The Open University of Israel
Department of Mathematics and Computer Science

Synchronization Complexity Metric

Thesis submitted as partial fulfillment of the requirements
towards an M.Sc. degree in Computer Science
The Open University of Israel
Computer Science Division

By
Peter Yastrebenetsky

Prepared under the supervision of Dr. Mark Trakhtenbrot

December 2009

Abstract

As multitasking environments and applications become more and more common, the quality and efficiency of these applications becomes more and more important. While for single-threaded programs there are many complexity metrics in use since the 1970's, so far there was no way to efficiently measure and compare the complexity of multitasking programs, and the affect of the concurrency on the overall program complexity.

In this thesis we propose a solution to this problem. Namely, we introduce a new metric that characterizes program complexity based on the kind and amount of the various synchronization means used for coordination between its concurrent components. Similar to McCabe's metric for single-threaded programs, the new Synchronization Complexity metric allows for assessment of the amount of tests needed to achieve a proper coverage in testing of a concurrent program. It also enables comparison between different system's implementations based on the synchronization complexity analysis.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Amir Pnueli starts his foreword to [6] with the following statement:
"It is widely agreed that the main obstacle to “help computers help us more” and relegate to these helpful partners even more complex and sensitive tasks is not inadequate speed and unsatisfactory raw computing power in the existing machines, but our limited ability to design and implement complex systems with sufficiently high degree of confidence in their correctness under all circumstances.”

This is the opening statement for the book dedicated entirely to the problem of model checking and formal verification, but the general intention of model checking and formal verification is, as also stated in the foreword, to ensure the correctness of the design at the earliest stage possible.

The methods of formal verification and model checking are used in software development in a limited way, partly because of their complexity and costs, and partly because of what is known as “the state explosion problem”. The state explosion problem, in a nutshell, is that the number of states in even a reasonably small system grows to be too large for it to be possible to be handled by a realistic computer. In order to avoid it, numerous techniques are used. Many of them basically take only a subset of the analyzed system, thus reducing the amount of system states to a number that can be handled [6, 29]. Another popular approach is that formal verification is performed not for the real system, but only for its abstraction, a design model. Once the design has been verified using model checking and formal verification methods, we still face the problem of validating the real system accordance to the design. To solve it we need the good old simulation and testing techniques, known in the industry as the “Software Testing” process [3].

The software testing is, generally speaking, a process based on performing experiments on the software prior to its deployment, and comparing the results of each of the experiments to the expected results based on the software specifications, customer expectations, or, in some cases where the behavior is otherwise undefined – common sense.

The definition of software testing, as stated in [3] is: “Testing is any activity aimed at evaluating an attribute or capability of a program or a system and determining that it meets its required results”.

In [26], it is defined slightly differently: “Software testing involves running an implementation of the software with test data. You examine the outputs of the software and its operational behavior to check that it is performing as required. Testing is a dynamic technique of verification and validation”.

As stated in [3], it is impossible to achieve total confidence by just testing, whatever form of testing it is. Due to the state explosion problem, it is not feasible to run tests that will reach **every possible state** of the system, especially in concurrent systems, where the theoretical amount of system states is exponential in the number of processes in the system

[17]. So, additional activities and methods are required to ensure software quality.

Some of the additional methods are based on the concept of “code inspection” [10]. The formal inspection process defined by Fagan in [10] is in fact in a widespread use in various, mainly safety critical systems' development processes [26]. Also, various alternative methods were developed for less formal peer code reviews, such as pair programming, or lightweight code reviews [7, 26]. Combination of system testing and code inspections is considered by some organizations as the optimal verification and validation technique [26], and even lightweight and informal code reviews help reduce the amount of errors found during the further development phases [7].

As noted in a software management oriented research published in 2001 ([4]), postponing error detection to later development phases results in very high fixing costs, up to 100 times more than if discovered during the design a coding phases, and higher.

Another observation in [4] is that the software development projects spend 40-50% of their effort on avoidable rework, in other words – almost half of the effort goes on fixing bugs which should have been avoided altogether, or at least discovered earlier. This observation also means that some rework is **unavoidable**, the fact which is important to the background of the current work.

As any code change, rework of existing code, whether necessary or unnecessary, will potentially introduce new defects. Part of this thesis will be dedicated to analysis of changes in order to estimate the effort needed to discover these defects.

The Goal of This Work

In this work it is shown that it is possible to evaluate the impact of the various concurrent programming patterns (mutual exclusions, accessing shared data, creating new threads and processes, etc.) on the programs' complexity.

For this, a novel Synachronization Complexity Metric is introduced, that provides the lower bound for the number of interleavings on the program language level, possible in the application under certain predefined conditions. This metric allows the programmers and testers to assess the impact of the various implementation choices for concurrency patterns on the overall complexity of their products. In particular, it shows the amount of unique tests required to cover the expected interleavings on the program language level (based on known branching and synchronization coverage models described below, and on the formal definition of “interleaving” given in chapter 2).

The novel metric is compatible with metrics currently in use for sequential programs, thus easing the effort of complexity comparison and competitive analysis of various solutions.

As part of the experiments during the work, the actual usage of the metric on real-life applications is demonstrated.

Metrics and Code Measures

Usage of metrics is based on selecting certain measures that are believed to be predictive of an aspect of system quality, and are used as an aid to requirements, design, test, and code reviews [3]. These measures include information which may be useful for the code readability and maintainability assessment. For example the ratio of lines of code per function – it is considered that a function should not be longer than two-three screens (some industry coding standards, like [15], set the limit to 200 lines of logical code). Another example: ratio of comments per line of code – it is considered to be bad practice to write code without any comments at all (as noted in another coding standard which is used widely in the industry – the MISRA C standard [13]). Knowing the ratio for the code pending the review will allow to save the time of the review if the code doesn't follow the minimum required on the metrics, and will assure better structured and better documented software. Static analysis tools often enforce these metric limitations (like the CCCC tool which will be described in details in chapter 4).

In [4] it is noted that peer reviews catch 60% of the defects. Although empirical findings in [18] show that the mere use of code metrics during peer review doesn't necessarily change the process effectiveness, the metrics can provide valuable information for both peer reviews [7] and software test designers [3].

There are various types of metrics, and one of them is a complexity metric. Probably the most well-known is the McCabe's cyclomatic complexity metric [19, 21], which is accepted as a basic measure in areas like safety critical development (e.g.: The MISRA C coding standards, already mentioned above, which are enforced throughout the motor industry), and generally in software testing [3, 18, 19, 26]. The McCabe's metric provides a number (CCN – Cyclomatic Complexity Number) which can be directly related to the amount of tests needed to perform full coverage testing based on the branch coverage model (various coverage models will be discussed later in this chapter). The number is also directly related to the branching of the code in question, thus providing an insight on the code complexity (hence the name of the metric).

The cyclomatic complexity number is calculated based on the control graph model of the program analysis. The graph consists of nodes and edges, where the nodes represent execution statements and the edges represent the transfer of control between those statements. For each possible execution flow of the program, there's a corresponding path in the control flow graph.

There are several equivalent definitions for the cyclomatic complexity number, described in details in [32]:

The cyclomatic complexity number value is calculated by the following formula:

$$CCN = e-n+2$$

In which:
e – The number of edges in the program's control flow graph,
n – The number of nodes in the program's control flow graph.

By this definition, the cyclomatic complexity number is in fact the number of different paths through the standard control flow graph model.

Additional and equivalent definition for the CCN is this:

$CCN = p + 1$

In which:
p – The number of binary decision predicates (i.e.: number of nodes with exactly two edges coming out of them).

Another and more intuitive definition for the CCN is this:

$CCN = R$

In which:
R - The number of "**regions**" – full circles defined by the edges of the control flow graph (including the region outside all the edges). This definition allows assessing the CCN value quickly by just looking at the flow graph, however it requires the graph to be planar, i.e.: no edges crossing each other.

Several examples of the CCN calculation will be given in chapters 2, 3 and 5.

The CCN is one of the first code complexity measures, and additional code complexity measures appeared later (for example data flow complexity measures; the effort measure; various object oriented programming metrics which measure coupling, inheritance depth, etc.) [18, 19, 33]. Code complexity is a number and for each one of code complexity measures, different numbers are considered "good". For CCN, for example, value over 10-15 [32] is considered bad, whereas for code coupling there's no precise definition of what a "good" or "bad" number is [15].

For sequential programs with a single thread of execution, CCN can provide valuable information with regards to the structured testing of the program. As described in [32], it is possible to assess the minimum number of tests needed for testing of the basic control paths (control paths which cannot be represented as a combination of any other control paths, i.e.: their representation on the control flow graph described earlier will differ by at least one edge) of the program (branch coverage).

However, in case of concurrent programs, the same code may provide different results on the same input, depending on the order (interleaving) in which the code statements are executed. In this case, the CCN won't provide enough information for assessing the amount of tests needed for adequate testing.

Metrics will be the main topic of this work, specifically a metric which allows assessing the amount of tests needed for adequate testing of a concurrent program. As other metrics, this will go together with a set of coverage detection techniques described below based on the prior work.

Test Coverage vs. Code Metrics

There are several different coverage models in existence, some of them are orthogonal, and some are somewhat overlapping. The common property of all the coverage models is that they come to measure test adequacy with respect to certain testing goal [10]. For example, test set which provides 100% statements coverage doesn't necessarily provide also 100% path coverage – statements might be covered during the tests fully, but not necessarily all the paths using the statements will be covered.
This is true in the following example:

Example 1.
The code:

```
#include <iostream>

int main(int argc, char *argv[]){
     bool fEnter = false;
     int val;
     std::cin >> val;
     if (argc==0){
          std::cout << "Nothing";
     }
     return 0;
}
```

The test set:

1. Run the code with no parameters; expect "Nothing" on the output.

The statements will all execute during the full execution of the test set, thus the test set provides 100% statements coverage. However, it doesn't provide full path coverage, because the path where no output is printed will never execute. The CCN value for the above code is 2 (The "if" is the only binary predicate, thus the complexity is 2), so we could know that one test is not sufficient for branch and path coverage by just looking at that number.

Obviously, number of tests by itself doesn't fix the problem. For example adding the test

2. Run the code with no parameters, expect nothing on the output

won't help, and the test will always fail.

Adding another test:

3. Run the code with a parameter, expect nothing on the output

will fix the issue and provide also 100% branching and path coverage.

In some cases, however, full statements coverage means also full path coverage, and the simplest example would be the classic C++ "Hello word!" program:

Example 2.
The code:

```
#include <iostream>
int main(){
     std::cout << “Hello World!”;
     return 0;
}
```

The test set:

1. Run the program; expect “Hello World!” on the output.

This set provides 100% statements, branching and path coverage. We could know that we only need one test in the set by looking at the CCN for the above program, which is 1 (no binary predicates).

Example 3

```
#include <iostream>
int main(){
     unsigned int I;
     std::cout << "Enter the loop counter value…";
     std::cin >> I;
     for (; I > 0; I--){
          std::cout << "\nIterating number " << I;
     }

     return 0;
}
```

The 100% branch coverage may be achieved easily by running the test set:

1. Run the program with I = 0
2. Run the program with I = 1.

However, that doesn't guarantee loop coverage. Loop coverage requires that each loop will be executed 0 times, 1 time, and more than 1 times, so in order to have full loop coverage, an additional test will be required:

3. Run the program with I > 1.

This example shows us that full coverage based on one criterion doesn't necessarily guarantee full coverage on any other criterion, and each test set should be measured for each coverage goal required separately.

Concurrency Coverage Models

Test coverage metrics provide valuable information regarding the test adequacy against the stated goals, and are in wide industrial usage. There are several tools and algorithms designed to provide full test coverage under certain coverage models. For example – the IBM® ConTest tool [5, 9], the Microsoft® CHESS tool [24], interleaving and concurrency oriented code review procedures [6], incremental structure testing [17], mock-based unit-testing [25], and many more

In [5, 12, 20, 31] there are several ways of describing context-aware/concurrency-aware

tests' development, including a coverage model for synchronization coverage adequacy described in [5, 20, 24]. The models described in [5, 20] will be described in details below, the model described in [24] is similar ot the model described in [5].

Concurrency Coverage Definitions

In [5] additional coverage model is defined. In it, all the pre-defined **synchronization points' types** are considered, and the coverage is considered full if during the tests all the synchronized sections have been observed to be "blocking" and "blocked". Synchronized sections that weren't observed in these states during the tests should be analyzed for either redundancy or test completion.

The **synchronization point** is a code statement which includes a potential of an interaction between the thread currently being executed and another execution thread, i.e.: **synchronization point is a point in the code which may cause an intentional interleaving**. The concept of interleaving and the concept of intentional interleaving will be defined later in this work, in chapter 2.

For example, in the code for SOUP message queue implementation (full listing can be found in appendix A), we can see this function:

```
void
soup_message_queue_append (SoupMessageQueue *queue,
SoupMessage *msg)
{
     g_mutex_lock (queue->mutex);
     if (queue->head) {
          queue->tail = g_list_append (queue->tail, msg);
          queue->tail = queue->tail->next;
     } else
          queue->head = queue->tail = g_list_append (NULL,
msg);

     g_object_add_weak_pointer (G_OBJECT (msg), &queue-
>tail->data);
     g_mutex_unlock (queue->mutex);
}
```

Statements `g_mutex_lock` and `g_mutex_unlock` are examples of a synchronization point. For the external callers, the function call to `soup_message_queue_append()` will, in turn, represent a synchronization point, encapsulating its internal structure.

There are many more various possible types of synchronization points, which will be described later in the work (Chapter 3), with detailed examples in different programming languages and systems.

This coverage model provides a way to calculate information regarding the test adequacy of the tests set for a concurrent system under tests. It assumes that there's a test set in existence and operates on the given tests, providing coverage information.

One of the difficulties in using this coverage criterion for the concurrency testing, is that it is not always easy to calculate, and requires certain code alterations. The main difficulty is to find at the run-time which execution path has been covered out of many possible.

Opposed to the single-threaded program when the given input defines the execution path uniquely, in multithreaded applications, it is not only the input for the given thread that defines the execution path but also inputs of other threads, timing and order of execution. These factors may not always be predictable or controllable during testing, so in order to know when a certain execution path is being executed certain alterations in the code should be made to make track of the execution of the program.

An example for such alterations can be found in the IBM's ConTest framework [9], the Mircosoft® CHESS tool [24], and a completing "desk checking" procedure described in [12].

The "desk checking" procedure is "an extremely effective code review technique used for early detection of sequential program errors" [12]. It defines a semi-formal code walkthrough which is, according to the authors' conclusion, a very beneficial in finding concurrency problems on the early stages of the development process.

The ConTest, designed as a framework for testing concurrent Java applications, relies on changes of the software bytecode that add, without changing the original functionality, calls to some of the ConTest modules during certain stages of the software execution (so called "coverage enabled irritators", as described in [9]). These "irritators" are design to make "interesting" iterations to occur, and in fact force certain orders of execution during different execution iterations to achieve the required level of coverage. The heuristics and methods to achieve that goal are described in more details in [9].

The results of each execution are analyzed by the ConTest components which then decided whether additional executions are necessary or the coverage requirement had been met The coverage model in use by the ConTest tool is defined in [5] and is described later in details in this work.

Another similar tool is CHESS [24], designed by a team of researches at Microsoft®. This tool is similar to the ConTest, except for wrapping Microsoft® .NET CLR or Windows API calls instead of the JVM libraries, and providing the coverage in much more systemized manner (it controls the scheduling and preemption of the tasks and allows covering all the possible interleavings systematically).

The metric suggested in this work allows using the ConTest, CHESS and other similar tools in a more controlled manner, so that at each point of the test execution the tester would be able to know how many paths were covered, and more importantly – how many are still remain to be covered.

The model described in [5] is based on these synchronization points' types:

Synchronization point	Description
Try-lock	Entrance to a mutual exclusion portion of the code, and represents a mutex call that can fail or succeed (but not necessarily block), for example a C pthread library `pthread_mutex_try_lock()` call.
Wait	Wait on a condition, such as a select call in a POSIX system, or `pthread_cond_wait()` call in a C pthread library.
Semaphore-wait	A semaphore or critical section entry point, where a task may proceed, or will be blocked if not
Semaphore-try-wait	A synchronization method similar to try-lock.
Notify	A synchronization method used to signal a waiting (on **Wait**) thread that the condition it is waiting for has been met. For example, POSIX `signal()` call.
Volatile access	Access to a volatile variable

Table 1: Synchronization points' types for the synchronization coverage model

The detailed examples of these synchronization points' types will be given in chapters 3 and 4 of this work.

There are several additional difficulties when using this coverage model. These difficulties were not addressed in [5, 9, 12], and the most important of them is the assumption that the software is encapsulated (i.e. no external events can change the internal state of the system).

This is not true for many concurrent programs on embedded platforms, which have also interaction with hardware using shared memory access (through, for example, C volatile variables) or interrupts handling. It is hard to consider system interrupts in the model, as they can occur independently of the software under test; however it is crucial for the testing process success that volatile variables access would be considered under the synchronization coverage models.

Thus, the model above, as it was defined in [5], cannot be considered complete, and requires certain supplements, which will be detailed in chapter 2.

Interleaving definition and Coverage Criteria Based On It

In [20] a formal definition for "interleaving" is provided:

"Access-Relation"-based Definition of "Interleaving"

"Consider a concurrent program P, executed under an input I, consisting of M threads: 1,2,…,M. Similar to previous work [33], we model the concurrent execution of P by a sequence of shared variable access events. We use E to denote the set of all shared variable accesses, and P_E for a program P with access set E under a given input. At any moment only one thread i is active and executes one event. When i finishes, one thread j (j might be equal to i) will be chosen and executes its next event. The event execution order within each thread is fixed. The order among different threads might change. Each different order to execute P_E is called an **interleaving**. Formally speaking, an *interleaving* $\prec$ of P_E is a total order relation on E. An event e is executed before an event e' iff $e \prec e'$. The whole interleaving domain of P_E is the set of all total order relations on E that maintain the sequential order within each thread."

In [20] several interleaving coverage criteria are proposed based on the above definition, starting with the most exhaustive definition. These coverage criteria are based on rules according to which the interleavings are being chosen to be considered for coverage. Basically they define "filters" on the set of all the possible interleavings in the system, which select only the certain types. These selection filters are based on certain usage patterns chosen by the authors (there are patterns not discussed in [20], for example "write-write" access interleaving).

The authors performed costs analysis for the criteria suggested in [20], and found that some of them can be achieved in time polynomial in the number of threads and shared variables.

However, there was no practical suggestion as to how to implement the coverage criteria in practice. The coverage criteria were defined based on a model representation of the system, which doesn't allow precise calculations based on the actual code. As it will be shown in chapter 5, even the simplest "innocent" changes to the code may influence greately the amount of possible interleavings, and when testing a model, rather than the actual source code, these difference may be lost.

Work Outline

In Chapter 2, an additional definition will be provided for the concept of "interleaving", and the relation between the two definitions shown in the work will be analyzed. These two different definitions are at base of two different synchronization coverage models that this work will relate to.

Then, there will be provided a formula which will allow calculating a code metric supporting the coverage models described above. The formula, as defined, will allow calculating the actual metric values from the real-life source code, as opposed to the cost

analysis done in the prior work [20], which only allows estimating boundaries based on the model representation of the system.

The definitions for concepts used in the formula will be provided and explained, and the connection between different definitions of the concept of "interleaving" (the one above, and an additional one that will be shown in chapter 2) will be shown and proven.

Also, the usage of the formula parameters to adjust the usage of the metric to the required coverage model (of these mentioned earlier and defined in [20]) will be explained and exampled,

In Chapter 3, the metric will be tested for metric properties defined by E.J.Weyuker in [33]. These properties provide a way to evaluate metric soundness, and several of well-known and accepted code metrics (e.g.: the McCabes cyclomatic complexity) have already been tested for these properties. Hence it is essential to check that the metric defined in this work also satisfies these soundness properties in a manner comparable with the existing measures that are in use in the industry. It is agreed upon [18] that not satisfying all of the properties doesn't necessarily disprove soundness of the metric; however a metric that does not satisfy several or most of the properties may not be accepted as valid and sound.

In Chapter 4 an example of a tool implementing the metric will be shown. The tool is based on an existing tool for C and C++ code measuring, the CCCC. The changes and additions to the tool in order to implement the metric will be shown and explained.

In Chapter 5, examples of usage of the metric will be provided and analyzed, including the examples of comparative analysis done on various implementations of the same functionality. Several different implementations of HTTP servers will be analyzed and compared using the metric, and the usage of the comparison results to improve the existing code will be demonstrated.

In Chapter 6, additional topics related to this work are considered for the future research.

Chapter 2

Interleavings

In the previous chapter a definition for interleavings was quoted from [20]. According to it, an interleaving is a total order relation on the set E of shared data access events in the system P. For each set of events P_E (for the given program P under a given input), there may be several different interleavings (total order relations between the events).

The order of event follows directly from the order of statements' execution by each thread, i.e.: for each execution flow path, there is an interleaving by definition quoted from [20] in the previous chapter.

In [21], the CCN is defined based on the graph representation of the execution flows. The graph as defined in [21] is for a single threaded program (single path of execution), but a similar graph can also be built for multithreaded programs as well.

The definition of such graph will be given below as part of an additional definition for the concept of “interleaving”. This definition will be marked ”Path” to distinct it from the definition previously quoted from [20], and which will be marked as ”Relation” (being defined through relations). Later in the work, it is shown that both definitions are closely related.

Examples of Interleaving

Below is a listing of a simple UNIX program illustrating the concept of interleaving:

The program creates (using the `fork` system call) two processes, each executing a different `printf` statement and a `sleep` statement.

```
#include <unistd.h>
#include <stdlib.h>
int main(){
     int pid;
     if ((pid = fork()) != 0) {
          sleep(1);
          printf("\nChild Process\n");
     }
     else {
          sleep(1);
          printf("\nParent Process\n");
     }
     return pid;
}
```

These are the results of several consecutive executions of the program (all on the same PC with Intel Core 2 Duo processor, running Microsoft Windows XP and Cygwin UNIX emulation environment). The program was run from the console, and below is the capture of the console with the results of the program several executions. In this capture we see

that each time the program executes the resulting output differs: sometimes the two strings are printed out separately, and sometimes there are concatenated, in different order. The interleaving occurs in all the cases when the new-line character is expected, probably because of the implementation of the `printf` function in the system

<Start of capture>
<Execution starts>

Child ProcessParent Process
<Execution ends>

<Execution starts>

Child Process

Parent Process
<Execution ends>

<Execution starts>

Parent ProcessChild Process
<Execution ends>

Parent Process
Child Process

<Execution ends>

<Execution starts>

Parent ProcessChild Process

<Execution ends>

<Execution starts>

Parent Process
<Execution ends>
<End of capture>

We can see that the order of the sentences printed by each of the two processes created by the program differs between runs. The processes run concurrently, in some cases the behavior of one process can affect the other (for example, in the last run the parent process terminated before the child process was woken after the sleep, and caused for it to terminate before executing part of its code). Each run that provides a different result is in fact an **interleaving**.

Another example of interleaving is using **volatile** variables:

```
volatile int* foo = 0x8200A;
while (*foo != 0);
```

In this example, the different execution paths can occur because of concurrent execution of operations in the code above and a hardware (or a separate software) application that changes the contents of the memory address 0x8200A. For example – it can result in infinite loop if the hardware never in fact updates the value in the memory location, or on the other hand the loop may never be executed if the value is already not 0 when the program execution reaches the `while` statement. Such examples are common in embedded systems or device drivers.

"Execution Path"-based Definition of "Interleaving"

Intuitively, if we have more than one thread, and it is possible that when executed several times, the order of the commands of all the threads will not remain constantly the same, then there must be an interleaving (like the examples shown above, where different runs of the same code provide different results, i.e.: clearly the execution steps were different although the original source code, and in the case of the first example – the input, never changed).

There had already been given a definition for "Interleaving" as a relation between shared variables' accesses. Here another definition will be given, which is based on execution paths. The reason for giving this new definition is so that it could serve as a "bridge" between the complexity measure used for single-threaded applications (the CCN) and the new synchronization complexity metric defined in this work. Later in the chapter a discussion will be held regarding the relation between the two diffinitions, in order to tie the new synchronization metric with the prior work.

In mathematical formalization, the definition of **Interleaving** is this:

Let G be a directed graph representing all the possible control flows of the application. Each path in the graph is a single execution flow of the application.

Let S be the set of states of G, and E the set of edges (transitions).

S is the product of states all n possible threads of the system:

$S = S1 \times S2 \times \ldots \times Sn$. According to this definition, each member in S is in fact a n-dimensional tuple, where n is the number of threads in the system. Each element in the tuple represents a state of the appropriate thread when the whole system is in the state represented by the tuple.

Each edge $e \in E$ connects two states $p, q \in S$, so that p and q differ by exactly one element of the tuple (i.e.: each transition in the path changes state of exactly one thread in the system, a transition cannot occur without changing a state of at least one thread, and it cannot change states of more than one thread).

Let *P* be a path in the graph *G* (which represents an execution flow of the application). E_P is the set of edges in the path, and S_P is the set of states. Since *P* is a path in the graph *G*, it means that $P \subseteq G$, thus $E_P \subseteq E$ and $S_P \subseteq S$.

Each different path in the graph *G* representing a different execution flow of the system given the same input (starting node of the path) is an **interleaving**. Each path (interleaving) can be represented by the set of edges that it covers.

This new definition differs slightly from the one given in chapter 1 in the semantics and methods of definition used, so that the common language would be kept when using the CCN in this work. However, as it will be shown in Lemma 1 below, the definitions define virtually the same thing. The difference is that the "relations" definition given in chapter 1 refers only to the order relation of shared variables' access events, whereas the new "paths" definition can be used to define interleavings in a finer granulation. However, as it will be shown in Lemma 1, for two different execution paths, there must be two different order relations, thus every interleaving by the "paths" definition, is also an interleaving by the "relations" definition. Thus the boundaries and costs calculated in [20] still hold when using the new definition.

Lemma 1: (a) For each interleaving by the "relations" definition there exists at least one path in the execution flow graph of the multithreaded program , and (b) for each such path there exists exactly one interleaving relation that implements this path.

The lemma specifically mentions multithreaded programs, since for single threaded programs there is an execution flow path which has no interleaving relation correlating to it. The reason for that is that by the "relations" definition (defined in chapter 1), there are no interleaving relations for single threaded programs since there's no shared data to access. Since the work is targeting multithreaded programs only, this corner case is excluded from the lemma.

Proof of Lemma 1:

a)

1. Let $\prec$ be an interleaving relation on a certain set of data access events during the program *P* execution with a certain input.
2. Shared data access is either a "read" (which means that a value of the shared data is assigned internally in the accessing thread), or "write" (which means that a value internal to the accessing thread is assigned to the shared data).
3. Each data access has to be an execution statement which changes a state of at least one thread (according to (a-2), either "read" access which includes an internal "write" or a "write" access to the shared variable), thus each event of data access ordered by the relation $\prec$ has also a representation as an edge in the flow graph *G* for the program *P* built as defined above.
4. Thus, there is a path in *G* in which all the data access events ordered by $\prec$ appear in the same order.

b)

1. Let *P* be a path of an execution flow in graph *G* for the given program with the

given input. Since we're limiting the discussion to multithreaded programs, there is a point in the program where a thread will be created, thus there has to be a point with shared data access (passing the control to the thread and its initialization based on the main thread state).

2. Let S'_P be set of states in P where the state of a thread changes as the result of a shared data access. Since there is at least once such state, as described above, this set is not empty.
3. By definition, this set correlates with a set of events E of shared data accesses as defined in [20], for the same run of the same program with the same inputs as represented by P: for each state $s \in S'_P$ there's an event $e \in E$, so that a state of a thread is changed in S'_P as the result of the event e.
4. The path P is a directed sub-graph of graph G, and the states in S'_P can be sorted by order of their appearance in P. The series of events correlating to S'_P and ordered so that each event will be in the same place as the state it correlates to, is interleaving by the "relations" definition.
5. Let P be a path of an execution flow as described in (**b-1**), and assume there's more than one interleaving it relates to. It means that more than one order of shared data accesses is possible in the same execution flow path.
6. However, the execution flow path defines the order of every state change, including those imposed by the shared data accesses (according to (a)). Thus there's a contradiction to the assumption => there could not be more than one interleavings relating to the same execution flow path.

■

The meaning of Lemma 1 for this work is that for each path of execution flow of a multithreaded program, which is an interleaving by the "paths" definition (defined in chapter 2), there's an exactly one interleaving by the "relatons" definition (defined in chapter 1). Thus, the boundaries described in [20] for various coverage models are valid when discussing interleavings based in the definition (2), including the boundaries provided by the cost analysis in [20].

From this point onward, this work will only discuss interleavings as defined in the "paths" definition in this chapter.

Intentional Interleaving

In actual software programs, interleavings can be caused at the machine instruction level, and even a single threaded program will have different interleavings (possible execution paths) because of the interrupts. For example, single threaded program can have different interleavings due to hardware interrupts.

Intuitively it is easy to see that there are two different kinds of interleavings – "intentional" and "unintentional". While unintentional interleaving are those caused, for example, by interrupts or other external unpredictable events, intentional interleaving is a case where the programmer intentionally added code that can potentially lead to more than one execution path as the result of concurrency.

Examples of Intentional and Unintentional Interleaving

In the examples above there are intentional and unintentional interleavings. Using the volatile variable may trigger an intentional interleaving at every access to it. Using calls like fork will cause intentional interleavings between the child and the parent tasks until the next statement for each task. Using the sleep statement will cause intentional interleavings as well since while one task sleeps, another will certainly be running.

However, there may be unintentional interleavings which the programmer didn't want to occur:

```
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>

#define THREADS 3

void *thread_start (void*);
char string[100] = {0};
int counter=0;
int main(){
     int i;
     pthread_t pid;
     pthread_attr_t attr;
     pthread_attr_init(&attr);
     for (i = 0; i < THREADS; i++){
          pthread_create(&pid, &attr, thread_start, NULL);
     }
     sleep(1);
     printf("\n%s - %d\n",string, counter);
     return 0;
}

void *thread_start(void*arg){
     pthread_t pid = pthread_self();
     sleep(1);
     sprintf(string, "Thread %d counter = %d\n",
               pid, counter++);
     printf(string);
     return NULL;
}
```

The program creates 3 concurrent threads, each of them incrementing a shared variable, and printing it out.

Below are the results of several consecutive executions of the program (all on the same PC with Intel Core 2 Duo processor, running Microsoft Windows XP and Cygwin UNIX emulation environment).

The program was run from the console, and below is the capture of the console with the results of the program several executions. In this capture we see that each time the program executes the resulting output differs: the order of the thread prints and the resulting values are not consistent between the runs. We can also see that in some cases all

the printouts or some of them are identical, although it would be expected that each thread would print a different string, with the last one being duplicated from the `main` function. In fact in all the results, the `main` function printed an unexpected result (marked **bold** in the capture) of only the `counter` value.

We can see clearly in the capture below that using unprotected global variables "`string`" and "`counter`" leads to unpredictable results in the output if all the threads are running in the same scheduled time (all sleep the same period, and are scheduled without priorities). This is clearly a behavior which is not wanted in a normal system: a so called **racing** between the threads; it is considered as a bug.

The access to the shared variables is an interleaving, but it was not intended by the programmer to have these interleavings.

<Start of capture>

<Execution starts>

- 0
Thread 6685208 counter = 2
Thread 6685208 counter = 2
Thread 6685208 counter = 2
<Execution ends>

<Execution starts>
Thread 6685072 counter = 1
Thread 6685344 counter = 2

Thread 6685344 counter = 2
- 3
<Execution ends>

<Execution starts>
Thread 6685208 counter = 0

Thread 6685344 counter = 2
- 1
Thread 6685344 counter = 2
Thread 6685344 counter = 2
<Execution ends>

<End of capture>

There are many more potential bugs related to the unintentional interleavings, these

include, for example, well-known problems like buffer overflows and other memory access violations which may result in data corruption, stack corruption, and other risks.

Also, we regard interrupts as unintentional interleavings, since these cannot be anticipated by the programmers and can (and will) occur at almost any possible point of time. Bugs occur in interrupt handlers as well, but these can be avoided by full and exhaustive testing of the interrupt handler routines (which are usually compact and uncomplicated by nature, thus exhaustive testing for them is feasible).

Work Assumptions

In this work we will ignore unintentional interleavings, and will only discuss intentional interleavings – code that was specifically and intentionally marked by the programmer as a potential interleaving trigger (by using the synchronization points, that will be discussed later on in the work).

For this work, it is also assumed that the synchronization is an atomic operation, and no interleaving on the machine instructions level is possible during the execution of a synchronization statement. This assumption is not restricting the application of the methods described in the work in the real world, since the atomicity can be achieved using various known techniques, for example [11] and [30].

Definition: Intentional Interleaving

After the intuitive explanation and examples, it is time to formally define the concept of the intentional interleavings.

Intentional interleaving is an interleaving (a different possible execution path for the program under the given input), caused by an explicit statement in the code which by its nature can cause several alternative execution flow paths for the multithreaded application. For example, the `fork` system call in the example above is such explicit statement.

It is similar to a branching statement for a single-threaded application, except that the result of this “concurrency” branching will be different order of state changes in different threads.

During the discussions below, the following notation will be used:

G: Graph of all the states and transitions (as defined above) possible in the given application (may be also referred as "the application *G*"). Reminder, the set of states and transitions of the graph *G* is defined as follows:

$S = S1 \times S2 \times \ldots \times Sn$. According to this definition, each member in S is in fact a n-dimensional tuple, where n is the number of threads in the system. Each element in the tuple represents a state of the appropriate thread when the whole application is in the state represented by the tuple.

Each edge $e \in E$ connects two states $p, q \in S$, so that p and q differ by exactly one element of the tuple (i.e.: each transition in the path changes state of exactly one thread in

the application, a transition cannot occur without changing a state of at least one thread, and it cannot change states of more than one thread).

Additional definitions:

$E_i(G)$: Set of all the interleavings possible in graph (systemapplication) *G* under a given input *i.*

$E_i^{int}(G)$: Set of all the interleavings in systemapplication *G* under a given input *i,* and only include the execution of the synchronization statements put intentionally by the programmer in the program (i.e.: *intentional* interleavings). This set will not include execution paths (*interleavings*) influenced by unintentional synchronization statements (for example – hardware interrupts).

In this thesis, only the intentional interleavings are considered when discussing the novel measure. This is because it is intended to show a measure that provides a **minimum** boundary for the complexity encurred by the synchronization constructs in the application. Based on the definition, the intentional interleavings are subset of the set of all interleavings possible in the application under the given input i (i.e.: under the given input i, $E_i^{int}(G) \subseteq E_i(G)$). In the next lemma, it is shown why the intentional interleavings are part of any minimal set of interleavings that can occur in the application running under a given input, thus ensuring that the rest of the discussion, concentrating on the intentional interleavings, does in fact provides the required minimum boundary.

In the lemma 2 below, the interleavings that can occur when the application is running under a given input *i* are discussed Using the lemma, it is shown that if the program under the given input *i* includes a synchronization statement that will be executed, then the *intentional interleavings* (interleavings incurred by that particular statement) will necessarily be part of the execution flow graph of the application reachable when running under that input. This to oppose the unintentional interleavings that cannot be anticipated by their nature.

Intuitively speaking, the proof of the lemma below shows that if, for example, we have a `fork` call in the application – we can assume that the least interleavings possible are the interleavings incurred by the `fork` call. It is possible however that there will be additional interleavings incurred, for instance, by the clock interrupt.

Lemma 2: The set of all intentional interleavings is the smallest set of interleavings possible in the application during the executions under a given input i
(i.e.: $\forall[E_i^{sub}(G) \subseteq E_i(G)],\ E_i^{int}(G) \subseteq E_i^{sub}(G)$).

Proof of Lemma 2. Intuitively the Lemma claims that for any execution of the application with the given input *i*, the total amount of possible interleavings cannot be less than the number of the intentional interleavings possible for an execution with the given input *i*.

Let *G* be an application.

Assume there's a set $E_i^{sub}(G) \subseteq E_i(G)$, so that $|E_i^{sub}(G)| < |E_i^{int}(G)|$. According to the definition above, $E_i^{int}(G) \subseteq E_i(G)$, i.e.: in an execution of a application, if there are intentional interleavings in the execution path under the given input *i* then they will be a part of all the interleavings possible during the executions of the system under the given input *i*. However, according to the assumption, $|E_i^{sub}(G)| < |E_i^{int}(G)| \Rightarrow$ there's an interleaving $e \in E_i(G)$, so that $e \in E_i^{int}(G)$ and $e \notin E_i^{sub}(G)$, which contradicts the definition.■

Minimizing Interleavings

It is obvious that the theoretical upper bound to the number of interleavings is the product of all the states in all the threads in the system (i.e.: exponential). However, in real life, this boundary is most likely not to be met. In [2], for example, it is shown that some values may never be assigned to variables on certain execution paths, thus the states of the system which include these values will not be reachable, and the actual interleavings represented by these states will never occur.

The goal of the work, as was mentioned in the first chapter, is to provide a way to calculate **boundaries** to the number of possible actual interleavings in an actual real-life program. The intention is to define a **lower bound**; however **upper bounds** will be discussed as well. The lower boundaries will be estimated using the synchronization complexity metric which is defined below.

Synchronization Complexity Metric (SCM)

Synchronization Points' Types

The synchronization metric is calculated based on a static code analysis; it helps to estimate the required amount of tests which would be needed to provide full coverage under the synchronization coverage model defined in [5], and the coverage models hierarchy defined in [20], and discussed above..

The metric is based on static code analysis, during which the statements which belong to one of the synchronization points' types defined here will be found and analyzed. The synchronization metric described in this work provides **synchronization branching coverage estimation**, i.e.: the metric will provide the number which will be the minimum amount of tests required in the test set, so that each branch of code execution which include a synchronization statement will be tested **at least once** with regards to synchronization (i.e.: **all the interleaving options for each occurence of a synchronization point will be covered at least once**).

In order to present the concept of the synchronization point in this work, let us first return to the first example of interleavings:

```
#include <unistd.h>
#include <stdlib.h>
int main(){
     int pid;
     if ((pid = fork()) != 0) {
          sleep(1);
          printf("\nChild Process\n");
     }
     else {
          sleep(1);
          printf("\nParent Process\n");
     }
     return pid;
}
```

In order to cover all the interleaving options for each synchronization point at least once, we need the following tests:

1. After the `fork` call let the child run
2. After the `fork` call let the parent run
3. After the `sleep` expiration for the parent – choose the child to run first
4. After the `sleep` expiration for the child – choose the parent to run first.

5. After the `sleep` expiration for the parent – choose the parent to run first
6. After the `sleep` expiration for the child – choose the child to run first.

The steps 3 - 6 may be redundant, depending on the system, since the order of getting into the waiting state for each task is preset by the order of execution (steps 1 & 2). However, since the times are identical and the sleep precision may not be small enough to allow distinction for when to wake each process, both of them may be scheduled by the operating system to wake up at the same time, and the choices will again be relevant.

Note, that the condition in the "`if`" statement affects the CCN of the program, had it been single threaded. However, that is misleading, since for each of the tasks (the parent and the child), there's no actual branching option: for the parent task the “if” statement will **always** evaluate to false, while for the child task the same statement will **always** evaluate to true. Thus, in fact there's no branching in the execution paths of the processes in question. This nuance will be discussed later in chapter 6.

The table below lists several **basic** synchronization points’ types. The types have been chosen based on the coverage definition in [5], and completed with additions in order to cover cases not covered by the types in [5] (the last three items in the table are the completion: volatile access to cover the cases where the system is not encapsulated to the software, thread/task creation and voluntary preemption cases not covered by [5]).

Synchronization point	Description
Try-lock	Entrance to a mutual exclusion portion of the code, and represents a mutex acquire call that can fail or succeed (without blocking), for example a C pthread library `pthread_mutex_trylock` call.
lock	Entrance to a mutual exclusion portion of the code, and represents a mutex acquire call that can block or succeed, for example a C pthread library `pthread_mutex_lock` call.
Unlock	Exit from the mutual exclusion portion of the code, and represents a mutext release call, for a example a C pthread library `pthread_mutex_unlock` call**.**
Wait	Wait on a condition, such as a select call in a POSIX system, or `pthread_cond_wait` call in a C pthread library. Functions like `sleep` or `delay` can also be considered as "wait" synchronization points.
Notify	A synchronization method used to signal a waiting (on **Wait**) thread that the condition it is waiting for has been met. For example, POSIX `signal` call, or `pthread_cond_signal` pthread library call.
Pass Control	A synchronization method used to release the CPU control by a thread. For example, Java method `yield`.
Volatile access	Access to a non-synchronized variable with concurrent access, for example a C volatile variable.
Task/Thread Initiation	Creation of a new thread, for example a C pthread library `pthread_create` call, a POSIX `fork` system call or C `exec` calls.

Table 2: Basic synchronization statements' types for the metric.

The synchronization points' types defined above are **basic**, since they define **types, or classes,** of synchronization points which will behave similarly in different systems and implementations, rather than specific functions.

For example, the **wait** synchronization point may be implemented as a call to either `sem_wait`, `pthread_cond_wait` or `mq_receive` pthread library function calls. Each of these implements different functionality, but the synchronization effect is the same: they will block until an event occurs.

Synchronization Patterns

Various synchronization mechanisms can be constructed using the implementations of the basic synchronizations, and such mechanisms will be called **synchronization patterns**. Many times such patterns appear as a single function call in the program code, thus for the analysis they can be treated as separate synchronization points . Good example is sending a message to a queue: a function call `SendMessage` (or a similar name) would often conceal a combination of (mutex) lock and unlock synchronization points.

Detailed examination of various examples of synchronization patterns will be done in chapter 3.

Cost Parameters Definition

For each type of a synchronization point listed above, the following parameters are defined:

I. **Interleaving potential** (IP) – this parameter is the novelty of the metric. The parameter gives a cost of the synchronization point as a linear function of potential interleavings, assuming another thread is synchronizing on the same point.

In more details, this is the *minimum* number of interleavings (branches in the execution paths graph) that are created by the use of the synchronization point of the given type on the given system or implementation. In other words, this is the *minimum* number of other threads that can preempt the current thread at the considered synchronization point.

This number should be calculated for each operating environment (specifically – the operating system scheduler and/or implementation of the synchronization point – C or Java, for instance). However, once calculated – the value will never change in this system (e.g.: once calculated the *IP* for the "Task Initiation" under the Linux real-time scheduler – it can be used on all the programs that use this synchronization type and intended for this system).

For example, consider the synchronization point of the "Task/Thread initation type", intended to spawn a new task/thread. When used in a system where tasks/threads with the same priority are treated by a Round-Robin or a similar algorithm (as in the case of UNIX `fork` system call), the control can switch to the newly created task/thread or not, in which case it remains in the calling task (or switches to some other ready to run task if such available, and the next parameter, the competition potential, will reflect it). Thus, the interleaving potential of this synchronization point type is *2*: if called, there are *at least* two possible interleavings. When the program is written, it is not known which of the two will occur at the run-time; at each run of the program a different interleaving may occur,

Situation is different when scheduling is based on fixed priorities and no two tasks can have the same priority, like in µCOS operating system used in various embedded devices. Namely, when a new task is spawn, there can only be one interleaving option: whether the current task will continue and the new task will wait (if its priority is lower than the current), or the new one will start immediately and the current will be put on hold (if the new task's priority is higher than the current). The decision is deterministic deterministic (since the priority is defined in the spawn command by the caller), and will always be the same when the program is run under the same input, Therefore, in this case the *IP* of the same synchronization point will be 1.

Thus, in order to properly use this parameter, the user of the metric (usually the programmer or tester) *must* be familiar with specifics of the target system for which the application is being tested. A sample analysis of such parameters for

some specific systems will be shown in chapter 3.

II. **Competition potential** (CP) – this parameter equals to the total number of threads competing for the synchronization point. The threads competing for the synchronization point are the threads that can possibly change states next to the execution of the synchronization point (i.e.: In the graph representation discussed above – these number of tuples in the graph that can be immediately reached from the current state as the result of the synchronization point execution). Intuitively *CP* equals to the number of threads that can be in their ready-to-run state immediately after execution of the considered synchronization point. During the static analysis stage, at which the SCM is calculated, his value can only be estimated, and can be different for the same synchronization point type, whet used in different portions of the program.

The more threads are competing for the same synchronization point, the higher is the competition potential. The competition potential value is an integer greater or equal to 1 (there has to be at least one thread in the system).

For example, let us look again at the task initiation synchronization point under the Round-Robin scheduling algorithm. The *IP* would be 2 (the control either goes to the newly created task, or not), but the competition potential, that influences the resulting SCM value as well, denotes the number of tasks to which the control may pass in this particular instance..

As opposed to *IP*, the *CP* does not depend on the underlying scheduling algorithm, as the amount of threads/tasks created by the program depends on its logic, not the scheduling algorithm that manages the program execution. Thus, as opposed to *IP*, which is constant, the *CP* varies for each occurrence of the synchronization statement in the code, based on the program logic.

Generally, we will treat all the threads as if they exist throughout the application execution life time. This is due to the fact that the metric in question is intended to be calculated based on the static code analysis, and at this stage it is hard to evaluate the exact periods at which different threads will exist in the system.

The possibility of extracting the information from the source code depends on the language and coding standards in use (examples of limitations of the existing tools will be shown in chapters 5 and 6) and sometimes it may not be feasible to extract that information (for example when the number of actual threads/tasks in the system depends on the input). When this information is not available from static code analysis, it should be assumed that there're at least 2 threads competing for each synchronization point. Else, if it is possible to extract this information from analyzing the source code, and the value of this parameter is 1 – it means that the synchronization point is only used in a single thread, and thus redundant.

For simpler programs, it may be possible to simulate execution and analyze the amount of threads accessing the synchronization points based on the given input (as it is done, for example, in [5]), but this is not feasible for more complex

applications, or applications with many different input options. The process of extracting the information is beyond the scope of this work.

Formal Definition of the SCM (Synchronization Complexity Metric)

The synchronization complexity metric (SCM) is a branching metric, and refers to execution paths' branching as a result of concurrent execution. While McCabe's CCN is the most common branching complexity metric for sequential programs (see Chapter 1), SCM can be viewed, in a sense, as its extension to the case of concurrent programs.

The metric, with the parameters defined above, can be used to estimate the minimum number of tests required to provide full branching coverage for every synchronization point in the program **at least once**, for branching as a result of concurrency ("concurrency" branching).

Calculation of SCM combines the classic branching metric with the newly defined parameters. The obtained value is affected by all the branching options – the single-thread condition branching (the McCabe's metric) and the synchronization branching based on the concurrency parameters defined above. This resulting value provides estimation for the amount of testing needed to achieve the complete branching coverage (i.e. every branch covered at least once) for all the possible system execution paths. Recall that we assume existence of only intentional interleavings, as discussed above, in the section titled "Work Assumptions" in this chapter).

The synchronization complexity metric will now be presented. The metric relies on the CCN, and provides an extension to the existing McCabes cyclomatic complexity measure.

As already mentioned before, the metric is based on static code analysis, during which the statements which are categorized to one of the synchronization points' types defined here will be found and analyzed. The synchronization metric provides branching coverage estimation (i.e.: all the interleaving options for each such statement or predicate will be covered at least once). For single threaded programs, the metric that provides the minimum number of test required for the full branch coverage testing (all interleavings covered for each predicate) is the CCN. Thus, the SCM metric is defined so that when applied on a single threaded program, its value will be the CCN for that program. It is designed in that way so that it could be more easily understood and integrated into the existing processes.

Following is the formal definition of the new metric:

$$SCM = \Sigma(CCN_{sp} * IP_{sp}^{\,CPsp-1}) \qquad (1)$$

Here for each synchronization point *sp* :

- CCN_{sp} is the cyclomatic complexity number of the branch at which the synchronization point was detected (see definition and a detailed example below)
- IP_{sp} is the interleaving potential of the synchronization point
- CP_{sp} is the competition potential of the synchronization point.

The CCN_{sp} is defined as follows. As a reminder, CCN is defined to be a value charactering the entire execution flow graph $G(S,E)$ of a single threaded program [16]. In order to find CCN_{sp} , a sub-graph $G_{sp} \subseteq G$ is defined as follows:

$\boldsymbol{S_{sp}}$ – all the states in S reachable from the state $b_{sp} \in S$. The state b_{sp} is the state in which the application was *after* executing the *most recent* branching statement (for example `if` or `while)`, on the execution path leading to *sp*.

$\boldsymbol{E_{sp}}$ – all edges in E, connected to states in S_{sp} (any direction).

CCN_{sp} is then defined as cyclomatic complexity of the graph G_{sp}

For example consider again the following program:

```
1.  #include <unistd.h>
2.  #include <stdlib.h>
3.  int main()
4.  {
5.     int pid;
6.     if ((pid = fork()) != 0)
7.     {
8.        sleep(1);
9.        printf("\nChild Process\n");
10.    }
11. else
12.    {
13.       sleep(1);
14.       printf("\nParent Process\n");
15.    }
16.    return pid;
17. }
```

The `fork` system call is a synchronization point on the top branching level (i.e.: the CCN_{sp} for it will be the CCN for the whole program if analyzed as a simple single threaded application). In this program, the CCN for the whole program is 2 ($CCN = p+1$, where p is the number of binary decision predicates . In this case there's only one such predicate – the `if` Statement).

The `sleep` system calls are each inside the branches created by the `if` statement. Each branch includes simple non-branching statements:

First branch:

```
8.        sleep(1);
9.        printf("\nChild Process\n");
```

Second branch:

```
13.       sleep(1);
14.       printf("\nParent Process\n");
```

Graphically, the execution paths graph for the example will look like the figure 1 below, with the synchronization statements marked in different colors (numbers in the circles

represent the relevant line numbers of the example code):

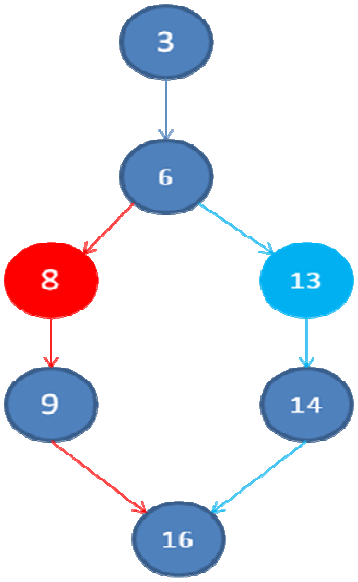

Fig. 1. Graphic representation of the possible execution paths of the sample program.

From the graph it can be clearly seen that each of the synchronization points is on a single possible execution path. Thus, in both cases the CCN_{sp} for the `sleep` statements will be 1 – the CCN value of the branches in which the synchronization point was found.

From the definition of SCM, it is clear that :

- in case of a single thread ($CP_{sp}=1$), the value of SCM will be the same as CCN for the same code branch
- the value of SCM can never be less than the CCN for the same program.

Based on the above definition, SCM can be used to describe the complexity incurred by the synchronization points over the existing code complexity. The usage of CCN is necessary to allow comparability of SCM values for single threaded (which would equal the CCN) and multithreaded implementations.

The value of SCM is exponential in the number of application threads competing for the same synchronization point (because of the *CP* that magnifies the concurrency impact exponentially based on the current number of threads); however this by itself should not be an immediate obstacle to using the metric. Most of the real-life applications have only handful of threads competing over the same resources, and thus require synchronization. For example, in [5], 16 classes out of 575 (3% of all classes) having synchronization primitives is considered reasonable. For such loosely coupled applications where there are no more than 2-3 threads competing for a single synchronization point, the number of interleaving options represented by the SCM will be feasible for full exhaustive testing.

Also, the metric can be used for **comparative analysis** between various implementations, thus only the ratio between the numbers would be required, and not the actual metric values. Examples of such comparative analysis using the SCM will be given in chapter 5.

Soundness of the SCM

As described in [22], validation of a code complexity measure through the measurement theory cannot be justified, i.e.: there's no definitive way to prove if a complexity metric is correct or not. This is because each complexity metric measures different aspects of the program, with some being more important to the developer than the others (for example, it is sometimes more important that the space complexity will be as low as possible on the account of the time complexity, and sometimes exactly the opposite would be important, etc). However there are several **soundness** properties defined which, when satisfied by the measure, define a complexity measure as "sound", according to the study described in [33].

These properties will be described in details and applied to the metric, and the results will be compared to the applications of these properties on other well-known metrics. It is important to note, that it's been claimed [18] that these properties are too strict, and the properties which are found useful and accepted as sound in the real world (such as the CCN) do not satisfy all of them [18, 33]. Thus the results application of the properties on the metric defined in this work do not, in their own stand, prove that the metric is or is not "useful" for real world applications.

