Noah Ruben
CMPSC 473
12/7/2020

Introduction

For the 3rd project of the semester, the student was given the task of creating and applying different page replacement algorithms to an initialized virtual memory system. The project incorporates many of the skills we have learned throughout the semester and certainly was a tough challenge.

Overall Design

My version of project 3 contains two functions: mm_init(), which is provided in the repository and is meant to help initialize the system, and mySignalHandler(), which is where most of the code is implemented and is used to handle segmentation faults in the memory in a specified order.

mm_init()

The mm_init() function has four main actions that are vital to the program. These are:

1) Initialize the parameters as global variables
2) Create an array of initialized data structures which contain the frames of the system
3) Route the mySignalHandler() function as the action for all segmentation faults with sigaction()
4) Set the permissions for all pages in the virtual memory to NONE by using mprotect()

These actions set us up for the inputs that will be coming through the system. The inputs will now start creating segmentation faults, which will be immediately handled by the second function mySignalHandler() with no function calls in the code.

mySignalHandler()

This function is where the segmentation faults are parsed and then used in a page replacement algorithm to be managed in the virtual memory.

From using sigaction(), the function has four parameters that are useful in finding different information about the segmentation fault. We are able to find the address which caused the segmentation fault from the “info” parameter by using info->addr. This helps us find which page and offset the segmentation fault took place. The other piece of information that is important to find is whether the segmentation fault was a read or write error. Digging deep into the parameter “context” helped me find this information, which allowed me to start working on the replacement policies.

FIFO

Once all of the information was gathered, I was ready to implement the first replacement policy. The first was the “First in first out” method. The easy I did this was by having a for loop which went through each frame in the frames[] array to check for different fault types, and finally if an eviction was needed. If an eviction was needed, a global variable “evict_frame” is there to keep track of which frame a page should be evicted from. Once a page was evicted, the variable is incremented by 1. When the variable increments greater than the total number of frames, it is declared back to zero. This was my own idea of a circular array. Once a fault type or eviction is chosen, mprotect() is used to set new permissions on the frame so that the segmentation fault does not happen again.

3rd Chance Algorithm

The implementation of the 3rd chance algorithm was very similar to FIFO with a few major changes. The first change was adding 2 more code blocks to catch fault types 3 and 4, which are used to track inputs without causing a segmentation fault in the physical memory. The second major change was creating a while loop in the eviction section which helps to keep the frames updated and choose which frame a page should be evicted from when needed. One challenge I noticed compared to the FIFO algorithm was that it is very important how the data is maintained in each frame for the implementation to work correctly.

Challenges

One of the main challenges with this project was learning how to use sigaction() and mprotect() correctly in order to implement the rest of the code in a correct manner. For example, after the permissions for the entire virtual memory are set to NONE, the program will create an infinite loop unless permissions are properly set to the requested form once again in the signal handler. This made it difficult to debug and find the issue with the program at times. Another challenge I had in creating this program was visualizing the algorithms (especially Third Chance) and translating them to the C code in relation to the virtual memory.

Conclusion

Although this project was very challenging, I feel that I have learned many different techniques which were required to make the program run successfully. I will be able to use these skills in future classes and in my career. It was certainly a great learning experience.

The project has been tested and matches with all of the test cases provided on a VMware Fusion Linux machine.