Operating Systems Project 3 (due Fri 5/7/21)

Groups: This is a group project, where the maximum group size is 2. So you can work on this by yourself or with one partner.

Grading: If you just do one part, your maximum score on the assignment is 80%. If you do both parts, your maximum score on the assignment is 100%.

Notes:

1. You can write these in whatever programming language you like.
2. All the thread synchronization needs to be done with locks, condition variables, or semaphores. Definitely do not use the sleep function to synchronize threads, though you can use it in #2 in those places it tells you to.
3. There is a text file included that has sample output.

***Warning**: You can look up small things online or ask friends for a little help, but if you go too far there will be consequences. For instance, taking someone else's code and changing some spacing and variable names is a definite no-no. Asking someone online to write it for you is also a definite no-no. If I suspect you gave or received excessive help, you will get a 0 on the assignment, and maybe worse. If I'm not sure, I may ask you to have a Zoom meeting with me where you walk me through your code to demonstrate your understanding of what you wrote.*

First problem

Write a program that satisfies the following requirements.

1. Create four threads, all running the same code. When created, pass each thread two arguments: a name (string) and a number (integer/long integer).
2. As soon as each thread starts, it should print its name and that it has started.
3. Then each thread will run a loop adds up all the integers from 1 up to the number from step 1.
4. Once the thread is done with its loop, it should print out the sum that it computed.
5. It then needs to wait until *all* the other threads are done with their counting. Once it's sure that they are all done, it prints out that it's done and exits.
6. Please use locks to protect the print statements so that the output doesn't end up all jumbled.

Hints: You'll want to use a global variable to count how many threads have finished their loops. Whichever thread finishes its loop last (based on the value of the count variable) should wake up the other threads. Please protect operations on this count variable using a lock.

You can use a condition variable or semaphores to allow the threads to wait for the others. Do *not* use the join method or sleep for this. If you use a condition variable, the `notifyAll` method will be helpful. If you use semaphores, make sure to call release after you call acquire or else you'll end up with a deadlock. If you're using C++ or Java, use a long variable for the number passed to the thread and for the variable that holds the sum.

Second problem

This problem is about simulating a small restaurant. Here are the specifications:

1. There are 5 tables, 1 server, and a waiting room.
2. If there are no customers, the server goes on break.
3. If the restaurant has no customers and one arrives, the server comes back.
4. If a customer enters and there are free tables, they take one. They stay for a random amount of time between 1 and 5 seconds and then leave. Use the sleep function to simulate the time the spend at their table.
5. If a customer enters and all 5 tables are full, then they will wait if there isn't more than 1 other customer waiting for a table. Otherwise they will leave.
6. Time passes as if in a real restaurant. Every 5 seconds of the simulation corresponds to a new hour. It starts at midnight and continuously runs through the days one hour at a time.
7. The customers arrive randomly, with more customers appearing at particular parts of the day than others. To simulate the arrival of customers, the program sleeps for a random amount of time in between customer arrivals. This is a random double that can be in the range from 0 to some max value that varies based on the hour. Those max values are given below. To do this, you can use `random.uniform` in Python or the `nextDouble` method from Java's `java.util.Random` library.

12	1	2	3	4	5	6	7	8	9	10	11	noon	1	2	3	4	5	6	7	8	9	10	11
5	5	7	10	8	5	3	2	1	.5	1	2	2	3	4	3	1	.25	.5	2	3	4	4	5

   Here's a list if you want to copy it into your program.

   ```
   [5, 5, 7, 10, 8, 5, 3, 2, 1, 0.5, 1, 2, 2, 3, 4, 3, 1, 0.25, 0.5, 2, 3, 4, 4, 5]
   ```

8. Create one thread for the server.
9. Create threads for each customer, each one running the same code. Each customer has a name, which is just which customer number they are (over the whole history of the simulation). The main part of your program is a good place for your program to create the new customers, running in an infinite loop.
10. Create a thread that keeps track of time. It shouldn't do much more than sleep for 5 seconds, update what hour it is, and print out the new time.
11. Your program should print out the following:
    (a) Whenever the time changes, print out the new time.
    (b) Whenever a customer arrives, print out their number and how many people are in the restaurant when they get there (not including themselves). If they need to wait or decide to leave, print that.
    (c) When a customer is seated, print out their number and that they are seated.
    (d) When a customer leaves, print out their name and that they are leaving, and print out how many customers are left.

Hints: Use locks to protect the print statement. Use a count variable for how many people are there, and protect that count with a lock. You can use a condition variable or semaphore to deal with the waiting room. Use a condition variable or semaphore to deal with the server going on break and coming back.

Note: For a small amount of bonus points, make sure your output displays everything cleanly. For instance, it should say "There is 1 customer" not "There are 1 customers," and times should display nicely, like "It's 1 pm" not "It's 37:00."