南京邮电大学

通达学院
课程设计 I 报告

（2017/2018 学年 第 2 学期）

题　　目： 时间片轮转进程调度算法

专　　业	计算机科学与技术
学 生 姓 名	
班 级 学 号	
指 导 教 师	
指 导 单 位	计算机学院计算机科学与技术系
日　　期	2018.5.23-6.1

指导教师成绩评定表

学生姓名		班级学号		专业	计算机科学与技术		
评分内容	评分标准			优秀	良好	中等	差
平时成绩	认真对待课程设计，遵守实验室规定，上机不迟到早退，不做和设计无关的事						
设计成果	设计的科学、合理性						
	功能丰富、符合题目要求						
	界面友好、外观漂亮、大方						
	程序功能执行的正确性						
	程序算法执行的效能						
设计报告	设计报告正确合理、反映系统设计流程						
	文档内容详实程度						
	文档格式规范、排版美观						
验收答辩	简练、准确阐述设计内容，能准确有条理回答各种问题，系统演示顺利。						
评分等级							
指导教师简短评语							
指导教师签名			日期				
备注	评分等级有五种：优秀、良好、中等、及格、不及格						

目录

时间片轮转进程调度算法

一、课题内容和要求

1.1 研究的背景及意义

得益于人们日益增长的计算需求和事务处理需求，电子计算机获得了长足稳定的发展。操作系统，作为计算机中不可或缺的资源管理者和人机交互枢纽，也随之不断发展演化。在操作系统诞生伊始便出现的时间片轮转调度算法，在操作系统的整个发展历程中，起到了重要的作用。

早期的计算机普遍采用批处理或分时操作系统。批处理操作系统虽然效率很高，但一旦处理开始后就无法交互，只能等待处理结果而无法得知具体进展情况，不利于程序调试和纠错。为了改善这一情况，分时操作系统应运而生。在分时操作系统中，多个联机用户可同时使用一个计算机系统，系统把处理器的时间划分为时间片，使用时间片轮转调度算法轮流分配给各个联机终端，因此响应速度得到了极大的提高。

到了现代，由于计算机架构的不断发展和硬件技术的不断成熟，操作系统也逐渐演变成为了兼具批处理、分时和实时全部功能的通用操作系统，但时间片轮转调度算法依旧在被使用。当多道互相独立的程序共同占用系统资源时，操作系统可以使用时间片轮转调度算法，使它们同处于开始到结束之间的状态并发执行，共享计算机系统资源。

对时间片轮转调度算法的研究，进一步涉及到进程的概念、进程状态转变、进程调度策略以及系统性能评价方法。因此，学习和理解该算法，在今天依旧意义非凡。

1.2 主要研究内容

此次设计将要实现时间片轮转调度算法的模拟过程。

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。采用不同的时间片大小 q，利用时间片轮转进程调度算法进行调度。模拟并输出整个调度过程，输出每个时刻进程的运行状态。计算每个进程的周转时间、带权周转时间，以及所有进程的平均周转时间、平均带权周转时间，以评估不同时间片大小 q 下的系统性能差异。

