

**Masters of Computer Science**  
**Project Report**

**DeMemory**  
**Simulation Tool for Multi-Level Memory System**  
**using Aging Algorithm**

Project Supervisor: Dr. Leonid Barenboim

**Gal Oren**

ID: 203127261

Mobile: 050-623-9129

[Galoren.com@gmail.com](mailto:Galoren.com@gmail.com)

Yigal Alon 164/8, Tel Aviv

Masters of Computer Science Project Report

Department of Computer Science

The Open University of Israel

September 2015

## **Table of Contents**

### **Background**

Basic Paging Mechanism	3
Paging in Other Computer Science Fields	3
Paging Algorithms Evolution	4

### **The Main Evolution (1960 - 1970)**

The Best Possible Page Replacement Algorithm	5
The Not Recently Used (NRU) Page Replacement Algorithm	6
The First-In-First-Out (FIFO) with Second-Chance and Aging Page Replacement Algorithm	7
The Least Recently Used (LRU) and NFU (Not Frequently Used) Page Replacement Algorithm	9

### **The Recent Evolution (2000 - Present)**

The Adaptive Replacement Cache (ARC) Page Replacement Algorithm	11
The CLOCK with Adaptive Replacement (CAR) Page Replacement Algorithm	13
The Token-ordered LRU Page Replacement Algorithm	14
The CLOCK-Pro Page Replacement Algorithm	15

<b>Summery</b>	18
----------------	----

<b>References</b>	21
-------------------	----

<b>Acknowledgements</b>	22
-------------------------	----

## Background

### Basic Paging Mechanism

A page fault is a type of interrupt, called trap, raised by the computer hardware when a running program accesses a memory page that is mapped into the virtual address space, but not actually loaded into main memory. It is well known [1] [2] [3] that when a page fault occurs, the operating system has to choose a page to remove from memory to make room for the page that has to be brought in. If the page to be removed has been modified while in memory, it must be rewritten to the disk to bring the disk copy up to date. If, however, the page has not been changed, the disk copy is already up to date, so no rewrite is needed. The page to be read in just overwrites the page being evicted. While it would be possible to pick a random page to evict at each page fault, system performance would be much better if a page that is not heavily used is chosen. If a heavily used page is removed, it will probably have to be brought back in quickly, resulting in extra overhead.

### The LRU and NFU Page Replacement Algorithms

A good approximation to the optimal algorithm is based on the observation that pages that have been heavily used in the last few instructions will probably be heavily used again in the next few. Conversely, pages that have not been used for ages will probably remain unused for a long time. This idea suggests a realizable algorithm: when a page fault occurs, throw out the page that has been unused for the longest time. This strategy is called **LRU (Least Recently Used) Paging**.

Although LRU is theoretically realizable, it is not cheap. To fully implement LRU, it is necessary to maintain a linked list of all pages in memory, with the most recently used page at the front and the least recently used page at the rear. The difficulty is that the list must be updated on every memory reference. Finding a page in the list, deleting it, and then moving it to the front

is a very time consuming operation, even in hardware (assuming that such hardware could be built).

However, there are other ways to implement LRU with special hardware:

1. The simplest method requires equipping the hardware with a counter that is automatically incremented after each instruction. Furthermore, each page table entry must also have a field large enough to contain the counter. After each memory reference, the current value of the counter is stored in the page table entry for the page just referenced. When a page fault occurs, the operating system examines all the counters in the page table to find the lowest one. That page is the least recently used.
2. The more sophisticated and known method suggest that for a machine with  $n$  page frames, the LRU hardware can maintain a matrix of  $n \times n$  bits, initially all zero. Whenever page frame  $k$  is referenced, the hardware first sets all the bits of row  $k$  to 1, then sets all the bits of column  $k$  to 0. At any instant, the row whose binary value is lowest is the least recently used, the row whose value is next lowest is next least recently used, and so forth.

