Experiment 1. Process scheduling experiment

[purpose requirements]

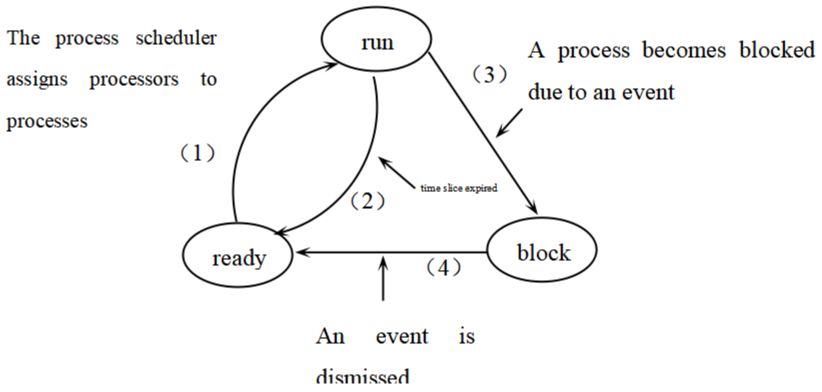
Write and debug a process scheduling program in high-level language to deepen the understanding of the concept of process and process scheduling algorithm.

[preparation knowledge]

1. Basic concepts
    1. The concept of process;
    2. The state of the process and the process control block;
    3. Process scheduling algorithm;

Two, process scheduling

1. The status of the process


The process scheduler assigns processors to processes
run
A process becomes blocked due to an event
(3)
(1)
time slice expired
(2)
ready
block
(4)
An event is dismissed

2. The structure of the process - PCB

A process is composed of a series of operations (actions) through which its tasks are accomplished. Therefore, different processes have different internal operations. In the operating system, besides the program and private data, the most important thing to describe a process is a data structure associated with the dynamic process, which is used to describe the external characteristics of the process (name, status, etc.) The data structure is called Process Control Block (PCB, Process Control Block) for information such as contact (communication relationship) of other processes .

The process control block PCB corresponds to the process one by one, and all the information required by the system, all the information required to describe the process situation, and all the information required to control the operation of the process are recorded in the PCB. Therefore, the system can manage the process through the PCB of the process.

[Test content]

Design a process scheduler with N processes running in parallel.

