**CS6456 (F2015) Operating Systems**

**Project 5**

**Title: Designing a Virtual Memory Manager**

**Due: November 9, 2015 (11:00 a.m.)**

**Points: 15**

**The Problem**

Write a program in C/C++ to solve the programming project shown on p. 432 in the textbook. You should change the number of frames from 256 to 128 and use LRU to handle page replacement in main memory. Also, FIFO should be used to handle page replacement in the TLB.

Read the project carefully and make sure you understand what the problem you need to solve is. You may first use 256 frames as indicated in the project and the files **addresses.txt** and **correct.txt** (will be available in the **Resources** folder on **collab**) to test your program without page replacement during a page fault. Once it works, you may change the number of frames to 128 and write code to include the LRU page replacement algorithm. **Note** that the file **addresses.txt** contains 1000 virtual addresses ranging from 0 to 65535 (inclusive). These addresses are stored in the file in such a way that each of them is a 5-digit number. However, they are NOT separated by any spaces (i.e. you will see a contiguous sequence of 5-digit numbers).

When a virtual address is read from the **addresses.txt** file, its page number portion is determined and hashed into the page table. (1) If the page is currently not in memory, your program should print a message indicating that a page fault occurs. The message should contain the following information: the virtual address causing the page fault and the page number containing the virtual address. For example, if the virtual address 33153 (in decimal) causes a page fault in main memory, then the message like "virtual address 33153 contained in page 129 causes a page fault" should be printed. Your program then obtains a frame (e.g. frame 37) from the free frame list using the first-fit placement algorithm. It first loads the page (i.e. 129) stored in the **BACKING\_STORE.bin** file into frame 37. It then prints a message like "page 129 is loaded into frame 37" to the screen followed by updating the information in the page table for the incoming page. However, if the memory is full (i.e. the free frame list is empty) at the time when the page fault occurs, your program should first use the LRU page replacement algorithm shown in class to select a page for replacement. Your program should next read the corresponding page from the **BACKING\_STORE.bin** file and store it in the frame that is selected using the LRU page replacement algorithm. Your program should then update the information in the page table for both the incoming and the outgoing pages. The last thing your program should do is to print a message indicating the frame into which the page is loaded. For the above example, if page 129 is loaded into frame 37, then the message like "page 129 is loaded into frame 37" should be printed. (2) If the page referenced is currently in memory, a message like "page 129 is contained in frame 37" should be printed.

After your program finishes handling all of the virtual addresses contained in the **addresses.txt** file, it should print the following to the screen: (1) contents of the page table (e.g. page 0: frame 79, page 1: not in memory, page 2: frame 213, etc.); (2) contents of page frames (e.g. frame 0: page 98, frame 1: page 151, frame 2: empty, etc.); (3) page-fault rate (e.g. 358 page faults out of 1000 references); and (4) TLB hit rate (e.g. 223 hits out of 1000 references). **Note** that first-fit placement algorithm is used for both main memory and TLB. **Note** also that LRU page replacement algorithm is used for both main memory and TLB.

**Data Structures**

You may consider using the following data structures to implement your program.

(1) TLB: a one-dimensional array (0..15) of struct type.

(2) page table: a one-dimensional array (0..255) of struct type.

(3) physical memory: a one-dimensional array (0..255) of one-dimensional arrays (0..64, each entry can be declared as an integer of 4 bytes, a total of 256 bytes in each frame).

**Miscellaneous**

(1) The project does NOT allow team work.

(2) The three files needed (**BACKLING\_STORE.bin**, **addresses.txt**, and **correct.txt**) will be available in the **Resources** folder on **collab**. The **BACKING\_STORE.bin** file name should be hardcoded into your program while the input file containing the virtual addresses should be entered at the command line. Your program should prompt the user to enter the input file name.

(3) You must use a makefile to compile your program.

(4) You must run your program in the Debian VM environment for this project.

(5) You must use the GNU C/C++ development tools (e.g. gcc, g++).

(6) Turn your project in via **collab** in a **tar** file consisting of (i) all headers you have created; (ii) source code file for project 5; (iii) a make file; and (iv) a write-up showing your design and implementation of the project (at least one page long). The **tar** file should be named as follows: **p5**, followed by the first letter of your first name, followed by your last name, and followed by the file extension (i.e. **tar**). For example, the tar file turned in by **John Smith** should be named **p5jsmith.tar**.

(7) Submit also a hardcopy of the **tar** file in class including the source code file, the header files you have created, and the write-up.

(8) Your program should be successfully compiled. Programs failing to compile will not get any points.