Definition of "Program"

The program definition for the measure evaluation will be similar to the one defined in [33], with certain additions required to take into the account the synchronization.

In [33] there are the following standard definitions for a "program" (quoted):

"Definitions:

1. ***Arithmetic expressions** are to be constructed using constants, identifiers and arithmetic operators, "+", "-", "*", "/", in the usual manner.*
2. *An **assignment statement** of the form: VAR <- EXP (where VAR is an identifier and EXP is an **arithmetic expression**).*
3. *A **predicate** is a Boolean expression having one of the forms: B1 = B2, B1 ≠ B2, B1 < B2, B1 ≤ B2 (where B1, B2 are either constant or identifier).*

A program is defined recursively:

*1) An **assignment statement** is a program body.*
*2) **IF** PRED **THEN** P*
*   **ELSE** Q*
*   **END***
*3) **IF** PRED **THEN** P **END***
*4) **WHILE** PRED **DO** P*
*   **ENDWHILE***
5) P
*   Q*

*Program of type 5) is said to be **composed from** P and Q, and will be denoted P;Q.*
*Program bodies of types 2), 3) and 4) will be referred to as **conditionals**.*

A program statement has the form:
PROGRAM*(variables)*
Where variables is a list of input variables.
An output statement has the form:
OUTPUT*(variables)*
Where variables is a list of output variables.

*Finally, a **program** consist of a **PROGRAM** statement, followed by the program body, followed by the **OUTPUT** statement."*

To this, we will add two more definitions, and change Definition 5). They add the concept of synchronization to the program model described above, and allow to use the extended model for a metric that addresses synchronization complexity. The additions are:

1. **Synchronized conditional ($\mathbf{PRED_s}$)**: This is a **conditional** which has a synchronization statement as part of the predicate. For example, access to a volatile variable as part of the predicate statement makes the predicate a "synchronized conditional".

2. **Synchronized assignment** ($\leftarrow_s$): This is an assignment that represents the "Volatile access" synchronization point type statement as the identifier, or as part of the expression. An example for a synchronized assignment would be a statement "`int result = fork();`" or "`sleep(1000);`" (Note that it is mentioned in the quoted definition, that an assignment statement is program body, but any statement can be considered assignment to some unused variable, which is discarded. For example, in C programming language, every function has to return a value, even if it is of a `void` type, which can be ignored when the function is actually called).

   It is worth noting that putting a synchronization statement like `fork` in a condition (like the C code "`if (fork() == 0)`") would be translated in a model to a "synchronized assignment" statement followed by a "regular conditional" statement. Accessing a synchronized variable will also be treated similarly – the mutual exclusion lock and unlock statements will be considered as "synchronized assignment" statements, while the actual access to the variable will be a regular statement or predicate.

3. **In addition to** the composition defined in type (5) above, the following kind of composition will be used: $P|Q$. It is called a parallel composition, and creates a program (called "parallel composed") with two components P and Q running concurrently.
   Launch of each of the components P and Q can be implemented as a synchronized assignment, that includes a synchronization statement of the "thread/task initiation" type. A program can only be considered as "parallel composed" if at it meets at least one of these conditions:
   - there's a common shared object on which P and Q synchronize at least once during their lifetime, or
   - they were both initiated by program(s) already considered as "parallel composed".

For all the evaluations below, every property which includes the sequential composition defined in [33], will also be evaluated using the parallel composition defined now.

When either interleaving or competition potential for the synchronized conditional or synchronized assignment is 1 (i.e.: no synchronization) – they will behave exactly as a conditional or assignment defined above.

For a given program *P*, the program complexity (value of the considered metric) will be marked as |*P*|.

Lemma 3: Synchronization points' competition potentials will not be reduced as the result of parallel composition.

Proof: Competition potential is the number of different entities competing for the synchronization point during the run of the program. Composing two programs (i.e.: running to programs in parallel) doesn't influence any other entity than the two composed programs.
Thus, if they don't compete for the same synchronization point – the other entities remain uninfluenced, thus continue to compete for the synchronization point, thus the competition potential won't change.
If the programs composed compete for the same synchronization point – the competition potential will have to grow.
■

Lemma 4: If P,Q are programs, then SCM(P;Q) ≥ SCM(P)+SCM(Q) (I.e.: |P;Q| ≥ |P|+|Q|).
In other words – if a program is composed by the sequential composition of two programs, its synchronization complexity will be at least as high as the sum of the original (i.e.: stand-alone) complexities of the composing programs.

Proof: Assume there are two programs *P* and *Q* so that: |*P;Q*| < |*P*|+|*Q*|). This means, that even if the competition potentials don't change (i.e.: both programs don't use threading or don't synchronize on the same data), the CCN of the main program is less than the CCN of the programs it's being composed of, which contradicts the definition of the CCN.
The competition potentials, by definition, cannot lessen as the result of composition, they can either remain the same (if the programs are not synchronizing on the same data), or grow (if the programs synchronize on the same data).
Thus, the SCM cannot lessen ⇒ the assumption is incorrect.
■

Lemma 5: If P,Q are programs, then SCM(P|Q) ≥ SCM(P)+SCM(Q) (I.e.: |(P|Q)| ≥ |P|+|Q|).
In other words – if a program is composed by the parallel composition of two programs, its synchronization complexity will be at least as high as the sum of the original (i.e.: stand-alone) complexities of the composing programs.

Proof: Assume the claim is not true, i.e.: there are programs *P*, *Q* so that |*(P|Q)*| < |*P*|+|*Q*|).

Since the CCN part of the SCM formula is not influenced by the parallel composition, it is true to say that the only change would occur in the synchronization dependency calculation. The P and Q are not changed by the parallel composition $\Rightarrow IP_{sp}{}^{CPsp-1}$ value for at least one of the synchronization points of at least one of the programs will be less in composition than when running stand-alone. The IP_{sp} of a synchronization point is a value that is not, by its definition, affected by the program being run in composition with another or not, thus the CP_{sp} (the competition potential) is the only value in the formula which is affected by the parallel composition (number of parallel executions competing over the given resource).

According to the assumption, the SCM of the composed program is less than the sum of the SCM's for the programs running standalone $\Rightarrow$ there is a synchronization point in at least one of the programs, for which the competition potential **lessens** as the result of the parallel composition, but according to Lemma 3, the competition potential **cannot lessen**, which contradicts the assumption $\Rightarrow |(P|Q)| \geq |P|+|Q|$.

■

Evaluating the Usability Properties of the SCM

The "soundness" properties are defined fully in [33], and are cited and evaluated here in the order of their presentation in that study. It has already been noted before, that a measure is not considered "sound" if it doesn't satisfy all the properties [18], and in fact some of the well-known measurements (like statements count, or the CCN) do not satisfy all the properties [33], yet they are by all means accepted as sound in the industry. However evaluating the properties will allow better assessment of the measure limitations and the measure values' impacts.

For programs without synchronization points, the SCM value is equal to CCN, which has already been analyzed in [33]. Thus, below we will only analyze programs which have synchronization points.

Property 1

The property: $(\exists P)(\exists Q)(|P| \neq |Q|)$

This property is clearly satisfied. For example, these two programs:

```
Program P
INPUT(V1, V2)
A ←s V1
B ←s A+V2
OUTPUT(B)

Program Q
INPUT(V1, V2)
A ←s V1+V2
OUTPUT(A)
```

The SCM of Q will be half of that of P, since (other variables equal) the interleaving

potential of the two synchronized assignments in P will be, combined, twice as high as of the single assignment in Q, thus:
$|P| = 2|Q|$.

■

Property 2

The property: Let c be a nonnegative number. Then there are only finitely many programs of complexity c.

It is shown in [33] that this property doesn't hold for CCN, which means it doesn't also hold for SCM for the trivial case when there are no synchronization points in the program.

However, for the case where the synchronization points exist and affect the value of the SCM for the program (i.e.: the interleaving and the competition potentials are both greater than 1), the situation is different.

In [33] it is shown that the statements count satisfies this property, i.e.: there're finitely many programs with statements count c, for any nonnegative c. Synchronization statements are subset of all the statements in the program, thus, for any nonnegative c, there are finitely many programs with synchronization statements count c (in a program with statement count c it is not possible to have more than c statements, synchronization or not). The synchronization statements' count directly affects the SCM value by its definition, but there are statements which do not affect the SCM – namely all the statements which do not result in branching of any kind of the execution path.

Thus, the SCM doesn't hold this property (similarly to the CCN [33]).

■

Property 3

The property: $(\exists P)(\exists Q)((P \neq Q) \wedge (|P| = |Q|))$

The SCM satisfies this property. For example the programs P and Q below:

```
Program P
INPUT(V1, V2)
A ←s V1+V2
OUTPUT(A)

Program Q
INPUT(V1)
A ←s V1
OUTPUT(A)
```

The SCM of the two programs is the same, even though the programs aren't identical.

■

Property 4

The property: $(\exists P)(\exists Q)((P \equiv Q) \wedge (|P| \neq |Q|))$

The SCM satisfies this property. The example given for property 1 is also valid here. Both *P* and *Q* in the example calculate the same function (*f(v1, v2)=v1+v2*), however their SCM values differ.

■

Property 5

The property: $(\forall P)(\forall Q)((|P| \leq |P;Q|) \wedge (|Q| \leq |P;Q|))$

This is Lemma 4, which has been proven for the SCM to be correct.

Also, the property has to be evaluated for the parallel composition:

The property: $(\forall P)(\forall Q)((|P| \leq |(P|Q)|) \wedge (|Q| \leq |(P|Q)|))$

This is Lemma 5, which also has been proven for the SCM to be correct, thus SCM satisfies this property.

■

Property 6

The property 6a: $(\exists P)(\exists Q)(\exists R)((|P| = |Q|) \wedge (|P;R| \neq |Q;R|))$

The property 6b: $(\exists P)(\exists Q)(\exists R)((|P| = |Q|) \wedge (|R;P| \neq |R;Q|))$

The proof is by example:

```
Program P
INPUT(V1, V2)
A ←s V1+V2
OUTPUT(A)

 Program Q
INPUT(V1)
B ←s V1
OUTPUT(A)
```

Assume $R = Q$, and x be the competition potential of the operation $\leftarrow_s$ (for which, since it's a synchronization point, the IP value is greater than 1).

$|P| = IP_{\leftarrow s}{}^{x-1}$

$|R| = |Q| = IP_{\leftarrow s}{}^{x-1}$

$|P;R| = |R;P| = 2*IP_{\leftarrow s}{}^{x-1}$, since the competition potentials don't change (the synchronization points are not related, thus programs running in sequentially don't compete on them).

$|Q;R| = |R;Q| = 2*IP_{\leftarrow_s}{}^{2x-1}$, since the competition potentials do change: Both Q and R compete for the synchronization point B on operation $\leftarrow_s$ in the same rate, thus th sequential composition doubles the competition potential (since it doubles the number of total threads competing).

$X > 1, IP_{\leftarrow_s} > 1 \Rightarrow 2*IP_{\leftarrow_s}{}^{2x-1} > 2*IP_{\leftarrow_s}{}^{x-1} \Rightarrow |R;P| \neq |R;Q|$ *and* $|P;R| \neq |Q;R|$.

Same property should also be evaluated with regards to the parallel composition (defined as an addition to the sequential composition used above):

The property 6c: $(\exists P)(\exists Q)(\exists R)((|P| = |Q|) \wedge (|(P|R)| \neq |(Q|R)|))$

The property 6d: $(\exists P)(\exists Q)(\exists R)((|P| = |Q|) \wedge (|(R|P)| \neq |(R|Q)|))$

Taking the same programs P, Q and R as above we will also receive the same results, since the analysis for parallel composition will also be influenced by the additional threads competing for $|(Q|R)|$ and $|(R|Q)|$ and will not for compositions of R and P.

■

Property 7

The property: there are programs *P* and *Q* such that *Q* is formed by permuting the order of the statements in *P*, and $|P| \neq |Q|$.

According to [11], neither the statements' count nor the CCN satisfy this property. Thus, for programs without any synchronization points, neither does the SCM.

However for programs with synchronization points, the analysis is different.

Example:

```
Program P
INPUT(V1, V2)
A ←s V1+V2
WHILE V1 > 0
     WHILE V2 > 0
          B ← B + 1;
          V2 ← V2−1;
     ENDWHILE
ENDWHILE
OUTPUT(B)

Program Q
INPUT(V1, V2)
WHILE V1 > 0
     WHILE V2 > 0
          A ←s V1+V2
          B ← B + 1;
          V2 ← V2−1;
     ENDWHILE
ENDWHILE
OUTPUT(B)
```

Let *IP* be the interleaving potential of the synchronized assignment to the variable `A`, and *CP* - the competition potential of that assignment. Assume *CP>1* and *IP>1* (i.e.: there's a potential interleaving).

The $|P| = SCM(P) = 1*(IP^{CP-1})+2$

The $|Q| = SCM(Q) = 1+2*(IP^{CP-1})$

$|Q| > |P| \Rightarrow$ the property holds for SCM provided there're synchronization statements in the program.

■

Property 8

The property: For any two programs P and Q such that Q is a renaming of P, $|P| = |Q|$.

The SCM satisfies the property since the naming conventions have no influence on the calculation formula

.■

Property 9

The property: $(\exists P)(\exists Q)(|P|+|Q| < |P;Q|)$.

For parallel composition: $(\exists P)(\exists Q)(|P|+|Q| < |(P|Q)|)$.

This is a stronger version of the fifth property.

Example given for property 7 is good for demonstration of the above: Both programs access the same synchronized variable *A*, thus when running **concurrently or sequentially** – in addition to the other entities they've been competing against, they will also be competing against each other, thus the competition potential of the synchronized access to A for both concurrently run programs will grow by 1.

Let *IP* be the interleaving potential of the synchronized assignment to the variable A, and *CP* - the competition potential of that assignment. Assume $CP>1$ and $IP>1$ (i.e.: there's a potential interleaving).

Before the composition:
The $|P| = SCM(P) = 1*(IP^{CP-1})+2$

The $|Q| = SCM(Q) = 1+2*(IP^{CP-1})$

$|P|+|Q| = 1*(IP^{CP-1})+2 + 1+2*(IP^{CP-1}) = 3+4*(IP^{CP-1})$

After the composition:

The $|P|' = SCM'(P) = 1*(IP^{CP-1+1})+2 = 1*(IP^{CP})+2$

The $|Q|' = SCM'(Q) = 1+2*(IP^{CP-1+1}) = 1+2*(IP^{CP})$

The $|P;Q| = |P|' +Q|' = 1*(IP^{CP})+2 + 1+2*(IP^{CP}) = 3+4*(IP^{CP}) > 3+4*(IP^{CP-1}) = |P|+|Q|$

■

Usability (Soundness) Properties Evaluation - Conclusion

As seen above, the SCM for programs with synchronization points satisfies 8 of the 9 properties of the complexity measure soundness, as defined in [33]. Comparing to other complexity measures evaluated against these properties in [33], the SCM is satisfies the most of them (for programs with synchronization points):

Property	Statements Count	CCN	Halstead's Programming Effort	Data Flow	SCM (for programs with synchronization points)
1	YES	YES	YES	YES	YES
2	YES	NO	YES	NO	NO
3	YES	YES	YES	YES	YES
4	YES	YES	YES	YES	YES
5	YES	YES	NO	NO	YES
6	NO	NO	YES	YES	YES
7	NO	NO	NO	YES	YES
8	YES	YES	YES	YES	YES
9	NO	NO	YES	YES	YES

Table 3: Metric soundness comparison table.

Chapter 3

Applying the SCM in Practice

Once all the theoretical parameters had been defined and properties tested, the time has come to apply the metric on the real world applications. The most important part of the metric is the synchronization points' interleaving potentials. These vary between various operating systems and implementation, as will be shown in this chapter, and thus provide the basis for comparison of the same code over different systems using the metric.

Once the interleaving potentials for all types of the synchronization points in use had been defined, the static code analysis tool implementing the metric can be applied to the code in question, to provide the required information on various levels. In the examples used for this work the CCCC [19] will be altered to implement the SCM and to provide the SCM values for each function in the C code under analysis.

Analysis of Interleaving Potentials

Interleaving Potentials are the core part of the metric calculation. The potentials are constant per each type of the synchronization points in the same system; however they may vary between different systems, as it will be shown.

In this chapter, each of the synchronization points' types defined in chapter 2 will be analyzed for various implementations, and the interleaving potential values will be given for each. Also, examples of synchronization patterns will be discussed and analyzed.

The analysis for the basic synchronization points' types will be provided with regards to the following non-preemptive scheduling algorithms:

1. Static Priority Scheduling.
2. Round Robin Scheduling.

The above two scheduling algorithms had been chosen because of their simplicity and popularity on the embedded and real-time software development market. Both algorithms are described in details in [27].

Other, more complicated scheduling algorithms exist, but those will not be covered in the analysis below. For example dynamic priority scheduling, combination scheduling algorithms (e.g.: combining priorities scheduling and round robin between tasks of the same priority), etc.

Non-preemptive priority based algorithms are in use in various real-time and/or embedded operating systems (for example VxWorks® or Embedded Linux® OS's).

Round Robin is one of the simplest basic scheduling algorithms which is easy to implement, and can be used on various applications which require their own thread-scheduling (for example when programming an embedded application which will run without an underlying operating system).

The scheduling systems are not usually being used in their "pure" non-preemptive implementation, and are frequently combined (for example Linux 2.6 kernel scheduler that combines priority and round-robin in its real-time scheduling algorithm [1]). However, for the simplicity of explanation and example calculations, the pure non-preemptive implementations will be considered in this chapter.

The analysis for the synchronization patterns will also be provided based on all, or some of the following distinctions:

a) C Pthread library (POSIX IEEE Std 1003.1c-1995 compliant) [14].
b) Unix (Open Group Specifications, POSIX IEEE Std 1003.1b-1993/1003.1i-1995 compliant) [28]
c) Standard ANSI C Implementations [16]
d) Microsoft Windows API [23].
e) Java standard thread and synchronized objects [8].

Interleaving Potentials Analysis for Basic Syncronization Types

Try-Lock synchronization point type

Synchronization statements of this type are intended to block the calling thread until a certain event occurs. It may not necessarily block, if the event has already occurred or non-blocking lock testing is requested.

This is the basic flowchart for this functionality:

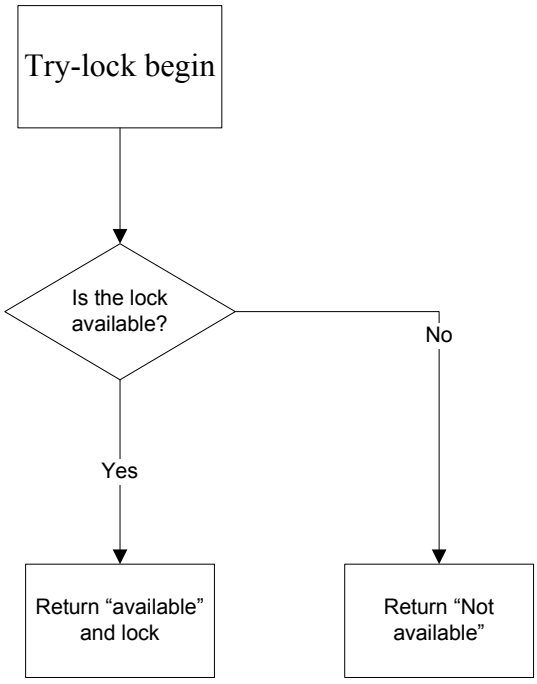

Fig. 2. Flowchart of the Try-Lock synchronization point type.

Interleaving potential analysis:

For the priority based scheduling: If the current thread is running, it will not be preempted as the result of the call, since the priority scheduling will let the highest priority thread running until a higher priority thread is ready. Having current thread running means it's a highest priority thread, and acquiring a lock will not release a higher priority thread. Thus, in non-preemptive priority based scheduling system, the synchronization point of this type will not have an interleaving potential greater than 1.

For round-robin based scheduling: The rationalization is the same – once the thread is running, it means it has its slice, and it will not be blocked until its time slice is over, regardless of the synchronization point. Thus, the interleaving potential will not be greater than 1.

Lock

Synchronization statements of this type are intended to block the calling thread until the requested resource is available.

The flowchart for the **lock** and **unlock** functionalities:

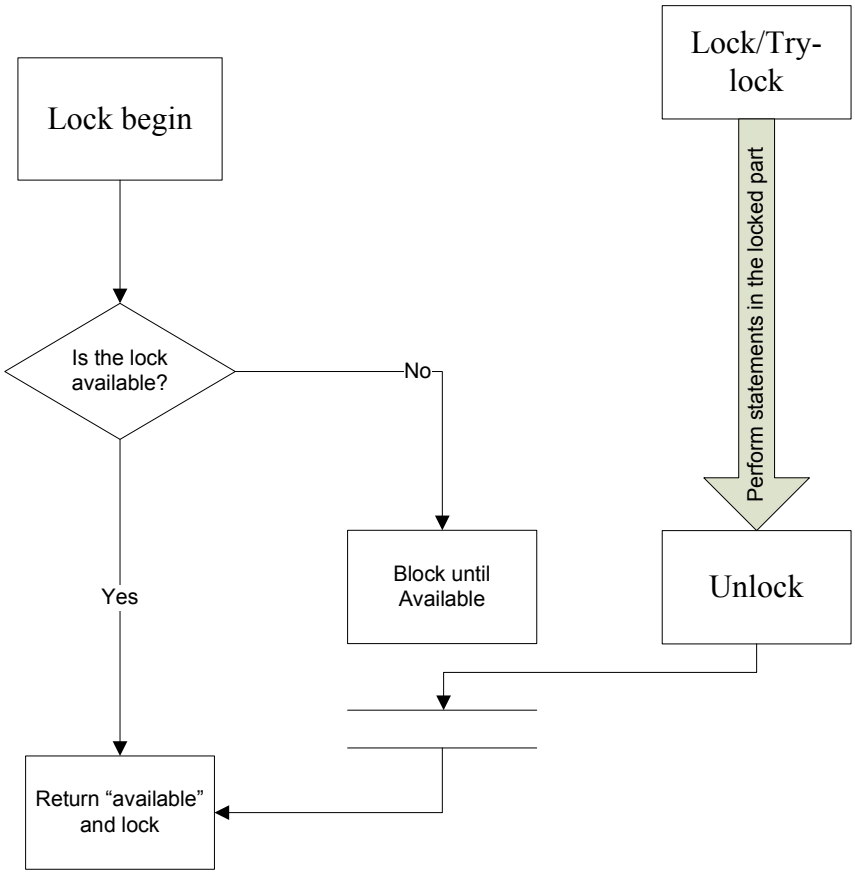

Fig. 3. Flowchart of the Lock and Unlock synchronization point types.

Interleaving potential analysis:

For the priority based scheduling: If the lock is released at the time of the call, the current thread will acquire the lock and will continue running, as in **try-lock**. If the lock is locked at the time of the call, the current thread will be blocked (will be in waiting state),

and the highest priority thread in ready state will be given the CPU. **The interleaving potential is the number of threads in the system**, because in the worse case, any other thread can theoretically be in ready state while the calling thread will be blocked waiting.

For round-robin based scheduling: For this functionality the analysis for the round-robin scheduling is the same, and so is the interleaving potential.

Unlock

Synchronization statements of this type are intended to release a lock acquired earlier by a **lock** or a **try-lock** call. The flowchart for this functionality can be seen under **lock**.

Interleaving potential analysis:

For the priority based scheduling: If the thread waiting on the lock has a higher priority – it will be released from waiting and allowed to run. Otherwise the thread waiting for the lock will be marked as "ready", and the current thread will be allowed to continue. Thus **the interleaving potential of this point is the amount of threads in the system with priority higher than the releasing thread** (i.e. potentially the amount of threads in the system, but in fact it is safe to assume that rarely a single lock will be shared between all the threads).

For round-robin based scheduling: The current thread will continue to run until it has exhausted the time slice given to it, while the thread waiting for the lock will be marked as waiting and will run the next time its turn comes. Thus **the interleaving potential is 1.**

Wait

Synchronization statements of this type are intended to block the calling thread until another thread notifies that the required event has occurred.

According to [5], a statement of this synchronization point type should be used in a loop in order to avoid known synchronization related bug pattern, thus increasing its de-facto synchronization impact (the CCN value in the SCM formula for the branch in which it is used).

The flowchart for the **wait** and **notify** functionalities:

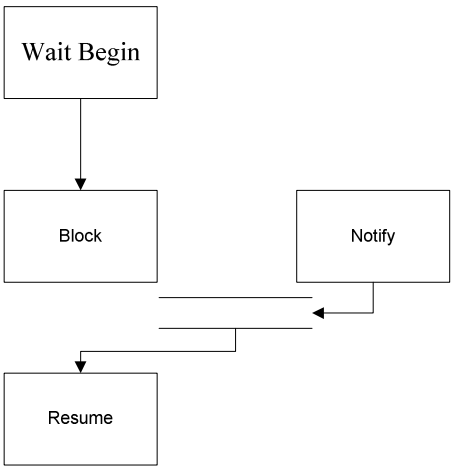

Fig. 4. Flowchart of the Wait and Notify synchronization point types.

Interleaving potential analysis:

The analysis is the same as for the **lock** synchronization point, since the functionality is basically the same except that **wait** will **always** lock.

Notify

Synchronization statements of this type are intended to release the thread waiting for an event (using the **wait** functionality). When "**notify**" is called, the waiting thread receives notification and is released from its block. The difference between the **unlock** and the **notify** points lay in the statements prior to the point itself: **unlock** synchronization point requires a **lock** or successful **try-lock** call before it can be used, whereas **notify** requires no prerequisites.

Interleaving potential analysis: The analysis for this point is the same as for the **Unlock** synchronization point.

Yield - Pass Control (Explicit)

Synchronization statements of this type are intended to pass the CPU control to a different thread.

JAVA

In Java, the explicit pass control synchronization point is a `yield` method call from `java.lang.Thread` class.

Interleaving potential analysis:

For the priority based scheduling: if the current thread is running – it means that it's the highest priority state available to run (this remark is based on the fact, that in this work we consider only static priorities; for dynamic priorities this property doesn't hold) . The synchronization point doesn't block, it only gives the scheduler a chance to reschedule, thus the running thread will remain the thread with the highest priority available to run, and will continue running. **The interleaving potential is 1.**

For round-robin based scheduling: The scheduler will reschedule, and may either return control to the current thread to finish its time slice, or give the control to the next thread according to the scheduling algorithm. Thus, **the interleaving potential is 2.**

Volatile access

Synchronization statements of this type are intended to access a variable which can be accessed by other threads without locking protection.

Interleaving potential analysis:

Regardless of scheduling algorithms, **the interleaving potential of this point is the number of threads that access the variable and are able to run concurrently**. For example, if the system runs on 5 processors with 20 threads being able to access the variable, the interleaving potential will be 5. The reasoning is that if the variable is accessed while no other thread can change it, there will be no **intentional** interleaving. Since volatile variables are usually used to share data with hardware components, there will be, usually, more than one processor which will be able to access it at a time.

Task/Thread initiation

Synchronization statements of this type are intended to create a new execution thread.

Interleaving potential analysis:

For the priority based scheduling: If the new thread has priority higher than the current – it will start running immediately, otherwise it will be marked "ready" and the current thread will continue to run. Thus, **the interleaving potential is 2.**

For round-robin based scheduling: The current thread will continue to run until it has exhausted the time slice given to it, while the new thread will be marked "ready" and will wait for its time slice. Thus, **the interleaving potential is 1.**

Synchronization Patterns Analysis

In this section several common synchronization patterns will be discussed and analyzed. Some of them are quite standard and have no differences with regards to the interleaving potentials, whereas others may be implemented differently on various systems the interleaving potentials of the same pattern may vary depending on the implementation.

Acquire/Release a Semaphore/Mutex

This pattern is the direct implementation of the "**lock**" or "**try-lock**" and "**unlock**" basic synchronization points, and the analysis provided.

On systems which don't have built-in support for such functionalities, they may be implemented using the "**wait**" and "**notify**" functionalities, through interrupts or other similar facilities.

Pthread library

Pthread library provides set of functions prefixed with "`pthread_mutex`" for mutexes' management, which implement the **lock, unlock** and the **try-lock** functionalities.

UNIX

In UNIX there are several functions for semaphore management and usage, which start with the "`sem_`" prefix. For example: `sem_wait, sem_trywait` and `sem_post` for "**lock**", "**try-lock**" and "**unlock**" accordingly.

Microsoft Windows API

There are several different WinAPI functions for semaphores. The simplests are `CreateSemaphore` and `ReleaseSemaphore` which are parallel to UNIX `sem_wait` and `sem_post` functions. As opposed to the POSIX complient functions, `CreateSemaphore` allows to create semaphores which can be locked multiple times. Simpler **Critical Section** constructs can also be used.

JAVA

Java provides the reserved word "`synchronized`" to mark the objects/methods which require to be locked for access. Every access to a synchronized method/variable performs the **lock** operation prior to the actual access, and the **unlock** operation once the access was finished. There's no **try-lock** construct built into the language.

Enter/Exit Critical Section

Synchronization statements that implement this pattern are intended to ensure a single thread to be executing the critical code section at a time.

The functionality can typically be implemented using semaphores; however some systems provide explicit support for this pattern.

Microsoft Windows API

In Microsoft Windows, there are functions for critical section management similar to other mutual exclusion mechanisms: Enter`CriticalSection` behaves like the **lock** synchronization point, `LeaveCriticalSection` behaves like the **unlock** synchronization point, and `TryEnterCriticalSection` behaves like the **try-lock** synchronization point.

JAVA

Critical sections can be implemented in Java using the `synchronized` block construct (example from [8]):

```
synchronized(syncObject) {
  // This code can be accessed
  // by only one thread at a time
}
```

The interleaving potential is the same as for the **lock** synchronization point type on the entry to the block, and the same as for the **unlock** synchronization point type on the exit from the block.

Send/Receive a Message

Synchronization statements that implement this pattern are intended to pass certain information from one thread to another. Sending a message can be implemented in several different ways:

Messaging can be implemented by the system developer using the synchronization patterns described earlier (semaphores and mutexes) and standard data structures (queues or cyclic buffers) protected by them. In this case the interleaving potentials will be calculated based on the chosen implementation details.

UNIX

There are several messaging options available in a UNIX system:

Using kernel-managed message queues (`mq_send/mq_recv` function), using sockets (`send`, `sendmsg` or `sendto/recv`, `recvmsg` or `recvfrom` functions) or using pipes (`write/read` function) – all the methods can be blocking or non-blocking, based on the queue/socket/pipe settings.

In case the blocking methods are used, the interleaving potentials are the same as for **lock** synchronization point, otherwise the same as **try-lock**.

Microsoft Windows API

There's no messaging support similar to the POSIX definitions in the Microsoft Windows API, but as for UNIX, sockets or files can be used for messaging. Also, shared memory objects can be used for that, using semaphore constructs to guard access. The closest thing to the message receiving functionality is `WaitForSingleObject`. The interleaving potential for this function is the same as for the **wait** synchronization point.

Also, as for UNIX, sockets can be used for messaging.

ANSI C Implementation

In Appendix A an example of messaging module is provided, taken under LGPL license from one of the open source projects available in the Internet. This implementation doesn't use operating system provided messaging mechanisms described above, and implements queue based messaging system, locking the access to the queue to a single thread at a time using the described above "Acquire/Release Semaphore/Mutex" pattern.

Adding a message to the queue is done using `soup_message_queue_append` call. The function uses the **lock/unlock** synchronization points to guard access to the queue, thus the IP value for the SCM calculation of the callers will be **the sum of IP values of the synchronization pints used in each branch of execution**.

The SCM of this function can be calculated as follows (assuming there are 2 threads in the system):

CCN = 2
IP for priority based scheduling:

IP = (IP(`g_mutex_lock`)+IP(`g_mutex_unlock`)) = 2 + 2 ***= 4***

IP for round robin based scheduling:

IP = (IP(`g_mutex_lock`)+IP(`g_mutex_unlock`)) = 2 + 1 ***= 3***

SCM = $2*4^{2-1}$=8 for priority based scheduling or $2*3^{2-1}$=6 for round robin based scheduling.

As it can be seen, using OS-provided functions like `mq_send`/`send` provide better SCM values for the implementation, since the synchronization is done internally by the kernel, and under the assumption of correctness, doesn't lead to multiple intentional interleavings.

Similarly to `soup_message_queue_append` call, the messaging module provided in Appendix A includes functions to remove (receive) messages from the queue: `soup_message_queue_first` and `soup_message_queue_next`. Similarly, these messages (which implement one way iteration over the queue) include locking functionality, and on each execution path of each of the functions, there's a **lock** and **unlock** synchronization point. The interleaving potential of each of the functions is the same as for `soup_message_queue_append`, by the same reasoning. It should be noted that the CCN value for the `soup_message_queue_first` and `soup_message_queue_next` is higher than for `soup_message_queue_append` because of the internal while loop, which means that the SCM value will also be higher.

Unprotected (Volatile) Shared Variables Access in ANSI C and in JAVA

Both ANSI C and JAVA provide reserved word `volatile` to mark variables that should

not be optimized by the compiler. This is used to mark variables which can be accessed concurrently from different threads of execution, specifically when sharing data with hardware (separate processors which access shared memory with the software systems) or with other threads, without using synchronization protection.

Thread/Task Initiation

Synchronization statements that implement this pattern are intended to create a new execution thread. When writing multi-threaded/multi-process system, the function calls used to create a new thread while leaving the current intact will perform as analyzed in the basic synchronization point section. However, there is a function that implements this pattern differently, and should be analyzed separately.

ANSI C

In the C language there are several functions defined which perform task initiations:

`system`: this function executes a shell command, and blocks the calling thread until the execution is finished. **The interleaving potential of this function is 2**: The call will always block until the requested command returns.

`exec`: the family of functions which **replace** the calling thread with a new one. These functions, if called, have **interleaving potential of 1**: If the call succeeds then the new thread will run, if the call fails – the old one.

Pass Control (Implicit)

Synchronization statements that implement this pattern are intended to pass the CPU control to a different thread, however it is done implicitly. A common occurrence of this pattern is using a blocking system call such as "`sleep`", which puts the calling thread into a "waiting" state, and allows running other "ready" threads in the system. Thus, regardless of the scheduling, the interleaving potential is 2, since the scheduler can either return control to the calling thread or give it to the next thread available, if the call blocks.

Interleaving Potentials Analysis for Syncronization Types and Patterns - Summary

The table below summarizes the interleaving potentials for the synchronization types and patterns analyzed above.

Synchronization Point/Pattern	**IP when Priority based scheduling (standard system call)**	**IP when Round Robin based scheduling (standard system call)**	**IP when non-standard implementation is used, or exceptions**
Try-Lock	1		
Lock, Critical Section Entry Pattern	Number of threads in the system		
Unlock	Number of threads with priority higher than the calling, accessing the lock	1	
Critical Section Exit Pattern	Number of threads with priority higher than the calling, accessing the critical section	1	
Wait	Number of threads in the system		
Notify	Same as Unlock, but not higher than number of threads accessing "Wait".		
Pass Control (Explicit)	1	2	System calls can be treated as such synchronization point (implicit), with IP=2.
Volatile Access	The higher of: number of threads accessing the data or number of processors in the system.		
Task/Thread initiation	1	2	
Message Send/Receive Pattern	Same as Lock		IP(Lock)+IP(Unlock)

Table 4: Synchronization Types and Patterns Analysis Summary – Interleaving Potentials

Competition Potentials Analysis for Basic Synchronization Types and Patterns

Data Dependant Competition Potentials

For all the synchronization types and patterns which rely on specific data, the competition potentials are the number of threads using that data.

For example: **Lock, Unlock, Try-Lock, Wait, Notify, Volatile Access** types and **Semaphore/Mutex/Critical Section** related patterns all have **competition potentials equal to the number of threads using the data** (using the lock/semaphore/mutex or accessing the code in the critical section, or using the signal used in the **wait-notify** construct (either waiting or notifying), or using the message queue (either sending or receiving), etc).

Worst case would be the total number of threads (if, for example, static code analysis doesn't allow analyzing the exact number of threads which will access certain synchronized data variable).

Data Independent Competition Potentials

Pass Control

When the control is passed, whether explicitly or implicitly, the thread given the CPU can be any of the threads available in the system, thus **the competition potential is the number of threads in the system**.

Task Initiation

Task initiation is essentially a pass control construct, except that each task initiation increases the following call competition potential. Initiating all the threads in the system in the initialization stage is a common practice in embedded software implementations with limited resources. Such systems acquire all the needed resources in the initialization stage, including threads initiations and memory allocations, in order to avoid resource deficit during the life time of the running system. In this case the competition potential for each thread initiation synchronization point will increase by 1 every call, so that for each new thread t, the CP_t will be $CP_{t-1}+1$, with $CP_0 = 0$ (thus $CP_t = i$).

A common practice is to initiate threads in one loop, in a code that looks similar to this:

```
int i, maxTasks = n; /* n is the total number of tasks to be
created */
for (i = 0; i < maxTasks; i++)
    createTask(i);
```

In this pattern usage the *CP* of the "createTask" synchronization point changes at each iteration, but for the static analysis the average value can be considered (calculated as the sum of all the numbers in the range, divided by the number of iterations of execution of this code). So, for such case, the *CP* for each call will be $\frac{1}{n}\sum_{t=1}^{n}(t+1) = \frac{n+1}{2}$, where n is the total number of threads.

In another case, when thread initiations and exits are performed in an arbitrary order (e.g. Web Server which starts a separate thread on each connection, and exits when the connection ends), the competition potential would be the ***maximum number*** of threads allowed to run in the system concurrently (i.e. n).

Competition Potentials Analysis for Synchronization Types and Patterns - Summary

The table below summarizes the interleaving potentials for the synchronization types and patterns analyzed above.

Synchronization Point/Pattern Type		**Competition Value (as function of *n* – number of threads in the system)**	**Comments**
Data Dependant		Number of threads in the system with the access to the data (can be used together with a coupling metric which can provide the needed data to assess the number of additional modules accessing the data as part of the static code analysis, or by prior design knowledge).	***O(n)*** – in worse case or when cannot be undetermined, all the threads are assumed to be able to access the variable. Since the metric is used when the system design is known, the worse case can and should be avoided.
Data Independent	Pass Control	***n*** – the number of threads in the system.	
	Thread/Process Initiation	Depends on the system design: ***n*** – the maximum number of threads in the system, if the threads are being initiated and terminated interchangeably; $\frac{n+1}{2}$ – when all the threads are being initialized before any thread has a chance to terminate, or never terminate.	The case where the threads are being initiated and never terminate is very common in embedded/reactive systems.

Table 5: Synchronization Types and Patterns Analysis Summary – Competition Potentials

Chapter 4

CCCC Introduction

The CCCC tool [19] is a tool developed by Tim Littlefair as part of his advanced graduate studies at the Edith Cowan University. The CCCC stands for C and C++ Code Counter, and as the name suggests – this is an utility that gathers metrics for C and C++ (and also Java) code.

The tool is distributed as an open-source software under the GPL license, through the "Sourceforge" project.

The tool is based on a command-line interface, its input is a C/C++/Java file (for the purpose of this thesis, C and C++ examples are used), and the output is a directory with a set of HTML pages which include the metrics calculated for the file, on the file, class and function levels with source cross-reference.

The tool implements measurement of the following metrics [18]:
Procedural metrics:

1. McCabe's Cyclomatic Complexity Number
2. Lines of Code
3. Lines of Comments

Object Oriented Design (OOD) Metrics:

1. Depth of Inheritance Tree
2. Number of Children
3. Coupling Between Objects
4. Weighted Methods per Class

Structural Metrics:

1. Fan-In
2. Fan-Out
3. Information Flow.

According to [18], a weak consensus was achieved between the participants of the tool evaluation experiment regarding the positive value of the procedural metrics the tool provides, while the OOD and the structural metrics received marginally negative responses.

Although the experiment conducted by Dr. Littlefair had shown that using the tool-provided metrics as part of the code review process doesn't change the process outcome significantly, it is reasonable to assume that the procedural metrics, especially the McCabe's CCN, can also be used as part of the testability evaluations by the QA and QC groups, and that the direct connection between the CCN and the test coverage, as shown above, is of benefit for these groups. This assumption is not put to test as part of this work, and is left to be tested as part of a future work on this topic.

In this thesis, the CCCC ability of measuring the McCabe's Cyclomatic Complexity Number (CCN) is used to implement the measurement of the SCM as defined above, as an

additional procedural metric the tool will provide. It is assumed (although the task to prove or disprove this assumption is left for future work) that the SCM will prove itself beneficial in the similar manner as the CCN is, for the purposes of testability evaluations of concurrent software.

The CCCC also provides totals per module analyzed. In case of the CCN (tagged in the CCCC reports as "MVG"for historical reasons) counter and the SCM counter, the totals are the sums of the relevant values for all the functions in the module.

CCCC Implementation

The CCCC is implemented in C++, using object oriented design and programming. The grammar for the C++ and Java languages is compiled using the PCCTS Antlr and Dlg tools.

Each token found by the parser is passed to the `CCCC_tok` class.

The McCabes, Lines of Code and Comments metrics are calculated directly in the parser. The increase of the CCN is triggered by the C++ reserved words:

```
break
for
if
return
switch
throw
while
```

Each encounter of any of the above reserved words triggers an increase of the CCN (the MVG counter) by one. As shown in chapter 6, the result is not necessarily correct in all cases, but is close enough to be usable for the purposes of demonstration.

The module `CCCC_utl` includes the implementation of the class `ParseStore`, which handles the metric counting.

The module `CCCC_met` is responsible for metrics' calculations and representations in the final reports (including excessive values marking).

The module `CCCC_htm` is responsible for creating the reports, including formatting.

The module `CCCC_tok` is respnsible for the source code parsing and the token handling.

There are additional modules that are responsible for interim data storage, specific metrics' calculations, etc., which were not changed for this work, and their source code is available on the internet.

CCCC Implementation Changes

The changes were made on the latest available stable version (version 3.1.4) that can be downloaded under the GPL from the Sourcforge project site [19]. Listings of all the non-trivial additional code added for the purpose of implementing the SCM are in the Appendix B of this work. Original sources are freely available over the Internet, for reference.

The trivial changes include the changes required for additional metric representation in the final report in the modules `CCCC_htm` and `CCCC_met`. The changes are made so that the SCM metric values will be represented immediately after the CCN in the reports.

The main non-trivial addition to the original code is the new `CCCC_ScmManager` class and the existing `ParseStore` class.

The SCM was implemented for and tested on C programs.

The `SCM Manager` class

The `CCCC_ScmManager` class is the core functionality added to the CCCC as part of this work. The class holds all the data needed for the SCM calculation: the competition potential, the interleaving potentials for each type of the synchronization points, the table which allows translating a specific code statement to an abstract synchronization point, and keeps track of `volatile` variables.

The class is implemented as a static C++ class (i.e.: the methods can be called directly without instantiation of the class).

The class has to be initialized prior to starting the calculation. The initialization will be done by a call to the `Initialize` method. In order to avoid the need to track the state of the class object (which is used as singletone) elsewhere, the Initialize method can be called by the `ConsumeToken` method (which will be described later) on the first call.

Initialization

The `Initialize` method calls `InitIds` and `InitPotentials` methods. These methods initialize the values for the interleaving and competitional potentials, and fill the list of the synchronization points' tokens.

Calculation

The `ConsumeToken` method is called by the `ANTLRToken::CountToken` method that scans the tokens stream. For each token in the stream, the `ConsumeToken` method checks whether it exists in the table of synchronization points' tokens (loaded from the initialization file as described above).

If the token is found in the table then the *IP* and *CP* values for that token and the current value of the "MVG" counter will be added to a list of synchronization points' values

encountered in the current function.

When the nesting level is decreased to 0 (i.e.: end of the function is reached, in the C code), the method will call the CalculateScm method of the SCM_Manager class, that calls the IncrementSCM function of the ParseStore class for each synchronization point stored.

For each of the *IP*'s/CP's pair saved for the current nesting level, the formula calculations will be performed, as defined in chapter 2 ("The Formal Definition of the SCM"). The CCN for the calculations will be taken from the current value of the "MVG" counter, which represents the cyclomatic complexity metric in the CCCC. The difference between the values of the "MVG" counter at the time the synchronization point was encountered in the code, and the end of the function, is taken as the CCN_{sp} value.

The CCN required for the calculation should be the CCN_{sp}, the cyclomatic complexity of the smallest sub-graph that includes the synchronization point. The CCCC is not able to calculate the CCN_{sp} as defined for all the possible cases (it is also not able to calculate CCN for all the possible cases as well, see chapter 6 for examples of such cases).

However, the calculation of the CCN (through the "MVG" counter) is implemented by incremental calculation based on the binary decision statements encountered during the single pass of the code analysis (for example, "`if`", "`switch-case`", "`for`", and other C constructs).

Such calculation provides intermediate values allowing calculation of the CCN_{sp} value, as it was defined in this work, when the code parser reaches the end of the function in which the synchronization points were found. The problems which prevent the CCCC to calculate the CCN correctly in certain cases have the same influence on the CCN_{sp} calculation as well (several examples are given in chapter 6). These corner cases were not dealt with during the work on this thesis.

The calculation is done in `ParseStore::IncrementSCM` member function (see below).

The `ParseStore` class

The `ParseStore` class is responsible for storing the metric values during the parsing process. It has `IncrementCount` member function which increments any given measure during the parsing and stores the current value.

The `IncrementCount` member function was changed so that it would increase the SCM measure each time the CCN is increased (since, by definition, unless a synchronization point is found, the SCM tracks the CCN).

A new member function, `IncrementSCM`, was added. This function performs the actual calculation of the SCM value based as described above, and stores the calculated value in the existing CCCC data structures, for further processing and formatting as HTML output (existing CCCC functionality).

Chapter 5

In this chapter several examples of usage of the changed CCCC tool will be provided. The actual output of the CCCC for each of the modules discussed is provided in appendix C.

BusyBox HTTP Server Analysis

"Busybox" is a Linux distribution targeting embedded systems' developers. This Linux distribution is characterized by small size, achieved by only bundling the minimal software required for running the device. Many common UNIX utilities have been rewritten to provide smaller and optimized replacements, sometimes using single executable for various related utilities. Busybox distribution is used in the industry in various Linux-powered embedded devices, such as cable and satellite TV set top boxes, network devices, home appliances, etc.

For this work, the HTTP server implementation in the Busybox package was chosen as a good candidate for analisys. The reasons to chose this particular utility are:

1) It is a stand-alone application with many synchronization points
2) The Busybox implementation has been improved significantly in the past years, so it is a good candidate for comparative analysis
3) This particular implementation is used in embedded devices where task synchronization problems are of high importance, while the schedulers are of a simpler design (for example simple priority based schedulers can be found in applications using this implementation).
4) There are many other similar HTTP server implementations which can be used for comparison (I chose to compare the BusyBox implementations with the IKI implementation, as described below)..

Busybox httpd.c file version used is 1.35 (dated Oct. 6, 2004), and the current (as of May 25, 2009) version, both are available under GPLv2 license from the BusyBox project at http://www.busybox.net/.

The source listings are available in appendix A of this work.

The analysis was performed based on an assumption of a single instance of the http daemon running, with a single connection (i.e.: competitional potential is 2: the main task and the listener task forked from it). Such implementations are used, for example, in some embedded devices with HTTP configuration interfaces.

The synchronization points defined (based on the scheduling constraints described in the table 4 above):

Synchronization Point	Type	IP
`create_and_bind_stream_or_die`	SCM_LOCK	2
`safe_read`	SCM_LOCK	2
`shutdown`	SCM_NOTIFY	2
`signal and sigaction`	SCM_NOTIFY	2
`send`	SCM_NOTIFY	2
`select`	SCM_WAIT	2
`read, write, full_write` and `full_read`	SCM_ NOTIFY	2
`accept, listen` and `xlisten`	SCM_WAIT	2
`fork` and `execv`	SCM_TASK_START	1

Table 6: Http Server Analysis Synchronization Points Values

The results, per function (only functions with synchronization points listed):

Functin Name	Old Version (2004)		New Version (2009)		Comments
	MVG (CCN)	SCM	MVG (CCN)	SCM	
`getLine`	8	24	9	25	In the new version – renamed to `get_line`
`handleIncoming`	78	466	94	288	In the new version renamed to `handle_incoming_and_exit`
`httpd_main`	19	39	16	38	
`log_and_exit`	N/A	N/A	1	5	A new function
`miniHttpd`	10	53	3	12	
`openServer`	1	9	4	16	
`sendCgi`	70	607	28	29	Changed to `send_cgi_and_exit`
`sendFile`	18	38	N/A	N/A	Changed to `send_file_and_exit`. Due to the code style used in the new version and the CCCC limitations, reliable calculations couldn't be performed.
`sendHeaders`	13	15	20	42	Changed to `send_headers_and_exit`
`sighup_handler`	1	7	--	--	No synchronization points in the new version
Module totals	374	1416	326	615	

Table 7: Busybox httpd.c Analysis Results

Using the results of analysis of the new and the old version, we can compare the versions with regards to their synchronization complexity.

We can see from the table 7 above, that while in some cases the CCN for a function become higher, the SCM become lower or changes insignificantly due to a more careful usage of synchronization points within the code (for example – following the simple guidline of limiting the CCN per function to below 20 [13, 15] will probably do marvels to the SCM values and the overall testability of the concurrent function as well).

A good example would be the function "`handleIncoming`". In the old implementation it had the CCN value of 78 and the SCM value of 466. In the new implementation, the CCN is 94 and the SCM – 288. We can see that although the CCN got higher, the SCM became considerably lower (improvement of over 38%). The listing of the function in the old implementation starts at the line 1481 of the file listing, and in the new implementation its on line 1769 (called "`handle_incoming_and_exit`").

The high difference in the SCM values in a function which basically implements the same functionality is explained in this case by the removal of the error handling from the `handleIncoming` function in the old implementation to the `send_headers_and_exit` in the new one (`sendHeaders` in the old implementation). This caused the increase of 27 in the SCM value from 15 to 42 in `send_headers_and_exit` but a dramatic decrease of 288 in the SCM value of the function `handle_incoming_and_exit`.

Overall the comparison shows that the improvements made over the time between the two versions improved significantly the testability of the program both in the "classical" way (the CCN per module reduced drastically, and especially for functions with large values like `sendCgi`), and also with regards to the synchronization complexity (as shown in example of the `handleIncoming` function, the changes in the SCM are not derived directly from changes of the CCN, as one might claim, but are actual changes in the usage of the synchronization mechanisms and code reorganisation).

IKI HTTP Server Analysis

Additional analyzed HTTP server implementation is the one written by Tero Kivinen; it can be found on the finnish site called IKI: http://www.iki.fi/iki/src/httpd.c.
It is published as is, and the source is provided in Appendix A with the copyright notice allowing the reprinting and the redistribution.

This version of the HTTP server is in use by the IKI site (www.iki.fi), and probably others.

For the analysis of this module, the same competition and interleaving potentials were used, for the same synchronization points, as for the Busybox implementations (see table 6).

The results are as follows (only functions with synchronization points listed):

Function Name	MVG (CCN)	SCM
`do_write`	10	22
`http_server`	29	75
`main`	25	90
`new_connection`	12	34
`open_service`	7	13
`read_data`	8	26
`read_page`	9	15
Total per module	312	487

Table 8: IKI httpd.c Analysis Results

Comparative Analysis

Above, the SCM and CCN values were measured for two versions of the BusyBox HTTP Server implementation, and for an additional (IKI) implementation of the same functionality.

The SCM number by itself may not be useful (especially with large *IP* and *CP* values), but it is very useful when we want to compare different implementations of the same functionality. For example, we may want to know the SCM values for the modules, and the additional complexity we get when using SCM versus the classic CCN.

In this case we're comparing three implementations of an HTTP server, all three assumed to be running in the same environment (percentage of the SCM addition is calculated as ((SCM/CCN) -1)%. In the table below we can see the totals of the CCN, the SCM and the SCM addition to the totals (sum of metric values for all the functions):

Module	CCN	SCM	SCM addition
Old BusyBox	374	1416	278%
New BusyBox	326	615	87%
IKI	312	487	56%

Table 9: HTTP Servers Comparison

It can be clearly seen that the synchronization complexity of the IKI implementation is less than that of the busybox implementation, both nominally and realtevly to the total CCN of the module.

