CS342 Operating Systems – Fall 2022
Project #3 – Two-Level Paging, Address Translation and Page Replacement

Assigned: Nov 25, 2022.
Due date: Dec 9, 2022, 23:59. Document version: 1.4

This project will be done in groups of two students. You can do it individually as well. You will program in C/Linux. Programs will be tested in Ubuntu Linux. Please start as soon as possible.

In this project, you will develop a **paging** simulator that will simulate the paging related activities in a computer for a process. The **virtual memory layout** of the process will be specified in an input file. Another input file will include a set of **virtual addresses** that the program is referencing. The process is given a number of **frames** for its use. When all the frames are filled up, page replacement will take place. For that you will use an algorithm (1-LRU or 2-FIFO). You will simulate **two level paging** with address split scheme: [10, 10, 12]. Virtual addresses and physical addresses are 32 bits long. As output, which will go to an output file, you will produce a list of **physical addresses** corresponding to the virtual addresses in the input file. For each virtual/physical address you will also indicate if that reference caused a **page fault** or not with an "**x**" sign. When a page fault occurs, if there is **free frame**, you will select the **frame** with **smallest number**; otherwise you will choose a **victim page** (hence the corresponding victim frame) according to page replacement algorithm.

The program will be called **pagesim** and will take the following parameters.

```
pagesim  <in1> <in2> <M> <out>  -a <alg> [-r <vmssize> -n <addrcount>]
```

Here, <in1> is the input file that will indicate the used **virtual regions** of the process. Each line in the file specifies a range [X, Y), where X is the start-address and Y is end-address+1. <in2> is the input file containing virtual addresses that process is referencing. <out> is the output file you will produce. <M> is the number of frames that the process can use. The –a <alg> parameter indicates the algorithm to use for page replacement: -a 1 indicates LRU, -a 2 indicates FIFO. The -r parameter is optional. When –r is specified, the program will not use <in1> input file but will use a single virtual memory region of size <vmsize> starting at virtual address 0 (in bytes and as a hexadecimal number) and virtual addresses will be generated randomly according to a random distribution instead of taking them from input file <in2>. Hence –r is specified, we will not have <in1> and <in2> specified. The -n option is used to specify the number of random addresses (<addrcount>) that will be generated.

You do not need to perform command line parameter checking. Assume they are given correctly.

If a **virtual address** given in <in2> is not falling into one of the ranges of given in <in1>, then you need to write the address as it is into the output file together with a "**e**" mark after a space ("e" means unused memory access exception). For example: `0xd3a01000 e`. Hence the output file will contain physical addresses except the lines with "e" mark which contain unconverted virtual addresses.

All numbers (in input files and output file) are in hexadecimal starting with 0x. For example, 0x0000a40c.

Example:

```
pagesim in1.txt in2.txt 10 out.txt -a 1
```

in.txt content:
```
0x00000000 0x00010000
0x00100000 0x001a0000
0x10000000 0x10c00000
```

in2.txt content:
```
0x00000000
0xd3a01000
0x00000010
0x00000011
0x10000d00
0x10000d00
0x10000c20
```

out.txt content:
```
0x00000000 x
0xd3a01000 e
....
```

Another invocation example is:
```
pagesim 10 out.txt  -a 1 –r 0x00100000 -n 1000
```

Submission:

Put all your files into a directory named with your Student ID. If the project is done as a group, the IDs of both students will be written, separated by a dash '-'. In a `README.txt` file, write your name, ID, etc. (if done as a group, all names and IDs will be included). The set of files in the directory will include `README.txt`, `Makefile`, and program source files. We should be able to compile your program by just typing `make`. No binary files (executable files) will be included in your submission. Then tar and gzip the directory, and submit it to Moodle.

For example, a project group with student IDs 21404312 214052104 will create a directory named "21404312-214052104" and will put their files there. Then, they will tar the directory (package the directory) as follows:

```
     tar cvf 21404312-214052104.tar 21404312-214052104
```

Then they will gzip the tar file as follows:

```
     gzip 21404312-214052104.tar
```

In this way they will obtain a file called 21404312-214052104.tar.gz. Then they will upload this file into Moodle. For a project done individually, just the ID of the student will be used as file or directory name.

References:

1. Operating System Concepts, A. Silberschatz et al. Wiley.

Tips and Clarifications:

- Minimum **frame count** (<M>) is 10, maximum is 1000.
- You will implement **two level paging**. You can NOT implement one level paging and produce results accordingly. If you do that, you will not get any points from the project.
- Assume the frames that the process can use are numbered 0 through <M>-1. <M> is the number of frames.
- Even though the maximum frame number is 1000, you will use all 20 bits in a physical address for the frame number (leading bits will just be zeros). 12 bits of the physical address will be offset. 20 + 12 = 32 bits. That means, each physical address must be 32 bits (8 hexadecimal characters) long. It is ok if a lot of leading bits are 0.
- If -r is specified, then you will generate **random addresses** from the specified virtual memory region. You don't need to generate (i.e., you will not generate) an address from outside of the specified region. Hence there will be no output marked with "e".
- When a page fault occurs, if there is free frame, you will select the frame with smallest number. Hence, **which free frame** will be used is **deterministic** (should be the same for all student projects). If there is no free frame, then you will choose a **victim page** according to page replacement algorithm (LRU or FIFO). Hence **which page** will be removed is **deterministic** (should be the same for all student projects). The frame containing the victim page is the victim frame. That page will be removed from that frame. The page causing page fault is loaded into that frame: that means now you will add that "frame number" to the page table entry (in an inner table) of the page that caused the page fault.
- If -r option is specified, then the output should include both the virtual address and the physical address (in the same line). For example: `0x00000000  0x00000000  x.`

- If a virtual address is not in a valid (used) virtual address region, then you will simply mark that virtual address as “e”, without trying to translate. In reality it is not like that but that is fine in the project.