Due: Sun Nov 8, 2020 11:59pm

**Learning outcomes**

This lab will give you experience with synchronization techniques.

**Introduction**

Using concurrent programming, we can write a program that simulates a real-world system by using individual processes (jobs) and synchronization techniques.  For this lab, we will simulate a popular food cart, having some number of workers. Workers are trained to take an order and make the order for the customer, as opposed to having a worker that just takes orders and having workers just be cooks.  Workers help customers using this process: 1) Take the customer’s order 2) Fulfill the entire order 3) Help next customer, and so on… Each customer “foodie” arrives at the food cart, waits in a single line until a worker becomes available, orders, and then receives their food.

The major parameters to this system are:

1. Number of workers
2. Rate at which new customers arrive (specified as an average interval between arrivals)
3. Average time it takes to service a customer

Some things to notice:

1. If (average arrival time > average service time/ # of workers) – then customers will not have to wait very long, but the workers will not be busy enough to sustain the number of workers working.
2. If (average arrival time < average service time / # of workers) – then the waiting line will grow without limit and the customers will become tired of waiting.
3. If (average arrival time == average service time / # of workers) – then the food cart will reach a steady state with a fairly constant waiting line length.

The simulation will proceed as follows:

1. One process will be responsible for creating new customers at random intervals, such that the average time between new customers is equal to the specified parameter. This will be constant throughout the simulation.
2. Each customer will be simulated by a process that is created when the customer arrives, works its way through the line, orders, receives their order and then leaves the food cart (process terminated).
3. The waiting line will be simulated by a general semaphore that uses a FIFO queue.
   1. The semaphore will be initialized to the number of workers. If there are n workers, then the first customers doing a wait() on this semaphore, will be able to proceed without delay.
   2. When each customer process stands in line, they will do a wait() on this semaphore. When they are finished being served, they will do a signal, allowing another customer to pass the semaphore and allow them (the next customer) to place their order.

**Requirements**

Your program must conform to the following specifications:

**Input**

The following values must be entered to your program on the command line and they must be in the order given.  You should also have a usage statement to indicate the correct list of parameters.

The parameters are:

1. Number of workers
2. Mean time between customer arrivals
3. Mean service time
4. Length of time to run the simulation

All the time values should be entered in simulated “world seconds”, with ten simulated world seconds simulated by one second of program run time. (This means that the times input by the user should be multiples of 10).

**Output**

Each customer process should write three lines to the screen at the following times:

1. When they arrive to the food cart and enter the waiting line.
2. When they step up to the worker and starts being served.
3. When they are through being served and leave the food cart.

Each line should consist of the current time (in real-world seconds since the start of the simulated day) and an appropriate message, and should identify the customer by number. That is, the first customer created is customer 1, the second customer is customer 2, etc.…).

Finally, at the end of the simulation, a summary message should be printed giving:

1. The total number of customers that have been served.
2. The average time spent waiting in line by customers that have been served.

Note: any customer that is present in line when the simulation ends must be allowed to be served – even if this means that some workers work overtime a bit.  This means that you must keep track of the number of customer processes that are still in existence and not finish the simulation until this becomes 0 after the simulation time is up.

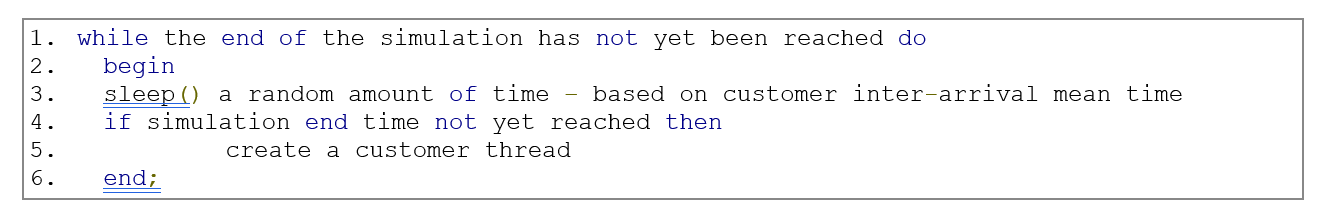
Note: You must treat both the screen and the variables that record the number of customers and cumulative data to be printed out at the end as shared resources to be accessed only by one process at a time. This will require some number of additional semaphores or mutexes. Even if your code appears to work, it will receive a poor grade if you do not properly ensure mutual exclusion on accesses to these!

