**操作系统实验**

**实验报告**

题目：[实验一 先来先服务FCFS和短作业优先SJF进程调度算法]

**专业：[软件工程]**

**年级：[2017级]**

**姓名：[陈思翔]**

**学号：[1725121003]**

目录

[一、需求分析 4](#_Toc27925936)

[1、程序设计的任务和目的 4](#_Toc27925937)

[2、输入的形式和输入值的范围 4](#_Toc27925938)

[3、输出的形式 4](#_Toc27925939)

[4、程序所能达到的功能 4](#_Toc27925940)

[5、测试数据，包括正确的输入及其输出结果和含有错误的输入及其输出结果 4](#_Toc27925941)

[正确用例 4](#_Toc27925942)

[错误用例（输入负值服务事件引起程序错误） 5](#_Toc27925943)

[二、概要设计 5](#_Toc27925944)

[1、抽象数据类型的定义 5](#_Toc27925945)

[2、主程序的流程 6](#_Toc27925946)

[3、各程序模块之间的层次(调用)关系 6](#_Toc27925947)

[三、详细设计 8](#_Toc27925948)

[1、FCFS算法 8](#_Toc27925949)

[2、SJF算法 8](#_Toc27925950)

[四、调试分析 9](#_Toc27925951)

[调试过程中遇到的问题以及解决方法，设计与实现的回顾讨论和分析 9](#_Toc27925952)

[算法的性能分析(包括基本操作和其它算法的时间复杂度和空间复杂度的分析)及其改进设想 10](#_Toc27925953)

[性能分析 10](#_Toc27925954)

[改进设想 10](#_Toc27925955)

[五、用户使用说明 10](#_Toc27925956)

[使用说明 10](#_Toc27925957)

[六、测试结果 10](#_Toc27925958)

[测试结果，包括输入和输出 10](#_Toc27925959)

[七、附录 11](#_Toc27925960)

[带注释的源程序 11](#_Toc27925961)

# 一、需求分析

## 1、程序设计的任务和目的

通过这次实验，加深对进程概念的理解，进一步掌握进程状态的转变、进程调度的策略及对系统性能的评价方法。

## 2、输入的形式和输入值的范围

进程个数n；每个进程的到达时间T1, … ,Tn和服务时间S1, … ,Sn；选择算法1-FCFS，2-SJF。

## 3、输出的形式

要求模拟整个调度过程，输出每个时刻的进程运行状态，如“时刻3：进程B开始运行”等等；要求输出计算出来的每个进程的周转时间、带权周转时间、所有进程的平均周转时间以及带权平均周转时间。 实现提示：

## 4、程序所能达到的功能

设计程序模拟进程的先来先服务FCFS和短作业优先SJF调度过程。假设有n个进程分别在T1, … ,Tn时刻到达系统，它们需要的服务时间分别为S1, … ,Sn。分别采用先来先服务FCFS和短作业优先SJF进程调度算法进行调度，计算每个进程的完成时间、周转时间和带权周转时间，并且统计n个进程的平均周转时间和平均带权周转时间。

## 5、测试数据，包括正确的输入及其输出结果和含有错误的输入及其输出结果

### 正确用例

#### 输入

请输入进程数量：5  
请输入进程1的到达时间ArrivalTime[1]：0  
请输入进程1的服务时间ServiceTime[1]：4  
请输入进程2的到达时间ArrivalTime[2]：1  
请输入进程2的服务时间ServiceTime[2]：3  
请输入进程3的到达时间ArrivalTime[3]：2  
请输入进程3的服务时间ServiceTime[3]：5  
请输入进程4的到达时间ArrivalTime[4]：3  
请输入进程4的服务时间ServiceTime[4]：2  
请输入进程5的到达时间ArrivalTime[5]：4  
请输入进程5的服务时间ServiceTime[5]：4  
  
请选择想要先使用的算法（1-FCFS，2-SJF)：1

#### 输出

您选择的是1-FCFS算法  
时刻0进程1开始运行  
时刻5进程2开始运行  
时刻8进程3开始运行  
时刻13进程4开始运行  
时刻15进程4开始运行  
  
周转信息如下表：  
 Process 1 Process 2 Process 3 Process 4 Process 5  
ArrivalTime 0 1 2 3 4  
ServiceTime 4 3 5 2 4  
FinishTime 4 7 12 14 18  
WholeTime 4 6 10 11 14  
WeightWholeTime 1.00 2.00 2.00 5.50 3.50  
平均周转时间: 9.00  
平均带权周转时间: 2.80

