CS3.301 **Assignment-4**

Computer Systems Engineering - I

Vikrant Dewangan
Roll no.- 2018111024

Contents

1 Problem Statements

1.1 Concurrent Quicksort

Perform a concurrent version of quicksort.
Input: N - Number of elements in the array A and the elements.
Output: Sorted elements .
Sample Input:
10
7 5 6 4 2 3 1 9 5 6
Sample Output:
1 2 3 4 5 5 6 6 7 9 .

1.2 Automating Biryani Serving

There are M robot chefs, each preparing r (random number) vessels of Biryani.Each vessel has capacity to serve P students. There are N tables present in the mess wherein each vessel's serving container can be loaded.
Input:
N - Number of tables.
M - Number of robots.
K - Number of students arriving.
Output: Sequence of events in the simulation so that every student is served a biryani .
Sample input
1 1 1
Sample output
Robot 0
Robot 0 has started preparing 1 vessels . It will take 3 time Student 0 will arrive at 2 time
Student 0 comes into the mess
Robot 0 finished preparing 1 vessels now will start loading
The table 0 is being filled by vessel from 0 robot
slots decided 1
Student 0 is allocated 0 table's 0 slot number
0 student stops eating at 0 table's slot number 0
Student 0 going bye
Robot 0 has started preparing 1 vessels . It will take 5 time Robot 0 finished preparing 1 vessels now will start loading
Robot done working bye from robot side

1.3 Ober Cab Service

It is requrired to maintain a cab service.There are N cabs, M riders and K payment servers. There are 2 types of cabs, pool and premier ride. There are 4 states of a cab, waitState, onRidePremier, onRideFull and onRidePoolFull.

Input:
N - Number of passengers.
M - Number of cabs.
K - Number of servers.
Output: Sequence of events in the simulation so that every passenger is given a cab .
Sample input
3 1 1
Sample output
Server 0 ready for payment
Passenger 2 arrived on board with request type POOL TYPE and wait time 1 and ride time 23
Passenger 2 boarded 0 cab ok
Passenger 0 arrived on board with request type PREMIUM TYPE and wait time 3 and ride time 7
Passenger 1 arrived on board with request type POOL TYPE and wait time 2 and ride time 4
Passenger 1 boarded 0 cab ok
Passenger 0 leaved without service
Passenger 1 left 0 cab
Payment of rider numbered 1 done by server numbered 0
Going bye 1
Passenger 2 left 0 cab
Payment of rider numbered 2 done by server numbered 0
Server 0 ready for payment
Going bye 2
Thank you for using Ober Cab Services. We wish to serve you again

2 Directory Structure

```
4Assignment
├── quicksort
│   ├── constraints.h
│   ├── simple_quicksort.c
│   ├── concurrent_quicksort.c
│   ├── multithread_quicksort.c
│   └── main.c
├── biryani
│   ├── global.h
│   └── main.c
└── ober
    ├── global.h
    └── ober_main.c
```

3 Quicksort

3.1 File Structure

constraints.h
Contains all the common functions to sorting functions like *kthSmallest,partition,insertion sort.*
simple_quicksort.c
Contains implementation of simple quicksort. The function *quickSort* takes care of it.
concurrent_quicksort.c
Contains implementation of concurrent quicksort . The *quicksort* function is responsible for creating the child processes which sort the array.
multithreaded_quicksort.c
Contains implementation of multithreaded quicksort . The function *quicksort* will check the thread number and call threads to sort their respective quarters.

3.2 Workflow

1. **For Simple Quicksort**
   The function calls *quickSort* which then calls the common functions in *constraints.h* appropriately. At the last it prints time of execution.

2. **For Concurrent Quicksort**
   In the main, first a shared memory segment `shm_array` is created so that all processes able to access it. Along with it, *shm_at* function is used to attach segment with id to address space of calling process. After that control is handed to *quicksort* which checks if array size is less than 5, in which case it performs insertion sort. It partitions the array around the median using the median found in median of median method. After fork is created, left child executes the left subpart, parent forks again and right child executes the right subpart. The parent is made to wait for both the children to join.

3. **For Multithreaded Quicksort**
   First, the array is partitioned one by one using 4 medians found during the process. They are stored in `median_array`. After that 4 seperate threads are called to sort each subpart of the array. The function *quicksort* takes care of it.