Although both of the previous two LRU algorithms are realizable in principle, few, if any, machines have this hardware, so they are of little use to the operating system designer who is making a system for a machine that does not have this hardware. Instead, a solution that can be implemented in software is needed. One possibility is called the **NFU (Not Frequently Used)** algorithm. It requires a software counter associated with each page, initially zero. At each clock interrupt, the operating system scans all the pages in memory. For each page, the  $R$  bit, which is 0 or 1, is added to the counter. In effect, the counters are an attempt to keep track of how often each page has been referenced. When a page fault occurs, the page with the lowest counter is chosen for replacement.

The main problem with NFU is that it never forgets anything, which consequently can cause the operating system to remove useful pages instead of pages no longer in use. Fortunately, a small modification to NFU makes it able to simulate LRU quite well. The modification has two parts. First, the counters are each shifted right 1 bit before the  $R$  bit is added in. Second, the  $R$  bit is added to the leftmost, rather than the rightmost bit. This modified version of the algorithm known as **Aging** algorithm.

### **The Aging Page Replacement Algorithm**

The Aging algorithm is a descendant of the NFU algorithm, with modifications to make it aware of the time span of use. Instead of just incrementing the counters of pages referenced, putting equal emphasis on page references regardless of the time, the modification has two parts: First, the counters are each shifted right 1 bit before the  $R$  bit is added in. Second, the  $R$  bit is added to the leftmost, rather than the rightmost bit.

For instance, if a page has referenced bits 1,0,0,1,1,0 in the past 6 clock ticks, its referenced counter will look like this: 10000000, 01000000, 00100000, 10010000, 11001000, 01100100. Hence, page references closer to the present time have more impact than page references long ago. This ensures that pages referenced more recently, though less frequently referenced, will have higher priority over pages more frequently referenced in the past. Thus, when a page needs to be swapped out, the page with the lowest counter will be chosen.

### **The N-Level Aging Page Replacement Algorithm**

The transition of the Aging algorithm to an N-level memory hierarchy model can even add another level of sophistication and optimization, especially because of the existence of a linear proportion between the degradation of the

referenced bits and the amount of time that a specific page was not in use. In this hypothesis, unlike in the other N-level memory hierarchies Paging algorithms adaptations and adjustments shown before, there is a possibility to create a direct link between the amount of zeros at the beginning of the page referenced bits to the level of memory that that page should be evicted to. Based on the knowledge that the amount of zeros points on the amount of unreferenced past clock ticks - and therefore on the page aging status - it would be wise to evict the page straight to its proportionate level of memory. Hence, by forming a dynamic pyramid hierarchy of both page and memory necessity it will be achievable to get the best performances for a Paging algorithm in an N-level memory hierarchy.

Formulation of the Aging algorithm for N-level memory hierarchy:

- **Insertion** of a new page:
  - *Set* memory levels to N ( $ML = N$ ).
  - *Set* memory level to the highest ( $L = 1$ ).
  - **Call** to the page.
  - *If* the page exists:
    - *Check* if placing in the L-level of memory is possible.
    - *If* placement possible:
      - *Place* page at the L-level of memory.
      - *Return* True.
    - *Else If* placement impossible:
      - *Find* the page with the lowest referenced counter:
        - **Remove** page with the lowest referenced counter.
        - **Do Insertion** of the page with the lowest referenced counter to the proportionate level of memory based on the amount of the zeros at the beginning of the page referenced bits. ( $L = \lfloor \frac{ZEROS\_BEGINING}{\text{page referenced bits}} \rfloor + 1$ )

(ALL\_REFERENCED\_BITS / ML) ||  
\*upwards\*

- *Place* the page at the L-level of memory.
  - *Return* True.
  - *Else If* page is not exists:
    - *Return* False.
- **Call** to a page:
  - *Calculate* the addressing of the page in the N-levels of memory.
  - *If* page found:
    - *Return* Real Addressing.
  - *Else If* page was not found:
    - *Return* NULL.
- **Remove** of a specific page:
  - *Store* the page in a temporary storage.
  - *Free* the addressing of the page.
  - *Return* the page form the temporary storage.
- **Update** of an exsiting page (*by the OS*):
  - *If* Read/Write action performed on the page:
    - *Set* R bit to 1 (R = 1).
  - *If* clock interrupt:
    - *Right Shift* 1 bit to all of the pages counters.
    - *Add* the R bit to the leftmost bit of all of the pages counters.

