CS4414 (SP2017) Operating Systems

Project 3

Title: Designing a Virtual Memory Manager
Due: April 18, 2017 (12:30 p.m.)
Points: 20

The Problem

Write a program in C/C++ to solve the programming project shown on p. 432 in the textbook. You must change the number of pages in the VAS from 256 to 16, the number of frames in the PAS from 256 to 8, and the number of entries in the TLB from 16 to 4. **Note** that the page size and frame size remain unchanged (*i.e.* 256 bytes). You must use LRU to handle page replacement in main memory. Also, you must use FIFO to handle replacement in the TLB.

Read the project carefully and make sure you understand what the problem you need to solve is. I will provide you with the **addresses.txt** file later so that you can use it to test your program. The file contains virtual addresses ranging from 0 to 4095 (inclusive) (The VAS ranges from 0 to 4095 because there are 16 pages in the VAS, each of which has 256 bytes). These virtual addresses are stored in the file in such a way that each of them is stored on a separate line terminated with '\n'.

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 must print a message indicating that a page fault occurs. The message must 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 3154 (in decimal) causes a page fault in main memory, then the message like "virtual address 3154 contained in page 12 causes a page fault" must be printed. Your program then obtains a frame (e.g. frame 6) from the free frame list using the **first-fit** placement algorithm. It first loads the page (*i.e.* page 12) stored in the **BACKING_STORE.bin** file (which will also be provided to you later) into frame 6. It then prints a message like "page 12 is loaded into frame 6" 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 must first use the LRU page replacement algorithm shown in class to select a page for replacement. Your program must 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 must then update the information in the page table for both the incoming and the outgoing pages. The last thing your program must do is to print a message indicating the frame into which the page is loaded. For the

above example, if page 12 is loaded into frame 6, then a message like "page 12 is loaded into frame 6" must be printed. (2) If the page referenced is currently in memory, a message like "page 12 is contained in frame 6" must be printed.

The TLB is handled in a similar way. When a page is referenced, it is searched in the TLB first for the frame containing it. If the frame number is stored in the TLB (called a TLB hit), a message like "page 12 is stored in frame 6 which is stored in entry 3 of the TLB" must be printed. However, if the page referenced cannot be found in the TLB (called a TLB miss), a message like "frame number for page 12 is missing in the TLB" must be printed. Your program then consults with the page table for the frame number by following the steps shown above. Once the frame number is obtained, it must be stored in an entry of the TLB determined using the **first-fit** placement algorithm. If the TLB is full, then the FIFO replacement algorithm must be used to select an entry for replacement. A message like "frame 6 containing page 12 is stored in entry 3 of the TLB" then must be printed for both cases (TLB is full or not full). Make sure that when a page is replaced from the physical memory, it must also be removed from the TLB entry if it is cached there.

After your program finishes handling all of the virtual addresses contained in the **addresses.txt** file, it must print the following to the screen: (1) contents of the page table (e.g. page 0: frame 7, page 1: not in memory, page 2: frame 3, *etc*.); (2) contents of page frames (e.g. frame 0: page 8, frame 1: page 5, frame 2: empty, *etc*.); (3) page-fault rate (e.g. 9 page faults out of 12 references); and (4) TLB hit rate (e.g. 6 hits out of 12 references). **Note** that first-fit placement algorithm must be used for both physical memory and the TLB. **Note** also that LRU page replacement algorithm must be used to handle page faults for physical memory and FIFO replacement algorithm must be used for the TLB.

Data Structures

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

(1) TLB: a one-dimensional **array** (0..3) of **struct** type.
(2) Page table: a one-dimensional **array** (0..15) of **struct** type.
(3) Physical memory: a one-dimensional **array** (0..7) of one-dimensional **arrays** (0..255).

Miscellaneous

(1) The project does NOT allow team work.
(2) The two files needed (*i.e.* **addresses.txt** and **BACKLING_STORE.bin**) will be available in the **Resources** folder on **Collab**. The file names of the two files must be hardcoded into your program and must NOT be entered at the command line.
(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 3; (iii) a make file; and (iv) a write-up showing your design and implementation of the project (at least two pages long in **PDF** format). The **tar** file should be named as follows: **p3**, followed by your computing ID, and followed by the file extension (*i.e.* **tar**). For example, the **tar** file turned in by **John Smith** should be named **p3js7nk.tar**, where **js7nk** is the computing ID of **John Smith**.
(7) Hardcopy of the **tar** file is NOT needed.
(8) Your program must be successfully compiled. Programs failing to compile will not get any points.
(9) Test cases used to test your program will be similar to the one that will be provided later.