Code breakdown for Automated Biryani Serving

Overview

Robo chef threads produce a random number of vessels with random capacity if they have 0 vessels left. Tables threads check if any robo chef has vessels remaining to fill its container (takes 1 vessel at a time). Student threads wait for random amount of time to simulate random arrival of students and keep checking if any table has slots. If they are there, it occupies that table slots and exits from the system (thread terminates) after eating.

Run the program

1. Run the command `gcc -pthread -o biryani biryani.c` .
2. Run `./biryani`
3. Input values for number of chefs, tables, and students for the simulation to begin.

Variables

Structs for chefs, tables, and students

- Chefs: Each Chef struct has details about chef number, vessels, and a mutex to ensure only one table reads and alters its information at a time. It also stores the Chef thread ID.

```
struct Chef
{
    int idx;
    int status;
    int vessels_left;
    int capacity;
    pthread_mutex_t chef_mutex;
    pthread_t chef_thread_id;
    pthread_cond_t cv_table;
};
```

- Tables: Each Table struct has details about table number, occupancy ,slots, and a mutex to ensure only one student reads and alters its information at a time. It also stores the table thread ID.

```
struct Table
{
    int idx;
    int capacity;
    int slots;
    int occupancy;
    pthread_mutex_t table_mutex;
    pthread_t table_thread_id;
    pthread_cond_t cv_student;
};
```

- Students : Each Student struct has details about student number and stores the student thread ID.

```
struct Student
{
    int idx;
    pthread_t student_thread_id;
};
```

Arrays

An array of the above structs is made as follows:

```
Chef chefs[1000]; Table tables[1000]; Student students[1000];
```

Int

- `waiting_students` : Keeps track of the number of students waiting at a time.

Functions

Chef Functions

Chef thread function

```
void * chef_thread(void* args)
{
    Chef * chef = (Chef*)args;

    while(1)
    {
        int r = rand()%10 + 1;
        printf("%sCHEF %d PREPARING %d VESSELS\n",KRED, chef->idx, r);
        int w = rand()%4 + 2;
        sleep(w);
        pthread_mutex_lock(&(chef -> chef_mutex));
        chef->vessels_left = r;
        chef->capacity = rand()%6 + 20;
        printf("%sCHEF %d HAS PREPARED %d VESSELS WITH CAPACITY %d\n",KRED, ch
        biryani_ready(chef);
    }

    return NULL;
}
```

- Sleeps for random amount of time to simulate preparation time
- Generates number of vessels and their capacity randomly.
- Calls `biryani_ready` function

biryani_ready

```
void biryani_ready(Chef *chef)
{
    while(1)
    {
        if(chef -> vessels_left == 0)
            break;
        else pthread_cond_wait(&(chef->cv_table), &(chef->chef_mutex));
    }

    printf("%sALL VESSELS PREPARED BY CHEF %d ARE EMPTY, RESUMING COOKING NOW\
    pthread_mutex_unlock(&(chef -> chef_mutex));
}
```

- Conditional waits to avoid busy waiting
- Returns to Robo chef when vessels_left become 0

Tables Functions

Tables thread function

- Loops through array of robo chefs in a while(1) loop to check if any chef has vessels available
- Accesses robo chefs values bounded inside the chef mutex so that only *1 table reads that chef's information at a time*

```
 for(int i=0; i < no_chefs; i++)
{
    pthread_mutex_lock(&(chefs[i].chef_mutex));
    if(chefs[i].vessels_left > 0)
    {
        ...
    }

    pthread_cond_signal(&(chefs[i].cv_table));
    pthread_mutex_unlock(&(chefs[i].chef_mutex));
}
```

- Upon finding a robo with vessel, the table decrements the chef's vessels variable, signals to conditional wait thread, and breaks
- Otherwise continues checking for availability of vessels
- Upon filling its serving container, it locks its mutex, and generates slots randomly.

```
    pthread_mutex_lock(&(table->table_mutex));
    table -> slots = rand() % 10 + 1;
```

- Then it calls `ready_to_serve` function

ready_to_serve

- Waits on the conditional variable until either

1. No waiting students:

```
    pthread_mutex_lock(&(waiting_mutex));
    if(waiting_students == 0)
    {
        printf("%sNO WAITING STUDENTS CURRENTLY, TABLE %d SLOTS EMPTIED\n",KYE
        pthread_mutex_unlock(&(waiting_mutex));
        break;
    }

    pthread_mutex_unlock(&(waiting_mutex));
```

2. All slots full:

```
    if(table -> slots == table -> occupancy)
    {
        printf("%sALL SLOTS FILLED FOR TABLE %d, RECHECKING CAPACITY\n",KCYN,t
        break;
    }
```

- Conditional waits otherwise

```
    else pthread_cond_wait(&(table-> cv_student), &(table-> table_mutex));
```

Student Functions

Student Thread function

- Sleeps for random time to simulate arrival of students at random times
- Calls `wait_for_slot` function

wait_for_slot

- Iterates over Tables aray to check if any has a free slot in a while(1) loop

```
for(int i=0; i < no_tables; i++)
{
    pthread_mutex_lock(&(tables[i].table_mutex));
    if(tables[i].slots - tables[i].occupancy > 0)
    {
        student_in_slot(i, idx);
    }

    pthread_cond_signal(&(tables[i].cv_student));
    pthread_mutex_unlock(&(tables[i].table_mutex));
}
```

- Calls `student_in_slot` function

student_in_slot

```
void student_in_slot(int i, int idx)
{
    pthread_mutex_lock(&(waiting_mutex));
    waiting_students--;
    pthread_mutex_unlock(&(waiting_mutex));
    printf("%sSTUDENT %d WAITING FOR SLOT AT TABLE %d\n",KRED,idx, i+1);
    printf("%sSTUDENT %d EATING AT TABLE %d\n",KGRN,idx, i+1);
    pthread_cond_signal(&(tables[i].cv_student));
    pthread_mutex_unlock(&(tables[i].table_mutex));
    return;
}
```

- Decreases global variable `waiting_students` inside a mutex
- Unlocks mutexes and prints when student sits, and eats

Main

- Takes inputs for number of chefs, tables, and students
- Initializes the random variables, structs, and mutexes
- Waits for all student threads to join (all students have eaten)
- Destroys mutexes and terminates the program