Using shared memory with semaphores for IPC
Lab. #6

Arif A. Balik

Undergraduate Student
Sytstems Programming
Department of Computer Science
Arel University
Büyükçekmece, İstanbul 34537
Email: arifbalik@outlook.com

May 17, 2019

Abstract

This reports covers some properties of IPC, shared memory and semaphores. It explains its theory of operation and answers some questions asked in the lab sheet. This report is powered by LaTeX

Theory of Operation

There is a main process, it initializes a shared memory, two semaphores and two child process, two child process then overlays themselves with **producer.c** and **consumer.c** respectively. Then producer program after locking the semaphore, fills an array and sets current position of array, then unlocks the semaphore and waits for consumer program to print the message. This process repeats given amount of time, then both process exits and parent program destroys both shared memory and semaphores.

Q&A

Q1. *What is the purpose of this lab work?*

To understand shared memory concept and semaphores thus IPC.

Q2. *Describe the operation of a producer-consumer system in general*

There is a shared memory for both consumer and producer programs. Both of these programs tries to access this memory concurrently, producer puts given amount of text in memory and consumer prints them, and they collaborate with semaphores to avoid race condition.

Q3. *How many processes do implement a producer-consumer system in this lab work?*

There are three processes, one is the main process and other two are producer and consumer.

Q4. *What are the roles (duties) of each of the processes in this lab work?*

The cons_prod_parent process creates producer and consumer as well as the shared memory and two semaphores. Producer attaches the shared memory and semaphores (so does the consumer) and tries to fill the shared memory with some messages, and consumer prints those messages. Finally cons_prod_parent detaches the shared memory and semaphores after waiting the termination of both processes.

Q5. *Will the producer-consumer system in this lab work, operate normally if the parent process initially creates the producer and the consumer processes, and only after that it will create shared memory and semaphores? Explain your answer.*

No. Because right after producer and consumer starts they attach already created semaphores and shared memory to themselves, and since there are no semaphores and shared memory in present, function **semget** and **shmget** will throw error.

Q6. *Initially the parent process and its two child (the producer and the consumer) use the same executable program* ***prod_cons_parent****. How is it done that later the producer and the consumer execute different executable programs?*

After fork operation, each process immediately forced to call the function **execl**, thus each process is overlaid by programs **producer.c** and **consumer.c**.

Q7. *Suppose that the producer wants to put a message into the shared message queue* *when this queue is full (that is. there are no empty slots). What will the producer do in this case? Explain in detail, using the source text of the producer.*

Suppose the length of queue is 10. When producer produces the 9th message and puts it in memory it increments the tails by one and takes the reminder of the increment divided by length, so it becomes 0 and on the next turn it overrides the first message.

So in fact, message queue will never be full, since its last index will be always empty.

Following codes shows the lines for this operation.

```
strcpy(memptr->buffer[memptr->tail], local);
memptr->tail = (memptr->tail + 1) % N_SLOTS;
```

Q8. *Suppose that the consumer wants to extract a message from the shared message queue when this queue is empty (that is. there are no messages in the queue). What will the consumer do? Explain in detail. using the source text of the consumer.*

It will do nothing and wait for the producer. Because in the enum the first parameter is *AVAIL_SLOTS*. If we change it and make the first one *TO_CONSUME*, it wont wait for producer anymore and will print empty string.

Q9. *Why do we need semaphores for the producer-consumer system? What can happen if no semaphores are used in this system?*

We need semaphores because we want to control the flow of our program, otherwise we cant predict what might happen, consumer might face with empty array situation, or race condition may occur.

Q10. *What can you tell about the values of variables* ***head*** *and* ***tail*** *when the message queue is empty?*

They are both zero, since they point nothing.

Q11. *What can you tell about the values of variables* ***head*** *and* ***tail*** *when the message queue is full?*

They both reset themselves to zero as explained in *Q7*.

Q12. *Suppose that the sleep time in the producer process is always zero, but the sleep time in the consumer is not zero. That is. the producer is capable to produce messages very fast. Can it really do this in the lab work, if semaphores are used? What will the producer do producing of messages in this case? Will it be blocked or not? If yes. then in what statement? Explain your answer with the of the source text.*

It will be slightly faster since it was waiting between 0 and 6 seconds in each cycle. But it still won't be as fast as it can be because it will be blocked by the consumer. The following code line keeps the producer waiting until consumer grants access.

```
semid = semget(myparid, 2, 0);
```

Q13. *Suppose that the parent process does not remove the shared memory and the semaphore set at the end. What will happen in this case? How can you see that the shared memory and the semaphores are not removed? Can you remove them manually? How?*

For semaphores,this is OS dependent and this behavior is not specified in Linux but it may cause resource leaks. For the shared memory, if not destroyed it will stay in the memory. And both of these can be destroyed manually by using **iprcm** command, and can be listed with **ipcs** command.

Q14.*The shared memory is the fastest mechanism of IPC. Explain Why*

Because otherwise data, signal, message should be physically carried in the memory by OS. In shared memory, all a process or thread needs to do is to access the shared variable.

Q15. *Explain the main disadvantage of shared mechanism of IPC.*

Synchronization is up to the developer. Developer must be careful, otherwise program can leave semaphores, shared memories which are not destroyed behind, and can cause race conditions etc.

Q16. *Is it always necessary to use semaphores with shared memory? Explain (see Introduction to OS textbooks).*

No. Developer can implement a custom signaling algorithm, or use mutex. Semaphore is just a method to lock access to shared memory by other processes, any method that can do this is applicable.

Q17. *Is it possible to the shared memory mechanism of IPC in UNIX for communication between processes run on different computers in a network? Explain.*

Yes. Since processes can run on different computers (locally), and since abbreviation of IPC is *Inter Process Communication*, it is possible.

Optimization

I had no time to make optimizations.

Appendix