本次实验成果具有以下特点：系统中模块划分明确，模块功能设计有较强的针对性。交互界面整洁简单，灵活性好，程序具有良好的鲁棒性。

二、需求分析

2.1 系统模块说明

2.1.1 输入模块

（1）函数 Input1()：文件读取模式的输入函数。用于在用户选择以文件读取模式输入后，从程序根目录下的 RR_data.txt 中读取相关初始信息。

（2）函数 Input2()：手动输入模式的输入函数。用于在用户选择以手动输入模式输入后，在交互界面读取用户从键盘输入的相关初始信息。

（3）函数 Output()：初始信息输出函数。用于在输入完成后，对输入结果进行输出显示，供用户确认。

2.1.2 算法模拟计算模块

（1）函数 RR_Simulate()：RR 算法模拟计算函数。用于模拟整个时间片轮转调度算法的过程，将过程的关键数据存储到对应的数据结构中。

（2）函数 RR_Calculate()：计算各个进程的周转时间、带权周转时间，以及所有进程的平均周转时间、平均带权周转时间，并进行存储。

2.1.3 输出模块

（1）函数 display1()：根据 RR_Simulate()函数的存储结果，输出模拟执行的整个过程。

（2）函数 display2()：根据 RR_Calculate()函数的存储结果，输出进程相关信息。

2.2 输入输出形式

2.2.1 输入形式

文件读取模式和手动输入模式的输入形式基本相同。

依次输入：进程个数 n

时间片长度 q

进程名

进程到达时间

进程服务时间

2.2.2 输出形式

（1）过程模拟的输出

依次输出：0 时刻 → T1 时刻　执行进程名 1

T1 时刻 → T2 时刻　执行进程名 2

……

Tn-1 时刻 → Tn 时刻　执行进程名 n

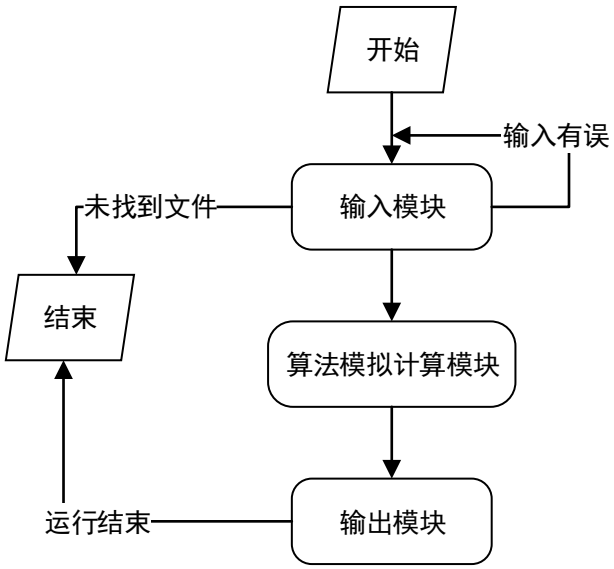
（2）进程相关信息的输出

进程名	到达时间	服务时间	完成时间	周转时间	带权周转时间

所有进程平均周转时间：

所有进程平均带权周转时间：

2.3 系统模块流程图



三、概要设计

3.1 设备环境

（1）硬件环境

处理器：i5-4200H CPU @ 2.80GHz，内存：8GB

（2）工作平台

Microsoft Visual Studio 2017(IDE)

3.2 数据结构

（1）结构体

```
//进程结构体
typedef struct {
    char name;        //进程名
    int ArrivalTime;  //到达时间
    int ServiceTime;  //服务时间
    int FinishedTime; //完成时间
    int WholeTime;    //周转时间
    double WeightWholeTime;  //带权周转时间
}RR;
```

（2）队列

```
static queue<RR>RRqueue;  //声明等待队列
static queue<RR>Pqueue;  //声明总进程队列
```

（3）数组

```
static RR RRarray[100];  //进程数组
static char processMoment[100];  //存储每个过程执行进程的名称
static int processSTime[50];  //存储每个过程的开始时刻
static int processFTime[50];  //存储每个过程的结束时刻
```

3.3 算法说明

3.3.1 时间片轮转调度算法模拟部分

（1）文字说明

所有进程先根据到达时间先后次序依次进入总进程队列，再随着当前时刻的变化依次从总进程队列队首出，转而进入等待队列。

调度程序每次把 CPU 分配给等待队列队首进程时使用规定的时间间隔，即时间片，通常为 10ms~200ms。就绪队列中的每个进程轮流地运行一个时间片，当时间片耗尽（或进程运行结束）时就强迫当前运行进程让出处理器，转而排列到就绪队列尾部，等候下一轮调度。

当等待队列非空时，调度按照上述方法进行，每次进程运行后保存该次过程的各种信息（开始时刻，结束时刻，运行进程名等），更新当前时刻，并将已经到达的进程加入等待队列中；当等待队列为空且总进程队列非空时，将当前时刻调整到最快将要进入就绪队列的进程的到达时间，将该进程放入等待队列，调度继续进行；当等待队列为空且总进程队列为空时，所有进程均运行完毕。