This suggests possibility of more efficient and careful usage of system calls and inter-process communication mechanism in the IKI implementation, and can be used as a basis for review of the implementations in order to find better ways (or identify better patterns) of doing things in one, learning from the other.

For example: the function `new_connection` in the IKI implementation and the function `handle_incoming_and_exit` in the (new) BusyBox implementation handle the new HTTP request.

However, the IKI implementation has much lower CCN and SCM values, since the parsing of the command portion of the request is not done in the scope of the function, thus reducing the potential of synchronization-related (and other) problems in the function which handles the actual connection.

Additional interesting observation seen in Table 9 is that the SCM doesn't grow exponentially, as might be expected based on the metric formula. This is because of the decoupling between the modules and very low de-facto competition potential for each synchronization point. For the analyzed HTTP servers' implementations, the *CP* value was *2*, since all the implementations used two tasks: the main task and a listener task forked from it

Conclusions

The goal of this work was to show that it is possible to evaluate the impact of using concurrent programming patterns (such as mutual exclusions, accessing shared data, creating new threads and processes, etc.) on the programs' complexity.

In this work a metric was defined that can represent this impact with relation to the amount of unique execution paths required to cover the expected interleavings on the program language level. Thus, this number also represents the amount of unique tests required for proper coverage of these paths (based on the known branching and synchronization coverage models described in chapter 1). This is important for example, when trying to reach full coverage based on the concurrency coverage criteria with tools like ConTest [5] or CHESS [24]. The CHESS tool is especially relevant since it attempts to systematically cover all the possible interleavings based on the synchronization statements divided by types, similarly to the classification described in this work. Thus SCM, as it is defined in this work, can be used for the estimation of the effort required for achieving the full coverage when using this tool.

In the comparative analysis done in the work for various implementations of the same concurrent programming patterns, we could see the direct and specific benefits that a developer could have gained, had he been using the SCM as part of the development process.

The conclusion is that the metric provides a valuable information for the developers and testers when considering usage of different implementations of the same functionality in their applications, or usage of the same implementation in different operation environments. This information is useful for assessment of the testability, the risk potential (the higher the SCM – the higher the risk and potential of different problems to occur), and the quality of the product.

Feasibility and Usability

The metric is defined by a formula which is exponential. However, the exponent, which, for every synchronization point analyzed represents the number of threads competing for it, is usually not large.

In the real life applications, the tendency is towards loose coupling between modules, thus creating very little dependencies and synchronization amongst threads in the system. For example, in [15] the requirement is that “[the] Source code should be developed as a set of modules as loosely coupled as reasonably feasible”. Similar requirements exist throughout the industry, and are measured by CCCC and other similar tools. Therefore, as it has already been mentioned before in this work, it can be assumed that most of the real-life applications have only handful of threads competing over the same resources, and thus require synchronization. For example, in [5], 16 classes out of 575 (3% of all classes) having synchronization primitives is considered reasonable. Although it is not clear from

[5] how many threads are in the system, it is reasonable to assume that in such a large-scale system, due to the loosely coupling, the amount of threads sharing data will be small relatevly to the total number of the threads. In [24] it is also shown that there are very limited amounts of threads for even large scale projects.

The example analysis performed in this work on a real life HTTP server applications has shown that the SCM provides valuable and useful information. On the actual real life HTTP server implementation, the SCM provided both an estimation of the test effort to achieve the coverage based on the branching/synchronization criteria, and a comparison between various implementations which can be used to suggest a better option to choose. In this real life program, the competition potential is as low as it can possibly be for a multi threaded application, and this program is not an exception, but rather the opposite – one of the reasons for it to be chosen is its being widely spread (BusyBox Linux distribution is very popular in the embedded devices world). Also, many other applications are built in a similar manner (in the BusyBox distribution additional server and client protocol implementations such as FTP, TFTP etc, are built with a similar architecture).

Although there are possible cases where the SCM will provide extremely large values which by themselves will appear not to be useful, in fact such values can still be useful and suggest that there are coupling issues in the code under analysis which should be checked. Even then, comparative analysis using the SCM to choose the better option out of several “bad” (with high SCM values) options can be performed.

Thus the conclusion is that the SCM is a measure which provides usable results for real life applications, and apart from it direct uses as test effort estimation and comparative analysis, it also provides an indication for the adequacy of the coupling between the threads in the system.

Chapter 6

Future Work

Coverage Models' Suitability

The SCM was developed based on the definitions of some of the coverage models described in Chapter 1. However, the implementation in Chapter 4 doesn't allow us to draw a direct line between the actual calculations of the SCM as implemented and the coverage criteria described in Chapter 1. The reason is that based on the static code analysis it is hard to identify and to order various types of accesses for the different variables. For example, during the static code analysis it is hard to identify the write-write access across a pair of threads, when it is not necessarily known in advance (i.e. before the execution) what threads are going to be there, and what code are they going to execute. Further development needs to be done on the implementation of the SCM in order to provide values precise enough to be used in pair with coverage calculations based on one of the models described in Chapter 1. This additional development is not part of this thesis, and is left for future work.

Real World Effectiveness

As part of future work it is left to evaluate the effectiveness of the SCM and its ability to prove itself useful in the real world of applications, for real-world quality control and quality assurance personnel.

It is assumed in this work, that having a metric which allows comparing the impact of the synchronization on the overall program complexity between various possible solutions may be beneficial for software developers and testing engineers. However, evaluation of the correctness of this assumption was not in the scope of this work and should be done as part of the future work left on this topic.

Implementation

The current CCCC implementation is very limited and is intended for demonstration and proof-of-concept usage rather than actual industrial application.

The current implementation (that was used for the SCM implementation) has limited language parser, and is limited to working on one file at a time, and cannot follow execution flows between different files or projects. It also cannot always extract the pre-compiler macros ("`#define`"s) or type substitutions correctly thus sometimes being unable to detect correct type usages or execution flows.

Below are several examples of code which would lead to incorrect metric calculations by the CCCC (in its current form).

If run on the following code, the CCCC will not be able to identify the `hidden_volatile` type variable "`test`" as volatile or as a pointer:

```
typedef volatile int * hidden_volatile;
hidden_volatile test;
```

In the current CCCC implementation, this would affect the SCM calculations and provide incorrect result.

If run on the following code, the CCCC will not detect execution flow branching, thus calculating the CCN and the SCM incorrectly:

```
#define DO_BRANCHING_HERE(x) \
 if (x) { select();} else {fork();}

void foo(bool some_flag){
    DO_BRANCHING_HERE(some_flag);
    return;
}
```

The CCCC will return CCN=1 and SCM=CCN for the function `foo`, both incorrect. If there will be no "`return`" statement (which is valid for functions with return type of `void`), the CCN reported would be 0 (for SCM this bug was fixed if there are synchronization points, but not if the SCM follows the CCN).

Another example was given just prior to defining the synchronization metric, in chapter 2. The example is :

```
#include <unistd.h>
#include <stdlib.h>
int main(){
     int pid;
     if ((pid = fork()) != 0) {
          sleep(1);
          printf("\nChild Process\n");
     }
     else {
          sleep(1);
          printf("\nParent Process\n");
     }
     return pid;
}
```

As it was mentioned in chapter 2, the condition in the "`if`" statement affects the CCN of the program, had it been single threaded. However, that is misleading, since for each of the tasks (the parent and the child), there's no branching option: for the parent task the "`if`"

statement will **always** evaluate to false, while for the child task the same statement will **always** evaluate to true. Thus, in fact there's no branching in the execution paths of the processes in question. Analyzing the code statically by tools like CCCC would provide incorrect calculations results for this code.

Additional example of problematic CCN calculation is in the HTTPD Busybox implementation of the function `getLine` analyzed below.

Here the root cause for the incorrect calculations is in "erratic" coding with many exit points and branching constructs that don't create actual branching (like the "`while (1)`" infinite loop) combined with "`break`" constructs which act as "`goto`" statements.

This is considered bad coding style, and in the MISRA C coding standard, for example [13], it is advised to keep a single exit point (single "`return`" statement, as opposed to the `getLine` implementation below), and avoid infinite loops with "`break`" statements (like the "`while (1)`" loop in the new get_line implementation below).

Thus the CCCC can help to detect coding style problems indirectly by showing the CCN values higher than expected for such trivial functions, but this of course is a side effect of the CCCC simple and limited implementation.

The old version:

```
static int getLine(void)
{
     int count = 0;
     char *buf = config->buf;

     while (read(config->accepted_socket, buf + count, 1) == 1) {
          if (buf[count] == '\r') continue;
          if (buf[count] == '\n') {
               buf[count] = 0;
               return count;
          }
          if (count < (MAX_MEMORY_BUFF-1))      /* check overflow */
               count++;
     }
     if (count) return count;
     else return -1;
}
```

The new version:

```
static int get_line(void)
{
     int count = 0;
     char c;

     alarm(HEADER_READ_TIMEOUT);
     while (1) {
          if (hdr_cnt <= 0) {
               hdr_cnt = safe_read(STDIN_FILENO, hdr_buf, sizeof(hdr_buf));
               if (hdr_cnt <= 0)
                    break;
               hdr_ptr = hdr_buf;
          }
          iobuf[count] = c = *hdr_ptr++;
          hdr_cnt--;

          if (c == '\r')
               continue;
          if (c == '\n') {
               iobuf[count] = '\0';
               break;
          }
          if (count < (IOBUF_SIZE - 1))      /* check overflow */
               count++;
     }
     return count;
}
```

Following is the execution flow graph (as a single threaded program) of the function `getLine` (the synchronization point is marked yellow):

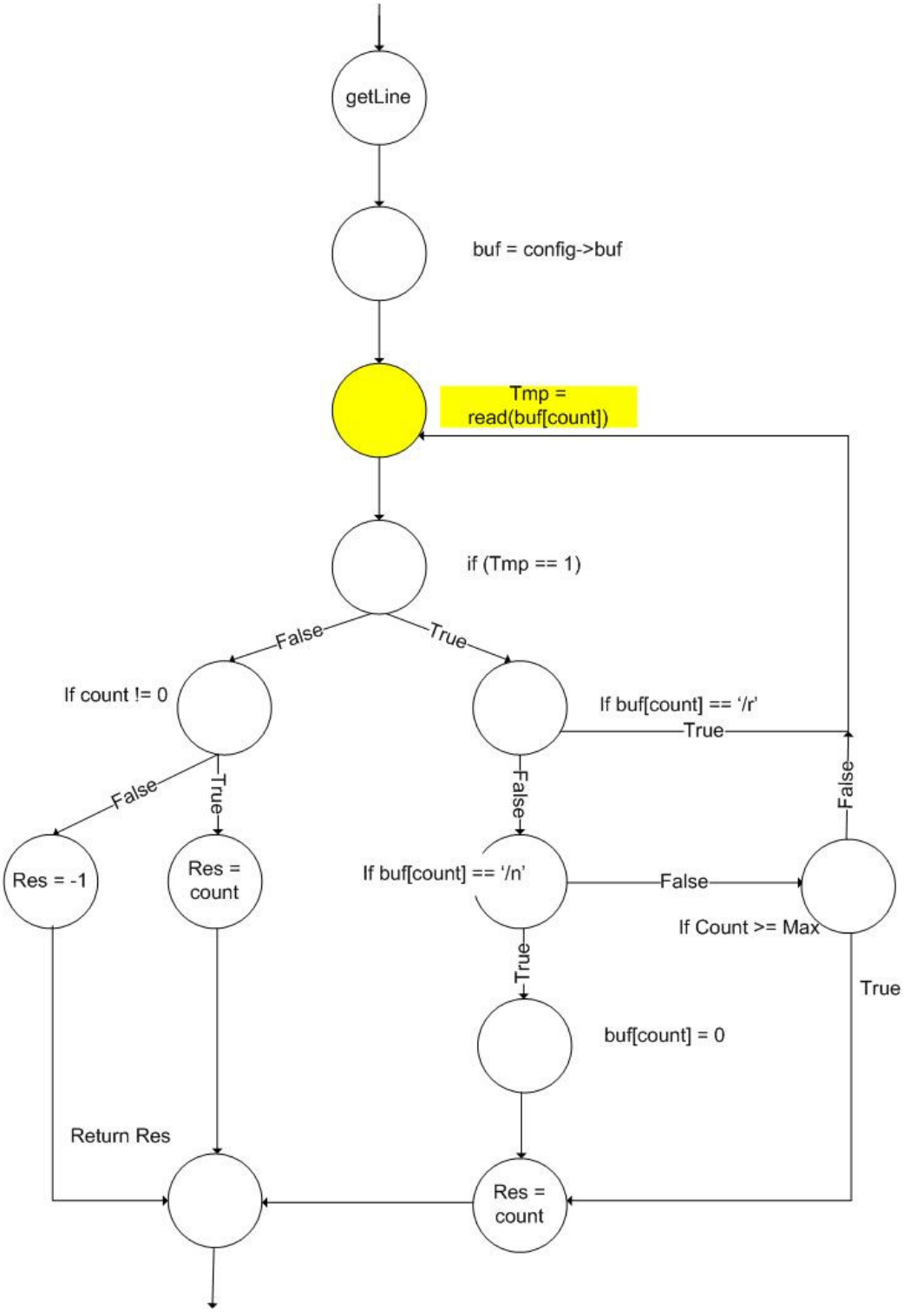

Fig. 5. Execution paths' graph for the getLine function.

The CCCC calculated the CCN for the first function to be 8, and the SCM to be 24. Analyzing the function manually we can see that this is incorrect. The SCM counter is

incorrect because of the wrong CCN calculation, which in turn is skewed by the "`return`" statement on the line 11.

The correct CCN value for this function is 6 (as can be seen directly from the graph above, which has 5 enclosed regions and 5 binary predicats), and the correct SCM value for this function would be 18 (see the calculation below).

Similarly, the new version is skewed by the "`while (1)`" statement on line 8 and "`break`" statements on lines 12 and 22, which lead the CCCC to calculate the CCN to be 9 instead of expected 6 and the SCM to be 25 instead of expected 18.

For both functions, the CCN_{sp} is the CCN of the whole function (sine the synchronization point is unavoidable in the function, and all the branching constructs in the function are reachable from the synchronization points in both the cases), so the additional of the synchronization point would be $6*2^1$ (where 6 is the CCN_{sp}, 2 is the *IP* and 1 is the *CP*-1), plus the SCM additions of other non-synchronized branching constructs, which equals to the CCN additions.

Handling all these and other possible corner cases that are currently not supported by the CCCC is not part of this work and is left for the future work on this topic.

Bibliography

[1]. J. Aas, "Understanding the Linux 2.6.8.1 CPU Scheduler", Silicon Graphics Inc. (SGI), 2005

[2]. M.Bilberstein, E.Farchi, S.Ur, Choosing among alternative pasts, Concurrency and Computation: Practice and Experience, Vol.19/3, John Wiley & Sons Ltd, pp.:341-353, 2006

[3]. B. M. Hetzel, The Complete Guide to Software Testing, 2nd ed. (John Wiley & Sons, 1993).

[4]. B. Boehm and V. Basili, "Software Defect Reduction Top 10 List", IEEE Computer, IEEE Computer Society, Vol. 34, No. 1, January 2001, pp. 135-137

[5]. A.Bron, E.Farchi, Y.Magid, Y.Nir, S. Ur, Applications of synchronization coverage, Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming, ACM New York, USA, pp.: 206-212, 2005

[6]. E.Clarke, O.Grumberg, D.Peled, Model Checking, The MIT Press, 1999.

[7]. Jason Cohen, Best Kept Secrets of Peer Code Review (Modern Approach. Practical Advice.), SmartbearSoftware.com, 2006

[8]. B. Eckel, "Thinking in Java", 3rd ed., Prentice-Hall, 2002

[9]. O.Edelstein, E.Farchi, E.Goldin, Y.Nir, G.Ratsaby, S.Ur, “Framework For Testing Mutithreaded Programs”, Concurrency and Computation: Practice and Experience 15(3-5): 485-499 (2003)

[10]. M.Fagan, Design and Code Inspections to Reduce Errors in Program Development, IBM Systems Journal, 15(3), 1976, pp.: 182-211

[11]. C. Fanagan, S. Freund, M. Lifshin, “Type Inference for Atomicity”, Proceedings of the 2005 ACM SIGPLAN international workshop on Types in languages design and implementation, pp.: 47-58, 2005.

[12]. A.Hayardeny, S.Fienblit, E.Farchi, Concurrent and Distributed Desk Checking, In 18th International Parallel and Distributed Processing Symposium (IPDPS'04) – Workshop 16, April 2004

[13]. "Guidelines for the use of the C language in vehicle based software" (MISRA-C:1998), Motor Industry Software Reliability Association, http://www.misra-c2.com/index.htm, 1998.

[14]. "IBM POSIX thread APIs", IBM®, http://publib.boulder.ibm.com/iseries/v5r2/ic2924/index.htm?info/apis/rzah4mst.htm

[15]. Joint Strike Fighter Air Vehicle C++ Coding Standards, Lockheed-Martin Corporation, Document Number 2RDU00001, Rev. C, 12/2005.

[16]. B. Karnighan, D. Ritchie, "The C Programming Language", 2nd ed., Prentice-Hall, 1988

[17]. P.V. Koppol, K. Thai, An Incremental Approach to Structural Testing, ACM SIGSOFT Software Engineering Notes, 21(3), 1996, pp.: 14-23

[18]. T. Littlefair, An Investigation into the use of Software Code Metrics in the Industrial Software Development Environment, PhD thesis, Faculty of Communications, Health, and Science, Edith Cowan University, Mount Lawley Campus, June 2001

[19]. T. Littlefair, C and C++ Code Counter, http://cccc.sourceforge.net, 2003

[20]. S. Lu, W. Jiang, Y. Zhou, A Study of Interleaving Coverage Criteria, POSTER SESSION: ESEC/FSE'07 posters, 2007, pp.: 533-536

[21]. T. McCabe, A Complexity Measure, IEEE Transactions on Software Engineering, SE-2(4), 1976, pp.:308-320.

[22]. Sanjay Misra, Hurevren Kilic, Measurement theory and validation criteria for software complexity measures, ACM SIGSOFT Software Engineering Notes, 32(2), 2007, pp.: 1-3

[23]. "MSDN – Microsoft Developer Network", Microsoft®, http://msdn.microsoft.com

[24]. M. Musuvathi, S. Qadeer, and T. Ball. CHESS: A Systematic Testing Tool for Concurrent Software. Microsoft Research Technical Report MSR-TR-2007-149, 2007.

[25]. B. Pasternak, S. Tyszberowicz, A. Yehudai. GenUTest: A Unit Test and Mock Aspect Generation Tool, Haifa Verification Conference, 2007.

[26]. I. Sommerville, "The Software Engineering", 7th edition, Addison-Wesley, 2004

[27]. A. Tanenbaum, "Modern Operating Systems", 2nd ed., Prentice-Hall, 1992.

[28]. "The Single UNIX Specifications, Version 2", The Open Group, http://www.opengroup.org/

[29]. A. Valmari, "The State Explosion Problem", Lectures on Petri Nets I: Basic Models, Lecture Notes in Computer Science 1941, Springer-Verlag 1998, pp.: 429-528.

[30]. L.Wang, S.Stoller, "Static Analysis of Atomicity for Programs with Non-Blocking Synchronization", Principles and Practice of Parallel Programming, Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming, 2005, pp.:61-71.

[31]. Z. Wang, S. Elbaum, D. Rosenblum, Automated Generation of Context-Aware Tests, Proceedings of the 29th International Conference on Software Engineering, 2007, pp.:406-415

[32]. S. N. Weiss. A formal framework for the study of concurrent program testing. In Proceedings of the Second Workshop on Software Testing, Verification and Analysis, 1988.

[33]. E.J. Weyuker, Evaluating Software Complexity Measures, IEEE Transactions on Software Engineering, 14(9), 1357-1365, September 1988

Appendix A

In this appendix there are code examples used in this work.

Message Queue Implementation

Soup-message-queue module was published under LGPL license at
http://www.angstrom-distribution.org/unstable/sources/libsoup-2.2.7.tar.bz2/libsoup-2.2.7/libsoup/

soup-message-queue.c

```
1  /* -*- Mode: C; tab-width: 8; indent-tabs-mode: t; c-basic-offset: 8 -*- */
2  /*
3   * soup-message-queue.c: Message queue
4   *
5   * Copyright (C) 2003, Ximian, Inc.
6   */
7
8  #ifdef HAVE_CONFIG_H
9  #include <config.h>
10 #endif
11
12 #include "soup-message-queue.h"
13
14 struct SoupMessageQueue {
15         GList *head, *tail;
16         GList *iters;
17
18         GMutex *mutex;
19 };
20
21 /**
22  * soup_message_queue_new:
```

```
 *
 * Creates a new #SoupMessageQueue
 *
 * Return value: a new #SoupMessageQueue object
 **/
SoupMessageQueue *
soup_message_queue_new (void)
{
        SoupMessageQueue *queue;

        queue = g_new0 (SoupMessageQueue, 1);
        queue->mutex = g_mutex_new ();
        return queue;
}

/**
 * soup_message_queue_destroy:
 * @queue: a message queue
 *
 * Frees memory associated with @queue, which must be empty.
 **/
void
soup_message_queue_destroy (SoupMessageQueue *queue)
{
        g_return_if_fail (queue->head == NULL);

        g_list_free (queue->head);
        g_list_free (queue->iters);
        g_mutex_free (queue->mutex);
        g_free (queue);
}

/**
 * soup_message_queue_append:
 * @queue: a queue
```

```
58   * @msg: a message
59   *
60   * Appends @msg to the end of @queue
61   **/
62  void
63  soup_message_queue_append (SoupMessageQueue *queue, SoupMessage *msg)
64  {
65          g_mutex_lock (queue->mutex);
66          if (queue->head) {
67                  queue->tail = g_list_append (queue->tail, msg);
68                  queue->tail = queue->tail->next;
69          } else
70                  queue->head = queue->tail = g_list_append (NULL, msg);
71
72          g_object_add_weak_pointer (G_OBJECT (msg), &queue->tail->data);
73          g_mutex_unlock (queue->mutex);
74  }
75
76  /**
77   * soup_message_queue_first:
78   * @queue: a queue
79   * @iter: pointer to a #SoupMessageQueueIter
80   *
81   * Initializes @iter and returns the first element of @queue. If you
82   * do not iterate all the way to the end of the list, you must call
83   * soup_message_queue_free_iter() to dispose the iterator when you are
84   * done.
85   *
86   * Return value: the first element of @queue, or %NULL if it is empty.
87   **/
88  SoupMessage *
89  soup_message_queue_first (SoupMessageQueue *queue, SoupMessageQueueIter *iter)
90  {
91          g_mutex_lock (queue->mutex);
92
```

```
93          if (!queue->head) {
94                  g_mutex_unlock (queue->mutex);
95                  return NULL;
96          }
97
98          queue->iters = g_list_prepend (queue->iters, iter);
99
100         iter->cur = NULL;
101         iter->next = queue->head;
102         g_mutex_unlock (queue->mutex);
103
104         return soup_message_queue_next (queue, iter);
105 }
106
107 static SoupMessage *
108 queue_remove_internal (SoupMessageQueue *queue, SoupMessageQueueIter *iter)
109 {
110         GList *i;
111         SoupMessageQueueIter *iter2;
112         SoupMessage *msg;
113
114         if (!iter->cur) {
115                 /* We're at end of list or this item was already removed */
116                 return NULL;
117         }
118
119         /* Fix any other iters pointing to iter->cur */
120         for (i = queue->iters; i; i = i->next) {
121                 iter2 = i->data;
122                 if (iter2 != iter) {
123                         if (iter2->cur == iter->cur)
124                                 iter2->cur = NULL;
125                         else if (iter2->next == iter->cur)
126                                 iter2->next = iter->cur->next;
127                 }
```

```
128         }
129
130         msg = iter->cur->data;
131         if (msg)
132                 g_object_remove_weak_pointer (G_OBJECT (msg), &iter->cur->data);
133
134         /* If deleting the last item, fix tail */
135         if (queue->tail == iter->cur)
136                 queue->tail = queue->tail->prev;
137
138         /* Remove the item */
139         queue->head = g_list_delete_link (queue->head, iter->cur);
140         iter->cur = NULL;
141
142         return msg;
143 }
144
145 /**
146  * soup_message_queue_next:
147  * @queue: a queue
148  * @iter: pointer to an initialized #SoupMessageQueueIter
149  *
150  * Returns the next element of @queue
151  *
152  * Return value: the next element, or %NULL if there are no more.
153  **/
154 SoupMessage *
155 soup_message_queue_next (SoupMessageQueue *queue, SoupMessageQueueIter *iter)
156 {
157         g_mutex_lock (queue->mutex);
158
159         while (iter->next) {
160                 iter->cur = iter->next;
161                 iter->next = iter->cur->next;
162                 if (iter->cur->data) {
```

```
163                         g_mutex_unlock (queue->mutex);
164                         return iter->cur->data;
165                 }
166
167                 /* Message was finalized, remove dead queue element */
168                 queue_remove_internal (queue, iter);
169         }
170
171         /* Nothing left */
172         iter->cur = NULL;
173         queue->iters = g_list_remove (queue->iters, iter);
174
175         g_mutex_unlock (queue->mutex);
176         return NULL;
177 }
178
179 /**
180  * soup_message_queue_remove:
181  * @queue: a queue
182  * @iter: pointer to an initialized #SoupMessageQueueIter
183  *
184  * Removes the queue element pointed to by @iter; that is, the last
185  * message returned by soup_message_queue_first() or
186  * soup_message_queue_next().
187  *
188  * Return value: the removed message, or %NULL if the element pointed
189  * to by @iter was already removed.
190  **/
191 SoupMessage *
192 soup_message_queue_remove (SoupMessageQueue *queue, SoupMessageQueueIter *iter)
193 {
194         SoupMessage *msg;
195
196         g_mutex_lock (queue->mutex);
197         msg = queue_remove_internal (queue, iter);
```

```
198         g_mutex_unlock (queue->mutex);
199
200         return msg;
201 }
202
203 /**
204  * soup_message_queue_remove_message:
205  * @queue: a queue
206  * @msg: a #SoupMessage
207  *
208  * Removes the indicated message from @queue.
209  **/
210 void
211 soup_message_queue_remove_message (SoupMessageQueue *queue, SoupMessage *msg)
212 {
213         SoupMessageQueueIter iter;
214         SoupMessage *msg2;
215
216         for (msg2 = soup_message_queue_first (queue, &iter); msg2; msg2 = soup_message_queue_next (queue, &iter))
217 {
218                 if (msg2 == msg) {
219                         soup_message_queue_remove (queue, &iter);
220                         soup_message_queue_free_iter (queue, &iter);
221                         return;
222                 }
223         }
224 }
225
226
227 /**
228  * soup_message_queue_free_iter:
229  * @queue: a queue
230  * @iter: pointer to an initialized #SoupMessageQueueIter
231  *
232  * Removes @iter from the list of active iterators in @queue.
```

```
233  **/
234 void
235 soup_message_queue_free_iter (SoupMessageQueue *queue,
236                               SoupMessageQueueIter *iter)
237 {
238         g_mutex_lock (queue->mutex);
239         queue->iters = g_list_remove (queue->iters, iter);
240         g_mutex_unlock (queue->mutex);
241 }
```

Busybox HTTP server implementation

HTTPD module was published under GPLv2 license at http://git.busybox.net/busybox/plain/networking/httpd.c.

