学号 P21714001 专业 计算机英才班 姓名 刘峰

实验日期 **2019.11.18**  教师签字 成绩

实验报告

【实验名称】 **时间片轮转法及FCFS、SJF进程调度算法**

【实验目的】

C++代码实现三种进程调度算法：

1. 时间片轮转法
2. 先来先服务
3. 短作业优先

【实验原理】

1. 先来先服务FCFS：

2. 短作业优先SJF：

3. 高级调度：根据某种算法，在外存中把处于后备队列中的那些作业调入内存，当作业完成时做善后处理

4. 中级调度

5. 低级调度：对象是进程（或内核级线程）；三个基本机制：排队器、分派器、上下文切换机制

6. 调度方式和算法的若干准则

1）面向用户的准则：周期时间短、响应时间快、截止时间的保证、优先权准则

2）面向系统的准则：系统吞吐量高、处理机利用率好、各类资源的平衡利用

7. 调度算法：根据系统的资源分配策略所规定的资源分配算法

8. FCFS调度算法有利于CPU繁忙型的作业，而不利于I/O繁忙型的作业或进程

1. 先来先服务调度算法FCFS：

1）是一种最简单的调度算法，适用于作业调度和进程调度

2）每次调度都是从后备队列中选择一个或者多个最先进入该队列的作业，将它们调入内存，分配资源，创建进程，然后放入就绪队列

3）FCFS算法比较有利于长作业（进程），不利于短作业（进程）

4）既可用于作业调度，也可用于进程调度

2. 周转时间 = 完成时间 - 到达时间

带权周转时间 = 周转时间/服务时间

【实验内容】

1. **时间片轮转法代码实现：**

// 操作系统\_实验二（RR调度算法）.cpp

//

#include <iostream>

#include <queue>

#include <iomanip>

#include <fstream>

using namespace std;

typedef struct

{

char name;

int ArrivalTime;

int ServiceTime;

int FinishedTime;

int WholeTime;

double WeightWholeTime;

}RR;

static queue<RR>RRqueue; //声明一个队列

static double AverageWT =0,AverageWWT=0;

static int q; //时间片

static int n; //进程个数

static RR RRarray[100]; //进程结构

void Input()

{

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"please input the number of process n = ";

cin>>n;

cout<<"please input the number of timeSlice q = ";

cin>>q;

cout<<"please input the name of process:"<<endl;

for (int i=0;i<n;i++)

{

cin>>RRarray[i].name;

}

cout<<"please input the ArrivalTime of process:"<<endl;

for (int i=0;i<n;i++)

{

cin>>RRarray[i].ArrivalTime;

}

cout<<"please input the ServiceTime of process:"<<endl;

for (int i=0;i<n;i++)

{

cin>>RRarray[i].ServiceTime;

}

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

//输出用户所输入的信息

cout<<"The information of processes is the following:"<<endl;

cout<<setw(10)<<"processName"<<" ";

cout<<setw(10)<<"ArrivalTime"<<" ";

cout<<setw(10)<<"ServiceTime"<<" "<<endl;

for (int i=0;i<n;i++)

{

cout<<setw(10)<<RRarray[i].name<<" ";

cout<<setw(10)<<RRarray[i].ArrivalTime<<" ";

cout<<setw(10)<<RRarray[i].ServiceTime<<" "<<endl;

}

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

}

void RRAlgorithm()

{

char processMoment[100]; //存储每个时间片p对应的进程名称

RRqueue.push(RRarray[0]);

int processMomentPoint = 0;

int CurrentTime=0;

int tempTime; //声明此变量控制CurrentTime的累加时间，当前进程的服务时间小于时间片q的时候，起到重要作用

int i=1; //指向还未处理的进程的下标

int finalProcessNumber = 0; //执行RR算法后，进程的个数

int processTime[50];

//CurrentTime的初始化

if (RRarray[0].ServiceTime>=q)

{

CurrentTime = q;

}

else

{

CurrentTime = RRarray[0].ServiceTime;

}

while(!RRqueue.empty())

{

for (int j=i;j<n;j++) //使得满足进程的到达时间小于当前时间的进程都进入队列

{

if (RRarray[j].name!=NULL && CurrentTime >= RRarray[j].ArrivalTime)

{

RRqueue.push(RRarray[j]);

i++;

}

}

if (RRqueue.front().ServiceTime<q)

