Q2

Automated Biryani Serving

Approach:

There are m chefs, n tables and k children. This program aims to automate the process of giving
biryani to the students such that everyone gets a portion.
In the main function I initialise the chefs, tables and students, in new threads, using their
respective init functions.
In chef_init, using a random number generator, we initialise a chef with the number of vessels
produced, each serving x people, and in how much time. Once produced we go through all the
tables to check if it's available to deposit a vessel. We do this till all our vessels are exhausted.
This part is shown in the biryani ready function.
In tables_init, we initialise a table with an Id and the number of slots in which it serves. After
creating the table element, it goes into the ready_to_serve function to show to the chefs that it
can be used to serve biryani.
In student_init, a student is initialised with an index. Then it goes into wait_for_slot function, in
order to get a slot. Once it gets an open slot, it goes into the in_slot function, where it waits until
all the slots of the table are filled and then eats biryani. We show the eating of biryani by
decrementing the number of students, number of slots and number if available portions.
The main reason that the code runs without deadlock is the use of mutex. The mutex locks on a
memory stops other items from accessing that piece of memory thus removing any incoherency

in values. Therefore whenever an object wishes to access
some memory it checks if it has been
locked. If not, it locks it and accesses the memory.

Implementation:

```c
typedef struct chef{
    int ind;
    int time;
    int num_portion_each;
    int num_vessel_each;
}chefo;
chefo chefs[1000];
typedef struct table{
    int ind;
    int num_slots;
    int available_portion;
}tablo;
tablo tables[1000];

typedef struct student{
    int ind;
    int status;
}stud ;
stud students[1000];
```


Chefs, tables and students maintain a list of the
respective objects, each element is initialised in
the form of a struct with the respective variables
as attributes.

```
void biryani_ready(int ind){
    while(chefs[ind].num_vessel_each){
        int j;
        for1(j,N-1){
            if(!pthread_mutex_trylock(&mutex_tables[N-j])){
                if(!tables[N-j].available_portion){
                    printf("Robot Chef %d is refilling Serving Container of Serving Table %d\n", chefs[ind].ind, tables[N-j].ind);
                    chefs[ind].num_vessel_each=chefs[ind].num_vessel_each-1;
                    tables[N-j].available_portion = chefs[ind].num_portion_each;
                    printf("Serving Container of Table %d is refilled by Robot Chef %d; Table %d resuming serving now\n", tables[N-j].in
                    sleep(1);
                }
                pthread_mutex_unlock(&mutex_tables[N-j]);
            if(chefs[ind].num_vessel_each==0)    break;
            }
        }
    }
    printf("All the vessels prepared by Robot Chef %d are emptied. Resuming cooking now.\n", chefs[ind].ind);
}
```

In biryani ready the chef goes through all the
tables until it has served all the vessels it
cooks. At
each table it checks if the table is not being used
using the trylock function and if the table
doesn't contain any portions. If both check out it
serves a vessel to that table and thus it has one
vessel less and the number of portions in the table
becomes equal to the number of portions in
each vessel. After serving it unlocks the table so
it can be used for eating.

```
void *chef_init(void *args){
    int check = 0;
    int ind = *(int *)args;
    while(check==0){
        if(num_students!=0){
            chefs[ind].num_vessel_each = generator(1, 10);
            chefs[ind].time = generator(2, 5);
            printf("Robot Chef %d is preparing %d vessels of Biryani\n",chefs[ind].ind, chefs[ind].num_vessel_each);
            chefs[ind].num_portion_each = generator(25, 50);
            sleep(chefs[ind].time);
            printf("Robot Chef %d has prepared %d vessels of Biryani. Waiting for all the vessels to be emptied to resume cooking\n
            biryani_ready(ind);
        }
        else  break;
    }
}
```