### 错误用例（输入负值服务事件引起程序错误）

#### 输入

请输入进程数量：3  
请输入进程1的到达时间ArrivalTime[1]：1  
请输入进程1的服务时间ServiceTime[1]：-5  
请输入进程2的到达时间ArrivalTime[2]：2  
请输入进程2的服务时间ServiceTime[2]：-4  
请输入进程3的到达时间ArrivalTime[3]：3  
请输入进程3的服务时间ServiceTime[3]：-3  
  
请选择想要先使用的算法（1-FCFS，2-SJF)：1

#### 输出

您选择的是1-FCFS算法  
  
Process finished with exit code 0

# 二、概要设计

## 1、抽象数据类型的定义

int n;//进程数  
int ArrivalTime[MaxNum];//进程到达时间T[i]  
int ServiceTime[MaxNum];//进程服务时间S[i]  
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 = 0;//平均周转时间  
double AverageWWT = 0;//平均带权周转时间  
int isAlgorithm;//进程算法输入值，用于验证输入,1为FCFS，2为SJF

## 2、主程序的流程

int main() {  
 //创建TimeCal类  
 TimeCal timeCal{};  
  
 //输入进程信息  
 //将输入的ArrivalTime信息与ServiceTime信息存储进数组  
 timeCal.InputProcess();  
  
 //获取算法选择输入  
 timeCal.InputAlgorithm();  
  
 if (timeCal.isAlgorithm == 1) //调用FCFS算法进行调度计算  
 {  
 timeCal.AlgorithmFCFS();  
 } else//调用SJF算法进行调度计算  
 {  
 timeCal.AlgorithmSJF();  
 }  
  
 //输出调度过程  
 timeCal.PrintSchedule();  
  
 //输出周转时间、带权周转时间、平均周转时间及带权平均周转时间  
 timeCal.Print();  
  
 return 0;  
}

## 3、各程序模块之间的层次(调用)关系

int main() {  
 ···  
 //输入进程信息  
 //将输入的ArrivalTime信息与ServiceTime信息存储进数组  
 timeCal.InputProcess();  
  
 //获取算法选择输入  
 timeCal.InputAlgorithm();  
  
 if (timeCal.isAlgorithm == 1) //调用FCFS算法进行调度计算  
 {  
 timeCal.AlgorithmFCFS();  
 } else//调用SJF算法进行调度计算  
 {  
 timeCal.AlgorithmSJF();  
 }  
  
 //输出调度过程  
 timeCal.PrintSchedule();  
  
 //输出周转时间、带权周转时间、平均周转时间及带权平均周转时间  
 timeCal.Print();  
  
 return 0;  
}  
  
//输入进程信息  
//将输入的ArrivalTime信息与ServiceTime信息存储进数组  
void InputProcess() {  
 ···  
}  
  
//获取算法选择输入  
void InputAlgorithm() {  
 ···  
 IsAlgorithm();  
}  
  
//算法存储  
void IsAlgorithm() {  
 ···  
 InputAlgorithm();  
}  
  
//调用FCFS算法进行调度计算  
void AlgorithmFCFS() {  
 //计算周转信息及调度情况  
 ···  
}  
  
//调用SJF算法进行调度计算  
void AlgorithmSJF() {  
 //计算周转信息及调度情况  
 ···  
}  
  
//输出调度过程  
void PrintSchedule() {  
 ···  
}  
  
//输出周转时间、带权周转时间、平均周转时间及带权平均周转时间  
void Print() {  
 ···  
}

# 三、详细设计

**实现程序模块的具体算法**

## 1、FCFS算法

//调用FCFS算法进行调度计算  
void AlgorithmFCFS() {  
  
 //初始化FinishTime[0]  
 FinishTime[0] = 0;  
  
 for (int i = 1; i <= n; i++) {  
 //计算完成时间  
 FinishTime[i] = FinishTime[i - 1] + ServiceTime[i];  
  
 //计算周转时间  
 WholeTime[i] = FinishTime[i] - ArrivalTime[i];  
  
 //计算带权周转时间  
 WeightWholeTime[i] = (double) WholeTime[i] / ServiceTime[i];  
  
 //计算平均周转时间  
 AverageWT += WholeTime[i];  
  
 //计算平局带权周转时间  
 AverageWWT += WeightWholeTime[i];  
 }  
 AverageWT = (double) AverageWT / n;  
 AverageWWT = (double) AverageWWT / n;  
}

