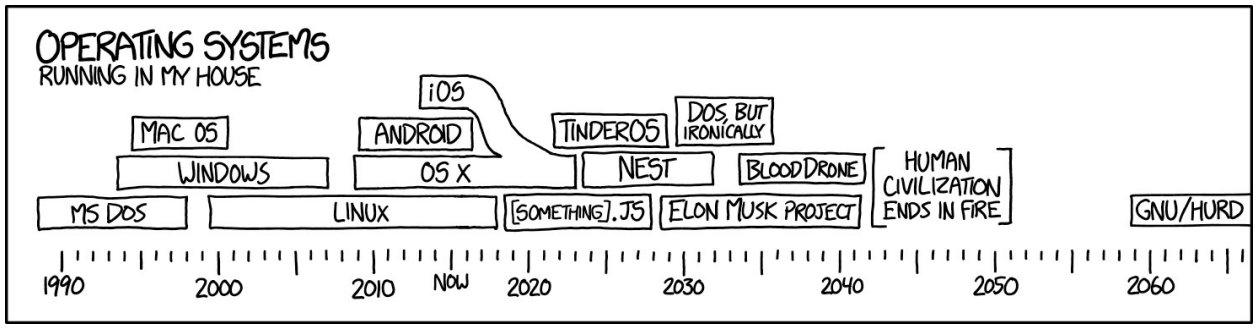
CS 370 – Project #5, Part A

Purpose: Familiarization with Frame Allocation
Points: 200



Source: xkcd.com/1508

Introduction

In a multiprocessing environment, processes may exceed the available memory. Virtual memory allows the *swapping* of pages to and from disk. This allows the aggregate set of processes to exceed the available memory transparently to each process.

To limit the impact of any process upon the set of all processes, each process is allocated a limited number of pages. When limited to a specific number of pages, the set of pages in memory are referred to as *frames*.

In this project, the impact of frame allocation and replacement algorithms will be explored. Implemented as a simulator, input to the simulator is a sequence of memory accesses. Each memory access is recorded as a page hit or fault while maintaining the frame table for a single process.

Project

Project #5 will be done entirely in a **Ubuntu Linux** environment (not xv6). The following steps must be completed in Part A

- Implement a memory access simulator
  - Where the page size (in bytes) is configurable
  - Where the frame table size (in pages) is configurable
  - The frame replacement algorithm is configurable
    - First-In-First-Out (FIFO)
    - Least Recently Used (LRU)
- Implementation **must** be in C, other languages will be rejected
- Very operation of the simulator

Simulator Specification

A template for implementing the simulator has been provided as paging-template.c. The template provides argument parsing, and default parameters, easing the development of the program body. Implement the functionality described by the arguments to the simulator.

Simulator Arguments

```
 14:24:10 ct@snarl ~
 514 ▸ ./paging --help
USAGE: paging [OPTION] <ACCESS 1> <ACC. 2> ... <ACC. N>

Simulates the impact of memory accesses for a given frame
replacement algorithm, frame table size, memory access
sequence.

OPTIONS:
  -p, --page-size    number of bytes in each page
  -f, --frame-size   number of frames for the process
  --fifo             first-in-first-out policy
  --lru              least-recently-used policy
  -v, --verbose      display the frame table at each step
  --help             this message

Every <ACCESS> is recorded as a hit or miss in the frame
table. The output of the program is the number of total
frame table hits and page table faults
 14:28:34 ct@snarl ~
 515 ▸
```

`-p, --page-size`: the number of bytes per page (default of 4096)
Each memory access is contained in a page in main memory, calculating the page of each access will be necessary.

`-f, --frame-size`:
the number of pages the process may keep in memory
The number of frames must be dynamic between invocations, they begin at 0.

`--fifo`: the default page replacement algorithm, first in first out.

`--lru`: the alternative page replacement algorithm, least recently used.

`-v, --verbose:` increases the amount of information displayed

Note: The simulator output must match the examples ***exactly*** with and without the `-verbose` option.

