Assignment 2: Multi-process and multi-threaded print server

Report

<u>Outline</u>

Working Portion

a) Include header files, define constants, and global variables.
b) Define the buffer_t and global_buffer_t structs.
c) Define the insertbuffer() and dequeuebuffer() functions to add and remove buffer_t elements to and from the shared buffer.
d) Define the producer() function, which generates a random number and inserts it into the shared buffer
e) Define the consumer() function, which removes a buffer_t element from the shared buffer and prints its contents, then continue looping until it reads a value of 0.
f) Define the sigint_handler() function, which is called when the user hits Ctrl-C to gracefully terminate the program and clean up any shared memories.
g) In the main() function:
    a. Take in command-line arguments to determine the number of producer and consumer threads to create.
    b. Initialize semaphores and shared memory for the shared buffer.
    c. Create the specified number of producer and consumer threads using the pthread_create() function.
    d. Wait for all threads to finish using the pthread_join() function.
    e. Calculate the total time elapsed during the program's execution using the clock() function.
    f. Print the average time each consumer thread spent waiting for a buffer_t element to be added to the shared buffer.
    g. Clean up any shared memory and unlinked the semaphores that we had pointed to with sem_open.

Limitations/Errors

While my code ran without issues I believe that the average wait time was not functioning correctly since at times it would be bigger than the total execution time. If I had used a different method to get the time I probably could have fixed it and out put the correct numbers but unfortunately I ran out of time.

<u>Explaining/Showing Logic</u>

Logic used to identify the terminating condition.

In the consumer function, I'm using an if statement as my terminating condition for the while loop. The If statement checks to see if the value has been dequeued properly by making sure the value is equal to 0. If it is, then it should break out of the while loop otherwise it should continue running.

How were the semaphores and other book-keeping variables shared between processes and threads?

For shared memory, I used shmget and shmat to allocate the shared memory for my ‘buffer_index’ and ‘my_buffer’ which is my global buffer/variable.

For semaphores I’m using sem_open to return a pointer to my three semaphores full_sem, empty_sem, and buffer_mutex. I have two counting semaphores(full_sem, and empty_sem) and one binary semaphore (buffer_mutex). In the actual code when using sem_open, I added an extra letter or two in the end to ensure that no one else is using the same name when using sem_open because earlier throughout the assignment I got an error code saying permission denied for my semaphore but thankfully changing the name fixed it.

I used the semaphores for synchronization of processes and thread and the shared memory to store certain variables that can be shared by the threads.

What was done in the signal handler for graceful termination?

For my signal handler function, I have several actions that are used to clean up resources and terminate the process. First, I have a sem_unlink() function that is called three times to remove the three semaphores that the sem_open function returns a pointer to. Second, is the shmctl() function to delete the shared memory segment for shmid4 and shmid5 which is used for the ‘my_buffer’ structure and my ‘buffer_index’.

I do not have anything in the signal handler function that kills the processes but, since I’m using exit(0) I didn’t need to kill the processes in my signal handler.

How did the LIFO and FIFO implementations differ in terms of your usage of either the buffer_index variable or in/out pointers (or some other method that you may use for the FIFO queue).

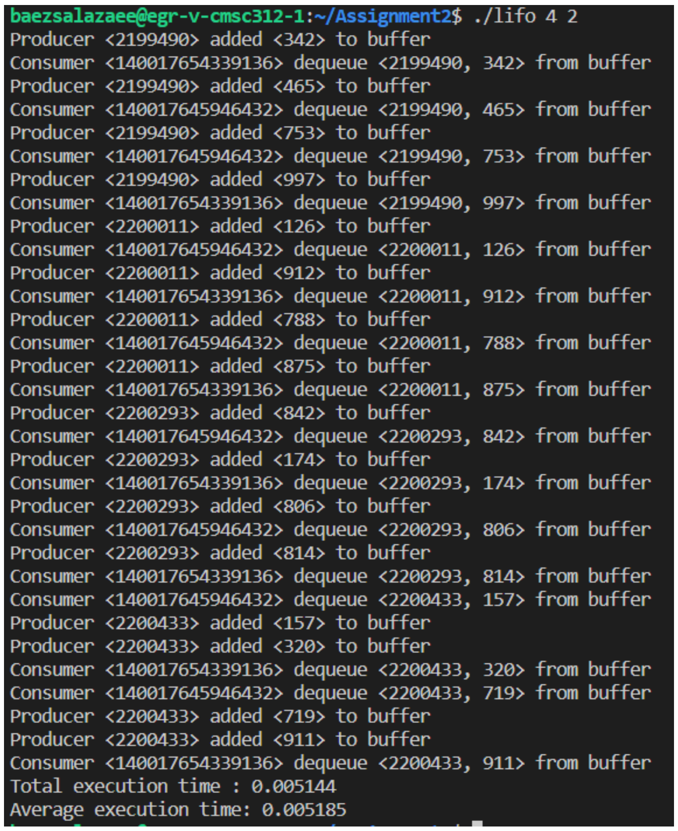
The only changes I made between my LIFO and FIFO implementations are in the insertbuffer() and dequeuebuffer() functions.

