**OPERATING SYSTEM**

**GROUP PROJECT  
University of Technology, Jamaica**

**GROUP MEMBERS:**

Shanice Facey 1701438

Dinito Thompson 1801202

Doneque Smith 1704299

Tyeree Tinker 1803646

**PROBLEM STATEMENT**

**The following Synchronization problem was given, with the task of coding a solution with the given constraints:**

The university of technology campus hosts a very popular bar from which students order beer from. In the passing days, the students have become dependent on beer to carry out their tasks. Students need beer to think, therefore it is constantly in high demand. With only having one bartender, who is constantly falling asleep on the job, the following takes place. When a student wants beer, they take a flask from a barrel as long as said barrel is not empty. If the barrel is empty however, the student wakes up the bartender and waits on the bartender to refill the flask.

Therefore:

* An unsynchronized SCIT student thread will run the following (pseudo-)code:

**while True:**

**get\_serving()**

**drink\_and\_think()**

* And the bartender will run the following unsynchronized code:

**while True:**

**refill\_barrel()**

CONSTRAINTS:

Any number of SCIT student threads can be running concurrently, but there is only one bartender thread. The problem has the following synchronization constraints:

* SCIT Students cannot get any light beer if the barrel is empty.
* The bartender can refill the barrel only if the barrel is empty. Also, the bartender will not always fill the barrel to maximum capacity.

**SIMULATION**

The simulation begins when a student thread is created and started in main. The students need beer to drink, hence, they will retrieve one from the barrel. The barrel will always start off at its max capacity being 50 beers, and student thread will retrieve beers and think (this being represented as 1000ms sleep time) until the barrel is empty.

Once the barrel is empty, student threads are put into a waiting state and the bartender thread is awoken. The bartender will then check if the maximum number of refills as been met. If not, the bartender will refill the barrel with a random number of beers in the range of 10 – 50 beers. At the end of this process, the bartender will update the remaining number of refills. On the other hand, if the maximum number of refills has been met, the bartender thread will output a notice that there will be no more refills and the program will terminate.

**ALGORITHM**

PRODUCER-CONSUMER ALGORITHM

A wildly used classical synchronization algorithm, it uses two processes which are the Producer (Bartender) and Consumer (Student). These processes share a buffer shared as a queue. The producer process generates the data that is placed into the buffer, as well as repeats the cycle as the consumer is retrieves from the buffer. The main issue in this algorithm is to ensure that data is not added to the buffer when it’s at its maximum capacity as well as data not being retrieves when the buffer is empty.

This can be solved in 2 ways. The first, is for the consumer sleep if the buffer is empty so it does not try to retrieve data, while at the same time, the producer is alerted and begins to place generated data in the buffer. The second is to discard data when the buffer is full until the consumer retrieves from the buffer. This algorithm was chosen for its similar nature to the problem that is to be solved, with the functionality requirements mirroring the process outlines in the project as well the actions to be taken

**SYNCHRONIZATION REQUIREMENT**

MONITORS

A monitor gives ways for mutual exclusion for threads, as well as giving them the ability to wait for specific conditions. This ensures that only one process can be run at any one time, whether it being a student or bartender thread. Furthermore, as the processed wait and allow access to the buffer, the monitors ensure that only a single process has access to the buffer at any singular point in time, as well as signals when other process are ready to access the buffer.

In this case, the student will retrieve from the barrel until it is empty, at which point they will signal to the bartender that the barrel is empty. The bartender then refills the barrel and then notifies the student that they can continue.

* Mutual Exclusion

Monitors allow for mutual exclusion and waiting. The student thread(s) waits using wait() until the bartender has refilled the buffer then notifies the thread using notify() to resume. At any time, at most one process (bartender/student) should be executing

* Progress

Processes can enter the critical section so long as that process is free. This can be seen in the student notifying the bartender to enter the critical section when the barrel is empty.

* Bounded Wait

If a process is trying to enter the critical section, will be given access when it has requesting and waiting to be notified. This can be seen as how the bartender has a hard limit of 3 refills to reach its