CMSC 312 Assignment 4: Memory Management Unit

Project due date: 11:59 pm EST, 4/24/19 (HARD deadline)

In this project, you will extend a program that emulates the processing of memory accesses. This program takes a sequence of memory accesses as inputs and emulates TLB, page table, and page fault handling to process that request. You will extend this program in four ways: (1) you will add the mechanisms to search the TLB and a simple, linear page table to determine the physical address of a memory request; (2) you will implement page replacement using three schemes (most frequently used, second chance (clock), and least frequently used) and (3) you will calculate effective memory-access times and effective (page fault) access times.

The program works as follows: type *cmsc312-p4 input-file output-file mechanism-number* at the prompt. The input files will be provided. The output file will contain the results of the processing.

The project will consist of the following tasks:

1. Download the following tarball Project 4 Code to your CS account file space. You should have one file CMSC312p4.tgz.

There is a Makefile in directory CMSC312p4, which makes cmsc312-p4.

2. You will need to implement virtual-to-physical address resolution for the emulated TLB (tlb_resolve_addr) and emulated page table (pt_resolve_addr). These functions take a virtual address (vaddr) and return a corresponding physical address (paddr). The page table emulation also returns whether the page table contains a valid entry for that virtual address. As these functions are run by the MMU, you will need to include calls to hw_update_pageref in each, which emulates the MMU updating the reference bits in the page table entry for the selected page. See how it is invoked in pt_demand_page.

3. You will need to implement *page replacement* when the page table/TLB do not contain a reference to that virtual address. The function pt_demand_page defines the demand paging mechanism (provided), but you will define the supporting page replacements and update the page tables (the TLB updating is provided).

Page table updates consist of allocating a page table entry that associates a page with a frame (pt_alloc_frame). In this function, a page table entry (ptentry) is associated with a frame and operation (read-only or read-write). The page table bits for the page (valid, and reference) must be updated also (call hw_update_pageref). The .h file defines these bits. If a page is being replaced, we need to invalidate that entry. You will also implement the function (pt_invalidate_mapping) to remove (invalidate) the mapping between a virtual page and a physical frame.

We will implement three page replacement mechanisms: (0) most frequently used; (1) second-chance (clock); and (2) least frequently used - use these numbers as the *mechanism-number* argument to the process. There are two functions that must be implemented for each page replacement mechanism: (1) update, which updates the page replacement data structures when a new page table entry is made valid (allocated); and (2) replace, which performs the page replacement mechanism. The replacement functions must make the same decisions as dictated by the page replacement algorithms.

See an example implementation of FIFO replacement to give you an idea of how to implement replacement mechanisms.

4. Finally, when all the memory access requests have been processed, we need to compute some summary information.

The two computations that you will need to add to write_results are: (1) effective memory-access time and (2) effective access time relative to page faults. Use the information for *tlb_hit_ratio* and *pf_ratio* (page fault) to compute these values. Use 20 nanoseconds for TLB lookup time, 100 nanoseconds for memory access time, and other #defines for overheads in cmsc312-p4.h.

5. Grading:
Address resolution: 15 points
Allocation/invalidation: 15 points
Page replacement algorithms (most frequently used, second chance (clock), least frequently used): 50 points
Effective time calculations: 10 points
Correct submission: 10 points

Extra notes/explanations/reminders:

1. To help you better understand what you are supposed to do, we provide a demo executable file project2-demo which implements FIFO algorithm. To run it, first write your own input file in the same directory, then type the command *project4-demo input-file output-file* in the terminal. We also provide the code to implement FIFO algorithm cmsc312-fifo.c, note that this code works as a reference, it cannot be compiled alone.

2. When creating your own input file, please be careful about the virtual address format. The page size is 0x1000 (PAGE_SIZE = 0x1000) and there are at most 64 virtual pages per process (VIRTUAL_PAGES = 64). That is to say, the maximum allowed virtual address is 64 * 0x1000 = 0x40000. Otherwise, a segmentation fault will occur.

3. In this project, if the accessed page offset is smaller than 0x200, it is write operation, otherwise, it is read operation.

4. When implementing most frequently used algorithm, you can use the field ct in ptentry to record how many times this page has been referred.

5. The time metrics such as TLB searching time, page table searching time, page fault overhead, etc, are constants defined in .h file. So you do not need to measure those in your project (since this is only a simulation project, it is meaningless to measure those time metrics). However, you do need to use your recorded tlb_hit_ratio and pf_ratio to calculate effective memory-access time and effective access time in your program.

6. In functions tlb_resolve_addr and pt_resolve_addr, if there is a hit, do not forget to call hw_update_pageref to update reference bit and (perhaps) dirty bit.

7. When invalidate a mapping, do not forget to check the dirty bit to determine whether it is necessary to overwrite the disk data (call pt_write_frame).

8. Although it is not a very difficult project, there are hundreds lines of codes you need to go through. So you may want to start early and ask TA or instructor for help when you get stuck.

Note: Late assignments will lose 5 points per day upto a maximum of 3 days. Code must compile and execute on the class Linux server.
