Process Synchronization

Automated Biryani Serving

Problem statement

The line for serving Biryani is growing day by day in mess. Annoyed by waiting for hours in the queue we should automate the entire pipeline of serving Biryani.

Introduction

We need to automate the biryani serving process with sample simulation. We will be having M robot chefs who will prepare r vessels with each of p portions. We have N tables to serve with slots being minimum of portions left in the container and random value between 1-10. We have K students to come to mess in random times and all have to be assigned a slot in any table and should be served. When all students are served, our simulation ends.

Requirements

1. **pthread.h** for threads

Compiling the code

gcc -g <filename> -lpthread

Input format

N(Number of robot chefs) M(Number of serving table) K(Number of students)

[OPTIONAL] *arrivalTime[K] (Arrival times of K students)*

Implementation

Robot chefs

For each robot chef, we need to make a thread so that they create biryanis parallely. To make threads, we can use pthread.h header file. We need to initialize pthread_t variable and create a thread to this variable using pthread_create function and call prepare_biryani() function in this thread. As the robot chef takes random time w to prepare r biryanis with each of p portions. W, r, and p can be found using rand() function. Sleep() can be used to halt the thread for x secs. A linked list is being maintained to add the chefs into it who have finished preparing biryanis. As this linked list can be accessed by all threads, we need to use a mutex lock so that only one thread will be able to update or retrieve data from it. When a robot chef prepares biryanis, the biryani array is being updated with number of vessels by ith chef. As biryani array is also accessible from all threads, we need to use mutex locks again, but using only one mutex lock for full array might be inefficient as it stops working of other chefs when one chef is updating the array. So, we can make M mutex locks for chefs so that for ith chef we lock ith chef mutex lock and update ith index of biryani array.

After this, chef calls biryani_ready() function which returns only when all vessels prepared by chef are loaded to any table. So, we implement a infinite while loop with checking ith index of biryani array using ith chef mutex lock and return when it becomes 0 (zero).

Serving tables

Similar to chef we need to create N table mutex locks for accessing the table array where table array stores the number of free slots available on ith table. After creating the threads for each serving table, we initialize the thread with wait_to_get_full() function which waits until the table is loaded. As the table container can be left with some portions of biryani after one round, we need to keep track of remaining biryani portions which is achieved by variables remainingbiryaniflag and remainingbiryaniportions. To access the biryani linked list which we created in chef, we need to lock the resource using biryani mutex lock. Once we get number of slots available we update the number of slots in table array using table mutex lock and pushing the slots and tableid into table linked list using table mutex lock.

After this we call ready_to_serve_table() function which checks whether all slots have been exhausted or not, if yes the function returns, or else it is stuck in infinite while loop until it exhausts the slots in the table.

Student

Once the student arrives into mess, he calls wait_for_slot() function. As the student arrival time is not same for all students, we take student arrival time as input.

In wait_for_slot() function, the student keeps waiting till a slot is assigned to him/her. To check whether a slot is free or not, we check the table linked list which contains number of slots in ith table. Until the student finds a slot, he will be waiting and when the student gets assigned a slot, he occupies the slot (decrementing number of slots in table i) and calls student_in_slot() function where he will be served and finishes eating biryani for x secs (taken randomly between 5 to 10). Then the function returns and the student exits the mess (thread exits).

When all K threads exit, the simulation stops.

Mutex locks used

1. *biryanimutex for biryani linked list*
2. *tablemutex for table linked list*
3. *cheflock[M] for M chefs*
4. *tablelock[N] for N tables*

Structs used

1. *chefstruct*
2. *tablestruct*
3. *studentstruct*
4. *biryaninode*
5. *tablenode*