{

tempTime = RRqueue.front().ServiceTime;

}

else

{

tempTime = q;

}

RRqueue.front().ServiceTime -= q; //进程每执行一次，就将其服务时间 -q

//将队首进程的名称放入数组中

processMoment[processMomentPoint] = RRqueue.front().name;

processMomentPoint++;

processTime[finalProcessNumber] = tempTime;

finalProcessNumber++;

if (RRqueue.front().ServiceTime <= 0) //把执行完的进程退出队列

{

//RRqueue.front().FinishedTime = CurrentTime;

RRqueue.pop(); //如果进程的服务时间小于等于，即该进程已经服务完了，将其退栈

}

else

{

//将队首移到队尾

RRqueue.push(RRqueue.front());

RRqueue.pop();

}

CurrentTime += tempTime;

}

//进程输出处理 每个时间段对应的执行的进程

cout<<"各进程的执行时刻信息："<<endl;

cout<<" "<<"0时刻 --> "<<setw(2)<<processTime[0]<<"时刻";

processTime[finalProcessNumber]=0;

int time = processTime[0];

int count = 0;

for (int i=0;i<finalProcessNumber;i++)

{

count = 0;

cout<<setw(3)<<processMoment[i]<<setw(3)<<endl;

while(RRarray[count].name!=processMoment[i] && count<n)

{

count++;

}

RRarray[count].FinishedTime = time;

if (i<finalProcessNumber - 1)

{

cout<<setw(3)<<time<<"时刻"<<" --> "<<setw(2)<<time + processTime[i+1]<<"时刻"<<setw(3);

time += processTime[i+1];

}

}

cout<<endl;

//周转时间、带权周转时间、平均周转时间、带权平均周转时间的计算

//1. 周转时间 = 完成时间 - 到达时间

//2. 带权周转时间 = 周转时间/服务时间

for (int i=0;i<n;i++)

{

RRarray[i].WholeTime = RRarray[i].FinishedTime - RRarray[i].ArrivalTime;

RRarray[i].WeightWholeTime = (double)RRarray[i].WholeTime/RRarray[i].ServiceTime;

}

double x=0,y=0;

for (int i=0;i<n;i++)

{

x += RRarray[i].WholeTime;

y += RRarray[i].WeightWholeTime;

}

AverageWT = x/n;

AverageWWT = y/n;

}

void display()

{

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"RR调度算法执行后：进程相关信息如下："<<endl;

cout<<setw(10)<<"进程名（ID）"<<" ";

cout<<setw(10)<<"到达时间"<<" ";

cout<<setw(10)<<"服务时间"<<" ";

cout<<setw(10)<<"完成时间"<<" ";

cout<<setw(10)<<"周转时间"<<" ";

cout<<setw(10)<<"带权周转时间"<<endl;

for (int i = 0;i<n;i++)

{

cout<<setw(10)<<RRarray[i].name<<" ";

cout<<setw(10)<<RRarray[i].ArrivalTime<<" ";

cout<<setw(10)<<RRarray[i].ServiceTime<<" ";

cout<<setw(10)<<RRarray[i].FinishedTime<<" ";

cout<<setw(10)<<RRarray[i].WholeTime<<" ";

cout<<setw(10)<<RRarray[i].WeightWholeTime<<" "<<endl;;

}

cout<<"所有进程的平均周转时间 = "<<AverageWT<<endl;

cout<<"所有进程的平均带权周转时间 = "<<AverageWWT<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

}

int main(int argc, char\* argv[])

{

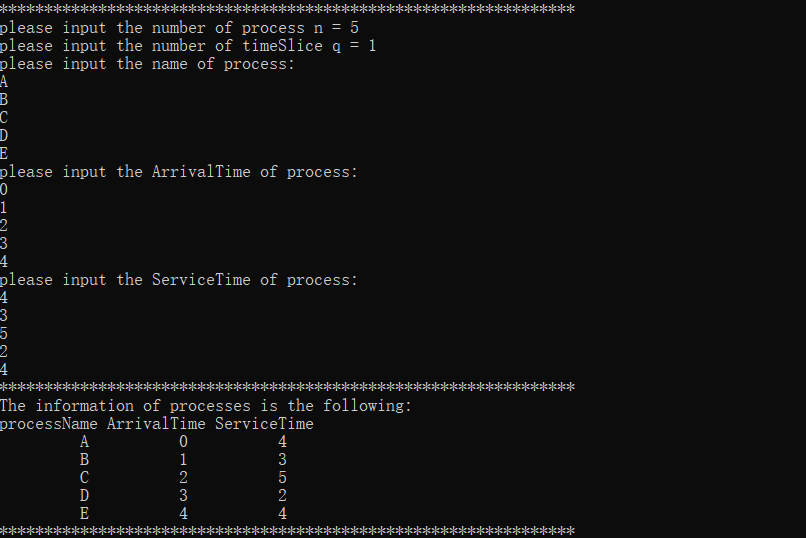
Input();

