Musical Mayhem

Structure of Implementation

Each stage is a struct which stores the id of the stage, the details of the musician performing, the status of the stage(Number of people performing on that stage) and the time of performance.

There is an array of stages which stores the pointer to the struct of every stage at the index equal to the id of the stage.

There are three type of semaphores regarding stages:

1. acoustic
2. electrical
3. duo

```
sem_t tshirt; // semaphore for allowing only less than a particular number
of students from taking tshirts simultaneously

sem_t acoustic; // Semaphore for acoustic stages.

sem_t electric; // Semaphore for electric stages

sem_t duo; // Semaphore for dual performances
```

Acoustic semaphore make sure that only certain number of perfomers can give performances at the acoustic stages. Electric semaphores do the same.Acoustic and electrical semaphores are used by both musicians and singers. duo semaphore is explicitly for singers.duo semaphore make sure that a singer can join a stage whenever a stage with musician is available.

```
struct stage
{
        int stageid; // To access the stage

        char personname[100]; // To know the name of musician performing on
the stage

        int personid; // Id of person performing on the stage

        pthread_mutex_t stagelock;

        int status; // To know how many people are performing on the stage
. Can be 0 , 1 ,2 (musician and a singer).

        int type; // Acoustic or electric
```

```
        int stagetime; // Time of performance of player on that stage
};

struct stage * stagearr[1000]; // To store the stage details so that others
can acess or update
```

MUSICIANS AND SINGERS ARRIVAL

I am considering both Musicians and singers under the group of performers. Here each performer is a thread. Each performer would have seperate requirements regarding stage. Singer can join a musicians performance or can make a solo performance.

```
//performer struct


struct performer
{
        // Musician and Singer both are performers
        char name[100]; // Name of the performer

        pthread_t pid;

        pthread_mutex_t performerlock;

        int performerid; // Id to access

        char singername[100]; // Name of the singer, if a singer joined the
performer

        int stage; // Stage number on which the performer is performing

        int tshirtstatus; // Collected or waiting

        int instrument; // Instrument that the performer plays

        int win; // For race condition . (Who got the semaphore first will
win the race).

        int impatienttime; // t given

        int performancetime; // Time for which peformer performs

        int waittime; // Arrival time
};
```

After a performer arrives , based on his requirements If the performer is able to join only one type of stage we will make him to do a timed wait on that particular semaphore. if he(musician) is able to do performance at more electrical and acoustic stages we make two threads one for acoustic stage and another for electic both sharing the same performer thread. Now the two threads go and wait on their respective semaphores(acoustic and electric). Which thread gets first pass the semaphore will change the performer->win to 1 and does the remaining work,so that the other thread will simply exits after entering without doing anything.

If the performer is a singer then we make three threads sharing a commom performer struct one for electric ,one for acoustic and another for duo performances. Whichever gets past the semaphore first will proceed further and remaining ones will simply exits after getting their respective semaphore.

If no semaphores are available the Performer waits until its max wait time.

There is no Busy waiting here and threads just wait on the semaphores.

```
void * performerfun(void * args)
{

        struct performer * per = (struct performer *)args;
        //      printf("HELLO WORLD %d \n",per->instrument);
        sleep(per->waittime);  // To simulate the arrival time
        printf(COLOR_RED"%s %c arrived\n"COLOR_RESET,per->name,
((char)reassign(per->instrument)));

        // DIVIDE THEM ACCORDING TO THEIR REQUIREMENTS
        // for those perform only on one stage, there is no need to have
extra threads
        //For musicians we must have one instance waiting on the acoustic
semaphore and another on the electric semaphore
        // For singers in additional we need to wait on the duo semaphore ,
since the singer can join solo performances.
        if(per->instrument != 5)
        {
                if(per->instrument == 3) // Only on acoustic stage
                {
                        acousticfun((void *)per);
                }
                else if(per->instrument == 4) // Only on electric stage
                {
                        electricfun((void *)per);
                }
                else // On bothe stages
                {
                        // Make two instances (simply two threads) and the
one which past their respective semaphore will survive.
                        pthread_t aa,bb,cc;
                        pthread_create(&(aa), NULL, acousticfun, (void *)
(per));
                        pthread_create(&(bb), NULL, electricfun, (void *)
(per));
                        pthread_join(aa, NULL);
```

```
                        pthread_join(bb, NULL);
                }

        }
        else if(per->instrument == 5) // For singer
        {
                pthread_t aa, bb, cc ;
                pthread_create(&(aa), NULL, acousticfun, (void *) (per));
                pthread_create(&(bb), NULL, electricfun, (void *) (per));
                pthread_create(&(cc), NULL, duofun, (void *) (per));
                pthread_join(aa, NULL);
                pthread_join(bb, NULL);
                pthread_join(cc, NULL);
        }
        return NULL;

}
```

