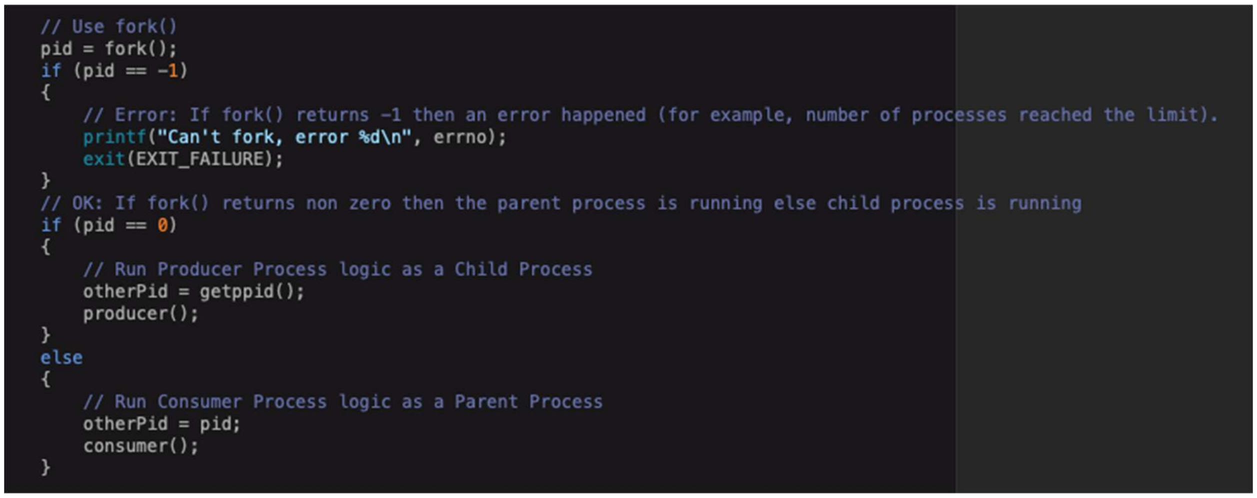
CST–321 Producer and Consumer

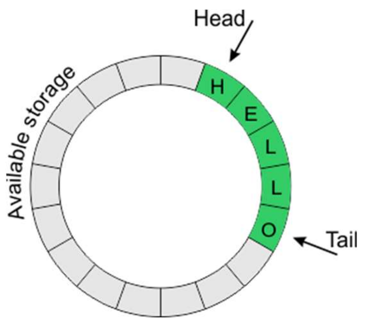
Activity Directions:

In this project, you will demonstrate your ability to create and manage processes. You will create two processes, *Producer* and *Consumer*, using the C functions, as shown in the code snippet below:

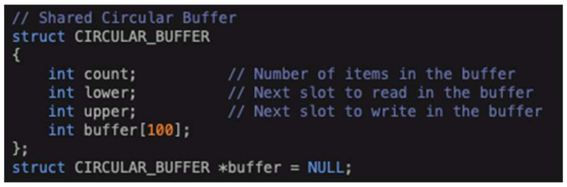


```
// Use fork()
pid = fork();
if (pid == -1)
{
    // Error: If fork() returns -1 then an error happened (for example, number of processes reached the limit).
    printf("Can't fork, error %d\n", errno);
    exit(EXIT_FAILURE);
}
// OK: If fork() returns non zero then the parent process is running else child process is running
if (pid == 0)
{
    // Run Producer Process logic as a Child Process
    otherPid = getppid();
    producer();
}
else
{
    // Run Consumer Process logic as a Parent Process
    otherPid = pid;
    consumer();
}
```

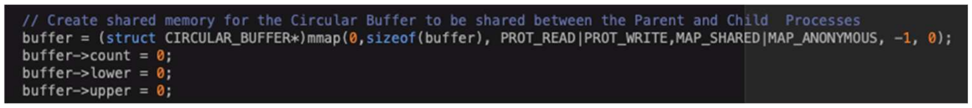
The *Producer* will be responsible for simply creating numbers, which are "passed" to the *Consumer* via a shared circular buffer. Each element of the circular buffer should hold a number. A good article on the circular buffer data structure can be found on Wikipedia at https://en.wikipedia.org/wiki/Circular_buffer and also illustrated in the following diagram.



The circular buffer should be placed in shared memory so that each process can access the buffer to support inter-process communication. The following code snippets can be used as guidance for implementing a shared memory buffer.

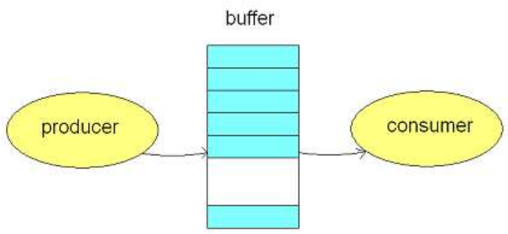


```c
// Shared Circular Buffer
struct CIRCULAR_BUFFER
{
    int count;          // Number of items in the buffer
    int lower;          // Next slot to read in the buffer
    int upper;          // Next slot to write in the buffer
    int buffer[100];
};
struct CIRCULAR_BUFFER *buffer = NULL;
```



```c
// Create shared memory for the Circular Buffer to be shared between the Parent and Child  Processes
buffer = (struct CIRCULAR_BUFFER*)mmap(0,sizeof(buffer), PROT_READ|PROT_WRITE,MAP_SHARED|MAP_ANONYMOUS, -1, 0);
buffer->count = 0;
buffer->lower = 0;
buffer->upper = 0;
```

Your objective is to implement the shared circular buffer along with *put()* and *get()* functions to access the buffer. Each process function (i.e., *producer()* and *consumer()*) will access the circular buffer where your objective is to ensure that the producer is always ahead of the consumer and the consumer does not have to wait. This is illustrated in the following diagram.



The use of signals should be used between the two processes. The consumer should put itself to sleep if there is no data in the buffer and wait for a signal from the producer indicating data has been placed in the buffer. The producer should put itself to sleep if the buffer is full and wait for a signal from the consumer that there is room in the buffer to start writing again.

<u>Deliverables</u>:

1. Cover sheet with your name, the name of this assignment, and the date.
2. An explanation of your approach to implementation and its reasoning.
3. Coding results:
    a. All the source code files. Comment your code, describing the solution and identifying the programmer. Zip up the source code (not the binaries) in a single zip file.
    b. A 3- to 5-minute screencast showing successful execution of the program and explanation of the code.
4. Package the URL for the screencast and description of the solution into a one document and upload it to the digital classroom.
5. Zip up the source code (not the binaries) in a single zip file and upload it to the digital classroom.