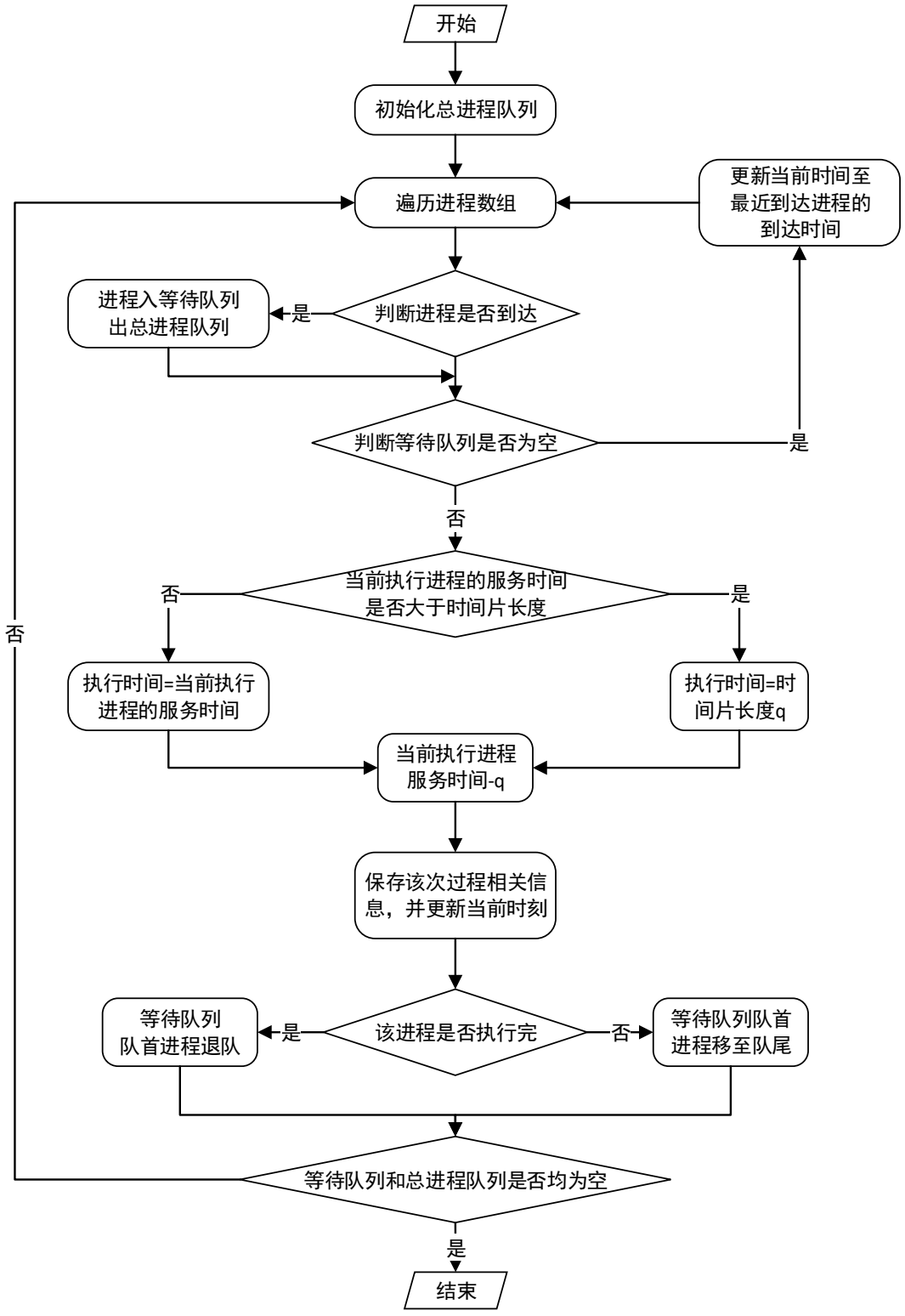
（2）伪代码说明

初始化总进程队列，将所有进程按到达时间先后依次进入队列;

```
while（等待队列不为空且总进程队列不为空）{
```

```
    for (遍历整个进程数组) {
        if (进程到达时间 < 当前时刻) {
            进程进入等待队列;
            进程出总进程队列;
        }
    }
    if (等待队列为空且总进程队列不为空) {
            当前时间 = 总进程队列队首进程的到达时间;
    }
    for (遍历整个进程数组) {
        if (进程到达时间 < 当前时刻) {
            进程进入等待队列;
            进程出总进程队列;
        }
    }
    if (等待队列队首进程服务时间 < 时间片长度q) {
        本次过程执行时间 = 等待队列队首进程服务时间;
    }
    else {
        本次过程执行时间 = 时间片长度q;
    }
    等待队列队首进程服务时间 - q;
    保存该次过程的开始时间;
     更新当前时刻;
    保存该次过程执行进程的名称;
    保存该次过程的结束时间;
    当前执行过程数 + 1;
    if (等待队列队首进程执行完) {
        等待队列队首进程退队;
    }
    else {
        等待队列队首进程移至队尾;
    }
}
```