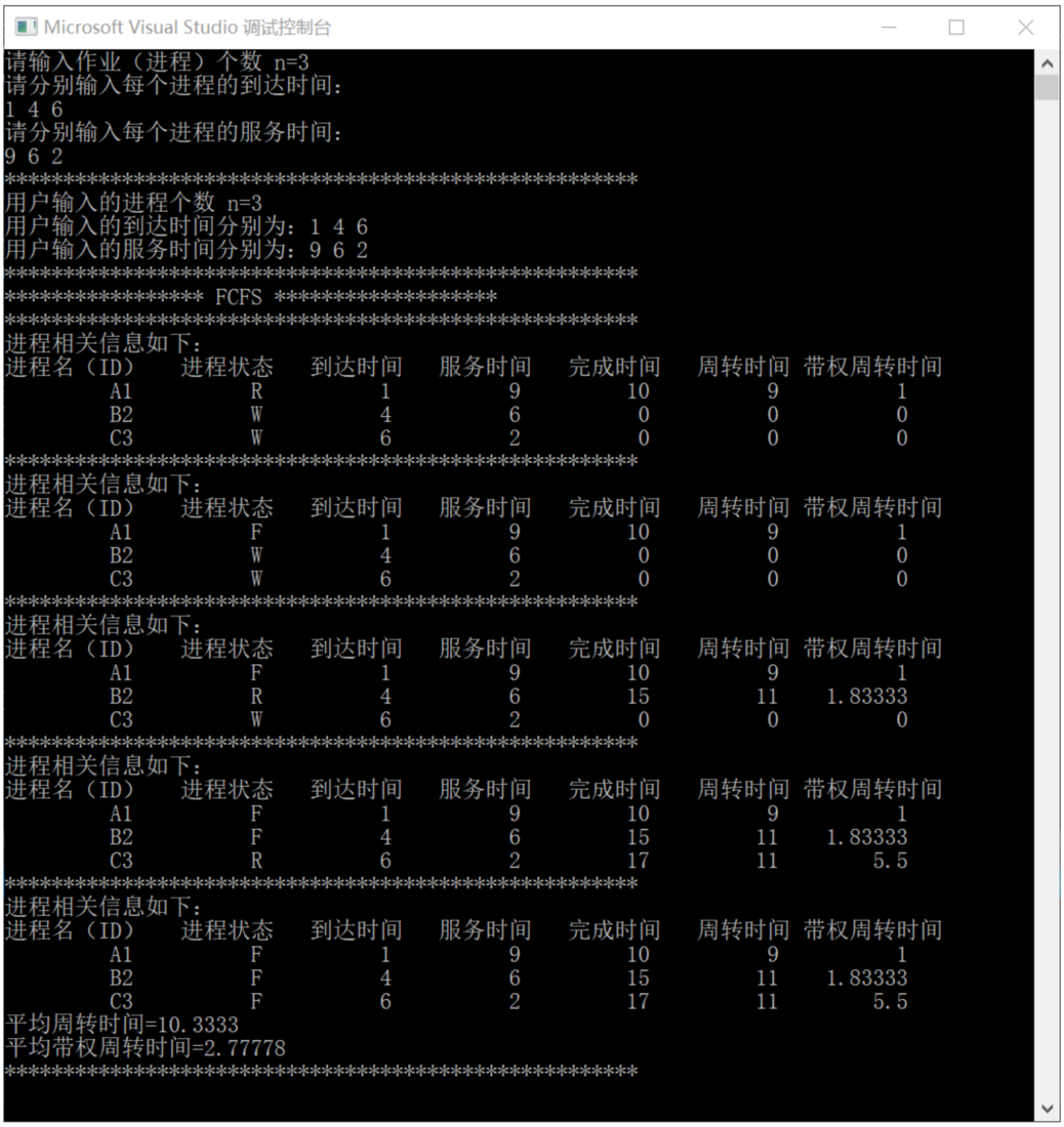
Process scheduling algorithm: use the scheduling algorithm with the highest priority number first (that is, assign the processor to the process with the highest priority number). Each process is represented by a Process Control Block (PCB). The process control block can contain the following information: process name, priority number, arrival time, required running time, used CPU time, process status and so on. The priority number of the process and the required

running time can be artificially specified in advance (it can also be generated by random numbers). The arrival time of the process is the time entered by the process. The running time of a process is calculated in units of time slices. The state of each process can be one of three states: ready W (Wait), running R (Run), or complete F (Finish). After the ready process obtains the CPU, it can only run for one time slice. It is expressed by adding 1 to the occupied CPU time. If after running a time slice, the occupied CPU time of the process has reached the required running time, the process will be canceled. If the occupied CPU time of the process has not reached the required running time after running a time slice, that is, the process It still needs to continue running. At this time, the priority number of the process should be reduced by 1 (that is, lowered by one level), and then inserted into the ready queue to wait for the CPU.
Every time the scheduler is executed, the running process, the ready queue, and the PCB of each process are printed for inspection.
Repeat the above process until all the required processes are completed.