For LIFO I made use of the buffer_index in the same way as the original code that was provided but instead of incrementing it like the original code (buffer[buffer_index++]), I had to add it onto the next line for it to run properly.

For FIFO I removed the ‘buffer_index’ and instead I used in and out variables that are in my global_buffer_t structure. The in and out variables are used to keep track of the read and write positions in the buffer. The % operator ensures that the buffer works as a circular buffer so that the write position always goes back around to the beginning of the buffer once it reaches the end.

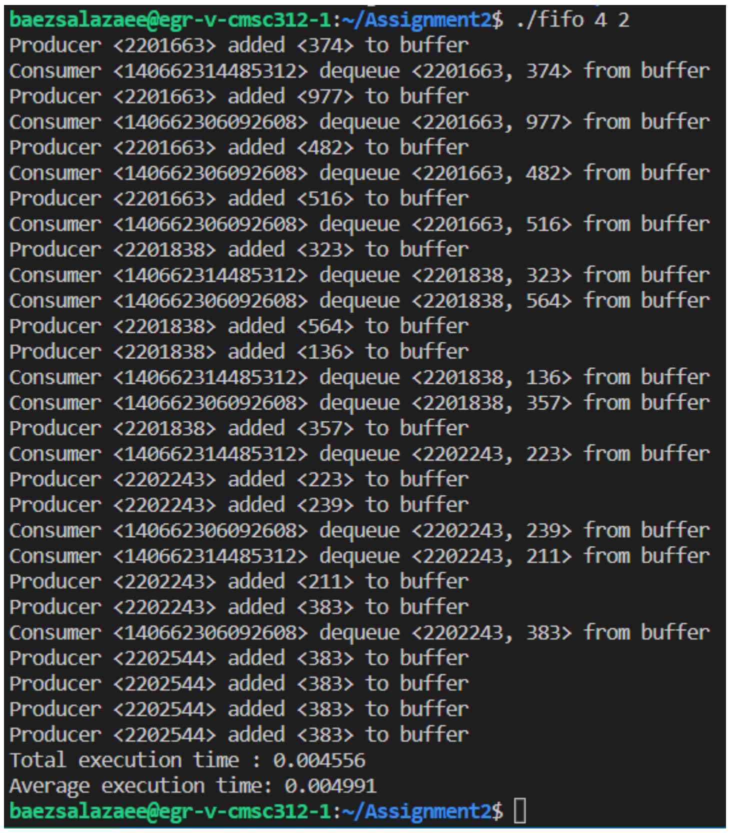
<u>Sample run of code</u>

LIFO



```
baezsalazaee@egr-v-cmsc312-1:~/Assignment2$ ./lifo 4 2
Producer <2199490> added <342> to buffer
Consumer <140017654339136> dequeue <2199490, 342> from buffer
Producer <2199490> added <465> to buffer
Consumer <140017645946432> dequeue <2199490, 465> from buffer
Producer <2199490> added <753> to buffer
Consumer <140017645946432> dequeue <2199490, 753> from buffer
Producer <2199490> added <997> to buffer
Consumer <140017654339136> dequeue <2199490, 997> from buffer
Producer <2200011> added <126> to buffer
Consumer <140017645946432> dequeue <2200011, 126> from buffer
Producer <2200011> added <912> to buffer
Consumer <140017654339136> dequeue <2200011, 912> from buffer
Producer <2200011> added <788> to buffer
Consumer <140017645946432> dequeue <2200011, 788> from buffer
Producer <2200011> added <875> to buffer
Consumer <140017654339136> dequeue <2200011, 875> from buffer
Producer <2200293> added <842> to buffer
Consumer <140017645946432> dequeue <2200293, 842> from buffer
Producer <2200293> added <174> to buffer
Consumer <140017654339136> dequeue <2200293, 174> from buffer
Producer <2200293> added <806> to buffer
Consumer <140017645946432> dequeue <2200293, 806> from buffer
Producer <2200293> added <814> to buffer
Consumer <140017654339136> dequeue <2200293, 814> from buffer
Consumer <140017645946432> dequeue <2200433, 157> from buffer
Producer <2200433> added <157> to buffer
Producer <2200433> added <320> to buffer
Consumer <140017654339136> dequeue <2200433, 320> from buffer
Consumer <140017645946432> dequeue <2200433, 719> from buffer
Producer <2200433> added <719> to buffer
Producer <2200433> added <911> to buffer
Consumer <140017654339136> dequeue <2200433, 911> from buffer
Total execution time : 0.005144
Average execution time: 0.005185
```