## 2、SJF算法

//调用SJF算法进行调度计算  
void AlgorithmSJF() {  
  
 //初始化数据  
 ···  
  
 //从小到大冒泡排序QueueServiceTime  
 ···  
  
 //计算完成时间  
 int temp = -1, old\_process = 0, new\_process = 0;//定义暂存值、上一个进程、本进程  
  
 for (int i = 1; i <= n; i++) {  
  
 old\_process = new\_process;  
  
 //查找服务时间最短的进程  
 if (i != 1) {  
 ···  
 } else//若为第1个进程，直接进入队列，完成时间即为服务时间  
 {  
 ···  
 }  
  
 //查找该最短服务时间对应的进程号  
 ···  
  
 //计算存储完成时间  
 if (i != 1) {  
 FinishTime[new\_process] = FinishTime[old\_process] + temp;  
 } else {  
 FinishTime[i] = ArrivalTime[i] + ServiceTime[i];  
 }  
  
 }  
  
 //计算周转信息及调度情况，同FCFS算法  
 ···  
}

# 四、调试分析

## 调试过程中遇到的问题以及解决方法，设计与实现的回顾讨论和分析

|  |  |  |
| --- | --- | --- |
| 问题 | 描述 | 解决方法 |
| 输出杂乱 | 数据较多，输出情况复杂，数据眼花缭乱 | 使用setw方法及setprecision方法规格化表格输出 |

## 算法的性能分析(包括基本操作和其它算法的时间复杂度和空间复杂度的分析)及其改进设想

### 性能分析

|  |  |  |
| --- | --- | --- |
| 算法 | 时间复杂度 | 空间复杂度 |
| FCFS算法 | T(n) = O(n) | S(n) = O(n) |
| SJF算法 | T(n) = O(n2) | S(n) = O(n) |
| PrintSchedule输出调度过程 | T(n) = O(n) | S(n) = O(n) |
| Print数据输出 | T(n) = O(n) | S(n) = O(n) |

### 改进设想

加入多次算法选择功能

# 五、用户使用说明

## 使用说明

* 控制台会提示要求用户进行输入，按提示输入内容即可
* 不可输入负值服务时间，会造成错误
* 选择算法输入完成后将会输出调度过程及周转信息

# 六、测试结果

## 测试结果，包括输入和输出

D:\Documents\MyCourse\OperatingSystem\cmake-build-debug\chapter01.exe  
请输入进程数量：5  
请输入进程1的到达时间ArrivalTime[1]：0  
请输入进程1的服务时间ServiceTime[1]：4  
请输入进程2的到达时间ArrivalTime[2]：1  
请输入进程2的服务时间ServiceTime[2]：3  
请输入进程3的到达时间ArrivalTime[3]：2  
请输入进程3的服务时间ServiceTime[3]：5  
请输入进程4的到达时间ArrivalTime[4]：3  
请输入进程4的服务时间ServiceTime[4]：2  
请输入进程5的到达时间ArrivalTime[5]：4  
请输入进程5的服务时间ServiceTime[5]：4  
  
请选择想要先使用的算法（1-FCFS，2-SJF)：1  
  
您选择的是1-FCFS算法  
时刻0进程1开始运行  
时刻5进程2开始运行  
时刻8进程3开始运行  
时刻13进程4开始运行  
时刻15进程4开始运行  
  
周转信息如下表：  
 Process 1 Process 2 Process 3 Process 4 Process 5  
ArrivalTime 0 1 2 3 4  
ServiceTime 4 3 5 2 4  
FinishTime 4 7 12 14 18  
WholeTime 4 6 10 11 14  
WeightWholeTime 1.00 2.00 2.00 5.50 3.50  
平均周转时间: 9.00  
平均带权周转时间: 2.80  
  
Process finished with exit code 0

# 七、附录

## 带注释的源程序

#include <iostream>  
#include <iomanip>  
  
using namespace std;  
#define MaxNum 100  
  
class TimeCal {  
public:  
  
 int n;//进程数  
 int ArrivalTime[MaxNum];//进程到达时间T[i]  
 int ServiceTime[MaxNum];//进程服务时间S[i]  
 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 = 0;//平均周转时间  
 double AverageWWT = 0;//平均带权周转时间  
 int isAlgorithm;//进程算法输入值，用于验证输入,1为FCFS，2为SJF  
  
