Faculty of Engineering & Technology
Electrical & Computer Engineering Department

OPERATING SYSTEMS – ENCS3390

Report of Assignment #1

Shared Memory

Prepared by:

Hatem Hussein	1200894
Momen Salem	1200034

Instructor: Dr. Bashar Tahayna

Section: 4

Date: 8-05-2023

Abstract/Objectives

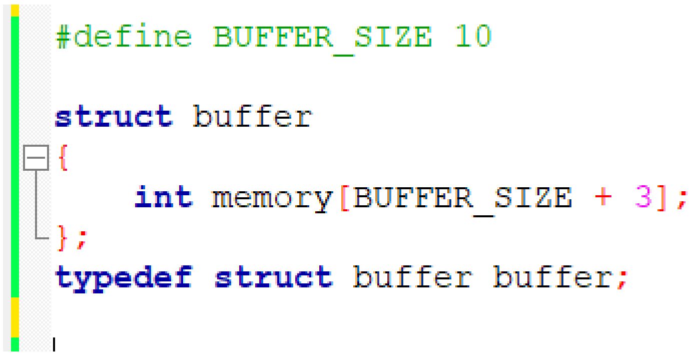
The aim of the assignment is to get to know more about the shared memory and how we can deal with it in the LINUX operating system by using C programming language. In addition, it aims to find a solution to the bounded buffer producer-consumer problem by using some functions such as fork, exec, and shared memory map.

Problem solution and discussion

We had a lot of problems at first we use virtual machine and download platform for c language after this we include many libraries where used in program. Finally, we had a big problem for resuming consumer after pausing it.

- *The buffer struct*

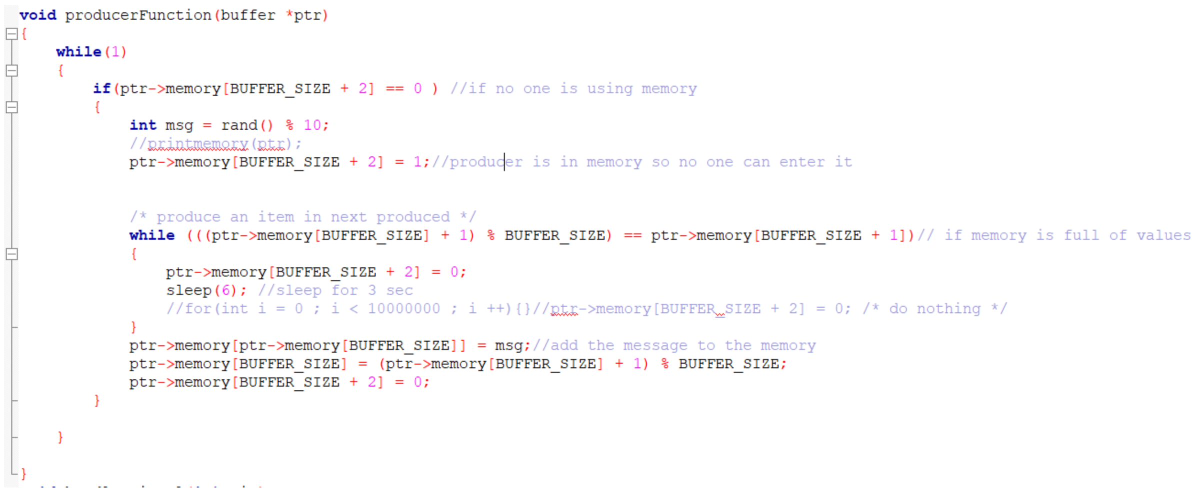
To solve the problem, we first start with structuring the buffer struct that has an int array inside of it representing the bounded buffer we want to deal with (producing and consuming). Moreover, as you can see from the below fragment of code, we defined a constant size called BUFFER_SIZE with value of 10 as requested. In the struct we added to the buffer size 3 additional indices so we can use them as in, out, and active flags (we prefer to keep using just array for simplicity). The in flag represents the next index where the buffer can be produced on. The out flag represents the index where the buffer can consume next. The active flag represents if the memory is currently used or not.



```c
#define BUFFER_SIZE 10

struct buffer
{
    int memory[BUFFER_SIZE + 3];
};
typedef struct buffer buffer;
```

