2.BIRYANI PROBLEM

In this problem we create a thread for each chef ,table and student.

Each chef and table has lock .

Locks are used so to prevent race condition.

FUNCTION FOR CHEF

```
void biryani_ready(Chef *chef){
    printf("Robot Chef %d has prepared %d vessels of Biryani. Waiting for all the vess
els to be emptied to resume cooking\n", chef->id+1, chef->vessels); fflush(stdout);
    pthread_mutex_t *lock = &(chef->lock);
    pthread_mutex_lock(lock);
    chef->isready = 1;
    if(chef->vessels && studentsToBeServed) pthread_cond_wait(&(chef->ccond), lock);
    chef->isready = 0;
    pthread_mutex_unlock(lock);
}

void* cook_biryani(void* arg){
    Chef *chef = (Chef*) arg;
    pthread_mutex_t * lock = &(chef->lock);
    while(studentsToBeServed){
        int w = rand()%4 + 2;
        chef->vessels = rand()%10 + 1;
        chef->p = rand()%26 + 25;
        printf("Robot Chef %d is preparing %d vessels of Biryani\n", chef->id+1, chef-
>vessels); fflush(stdout);
        sleep(w);
        biryani_ready(chef);
        if(!chef->vessels) printf("All the vessels prepared by Robot Chef %d are empti
ed. Resuming cooking now\n", chef->id+1), fflush(stdout);
    }
}

void load_biryani(Table *table){
    int status = 1;
    pthread_mutex_t *clock, *tlock;
    while(status && studentsToBeServed){
        for(int i=0; i<M; i++){
            clock = &(chefs[i].lock);
            pthread_mutex_lock(clock);
            if(chefs[i].isready){
                tlock = &(table->lock);
                pthread_mutex_lock(tlock);
                table->p = chefs[i].p;
                printf("Robot Chef %d is refilling Serving Container of Serving Table
%d\n", chefs[i].id+1, table->id+1); fflush(stdout);
                printf("Serving Container of Table %d is refilled by Robot Chef %d; Ta
ble %d is resuming serving now\n", table->id+1, chefs[i].id+1, table->id+1); fflush(st
dout);
                pthread_mutex_unlock(tlock);
                chefs[i].vessels--;
                if(chefs[i].vessels == 0){
                    pthread_cond_broadcast(&(chefs[i].ccond));
                }
                pthread_mutex_unlock(clock);
                status = 0;
                break;
            }
            pthread_mutex_unlock(clock);
        }
    }
}
```

For each chef we create a thread and it then enter the func robot_chef it randomly assign value to how many vessel chef prepares and how many each student each vessel can serve . Then it enters biryani_ready it reamins here untill all the vessel becomes empty.

FUNCTION FOR TABLE

```
void ready_to_serve_table(Table *table){
    printf("Serving table %d entering Serving Phase\n", table->id+1); fflush(stdout);
    pthread_mutex_t * lock = &(table->lock);
    pthread_mutex_lock(lock);
    if(table->slots || studentsToBeServed){
        pthread_cond_wait(&(table->tcond), lock);
    }
    pthread_mutex_unlock(lock);
}

void* serve_biryani(void* arg){
    int status;
    Table *table = (Table*) arg;
    pthread_mutex_t * lock = &(table->lock);
    while(studentsToBeServed){
        status = 0;
        printf("Serving Container of Table %d is empty, waiting for refill\n", table->
id+1); fflush(stdout);
        load_biryani(table);
        if(studentsToBeServed){
            status = 1;
            pthread_mutex_lock(lock);
        }
        while(studentsToBeServed && table->p){
            status = 0;
            table->slots = rand()%10%table->p + 1;
            table->p -= table->slots;
            printf("Serving Table %d is ready to serve with %d slots\n", table->id+1,
table->slots); fflush(stdout);
            pthread_mutex_unlock(lock);
            ready_to_serve_table(table);
        }
        if(status) pthread_mutex_unlock(lock);
    }
}
```

For each table we create a new thread and then it enters the function serve_table which serches for a chef that has some vessel of buryani prepared and then chef fill the container at the table .Then each time we create some random no of slots and serve the students untill the container becomes empty .
Then we refill it again untill all students are served.

FUNCTION FOR STUDENTS

```
void student_in_slot(Student* student){
    printf("Student %d is assigned a slot on the serving table %d and waiting to be se
rved\n", student->id+1, student->tableid+1);
    fflush(stdout);
    pthread_mutex_t *lock = &(tables[student->tableid].lock);
    pthread_mutex_lock(lock);
    studentsToBeServed--;
    if(tables[student->tableid].slots == 0){
        while(tables[student->tableid].top > -1){
            printf("Student %d on Serving Table %d has been served\n", pop(tables[stud
ent->tableid].st, tables[student->tableid].top)+1, tables[student->tableid].id+1);
            tables[student->tableid].top--;
            fflush(stdout);
        }
        pthread_cond_broadcast(&(tables[student->tableid].tcond));
    }
    pthread_mutex_unlock(lock);
}


void *wait_for_slot(void* arg){
    Student *student = (Student*) arg;
    int arrivalTime = rand()%10;
    sleep(arrivalTime);
    printf("Student %d has arrived\n", student->id+1); fflush(stdout);
    printf("Student %d is waiting to be allocated a slot on the serving table\n", stud
ent->id+1); fflush(stdout);
    int status = 1;
    pthread_mutex_t *lock;
    while(status){
        for(int i=0; i<N; i++){
            lock = &(tables[i].lock);
            pthread_mutex_lock(lock);
            if(tables[i].slots != 0){
                student->tableid = tables[i].id;
                student->slot = tables[i].slots;
                tables[i].slots--;
                push(tables[i].st, tables[i].top, student->id);
                tables[i].top++;
                status = 0;
                pthread_mutex_unlock(lock);
                break;
            }
            pthread_mutex_unlock(lock);
        }
    }
    student_in_slot(student);
}
```

For each student we create a new thread and it enters wait_for_slot which search for table which has a free slots and take food from there and leaves till all students are served