Usage

The sequence of memory accesses are provided as the final list of arguments on the command line. The template handles the argument parsing as well as processing each memory access one at a time. Once compiled, an example invocation of `paging` would be:

```
$ #FIFO, 4096 byte pages, 4 frames
$ ./paging 15872 19860 15731 703 4883 19225
```

Each memory access is given in decimal, 15872 would be 0x3e00 in hexadecimal. Given the page size (4096) the simulator must calculate the page which 15872 resides in, which is 3 in this example. The page must be moved into memory and placed in the frame table, the simulator simulates this operation by placing the frame number in the table and recording a page fault.

Example output for **`./paging 15872 19860 15731 703 4883 19225`**

```
14:39:57 ct@snarl ~
● 530 ▸ ./paging 15872 19860 15731 703 4883 19225

Beginning simulation
[ 0] Access    15872 (dec) 0x00003e00 (hex) page 3    frame 0    fault yes
[ 1] Access    19860 (dec) 0x00004d94 (hex) page 4    frame 0    fault yes
[ 2] Access    15731 (dec) 0x00003d73 (hex) page 3    frame 1    fault no
[ 3] Access      703 (dec) 0x000002bf (hex) page 0    frame 0    fault yes
[ 4] Access     4883 (dec) 0x00001313 (hex) page 1    frame 0    fault yes
[ 5] Access    19225 (dec) 0x00004b19 (hex) page 4    frame 2    fault no
Page Hits:   2
Page Faults: 4
14:39:58 ct@snarl ~
● 531 ▸
```

Example output for **`./paging --lru 15872 19860 15731 703 4883 19225`**

```
14:51:13 ct@snarl ~
● 536 ▸ ./paging --lru 15872 19860 15731 703 4883 19225

Beginning simulation
[ 0] Access    15872 (dec) 0x00003e00 (hex) page 3    frame 0    fault yes
[ 1] Access    19860 (dec) 0x00004d94 (hex) page 4    frame 1    fault yes
[ 2] Access    15731 (dec) 0x00003d73 (hex) page 3    frame 0    fault no
[ 3] Access      703 (dec) 0x000002bf (hex) page 0    frame 2    fault yes
[ 4] Access     4883 (dec) 0x00001313 (hex) page 1    frame 3    fault yes
[ 5] Access    19225 (dec) 0x00004b19 (hex) page 4    frame 1    fault no
Page Hits:   2
Page Faults: 4
14:51:17 ct@snarl ~
● 537 ▸
```

With the **`--verbose`** option, the state of the initial page table must be displayed before each access. After each access the table is displayed once again.
Example output for **`./paging 15872 19860 15731 703 4883 19225 --verbose`**

```
 14:46:06 ct@snarl ~
● 533 ▶ ./paging 15872 19860 15731 703 4883 19225  -v | less
 14:46:43 ct@snarl ~
● 534 ▶ ./paging 15872 19860 15731 703 4883 19225  -v
page-size: 4096
frame-size: 4
policy: FIFO

Initial table
Frame | Page | Age
------+------+----
   0  |  -   | -
   1  |  -   | -
   2  |  -   | -
   3  |  -   | -

Beginning simulation
[ 0] Access    15872 (dec) 0x00003e00 (hex) page 3    frame 0    fault yes
Frame | Page | Age
------+------+----
   0  |  3   | 0
   1  |  -   | -
   2  |  -   | -
   3  |  -   | -

[ 1] Access    19860 (dec) 0x00004d94 (hex) page 4    frame 0    fault yes
Frame | Page | Age
------+------+----
   0  |  4   | 0
   1  |  3   | 0
   2  |  -   | -
   3  |  -   | -

[ 2] Access    15731 (dec) 0x00003d73 (hex) page 3    frame 1    fault no
Frame | Page | Age
------+------+----
   0  |  4   | 0
   1  |  3   | 0
   2  |  -   | -
   3  |  -   | -

[ 3] Access      703 (dec) 0x000002bf (hex) page 0    frame 0    fault yes
Frame | Page | Age
------+------+----
   0  |  0   | 0
   1  |  4   | 0
   2  |  3   | 0
   3  |  -   | -

[ 4] Access     4883 (dec) 0x00001313 (hex) page 1    frame 0    fault yes
Frame | Page | Age
------+------+----
   0  |  1   | 0
   1  |  0   | 0
   2  |  4   | 0
   3  |  3   | 0

[ 5] Access    19225 (dec) 0x00004b19 (hex) page 4    frame 2    fault no
Frame | Page | Age
------+------+----
   0  |  1   | 0
   1  |  0   | 0
   2  |  4   | 0
   3  |  3   | 0

Page Hits:   2
Page Faults: 4
 14:46:47 ct@snarl ~
● 535 ▶
```