## The DeMemory Package

### File System

The DeMemory simulation contains multiply sections which requires different files and directories to operate. As shown in Fig. 1, all of those files and directories placed in the dememory package.

```
galoren@h-MacBook-Pro-sl-Gal:~/Desktop/THESIS/dememory$ ls -lah
total 1624
drwxr-xr-x@ 24 galoren  staff  816B 12:26 24  נוב .
drwxr-xr-x  5 galoren  staff  170B 15:37 21  נוב ..
-rw-r--r--@  1 galoren  staff   8.0K 11:23 24  נוב .DS_Store
drwxr-xr-x  13 galoren  staff  442B 12:26 24  נוב .git
-rwxr-xr-x@  1 galoren  staff    8B 2015  16  י מנז .gitignore
drwxr-xr-x  9 galoren  staff  306B 12:26 24  נוב .idea
-rw-r--r--  1 galoren  staff  114B 23:09 23  נוב ~/.lock.DMemory.doc
-rw-r--r--@  1 galoren  staff  351K 23:09 23  נוב DeMemory.doc
-rw-r--r--  1 galoren  staff  102K 10:40 24  נוב Doxyfile
-rw-r--r--  1 galoren  staff  102K 22:56 23  נוב Doxyfile.bak
-rwxr-xr-x@  1 galoren  staff   1.0K 15:35 21  נוב LICENSE
-rw-r--r--  1 galoren  staff   40B 18:28 23  נוב README.md
drwxr-xr-x  3 galoren  staff  102B 10:40 24  נוב def
-rwxr-xr-x  1 galoren  staff   33K 18:30 23  נוב dememory
-rwxr-xr-x  1 galoren  staff   61K 18:21 23  נוב dememory.c
-rw-r--r--  1 galoren  staff   9.4K 18:30 23  נוב dememory.log
drwxr-xr-x  11 galoren  staff  374B 10:40 24  נוב docbook
-rw-r--r--  1 galoren  staff  102K 10:37 24  נוב doxygen.file
drwxr-xr-x  53 galoren  staff   1.8K 10:40 24  נוב html
drwxr-xr-x  10 galoren  staff  340B 10:40 24  נוב latex
-rwxr-xr-x@  1 galoren  staff  308B 15:36 21  נוב makefile
drwxr-xr-x  3 galoren  staff  102B 10:40 24  נוב man
drwxr-xr-x  3 galoren  staff  102B 10:40 24  נוב rtf
-rwxr-xr-x  1 galoren  staff  769B 22:26 21  נוב runner.py
```

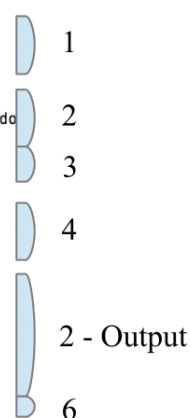


Fig. 1: The DeMemory file system

The usage of the files and directories is based on six different groups of files and directories, and it is differentiate as following (base on the six grouping forms in Fig 1.):

- 1) **GIT**: Git (git-scm.com) is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows. The DeMemory package is communicating via this Git repository to an online secure repository named BitBucket (bitbucket.org). In this way there is an option to securely locate the package online and also share it with the scientific community (Fig. 2).



- 2) Doxygen: Doxygen (doxygen.org) is the de facto standard tool for generating documentation from annotated C++ sources, but it also supports other popular programming languages such as C. Doxygen can generate an on-line documentation browser (in HTML) and/or an off-line reference manual (in LaTeX) from a set of documented source files. There is also support for generating output in RTF (MS-Word), PostScript, hyperlinked PDF, compressed HTML, and Unix man pages (as shown in section 2.1 in Fig. 1). The documentation is extracted directly from the sources, which makes it much easier to keep the documentation consistent with the source code. The DeMemory simulation code is completely documented with Doxygen, and the outcome of this documentation is presented in the code section of this file.
- 3) License: Includes the license usage of this package and the copyright.
- 4) Code: The DeMemory simulation code and binaries, including the log file of the simulation, which contains the summery results of the simulation, and its permanent part of the code because its always appended and not erased.
- 5) Runner: A Python script which runs the DeMemory simulation with different arguments for multiply times. Good for benchmarking mainly.