**When you commit your code, include the output from a sample run.**

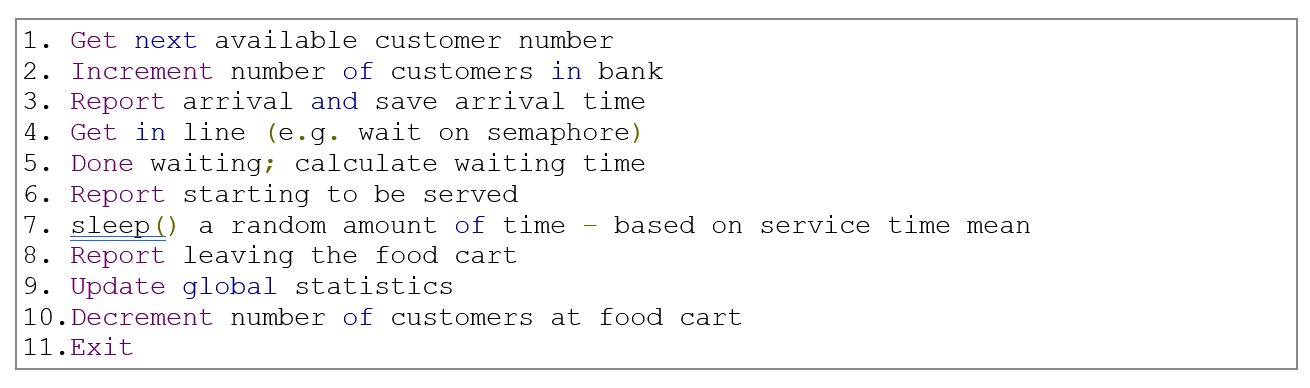
**Implementation**

Because all the processes will be updating certain shared variables, the most natural way to write this program is to use shared memory.

The process that generates the customer processes can be structured as follows:



A customer process can be structured as follows:



For an implementation language, you can use any OO language like Python, C++ or C#.

**C++**

For C++, you must use the pthreads library to create threads. Use the function called pthread\_create(3) where 3 indicates the man page for this function is in section 3 (section on subroutines).  Please also use the POSIX unnamed semaphore for synchronization.

1. To use pthreads and POSIX semaphores, you should include the following lines in your program:

#include <semaphore.h>

#include <pthread.h>

1. When building your program, you will need to include -lpthread as an option to g++ so that it links your program with the pthreads library.

1. You can find detailed information about pthreads (pthreads(5), pthreads\_create(3), pthread\_join(3), and pthread\_exit(3)) and the POSIX unnamed semaphores (sem\_init(3), sem\_wait(3), sem\_post(3), and sem\_destroy(3)) in the manual pages using the "man" command.

1. For randomness and timing, you can use any libraries you are familiar with or do some research on the best method for generating random numbers and for calculating timespans.

**C#**

1. Put a using statement for System.Threading;
2. Use the Semaphore and Mutex classes in System.Threading.
3. Store the customers in a BlockingCollection<Thread>() collection (put a using statement for System.Collections.Concurrent).
4. Create customers as new Thread(CustomerProcess); where CustomerProcess is the method that contains the definition of the customer’s process.
5. For timing, you may want to check out StopWatch or TimeSpan.

**Deliverables**

1. Screen shot or raw text of your output.
2. Your code committed and pushed to master with a tag/release submitted to Canvas.

**Appendix A: Sample program output**

Mean inter-arrival time: 10

Mean service time: 30

Number of tellers: 3

Length of simulation: 70

At time       30, customer        1 arrives in line

At time       30, customer        1 starts being served

At time       30, customer        2 arrives in line

At time       30, customer        2 starts being served

At time       30, customer        3 arrives in line

At time       30, customer        3 starts being served

At time       40, customer        1 leaves the food cart

At time       50, customer        4 arrives in line

At time       50, customer        4 starts being served

At time       50, customer        4 leaves the food cart

At time       50, customer        5 arrives in line

At time       50, customer        5 starts being served

At time       50, customer        6 arrives in line

At time       50, customer        7 arrives in line

At time       60, customer        3 leaves the food cart

At time       60, customer        5 leaves the food cart

At time       60, customer        6 starts being served

At time       60, customer        7 starts being served

At time       70, customer        7 leaves the food cart

At time       70, customer        8 arrives in line

At time       70, customer        8 starts being served

At time       70, customer        8 leaves the food cart

At time      100, customer        6 leaves the food cart

At time      110, customer        2 leaves the food cart

Simulation terminated after 8 customers served

Average waiting time =     2.50