- *The producer function*

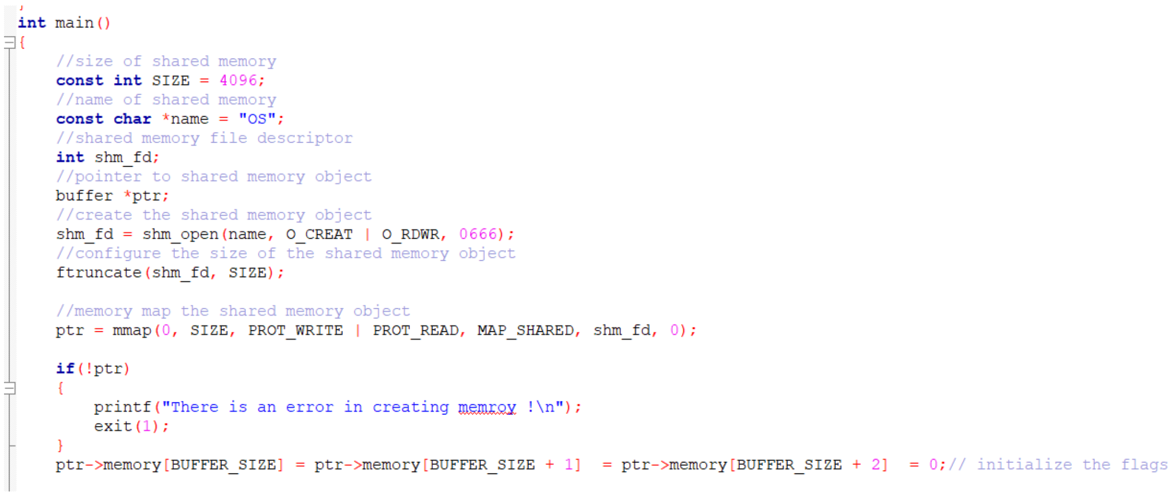
The producer function its main role is to produce processes on the shared memory. It takes a pointer of type buffer ‘struct’ as a parameter. The function keeps looping infinitely and each time asking if the memory is used or not. If the memory isn’t used it writes a random number got from the rand() built-in function. In addition, if the memory is full the active flag updated with zero and makes some delay time by using the sleep() function. If all the conditions are satisfied, the producer adds the message to the memory with updating the in flag and the active flag with zero so any one now can use the memory.



```
void producerFunction(buffer *ptr)
{
    while(1)
    {
        if(ptr->memory[BUFFER_SIZE + 2] == 0 ) //if no one is using memory
        {
            int msg = rand() % 10;
            //printmemory(ptr);
            ptr->memory[BUFFER_SIZE + 2] = 1;//producer is in memory so no one can enter it


            /* produce an item in next produced */
            while (((ptr->memory[BUFFER_SIZE] + 1) % BUFFER_SIZE) == ptr->memory[BUFFER_SIZE + 1])// if memory is full of values
            {
                ptr->memory[BUFFER_SIZE + 2] = 0;
                sleep(6); //sleep for 3 sec
                //for(int i = 0 ; i < 10000000 ; i ++){}//ptr->memory[BUFFER_SIZE + 2] = 0; /* do nothing */
            }
            ptr->memory[ptr->memory[BUFFER_SIZE]] = msg;//add the message to the memory
            ptr->memory[BUFFER_SIZE] = (ptr->memory[BUFFER_SIZE] + 1) % BUFFER_SIZE;
            ptr->memory[BUFFER_SIZE + 2] = 0;
        }

    }

}
```

