Assignment-4

Question-3

Musical Mayhem

The college is opening for the new semester, and the campus is overrun with exciting first years and their parents. The Music Club has decided to host an event on the first day to entertain all the newcomers. You have been given the responsibility of managing the event.

Global Structs

```
typedef struct performer
{
    int id;            //
    int stage;         //
    int stageid;       //
    int arrivalTime;   //
    int extendTime;    //
    int performTime;   //
    char *name;        //
    char *instrument;  //
    bool singer;       //
    bool finished;     //
    pthread_t tid;     //
    pthread_mutex_t m; //
    pthread_cond_t cv; //
    sem_t tshirt;      //
    // 0 is no choice,
    // 1 is acoustic,
    // -1 is electric
} performer;
typedef struct stage
{
    int id;             //
    int type;           //
    int performer;      //
    int performer_type; //
    bool noExtension;   //
    pthread_t tid;      //
    sem_t sem;          //
    sem_t extension;    //
} stage;
typedef struct coordinator
{
    int id;        //
```

```
    pthread_t tid; //
} coordinator;
```

Explanation

There are three global structs -- because threads share global address space, this is the easiest way to handle them.

- Struct *performer* coordinates between stage and the performer (musician and singer). Mutex *m* ensures that a performer must perform on one stage. Condition variable *cv* allows time waiting. Semaphore *tshirt* acts as a binary rendezvous between the performer and the coordinator.
- Struct *stage* has a boolean variable *noExtension* and semaphore *extension*, which provides coordination between musician and singer if they share the same stage. Semaphore *sem* coordinates between stage and performer, which will be signaled by the performer when his/her performance overs.

Performers (musicians and singers)

Performer function

```
static void *Performer(void *p1)
{
    int rc = 0;
    struct timespec ts;
    performer *p = (performer *)p1;
    sleep(p->arrivalTime);
    printf("\033[0;32m%s %s arrived\n", p->name, p->instrument);
    fflush(stdout);
    pthread_mutex_lock(&p->m);
    clock_gettime(CLOCK_REALTIME, &ts);
    ts.tv_sec += MaxWaitingTime;
    p->stageid = -1;
    while (p->stageid == -1 && rc == 0)
    {
        rc = pthread_cond_timedwait(&p->cv, &p->m, &ts);
    }
    if (rc != 0)
    {
        p->stageid = 0;
    }
    pthread_mutex_unlock(&p->m);
    if (rc == 0)
    {
        givePerformance(p);
        sem_post(&stages[p->stageid].sem);
        gettshirt(p);
    }
    else
    {
```

```
        printf("\033[0;36m%s %s left because of impatience\n", p->name, p->instrument);
        fflush(stdout);
    }
    pthread_mutex_lock(&lock);
    PerformedPerformers++;
    pthread_mutex_unlock(&lock);
    return NULL;
}
```

Explanation

This function handles all the details of a performer. It uses a mutex to access the shared global variable *PerformedPerformers* to be no race condition and deadlock. The performer is waiting for a stage to be available for the performance. He/She waits for maximum *MaxWaitingTime* for the stage to be available for performance.

givePerformance function

```
static void givePerformance(performer *p)
{
    // int performTime = getBrowseTime();
    stages[p->stageid].type == 1 ? printf("\033[01;33m%s performing %s at acoustic stage #%d for %d seconds\n", p->name, p->instrument, p->stageid + 1, p->performTime)
                                 : printf("\033[0;33m%s performing %s at electric stage #%d for %d seconds\n", p->name, p->instrument, p->stageid + 1, p->performTime);
    fflush(stdout);
    sleep(p->performTime);
    sem_wait(&stages[p->stageid].extension);
    sleep(p->extendTime);
    stages[p->stageid].noExtension = true;
    sem_post(&stages[p->stageid].extension);
    stages[p->stageid].type == 1 ? printf("\033[01;33m%s performance at acoustic stage #%d ended\n", p->name, p->stageid + 1)
                                 : printf("\033[0;33m%s performance at electric stage #%d ended\n", p->name, p->stageid + 1);
    fflush(stdout);
}
```

Explanation

In this function, a performer gives his/her performance for a random amount of time *performTime*. After completion, it checks whether any extension due to joining of a singer, if not, signals the respective stage.

gettshirt function

```
static void gettshirt(performer *p)
{
    // singers will also get t-shirt
    p->finished = true;
    sem_wait(&p->tshirt);
}
```

Explanation

- This is a straightforward function. The performer is waiting until he/she get the t-shirt.
- Both musicians and singers will get a t-shirt. How? Because I am considering both of them as a performer performing on the same stage (stage number) and will complete their performance simultaneously.

