CS105 – Computer Systems Spring 2020

Assignment 5: Virtual Memory Homework

Due: Tuesday, April 14, 2020 at 11:59pm

For this assignment, you will emulate virtual memory at the user level. You may complete this assignment either individually or with a partner. If you choose to complete it with a partner, you must follow the rules of peer-programming (you must both be working on it together, one person should type, but you should both be able to see the code). Note that zoom has an option for screensharing if you do decide to work with a partner.

The starter code for this assignment is available both on the course website and on the course VM. You may complete it either on your local machine or on the course VM (the file path is `/data/vm.tar`. Note that you will need to be connected to the Pomona VPN to access the VM.

Once you have a copy copy of the tar file in the directory of your choice, unpacking the file with "`tar xvf vm.tar`" will create a subdirectory named `vm` containing the writeup, a `makefile`, and the starter code `vm.c`.

1 Background

Recall that virtual memory allows a process to use more memory than is available on the physical machine. Virtual memory is divided into pages, and some pages are stored in frames in physical memory while others are written out to the disk. When a process attempts to access a value in "memory" that is not actually in physical memory, a page fault occurs. The operating system kernel handles page faults by (1) identifying the missing page, (2) evicting some other page from physical memory, (3) reading the missing page from disk and storing it in the newly available frame, (4) updating the process's page table, and (5) restarting the faulting instruction.

2 Starter Code

The starter code for this assignment contains most of the code necessary to simulate virtual memory. Physical memory is implemented as an array of bytes; the address of this array is stored in the global variable `physical_memory`. The physical address of a value is the index of (the start of) that value in the array `physical_memory`.

In the simulation, the page table is stored an array of `page_table_entry_t` values (page table entries) that is stored on the heap; the address of this array is stored in the global variable `page_table`. The type `page_table_entry_t` is defined at the top of the file; it contains a frame number, a valid bit, and a counter (which will be used later). For simplicity, there are no access control bits. If the valid bit is set, then that virtual page is stored in physical memory (i.e., in the array `physical_memory`) in the specified frame. If the valid bit is not set, then that virtual page is currently paged out to disk; in this case that virtual page is stored in a file located in the directory `files` whose filename is the page number. For simplicity, you may assume that there are no uninitialized pages in this simulation.

For the purposes of this simulation, we will assume that the only thing stored in virtual memory is an array of random integers (we will ignore the fact that code, global variables, stack frames, etc. are also stored in memory). This random array of integers is generated by the function `gen_array` and stored in our simulated virtual memory system by the funtion `initialize`.

The function `handle_page_fault` simulates the kernel code for handling page faults (it identifies the missing page, evicts some other page from memory, reads the missing page from disk, and updates the simulated page table). The wrapper functions `load` and `store` simulate memory accesses that call the page fault handler code when necessary.

The function `print_simulation_state` prints the current state of the page table and of virtual memory. This might be helpful for debugging purposes. However, I strongly recommend that you solve this assignment on a system where you can run gdb, as that will be a more efficient way to debug.

There are also a series of functions that implement different sorting algorithms; these functions will be used to evaluate the simulation. The function `print_array` prints the current state of the array (that you are in the process of sorting). **Hint:** you might find this function useful for testing your solution, as most bugs will result in an incorrectly sorted array. Or a segfault.

3 Address Translation

Your first task is to implement the function `translate_addr`. This function should take a virtual address and return the corresponding physical address (if the page is in physical memory) or `-1` otherwise. Note that in this simulation, the virtual address is the offset from the beginning of the integer array (so the virtual address of `array[i]` woud be `i*4`) and the physical address is the offset from the beginning of the byte array `physical_memory`.

4 Eviction

The starter code uses a random frame to evict from physical memory; this is not always an efficient eviction algorithm. Your second task is to implement a least recently used (LRU) eviction policy, in which the page that has been accessed least recently will be evicted when a page fault occurs.

Hint: You will probably want to use some global variables to keep track of the order in which pages were accessed. You will also probably need to modify the `initialize` and `address_translate` functions.

5 Evaluation

Once you have implemented and tested your address translation and page eviction code, modify the main function to call the function evaluate (if you haven't made any changes to the main function, it should do this already). Double check that the constants at the top are set correctly before running this code! Redirect or copy the output to a file named results.txt. At the end of that file, add a few sentences summarizing what these results tell you about page eviction algorithms.

Hint: You might also want to try running the evaluate function with various other possible values for the global constants at the top, but make sure the ones you include in your submission are the right ones.

Feedback

Create a file called `feedback.txt` that answers the following questions:

1. How long did each of you spend on this assignment?
2. Any comments on this assignment?

As always, how you answer these questions **will not affect your grade**, but whether you answer them will.

Submission

Use the course submission site to submit your competed program `vm.c`, your evaluation results `results.txt`, and your feedback file. If you are working with a partner, make sure that you tag your partner as a collaborator when you submit. Also, be sure the names of all team members are *clearly* and *prominently* documented in the comments at the top of all submitted files.