Josh Romisher

10/31/22

Operating Systems IPC Program Report

**Implementation**

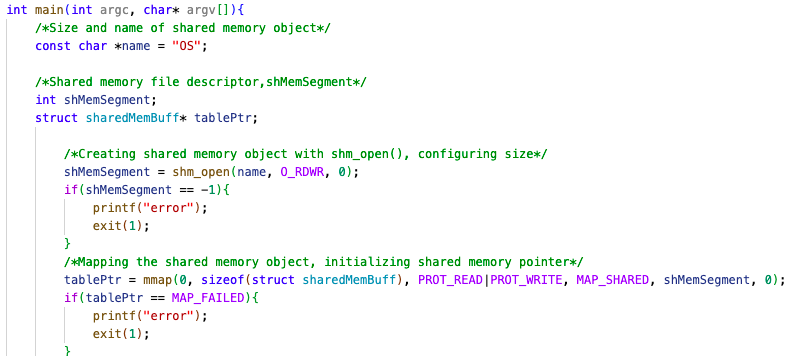
This was my first attempt coding a systems program, and my first attempt at coding in C. I did a good amount of research around semaphores, pthreads, POSIX API, and using shared memory to see how other programmers had implemented solutions to the producer/consumer problem. It was very helpful to see multiple solutions and figure out which ones were applicable to the program we were asked to create.

My first step was figuring out how to implement the shared memory segment completely. I ran into issues here in trying to figure out the correct way to use my shared memory pointer to access and modify the elements shared between the producer and consumer process. I initially implemented this incorrectly, declaring two sem\_t variables for the producer process, and two separate sem\_t variables for the consumer process. Realizing that these two values would change independently of each other, I had to figure out a way for these two processes to share the sem\_t variables, as well as the “table” buffer.

Graphical user interface, text, application

Description automatically generated

Seeding rand(), initializing shared memory segment, creating a pointer to shared memory segment



Opening shared memory in consumer.c

The solution to this issue was to create a shared memory structure in a header file and include this header in both processes. The shared memory structure I created, sharedMemBuff, was used to hold two sem\_t variables (empty and full), as well as the “table” buffer, int buffer[2]. This way, both the consumer and producer processes were able to access these elements in shared memory and use them to read and write values.

Graphical user interface, text, application

Description automatically generated

Shared memory structure for both processes

My next issue in implementation came when I was attempting to use pthread\_mutex\_t locks. I found that when I tried to use a mutex, both of my processes were able to start, but hit a deadlock when I used a mutex. I tried many different combinations of implementing the mutex lock, but in the end I was able to get the two processes to communicate correctly only using semaphores. Once I had the basic structure of both processes, I focused on creating a routine for the consumer and producer processes. I was able to add some prinft() statements to show when the producer or consumer were writing or reading elements to or from the table, which helped when debugging the programs and added a simple interface to the interaction between the two processes.

Graphical user interface, text

Description automatically generated

Producer routine

Before the producer routine I initialize the shared semaphores, empty and full. The producer routine then executes a total of 6 times, using rand() to fill the open buffer slots with a random number between 0 and 99. It uses sem\_wait() and sem\_full() to signal to consumer.c .

Graphical user interface, text, application, email

Description automatically generated

Consumer Routine

Above is the consumer routine, executing a total of 6 times, and reading values added to the “table” buffer from producer.c. Consumer.c uses the semaphores empty and full to signal to producer.c.

**Process Execution Example**

Below is an image of the two processes at work.

**Graphical user interface, text, application

Description automatically generated**

**Conclusions**

This project was a great opportunity to combine many of the topics we have studied this semester in Operating Systems. Hands on implementation is a great way to learn the ins and outs of new material. My final implementation of producer.c and consumer.c were not perfect, but after a lot of small tweaking and reorganizing the two routines, I was able to get a very close execution to what I was expecting and aiming to accomplish. Before completing the project, I had an abstract idea of what IPC and shared memory were supposed to accomplish, but I still struggled to understand how the implementation would work. I enjoyed learning more about coding in C, including shared memory flags to restrict access to certain processes and using format specifiers and different arguments with printf(). Upon project completion, I feel more confident in my understanding of IPC and shared memory and how they are implemented.