httpd.c – Current Version as of May 25, 2009.

```
1  /* vi: set sw=4 ts=4: */
2  /*
3   * httpd implementation for busybox
4   *
5   * Copyright (C) 2002,2003 Glenn Engel <glenne@engel.org>
6   * Copyright (C) 2003-2006 Vladimir Oleynik <dzo@simtreas.ru>
7   *
8   * simplify patch stolen from libbb without using strdup
9   *
10  * Licensed under GPLv2 or later, see file LICENSE in this tarball for details.
11  *
12  *****************************************************************************
13  *
14  * Typical usage:
15  *   for non root user
16  * httpd -p 8080 -h $HOME/public_html
17  *   or for daemon start from rc script with uid=0:
18  * httpd -u www
19  * This is equivalent if www user have uid=80 to
20  * httpd -p 80 -u 80 -h /www -c /etc/httpd.conf -r "Web Server Authentication"
21  *
22  *
23  * When a url starts by "/cgi-bin/" it is assumed to be a cgi script.  The
24  * server changes directory to the location of the script and executes it
25  * after setting QUERY_STRING and other environment variables.
26  *
27  * Doc:
28  * "CGI Environment Variables": http://hoohoo.ncsa.uiuc.edu/cgi/env.html
29  *
```

```
30  * The applet can also be invoked as a url arg decoder and html text encoder
31  * as follows:
32  *  foo=`httpd -d $foo`            # decode "Hello%20World" as "Hello World"
33  *  bar=`httpd -e "<Hello World>"`  # encode as "&#60Hello&#32World&#62"
34  * Note that url encoding for arguments is not the same as html encoding for
35  * presentation.  -d decodes an url-encoded argument while -e encodes in html
36  * for page display.
37  *
38  * httpd.conf has the following format:
39  *
40  * H:/serverroot     # define the server root. It will override -h
41  * A:172.20.         # Allow address from 172.20.0.0/16
42  * A:10.0.0.0/25     # Allow any address from 10.0.0.0-10.0.0.127
43  * A:10.0.0.0/255.255.255.128  # Allow any address that previous set
44  * A:127.0.0.1       # Allow local loopback connections
45  * D:*               # Deny from other IP connections
46  * E404:/path/e404.html # /path/e404.html is the 404 (not found) error page
47  * I:index.html      # Show index.html when a directory is requested
48  *
49  * P:/url:[http://]hostname[:port]/new/path
50  *                   # When /urlXXXXXX is requested, reverse proxy
51  *                   # it to http://hostname[:port]/new/pathXXXXXX
52  *
53  * /cgi-bin:foo:bar  # Require user foo, pwd bar on urls starting with /cgi-bin/
54  * /adm:admin:setup  # Require user admin, pwd setup on urls starting with /adm/
55  * /adm:toor:PaSsWd  # or user toor, pwd PaSsWd on urls starting with /adm/
56  * .au:audio/basic   # additional mime type for audio.au files
57  * *.php:/path/php   # run xxx.php through an interpreter
58  *
59  * A/D may be as a/d or allow/deny - only first char matters.
60  * Deny/Allow IP logic:
61  *  - Default is to allow all (Allow all (A:*) is a no-op).
62  *  - Deny rules take precedence over allow rules.
63  *  - "Deny all" rule (D:*) is applied last.
64  *
```

```
 * Example:
 *   1. Allow only specified addresses
 *     A:172.20          # Allow any address that begins with 172.20.
 *     A:10.10.          # Allow any address that begins with 10.10.
 *     A:127.0.0.1       # Allow local loopback connections
 *     D:*               # Deny from other IP connections
 *
 *   2. Only deny specified addresses
 *     D:1.2.3.        # deny from 1.2.3.0 - 1.2.3.255
 *     D:2.3.4.        # deny from 2.3.4.0 - 2.3.4.255
 *     A:*             # (optional line added for clarity)
 *
 * If a sub directory contains a config file it is parsed and merged with
 * any existing settings as if it was appended to the original configuration.
 *
 * subdir paths are relative to the containing subdir and thus cannot
 * affect the parent rules.
 *
 * Note that since the sub dir is parsed in the forked thread servicing the
 * subdir http request, any merge is discarded when the process exits.  As a
 * result, the subdir settings only have a lifetime of a single request.
 *
 * Custom error pages can contain an absolute path or be relative to
 * 'home_httpd'. Error pages are to be static files (no CGI or script). Error
 * page can only be defined in the root configuration file and are not taken
 * into account in local (directories) config files.
 *
 * If -c is not set, an attempt will be made to open the default
 * root configuration file.  If -c is set and the file is not found, the
 * server exits with an error.
 *
 */
 /* TODO: use TCP_CORK, parse_config() */

#include "libbb.h"
```

```
100  #if ENABLE_FEATURE_HTTPD_USE_SENDFILE
101  # include <sys/sendfile.h>
102  #endif
103
104  #define DEBUG 0
105
106  #define IOBUF_SIZE 8192    /* IO buffer */
107
108  /* amount of buffering in a pipe */
109  #ifndef PIPE_BUF
110  # define PIPE_BUF 4096
111  #endif
112  #if PIPE_BUF >= IOBUF_SIZE
113  # error "PIPE_BUF >= IOBUF_SIZE"
114  #endif
115
116  #define HEADER_READ_TIMEOUT 60
117
118  static const char DEFAULT_PATH_HTTPD_CONF[] ALIGN1 = "/etc";
119  static const char HTTPD_CONF[] ALIGN1 = "httpd.conf";
120  static const char HTTP_200[] ALIGN1 = "HTTP/1.0 200 OK\r\n";
121
122  typedef struct has_next_ptr {
123          struct has_next_ptr *next;
124  } has_next_ptr;
125
126  /* Must have "next" as a first member */
127  typedef struct Htaccess {
128          struct Htaccess *next;
129          char *after_colon;
130          char before_colon[1];  /* really bigger, must be last */
131  } Htaccess;
132
133  /* Must have "next" as a first member */
134  typedef struct Htaccess_IP {
```

```
135         struct Htaccess_IP *next;
136         unsigned ip;
137         unsigned mask;
138         int allow_deny;
139 } Htaccess_IP;
140
141 /* Must have "next" as a first member */
142 typedef struct Htaccess_Proxy {
143         struct Htaccess_Proxy *next;
144         char *url_from;
145         char *host_port;
146         char *url_to;
147 } Htaccess_Proxy;
148
149 enum {
150         HTTP_OK = 200,
151         HTTP_PARTIAL_CONTENT = 206,
152         HTTP_MOVED_TEMPORARILY = 302,
153         HTTP_BAD_REQUEST = 400,       /* malformed syntax */
154         HTTP_UNAUTHORIZED = 401, /* authentication needed, respond with auth hdr */
155         HTTP_NOT_FOUND = 404,
156         HTTP_FORBIDDEN = 403,
157         HTTP_REQUEST_TIMEOUT = 408,
158         HTTP_NOT_IMPLEMENTED = 501,   /* used for unrecognized requests */
159         HTTP_INTERNAL_SERVER_ERROR = 500,
160         HTTP_CONTINUE = 100,
161 #if 0   /* future use */
162         HTTP_SWITCHING_PROTOCOLS = 101,
163         HTTP_CREATED = 201,
164         HTTP_ACCEPTED = 202,
165         HTTP_NON_AUTHORITATIVE_INFO = 203,
166         HTTP_NO_CONTENT = 204,
167         HTTP_MULTIPLE_CHOICES = 300,
168         HTTP_MOVED_PERMANENTLY = 301,
169         HTTP_NOT_MODIFIED = 304,
```

```
170         HTTP_PAYMENT_REQUIRED = 402,
171         HTTP_BAD_GATEWAY = 502,
172         HTTP_SERVICE_UNAVAILABLE = 503, /* overload, maintenance */
173         HTTP_RESPONSE_SETSIZE = 0xffffffff
174 #endif
175 };
176
177 static const uint16_t http_response_type[] ALIGN2 = {
178         HTTP_OK,
179 #if ENABLE_FEATURE_HTTPD_RANGES
180         HTTP_PARTIAL_CONTENT,
181 #endif
182         HTTP_MOVED_TEMPORARILY,
183         HTTP_REQUEST_TIMEOUT,
184         HTTP_NOT_IMPLEMENTED,
185 #if ENABLE_FEATURE_HTTPD_BASIC_AUTH
186         HTTP_UNAUTHORIZED,
187 #endif
188         HTTP_NOT_FOUND,
189         HTTP_BAD_REQUEST,
190         HTTP_FORBIDDEN,
191         HTTP_INTERNAL_SERVER_ERROR,
192 #if 0   /* not implemented */
193         HTTP_CREATED,
194         HTTP_ACCEPTED,
195         HTTP_NO_CONTENT,
196         HTTP_MULTIPLE_CHOICES,
197         HTTP_MOVED_PERMANENTLY,
198         HTTP_NOT_MODIFIED,
199         HTTP_BAD_GATEWAY,
200         HTTP_SERVICE_UNAVAILABLE,
201 #endif
202 };
203
204 static const struct {
```

```
205         const char *name;
206         const char *info;
207 } http_response[ARRAY_SIZE(http_response_type)] = {
208         { "OK", NULL },
209 #if ENABLE_FEATURE_HTTPD_RANGES
210         { "Partial Content", NULL },
211 #endif
212         { "Found", NULL },
213         { "Request Timeout", "No request appeared within 60 seconds" },
214         { "Not Implemented", "The requested method is not recognized" },
215 #if ENABLE_FEATURE_HTTPD_BASIC_AUTH
216         { "Unauthorized", "" },
217 #endif
218         { "Not Found", "The requested URL was not found" },
219         { "Bad Request", "Unsupported method" },
220         { "Forbidden", ""  },
221         { "Internal Server Error", "Internal Server Error" },
222 #if 0   /* not implemented */
223         { "Created" },
224         { "Accepted" },
225         { "No Content" },
226         { "Multiple Choices" },
227         { "Moved Permanently" },
228         { "Not Modified" },
229         { "Bad Gateway", "" },
230         { "Service Unavailable", "" },
231 #endif
232 };
233
234
235 struct globals {
236         int verbose;            /* must be int (used by getopt32) */
237         smallint flg_deny_all;
238
239         unsigned rmt_ip;       /* used for IP-based allow/deny rules */
```

```
240         time_t last_mod;
241         char *rmt_ip_str;       /* for $REMOTE_ADDR and $REMOTE_PORT */
242         const char *bind_addr_or_port;
243
244         const char *g_query;
245         const char *opt_c_configFile;
246         const char *home_httpd;
247         const char *index_page;
248
249         const char *found_mime_type;
250         const char *found_moved_temporarily;
251         Htaccess_IP *ip_a_d;    /* config allow/deny lines */
252
253         IF_FEATURE_HTTPD_BASIC_AUTH(const char *g_realm;)
254         IF_FEATURE_HTTPD_BASIC_AUTH(char *remoteuser;)
255         IF_FEATURE_HTTPD_CGI(char *referer;)
256         IF_FEATURE_HTTPD_CGI(char *user_agent;)
257         IF_FEATURE_HTTPD_CGI(char *host;)
258         IF_FEATURE_HTTPD_CGI(char *http_accept;)
259         IF_FEATURE_HTTPD_CGI(char *http_accept_language;)
260
261         off_t file_size;        /* -1 - unknown */
262 #if ENABLE_FEATURE_HTTPD_RANGES
263         off_t range_start;
264         off_t range_end;
265         off_t range_len;
266 #endif
267
268 #if ENABLE_FEATURE_HTTPD_BASIC_AUTH
269         Htaccess *g_auth;       /* config user:password lines */
270 #endif
271         Htaccess *mime_a;       /* config mime types */
272 #if ENABLE_FEATURE_HTTPD_CONFIG_WITH_SCRIPT_INTERPR
273         Htaccess *script_i;     /* config script interpreters */
274 #endif
```

```
275         char *iobuf;            /* [IOBUF_SIZE] */
276 #define hdr_buf bb_common_bufsiz1
277         char *hdr_ptr;
278         int hdr_cnt;
279 #if ENABLE_FEATURE_HTTPD_ERROR_PAGES
280         const char *http_error_page[ARRAY_SIZE(http_response_type)];
281 #endif
282 #if ENABLE_FEATURE_HTTPD_PROXY
283         Htaccess_Proxy *proxy;
284 #endif
285 };
286 #define G (*ptr_to_globals)
287 #define verbose           (G.verbose          )
288 #define flg_deny_all      (G.flg_deny_all     )
289 #define rmt_ip            (G.rmt_ip           )
290 #define bind_addr_or_port (G.bind_addr_or_port)
291 #define g_query           (G.g_query          )
292 #define opt_c_configFile  (G.opt_c_configFile )
293 #define home_httpd        (G.home_httpd       )
294 #define index_page        (G.index_page       )
295 #define found_mime_type   (G.found_mime_type  )
296 #define found_moved_temporarily (G.found_moved_temporarily)
297 #define last_mod          (G.last_mod         )
298 #define ip_a_d            (G.ip_a_d           )
299 #define g_realm           (G.g_realm          )
300 #define remoteuser        (G.remoteuser       )
301 #define referer           (G.referer          )
302 #define user_agent        (G.user_agent       )
303 #define host              (G.host             )
304 #define http_accept       (G.http_accept      )
305 #define http_accept_language (G.http_accept_language)
306 #define file_size         (G.file_size        )
307 #if ENABLE_FEATURE_HTTPD_RANGES
308 #define range_start       (G.range_start      )
309 #define range_end         (G.range_end        )
```

```
310 #define range_len         (G.range_len        )
311 #else
312 enum {
313         range_start = 0,
314         range_end = MAXINT(off_t) - 1,
315         range_len = MAXINT(off_t),
316 };
317 #endif
318 #define rmt_ip_str        (G.rmt_ip_str       )
319 #define g_auth            (G.g_auth           )
320 #define mime_a            (G.mime_a           )
321 #define script_i          (G.script_i         )
322 #define iobuf             (G.iobuf            )
323 #define hdr_ptr           (G.hdr_ptr          )
324 #define hdr_cnt           (G.hdr_cnt          )
325 #define http_error_page   (G.http_error_page  )
326 #define proxy             (G.proxy            )
327 #define INIT_G() do { \
328         SET_PTR_TO_GLOBALS(xzalloc(sizeof(G))); \
329         IF_FEATURE_HTTPD_BASIC_AUTH(g_realm = "Web Server Authentication";) \
330         bind_addr_or_port = "80"; \
331         index_page = "index.html"; \
332         file_size = -1; \
333 } while (0)
334
335
336 #define STRNCASECMP(a, str) strncasecmp((a), (str), sizeof(str)-1)
337
338 /* Prototypes */
339 enum {
340         SEND_HEADERS     = (1 << 0),
341         SEND_BODY        = (1 << 1),
342         SEND_HEADERS_AND_BODY = SEND_HEADERS + SEND_BODY,
343 };
344 static void send_file_and_exit(const char *url, int what) NORETURN;
```

```
345
346 static void free_llist(has_next_ptr **pptr)
347 {
348         has_next_ptr *cur = *pptr;
349         while (cur) {
350                 has_next_ptr *t = cur;
351                 cur = cur->next;
352                 free(t);
353         }
354         *pptr = NULL;
355 }
356
357 static ALWAYS_INLINE void free_Htaccess_list(Htaccess **pptr)
358 {
359         free_llist((has_next_ptr**)pptr);
360 }
361
362 static ALWAYS_INLINE void free_Htaccess_IP_list(Htaccess_IP **pptr)
363 {
364         free_llist((has_next_ptr**)pptr);
365 }
366
367 /* Returns presumed mask width in bits or < 0 on error.
368  * Updates strp, stores IP at provided pointer */
369 static int scan_ip(const char **strp, unsigned *ipp, unsigned char endc)
370 {
371         const char *p = *strp;
372         int auto_mask = 8;
373         unsigned ip = 0;
374         int j;
375
376         if (*p == '/')
377                 return -auto_mask;
378
379         for (j = 0; j < 4; j++) {
```

```
380                 unsigned octet;
381
382                 if ((*p < '0' || *p > '9') && *p != '/' && *p)
383                         return -auto_mask;
384                 octet = 0;
385                 while (*p >= '0' && *p <= '9') {
386                         octet *= 10;
387                         octet += *p - '0';
388                         if (octet > 255)
389                                 return -auto_mask;
390                         p++;
391                 }
392                 if (*p == '.')
393                         p++;
394                 if (*p != '/' && *p)
395                         auto_mask += 8;
396                 ip = (ip << 8) | octet;
397         }
398         if (*p) {
399                 if (*p != endc)
400                         return -auto_mask;
401                 p++;
402                 if (*p == '\0')
403                         return -auto_mask;
404         }
405         *ipp = ip;
406         *strp = p;
407         return auto_mask;
408 }
409
410 /* Returns 0 on success. Stores IP and mask at provided pointers */
411 static int scan_ip_mask(const char *str, unsigned *ipp, unsigned *maskp)
412 {
413         int i;
414         unsigned mask;
```

```
415         char *p;
416
417         i = scan_ip(&str, ipp, '/');
418         if (i < 0)
419                 return i;
420
421         if (*str) {
422                 /* there is /xxx after dotted-IP address */
423                 i = bb_strtou(str, &p, 10);
424                 if (*p == '.') {
425                         /* 'xxx' itself is dotted-IP mask, parse it */
426                         /* (return 0 (success) only if it has N.N.N.N form) */
427                         return scan_ip(&str, maskp, '\0') - 32;
428                 }
429                 if (*p)
430                         return -1;
431         }
432
433         if (i > 32)
434                 return -1;
435
436         if (sizeof(unsigned) == 4 && i == 32) {
437                 /* mask >>= 32 below may not work */
438                 mask = 0;
439         } else {
440                 mask = 0xffffffff;
441                 mask >>= i;
442         }
443         /* i == 0 -> *maskp = 0x00000000
444          * i == 1 -> *maskp = 0x80000000
445          * i == 4 -> *maskp = 0xf0000000
446          * i == 31 -> *maskp = 0xfffffffe
447          * i == 32 -> *maskp = 0xffffffff */
448         *maskp = (uint32_t)(~mask);
449         return 0;
```

```
}

/*
 * Parse configuration file into in-memory linked list.
 *
 * Any previous IP rules are discarded.
 * If the flag argument is not SUBDIR_PARSE then all /path and mime rules
 * are also discarded.  That is, previous settings are retained if flag is
 * SUBDIR_PARSE.
 * Error pages are only parsed on the main config file.
 *
 * path   Path where to look for httpd.conf (without filename).
 * flag   Type of the parse request.
 */
/* flag param: */
enum {
        FIRST_PARSE    = 0, /* path will be "/etc" */
        SIGNALED_PARSE = 1, /* path will be "/etc" */
        SUBDIR_PARSE   = 2, /* path will be derived from URL */
};
static void parse_conf(const char *path, int flag)
{
        /* internally used extra flag state */
        enum { TRY_CURDIR_PARSE = 3 };

        FILE *f;
        const char *filename;
        char buf[160];

        /* discard old rules */
        free_Htaccess_IP_list(&ip_a_d);
        flg_deny_all = 0;
        /* retain previous auth and mime config only for subdir parse */
        if (flag != SUBDIR_PARSE) {
                free_Htaccess_list(&mime_a);
```

```
485 #if ENABLE_FEATURE_HTTPD_BASIC_AUTH
486                 free_Htaccess_list(&g_auth);
487 #endif
488 #if ENABLE_FEATURE_HTTPD_CONFIG_WITH_SCRIPT_INTERPR
489                 free_Htaccess_list(&script_i);
490 #endif
491         }
492
493         filename = opt_c_configFile;
494         if (flag == SUBDIR_PARSE || filename == NULL) {
495                 filename = alloca(strlen(path) + sizeof(HTTPD_CONF) + 2);
496                 sprintf((char *)filename, "%s/%s", path, HTTPD_CONF);
497         }
498
499         while ((f = fopen_for_read(filename)) == NULL) {
500                 if (flag >= SUBDIR_PARSE) { /* SUBDIR or TRY_CURDIR */
501                         /* config file not found, no changes to config */
502                         return;
503                 }
504                 if (flag == FIRST_PARSE) {
505                         /* -c CONFFILE given, but CONFFILE doesn't exist? */
506                         if (opt_c_configFile)
507                                 bb_simple_perror_msg_and_die(opt_c_configFile);
508                         /* else: no -c, thus we looked at /etc/httpd.conf,
509                          * and it's not there. try ./httpd.conf: */
510                 }
511                 flag = TRY_CURDIR_PARSE;
512                 filename = HTTPD_CONF;
513         }
514
515 #if ENABLE_FEATURE_HTTPD_BASIC_AUTH
516         /* in "/file:user:pass" lines, we prepend path in subdirs */
517         if (flag != SUBDIR_PARSE)
518                 path = "";
519 #endif
```

```
520         /* The lines can be:
521          *
522          * I:default_index_file
523          * H:http_home
524          * [AD]:IP[/mask]   # allow/deny, * for wildcard
525          * Ennn:error.html  # error page for status nnn
526          * P:/url:[http://]hostname[:port]/new/path # reverse proxy
527          * .ext:mime/type   # mime type
528          * *.php:/path/php  # run xxx.php through an interpreter
529          * /file:user:pass  # username and password
530          */
531         while (fgets(buf, sizeof(buf), f) != NULL) {
532                 unsigned strlen_buf;
533                 unsigned char ch;
534                 char *after_colon;
535
536                 { /* remove all whitespace, and # comments */
537                         char *p, *p0;
538
539                         p0 = buf;
540                         /* skip non-whitespace beginning. Often the whole line
541                          * is non-whitespace. We want this case to work fast,
542                          * without needless copying, therefore we don't merge
543                          * this operation into next while loop. */
544                         while ((ch = *p0) != '\0' && ch != '\n' && ch != '#'
545                          && ch != ' ' && ch != '\t'
546                         ) {
547                                 p0++;
548                         }
549                         p = p0;
550                         /* if we enter this loop, we have some whitespace.
551                          * discard it */
552                         while (ch != '\0' && ch != '\n' && ch != '#') {
553                                 if (ch != ' ' && ch != '\t') {
554                                         *p++ = ch;
```

```
555                     }
556                     ch = *++p0;
557             }
558             *p = '\0';
559             strlen_buf = p - buf;
560             if (strlen_buf == 0)
561                     continue; /* empty line */
562     }
563
564     after_colon = strchr(buf, ':');
565     /* strange line? */
566     if (after_colon == NULL || *++after_colon == '\0')
567             goto config_error;
568
569     ch = (buf[0] & ~0x20); /* toupper if it's a letter */
570
571     if (ch == 'I') {
572             index_page = xstrdup(after_colon);
573             continue;
574     }
575
576     /* do not allow jumping around using H in subdir's configs */
577     if (flag == FIRST_PARSE && ch == 'H') {
578             home_httpd = xstrdup(after_colon);
579             xchdir(home_httpd);
580             continue;
581     }
582
583     if (ch == 'A' || ch == 'D') {
584             Htaccess_IP *pip;
585
586             if (*after_colon == '*') {
587                     if (ch == 'D') {
588                             /* memorize "deny all" */
589                             flg_deny_all = 1;
```

```
590                                     }
591                                     /* skip assumed "A:*", it is a default anyway */
592                                     continue;
593                             }
594                             /* store "allow/deny IP/mask" line */
595                             pip = xzalloc(sizeof(*pip));
596                             if (scan_ip_mask(after_colon, &pip->ip, &pip->mask)) {
597                                     /* IP{/mask} syntax error detected, protect all */
598                                     ch = 'D';
599                                     pip->mask = 0;
600                             }
601                             pip->allow_deny = ch;
602                             if (ch == 'D') {
603                                     /* Deny:from_IP - prepend */
604                                     pip->next = ip_a_d;
605                                     ip_a_d = pip;
606                             } else {
607                                     /* A:from_IP - append (thus all D's precedes A's) */
608                                     Htaccess_IP *prev_IP = ip_a_d;
609                                     if (prev_IP == NULL) {
610                                             ip_a_d = pip;
611                                     } else {
612                                             while (prev_IP->next)
613                                                     prev_IP = prev_IP->next;
614                                             prev_IP->next = pip;
615                                     }
616                             }
617                             continue;
618                     }
619
620     #if ENABLE_FEATURE_HTTPD_ERROR_PAGES
621                     if (flag == FIRST_PARSE && ch == 'E') {
622                             unsigned i;
623                             int status = atoi(buf + 1); /* error status code */
624
```

```
625                         if (status < HTTP_CONTINUE) {
626                                 goto config_error;
627                         }
628                         /* then error page; find matching status */
629                         for (i = 0; i < ARRAY_SIZE(http_response_type); i++) {
630                                 if (http_response_type[i] == status) {
631                                         /* We chdir to home_httpd, thus no need to
632                                          * concat_path_file(home_httpd, after_colon)
633                                          * here */
634                                         http_error_page[i] = xstrdup(after_colon);
635                                         break;
636                                 }
637                         }
638                         continue;
639                 }
640 #endif
641
642 #if ENABLE_FEATURE_HTTPD_PROXY
643                 if (flag == FIRST_PARSE && ch == 'P') {
644                         /* P:/url:[http://]hostname[:port]/new/path */
645                         char *url_from, *host_port, *url_to;
646                         Htaccess_Proxy *proxy_entry;
647
648                         url_from = after_colon;
649                         host_port = strchr(after_colon, ':');
650                         if (host_port == NULL) {
651                                 goto config_error;
652                         }
653                         *host_port++ = '\0';
654                         if (strncmp(host_port, "http://", 7) == 0)
655                                 host_port += 7;
656                         if (*host_port == '\0') {
657                                 goto config_error;
658                         }
659                         url_to = strchr(host_port, '/');
```

```
660                         if (url_to == NULL) {
661                                 goto config_error;
662                         }
663                         *url_to = '\0';
664                         proxy_entry = xzalloc(sizeof(*proxy_entry));
665                         proxy_entry->url_from = xstrdup(url_from);
666                         proxy_entry->host_port = xstrdup(host_port);
667                         *url_to = '/';
668                         proxy_entry->url_to = xstrdup(url_to);
669                         proxy_entry->next = proxy;
670                         proxy = proxy_entry;
671                         continue;
672                 }
673 #endif
674                 /* the rest of directives are non-alphabetic,
675                  * must avoid using "toupper'ed" ch */
676                 ch = buf[0];
677
678                 if (ch == '.' /* ".ext:mime/type" */
679 #if ENABLE_FEATURE_HTTPD_CONFIG_WITH_SCRIPT_INTERPR
680                  || (ch == '*' && buf[1] == '.') /* "*.php:/path/php" */
681 #endif
682                 ) {
683                         char *p;
684                         Htaccess *cur;
685
686                         cur = xzalloc(sizeof(*cur) /* includes space for NUL */ + strlen_buf);
687                         strcpy(cur->before_colon, buf);
688                         p = cur->before_colon + (after_colon - buf);
689                         p[-1] = '\0';
690                         cur->after_colon = p;
691                         if (ch == '.') {
692                                 /* .mime line: prepend to mime_a list */
693                                 cur->next = mime_a;
694                                 mime_a = cur;
```

```
695                         }
696 #if ENABLE_FEATURE_HTTPD_CONFIG_WITH_SCRIPT_INTERPR
697                         else {
698                                 /* script interpreter line: prepend to script_i list */
699                                 cur->next = script_i;
700                                 script_i = cur;
701                         }
702 #endif
703                         continue;
704                 }
705
706 #if ENABLE_FEATURE_HTTPD_BASIC_AUTH
707                 if (ch == '/') { /* "/file:user:pass" */
708                         char *p;
709                         Htaccess *cur;
710                         unsigned file_len;
711
712                         /* note: path is "" unless we are in SUBDIR parse,
713                          * otherwise it does NOT start with "/" */
714                         cur = xzalloc(sizeof(*cur) /* includes space for NUL */
715                                 + 1 + strlen(path)
716                                 + strlen_buf
717                                 );
718                         /* form "/path/file" */
719                         sprintf(cur->before_colon, "/%s%.*s",
720                                 path,
721                                 after_colon - buf - 1, /* includes "/", but not ":" */
722                                 buf);
723                         /* canonicalize it */
724                         p = bb_simplify_abs_path_inplace(cur->before_colon);
725                         file_len = p - cur->before_colon;
726                         /* add "user:pass" after NUL */
727                         strcpy(++p, after_colon);
728                         cur->after_colon = p;
729
```

```
730                         /* insert cur into g_auth */
731                         /* g_auth is sorted by decreased filename length */
732                         {
733                                 Htaccess *auth, **authp;
734
735                                 authp = &g_auth;
736                                 while ((auth = *authp) != NULL) {
737                                         if (file_len >= strlen(auth->before_colon)) {
738                                                 /* insert cur before auth */
739                                                 cur->next = auth;
740                                                 break;
741                                         }
742                                         authp = &auth->next;
743                                 }
744                                 *authp = cur;
745                         }
746                         continue;
747                 }
748 #endif /* BASIC_AUTH */
749
750                 /* the line is not recognized */
751  config_error:
752                 bb_error_msg("config error '%s' in '%s'", buf, filename);
753         } /* while (fgets) */
754
755         fclose(f);
756 }
757
758 #if ENABLE_FEATURE_HTTPD_ENCODE_URL_STR
759 /*
760  * Given a string, html-encode special characters.
761  * This is used for the -e command line option to provide an easy way
762  * for scripts to encode result data without confusing browsers.  The
763  * returned string pointer is memory allocated by malloc().
764  *
```

```
765  * Returns a pointer to the encoded string (malloced).
766  */
767 static char *encodeString(const char *string)
768 {
769         /* take the simple route and encode everything */
770         /* could possibly scan once to get length.     */
771         int len = strlen(string);
772         char *out = xmalloc(len * 6 + 1);
773         char *p = out;
774         char ch;
775
776         while ((ch = *string++)) {
777                 /* very simple check for what to encode */
778                 if (isalnum(ch))
779                         *p++ = ch;
780                 else
781                         p += sprintf(p, "&#%d;", (unsigned char) ch);
782         }
783         *p = '\0';
784         return out;
785 }
786 #endif          /* FEATURE_HTTPD_ENCODE_URL_STR */
787
788 /*
789  * Given a URL encoded string, convert it to plain ascii.
790  * Since decoding always makes strings smaller, the decode is done in-place.
791  * Thus, callers should xstrdup() the argument if they do not want the
792  * argument modified.  The return is the original pointer, allowing this
793  * function to be easily used as arguments to other functions.
794  *
795  * string    The first string to decode.
796  * option_d  1 if called for httpd -d
797  *
798  * Returns a pointer to the decoded string (same as input).
799  */
```

```
800 static unsigned hex_to_bin(unsigned char c)
801 {
802         unsigned v;
803
804         v = c - '0';
805         if (v <= 9)
806                 return v;
807         /* c | 0x20: letters to lower case, non-letters
808          * to (potentially different) non-letters */
809         v = (unsigned)(c | 0x20) - 'a';
810         if (v <= 5)
811                 return v + 10;
812         return ~0;
813 }
814 /* For testing:
815 void t(char c) { printf("'%c'(%u) %u\n", c, c, hex_to_bin(c)); }
816 int main() { t(0x10); t(0x20); t('0'); t('9'); t('A'); t('F'); t('a'); t('f');
817 t('0'-1); t('9'+1); t('A'-1); t('F'+1); t('a'-1); t('f'+1); return 0; }
818 */
819 static char *decodeString(char *orig, int option_d)
820 {
821         /* note that decoded string is always shorter than original */
822         char *string = orig;
823         char *ptr = string;
824         char c;
825
826         while ((c = *ptr++) != '\0') {
827                 unsigned v;
828
829                 if (option_d && c == '+') {
830                         *string++ = ' ';
831                         continue;
832                 }
833                 if (c != '%') {
834                         *string++ = c;
```

```
                        continue;
                }
                v = hex_to_bin(ptr[0]);
                if (v > 15) {
 bad_hex:
                        if (!option_d)
                                return NULL;
                        *string++ = '%';
                        continue;
                }
                v = (v * 16) | hex_to_bin(ptr[1]);
                if (v > 255)
                        goto bad_hex;
                if (!option_d && (v == '/' || v == '\0')) {
                        /* caller takes it as indication of invalid
                         * (dangerous wrt exploits) chars */
                        return orig + 1;
                }
                *string++ = v;
                ptr += 2;
        }
        *string = '\0';
        return orig;
}

#if ENABLE_FEATURE_HTTPD_BASIC_AUTH
/*
 * Decode a base64 data stream as per rfc1521.
 * Note that the rfc states that non base64 chars are to be ignored.
 * Since the decode always results in a shorter size than the input,
 * it is OK to pass the input arg as an output arg.
 * Parameter: a pointer to a base64 encoded string.
 * Decoded data is stored in-place.
 */
static void decodeBase64(char *Data)
```

```
{
        const unsigned char *in = (const unsigned char *)Data;
        /* The decoded size will be at most 3/4 the size of the encoded */
        unsigned ch = 0;
        int i = 0;

        while (*in) {
                int t = *in++;

                if (t >= '0' && t <= '9')
                        t = t - '0' + 52;
                else if (t >= 'A' && t <= 'Z')
                        t = t - 'A';
                else if (t >= 'a' && t <= 'z')
                        t = t - 'a' + 26;
                else if (t == '+')
                        t = 62;
                else if (t == '/')
                        t = 63;
                else if (t == '=')
                        t = 0;
                else
                        continue;

                ch = (ch << 6) | t;
                i++;
                if (i == 4) {
                        *Data++ = (char) (ch >> 16);
                        *Data++ = (char) (ch >> 8);
                        *Data++ = (char) ch;
                        i = 0;
                }
        }
        *Data = '\0';
}
```

```
905 #endif
906
907 /*
908  * Create a listen server socket on the designated port.
909  */
910 static int openServer(void)
911 {
912         unsigned n = bb_strtou(bind_addr_or_port, NULL, 10);
913         if (!errno && n && n <= 0xffff)
914                 n = create_and_bind_stream_or_die(NULL, n);
915         else
916                 n = create_and_bind_stream_or_die(bind_addr_or_port, 80);
917         xlisten(n, 9);
918         return n;
919 }
920
921 /*
922  * Log the connection closure and exit.
923  */
924 static void log_and_exit(void) NORETURN;
925 static void log_and_exit(void)
926 {
927         /* Paranoia. IE said to be buggy. It may send some extra data
928          * or be confused by us just exiting without SHUT_WR. Oh well. */
929         shutdown(1, SHUT_WR);
930         /* Why??
931         (this also messes up stdin when user runs httpd -i from terminal)
932         ndelay_on(0);
933         while (read(STDIN_FILENO, iobuf, IOBUF_SIZE) > 0)
934                 continue;
935         */
936
937         if (verbose > 2)
938                 bb_error_msg("closed");
939         _exit(xfunc_error_retval);
```

```
940 }
941
942 /*
943  * Create and send HTTP response headers.
944  * The arguments are combined and sent as one write operation.  Note that
945  * IE will puke big-time if the headers are not sent in one packet and the
946  * second packet is delayed for any reason.
947  * responseNum - the result code to send.
948  */
949 static void send_headers(int responseNum)
950 {
951         static const char RFC1123FMT[] ALIGN1 = "%a, %d %b %Y %H:%M:%S GMT";
952
953         const char *responseString = "";
954         const char *infoString = NULL;
955         const char *mime_type;
956 #if ENABLE_FEATURE_HTTPD_ERROR_PAGES
957         const char *error_page = NULL;
958 #endif
959         unsigned i;
960         time_t timer = time(NULL);
961         char tmp_str[80];
962         int len;
963
964         for (i = 0; i < ARRAY_SIZE(http_response_type); i++) {
965                 if (http_response_type[i] == responseNum) {
966                         responseString = http_response[i].name;
967                         infoString = http_response[i].info;
968 #if ENABLE_FEATURE_HTTPD_ERROR_PAGES
969                         error_page = http_error_page[i];
970 #endif
971                         break;
972                 }
973         }
974         /* error message is HTML */
```

```
975         mime_type = responseNum == HTTP_OK ?
976                                 found_mime_type : "text/html";
977
978         if (verbose)
979                 bb_error_msg("response:%u", responseNum);
980
981         /* emit the current date */
982         strftime(tmp_str, sizeof(tmp_str), RFC1123FMT, gmtime(&timer));
983         len = sprintf(iobuf,
984                         "HTTP/1.0 %d %s\r\nContent-type: %s\r\n"
985                         "Date: %s\r\nConnection: close\r\n",
986                         responseNum, responseString, mime_type, tmp_str);
987
988 #if ENABLE_FEATURE_HTTPD_BASIC_AUTH
989         if (responseNum == HTTP_UNAUTHORIZED) {
990                 len += sprintf(iobuf + len,
991                                 "WWW-Authenticate: Basic realm=\"%s\"\r\n",
992                                 g_realm);
993         }
994 #endif
995         if (responseNum == HTTP_MOVED_TEMPORARILY) {
996                 len += sprintf(iobuf + len, "Location: %s/%s%s\r\n",
997                                 found_moved_temporarily,
998                                 (g_query ? "?" : ""),
999                                 (g_query ? g_query : ""));
1000        }
1001
1002 #if ENABLE_FEATURE_HTTPD_ERROR_PAGES
1003        if (error_page && access(error_page, R_OK) == 0) {
1004                strcat(iobuf, "\r\n");
1005                len += 2;
1006
1007                if (DEBUG)
1008                        fprintf(stderr, "headers: '%s'\n", iobuf);
1009                full_write(STDOUT_FILENO, iobuf, len);
```

```
1010                if (DEBUG)
1011                        fprintf(stderr, "writing error page: '%s'\n", error_page);
1012                return send_file_and_exit(error_page, SEND_BODY);
1013        }
1014 #endif
1015
1016        if (file_size != -1) {    /* file */
1017                strftime(tmp_str, sizeof(tmp_str), RFC1123FMT, gmtime(&last_mod));
1018 #if ENABLE_FEATURE_HTTPD_RANGES
1019                if (responseNum == HTTP_PARTIAL_CONTENT) {
1020                        len += sprintf(iobuf + len, "Content-Range: bytes %"OFF_FMT"d-
1021 %"OFF_FMT"d/%"OFF_FMT"d\r\n",
1022                                        range_start,
1023                                        range_end,
1024                                        file_size);
1025                        file_size = range_end - range_start + 1;
1026                }
1027 #endif
1028                len += sprintf(iobuf + len,
1029 #if ENABLE_FEATURE_HTTPD_RANGES
1030                        "Accept-Ranges: bytes\r\n"
1031 #endif
1032                        "Last-Modified: %s\r\n%s %"OFF_FMT"d\r\n",
1033                                tmp_str,
1034                                "Content-length:",
1035                                file_size
1036                );
1037        }
1038        iobuf[len++] = '\r';
1039        iobuf[len++] = '\n';
1040        if (infoString) {
1041                len += sprintf(iobuf + len,
1042                                "<HTML><HEAD><TITLE>%d %s</TITLE></HEAD>\n"
1043                                "<BODY><H1>%d %s</H1>\n%s\n</BODY></HTML>\n",
1044                                responseNum, responseString,
```

```
1045                                responseNum, responseString, infoString);
1046        }
1047        if (DEBUG)
1048                fprintf(stderr, "headers: '%s'\n", iobuf);
1049        if (full_write(STDOUT_FILENO, iobuf, len) != len) {
1050                if (verbose > 1)
1051                        bb_perror_msg("error");
1052                log_and_exit();
1053        }
1054 }
1055
1056 static void send_headers_and_exit(int responseNum) NORETURN;
1057 static void send_headers_and_exit(int responseNum)
1058 {
1059        send_headers(responseNum);
1060        log_and_exit();
1061 }
1062
1063 /*
1064  * Read from the socket until '\n' or EOF. '\r' chars are removed.
1065  * '\n' is replaced with NUL.
1066  * Return number of characters read or 0 if nothing is read
1067  * ('\r' and '\n' are not counted).
1068  * Data is returned in iobuf.
1069  */
1070 static int get_line(void)
1071 {
1072        int count = 0;
1073        char c;
1074
1075        alarm(HEADER_READ_TIMEOUT);
1076        while (1) {
1077                if (hdr_cnt <= 0) {
1078                        hdr_cnt = safe_read(STDIN_FILENO, hdr_buf, sizeof(hdr_buf));
1079                        if (hdr_cnt <= 0)
```

```
1080                                    break;
1081                            hdr_ptr = hdr_buf;
1082                    }
1083                    iobuf[count] = c = *hdr_ptr++;
1084                    hdr_cnt--;
1085
1086                    if (c == '\r')
1087                            continue;
1088                    if (c == '\n') {
1089                            iobuf[count] = '\0';
1090                            break;
1091                    }
1092                    if (count < (IOBUF_SIZE - 1))      /* check overflow */
1093                            count++;
1094            }
1095            return count;
1096    }
1097
1098    #if ENABLE_FEATURE_HTTPD_CGI || ENABLE_FEATURE_HTTPD_PROXY
1099
1100    /* gcc 4.2.1 fares better with NOINLINE */
1101    static NOINLINE void cgi_io_loop_and_exit(int fromCgi_rd, int toCgi_wr, int post_len) NORETURN;
1102    static NOINLINE void cgi_io_loop_and_exit(int fromCgi_rd, int toCgi_wr, int post_len)
1103    {
1104            enum { FROM_CGI = 1, TO_CGI = 2 }; /* indexes in pfd[] */
1105            struct pollfd pfd[3];
1106            int out_cnt; /* we buffer a bit of initial CGI output */
1107            int count;
1108
1109            /* iobuf is used for CGI -> network data,
1110             * hdr_buf is for network -> CGI data (POSTDATA) */
1111
1112            /* If CGI dies, we still want to correctly finish reading its output
1113             * and send it to the peer. So please no SIGPIPEs! */
1114            signal(SIGPIPE, SIG_IGN);
```

```
1115
1116        // We inconsistently handle a case when more POSTDATA from network
1117        // is coming than we expected. We may give *some part* of that
1118        // extra data to CGI.
1119
1120        //if (hdr_cnt > post_len) {
1121        //      /* We got more POSTDATA from network than we expected */
1122        //      hdr_cnt = post_len;
1123        //}
1124        post_len -= hdr_cnt;
1125        /* post_len - number of POST bytes not yet read from network */
1126
1127        /* NB: breaking out of this loop jumps to log_and_exit() */
1128        out_cnt = 0;
1129        while (1) {
1130                memset(pfd, 0, sizeof(pfd));
1131
1132                pfd[FROM_CGI].fd = fromCgi_rd;
1133                pfd[FROM_CGI].events = POLLIN;
1134
1135                if (toCgi_wr) {
1136                        pfd[TO_CGI].fd = toCgi_wr;
1137                        if (hdr_cnt > 0) {
1138                                pfd[TO_CGI].events = POLLOUT;
1139                        } else if (post_len > 0) {
1140                                pfd[0].events = POLLIN;
1141                        } else {
1142                                /* post_len <= 0 && hdr_cnt <= 0:
1143                                 * no more POST data to CGI,
1144                                 * let CGI see EOF on CGI's stdin */
1145                                close(toCgi_wr);
1146                                toCgi_wr = 0;
1147                        }
1148                }
1149
```

```
1150                /* Now wait on the set of sockets */
1151                count = safe_poll(pfd, 3, -1);
1152                if (count <= 0) {
1153    #if 0
1154                        if (safe_waitpid(pid, &status, WNOHANG) <= 0) {
1155                                /* Weird. CGI didn't exit and no fd's
1156                                 * are ready, yet poll returned?! */
1157                                continue;
1158                        }
1159                        if (DEBUG && WIFEXITED(status))
1160                                bb_error_msg("CGI exited, status=%d", WEXITSTATUS(status));
1161                        if (DEBUG && WIFSIGNALED(status))
1162                                bb_error_msg("CGI killed, signal=%d", WTERMSIG(status));
1163    #endif
1164                        break;
1165                }
1166
1167                if (pfd[TO_CGI].revents) {
1168                        /* hdr_cnt > 0 here due to the way pfd[TO_CGI].events set */
1169                        /* Have data from peer and can write to CGI */
1170                        count = safe_write(toCgi_wr, hdr_ptr, hdr_cnt);
1171                        /* Doesn't happen, we dont use nonblocking IO here
1172                         *if (count < 0 && errno == EAGAIN) {
1173                         *      ...
1174                         *} else */
1175                        if (count > 0) {
1176                                hdr_ptr += count;
1177                                hdr_cnt -= count;
1178                        } else {
1179                                /* EOF/broken pipe to CGI, stop piping POST data */
1180                                hdr_cnt = post_len = 0;
1181                        }
1182                }
1183
1184                if (pfd[0].revents) {
```

```
1185                    /* post_len > 0 && hdr_cnt == 0 here */
1186                    /* We expect data, prev data portion is eaten by CGI
1187                     * and there *is* data to read from the peer
1188                     * (POSTDATA) */
1189                    //count = post_len > (int)sizeof(hdr_buf) ? (int)sizeof(hdr_buf) : post_len;
1190                    //count = safe_read(STDIN_FILENO, hdr_buf, count);
1191                    count = safe_read(STDIN_FILENO, hdr_buf, sizeof(hdr_buf));
1192                    if (count > 0) {
1193                            hdr_cnt = count;
1194                            hdr_ptr = hdr_buf;
1195                            post_len -= count;
1196                    } else {
1197                            /* no more POST data can be read */
1198                            post_len = 0;
1199                    }
1200            }
1201
1202            if (pfd[FROM_CGI].revents) {
1203                    /* There is something to read from CGI */
1204                    char *rbuf = iobuf;
1205
1206                    /* Are we still buffering CGI output? */
1207                    if (out_cnt >= 0) {
1208                            /* HTTP_200[] has single "\r\n" at the end.
1209                             * According to http://hoohoo.ncsa.uiuc.edu/cgi/out.html,
1210                             * CGI scripts MUST send their own header terminated by
1211                             * empty line, then data. That's why we have only one
1212                             * <cr><lf> pair here. We will output "200 OK" line
1213                             * if needed, but CGI still has to provide blank line
1214                             * between header and body */
1215
1216                            /* Must use safe_read, not full_read, because
1217                             * CGI may output a few first bytes and then wait
1218                             * for POSTDATA without closing stdout.
1219                             * With full_read we may wait here forever. */
```

```
                        count = safe_read(fromCgi_rd, rbuf + out_cnt, PIPE_BUF - 8);
                        if (count <= 0) {
                                /* eof (or error) and there was no "HTTP",
                                 * so write it, then write received data */
                                if (out_cnt) {
                                        full_write(STDOUT_FILENO, HTTP_200, sizeof(HTTP_200)-1);
                                        full_write(STDOUT_FILENO, rbuf, out_cnt);
                                }
                                break; /* CGI stdout is closed, exiting */
                        }
                        out_cnt += count;
                        count = 0;
                        /* "Status" header format is: "Status: 302 Redirected\r\n" */
                        if (out_cnt >= 8 && memcmp(rbuf, "Status: ", 8) == 0) {
                                /* send "HTTP/1.0 " */
                                if (full_write(STDOUT_FILENO, HTTP_200, 9) != 9)
                                        break;
                                rbuf += 8; /* skip "Status: " */
                                count = out_cnt - 8;
                                out_cnt = -1; /* buffering off */
                        } else if (out_cnt >= 4) {
                                /* Did CGI add "HTTP"? */
                                if (memcmp(rbuf, HTTP_200, 4) != 0) {
                                        /* there is no "HTTP", do it ourself */
                                        if (full_write(STDOUT_FILENO, HTTP_200, sizeof(HTTP_200)-1) !=
sizeof(HTTP_200)-1)
                                                break;
                                }
                                /* Commented out:
                                if (!strstr(rbuf, "ontent-")) {
                                        full_write(s, "Content-type: text/plain\r\n\r\n", 28);
                                }
                                 * Counter-example of valid CGI without Content-type:
                                 * echo -en "HTTP/1.0 302 Found\r\n"
                                 * echo -en "Location: http://www.busybox.net\r\n"
```

```
1255                                        * echo -en "\r\n"
1256                                        */
1257                                       count = out_cnt;
1258                                       out_cnt = -1; /* buffering off */
1259                               }
1260                       } else {
1261                               count = safe_read(fromCgi_rd, rbuf, PIPE_BUF);
1262                               if (count <= 0)
1263                                       break;  /* eof (or error) */
1264                       }
1265                       if (full_write(STDOUT_FILENO, rbuf, count) != count)
1266                               break;
1267                       if (DEBUG)
1268                               fprintf(stderr, "cgi read %d bytes: '%.*s'\n", count, count, rbuf);
1269               } /* if (pfd[FROM_CGI].revents) */
1270       } /* while (1) */
1271       log_and_exit();
1272 }
1273 #endif
1274
1275 #if ENABLE_FEATURE_HTTPD_CGI
1276
1277 static void setenv1(const char *name, const char *value)
1278 {
1279       setenv(name, value ? value : "", 1);
1280 }
1281
1282 /*
1283  * Spawn CGI script, forward CGI's stdin/out <=> network
1284  *
1285  * Environment variables are set up and the script is invoked with pipes
1286  * for stdin/stdout.  If a POST is being done the script is fed the POST
1287  * data in addition to setting the QUERY_STRING variable (for GETs or POSTs).
1288  *
1289  * Parameters:
```

```
1290  * const char *url              The requested URL (with leading /).
1291  * int post_len                 Length of the POST body.
1292  * const char *cookie           For set HTTP_COOKIE.
1293  * const char *content_type     For set CONTENT_TYPE.
1294  */
1295 static void send_cgi_and_exit(
1296                 const char *url,
1297                 const char *request,
1298                 int post_len,
1299                 const char *cookie,
1300                 const char *content_type) NORETURN;
1301 static void send_cgi_and_exit(
1302                 const char *url,
1303                 const char *request,
1304                 int post_len,
1305                 const char *cookie,
1306                 const char *content_type)
1307 {
1308         struct fd_pair fromCgi;  /* CGI -> httpd pipe */
1309         struct fd_pair toCgi;    /* httpd -> CGI pipe */
1310         char *script;
1311         int pid;
1312
1313         /* Make a copy. NB: caller guarantees:
1314          * url[0] == '/', url[1] != '/' */
1315         url = xstrdup(url);
1316
1317         /*
1318          * We are mucking with environment _first_ and then vfork/exec,
1319          * this allows us to use vfork safely. Parent doesn't care about
1320          * these environment changes anyway.
1321          */
1322
1323         /* Check for [dirs/]script.cgi/PATH_INFO */
1324         script = (char*)url;
```

```
1325        while ((script = strchr(script + 1, '/')) != NULL) {
1326                struct stat sb;
1327
1328                *script = '\0';
1329                if (!is_directory(url + 1, 1, &sb)) {
1330                        /* not directory, found script.cgi/PATH_INFO */
1331                        *script = '/';
1332                        break;
1333                }
1334                *script = '/'; /* is directory, find next '/' */
1335        }
1336        setenv1("PATH_INFO", script);   /* set to /PATH_INFO or "" */
1337        setenv1("REQUEST_METHOD", request);
1338        if (g_query) {
1339                putenv(xasprintf("%s=%s?%s", "REQUEST_URI", url, g_query));
1340        } else {
1341                setenv1("REQUEST_URI", url);
1342        }
1343        if (script != NULL)
1344                *script = '\0';         /* cut off /PATH_INFO */
1345
1346        /* SCRIPT_FILENAME is required by PHP in CGI mode */
1347        if (home_httpd[0] == '/') {
1348                char *fullpath = concat_path_file(home_httpd, url);
1349                setenv1("SCRIPT_FILENAME", fullpath);
1350        }
1351        /* set SCRIPT_NAME as full path: /cgi-bin/dirs/script.cgi */
1352        setenv1("SCRIPT_NAME", url);
1353        /* http://hoohoo.ncsa.uiuc.edu/cgi/env.html:
1354         * QUERY_STRING: The information which follows the ? in the URL
1355         * which referenced this script. This is the query information.
1356         * It should not be decoded in any fashion. This variable
1357         * should always be set when there is query information,
1358         * regardless of command line decoding. */
1359        /* (Older versions of bbox seem to do some decoding) */
```

```
1360          setenv1("QUERY_STRING", g_query);
1361          putenv((char*)"SERVER_SOFTWARE=busybox httpd/"BB_VER);
1362          putenv((char*)"SERVER_PROTOCOL=HTTP/1.0");
1363          putenv((char*)"GATEWAY_INTERFACE=CGI/1.1");
1364          /* Having _separate_ variables for IP and port defeats
1365           * the purpose of having socket abstraction. Which "port"
1366           * are you using on Unix domain socket?
1367           * IOW - REMOTE_PEER="1.2.3.4:56" makes much more sense.
1368           * Oh well... */
1369          {
1370                  char *p = rmt_ip_str ? rmt_ip_str : (char*)"";
1371                  char *cp = strrchr(p, ':');
1372                  if (ENABLE_FEATURE_IPV6 && cp && strchr(cp, ']'))
1373                          cp = NULL;
1374                  if (cp) *cp = '\0'; /* delete :PORT */
1375                  setenv1("REMOTE_ADDR", p);
1376                  if (cp) {
1377                          *cp = ':';
1378  #if ENABLE_FEATURE_HTTPD_SET_REMOTE_PORT_TO_ENV
1379                          setenv1("REMOTE_PORT", cp + 1);
1380  #endif
1381                  }
1382          }
1383          setenv1("HTTP_USER_AGENT", user_agent);
1384          if (http_accept)
1385                  setenv1("HTTP_ACCEPT", http_accept);
1386          if (http_accept_language)
1387                  setenv1("HTTP_ACCEPT_LANGUAGE", http_accept_language);
1388          if (post_len)
1389                  putenv(xasprintf("CONTENT_LENGTH=%d", post_len));
1390          if (cookie)
1391                  setenv1("HTTP_COOKIE", cookie);
1392          if (content_type)
1393                  setenv1("CONTENT_TYPE", content_type);
1394  #if ENABLE_FEATURE_HTTPD_BASIC_AUTH
```

```
1395         if (remoteuser) {
1396                 setenv1("REMOTE_USER", remoteuser);
1397                 putenv((char*)"AUTH_TYPE=Basic");
1398         }
1399 #endif
1400         if (referer)
1401                 setenv1("HTTP_REFERER", referer);
1402         setenv1("HTTP_HOST", host); /* set to "" if NULL */
1403         /* setenv1("SERVER_NAME", safe_gethostname()); - don't do this,
1404          * just run "env SERVER_NAME=xyz httpd ..." instead */
1405
1406         xpiped_pair(fromCgi);
1407         xpiped_pair(toCgi);
1408
1409         pid = vfork();
1410         if (pid < 0) {
1411                 /* TODO: log perror? */
1412                 log_and_exit();
1413         }
1414
1415         if (!pid) {
1416                 /* Child process */
1417                 char *argv[3];
1418
1419                 xfunc_error_retval = 242;
1420
1421                 /* NB: close _first_, then move fds! */
1422                 close(toCgi.wr);
1423                 close(fromCgi.rd);
1424                 xmove_fd(toCgi.rd, 0);  /* replace stdin with the pipe */
1425                 xmove_fd(fromCgi.wr, 1);  /* replace stdout with the pipe */
1426                 /* User seeing stderr output can be a security problem.
1427                  * If CGI really wants that, it can always do dup itself. */
1428                 /* dup2(1, 2); */
1429
```

```
1430                /* Chdiring to script's dir */
1431                script = strrchr(url, '/');
1432                if (script != url) { /* paranoia */
1433                        *script = '\0';
1434                        if (chdir(url + 1) != 0) {
1435                                bb_perror_msg("chdir %s", url + 1);
1436                                goto error_execing_cgi;
1437                        }
1438                        // not needed: *script = '/';
1439                }
1440                script++;
1441
1442                /* set argv[0] to name without path */
1443                argv[0] = script;
1444                argv[1] = NULL;
1445
1446 #if ENABLE_FEATURE_HTTPD_CONFIG_WITH_SCRIPT_INTERPR
1447                {
1448                        char *suffix = strrchr(script, '.');
1449
1450                        if (suffix) {
1451                                Htaccess *cur;
1452                                for (cur = script_i; cur; cur = cur->next) {
1453                                        if (strcmp(cur->before_colon + 1, suffix) == 0) {
1454                                                /* found interpreter name */
1455                                                argv[0] = cur->after_colon;
1456                                                argv[1] = script;
1457                                                argv[2] = NULL;
1458                                                break;
1459                                        }
1460                                }
1461                        }
1462                }
1463 #endif
1464                /* restore default signal dispositions for CGI process */
```

```
1465                bb_signals(0
1466                        | (1 << SIGCHLD)
1467                        | (1 << SIGPIPE)
1468                        | (1 << SIGHUP)
1469                        , SIG_DFL);
1470
1471                /* _NOT_ execvp. We do not search PATH. argv[0] is a filename
1472                 * without any dir components and will only match a file
1473                 * in the current directory */
1474                execv(argv[0], argv);
1475                if (verbose)
1476                        bb_perror_msg("exec %s", argv[0]);
1477 error_execing_cgi:
1478                /* send to stdout
1479                 * (we are CGI here, our stdout is pumped to the net) */
1480                send_headers_and_exit(HTTP_NOT_FOUND);
1481        } /* end child */
1482
1483        /* Parent process */
1484
1485        /* Restore variables possibly changed by child */
1486        xfunc_error_retval = 0;
1487
1488        /* Pump data */
1489        close(fromCgi.wr);
1490        close(toCgi.rd);
1491        cgi_io_loop_and_exit(fromCgi.rd, toCgi.wr, post_len);
1492 }
1493
1494 #endif          /* FEATURE_HTTPD_CGI */
1495
1496 /*
1497  * Send a file response to a HTTP request, and exit
1498  *
1499  * Parameters:
```

```
1500  * const char *url  The requested URL (with leading /).
1501  * what             What to send (headers/body/both).
1502  */
1503 static NOINLINE void send_file_and_exit(const char *url, int what)
1504 {
1505         static const char *const suffixTable[] = {
1506         /* Warning: shorter equivalent suffix in one line must be first */
1507                 ".htm.html", "text/html",
1508                 ".jpg.jpeg", "image/jpeg",
1509                 ".gif",      "image/gif",
1510                 ".png",      "image/png",
1511                 ".txt.h.c.cc.cpp", "text/plain",
1512                 ".css",      "text/css",
1513                 ".wav",      "audio/wav",
1514                 ".avi",      "video/x-msvideo",
1515                 ".qt.mov",   "video/quicktime",
1516                 ".mpe.mpeg", "video/mpeg",
1517                 ".mid.midi", "audio/midi",
1518                 ".mp3",      "audio/mpeg",
1519 #if 0                         /* unpopular */
1520                 ".au",       "audio/basic",
1521                 ".pac",      "application/x-ns-proxy-autoconfig",
1522                 ".vrml.wrl", "model/vrml",
1523 #endif
1524                 NULL
1525         };
1526
1527         char *suffix;
1528         int fd;
1529         const char *const *table;
1530         const char *try_suffix;
1531         ssize_t count;
1532
1533         fd = open(url, O_RDONLY);
1534         if (fd < 0) {
```

```
1535                if (DEBUG)
1536                        bb_perror_msg("can't open '%s'", url);
1537                /* Error pages are sent by using send_file_and_exit(SEND_BODY).
1538                 * IOW: it is unsafe to call send_headers_and_exit
1539                 * if what is SEND_BODY! Can recurse! */
1540                if (what != SEND_BODY)
1541                        send_headers_and_exit(HTTP_NOT_FOUND);
1542                log_and_exit();
1543        }
1544        /* If you want to know about EPIPE below
1545         * (happens if you abort downloads from local httpd): */
1546        signal(SIGPIPE, SIG_IGN);
1547
1548        suffix = strrchr(url, '.');
1549
1550        /* If not found, set default as "application/octet-stream";  */
1551        found_mime_type = "application/octet-stream";
1552        if (suffix) {
1553                Htaccess *cur;
1554                for (table = suffixTable; *table; table += 2) {
1555                        try_suffix = strstr(table[0], suffix);
1556                        if (try_suffix) {
1557                                try_suffix += strlen(suffix);
1558                                if (*try_suffix == '\0' || *try_suffix == '.') {
1559                                        found_mime_type = table[1];
1560                                        break;
1561                                }
1562                        }
1563                }
1564                for (cur = mime_a; cur; cur = cur->next) {
1565                        if (strcmp(cur->before_colon, suffix) == 0) {
1566                                found_mime_type = cur->after_colon;
1567                                break;
1568                        }
1569                }
```

```
1570            }
1571
1572            if (DEBUG)
1573                    bb_error_msg("sending file '%s' content-type: %s",
1574                            url, found_mime_type);
1575
1576    #if ENABLE_FEATURE_HTTPD_RANGES
1577            if (what == SEND_BODY)
1578                    range_start = 0; /* err pages and ranges don't mix */
1579            range_len = MAXINT(off_t);
1580            if (range_start) {
1581                    if (!range_end) {
1582                            range_end = file_size - 1;
1583                    }
1584                    if (range_end < range_start
1585                     || lseek(fd, range_start, SEEK_SET) != range_start
1586                    ) {
1587                            lseek(fd, 0, SEEK_SET);
1588                            range_start = 0;
1589                    } else {
1590                            range_len = range_end - range_start + 1;
1591                            send_headers(HTTP_PARTIAL_CONTENT);
1592                            what = SEND_BODY;
1593                    }
1594            }
1595    #endif
1596            if (what & SEND_HEADERS)
1597                    send_headers(HTTP_OK);
1598    #if ENABLE_FEATURE_HTTPD_USE_SENDFILE
1599            {
1600                    off_t offset = range_start;
1601                    while (1) {
1602                            /* sz is rounded down to 64k */
1603                            ssize_t sz = MAXINT(ssize_t) - 0xffff;
1604                            IF_FEATURE_HTTPD_RANGES(if (sz > range_len) sz = range_len;)
```

```
1605                        count = sendfile(STDOUT_FILENO, fd, &offset, sz);
1606                        if (count < 0) {
1607                                if (offset == range_start)
1608                                        break; /* fall back to read/write loop */
1609                                goto fin;
1610                        }
1611                        IF_FEATURE_HTTPD_RANGES(range_len -= sz;)
1612                        if (count == 0 || range_len == 0)
1613                                log_and_exit();
1614                }
1615        }
1616 #endif
1617        while ((count = safe_read(fd, iobuf, IOBUF_SIZE)) > 0) {
1618                ssize_t n;
1619                IF_FEATURE_HTTPD_RANGES(if (count > range_len) count = range_len;)
1620                n = full_write(STDOUT_FILENO, iobuf, count);
1621                if (count != n)
1622                        break;
1623                IF_FEATURE_HTTPD_RANGES(range_len -= count;)
1624                if (range_len == 0)
1625                        break;
1626        }
1627        if (count < 0) {
1628  IF_FEATURE_HTTPD_USE_SENDFILE(fin:)
1629                if (verbose > 1)
1630                        bb_perror_msg("error");
1631        }
1632        log_and_exit();
1633 }
1634
1635 static int checkPermIP(void)
1636 {
1637        Htaccess_IP *cur;
1638
1639        for (cur = ip_a_d; cur; cur = cur->next) {
```

```
1640  #if DEBUG
1641                  fprintf(stderr,
1642                          "checkPermIP: '%s' ? '%u.%u.%u.%u/%u.%u.%u.%u'\n",
1643                          rmt_ip_str,
1644                          (unsigned char)(cur->ip >> 24),
1645                          (unsigned char)(cur->ip >> 16),
1646                          (unsigned char)(cur->ip >> 8),
1647                          (unsigned char)(cur->ip),
1648                          (unsigned char)(cur->mask >> 24),
1649                          (unsigned char)(cur->mask >> 16),
1650                          (unsigned char)(cur->mask >> 8),
1651                          (unsigned char)(cur->mask)
1652                  );
1653  #endif
1654                  if ((rmt_ip & cur->mask) == cur->ip)
1655                          return (cur->allow_deny == 'A'); /* A -> 1 */
1656          }
1657
1658          return !flg_deny_all; /* depends on whether we saw "D:*" */
1659  }
1660
1661  #if ENABLE_FEATURE_HTTPD_BASIC_AUTH
1662  /*
1663   * Config file entries are of the form "/<path>:<user>:<passwd>".
1664   * If config file has no prefix match for path, access is allowed.
1665   *
1666   * path                 The file path
1667   * user_and_passwd      "user:passwd" to validate
1668   *
1669   * Returns 1 if user_and_passwd is OK.
1670   */
1671  static int check_user_passwd(const char *path, const char *user_and_passwd)
1672  {
1673          Htaccess *cur;
1674          const char *prev = NULL;
```

```
1675
1676        for (cur = g_auth; cur; cur = cur->next) {
1677                const char *dir_prefix;
1678                size_t len;
1679
1680                dir_prefix = cur->before_colon;
1681
1682                /* WHY? */
1683                /* If already saw a match, don't accept other different matches */
1684                if (prev && strcmp(prev, dir_prefix) != 0)
1685                        continue;
1686
1687                if (DEBUG)
1688                        fprintf(stderr, "checkPerm: '%s' ? '%s'\n", dir_prefix, user_and_passwd);
1689
1690                /* If it's not a prefix match, continue searching */
1691                len = strlen(dir_prefix);
1692                if (len != 1 /* dir_prefix "/" matches all, don't need to check */
1693                 && (strncmp(dir_prefix, path, len) != 0
1694                    || (path[len] != '/' && path[len] != '\0'))
1695                ) {
1696                        continue;
1697                }
1698
1699                /* Path match found */
1700                prev = dir_prefix;
1701
1702                if (ENABLE_FEATURE_HTTPD_AUTH_MD5) {
1703                        char *md5_passwd;
1704
1705                        md5_passwd = strchr(cur->after_colon, ':');
1706                        if (md5_passwd && md5_passwd[1] == '$' && md5_passwd[2] == '1'
1707                         && md5_passwd[3] == '$' && md5_passwd[4]
1708                        ) {
1709                                char *encrypted;
```

```
1710                                int r, user_len_p1;
1711
1712                                md5_passwd++;
1713                                user_len_p1 = md5_passwd - cur->after_colon;
1714                                /* comparing "user:" */
1715                                if (strncmp(cur->after_colon, user_and_passwd, user_len_p1) != 0) {
1716                                        continue;
1717                                }
1718
1719                                encrypted = pw_encrypt(
1720                                        user_and_passwd + user_len_p1 /* cleartext pwd from user */,
1721                                        md5_passwd /*salt */, 1 /* cleanup */);
1722                                r = strcmp(encrypted, md5_passwd);
1723                                free(encrypted);
1724                                if (r == 0)
1725                                        goto set_remoteuser_var; /* Ok */
1726                                continue;
1727                        }
1728                }
1729
1730                /* Comparing plaintext "user:pass" in one go */
1731                if (strcmp(cur->after_colon, user_and_passwd) == 0) {
1732 set_remoteuser_var:
1733                        remoteuser = xstrndup(user_and_passwd,
1734                                        strchrnul(user_and_passwd, ':') - user_and_passwd);
1735                        return 1; /* Ok */
1736                }
1737        } /* for */
1738
1739        /* 0(bad) if prev is set: matches were found but passwd was wrong */
1740        return (prev == NULL);
1741 }
1742 #endif  /* FEATURE_HTTPD_BASIC_AUTH */
1743
1744 #if ENABLE_FEATURE_HTTPD_PROXY
```

```
1745  static Htaccess_Proxy *find_proxy_entry(const char *url)
1746  {
1747          Htaccess_Proxy *p;
1748          for (p = proxy; p; p = p->next) {
1749                  if (strncmp(url, p->url_from, strlen(p->url_from)) == 0)
1750                          return p;
1751          }
1752          return NULL;
1753  }
1754  #endif
1755
1756  /*
1757   * Handle timeouts
1758   */
1759  static void send_REQUEST_TIMEOUT_and_exit(int sig) NORETURN;
1760  static void send_REQUEST_TIMEOUT_and_exit(int sig UNUSED_PARAM)
1761  {
1762          send_headers_and_exit(HTTP_REQUEST_TIMEOUT);
1763  }
1764
1765  /*
1766   * Handle an incoming http request and exit.
1767   */
1768  static void handle_incoming_and_exit(const len_and_sockaddr *fromAddr) NORETURN;
1769  static void handle_incoming_and_exit(const len_and_sockaddr *fromAddr)
1770  {
1771          static const char request_GET[] ALIGN1 = "GET";
1772          struct stat sb;
1773          char *urlcopy;
1774          char *urlp;
1775          char *tptr;
1776  #if ENABLE_FEATURE_HTTPD_CGI
1777          static const char request_HEAD[] ALIGN1 = "HEAD";
1778          const char *prequest;
1779          char *cookie = NULL;
```

```
1780            char *content_type = NULL;
1781            unsigned long length = 0;
1782    #elif ENABLE_FEATURE_HTTPD_PROXY
1783    #define prequest request_GET
1784            unsigned long length = 0;
1785    #endif
1786    #if ENABLE_FEATURE_HTTPD_BASIC_AUTH
1787            smallint authorized = -1;
1788    #endif
1789            smallint ip_allowed;
1790            char http_major_version;
1791    #if ENABLE_FEATURE_HTTPD_PROXY
1792            char http_minor_version;
1793            char *header_buf = header_buf; /* for gcc */
1794            char *header_ptr = header_ptr;
1795            Htaccess_Proxy *proxy_entry;
1796    #endif
1797
1798            /* Allocation of iobuf is postponed until now
1799             * (IOW, server process doesn't need to waste 8k) */
1800            iobuf = xmalloc(IOBUF_SIZE);
1801
1802            rmt_ip = 0;
1803            if (fromAddr->u.sa.sa_family == AF_INET) {
1804                    rmt_ip = ntohl(fromAddr->u.sin.sin_addr.s_addr);
1805            }
1806    #if ENABLE_FEATURE_IPV6
1807            if (fromAddr->u.sa.sa_family == AF_INET6
1808             && fromAddr->u.sin6.sin6_addr.s6_addr32[0] == 0
1809             && fromAddr->u.sin6.sin6_addr.s6_addr32[1] == 0
1810             && ntohl(fromAddr->u.sin6.sin6_addr.s6_addr32[2]) == 0xffff)
1811                    rmt_ip = ntohl(fromAddr->u.sin6.sin6_addr.s6_addr32[3]);
1812    #endif
1813            if (ENABLE_FEATURE_HTTPD_CGI || DEBUG || verbose) {
1814                    /* NB: can be NULL (user runs httpd -i by hand?) */
```

```
1815                rmt_ip_str = xmalloc_sockaddr2dotted(&fromAddr->u.sa);
1816        }
1817        if (verbose) {
1818                /* this trick makes -v logging much simpler */
1819                if (rmt_ip_str)
1820                        applet_name = rmt_ip_str;
1821                if (verbose > 2)
1822                        bb_error_msg("connected");
1823        }
1824
1825        /* Install timeout handler. get_line() needs it. */
1826        signal(SIGALRM, send_REQUEST_TIMEOUT_and_exit);
1827
1828        if (!get_line()) /* EOF or error or empty line */
1829                send_headers_and_exit(HTTP_BAD_REQUEST);
1830
1831        /* Determine type of request (GET/POST) */
1832        urlp = strpbrk(iobuf, " \t");
1833        if (urlp == NULL)
1834                send_headers_and_exit(HTTP_BAD_REQUEST);
1835        *urlp++ = '\0';
1836 #if ENABLE_FEATURE_HTTPD_CGI
1837        prequest = request_GET;
1838        if (strcasecmp(iobuf, prequest) != 0) {
1839                prequest = request_HEAD;
1840                if (strcasecmp(iobuf, prequest) != 0) {
1841                        prequest = "POST";
1842                        if (strcasecmp(iobuf, prequest) != 0)
1843                                send_headers_and_exit(HTTP_NOT_IMPLEMENTED);
1844                }
1845        }
1846 #else
1847        if (strcasecmp(iobuf, request_GET) != 0)
1848                send_headers_and_exit(HTTP_NOT_IMPLEMENTED);
1849 #endif
```

```
1850        urlp = skip_whitespace(urlp);
1851        if (urlp[0] != '/')
1852                send_headers_and_exit(HTTP_BAD_REQUEST);
1853
1854        /* Find end of URL and parse HTTP version, if any */
1855        http_major_version = '0';
1856        IF_FEATURE_HTTPD_PROXY(http_minor_version = '0';)
1857        tptr = strchrnul(urlp, ' ');
1858        /* Is it " HTTP/"? */
1859        if (tptr[0] && strncmp(tptr + 1, HTTP_200, 5) == 0) {
1860                http_major_version = tptr[6];
1861                IF_FEATURE_HTTPD_PROXY(http_minor_version = tptr[8];)
1862        }
1863        *tptr = '\0';
1864
1865        /* Copy URL from after "GET "/"POST " to stack-allocated char[] */
1866        urlcopy = alloca((tptr - urlp) + 2 + strlen(index_page));
1867        /*if (urlcopy == NULL)
1868         *      send_headers_and_exit(HTTP_INTERNAL_SERVER_ERROR);*/
1869        strcpy(urlcopy, urlp);
1870        /* NB: urlcopy ptr is never changed after this */
1871
1872        /* Extract url args if present */
1873        g_query = NULL;
1874        tptr = strchr(urlcopy, '?');
1875        if (tptr) {
1876                *tptr++ = '\0';
1877                g_query = tptr;
1878        }
1879
1880        /* Decode URL escape sequences */
1881        tptr = decodeString(urlcopy, 0);
1882        if (tptr == NULL)
1883                send_headers_and_exit(HTTP_BAD_REQUEST);
1884        if (tptr == urlcopy + 1) {
```

```
1885                /* '/' or NUL is encoded */
1886                send_headers_and_exit(HTTP_NOT_FOUND);
1887        }
1888
1889        /* Canonicalize path */
1890        /* Algorithm stolen from libbb bb_simplify_path(),
1891         * but don't strdup, retain trailing slash, protect root */
1892        urlp = tptr = urlcopy;
1893        do {
1894                if (*urlp == '/') {
1895                        /* skip duplicate (or initial) slash */
1896                        if (*tptr == '/') {
1897                                continue;
1898                        }
1899                        if (*tptr == '.') {
1900                                /* skip extra "/./" */
1901                                if (tptr[1] == '/' || !tptr[1]) {
1902                                        continue;
1903                                }
1904                                /* "..": be careful */
1905                                if (tptr[1] == '.' && (tptr[2] == '/' || !tptr[2])) {
1906                                        ++tptr;
1907                                        if (urlp == urlcopy) /* protect root */
1908                                                send_headers_and_exit(HTTP_BAD_REQUEST);
1909                                        while (*--urlp != '/') /* omit previous dir */;
1910                                                continue;
1911                                }
1912                        }
1913                }
1914                *++urlp = *tptr;
1915        } while (*++tptr);
1916        *++urlp = '\0';       /* terminate after last character */
1917
1918        /* If URL is a directory, add '/' */
1919        if (urlp[-1] != '/') {
```

```
1920                if (is_directory(urlcopy + 1, 1, &sb)) {
1921                        found_moved_temporarily = urlcopy;
1922                }
1923        }
1924
1925        /* Log it */
1926        if (verbose > 1)
1927                bb_error_msg("url:%s", urlcopy);
1928
1929        tptr = urlcopy;
1930        ip_allowed = checkPermIP();
1931        while (ip_allowed && (tptr = strchr(tptr + 1, '/')) != NULL) {
1932                /* have path1/path2 */
1933                *tptr = '\0';
1934                if (is_directory(urlcopy + 1, 1, &sb)) {
1935                        /* may have subdir config */
1936                        parse_conf(urlcopy + 1, SUBDIR_PARSE);
1937                        ip_allowed = checkPermIP();
1938                }
1939                *tptr = '/';
1940        }
1941
1942 #if ENABLE_FEATURE_HTTPD_PROXY
1943        proxy_entry = find_proxy_entry(urlcopy);
1944        if (proxy_entry)
1945                header_buf = header_ptr = xmalloc(IOBUF_SIZE);
1946 #endif
1947
1948        if (http_major_version >= '0') {
1949                /* Request was with "... HTTP/nXXX", and n >= 0 */
1950
1951                /* Read until blank line for HTTP version specified, else parse immediate */
1952                while (1) {
1953                        if (!get_line())
1954                                break; /* EOF or error or empty line */
```

```
1955                        if (DEBUG)
1956                                bb_error_msg("header: '%s'", iobuf);
1957
1958    #if ENABLE_FEATURE_HTTPD_PROXY
1959                        /* We need 2 more bytes for yet another "\r\n" -
1960                         * see near fdprintf(proxy_fd...) further below */
1961                        if (proxy_entry && (header_ptr - header_buf) < IOBUF_SIZE - 2) {
1962                                int len = strlen(iobuf);
1963                                if (len > IOBUF_SIZE - (header_ptr - header_buf) - 4)
1964                                        len = IOBUF_SIZE - (header_ptr - header_buf) - 4;
1965                                memcpy(header_ptr, iobuf, len);
1966                                header_ptr += len;
1967                                header_ptr[0] = '\r';
1968                                header_ptr[1] = '\n';
1969                                header_ptr += 2;
1970                        }
1971    #endif
1972
1973    #if ENABLE_FEATURE_HTTPD_CGI || ENABLE_FEATURE_HTTPD_PROXY
1974                        /* Try and do our best to parse more lines */
1975                        if ((STRNCASECMP(iobuf, "Content-length:") == 0)) {
1976                                /* extra read only for POST */
1977                                if (prequest != request_GET
1978    #if ENABLE_FEATURE_HTTPD_CGI
1979                                 && prequest != request_HEAD
1980    #endif
1981                                ) {
1982                                        tptr = skip_whitespace(iobuf + sizeof("Content-length:") - 1);
1983                                        if (!tptr[0])
1984                                                send_headers_and_exit(HTTP_BAD_REQUEST);
1985                                        /* not using strtoul: it ignores leading minus! */
1986                                        length = bb_strtou(tptr, NULL, 10);
1987                                        /* length is "ulong", but we need to pass it to int later */
1988                                        if (errno || length > INT_MAX)
1989                                                send_headers_and_exit(HTTP_BAD_REQUEST);
```

```
1990                                    }
1991                            }
1992    #endif
1993    #if ENABLE_FEATURE_HTTPD_CGI
1994                            else if (STRNCASECMP(iobuf, "Cookie:") == 0) {
1995                                    cookie = xstrdup(skip_whitespace(iobuf + sizeof("Cookie:")-1));
1996                            } else if (STRNCASECMP(iobuf, "Content-Type:") == 0) {
1997                                    content_type = xstrdup(skip_whitespace(iobuf + sizeof("Content-Type:")-1));
1998                            } else if (STRNCASECMP(iobuf, "Referer:") == 0) {
1999                                    referer = xstrdup(skip_whitespace(iobuf + sizeof("Referer:")-1));
2000                            } else if (STRNCASECMP(iobuf, "User-Agent:") == 0) {
2001                                    user_agent = xstrdup(skip_whitespace(iobuf + sizeof("User-Agent:")-1));
2002                            } else if (STRNCASECMP(iobuf, "Host:") == 0) {
2003                                    host = xstrdup(skip_whitespace(iobuf + sizeof("Host:")-1));
2004                            } else if (STRNCASECMP(iobuf, "Accept:") == 0) {
2005                                    http_accept = xstrdup(skip_whitespace(iobuf + sizeof("Accept:")-1));
2006                            } else if (STRNCASECMP(iobuf, "Accept-Language:") == 0) {
2007                                    http_accept_language = xstrdup(skip_whitespace(iobuf + sizeof("Accept-Language:")-
2008    1));
2009                            }
2010    #endif
2011    #if ENABLE_FEATURE_HTTPD_BASIC_AUTH
2012                            if (STRNCASECMP(iobuf, "Authorization:") == 0) {
2013                                    /* We only allow Basic credentials.
2014                                     * It shows up as "Authorization: Basic <user>:<passwd>" where
2015                                     * "<user>:<passwd>" is base64 encoded.
2016                                     */
2017                                    tptr = skip_whitespace(iobuf + sizeof("Authorization:")-1);
2018                                    if (STRNCASECMP(tptr, "Basic") != 0)
2019                                            continue;
2020                                    tptr += sizeof("Basic")-1;
2021                                    /* decodeBase64() skips whitespace itself */
2022                                    decodeBase64(tptr);
2023                                    authorized = check_user_passwd(urlcopy, tptr);
2024                            }
```

```
#endif
#if ENABLE_FEATURE_HTTPD_RANGES
                        if (STRNCASECMP(iobuf, "Range:") == 0) {
                                /* We know only bytes=NNN-[MMM] */
                                char *s = skip_whitespace(iobuf + sizeof("Range:")-1);
                                if (strncmp(s, "bytes=", 6) == 0) {
                                        s += sizeof("bytes=")-1;
                                        range_start = BB_STRTOOFF(s, &s, 10);
                                        if (s[0] != '-' || range_start < 0) {
                                                range_start = 0;
                                        } else if (s[1]) {
                                                range_end = BB_STRTOOFF(s+1, NULL, 10);
                                                if (errno || range_end < range_start)
                                                        range_start = 0;
                                        }
                                }
                        }
#endif
                } /* while extra header reading */
        }

        /* We are done reading headers, disable peer timeout */
        alarm(0);

        if (strcmp(bb_basename(urlcopy), HTTPD_CONF) == 0 || !ip_allowed) {
                /* protect listing [/path]/httpd.conf or IP deny */
                send_headers_and_exit(HTTP_FORBIDDEN);
        }

#if ENABLE_FEATURE_HTTPD_BASIC_AUTH
        /* Case: no "Authorization:" was seen, but page does require passwd.
         * Check that with dummy user:pass */
        if (authorized < 0)
                authorized = check_user_passwd(urlcopy, ":");
        if (!authorized)
```

```
2060                send_headers_and_exit(HTTP_UNAUTHORIZED);
2061    #endif
2062
2063            if (found_moved_temporarily) {
2064                    send_headers_and_exit(HTTP_MOVED_TEMPORARILY);
2065            }
2066
2067    #if ENABLE_FEATURE_HTTPD_PROXY
2068            if (proxy_entry != NULL) {
2069                    int proxy_fd;
2070                    len_and_sockaddr *lsa;
2071
2072                    proxy_fd = socket(AF_INET, SOCK_STREAM, 0);
2073                    if (proxy_fd < 0)
2074                            send_headers_and_exit(HTTP_INTERNAL_SERVER_ERROR);
2075                    lsa = host2sockaddr(proxy_entry->host_port, 80);
2076                    if (lsa == NULL)
2077                            send_headers_and_exit(HTTP_INTERNAL_SERVER_ERROR);
2078                    if (connect(proxy_fd, &lsa->u.sa, lsa->len) < 0)
2079                            send_headers_and_exit(HTTP_INTERNAL_SERVER_ERROR);
2080                    fdprintf(proxy_fd, "%s %s%s%s%s HTTP/%c.%c\r\n",
2081                                    prequest, /* GET or POST */
2082                                    proxy_entry->url_to, /* url part 1 */
2083                                    urlcopy + strlen(proxy_entry->url_from), /* url part 2 */
2084                                    (g_query ? "?" : ""), /* "?" (maybe) */
2085                                    (g_query ? g_query : ""), /* query string (maybe) */
2086                                    http_major_version, http_minor_version);
2087                    header_ptr[0] = '\r';
2088                    header_ptr[1] = '\n';
2089                    header_ptr += 2;
2090                    write(proxy_fd, header_buf, header_ptr - header_buf);
2091                    free(header_buf); /* on the order of 8k, free it */
2092                    /* cgi_io_loop_and_exit needs to have two distinct fds */
2093                    cgi_io_loop_and_exit(proxy_fd, dup(proxy_fd), length);
2094            }
```

```
#endif

        tptr = urlcopy + 1;      /* skip first '/' */

#if ENABLE_FEATURE_HTTPD_CGI
        if (strncmp(tptr, "cgi-bin/", 8) == 0) {
                if (tptr[8] == '\0') {
                        /* protect listing "cgi-bin/" */
                        send_headers_and_exit(HTTP_FORBIDDEN);
                }
                send_cgi_and_exit(urlcopy, prequest, length, cookie, content_type);
        }
#if ENABLE_FEATURE_HTTPD_CONFIG_WITH_SCRIPT_INTERPR
        {
                char *suffix = strrchr(tptr, '.');
                if (suffix) {
                        Htaccess *cur;
                        for (cur = script_i; cur; cur = cur->next) {
                                if (strcmp(cur->before_colon + 1, suffix) == 0) {
                                        send_cgi_and_exit(urlcopy, prequest, length, cookie, content_type);
                                }
                        }
                }
        }
#endif
        if (prequest != request_GET && prequest != request_HEAD) {
                send_headers_and_exit(HTTP_NOT_IMPLEMENTED);
        }
#endif  /* FEATURE_HTTPD_CGI */

        if (urlp[-1] == '/')
                strcpy(urlp, index_page);
        if (stat(tptr, &sb) == 0) {
                file_size = sb.st_size;
                last_mod = sb.st_mtime;
```

```
2130            }
2131    #if ENABLE_FEATURE_HTTPD_CGI
2132            else if (urlp[-1] == '/') {
2133                    /* It's a dir URL and there is no index.html
2134                     * Try cgi-bin/index.cgi */
2135                    if (access("/cgi-bin/index.cgi"+1, X_OK) == 0) {
2136                            urlp[0] = '\0';
2137                            g_query = urlcopy;
2138                            send_cgi_and_exit("/cgi-bin/index.cgi", prequest, length, cookie, content_type);
2139                    }
2140            }
2141    #endif
2142            /* else {
2143             *      fall through to send_file, it errors out if open fails
2144             * }
2145             */
2146
2147            send_file_and_exit(tptr,
2148    #if ENABLE_FEATURE_HTTPD_CGI
2149                    (prequest != request_HEAD ? SEND_HEADERS_AND_BODY : SEND_HEADERS)
2150    #else
2151                    SEND_HEADERS_AND_BODY
2152    #endif
2153            );
2154    }
2155
2156    /*
2157     * The main http server function.
2158     * Given a socket, listen for new connections and farm out
2159     * the processing as a [v]forked process.
2160     * Never returns.
2161     */
2162    #if BB_MMU
2163    static void mini_httpd(int server_socket) NORETURN;
2164    static void mini_httpd(int server_socket)
```

```
2165  {
2166          /* NB: it's best to not use xfuncs in this loop before fork().
2167           * Otherwise server may die on transient errors (temporary
2168           * out-of-memory condition, etc), which is Bad(tm).
2169           * Try to do any dangerous calls after fork.
2170           */
2171          while (1) {
2172                  int n;
2173                  len_and_sockaddr fromAddr;
2174
2175                  /* Wait for connections... */
2176                  fromAddr.len = LSA_SIZEOF_SA;
2177                  n = accept(server_socket, &fromAddr.u.sa, &fromAddr.len);
2178
2179                  if (n < 0)
2180                          continue;
2181                  /* set the KEEPALIVE option to cull dead connections */
2182                  setsockopt(n, SOL_SOCKET, SO_KEEPALIVE, &const_int_1, sizeof(const_int_1));
2183
2184                  if (fork() == 0) {
2185                          /* child */
2186                          /* Do not reload config on HUP */
2187                          signal(SIGHUP, SIG_IGN);
2188                          close(server_socket);
2189                          xmove_fd(n, 0);
2190                          xdup2(0, 1);
2191
2192                          handle_incoming_and_exit(&fromAddr);
2193                  }
2194                  /* parent, or fork failed */
2195                  close(n);
2196          } /* while (1) */
2197          /* never reached */
2198  }
2199  #else
```

```
2200 static void mini_httpd_nommu(int server_socket, int argc, char **argv) NORETURN;
2201 static void mini_httpd_nommu(int server_socket, int argc, char **argv)
2202 {
2203         char *argv_copy[argc + 2];
2204
2205         argv_copy[0] = argv[0];
2206         argv_copy[1] = (char*)"-i";
2207         memcpy(&argv_copy[2], &argv[1], argc * sizeof(argv[0]));
2208
2209         /* NB: it's best to not use xfuncs in this loop before vfork().
2210          * Otherwise server may die on transient errors (temporary
2211          * out-of-memory condition, etc), which is Bad(tm).
2212          * Try to do any dangerous calls after fork.
2213          */
2214         while (1) {
2215                 int n;
2216                 len_and_sockaddr fromAddr;
2217
2218                 /* Wait for connections... */
2219                 fromAddr.len = LSA_SIZEOF_SA;
2220                 n = accept(server_socket, &fromAddr.u.sa, &fromAddr.len);
2221
2222                 if (n < 0)
2223                         continue;
2224                 /* set the KEEPALIVE option to cull dead connections */
2225                 setsockopt(n, SOL_SOCKET, SO_KEEPALIVE, &const_int_1, sizeof(const_int_1));
2226
2227                 if (vfork() == 0) {
2228                         /* child */
2229                         /* Do not reload config on HUP */
2230                         signal(SIGHUP, SIG_IGN);
2231                         close(server_socket);
2232                         xmove_fd(n, 0);
2233                         xdup2(0, 1);
2234
```

```
                        /* Run a copy of ourself in inetd mode */
                        re_exec(argv_copy);
                }
                /* parent, or vfork failed */
                close(n);
        } /* while (1) */
        /* never reached */
}
#endif

/*
 * Process a HTTP connection on stdin/out.
 * Never returns.
 */
static void mini_httpd_inetd(void) NORETURN;
static void mini_httpd_inetd(void)
{
        len_and_sockaddr fromAddr;

        memset(&fromAddr, 0, sizeof(fromAddr));
        fromAddr.len = LSA_SIZEOF_SA;
        /* NB: can fail if user runs it by hand and types in http cmds */
        getpeername(0, &fromAddr.u.sa, &fromAddr.len);
        handle_incoming_and_exit(&fromAddr);
}

static void sighup_handler(int sig UNUSED_PARAM)
{
        parse_conf(DEFAULT_PATH_HTTPD_CONF, SIGNALED_PARSE);
}

enum {
        c_opt_config_file = 0,
        d_opt_decode_url,
        h_opt_home_httpd,
```

```
2270            IF_FEATURE_HTTPD_ENCODE_URL_STR(e_opt_encode_url,)
2271            IF_FEATURE_HTTPD_BASIC_AUTH(    r_opt_realm     ,)
2272            IF_FEATURE_HTTPD_AUTH_MD5(      m_opt_md5       ,)
2273            IF_FEATURE_HTTPD_SETUID(        u_opt_setuid    ,)
2274            p_opt_port      ,
2275            p_opt_inetd     ,
2276            p_opt_foreground,
2277            p_opt_verbose   ,
2278            OPT_CONFIG_FILE = 1 << c_opt_config_file,
2279            OPT_DECODE_URL  = 1 << d_opt_decode_url,
2280            OPT_HOME_HTTPD  = 1 << h_opt_home_httpd,
2281            OPT_ENCODE_URL  = IF_FEATURE_HTTPD_ENCODE_URL_STR((1 << e_opt_encode_url)) + 0,
2282            OPT_REALM       = IF_FEATURE_HTTPD_BASIC_AUTH(    (1 << r_opt_realm     )) + 0,
2283            OPT_MD5         = IF_FEATURE_HTTPD_AUTH_MD5(      (1 << m_opt_md5       )) + 0,
2284            OPT_SETUID      = IF_FEATURE_HTTPD_SETUID(        (1 << u_opt_setuid    )) + 0,
2285            OPT_PORT        = 1 << p_opt_port,
2286            OPT_INETD       = 1 << p_opt_inetd,
2287            OPT_FOREGROUND  = 1 << p_opt_foreground,
2288            OPT_VERBOSE     = 1 << p_opt_verbose,
2289    };
2290
2291
2292    int httpd_main(int argc, char **argv) MAIN_EXTERNALLY_VISIBLE;
2293    int httpd_main(int argc UNUSED_PARAM, char **argv)
2294    {
2295            int server_socket = server_socket; /* for gcc */
2296            unsigned opt;
2297            char *url_for_decode;
2298            IF_FEATURE_HTTPD_ENCODE_URL_STR(const char *url_for_encode;)
2299            IF_FEATURE_HTTPD_SETUID(const char *s_ugid = NULL;)
2300            IF_FEATURE_HTTPD_SETUID(struct bb_uidgid_t ugid;)
2301            IF_FEATURE_HTTPD_AUTH_MD5(const char *pass;)
2302
2303            INIT_G();
2304
```

```
2305  #if ENABLE_LOCALE_SUPPORT
2306          /* Undo busybox.c: we want to speak English in http (dates etc) */
2307          setlocale(LC_TIME, "C");
2308  #endif
2309
2310          home_httpd = xrealloc_getcwd_or_warn(NULL);
2311          /* -v counts, -i implies -f */
2312          opt_complementary = "vv:if";
2313          /* We do not "absolutize" path given by -h (home) opt.
2314           * If user gives relative path in -h,
2315           * $SCRIPT_FILENAME will not be set. */
2316          opt = getopt32(argv, "c:d:h:"
2317                         IF_FEATURE_HTTPD_ENCODE_URL_STR("e:")
2318                         IF_FEATURE_HTTPD_BASIC_AUTH("r:")
2319                         IF_FEATURE_HTTPD_AUTH_MD5("m:")
2320                         IF_FEATURE_HTTPD_SETUID("u:")
2321                         "p:ifv",
2322                         &opt_c_configFile, &url_for_decode, &home_httpd
2323                         IF_FEATURE_HTTPD_ENCODE_URL_STR(, &url_for_encode)
2324                         IF_FEATURE_HTTPD_BASIC_AUTH(, &g_realm)
2325                         IF_FEATURE_HTTPD_AUTH_MD5(, &pass)
2326                         IF_FEATURE_HTTPD_SETUID(, &s_ugid)
2327                         , &bind_addr_or_port
2328                         , &verbose
2329                  );
2330          if (opt & OPT_DECODE_URL) {
2331                  fputs(decodeString(url_for_decode, 1), stdout);
2332                  return 0;
2333          }
2334  #if ENABLE_FEATURE_HTTPD_ENCODE_URL_STR
2335          if (opt & OPT_ENCODE_URL) {
2336                  fputs(encodeString(url_for_encode), stdout);
2337                  return 0;
2338          }
2339  #endif
```

```
2340  #if ENABLE_FEATURE_HTTPD_AUTH_MD5
2341          if (opt & OPT_MD5) {
2342                  puts(pw_encrypt(pass, "$1$", 1));
2343                  return 0;
2344          }
2345  #endif
2346  #if ENABLE_FEATURE_HTTPD_SETUID
2347          if (opt & OPT_SETUID) {
2348                  xget_uidgid(&ugid, s_ugid);
2349          }
2350  #endif
2351
2352  #if !BB_MMU
2353          if (!(opt & OPT_FOREGROUND)) {
2354                  bb_daemonize_or_rexec(0, argv); /* don't change current directory */
2355          }
2356  #endif
2357
2358          xchdir(home_httpd);
2359          if (!(opt & OPT_INETD)) {
2360                  signal(SIGCHLD, SIG_IGN);
2361                  server_socket = openServer();
2362  #if ENABLE_FEATURE_HTTPD_SETUID
2363                  /* drop privileges */
2364                  if (opt & OPT_SETUID) {
2365                          if (ugid.gid != (gid_t)-1) {
2366                                  if (setgroups(1, &ugid.gid) == -1)
2367                                          bb_perror_msg_and_die("setgroups");
2368                                  xsetgid(ugid.gid);
2369                          }
2370                          xsetuid(ugid.uid);
2371                  }
2372  #endif
2373          }
2374
```

```
2375  #if 0
2376          /* User can do it himself: 'env - PATH="$PATH" httpd'
2377           * We don't do it because we don't want to screw users
2378           * which want to do
2379           * 'env - VAR1=val1 VAR2=val2 httpd'
2380           * and have VAR1 and VAR2 values visible in their CGIs.
2381           * Besides, it is also smaller. */
2382          {
2383                  char *p = getenv("PATH");
2384                  /* env strings themself are not freed, no need to xstrdup(p): */
2385                  clearenv();
2386                  if (p)
2387                          putenv(p - 5);
2388  //              if (!(opt & OPT_INETD))
2389  //                      setenv_long("SERVER_PORT", ???);
2390          }
2391  #endif
2392
2393          parse_conf(DEFAULT_PATH_HTTPD_CONF, FIRST_PARSE);
2394          if (!(opt & OPT_INETD))
2395                  signal(SIGHUP, sighup_handler);
2396
2397          xfunc_error_retval = 0;
2398          if (opt & OPT_INETD)
2399                  mini_httpd_inetd();
2400  #if BB_MMU
2401          if (!(opt & OPT_FOREGROUND))
2402                  bb_daemonize(0); /* don't change current directory */
2403          mini_httpd(server_socket); /* never returns */
2404  #else
2405          mini_httpd_nommu(server_socket, argc, argv); /* never returns */
2406  #endif
2407          /* return 0; */
2408  }
```