（3）流程图说明

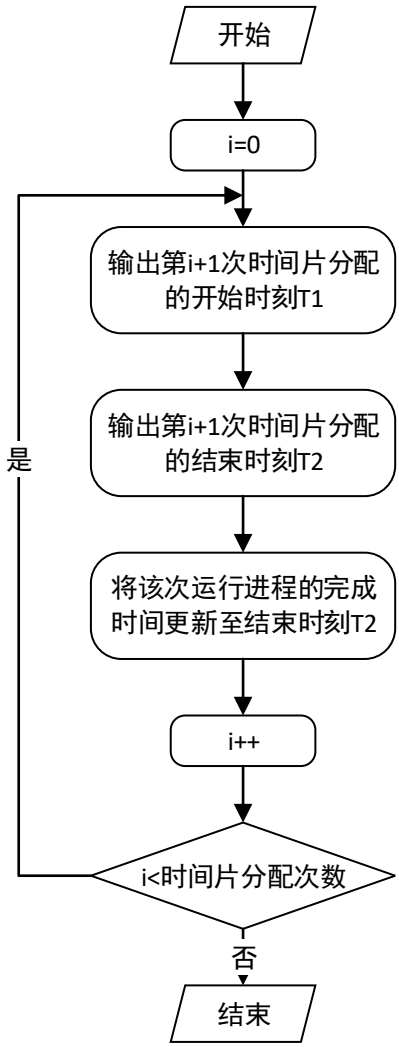

开始
初始化总进程队列
遍历进程数组
更新当前时间至
最近到达进程的
到达时间
进程入等待队列
出总进程队列
是
判断进程是否到达
判断等待队列是否为空
是
否
当前执行进程的服务时间
是否大于时间片长度
否
是
否
执行时间=当前执行
进程的服务时间
执行时间=时
间片长度q
当前执行进程
服务时间-q
保存该次过程相关信
息，并更新当前时刻
等待队列
队首进程退队
是
该进程是否执行完
否
等待队列队首
进程移至队尾
等待队列和总进程队列是否均为空
是
结束

3.3.2 模拟执行过程输出部分

（1）文字说明

算法模拟部分已经保存了每次时间片分配过程的开始时刻、结束时刻和运行进程名。将其按照次序依次输出，每次输出前在进程数组中按照进程名查找到该次运行的进程，将其完成时间更新为该次时间片分配的结束时刻。

