Umang Srivastava
2019101090

OSN Assignment 4

Q3 : Musical Mayhem

Overview

In this code, we create separate threads for musicians/singers and coordinators. Musicians check for the availability of the associated type of stage to perform and coordinators check for musician/singer who has completed his/her performance and will now get the T-shirt.

Variables and Data Structures

1. Composite Variables

- **Musician**

```
typedef struct Musician
{
      int id; // id of the musician
      int arrivalTime; // arrival time of the musician
      int stageType; // type of the stage on which musician performs...
1-Acoustic  2-Electric  3-Both
      int maxWaitTime; // maximum waiting time of the musician
      int PerformanceTime; // time for which musician will perform
      char instrumentChar; // instrument character of the musician
      int stage_no; // number of the stage on which musician performs
      int status; // performance status of the musician... 1-Performing
0-Not Performing
      char name[25]; // Name of the musician
      sem_t collect; // semaphore
      pthread_t musician_thread_id; // thread for the musician
      pthread_mutex_t musician_mutex; // mutex for the musician
```

```
} Musician;
```

- **Stage**

```
typedef struct Stage
{
      int id; // id of the stage
      int type; // type of the stage... 1-Acoustic  2-Electric
      int status; // status of the stage... 1-Occupied  0-Available
      int curr_mus; // ID of the musician currently performing on the stage
      bool singerflag; // flag to check if a singer is performing on the
stage or not
      pthread_mutex_t stage_mutex; // mutex for stage
} Stage;
```

- **Coordinator**

```
typedef struct Coordinator
{
      int id; // id of the coordinator
      int status; // status of the coordinator... 1-Busy  0-Availale
      int musician_no; // ID of the musician who is taking T-Shirt from
this coordinator
      pthread_t coordinator_thread_id; // thread for coordinator
} Coordinator;
```

2. Arrays

```
Stage stages[1000]; // array of stages
Musician musicians[1000]; // array of musicians
Coordinator coordinators[1000]; // array of coordinators
```

Functions

1. **musician_thread**

   Sleeps for **arrivalTime** to stimulate arrival of musicians.

   clock() is used to keep track of time elapsed.

```
bool wait_time_over = 0;
clock_t tpp = clock();
      while(!stage_alloc)
      {
            clock_t time_taken = clock() - tpp;
            int times = (int)(time_taken/CLOCKS_PER_SEC)-7;
            if(times > m->maxWaitTime)
            {
                  printf("%sMusician %s waited for %d
seconds\n\n",RED,m->name,m->maxWaitTime);
                  wait_time_over = 1;
                  break;
            }
```

   We loop through the stages and check for the availability for musicians/singers. And choose the corresponding stage accordingly.

   Mutexes are present so that only one musician has access at a given time.

```
if(m->stageType == 1)
{
      for(int i=0;i<a;i++)
      {
            pthread_mutex_lock(&(stages[i].stage_mutex));
            if(stages[i].status == 0)
            {
                  // printf("Musician %s alloted stage %d\n\n", &m->name,
stages[i].type);
                  stage_alloc = stages[i].status = 1;
                  stages[i].curr_mus = m->id-1;
                  m->stage_no = i+1;
```

```
                pthread_mutex_unlock(&(stages[i].stage_mutex));
                break;
            }
            pthread_mutex_unlock(&(stages[i].stage_mutex));
      }
}
else if(m->stageType == 2)
{
      for(int i=a; i<tot_stages; i++)
      {
            pthread_mutex_lock(&(stages[i].stage_mutex));
            if(stages[i].status == 0)
            {
            // printf("Musician %s alloted stage %d\n\n", &m->name,
stages[i].type);
            stage_alloc = 1;
            m->stage_no = i+1;
            stages[i].status = 1;
            stages[i].curr_mus = m->id-1;
            pthread_mutex_unlock(&(stages[i].stage_mutex));
            break;
      }
      pthread_mutex_unlock(&(stages[i].stage_mutex));
}
```

If the wait time exceeds the maximum waiting time, the message is displayed.

```
if(wait_time_over)
{
      printf("%sMusician %s %c left because of
impatience!\n\n",MAG,m->name, m->instrumentChar);
      return NULL;
}
```

Else, the performing of the musician/singer is printed, along with the stage type and number.

```
if(m->instrumentChar !='s')
{
      if(stages[m->stage_no-1].type == 1)
            printf("%sMusician %s performing %c at Acoustic stage number %d
for %d seconds\n\n",BLU,m->name, m->instrumentChar,stages[m->stage_no-1].id
, m->PerformanceTime);
      else
            printf("%sMusician %s performing %c at Electric stage number %d
for %d seconds\n\n",BLU,m->name,
m->instrumentChar,stages[m->stage_no-1].id, m->PerformanceTime);
}
else
{
      .
      .
      .
      //Code for performance
```

Now sleep for the **performanceTime .**

```
sleep(m->PerformanceTime);
```

We change the state of stage on which the musician/singer is performing within the mutexes. Then signals the **musician_get_tshirt** semaphore that musician/singer is ready to get T-Shirt

```
pthread_mutex_lock(&(stages[no].stage_mutex));
if(m->instrumentChar != 's')
{
      if(stages[no].status==1)
      {
            stages[no].status = 0;
      }
      else
      {
            stages[no].status = 1;
      }
}
.
.
```

```
//Code for performance finish
.
.
sem_post(&musician_get_tshirt);
pthread_mutex_unlock(&(stages[no].stage_mutex));
sem_wait(&(m->collect)); // wait for musician/singer to collect t
shirt
```

