Producer Consumer Project

1. Problem Statement

The assigned task was to use the virtual machine from the previous project to write a C/C++ program that will simulate the Producer/Consumer problem. The one Producer and the ten Consumers must be created using the fork(). These must be separate processes and must use Linux semaphores. The buffer size will be 1,000 and the producer will produce 1,000,000 items. The importance of this project will allow the user to know more information about forks(), semaphores, processes and one of the many ways to approach the Producer/Consumer problem.

1. Approach

The tools used were VirtualBox to host my virtual machine and the Ubuntu distribution. I also used g++, touch, vim and make to help me run and compile my code. I also included many libraries that were needed to run forks() and semaphores.

The decision was made to do it all in one file as this was my first time doing a medium sized project with forks(). Doing it all in one file helps me slow down and see all the individual parts interacting with one another. This helped me tremendously in the debugging process and made it much easier to fix my code. However, the downside is that it will now be much harder to scale up the project for later.

My approach was to first initialize the necessary memory that may or not be shared. The buffer was shared and was also to be a vector of integers and the size of the buffer would be kept track using a semaphore. I also used a variable that kept track of how many products left the producer should still produce.

Three semaphores were needed. One to handle mutex, which kept track if memory is available to access. Another kept track of how full the buffer is and will increment when a product is put in the buffer and will decrement when consumed in the buffer. The last one kept track of how empty the buffer is and will increment when a product is consumed in the buffer and decrement when a product is put in the buffer. These are the main variables to keep track of the simulation.

The next step in the program would be to create the processes. I created the processes using fork() and put the pid inside an array of pids. The program will currently create 11 processes. The parent is ignored while the children will either be a producer or a consumer. Only the first pid in the pid array is the producer while every other pid in the pid array is a consumer. This means that there will be 1 producer and 10 consumers.

In the program, the producer will continuously check the semaphores if it can create a product and put it in the buffer. If it can then the size of the vector will increase by one, the buffer will get closer to getting filled up and the count of the products needed to be produced will decrement closer to 0. Once 1,000,000 products have been created, the producer will kill itself.

The consumer will continuously check the semaphores if it can consume a product from the buffer and also if it is getting closer to the limit of consumable products. If it can consume a product from the buffer then. The size of the vector will decrement by one and the buffer will lose one product. Once it gets closer and closer to having 0 products ready to be produced and closer and closer to having 0 items in the buffer, the consumers will slowly kill themselves.

The program keeps a log of everything going on in the simulation. The producer knows how full the buffer is, how many products it should still make, and how many products it has produced so far and if it has killed itself yet. The consumer knows its own pid number, the buffer size, how many products will still be produced and how many it has eaten so far independently of other consumers.

1. Solution

The assigned task was to design, implement a program that would run a simulated Producer/Consumer problem. It would have to output the log of what happened during the simulation. It would also need to kill any zombies processes and clean up any semaphores.

To run the program, make sure you are building, compiling and running the program on a Linux virtual machine, preferably Ubuntu. You can compile the program with g++, but make sure you include -lpthread when compiling. You can also use the makefile that I have provided for you.

If you wish to change the buffer size, change the sizeOfBuffer variable to a different integer. If you wish to change the maximum items generated by the producer, change the \*maxItemsGenerated variable. If you wish to change the number of consumers, change the size of the int pid[] array from 11 to the integer you wish and also change the integer in the for loop that creates the processes from 11 to a number you wish. Right now changing the number of producers feature is currently not available.

I got the structure of the fork() code from the professor’s lecture. You can see it for yourself in the Blackboard Collaborate Recordings with the date of the lecture being 9/21/2020. The structure of the producer and consumer code is very similar to the textbook, and more specifically the code of the producer and consumer came from Figure 5.16. Other than that, everything is original.

Figure 1 demonstrates that the consumer will kill itself once the number of products in the simulation gets closer to 0. It will output the buffer size, how many products still need to be produced and how many it has consumed so far. It will kill itself once done.

When you add up what all the consumers have eaten before they died.

99953 + 100011 + 99999 + 100004 + 99970 + 100021 + 100016 + 100023 + 99980 + 100023 =

You get exactly 1,000,000 , which shows that this program works as intended.

Figure 2 shows that the producer will kill itself once done producing.

Figure 3 shows that the program keeps track of the logs in the simulation. It is very long so not the entire log could be shown in one picture. Run the program to have your own log of the simulation.

Figure 1.

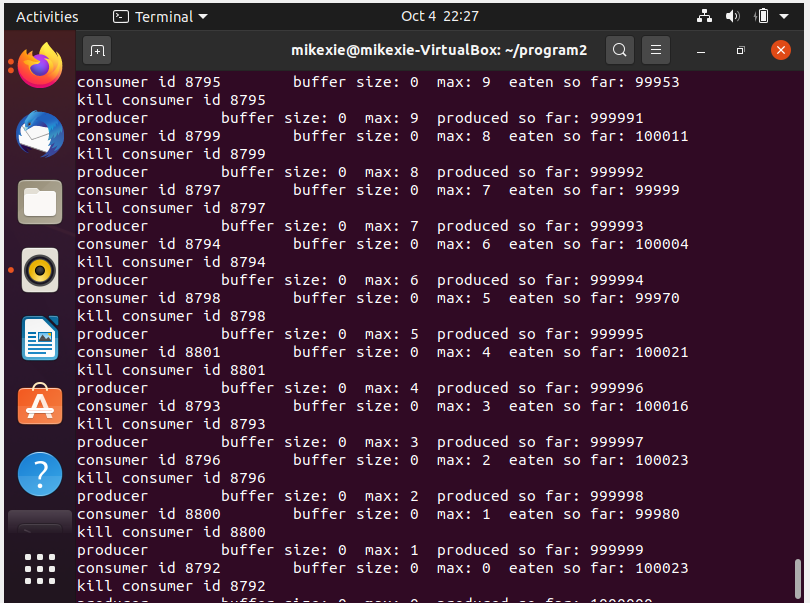


Figure 2.

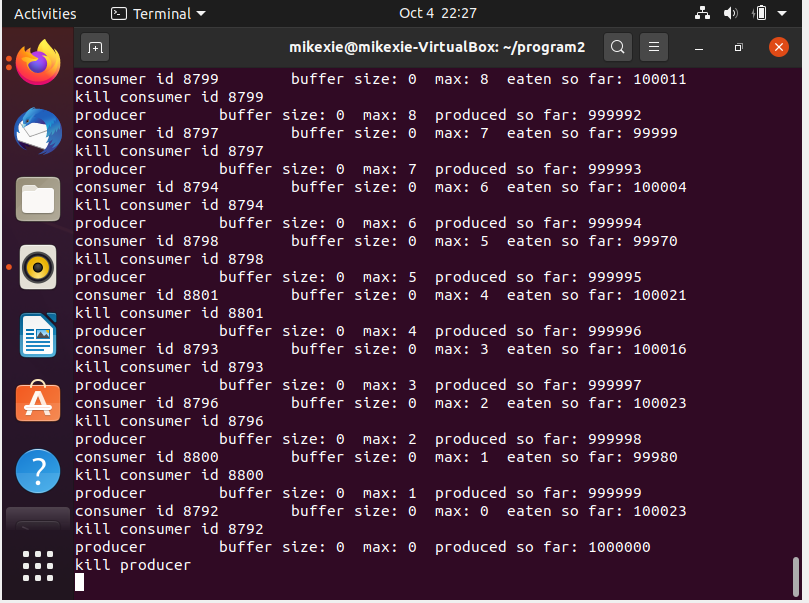


Figure 3.