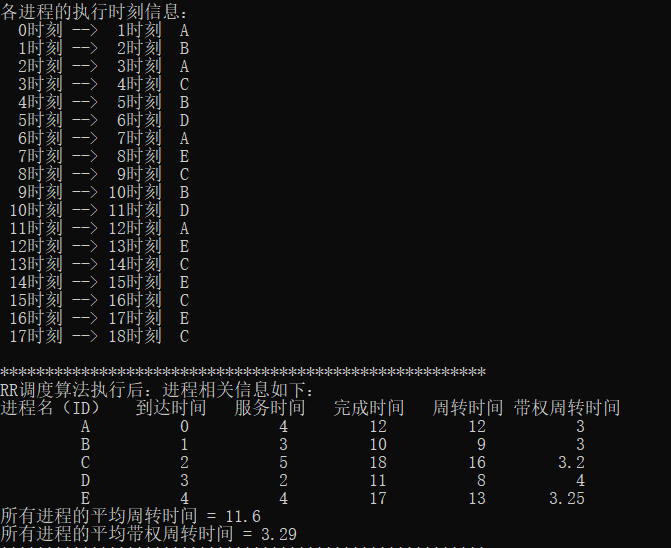
RRAlgorithm();

display();

return 0;

}





1. **先来先服务FCFS和短作业优先SJF：**

// 操作系统\_实验一.cpp :

//

/\*

//实验题目：先来先服务FCFS和短作业优先SJF进程调度算法

//#include "stdafx.h"

#include <iostream>

#include <iomanip>

using namespace std;

#define MaxNum 100

//typedef struct

//{

// int ArrivalTime[MaxNum]; //到达时间

// int ServiceTime[MaxNum]; //服务时间

// int FinishTime[MaxNum]; //完成时间

//

//

//}Process;

/\* 算法思想：

1. Initial()进行初始化

2. input()对到达时间和服务时间进行输入

3. get\_firstProcess()获得第一个进程，FCFS和SJF算法的实现相同

4. FCFS()对算法进行处理

5. SJF()对算法进行处理

6. choose\_Algorithm();对实现算法的类别进行选择：具有容错性特征

\*/

//相同的数组下标对应同一个进程的信息

int ArrivalTime[MaxNum]; //到达时间

int ServiceTime[MaxNum]; //服务时间

int FinishTime[MaxNum]; //完成时间

int WholeTime[MaxNum]; //周转时间

double WeightWholeTime[MaxNum]; //带权周转时间

double AverageWT\_FCFS,AverageWT\_SJF; //FCFS算法的平均周转时间，SJF算法的平均周转时间

double AverageWWT\_FCFS,AverageWWT\_SJF; //FCFS算法的平均带权周转时间，SJF算法的平均带权周转时间

bool isFinished\_FCFS[MaxNum];

bool isFinished\_SJF[MaxNum];

static int n;

void Initial() //确定进程个数后再初始化

{

cout<<"请输入作业（进程）个数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()

{

cout<<"请分别输入每个进程的到达时间："<<endl;

for (int i=0;i<n;i++)

{

cin>>ArrivalTime[i];

}

cout<<"请分别输入每个进程的服务时间："<<endl;

for (int i=0;i<n;i++)

{

cin>>ServiceTime[i];

}

//输出用户输入的信息

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"用户输入的进程个数n="<<n<<endl;

cout<<"用户输入的服务时间分别为："<<endl;

for (int i=0;i<n;i++)

{

cout<<ArrivalTime[i]<<" ";

}

cout<<endl;

cout<<"用户输入的服务时间分别为："<<endl;

for (int i=0;i<n;i++)

{

cout<<ServiceTime[i]<<" ";

}

cout<<endl<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

}

int get\_firstProcess()

