# ITF22519: Introduction to Operating Systems

Fall Semester, 2021

Lab12: Memory Management

Submission Deadline: November 20th, 2021 23:59

This lab is mandatory!

In this lab, you will write a C program that simulates two page replacement algorithms: first-in-first-out (FIFO) and least-recently-used (LRU) algorithms. When there is a `page fault` and memory is full, OS has to choose one page in the memory to remove and to make room for the incomming page. The selected page is determined by the page replacement algorithm implemented in OS. Let see which page FIFO and LRU algorithm will select.

Before you start, remember to `commit` and `push` your previous lab to your git repository. Then, try to `pull` the new lab to have all necessary materials:

```
$ cd Introduction2OS/labs
$ git pull upstream main
$ cd lab12
```

## 1 Report

Include the followings in the report of your lab assignment:

- GitHub: <yourGitHubAccount> on the top of your report. Otherwise, your report will not be graded.
- Source code of algorithms you implemented.
- The output of testcase(s) provided under lab12 repository.

## 2 Page Replacement Algorithms

You will simulate two virtual memory page replacement algorithms: FIFO and LRU. Before you start your implementation, lets see why page replacement algorithms are necessary.

A process in a computer may want to have more and more resource in the system; one of the resources can be memory space. Often, the total amount RAM required by all processes is much more than what can fit in in the system. One way to solve this problem is to use Virtual Memory (VM). The system maintains physical memory (RAM) and VM whose capacity is much larger than that of RAM. In short, in VM, both virtual memory and physical memory are divided into fixed-unit blocks called ***pages***. An application is also broken up into pages, some are in the RAM and some are in the disk. When a program references a part of its address space, if the address is in RAM, hardware performs necessary mapping between physical address and virtual address. Otherwise, there is a page fault. At this time, if the memory is full, OS has to select one page in the memory to remove. Different page replacement algorithm may suggest different page for the OS.

**FIFO:** FIFO is the simplest page-replacement algorithm. When a page needs to be replaced, the OS will simply selects the one which has been in memory for the longest time. FIFO performs poorly in practice since it ignores the usage history of a page and only focuses on the time of arrival. Some pages may be important, when they get old, replacing them will cause an immediate Page Fault. Thus, FIFO is rarely used in its original form. Look at the *PageReplacementAlgorithms.pdf* for an example of FIFO.

**LRU:** The logic behind LRU algorithm is that the page which has been used in only few instructions in the past is likely to be used in the next few instructions too. Therefore, in this algorithm, the page that has been used least recently is evicted. Look at the *PageReplacementAlgorithms.pdf* for an example of LRU.

# 3 Assignment Description

We assume that there are 16 page frames in the memory each of which can hold a single page. For each page access, the OS first checks in the memory. If the page is in the memory, there is `page hit`; otherwise, there is page fault. On a page fault, if the memory is full i.e, all 16 page frames are occupied, then OS has to find one page to be replaced.

In this assignment, we assume there is page fault and that the memory is full. You are asked to implement FIFO and LRU algorithms to recommend the page for the OS to evict from the system. Part of the code is already provided for you in the file *pra.c*. Your job is to add your code on the top of the provided code.

## 3.1 Page frame Structure

We maintain one structure for page frame. The file *pra.c* contains the main function that will call two page replacement algorithms implemented by you. We maintain one structure for page frame. The frame structure is defined as follow:

```
struct PageFrame{
     int Id;    /* Page ID */
     int ArrivalTime;  /* Time that the page arrives */
     int LastRefTime;  /* the Last time that the page was refered in the past */
     int Rbit;         /* Referenced(R)  bit */
     int Mbit;         /* Modified(M) bit */
};
```

In the page frame structure, `Id` is the ID of the page which ranges from 0 to 16. `ArrivalTime` is the time that the page is loaded into the system. `LastRefTime` is the last time that the page was referred in the past. `Rbit` is the reference bit; it is set when the page is read or written. `Mbit` is the modified bit and is set when the page is modified. You do not have to use `Rbit` and `Mbit` for FIFO and LRU algorithms but may be for other algorithms. All fields are inputs for FIFO and LRU algorithms.

## 3.2 Input file

The *pra.c* program reads the five values for each page from the *Testcase.txt* file and then stores the input information for each in a page array. You can take a look at *Testcase.txt* for more information about the format of the file. In general, it contains three columns that are corresponding to the five values of the page frame structure. Each line corresponds to one page.

# 4 Your Implementation

You are asked to implement `first_in_first_out()` and `least_recently_used()` functions. You are not supposed to modify the code provided but you are free to add new variables, functions... on the top of it.

## 4.1 void first_in_first_out()

Your implementation for FIFO algorithm is in this function.

## 4.2 void least_recently_used()

Your implementation for LRU algorithm is in this function.

## 4.3 Output

For each page replacement algorithm, print out the information of the selected page. The sample output is given below:

```
First-in-First-out...
Page selected: Page 3, Loaded at time 110,  Last Refered at time 285, Rbit 1, Mbit 1.

Least-Recently-Used...
Page selected: Page 1, Loaded at time 230,  Last Refered at time 265, Rbit 0, Mbit 1.
```

## 4.4 Gradings

Each algorithm implementation weighs for 50 points. For each algorithm, your implementation will be tested with a number of testcases.

- If your implementation produces the correct outputs for all testcases, you will get 50 points.
- If your implementation does not produce the correct output for any testcase, you will get 0 points.
- If your implementation produces the correct outputs for some testcases but not all, we will look at your code and your report to determine your scores.

# 5 What To Submit

Complete this lab and put your files into the lab12 directory of your repository. Run git `add .` and `git status` to ensure the files have been added and commit the changes by running `git commit -m Commit Message`. Finally, submit your files to GitHub by running `git push`. Check the GitHub website to make sure all files have been submitted.