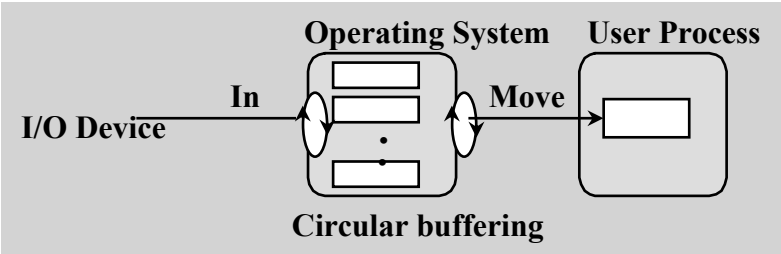
SYSC 4001 Operating Systems, Fall 2022
Assignment 2
Due: Tuesday Nov. 8th 12pm

Objective: The objective is to implement the finite circular buffer for the producer/consumer problem (PC) by using Unix/Linux shared memory and semaphores.

Assignment description:

You need to understand the PC problem discussed in class and the programming details – semaphores and shared memory – for Linux. There are two sets of interface functions for semaphores: One is System V semaphores (described in Chapter 14 of the Linux book and used for Lab 7), which are commonly used for process synchronization; the other set is taken from POSIX Real-time Extensions initially used for threads and has been extended to processes (described in Chapter 12 Threads of the Linux book). For this assignment, System V semaphores (described in Chapter 14) are to be used.

The PC problem is commonly used in practice, e.g., I/O buffering, as shown in the figure in Chapter 11 of the textbook:



The following describes the tasks for the assignment.

A. Implementation of the PC problem

The algorithm for the PC problem is presented in Chapter 5 *Concurrency Control*. The first task is a straight implementation of the algorithm. The key components are shared memory, producer, and consumer. Producer and consumer are two independent programs; there is no parent-child relationship between them. The main components are described as follows:

Shared memory: The shared memory contains 100 buffers (or NUM_BUFFER), each having 128 bytes (or BUFFER_SIZE), i.e., a string of 128 characters, and an integer member count. In other words, each buffer is a structure that contains a string and a count (the number of bytes stored in the buffer). Each time the producer writes text to a shared buffer, it also writes the number of bytes to the count member for reliability purpose.

Producer: The producer will repeatedly read text from a file and write the text it just read into one of the buffers in shared memory until the end of file is reached. The basic functions for reading from and writing to a file are read() and write(). See Chapter 3 of the Linux book for more description on read() and write() operations and examples.

When the producer reads text from a file into a input buffer, use the default buffer size, i.e., BUFSIZ for the read() function. For instance,

```
len = read (input_fd, &input_buffer, BUFSIZ);  // Should add error handling for read()
```

Note that BUFSIZ is defined in stdio.h. Its value may be platform dependent.

Before writing to each buffer in the shared memory, the producer divides the text stored in the input buffer into smaller segments, as each buffer in the shared memory only has 128 bytes which is smaller than BUFSIZ.

When the end of file is reached, the producer also prints the total number of bytes written to the shared memory.

Consumer: The consumer will read from buffers in the shared memory and print the text into an output file.

At the end, the consumer calculates the total number of bytes read from the producer and prints the value. This value should be identical to the one printed from the producer.

B. Analysis and verification of the PC problem without one of the semaphores

The second part of the assignment is to verify if indeed all three semaphores, S, N, and E, are needed for a scenario where there are only one producer and one consumer (*1P1C*).

Walk through the PC algorithm carefully by hand first. Make sure you fully understand the algorithm. Next, **identify one of the three semaphores that can be removed**. Verify if the solution can still work correctly. Find the reason behind it. What could be the benefit of the solution?

You may need to evaluate three semaphores one at a time if needed.

Consider other scenarios: what if there are multiple customers, i.e., one producer and multiple consumers (*1PmC*)? Will the proposed solution still work? What may be the problem? What needs to be changed without affecting (or changing) the producer process?

Similarly, if there are multiple producers and one single consumer (*mP1C*), what is the solution without changing the customer processes?

Lastly, consider the scenario where there are multiple producers and multiple consumers (*mPmC*), what solution can be used to isolate producers and consumers? In other words, between the producer and consumer processes, there is no sharing of the semaphore that was identified.

Write a concise summary (limited to one page) of your solution to various scenarios. It is part of the assignment submission.

After your verification, test it out by commenting out the semaphore you identified for *1P1C* and run your code multiple times. Verify the correctness of your solution.

Optional: If you are interested, you could verify other alternatives, e.g., *1PmC*, *mP1C*, and *mPmC*, and even compare the performance of different options. It would be easier to simply generate distinct numbers for multiple producers.

Implementation issues:

1. If a producer terminates early, the semaphores it has allocated may be released. To avoid this issue, keep the producer alive using sleep() or replace SEM_UNDO with 0 in semaphore operations.

2. There are different ways to inform the consumer that the end is reached. You may consider using a special text, e.g., end, and put it in a separate buffer. But the special text needs to be unique. Another way is to send the total byte count of the file to the consumer. The consumer keeps updating the byte count read so far and stops when the byte count reaches the file size. Two models may be considered for finding out the size of a file:

```
fseek(fd, 0, SEEK_END); // seek to end of file, fd is the file descriptor obtained from fopen()
size = ftell(fd);       // get current file pointer
fseek(fd, 0, SEEK_SET); // seek back to beginning of file
                        // proceed with allocating memory and reading the file

#include <sys/stat.h>
struct stat st;         // The stat() function shall obtain information about the named file and
                        // write it to the area pointed to by the buf argument.
stat(filename, &st);    // Or fstat(fd, &st); if file descriptor is available
size = st.st_size;
```

Grading criteria:

Marking is based on the following criteria:

Correctness (including error checking, message labeling, output): 80%
Documentation: 10%
Program structure: 5%
Style and readability: 5%

Additional notes:

Finally, as we know that customer requirements are constantly changing in practice, I also reserve the right to make changes (only minor ones) to the assignment, but the requirements will be finalized as least one week before the due date.