CMPE142 : Operating Systems
Assignment 3: Memory Management
Version 1

Due: May 6 2019 Before Midnight

In this assignment you will create a program that simulates the memory management/swap policies we learned about in class. Your program will read process memory activity traces from a file, and simulate the swap/memory management algorithms from the book, outputting statistics for each type.

This assignment is worth up to 20 points and may be done in groups of up to **three people max.** Each team member can receive up to 20 points. It is expected that groups of more than one student will find an equitable way to distribute the work outlined in this assignment. Groups of fewer than three people will still be expected to do the same assignment (eg, no skipping parts if you are working solo).

Prerequisites

- You will need a Linux development environment. We outlined how to create this in class.
- A github repository to store your code (see below)

The Assignment

Your program will read a file called ‘memory.dat’ containing information about the processes in the system, and how they are accessing memory. Each line of the memory.dat adheres to the following format:

PROCESS_ID   ACTION   PAGE

The PROCESS_ID field is a numeric value corresponding to a process running on the system.

The ACTION field is interpreted variously:

- ‘C’ means the process is created   (‘PAGE’ field not present)
- ‘T’ means the process terminated (‘PAGE’ field not present)
- ‘A’ means the process allocated memory at address ‘PAGE’
- ‘R’ means the process read ‘PAGE’
- ‘W’ means the process wrote to ‘PAGE’
- ‘F’ means the process freed memory at address ‘PAGE’

For example, a sample memory.dat file for two processes might contain:

```
100    C
100    A    1
100    W    1
100    R    1
200    C
100    A    2
200    A    1
200    W    1
200    R    1
100    T
200    T
```

Your program should implement a virtual memory system with 20 total physical pages, and infinite swap. For each line read from the memory.dat file, simulate what the system would do with each of the following memory/swap policies:

- FIFO
- LRU
- Random

For each of the above policies, **first** check if there are unmodified pages available to be immediately reclaimed. If so, use those pages first. If no unmodified pages exist, fall back to the algorithm currently in use.

It may be easier to perform each of the above simulations separately (eg, read memory.dat, run the FIFO simulation, then reset, read memory.dat again, perform LRU, etc).

At the conclusion of each of the above simulations, output the following:

- The page table for each process
- The content of the 20 pages of physical memory

You do not need to consider the following:
- Pages used for the OS
- Pages used for the page tables themselves

You may choose any swap placement scheme you wish, but do keep track!

A sample output might look like the following (note this does not match the previous example):

```
PROCESS 1
        Virtual     1       Physical     4
        Virtual     3       Physical     17

PROCESS 2
        Virtual     7       Physical     1
        Virtual     6       Swap

SWAP
        Process     2       Virtual      6

PHYSICAL
        0                   FREE
        1                   Process      2
        2                   FREE
        3                   FREE
        4                   Process      1
        …
        …
        20                  FREE
```

If a process accesses or frees a page it has not previously allocated, that process is immediately killed. In this case, your output should reflect this:

```
PROCESS 3                   KILLED
```

Notes:
- When a process terminates, all its pages are freed
- If a process is not terminated when the simulation ends, print its final configuration (without freeing its pages)
- Process IDs can repeat, but will always be ‘T’erminated before re-’C’reated
- You do not need to consider code vs data pages
- A process will always be ‘C’reated before any memory is accessed by that process

I don’t have a requirement for the precise output format, but it should be easy to understand, such as shown above.

Start by creating a new github repository for your code. You can sign up for a free github account online if you don’t have one already. Each team member should have their own account, but it does not matter which account owns the repository for this assignment. The repository can be either public or private. Please do not re-use the assignment 1 or 2 repository (make a new one for this assignment).

Next, develop the basic framework – reading reading the memory.dat file, for example. After you have that working, you can split up the remaining work among your team members. It is up to each team to agree on an equitable division of work among the team members.

Teams are **strongly** urged to “commit early, commit often” - this means as soon as you have something working, push your changes to your team’s repository. This ensures that everyone has access to the latest code, and also serves as a backup in case your copy of the code gets messed up.

You may use any programming language you wish.

<u>Submission and Grading</u>

I will use 5 different memory.dat files as input. For each memory/swap algorithm that outputs a correct value (for each memory.dat file), you will receive 1 point, for a maximum of 15 points. Each team member will be awarded up to 5 points based on the answers to the questions below.

On or before the due date, send answers to the questions below via E-Mail or Canvas, and grant me access to your team’s github repository (my github ID is ‘mlarkin2015’). **Each team member must answer the questions separately.**

The answers to the questions below shall be made via email to the email addresses listed in the class syllabus/green sheet, or via Canvas. DO NOT WAIT UNTIL LATE ON THE DUE DATE, as email server lags or delays may result in a late submission. This is one area that I am extremely picky with – even 1 second late will result in a zero score for that part of the assignment.

I will be comparing all submissions to ensure no collaboration has taken place. Make sure you do not copy another group's work. If you copy another group’s work, members of both groups will receive an F in the class and be reported to the department chair for disciplinary action. If you are working in a group, make sure your partners do not copy another group’s work without your knowledge, as all group members will be penalized if cheating is found.

Make sure your final changes are pushed to github before midnight of the due date. I will be cloning your repository “as of” midnight on that date. This means any changes made afterward will not be considered.

<u>Questions</u>

1. Provide the name of the github repository (including the owning account, eg “mlarkin2015/142-assignment-3”)
2. Provide instructions I need to follow to build your assignment.
3. What is the name of your assignment? (Eg, what do I type to invoke it?)
4. For each member in your team, provide 1 paragraph detailing what parts of the lab that member implemented / researched. This may seem obvious but this is the way I use to ensure that each team member was actually contributing usefully to the team. If you worked on the assignment by yourself, you can just send me an email saying “I did the project by myself”, and that will be sufficient.