Question-3 Report

Idea

I am taking every performer as a thread. when the performer is arrived.I divided threads as per their instruments. main classification is

Performer is a musician:

After dividing the performers by instruments we will talk only for musician here . If the musician has chance of taking two stages then i am creating two threads and keeping a race between theese two threads which ever takes the stage first the performer will perform on that stage which is decided by thread like if the acoustic thread aquires a stage then the performer will perform on acoustic stage. so I have two fn's here givingac() and givingec() the acoustic thread will call givingac() simlarly electric thread in that fn In that fns when the thread calls the fn stat varialble in perfm struct will assign the stage. so after going to these fn first i will check the stat is assigned or not if not assigned then goto this stage if assigned then leave this fn like that So the performer goes to the stage and In that fn's we have loop looping through respective stages i.e. acoustic stages in givingac(). as he is musician we will mark the stage as a singer can join and after if the singer joins we will wait for two seconds else normal and the performer has duration to perform after this duration the musician an singer will leave the stage if singer joined musician and if singer is joined then after leaving the stage I am calling two threads one is musician and singer and passes to coordinate fn to collect tshirts parallely after that the musician and joined singer will exit. And if the muscian can perform only in one stage then no racing and this thread will normally call and do the same.

performer is a singer:

As singer can join a musician , so I am creating three threads acoustic ,electric ,joiningmusician and keeping race between them if the electric wins that means singer performs on electric stage and nobody can join singer similarly acoustic wins, If joiningmusician wins that means singer has to join the stage with musician. so the acoustic and electric parts are same for singer and musician, as joiningmusician wins as the thread calls singerjoin() fn inthat same as givinga() to check whethere the stage is assigned or not and after that we will loop through all stages because he can join any stage, and checking the stage can hae joining option if there then joins.

Implementation functions

variables

In this code I have 4 semaphores name ac,ec,js,co. as ac is the semaphore which takes care of how many acoustics available similarly ec , js is the semaphore that which takes care of how many stages can the singer may join, co is the semaphore that takes care about how many coordinators for giving tshirt. and I have structs per which is performer things and st which is stage things and coo is the struct which is passing to coordinator fn to collect tshirts for performer: as comments and the names says the functionality of that variable

```
typedef struct per
{
    int id;
    char *name;
    char instrument;
    int singer;                         //if perfomer is a muscian then this
the joining singer id if he is a singer then it is same as id
    int performingstage;                 //stage id where he is performing
to be
    int stat;                   //0 if stage is not assigned 1 is stage
assigned 2 is he left the stage
    int arrivaltime;
    pthread_mutex_t lock;
}per;
```

for stage : as comments and the names says the functionality of that variable

```
typedef struct st
{
    int id;
    int perfm;                     //performer id who is performing on
stage
    int stat;                     //0 if stage is not filled 1 if stage is
filled with singer 2 if stage can not be filled with singer
    int type;                     //0 for acoustic 1 for electric
    pthread_mutex_t lock;
}st;
```

for passing to coordinator(): as comments and the names says the functionality of that variable

```
{
    int id;
    char *name;
}coo;
```

I have arrays perfm and stage to with datatypes as respective structs, ressembles as each instance of array is a performer and identified by id of the performer and similarly for stage.

usefull funtions: myrandom() gives the random number between the l and r;

```
int myrandom(int l, int r)
{
    return (rand() % (r - l + 1)) + l;
}
```

regular():