httpd.c – Older Version (Ver. 1.35, Oct. 6, 2004)

```
1  /* vi: set sw=4 ts=4: */
2  /*
3   * httpd implementation for busybox
4   *
5   * Copyright (C) 2002,2003 Glenn Engel <glenne@engel.org>
6   * Copyright (C) 2003-2006 Vladimir Oleynik <dzo@simtreas.ru>
7   *
8   * simplify patch stolen from libbb without using strdup
9   *
10  * Licensed under GPLv2 or later, see file LICENSE in this tarball for details.
11  *
12  *****************************************************************************
13  *
14  * Typical usage:
15  *   for non root user
16  * httpd -p 8080 -h $HOME/public_html
17  *   or for daemon start from rc script with uid=0:
18  * httpd -u www
19  * This is equivalent if www user have uid=80 to
20  * httpd -p 80 -u 80 -h /www -c /etc/httpd.conf -r "Web Server Authentication"
21  *
22  *
23  * When a url contains "cgi-bin" it is assumed to be a cgi script.  The
24  * server changes directory to the location of the script and executes it
25  * after setting QUERY_STRING and other environment variables.
26  *
27  * Doc:
28  * "CGI Environment Variables": http://hoohoo.ncsa.uiuc.edu/cgi/env.html
29  *
```

```
30   * The server can also be invoked as a url arg decoder and html text encoder
31   * as follows:
32   *  foo=`httpd -d $foo`           # decode "Hello%20World" as "Hello World"
33   *  bar=`httpd -e "<Hello World>"`  # encode as "&#60Hello&#32World&#62"
34   * Note that url encoding for arguments is not the same as html encoding for
35   * presentation.  -d decodes a url-encoded argument while -e encodes in html
36   * for page display.
37   *
38   * httpd.conf has the following format:
39   *
40   * A:172.20.         # Allow address from 172.20.0.0/16
41   * A:10.0.0.0/25     # Allow any address from 10.0.0.0-10.0.0.127
42   * A:10.0.0.0/255.255.255.128  # Allow any address that previous set
43   * A:127.0.0.1       # Allow local loopback connections
44   * D:*               # Deny from other IP connections
45   * /cgi-bin:foo:bar  # Require user foo, pwd bar on urls starting with /cgi-bin/
46   * /adm:admin:setup  # Require user admin, pwd setup on urls starting with /adm/
47   * /adm:toor:PaSsWd  # or user toor, pwd PaSsWd on urls starting with /adm/
48   * .au:audio/basic   # additional mime type for audio.au files
49   * *.php:/path/php   # running cgi.php scripts through an interpreter
50   *
51   * A/D may be as a/d or allow/deny - first char case insensitive
52   * Deny IP rules take precedence over allow rules.
53   *
54   *
55   * The Deny/Allow IP logic:
56   *
57   *  - Default is to allow all.  No addresses are denied unless
58   *         denied with a D: rule.
59   *  - Order of Deny/Allow rules is significant
60   *  - Deny rules take precedence over allow rules.
61   *  - If a deny all rule (D:*) is used it acts as a catch-all for unmatched
62   *       addresses.
63   *  - Specification of Allow all (A:*) is a no-op
64   *
```

```
 * Example:
 *   1. Allow only specified addresses
 *     A:172.20          # Allow any address that begins with 172.20.
 *     A:10.10.          # Allow any address that begins with 10.10.
 *     A:127.0.0.1       # Allow local loopback connections
 *     D:*               # Deny from other IP connections
 *
 *   2. Only deny specified addresses
 *     D:1.2.3.        # deny from 1.2.3.0 - 1.2.3.255
 *     D:2.3.4.        # deny from 2.3.4.0 - 2.3.4.255
 *     A:*             # (optional line added for clarity)
 *
 * If a sub directory contains a config file it is parsed and merged with
 * any existing settings as if it was appended to the original configuration.
 *
 * subdir paths are relative to the containing subdir and thus cannot
 * affect the parent rules.
 *
 * Note that since the sub dir is parsed in the forked thread servicing the
 * subdir http request, any merge is discarded when the process exits.  As a
 * result, the subdir settings only have a lifetime of a single request.
 *
 *
 * If -c is not set, an attempt will be made to open the default
 * root configuration file.  If -c is set and the file is not found, the
 * server exits with an error.
 *
*/

#include "libbb.h"

/* amount of buffering in a pipe */
#ifndef PIPE_BUF
# define PIPE_BUF 4096
#endif
```

```
100
101 static const char httpdVersion[] = "busybox httpd/1.35 6-Oct-2004";
102 static const char default_path_httpd_conf[] = "/etc";
103 static const char httpd_conf[] = "httpd.conf";
104 static const char home[] = "./";
105
106 #define TIMEOUT 60
107
108 // Note: busybox xfuncs are not used because we want the server to keep running
109 //       if something bad happens due to a malformed user request.
110 //       As a result, all memory allocation after daemonize
111 //       is checked rigorously
112
113 //#define DEBUG 1
114 #define DEBUG 0
115
116 #define MAX_MEMORY_BUFF 8192    /* IO buffer */
117
118 typedef struct HT_ACCESS {
119         char *after_colon;
120         struct HT_ACCESS *next;
121         char before_colon[1];         /* really bigger, must last */
122 } Htaccess;
123
124 typedef struct HT_ACCESS_IP {
125         unsigned int ip;
126         unsigned int mask;
127         int allow_deny;
128         struct HT_ACCESS_IP *next;
129 } Htaccess_IP;
130
131 typedef struct {
132         char buf[MAX_MEMORY_BUFF];
133
134         USE_FEATURE_HTTPD_BASIC_AUTH(const char *realm;)
```

```
135         USE_FEATURE_HTTPD_BASIC_AUTH(char *remoteuser;)
136
137         const char *query;
138
139         USE_FEATURE_HTTPD_CGI(char *referer;)
140
141         const char *configFile;
142
143         unsigned int rmt_ip;
144 #if ENABLE_FEATURE_HTTPD_CGI || DEBUG
145         char *rmt_ip_str;        /* for set env REMOTE_ADDR */
146 #endif
147         unsigned port;           /* server initial port and for
148                                                 set env REMOTE_PORT */
149         const char *found_mime_type;
150         const char *found_moved_temporarily;
151
152         off_t ContentLength;          /* -1 - unknown */
153         time_t last_mod;
154
155         Htaccess_IP *ip_a_d;          /* config allow/deny lines */
156         int flg_deny_all;
157 #if ENABLE_FEATURE_HTTPD_BASIC_AUTH
158         Htaccess *auth;               /* config user:password lines */
159 #endif
160 #if ENABLE_FEATURE_HTTPD_CONFIG_WITH_MIME_TYPES
161         Htaccess *mime_a;             /* config mime types */
162 #endif
163
164         int server_socket;
165         int accepted_socket;
166         volatile int alarm_signaled;
167
168 #if ENABLE_FEATURE_HTTPD_CONFIG_WITH_SCRIPT_INTERPR
169         Htaccess *script_i;           /* config script interpreters */
```

```
#endif
} HttpdConfig;

static HttpdConfig *config;

static const char request_GET[] = "GET";    /* size algorithmic optimize */

static const char* const suffixTable [] = {
/* Warning: shorted equivalent suffix in one line must be first */
        ".htm.html", "text/html",
        ".jpg.jpeg", "image/jpeg",
        ".gif", "image/gif",
        ".png", "image/png",
        ".txt.h.c.cc.cpp", "text/plain",
        ".css", "text/css",
        ".wav", "audio/wav",
        ".avi", "video/x-msvideo",
        ".qt.mov", "video/quicktime",
        ".mpe.mpeg", "video/mpeg",
        ".mid.midi", "audio/midi",
        ".mp3", "audio/mpeg",
#if 0                           /* unpopular */
        ".au", "audio/basic",
        ".pac", "application/x-ns-proxy-autoconfig",
        ".vrml.wrl", "model/vrml",
#endif
        0, "application/octet-stream" /* default */
};

typedef enum {
        HTTP_OK = 200,
        HTTP_MOVED_TEMPORARILY = 302,
        HTTP_BAD_REQUEST = 400,       /* malformed syntax */
        HTTP_UNAUTHORIZED = 401, /* authentication needed, respond with auth hdr */
        HTTP_NOT_FOUND = 404,
```

```
205         HTTP_FORBIDDEN = 403,
206         HTTP_REQUEST_TIMEOUT = 408,
207         HTTP_NOT_IMPLEMENTED = 501,   /* used for unrecognized requests */
208         HTTP_INTERNAL_SERVER_ERROR = 500,
209 #if 0 /* future use */
210         HTTP_CONTINUE = 100,
211         HTTP_SWITCHING_PROTOCOLS = 101,
212         HTTP_CREATED = 201,
213         HTTP_ACCEPTED = 202,
214         HTTP_NON_AUTHORITATIVE_INFO = 203,
215         HTTP_NO_CONTENT = 204,
216         HTTP_MULTIPLE_CHOICES = 300,
217         HTTP_MOVED_PERMANENTLY = 301,
218         HTTP_NOT_MODIFIED = 304,
219         HTTP_PAYMENT_REQUIRED = 402,
220         HTTP_BAD_GATEWAY = 502,
221         HTTP_SERVICE_UNAVAILABLE = 503, /* overload, maintenance */
222         HTTP_RESPONSE_SETSIZE = 0xffffffff
223 #endif
224 } HttpResponseNum;
225
226 typedef struct {
227         HttpResponseNum type;
228         const char *name;
229         const char *info;
230 } HttpEnumString;
231
232 static const HttpEnumString httpResponseNames[] = {
233         { HTTP_OK, "OK", NULL },
234         { HTTP_MOVED_TEMPORARILY, "Found", "Directories must end with a slash." },
235         { HTTP_REQUEST_TIMEOUT, "Request Timeout",
236                 "No request appeared within a reasonable time period." },
237         { HTTP_NOT_IMPLEMENTED, "Not Implemented",
238                 "The requested method is not recognized by this server." },
239 #if ENABLE_FEATURE_HTTPD_BASIC_AUTH
```

```
240         { HTTP_UNAUTHORIZED, "Unauthorized", "" },
241 #endif
242         { HTTP_NOT_FOUND, "Not Found",
243                 "The requested URL was not found on this server." },
244         { HTTP_BAD_REQUEST, "Bad Request", "Unsupported method." },
245         { HTTP_FORBIDDEN, "Forbidden", "" },
246         { HTTP_INTERNAL_SERVER_ERROR, "Internal Server Error",
247                 "Internal Server Error" },
248 #if 0                               /* not implemented */
249         { HTTP_CREATED, "Created" },
250         { HTTP_ACCEPTED, "Accepted" },
251         { HTTP_NO_CONTENT, "No Content" },
252         { HTTP_MULTIPLE_CHOICES, "Multiple Choices" },
253         { HTTP_MOVED_PERMANENTLY, "Moved Permanently" },
254         { HTTP_NOT_MODIFIED, "Not Modified" },
255         { HTTP_BAD_GATEWAY, "Bad Gateway", "" },
256         { HTTP_SERVICE_UNAVAILABLE, "Service Unavailable", "" },
257 #endif
258 };
259
260
261 static const char RFC1123FMT[] = "%a, %d %b %Y %H:%M:%S GMT";
262
263
264 #define STRNCASECMP(a, str) strncasecmp((a), (str), sizeof(str)-1)
265
266
267 static int scan_ip(const char **ep, unsigned int *ip, unsigned char endc)
268 {
269         const char *p = *ep;
270         int auto_mask = 8;
271         int j;
272
273         *ip = 0;
274         for (j = 0; j < 4; j++) {
```

```
275                 unsigned int octet;
276
277                 if ((*p < '0' || *p > '9') && (*p != '/' || j == 0) && *p != 0)
278                         return -auto_mask;
279                 octet = 0;
280                 while (*p >= '0' && *p <= '9') {
281                         octet *= 10;
282                         octet += *p - '0';
283                         if (octet > 255)
284                                 return -auto_mask;
285                         p++;
286                 }
287                 if (*p == '.')
288                         p++;
289                 if (*p != '/' && *p != 0)
290                         auto_mask += 8;
291                 *ip = ((*ip) << 8) | octet;
292         }
293         if (*p != 0) {
294                 if (*p != endc)
295                         return -auto_mask;
296                 p++;
297                 if (*p == 0)
298                         return -auto_mask;
299         }
300         *ep = p;
301         return auto_mask;
302 }
303
304 static int scan_ip_mask(const char *ipm, unsigned int *ip, unsigned int *mask)
305 {
306         int i;
307         unsigned int msk;
308
309         i = scan_ip(&ipm, ip, '/');
```

```
310         if (i < 0)
311                 return i;
312         if (*ipm) {
313                 const char *p = ipm;
314
315                 i = 0;
316                 while (*p) {
317                         if (*p < '0' || *p > '9') {
318                                 if (*p == '.') {
319                                         i = scan_ip(&ipm, mask, 0);
320                                         return i != 32;
321                                 }
322                                 return -1;
323                         }
324                         i *= 10;
325                         i += *p - '0';
326                         p++;
327                 }
328         }
329         if (i > 32 || i < 0)
330                 return -1;
331         msk = 0x80000000;
332         *mask = 0;
333         while (i > 0) {
334                 *mask |= msk;
335                 msk >>= 1;
336                 i--;
337         }
338         return 0;
339 }
340
341 #if ENABLE_FEATURE_HTTPD_BASIC_AUTH \
342  || ENABLE_FEATURE_HTTPD_CONFIG_WITH_MIME_TYPES \
343  || ENABLE_FEATURE_HTTPD_CONFIG_WITH_SCRIPT_INTERPR
344 static void free_config_lines(Htaccess **pprev)
```

```
{
        Htaccess *prev = *pprev;

        while (prev) {
                Htaccess *cur = prev;

                prev = cur->next;
                free(cur);
        }
        *pprev = NULL;
}
#endif

/* flag */
#define FIRST_PARSE          0
#define SUBDIR_PARSE         1
#define SIGNALED_PARSE       2
#define FIND_FROM_HTTPD_ROOT 3
/*****************************************************************************
 *
 > $Function: parse_conf()
 *
 * $Description: parse configuration file into in-memory linked list.
 *
 * The first non-white character is examined to determine if the config line
 * is one of the following:
 *    .ext:mime/type   # new mime type not compiled into httpd
 *    [adAD]:from      # ip address allow/deny, * for wildcard
 *    /path:user:pass  # username/password
 *
 * Any previous IP rules are discarded.
 * If the flag argument is not SUBDIR_PARSE then all /path and mime rules
 * are also discarded.  That is, previous settings are retained if flag is
 * SUBDIR_PARSE.
 *
```

```
 * $Parameters:
 *      (const char *) path . . null for ip address checks, path for password
 *                               checks.
 *      (int) flag  . . . . . . the source of the parse request.
 *
 * $Return: (None)
 *
 ******************************************************************************/
static void parse_conf(const char *path, int flag)
{
        FILE *f;
#if ENABLE_FEATURE_HTTPD_BASIC_AUTH
        Htaccess *prev;
#endif
#if ENABLE_FEATURE_HTTPD_BASIC_AUTH \
 || ENABLE_FEATURE_HTTPD_CONFIG_WITH_MIME_TYPES \
 || ENABLE_FEATURE_HTTPD_CONFIG_WITH_SCRIPT_INTERPR
        Htaccess *cur;
#endif

        const char *cf = config->configFile;
        char buf[160];
        char *p0 = NULL;
        char *c, *p;

        /* free previous ip setup if present */
        Htaccess_IP *pip = config->ip_a_d;

        while (pip) {
                Htaccess_IP *cur_ipl = pip;

                pip = cur_ipl->next;
                free(cur_ipl);
        }
        config->ip_a_d = NULL;
```

```
415
416         config->flg_deny_all = 0;
417
418 #if ENABLE_FEATURE_HTTPD_BASIC_AUTH \
419  || ENABLE_FEATURE_HTTPD_CONFIG_WITH_MIME_TYPES \
420  || ENABLE_FEATURE_HTTPD_CONFIG_WITH_SCRIPT_INTERPR
421         /* retain previous auth and mime config only for subdir parse */
422         if (flag != SUBDIR_PARSE) {
423 #if ENABLE_FEATURE_HTTPD_BASIC_AUTH
424                 free_config_lines(&config->auth);
425 #endif
426 #if ENABLE_FEATURE_HTTPD_CONFIG_WITH_MIME_TYPES
427                 free_config_lines(&config->mime_a);
428 #endif
429 #if ENABLE_FEATURE_HTTPD_CONFIG_WITH_SCRIPT_INTERPR
430                 free_config_lines(&config->script_i);
431 #endif
432         }
433 #endif
434
435         if (flag == SUBDIR_PARSE || cf == NULL) {
436                 cf = alloca(strlen(path) + sizeof(httpd_conf) + 2);
437                 if (cf == NULL) {
438                         if (flag == FIRST_PARSE)
439                         bb_error_msg_and_die(bb_msg_memory_exhausted);
440                         return;
441                 }
442                 sprintf((char *)cf, "%s/%s", path, httpd_conf);
443         }
444
445         while ((f = fopen(cf, "r")) == NULL) {
446                 if (flag == SUBDIR_PARSE || flag == FIND_FROM_HTTPD_ROOT) {
447                         /* config file not found, no changes to config */
448                         return;
449                 }
```

```
450                 if (config->configFile && flag == FIRST_PARSE) /* if -c option given */
451                         bb_perror_msg_and_die("%s", cf);
452                 flag = FIND_FROM_HTTPD_ROOT;
453                 cf = httpd_conf;
454         }
455
456 #if ENABLE_FEATURE_HTTPD_BASIC_AUTH
457                 prev = config->auth;
458 #endif
459                 /* This could stand some work */
460         while ((p0 = fgets(buf, sizeof(buf), f)) != NULL) {
461                 c = NULL;
462                 for (p = p0; *p0 != 0 && *p0 != '#'; p0++) {
463                         if (!isspace(*p0)) {
464                                 *p++ = *p0;
465                                 if (*p0 == ':' && c == NULL)
466                                 c = p;
467                         }
468                 }
469                 *p = 0;
470
471                 /* test for empty or strange line */
472                 if (c == NULL || *c == 0)
473                         continue;
474                 p0 = buf;
475                 if (*p0 == 'd')
476                                 *p0 = 'D';
477                 if (*c == '*') {
478                         if (*p0 == 'D') {
479                                 /* memorize deny all */
480                                 config->flg_deny_all++;
481                         }
482                         /* skip default other "word:*" config lines */
483                         continue;
484                 }
```

```
485
486                 if (*p0 == 'a')
487                         *p0 = 'A';
488                 else if (*p0 != 'D' && *p0 != 'A'
489 #if ENABLE_FEATURE_HTTPD_BASIC_AUTH
490                          && *p0 != '/'
491 #endif
492 #if ENABLE_FEATURE_HTTPD_CONFIG_WITH_MIME_TYPES
493                          && *p0 != '.'
494 #endif
495 #if ENABLE_FEATURE_HTTPD_CONFIG_WITH_SCRIPT_INTERPR
496                          && *p0 != '*'
497 #endif
498                         )
499                          continue;
500                 if (*p0 == 'A' || *p0 == 'D') {
501                         /* storing current config IP line */
502                         pip = xzalloc(sizeof(Htaccess_IP));
503                         if (pip) {
504                                 if (scan_ip_mask(c, &(pip->ip), &(pip->mask))) {
505                                         /* syntax IP{/mask} error detected, protect all */
506                                         *p0 = 'D';
507                                         pip->mask = 0;
508                                 }
509                                 pip->allow_deny = *p0;
510                                 if (*p0 == 'D') {
511                                         /* Deny:form_IP move top */
512                                         pip->next = config->ip_a_d;
513                                         config->ip_a_d = pip;
514                                 } else {
515                                         /* add to bottom A:form_IP config line */
516                                         Htaccess_IP *prev_IP = config->ip_a_d;
517
518                                         if (prev_IP == NULL) {
519                                                 config->ip_a_d = pip;
```

```
520                                     } else {
521                                             while (prev_IP->next)
522                                                     prev_IP = prev_IP->next;
523                                             prev_IP->next = pip;
524                                     }
525                                 }
526                             }
527                             continue;
528                     }
529     #if ENABLE_FEATURE_HTTPD_BASIC_AUTH
530                     if (*p0 == '/') {
531                             /* make full path from httpd root / curent_path / config_line_path */
532                             cf = flag == SUBDIR_PARSE ? path : "";
533                             p0 = malloc(strlen(cf) + (c - buf) + 2 + strlen(c));
534                             if (p0 == NULL)
535                                     continue;
536                             c[-1] = 0;
537                             sprintf(p0, "/%s%s", cf, buf);
538
539                             /* another call bb_simplify_path */
540                             cf = p = p0;
541
542                             do {
543                                     if (*p == '/') {
544                                             if (*cf == '/') {    /* skip duplicate (or initial) slash */
545                                                     continue;
546                                             } else if (*cf == '.') {
547                                                     if (cf[1] == '/' || cf[1] == 0) { /* remove extra '.' */
548                                                             continue;
549                                                     } else if ((cf[1] == '.') && (cf[2] == '/' || cf[2] == 0)) {
550                                                             ++cf;
551                                                             if (p > p0) {
552                                                                     while (*--p != '/') /* omit previous dir */;
553                                                             }
554                                                             continue;
```

```
555                                             }
556                                         }
557                                 }
558                                 *++p = *cf;
559                         } while (*++cf);
560
561                         if ((p == p0) || (*p != '/')) {      /* not a trailing slash */
562                                 ++p;                            /* so keep last character */
563                         }
564                         *p = 0;
565                         sprintf(p0, "%s:%s", p0, c);
566                 }
567 #endif
568
569 #if ENABLE_FEATURE_HTTPD_BASIC_AUTH \
570  || ENABLE_FEATURE_HTTPD_CONFIG_WITH_MIME_TYPES \
571  || ENABLE_FEATURE_HTTPD_CONFIG_WITH_SCRIPT_INTERPR
572                 /* storing current config line */
573                 cur = xzalloc(sizeof(Htaccess) + strlen(p0));
574                 if (cur) {
575                         cf = strcpy(cur->before_colon, p0);
576                         c = strchr(cf, ':');
577                         *c++ = 0;
578                         cur->after_colon = c;
579 #if ENABLE_FEATURE_HTTPD_CONFIG_WITH_MIME_TYPES
580                         if (*cf == '.') {
581                                 /* config .mime line move top for overwrite previous */
582                                 cur->next = config->mime_a;
583                                 config->mime_a = cur;
584                                 continue;
585                         }
586 #endif
587 #if ENABLE_FEATURE_HTTPD_CONFIG_WITH_SCRIPT_INTERPR
588                         if (*cf == '*' && cf[1] == '.') {
589                                 /* config script interpreter line move top for overwrite previous */
```

```
590                                 cur->next = config->script_i;
591                                 config->script_i = cur;
592                                 continue;
593                         }
594 #endif
595 #if ENABLE_FEATURE_HTTPD_BASIC_AUTH
596                         free(p0);
597                         if (prev == NULL) {
598                                 /* first line */
599                                 config->auth = prev = cur;
600                         } else {
601                                 /* sort path, if current lenght eq or bigger then move up */
602                                 Htaccess *prev_hti = config->auth;
603                                 size_t l = strlen(cf);
604                                 Htaccess *hti;
605
606                                 for (hti = prev_hti; hti; hti = hti->next) {
607                                         if (l >= strlen(hti->before_colon)) {
608                                                 /* insert before hti */
609                                                 cur->next = hti;
610                                                 if (prev_hti != hti) {
611                                                         prev_hti->next = cur;
612                                                 } else {
613                                                         /* insert as top */
614                                                         config->auth = cur;
615                                                 }
616                                                 break;
617                                         }
618                                         if (prev_hti != hti)
619                                                 prev_hti = prev_hti->next;
620                                 }
621                                 if (!hti) {       /* not inserted, add to bottom */
622                                         prev->next = cur;
623                                         prev = cur;
624                                 }
```

```
                        }
#endif
                }
#endif
        }
        fclose(f);
}

#if ENABLE_FEATURE_HTTPD_ENCODE_URL_STR
/*****************************************************************************
 *
 > $Function: encodeString()
 *
 * $Description: Given a string, html encode special characters.
 *   This is used for the -e command line option to provide an easy way
 *   for scripts to encode result data without confusing browsers.  The
 *   returned string pointer is memory allocated by malloc().
 *
 * $Parameters:
 *      (const char *) string . . The first string to encode.
 *
 * $Return: (char *) . . . . .. . A pointer to the encoded string.
 *
 * $Errors: Returns a null string ("") if memory is not available.
 *
 *****************************************************************************/
static char *encodeString(const char *string)
{
        /* take the simple route and encode everything */
        /* could possibly scan once to get length.     */
        int len = strlen(string);
        char *out = xmalloc(len * 6 + 1);
        char *p = out;
        char ch;

