**操作系统实验**

**实验报告**

题目：[实验二 时间片轮转RR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](#_Toc27926603)

[1、程序设计的任务和目的 4](#_Toc27926604)

[2、输入的形式和输入值的范围 4](#_Toc27926605)

[3、输出的形式 4](#_Toc27926606)

[4、程序所能达到的功能 4](#_Toc27926607)

[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](#_Toc27926608)

[正确用例 4](#_Toc27926609)

[错误用例（输入负值服务时间造成错误） 5](#_Toc27926610)

[二、概要设计 6](#_Toc27926611)

[1、抽象数据类型的定义 6](#_Toc27926612)

[2、主程序的流程 6](#_Toc27926613)

[3、各程序模块之间的层次(调用)关系 6](#_Toc27926614)

[三、详细设计 7](#_Toc27926615)

[1、RR算法调度计算 8](#_Toc27926616)

[四、调试分析 9](#_Toc27926617)

[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](#_Toc27926618)

[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 9](#_Toc27926619)

[性能分析 9](#_Toc27926620)

[改进设想 9](#_Toc27926621)

[五、用户使用说明 9](#_Toc27926622)

[使用说明 9](#_Toc27926623)

[六、测试结果 10](#_Toc27926624)

[测试结果，包括输入和输出 10](#_Toc27926625)

[七、附录 11](#_Toc27926626)

[带注释的源程序 11](#_Toc27926627)

# 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；输入时间片大小q。

## 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。

## 4、程序所能达到的功能

设计程序模拟进程的时间片轮转RR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。分别利用不同的时间片大小q，采用时间片轮转RR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  
请输入时间片大小：4

#### 输出

时刻1进程1开始运行  
时刻4进程1结束运行  
时刻5进程2开始运行  
时刻7进程2结束运行  
时刻8进程3开始运行  
时刻12进程4开始运行  
时刻13进程4结束运行  
时刻14进程5开始运行  
时刻17进程5结束运行  
时刻18进程3开始运行  
时刻18进程3结束运行  
  
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 18 13 17  
WholeTime 4 6 16 10 13  
WeightWholeTime 1.00 2.00 3.20 5.00 3.25  
平均周转时间:9.80  
平均带权周转时间:2.89

### 错误用例（输入负值服务时间造成错误）

#### 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  
请输入时间片大小：1

#### 输出

时刻1进程1开始运行  
时刻-5进程1结束运行  
时刻-4进程2开始运行  
时刻-9进程2结束运行  
时刻-8进程3开始运行  
时刻-12进程3结束运行  
  
周转信息如下表：  
 Process 1 Process 2 Process 3  
ArrivalTime 1 2 3  
ServiceTime -5 -4 -3  
FinishTime -5 -9 -12  
WholeTime -6 -11 -15  
WeightWholeTime 1.20 2.75 5.00  
平均周转时间:-10.67  
平均带权周转时间:2.98

# 二、概要设计

## 1、抽象数据类型的定义

int n;//进程数  
int q;//时间片大小  
int ArrivalTime[MaxNum];//进程到达时间T[i]  
int ServiceTime[MaxNum];//进程服务时间S[i]  
double AverageWT = 0;//平均周转时间  
double AverageWWT = 0;//平均带权周转时间  
  
//定义进程的数据结构  
typedef struct {  
 int index;  
 int ArrivalTime;//进程到达时间  
 int ServiceTime;//进程服务时间  
 int PServiceTime;//剩余进程服务时间P[i]  
 int FinishedTime;//完成时间  
 int WholeTime;//周转时间  
 double WeightWholeTime;//带权周转时间  
 bool Finished;//完成状态  
} Progress;

## 2、主程序的流程

int main() {  
 RRSchedule rrSchedule{};  
 rrSchedule.InputProcess();  
 rrSchedule.InitQueue();  
 rrSchedule.AlgorithmRR();  
 rrSchedule.Print();  
 return 0;  
}

## 3、各程序模块之间的层次(调用)关系

int main() {  
 RRSchedule rrSchedule{};  
 rrSchedule.InputProcess();  
 rrSchedule.InitQueue();  
 rrSchedule.AlgorithmRR();  
 rrSchedule.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() {  
 ···  
}  
  