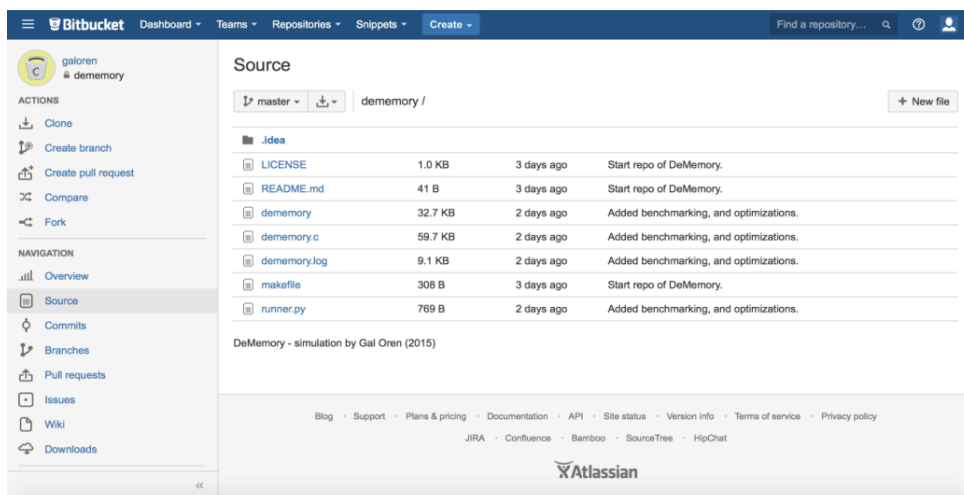


Fig. 2: The DeMemory file system on Bitbucket repository

## Usage

In Linux/UNIX systems, under GCC compiler, and using the Makefile platform, the usage will be in the following form:

```
>> make clean
```

```
>> make
```

```
>> ./dememory [algorithm] [num_frames] [show_process]
```

```
                [debug] [ref] [page_calls]
```

algorithm        - page algorithm to use {A = Aging, B = N-Level Aging}

num\_frames      - number of page frames {int > 0}

show\_process    - print page table after each ref is processed {1 or 0}

debug           - verbose debugging output {1 or 0}

ref             - page ref upper bound {ALL}

page\_calls      - max\_page\_calls {int > 0}

Because of a heavy usage of the memory system, it is highly recommended to clean the Cache after each run of the simulation. Under Mac/Linux systems it is can be done quickly using the Purge command from the Terminal under Administrator permissions.

```
>> sudo purge
```

## Example of Usage

```
>> ./dememory B 10 1 0 100 1000
```

Run the simulation of the 3-Level Aging algorithm, with 10 memory frames, printing of the page table during the process, without showing debug verbose, using 100 different page references and repetition of the procedure for 1000 times.