If the thread find a semaphore before its maxtime, then it go through the array of stages and finds a valid stage for itself. When searching for a stage a stagelock is employeed. We maintain a mutex lock for every stage. We employ the lock no two performers should take up the same stage. To solve the issue we employ a lock. For a musician valid stage is empty stage satisfying his requirements (acoustic/electrical).A musician changes the status of the stage to 1. For a singer it can be a empty stage or stage containig containing only the musician(duo semaphore). If a singer does a solo performance If the semaphore allows them the there is an valid stage for them for sure.

LEAVING THE STAGE

After the performance get ended , the musician thread checks the status of the stage (that he is in now), if it changes to 2 it means a singer has joined him .So he sleeps for two more seconds to simulate the performance extension by two seconds. When a singer joins a musician he fills the singername attribute in the performer struct with the singer name. There is a possibity that a singer enters and in the search of the stage meanwhile the performer may exits . The singer enters knowing that a musician might exists but by that time of search he may exit. So counter this situation we have used sem_trywait so that a singer who has come by acknowledging a musician will always find a musician.

After the exits he makes the status of stage set back to 0.

```
// Here the for loop is emplyed for running just once since it gives
facility to break :)
 for(int kk = 0;kk<1;kk++)
{
        pthread_mutex_lock(&((stagearr[per->stage])->stagelock));
        if((stagearr[per->stage])->status == 2) // check if any singer has
joined him or not
        {

                yes = 1;
                stagearr[per->stage]->status = 0;
```

```
                pthread_mutex_unlock(&((stagearr[per->stage])->stagelock));
                sleep(2);
                break;
        }
        pthread_mutex_unlock(&((stagearr[per->stage])->stagelock));

        if((stagearr[per->stage])->status != 2)
        {

                int val = sem_trywait(&duo);
                if(val==-1) // Check if any singer has pastthrough the
semaphore and waiting for the slot
                {
                        sleep(1);
                        yes = 1;
                        sleep(2); // if yes decrement the value;
                        pthread_mutex_lock(&((stagearr[per->stage])-
>stagelock));
                        stagearr[per->stage]->status = 0;
                        pthread_mutex_unlock(&((stagearr[per->stage])-
>stagelock));

                }
                else
                {               // reset the values and exit
                        pthread_mutex_lock(&((stagearr[per->stage])-
>stagelock));
                        stagearr[per->stage]->status = 0;
                        pthread_mutex_unlock(&((stagearr[per->stage])-
>stagelock));
                }
        }
}
```

COLLECTING T-SHIRTS

After the exit the musician enters the coordiantor function to collect the t-shirts. We employ tshirt semaphore there. So that at any time only k(Number of co-ordinators) number of students of students can be collecting tshirts.

