COP 4600-002: Operating Systems
Fall 2017

Project #3: Shared Memory

Alan Rodriguez, U86831061

I. INTRODUCTION

THIS project demonstrates the use of semaphores and how they can protect processes by locking and unlocking a process's critical section. There are two processes, Producer, and Consumer. They both modify a shared memory segment which is declared as an array of characters. The Producer reads the input of a file and adds it to the shared memory segment until it is full. The Consumer reads from the buffer and prints to the console the contents of the character array until it is empty. There is a mutually exclusive relation between the two processes for the fact that they cannot operate their critical section at the same time.

The semaphores created are mutex, full, and empty. These semaphores use the function sem_wait to decrement and sem_post to increment. Decrementing a semaphore locks all processes from modifying the shared memory segment referenced in the critical sections. Incrementing unlocks the semaphore and allows the process to access shared memory.

Using pthread we can run concurrent processes of Producer and Consumer. Using sem_wait(empty) before Producer's critical, and sem_post(full) after it properly waits to access the character array and only adds to it when it is not full.
Using sem_wait(full) before Consumer's critical, and sem_post(empty) after, the consumer only reads and prints when the character array is not empty.

The threads wait for the processes to finish and afterwards, the semaphores are all destroyed.
Once all the processes have completed, the shared memory must be detached and removed. This is done by passing our shared variable to the function call, shmdt.
Now that shared memory is detached, it can be removed with shmctl(shmid, IPC_RMID, NULL); This ensures that shared memory is removed and there is no memory leakage.

II. RESULTS

Initially I made the file a small amount of characters to test if the buffer size had any impact on the semaphores yet it seemed to be indifferent. The initial file was this stream:
0123456789ABCDEF
I decided to increase the number of characters to a value larger than the threshold and the characters that were not within the threshold were not preserved through the producer and consumer processes.

III. DATA ANALYSIS:

After running the program several times, my results were consistent. Shared memory data was preserved through all processes. Changing the size of the file stream showed how only the characters that were within the threshold of 150 were preserved. Increasing the max characters of message to produce and consume yielded consistent results as well.

IV. CONCLUSION:

In conclusion, it is clear that semaphores play a vital role in protection for shared memory segments. However shared memory is used, via multiple processes requesting access such as forks in processes, or parallel threads modifying a shared variable, semaphores guarantee that data modified is only accessed when the semaphore's lock has been "unlocked".