```

```
660         while ((ch = *string++)) {
661                 // very simple check for what to encode
662                 if (isalnum(ch)) *p++ = ch;
663                 else p += sprintf(p, "&#%d;", (unsigned char) ch);
664         }
665         *p = '\0';
666         return out;
667 }
668 #endif          /* FEATURE_HTTPD_ENCODE_URL_STR */
669
670 /*****************************************************************************
671  *
672  > $Function: decodeString()
673  *
674  * $Description: Given a URL encoded string, convert it to plain ascii.
675  *   Since decoding always makes strings smaller, the decode is done in-place.
676  *   Thus, callers should strdup() the argument if they do not want the
677  *   argument modified.  The return is the original pointer, allowing this
678  *   function to be easily used as arguments to other functions.
679  *
680  * $Parameters:
681  *      (char *) string . . . The first string to decode.
682  *      (int)    option_d . . 1 if called for httpd -d
683  *
684  * $Return: (char *)  . . . . A pointer to the decoded string (same as input).
685  *
686  * $Errors: None
687  *
688  *****************************************************************************/
689 static char *decodeString(char *orig, int option_d)
690 {
691         /* note that decoded string is always shorter than original */
692         char *string = orig;
693         char *ptr = string;
694         char c;
```

```
695
696         while ((c = *ptr++) != '\0') {
697                 unsigned value1, value2;
698
699                 if (option_d && c == '+') {
700                         *string++ = ' ';
701                         continue;
702                 }
703                 if (c != '%') {
704                         *string++ = c;
705                         continue;
706                 }
707                 if (sscanf(ptr, "%1X", &value1) != 1
708                  || sscanf(ptr+1, "%1X", &value2) != 1
709                 ) {
710                         if (!option_d)
711                                 return NULL;
712                         *string++ = '%';
713                         continue;
714                 }
715                 value1 = value1 * 16 + value2;
716                 if (!option_d && (value1 == '/' || value1 == '\0')) {
717                         /* caller takes it as indication of invalid
718                          * (dangerous wrt exploits) chars */
719                         return orig + 1;
720                 }
721                 *string++ = value1;
722                 ptr += 2;
723         }
724         *string = '\0';
725         return orig;
726 }
727
728
729 #if ENABLE_FEATURE_HTTPD_CGI
```

```
730  /*****************************************************************************
731   * setenv helpers
732   *****************************************************************************/
733  static void setenv1(const char *name, const char *value)
734  {
735          if (!value)
736                  value = "";
737          setenv(name, value, 1);
738  }
739  static void setenv_long(const char *name, long value)
740  {
741          char buf[sizeof(value)*3 + 1];
742          sprintf(buf, "%ld", value);
743          setenv(name, buf, 1);
744  }
745  #endif
746
747  #if ENABLE_FEATURE_HTTPD_BASIC_AUTH
748  /*****************************************************************************
749   *
750   > $Function: decodeBase64()
751   *
752   > $Description: Decode a base 64 data stream as per rfc1521.
753   *    Note that the rfc states that none base64 chars are to be ignored.
754   *    Since the decode always results in a shorter size than the input, it is
755   *    OK to pass the input arg as an output arg.
756   *
757   * $Parameter:
758   *      (char *) Data . . . . A pointer to a base64 encoded string.
759   *                            Where to place the decoded data.
760   *
761   * $Return: void
762   *
763   * $Errors: None
764   *
```

```
765  ********************************************************************************/
766 static void decodeBase64(char *Data)
767 {
768         const unsigned char *in = (const unsigned char *)Data;
769         // The decoded size will be at most 3/4 the size of the encoded
770         unsigned ch = 0;
771         int i = 0;
772
773         while (*in) {
774                 int t = *in++;
775
776                 if (t >= '0' && t <= '9')
777                         t = t - '0' + 52;
778                 else if (t >= 'A' && t <= 'Z')
779                         t = t - 'A';
780                 else if (t >= 'a' && t <= 'z')
781                         t = t - 'a' + 26;
782                 else if (t == '+')
783                         t = 62;
784                 else if (t == '/')
785                         t = 63;
786                 else if (t == '=')
787                         t = 0;
788                 else
789                         continue;
790
791                 ch = (ch << 6) | t;
792                 i++;
793                 if (i == 4) {
794                         *Data++ = (char) (ch >> 16);
795                         *Data++ = (char) (ch >> 8);
796                         *Data++ = (char) ch;
797                         i = 0;
798                 }
799         }
```

```
800         *Data = '\0';
801 }
802 #endif
803
804
805 /*****************************************************************************
806  *
807  > $Function: openServer()
808  *
809  * $Description: create a listen server socket on the designated port.
810  *
811  * $Return: (int)  . . . A connection socket. -1 for errors.
812  *
813  * $Errors: None
814  *
815  *****************************************************************************/
816 static int openServer(void)
817 {
818         int fd;
819
820         /* create the socket right now */
821         fd = create_and_bind_stream_or_die(NULL, config->port);
822         xlisten(fd, 9);
823         return fd;
824 }
825
826 /*****************************************************************************
827  *
828  > $Function: sendHeaders()
829  *
830  * $Description: Create and send HTTP response headers.
831  *   The arguments are combined and sent as one write operation.  Note that
832  *   IE will puke big-time if the headers are not sent in one packet and the
833  *   second packet is delayed for any reason.
834  *
```

```
 * $Parameter:
 *      (HttpResponseNum) responseNum . . . The result code to send.
 *
 * $Return: (int)  . . . . writing errors
 *
 ************************************************************************/
static int sendHeaders(HttpResponseNum responseNum)
{
        char *buf = config->buf;
        const char *responseString = "";
        const char *infoString = 0;
        const char *mime_type;
        unsigned i;
        time_t timer = time(0);
        char timeStr[80];
        int len;
        enum {
                numNames = sizeof(httpResponseNames) / sizeof(httpResponseNames[0])
        };

        for (i = 0; i < numNames; i++) {
                if (httpResponseNames[i].type == responseNum) {
                        responseString = httpResponseNames[i].name;
                        infoString = httpResponseNames[i].info;
                        break;
                }
        }
        /* error message is HTML */
        mime_type = responseNum == HTTP_OK ?
                                config->found_mime_type : "text/html";

        /* emit the current date */
        strftime(timeStr, sizeof(timeStr), RFC1123FMT, gmtime(&timer));
        len = sprintf(buf,
                        "HTTP/1.0 %d %s\r\nContent-type: %s\r\n"
```

```
870                         "Date: %s\r\nConnection: close\r\n",
871                         responseNum, responseString, mime_type, timeStr);
872
873 #if ENABLE_FEATURE_HTTPD_BASIC_AUTH
874         if (responseNum == HTTP_UNAUTHORIZED) {
875                 len += sprintf(buf+len,
876                                 "WWW-Authenticate: Basic realm=\"%s\"\r\n",
877                                 config->realm);
878         }
879 #endif
880         if (responseNum == HTTP_MOVED_TEMPORARILY) {
881                 len += sprintf(buf+len, "Location: %s/%s%s\r\n",
882                                 config->found_moved_temporarily,
883                                 (config->query ? "?" : ""),
884                                 (config->query ? config->query : ""));
885         }
886
887         if (config->ContentLength != -1) {    /* file */
888                 strftime(timeStr, sizeof(timeStr), RFC1123FMT, gmtime(&config->last_mod));
889                 len += sprintf(buf+len, "Last-Modified: %s\r\n%s %"OFF_FMT"d\r\n",
890                         timeStr, "Content-length:", config->ContentLength);
891         }
892         strcat(buf, "\r\n");
893         len += 2;
894         if (infoString) {
895                 len += sprintf(buf+len,
896                                 "<HEAD><TITLE>%d %s</TITLE></HEAD>\n"
897                                 "<BODY><H1>%d %s</H1>\n%s\n</BODY>\n",
898                                 responseNum, responseString,
899                                 responseNum, responseString, infoString);
900         }
901         if (DEBUG)
902                 fprintf(stderr, "headers: '%s'\n", buf);
903         i = config->accepted_socket;
904         if (i == 0) i++; /* write to fd# 1 in inetd mode */
```

```
905         return full_write(i, buf, len);
906 }
907
908 /*********************************************************************
909  *
910  > $Function: getLine()
911  *
912  * $Description: Read from the socket until an end of line char found.
913  *
914  *   Characters are read one at a time until an eol sequence is found.
915  *
916  * $Return: (int) . . . . number of characters read.  -1 if error.
917  *
918  *********************************************************************/
919 static int getLine(void)
920 {
921         int count = 0;
922         char *buf = config->buf;
923
924         while (read(config->accepted_socket, buf + count, 1) == 1) {
925                 if (buf[count] == '\r') continue;
926                 if (buf[count] == '\n') {
927                         buf[count] = 0;
928                         return count;
929                 }
930                 if (count < (MAX_MEMORY_BUFF-1))      /* check overflow */
931                         count++;
932         }
933         if (count) return count;
934         else return -1;
935 }
936
937 #if ENABLE_FEATURE_HTTPD_CGI
938 /*********************************************************************
939  *
```

```
 > $Function: sendCgi()
 *
 * $Description: Execute a CGI script and send it's stdout back
 *
 *   Environment variables are set up and the script is invoked with pipes
 *   for stdin/stdout.  If a post is being done the script is fed the POST
 *   data in addition to setting the QUERY_STRING variable (for GETs or POSTs).
 *
 * $Parameters:
 *      (const char *) url . . . . . The requested URL (with leading /).
 *      (int bodyLen)  . . . . . Length of the post body.
 *      (const char *cookie) . . . . . For set HTTP_COOKIE.
 *      (const char *content_type) . . For set CONTENT_TYPE.
 *
 * $Return: (char *)  . . . . A pointer to the decoded string (same as input).
 *
 * $Errors: None
 *
 ******************************************************************************/
static int sendCgi(const char *url,
               const char *request, int bodyLen, const char *cookie,
               const char *content_type)
{
        int fromCgi[2];  /* pipe for reading data from CGI */
        int toCgi[2];    /* pipe for sending data to CGI */

        static char * argp[] = { 0, 0 };
        int pid = 0;
        int inFd;
        int outFd;
        int buf_count;
        int status;
        size_t post_read_size, post_read_idx;

        if (pipe(fromCgi) != 0)
```

```
975                 return 0;
976         if (pipe(toCgi) != 0)
977                 return 0;
978
979 /*
980  * Note: We can use vfork() here in the no-mmu case, although
981  * the child modifies the parent's variables, due to:
982  * 1) The parent does not use the child-modified variables.
983  * 2) The allocated memory (in the child) is freed when the process
984  *    exits. This happens instantly after the child finishes,
985  *    since httpd is run from inetd (and it can't run standalone
986  *    in uClinux).
987  */
988 #if !BB_MMU
989         pid = vfork();
990 #else
991         pid = fork();
992 #endif
993         if (pid < 0)
994                 return 0;
995
996         if (!pid) {
997                 /* child process */
998                 char *script;
999                 char *purl;
1000                char realpath_buff[MAXPATHLEN];
1001
1002                if (config->accepted_socket > 1)
1003                        close(config->accepted_socket);
1004                if (config->server_socket > 1)
1005                        close(config->server_socket);
1006
1007                dup2(toCgi[0], 0);  // replace stdin with the pipe
1008                dup2(fromCgi[1], 1);  // replace stdout with the pipe
1009                /* Huh? User seeing stderr can be a security problem...
```

```
1010                 * and if CGI really wants that, it can always dup2(1,2)...
1011                if (!DEBUG)
1012                        dup2(fromCgi[1], 2);  // replace stderr with the pipe
1013                */
1014                /* I think we cannot inadvertently close 0, 1 here... */
1015                close(toCgi[0]);
1016                close(toCgi[1]);
1017                close(fromCgi[0]);
1018                close(fromCgi[1]);
1019
1020                /*
1021                 * Find PATH_INFO.
1022                 */
1023                xfunc_error_retval = 242;
1024                purl = xstrdup(url);
1025                script = purl;
1026                while ((script = strchr(script + 1, '/')) != NULL) {
1027                        /* have script.cgi/PATH_INFO or dirs/script.cgi[/PATH_INFO] */
1028                        struct stat sb;
1029
1030                        *script = '\0';
1031                        if (is_directory(purl + 1, 1, &sb) == 0) {
1032                                /* not directory, found script.cgi/PATH_INFO */
1033                                *script = '/';
1034                                break;
1035                        }
1036                        *script = '/';          /* is directory, find next '/' */
1037                }
1038                setenv1("PATH_INFO", script);   /* set /PATH_INFO or "" */
1039                /* setenv1("PATH", getenv("PATH")); redundant */
1040                setenv1("REQUEST_METHOD", request);
1041                if (config->query) {
1042                        char *uri = alloca(strlen(purl) + 2 + strlen(config->query));
1043                        if (uri)
1044                                sprintf(uri, "%s?%s", purl, config->query);
```

```
1045                    setenv1("REQUEST_URI", uri);
1046            } else {
1047                    setenv1("REQUEST_URI", purl);
1048            }
1049            if (script != NULL)
1050                    *script = '\0';         /* cut off /PATH_INFO */
1051             /* SCRIPT_FILENAME required by PHP in CGI mode */
1052            if (!realpath(purl + 1, realpath_buff))
1053                    goto error_execing_cgi;
1054            setenv1("SCRIPT_FILENAME", realpath_buff);
1055            /* set SCRIPT_NAME as full path: /cgi-bin/dirs/script.cgi */
1056            setenv1("SCRIPT_NAME", purl);
1057            /* http://hoohoo.ncsa.uiuc.edu/cgi/env.html:
1058             * QUERY_STRING: The information which follows the ? in the URL
1059             * which referenced this script. This is the query information.
1060             * It should not be decoded in any fashion. This variable
1061             * should always be set when there is query information,
1062             * regardless of command line decoding. */
1063            /* (Older versions of bbox seem to do some decoding) */
1064            setenv1("QUERY_STRING", config->query);
1065            setenv1("SERVER_SOFTWARE", httpdVersion);
1066            putenv((char*)"SERVER_PROTOCOL=HTTP/1.0");
1067            putenv((char*)"GATEWAY_INTERFACE=CGI/1.1");
1068            /* Having _separate_ variables for IP and port defeats
1069             * the purpose of having socket abstraction. Which "port"
1070             * are you using on Unix domain socket?
1071             * IOW - REMOTE_PEER="1.2.3.4:56" makes much more sense.
1072             * Oh well... */
1073            {
1074                    char *p = config->rmt_ip_str ? : (char*)"";
1075                    char *cp = strrchr(p, ':');
1076                    if (ENABLE_FEATURE_IPV6 && cp && strchr(cp, ']'))
1077                            cp = NULL;
1078                    if (cp) *cp = '\0'; /* delete :PORT */
1079                    setenv1("REMOTE_ADDR", p);
```

```
1080                }
1081    #if ENABLE_FEATURE_HTTPD_SET_REMOTE_PORT_TO_ENV
1082                setenv_long("REMOTE_PORT", config->port);
1083    #endif
1084                if (bodyLen)
1085                        setenv_long("CONTENT_LENGTH", bodyLen);
1086                if (cookie)
1087                        setenv1("HTTP_COOKIE", cookie);
1088                if (content_type)
1089                        setenv1("CONTENT_TYPE", content_type);
1090    #if ENABLE_FEATURE_HTTPD_BASIC_AUTH
1091                if (config->remoteuser) {
1092                        setenv1("REMOTE_USER", config->remoteuser);
1093                        putenv((char*)"AUTH_TYPE=Basic");
1094                }
1095    #endif
1096                if (config->referer)
1097                        setenv1("HTTP_REFERER", config->referer);
1098
1099                /* set execve argp[0] without path */
1100                argp[0] = strrchr(purl, '/') + 1;
1101                /* but script argp[0] must have absolute path and chdiring to this */
1102                script = strrchr(realpath_buff, '/');
1103                if (!script)
1104                        goto error_execing_cgi;
1105                *script = '\0';
1106                if (chdir(realpath_buff) == 0) {
1107                        // Now run the program.  If it fails,
1108                        // use _exit() so no destructors
1109                        // get called and make a mess.
1110    #if ENABLE_FEATURE_HTTPD_CONFIG_WITH_SCRIPT_INTERPR
1111                        char *interpr = NULL;
1112                        char *suffix = strrchr(purl, '.');
1113
1114                        if (suffix) {
```

```
1115                                    Htaccess *cur;
1116                                    for (cur = config->script_i; cur; cur = cur->next) {
1117                                            if (strcmp(cur->before_colon + 1, suffix) == 0) {
1118                                                    interpr = cur->after_colon;
1119                                                    break;
1120                                            }
1121                                    }
1122                            }
1123    #endif
1124                            *script = '/';
1125    #if ENABLE_FEATURE_HTTPD_CONFIG_WITH_SCRIPT_INTERPR
1126                            if (interpr)
1127                                    execv(interpr, argp);
1128                            else
1129    #endif
1130                                    execv(realpath_buff, argp);
1131                    }
1132     error_execing_cgi:
1133                    /* send to stdout (even if we are not from inetd) */
1134                    config->accepted_socket = 1;
1135                    sendHeaders(HTTP_NOT_FOUND);
1136                    _exit(242);
1137            } /* end child */
1138
1139            /* parent process */
1140
1141            buf_count = 0;
1142            post_read_size = 0;
1143            post_read_idx = 0; /* for gcc */
1144            inFd = fromCgi[0];
1145            outFd = toCgi[1];
1146            close(fromCgi[1]);
1147            close(toCgi[0]);
1148            signal(SIGPIPE, SIG_IGN);
1149
```

```
1150        while (1) {
1151                fd_set readSet;
1152                fd_set writeSet;
1153                char wbuf[128];
1154                int nfound;
1155                int count;
1156
1157                FD_ZERO(&readSet);
1158                FD_ZERO(&writeSet);
1159                FD_SET(inFd, &readSet);
1160                if (bodyLen > 0 || post_read_size > 0) {
1161                        FD_SET(outFd, &writeSet);
1162                        nfound = outFd > inFd ? outFd : inFd;
1163                        if (post_read_size == 0) {
1164                                FD_SET(config->accepted_socket, &readSet);
1165                                if (nfound < config->accepted_socket)
1166                                        nfound = config->accepted_socket;
1167                        }
1168                        /* Now wait on the set of sockets! */
1169                        nfound = select(nfound + 1, &readSet, &writeSet, NULL, NULL);
1170                } else {
1171                        if (!bodyLen) {
1172                                close(outFd); /* no more POST data to CGI */
1173                                bodyLen = -1;
1174                        }
1175                        nfound = select(inFd + 1, &readSet, NULL, NULL, NULL);
1176                }
1177
1178                if (nfound <= 0) {
1179                        if (waitpid(pid, &status, WNOHANG) <= 0) {
1180                                /* Weird. CGI didn't exit and no fd's
1181                                 * are ready, yet select returned?! */
1182                                continue;
1183                        }
1184                        close(inFd);
```

```
1185                    if (DEBUG && WIFEXITED(status))
1186                            bb_error_msg("piped has exited with status=%d", WEXITSTATUS(status));
1187                    if (DEBUG && WIFSIGNALED(status))
1188                            bb_error_msg("piped has exited with signal=%d", WTERMSIG(status));
1189                    break;
1190            }
1191
1192            if (post_read_size > 0 && FD_ISSET(outFd, &writeSet)) {
1193                    /* Have data from peer and can write to CGI */
1194            // huh? why full_write? what if we will block?
1195            // (imagine that CGI does not read its stdin...)
1196                    count = full_write(outFd, wbuf + post_read_idx, post_read_size);
1197                    if (count > 0) {
1198                            post_read_idx += count;
1199                            post_read_size -= count;
1200                    } else {
1201                            post_read_size = bodyLen = 0; /* broken pipe to CGI */
1202                    }
1203            } else if (bodyLen > 0 && post_read_size == 0
1204             && FD_ISSET(config->accepted_socket, &readSet)
1205            ) {
1206                    /* We expect data, prev data portion is eaten by CGI
1207                     * and there *is* data to read from the peer
1208                     * (POSTDATA?) */
1209                    count = bodyLen > (int)sizeof(wbuf) ? (int)sizeof(wbuf) : bodyLen;
1210                    count = safe_read(config->accepted_socket, wbuf, count);
1211                    if (count > 0) {
1212                            post_read_size = count;
1213                            post_read_idx = 0;
1214                            bodyLen -= count;
1215                    } else {
1216                            bodyLen = 0;    /* closed */
1217                    }
1218            }
1219
```

```
#define PIPESIZE PIPE_BUF
#if PIPESIZE >= MAX_MEMORY_BUFF
# error "PIPESIZE >= MAX_MEMORY_BUFF"
#endif
                if (FD_ISSET(inFd, &readSet)) {
                        /* There is something to read from CGI */
                        int s = config->accepted_socket;
                        char *rbuf = config->buf;

                        /* Are we still buffering CGI output? */
                        if (buf_count >= 0) {
                                static const char HTTP_200[] = "HTTP/1.0 200 OK\r\n";
                                /* Must use safe_read, not full_read, because
                                 * CGI may output a few first bytes and then wait
                                 * for POSTDATA without closing stdout.
                                 * With full_read we may wait here forever. */
                                count = safe_read(inFd, rbuf + buf_count, PIPESIZE - 4);
                                if (count <= 0) {
                                        /* eof (or error) and there was no "HTTP",
                                         * so add one and write out the received data */
                                        if (buf_count) {
                                                full_write(s, HTTP_200, sizeof(HTTP_200)-1);
                                                full_write(s, rbuf, buf_count);
                                        }
                                        break;  /* closed */
                                }
                                buf_count += count;
                                count = 0;
                                if (buf_count >= 4) {
                                        /* check to see if CGI added "HTTP" */
                                        if (memcmp(rbuf, HTTP_200, 4) != 0) {
                                                /* there is no "HTTP", do it ourself */
                                                if (full_write(s, HTTP_200, sizeof(HTTP_200)-1) !=
sizeof(HTTP_200)-1)
                                                        break;
```

```
1255                                    }
1256                                    /* example of valid CGI without "Content-type:"
1257                                     * echo -en "HTTP/1.0 302 Found\r\n"
1258                                     * echo -en "Location: http://www.busybox.net\r\n"
1259                                     * echo -en "\r\n"
1260                                    if (!strstr(rbuf, "ontent-")) {
1261                                            full_write(s, "Content-type: text/plain\r\n\r\n", 28);
1262                                    }
1263                                     */
1264                                    count = buf_count;
1265                                    buf_count = -1; /* buffering off */
1266                            }
1267                    } else {
1268                            count = safe_read(inFd, rbuf, PIPESIZE);
1269                            if (count <= 0)
1270                                    break;  /* eof (or error) */
1271                    }
1272                    if (full_write(s, rbuf, count) != count)
1273                            break;
1274                    if (DEBUG)
1275                            fprintf(stderr, "cgi read %d bytes: '%.*s'\n", count, count, rbuf);
1276            } /* if (FD_ISSET(inFd)) */
1277    } /* while (1) */
1278    return 0;
1279 }
1280 #endif          /* FEATURE_HTTPD_CGI */
1281
1282 /*****************************************************************************
1283  *
1284  > $Function: sendFile()
1285  *
1286  * $Description: Send a file response to a HTTP request
1287  *
1288  * $Parameter:
1289  *      (const char *) url . . The URL requested.
```

```
 *
 * $Return: (int)  . . . . . Always 0.
 *
 ****************************************************************************/
static int sendFile(const char *url)
{
        char * suffix;
        int  f;
        const char * const * table;
        const char * try_suffix;

        suffix = strrchr(url, '.');

        for (table = suffixTable; *table; table += 2)
                if (suffix != NULL && (try_suffix = strstr(*table, suffix)) != 0) {
                        try_suffix += strlen(suffix);
                        if (*try_suffix == 0 || *try_suffix == '.')
                                break;
                }
        /* also, if not found, set default as "application/octet-stream";  */
        config->found_mime_type = table[1];
#if ENABLE_FEATURE_HTTPD_CONFIG_WITH_MIME_TYPES
        if (suffix) {
                Htaccess * cur;

                for (cur = config->mime_a; cur; cur = cur->next) {
                        if (strcmp(cur->before_colon, suffix) == 0) {
                                config->found_mime_type = cur->after_colon;
                                break;
                        }
                }
        }
#endif  /* FEATURE_HTTPD_CONFIG_WITH_MIME_TYPES */

        if (DEBUG)
```

```
1325                fprintf(stderr, "sending file '%s' content-type: %s\n",
1326                        url, config->found_mime_type);
1327
1328        f = open(url, O_RDONLY);
1329        if (f >= 0) {
1330                int count;
1331                char *buf = config->buf;
1332
1333                sendHeaders(HTTP_OK);
1334                /* TODO: sendfile() */
1335                while ((count = full_read(f, buf, MAX_MEMORY_BUFF)) > 0) {
1336                        int fd = config->accepted_socket;
1337                        if (fd == 0) fd++; /* write to fd# 1 in inetd mode */
1338                        if (full_write(fd, buf, count) != count)
1339                                break;
1340                }
1341                close(f);
1342        } else {
1343                if (DEBUG)
1344                        bb_perror_msg("cannot open '%s'", url);
1345                sendHeaders(HTTP_NOT_FOUND);
1346        }
1347
1348        return 0;
1349  }
1350
1351  static int checkPermIP(void)
1352  {
1353        Htaccess_IP * cur;
1354
1355        /* This could stand some work */
1356        for (cur = config->ip_a_d; cur; cur = cur->next) {
1357  #if ENABLE_FEATURE_HTTPD_CGI && DEBUG
1358                fprintf(stderr, "checkPermIP: '%s' ? ", config->rmt_ip_str);
1359  #endif
```

```
#if DEBUG
                fprintf(stderr, "'%u.%u.%u.%u/%u.%u.%u.%u'\n",
                        (unsigned char)(cur->ip >> 24),
                        (unsigned char)(cur->ip >> 16),
                        (unsigned char)(cur->ip >> 8),
                        (unsigned char)(cur->ip),
                        (unsigned char)(cur->mask >> 24),
                        (unsigned char)(cur->mask >> 16),
                        (unsigned char)(cur->mask >> 8),
                        (unsigned char)(cur->mask)
                );
#endif
                if ((config->rmt_ip & cur->mask) == cur->ip)
                        return cur->allow_deny == 'A';   /* Allow/Deny */
        }

        /* if unconfigured, return 1 - access from all */
        return !config->flg_deny_all;
}

/************************************************************************
 *
 > $Function: checkPerm()
 *
 * $Description: Check the permission file for access password protected.
 *
 *   If config file isn't present, everything is allowed.
 *   Entries are of the form you can see example from header source
 *
 * $Parameters:
 *      (const char *) path  . . . . The file path.
 *      (const char *) request . . . User information to validate.
 *
 * $Return: (int)  . . . . . . . . . 1 if request OK, 0 otherwise.
 *
```

```
 ****************************************************************************/

#if ENABLE_FEATURE_HTTPD_BASIC_AUTH
static int checkPerm(const char *path, const char *request)
{
        Htaccess * cur;
        const char *p;
        const char *p0;

        const char *prev = NULL;

        /* This could stand some work */
        for (cur = config->auth; cur; cur = cur->next) {
                size_t l;

                p0 = cur->before_colon;
                if (prev != NULL && strcmp(prev, p0) != 0)
                        continue;       /* find next identical */
                p = cur->after_colon;
                if (DEBUG)
                        fprintf(stderr, "checkPerm: '%s' ? '%s'\n", p0, request);

                l = strlen(p0);
                if (strncmp(p0, path, l) == 0
                 && (l == 1 || path[l] == '/' || path[l] == '\0')
                ) {
                        char *u;
                        /* path match found.  Check request */
                        /* for check next /path:user:password */
                        prev = p0;
                        u = strchr(request, ':');
                        if (u == NULL) {
                                /* bad request, ':' required */
                                break;
                        }
```

```
1430
1431                        if (ENABLE_FEATURE_HTTPD_AUTH_MD5) {
1432                                char *cipher;
1433                                char *pp;
1434
1435                                if (strncmp(p, request, u-request) != 0) {
1436                                        /* user uncompared */
1437                                        continue;
1438                                }
1439                                pp = strchr(p, ':');
1440                                if (pp && pp[1] == '$' && pp[2] == '1' &&
1441                                                pp[3] == '$' && pp[4]) {
1442                                        pp++;
1443                                        cipher = pw_encrypt(u+1, pp);
1444                                        if (strcmp(cipher, pp) == 0)
1445                                                goto set_remoteuser_var;   /* Ok */
1446                                        /* unauthorized */
1447                                        continue;
1448                                }
1449                        }
1450
1451                        if (strcmp(p, request) == 0) {
1452    set_remoteuser_var:
1453                                config->remoteuser = strdup(request);
1454                                if (config->remoteuser)
1455                                        config->remoteuser[(u - request)] = 0;
1456                                return 1;   /* Ok */
1457                        }
1458                        /* unauthorized */
1459                }
1460        }   /* for */
1461
1462        return prev == NULL;
1463    }
1464
```

```
1465  #endif  /* FEATURE_HTTPD_BASIC_AUTH */
1466
1467  /*****************************************************************************
1468   *
1469   > $Function: handle_sigalrm()
1470   *
1471   * $Description: Handle timeouts
1472   *
1473   *****************************************************************************/
1474
1475  static void handle_sigalrm(int sig)
1476  {
1477          sendHeaders(HTTP_REQUEST_TIMEOUT);
1478          config->alarm_signaled = sig;
1479  }
1480
1481  /*****************************************************************************
1482   *
1483   > $Function: handleIncoming()
1484   *
1485   * $Description: Handle an incoming http request.
1486   *
1487   *****************************************************************************/
1488  static void handleIncoming(void)
1489  {
1490          char *buf = config->buf;
1491          char *url;
1492          char *purl;
1493          int  blank = -1;
1494          char *test;
1495          struct stat sb;
1496          int ip_allowed;
1497  #if ENABLE_FEATURE_HTTPD_CGI
1498          const char *prequest = request_GET;
1499          unsigned long length = 0;
```

```
1500         char *cookie = 0;
1501         char *content_type = 0;
1502 #endif
1503         fd_set s_fd;
1504         struct timeval tv;
1505         int retval;
1506         struct sigaction sa;
1507
1508 #if ENABLE_FEATURE_HTTPD_BASIC_AUTH
1509         int credentials = -1;  /* if not required this is Ok */
1510 #endif
1511
1512         sa.sa_handler = handle_sigalrm;
1513         sigemptyset(&sa.sa_mask);
1514         sa.sa_flags = 0; /* no SA_RESTART */
1515         sigaction(SIGALRM, &sa, NULL);
1516
1517         do {
1518                 int count;
1519
1520                 (void) alarm(TIMEOUT);
1521                 if (getLine() <= 0)
1522                         break;  /* closed */
1523
1524                 purl = strpbrk(buf, " \t");
1525                 if (purl == NULL) {
1526  BAD_REQUEST:
1527                         sendHeaders(HTTP_BAD_REQUEST);
1528                         break;
1529                 }
1530                 *purl = '\0';
1531 #if ENABLE_FEATURE_HTTPD_CGI
1532                 if (strcasecmp(buf, prequest) != 0) {
1533                         prequest = "POST";
1534                         if (strcasecmp(buf, prequest) != 0) {
```

```
1535                            sendHeaders(HTTP_NOT_IMPLEMENTED);
1536                            break;
1537                    }
1538            }
1539    #else
1540            if (strcasecmp(buf, request_GET) != 0) {
1541                    sendHeaders(HTTP_NOT_IMPLEMENTED);
1542                    break;
1543            }
1544    #endif
1545            *purl = ' ';
1546            count = sscanf(purl, " %[^ ] HTTP/%d.%*d", buf, &blank);
1547
1548            if (count < 1 || buf[0] != '/') {
1549                    /* Garbled request/URL */
1550                    goto BAD_REQUEST;
1551            }
1552            url = alloca(strlen(buf) + sizeof("/index.html"));
1553            if (url == NULL) {
1554                    sendHeaders(HTTP_INTERNAL_SERVER_ERROR);
1555                    break;
1556            }
1557            strcpy(url, buf);
1558            /* extract url args if present */
1559            test = strchr(url, '?');
1560            config->query = NULL;
1561            if (test) {
1562                    *test++ = '\0';
1563                    config->query = test;
1564            }
1565
1566            test = decodeString(url, 0);
1567            if (test == NULL)
1568                    goto BAD_REQUEST;
1569            if (test == url+1) {
```

```
1570                    /* '/' or NUL is encoded */
1571                    sendHeaders(HTTP_NOT_FOUND);
1572                    break;
1573            }
1574
1575            /* algorithm stolen from libbb bb_simplify_path(),
1576             * but don't strdup and reducing trailing slash and protect out root */
1577            purl = test = url;
1578            do {
1579                    if (*purl == '/') {
1580                            /* skip duplicate (or initial) slash */
1581                            if (*test == '/') {
1582                                    continue;
1583                            }
1584                            if (*test == '.') {
1585                                    /* skip extra '.' */
1586                                    if (test[1] == '/' || !test[1]) {
1587                                            continue;
1588                                    }
1589                                    /* '..': be careful */
1590                                    if (test[1] == '.' && (test[2] == '/' || !test[2])) {
1591                                            ++test;
1592                                            if (purl == url) {
1593                                                    /* protect out root */
1594                                                    goto BAD_REQUEST;
1595                                            }
1596                                            while (*--purl != '/') /* omit previous dir */;
1597                                                    continue;
1598                                    }
1599                            }
1600                    }
1601                    *++purl = *test;
1602            } while (*++test);
1603            *++purl = '\0';       /* so keep last character */
1604            test = purl;          /* end ptr */
```

```
1605
1606                    /* If URL is directory, adding '/' */
1607                    if (test[-1] != '/') {
1608                            if (is_directory(url + 1, 1, &sb)) {
1609                                    config->found_moved_temporarily = url;
1610                            }
1611                    }
1612                    if (DEBUG)
1613                            fprintf(stderr, "url='%s', args=%s\n", url, config->query);
1614
1615                    test = url;
1616                    ip_allowed = checkPermIP();
1617                    while (ip_allowed && (test = strchr(test + 1, '/')) != NULL) {
1618                            /* have path1/path2 */
1619                            *test = '\0';
1620                            if (is_directory(url + 1, 1, &sb)) {
1621                                    /* may be having subdir config */
1622                                    parse_conf(url + 1, SUBDIR_PARSE);
1623                                    ip_allowed = checkPermIP();
1624                            }
1625                            *test = '/';
1626                    }
1627                    if (blank >= 0) {
1628                            /* read until blank line for HTTP version specified, else parse immediate */
1629                            while (1) {
1630                                    alarm(TIMEOUT);
1631                                    count = getLine();
1632                                    if (count <= 0)
1633                                            break;
1634
1635                                    if (DEBUG)
1636                                            fprintf(stderr, "header: '%s'\n", buf);
1637
1638    #if ENABLE_FEATURE_HTTPD_CGI
1639                                    /* try and do our best to parse more lines */
```

```
1640                                if ((STRNCASECMP(buf, "Content-length:") == 0)) {
1641                                        /* extra read only for POST */
1642                                        if (prequest != request_GET) {
1643                                                test = buf + sizeof("Content-length:")-1;
1644                                                if (!test[0])
1645                                                        goto bail_out;
1646                                                errno = 0;
1647                                                /* not using strtoul: it ignores leading munis! */
1648                                                length = strtol(test, &test, 10);
1649                                                /* length is "ulong", but we need to pass it to int later */
1650                                                /* so we check for negative or too large values in one go: */
1651                                                /* (long -> ulong conv caused negatives to be seen as > INT_MAX) */
1652                                                if (test[0] || errno || length > INT_MAX)
1653                                                        goto bail_out;
1654                                        }
1655                                } else if ((STRNCASECMP(buf, "Cookie:") == 0)) {
1656                                        cookie = strdup(skip_whitespace(buf + sizeof("Cookie:")-1));
1657                                } else if ((STRNCASECMP(buf, "Content-Type:") == 0)) {
1658                                        content_type = strdup(skip_whitespace(buf + sizeof("Content-Type:")-1));
1659                                } else if ((STRNCASECMP(buf, "Referer:") == 0)) {
1660                                        config->referer = strdup(skip_whitespace(buf + sizeof("Referer:")-1));
1661                                }
1662    #endif
1663
1664    #if ENABLE_FEATURE_HTTPD_BASIC_AUTH
1665                                if (STRNCASECMP(buf, "Authorization:") == 0) {
1666                                        /* We only allow Basic credentials.
1667                                         * It shows up as "Authorization: Basic <userid:password>" where
1668                                         * the userid:password is base64 encoded.
1669                                         */
1670                                        test = skip_whitespace(buf + sizeof("Authorization:")-1);
1671                                        if (STRNCASECMP(test, "Basic") != 0)
1672                                                continue;
1673                                        test += sizeof("Basic")-1;
1674                                        /* decodeBase64() skips whitespace itself */
```

```
1675                                    decodeBase64(test);
1676                                    credentials = checkPerm(url, test);
1677                                }
1678    #endif           /* FEATURE_HTTPD_BASIC_AUTH */
1679
1680                            } /* while extra header reading */
1681                    }
1682                    alarm(0);
1683                    if (config->alarm_signaled)
1684                            break;
1685
1686                    if (strcmp(strrchr(url, '/') + 1, httpd_conf) == 0 || ip_allowed == 0) {
1687                            /* protect listing [/path]/httpd_conf or IP deny */
1688    #if ENABLE_FEATURE_HTTPD_CGI
1689     FORBIDDEN:             /* protect listing /cgi-bin */
1690    #endif
1691                            sendHeaders(HTTP_FORBIDDEN);
1692                            break;
1693                    }
1694
1695    #if ENABLE_FEATURE_HTTPD_BASIC_AUTH
1696                    if (credentials <= 0 && checkPerm(url, ":") == 0) {
1697                            sendHeaders(HTTP_UNAUTHORIZED);
1698                            break;
1699                    }
1700    #endif
1701
1702                    if (config->found_moved_temporarily) {
1703                            sendHeaders(HTTP_MOVED_TEMPORARILY);
1704                            /* clear unforked memory flag */
1705                            config->found_moved_temporarily = NULL;
1706                            break;
1707                    }
1708
1709                    test = url + 1;      /* skip first '/' */
```

```
1710
1711  #if ENABLE_FEATURE_HTTPD_CGI
1712                  if (strncmp(test, "cgi-bin", 7) == 0) {
1713                          if (test[5] == '/' && test[8] == 0)
1714                                  goto FORBIDDEN;     /* protect listing cgi-bin/ */
1715                          sendCgi(url, prequest, length, cookie, content_type);
1716                          break;
1717                  }
1718  #if ENABLE_FEATURE_HTTPD_CONFIG_WITH_SCRIPT_INTERPR
1719                  {
1720                          char *suffix = strrchr(test, '.');
1721                          if (suffix) {
1722                                  Htaccess *cur;
1723                                  for (cur = config->script_i; cur; cur = cur->next) {
1724                                          if (strcmp(cur->before_colon + 1, suffix) == 0) {
1725                                                  sendCgi(url, prequest, length, cookie, content_type);
1726                                                  goto bail_out;
1727                                          }
1728                                  }
1729                          }
1730                  }
1731  #endif
1732                  if (prequest != request_GET) {
1733                          sendHeaders(HTTP_NOT_IMPLEMENTED);
1734                          break;
1735                  }
1736  #endif  /* FEATURE_HTTPD_CGI */
1737                  if (purl[-1] == '/')
1738                          strcpy(purl, "index.html");
1739                  if (stat(test, &sb) == 0) {
1740                          /* It's a dir URL and there is index.html */
1741                          config->ContentLength = sb.st_size;
1742                          config->last_mod = sb.st_mtime;
1743                  }
1744  #if ENABLE_FEATURE_HTTPD_CGI
```

```
1745                else if (purl[-1] == '/') {
1746                        /* It's a dir URL and there is no index.html
1747                         * Try cgi-bin/index.cgi */
1748                        if (access("/cgi-bin/index.cgi"+1, X_OK) == 0) {
1749                                purl[0] = '\0';
1750                                config->query = url;
1751                                sendCgi("/cgi-bin/index.cgi", prequest, length, cookie, content_type);
1752                                break;
1753                        }
1754                }
1755 #endif  /* FEATURE_HTTPD_CGI */
1756                 sendFile(test);
1757                 config->ContentLength = -1;
1758         } while (0);
1759
1760 #if ENABLE_FEATURE_HTTPD_CGI
1761  bail_out:
1762 #endif
1763
1764         if (DEBUG)
1765                 fprintf(stderr, "closing socket\n\n");
1766 #if ENABLE_FEATURE_HTTPD_CGI
1767         free(cookie);
1768         free(content_type);
1769         free(config->referer);
1770         config->referer = NULL;
1771 # if ENABLE_FEATURE_HTTPD_BASIC_AUTH
1772         free(config->remoteuser);
1773         config->remoteuser = NULL;
1774 # endif
1775 #endif
1776         shutdown(config->accepted_socket, SHUT_WR);
1777
1778         /* Properly wait for remote to closed */
1779         FD_ZERO(&s_fd);
```

```
1780          FD_SET(config->accepted_socket, &s_fd);
1781
1782          do {
1783                  tv.tv_sec = 2;
1784                  tv.tv_usec = 0;
1785                  retval = select(config->accepted_socket + 1, &s_fd, NULL, NULL, &tv);
1786          } while (retval > 0 && read(config->accepted_socket, buf, sizeof(config->buf) > 0));
1787
1788          shutdown(config->accepted_socket, SHUT_RD);
1789          /* In inetd case, we close fd 1 (stdout) here. We will exit soon anyway */
1790          close(config->accepted_socket);
1791  }
1792
1793  /*****************************************************************************
1794   *
1795   > $Function: miniHttpd()
1796   *
1797   * $Description: The main http server function.
1798   *
1799   *   Given an open socket fildes, listen for new connections and farm out
1800   *   the processing as a forked process.
1801   *
1802   * $Parameters:
1803   *      (int) server. . . The server socket fildes.
1804   *
1805   * $Return: (int) . . . . Always 0.
1806   *
1807   *****************************************************************************/
1808  static int miniHttpd(int server)
1809  {
1810          fd_set readfd, portfd;
1811
1812          FD_ZERO(&portfd);
1813          FD_SET(server, &portfd);
1814
```

```
1815          /* copy the ports we are watching to the readfd set */
1816          while (1) {
1817                  int s;
1818                  union {
1819                          struct sockaddr sa;
1820                          struct sockaddr_in sin;
1821                          USE_FEATURE_IPV6(struct sockaddr_in6 sin6;)
1822                  } fromAddr;
1823                  socklen_t fromAddrLen = sizeof(fromAddr);
1824
1825                  /* Now wait INDEFINITELY on the set of sockets! */
1826                  readfd = portfd;
1827                  if (select(server + 1, &readfd, 0, 0, 0) <= 0)
1828                          continue;
1829                  if (!FD_ISSET(server, &readfd))
1830                          continue;
1831                  s = accept(server, &fromAddr.sa, &fromAddrLen);
1832                  if (s < 0)
1833                          continue;
1834                  config->accepted_socket = s;
1835                  config->rmt_ip = 0;
1836                  config->port = 0;
1837  #if ENABLE_FEATURE_HTTPD_CGI || DEBUG
1838                  free(config->rmt_ip_str);
1839                  config->rmt_ip_str = xmalloc_sockaddr2dotted(&fromAddr.sa, fromAddrLen);
1840  #if DEBUG
1841                  bb_error_msg("connection from '%s'", config->rmt_ip_str);
1842  #endif
1843  #endif /* FEATURE_HTTPD_CGI */
1844                  if (fromAddr.sa.sa_family == AF_INET) {
1845                          config->rmt_ip = ntohl(fromAddr.sin.sin_addr.s_addr);
1846                          config->port = ntohs(fromAddr.sin.sin_port);
1847                  }
1848  #if ENABLE_FEATURE_IPV6
1849                  if (fromAddr.sa.sa_family == AF_INET6) {
```

```
1850                        //config->rmt_ip = ntohl(fromAddr.sin.sin_addr.s_addr);
1851                        config->port = ntohs(fromAddr.sin6.sin6_port);
1852                }
1853    #endif
1854
1855                /* set the KEEPALIVE option to cull dead connections */
1856                setsockopt(s, SOL_SOCKET, SO_KEEPALIVE, &const_int_1, sizeof(const_int_1));
1857
1858                if (DEBUG || fork() == 0) {
1859                        /* child */
1860    #if ENABLE_FEATURE_HTTPD_RELOAD_CONFIG_SIGHUP
1861                        /* protect reload config, may be confuse checking */
1862                        signal(SIGHUP, SIG_IGN);
1863    #endif
1864                        handleIncoming();
1865                        if (!DEBUG)
1866                                exit(0);
1867                }
1868                close(s);
1869        } /* while (1) */
1870        return 0;
1871    }
1872
1873    /* from inetd */
1874    static int miniHttpd_inetd(void)
1875    {
1876        union {
1877                struct sockaddr sa;
1878                struct sockaddr_in sin;
1879                USE_FEATURE_IPV6(struct sockaddr_in6 sin6;)
1880        } fromAddr;
1881        socklen_t fromAddrLen = sizeof(fromAddr);
1882
1883        getpeername(0, &fromAddr.sa, &fromAddrLen);
1884        config->rmt_ip = 0;
```

```
1885            config->port = 0;
1886    #if ENABLE_FEATURE_HTTPD_CGI || DEBUG
1887            free(config->rmt_ip_str);
1888            config->rmt_ip_str = xmalloc_sockaddr2dotted(&fromAddr.sa, fromAddrLen);
1889    #endif
1890            if (fromAddr.sa.sa_family == AF_INET) {
1891                    config->rmt_ip = ntohl(fromAddr.sin.sin_addr.s_addr);
1892                    config->port = ntohs(fromAddr.sin.sin_port);
1893            }
1894    #if ENABLE_FEATURE_IPV6
1895            if (fromAddr.sa.sa_family == AF_INET6) {
1896                    //config->rmt_ip = ntohl(fromAddr.sin.sin_addr.s_addr);
1897                    config->port = ntohs(fromAddr.sin6.sin6_port);
1898            }
1899    #endif
1900            handleIncoming();
1901            return 0;
1902    }
1903
1904    #if ENABLE_FEATURE_HTTPD_RELOAD_CONFIG_SIGHUP
1905    static void sighup_handler(int sig)
1906    {
1907            /* set and reset */
1908            struct sigaction sa;
1909
1910            parse_conf(default_path_httpd_conf, sig == SIGHUP ? SIGNALED_PARSE : FIRST_PARSE);
1911            sa.sa_handler = sighup_handler;
1912            sigemptyset(&sa.sa_mask);
1913            sa.sa_flags = SA_RESTART;
1914            sigaction(SIGHUP, &sa, NULL);
1915    }
1916    #endif
1917
1918    enum {
1919            c_opt_config_file = 0,
```

```
1920            d_opt_decode_url,
1921            h_opt_home_httpd,
1922            USE_FEATURE_HTTPD_ENCODE_URL_STR(e_opt_encode_url,)
1923            USE_FEATURE_HTTPD_BASIC_AUTH(    r_opt_realm     ,)
1924            USE_FEATURE_HTTPD_AUTH_MD5(      m_opt_md5       ,)
1925            USE_FEATURE_HTTPD_SETUID(        u_opt_setuid    ,)
1926            p_opt_port      ,
1927            p_opt_inetd     ,
1928            p_opt_foreground,
1929            OPT_CONFIG_FILE = 1 << c_opt_config_file,
1930            OPT_DECODE_URL  = 1 << d_opt_decode_url,
1931            OPT_HOME_HTTPD  = 1 << h_opt_home_httpd,
1932            OPT_ENCODE_URL  = USE_FEATURE_HTTPD_ENCODE_URL_STR((1 << e_opt_encode_url)) + 0,
1933            OPT_REALM       = USE_FEATURE_HTTPD_BASIC_AUTH(    (1 << r_opt_realm     )) + 0,
1934            OPT_MD5         = USE_FEATURE_HTTPD_AUTH_MD5(      (1 << m_opt_md5       )) + 0,
1935            OPT_SETUID      = USE_FEATURE_HTTPD_SETUID(        (1 << u_opt_setuid    )) + 0,
1936            OPT_PORT        = 1 << p_opt_port,
1937            OPT_INETD       = 1 << p_opt_inetd,
1938            OPT_FOREGROUND  = 1 << p_opt_foreground,
1939    };
1940
1941
1942    int httpd_main(int argc, char **argv);
1943    int httpd_main(int argc, char **argv)
1944    {
1945            unsigned opt;
1946            const char *home_httpd = home;
1947            char *url_for_decode;
1948            USE_FEATURE_HTTPD_ENCODE_URL_STR(const char *url_for_encode;)
1949            const char *s_port;
1950            USE_FEATURE_HTTPD_SETUID(const char *s_ugid = NULL;)
1951            USE_FEATURE_HTTPD_SETUID(struct bb_uidgid_t ugid;)
1952            USE_FEATURE_HTTPD_AUTH_MD5(const char *pass;)
1953
1954    #if ENABLE_LOCALE_SUPPORT
```

```
1955         /* Undo busybox.c: we want to speak English in http (dates etc) */
1956         setlocale(LC_TIME, "C");
1957 #endif
1958
1959         config = xzalloc(sizeof(*config));
1960 #if ENABLE_FEATURE_HTTPD_BASIC_AUTH
1961         config->realm = "Web Server Authentication";
1962 #endif
1963         config->port = 80;
1964         config->ContentLength = -1;
1965
1966         opt = getopt32(argc, argv, "c:d:h:"
1967                         USE_FEATURE_HTTPD_ENCODE_URL_STR("e:")
1968                         USE_FEATURE_HTTPD_BASIC_AUTH("r:")
1969                         USE_FEATURE_HTTPD_AUTH_MD5("m:")
1970                         USE_FEATURE_HTTPD_SETUID("u:")
1971                         "p:if",
1972                         &(config->configFile), &url_for_decode, &home_httpd
1973                         USE_FEATURE_HTTPD_ENCODE_URL_STR(, &url_for_encode)
1974                         USE_FEATURE_HTTPD_BASIC_AUTH(, &(config->realm))
1975                         USE_FEATURE_HTTPD_AUTH_MD5(, &pass)
1976                         USE_FEATURE_HTTPD_SETUID(, &s_ugid)
1977                         , &s_port
1978                 );
1979         if (opt & OPT_DECODE_URL) {
1980                 printf("%s", decodeString(url_for_decode, 1));
1981                 return 0;
1982         }
1983 #if ENABLE_FEATURE_HTTPD_ENCODE_URL_STR
1984         if (opt & OPT_ENCODE_URL) {
1985                 printf("%s", encodeString(url_for_encode));
1986                 return 0;
1987         }
1988 #endif
1989 #if ENABLE_FEATURE_HTTPD_AUTH_MD5
```

```
1990            if (opt & OPT_MD5) {
1991                    puts(pw_encrypt(pass, "$1$"));
1992                    return 0;
1993            }
1994    #endif
1995            if (opt & OPT_PORT)
1996                    config->port = xatou16(s_port);
1997
1998    #if ENABLE_FEATURE_HTTPD_SETUID
1999            if (opt & OPT_SETUID) {
2000                    if (!get_uidgid(&ugid, s_ugid, 1))
2001                            bb_error_msg_and_die("unrecognized user[:group] "
2002                                                 "name '%s'", s_ugid);
2003            }
2004    #endif
2005
2006            xchdir(home_httpd);
2007            if (!(opt & OPT_INETD)) {
2008                    signal(SIGCHLD, SIG_IGN);
2009                    config->server_socket = openServer();
2010    #if ENABLE_FEATURE_HTTPD_SETUID
2011                    /* drop privileges */
2012                    if (opt & OPT_SETUID) {
2013                            if (ugid.gid != (gid_t)-1) {
2014                                    if (setgroups(1, &ugid.gid) == -1)
2015                                            bb_perror_msg_and_die("setgroups");
2016                                    xsetgid(ugid.gid);
2017                            }
2018                            xsetuid(ugid.uid);
2019                    }
2020    #endif
2021            }
2022
2023    #if ENABLE_FEATURE_HTTPD_CGI
2024            {
```

```
2025                    char *p = getenv("PATH");
2026                    p = xstrdup(p); /* if gets NULL, returns NULL */
2027                    clearenv();
2028                    if (p)
2029                            setenv1("PATH", p);
2030                    if (!(opt & OPT_INETD))
2031                            setenv_long("SERVER_PORT", config->port);
2032            }
2033    #endif
2034
2035    #if ENABLE_FEATURE_HTTPD_RELOAD_CONFIG_SIGHUP
2036            sighup_handler(0);
2037    #else
2038            parse_conf(default_path_httpd_conf, FIRST_PARSE);
2039    #endif
2040
2041            if (opt & OPT_INETD)
2042                    return miniHttpd_inetd();
2043
2044            if (!(opt & OPT_FOREGROUND))
2045                    bb_daemonize(0);     /* don't change current directory */
2046            return miniHttpd(config->server_socket);
2047    }
```

IKI HTTP server implementation

HTTPD module was published as is , with permission to use freely, at http://www.iki.fi/iki/src/httpd.c.