Example output for **`./paging --lru 15872 19860 15731 703 4883 19225 -v`**

```
 14:51:17 ct@snarl ~
● 537 ▶ ./paging --lru 15872 19860 15731 703 4883 19225 -v
page-size: 4096
frame-size: 4
policy: LRU

Initial table
Frame | Page | Age
------+------+----
   0  |  -   | -
   1  |  -   | -
   2  |  -   | -
   3  |  -   | -

Beginning simulation
[ 0] Access    15872 (dec) 0x00003e00 (hex) page 3    frame 0    fault yes
Frame | Page | Age
------+------+----
   0  |  3   | 0
   1  |  -   | -
   2  |  -   | -
   3  |  -   | -

[ 1] Access    19860 (dec) 0x00004d94 (hex) page 4    frame 1    fault yes
Frame | Page | Age
------+------+----
   0  |  3   | 1
   1  |  4   | 0
   2  |  -   | -
   3  |  -   | -

[ 2] Access    15731 (dec) 0x00003d73 (hex) page 3    frame 0    fault no
Frame | Page | Age
------+------+----
   0  |  3   | 0
   1  |  4   | 1
   2  |  -   | -
   3  |  -   | -

[ 3] Access      703 (dec) 0x000002bf (hex) page 0    frame 2    fault yes
Frame | Page | Age
------+------+----
   0  |  3   | 1
   1  |  4   | 2
   2  |  0   | 0
   3  |  -   | -

[ 4] Access     4883 (dec) 0x00001313 (hex) page 1    frame 3    fault yes
Frame | Page | Age
------+------+----
   0  |  3   | 2
   1  |  4   | 3
   2  |  0   | 1
   3  |  1   | 0

[ 5] Access    19225 (dec) 0x00004b19 (hex) page 4    frame 1    fault no
Frame | Page | Age
------+------+----
   0  |  3   | 3
   1  |  4   | 0
   2  |  0   | 2
   3  |  1   | 1

Page Hits:   2
Page Faults: 4
 14:52:34 ct@snarl ~
● 538 ▶
 14:53:06 ct@snarl ~
```

Submission
When complete, submit:

- All source files must compile, assemble, and execute on Ubuntu using **gcc**.
- Upload the source via the **codeGrade** link
- *Note*, late submissions will not be accepted.
- You may re-submit any number of times before the due date. However, codeGrade limits the number of submissions to 5 per hour.

Program Header Block
All source files must include your name, section number, assignment, NSHE number, and program description. The required format is as follows:

```
/*
  Name: <your name>
  NSHE ID: <your id>
  Section: <section>
  Assignment: <assignment number>
  Description: <short description>
  Input: <short description of program input>
  Output: <short description of program output>
*/
```

Failure to include your name in this format will result in a loss of up to 3%.

Scoring Rubric
Scoring will include functionality, code quality, and documentation. Below is a summary of the scoring rubric for this assignment.

Criteria	Weight	Summary
Compilation	-	Failure to compile will result in a score of 0.
Program Header and General Comments	10%	Must include header block in the required format (see above). Must include appropriate comments for your algorithm.
FIFO Functionality	45%	Program must meet the FIFO functional requirements as outlined in the assignment.
LRU Functionality	45%	Program must meet the LRU functional requirements as outlined in the assignment.