（2）流程图说明



3.3.3 计算及结果输出部分

（1）文字说明

周转时间=完成时间-到达时间

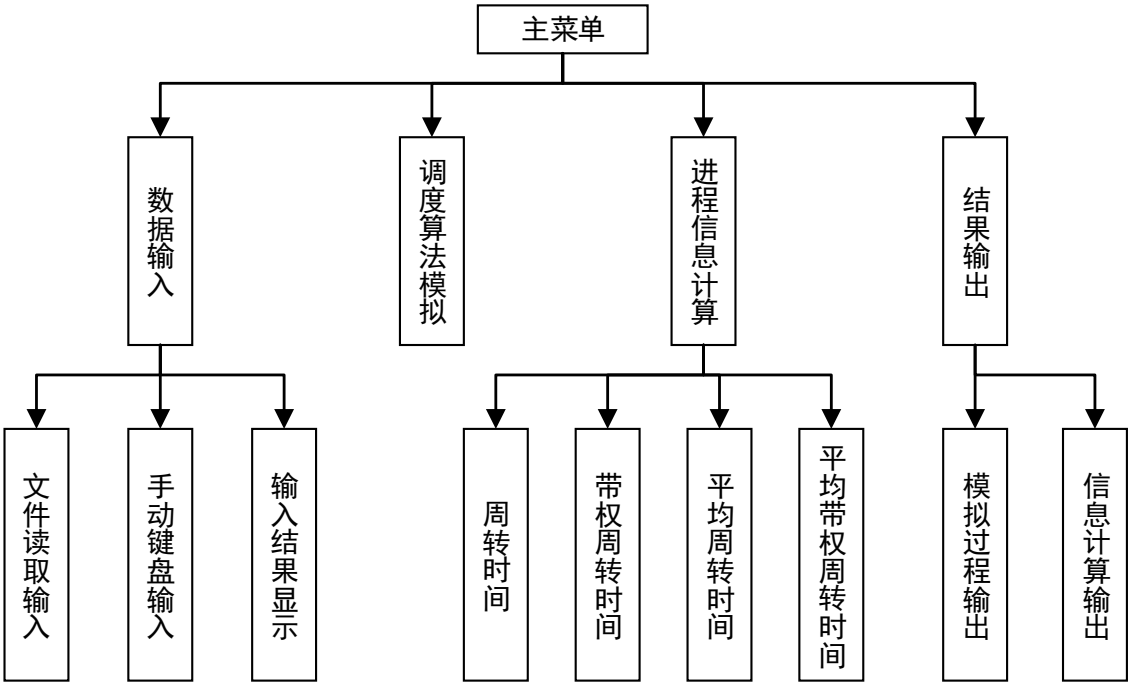
带权周转时间=周转时间/服务时间

平均周转时间=周转时间和/进程总数

平均带权周转时间=带权周转时间和/进程总数

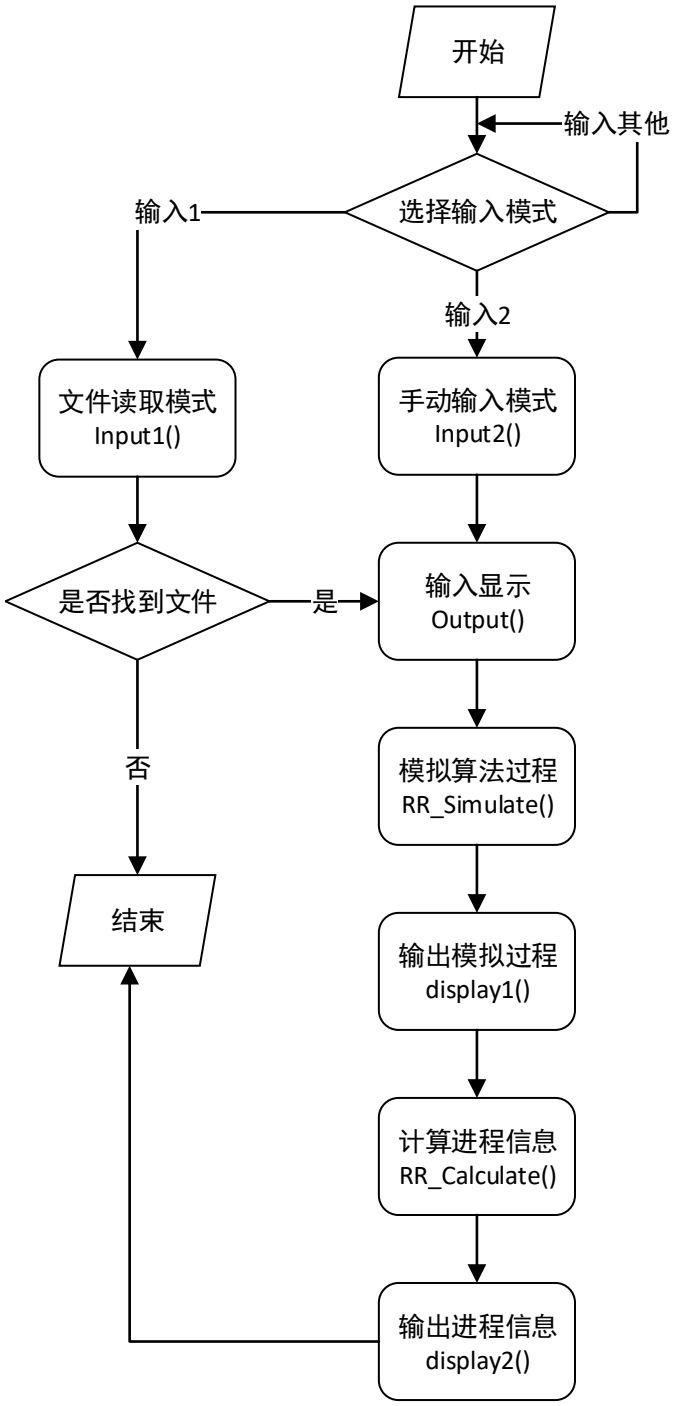
具体实现过程简单，不再赘述。

3.4 系统结构图



四、详细设计

4.1 程序流程图



4.2 主要函数核心代码

4.2.1 算法模拟计算模块

（1）模拟算法过程 RR_Simulate()函数

```
//模拟函数_模拟调度算法过程
void RR_Simulate() {
    int CurrentTime = 0;    //当前时间
```

```
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时起作用
int ncopy = n;   //存放进程总数的拷贝
int shortest = 0;   //存放到达时间最近的进程的下标
RR RRcopy[100];   //存放进程数组的拷贝
```

为方便排序，将原进程数组拷贝，以便对拷贝数组进行操作

```
//拷贝进程数组
for (int m = 0; m < n; m++) {
    RRcopy[m] = RRarray[m];
}
```

