Project 2

Temple Tran and Zachary Krevis

Work Distribution: Zachary Krevis -> FIFO, Temple Tran -> Third Chance Replacement

First in First Out:

For First in First out, I used an array to simulate the physical memory. This array was initialized in mm_init. I also keep a singly-linked list of virtual pages, that holds the page number and the fault type associated with the virtual page. I used this list to decide which page should be evicted when a call to our virtual memory is made and our physical memory is full. Because of the properties of first-in-first-out, I would eject the head of that queue whenever space was needed.

If a call was made to a page in memory that is already in memory, it would have to be a call type that doesn't match the permissions given to it previously by my handler to cause another fault. That is how I decided if a fault of type 2 occurred. If a write call was made to a portion that was read only, we'd still need to log that in our logger, but not eject any portion of memory from our physical memory, so all I would do in that situation is update the permissions given to that page in memory to allow for write calls.

In this project, I learned how different paging algorithms are more efficient than others. Before this project I had assumed that first-in-first-out would be the obvious choice, but of course, this caused a lot of unnecessary faults that could have been avoided using a more efficient algorithm.

Third Chance Replacement:

For third chance replacement, I used an array to simulate a circular linked list. In order to do this, I kept track of the front and back of the list with variables holding the indices. When adding a page to the end of the list, I would increase the end of the list variable by one and modding by the size of the array. When evicting a page from the front, I would similarly increase the front of the list variable.

In order to determine which page should be evicted, each page entry contained a r-bit as well as an m-bit. Whenever there was a read access, the r-bit would be set to 1. Whenever there was a write access, the r-bit and m-bit were both set to 1. During the page eviction process, if the (r-bit, m-bit) were either (0, 0) or (-1, 1), the corresponding page would be evicted. On each pass, the bits would be updated in the following manner:

(r-bit, m-bit): (1, 1) -> (0, 1) -> (-1, 1) -> evicted

(r-bit, m-bit): (1, 0) -> (0, 0) -> evicted

However, these bits are reset on each reference/access. In order to raise a sigsegv to track references/accesses, I would mprotect(..., …, PROT_NONE) on certain bit combinations even though the page was technically in the physical memory.

In this project, I learned how to set up and use signal handlers as well as learned how to implement third chance replacement. Initially it was hard to understand what this project entails because it is a simulation of memory management. But after rereading the project multiple times, going to office hours, and referring to class notes, it all came together. It was also beneficial to have a good implementation plan in mind before fully implementing because it allowed for minimal bugs and was time efficient.