Lab content: FCFS algorithm

```
#include <iostream>
#include <iomanip>
using namespace std;
#define MaxNum100 _

int ArrivalTime[ MaxNum ]; //arrival time
int ServiceTime[ MaxNum ]; //service time
int FinishTime[ MaxNum ]; //Completion time
int WholeTime[ MaxNum ]; //Turnaround time

int Priority[ MaxNum ]; //priority
char State[ MaxNum ]; //state
char Name[ MaxNum ]; //Name

double WeightWholeTime[ MaxNum ]; // weighted turnaround time
double AverageWT_FCFS, AverageWT_SJF; //The average turnaround time of the FCFS algorithm,
the average turnaround time of the SJF algorithm
double AverageWWT_FCFS, AverageWWT_SJF; //The average weighted turnaround time of the FCFS
algorithm, the average weighted turnaround time of the SJF algorithm

bool isFinished_FCFS[ MaxNum ];
bool isFinished_SJF[ MaxNum ];

static int n;


void Initial() //Initialize after determining the number of processes
```

```
{
    cout << "Please enter the number of jobs (processes) n=" ;
    cin >> n;

    for ( int i = 0; i < n; i++)
    {
        ArrivalTime[i] = 0;
        ServiceTime[i] = 0;
        FinishTime[i] = 0;
        WholeTime[i] = 0;
        WeightWholeTime[i] = 0;
        AverageWT_FCFS = 0;
        AverageWT_SJF = 0;
        AverageWWT_FCFS = 0;
        AverageWWT_SJF = 0;
        isFinished_FCFS[i] = false ;
        isFinished_SJF[i] = false ;
    }
}

void input() //Input the arrival time of each process [ArrivalTime], service time
[ServiceTime] and display
{
    cout << "Please enter the arrival time for each process separately:" << endl;
    for ( int i = 0; i < n; i++)
    {
        cin >> ArrivalTime[i];
    }

    cout << "Please enter the service time for each process separately:" << endl;
    for ( int i = 0; i < n; i++)
    {
        cin >> ServiceTime[i];
    }

    //Output the information entered by the user
    cout << "************************************************* *****" << endl;
    cout << "The number of processes entered by the user n=" << n << endl;

    cout << "The arrival times entered by the user are:" ;
    for ( int i = 0; i < n; i++)
    {
        cout << ArrivalTime[i] << " " ;
    }
    cout << endl;

    cout << "The service time entered by the user is:" ;
    for ( int i = 0; i < n; i++)
    {
        cout << ServiceTime[i] << " " ;
    }
    cout << endl << "************************************************** *****" << endl;
}

int get_firstProcess() //Get the highest priority process
{
    int first = MaxNum ;

    //cout << ArrivalTime[first]<<endl;
    for ( int i = 0; i < n; i++)
    {
        if (ArrivalTime[i] <= ArrivalTime[first])
        {
            first = i;
        }
        Name[i] = 65 + i;
        State[i] = 'W' ;
        Priority[i] = 0;
    }
    //cout << first+1 << endl;
    return first;
}
void display()
{
    cout << "************************************************** *****" << endl;
    cout << "Process-related information is as follows:" << endl;
    cout << setw(10) << "Process name (ID)" << " " ;
    cout << setw(10) << "Process Status" << " " ;

    cout << setw(10) << "Time of arrival" << " " ;
    cout << setw(10) << "service hours" << " " ;
    cout << setw(10) << "Complete time" << " " ;
    cout << setw(10) << "Turnaround time" << " " ;
    cout << setw(10) << "Turnaround time with rights" << endl;
    for ( int i = 0; i < n; i++)
    {
        cout << setw(10) << Name[i] << i + 1 << " " ;
        cout << setw(10) << State[i] << " " ;
        cout << setw(10) << ArrivalTime[i] << " ";
        cout << setw(10) << ServiceTime[i] << " ";
        cout << setw(10) << FinishTime[i] << " ";
        cout << setw(10) << WholeTime[i] << " ";
        cout << setw(10) << WeightWholeTime[i] << " " << endl;