httpd.c – Current Version as of May 25, 2009.

```
1  /*
2   * httpd -- Simple httpd-server
3   * Copyright (c) 1995 Tero Kivinen
4   * All Rights Reserved.
5   *
6   * Permission to use, copy, modify and distribute this software and its
7   * documentation is hereby granted, provided that both the copyright
8   * notice and this permission notice appear in all copies of the
9   * software, derivative works or modified versions, and any portions
10  * thereof, and that both notices appear in supporting documentation.
11  *
12  * TERO KIVINEN ALLOWS FREE USE OF THIS SOFTWARE IN ITS "AS IS"
13  * CONDITION.  TERO KIVINEN DISCLAIMS ANY LIABILITY OF ANY KIND FOR
14  * ANY DAMAGES WHATSOEVER RESULTING FROM THE USE OF THIS SOFTWARE.
15  *
16  */
17 /*
18  *        Program: simple httpd-server
19  *        $Source: /iki/src/simple-httpd/RCS/httpd.c,v $
20  *        $Author: kivinen $
21  *
22  *        (C) Tero Kivinen 1995 <Tero.Kivinen@hut.fi>
23  *
24  *        Creation          : 23:47 Mar 23 1995 kivinen
25  *        Last Modification : 14:40 Nov 24 2006 kivinen
26  *        Last check in     : $Date: 2006/11/24 12:45:36 $
```

```
27    *        Revision number   : $Revision: 1.25 $
28    *        State             : $State: Exp $
29    *        Version           : 1.915
30    *        Edit time         : 549 min
31    *
32    *        Description       : Simple http-server
33    *
34    *
35    *        $Log: httpd.c,v $
36    *        Revision 1.25  2006/11/24 12:45:36  kivinen
37    *              Fixed opendir + telldir + closedir + opendir + seekdir to so
38    *              it willnot reopen the directory, and do not do extra seeks
39    *              etc. The seekdir cannot be used after directory has been
40    *              closed. Changed most of strings to unsigned char's to remove
41    *              warnings.
42    *
43    *        Revision 1.24  2006/11/02 18:57:04  kivinen
44    *              Added css to known mime types.
45    *
46    *        Revision 1.23  2006/10/05 13:32:27  kivinen
47    *              Added charset parameter to text/plain and text/html.
48    *
49    *        Revision 1.22  2002/11/12 18:16:57  kivinen
50    *              Added code that will exit if connections do not finish in 30
51    *              seconds after quit.
52    *
53    *        Revision 1.21  2002/10/06 13:32:04  kivinen
54    *              Added code that will raise the rlimit_nofile to max.
55    *
56    *        Revision 1.20  2002/10/06 13:06:47  kivinen
57    *              Added some memory allocation checks. Changed version number to
58    *              1.2. Optimized the code to use only one write instead of two
59    *              when sending reply.
60    *
61    *        Revision 1.19  1998/12/03 20:13:55  kivinen
```

```
62    *             Added ignoring of sigpipe.
63    *
64    *       Revision 1.18  1997/12/06 12:06:22  kivinen
65    *             Fixed 2 bugs.
66    *
67    *       Revision 1.17  1997/10/09 03:14:19  kivinen
68    *             Fixed bug reported by Jon Wickstrom about weekday being off by
69    *             one.
70    *
71    *       Revision 1.16  1997/05/13 16:08:57  kivinen
72    *             Added changes from liw for solaris.
73    *
74    *       Revision 1.15  1996/11/19 21:05:13  kivinen
75    *             Added post command to be processed just like get.
76    *
77    *       Revision 1.14  1996/09/13 20:18:08  kivinen
78    *             Added status command.
79    *
80    *       Revision 1.13  1996/08/21 14:09:46  kivinen
81    *             getservbyname returns port number in network byte order,
82    *             removed htons from using servent->s_port.
83    *
84    *       Revision 1.12  1996/07/15 16:14:51  kivinen
85    *             Added printing of errno in case of errors.
86    *             If read fails set write_state to STATE_ERROR too, so it don't
87    *             try to write to socket.
88    *
89    *       Revision 1.11  1996/07/15 15:59:45  kivinen
90    *             Changed to use LOG_LOCAL0 instead of LOG_DAEMON.
91    *
92    *       Revision 1.10  1996/07/11 03:06:08  kivinen
93    *             Fixed bug in last_info_hour.
94    *
95    *       Revision 1.9  1996/07/11 02:08:52  kivinen
96    *             Modified all times to use MINUTES and HOUR defines.
```

```
 97  *              Added statistic output. Changed LOG_NOTICE to LOG_INFO and
 98  *              added all statistics to be on level LOG_NOTICE.
 99  *
100  *        Revision 1.8  1996/03/01 14:36:08  kivinen
101  *              Added png.
102  *
103  *        Revision 1.7  1995/12/14 19:28:32  kivinen
104  *              Fixed %-n.ns to %.ns.
105  *
106  *        Revision 1.6  1995/11/29 06:48:32  kivinen
107  *              Increased command length to 2048, so now the headers can also
108  *              fit there.
109  *              Added READ_TIME_OUT that will tell when the request reads time
110  *              out when reading headers.
111  *              Added read_state, write_state and headers to connection_t.
112  *              Changed all %s to %-n.ns in syslogs.
113  *              Changed main loop so it will close socket only after all of
114  *              the request have been read from the socket.
115  *
116  *        Revision 1.5  1995/11/16 18:00:43  kivinen
117  *              Added timeout code.
118  *
119  *        Revision 1.4  1995/07/26 12:14:49  kivinen
120  *              Fixed bug in temporary redirection page generation.
121  *
122  *        Revision 1.3  1995/07/20 04:14:51  kivinen
123  *              Raised BUF_LENGTH from 2048 to 3000, because it now must be
124  *              large enough for 2 URL's.
125  *              Moved setsockopt to correct place before bind.
126  *              Changed rfc850date to rfc1123date.
127  *              Added URI-field, renamed Date: to X-Date (Date must be current
128  *              date, and our date-field was the date when the page was
129  *              created in the memory).
130  *              Added url-decoding.
131  *              Changed protocol version check to check only HTTP/1.
```

```
 *
 *        Revision 1.2  1995/07/16 16:30:32  kivinen
 *              Fixed quit code.
 *
 *        Revision 1.1  1995/07/16 11:45:07  kivinen
 *              Created.
 *
 *
 *
 *
 *
 *
 */
/*
 * If you have any useful modifications or extensions please send them to
 * Tero.Kivinen@hut.fi
 */

/* Make sure this is big enough. */
#define FD_SETSIZE 1024

#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <fcntl.h>
#include <dirent.h>
#include <syslog.h>
#include <stdio.h>
#include <ctype.h>
#include <errno.h>
#include <memory.h>
```

```
167  #include <string.h>
168  #include <unistd.h>
169  #include <limits.h>
170  #include <sys/time.h>
171  #include <locale.h>
172  #include <stdarg.h>
173  #include <pwd.h>
174  #ifdef __sgi__
175  #include <bstring.h>
176  #endif
177  #include <signal.h>
178  #include <sys/resource.h>
179
180  #ifdef __sun__
181  #include <sys/file.h>
182  #define getdtablesize() 1024
183  #endif
184
185  #undef DEBUG
186  #undef NIKSULAROOT
187  #undef SHADOWSROOT
188  #define HTTPD_GID 80
189  #define HTTPD_UID 80
190
191  #define VERSION "SimpleHTTP/1.2"
192
193  /* Local http-root */
194  #ifdef NIKSULAROOT
195  #define LOCAL_ROOT_URL "http://nukkekoti.cs.hut.fi"
196  #else
197  #ifdef SHADOWSROOT
198  #define LOCAL_ROOT_URL "http://shadows.cs.hut.fi"
199  #else
200  #define LOCAL_ROOT_URL "http://www.iki.fi"
201  #endif
```

```
202 #endif
203
204 #define MINUTE         (60)
205 #define HOUR           (MINUTE * 60)
206
207 /* How long wait for the http connections to finish before exiting. */
208 #define QUIT_TIME      30
209
210 /* How long to keep the temporal pages in memory */
211 #define KEEP_TIME      (10 * MINUTE)
212
213 /* How often to check if we have temporal pages in memory we can throw away */
214 #define CHECK_TIME     (MINUTE)
215
216 /* This tells how often we try to stat files, to see if they have changed */
217 #define STAT_TIME      (5 * MINUTE)
218
219 /* This tells long we wait for command */
220 #define KICK_TIME      (2 * MINUTE)
221
222 /* This tells long we wait for headers */
223 #define READ_TIME_OUT  (2 * MINUTE)
224
225 /* Max line length. Used to read data from redirections file. */
226 #define LINE_LENGTH    1024
227
228 /* Max command length. Maximum of this much is read from socket to find url */
229 #define COMMAND_LENGTH 2048
230
231 /* Maximum length of url. Must be larger or same as COMMAND_LENGTH,
232  * LINE_LENGTH, and PATH_MAX. */
233 #define URL_LENGTH     2048
234
235 /* Misc buffer length. Must be at least URL_LENGTH * 2 + TIME_LENGTH +
236   *  300 (misc headers) */
```

```
237  #define BUF_LENGTH     5000
238
239  /* Length of time buffers "Weekday, dd-Mon-95 hh:mm:ss GMT" = 34 chars */
240  #define TIME_LENGTH    64
241
242
243  /* Listen backlog value */
244  #define LISTEN_BACKLOG 10
245
246  /* Debug output */
247  #ifdef DEBUG
248  #define DPRINT(x) dprint x
249  #else
250  #define DPRINT(x)
251  #endif
252
253  char *program;
254  int f_inetd = 0;
255  int f_daemon = 0;
256  gid_t f_gid = -1;
257  pid_t f_uid = -1;
258  char *f_port = "http";
259  int port = -1;
260
261  typedef enum {
262    STATE_NONE,
263    STATE_DOING,
264    STATE_COMMAND_READ,          /* Only for read */
265    STATE_ERROR,
266    STATE_TIMED_OUT,             /* Only for read */
267    STATE_DONE,
268  } state_t;
269
270  typedef enum {
271    COMMAND_HEAD,
```

```
272     COMMAND_BODY,
273     COMMAND_BOTH
274   } command_t;
275
276   typedef struct page_s
277   {
278     unsigned char *url;          /* Url of the page */
279     unsigned char *reply_head;   /* Headers of the reply */
280     long reply_head_len;         /* Length  of headers */
281     unsigned char *reply_body;   /* Body of the reply, this follows directly
282                                     the headers, i.e it is in same buffer. */
283     long reply_body_len;         /* Length of the body */
284     unsigned char *filename;     /* Filename of page or NULL if none */
285     time_t last_modification;    /* Last modification time of the page */
286     time_t last_stat;            /* Last stat for the file */
287     time_t ref_count;            /* Reference count. Initially set to 1 for all
288                                   * permanent pages (files, permanent
289                                   * redirections). When it gets to 0 the
290                                   * page is freed (permanent pages are never
291                                   * freed) */
292     time_t last_ref;             /* Last time the page was referenced */
293     int permanent;               /* Permanent page */
294   } *page_t;
295
296   typedef struct redirection_s
297   {
298     unsigned char *from;         /* The path component of url to redirect */
299     unsigned char *to;           /* The initial url where to redirect */
300   } *redirection_t;
301
302   typedef struct connection_s
303   {
304     int socket;
305     struct in_addr addr;
306     time_t last_time;
```

```
307     state_t read_state, write_state;
308
309     long total_bytes;
310
311     unsigned char *out_data;
312     long out_data_len;
313     unsigned char *out_data_ptr;
314
315     unsigned char *in_data;
316     long in_data_len;
317     unsigned char *in_data_ptr;
318
319     unsigned char *url;
320     unsigned char *status;
321     unsigned char *command;
322     unsigned char *headers;
323
324     page_t page;
325   } *connection_t;
326
327   page_t *pages, *temp_pages;
328   redirection_t *redirections;
329   connection_t *connections;
330   int max_pages, num_pages, max_redirections, num_redirections;
331   int max_temp_pages;
332   int master_server = -1;
333   time_t now, last_info_hour, quit_time = 0;
334   unsigned char local_root_url[URL_LENGTH];
335   int total_bytes_in_cache = 0;
336   fd_set fdrdset, fdwrset;
337   int max_connections, number_of_sockets;
338
339   int total_connections, total_max_connections,
340     total_pages, total_errors, total_errors_no_page,
341     total_temp_cache_hit, total_temp_make, total_perm_page,
```

```
342     total_user_home, total_stat, total_read,
343     total_bytes_sent, total_bytes;
344
345   /*
346    * Debug printf
347    */
348   void dprint(const unsigned char *fmt, ...)
349   {
350     va_list args;
351     char buffer[BUF_LENGTH];
352
353     va_start(args, fmt);
354     vsprintf(buffer, (char *) fmt, args);
355     va_end(args);
356     fprintf(stderr, "%s\n", buffer);
357   }
358
359   /*
360    * Open server socket
361    */
362   int open_service(const char *serv)
363   {
364     struct sockaddr_in sin;
365     int err, socks;
366     struct servent *servent;
367     int one;
368
369     DPRINT(("opening service %s", serv));
370
371     memset(&sin, 0, sizeof(sin));
372     sin.sin_family = AF_INET;
373
374     servent = getservbyname(serv, "tcp");
375     if (servent == NULL)
376       {
```

```
377         errno = 0;
378         port = atoi(serv);
379         if (port == 0)
380           {
381             syslog(LOG_CRIT, "Error unknown service %.200s, exiting", serv);
382             exit(1);
383           }
384         sin.sin_port = htons((unsigned short) port);
385       }
386     else
387       {
388         sin.sin_port = servent->s_port;
389         port = servent->s_port;
390       }
391
392     DPRINT(("found port %d", ntohs(sin.sin_port)));
393
394     socks = socket(AF_INET, SOCK_STREAM,0);
395
396     if (socks < 0)
397       {
398         syslog(LOG_CRIT, "Error in socket at open_service opening service %.200s, exiting",
399                 serv);
400         exit(1);
401       }
402
403     one = 1;
404
405     if (setsockopt(socks, SOL_SOCKET, SO_REUSEADDR, (char *) &one,
406                    sizeof(int)) == -1)
407       {
408         syslog(LOG_ERR, "Error in setsockopt REUSEADDR");
409       }
410
411     err = bind(socks, (struct sockaddr *) &sin, sizeof(sin));
```

```
412     if (err)
413       {
414         close(socks);
415         syslog(LOG_CRIT, "Error in bind at open_service opening service %.200s, exiting",
416                serv);
417         exit(1);
418       }
419
420     err = listen(socks, LISTEN_BACKLOG);
421     if (err)
422       {
423         close(socks);
424         syslog(LOG_CRIT, "Error in listen at open_service opening service %.200s, exiting",
425                serv);
426         exit(1);
427       }
428     DPRINT(("service opened"));
429     return socks;
430   }
431
432   char *wkday[5] = { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };
433   char *month[12] = { "Jan", "Feb", "Mar", "Apr", "May", "Jun",
434             "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" };
435
436   /*
437    * Convert unix date integer to rfc1123 date
438    */
439   unsigned char *rfc1123date(unsigned char *strbuf, time_t t)
440   {
441     struct tm *tm;
442     tm = localtime(&t);
443     sprintf((char *) strbuf, "%s, %02d %s %04d %02d:%02d:%02d GMT",
444             wkday[tm->tm_wday], tm->tm_mday, month[tm->tm_mon],
445             tm->tm_year+1900, tm->tm_hour, tm->tm_min, tm->tm_sec);
446     DPRINT(("rfc1123date: %s", strbuf));
```

```
447     return strbuf;
448   }
449
450   /*
451    * Skip all whitespace characters. Returns the pointer to first non-whitespace
452    * character.
453    */
454   unsigned char *skip_white(unsigned char *p)
455   {
456     if (p == NULL)
457       return NULL;
458
459     while(isspace(*p))
460       p++;
461     return p;
462   }
463
464   /*
465    * Skip all non whitespace characters. Returns the pointer to first whitespace
466    * character.
467    */
468   unsigned char *skip_non_white(unsigned char *p)
469   {
470     if (p == NULL)
471       return NULL;
472
473     while(*p && !isspace(*p))
474       p++;
475     return p;
476   }
477
478   /*
479    * Compare page entries using url field.
480    */
481   int compr_url(const void *a, const void *b)
```

```
{
  const page_t ap = *(page_t *) a;
  const page_t bp = *(page_t *) b;
  return strcasecmp((char *) ap->url, (char *) bp->url);
}

/*
 * Compare page entries using url field.
 */
int compr_redirection(const void *a, const void *b)
{
  const redirection_t ar = *(redirection_t *) a;
  const redirection_t br = *(redirection_t *) b;
  return strcasecmp((char *) ar->from, (char *) br->from);
}

/*
 * Set head and body data. If the buffer is NULL then it is not copied,
 * if the size is 0 then strlen is used.
 */
void set_head_and_body(page_t page, unsigned char *header, long header_len,
                       unsigned char *body, long body_len)
{
  long length;
  unsigned char *data;

  if (header_len == 0 && header != NULL)
    header_len = strlen((char *) header);
  if (body_len == 0 && body != NULL)
    body_len = strlen((char *) body);

  length = header_len + body_len;
  data = malloc(length);

  if (data == NULL)
```

```
517       {
518         syslog(LOG_CRIT, "Out of memory, exiting");
519         exit(1);
520       }
521
522     page->reply_head = data;
523     page->reply_head_len = header_len;
524     page->reply_body = data + header_len;
525     page->reply_body_len = body_len;
526
527     if (header != NULL)
528       memcpy(page->reply_head, header, header_len);
529     if (body != NULL)
530       memcpy(page->reply_body, body, body_len);
531   }
532
533   /*
534    * Add redirection page
535    */
536   void add_redir_page(unsigned char *url, unsigned char *to_url)
537   {
538     long length;
539     unsigned char buffer[BUF_LENGTH], hbuffer[BUF_LENGTH], timebuf[TIME_LENGTH];
540
541     DPRINT(("Adding redir page: %s -> %s", url, to_url));
542     pages[num_pages] = calloc(1, sizeof(struct page_s));
543     if (pages[num_pages] == NULL)
544       {
545         syslog(LOG_CRIT, "Out of memory, exiting");
546         exit(1);
547       }
548     pages[num_pages]->url = (unsigned char *) strdup((char *) url);
549     if (pages[num_pages]->url == NULL)
550       {
551         syslog(LOG_CRIT, "Out of memory, exiting");
```

```
552         exit(1);
553       }
554
555     sprintf((char *) buffer, "<HEAD><TITLE>Redirection</TITLE></HEAD>\n<BODY><H1>Query redirected to another
556   address</H1>\nThis is only a redirection service, the document can be found <A HREF=\"%s\">here</A>.<P></BODY>",
557   to_url);
558     length = strlen((char *) buffer);
559
560     sprintf((char *) hbuffer, "HTTP/1.0 302 Found\r\nX-Date: %s\r\nServer: %s\r\nMIME-version: 1.0\r\nLocation:
561   %s\r\nURI: <%s>\r\nContent-type: text/html\r\nContent-Length: %ld\r\n\r\n", rfc1123date(timebuf, now), VERSION,
562   to_url, to_url, length);
563
564     set_head_and_body(pages[num_pages], hbuffer, 0L, buffer, 0L);
565     pages[num_pages]->filename = NULL;
566     pages[num_pages]->last_modification = 0;
567     pages[num_pages]->last_stat = 0;
568     pages[num_pages]->ref_count = 0;
569     pages[num_pages]->permanent = 1;
570
571     num_pages++;
572     if (num_pages >= max_pages)
573       {
574         pages = realloc(pages, sizeof(page_t) * 2 * max_pages);
575         if (pages == NULL)
576           {
577             syslog(LOG_CRIT, "Out of memory, exiting");
578             exit(1);
579           }
580         memset(pages + max_pages, 0, sizeof(page_t) * max_pages);
581         max_pages *= 2;
582       }
583   }
584
585   void add_forwarding_page(unsigned char *url, unsigned char *to_url)
586   {
```

```
587
588     DPRINT(("Adding forwarding page: %s -> %s", url, to_url));
589
590     redirections[num_redirections] = calloc(1, sizeof(struct redirection_s));
591     if (redirections[num_redirections] == NULL)
592       {
593         syslog(LOG_CRIT, "Out of memory, exiting");
594         exit(1);
595       }
596
597     redirections[num_redirections]->from =
598       (unsigned char *) strdup((char *) url);
599     redirections[num_redirections]->to =
600       (unsigned char *) strdup((char *) to_url);
601     if (redirections[num_redirections]->from == NULL ||
602         redirections[num_redirections]->to == NULL)
603       {
604         syslog(LOG_CRIT, "Out of memory, exiting");
605         exit(1);
606       }
607     num_redirections++;
608     if (num_redirections >= max_redirections)
609       {
610         redirections = realloc(redirections,
611                                sizeof(redirection_t) * 2 * max_redirections);
612         if (redirections == NULL)
613           {
614             syslog(LOG_CRIT, "Out of memory, exiting");
615             exit(1);
616           }
617         memset(redirections + max_redirections, 0,
618                sizeof(redirection_t) * max_redirections);
619         max_redirections *= 2;
620       }
621 }
```

```
622
623   /*
624    * Read redirections.
625    */
626   void read_redirections(unsigned char *file_name)
627   {
628     FILE *file;
629     unsigned char line[LINE_LENGTH];
630     long length;
631     unsigned char *from, *to;
632
633     file = fopen((char *) file_name, "r");
634     if (file == NULL)
635       {
636         syslog(LOG_CRIT, "Cannot open redirections file, exiting");
637         exit(1);
638       }
639     while(fgets((char *) line, LINE_LENGTH, file) != NULL)
640       {
641         length = strlen((char *) line);
642         if (length == 0)
643           continue;
644         while (length > 0 && isspace(line[length - 1]))
645           {
646             length--;
647           }
648         line[length] = '\0';
649
650         from = skip_white(line);
651         to = skip_non_white(from);
652         *to++ = '\0';
653         to = skip_white(to);
654         if (strlen((char *) from) == 0 || strlen((char *) to) == 0)
655           continue;
656
```

```
657         add_redir_page(from, to);
658         if (from[strlen((char *) from) - 1] == '/')
659           {
660             add_forwarding_page(from, to);
661           }
662       }
663     fclose(file);
664   }
665
666   /*
667    * Convert filename to type
668    */
669   char *match_type(char *name)
670   {
671     char *dot;
672     dot = strrchr(name, '.');
673     if (dot == NULL)
674       return "text/plain; charset=ISO-8859-1";
675     else if (strcasecmp(dot, ".html") == 0)
676       return "text/html; charset=ISO-8859-1";
677     else if (strcasecmp(dot, ".css") == 0)
678       return "text/css";
679     else if (strcasecmp(dot, ".htm") == 0)
680       return "text/html; charset=ISO-8859-1";
681     else if (strcasecmp(dot, ".txt") == 0)
682       return "text/plain; charset=ISO-8859-1";
683     else if (strcasecmp(dot, ".aiff") == 0)
684       return "audio/x-aiff";
685     else if (strcasecmp(dot, ".au") == 0)
686       return "audio/x-au";
687     else if (strcasecmp(dot, ".gif") == 0)
688       return "image/gif";
689     else if (strcasecmp(dot, ".png") == 0)
690       return "image/png";
691     else if (strcasecmp(dot, ".bmp") == 0)
```

```
692       return "image/bmp";
693     else if (strcasecmp(dot, ".jpeg") == 0)
694       return "image/jpeg";
695     else if (strcasecmp(dot, ".jpg") == 0)
696       return "image/jpeg";
697     else if (strcasecmp(dot, ".tiff") == 0)
698       return "image/tiff";
699     else if (strcasecmp(dot, ".tif") == 0)
700       return "image/tiff";
701     else if (strcasecmp(dot, ".pnm") == 0)
702       return "image/x-portable-anymap";
703     else if (strcasecmp(dot, ".pbm") == 0)
704       return "image/x-portable-bitmap";
705     else if (strcasecmp(dot, ".pgm") == 0)
706       return "image/x-portable-graymap";
707     else if (strcasecmp(dot, ".ppm") == 0)
708       return "image/x-portable-pixmap";
709     else if (strcasecmp(dot, ".rgb") == 0)
710       return "image/rgb";
711     else if (strcasecmp(dot, ".xbm") == 0)
712       return "image/x-bitmap";
713     else if (strcasecmp(dot, ".xpm") == 0)
714       return "image/x-pixmap";
715     else if (strcasecmp(dot, ".mpeg") == 0)
716       return "video/mpeg";
717     else if (strcasecmp(dot, ".mpg") == 0)
718       return "video/mpeg";
719     else if (strcasecmp(dot, ".ps") == 0)
720       return "application/ps";
721     else if (strcasecmp(dot, ".eps") == 0)
722       return "application/ps";
723     else if (strcasecmp(dot, ".dvi") == 0)
724       return "application/x-dvi";
725     return "text/plain";
726   }
```

```
727
728  /*
729   * Read one www-page, fill in the data to page-struct given. Return 1 if
730   * success, 0 otherwise.
731   */
732
733  int read_page(page_t *page, unsigned char *file, struct stat *st,
734                unsigned char *url)
735  {
736    int fd;
737    unsigned char buffer[BUF_LENGTH], timebuf1[TIME_LENGTH],
738      timebuf2[TIME_LENGTH];
739
740    DPRINT(("Reading file = %s, url = %s", file, url));
741    fd = open((char *) file, O_RDONLY, 0666);
742    if (fd < 0)
743      {
744        syslog(LOG_ERR, "Error cannot open file %.200s", file);
745        return 0;
746      }
747
748    if (*page == NULL)
749      {
750        *page = calloc(1, sizeof(struct page_s));
751        if (*page == NULL)
752          {
753            syslog(LOG_CRIT, "Out of memory, exiting");
754            exit(1);
755          }
756      }
757    (*page)->url = (unsigned char *) strdup((char *) url);
758    if ((*page)->url == NULL)
759      {
760        syslog(LOG_CRIT, "Out of memory, exiting");
761        exit(1);
```

```
762       }
763
764     sprintf((char *) buffer, "HTTP/1.0 200 OK\r\nX-Date: %s\r\nServer: %s\r\nMIME-version: 1.0\r\nLast-Modified:
765   %s\r\nContent-type: %s\r\nContent-Length: %ld\r\n\r\n", rfc1123date(timebuf1, now), VERSION,
766   rfc1123date(timebuf2, st->st_mtime), match_type((char *) file), (unsigned long) st->st_size);
767
768     set_head_and_body(*page, buffer, 0L, NULL, (long) st->st_size);
769     (*page)->filename = (unsigned char *) strdup((char *) file);
770     if ((*page)->filename == NULL)
771       {
772         syslog(LOG_CRIT, "Out of memory, exiting");
773         exit(1);
774       }
775     (*page)->last_modification = st->st_mtime;
776     (*page)->last_stat = now;
777     (*page)->ref_count = 0;
778
779     if (read(fd, (*page)->reply_body, st->st_size) != st->st_size)
780       {
781         close(fd);
782         syslog(LOG_ERR, "Error reading file %.200s", file);
783         free((*page)->reply_head);
784         free((*page)->url);
785         free((*page));
786         (*page) = NULL;
787         return 0;
788       }
789     close(fd);
790     return 1;
791   }
792
793   /*
794    * Read htdocs.
795    */
796
```

```
797  void read_htdocs(unsigned char *htdocs)
798  {
799    DIR *dir;
800    struct dirent *de;
801    struct stat st;
802    unsigned char url[URL_LENGTH], to_url[URL_LENGTH], file[PATH_MAX];
803
804    DPRINT(("Reading htdocs = %s", htdocs));
805
806    if (htdocs[0] == '.' && htdocs[1] == '\0')
807      {
808        sprintf((char *) to_url, "%s/index.html", local_root_url);
809        add_redir_page((unsigned char *) "/", to_url);
810      }
811    else
812      {
813        if (htdocs[0] == '.' && htdocs[1] == '/')
814          {
815            sprintf((char *) url, "/%s", htdocs + 2);
816          }
817        else
818          {
819            sprintf((char *) url, "/%s", htdocs);
820          }
821        sprintf((char *) to_url, "%s%s/index.html", local_root_url, url);
822        add_redir_page(url, to_url);
823        strcat((char *) url, "/");
824        add_redir_page(url, to_url);
825      }
826
827    sprintf((char *) file, "%s/.", htdocs);
828
829    dir = opendir((char *) file);
830
831    if (dir == NULL)
```

```
832       {
833         syslog(LOG_ERR, "Error opendir(%.200s) failed", htdocs);
834         return;
835       }
836     while((de = readdir(dir)) != NULL)
837       {
838         sprintf((char *) file, "%s/%s", htdocs, de->d_name);
839         if (htdocs[0] == '.' && htdocs[1] == '\0')
840           {
841             sprintf((char *) url, "/%s", de->d_name);
842           }
843         else if (htdocs[0] == '.' && htdocs[1] == '/')
844           {
845             sprintf((char *) url, "/%s/%s", htdocs + 2, de->d_name);
846           }
847         else
848           {
849             sprintf((char *) url, "/%s/%s", htdocs, de->d_name);
850           }
851
852         if (stat((char *) file, &st) < 0)
853           {
854             syslog(LOG_ERR, "Error stat to %.200s failed", de->d_name);
855             continue;
856           }
857         if (st.st_mode & S_IFDIR)
858           {
859             if (strcmp((char *) de->d_name, ".") != 0 &&
860                 strcmp((char *) de->d_name, "..") != 0)
861               {
862                 long pos;
863
864                 pos = telldir(dir);
865                 //              closedir(dir);
866
```

```
867               read_htdocs(file);
868               // dir = opendir((char *) htdocs);
869               // if (dir == NULL)
870               // {
871               // syslog(LOG_ERR, "Error re-opendir(%.200s) failed", htdocs);
872               //  return;
873               // }
874               // seekdir(dir, pos);
875             }
876         }
877       else
878         {
879           if (read_page(&(pages[num_pages]), file, &st, url)) {
880             pages[num_pages]->permanent = 1;
881             num_pages++;
882           }
883           if (num_pages >= max_pages)
884             {
885               pages = realloc(pages, sizeof(page_t) * 2 * max_pages);
886               if (pages == NULL)
887                 {
888                   syslog(LOG_CRIT, "Out of memory, exiting");
889                   exit(1);
890                 }
891               memset(pages + max_pages, 0, sizeof(page_t) * max_pages);
892               max_pages *= 2;
893             }
894         }
895     }
896   closedir(dir);
897 }
898
899 /*
900  * Read www-pages. First read redirections file, which contain from-http,
901  * to-http pairs one at a line separated by spaces.
```

```
902    *
903    * Then read all files from htdocs directory.
904    */
905   void read_pages(unsigned char *file, unsigned char *htdocs)
906   {
907     max_pages = 64;
908     num_pages = 0;
909     pages = calloc(max_pages, sizeof(page_t));
910
911     max_redirections = 64;
912     num_redirections = 0;
913     redirections = calloc(max_redirections, sizeof(redirection_t));
914
915     max_temp_pages = 64;
916     temp_pages = calloc(max_temp_pages, sizeof(page_t));
917
918     if (pages == NULL || redirections == NULL || temp_pages == NULL)
919       {
920         syslog(LOG_CRIT, "Out of memory, exiting");
921         exit(1);
922       }
923     read_redirections(file);
924     chdir((char *) htdocs);
925     read_htdocs((unsigned char *) ".");
926     qsort(pages, num_pages, sizeof(page_t), compr_url);
927     qsort(redirections, num_redirections, sizeof(redirection_t),
928           compr_redirection);
929   }
930
931   /*
932    * Do writing of data.
933    */
934   void do_write(int i)
935   {
936     int ret;
```

```
937
938     DPRINT(("Writing data to connection %d", i));
939     if (connections[i]->write_state == STATE_ERROR ||
940         connections[i]->write_state == STATE_DONE)
941       return;
942     connections[i]->write_state = STATE_DOING;
943
944     while (1)
945       {
946         if (connections[i]->out_data_len > 0)
947           {
948             ret = write(connections[i]->socket,
949                         connections[i]->out_data_ptr,
950                         connections[i]->out_data_len);
951             if (ret < 0)
952               {
953                 if (errno == EWOULDBLOCK)
954                   return;
955                 syslog(LOG_ERR, "Write failed for %.200s : %d",
956                        inet_ntoa(connections[i]->addr), errno);
957                 connections[i]->write_state = STATE_ERROR;
958                 return;
959               }
960             connections[i]->out_data_ptr += ret;
961             connections[i]->out_data_len -= ret;
962           }
963         else
964           {
965             connections[i]->write_state = STATE_DONE;
966             FD_CLR(connections[i]->socket, &fdwrset);
967             return;
968           }
969       }
970   }
971
```

```
/*
 * Close connection
 */
void close_connection(int i)
{
  DPRINT(("Closing connection %d", i));
  syslog(LOG_INFO, "%.40s %.200s (%ld/%ld bytes) from %.30s %.100s.",
         connections[i]->command,
         connections[i]->url,
         connections[i]->total_bytes -
         connections[i]->out_data_len,
         connections[i]->total_bytes,
         inet_ntoa(connections[i]->addr),
         connections[i]->status);

  total_pages++;
  total_bytes += connections[i]->total_bytes;
  total_bytes_sent += connections[i]->total_bytes -
    connections[i]->out_data_len;

  FD_CLR(connections[i]->socket, &fdwrset);
  FD_CLR(connections[i]->socket, &fdrdset);
  number_of_sockets--;
  close(connections[i]->socket);
  if (connections[i]->page != NULL)
    {
      connections[i]->page->ref_count--;
      if (connections[i]->page->ref_count == 0)
        {
          DPRINT(("Marking page %s last reference time to %d",
                  connections[i]->page->url, now));
          connections[i]->page->last_ref = now;
        }
    }
  connections[i]->socket = -1;
```

```
1007      connections[i]->total_bytes = 0;
1008      connections[i]->last_time = now;
1009      connections[i]->out_data = NULL;
1010      connections[i]->out_data_len = 0L;
1011      connections[i]->out_data_ptr = NULL;
1012      connections[i]->in_data_len = 0L;
1013    }
1014
1015    unsigned char bad_request[] = "<HEAD><TITLE>400 Bad Request</TITLE></HEAD>\n<BODY><H1>400 Bad Request</H1>\nYour
1016    client sent a query that this server could not understand.<P>\n</BODY>\n";
1017
1018    unsigned char not_found[] = "HTTP/1.0 404 Not found\r\nServer: " VERSION "\r\nMIME-version: 1.0\r\nContent-type:
1019    text/html\r\n\r\n<HEAD><TITLE>404 Not Found</TITLE></HEAD>\n<BODY><H1>404 Not Found</H1>\nThe requested URL was
1020    not found on this server.<P>\n</BODY>\n";
1021
1022    unsigned char done[] = "HTTP/1.0 200 OK\r\n\r\n<HEAD><TITLE>Done</TITLE></HEAD>\n<BODY><H1>Done</H1>\n</BODY>\n";
1023
1024    /* Do not change header unless you change the status_head_len also. */
1025    unsigned char status[1024] = "HTTP/1.0 200 OK\r\n\r\n";
1026    long status_head_len = 19L;
1027
1028    /*
1029     * Return error message for connection i. Start writing of error
1030     */
1031    void return_error(int i)
1032    {
1033      DPRINT(("Returning bad request error to connection %d", i));
1034      connections[i]->out_data = bad_request;
1035      connections[i]->out_data_len = sizeof(bad_request) - 1;
1036      connections[i]->out_data_ptr = bad_request;
1037      connections[i]->total_bytes = sizeof(bad_request) - 1;
1038      connections[i]->status = (unsigned char *) "bad request error";
1039      connections[i]->page = NULL;
1040      total_errors++;
1041      do_write(i);
```

```
}

/*
 * Set out data from the from the string and total length and optional
 * header length. If the header length is not given then the first \r\n\r\n
 * is searched and that is used as end of header.
 */
void set_out_data(connection_t connection, unsigned char *str, size_t tlen,
                  size_t hlen, command_t cmd)
{
  if (cmd == COMMAND_BOTH)
    {
      connection->out_data = str;
      connection->out_data_len = tlen;
      connection->out_data_ptr = str;
    }
  else
    {
      if (hlen == 0)
        {
          unsigned char *p;

          p = str;
          while (memcmp(p, "\r\n\r\n", 4) != 0)
            {
              p = memchr(p + 1, '\r', tlen - (p - str - 1));
              if (p == NULL)
                {
                  syslog(LOG_CRIT, "Internal error, no end of header found");
                  exit(1);
                }
            }
          hlen = (p - str) + 4;
        }
      if (cmd == COMMAND_BODY)
```

```
1077          {
1078            connection->out_data = str + hlen;
1079            connection->out_data_len = tlen - hlen;
1080            connection->out_data_ptr = str + hlen;
1081          }
1082        else
1083          {
1084            connection->out_data = str;
1085            connection->out_data_len = hlen;
1086            connection->out_data_ptr = str;
1087          }
1088      }
1089  }
1090
1091  /*
1092   * Find free temporary page
1093   */
1094  page_t *find_free_temp_page()
1095  {
1096    int i;
1097
1098    for(i = 0; i < max_temp_pages; i++)
1099      {
1100        if (temp_pages[i] == NULL ||
1101            temp_pages[i]->url == NULL)
1102          break;
1103      }
1104    if (i >= max_temp_pages)
1105      {
1106        temp_pages = realloc(temp_pages, sizeof(page_t) * 2 * max_temp_pages);
1107        if (temp_pages == NULL)
1108          {
1109            syslog(LOG_CRIT, "Out of memory, exiting");
1110            exit(1);
1111          }
```

```
1112        memset(temp_pages + max_temp_pages, 0, sizeof(page_t) * max_temp_pages);
1113        max_temp_pages *= 2;
1114      }
1115    return &(temp_pages[i]);
1116  }
1117
1118  /*
1119   * Make redirection page
1120   */
1121  page_t *make_redirection_page(redirection_t redirection,
1122                                unsigned char *rest_of_url)
1123  {
1124    long length;
1125    unsigned char buffer[BUF_LENGTH], hbuffer[BUF_LENGTH], timebuf[TIME_LENGTH],
1126      url[URL_LENGTH];
1127    page_t *page;
1128
1129    page = find_free_temp_page();
1130
1131    if (*page == NULL)
1132      {
1133        *page = calloc(1, sizeof(struct page_s));
1134        if (*page == NULL)
1135          {
1136            syslog(LOG_CRIT, "Out of memory, exiting");
1137            exit(1);
1138          }
1139      }
1140    sprintf((char *) url, "%s%s", redirection->from, rest_of_url);
1141    (*page)->url = (unsigned char *) strdup((char *) url);
1142    if ((*page)->url == NULL)
1143      {
1144        syslog(LOG_CRIT, "Out of memory, exiting");
1145        exit(1);
1146      }
```

```
1147
1148      sprintf((char *) url, "%s%s", redirection->to, rest_of_url);
1149
1150      DPRINT(("Making redirection page from %s -> %s", (*page)->url, url));
1151      sprintf((char *) buffer, "<HEAD><TITLE>Redirection</TITLE></HEAD>\n<BODY><H1>Query redirected to another
1152    address</H1>\nThis is only a redirection service, the document can be found <A HREF=\"%s\">here</A>.<P></BODY>",
1153    url);
1154      length = strlen((char *) buffer);
1155      total_bytes_in_cache += length;
1156
1157      sprintf((char *) hbuffer, "HTTP/1.0 302 Found\r\nX-Date: %s\r\nServer: %s\r\nMIME-version: 1.0\r\nLocation:
1158    %s\r\nURI: <%s>\r\nContent-type: text/html\r\nContent-Length: %ld\r\n\r\n", rfc1123date(timebuf, now), VERSION,
1159    url, url, length);
1160      set_head_and_body(*page, hbuffer, 0L, buffer, 0L);
1161      (*page)->filename = NULL;
1162      (*page)->last_modification = now;
1163      (*page)->last_stat = now;
1164      (*page)->ref_count = 0;
1165      (*page)->permanent = 0;
1166      return page;
1167    }
1168
1169    /*
1170     * Decode url
1171     */
1172    void decode_url_in_place(unsigned char *url)
1173    {
1174      unsigned char *where, *to;
1175      for(where = url, to = url; *where; )
1176        {
1177          if (*where == '%' &&
1178              isxdigit(where[1]) &&
1179              isxdigit(where[2]))
1180            {
1181              *to++ = (isdigit(where[1]) ? (where[1] - '0') :
```

```
1182                  (tolower(where[1]) - 'a' + 10)) * 16 +
1183                    (isdigit(where[2]) ? (where[2] - '0') :
1184                     (tolower(where[2]) - 'a' + 10));
1185            where += 3;
1186
1187          }
1188        else
1189          {
1190            *to++ = *where++;
1191          }
1192      }
1193    *to = '\0';
1194  }
1195
1196
1197  /*
1198   * Set the www-page for transfer
1199   */
1200  void do_get(int i, unsigned char *url, command_t cmd)
1201  {
1202    page_t *page, key_page_ptr;
1203    struct page_s key_page;
1204    unsigned char decoded_url[URL_LENGTH];
1205
1206    strcpy((char *) decoded_url, (char *) url);
1207    decode_url_in_place(decoded_url);
1208    key_page.url = decoded_url;
1209    key_page_ptr = &key_page;
1210    connections[i]->url = url;
1211
1212    DPRINT(("Finding url %s for connection %d", url, i));
1213    page = bsearch(&key_page_ptr, pages, num_pages, sizeof(page_t), compr_url);
1214    if (page == NULL)
1215      {
1216        int j;
```

```
1217
1218        DPRINT(("No permanent page found, finding temp"));
1219        for(j = 0; j < max_temp_pages; j++)
1220          {
1221            if (temp_pages[j] != NULL &&
1222                temp_pages[j]->url != NULL &&
1223                strcmp((char *) temp_pages[j]->url, (char *) decoded_url) == 0) {
1224              DPRINT(("Temp page found, created at %d",
1225                      temp_pages[j]->last_stat));
1226              page = &(temp_pages[j]);
1227              total_temp_cache_hit++;
1228              break;
1229            }
1230          }
1231      } else {
1232        total_perm_page++;
1233      }
1234    if (page == NULL)
1235      {
1236        redirection_t *redirection, key_redirection_ptr;
1237        struct redirection_s key_redirection;
1238        unsigned char *first_slash, *rest_of_url, url_buffer[URL_LENGTH];
1239
1240        DPRINT(("No page found, checking for forwards"));
1241
1242        first_slash = (unsigned char *) strchr((char *) url + 1, '/');
1243        if (first_slash != 0)
1244          {
1245            rest_of_url = ++first_slash;
1246          }
1247        else
1248          {
1249            rest_of_url = (unsigned char *) "";
1250          }
1251        strcpy((char *) url_buffer, (char *) decoded_url);
```

```
1252        first_slash = (unsigned char *) strchr((char *) url_buffer + 1, '/');
1253        if (first_slash != 0)
1254          {
1255            *++first_slash = '\0';
1256          }
1257
1258        key_redirection.from = url_buffer;
1259        key_redirection_ptr = &key_redirection;
1260
1261        redirection = bsearch(&key_redirection_ptr, redirections,
1262                              num_redirections, sizeof(redirection_t),
1263                              compr_redirection);
1264        if (redirection == NULL)
1265          {
1266            page = NULL;
1267          }
1268        else
1269          {
1270            page = make_redirection_page(*redirection, rest_of_url);
1271            total_temp_make++;
1272          }
1273      }
1274    if (page == NULL)
1275      {
1276        if (decoded_url[0] == '/' && decoded_url[1] == '~')
1277          {
1278            unsigned char *username, *slash, filepart[URL_LENGTH],
1279              fullpath[PATH_MAX];
1280            struct stat st;
1281            struct passwd *pw;
1282
1283            DPRINT(("Users home directory request : %s", decoded_url));
1284            total_user_home++;
1285
1286            username = decoded_url + 2;
```

```
1287            slash = (unsigned char *) strchr((char *) username, '/');
1288            if (slash == NULL)
1289              {
1290                strcpy((char *) filepart, "/");
1291              }
1292            else
1293              {
1294                strcpy((char *) filepart, (char *) slash);
1295                *slash = '\0';
1296              }
1297            DPRINT(("Finding user %s, filepart = %s", username, filepart));
1298            pw = getpwnam((char *) username);
1299            if (slash)
1300              *slash = '/';
1301            if (pw != NULL)
1302              {
1303                if (strlen(pw->pw_dir) + 20 + strlen((char *) filepart) <
1304                    PATH_MAX)
1305                  {
1306                    sprintf((char *) fullpath, "%s/public_html%s", pw->pw_dir,
1307                            filepart);
1308                    DPRINT(("Statting path = %s", fullpath));
1309                    if (stat((char *) fullpath, &st) >= 0)
1310                      {
1311                        if (st.st_mode & S_IFDIR)
1312                          {
1313                            if (strcmp((char *) filepart, ".") != 0 &&
1314                                strcmp((char *) filepart, "..") != 0)
1315                              {
1316                                struct redirection_s redirection;
1317
1318                                redirection.from = decoded_url;
1319                                if (decoded_url[strlen((char *) decoded_url)
1320                                                - 1] == '/')
1321                                  sprintf((char *) fullpath, "%s%sindex.html",
```

```
1322                                    local_root_url, decoded_url);
1323                            else
1324                              sprintf((char *) fullpath, "%s%s/index.html",
1325                                    local_root_url, decoded_url);
1326
1327                            redirection.to = fullpath;
1328                            page =
1329                              make_redirection_page(&redirection,
1330                                                    (unsigned char *) "");
1331                          }
1332                      }
1333                    else
1334                      {
1335                        page_t *new_page;
1336                        new_page = find_free_temp_page();
1337                        if (read_page(new_page, fullpath, &st, decoded_url))
1338                          {
1339                            page = new_page;
1340                            (*page)->permanent = 0;
1341                            total_bytes_in_cache += (*page)->reply_body_len;
1342                          }
1343                      }
1344                  }
1345              }
1346          }
1347      }
1348    }
1349  if (page == NULL)
1350    {
1351      DPRINT(("No page found, returning error"));
1352      set_out_data(connections[i], not_found, sizeof(not_found) - 1, 0, cmd);
1353      connections[i]->status = (unsigned char *) "page not found";
1354      connections[i]->page = NULL;
1355      total_errors_no_page++;
1356    }
```

```
1357    else
1358      {
1359        DPRINT(("Page found, checking age"));
1360        if ((*page)->filename != NULL &&
1361            (*page)->ref_count == 0 &&
1362            (*page)->last_stat + STAT_TIME < now)
1363          {
1364            struct stat st;
1365
1366            DPRINT(("Statting"));
1367            total_stat++;
1368            if (stat((char *) ((*page)->filename), &st) < 0)
1369              {
1370                syslog(LOG_ERR, "Error stat to %.200s failed",
1371                       (*page)->filename);
1372              }
1373            else
1374              {
1375                (*page)->last_stat = now;
1376                DPRINT(("Last modification is %d (was %d)", st.st_mtime,
1377                        (*page)->last_modification));
1378                if (st.st_mtime != (*page)->last_modification)
1379                  {
1380                    page_t new_page = NULL;
1381
1382                    total_read++;
1383                    if (read_page(&new_page, (*page)->filename, &st,
1384                                  (*page)->url))
1385                      {
1386                        if ((*page)->permanent)
1387                          {
1388                            new_page->permanent = 1;
1389                          }
1390                        else
1391                          {
```

```
1392                        total_bytes_in_cache -= (*page)->reply_body_len;
1393                        total_bytes_in_cache += new_page->reply_body_len;
1394                      }
1395                    DPRINT(("Freeing old page and using new"));
1396                    free((*page)->url);
1397                    free((*page)->filename);
1398                    free((*page)->reply_head);
1399                    free(*page);
1400                    *page = new_page;
1401                  }
1402                }
1403              }
1404            }
1405
1406          set_out_data(connections[i], (*page)->reply_head,
1407                       (*page)->reply_head_len + (*page)->reply_body_len,
1408                       (*page)->reply_head_len, cmd);
1409          connections[i]->status = (unsigned char *) "page found";
1410          connections[i]->page = *page;
1411          (*page)->ref_count++;
1412        }
1413    }
1414
1415    /*
1416     * Remove expired pages
1417     */
1418    void cleanup_temp_cache()
1419    {
1420      int i;
1421
1422      for(i = 0; i < max_temp_pages; i++)
1423        {
1424          if (temp_pages[i] != NULL &&
1425              temp_pages[i]->url != NULL &&
1426              temp_pages[i]->ref_count == 0 &&
```

```
1427            temp_pages[i]->last_ref + KEEP_TIME < now)
1428          {
1429            DPRINT(("Clening page %s, %d bytes (last ref = %d)",
1430                    temp_pages[i]->url, temp_pages[i]->reply_body_len,
1431                    temp_pages[i]->last_ref));
1432            total_bytes_in_cache -= temp_pages[i]->reply_body_len;
1433            free(temp_pages[i]->url);
1434            free(temp_pages[i]->reply_head);
1435            if (temp_pages[i]->filename != NULL)
1436              free(temp_pages[i]->filename);
1437            temp_pages[i]->url = NULL;
1438            temp_pages[i]->reply_head = NULL;
1439            temp_pages[i]->reply_body = NULL;
1440            temp_pages[i]->filename = NULL;
1441            temp_pages[i]->reply_head_len = 0L;
1442            temp_pages[i]->reply_body_len = 0L;
1443            temp_pages[i]->last_modification = 0;
1444            temp_pages[i]->last_stat = 0;
1445            temp_pages[i]->last_ref = 0;
1446          }
1447      }
1448    DPRINT(("Cleaned cache, total %d bytes remaining", total_bytes_in_cache));
1449  }
1450
1451  /*
1452   * New connection
1453   */
1454  void new_connection()
1455  {
1456    struct sockaddr_in rsin;
1457    socklen_t rsinlen;
1458    int i;
1459
1460    rsinlen = sizeof(rsin);
1461
```

```
1462    total_connections++;
1463    for(i = 0; i < max_connections; i++)
1464      {
1465        if (connections[i] == NULL ||
1466            connections[i]->socket == -1)
1467          break;
1468      }
1469    if (i > total_max_connections)
1470      total_max_connections = i;
1471    if (i == max_connections)
1472      {
1473        int client;
1474        client = accept(master_server, (struct sockaddr *) &rsin, &rsinlen);
1475        syslog(LOG_ERR, "Too many connections, dropping connection from %.200s",
1476               inet_ntoa(rsin.sin_addr));
1477        close(client);
1478      }
1479    else
1480      {
1481        if (connections[i] == NULL)
1482          {
1483            connections[i] = calloc(1, sizeof(struct connection_s));
1484            if (connections[i] == NULL)
1485              {
1486                syslog(LOG_CRIT, "Out of memory, exiting");
1487                exit(1);
1488              }
1489            connections[i]->socket = -1;
1490            connections[i]->in_data = malloc(COMMAND_LENGTH);
1491            if (connections[i]->in_data == NULL)
1492              {
1493                syslog(LOG_CRIT, "Out of memory, exiting");
1494                exit(1);
1495              }
1496          }
```

```
1497        connections[i]->last_time = now;
1498        connections[i]->write_state = STATE_NONE;
1499        connections[i]->read_state = STATE_NONE;
1500        connections[i]->in_data_ptr = connections[i]->in_data;
1501        connections[i]->command = (unsigned char *) "(no command)";
1502        connections[i]->headers = (unsigned char *) "(no headers)";
1503        connections[i]->url = (unsigned char *) "(no url)";
1504        connections[i]->in_data_len = 0L;
1505        connections[i]->socket = accept(master_server, (struct sockaddr *) &rsin,
1506                                        &rsinlen);
1507        if (connections[i]->socket < 0)
1508          {
1509            syslog(LOG_ERR, "Accept failed from %.200s", inet_ntoa(rsin.sin_addr));
1510            connections[i]->socket = -1;
1511          }
1512        else
1513          {
1514            if (fcntl(connections[i]->socket, F_SETFL,
1515                      fcntl(connections[i]->socket, F_GETFL, 0) | FNDELAY) < 0)
1516              {
1517                syslog(LOG_ERR, "fcntl failed for %.200s", inet_ntoa(rsin.sin_addr));
1518                close(connections[i]->socket);
1519                connections[i]->socket = -1;
1520              }
1521            else
1522              {
1523                int one;
1524                one = 1;
1525
1526                if (setsockopt(connections[i]->socket, SOL_SOCKET,
1527                               SO_REUSEADDR, (char *) &one, sizeof(int)) == -1)
1528                  {
1529                    syslog(LOG_ERR, "Setsockopt REUSEADDR fails");
1530                  }
1531                connections[i]->status = (unsigned char *) "connected";
```

```
1532                FD_SET(connections[i]->socket, &fdrdset);
1533                connections[i]->addr = rsin.sin_addr;
1534                number_of_sockets++;
1535                DPRINT(("New connection %d from %s", i,
1536                        inet_ntoa(rsin.sin_addr)));
1537              }
1538            }
1539        }
1540    }
1541
1542    /*
1543     * Data read from socket
1544     */
1545    void read_data(int i)
1546    {
1547      int data_read;
1548
1549      data_read = read(connections[i]->socket, connections[i]->in_data_ptr,
1550                       COMMAND_LENGTH - connections[i]->in_data_len);
1551      if (connections[i]->read_state == STATE_ERROR ||
1552          connections[i]->read_state == STATE_TIMED_OUT ||
1553          connections[i]->read_state == STATE_DONE)
1554        return;
1555      if (data_read <= 0)
1556        {
1557          if (data_read < 0)
1558            {
1559              if (errno == EWOULDBLOCK)
1560                return;
1561              syslog(LOG_WARNING, "Read error from host %.200s : %d",
1562                     inet_ntoa(connections[i]->addr), errno);
1563              connections[i]->status = (unsigned char *) "read error";
1564              connections[i]->read_state = STATE_ERROR;
1565              connections[i]->write_state = STATE_ERROR;
1566            }
```

```
1567        connections[i]->status = (unsigned char *) "other end of socket closed";
1568        connections[i]->read_state = STATE_DONE;
1569        FD_CLR(connections[i]->socket, &fdrdset);
1570      }
1571    else
1572      {
1573        DPRINT(("Read %d bytes", data_read));
1574        connections[i]->in_data_ptr += data_read;
1575        connections[i]->in_data_len += data_read;
1576      }
1577  }
1578
1579  /*
1580   * parse command
1581   */
1582  void parse_command(int i)
1583  {
1584    unsigned char *eol1, *eol2;
1585    unsigned char *command, *uri, *protocol, *tmp;
1586
1587    eol1 = memchr(connections[i]->in_data, '\n', connections[i]->in_data_len);
1588    eol2 = memchr(connections[i]->in_data, '\r', connections[i]->in_data_len);
1589    if (eol1 == NULL && eol2 == NULL)
1590      {
1591        if (connections[i]->in_data_len >= COMMAND_LENGTH)
1592          {
1593            syslog(LOG_WARNING, "No url in buffer range from %.200s",
1594                   inet_ntoa(connections[i]->addr));
1595            connections[i]->status = (unsigned char *) "read buffer overflow";
1596            connections[i]->read_state = STATE_ERROR;
1597            connections[i]->write_state = STATE_ERROR;
1598          }
1599        if (connections[i]->read_state == STATE_DONE)
1600          {
1601            return_error(i);
```

```
1602          }
1603        return;
1604      }
1605
1606    connections[i]->read_state = STATE_COMMAND_READ;
1607    FD_SET(connections[i]->socket, &fdwrset);
1608
1609    if (eol1 == NULL)
1610      eol1 = eol2;
1611    if (eol2 == NULL)
1612      eol2 = eol1;
1613    if (eol1 > eol2)
1614      eol1 = eol2;
1615
1616    *eol1 = '\0';
1617    connections[i]->headers = ++eol1;
1618
1619    command = connections[i]->in_data;
1620    command = skip_white(command);
1621    connections[i]->command = command;
1622    uri = skip_non_white(command);
1623    if (*uri != '\0')
1624      {
1625        *uri++ = '\0';
1626        uri = skip_white(uri);
1627        protocol = skip_non_white(uri);
1628        if (*protocol != '\0')
1629          {
1630            *protocol++ = '\0';
1631            protocol = skip_white(protocol);
1632            tmp = skip_non_white(protocol);
1633            *tmp++ = '\0';
1634          }
1635      }
1636    else
```

```
1637      {
1638        protocol = uri;
1639        *connections[i]->headers = '\0';
1640      }
1641
1642    connections[i]->url = uri;
1643    DPRINT(("Parsed command = %s, uri = %s, protocol = %s", command, uri,
1644           protocol));
1645
1646    if (strcasecmp((char *) command, "get") == 0 ||
1647        strcasecmp((char *) command, "head") == 0 ||
1648        strcasecmp((char *) command, "post") == 0)
1649      {
1650        command_t cmd;
1651
1652        cmd = COMMAND_BOTH;
1653        if (!*protocol)
1654          cmd = COMMAND_BODY;
1655        else if (strcasecmp((char *) command, "head") == 0)
1656          cmd = COMMAND_HEAD;
1657
1658        if (*uri == '\0')
1659          {
1660            return_error(i);
1661            return;
1662          }
1663        do_get(i, uri, cmd);
1664
1665        if (cmd != COMMAND_BODY &&
1666            strncasecmp((char *) protocol, "http/1.", 6) != 0)
1667          {
1668            return_error(i);
1669            return;
1670          }
1671        connections[i]->total_bytes = connections[i]->out_data_len;
```

```
1672        do_write(i);
1673        return;
1674      }
1675    else if (strcasecmp((char *) command, "quit") == 0)
1676      {
1677        quit_time = now + QUIT_TIME;
1678        close(master_server);
1679        FD_CLR(master_server, &fdrdset);
1680        number_of_sockets--;
1681        set_out_data(connections[i], done, sizeof(done) - 1, 0, COMMAND_BOTH);
1682        connections[i]->total_bytes = connections[i]->out_data_len;
1683        connections[i]->status = (unsigned char *) "quit done";
1684        do_write(i);
1685        return;
1686      }
1687    else if (strcasecmp((char *) command, "status") == 0)
1688      {
1689        sprintf((char *) status + status_head_len,
1690                "<HEAD><TITLE>Status</TITLE></HEAD>\n<BODY><H1>Status</H1>\nStatistics time = %ld<BR>\n%d
1691   connections (%d max)<BR>\n%d pages (%d no page errors/%d errors)<BR>\n%d perm, %d temp cache, %d temp make, %d
1692   user home<BR>\n%d stated, %d reread<BR>\n%d/%d bytes sent<BR>\nCache: %d pages, %d redirections, %d bytes in temp
1693   cache<BR>\n</BODY>\n",
1694                now - last_info_hour,
1695                total_connections, total_max_connections,
1696                total_pages, total_errors_no_page, total_errors,
1697                total_perm_page, total_temp_cache_hit, total_temp_make,
1698                total_user_home,
1699                total_stat, total_read,
1700                total_bytes_sent, total_bytes,
1701                num_pages, num_redirections, total_bytes_in_cache);
1702        set_out_data(connections[i], status, strlen((char *) status),
1703                     status_head_len,
1704                     COMMAND_BOTH);
1705        connections[i]->total_bytes = connections[i]->out_data_len;
1706        connections[i]->status = (unsigned char *) "status info done";
```

```
      do_write(i);
      return;
    }
  return_error(i);
}

/*
 * parse command
 */
void parse_headers(int i)
{
  unsigned char *p;
  long len;

  len = connections[i]->in_data_len -
    (connections[i]->headers - connections[i]->in_data);

  for(p = connections[i]->headers; len > 0 ; len--, p++)
    {
      if ((p[0] == '\n' &&
           ((len >= 4 && p[1] == '\r' && p[2] == '\n' && p[3] == '\r') ||
            (len >= 2 && p[1] == '\n'))) ||
          (p[0] == '\r' &&
           ((len >= 4 && p[1] == '\n' && p[2] == '\r' && p[3] == '\n') ||
            (len >= 2 && p[1] == '\r'))))
        {
          p[1] = '\0';
          connections[i]->read_state = STATE_DONE;
          return;
        }
    }
}


/*
```

```
1742    * master server
1743    */
1744   void http_server()
1745   {
1746     fd_set tmprdset, tmpwrset;
1747     int nfound, i;
1748     time_t last_cleanup_time;
1749     struct timeval tm;
1750     struct tm *t;
1751
1752     max_connections = getdtablesize();
1753
1754     connections = calloc(max_connections, sizeof(connection_t));
1755     if (connections == NULL)
1756       {
1757         syslog(LOG_CRIT, "Out of memory, exiting");
1758         exit(1);
1759       }
1760
1761     FD_ZERO(&fdrdset);
1762     FD_ZERO(&fdwrset);
1763     FD_SET(master_server, &fdrdset);
1764     number_of_sockets = 1;
1765
1766     now = time(NULL);
1767     last_cleanup_time = now;
1768     last_info_hour = now;
1769     /* Round to exact hour */
1770     t = localtime(&last_info_hour);
1771     last_info_hour -= t->tm_sec + t->tm_min * 60;
1772
1773     while (number_of_sockets > 0)
1774       {
1775         now = time(NULL);
1776         if (quit_time != 0 && now > quit_time)
```

```
1777          {
1778            syslog(LOG_NOTICE, "Server didn't die after %d seconds, exiting",
1779                   QUIT_TIME);
1780            break;
1781          }
1782        if (last_cleanup_time + CHECK_TIME <= now)
1783          {
1784            cleanup_temp_cache();
1785            last_cleanup_time = now;
1786          }
1787        if (last_info_hour + HOUR <= now)
1788          {
1789            syslog(LOG_NOTICE, "%d connections (%d max), %d pages (%d no page errors/%d errors), %d perm, %d temp
1790    cache, %d temp make, %d user home, %d stated, %d reread, %d/%d bytes sent. Cache: %d pages, %d redirections, %d
1791    bytes in temp cache",
1792                   total_connections, total_max_connections,
1793                   total_pages, total_errors_no_page, total_errors,
1794                   total_perm_page, total_temp_cache_hit, total_temp_make,
1795                   total_user_home,
1796                   total_stat, total_read,
1797                   total_bytes_sent, total_bytes,
1798                   num_pages, num_redirections, total_bytes_in_cache);
1799            total_connections = 0;
1800            total_max_connections = 0;
1801            total_pages = 0;
1802            total_errors_no_page = 0;
1803            total_errors = 0;
1804            total_perm_page = 0;
1805            total_temp_cache_hit = 0;
1806            total_temp_make = 0;
1807            total_user_home = 0;
1808            total_stat = 0;
1809            total_read = 0;
1810            total_bytes_sent = 0;
1811            total_bytes = 0;
```

```
1812            last_info_hour += HOUR;
1813          }
1814        tmprdset = fdrdset;
1815        tmpwrset = fdwrset;
1816        tm.tv_sec = last_cleanup_time + CHECK_TIME - now;
1817        tm.tv_usec = 0;
1818        nfound = select(FD_SETSIZE, &tmprdset, &tmpwrset, NULL, &tm);
1819        if(nfound < 0 && errno != EINTR)
1820          {
1821            syslog(LOG_CRIT, "Select failed in main_loop, exiting");
1822            exit(1);
1823          }
1824        if (nfound < 0 && errno == EINTR)
1825          continue;
1826
1827        if (FD_ISSET(master_server, &tmprdset))
1828          new_connection();
1829        for(i = 0; i < max_connections; i++)
1830          {
1831            if (connections[i] == NULL)
1832              break;
1833            if (connections[i]->socket == -1)
1834              continue;
1835            if (FD_ISSET(connections[i]->socket, &tmprdset))
1836              {
1837                connections[i]->last_time = now;
1838                if (connections[i]->read_state == STATE_NONE)
1839                  {
1840                    connections[i]->read_state = STATE_DOING;
1841                  }
1842
1843                read_data(i);
1844
1845                if (connections[i]->read_state == STATE_DOING)
1846                  {
```

```
1847                parse_command(i);
1848              }
1849            if (connections[i]->read_state == STATE_COMMAND_READ)
1850              {
1851                parse_headers(i);
1852              }
1853          }
1854
1855        if (connections[i]->read_state == STATE_COMMAND_READ &&
1856            now - connections[i]->last_time > READ_TIME_OUT)
1857          {
1858            connections[i]->read_state = STATE_TIMED_OUT;
1859          }
1860
1861        if (FD_ISSET(connections[i]->socket, &tmpwrset))
1862          {
1863            connections[i]->last_time = now;
1864            do_write(i);
1865          }
1866
1867        if ((connections[i]->read_state == STATE_ERROR ||
1868             connections[i]->read_state == STATE_TIMED_OUT ||
1869             connections[i]->read_state == STATE_DONE) &&
1870            (connections[i]->write_state == STATE_ERROR ||
1871             connections[i]->write_state == STATE_DONE))
1872          close_connection(i);
1873
1874        if (now - connections[i]->last_time > KICK_TIME)
1875          {
1876            close_connection(i);
1877          }
1878      }
1879  }
```

```
1880    syslog(LOG_NOTICE, "%d connections (%d max), %d pages (%d no page errors/%d errors), %d perm, %d temp cache, %d
1881  temp make, %d user home, %d stated, %d reread, %d/%d bytes sent. Cache: %d pages, %d redirections, %d bytes in
1882  temp cache",
1883           total_connections, total_max_connections,
1884           total_pages, total_errors_no_page, total_errors,
1885           total_perm_page, total_temp_cache_hit, total_temp_make,
1886           total_user_home,
1887           total_stat, total_read,
1888           total_bytes_sent, total_bytes,
1889           num_pages, num_redirections, total_bytes_in_cache);
1890  }
1891
1892  int main(int argc, char **argv)
1893  {
1894    extern char *optarg;
1895    extern int optind;
1896    int c, errflg = 0;
1897
1898    signal(SIGPIPE, SIG_IGN);
1899
1900    now = time(NULL);
1901    total_connections = 0;
1902    total_max_connections = 0;
1903    total_pages = 0;
1904    total_errors_no_page = 0;
1905    total_errors = 0;
1906    total_perm_page = 0;
1907    total_temp_cache_hit = 0;
1908    total_temp_make = 0;
1909    total_user_home = 0;
1910    total_stat = 0;
1911    total_read = 0;
1912    total_bytes_sent = 0;
1913    total_bytes = 0;
1914
```

```
  program = strrchr(argv[0], '/');
  if (program == NULL)
    program = argv[0];
  else
    program++;

#ifdef RLIMIT_NOFILE
  {
    struct rlimit rl;

    if (getrlimit(RLIMIT_NOFILE, &rl) >= 0)
      {
        rl.rlim_cur = rl.rlim_max;
        setrlimit(RLIMIT_NOFILE, &rl);
      }
  }
#endif /* RLIMIT_NOFILE */

  openlog(program, LOG_PID, LOG_LOCAL0);

  while ((c = getopt(argc, argv, "dig:u:p:")) != EOF)
    {
      switch (c)
        {
        case 'd': f_daemon++; break;
        case 'i': f_inetd++; break;
        case 'g': f_gid = atoi(optarg); break;
        case 'u': f_uid = atoi(optarg); break;
        case 'p': f_port = optarg; break;
        case '?': errflg++; break;
        }
    }
  if (errflg || argc - optind < 2)
    {
      fprintf(stderr, "Usage: %s [-di] [-g gid] [-u uid] [-p service] redir-file htdocs-dir\n",
```

```
1950                program);
1951          exit(1);
1952        }
1953
1954      if (f_daemon)
1955        {
1956          int pid, i;
1957
1958          pid = fork();
1959
1960          if (pid)
1961            {
1962              if (pid == -1)
1963                {
1964                  syslog(LOG_CRIT, "Can't fork a child, exiting!");
1965                  exit(1);
1966                }
1967
1968              printf("%d\n", pid);
1969              fflush(stdout);
1970              exit(0);
1971            }
1972          closelog();
1973          for(i = getdtablesize() - 1; i >= 0; i--)
1974            if (!f_inetd || i != 0)
1975              close(i);
1976          setsid();
1977          openlog(program, LOG_PID, LOG_DAEMON);
1978        }
1979
1980      if (f_inetd)
1981        master_server = 0;
1982      else
1983        master_server = open_service(f_port);
1984
```

```
1985    if (getuid() == 0)
1986      {
1987        if (f_gid != -1)
1988          setgid(f_gid);
1989        if (f_uid != -1)
1990          setuid(f_uid);
1991      }
1992
1993    if (port == 80 || port == -1)
1994      {
1995        sprintf((char *) local_root_url, "%s", LOCAL_ROOT_URL);
1996      }
1997    else
1998      {
1999        sprintf((char *) local_root_url, "%s:%d", LOCAL_ROOT_URL, port);
2000      }
2001    read_pages((unsigned char *) argv[optind],
2002               (unsigned char *) argv[optind + 1]);
2003    http_server();
2004    return 0;
2005  }
```