{

int first = MaxNum;

for (int i=0;i<n;i++)

{

if (ArrivalTime[i]<=ArrivalTime[first])

{

first = i;

}

}

return first;

}

void display()

{

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"进程相关信息如下："<<endl;

cout<<setw(10)<<"进程名（ID）"<<" ";

cout<<setw(10)<<"到达时间"<<" ";

cout<<setw(10)<<"服务时间"<<" ";

cout<<setw(10)<<"完成时间"<<" ";

cout<<setw(10)<<"周转时间"<<" ";

cout<<setw(10)<<"带权周转时间"<<endl;

for (int i = 0;i<n;i++)

{

cout<<setw(10)<<i+1<<" ";

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()

{

/\*

1. 找到最先到达的进程的坐标，并计算相关信息

2. 依次找到接下去到达的进程

\*/

int startWorkTime = 0; //表示开始执行时间 = 当前进程之前的所有服务时间之和

int first = get\_firstProcess(); //获得第一个进程

isFinished\_FCFS[first] = true;

FinishTime[first] = ArrivalTime[first] + ServiceTime[first];

startWorkTime += ServiceTime[first]; //下一个进程的开始执行时间

WholeTime[first] = FinishTime[first] - ArrivalTime[first]; //周转时间 = 完成时间 - 到达时间

WeightWholeTime[first] = WholeTime[first]/ServiceTime[first]; //带权周转时间 = 周转时间/服务时间

//接下去的进程

int nextProcess = n; //初始化下一个进程的下标超出界限

for (int i=1;i<n;i++)

{

nextProcess = n; //每次对下一个进程的下标进行更新

for (int j=0;j<n;j++)

{

if (!isFinished\_FCFS[j]) //表示当前进程还未完成相关信息的计算

{

if (ArrivalTime[j]<=startWorkTime) //满足到达时间小于等于开始执行时间的情况下

{

if (nextProcess==n)

{

nextProcess = j;

}

else

{

if (ArrivalTime[nextProcess]>ArrivalTime[j]) //筛选出最先到达的进程

{

nextProcess=j; //获得当前进程中：最先到达的进程

}

}

}

}

}//for(j)

//获得当前需要处理的进程nextProcess后，对相关信息进行计算

isFinished\_FCFS[nextProcess] = true;

FinishTime[nextProcess] = ServiceTime[nextProcess] + startWorkTime;

startWorkTime += ServiceTime[nextProcess]; //获得下一个进程对应的“开始执行时间”

WholeTime[nextProcess] = FinishTime[nextProcess] - ArrivalTime[nextProcess];

WeightWholeTime[nextProcess] = (double)WholeTime[nextProcess]/ServiceTime[nextProcess];

}//for(i)

//计算平均周转时间和平均带权周转时间

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;

//输出检测

display();

cout<<"平均周转时间="<<AverageWT\_FCFS<<endl;

cout<<"平均带权周转时间="<<AverageWWT\_FCFS<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

}

void SJF()

{

//与SCSF类似，相同的方法获得第一个进程

int startWorkTime\_SJF = 0; //表示开始执行时间 = 当前进程之前的所有服务时间之和

//第一个进程的处理

int first = get\_firstProcess(); //获得第一个进程

isFinished\_SJF[first] = true;

FinishTime[first] = ArrivalTime[first] + ServiceTime[first];

startWorkTime\_SJF += ServiceTime[first]; //下一个进程的开始执行时间

WholeTime[first] = FinishTime[first] - ArrivalTime[first]; //周转时间 = 完成时间 - 到达时间

WeightWholeTime[first] = (double)WholeTime[first]/ServiceTime[first]; //带权周转时间 = 周转时间/服务时间

//获得下一个进程的下标

int nextProcess\_SJF = n;

for (int i=1;i<n;i++)

{

nextProcess\_SJF = n;

for (int j=0;j<n;j++)

{

if (!isFinished\_SJF[j])

{

if (ArrivalTime[j]<=startWorkTime\_SJF)

{

if (nextProcess\_SJF==n)

{

nextProcess\_SJF = j;

}

else

{

if (ServiceTime[nextProcess\_SJF]>ServiceTime[j])

{

nextProcess\_SJF = j; //获得运行时间最短的作业的下标

}

}

}

}

}//for(j)

//对获得的进程进行处理

isFinished\_SJF[nextProcess\_SJF] = true;