## Analysis of Output

The output version as shown in Fig. 1 is based on a simulation of a 3 level memory hierarchy with 10 frames for each level, when there are 100 references and the simulations run for 5 times. In this case there is a complete printing - both debug and show process options were on. Fig. 3 presents only the last steps of the simulation.

```
>> ./dememory B 10 1 1 100 5
```

```
AGING_N Algorithm
LEVEL: [0] - Frames in Mem: 10, Refs to Mem: 100, Hits: 0, Misses: 4, Hit Ratio: 0.000000, [Max Page calls: 5]
Frame # :      0       1       2       3       4       5       6       7       8       9
Page Ref :      7      73      72      23      0      0      0      0      0      0
Extra    :   1250000  2500000  5000000      0      0      0      0      0      0
Time     :  48296258  48296258  48296258  48296258  48296258  48296258  48296258  48296258  48296258  48296258

Frame # :      0       1       2       3       4       5       6       7       8       9
Page Ref :      0      0      0      0      0      0      0      0      0      0
Extra    :      0      0      0      0      0      0      0      0      0      0
Time     :  48296258  48296258  48296258  48296258  48296258  48296258  48296258  48296258  48296258  48296258

Frame # :      0       1       2       3       4       5       6       7       8       9
Page Ref :      0      0      0      0      0      0      0      0      0      0
Extra    :      0      0      0      0      0      0      0      0      0      0
Time     :  48296258  48296258  48296258  48296258  48296258  48296258  48296258  48296258  48296258  48296258

>>>>>>>>>> Current Level: [0]

frame->index:[0], frame->extra:[625000], count_zeros_before:[4]
**** calculate_direct_level ****: 1
**** [00001000] ****
INDEX: [0]
INDEXES [0]:[0]
**** REMOVED : [1]****
INSERT-IN :: frame->index:[0], frame->page:[7]
INSERT :: frame->index:[0], frame->page:[7]
**** INSERT : [1]****
frame->index:[1], frame->extra:[1250000], count_zeros_before:[3]
frame->index:[2], frame->extra:[2500000], count_zeros_before:[2]
frame->index:[3], frame->extra:[5000000], count_zeros_before:[1]
>>>>>>>>>> Current Level: [1]

frame->index:[0], frame->extra:[625000], count_zeros_before:[5]
>>>>>>>>>> Current Level: [2]

AGING_N Algorithm
LEVEL: [0] - Frames in Mem: 10, Refs to Mem: 100, Hits: 0, Misses: 5, Hit Ratio: 0.000000, [Max Page calls: 5]
Frame # :      0       1       2       3       4       5       6       7       8       9
Page Ref :      7      73      72      23      65      0      0      0      0      0
Extra    :      0   1250000  2500000  5000000      0      0      0      0      0      0
Time     :      0  48296258  48296258  48296258  48296258  48296258  48296258  48296258  48296258  48296258

Frame # :      0       1       2       3       4       5       6       7       8       9
Page Ref :      7      0      0      0      0      0      0      0      0      0
Extra    :   625000      0      0      0      0      0      0      0      0      0
Time     :  48296258  48296258  48296258  48296258  48296258  48296258  48296258  48296258  48296258  48296258

Frame # :      0       1       2       3       4       5       6       7       8       9
Page Ref :      0      0      0      0      0      0      0      0      0      0
Extra    :      0      0      0      0      0      0      0      0      0      0
Time     :  48296258  48296258  48296258  48296258  48296258  48296258  48296258  48296258  48296258  48296258

AGING_N Algorithm
LEVEL: [0] - Frames in Mem: 10, Refs to Mem: 100, Hits: 0, Misses: 5, Hit Ratio: 0.000000, [Max Page calls: 5]
Elapsed: 0.000706 seconds
```

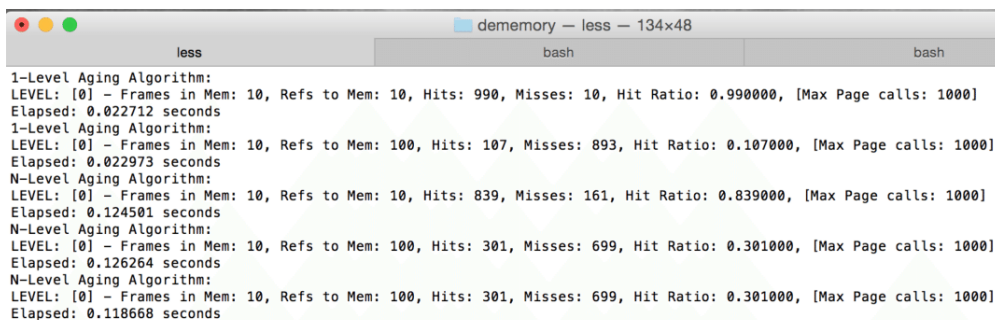
Fig. 3: Example of Output in a 3-Level Memory Hierarchy

As shown in Fig. 3, at the beginning the simulation prints the current status of the pages in all of the memory levels. After the N-level Aging algorithm was run on the first level of the memory (Current Level: [0]), it was discovered that the first page (index = 0) was not referenced for 4 cycles (extra = 625000, count\_zeros\_before = 4) and the algorithm calculated that it should be evicted to a lower level of memory. Therefore, the page is been successfully evicted from the first level of memory (level = 0 , REMOVED: [1]) and successfully inserted into the second level of memory (level = 1, INSERT: [1]).

