

# CprE 308 Laboratory 6: Memory Management

Department of Electrical and Computer Engineering  
Iowa State University

## 1 Submission

Submit the following items on Canvas:

- (10 pts) A summary of what you learned in the lab session, no more than two paragraphs
- (90 pts) A zip file containing your report, all source code files along with a Makefile. Make sure your source code is well-commented.

**Grading criteria of the program code:**

- (10 pts) Program compiles
- (25 pts) FIFO is implemented and produces correct output
- (25 pts) LRU is implemented and produces correct output
- (30 pts) OPT is implemented and produces correct output

## 2 Task

Download the file `memory_mgmt.c` and understand the different parts involved. Then fill the missing code for the three functions:

- `PRAlgo_FIFO(...)`
- `PRAlgo_LRU(...)`
- `PRAlgo_OPT(...)`

## 3 Introduction and Background

In this assignment, you will write a program that simulates three virtual memory page replacement algorithms: Optimal (OPT), Least Recently Used (LRU) and First-In-First-Out (FIFO). Here is a brief description of the three algorithms that you are going to implement in your code.

**FIFO:** When a page needs to be replaced, the page that is chosen is the one which arrived earliest into memory, among all the page frames currently in memory.

**LRU:** The page to be swapped out is the page whose most recent access time is the earliest. LRU works on the idea that pages that have been used in the past few instructions are most likely to be used in the next few instructions too.

**OPT:** When a page need arrives into the memory, the OS replaces the page whose next use will occur farthest in the future. Note that this algorithm cannot be implemented in the general-purpose OS, because the pattern of future page references is usually unknown.

## 4 Program Description

The given program `memory_mgmt.c` runs an experiment that evaluates the performance of different page replacement algorithms. The code contains three parts:

1. Generating different page access sequences
2. Processing the page access sequence and checks whether each page is in memory or not
3. Deciding which page in memory should be replaced when there is a page fault AND there is no space for additional pages in the memory

The first two parts of the code are already written, while the third part is to be written by you. We assume there are:

- 128 virtual pages in total (`NUM_PAGES`)
- 10,000 page accesses in a page access sequence (`NUM_ACCESSES`)
- 16 physical page frames, each of which can hold a virtual page (`NUM_FRAMES`)

Initially, the memory is empty and any page access will result in a **page fault**, which means that the page is not in the memory and must be collected from the disk. For each page access, the OS first checks the memory. If the page is in memory, then it is a **page hit** where the OS access the page from the memory. If the page is not in the memory, then it results in a page fault, where the page is retrieved from the disk. On a page fault, if the memory is not full (there is an empty page frame), then the page is placed in an empty page frame. If the memory is full (i.e. all 16 page frames are occupied), then the OS should find a page to be replaced. The OS should find a page to be replaced by calling the three functions: OPT, LRU, and FIFO.

A page access sequence contains the IDs of the pages that are accessed. **The time of the access is equal to the index of the entry.** There are three types of page reference sequences:

1. First sequence will be sequential (SEQ). The application accesses the pages 1,.. ,128 in order, and this sequence is then repeated.
2. The second sequence is random (RAN): there will be 10000 page IDs, each of them a random number between 1 and 128.
3. The third sequence (LR) is constructed to obey a “locality of reference pattern”. With 90% probability the next page to be accessed is one of the 5 pages that were recently been accessed, and with probability 10% a random page among the 128 pages will be accessed.

We maintain a structure for each page frame, which contains the (1) ID of the page in this page frame (2) most recent access time of the page, and (3) arrival time of the page. On a page hit, the OS updates the time of most recent access. On a page fault, the OS brings the new page to the memory and records the last access and memory arrival times to the same value which is the current time. Here is the structure representing a page frame.

```
typedef struct {
    int page_id;
    int time_of_access;
    int time_of_arrival;
} PageFrame;
```