Assignment 4
Concurrency
Computer Systems Engineering 1
Monsoon 2019

Deadline: 12th October, 11:59 PM.
Note that this is a hard deadline and will not be extended.

General Instructions

- This is an individual assignment, no collaboration is permitted
- Any kinds of plagiarism will be penalized.
- Submission by email won't be accepted.
- You are required to submit a detailed report explaining your implementations of the assignment tasks and their respective results.
- The report will be graded. Note that verbosity will not fetch you marks. Your report should be a high-level explanation of your code and results.

Concurrent Quicksort

Quicksort is a Divide and Conquer sorting algorithm. It is one of the most efficient sorting algorithms and is based on the idea of splitting an array into smaller ones. The algorithm first picks an element called a pivot and partitions the array into a **low subarray**, elements smaller than the pivot and a **high subarray**, elements greater than the pivot. This step is called the partitioning step.
Quicksort algorithm recursively applies the above step to the sub-arrays, resulting in sorting the array.
In this task, you are given the number of elements in the array N, and the elements of the array. You are required to sort the numbers using a Concurrent version of Quicksort algorithm.

Tasks

1. Implement a Concurrent version of Quicksort algorithm
2. Make a detailed report
    a. Explains your implementation for the Concurrent Quicksort algorithm
    b. Compares the performance of Concurrent Quicksort with normal Quicksort

Bonus Task

1. Implement a variant of Concurrent Quicksort algorithm, which uses threads instead of processes.
2. Compare the performance of this variant of the algorithm with the ones implemented as a part of the above mentioned tasks.

Instructions

- The median of the subarray should be chosen as the pivot. It is also allowed to choose a random element from the subarray as your pivot.
- Partition the array around the pivot such that all the elements with a value less than the pivot are positioned before it, while all the elements with a value greater than the pivot are positioned after. In case of equality, they can go on either side of the partition.
- Recursively make two child processes. One of which will sort the low subarray and the other will sort the high subarray.
- When the number of elements in the array for a process is less than 5, perform an Insertion sort to sort the elements of that array.
- Use the shmget, shmat functions for accessing the shared memory.

Automating Biryani Serving

The line for serving Biryani is growing day by day in the Kadamb Mess. Annoyed by waiting for hours in the queue you have decided to automate the entire pipeline of serving Biryani.

Infrastructure

Robot Chefs

- From now on, there will be **M Robot Chefs** present in the kitchen preparing vessels of Biryani.
- Each Robot Chef can prepare a random number (greater than or equal to 1) of Biryani vessels at a time.
- Each Biryani vessel has a capacity to serve **P Students**.

Serving Tables

- There will be **N Serving Tables** present in the mess.
- Each Serving Table has a **Serving Container** which loads a Biryani vessel prepared by any of our Robot Chefs. Only one vessel of Biryani can be loaded at a time into the Serving Container. The Serving Container can be refilled only when it is empty.

- The Serving Table incorporates the latest state of the art technologies and has automated the process of serving Biryani through multiple slots available on it.
- As long as a serving table has enough portions of Biryani left in the serving container, it makes a random number of slots available for the students to collect a portion of Biryani from the Serving Table. Note that the number of slots should always be less than or equal to the portions left in the Serving Container. Read down for exact limits.
- Thus there is a possibility that once a Serving Table has **x** available slots and later has greater than **x** or less than **x** slots available. After serving **x** portions (and consequently **x** students) the portions available in the Serving Container is updated.

Student

- **K Students** have registered for the Biryani.

<u>**Pipeline**</u>

Robot Chef

- Each Robot chef takes **w seconds** (random between 2-5) to prepare **r vessels** (random between 1-10) of biryani at any particular time. Each vessel has a capacity to feed **p students** (random between 25-50).
- Once the Robot Chef is done preparing the Biryani Vessels, he invokes the function biryani_ready() and does not return from it until all the biryani vessels he has cooked are loaded into any of the serving containers.
- Once all the biryani vessels are empty, the biryani_ready() function returns and the Robot Chef resumes making another batch of Biryani.