In regular fn i will create the threads for racing between the stages of the performer, and for singer i created three threads as it has to has three options so race between three options. This funtion takes care about creating threads according to number of stages and for singer one extra thread for joiningmusician. after all threads executed will wait the threads to join and exits the fn.

```
void *regular(void * args)
{
    int id=((per*)args)->id;
    sleep(perfm[id].arrivaltime);    //waiting for the student to arrive
   // printf("\033[0;33m");
    printf("\033[0;33m%s %c arrived\n\033[0m", perfm[id].name,
perfm[id].instrument);
   // printf("\033[0m");
    pthread_t actid, ectid,jstid;//according to his instrument i created
noof threads as no of stages he can participate and keeping a race between
then who wons they will take that stage (electric or acoustic) and proceed.
    if(perfm[id].instrument=='v')
    {
        pthread_create(&actid,NULL,givingac,&perfm[id]);
    }
    else if (perfm[id].instrument == 'b')
    {
        pthread_create(&ectid, NULL, givingec, &perfm[id]);
    }
    else if (perfm[id].instrument == 'p' || perfm[id].instrument=='g')
    {
        pthread_create(&actid, NULL, givingac, &perfm[id]);
        pthread_create(&ectid,NULL,givingec,&perfm[id]);
    }
    else if (perfm[id].instrument == 's')   //for singer extra is joining
so we have another thread for joining and the race is between he takes
acoustic or electric or he joins with other who wins the race they will
perform correspondingly
    {
        pthread_create(&actid, NULL, givingac, &perfm[id]);
        pthread_create(&ectid, NULL, givingec, &perfm[id]);
        pthread_create(&jstid, NULL, singerjoin, &perfm[id]);
    }



    if (perfm[id].instrument == 'v')
    {
        pthread_join(actid, NULL);
    }
    else if (perfm[id].instrument == 'b')
    {
        pthread_join(ectid, NULL);
    }
    else if (perfm[id].instrument == 'p' || perfm[id].instrument == 'g')
```

```
    {
        pthread_join(actid, NULL);
        pthread_join(ectid, NULL);
    }
    else if (perfm[id].instrument == 's')
    {
        pthread_join(actid, NULL);
        pthread_join(ectid, NULL);
        pthread_join(jstid, NULL);
    }
}
```

givingac():

As we are keeping the race between threads one of the threads call this fn or the musician with only one stage will create thread and call this fn As the thread called this fn first we do timed wait on semaphore ac (i.e ac is no acoustic stages available). that means the performer only wait t seconds for a to be assigned to a stage, if the stage is not avaible for more than t seconds the performer will exit due to impatience. i.e. when the timewait fails we have to check that the performer status as left or not and not assigned. if the status is left then the thread will just exit if he did not left and stage is assigned then also the thread just exits. if status is not left and not assigned then we have to assigned the status as left and exits the fn. and if it succeded the againg we have to check the status. if the status is assigned i.e the other thread came first and taken the stage in that case this thread no longer should exist and if the status is left i.e if the other thread came first and exited because of timed wait or whole performance is over so as the win of the race is decided by who came first . if the first one lefted then the other threads comperting this also should be exited and as this if condition is checked i.e it decremented the semaphore so while we leaving we have to increment the semaphore and exit.

```
int id = ((per *)args)->id;
    struct timespec ts;
    clock_gettime(CLOCK_REALTIME, &ts);
    ts.tv_sec+=t;
    int s = sem_timedwait(&ac, &ts); //timed wait for t sec
    pthread_mutex_lock(&perfm[id].lock);
    if(s==-1)
    {
        if (perfm[id].stat == 0 && perfm[id].stat != 2) //if failed then we
have to check that performer status is left or not and stage not assigned
        {
            perfm[id].stat = 2; //then we have to mark this performer as
left.
                                //  printf("\033[0;35m");
            printf("\033[0;35m%s %c left because of impatience\n\033[0m",
perfm[id].name, perfm[id].instrument);
          //  printf("\033[0m");
        }
        pthread_mutex_unlock(&perfm[id].lock); //unlocking before returning
        return NULL;
    }
```

```
    if (perfm[id].stat == 1 || perfm[id].stat == 2) //I was checking that
whether the performer is assigned to stage or left?
    {
        sem_post(&ac); //releasing semaphore.
        pthread_mutex_unlock(&perfm[id].lock); //unlocking before retrun.
        return NULL;
    }
```

if all the above condistions satisfiesd then we have to loop through the stages as this fn is acoustic stage so we have iterate through acoustic stage only we will now lock the stage to not acces the inner data by multiple performers and check the stage is assigned or not if not assigned then we wil update the performers performingstage and performer status as stage assigned and stage stores the performer and checks if the performer is singer or not if performer is singer then the stage status updates as nosinger can join, if the performer is musician then the stage staus is updated to singer can join and increment the semaphore js(i.e available stages to join).after that unlocking the stage and breaking out of the loop.

```
for (int i = 0; i < a; i++) //if all conditions are satisfied that means no
stage is assigned to it then loop through acoustic stages.
    {
        pthread_mutex_lock(&stage[i].lock);
        if (stage[i].stat == 0) //checking the stage is assigned to a
performer or not
        {

            perfm[id].performingstage = i; //if not assigned the setting
performers stage to this stage
            perfm[id].stat = 1;            //and updating status as as
stage assigned for performer.
            stage[i].perfm = id;           //setting the performerid of the
stage to performer id
            if (perfm[id].singer == id)
            {
                stage[i].stat = 2; //if he is singer then we have to update
the staus of stage as singer cannot join to this stage
            }
            else
            {
                stage[i].stat = 1; //if he is performer the singer can join
and updating the status to can join singer at this stage
                sem_post(&js);     //increasing the semaphore js that means
singer can join
            }
            pthread_mutex_unlock(&stage[i].lock);
            break;
        }
        pthread_mutex_unlock(&stage[i].lock);

    }
    pthread_mutex_unlock(&perfm[id].lock);
```