FinishTime[nextProcess\_SJF] = ServiceTime[nextProcess\_SJF] + startWorkTime\_SJF;

startWorkTime\_SJF += ServiceTime[nextProcess\_SJF];

WholeTime[nextProcess\_SJF] = FinishTime[nextProcess\_SJF] - ArrivalTime[nextProcess\_SJF];

WeightWholeTime[nextProcess\_SJF] = (double)WholeTime[nextProcess\_SJF]/ServiceTime[nextProcess\_SJF];

}//for(i)

double totalWT = 0;

double totalWWT = 0;

for (int i=0;i<n;i++)

{

totalWT+=WholeTime[i];

totalWWT+=WeightWholeTime[i];

}

AverageWT\_SJF = totalWT/n;

AverageWWT\_SJF = totalWWT/n;

//输出检测

display();

cout<<"平均周转时间="<<AverageWT\_SJF<<endl;

cout<<"平均带权周转时间="<<AverageWWT\_SJF<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

}

void choose\_Algorithm()

{

cout<<"请选择算法“1-FCFS，2-SJF”"<<endl;

int choose;

cin>>choose;

if (choose==1)

{

FCFS();

}

else if(choose==2)

{

SJF();

}

else

{

cout<<"请输入正确的选择“1-FCFS，2-SJF”"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

choose\_Algorithm(); //递归调用，实现排除错误的选择也可以继续输入

}

}

int main(int argc, char\* argv[])

{

Initial();

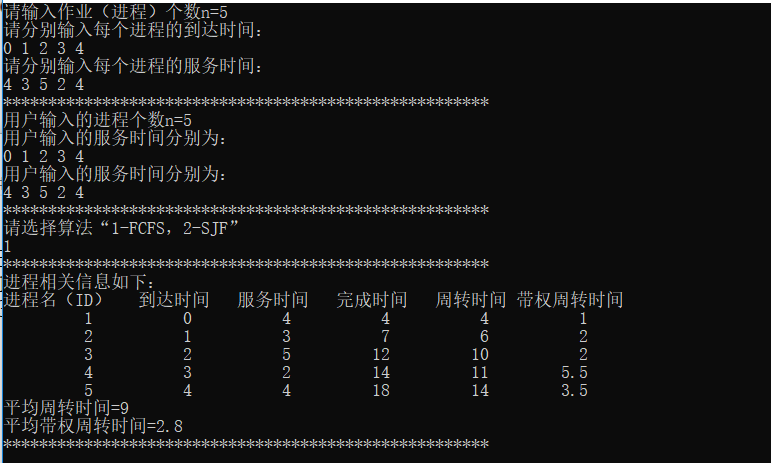
input();

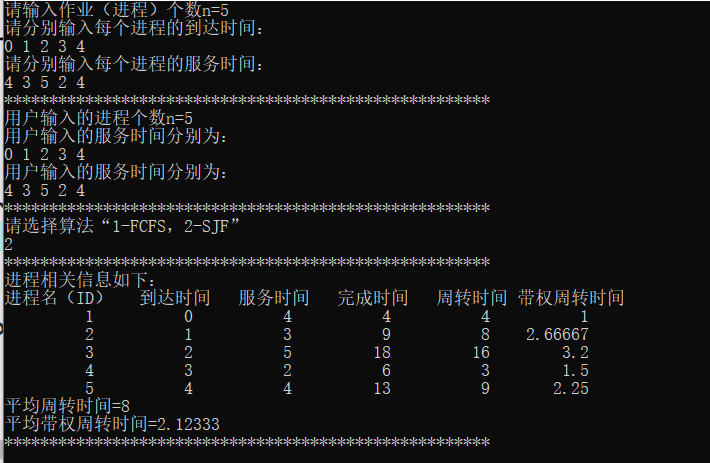
choose\_Algorithm();

return 0;

}

先来先服务和短时间优先算法实验结果“





【实验总结】

经过本次实验，我对时间片轮转法，先来先服务，短时间优先这三个操作系统处理机分配算法有了更加深入的认识，应用c++将三类算法过程实现。另外这三个实验代码相对比较长，算是我本科从大一到大三完成的代码量最长的一次实验，完成花费时间非常之长，也遇到了许多难点，但都通过查询相关书籍以及互联网等工具解决，对c++代码能力有了较大幅度的提升。总之本次实验受益匪浅！