For a singer if he does a solo performance he collects just like any other musician , but if he joins a duo then after the exit the musician with which the singer has joined will create a new thread for the singer and initialize the pariticulars of the singer and let it go the co-ordinator function for collecting t-shirts.

```
// Co-ordinator semaphore regulates that only some particular number of
students can be collecting the tshirts
void * coordinator(void *args)
{
        struct performer * per= (struct performer * )args;
```

```
        sem_wait(&tshirt);
        printf(COLOR_YELLOW"%s collecting tshirt\n"COLOR_RESET, per->name
);
        sem_post(&tshirt);
        return NULL;

}
```

OTHER IMPORTANT FUNCTIONS

```
// Function where singer search for a musician to join
void * duofun(void * args)
{
        int yes = 0;
        struct performer * per = (struct performer *)args;

        struct timespec * tm3 = (struct timespec *)malloc(sizeof(struct
timespec)); // to let the singer wait until the maximum waiting time;
        clock_gettime(CLOCK_REALTIME , tm3);
        tm3->tv_sec +=tseconds;


        int rt = sem_timedwait(&duo, tm3);// To ensure only a particular
number of people are allowed at a instant of time

        pthread_mutex_lock(&per->performerlock); // Since all the instance
threads of a performer may be try to change this simulatenously
        // To avoid that we employ a lock.
        if(per->win == 1)
        {
                pthread_mutex_unlock(&per->performerlock);
                if(rt!=-1)      // Increase the semaphore value only if
enters before max waiting time.
                {
                        sem_post(&duo);
                }
                return NULL;
        }
        else
        {
                per->win = 1;           // Change the value of win so that
only one instance of performer can execute
        }
        pthread_mutex_unlock(&per->performerlock);

        if(rt == -1)// Sem_timedwait return -1 when the maxwaittime exceeds
        {
                printf("%s left beccause of impatience\n", per->name);
                return NULL;
```

```
        }
        else
        {

                for(int i=0;i<numstages;i++)
                {
                        pthread_mutex_lock(&(stagearr[i]->stagelock)); //
Since the stage array is accessed by many performer threads to avoid
perfection issues , employ a lock


                        if(stagearr[i]->status == 1) // A singer can enter
if and only if there is a musician performing.
                        {
                                stagearr[i]->status = 2; // Change the
status of the stage to represent that the singer has joined the stage.
                                printf(COLOR_CYAN" Singer %s joined %s
performance ont the stage %d, performance extended by 2
seconds\n"COLOR_RESET, per->name , stagearr[i]->personname, i);
                                //sem_post(&duo);
                                strcpy(performerarr[stagearr[i]->personid]-
>singername,per->name ); // To the performer know the name of the singer
who joined
                                // The perfomer use the name when creating
the thread for singer to collect t-shirt
                                per->stage =i;  // To record on which does
the performance is going on
                                pthread_mutex_unlock(&(stagearr[i]-
>stagelock));
                                break;
                        }

                        pthread_mutex_unlock(&(stagearr[i]->stagelock));
                }


        }
        return NULL;
}
```

