Question 2:

Automated Biryani Service

Structures Used

Struct Chef:
The chef structure has an index, and preparation time, num_vessels and capacity which are determined by a rand() function. These are calculated every time the chef has to cook biryani.

Struct Table:
The table structure consists of an index, how many people the serving_container can serve currently, and number of slots the table has alloted.

Struct Students:
The student structure contains an index, his status of the meal, 0 if he hasn't eaten and 1 if he has and the entry_time at which he arrives at Kadamb.

```
struct robot_chef{
	int index;
	int preptime;
	int num_vessels;
	int capacity;
} * chefs;

struct serving_table{
	int index;
	int serving_container;
	int slots;
} * tables;

struct students{
	int index;
	int status;
	int entry_time;
} * students;
```

```
void biryani_ready(int index){
    int lolflag=0;
    while(chefs[index].num_vessels){
        for(int j=0; j<n; j++){
            if(pthread_mutex_trylock(&mutex_tables[j])){
                continue;
            }
            if(tables[j].serving_container==0){
                printf("Robot Chef %d is refilling Serving Contain
                tables[j].serving_container=chefs[index].capacity;
                chefs[index].num_vessels-=1;
                printf("Serving Container of Table %d is refilled
                //containers--;
                //printf("Chef %d served table %d with capacity %d
                fflush(stdout);
            }
            pthread_mutex_unlock(&mutex_tables[j]);
            if(!chefs[index].num_vessels){
                lolflag=1;
                printf("All the vessels prepared by Robot Chef %d
                break;
            }
        }
        if(!chefs[index].num_vessels && lolflag==0){
            printf("All the vessels prepared by Robot Chef %d are e
            break;
        }
    }
}
```

Robot Chefs

The chef threads are initialised in a chef_init() function where the values for preparation time, number of vessels, etc are determined.
The threads then enter the biryani_ready() function where we constantly poll the serving containers of tables to see if they are empty. If they are, we fill the table's container safely using mutex. This is done until number of vessels of the chef goes to 0 in which case he stops serving and goes back to the initialisation function where he prepares more biryani.

Serving Tables

The serving tables are initialised using a table_init() function in which it waits until a robot has given it a vessel and then it randomises a value for the number of slots and goes to the ready_to_serve_table function. In which it serves biryani until either the slots are finished or the students remaining are none in which case it returns to randomise the value of slots.

```
void ready_to_serve_table(int number_of_slots, int index){
     while(1){
          if(pthread_mutex_trylock(&mutex_tables[index])){
               continue;
          }
          if(tables[index].slots==0 || studentsremain==0){
               if(tables[index].serving_container==0){
                    printf("Serving Container of Table %d is
               }
               pthread_mutex_unlock(&mutex_tables[index]);
               break;
          }
          pthread_mutex_unlock(&mutex_tables[index]);
     }
}
```

```
while(cond){
     for(int j=0; j<n; j++){
          if(pthread_mutex_trylock(&mutex_tables[j])){
               continue;
          }
          if(tables[j].slots>0){
               printf("Student %d assigned a slot on the servi
               fflush(stdout);
               student_in_slot(index,j);
               cond=0;
               printf("Student %d done eating at table %d whic
               fflush(stdout);
               //printf("Students ate at table %d. Slots left
               //pthread_mutex_unlock(&mutex_tables[j]);
               break;
          }
          pthread_mutex_unlock(&mutex_tables[j]);
          continue;
     }
}
```

```
void student_in_slot(int index,int j){
     tables[j].slots-=1;//chefs[index].capacity;
     tables[j].serving_container-=1;
     studentsremain--;
     pthread_mutex_lock(&mutex_waiting_students);
     students_waiting--;
     pthread_mutex_unlock(&mutex_waiting_students);
     pthread_mutex_unlock(&mutex_tables[j]);
     while(tables[j].slots!=0 && students_waiting!=0){

     }
     //sleep(1);
     // printf("Students ate at table %d. Slots left is
}
void wait_for_slot(int index){
```

Students

The students are initialised in a student_init function in which they receive there arrival time. As the students arriving, the counter on students_waiting increases. Once the student arrives, he polls the tables to check whether a table has a free slot, if it does he goes and sits there and calls the function student_in_slot(). In the student_in_slot function, **students sit at the table and biryani is only served when the slots are all finished, or the number of people waiting are 0 currently.** Once biryani is served the student set his status to 1 and exits.