Musical Mayhem

Breakdown of the implemented Code

(Most of the code is well commented)

Main highlights of Code

Struct Used

```c
sem_t semph[5];
// semph[1]->acouctic
// semph[2]->electric
// semph[3]->coordinator
int array_of_stage[SIZE], array_singer[SIZE];
int stages[6][SIZE];
int student_stage[SIZE];
//for array_of_stage[SIZE] 0 -> no stage , 1-> got stage
//for array_singer[SIZE] initialised  = -1 , person_onstage_without_singer = -2
// with_singer = singer_id >=0  (musician P+G+V+B)
//variables as per question
int k;
int a, e, c, t1, t2, t;

//structure
typedef struct Musician
{
    int id, arrival_time; //for input
    //piano = 1, guitar = 2, violin = 3, bass = 4, singer = 5
    char name[75], instrument;
    pthread_mutex_t music_mutex[5];
    //  music_mutex[1] -> acoustics
    //  music_mutex[2]; -> electrical
    //  music_mutex[3]; -> needed for singer
    pthread_t music_pthread;
} Musician;

Musician musicians[SIZE];
```

Main() Function

```
    //initialization of semaphores used for stage and coordinator
    sem_init(&semph[1], 0, a);
    sem_init(&semph[2], 0, e);
    sem_init(&semph[3], 0, c);
    for (int i = 0; i <= (k - 1); ++i)
    {
        .
        .
        .
        pthread_mutex_init(&(musicians[i].music_mutex[1]), NULL);
        pthread_mutex_init(&(musicians[i].music_mutex[3]), NULL);
    }
    // creating , joining and destroying threads and calling musicianiniti function
    for (int i = 0; i <= (k - 1); ++i)
        pthread_create(&(musicians[i].music_pthread), NULL, musiciansinit,
&musicians[i]);
    for (int i = 0; i <= (k - 1); ++i)
        pthread_join(musicians[i].music_pthread, NULL);
    for (int i = 0; i < (k - 1); ++i)
    {
        pthread_mutex_destroy(&musicians[i].music_mutex[1]);
        pthread_mutex_destroy(&musicians[i].music_mutex[3]);
    }
```

void *musiciansinit(void *m)

It is just the caller function for the various cases like the acoustic and electic stages and also the singer function and the join function where the singer joins stage.

```
void *musiciansinit(void *m)
{
    Musician *temp = (Musician *)m;
    sleep(temp->arrival_time); //do later
    green();
    printf("%s %c arrived \n", temp->name, temp->instrument);
    reset();
    if (temp->instrument == 'p')
        case_ae(temp->id);
    else if (temp->instrument == 'g')
        case_ae(temp->id);
    else if (temp->instrument == 'v')
        case_unity(temp->id, 1);
    else if (temp->instrument == 'b')
        case_unity(temp->id, 2);
    else if (temp->instrument == 's')
        case_5(temp->id);
    else
    {
        printf("\x1B[31mError :\x1B[0m No instrument matched\n");
    }
}
```

void case_ae(int id)

It is the common function for acoustic and electric and it creates thread and calls thread_function for acoustic and electric.

```
void case_ae(int id)
{
    pthread_t thread_p1, thread_p2;
    pthread_create(&thread_p1, NULL, thread_func_a, (void *)&musicians[id]);
    pthread_t thread_p2;
    pthread_create(&thread_p2, NULL, thread_func_e, (void *)&musicians[id]);
    pthread_join(thread_p1, NULL), pthread_join(thread_p2, NULL);
}
```

void *thread_func_a(void *m) and void *thread_func_e (void *m)

Basically both the functions are similar , they use timedwait and return the threads which become impatient after the given time else gives the stage for performance.

```
void *thread_func_a(void *m)
{
    Musician *temp = (Musician *)m;
    struct timespec tim;
    .
    .
    tim.tv_sec = tim.tv_sec + t;
    flag = sem_timedwait(&semph[1], &tim);
    if (flag != -1) //getting stage
    {
        pthread_mutex_lock(&temp->music_mutex[1]);
        if (array_of_stage[temp->id] != 0)
            pthread_mutex_unlock(&temp->music_mutex[1]);
        else
        {
            array_of_stage[temp->id] = 1;
            pthread_mutex_unlock(&temp->music_mutex[1]);
            stage(temp->id, 1);
        }
        sem_post(&semph[1]);
    }
    else
    {
        pthread_mutex_lock(&temp->music_mutex[1]);
        if (!array_of_stage[temp->id])
        {
            printf("\033[0;36m%s %c left because of impatience\033[0m\n", temp->name,
temp->instrument);
            array_of_stage[temp->id] = 1;
        }
        pthread_mutex_unlock(&temp->music_mutex[1]);
    }
}

void *thread_func_e(void *m)
{
    Musician *temp = (Musician *)m;
    .
    .
    flag = sem_timedwait(&semph[2], &tim);

    if (flag != -1) //getting stage
    {
        pthread_mutex_lock(&temp->music_mutex[1]);
        if (array_of_stage[temp->id] != 0)
            pthread_mutex_unlock(&temp->music_mutex[1]);
        else
        {
            array_of_stage[temp->id] = 1;
            pthread_mutex_unlock(&temp->music_mutex[1]);
            stage(temp->id, 2);
        }
        sem_post(&semph[2]);
    }
    else
    {
        pthread_mutex_lock(&temp->music_mutex[1]);
        if (!array_of_stage[temp->id])
        {
            printf("\033[0;36m%s %c left because of impatience\033[0m\n", temp->name,
temp->instrument);
            array_of_stage[temp->id] = 1;
        }
        pthread_mutex_unlock(&temp->music_mutex[1]);
    }
}
```

