COSE341 – Operating Systems – Project 1

Department: Computer Science

Student Name: Alanna Morris

Student Number (학번): 2023952711

Submission Date: April 11th, 2024

Free Days Used: 0

Project Development Environment and Main Goals

The main purpose of this project was to compile and modify the Linux kernel source code and implement a custom queue data structure. This required modifying several source files and creating our own implementation and application files.

The development environment we utilized for this project was the Ubuntu 18.04.02 (64-bit) with the Linux kernel version 4.20.11. It was important to use this specific pairing of operating system and kernel seeing as confusion or complications could be experienced if you had the incorrect combination whilst attempting to carry out this project. To conduct the development and testing we were instructed to use a Virtual Machine environment; specially VirtualBox. A convenient benefit of using VirtualBox is that you can comfortably work inside it without externally affecting your host operating systems. Overall, by using these recommended tools for conducting our implementations, it proved to be an efficient and reliable approach for our project.

Explanation of System Calls

A key aspect of this project was to create and implement our own system calls, specifically; a system call to enqueue and dequeue the queue data structures we created. We needed to add the system call handler names into syscall_64.tbl and define the prototypes of system call functions in syscalls.h. System calls are requests which are made from the user program so that the resources protected in the operating system can be accessed and utilized. Typically, these resources are only managed by the operating system, but system calls are a means to gain safe and controlled access.

In Project 1 we needed to use system calls to manage our queue data structure. We created an enqueue system call and a dequeue system call. Our queue data structure was an int array, storing only unique numbers. The enqueue function would intake new integers and insert them in; whilst omitting repeated values and halting if the queue was full. The dequeue function would remove values from the queue and not initiate if the queue was empty. Together enqueue and dequeue worked in a first-in, first-out basis; meaning the values stored in the queue the longest were the first to be dequeued. By developing system calls to perform enqueue/dequeue, we got to appreciate the practical use of system-level programming and practice the process of accessing functions within the operating system.

Implementation:

The main implementations developed in this project reside in the ‘call_my_queue.c’ and ‘sslab_my_queue.c’ files.

In ‘sslab_my_queue’ the enqueue and dequeue operations are handled. The enqueue system call receives an integer ‘a’ to input into the queue. Enqueue first checks for edge cases such as a full queue or a repeated integer entry; and prints an error statement if these scenarios are encountered. Otherwise, the new element is inserted into the front of the array and the “rear” integer variable is incremented to denote the next location of insertion. The enqueue function announces the system call and prints the queue to the kernel which can be seen in the screen shot on the final page of this report. The dequeue function removes the oldest value sitting in the queue and shifts the numbers leftwards in the array. If the queue is empty this function doesn’t operate and an error message is printed. Similar to enqueue, the dequeue function announces its commencement and prints the queue in the kernel each time it is called.

In ‘call_my_queue’ a very basic set of operations is implemented to test out the system calls created. Three random values are coded in to be enqueued and three random values are coded to be dequeued. This file was a very simple test made to see if the system calls for the enqueue/dequeue functions worked properly.

In other scenarios or if the project had a different purpose, I think I would have coded each of the above files differently. However, since both files fulfilled the basic requirements of the project, I left the files as is. In hindsight, in another attempt I think I would have introduced further efficiencies into my code, included more error handlings, and removed redundancies given more time. For me I found the ‘make’ and ‘install’ commands extremely time consuming which deterred me from making extensive changes.

