REPORT
QUESTION 2

Submitted by : Tanvi Kamble
2018114004

Aim: Implementing a Biryani system

Code:

- Global variables:
  - int robo_prep[1000], robo_v[1000]---------- robo_prep = time taken by each chef and robo_v number of vessels prepared by each chef
  - int front = -1, rear = -1, size = 100---------- for queue of students
  - int table_free[100]---------- check if container is full
  - portion_alot[100]----------number of portions left in each container
  - Table_cap[100]---------- number of slots at any point of time
  - Stud_queue[100]---------- students waiting in the queue
  - stud_stat[100]---------- check if student has gotten the biryani or not
  - int bir_cooked[100]---------- when chef is ready with the vessels
  - no_fed = 0;---------- total number of students fed till now
  - int m, n, k; ----------m = no. of chefs n = no of tables, k = no. of students
  - struct cap *ves_cap[100];---for each chef the capacity of each vessel prepared
  - struct info *tab_info[100];----------stores the info about which vessel of which chef is placed on the table

FUNCTIONS USED

- Enqueue:

```c
void enQueue(int value, int arr[])
{
    if ((front == 0 && rear == size - 1) ||
        (rear == (front - 1) % (size - 1)))
    {
        printf("\nQueue is Full");
        return;
    }

    else if (front == -1) /* Insert First Element */
    {
        front = rear = 0;
        arr[rear] = value;
    }

    else if (rear == size - 1 && front != 0)
    {
        rear = 0;
        arr[rear] = value;
    }

    else
    {
        rear++;
        arr[rear] = value;
    }
    //sleep(1);
}
```

- Used to insert elements in the student queue

➢ total_enque():

```c
int total_inq()
{
    int sz;
    if (front == -1 && rear == -1)
    {
        sz = 0;
    }
    else
    {
        sz = front > rear ? (size - front + rear + 1) : (rear - front + 1)
        ;
    }
    return sz;
}
```

- Finds the total number of students present in the queue at a given time

➢ Dequeue:

```
int deQueue(int arr[])
{
    if (front == -1)
    {
        printf("\nQueue is Empty");
        return -1;
    }

    int data = arr[front];
    arr[front] = -1;
    if (front == rear)
    {
        front = -1;
        rear = -1;
    }
    else if (front == size - 1)
        front = 0;
    else
        front++;

    return data;
}
```

- Extract the student present in the front

➢ Wait_for_slot():

```
void wait_for_slot(int i)
{
    while (1)
    {
        if (i == stud_queue[front])
        {
            break;
        }
    }
    while(i == stud_queue[front]){
        continue;
    }
    printf("Student %d ready to be assigned\n");
    return;
}
```

- Student calls this function
- This returns when the student is ready to get a slot

➢ Robothread:

```
pthread_mutex_lock(&mutex_wait);
struct serve *ki = i;
int num = ki->num;
// pthread_mutex_lock(&mutex[n]);
pthread_mutex_unlock(&mutex_wait);
while (1)
{
    if (no_fed == k)
    {
        break;
    }
    robo_prep[num] = rand() % 4 + 2;
    robo_v[num] = rand() % 10 + 1;
    int cap_count = 0;
    ves_cap[num] = malloc(sizeof(struct cap));

    while (cap_count < robo_v[num])
        ves_cap[num]->a[cap_count++] = rand() % 26 + 25;
    //printf("halo\n");
    printf("robot %d, vessels %d, prep time %d\n", ki->num, robo_v[num], robo_prep[num]);

    sleep(robo_prep[num]);
    printf("Robot %d finished cooking %d vessels in %d time.\n", num, robo_v[num], robo_prep[num]);
    biryani_ready(num);
```

- Called when thread creation
- Randomly generates the prep time and vessels with their respective portions
- Robot starts cooking
- After the food is cooked call biryani_ready()

➢ Biryani ready:

- If a table is found empty load the container of the chef.
- If all the tables are full wait till a table becomes empty and then assign the container
- Returns only when all the vessels of the chef are assigned to the tables and are all empty

```c
pthread_mutex_lock(&mutex_serve[i]);
bir_cooked[i] = 1;
int c = 0;
while (c < robo_v[i])
{
    for (int j = 1; j <= n && c < robo_v[i]; j++)
    {
        if (table_free[j] == 0)
        {
            portion_alot[j] = ves_cap[i]->a[c++];
            table_free[j] = 1;
            printf("Service table %d has been given %d portions from %d chef container %d\n", j, portion_alot[j], i, c);
            tab_info[j]->chef = i;
            tab_info[j]->v_no = c;
        }
    }
    if (c < robo_v[i])
    {
        int x = 1;
        while (1)
        {
            while (x <= n)
            {
                if (table_free[x] == 0)
                {
                    break;
                }
                x++;
            }
```

➢ Tablethread:

- Called when the thread for serving is created
- as soon as biryani is loaded, randomly generate slots which are no more than the portions left in the container and the students in the queue.
- Call ready to serve
- If portion = 0 when slots aren't full wait for another container by changing table_free to 0
- When total slots (100) are full or all students are fed return

```
if (table_free[num] == 1)
{
    table_cap[num] = rand() % 10 + 1;
    if (table_cap[num] > portion_alot[num])
    {
        table_cap[num] = portion_alot[num];
    }
    if (table_cap[num] > total_inq())
    {
        table_cap[num] = total_inq();
    }

    int served = ready_to_serve(num, table_cap[num], served);

    //printf("For table %d portion left = %d, table_total = %d, no__fed = %d\n", num, portion_alot[num], table_total[num]->num, no_fed);
    if (portion_alot[num] == 0)
    {
        int temp = tab_info[num]->chef;
        int ves_num = tab_info[num]->v_no;
        ves_cap[temp]->a[ves_num] = 0;
        printf("Serving Container of Table %d is empty, waiting for refill\d", num);
    }
    if (portion_alot[num] == 0 && table_total[num]->num != 0 && no_fed != k)
    {
        table_free[num] = 0;
    }
}
if (table_free[num] == 0)
{
    portion_alot[num] = 0;
    table_cap[num] = 0;
    //pthread_cond_signal(&cond[num]);
}
if (table_total[num]->num == 0 || no_fed == k)
{
    break;
}
```

➢ Ready_to_serve:

```
printf("Serving Table %d is ready to serve with %d slots\n", i, no);
table_total[i]->num -= no;
portion_alot[i] -= no;
table_cap[i] = 0;
no_fed += no;
for (int j = 0; j < no; j++)
{
    if (front == -1)
    {
        j--;
        continue;
    }
    int temp = deQueue(stud_queue);
    stud_stat[temp] = 1;
    printf("Student %d assigned a slot on the serving table %d and waiting to be served\n", temp, i);
    served++;
}
return served;
```

- For the given number of students, dequeue the student, assign him a slot in the table and decrease the slots remaining and portions accordingly

➢ StudentThread:

- Called while creating the student thread
- Whenever a student comes add to the queue and call wait_for_slot until dequeued

```
pthread_mutex_lock(&mutex_wait);
struct serve *ki = i;
int num = ki->num;
// pthread_mutex_lock(&mutex[n]);
printf("Student %d arrived\n", num);
enQueue(num, stud_queue);

//sleep(1);
pthread_mutex_unlock(&mutex_wait);
printf("Student %d is waiting to be allocated a slot on the serving table\n", num);
wait_for_slot(num);
```