Programming Assignment 3 – Process Synchronize

薛劭杰 1930026143

Requirement:

We should design a program that students can ask for TA questions without conflict with each other. There are three chairs which is used for students to wait the TA if there is a student asking the TA. In order to guarantee the uncertainty of events, the help time and the programming of the students is random, here we set the range of the random number be from 0~9. Each student cannot access TA simultaneously and there are only three chairs which means that up to 3 students are allowed to be on hold.

How to solve question:

Step1: Define some semaphores, mutex and the threads as the global variables and we can initialize them in the main function. Semaphore is including TA, chairs and student. Mutex is used to prevent chair resource conflicts. And we have a TA thread, five students' threads.

```
pthread_t *students;
pthread_t TA;
sem_t ta_sem;
sem_t stu_sem;
sem_t chair_sem[chair_number];
pthread_mutex_t chair_available;
```

Step2: Initialize the semaphore, mutex and create the thread in the main function. For each thread, we have a special function to implement some operations.

```
sem_init(&ta_sem, 0, 0);
sem_init(&stu_sem, 0, 0);
for(stu_id = 0; stu_id < chair_number; ++stu_id)
    sem_init(&chair_sem[stu_id], 0, 0);
pthread_mutex_init(&chair_available, NULL);
```

Step3: Define the student thread function. In the function, we have a loop which used to make the program go on and on. First, the initial state of the student is programming. The time of the programming is random. And then, we should determine the number of the available chairs. If there three available chairs which means that there is no student in asking state, the student can sit on first empty chair, and wake up the TA.

```
if(count < chair_number) {
    if(count == 0){
        sem_post(&ta_sem);
    }
```

If the number of the available chairs is from 1 to 3, which means that there is a student is asking the ta and he must wait in the chair. Here we must use to a mutex to

lock the number of the chairs in order to prevent others from changing at the same time. And then, above the two cases should make the semaphore of chair and the student wait because student should leave his chair to receive help and wait to go next.

```
sem_wait(&chair_sem[index]);
printf("Student %ld receiving help.\n", (long)threadID);
sem_wait(&stu_sem);
```

If the number of the available chairs is 0, which means the student didn't find any chair to sit on, he should try asking later.

Step4: Define the ta thread function. In the student method, if the student can ask ta question, it will post the ta semaphore which means that ta should wake up and help students. Before them, ta is sleeping state which means that we should use a wait to initialize it.

Then, we should deal with the chair problem. When a student go to ask the ta, the chair number will increase 1, the current chair will post and the index of the current chair should change to the next one.

```
sem_post(&chair_sem[current_id]);
printf("TA is serving student");
chair_count--;

int help_time = rand() % 10 ;
printf("Helping a student for %d seconds waiting students = %d\n", help_time, chair_count);
```

Run the code by this command:

Remember to add the parameter $-lpthread$

```
root@P930026143:~/Desktop/os/homework/p3# gcc -o ta ta.c -lpthread
root@P930026143:~/Desktop/os/homework/p3# ./ta
```

And here is the running result:

```
        Student 0 hanging out for programming for 4 seconds
        Student 1 hanging out for programming for 3 seconds
        Student 3 hanging out for programming for 7 seconds
        Student 2 hanging out for programming for 9 seconds
        Student 4 hanging out for programming for 6 seconds
TA is serving studentHelping a student for 5 seconds waiting students = 0
Student 1 receiving help.
                Student 4 takes a seat  waiting students = 1
                Student 3 takes a seat  waiting students = 2
        Student 1 hanging out for programming for 9 seconds
TA is serving studentHelping a student for 8 seconds waiting students = 2
Student 0 receiving help.
                Student 2 takes a seat  waiting students = 2
        Student 0 hanging out for programming for 3 seconds
TA is serving studentHelping a student for 7 seconds waiting students = 2
Student 4 receiving help.
                Student 1 takes a seat  waiting students = 2
                Student 0 will try later.
        Student 0 hanging out for programming for 3 seconds
                Student 0 will try later.
        Student 0 hanging out for programming for 2 seconds
        Student 4 hanging out for programming for 1 seconds
TA is serving studentHelping a student for 0 seconds waiting students = 2
Student 3 receiving help.
        Student 3 hanging out for programming for 8 seconds
TA is serving studentHelping a student for 8 seconds waiting students = 1
Student 2 receiving help.
                Student 0 takes a seat  waiting students = 1
                Student 4 takes a seat  waiting students = 2
                Student 3 will try later.
        Student 3 hanging out for programming for 2 seconds
        Student 2 hanging out for programming for 3 seconds
TA is serving studentHelping a student for 2 seconds waiting students = 2
Student 1 receiving help.
                Student 3 takes a seat  waiting students = 2
        Student 1 hanging out for programming for 4 seconds
TA is serving studentHelping a student for 1 seconds waiting students = 2
Student 0 receiving help.
                Student 2 takes a seat  waiting students = 2
        Student 0 hanging out for programming for 5 seconds
TA is serving studentHelping a student for 8 seconds waiting students = 2
Student 4 receiving help.
```

```
                Student 4 takes a seat  waiting students = 1
                Student 3 takes a seat  waiting students = 2
                Student 2 will try later.
        Student 2 hanging out for programming for 8 seconds
        Student 0 hanging out for programming for 7 seconds
TA is serving studentHelping a student for 3 seconds waiting students = 2
Student 1 receiving help.
        Student 1 hanging out for programming for 6 seconds
TA is serving studentHelping a student for 3 seconds waiting students = 1
Student 4 receiving help.
                Student 2 takes a seat  waiting students = 1
        Student 4 hanging out for programming for 7 seconds
TA is serving studentHelping a student for 5 seconds waiting students = 1
Student 3 receiving help.
                Student 0 takes a seat  waiting students = 1
                Student 1 takes a seat  waiting students = 2
        Student 3 hanging out for programming for 0 seconds
                Student 3 will try later.
        Student 3 hanging out for programming for 6 seconds
TA is serving studentHelping a student for 5 seconds waiting students = 2
Student 2 receiving help.
                Student 4 takes a seat  waiting students = 2
        Student 2 hanging out for programming for 3 seconds
TA is serving studentHelping a student for 6 seconds waiting students = 2
Student 0 receiving help.
                Student 3 takes a seat  waiting students = 2
                Student 2 will try later.
        Student 2 hanging out for programming for 9 seconds
        Student 0 hanging out for programming for 1 seconds
TA is serving studentHelping a student for 3 seconds waiting students = 2
Student 1 receiving help.
                Student 0 takes a seat  waiting students = 2
        Student 1 hanging out for programming for 4 seconds
TA is serving studentHelping a student for 5 seconds waiting students = 2
Student 4 receiving help.
```

Encountered Problems:

1. How ta can know the index of the student? Or where the ta begin to teach the student and print the information “ta is Serving student”,

We can pass the parameter stu_id to the ta function or find the position where post the semaphores. But the process is complex and I think if cannot infer it, we can try the position by traversing.