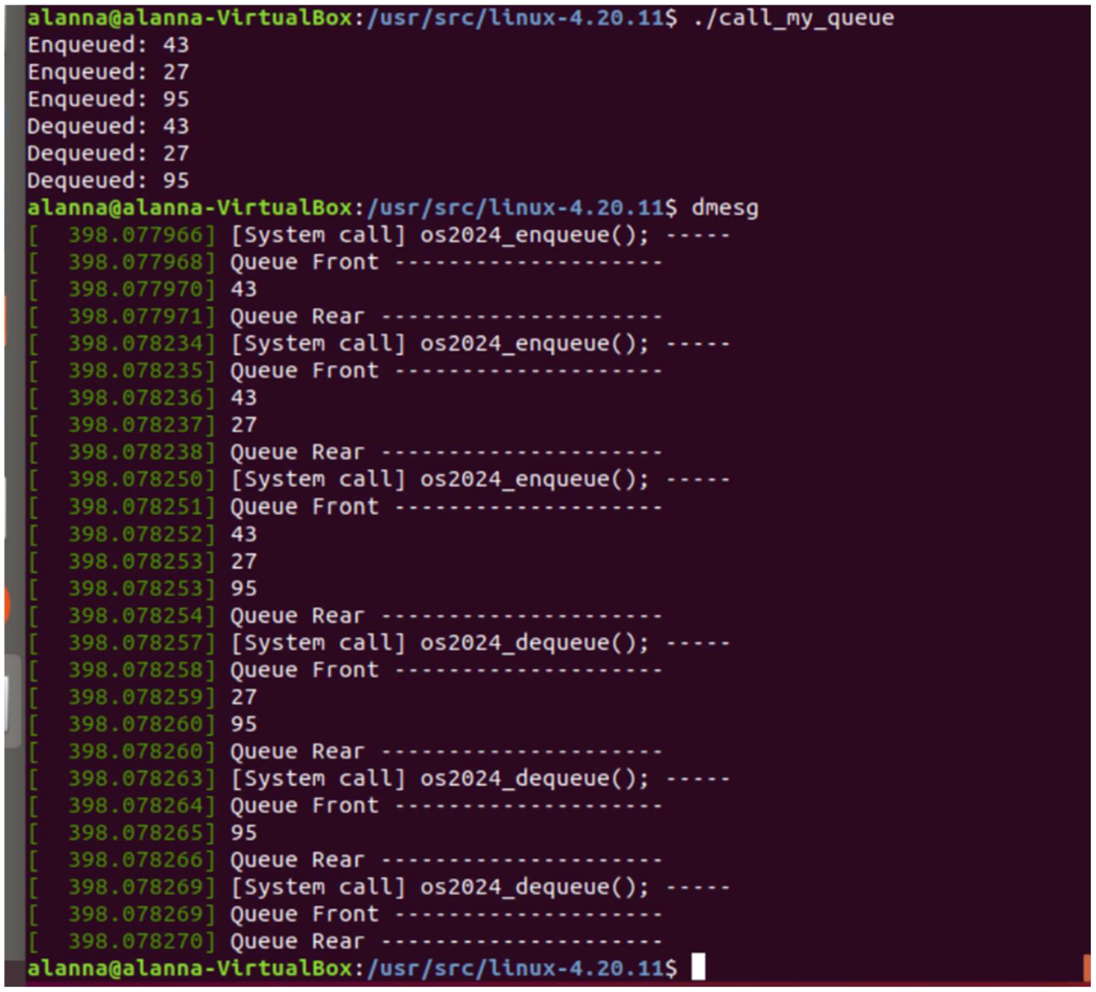
Difficulties and Efforts Encountered During the Project:

Initially, I struggled with my implementation since I wasn’t working in the correct kernel after completing the assignment steps. I was confused why outputs weren’t showing the way I expected them to. After consulting the TA I was directed to a step-by-step work around to this problem located on FAQ #5 online. After adjusting the GRUB files and initializing a menu to select the correct kernel I was able to continue. In the future, I think it could be helpful to include the steps in FAQ #5 into the project guidelines.

Another difficulty I encountered was the considerable amount of time required for executing the "sudo make" and "sudo make install" commands. I made a slight mistake in my print statements in my sslab_my_queue.c file and went back to edit them. However, after editing my code to correct print statements and doing other troubleshooting, I still faced persistent output issues. Ultimately, I had to delete and recreate my implementation files, adjust the Makefile, and repeat the "sudo make" and "sudo make install" processes, which took a lot of time. I haven't found an alternative yet but maybe there is a different approach. In the future I hope time-consuming operations like make and install won't be necessary. However, in the meantime, I did find some ways to accelerate these tasks such as "sudo make -j8" or "-j12". It was helpful to learn of a way to execute multiple compilation jobs at once and decrease my wait times during these tasks.

Snapshot:

Below are the results of the terminal and kernel logs after running the user application:



```
alanna@alanna-VirtualBox:/usr/src/linux-4.20.11$ ./call_my_queue
Enqueued: 43
Enqueued: 27
Enqueued: 95
Dequeued: 43
Dequeued: 27
Dequeued: 95
alanna@alanna-VirtualBox:/usr/src/linux-4.20.11$ dmesg
[  398.077966] [System call] os2024_enqueue(); -----
[  398.077968] Queue Front --------------------
[  398.077970] 43
[  398.077971] Queue Rear --------------------
[  398.078234] [System call] os2024_enqueue(); -----
[  398.078235] Queue Front --------------------
[  398.078236] 43
[  398.078237] 27
[  398.078238] Queue Rear --------------------
[  398.078250] [System call] os2024_enqueue(); -----
[  398.078251] Queue Front --------------------
[  398.078252] 43
[  398.078253] 27
[  398.078253] 95
[  398.078254] Queue Rear --------------------
[  398.078257] [System call] os2024_dequeue(); -----
[  398.078258] Queue Front --------------------
[  398.078259] 27
[  398.078260] 95
[  398.078260] Queue Rear --------------------
[  398.078263] [System call] os2024_dequeue(); -----
[  398.078264] Queue Front --------------------
[  398.078265] 95
[  398.078266] Queue Rear --------------------
[  398.078269] [System call] os2024_dequeue(); -----
[  398.078269] Queue Front --------------------
[  398.078270] Queue Rear --------------------
alanna@alanna-VirtualBox:/usr/src/linux-4.20.11$
```