2. **coordinator_thread**

We wait for the **musician_get_tshirt** semaphore in while loop.

```
Coordinator * crd = (Coordinator *)args;
while(1>0)
{
      sem_wait(&musician_get_tshirt);
      crd->status = 1; // coordinator is busy finding tshirt
```

In the critical section, we check for musicians/singers who have completed the performance. To give T-Shirt to a musician/singer, we lock mutex so that only one musician/singer gets the t-shirt at a given moment.

```
for(int i=0;i<k;i++)
{
      pthread_mutex_lock(&(musicians[i].musician_mutex));
      if(musicians[i].status != 0)
      {
            crd->musician_no = i+1;
            if(musicians[i].instrumentChar != 's')
                  printf("%sMusician %s collecting T-shirt\n\n", WHT,
musicians[i].name);
            else
                  printf("%sSinger %s collecting T-shirt\n\n", WHT,
musicians[i].name);
            musicians[i].status = 0;
            pthread_mutex_unlock(&(musicians[i].musician_mutex));
```

```
            break;
      }
      pthread_mutex_unlock(&(musicians[i].musician_mutex));
}
```

We use **sleep(2)** to show that it takes 2 seconds for a coordinator to find the correct size tshirt. Then we print that the musician/singer is exiting and post to the musician's collect semaphore so that it can exit from the system.

```
sleep(2);
if(musicians[crd->musician_no-1].instrumentChar != 's')
      printf("%sMusician %s is exiting\n\n",BLU,
musicians[crd->musician_no-1].name);
else
      printf("%sSinger %s is exiting\n\n",BLU,
musicians[crd->musician_no-1].name);
sem_post(&(musicians[crd->musician_no - 1].collect));
crd->musician_no = 0;
crd->status = 0;
return NULL;
```

3. main

Take input:

- Number of Musicians/Singers
- Acoustic stages,
- Stages
- Coordinators
- Max Waiting Time
- Lower Limit of Performance Time
- Upper Limit of Performance Time

- Name, Instrument character and arrival Time of all musicians/singers.

Initialize all variables.

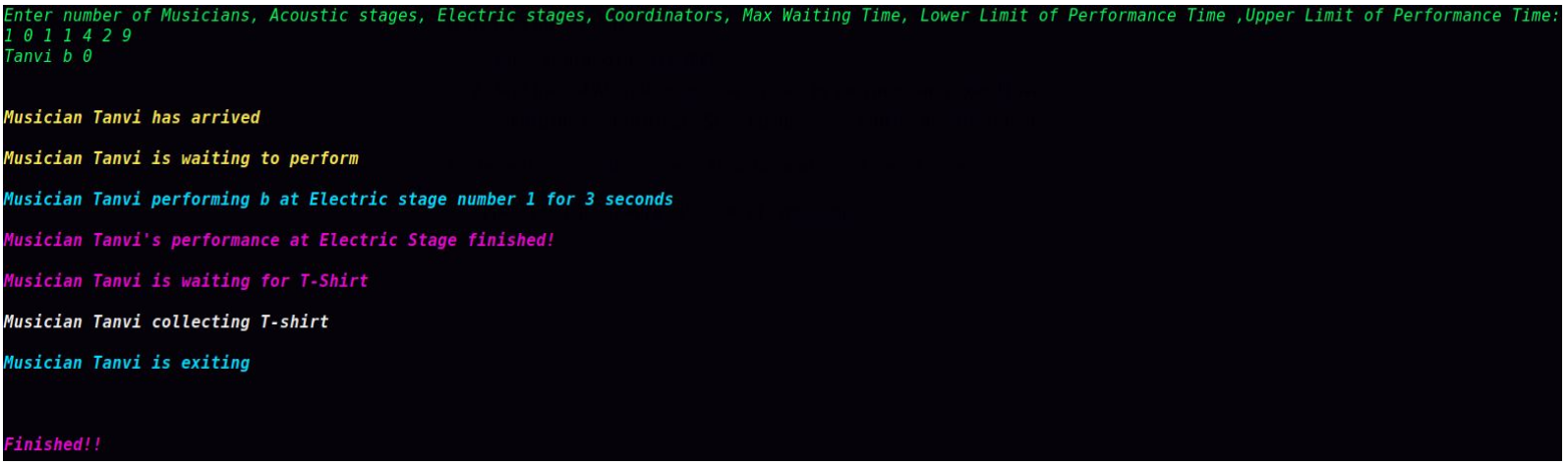
Create musician threads and wait to join them all.

BONUSES implemented:

1. Singers can also collect T-Shirts.
2. Kept track of which stage number is joined by each musician/singer. I have printed the Acoustic/Electric Stage number along with the musician/singer.

Implementation of all 7 statuses of Musician and all 5 statuses of singer

A snippet of output , showing all 7 statuses of a musician.



Assumptions:

For computation of waiting time, I subtract 7 from the calculated time and then compare current elapsed time since I tried many test cases and found that comparing the exact elapsed time to maxWaitTime, I get wrong output because the print and other function in program take a small time to execute. By trial and error, I found that subtracting 7 solved the issue for the maximum number of cases.