3.3 Constraints

- Please do not give more than 2000 numbers for concurrent quicksort.
- The running times of execution for normal size of inputs are as follows -

  simple < multithreaded < concurrent

  However note that for larger inputs, multithreaded tends to get faster than simple.

simple > multithreaded < concurrent

- At less than 5 elements, their times are almost same.

4 Automatic Biryani Serving

4.1 File Structure

global.h
Contains all the macros ,structs as well as mutexes . Also contains the initialization function for students,tables and robots.
main.c
Contains all the necessary functions -

1. **For robots**
   (a) *prepare_biryani*
   The function produces r vessels of biryani with capacity p having x slots when it will be put to table. Note here that the number of slots it generates will be computed before putting to the table.
   (b) *biryani_ready*
   The robot, when ready to serve biryani, goes into this function to do polling and places the vessel wherever the table is free.
2. **For students**
   (a) *wait_for_slots*
   The function makes the student wait until he arrives. After that it calls, *student_in_slot()* function.
   (b) *student_in_slot*
   Checks for nearest available slot in any table which in empty. `lock_students` prevents dirty access. It also makes the table status 0 when all students have occupied the slot meaning all of them can eat .
3. **For tables**
   (a) *serving_table*
   Just makes the thread wait until all have finished eating. Student searches for the table by him/herself .

4.2 Workflow

1. **For robots**
   The function *prepare_biryani* is called by the thread of robots.It prepares a random number of vessels in which it takes some time and then calls *biryani_ready* function.The *biryani_ready* function checks for every table if it is empty or not by using mutexes for

protection. If it is it places the vessels over there. The robot does not exit from this function to prepare more biryani until all vessels are kept.

2. **For students**
   The function *student_arrives* is called by the thread and it makes it wait until the student's arrival time.After that it calls *wait_for_slot* function which checks over all the tables using a mutex which table has which slot free.After that it calls *student_in_slot* function and passes it the slot numbers. If the slot number is the last slot, then all students vacate the table and have finished eating.Else , a student is stuck as long as all the students don't arrive in the table's slots.

3. **For tables**
   It waits just to check the students have finished eating or not in *serving_table* function.

4.3 Constraints

1. Any number of robots, students or tables may be given provided it is sufficiently small for malloc to be able to allocate memory.

2. The program's running time may get large as printing may take time.

3. Note that if many events occur simultaneously, it may be possible for the printing order to get random depending upon whichever prints first.

4. The possibility of busy waiting may occur if a student repeatedly checks for slots in table while there isn't any slot.

5 Ober Cab Services

5.1 File Structure

global.h
Contains all the macros,structs ,mutexes and semaphores.Also contains the initialization function for passengers,cabs and servers.
ober_main.c
Contains all the necessary functions -

1. **For passengers**

   (a) *function_pass*
   Delay of arrival time is introduced.myCond function is called until max_wait_time.After that the thread exits if not serviced.If serviced, it unlocks ride_done semaphore which unlocks a srver to the payment.

   (b) *myCond*
   Continously polls over the cab struct array using a binary semaphore whether any of the cab of it's type is free or not.

2. **For servers**

   (a) *function_srvr*
   This function waits for any rider to complete his/her ride. After so, it checks which rider has completed the rider (by applying a lock) and marks it's status as payment completed.

5.2 Workflow

First all semaphores are initialized to their respective values. Most semaphores I have used in the code (like `ride_done` , `available_servers`) work as binary semaphores. After that all structs are initialized as well the mutex locks are also initialized .

1. **For passengers**
   function_pass is called for each thread. Note that since , every passenger comes only once, their time of arrival, ride_time, and wait_time are precomputed .It introduces a delay until the passenger as arrived. After that , the function waits on the respective request type using *pthread_cond_timedwait* function. It will either exit out of it if timed_wait happens in which case the thread exits completely. If not, it will look for the available cabs free by iterating over them using a mutex for each cabs to prevent concurrent access.
   After completing the ride, it signals the semaphore `ride_done` so that a payment server, if free is allotted to it's payment.

2. **For servers**
   function_srvr is called for each thread. It waits on `ride_done` semaphore until it is signalled by the passenger. After which, it iterates over the passenger struct to check which passenger had completed his/her ride (using a mutex lock to prevent concurrent access).

5.3 Constraints

1. Any number of passengers, servers or cabs may be given provided it is sufficiently small for malloc to be able to allocate memory.

2. The program's running time may get large as printing may take time.

3. Note that if many events occur simultaneously, it may be possible for the printing order to get random depending upon whichever prints first.

4. No busy waiting is there as conditional variables are used.

****