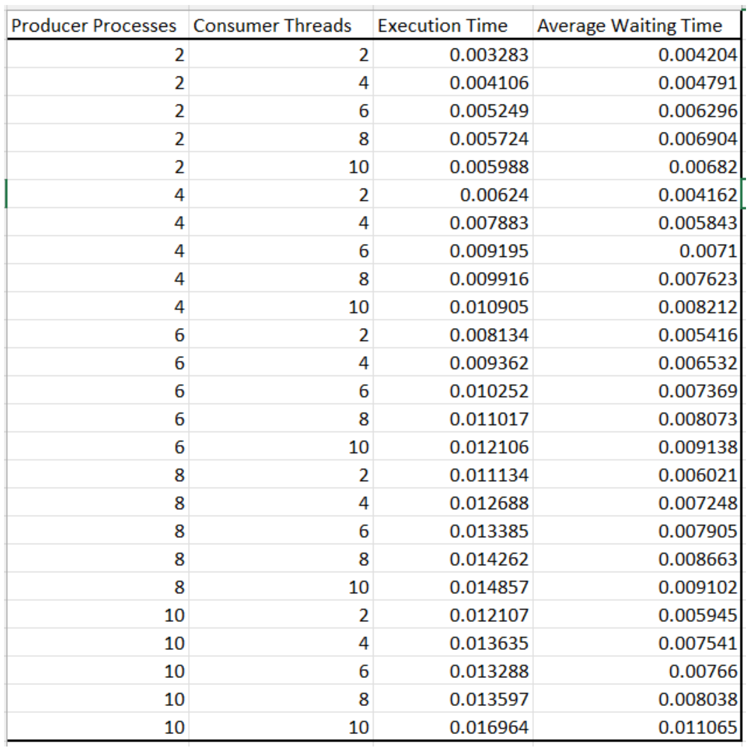
FIFO



```
baezsalazaee@egr-v-cmsc312-1:~/Assignment2$ ./fifo 4 2
Producer <2201663> added <374> to buffer
Consumer <140662314485312> dequeue <2201663, 374> from buffer
Producer <2201663> added <977> to buffer
Consumer <140662306092608> dequeue <2201663, 977> from buffer
Producer <2201663> added <482> to buffer
Consumer <140662306092608> dequeue <2201663, 482> from buffer
Producer <2201663> added <516> to buffer
Consumer <140662306092608> dequeue <2201663, 516> from buffer
Producer <2201838> added <323> to buffer
Consumer <140662314485312> dequeue <2201838, 323> from buffer
Consumer <140662306092608> dequeue <2201838, 564> from buffer
Producer <2201838> added <564> to buffer
Producer <2201838> added <136> to buffer
Consumer <140662314485312> dequeue <2201838, 136> from buffer
Consumer <140662306092608> dequeue <2201838, 357> from buffer
Producer <2201838> added <357> to buffer
Consumer <140662314485312> dequeue <2202243, 223> from buffer
Producer <2202243> added <223> to buffer
Producer <2202243> added <239> to buffer
Consumer <140662306092608> dequeue <2202243, 239> from buffer
Consumer <140662314485312> dequeue <2202243, 211> from buffer
Producer <2202243> added <211> to buffer
Producer <2202243> added <383> to buffer
Consumer <140662306092608> dequeue <2202243, 383> from buffer
Producer <2202544> added <383> to buffer
Producer <2202544> added <383> to buffer
Producer <2202544> added <383> to buffer
Producer <2202544> added <383> to buffer
Total execution time : 0.004556
Average execution time: 0.004991
baezsalazaee@egr-v-cmsc312-1:~/Assignment2$
```