    }
}
void FCFS()
{
    cout << "****************** FCFS *********************" << endl;
    /*
        1. Find the coordinates of the first arriving process and calculate related
information
        2. Find the next arriving process in turn
    */

    int startWorkTime = 0; //Indicates that the start execution time = the sum of all service
times before the current process
    int first = get_firstProcess(); // get the first process

    isFinished_FCFS[first] = true ;
    FinishTime[first] = ArrivalTime[first] + ServiceTime[first];
    startWorkTime += ServiceTime[first]; //The start execution time of the next process
    WholeTime[first] = FinishTime[first] - ArrivalTime[first]; //Turnaround Time = Finish
Time - Arrival Time
    WeightWholeTime[first] = WholeTime[first] / ServiceTime[first]; //weighted turnaround
time = turnaround time/service time

    State[first] = 'R' ;
    display();
    State[first] = 'F' ;
    //The next process
    int nextProcess = n; //Initialize the subscript of the next process out of bounds
    int m = first;
    for ( int i = 1; i < n+1; i++) //************ here i+1 modification please note
    {

        display();
        if (i > 1)
            m = nextProcess;

        nextProcess = n; //Update the subscript of the next process each time
        for ( int j = 0; j < n; j++)
        {
            if (!isFinished_FCFS[j]) //Indicates that the current process has not yet
completed the calculation of relevant information
            {
                if (ArrivalTime[j] <= startWorkTime) //When the arrival time is less than
or equal to the execution start time
                {
                    if (nextProcess == n)
                    {
                        nextProcess = j;
                    }
                    else
                    {
                        if (ArrivalTime[nextProcess] > ArrivalTime[j]) //screen out the
process that arrives first
                        {
                            nextProcess = j; //Get the current process: the process that
arrives first
                        }
                    }
                }
            }
        } //for(j)
        / / After obtaining the process nextProcess that needs to be processed currently,
calculate the relevant information

        isFinished_FCFS[nextProcess] = true ;
        FinishTime[nextProcess] = ServiceTime[nextProcess] + startWorkTime; //【Completion
Time】 = 【Service Time】 + 【Start Time Point】
        startWorkTime += ServiceTime[nextProcess]; //Get the "start execution time"
corresponding to the next process [The start time point is updated to prepare for the next
process]
        WholeTime[nextProcess] = FinishTime[nextProcess] - ArrivalTime[nextProcess]; //
【Turnaround time】 = 【Completion time】 - 【Arrival time】
        WeightWholeTime[nextProcess] = ( double )WholeTime[nextProcess] /
ServiceTime[nextProcess]; // 【Turnaround time with weight】 = 【Turnaround time】 / 【Service
time】
        State[m] = 'F' ;
        State[nextProcess] = 'R' ;

    } //for(i)

    //Calculate average turnaround time and average weighted turnaround time
    double totalWT = 0;
    double totalWWT = 0;
    for ( int i = 0; i < n; i++)
    {
        totalWT += WholeTime[i];
        totalWWT += WeightWholeTime[i];
    }
    AverageWT_FCFS = totalWT/n;
    AverageWWT_FCFS = totalWWT / n;



    // output detection
    display();
    cout << "average turnaround time=" << AverageWT_FCFS << endl;
    cout << "Average Turnaround Time with Rights=" << AverageWWT_FCFS << endl;
    cout << "************************************************** *****" << endl;
}



int main()
{
    Initial();
input();
    FCFS();

  }
```