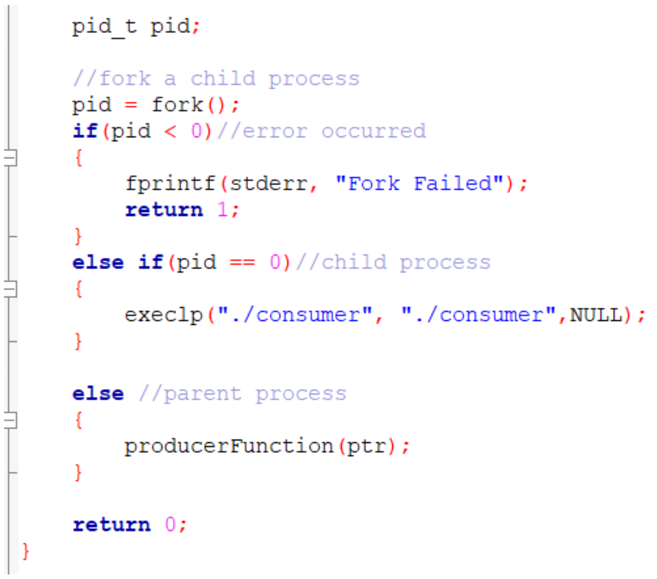
- *The main function*

Here the program starts by defining the shared memory and opening it by using the function shm_open(). Then, we used the function ftruncate() and mmap() to specify the area of the shared memory. Moreover, we initialized all the flags with zeros at first.



```
int main()
{
    //size of shared memory
    const int SIZE = 4096;
    //name of shared memory
    const char *name = "OS";
    //shared memory file descriptor
    int shm_fd;
    //pointer to shared memory object
    buffer *ptr;
    //create the shared memory object
    shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);
    //configure the size of the shared memory object
    ftruncate(shm_fd, SIZE);

    //memory map the shared memory object
    ptr = mmap(0, SIZE, PROT_WRITE | PROT_READ, MAP_SHARED, shm_fd, 0);

    if(!ptr)
    {
        printf("There is an error in creating memroy !\n");
        exit(1);
    }
    ptr->memory[BUFFER_SIZE] = ptr->memory[BUFFER_SIZE + 1]  = ptr->memory[BUFFER_SIZE + 2]  = 0;// initialize the flags
```

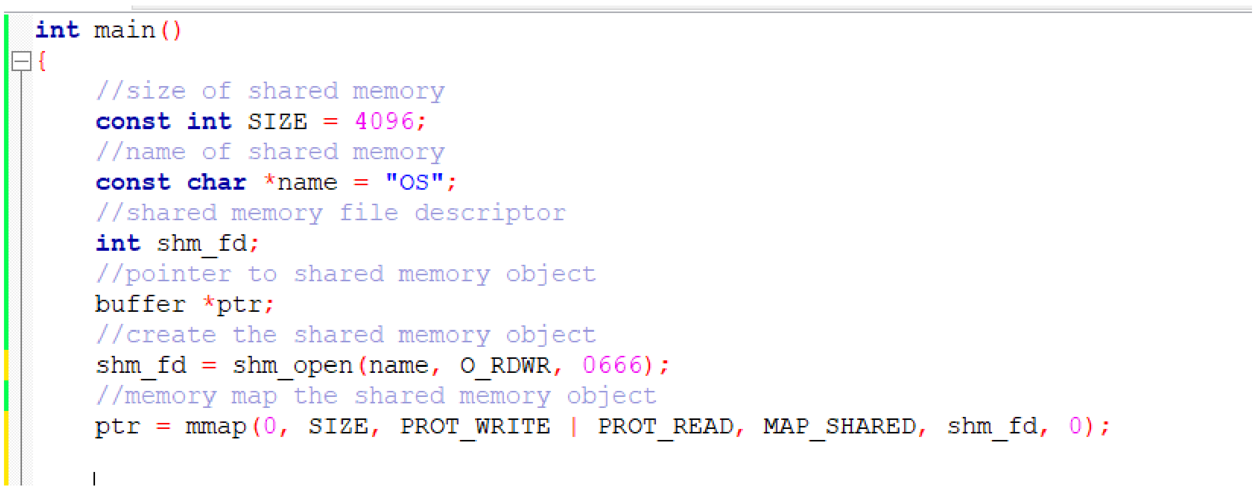
Then, we defined a process id and created a new child that is a duplicate from the parent. If the process id was less than zero prints an error message, and if the process id was zero "representing the child id" call the consumer from another file by using execlp() function. On the other hand, if the process id was more than zero call the producer and continue the parent process.



```
    pid_t pid;

    //fork a child process
    pid = fork();
    if(pid < 0)//error occurred
    {
        fprintf(stderr, "Fork Failed");
        return 1;
    }
    else if(pid == 0)//child process
    {
        execlp("./consumer", "./consumer",NULL);
    }

    else //parent process
    {
        producerFunction(ptr);
    }

    return 0;
}
```

- *<u>The consumer file</u>*

Firstly, we defined the same struct we defined in the producer file so it's shared among the two files. Moreover, we defined the same shared memory as in the producer but without calling the ftruncate() function.



```
int main()
{
    //size of shared memory
    const int SIZE = 4096;
    //name of shared memory
    const char *name = "OS";
    //shared memory file descriptor
    int shm_fd;
    //pointer to shared memory object
    buffer *ptr;
    //create the shared memory object
    shm_fd = shm_open(name, O_RDWR, 0666);
    //memory map the shared memory object
    ptr = mmap(0, SIZE, PROT_WRITE | PROT_READ, MAP_SHARED, shm_fd, 0);
```