//初始化队列  
void InitQueue() {  
 for (int i = 1; i <= n; i++) {  
 ···  
 queueRR.push(progress[i]);  
 }  
}  
  
//调用RR算法进行调度计算  
void AlgorithmRR() {  
 int finishTime = 0;  
 while (!queueRR.empty()) {  
 //按进程顺序读取队列首值进行操作，并出队列，判断其是否完成，若无，则插入队列尾，并在执行过程中输出调度过程  
 ···  
 }  
  
 //计算属性  
 for (int i = 1; i <= n; i++) {  
 ···  
 }  
  
 //计算平均周转时间  
 AverageWT /= (double) n;  
  
 //计算平局带权周转时间  
 AverageWWT /= (double) n;  
}  
  
//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、RR算法调度计算

//调用RR算法进行调度计算  
void AlgorithmRR() {  
 int finishTime = 0;  
 while (!queueRR.empty()) {  
 //按进程顺序读取队列首值进行操作，并出队列，判断其是否完成，若无，则插入队列尾，并在执行过程中输出调度过程  
 int x = queueRR.front().index;  
 if (progress[x].PServiceTime > q) {//进程未结束  
 cout << "时刻" << finishTime + 1 << "进程" << x << "开始运行" << endl;  
 finishTime += q;  
 progress[x].PServiceTime -= q;  
 queueRR.pop();  
 queueRR.push(progress[x]);  
 } else {//进程结束  
 cout << "时刻" << finishTime + 1 << "进程" << x << "开始运行" << endl;  
 progress[x].Finished = true;  
 finishTime += progress[x].PServiceTime;  
 progress[x].PServiceTime = 0;  
 progress[x].FinishedTime = finishTime;  
 cout << "时刻" << finishTime << "进程" << x << "结束运行" << endl;  
 queueRR.pop();  
 }  
 }  
  
 //计算属性  
 for (int i = 1; i <= n; i++) {  
 //计算周转时间  
 progress[i].WholeTime = progress[i].FinishedTime - progress[i].ArrivalTime;  
  
 //计算带权周转时间  
 progress[i].WeightWholeTime = (double) progress[i].WholeTime / progress[i].ServiceTime;  
  
 //计算平均周转时间  
 AverageWT += progress[i].WholeTime;  
  
 //计算平局带权周转时间  
 AverageWWT += progress[i].WeightWholeTime;  
 }  
  
 //计算平均周转时间  
 AverageWT /= (double) n;  
  
 //计算平局带权周转时间  
 AverageWWT /= (double) n;  
}

# 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 |
| 队列更改 | 使用数据结构数组队列进行算法调度，无法改变数据 | 在数据结构中设计index位用于更改数组中数据 |

## 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 |
| AlgorithmRR算法调度计算 | T(n) = O(n2) | S(n) = O(n) |
| InitQueue初始化队列算法 | 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\chapter02.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  
请输入时间片大小：4  
  
时刻1进程1开始运行  
时刻4进程1结束运行  
时刻5进程2开始运行  
时刻7进程2结束运行  
时刻8进程3开始运行  
时刻12进程4开始运行  
时刻13进程4结束运行  
时刻14进程5开始运行  
时刻17进程5结束运行  
时刻18进程3开始运行  
时刻18进程3结束运行  
  
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 18 13 17  
WholeTime 4 6 16 10 13  
WeightWholeTime 1.00 2.00 3.20 5.00 3.25  
平均周转时间:9.80  
平均带权周转时间:2.89  
  
Process finished with exit code 0

# 七、附录

## 带注释的源程序

#include <iostream>  
#include <queue>  
#include <iomanip>  
  
using namespace std;  
#define MaxNum 100  
  
class RRSchedule {  
public:  
  
 int n;//进程数  
 int q;//时间片大小  
 int ArrivalTime[MaxNum];//进程到达时间T[i]  
 int ServiceTime[MaxNum];//进程服务时间S[i]  
 double AverageWT = 0;//平均周转时间  
 double AverageWWT = 0;//平均带权周转时间  
  