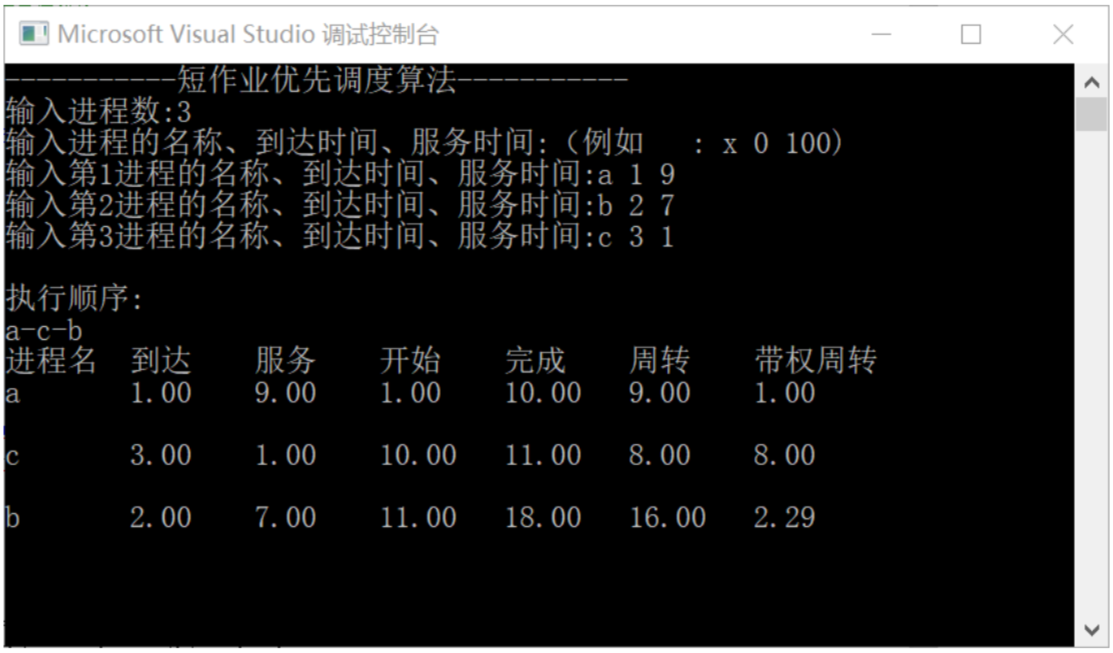


```
请输入作业（进程）个数 n=3
请分别输入每个进程的到达时间:
1 4 6
请分别输入每个进程的服务时间:
9 6 2
*******************************************************
用户输入的进程个数 n=3
用户输入的到达时间分别为: 1 4 6
用户输入的服务时间分别为: 9 6 2
*******************************************************
***************** FCFS ******************
*******************************************************
进程相关信息如下:
进程名（ID）   进程状态   到达时间   服务时间   完成时间   周转时间 带权周转时间
        A1          R          1          9         10          9          1
        B2          W          4          6          0          0          0
        C3          W          6          2          0          0          0
*******************************************************
进程相关信息如下:
进程名（ID）   进程状态   到达时间   服务时间   完成时间   周转时间 带权周转时间
        A1          F          1          9         10          9          1
        B2          W          4          6          0          0          0
        C3          W          6          2          0          0          0
*******************************************************
进程相关信息如下:
进程名（ID）   进程状态   到达时间   服务时间   完成时间   周转时间 带权周转时间
        A1          F          1          9         10          9          1
        B2          R          4          6         15         11    1.83333
        C3          W          6          2          0          0          0
*******************************************************
进程相关信息如下:
进程名（ID）   进程状态   到达时间   服务时间   完成时间   周转时间 带权周转时间
        A1          F          1          9         10          9          1
        B2          F          4          6         15         11    1.83333
        C3          R          6          2         17         11        5.5
*******************************************************
进程相关信息如下:
进程名（ID）   进程状态   到达时间   服务时间   完成时间   周转时间 带权周转时间
        A1          F          1          9         10          9          1
        B2          F          4          6         15         11    1.83333
        C3          F          6          2         17         11        5.5
平均周转时间=10.3333
平均带权周转时间=2.77778
*******************************************************
```