Afterwards, the algorithm is checking the second and the third level of memory to figure if there is a need to upgrade or downgrade any of its files like its been done in the first level, but concludes there is nothing to do. At the end of the algorithm run the simulation re-print the status of the memory and there is an option to see that the actions were actually took place.

At the end of the output the program prints the status and the statistics of the simulation so far (and also when the simulation ends), including the number of hits and misses, the total hit ratio and the time elapsed ever since the programm started. Those statistics constantly append into a log file at the local directory (dememory.log, Fig. 4), and it is possible to examine it using:

```
>> less dememory.log
```



```

1-Level Aging Algorithm:
LEVEL: [0] - Frames in Mem: 10, Refs to Mem: 10, Hits: 990, Misses: 10, Hit Ratio: 0.990000, [Max Page calls: 1000]
Elapsed: 0.022712 seconds
1-Level Aging Algorithm:
LEVEL: [0] - Frames in Mem: 10, Refs to Mem: 100, Hits: 107, Misses: 893, Hit Ratio: 0.107000, [Max Page calls: 1000]
Elapsed: 0.022973 seconds
N-Level Aging Algorithm:
LEVEL: [0] - Frames in Mem: 10, Refs to Mem: 10, Hits: 839, Misses: 161, Hit Ratio: 0.839000, [Max Page calls: 1000]
Elapsed: 0.124501 seconds
N-Level Aging Algorithm:
LEVEL: [0] - Frames in Mem: 10, Refs to Mem: 100, Hits: 301, Misses: 699, Hit Ratio: 0.301000, [Max Page calls: 1000]
Elapsed: 0.126264 seconds
N-Level Aging Algorithm:
LEVEL: [0] - Frames in Mem: 10, Refs to Mem: 100, Hits: 301, Misses: 699, Hit Ratio: 0.301000, [Max Page calls: 1000]
Elapsed: 0.118668 seconds

```

Fig. 4: Example of Output to the log file dememory.log

When there is a need to follow several runs of the simulations it is possible to see the results appended in live streaming using:

```
>> tail -f dememory.log
```

## The DeMemory Code

The DeMemory code documentation is presented as processed directly from the code Doxygen comments, and it is available in LaTeX, rtf and HTML as well.

### dememory.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <math.h>
#include <sys/queue.h>
```

### Classes

struct	<a href="#">Frame</a>
struct	<a href="#">Algorithm_Data</a>
struct	<a href="#">Algorithm</a>

### Macros

#define	<a href="#">amount_of_memory_levels</a>	3
#define	<a href="#">LEVEL_A</a>	0
#define	<a href="#">LEVEL_B</a>	1
#define	<a href="#">LEVEL_C</a>	2

### Typedefs

typedef struct	<a href="#">Frame</a>	<a href="#">Frame</a>
----------------	-----------------------	-----------------------

### Functions

	<a href="#">LIST_HEAD</a> ( <a href="#">Page_Ref_List</a> , <a href="#">Page_Ref</a> )
int	<a href="#">init</a> ()
void	<a href="#">gen_page_refs</a> ()
<a href="#">Page_Ref</a> *	<a href="#">gen_ref</a> ()
<a href="#">Algorithm_Data</a> *	<a href="#">create_algo_data_store</a> ()
<a href="#">Frame</a> *	<a href="#">create_empty_frame</a> (int index)
int	<a href="#">cleanup</a> ()
int	<a href="#">event_loop</a> ()
int	<a href="#">page</a> (int page_ref)
int	<a href="#">get_ref</a> ()
int	<a href="#">add_victim</a> (struct <a href="#">Frame_List</a> *victim_list, struct <a href="#">Frame</a> *frame, int level)
int	<a href="#">print_help</a> (const char *binary)
int	<a href="#">print_list</a> (struct <a href="#">Frame</a> *head, const char *index_label, const char *value_label, int

level)
int <code>print_stats</code> ( <code>Algorithm</code> algo)
int <code>print_summary</code> ( <code>Algorithm</code> algo)
int <code>delete_frame_from_list</code> ( <code>Algorithm_Data</code> *data, int index, int level)
int <code>insert_frame_into_list</code> ( <code>Algorithm_Data</code> *data, struct <code>Frame</code> *framep_in, int level)
int <code>calculate_direct_level</code> (struct <code>Frame</code> *framep)
int <code>AGING</code> ( <code>Algorithm_Data</code> *data)
int <code>AGING_N</code> ( <code>Algorithm_Data</code> *data)
int <code>main</code> (int argc, char *argv[])

## Variables

<code>Page_Ref</code>
int <code>num_frames</code>
int <code>page_ref_upper_bound</code>
int <code>max_page_calls</code>
int <code>debug</code> = 0
int <code>printrefs</code> = 0
<code>Algorithm</code> <code>algos</code> [2]
int <code>counter</code> = 0
int <code>last_page_ref</code> = -1
int <code>last_page_ref2</code> = -1
size_t <code>num_algos</code> = 2
int * <code>optimum_find_test</code>
int <code>num_refs</code> = 0
FILE * <code>pFile</code>
int <code>print_count</code> = 0
double <code>time_spent_global</code> = 0

## Macro Definition Documentation

<code>#define amount_of_memory_levels 3</code>
<code>#define LEVEL_A 0</code>
<code>#define LEVEL_B 1</code>
<code>#define LEVEL_C 2</code>

## Typedef Documentation

