計算機程式設計 HW8report

103061223 李俊穎

1. Introduction of the problem and your solution

The homework 8 is a “Task Arrangement” that we would like to calculate how much resources we need to complete a set of tasks and assign all the tasks with the correct resource number. The first we need to complete is to create at least one hundred tasks randomly. The problem of this is that we can’t use the srand() in the **Class** because it will produce the same number in the one hundred tasks and then violates the requirement. To solve this problem, we need to use pusk.back to store the one hundred tasks in the vector. The second step we do is sorting the data tasks from lowest to highest according to the start time, and then store them in the resources without overlapping. It means that one resource can’t deal with the work at the same time. After discussing with classmates, I use two for loop to judge the condition and store them into resources. The following is my solution.

2. Implementation details ,Additional features of your program (data structure, flows and algorithms)

In the first step I use the selection sort to sort the data in sequence, what I need to be careful is the usage of the class function:

```
void sequence(vector<Task>& task,int& num)
{
    int earliest_task,minindex,mintime;
    for(earliest_task=0;earliest_task<num;earliest_task++)
    {
        minindex=earliest_task;
        mintime=task[earliest_task].GetCoordinate().gettime();
        for(int index=earliest_task+1;index<num;index++)
        {
            if(task[index].GetCoordinate().gettime()<mintime)
            {
                mintime=task[index].GetCoordinate().gettime();
                minindex=index;
            }
        }
        task[minindex].SetCoordinate1(task[earliest_task].GetCoordinate().gettime());
        task[earliest_task].SetCoordinate1(mintime);
    }
}
```

This part I just reference at the textbook.

The second part of code:

```
for(int i=1;i<task.size();i++)
{
    int buffer=0,resource=0;
    bool check=true;
    while(check){
        for(int j=0;j<i;j++)
        {
            if(task[j].GetResource()==resource&&task[j].GetTime()+task[j].GetDuration()>task[i].GetTime())
            {
                buffer++;
            }
        }
        if(buffer!=0)
        {
            resource++;
            buffer=0;
        }
        else if(buffer==0)
        {
            task[i].SetCoordinate(resource);
            check=false;
        }
    }
}
```

What we need to check is if any task is overlapped with the others tasks, so we select two tasks compare with each other. At the first we set all the tasks resource in resource 0, and we check the time and duration. We choose the task 1 compared with task 0. The initial resource for task 0 is resource 0, so the **task[j].GetResource() ==resource** is true, and if the task 1's start time is less than the sum of task 0's start time and duration, I let a variable buffer +1. It means that the task 1 is overlapped with task 0 ,so I need to store it to resource 1. Buffer +1 makes me enter the **if** and then the resource will add one, because as long as **buffer>0**， it means that the task[i] must at least overlapped with the other task, then previous resource can't be stored. After the next time **while loop** running, it will have the initial value of r=1, and if this time the task 2 isn't overlapped with task 0 or task 1, then it will have buffer =0 at that time, and then the resource value at that time will store at task 2. Check = false is to break the while loop and we can check the task 3. After these loop, the tasks will stored in the correct resources.

The next procedure is store the cards value to the players at array. This isn't the big problem because it just need to set three for loops, and then this problem is solved. The previous Deal function is written, so what we need to do is pass the value to the function.

Finally, we have to input a time and check how much resource we need at the time. The following is my solution:

```
cout<<"Enter the time you want to see how much resources are working"<<endl;
cin>>time1;
for(int i=0;i<100;i++)
{
    if(time1>=task[i].GetTime()&&time1<=task[i].GetTime()+task[i].GetDuration())
    {
        work++;
    }
}
cout<<"We need the working resources are "<<work<<endl;
```

I check if the input time is in a time period because we have already stored the correct sequence in any resources, the time in any resource must only have one task or even have no task. That's why the calculation of work value is equal to resources value. That's all.