after taking the stage the performance of the performer will have a duration between t1 and t2 randomly so myrandom fn returns random no between t1 and t2, so sleeping that much time indicating the performance is going on for that time. after this we will check that if the singer is joined or not i.e performers singer variable is the id of the singer, if that id is same as this performer id then the performer is a singer if it is -1 that means no singer is joined other wise singer is joined, as singer is joined the performance has to be extended by 2 seconds. if not joined I am using sem_trywait() it returns -1 as the semaphore is 0 and returns 0 as the semaphore is >0 here the semantic is if the semtrywait() fails i.e the js is 0, that means as he is a musician he incremented the js that means js have to be atleast 1 but now if the js is 0 that means the singer takes the semaphore but he is not joined so we will wait for 2sec to be joined. As the performane is over the performers to be left as the perfomers are leaving we have to reset the stage variables and set the performers status as left, After this these performers have to be get tshirts so I created threads as many as no of performers and called the coordinator fn because the tshirts should be collected parallelly.

```
int rantime = myrandom(t1, t2); //setting the time of duration of
performance
                                    //   printf("\033[0;36m");
    printf("\033[0;36m%s performing %c at acoustic stage for %d
sec\n\033[0m", perfm[id].name, perfm[id].instrument, rantime);
  //  printf("\033[0m");
sleep(rantime); //sleeping means that this performance is going on
    if (perfm[id].singer != id && perfm[id].singer != -1) //checking singer
is joined performer
    {
        sleep(2); //if joined waiting two seconds
    }
    else if (perfm[id].singer!=id)
    {
        int stry = sem_trywait(&js); //checking trywait that if the js=0
i.e singer is about to joing and not came to stage
        if (stry == -1)
        {
            sleep(2); //sleeping two seconds is waiting to join the singer
to musician
        }
    }
    // if(perfm[id].singer!=-1 && perfm[id].singer!=id)
    // {
    //     sleep(2);
    // }

    pthread_mutex_lock(&stage[perfm[id].performingstage].lock);
    stage[perfm[id].performingstage].stat = 0; //reseting the stage here
    stage[perfm[id].performingstage].perfm=-1;
    pthread_mutex_unlock(&stage[perfm[id].performingstage].lock);
    pthread_mutex_lock(&perfm[id].lock);
    perfm[id].stat = 2; //updating status to that he has left the stage
                        // printf("\033[0;36m");
    printf("\033[0;36m%s performance at acoustic stage is ended\n\033[0m",
perfm[id].name);
```

```
   // printf("\033[0m");
    if (perfm[id].singer != -1 && perfm[id].singer != id)
    {
        pthread_mutex_lock(&perfm[perfm[id].singer].lock);
        perfm[perfm[id].singer].stat = 2; //if singer is there then
updating the status of the singer as left the stage.

        pthread_mutex_unlock(&perfm[perfm[id].singer].lock);
    }
    pthread_mutex_unlock(&perfm[id].lock);
    if (c != 0) //checking coordinators are zero or not
    {
    pthread_t music_thread,singer_thread;
    coo music, singer;
    music.id=perfm[id].id;
    music.name=malloc(sizeof(perfm[id].name)*2);
    strcpy(music.name,perfm[id].name);
    pthread_create(&music_thread, NULL, coordinator, &music); //creating
threads of performer to collect tshirt parallely
    if(perfm[id].singer!=-1 && perfm[id].singer!=id)
    {
        singer.id=perfm[id].singer;
        singer.name=malloc(sizeof(perfm[perfm[id].singer].name)*2);
        strcpy(singer.name, perfm[perfm[id].singer].name);
        pthread_create(&singer_thread, NULL, coordinator, &singer); //if
joined singer is there then also thread will be created to collect tshirt
parallely
    }
    pthread_join(music_thread,NULL);
    if (perfm[id].singer != -1 && perfm[id].singer != id)
    {
        pthread_join(singer_thread,NULL);
    }
    }
    sem_post(&ac);     //releasing semaphore
    return NULL;
```

