COSC 439 Program #4

Michael Santoro

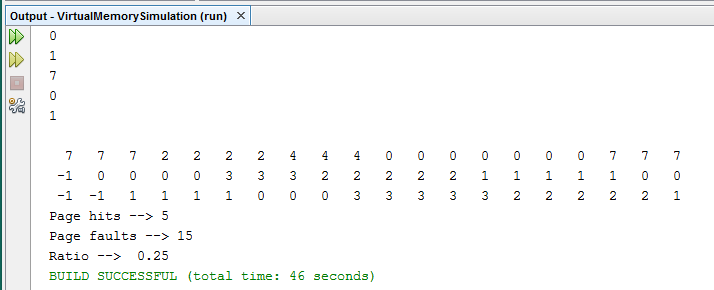
**README**

**PART 1:**

This program is designed to implement the demand paging scheme and page replacement that is used in virtual memory. I have developed this program in java NetBeans IDE on a windows 8 machine.

With virtual memory management, the MMU is used to map the virtual to physical addresses at runtime. Paging is when the physical address space of a process can be noncontiguous and is allocated whenever memory is free. One advantage of this is it avoids external fragmentation, however it still maintains internal fragmentation. We implement virtual memory to separate logical and physical memory. This allows us to have noncontiguous pages in physical memory that when mapped to virtual are now contiguous, and as mentioned before this is done by the MMU.

Demand Paging is an implementation of paging that saves space by only loading a page into virtual memory when it is required during program execution. So if a page is never accessed it will never have to be loaded into virtual memory. This is done using a lazy swapper or lazy pager. A valid-invalid bit is used to determine if the page is valid and in memory, or invalid meaning not an address that can be associated with anything, or an address that is valid but not in memory at the moment. If a process attempts to access an invalid page, this is considered a page fault…and then will handle this page using an OS trap and a save of the registers and process state.

For page replacement, this is simply a way to free up frames. If there are not any frames free, it will find one that isn’t actively being used and it will free it up swapping out and using the page table. The FIFO algorithm is the simplest form of page replacement and will be the one I have implemented in java. Due to time constraints there is no GUI or full running simulation however, I have taken a look at many of the simulations out there to get a better idea of how things work. The jave file “PageRepacement.java” is an adaption of a FIFO page replacement algorithm from http://campuscoke.blogspot.com/2015/01/fifo-page-replacement-algorithm-in-java.html . Essentially what I am doing here is taking in some user input to define the frame and reference string….and then doing the same processing that we had to do for our quiz question in chapter 8. It simply goes through the string and checks memory to determine whether there is a hit or fault…and then it proceeds to remove the number that has been in memory the longest. At the end it does a simply print statement of the accumulated page faults and hits. Below is an example of execution using a reference string from the textbook to confirm that it is working correctly. 

This program is very simple implementation of FIFO page replacement and shows the step by step processing just like our examples from the book. This could also show demand paging scheme in a way because the pages that are placed in frames are only those requested…however, that is a bit of a stretch.

**PART 2:**

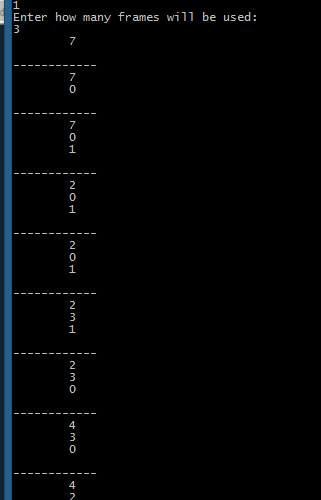
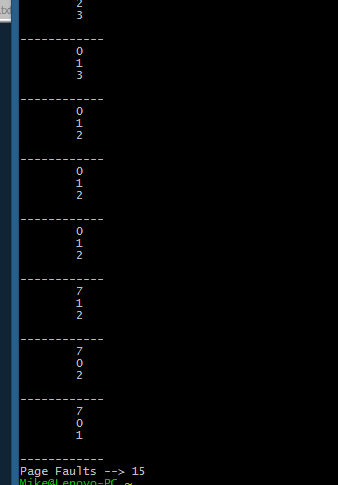
As an extension to this implementation. I decided to do a very similar simulation of demand paging and FIFO page replacement using C. I wrote this code again on a windows 8 machine however using notepad++ as a text editor and Cygwin to compile my and execute the code.

Similar to my java program this simulation does not actually create processes or threads to demonstrate the page faults. It prompts for user input to set up pages, frames, and the reference string. Based on the reference string and frame size, the program continues parsing the string and comparing it to what is stored in our page structure. For this C program, instead of just using the buffer and arrays like the java program…we set up a structure which call “page”…giving it a frame, and 2 arrays storing the count and current.

Once the user input has been obtained…we call the pageReplacement function and pass it the number of pages and the array of pages or the reference string that was built from the user input. We use a for loop to initialize the count for our pages to -1…to indicate empty essentially. And maximize the count. We then parse through as many times through our for loop as the length of the reference string…and check to see if the current reference input…is in the page structure or array. If it is then we break, however if it is not….we declare a page fault and update the page structure…eliminating the current and adding the new number from the reference string.

Below I have attached a picture of execution based on the reference string from the textbook.





As we can see…again we get 15 faults for this refernce string….for this second part of the assignment I will be attaching just a single c file named “demandPaging.c”

**Lessons Learned:**

Overall deand paging and FIFO page replacement are failrly simple concepts, however they are certainly not always the most efficient algorithms and they have their flaws.

**Referenced Examples**

<https://www.youtube.com/watch?v=jgv6UrPhmFI>

<http://www.ontko.com/moss/memory/user_guide.html>

<https://www.youtube.com/watch?v=yoj064PZCsQ>

<http://campuscoke.blogspot.com/2015/01/fifo-page-replacement-algorithm-in-java.html>