void stage(int id, int type)

It allocates the stage on the basis of type (acoustic or electric) and monitors the performance of musician and singers.

```
void stage(int id, int type)
{   pthread_mutex_lock(&(musicians[id].music_mutex[3]));
    int sleep_time, z = 0;
    array_singer[id] = -2;
    pthread_mutex_unlock(&(musicians[id].music_mutex[3])),
        pthread_mutex_lock(&(musicians[id].music_mutex[3]));
    int till;
    if (type == 1)
    {
        till = a;
    }
```

```c
    else
    {
        till = e;
    }
    for (z = 0; z < till; ++z)
    {
        if (stages[type][z] == -1)
        {
            stages[type][z] = 1;
            student_stage[id] = z + 1;
            break;
        }
    }
    pthread_mutex_unlock(&(musicians[id].music_mutex[3]));
    sleep_time = rand() % (1 + (t2 - t1)) + t1;
    char strr[20];
    if (type == 1)
    {
        strcpy(strr, " Acoustic ");
    }
    if (type == 2)
    {
        strcpy(strr, " Electric ");
    }
    printf("\033[0;33m%s performing %c at %s stage %d for %d sec\033[0m\n",
musicians[id].name, musicians[id].instrument, strr, student_stage[id], sleep_time);
    sleep(sleep_time); // performing
    pthread_mutex_lock(&(musicians[id].music_mutex[3]));
    if (array_singer[id] > -1)
    {
        sleep(2);
        pthread_mutex_unlock(&(musicians[id].music_mutex[3]));
    }
    else
    {
        pthread_mutex_unlock(&(musicians[id].music_mutex[3]));
    }
    printf("\033[1;33m%s performance %c at %s stage %d  ended. \033[0m\n",
musicians[id].name, musicians[id].instrument, strr, student_stage[id]);
    sem_wait(&semph[3]); //coordinator available
    printf("\033[0;34m%s collecting T-shirt\033[0m\n", musicians[id].name);
    if (array_singer[id] > -1)
      printf("\033[0;34m%s collecting T-shirt\033[0m\n",
musicians[array_singer[id]].name);
    pthread_mutex_lock(&(musicians[id].music_mutex[3]));
    stages[type][z] = -1;
    pthread_mutex_unlock(&(musicians[id].music_mutex[3]));
    sem_post(&semph[3]);
    return;
}
```

void case_5(int id)

This implements threads for the singers and calls the respective function.

```
void case_5(int id)
{
    pthread_t thread_s1;
    pthread_create(&thread_s1, NULL, thread_func_a, (void *)&musicians[id]);
    pthread_t thread_s2, thread_s3;
    pthread_create(&thread_s2, NULL, thread_func_e, (void *)&musicians[id]),
        pthread_create(&thread_s3, NULL, join, (void *)&musicians[id]);
    pthread_join(thread_s1, NULL),
        pthread_join(thread_s2, NULL),
        pthread_join(thread_s3, NULL);
}
```

void *join(void *m)

It is used to implement if singer joins anyone already performing on stage.

```
void *join(void *m)
{
    Musician *temp = (Musician *)m;
    int typ = 1;
    while (!(array_of_stage[temp->id]))
    {
        for (int i = 0; i <= (k - 1); ++i)
        {
            pthread_mutex_lock(&(temp->music_mutex[3]));
            if (!(array_singer[i] + 2) && !(array_of_stage[temp->id]))
            {
                array_singer[i] = temp->id;
                array_of_stage[temp->id] = typ;
                printf("\033[1;35m%s joined %s's performance, extending performance by
2 secs .\033[0m\n", temp->name, musicians[i].name);
                break;
            }
            pthread_mutex_unlock(&(temp->music_mutex[3]));
        }
    }
}
```

First Bonus

Shirts for singer is also distributed. array_singer. It is implemented by array_singer[id] ie if it is greater than or equal to 0 then its value becomes equal to id of singer and while distributing the tshirt to musician we check if there were any musicians performing with them.

Second Bonus

All stages[] are given id and checked if stages are empty if it is -1 then no-one is preforming else stage is occupied. If we get the first stage where someone is not performing then we print that stage.

Assumption

Singer join if another singer performing on stage since nothing was mentioned about this in the question.