为保证程序健壮性，将拷贝数组中的进程按到达时间先后顺序进入总进程，使得用户在输入进程时可以随意输入，而不必按照到达时间顺序输入。

```
//遍历进程数组，将除到达时间最大进程以外的所有进程，按到达顺序依次进入总进程队列
while (ncopy != 1) {
    shortest = 0;
    //找到最近到达的进程下标，存入shortest
    for (int m = 0; m < ncopy - 1; m++) {
        if (RRcopy[m].ArrivalTime > RRcopy[m + 1].ArrivalTime) {
            shortest = m + 1;
        }
    }
    Pqueue.push(RRcopy[shortest]);   //将最近到达的进程进入总队列
    //将进队的进程移除出拷贝进程数组
    if (shortest == (ncopy - 1)) {
        ncopy--;
    }
    else {
        for (int m = shortest; m < ncopy - 1; m++) {
            RRcopy[m] = RRcopy[m + 1];
        }
        ncopy--;
    }
}
Pqueue.push(RRcopy[0]);   //将到达时间最大的进程进入总队列
```

下面介绍主体部分逻辑：

①首先依次检查总进程队列队首是否到达，若到达则出总进程队列进等待队列。

②然后检查两队列情况，若等待队列为空且总进程队列非空，则将CurrentTime更新至总进程队列队首进程的到达时间。

③再次执行第一步，检查并更新队列。

④确认该次时间片的分配时间tempTime。若该次过程执行进程的服务时间小于时间片长度q，则tempTime=服务时间；否则tempTime=时间片长度q。

⑤更新该次过程执行进程的服务时间，记录该次过程的各种信息。

⑥再次执行第一步，检查并更新队列。

⑦更新该次过程执行进程（即当前等待队列队首进程）的状态。若其已执行完，则退队；若其还未执行完，则将其从队首移至队尾。

```
//当等待队列为空且总进程队列为空时，跳出循环
while (!RRqueue.empty() || !Pqueue.empty()) {
    //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
    while (!Pqueue.empty() && Pqueue.front().ArrivalTime <= CurrentTime) {
        RRqueue.push(Pqueue.front());
        Pqueue.pop();
    }
    //当等待队列进程已全部执行完，但还有进程没有到达时，将当前时间更新至最快到达进程的到达时间
    if (RRqueue.empty() && !Pqueue.empty()) {
        CurrentTime = Pqueue.front().ArrivalTime;
    }
    //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
    while (!Pqueue.empty() && Pqueue.front().ArrivalTime <= CurrentTime) {
        RRqueue.push(Pqueue.front());
        Pqueue.pop();
    }

    //确认该次时间片分配的时间tempTime
    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
    processSTime[processMomentPoint] = CurrentTime;   //记录每个过程的开始时刻
    CurrentTime += tempTime;
    processMoment[processMomentPoint] = RRqueue.front().name;//记录每个过程执行的进程名
```

```
            processFTime[processMomentPoint] = CurrentTime;    //记录每个过程的结束时刻
            processMomentPoint++;

            //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
            while (!Pqueue.empty() && Pqueue.front().ArrivalTime <= CurrentTime) {
                RRqueue.push(Pqueue.front());
                Pqueue.pop();
            }

            //把执行完的进程退出队列
            if (RRqueue.front().ServiceTime <= 0) {
                RRqueue.pop();    //如果进程的服务时间小于等于0，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();
            }
        }
}
```

4.2.2 输出模块

（1）输出模拟过程 display1()函数

```
//输出函数_输出模拟执行过程
void display1() {
    int time1 = 0;    //标明取出第几次过程的开始时刻
    int time2 = 0;    //标明取出第几次过程的结束时刻
    int count = 0;
    cout << "各进程的执行时刻信息：" << endl;
    cout << "    " << processSTime[time1] << "时刻  --> " << setw(2) << processFTime[time2]
<< "时刻";    //输出第一次过程的开始时刻和结束时刻
    for (int i = 0; i < processMomentPoint; i++) {
        count = 0;
        cout << setw(3) << processMoment[i] << setw(3) << endl;

        while (RRarray[count].name != processMoment[i] && count < n) {
            count++;
        }
        RRarray[count].FinishedTime = processFTime[time2];

        if (i < processMomentPoint - 1) {
            time1++;
            time2++;
```

```
                cout << setw(3) << processSTime[time1] << "时刻" << " --> " << setw(2) <<
processFTime[time2] << "时刻" << setw(3);
            }
        }
        cout << endl;
}
```