ELECTRIC FUNCTION

```
// The performer searches for an Electric stage.
void * electricfun(void * args)
{
        int yes = 0;
        struct performer * per = (struct performer *)args;

        struct timespec * tm2 = (struct timespec *)malloc(sizeof(struct
timespec));// to let them wait only for a certain time
```

```
        clock_gettime(CLOCK_REALTIME , tm2);
        tm2->tv_sec +=tseconds;
        //printf(" %s WAITING FOR %d secondds in electric\n",per-
>name,tseconds);



        int rt = sem_timedwait(&electric, tm2);// To ensure only a
particular number of people are allowed at a instant of time


        pthread_mutex_lock(&per->performerlock); // Since all the instance
threads of a performer may be try to change this simulatenously
        // To avoid that we employ a lock.

        if(per->win == 1)
        {

                pthread_mutex_unlock(&per->performerlock);// Increase the
semaphore value only if enters before max waiting time.

                if(rt!=-1)
                {
                        sem_post(&electric);
                }
                return NULL;
        }
        else
        {
                per->win = 1;           // Change the value of win so that
only one instance of performer can execute

        }
        pthread_mutex_unlock(&per->performerlock);

        if(rt == -1) // Sem_timedwait return -1 when the maxwaittime
exceeds
        {
                printf("%s left beccause of impatience\n", per->name);
                return NULL;
        }
        else
        {

                for(int i=0;i<numstages;i++)
                {
                        pthread_mutex_lock(&(stagearr[i]->stagelock));//
lock the stage since there is possibilty that two musicians take up same
stage

                        if(stagearr[i]->type == 1)
                        {
                                if(stagearr[i]->status == 0) // Check
```

```
whether it is the required stage or not
                                {
                                        strcpy(stagearr[i]->personname
,per->name);// fill in attributes. TO let the stage know which performer is
performing

                                        stagearr[i]->personid = per-
>performerid;// fill in attributes. TO let the stage know which performer
is performing

                                        stagearr[i]->status = 1;// Set the
status to one since only one is performing for now

                                        //printf("%s got a stage of id
%d\n", per->name , i);
                                        if(per->instrument != 5)// Increase
the available slots for a singer to join . don't increment if singer gives
solo performance
                                        {
                                                sem_post(&duo);
                                        }

                                        per->stage = i;
                                        pthread_mutex_unlock(&(stagearr[i]-
>stagelock));
                                        break;
                                }
                        }
                        pthread_mutex_unlock(&(stagearr[i]->stagelock));
                }

                int time =  t1 + rand()%(t2-t1+1);// calcuate the time of
performance

                printf(COLOR_BLUE"%s is performing %c at the electic stage
(id %d) for %d seconds\n"COLOR_BLUE,per->name ,((char)(reassign(per-
>instrument))),per->stage,time);
                sleep(time);// To simulate the time of performance.
                for(int kk = 0;kk<1;kk++)
                {
                        //printf("ENTERED\n");

                        pthread_mutex_lock(&(stagearr[per->stage]-
>stagelock));
                        if((stagearr[per->stage])->status == 2)// check if
any singer has joined him or not
                        {       sleep(1);

                                yes = 1;

                                stagearr[per->stage]->status = 0;
                                pthread_mutex_unlock(&(stagearr[per-
>stage]->stagelock));
                                sleep(2);
```

```
                                break;
                        }
                        pthread_mutex_unlock(&(stagearr[per->stage]-
>stagelock));
                        //printf("EXITED once\n");
                        if(stagearr[per->stage]->status != 2)
                        {
                                int val = sem_trywait(&duo);
                                if(val==-1) // Check if any singer has
pastthrough the semaphore and waiting for the slot
                                {
                                        sleep(1);
                                        yes =  1;                       //
printf("enterd wrongfastly\n");
                                        sleep(2);// if yes decrement the
value;
                                        pthread_mutex_lock(&((stagearr[per-
>stage])->stagelock));
                                        stagearr[per->stage]->status = 0;
                                        pthread_mutex_unlock(&
((stagearr[per->stage])->stagelock));
                                }
                                else
                                {

                                        // reset the values and exit
                                        //("enterd corectly\n");
                                        pthread_mutex_lock(&((stagearr[per-
>stage])->stagelock));
                                        stagearr[per->stage]->status = 0;
                                        pthread_mutex_unlock(&
((stagearr[per->stage])->stagelock));
                                        //              printf("exitedd
corectly\n");

                                }
                        }
                }

        }
        printf(COLOR_GREEN"%s performance at electric stage
finished\n"COLOR_RESET,per->name);

        sem_post(&electric);
        if(yes)// if a singer has joined musician make a thread for the
singer to let him collect the t-shirt.
        {
                pthread_t singer;
                struct performer * ss = (struct performer *)
(malloc(sizeof(struct performer)));
                strcpy(ss->name , per->singername);
                pthread_create(&singer, NULL , coordinator, (void *)ss);
        }
        coordinator((void *)per);
```

```
        //printf("done\n");
        return NULL;
}
```

The performer searching for an acoustic stage is simulated by function "acoustic". The function is similar to function "electric".