 //定义进程的数据结构  
 typedef struct {  
 int index;  
 int ArrivalTime;//进程到达时间  
 int ServiceTime;//进程服务时间  
 int PServiceTime;//剩余进程服务时间P[i]  
 int FinishedTime;//完成时间  
 int WholeTime;//周转时间  
 double WeightWholeTime;//带权周转时间  
 bool Finished;//完成状态  
 } Progress;  
  
 queue<Progress> queueRR;  
 Progress progress[MaxNum];  
  
 //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];  
 }  
 cout << "请输入时间片大小：";  
 cin >> q;  
 cout << endl;  
 }  
  
 void InitQueue() {  
 for (int i = 1; i <= n; i++) {  
 progress[i].index = i;  
 progress[i].ArrivalTime = ArrivalTime[i];  
 progress[i].ServiceTime = ServiceTime[i];  
 progress[i].PServiceTime = progress[i].ServiceTime;  
 progress[i].Finished = false;  
 progress[i].FinishedTime = -1;  
 queueRR.push(progress[i]);  
 }  
 }  
  
 //调用RR算法进行调度计算  
 void AlgorithmRR() {  
 int finishTime = 0;  
 while (!queueRR.empty()) {  
 //按进程顺序读取队列首值进行操作，并出队列，判断其是否完成，若无，则插入队列尾，并在执行过程中输出调度过程  
 int x = queueRR.front().index;  
 if (progress[x].PServiceTime > q) {//进程未结束  
 cout << "时刻" << finishTime + 1 << "进程" << x << "开始运行" << endl;  
 finishTime += q;  
 progress[x].PServiceTime -= q;  
 queueRR.pop();  
 queueRR.push(progress[x]);  
 } else {//进程结束  
 cout << "时刻" << finishTime + 1 << "进程" << x << "开始运行" << endl;  
 progress[x].Finished = true;  
 finishTime += progress[x].PServiceTime;  
 progress[x].PServiceTime = 0;  
 progress[x].FinishedTime = finishTime;  
 cout << "时刻" << finishTime << "进程" << x << "结束运行" << endl;  
 queueRR.pop();  
 }  
 }  
  
 //计算属性  
 for (int i = 1; i <= n; i++) {  
 //计算周转时间  
 progress[i].WholeTime = progress[i].FinishedTime - progress[i].ArrivalTime;  
  
 //计算带权周转时间  
 progress[i].WeightWholeTime = (double) progress[i].WholeTime / progress[i].ServiceTime;  
  
 //计算平均周转时间  
 AverageWT += progress[i].WholeTime;  
  
 //计算平局带权周转时间  
 AverageWWT += progress[i].WeightWholeTime;  
 }  
  
 //计算平均周转时间  
 AverageWT /= (double) n;  
  
 //计算平局带权周转时间  
 AverageWWT /= (double) n;  
 }  
  
 //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() {  
 cout << endl << 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) << progress[i].ArrivalTime;  
 }  
 cout << endl;  
  
 cout << left << setw(15) << "ServiceTime";  
 for (int i = 1; i <= n; i++) {  
 cout << right << setw(10) << progress[i].ServiceTime;  
 }  
 cout << endl;  
  
 cout << left << setw(15) << "FinishTime";  
 for (int i = 1; i <= n; i++) {  
 cout << right << setw(10) << progress[i].FinishedTime;  
 }  
 cout << endl;  
  
 cout << left << setw(15) << "WholeTime";  
 for (int i = 1; i <= n; i++) {  
 cout << right << setw(10) << progress[i].WholeTime;  
 }  
 cout << endl;  
  
 cout << left << setw(15) << "WeightWholeTime";  
 for (int i = 1; i <= n; i++) {  
 cout << right << setw(10) << fixed << setprecision(2) << progress[i].WeightWholeTime;  
 }  
 cout << endl;  
  
 cout << "平均周转时间:" << setprecision(2) << AverageWT << endl;  
 cout << "平均带权周转时间:" << setprecision(2) << AverageWWT << endl;  
 }  
  
  
};  
  
int main() {  
 RRSchedule rrSchedule{};  
 rrSchedule.InputProcess();  
 rrSchedule.InitQueue();  
 rrSchedule.AlgorithmRR();  
 rrSchedule.Print();  
 return 0;  
}