Appendix B

In this appendix there are listings of the CCCC SCM implementation.

The SCM Manager Class

cccc_scm.h

```
1  // cccc_scm.h: interface for the CCCC_ScmManager class.
2  //
3  //////////////////////////////////////////////////////////////////////
4
5  #if !defined(AFX_CCCC_SCM_H__74D8CAD6_8692_4DC1_A570_45DD319FB424__INCLUDED_)
6  #define AFX_CCCC_SCM_H__74D8CAD6_8692_4DC1_A570_45DD319FB424__INCLUDED_
7
8  #if _MSC_VER > 1000
9  #pragma once
10 #endif // _MSC_VER > 1000
11
12 #include <map>
13 #include <string>
14 #include <vector>
15 using namespace std;
16
17 typedef enum
18 {
19     SCM_NORMAL = 0,
20     SCM_TRY_LOCK,
21     SCM_LOCK,
22     SCM_UNLOCK,
23     SCM_WAIT,
24     SCM_NOTIFY,
25     SCM_YIELD, // PASS CONTROL
```

```
26      SCM_VOLATILE,
27      SCM_TASK_START,
28      SCM_THREAD_START,
29      SCM_TASK_STOP,
30      SCM_THREAD_STOP,
31      SCM_LAST,
32  } TSCMTypes;
33
34  class SyncPoint
35  {
36  public:
37
38      SyncPoint()
39      {
40          Init(SCM_NORMAL, -1, -1);
41      };
42
43      SyncPoint(TSCMTypes type, int IP, int CP)
44      {
45          Init(type, IP, CP);
46      };
47
48      SyncPoint(TSCMTypes type)
49      {
50          Init(type, -1, -1);
51      };
52
53      SyncPoint(TSCMTypes type, int CP)
54      {
55          Init(type, -1, CP);
56      };
57
58      SyncPoint(const SyncPoint &orig)
59      {
60          Init(orig.m_type, orig.m_IP, orig.m_CP);
```

```
61      };
62
63
64      void Init(TSCMTypes type)
65      {
66          Init(type, -1, -1);
67      };
68
69      void Init(TSCMTypes type, int CP)
70      {
71          Init(type, -1,CP);
72      };
73
74      void Init(TSCMTypes type, int IP, int CP)
75      {
76          m_type = type;
77          m_IP = IP;
78          m_CP = CP;
79      };
80
81      int GetCP(){return m_CP;};
82      int GetIP(){return m_IP;};
83      TSCMTypes GetType(){return m_type;};
84
85      ~SyncPoint(){};
86  private:
87      TSCMTypes m_type;
88      int m_IP;
89      int m_CP;
90  } ;
91
92  typedef struct
93  {
94      int IP;
95      int CP;
```

```
96      unsigned MVG_sp;
97  } tsSpData;
98
99  class CCCC_ScmManager
100 {
101 public:
102         CCCC_ScmManager();
103     static void PrepareScmForVolatile();
104     static void Initialize(string IdsFile = "scmids.dat", string PotentialFiles = "scmpotentials.dat");
105     static void FillDefaultPotentials();
106     static void FillDefaultIds();
107     static TSCMTypes ConsumeToken(ANTLRToken *pToken);
108     static void CalculateScm();
109         virtual ~CCCC_ScmManager();
110 protected:
111     static string m_PreviousToken;
112     static map<string, SyncPoint> m_ScmIdentifiers;
113     static int m_Potentials[(int)SCM_LAST];
114     static int m_Competition;
115     static bool m_NextIsVolatile;
116     static bool m_Initialized;
117     static void InitIds(string fileName);
118     static void InitPotentials(string fileName);
119     static vector<tsSpData > m_SpValues;
120 };
121
122 #endif // !defined(AFX_CCCC_SCM_H__74D8CAD6_8692_4DC1_A570_45DD319FB424__INCLUDED_)
```

cccc_scm.cc

```
1  // cccc_scm.cc: implementation of the CCCC_ScmManager class.
2  //
3  ////////////////////////////////////////////////////////////////////
4
5  #include "cccc_tok.h"
6  #include "cccc_scm.h"
7  #include "cccc_utl.h"
8
9  map<string, SyncPoint> CCCC_ScmManager::m_ScmIdentifiers;
10 bool CCCC_ScmManager::m_NextIsVolatile = FALSE;
11 string CCCC_ScmManager::m_PreviousToken = "";
12 int CCCC_ScmManager::m_Potentials[(int)SCM_LAST] = {0};
13 int CCCC_ScmManager::m_Competition = 0;
14 bool CCCC_ScmManager::m_Initialized = false;
15 vector<tsSpData> CCCC_ScmManager::m_SpValues;
16
17 ////////////////////////////////////////////////////////////////////
18 // Construction/Destruction
19 ////////////////////////////////////////////////////////////////////
20
21 #include <string>
22 #include <fstream>
23 using namespace std;
24
25 CCCC_ScmManager::CCCC_ScmManager()
26 {
27 }
28
29 void CCCC_ScmManager::FillDefaultIds()
30 {
31     SyncPoint sp;
32 // HTTPD - synchronization points for use with UN*X HTTP server implementations.
33     sp.Init(SCM_LOCK);
```

```
34      m_ScmIdentifiers["create_and_bind_stream_or_die"] = sp;
35      m_ScmIdentifiers["safe_read"] = sp;
36
37      sp.Init(SCM_NOTIFY);
38      m_ScmIdentifiers["shutdown"] = sp;
39      m_ScmIdentifiers["signal"] = sp;
40      m_ScmIdentifiers["sigaction"] = sp;
41          m_ScmIdentifiers["send"] = sp;
42          m_ScmIdentifiers["write"] = sp;
43
44      sp.Init(SCM_WAIT);
45      m_ScmIdentifiers["select"] = sp;
46      m_ScmIdentifiers["full_write"] = sp;
47      m_ScmIdentifiers["full_read"] = sp;
48      m_ScmIdentifiers["read"] = sp;
49      m_ScmIdentifiers["accept"] = sp;
50      m_ScmIdentifiers["xlisten"] = sp;
51      m_ScmIdentifiers["listen"] = sp;
52
53
54      sp.Init(SCM_TASK_START);
55      m_ScmIdentifiers["fork"] = sp;
56      m_ScmIdentifiers["execv"] = sp;
57
58  }
59
60  void CCCC_ScmManager::FillDefaultPotentials()
61  {
62      // CP and IP for the HTTPD comparison
63      m_Competition = 2;
64      m_Potentials[SCM_NORMAL] = 0;
65      m_Potentials[SCM_TRY_LOCK] = 0;
66      m_Potentials[SCM_LOCK] = 2;
67      m_Potentials[SCM_UNLOCK] = 1;
68      m_Potentials[SCM_WAIT] = 2;
```

```
69      m_Potentials[SCM_NOTIFY] = 2;
70      m_Potentials[SCM_YIELD] = 0;
71      m_Potentials[SCM_VOLATILE] = 2;
72      m_Potentials[SCM_TASK_START] = 1;
73      m_Potentials[SCM_THREAD_START] = 2;
74      m_Potentials[SCM_TASK_STOP] = 0;
75      m_Potentials[SCM_THREAD_STOP] = 0;
76  }
77
78
79  void CCCC_ScmManager::InitPotentials(string fileName)
80  {
81      TSCMTypes t=SCM_NORMAL;
82      fstream inFile;
83      inFile.open(fileName.c_str(), ios::in);
84      int i;
85
86      bool success = (! inFile.fail());
87
88      while ((! inFile.fail()) && (t != SCM_LAST))
89          {
90          if (SCM_NORMAL == t)
91              {
92              inFile >> m_Competition;
93              }
94          t = (TSCMTypes)(((int)t)+1);
95          inFile >> i;
96          m_Potentials[(int)t] = i;
97          }
98      inFile.close();
99
100     if (! success)
101         {
102         FillDefaultPotentials();
103         }
```

```
104 }
105
106
107 void CCCC_ScmManager::InitIds(string fileName)
108 {
109     string TypeStrings[SCM_LAST] =
110     {
111         "SCM_NORMAL",
112         "SCM_TRY_LOCK",
113         "SCM_LOCK",
114         "SCM_UNLOCK",
115         "SCM_WAIT",
116         "SCM_NOTIFY",
117         "SCM_YIELD",
118         "SCM_VOLATILE",
119         "SCM_TASK_START",
120         "SCM_THREAD_START",
121         "SCM_TASK_STOP",
122         "SCM_THREAD_STOP"
123     };
124     bool success;
125     string identifier, type;
126     int IP, CP;
127     fstream inFile;
128     inFile.open(fileName.c_str(), ios::in);
129
130     success = (! inFile.fail());
131
132     while (! inFile.fail())
133         {
134         inFile >> identifier >> type >> IP >> CP;
135         for (int t = (int)SCM_NORMAL; t < (int)SCM_LAST; t++)
136             {
137             if (TypeStrings[t] == type)
138                 {
```

```
                SyncPoint sp((TSCMTypes)t, IP, CP);
                m_ScmIdentifiers[identifier] = sp;
                }
            }
        }

    inFile.close();

    if (!success)
        {
        FillDefaultIds();
        }
}

void CCCC_ScmManager::PrepareScmForVolatile()
{
    m_NextIsVolatile = true;
}

void CCCC_ScmManager::Initialize(string IdsFile, string PotentialFiles)
{
    InitIds(IdsFile);
    InitPotentials(PotentialFiles);
    m_Initialized = true;
}

TSCMTypes CCCC_ScmManager::ConsumeToken(ANTLRToken *pToken)
{
    TSCMTypes type = SCM_NORMAL;
        string sToken = pToken->getText();

    // if first token - initialize
    if (! m_Initialized)
        {
        Initialize();
```

```
174         }
175
176     if (m_NextIsVolatile)
177         {
178         // if delimiter found - the previous token was identifier. If we're in a volatile definition - take the
179 identifier as volatile
180         if ((sToken == ",")||(sToken == ";"))
181             {
182             m_ScmIdentifiers[m_PreviousToken] = SCM_VOLATILE;
183             m_NextIsVolatile = false;
184             }
185         }
186     else if (sToken == "volatile")
187         {
188         // if the token is "volatile" - raise the flag to wait for the identifier.
189         // although it's supposed to be raised through the parser, raise it here as well.
190         m_NextIsVolatile = true;
191         }
192     else
193         {
194         // get the type of the current identifier
195         if ((m_ScmIdentifiers.count(sToken))||(sToken == "t_start"))
196             {
197             // calculate SCM for the current token
198             type = m_ScmIdentifiers[sToken].GetType();
199             fprintf(stderr,"\nSCM: For token %s found type %d.\n", sToken, type);
200             // insert the current token into the nesting map
201             sData.IP = m_Potentials[type];
202             sData.CP = m_Competition;
203             sData.MVG_sp = ParseStore::currentInstance()->GetCCNspValue();
204             m_SpValues.push_back(sData);
205             }
206         }
207
208     m_PreviousToken = sToken;
```

```
209         return type;
210     }
211
212     void CCCC_ScmManager::CalculateScm()
213     {
214         tsSpData sData;
215         vector<tsSpData>::iterator it;
216         // get the IP and pass to the ParseStore with the current CP,
217         // for each of the encountered synchronization points
218         // in this function
219          for (it = m_SpValues.begin(); it != m_SpValues.end();  it++)
220              {
221              sData = *it;
222              ParseStore::currentInstance()->IncrementSCM(sData.IP, sData.CP, sData.MVG_sp);
223              }
224          // delete the IP's of the synchronization points for the current nesting level.
225          m_SpValues.clear();
226          // clear the CCNsg counter
227          ParseStore::currentInstance()->PerformReturn();
228     }
229
230     CCCC_ScmManager::~CCCC_ScmManager()
231     {
232     }
```

The ParseStore Class

cccc_utils.h

The listing below of the whole cccc_utils.h file, which includes utility classes and functions for the CCCC. The changes relevant to this work are between lines 227 and 250, and are marked with the yellow background. A new member `m_CCNSpHelper` that was added to the class on line 344 is initialized in its constructor. This trivial (and only) change in `cccc_utils.cc` is not shown here.

```
/*
    CCCC - C and C++ Code Counter
    Copyright (C) 1994-2005 Tim Littlefair (tim_littlefair@hotmail.com)

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301 USA
*/
// cccc_utl.h

#ifndef __CCCC_UTL_H
#define __CCCC_UTL_H

#include "cccc.h"
#include <map>
#include <vector>
#include "cccc_tok.h"
#include "AParser.h"

class ANTLRAbstractToken;
class ANTLRTokenPtr;
class CCCC_Item;

// this file declares all enumeration datatypes used in the project, and
```

```
35  // also the parse state class, which is used to capture information in the
36  // parse and transfer it to the code database for later report generation
37
38  // for each enumeration, a single character code is defined for each member
39  // these codes are shown in the inline comments
40
41  // the enumerations are designed to support resolution of incomplete
42  // knowledge about several sections of code which relate to the same
43  // object to give the most complete picture available
44
45  class AST;
46
47  // the languages which can be parsed
48  // only C and C++ are implemented as yet
49  enum Language { lAUTO, lCPLUSPLUS, lANSIC, lJAVA, lADA };
50  extern Language global_language, file_language;
51
52  enum Visibility {
53    vPUBLIC='0',vPROTECTED='1',vPRIVATE='2',vIMPLEMENTATION='3',
54    vDONTKNOW='?',vDONTCARE='X',vINVALID='*'
55  };
56  ostream& operator << (ostream&, Visibility);
57  istream& operator >> (istream&, Visibility&);
58
59  enum AugmentedBool {
60    abFALSE='F', abTRUE='T', abDONTKNOW='?', abDONTCARE='X', abINVALID='*'
61  };
62  ostream& operator << (ostream& os, AugmentedBool ab);
63  istream& operator >> (istream& is, AugmentedBool& ab);
64
65  enum UseType {
66    utDECLARATION='D', utDEFINITION='d',  // of methods and classes
67    utINHERITS='I',                       // inheritance, including Java
68    // extends and implements relations
69    utHASBYVAL='H', utHASBYREF='h',       // class data member
```

```
70    utPARBYVAL='P', utPARBYREF='p',       // method parameter or return value
71    utVARBYVAL='V', utVARBYREF='v',       // local variable within a method
72    utTEMPLATE_NAME='T',                  // typedef alias for a template
73    utTEMPLATE_TYPE='t',                  // type over which a template is
74    // instantiated
75    utINVOKES='i',                        // C function invocation
76    utREJECTED='r',                       // for extents rejected by the parser
77    utWITH='w',                           // Ada 'with' keyword context
78    utDONTKNOW='?', utDONTCARE='X', utINVALID='*'
79  };
80
81  // the parse state object consists of a number of strings representing
82  // knowledge about the identification of the source code object currently
83  // being processed, a number of flags of type AugmentedBool, and
84  // items representing knowledge about the
85  // concerning the object's nature, and also its visibility
86
87  enum PSString {
88    pssFILE, pssRULE, pssFLAGS, // the context of the parse
89    pssMODTYPE, pssMODULE,      // the syntactic class and name of the module
90    pssUTYPE,                    // unqualified type of the current member
91    pssINDIR,                   // indirection associated with the type above
92    pssITYPE,                   // type qualified with indirection
93    pssMEMBER, pssPARAMS,       // name, parameter list of a member
94    pssDESCRIPTION,             // textual description of the relationship type
95    pssLAST                     // used to dimension the array
96  };
97
98  enum PSFlag {
99    psfCONST, psfSTATIC, psfEXTERN, psfVIRTUAL, // AugmentedBool
100   psfVISIBILITY,                              // Visibility
101   psfLAST                                     // used to dimension the array
102 };
103 enum PSVerbosity { psvSILENT, psvQUIET, psvLOUD };
104
```

```
105  #define MAX_STACK_DEPTH 1000
106
107  // I have moved some actions originally embedded within the C++ grammar
108  // out of the grammar into the class ParseUtility defined below, so that
109  // other grammars can use them as well for consistency and efficiency.
110  // The ParseUtility::resynchronize() method provides a standardised way
111  // of 1) resynchronising the parser, and 2) reporting the parse error
112  // which caused the problem.  Unfortunately, to do the resynchronisation
113  // it requires access to protected functions of ANTLRParser.
114  // The class ANTLR_Assisted_Parser below is a hack to enable ParseUtility
115  // to violate the protection of the functions required: ParseUtility is
116  // passed a pointer to a real parser which is of a subclass of ANTLRParser,
117  // and casts it to this artificial subclass, so as to give ParseUtility
118  // friend rights and to access the protected functions.
119  // This hack is necessary because the class definition we need to affect
120  // is generated by PCCTS: I am not proud of it and if anyone can suggest
121  // a way of doing without modifying PCCTS or its support code, I will be
122  // very happy to hear about it.
123  class ANTLR_Assisted_Parser : public ANTLRParser
124  {
125    ANTLR_Assisted_Parser(ANTLRParser& parser) : ANTLRParser(parser) {}
126    friend class ParseUtility;
127  };
128
129  // The parse utility class is intended to assist the parser in a number
130  // of ways.  In earlier versions, this class had at least two distinct
131  // roles:
132  // 1) as a place for common functions which each parser might call
133  //    for diagnostics, resynchronisation etc; and
134  // 2) as a general storage area for state which needs to be remembered
135  //    for any length of time during the parsing process.
136  // The class ParseStore has been added to support the second role,
137  // and it is hoped that the amount of stored state can be reduced
138  // in the near future.
139  class ParseUtility {
```

```
140
141  public:
142   ParseUtility(ANTLRParser *parser);
143   ~ParseUtility();
144
145   // the following methods are used to service the standard tracein/traceout
146   // and syntax error reporting calls generated by PCCTS
147   void tracein(const char *rulename, int guessing, ANTLRAbstractToken *tok);
148   void traceout(const char *rulename, int guessing, ANTLRAbstractToken *tok);
149   void syn(_ANTLRTokenPtr tok, ANTLRChar *egroup, SetWordType *eset,
150            ANTLRTokenType etok, int k);
151
152   // this method consolidates the text of the next n tokens of lookahead
153   string lookahead_text(int n);
154
155   // this method searches for a string of tokens at the specified nesting
156   // depth from the specified token class, and uses them as a marker to
157   // resynchronise the parser
158   void resynchronize(
159                     int initial_nesting, SetWordType *resync_token_class,
160                     ANTLRTokenPtr& resync_token);
161
162   // This utility function is used to create
163   // a composite scope name from a qualifier scope
164   // and a relative name.
165   string scopeCombine(const string& baseScope, const string& name);
166
167   // Only one instance of this class should exist at any time.
168   // This method allows the parsers and lexers to access the instance.
169   static ParseUtility *currentInstance() { return theCurrentInstance; }
170
171  private:
172   static ParseUtility *theCurrentInstance;
173
174   ANTLR_Assisted_Parser *parser;
```

```
175     int trace_depth;
176     static int stack_depth;
177     static string   stack_tokentext[MAX_STACK_DEPTH];
178     static int           stack_tokenline[MAX_STACK_DEPTH];
179     static string   stack_rules[MAX_STACK_DEPTH];
180
181     // copy constructor and assignment operator are private to
182     // prevent unexpected copying
183     ParseUtility(const ParseUtility&);
184     const ParseUtility& operator=(const ParseUtility&);
185   };
186
187     // LOC, COM and MVG are all counted by the lexical analyzer,
188     // but the counts must be apportioned after the parser has
189     // identified the extents of the various declarations and definitions
190     // they belong to.
191     // This is achieved by the lexer maintaining counts of each
192     // which are reported to the ParseUtility class on a line by line
193     // basis.  ParseUtility uses this data to create a store which is
194     // used to apportion counts as the parser reports extents.
195     enum LexicalCount { tcCOMLINES, tcCODELINES, tcMCCABES_VG, tcSCM, tcLAST };
196
197
198   // The ParseStore class encapsulates all information storage
199   // requirements related to the parser, and also manages
200   // the process of feeding that information to the database
201   // when it is complete.
202   // In particular, the class is responsible for receiving and
203   // retaining counts of the lexical metrics (LOC, COM,
204   // MVG) on a line-by-line basis.  These are counted in the
205   // lexical analyzer, and the line-by-line counts must be
206   // integrated to allocate the counts to the extents identified
207   // by the parser as belonging to significant declarations and
208   // definitions.
209   class ParseStore
```

```
{
 public:
  ParseStore(const string& filename);
  ~ParseStore();

  void IncrementCount(LexicalCount lc)
  {
      pendingLexicalCounts[lc]++;
      if (tcMCCABES_VG == lc)
          {
          fprintf(stderr,"\nSCM: McCabe is incremented - incrementing SCM too.\n");
          pendingLexicalCounts[tcSCM]++;
          // remember the current MVG value
          m_CCNSpHelper++;
          }
  }

  int GetCCNspValue() { return m_CCNSpHelper;};

  void PerformReturn()
  {
      m_CCNSpHelper = 0;
  };

  void IncrementSCM(int IP, int CP, int CCN_sp)
  {
          long p = pow(IP, (CP)?CP-1:0);
      int useMcCabe = m_CCNSpHelper - CCN_sp + 1;
      // SCM follows McCabe (see IncrementCount in cccc.g), and here we multiple the branching
      // calculated so far by the Interleaving/Competition potentials' coefficent
          if (p>0)
          {
      pendingLexicalCounts[tcSCM] += useMcCabe * (long)p;
          }
  }
```

```
245
246     void endOfLine(int line);
247
248
249     // each of the functions below writes one or more records into
250     // the database of code
251     void record_module_extent(int startLine, int endLine,
252                               const string& moduleName,
253                               const string& moduleType,
254                               const string& description,
255                               UseType ut);
256     void record_function_extent(int startLine, int endLine,
257                                 const string& returnType,
258                                 const string& moduleName,
259                                 const string& memberName,
260                                 const string& paramList,
261                                 const string& description,
262                                 Visibility visibility,
263                                 UseType ut);
264     void record_userel_extent(int startLine, int endLine,
265                               const string& clientName,
266                               const string& memberName,
267                               const string& serverName,
268                               const string& description,
269                               Visibility visibility,
270                               UseType ut);
271     void record_other_extent(int startLine, int endLine,
272                                 const string& description);
273     void record_file_balance_extent(string);
274
275     // Each of the record_XXX methods above uses this function to
276     // add an extent record.
277     void insert_extent(CCCC_Item&, int, int,
278                        const string&, const string&,
279                        UseType, bool allocate_lexcounts);
```

```
280
281     // the class maintains a number of strings and flags which reflect
282     // the most recently recognized module, member, type (with and without
283     // indirection) etc, and the visibility of items occuring at the current
284     // context
285     int get_flag(PSFlag) const;
286     void set_flag(PSFlag,int);
287     void set_flag(Visibility);
288     Visibility get_visibility();
289     string filename();
290
291     char *flags() { return &(*flag.begin()); }
292
293     // We also need the automatically generated copy constructor
294     // and assignment operator to allow us to save state in the
295     // parser.
296
297     // Only one instance of this class should exist at any time.
298     // This method allows the parsers and lexers to access the instance.
299     static ParseStore *currentInstance() { return theCurrentInstance; }
300   private:
301     static ParseStore *theCurrentInstance;
302
303     int m_CCNSpHelper;
304
305     string theFilename;
306
307     typedef std::vector<int> LexicalCountArray;
308     LexicalCountArray pendingLexicalCounts;
309
310     typedef std::map<int,LexicalCountArray> LineLexicalCountMatrix;
311     LineLexicalCountMatrix lineLexicalCounts;
312
313     typedef std::vector<char> CharArray;
314     CharArray flag;
```

```
315
316     // copy constructor and assignment operator are private to
317     // prevent unexpected copying
318     ParseStore(const ParseStore&);
319     const ParseStore& operator=(const ParseStore&);
320   };
321
322   #endif
```

The ANTLRToken Class

cccc_tok.cc

The listing below of the whole cccc_tok.cc file, which includes the ANTLRToken class implementation. The changes relevant to this work are between lines 161 and 167, and are marked with the yellow background.

```
1   /*
2       CCCC - C and C++ Code Counter
3       Copyright (C) 1994-2005 Tim Littlefair (tim_littlefair@hotmail.com)
4
5       This program is free software; you can redistribute it and/or modify
6       it under the terms of the GNU General Public License as published by
7       the Free Software Foundation; either version 2 of the License, or
8       (at your option) any later version.
9
10      This program is distributed in the hope that it will be useful,
11      but WITHOUT ANY WARRANTY; without even the implied warranty of
12      MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
13      GNU General Public License for more details.
14
15      You should have received a copy of the GNU General Public License
16      along with this program; if not, write to the Free Software
17      Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301 USA
18  */
```

```
19  /*
20   * cccc_tok.C
21   * implementation of a token class for the cccc project
22   *
23   */
24
25  #include "cccc.h"
26  #include "cccc_tok.h"
27  #include "cccc_utl.h"
28  #include "cccc_scm.h"
29
30  /* static variables */
31  int ANTLRToken::RunningNesting=0;
32  int ANTLRToken::bCodeLine=0;
33  int ANTLRToken::numAllocated=0;
34  int toks_alloc1=0, toks_alloc2=0, toks_alloc3=0, toks_freed=0;
35
36  ANTLRToken currentLexerToken;
37
38  /*
39  ** Token objects are used to count the occurences of states which
40  ** our analyser is interested in within the text.  Any metric which
41  ** can be reduced to lexical counting on the text can be recorded
42  ** this way.
43  **
44  ** This implementation counts the following features:
45  **   tokens
46  **   comment lines
47  **   lines containing at least one token of code
48  **
49  ** It also makes a lexical count for the following tokens, each of which
50  ** is expected to increase McCabe's cyclomatic complexity (Vg) for the
51  ** section of code by one unit:
52  **  IF FOR WHILE SWITCH BREAK RETURN ? && ||
53  **
```

```
54  ** Note that && and || create additional paths through the code due to C/C++
55  ** short circuit evaluation of logical expressions.
56  **
57  ** Also note the way SWITCH constructs are counted: the desired increment
58  ** in Vg is equal to the number of cases provided for, including the
59  ** default case, whether or not an action is defined for it.  This is acheived
60  ** by counting the SWITCH at the head of the construct as a surrogate for
61  ** the default case, and counting BREAKs as surrogates for the individual
62  ** cases.  This approach yields the correct results provided that the
63  ** coding style in use ensures the use of BREAK after all non-default
64  ** cases, and forbids 'drop through' from one case to another other than
65  ** in the case where two or more values of the switch variable require
66  ** identical actions, and no executable code is defined between the
67  ** case gates (as in the switch statement in ANTLRToken::CountToken() below).
68  */
69
70  /* default constructor */
71  ANTLRToken::ANTLRToken() : ANTLRCommonToken() {
72    toks_alloc1++;
73    CurrentNesting=-99;
74  }
75
76  /*
77  ** constructor used by makeToken below
78  */
79  ANTLRToken::ANTLRToken(ANTLRTokenType t, ANTLRChar *s) :
80    ANTLRCommonToken(t,s) {
81    setType(t);
82    setText(s);
83    CountToken();
84
85    toks_alloc2++;
86  }
87
88  /* copy constructor */
```

```
89  ANTLRToken::ANTLRToken(ANTLRToken& copyTok) {
90    setType(copyTok.getType());
91    setText(copyTok.getText());
92    setLine(copyTok.getLine());
93    CurrentNesting=copyTok.CurrentNesting;
94    toks_alloc3++;
95  }
96
97  /*
98  ** the virtual pseudo-constructor
99  ** This is required because the PCCTS support code does not know the
100 ** exact nature of the token which will be created by the user's code,
101 ** and indeed does not forbid the user creating more than one kind of
102 ** token, so long as ANTLRToken is defined and all token classes are
103 ** subclassed from ANTLRAbstractToken
104 */
105 ANTLRAbstractToken *ANTLRToken::makeToken(
106                                         ANTLRTokenType tt, ANTLRChar *txt, int line
107                                         ) {
108
109   ANTLRToken *new_t = new ANTLRToken(tt,txt);
110   if(new_t==0) {
111     cerr << "Memory overflow in "
112       "ANTLRToken::makeToken(" << static_cast<int>(tt) << ","
113         << txt << "," << line << ")" << endl;
114     exit(2);
115   }
116   new_t->setLine(line);
117
118   DbgMsg(
119         LEXER,cerr,
120         "makeToken(tt=>" << static_cast<int>(tt) <<
121         ", txt=>" << txt <<
122         ",line=>" << line <<
123         ")" << endl
```

```
124           );
125
126     return new_t;
127   }
128
129   /* the destructor */
130   ANTLRToken::~ANTLRToken() {
131     toks_freed++;
132     DbgMsg(MEMORY,cerr,"freeing token " << getText()
133           << " on line " << getLine()
134           << " c1:" << toks_alloc1 << " c2:" << toks_alloc2
135           << " c3:" << toks_alloc3 << " freed:" << toks_freed << endl);
136   }
137
138   /* the assignment operator */
139   ANTLRToken& ANTLRToken::operator=(ANTLRToken& copyTok) {
140     setType(copyTok.getType());
141     setText(copyTok.getText());
142     setLine(copyTok.getLine());
143     CurrentNesting=copyTok.CurrentNesting;
144     return *this;
145   }
146
147   /*
148   ** ANTLRToken::CountToken performs counting of features which are traced
149   ** back to individual tokens created up by the lexer, i.e. the token count
150   ** and SCM values.  Code lines and comment lines are both identified during
151   ** the processing of text which the lexer will (usually) skip, so the code
152   ** to increment these counts is in the relevant lexer rules in the file
153   ** cccc.g, and so is the code relevant for the McCabe's VG.
154   */
155   void ANTLRToken::CountToken()
156   {
157     // we have seen a non-skippable pattern => this line counts toward LOC
158     bCodeLine=1;
```

```
159     int type =1;
160     CurrentNesting=RunningNesting;
161     // see the exact token and decide wether to count it towards the SCM value counter
162     CCCC_ScmManager::ConsumeToken(this);
163     if (! RunningNesting)
164         {
165         // we're out of the latest function scope
166         CCCC_ScmManager::CalculateScm();
167         }
168     DbgMsg(COUNTER,cerr,*this);
169   }
170
171   char *ANTLRToken::getTokenTypeName() { return ""; }
172
173   /*
174   ** structured output method for token objects
175   */
176   ostream& operator << (ostream& out, ANTLRToken& t) {
177     int i;
178
179     out << "TOK: " << t.getTokenTypeName()
180         << " " << t.getText()
181         << " " << t.getLine()
182         << " " << t.getNestingLevel();
183
184     out << endl;
185     return out;
186   }
```

Appendix C

In this appendix there are the full results provided by the CCCC for the modules analyzed in chapter 5.

BusyBox – Old Version Analysis Results

Detailed report on module anonymous

Metric	Tag	Overall	Per Function
Lines of Code	LOC	*1135*	******
McCabe's Cyclomatic Number	MVG	*374*	******
The SCM Value	SCM	1416	******
Lines of Comment	COM	406	********
LOC/COM	L_C	2.796	
MVG/COM	M_C	0.921	
SCM/COM	SCM_C	3.488	
Weighted Methods per Class (weighting = unity)	WMC1	22	
Weighted Methods per Class (weighting = visible)	WMCv	22	
Depth of Inheritance Tree	DIT	0	
Number of Children	NOC	0	
Coupling between objects	CBO	0	
Information Flow measure (inclusive)	IF4	0	********

Information Flow measure (visible)	IF4v	0	********
Information Flow measure (concrete)	IF4c	0	********

Definitions and Declarations

Description	LOC	MVG	SCM	COM	L_C	M_C	SCM_C
No module extents have been identified for this module							

Functions

Function prototype	LOC	MVG	SCM	COM	L_C	M_C	SCM_C
checkPerm(const char *, const char *) definition busybox_httpd.c:1389	*51*	*22*	22	27	1.889	0.815	0.815
checkPermIP(void) definition busybox_httpd.c:1342	20	4	4	3	6.667	------	------
decodeBase64(char *) definition busybox_httpd.c:758	*32*	*11*	11	19	1.684	0.579	0.579
decodeString(char *, int) definition busybox_httpd.c:681	*33*	*13*	13	23	1.435	0.565	0.565
encodeString(const char *) definition busybox_httpd.c:643	13	3	3	20	------	------	------

free_config_lines(Htaccess **) definition busybox_httpd.c:342	10	1	1	0	------	------	------
getLine(void) definition busybox_httpd.c:911	16	8	24	12	------	0.667	2.000
handleIncoming(void) definition busybox_httpd.c:1479	**222**	**78**	**466**	51	4.353	1.529	9.137
handle_sigalrm(int) definition busybox_httpd.c:1466	5	0	2	8	------	------	------
httpd_main(int, char **) declaration busybox_httpd.c:1933 definition busybox_httpd.c:1934	*78*	*19*	39	4	*19.500*	4.750	9.750
miniHttpd(int) definition busybox_httpd.c:1799	*45*	10	53	23	1.957	0.435	2.304
miniHttpd_inetd(void) definition busybox_httpd.c:1865	23	3	3	2	*11.500*	------	------
openServer(void) definition busybox_httpd.c:808	7	1	9	12	------	------	0.750
parse_conf(const char *, int)	**171**	**64**	64	53	3.226	1.208	1.208

definition busybox_httpd.c:386							
scan_ip(const char **, int *, char) definition busybox_httpd.c:267	*34*	*20*	20	0	******	******	******
scan_ip_mask(const char *, int *, int *) definition busybox_httpd.c:304	*34*	*14*	14	0	******	******	******
sendCgi(const char *, const char *, int, const char *, const char *) definition busybox_httpd.c:951	**217**	**70**	**607**	109	1.991	0.642	5.569
sendFile(const char *) definition busybox_httpd.c:1285	*45*	*18*	38	17	2.647	1.059	2.235
sendHeaders(HttpResponseNum) definition busybox_httpd.c:833	*58*	*13*	15	19	3.053	0.684	0.789
setenv1(const char *, const char *) definition busybox_httpd.c:725	6	1	1	3	------	------	------
setenv_long(const char *, long) definition busybox_httpd.c:731	6	0	0	0	------	------	------
sighup_handler(int) definition busybox_httpd.c:1896	9	1	7	1	------	------	7.000

Relationships

Clients	Suppliers

BusyBox – New Version Analysis Results

Detailed report on module anonymous

Metric	Tag	Overall	Per Function
Lines of Code	LOC	*1118*	******
McCabe's Cyclomatic Number	MVG	*326*	******
The SCM Value	SCM	615	******
Lines of Comment	COM	374	********
LOC/COM	L_C	2.989	
MVG/COM	M_C	0.872	
SCM/COM	SCM_C	1.644	
Weighted Methods per Class (weighting = unity)	WMC1	28	
Weighted Methods per Class (weighting = visible)	WMCv	28	
Depth of Inheritance Tree	DIT	0	
Number of Children	NOC	0	
Coupling between objects	CBO	0	
Information Flow measure (inclusive)	IF4	0	********
Information Flow measure (visible)	IF4v	0	********
Information Flow measure (concrete)	IF4c	0	********

Definitions and Declarations

Description	LOC	MVG	SCM	COM	L_C	M_C	SCM_C
No module extents have been identified for this module							

Functions

Function prototype	LOC	MVG	SCM	COM	L_C	M_C	SCM_C
checkPermIP(void) definition httpd.c:1627	21	4	4	2	*10.500*	------	------
check_user_passwd(const char *, const char *) definition httpd.c:1663	*52*	*19*	19	23	2.261	0.826	0.826
decodeBase64(char *) definition httpd.c:863	*32*	*11*	11	9	3.556	1.222	1.222
decodeString(char *, int) definition httpd.c:813	*35*	*13*	13	8	4.375	1.625	1.625
encodeString(const char *) definition httpd.c:761	15	3	3	11	------	------	------
find_proxy_entry(const char *) definition httpd.c:1737	9	4	4	1	------	------	------

free_llist(has_next_ptr **) definition httpd.c:340	10	1	1	0	------	------	------
get_line(void) definition httpd.c:1063	25	9	25	8	3.125	1.125	3.125
handle_incoming_and_exit(const len_and_sockaddr *) declaration httpd.c:1760 definition httpd.c:1761	**267**	**94**	*288*	69	3.870	1.362	4.174
hex_to_bin(char) definition httpd.c:794	11	5	5	15	------	0.333	0.333
httpd_main(...) definition httpd.c:2284	*75*	*16*	38	21	3.571	0.762	1.810
httpd_main(int, char **) declaration httpd.c:2283	1	0	0	0	------	------	------
log_and_exit(void) declaration httpd.c:918 definition httpd.c:919	8	1	5	11	------	------	0.455
mini_httpd(int) declaration httpd.c:2154 definition httpd.c:2155	21	3	12	18	1.167	------	0.667

mini_httpd_inetd(void) declaration httpd.c:2240 definition httpd.c:2241	9	0	0	5	------	------	------
mini_httpd_nommu(int, int, char **) declaration httpd.c:2191 definition httpd.c:2192	25	3	11	13	1.923	------	0.846
openServer(void) definition httpd.c:904	10	4	16	3	------	------	5.333
parse_conf(const char *, int) definition httpd.c:464	**195**	**54**	54	64	3.047	0.844	0.844
scan_ip(const char **, unsigned *, char) definition httpd.c:363	*37*	*21*	21	2	*18.500*	**10.500**	10.500
scan_ip_mask(const char *, unsigned *, unsigned *) definition httpd.c:405	27	*12*	12	10	2.700	1.200	1.200
send_REQUEST_TIMEOUT_and_exit(...) definition httpd.c:1752	4	0	0	0	------	------	------
send_REQUEST_TIMEOUT_and_exit(int) declaration httpd.c:1751	1	0	0	3	------	------	------
send_cgi_and_exit(const char *, const char *, int, const char *,	**130**	*28*	30	68	1.912	0.412	0.441

const char *) declaration httpd.c:1287 definition httpd.c:1293							
send_file_and_exit(const char *, int) declaration httpd.c:338	1	0	0	0	------	------	------
send_headers(int) definition httpd.c:943	*83*	*20*	42	10	*8.300*	2.000	4.200
send_headers_and_exit(int) declaration httpd.c:1049 definition httpd.c:1050	6	0	0	0	------	------	------
setenv1(const char *, const char *) definition httpd.c:1269	4	1	1	0	------	------	------
sighup_handler(...) definition httpd.c:2252	4	0	0	0	------	------	------

Relationships

Clients	Suppliers

IKI Analysis Results

Detailed report on module anonymous

Metric	Tag	Overall	Per Function
Lines of Code	LOC	*1376*	******
McCabe's Cyclomatic Number	MVG	*312*	******
The SCM Value	SCM	487	******
Lines of Comment	COM	103	********
LOC/COM	L_C	*13.359*	
MVG/COM	M_C	3.029	
SCM/COM	SCM_C	4.728	
Weighted Methods per Class (weighting = unity)	WMC1	30	
Weighted Methods per Class (weighting = visible)	WMCv	30	
Depth of Inheritance Tree	DIT	0	
Number of Children	NOC	0	
Coupling between objects	CBO	0	
Information Flow measure (inclusive)	IF4	0	********
Information Flow measure (visible)	IF4v	0	********
Information Flow measure (concrete)	IF4c	0	********

Definitions and Declarations

Description	LOC	MVG	SCM	COM	L_C	M_C	SCM_C
No module extents have been identified for this module							

Functions

Function prototype	LOC	MVG	SCM	COM	L_C	M_C	SCM_C
add_forwarding_page(char *, char *) definition iki_httpd.c:581	*34*	5	5	0	******	******	******
add_redir_page(char *, char *) definition iki_httpd.c:536	*39*	4	4	3	*13.000*	------	------
cleanup_temp_cache( ) definition iki_httpd.c:1405	*31*	6	6	3	*10.333*	2.000	2.000
close_connection(int) definition iki_httpd.c:969	*37*	2	2	3	*12.333*	------	------
compr_redirection(const void *, const void *) definition iki_httpd.c:491	6	1	1	3	------	------	------
compr_url(const void *, const void *) definition iki_httpd.c:481	6	1	1	3	------	------	------

decode_url_in_place(char *) definition iki_httpd.c:1159	22	6	6	3	*7.333*	2.000	2.000
do_get(int, char *, command_t) definition iki_httpd.c:1187	**200**	**31**	31	3	**66.667**	**10.333**	10.333
do_write(int) definition iki_httpd.c:928	*35*	10	22	3	*11.667*	3.333	7.333
dprint(...) definition iki_httpd.c:348	9	0	0	3	------	------	------
find_free_temp_page( ) definition iki_httpd.c:1085	22	7	7	3	*7.333*	2.333	2.333
http_server( ) definition iki_httpd.c:1728	**130**	*29*	75	4	**32.500**	*7.250*	18.750
main(int, char **) definition iki_httpd.c:1872	*98*	*25*	90	1	**98.000**	**25.000**	90.000
make_redirection_page(redirection_t, char *) definition iki_httpd.c:1112	*38*	4	4	3	*12.667*	------	------
match_type(char *) definition iki_httpd.c:665	*58*	**53**	53	3	*19.333*	**17.667**	17.667

new_connection( ) definition iki_httpd.c:1441	*84*	*12*	34	3	*28.000*	4.000	11.333
open_service(const char *) definition iki_httpd.c:362	*59*	7	13	3	*19.667*	2.333	4.333
parse_command(int) definition iki_httpd.c:1569	**117**	*25*	25	3	**39.000**	*8.333*	8.333
parse_headers(int) definition iki_httpd.c:1700	21	*16*	16	3	7.000	*5.333*	5.333
read_data(int) definition iki_httpd.c:1532	*32*	8	26	3	*10.667*	2.667	8.667
read_htdocs(char *) definition iki_httpd.c:791	*85*	*18*	18	11	*7.727*	1.636	1.636
read_page(page_t *, char *, stat *, char *) definition iki_httpd.c:729	*52*	9	15	4	*13.000*	2.250	3.750
read_pages(char *, char *) definition iki_httpd.c:899	22	3	3	6	3.667	------	------
read_redirections(char *) definition iki_httpd.c:622	*36*	8	8	3	*12.000*	2.667	2.667

return_error(int) definition iki_httpd.c:1022	12	0	0	3	------	------	------
rfc1123date(char *, time_t) definition iki_httpd.c:439	10	1	1	3	------	------	------
set_head_and_body(page_t, char *, long, char *, long) definition iki_httpd.c:502	25	7	7	4	6.250	1.750	1.750
set_out_data(connection_t, char *, size_t, size_t, command_t) definition iki_httpd.c:1040	*40*	5	5	5	*8.000*	1.000	1.000
skip_non_white(char *) definition iki_httpd.c:468	8	5	5	4	------	1.250	1.250
skip_white(char *) definition iki_httpd.c:454	8	4	4	4	------	------	------

Relationships

Clients	Suppliers

תקציר

ככל שסביבות ומערכות מרובות תהליכים נהיות יותר ויותר נפוצות, איכות ויעילות של המערכות הללו נהיית יותר ויותר חשובה.
בזמן שעבור מערכות סדרתיות ישנן מדדי סיבוכיות אשר פותחו עוד בשנות ה70 של המאה הקודמת, עד עתה לא הייתה דרך לבצע מדידה והשוואת סיבוכיות תוכנות מרובות תהליכים בצורה אמינה ויעילה, ולאמוד את השפעת היותה של המערכת מקבילית על האיכות הכוללת שלה.

בעבודה זו אנו מציעים פתרון לבעיה הזאת. בפרט, אנחנו מציעים מדד חדש, אשר יאפיין סיבוכיות תוכנה על בסיס סוג וכמות אמצעי הסנכרון אשר משמשים לשם תיאום בין רכיביה המקביליים השונים. בדומה למדד הסיבוכיות הציקלומטית של מקקייב עבור המערכות הסדרתיות, מדד סיבוכיות סנכרון החדש מאפשר לאמוד את מספר הבדיקות הנדרשות לכיסוי הולם בבדיקות של מערכת מקבילית. בנוסף, המדד מאפשר השוואה בין מימושים שונים של מערכת בהתבסס על ניתוח סיבוכיות הסנכרון.

האוניברסיטה הפתוחה
המחלקה למתמטיקה ולמדעי המחשב

מדד לסיבוכיות סנכרון

עבודת תזה זו הוגשה כחלק מהדרישות לקבלת תואר
"מוסמך במדעים" .M.Sc במדעי המחשב
באוניברסיטה הפתוחה
החטיבה למדעי המחשב

על-ידי
פטר יסטרבנצקי

העבודה הוכנה בהדרכתו של ד"ר מרק טרכטנברוט

דצמבר 2009