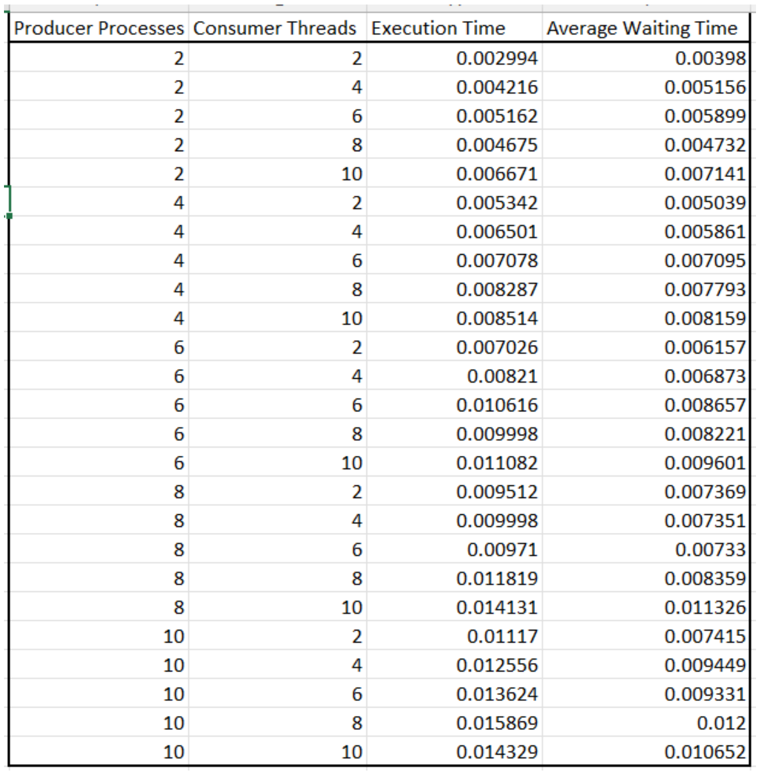
Plot/Table for runtime

LIFO (execution time in milliseconds)



Producer Processes	Consumer Threads	Execution Time	Average Waiting Time
2	2	0.003283	0.004204
2	4	0.004106	0.004791
2	6	0.005249	0.006296
2	8	0.005724	0.006904
2	10	0.005988	0.00682
4	2	0.00624	0.004162
4	4	0.007883	0.005843
4	6	0.009195	0.0071
4	8	0.009916	0.007623
4	10	0.010905	0.008212
6	2	0.008134	0.005416
6	4	0.009362	0.006532
6	6	0.010252	0.007369
6	8	0.011017	0.008073
6	10	0.012106	0.009138
8	2	0.011134	0.006021
8	4	0.012688	0.007248
8	6	0.013385	0.007905
8	8	0.014262	0.008663
8	10	0.014857	0.009102
10	2	0.012107	0.005945
10	4	0.013635	0.007541
10	6	0.013288	0.00766
10	8	0.013597	0.008038
10	10	0.016964	0.011065

FIFO (execution time in milliseconds)



Producer Processes	Consumer Threads	Execution Time	Average Waiting Time
2	2	0.002994	0.00398
2	4	0.004216	0.005156
2	6	0.005162	0.005899
2	8	0.004675	0.004732
2	10	0.006671	0.007141
4	2	0.005342	0.005039
4	4	0.006501	0.005861
4	6	0.007078	0.007095
4	8	0.008287	0.007793
4	10	0.008514	0.008159
6	2	0.007026	0.006157
6	4	0.00821	0.006873
6	6	0.010616	0.008657
6	8	0.009998	0.008221
6	10	0.011082	0.009601
8	2	0.009512	0.007369
8	4	0.009998	0.007351
8	6	0.00971	0.00733
8	8	0.011819	0.008359
8	10	0.014131	0.011326
10	2	0.01117	0.007415
10	4	0.012556	0.009449
10	6	0.013624	0.009331
10	8	0.015869	0.012
10	10	0.014329	0.010652

2-d Graph

*****My graphs are on the excel file that has been attached with the submission. Was having some issues placing the graphs in the word doc*****