after this these thread will go to coordinator fn and waits for the co semaphore(i.e noof coordinators available) prints he is collected tshirt and sem_post the co and returns from the fn.

```
void *coordinator(void *args)
{
    int id = ((coo *)args)->id;
    sem_wait(&co);                                  //holding the
coordinator to give t shirt
    //printf("\033[1;35m");
    printf("\033[1;35m%s collecting T-shirt\n\033[0m", ((coo *)args)-
>name);
  //  printf("\033[0m");
    sem_post(&co);                                   //after collecting
tshirt coordinater will be released
```

```
    return NULL;
}
```

givingec()

This function is same as givingac() fn just is the difference is using semaphore ec instead of ac and looping through electirc stage i.e i is looping through a to a+e. the threads have race so one of the thread goes to following fn and change their statuses before the other thread changes the change the status and only one thread continues to allocating and performing on the stage.

singerjoin()

This fn is the called by one of the threads created by the singer which are in race. the checking of timed wait and the stage is assigned or not conditions are same only difference is it loops through all the stages and check the status of the stage if the status of the stage is singer can join the the singer updates his status to stage assigned and the stage staus will be assigned to nosingerjoin and the musician's singer id is change to this id and who is performing on the stage. and breaking out of the loop and exits. Here the semaphore used is js.

```
void *singerjoin(void *args)                                    //this is the fn
that takes care of joining singer
{
    int id = ((per *)args)->id;
    struct timespec ts;
    clock_gettime(CLOCK_REALTIME, &ts);
    ts.tv_sec += t;
    int s = sem_timedwait(&js, &ts);                 //timed wait for given
tsec
    pthread_mutex_lock(&perfm[id].lock);
    if (s == -1)
    {
        if (perfm[id].stat == 0 && perfm[id].stat != 2)        //if failed
then we have to check that performer status is left or not and stage not
assigned
        {
            perfm[id].stat = 2;                                  //then we
have to mark this performer as left.
           // printf("\033[0;35m");
            printf("\033[0;35m%s %c left because of impatience\n\033[0m",
perfm[id].name, perfm[id].instrument);
           // printf("\033[0m");
        }
        pthread_mutex_unlock(&perfm[id].lock);      //unlocking before
returning
        return NULL;
    }
    if (perfm[id].stat == 1 || perfm[id].stat == 2)               //I was
checking that the performer left or stage assigned then
    {
        sem_post(&js);                                           //releasing
```

```
the semaphore before return.
        pthread_mutex_unlock(&perfm[id].lock);                  //unlocking
the lock
        return NULL;
    }
    for (int i = 0; i < a+e; i++)                               //if all the
conditions above are satisfied then it comes to for loop.
    {
        pthread_mutex_lock(&stage[i].lock);
        if (stage[i].stat == 1)                                  //checking
the stage can join the performer
        {

            perfm[id].performingstage = i;                          //these
are storing the singers stage and assigning staging for singer.
            perfm[id].stat = 1;
            stage[i].stat = 2;                                       //sets
the stage status to not joinging by singer
            perfm[stage[i].perfm].singer=perfm[id].id;
//setting the musican singer's id to singer's id
         //   printf("\033[1;32m");
            printf("\033[1;32m%s joined %s's performance ,performance
extended by 2 seconds\n\033[0m", perfm[id].name,
perfm[stage[i].perfm].name);
          //  printf("\033[0m");
            pthread_mutex_unlock(&stage[i].lock);
            break;
        }
        pthread_mutex_unlock(&stage[i].lock);
    }
    pthread_mutex_unlock(&perfm[id].lock);
    return NULL;
}
```

So Afher the singer has joined the stage .this will be resumed in the musicians stage fn i.e givingac() or givingec() in that they will check the muscian is joined or not. And they create thread of the singer to collect tshirt there. So singer also will collect the tshirt and the performers who left cannot get the tshirt.