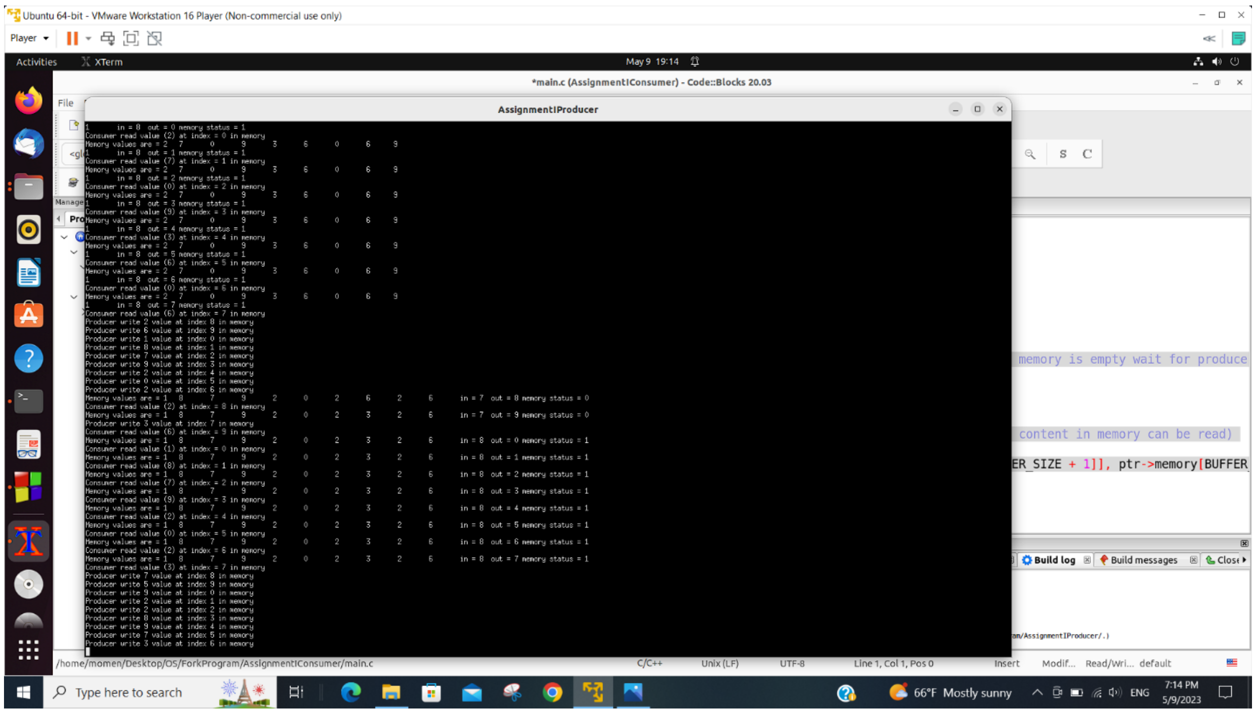
The consumer function represented by the below fragment of code. The consumer checks firstly if the active flag is zero so the memory is not used. In addition, if the first condition satisfied it checks if the in flag equals the out flag so the buffer is empty and it makes the active flag zero and calls sleep() function. In short, if the memory is not used by anyone and the buffer is not empty so the consumer works and reads from the memory with updating the out flag and the active flag with zero. Finally, we unlink the shared memory by calling the shm_unlink().

```
while(1)
{

    if(ptr->memory[BUFFER_SIZE + 2] == 0)
    {
        printmemory(ptr);
        ptr->memory[BUFFER_SIZE + 2] = 1;
        /* produce an item in next produced */
        while (ptr->memory[BUFFER_SIZE] == ptr->memory[BUFFER_SIZE + 1])
        {
            ptr->memory[BUFFER_SIZE + 2] = 0;
            sleep(1);
        }

        ptr->memory[BUFFER_SIZE + 1] = (ptr->memory[BUFFER_SIZE + 1] + 1) % BUFFER_SIZE;
        printf("read index = %d\t", ptr->memory[BUFFER_SIZE + 1]);
        printf("consumer value = %d\n", ptr->memory[ptr->memory[BUFFER_SIZE + 1]]);
        ptr->memory[BUFFER_SIZE + 2] = 0;

    }
}


//remove the shared memory object
shm_unlink(name);

return 0;
```

Some test cases :

1- When buffer is full

The producer will wait until the consumer read data as bellow figure:

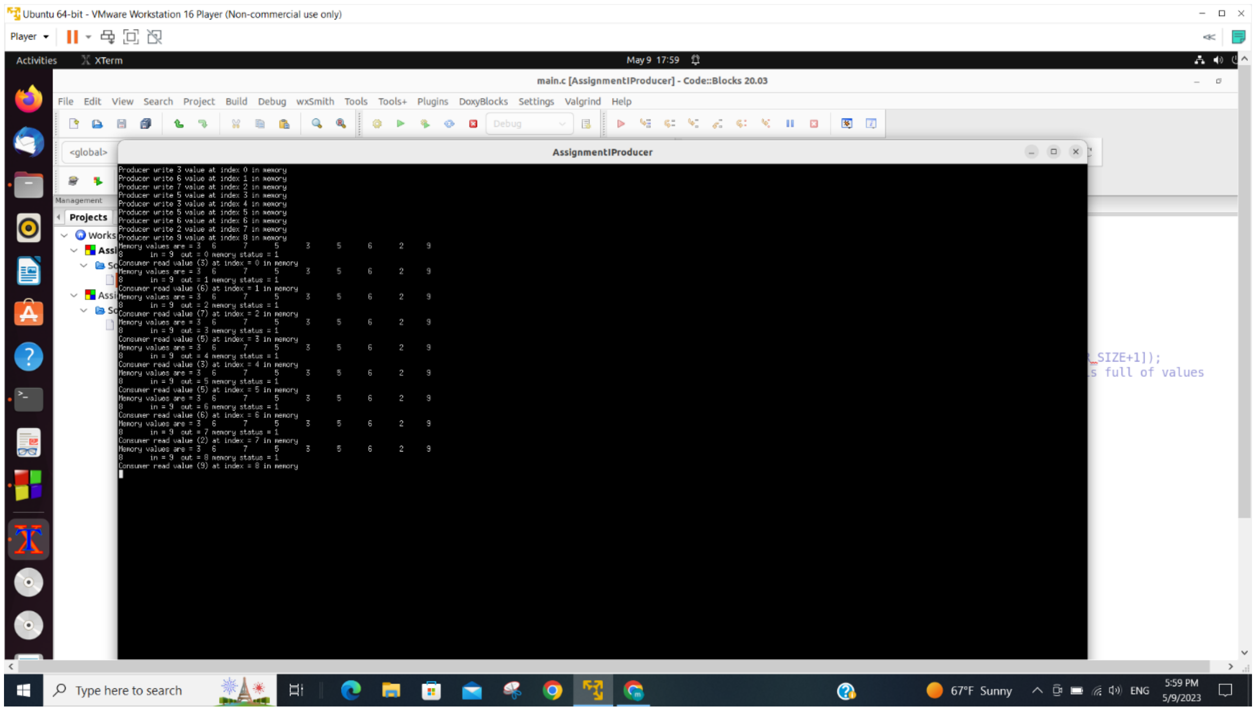


```
1       in = 8  out = 0 memory status = 1
Consumer read value (2) at index = 0 in memory
Memory values are = 2   7       0       9       3       6       0       6       9
1       in = 8  out = 1 memory status = 1
Consumer read value (7) at index = 1 in memory
Memory values are = 2   7       0       9       3       6       0       6       9
1       in = 8  out = 2 memory status = 1
Consumer read value (0) at index = 2 in memory
Memory values are = 2   7       0       9       3       6       0       6       9
1       in = 8  out = 3 memory status = 1
Consumer read value (9) at index = 3 in memory
Memory values are = 2   7       0       9       3       6       0       6       9
1       in = 8  out = 4 memory status = 1
Consumer read value (3) at index = 4 in memory
Memory values are = 2   7       0       9       3       6       0       6       9
1       in = 8  out = 5 memory status = 1
Consumer read value (6) at index = 5 in memory
Memory values are = 2   7       0       9       3       6       0       6       9
1       in = 8  out = 6 memory status = 1
Consumer read value (0) at index = 6 in memory
Memory values are = 2   7       0       9       3       6       0       6       9
1       in = 8  out = 7 memory status = 1
Consumer read value (6) at index = 7 in memory
Producer write 2 value at index 8 in memory
Producer write 6 value at index 9 in memory
Producer write 1 value at index 0 in memory
Producer write 8 value at index 1 in memory
Producer write 7 value at index 2 in memory
Producer write 9 value at index 3 in memory
Producer write 2 value at index 4 in memory
Producer write 0 value at index 5 in memory
Producer write 2 value at index 6 in memory
Memory values are = 1   8       7       9       2       0       2       6       2       6       in = 7  out = 8 memory status = 0
Consumer read value (2) at index = 8 in memory
Memory values are = 1   8       7       9       2       0       2       3       2       6       in = 7  out = 9 memory status = 0
Producer write 3 value at index 7 in memory
Consumer read value (6) at index = 9 in memory
Memory values are = 1   8       7       9       2       0       2       3       2       6       in = 8  out = 0 memory status = 1
Consumer read value (1) at index = 0 in memory
Memory values are = 1   8       7       9       2       0       2       3       2       6       in = 8  out = 1 memory status = 1
Consumer read value (8) at index = 1 in memory
Memory values are = 1   8       7       9       2       0       2       3       2       6       in = 8  out = 2 memory status = 1
Consumer read value (7) at index = 2 in memory
Memory values are = 1   8       7       9       2       0       2       3       2       6       in = 8  out = 3 memory status = 1
Consumer read value (9) at index = 3 in memory
Memory values are = 1   8       7       9       2       0       2       3       2       6       in = 8  out = 4 memory status = 1
Consumer read value (2) at index = 4 in memory
Memory values are = 1   8       7       9       2       0       2       3       2       6       in = 8  out = 5 memory status = 1
Consumer read value (0) at index = 5 in memory
Memory values are = 1   8       7       9       2       0       2       3       2       6       in = 8  out = 6 memory status = 1
Consumer read value (2) at index = 6 in memory
Memory values are = 1   8       7       9       2       0       2       3       2       6       in = 8  out = 7 memory status = 1
Consumer read value (3) at index = 7 in memory
Producer write 7 value at index 8 in memory
Producer write 5 value at index 9 in memory
Producer write 9 value at index 0 in memory
Producer write 2 value at index 1 in memory
Producer write 2 value at index 2 in memory
Producer write 8 value at index 3 in memory
Producer write 9 value at index 4 in memory
Producer write 7 value at index 5 in memory
Producer write 3 value at index 6 in memory
```

```
 memory is empty wait for produce
 content in memory can be read)
ER_SIZE + 1]], ptr->memory[BUFFER
```