Stage

Stage function

```
static void *Stage(void *s1)
{
    stage *s = (stage *)s1;
    while (PerformedPerformers != NumPerformers)
    {
        time_t start_time, perform_time;
        bool success = false;
        while (!success && PerformedPerformers != NumPerformers)
        {
            for (int i = 0; i < NumPerformers && PerformedPerformers !=
NumPerformers; i++)
            {
                if (performers[i].stageid != -1)
                {
                    continue;
                }
                pthread_mutex_lock(&performers[i].m);
                if (performers[i].stageid == -1 &&
                    (performers[i].stage == 0 ||
                     performers[i].stage == s->type))
                {
                    s->noExtension = false;
                    s->performer = performers[i].id;
                    s->performer_type = performers[i].singer;
                    start_time = time(NULL);
                    performers[i].stageid = s->id;
                    performers[i].performTime = getBrowseTime();
                    perform_time = performers[i].performTime;
                    success = true;
                }
```

```
                pthread_mutex_unlock(&performers[i].m);
                if (success)
                {
                    pthread_cond_signal(&performers[i].cv);
                    break;
                }
            }
        }
        if (!success)
        {
            break;
        }
        // solo singer
        if (s->performer_type > 0)
        {
            sem_wait(&s->sem);
            continue;
        }
        // performance joining by singer
        int secondPerformer = -1;
        while (secondPerformer == -1 && PerformedPerformers !=
NumPerformers && !s->noExtension)
        {
            for (int i = 0; i < NumPerformers && PerformedPerformers !=
NumPerformers && !s->noExtension; i++)
            {
                if (performers[i].stageid != -1)
                {
                    continue;
                }
                pthread_mutex_lock(&performers[i].m);
                if (performers[i].stageid == -1 &&
                    performers[i].singer)
                {
                    sem_wait(&s->extension);
                    if (!s->noExtension)
                    {
                        printf("\033[1;32m%s joined %s's performance,
performance extended by %d seconds\n", performers[i].name, performers[s-
>performer].name, ExtensionTime);
                        fflush(stdout);
                        performers[i].stageid = s->id;
                        performers[i].performTime = ExtensionTime +
perform_time - (time(NULL) - start_time);
                        performers[s->performer].extendTime =
ExtensionTime;
                        secondPerformer++;
                    }
                    secondPerformer++;
                    sem_post(&s->extension);
                }
                pthread_mutex_unlock(&performers[i].m);
                if (secondPerformer != -1)
                {
```

```
                    if (secondPerformer > 0)
                    {
                        pthread_cond_signal(&performers[i].cv);
                    }
                    break;
                }
            }
        }
        if (secondPerformer > 0)
        {
            sem_wait(&s->sem);
        }
        sem_wait(&s->sem);
    }
    return NULL;
}
```

Explanation

- This function handles all the details of a stage, which is the longest function in the whole code.
- This function checks if any performer is waiting for a stage whose instrument matches this stage, then signal that performer and assign it to him/her.
- As a singer can join a musician, the stage will increase the performance time for the musician performing already and signal the musician.
- The *performTime* for the signer will be calculated based on the time for which the musician has already performed so that they will complete their performance simultaneously.
- If all the performers are done with their performance or exited due to impatience, it will return.

Coordinator

Coordinator function

```
static void *Coordinator(void *c1)
{
    coordinator *c = (coordinator *)c1;
    while (PerformedPerformers != NumPerformers)
    {
        for (int i = 0; i < NumPerformers && PerformedPerformers !=
NumPerformers; i++)
        {
            if (!performers[i].finished)
            {
                continue;
            }
            pthread_mutex_lock(&performers[i].m);
            if (performers[i].finished)
            {
                printf("\033[0;35m%s collecting t-shirt\n",
performers[i].name);
```

```
                fflush(stdout);
                performers[i].finished = false;
                sleep(tshirtTime);
                sem_post(&performers[i].tshirt);
            }
            pthread_mutex_unlock(&performers[i].m);
        }
    }
    return NULL;
}
```

Explanation

It is a straightforward function that checks if a performer has finished his/her performance and not yet given a t-shirt, then gives him/her a t-shirt taking the time required for searching into account for a particular size. Mutex lock is used so that no two coordinators will be giving a t-shirt to the same performer.

Analysis

I analyzed the code for the test cases provided. Input and output are given below.

Test case-1

Input

```
5 1 1 4 2 10 5
Tanvi p 0
Sudh b 1
Manas g 0
Sriya v 1
Pahun s 1
```

Output

```
Tanvi piano arrived
Manas guitar arrived
Tanvi performing piano at acoustic stage #1 for 2 seconds
Manas performing guitar at electric stage #2 for 9 seconds
Sudh bass arrived
Sriya violin arrived
Pahun singer arrived
Pahun joined Manas's performance, performance extended by 2 seconds
Pahun performing singer at electric stage #2 for 10 seconds
Tanvi performance at acoustic stage #1 ended
Tanvi collecting t-shirt
Sriya performing violin at acoustic stage #1 for 4 seconds
Sriya performance at acoustic stage #1 ended
```

```
Sudh bass left because of impatience
Sriya collecting t-shirt
Manas performance at electric stage #2 ended
Manas collecting t-shirt
Pahun performance at electric stage #2 ended
Pahun collecting t-shirt
Finished
```

Test case-2

Input

```
1 1 1 4 2 7 1
Abhinav b 3
```

Output

```
Abhinav bass arrived
Abhinav performing bass at electric stage #2 for 7 seconds
Abhinav performance at electric stage #2 ended
Abhinav collecting t-shirt
Finished
```

Bonuses

- Singers will also get t-shirt after performance.
- Stage number is also printed in the output.

Thank you!