4.2.3 主函数

```
int main() {
        cout << "请选择输入模式：" << endl;
        cout << "1.文件读取模式" << endl;
        cout << "2.用户输入模式" << endl;
        int i, j;
        j = 0;
        while (j==0) {
            cout << "请输入(1/2)：";
            cin >> i;
            cout <<
"**************************************************************************" << endl;
            switch (i) {
            case 1:
                int a;
                a = Input1();
                if (a == -1) {
                    cout << "请检查文件是否存在!" << endl;
                    system("pause");
                    return -1;
                }
                j = 1;
                break;
            case 2:
                Input2();
                j = 1;
                break;
            default:cout << "输入有误，请重新输入!" << endl;
            }
        }
        Output();
        RR_Simulate();
        display1();
        RR_Calculate();
        display2();
        system("pause");
        return 0;
}
```

五、测试数据及其结果分析

5.1 普通测试用例

5.1.1 时间片长度 q=1

（1）初始输入及结果输出

进程名	A	B	C	D	E	平均
到达时间	0	1	2	3	4	
服务时间	4	3	5	2	4	
完成时间	12	10	18	11	17	
周转时间	12	9	16	8	13	11.6
带权周转时间	3	3	3.2	4	3.25	3.29

（2）结果截图

```
请选择输入模式:
1.文件读取模式
2.用户输入模式
请输入(1/2)：1
**********************************************************************
文件读取成功!
**********************************************************************
The information of processes is the following:
processName ArrivalTime ServiceTime
         A           0           4
         B           1           3
         C           2           5
         D           3           2
         E           4           4
**********************************************************************
各进程的执行时刻信息:
  0时刻 -->  1时刻  A
  1时刻 -->  2时刻  B
  2时刻 -->  3时刻  A
  3时刻 -->  4时刻  C
  4时刻 -->  5时刻  B
  5时刻 -->  6时刻  D
  6时刻 -->  7时刻  A
  7时刻 -->  8时刻  E
  8时刻 -->  9时刻  C
  9时刻 --> 10时刻  B
 10时刻 --> 11时刻  D
 11时刻 --> 12时刻  A
 12时刻 --> 13时刻  E
 13时刻 --> 14时刻  C
 14时刻 --> 15时刻  E
 15时刻 --> 16时刻  C
 16时刻 --> 17时刻  E
 17时刻 --> 18时刻  C

**********************************************************************
RR调度算法执行后：进程相关信息如下:
进程名（ID）    到达时间    服务时间    完成时间    周转时间  带权周转时间
         A           0           4          12          12           3
         B           1           3          10           9           3
         C           2           5          18          16         3.2
         D           3           2          11           8           4
         E           4           4          17          13        3.25
所有进程的平均周转时间 = 11.6
所有进程的平均带权周转时间 = 3.29
**********************************************************************
```

5.1.2 时间片长度 q=4

（1）初始输入及结果输出

进程名	A	B	C	D	E	平均
到达时间	0	1	2	3	4	
服务时间	4	3	5	2	4	
完成时间	4	7	18	13	17	
周转时间	4	6	16	10	13	9.8
带权周转时间	1	2	3.2	5	3.25	2.89

（2）结果截图

```
请选择输入模式:
1.文件读取模式
2.用户输入模式
请输入(1/2)：1
**********************************************************************
文件读取成功!
**********************************************************************
The information of processes is the following:
processName ArrivalTime ServiceTime
         A           0           4
         B           1           3
         C           2           5
         D           3           2
         E           4           4
**********************************************************************
各进程的执行时刻信息:
  0时刻 -->  4时刻  A
  4时刻 -->  7时刻  B
  7时刻 --> 11时刻  C
 11时刻 --> 13时刻  D
 13时刻 --> 17时刻  E
 17时刻 --> 18时刻  C

**********************************************************************
RR调度算法执行后：进程相关信息如下:
进程名（ID）    到达时间    服务时间    完成时间    周转时间  带权周转时间
         A           0           4           4           4           1
         B           1           3           7           6           2
         C           2           5          18          16         3.2
         D           3           2          13          10           5
         E           4           4          17          13        3.25
所有进程的平均周转时间 = 9.8
所有进程的平均带权周转时间 = 2.89
**********************************************************************
```