SJF algorithm

```
#include <stdio.h>
#include <iostream>
using namespace std;
structure sjf { //Define the structure of the process
    char name[10]; //process name
    float arrivetime; //arrival time
    float servicetime; //service time
    float starttime; //start time
    float finishtime; //finish time
    float zztime; //Turnaround time
    float dqzztime; //weighted turnaround time

};
```

```
structure sjf b[100];
//Define the maximum number of short job first algorithm processes
void Sinput( struct sjf * p , int N ) { // input function
    int i;
    printf("Enter the process name, arrival time, service time: (for example: x 0 100)\n" );
    for (i = 0; i <= N - 1; i++) {
        printf( "Enter the name, arrival time and service time of the %d process:" , i +
1);
        cin >> p [i].name >> p [i].arrivetime >> p [i].servicetime;

    }

}

// output function
void SPrint(struct sjf *p, float arrivetime, float servicetime,
    float starttime, float finishtime, float zztime, float dqzztime, int N)
{
    int k;
    printf( "\nExecution sequence:\n" );
    printf( "%s" , p [0].name);
    for (k = 1; k < N ; k++) {
        printf( "-%s" , p [k].name);
    }
    printf( "\nProcess name\tArrival\tService\tStart\tComplete\tTurnover\tTurnover with
authority\n" );
    for (k = 0; k <= N - 1; k++) {
        printf( "%s\t%-.2f\t%-.2f\t%-.2f\t%-.2f\t%-.2f\t%-.2f\t\t\n\n" ,
            p [k].name, p [k].arrivetime, p [k].servicetime, p [k].starttime, p
[k].finishtime,
            p [k].zztime, p [k].dqzztime);


    }
    printf( "\n" );

}
void Ssort( struct sjf * p , int N )
{ //Sort by shortest job first algorithm
    int i, j;
    for (i = 1; i <= N - 1; i++)
        for (j = 1; j <= i; j++)
            if (p[i].servicetime < p[j].servicetime) {
                struct sjf temp;
                temp = p[i];
                p[i] = p[j];
                p[j] = temp;

            }
}
//运行结果
void Sdeal(struct sjf *p, float arrivetime, float servicetime,
    float starttime, float finishtime, float zztime, float dqzztime, int N)

{
    int k;
    for (k = 0; k <= N - 1; k++) {
        if (k == 0) {
            p [k].starttime = p [k].arrivetime;
            p [k].finishtime = p [k].arrivetime + p [k].servicetime;
        }
        else {
            p [k].starttime = p [k - 1].finishtime;
            // start time = the completion time of the previous process
            p [k].finishtime = p [k - 1].finishtime + p [k].servicetime;
            //End time = completion time of the previous process + service time of the
current process


        }

    }




    for (k = 0; k <= N - 1; k++) {
        p [k].zztime = p [k].finishtime - p [k].arrivetime;
        // turnaround time = completion time - arrival time
        p [k].dqzztime = p [k].zztime / p [k].servicetime;
        //Powered turnaround time = turnaround time/service time

    }

}

void SJF(struct sjf *p, int N)

{
    float arrivetime = 0, servicetime = 0, starttime = 0,
        finishtime = 0, zztime = 0, dqzztime = 0;
    Ssort(p, N);
    Sdeal(p, arrivetime, servicetime, starttime, finishtime, zztime, dqzztime, N);
    SPrint(p, arrivetime, servicetime, starttime, finishtime, zztime, dqzztime, N);
}

int main() { //main function

    int M;
    printf( "-----------Short Job First Scheduling Algorithm -----------\n" );
    printf( "Enter the number of processes:" );
    cin >> M;
    Sinput(b, M);
    SJF(b, M);
    return 0;

}
```



```
-----------短作业优先调度算法-----------
输入进程数:3
输入进程的名称、到达时间、服务时间:(例如   : x 0 100)
输入第1进程的名称、到达时间、服务时间:a 1 9
输入第2进程的名称、到达时间、服务时间:b 2 7
输入第3进程的名称、到达时间、服务时间:c 3 1

执行顺序:
a-c-b
进程名  到达    服务    开始    完成    周转    带权周转
a       1.00    9.00    1.00    10.00   9.00    1.00

c       3.00    1.00    10.00   11.00   8.00    8.00

b       2.00    7.00    11.00   18.00   16.00   2.29
```