```
typedef struct Frame Frame
```

## Function Documentation

```
int add_victim ( struct Frame_List * victim_list,  
                 struct Frame * frame,  
                 int level  
                )
```

int add\_victim(struct Frame\_List \*victim\_list, struct Frame \*frame)

Add victim frame evicted from page table to list of victims

### Parameters

**index** {Frame\_List} list of victims

**page** {Frame} page frame evicted

0

```
int AGING ( Algorithm_Data * data )
```

int AGING(Algorithm\_Data \*data)

AGING Page Replacement [Algorithm](#) for 1-Level memory hierarchy

### Parameters

**\*data** {Algorithm\_Data} struct holding algorithm data

return {int} did page fault, 0 or 1

```
int AGING_N ( Algorithm_Data * data )
```

int AGING\_N(Algorithm\_Data \*data)

AGING Page Replacement [Algorithm](#) for N-Level memory hierarchy

### Parameters

**\*data** {Algorithm\_Data} struct holding algorithm data

return {int} did page fault, 0 or 1



```
int calculate_direct_level ( struct Frame * framep )
```

```
int calculate_direct_level(struct Frame *framep)
```

Calculates the direct level to map the frame based on the N-Level Aging algorithm

**Parameters**

**page** {Frame} page frame to calculate

{int} the relative deviation level

```
int cleanup ( )
```

```
int cleanup()
```

Clean up memory

**Returns**

0

```
Algorithm_Data * create_algo_data_store ( )
```

```
Algorithm_Data* create_algo_data_store(int num_frames)
```

Creates an empty **Algorithm\_Data** to init an **Algorithm**

**Returns**

{Algorithm\_Data\*} empty **Algorithm\_Data** struct for an **Algorithm**

```
int delete_frame_from_list ( Algorithm_Data * data,  
                           int          index,  
                           int          level  
                           )
```

**Algorithm** functions

```
int delete_frame_from_list(Algorithm_Data *data, int index, int level)
```

Deletes a frame from specific memory level based on the index.

**Parameters**

**data** {Algorithm\_Data} the algorithm data

**index** {int} index of frame

**level** {int} level of frame

{int} 0 or 1 base on success or failure



**Frame \* create\_empty\_frame ( int index )**

Frame\* create\_empty\_frame(int num\_frames)

Creates an empty **Frame** for page table list

**Returns**

{Frame\*} empty **Frame** entry for page table list

**int event\_loop ( )**

Control functions

int event\_loop()

page all selected algorithms with input ref

**Parameters**

**page\_ref** {int} page to ref

**Returns**

0

**void gen\_page\_refs ( )**

void gen\_page\_refs()

Generate all page refs to use in tests

**Returns**

0

**Page\_Ref \* gen\_ref ( )**

Page\_Ref\* gen\_ref()

generate a random page ref within bounds

**Returns**

{Page\_Ref\*}

**int get\_ref ( )**

int get\_ref()

get a random ref

**Returns**

{int}