 //输入进程信息  
 //将输入的ArrivalTime信息与ServiceTime信息存储进数组  
 void InputProcess() {  
 cout << "请输入进程数量：";  
 cin >> n;  
 for (int i = 1; i <= n; i++) {  
 cout << "请输入进程" << i << "的到达时间ArrivalTime[" << i << "]：";  
 cin >> ArrivalTime[i];  
 cout << "请输入进程" << i << "的服务时间ServiceTime[" << i << "]：";  
 cin >> ServiceTime[i];  
 }  
 }  
  
 //获取算法选择输入  
 void InputAlgorithm() {  
 cout << endl << "请选择想要先使用的算法（1-FCFS，2-SJF)：";  
 cin >> isAlgorithm;  
 IsAlgorithm();  
 }  
  
 //算法存储  
 void IsAlgorithm() {  
 if (isAlgorithm == 1) {  
 cout << endl << "您选择的是1-FCFS算法" << endl;  
 } else if (isAlgorithm == 2) {  
 cout << endl << "您选择的是2-SJF算法" << endl;  
 } else {  
 cout << "算法值" << isAlgorithm << "有误,请重新输入正确的算法类型（1-FCFS，2-SJF）" << endl;  
 InputAlgorithm();  
 }  
 }  
  
 //调用FCFS算法进行调度计算  
 void AlgorithmFCFS() {  
  
 //初始化FinishTime[0]  
 FinishTime[0] = 0;  
  
 for (int i = 1; i <= n; i++) {  
 //计算完成时间  
 FinishTime[i] = FinishTime[i - 1] + ServiceTime[i];  
  
 //计算周转时间  
 WholeTime[i] = FinishTime[i] - ArrivalTime[i];  
  
 //计算带权周转时间  
 WeightWholeTime[i] = (double) WholeTime[i] / ServiceTime[i];  
  
 //计算平均周转时间  
 AverageWT += WholeTime[i];  
  
 //计算平局带权周转时间  
 AverageWWT += WeightWholeTime[i];  
 }  
 AverageWT = (double) AverageWT / n;  
 AverageWWT = (double) AverageWWT / n;  
 }  
  
 //调用SJF算法进行调度计算  
 void AlgorithmSJF() {  
  
 int QueueServiceTime[MaxNum];//排序服务时间  
 int CopyServiceTime[MaxNum];//备份服务时间  
  
 //初始化QueueServiceTime  
 for (int i = 1; i <= n; i++) {  
 QueueServiceTime[i] = ServiceTime[i];  
 }  
  
 //初始化CopyServiceTime  
 for (int i = 1; i <= n; i++) {  
 CopyServiceTime[i] = ServiceTime[i];  
 }  
  
 //给FinishTime[0]附初值  
 FinishTime[0] = 0;  
  
 //从小到大冒泡排序QueueServiceTime  
 for (int i = 1; i <= n; i++) {  
 for (int j = 1; j <= n - i; j++) {  
 if (QueueServiceTime[j] > QueueServiceTime[j + 1]) {  
 int temp = QueueServiceTime[j];  
 QueueServiceTime[j] = QueueServiceTime[j + 1];  
 QueueServiceTime[j + 1] = temp;  
 }  
 }  
 }  
  
 //计算完成时间  
 int temp = -1, old\_process = 0, new\_process = 0;//定义暂存值、上一个进程、本进程  
  
 for (int i = 1; i <= n; i++) {  
  
 old\_process = new\_process;  
  
 //查找服务时间最短的进程  
 if (i != 1) {  
 for (int j = 1; j <= n; j++) {  
 if (QueueServiceTime[j] > 0 && ArrivalTime[j] <= FinishTime[old\_process] + 1) {  
 temp = QueueServiceTime[j];  
 QueueServiceTime[j] = -1;  
 break;  
 }  
 }  
 } else//若为第1个进程，直接进入队列，完成时间即为服务时间  
 {  
 for (int j = 1; j <= n; j++) {  
 if (QueueServiceTime[j] == ServiceTime[i]) {  
 temp = QueueServiceTime[j];  
 QueueServiceTime[j] = -1;  
 break;  
 }  
 }  
 }  
  
 //查找该最短服务时间对应的进程号  
 for (int j = 1; j <= n; j++) {  
 if (CopyServiceTime[j] == temp && CopyServiceTime[j] > 0) {  
 new\_process = j;  
 CopyServiceTime[j] = -1;  
 break;  
 }  
 }  
  
 //计算存储完成时间  
 if (i != 1) {  
 FinishTime[new\_process] = FinishTime[old\_process] + temp;  
 } else {  
 FinishTime[i] = ArrivalTime[i] + ServiceTime[i];  
 }  
  