5.2 极端情况测试用例

（1）初始输入及结果输出

该次输入测试一些极端情况下程序的稳定性。首先进程输入不按照到达时间顺序输入，而是随意输入；其次在前进程执行完毕之后，后一个进程还未到达。其中时间片长度 q=4。

进程名	A	B	C	D	E	平均
到达时间	20	15	10	5	0	
服务时间	4	3	5	2	4	
完成时间	24	18	15	7	4	
周转时间	4	3	5	2	4	3.6
带权周转时间	1	1	1	1	1	1

（2）结果截图

```
请选择输入模式:
1.文件读取模式
2.用户输入模式
请输入(1/2)：1
**************************************************************************
文件读取成功!
**************************************************************************
The information of processes is the following:
processName ArrivalTime ServiceTime
         A         20          4
         B         15          3
         C         10          5
         D          5          2
         E          0          4
**************************************************************************
各进程的执行时刻信息:
  0时刻 -->  4时刻  E
  5时刻 -->  7时刻  D
 10时刻 --> 14时刻  C
 14时刻 --> 15时刻  C
 15时刻 --> 18时刻  B
 20时刻 --> 24时刻  A

**************************************************************************
RR调度算法执行后：进程相关信息如下:
进程名（ID）    到达时间    服务时间    完成时间    周转时间  带权周转时间
         A         20          4          24          4          1
         B         15          3          18          3          1
         C         10          5          15          5          1
         D          5          2           7          2          1
         E          0          4           4          4          1
所有进程的平均周转时间 = 3.6
所有进程的平均带权周转时间 = 1
**************************************************************************
```

六、调试过程中的问题

【问题 1】当执行输入模块，用户没有按照进程到达顺序依次输入时，程序的等待队列出错。（已解决）

解决方法：设置总进程队列和等待队列两个队列。先将所有进程按到达时间先后顺序进入总进程队列。然后依次检查总进程队列队首进程是否到达，若到达就出总进程队列进入等待队列，直到总进程队列队首进程还未到达或总进程队列

为空为止。

【问题 2】等待队列中所有进程均执行完而下一个进程还没有到达，等待队列为空且总进程队列不为空。此时程序认为所有进程均执行完，程序结束。（已解决）

解决方法：在循环判断时，若同时满足等待队列为空、总进程队列不为空且总进程队列队首进程还未到达时，将当前时刻更新至总进程队列队首进程的到达时间，再依次检查总进程队列队首进程是否到达。

【问题 3】在一个时间片分配完毕后，本应先将到达进程进入等待队列再处理该次执行进程，但该次执行进程先一步从等待队列队首进入了队尾，使得执行顺序出错。（已解决）

解决方法：遵循每次更新 CurrentTime 就检查一遍总进程队列队首是否到达的原则，在处理当前执行进程前添加相关检查代码，使得到达进程先进队。

七、参考文献和查阅的资料

[1] 谭浩强. C 程序设计（第四版）. 清华大学出版社 ,2010.

[2] 姚琳. C++程序设计. 人民邮电出版社 ,2011.

[3] 严蔚敏/吴伟民. 数据结构（C 语言版）. 清华大学出版社 ,2007.

[4] 费祥林. 操作系统教程（第五版）. 人民邮电出版社 ,2014.

[5] 《编程之美》小组. 编程之美. 电子工业出版社 ,2008.

八、程序设计总结

经过一个星期的设计和 4 个版本的修改，“时间片轮转调度算法模拟程序”已经基本完成。在程序的开发过程中，我复习了操作系统有关进程调度的知识，亲手实现并验证了时间片轮转调度算法的调度过程，并对进程调度算法的对系统效率的影响有了更深刻的认识。

在整个设计过程中，主要工作有：

- 设计多种输入方式，使用户能使用合适的输入方式进行初始信息的输入。设计中力求交互界面友好、简洁，易于操作。
- 实现调度算法模拟、周转时间计算等要求的功能，代码部分尽量避免逻辑错误，算法设计简单合理，尽量使程序具有良好的可读性。

- 保证程序的逻辑结构，编程时注意多使用通用方法（函数和过程）。

在实现实验要求的基本功能以外，本程序极力确保程序的安全性和健壮性，在进行不同情况的测试后对暴露出的潜在 bug 反复修改，共计修改了 4 个版本，力求将程序做到使用简单且不易出错。

当然，在设计中还有很多不足的地方。比如程序中的排序过程还可以选择复杂度更低的排序算法实现，程序的输入过程还不够简洁等。在设计时，由于时间和能力有限，还有一些额外的想法没有实践，希望能在以后的版本得以实现。