In chef_init we pass the id of the chef as argument. In the function the chef with the id passed is
randomly given attributes using a generator function. After that we make the process wait for a set amount of time till the chef cooks the biryani. After this the chef has cooked the biryani and will search for a table to serve it in using the biryani_ready function.

```c
void ready_to_serve_table(int num_slots, int ind){
    int check = 0;
    while(check==0){
        if(pthread_mutex_trylock(&mutex_tables[ind])==0){
            if(num_students==0 || !tables[ind].num_slots ){
                pthread_mutex_unlock(&mutex_tables[ind]);
                printf("Serving table %d entering Serving Phase\n", tables[ind].ind);


                break;
            }
            pthread_mutex_unlock(&mutex_tables[ind]);
        }}
}
```

```
void *table_init(void *args){
    int ind = *(int *)args;
    int check = 0;
    while(check==0){
        if(num_students==0)
            break;
        if(tables[ind].available_portion!=0){
            int num_slots = generator(1, 10);
            if(tables[ind].available_portion<num_slots)
                tables[ind].num_slots = tables[ind].available_portion;
            else tables[ind].num_slots = num_slots;

            printf("%d slots are ready to be served by Table %d\n", tables[ind].num_slots, tables[ind].ind);
            ready_to_serve_table(tables[ind].num_slots, tables[ind].ind);
        }}
    }
```

In table_init the table the id of the table is
passed through as an argument. In the function the
table with the passed id is initialised as a
serving table if it contains biryani by defining
the
number of slots it will serve in. After
initialising it is passed through the
ready_to_serve_table
function so that students know that the specific
table is open for eating.
In the ready_to_serve function the table
essentially waits till the number of slots or
number of
remaining students becomes 0. After this the table
mutex unlocks so it can take more biryani
from the chef.

```c
void student_in_slot(int ind, int j){
    tables[j].available_portion=tables[j].available_portion-1;
    if(!tables[j].available_portion)
        printf("Serving Container of Table %d is empty, waiting for refill\n", tables[j].ind);
    num_students-=1;
    tables[j].num_slots-=1;
    pthread_mutex_unlock(&mutex_tables[j]);
    while(num_students!=0 && tables[j].num_slots!=0){
    }
    printf("Student %d on serving table %d has been served\n", students[ind].ind, tables[j].ind);
}
```

```c
void wait_for_slots(int ind){
    int check = 0;
    printf("Student %d is waiting to be allocated a slot on the serving table\n", ind);
    while(check==0){
        int j = N;
        while(j--){
            if(!pthread_mutex_trylock(&mutex_tables[N-j])){
                if(tables[N-j].num_slots!=0){
                    printf("Student %d assigned a slot on the serving table %d and waiting to be served.\n", ind, N-j);
                    check = 1;
                    student_in_slot(ind, N-j);
                    pthread_mutex_unlock(&mutex_tables[N-j]);
                    break;
                }
                pthread_mutex_unlock(&mutex_tables[N-j]);
            }
        }
    }
}
```

```c
void *student_init(void *args){
    int ind = *(int *)args;
    printf("Student %d has arrived.\n", ind);
    wait_for_slots(ind);
}
```

In student_init we pass the id as the argument.
Once the student is initialized it waits for an
available slot by entering the wait_for_slot
function.
In wait_for_slot it continuously goes through all
the tables and check whether it is not locked
and there are available slots. If so it is assigned
a slot and enters the in_slot function.
In the in_slot function the student waits till all
slots are full in the table and then it gets
biryani

after which it immediately leaves and the table in which it was eating becomes free and is therefore unlocked so it can be used again.

```
for1(i,M-1){
    usleep(100);

    chefs[M-i].ind = M-i;
    pthread_create(&cheftids[M-i], NULL, chef_init, (void *)&chefs[M-i].ind);
}


for1(i,N-1){
    usleep(100);

    tables[N-i].ind = N-i;
    pthread_create(&tabletids[N-i], NULL, table_init, (void *)&tables[N-i].ind);
}

for1(i,K-1){
    usleep(100);
    students[K-i].ind = K-i;
    pthread_create(&studenttids[K-i], NULL, student_init, (void *)&students[K-i].ind);
}
for1(i,K-1)
    pthread_join(studenttids[K-i],NULL);
```

This is a part of the main function. In this we continuously initialise all the students, tables and
chefs in separate threads. At the end we wait for all students to finish by using the join function.

Name - Zia Bashir

Roll no - 2018101006