Operating Systems Project 3

A shared Protected Circular Queue and Communication between threads

Jason Mellinger
U71145540

***Abstract*— the purpose of this project is to experiment with synchronization mechanisms, specifically using semaphores to protect a limited size resource. There will be a circular buffer with 15 positions used to communicate information between a producer and consumer thread. The producer thread places characters into a buffer 1 by 1 while the consumer thread reads the characters from the shared buffer and prints them to the screen 1 by 1.**

CODE

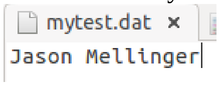
There is the standard three semaphores created globally as they will be used across multiple sections: empty, full, and mutex. There are also two separate threads, producer and consumer, that input data from a dat file into a circular buffer and output data to the screen respectively. The producer thread reads the characters one by one until the EOF marker is reached, and then it inputs an asterisk into the buffer.

The consumer thread will read the characters, one by one, from the shared buffer and print them to the screen. The semaphores along with a 1 second sleep in the consumer function are used to ensure that the processes are synchronized and not accessing the shared resource at the same time. Both producer and consumer threads are created by a parent process, and once those threads have completed the semaphores will be destroyed and shared memory will be released.

Data Analysis

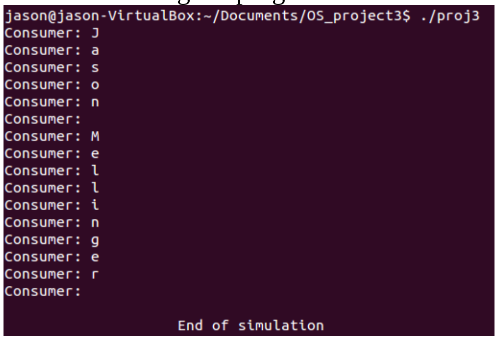
Test 1

Contents of “mytest.dat”:



```
Jason Mellinger
```

Result of running the program:



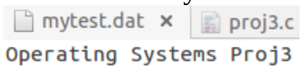
```
jason@jason-VirtualBox:~/Documents/OS_project3$ ./proj3
Consumer: J
Consumer: a
Consumer: s
Consumer: o
Consumer: n
Consumer:
Consumer: M
Consumer: e
Consumer: l
Consumer: l
Consumer: i
Consumer: n
Consumer: g
Consumer: e
Consumer: r
Consumer:

                    End of simulation
```

Here it can be seen that one by one the correct characters have been accessed from the file, put into the buffer, printed from the buffer to the screen. With the last asterisk not being printed, and having the program end right afterwards.

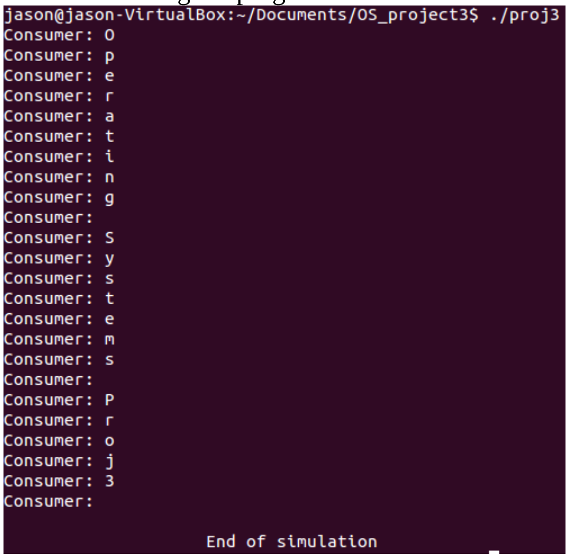
Test 2

Contents of “mytest.dat”:



```
Operating Systems Proj3
```

Result of running the program:



```
jason@jason-VirtualBox:~/Documents/OS_project3$ ./proj3
Consumer: O
Consumer: p
Consumer: e
Consumer: r
Consumer: a
Consumer: t
Consumer: i
Consumer: n
Consumer: g
Consumer:
Consumer: S
Consumer: y
Consumer: s
Consumer: t
Consumer: e
Consumer: m
Consumer: s
Consumer:
Consumer: P
Consumer: r
Consumer: o
Consumer: j
Consumer: 3
Consumer:

                    End of simulation
```

The first test was with a 15 character dat file, the same size as the buffer, for test 2 I used a much longer dat file and the program still executed correctly.

Conclusion

The tests show that the producer and consumer threads were interacting exactly as planned and the asterisk was able to signal that the file had ended when read from the consumer thread in the buffer.

Afterwards I used the IPCS command in the terminal to see if there was any shared memory or semaphore arrays unreleased from the program. There wasn’t any and so I know the shared memory was released correctly.

Overall I see no issues with the program and it seems to be running exactly as planned.