```
int init ( )
```

Init/cleanup functions

```
int init()
```

Initialize lists and variables

**Returns**

0

```
int insert_frame_into_list ( Algorithm_Data * data,
                           struct Frame *   framep_in,
                           int               level
                           )
```

```
int insert_frame_into_list(Algorithm_Data *data, struct Frame *framep_in, int level)
```

Inserts a frame to specific memory level.

**Parameters**

**data** {**Algorithm\_Data**} struct holding algorithm data

**frame** {**Frame**} the frame to insert

**level** {int} level of frame

{int} 0 or 1 base on success or failure

```
LIST_HEAD ( Page_Ref_List ,
           Page_Ref
           )
```

Data structures

```
int main ( int   argc,
           char * argv[]
           )
```

```
int main(int argc, char *argv[])
```

**Parameters**

**argc** {int} number of commandline terms eg { './pagesim' => argc=1 }

**argv** {char \*\*} arguments passed in

Run algo if given correct arguments, else terminate with error

```
int page ( int page_ref )
```

```
int page()
```

page all selected algorithms with input ref

**Parameters**

**page\_ref** {int} page to ref

**Returns**

0

```
int print_help ( const char * binary )
```

Output functions

```
int print_help(const char *binary)
```

**Parameters**

**prog\_name** {const char} the name of the program

Function to print results after algo is run

0

```
int print_list ( struct Frame * head,  
                 const char * index_label,  
                 const char * value_label,  
                 int level  
                )
```

```
int print_list(struct Frame head, const char index_label, const char* value_label, int level)
```

Print list

**Parameters**

**head** {Frame} head of frame list

**index\_label** {const char\*} label for index frame field

**value\_label** {const char\*} label for value frame field

**level** {int} level of memory

0

```
int print_stats ( Algorithm algo )
```

```
int print_stats(Algorithm algo)
```

Function to print results after algo is run

**Parameters**

**\*data** {Algorithm\_Data} struct holding algorithm data

0

```
int print_summary ( Algorithm algo )
```

```
int print_summary(Algorithm algo)
```

Function to print summary report of an Algorithm

**Parameters**

**\*data** {Algorithm\_Data} struct holding algorithm data

0

## Variable Documentation

```
Algorithm algos[2]
```

**Initial value:**

```
= { {"AGING", &AGING, 0, NULL},  
    {"AGING_N", &AGING_N, 0, NULL} }
```

Array of algorithm functions that can be enabled

```
int counter = 0
```

Runtime variables, don't touch

```
int debug = 0
```

```
int last_page_ref = -1
```

```
int last_page_ref2 = -1
```

```
int max_page_calls
```

```
size_t num_algos = 2
```

```
int num_frames
```

Configuration variables

```
int num_refs = 0
```

```
int* optimum_find_test
```

```
Page_Ref
```

```
int page_ref_upper_bound
```

```
FILE* pFile
```

```
int print_count = 0
```

```
int printrefs = 0
```

```
double time_spent_global = 0
```

Generated by  1.8.10

## References

1. Andrew S. Tanenbaum ,*Modern Operating Systems*, Prentice Hall, 2nd edition, 2001.
2. Abraham Silberschatz, Greg Gagne, Peter B. Galvin, *Operating System Concepts*, Wiley, 8th edition, 2008.
3. Harvey M. Deitel, Paul Deitel, David R. Choffnes, *Operating Systems*, Prentice Hall, 3rd edition, 2003.
4. *Page Replacement Algorithm*, n.d., In Wikipedia, Retrieved September 2015 from [https://en.wikipedia.org/wiki/Page\\_replacement\\_algorithm](https://en.wikipedia.org/wiki/Page_replacement_algorithm)

## **Acknowledgements**

I would like to express my sincere gratitude to my scientific supervisor over the last year, Dr. Leonid Barenboim, and to my scientific mentor over the last three years, Dr. Lior Amar, which led me to my current achievements in the field of computer science - in theory as well as in practice.

Also, I would like to thank my research facility and its executives for giving me the opportunity to learn for this masters degree as a permanent part of my professional experience.