Serving Tables

- Each Serving table's serving container is initially empty. It waits for any of the Robot Chefs to load a vessel of Biryani into the serving container. The serving table cannot go into serving mode as long as its container is empty.
- Once the serving container is full, it enters into the serving mode. It invokes a function ready_to_serve_table(int number_of_slots) in which number_of_slots (chosen randomly between 1-10) denotes the number of slots available at the particular serving table at that instant. The function must not return until either all the serving slots of the serving table are full or all the waiting students have been assigned a slot. Note that student can be assigned a slot at any of the serving tables. (If there is no waiting student, then the function returns)

Students

- Once a student arrives in the mess, he/she invokes a function wait_for_slot(). This function does not return until a Serving table with a free slot is available, that is ready_to_serve_table method is in progress. Once the function returns the students goes to the allocated slot and waits for the slot to serve Biryani.
- Once the student is in the slot, he/she will call the function student_in_slot() to let the Serving Table know that he/she has arrived at the slot.
- Once all the students have been served you are done for the day.

To convince the Mess Committee you have decided to build a simulation of your efficient serving pipeline.

Instructions

- Each Robot Chef, Serving Table and Student are threads.
- Stop the simulation when all the students are served.
- A Serving Table is used multiple times for serving. That means if there are still students left to be served, the serving table should invoke ready_to_serve_table().
- Use appropriate small delays for the simulation.
- Prepare a detailed report explaining your implementation and assumptions. Note that the report will be graded.
- The use of semaphores is not allowed. You can use only one mutex lock per Serving Table and Robot Chef.
- Your simulation should allow for multiple students to arrive at a Serving Table simultaneously. It must be possible for several students to have called wait_for_slot() function and the wait_for_slot function returned for each of the student before any of the student calling student_in_slot function
- Your simulation must not result in dead-locks.
- You are allowed to declare more functions if you require.

The goal of this problem is for you to appreciate the use of threads and synchronisation techniques. There are no strict restrictions on the output format, as long as the simulation functions smoothly and displays appropriate messages.

Ober Cab Services:

Ober is a new cab service, which needs your help to implement the simple system. The requirements of the system are as follows given N cabs, M riders and K payment servers you need to implement a working system which ensures correctness and idempotency. Each cab has one driver. In Ober, payments by the riders should be done in Payment Servers only. Ober provides two types of cab services namely pool and premier ride. In pool ride maximum of two riders can share a cab whereas in premier ride only one rider. There are four states for a cab namely **waitState** (no rider in cab), **onRidePremier**, **onRidePoolFull** (pool ride with two riders), **onRidePoolOne** (pool ride with only one rider).

As a part of this system you need to implement the following functionalities.

1. **<u>Riders</u>**:
    a. BookCab: This function takes three inputs namely *cabType*, *maxWaitTime*, *RideTime*. If the rider doesn't find any cab (all cabs are in usage) until *maxWaitTime* he exits from the system with *TimeOut* message.
    b. MakePayment: This function should be called by rider only after the end of the ride. If all the K payment servers are busy, then the rider should wait for the payment servers to get free. After payment is done rider can exit from the system.
2. **<u>Drivers</u>**:
    a. AcceptRide: If ride is premier cab in *waitState* should accept ride and change its state to *onRidePremier*. If the ride is pool and there is a cab with state *onRidePoolOne* then that cab should accept the ride and changes its state to *onRidePoolFull*. If the ride is pool and there is no cab with the state *onRidePoolOne* then cab in wait state should accept ride and change its state to *onRidePoolOne*. Assume that there is no time gap between accept rider and pickup rider, i.e., ride starts immediately after accepting ride. If there are multiple cabs available with required state take any random cab.
    b. OnRide: When pool cab is on ride and state is *onRidePoolOne* then driver can accept another pool ride. No new rider is accepted when cab is on premier ride.
    c. EndRide: The driver ends the ride after completion of the *RideTime* of the rider. If the ride is a premier it goes to *waitState* (ready to accept new rides) once ride is done. If cab's state is *onRidePoolOne* then driver ends the ride and goes to *waitState*. If cab's state is *onRidePoolFull* then it goes to *onRidePoolOne* state (Notably, it can accept another pool rider). Driver or cab need not wait for the payment to be done by the rider.
3. **<u>Payment Servers:</u>**
    a. Accept payment: Payment servers accepts a payment and assume the payment time is constant for all the riders (assume 2sec for this assignment).

<u>Instructions</u>:

1. Your code must not result in busy-waiting (deadlocks).
2. Use appropriate times for RideTime and the maxWaitTime.

3. You need not follow above mentioned functions you can create functions appropriately.
4. Use Semaphores and mutex accordingly.
5. Prepare a detailed report explaining your implementation and assumptions. Note that the report will be graded.
6. Use separate threads for each of the drivers, riders and payment servers.
7. Use only four states of Cab as mentioned above.
8. Print appropriate statements for every update.
9. Write a proper report or Readme explaining the code.