2- When buffer is empty

The consumer will wait until producer write new data as bellow figure.:



```
Producer write 3 value at index 0 in memory
Producer write 6 value at index 1 in memory
Producer write 7 value at index 2 in memory
Producer write 5 value at index 3 in memory
Producer write 3 value at index 4 in memory
Producer write 5 value at index 5 in memory
Producer write 6 value at index 6 in memory
Producer write 2 value at index 7 in memory
Producer write 9 value at index 8 in memory
Memory values are = 3   6       7       5       3       5       6       2       9
8       in = 9  out = 0 memory status = 1
Consumer read value (3) at index = 0 in memory
Memory values are = 3   6       7       5       3       5       6       2       9
8       in = 9  out = 1 memory status = 1
Consumer read value (6) at index = 1 in memory
Memory values are = 3   6       7       5       3       5       6       2       9
8       in = 9  out = 2 memory status = 1
Consumer read value (7) at index = 2 in memory
Memory values are = 3   6       7       5       3       5       6       2       9
8       in = 9  out = 3 memory status = 1
Consumer read value (5) at index = 3 in memory
Memory values are = 3   6       7       5       3       5       6       2       9
8       in = 9  out = 4 memory status = 1
Consumer read value (3) at index = 4 in memory
Memory values are = 3   6       7       5       3       5       6       2       9
8       in = 9  out = 5 memory status = 1
Consumer read value (5) at index = 5 in memory
Memory values are = 3   6       7       5       3       5       6       2       9
8       in = 9  out = 6 memory status = 1
Consumer read value (6) at index = 6 in memory
Memory values are = 3   6       7       5       3       5       6       2       9
8       in = 9  out = 7 memory status = 1
Consumer read value (2) at index = 7 in memory
Memory values are = 3   6       7       5       3       5       6       2       9
8       in = 9  out = 8 memory status = 1
Consumer read value (9) at index = 8 in memory
```

So from these two cases our program is working properlly.

Link for our program execution in google drive:

https://drive.google.com/file/d/1M2nofEDYd9jLqRgjR_ry-m0k3RwVr-Ys/view?fbclid=IwAR23jJIMUEMFkJVO2XXV1Ihe3FViu_-JLc_bcwJkzPbOAyNJHF_xqEW8Ey0