Programming Assignment 3 – Bounded Buffer

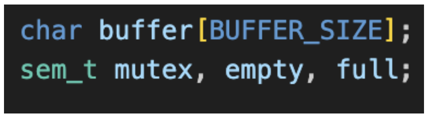
Brianna Patten
Student ID #113632860
University of Oklahoma
brianna.patten@ou.edu

I. Introduction/Project Overview

This project's main objective was for students to learn about circular queues, synchronized mechanisms, semaphore syntax, and thread communication. We were asked to create a circular buffer that read and output the contents of a given file, rotating the characters through the buffer so producer and consumer threads could appropriately communicate. The output would result in the file contents, printed one at a time.

II. Finding a Solution to the Problem

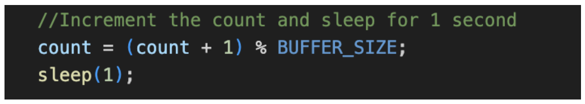
The first problem I ran into was understanding what three semaphores were needed. I knew, based on the project description, that three would be used, but I wasn't sure what for. After some research, I found that a 'mutex' semaphore would be used to provide mutual exclusion to the buffer for threads, the 'full' semaphore would represent the number of filled slots in the buffer that the consumer can read from, and the 'empty' semaphore would represent the number of empty/available slots for the producer to write to.



```c
char buffer[BUFFER_SIZE];
sem_t mutex, empty, full;
```

Figure 1

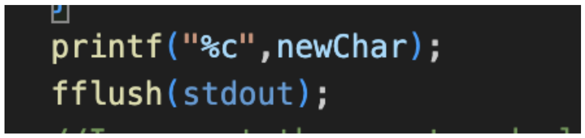
After writing the needed code for the semaphores, I had to figure out what code was needed for the producer and consumer threads. I knew I would need a producer to read from the file and write to the buffer, and a consumer to read from the buffer and print to the user. I began writing code based on previous projects I've done with file management and buffers, and wrote two functions respectively. I kept a counter to know where in the buffer to place the character, which was incremented by 1 and compared to the buffer size (15) to ensure no out-of-bounds errors.



```c
//Increment the count and sleep for 1 second
count = (count + 1) % BUFFER_SIZE;
sleep(1);
```

Figure 2

After writing the rest of the code, I realized that nothing was printing. I attempted to redo the file-related code, but nothing worked. After adding error-handling print statements, I realized the issue was in the consumer function. I added a "flush" that I had previously forgotten, which fixed my issue. After understanding semaphores, producer/consumer threads, and using error-checking, I was able to successfully read from, and print out, the file contents.



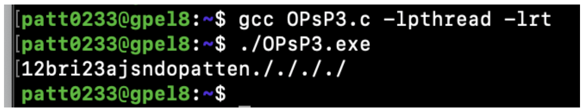
```c
printf("%c",newChar);
fflush(stdout);
```

Figure 3

III. Post-Execution Discussion

The code was able to successfully read, write, and print characters from the "mytest.dat" file, one at a time. Similarly, no characters were skipped or out of place, which was important to note. This showed that both the consumer and producer were successful, and that the memory was accessed as needed.

(In mytest.dat: 12bri23ajsndopatten./././)



```
[patt0233@gpel8:~$ gcc OPsP3.c -lpthread -lrt
[patt0233@gpel8:~$ ./OPsP3.exe
[12bri23ajsndopatten./././
 patt0233@gpel8:~$ _
```

Figure 4

IV. Conclusion

Overall, this project was challenging and a lot different from the first two. It was interesting to learn about consumer/producer threads and how they can interact with semaphores in a program. I look forward to utilizing these skills in future projects.