 }  
  
 for (int i = 1; i <= n; i++) {  
 //计算周转时间  
 WholeTime[i] = FinishTime[i] - ArrivalTime[i];  
  
 //计算带权周转时间  
 WeightWholeTime[i] = (double) WholeTime[i] / ServiceTime[i];  
  
 //计算平均周转时间  
 AverageWT += WholeTime[i];  
  
 //计算平局带权周转时间  
 AverageWWT += WeightWholeTime[i];  
 }  
 cout << endl;  
  
 AverageWT = (double) AverageWT / n;  
 AverageWWT = (double) AverageWWT / n;  
 }  
  
 //输出调度过程  
 void PrintSchedule() {  
 int temp = 0;  
 int CopyFinishTime[MaxNum];//备份服务时间  
  
 //初始化CopyServiceTime  
 for (int i = 1; i <= n; i++) {  
 CopyFinishTime[i] = FinishTime[i];  
 }  
  
 if (FinishTime[1] >= 0)//判断是否已有数据输入  
 //根据完成时间排序输出  
 for (int i = 1; i <= n; i++) {  
  
 if (i != 1) {  
 for (int j = 1; j <= n; j++) {  
 if (temp != 0) {  
 if (CopyFinishTime[j] > 0) {  
 temp = (CopyFinishTime[temp] < CopyFinishTime[j] ? temp : j);  
 }  
 } else {  
 if (CopyFinishTime[j] > 0) {  
 temp = j;  
 }  
 }  
 }  
 if (i != 2) { cout << "进程" << temp << "开始运行" << endl; }  
 cout << "时刻" << FinishTime[temp] + 1;  
 if (i == n) {  
 for (int j = 1; j < n; j++) {  
 if (CopyFinishTime[j] > 0) {  
 temp = j;  
 }  
 }  
 cout << "进程" << temp << "开始运行" << endl << endl;  
 }  
 CopyFinishTime[temp] = -1;  
 temp = 0;  
 } else {  
 cout << "时刻0进程" << i << "开始运行" << endl;  
 }  
 }  
 }  
  
 //输出周转时间、带权周转时间、平均周转时间及带权平均周转时间  
 void Print() {  
 if (FinishTime[1] >= 0)//判断是否已有数据输入  
 {  
 cout << left << setw(15) << "周转信息如下表：" << endl;  
  
 cout << left << setw(15) << "";  
 for (int i = 1; i <= n; i++) {  
 cout << right << setw(8) << "Process" << setw(2) << i;  
 }  
 cout << endl;  
  
 cout << left << setw(15) << "ArrivalTime";  
 for (int i = 1; i <= n; i++) {  
 cout << right << setw(10) << ArrivalTime[i];  
 }  
 cout << endl;  
  
 cout << left << setw(15) << "ServiceTime";  
 for (int i = 1; i <= n; i++) {  
 cout << right << setw(10) << ServiceTime[i];  
 }  
 cout << endl;  
  
 cout << left << setw(15) << "FinishTime";  
 for (int i = 1; i <= n; i++) {  
 cout << right << setw(10) << FinishTime[i];  
 }  
 cout << endl;  
  
 cout << left << setw(15) << "WholeTime";  
 for (int i = 1; i <= n; i++) {  
 cout << right << setw(10) << WholeTime[i];  
 }  
 cout << endl;  
  
 cout << left << setw(15) << "WeightWholeTime";  
 for (int i = 1; i <= n; i++) {  
 cout << right << setw(10) << fixed << setprecision(2) << WeightWholeTime[i];  
 }  
 cout << endl;  
  
 cout << "平均周转时间:" << setw(10) << setprecision(2) << AverageWT << endl;  
 cout << "平均带权周转时间:" << setw(10) << setprecision(2) << AverageWWT << endl;  
 }  
 }  
};  
  
  
int main() {  
 //创建TimeCal类  
 TimeCal timeCal{};  
  
 //输入进程信息  
 //将输入的ArrivalTime信息与ServiceTime信息存储进数组  
 timeCal.InputProcess();  
  
 //获取算法选择输入  
 timeCal.InputAlgorithm();  
  
 if (timeCal.isAlgorithm == 1) //调用FCFS算法进行调度计算  
 {  
 timeCal.AlgorithmFCFS();  
 } else//调用SJF算法进行调度计算  
 {  
 timeCal.AlgorithmSJF();  
 }  
  
 //输出调度过程  
 timeCal.PrintSchedule();  
  
 //输出周转时间、带权周转时间、平均周转时间及带权平均周转时间  
 timeCal.Print();  
  
 return 0;  
}