



可视化模拟实现进程调度算法

学 院 计算机学院 专 业 计算机科学与技术

年级班别 17级07班

学 号 3117008753

学生姓名 吴 缺

指导教师 彭重嘉

**目录**

[**一．设计思想说明 1**](#_Toc29224924)

[**二．系统结构的说明 1**](#_Toc29224925)

[**三．数据结构的说明 1**](#_Toc29224926)

[**四．程序的主流程图以及各模块的代码 3**](#_Toc29224927)

[**五．程序运行及清单 12**](#_Toc29224928)

[**六．使用说明书 14**](#_Toc29224929)

[**七．心得体会 14**](#_Toc29224930)

[**八.参考文献 14**](#_Toc29224931)

# 一．设计思想说明

软件要求：

进程调度系统主要是记录输入进程后每个进程在每一个时间断中的状态，并且不同的调度方式把处理器分配给就绪队列中的进程，FIFO，SJF，高响应比优先，基于时间片轮转和FIFO的调度算法，需要随机输入多个进程后，可以得到调度结果。设计出界面来使进程调度方式可视化。

# 二．系统结构的说明

此系统需要对进程进行输入与调度，最后输出结果，首先需要输入进程三个信息：进程名，到达时间以及需求时间，再是选择进程调度，作业调度的方式，内存的大小。输入完后就是系统的主要功能实现，大致分为三个主要模块：

a.实现将进程移入外存，以及判断进程调度与作业调度的时机，完成主程序的框架

b.进程调度：FIFO，选出外存中到达时间最早的进程放入内存中。SJF，选出外存中需求时间最少的进程放入内存中。HRRN，选出外存中响应比最高的进程放入内存中。

c.作业调度：FIFO，执行内存中到达时间最早的进程。RR，只执行内存中队头的进程，当内存中的进程执行到一个给定的时间片后便将当前进程阻断并放到队列末尾然后再执行队头的进程。

# 三．数据结构的说明

本程序声明的类结构如下：

public class Pcb

{ /\* 定义进程控制块PCB \*/

public string name; //进程名

public string state; //状态

public int super; //优先级

public int dtime; //到达时间

public int ntime; //需要运行时间

public int stime; //剩下时间

public int rtime; //已用时间

public int ftime; //完成时刻

};

List<Pcb> pcb = new List<Pcb>();

采用C#中list泛型集合的方式将输入的每个进程的信息保存为集合，并为其添加序号，并且方便增添删改以及查询。

List<int> noa = new List<int>();//未创建序列

List<int> wca = new List<int>();//外存序列

List<int> nca = new List<int>();//内存序列

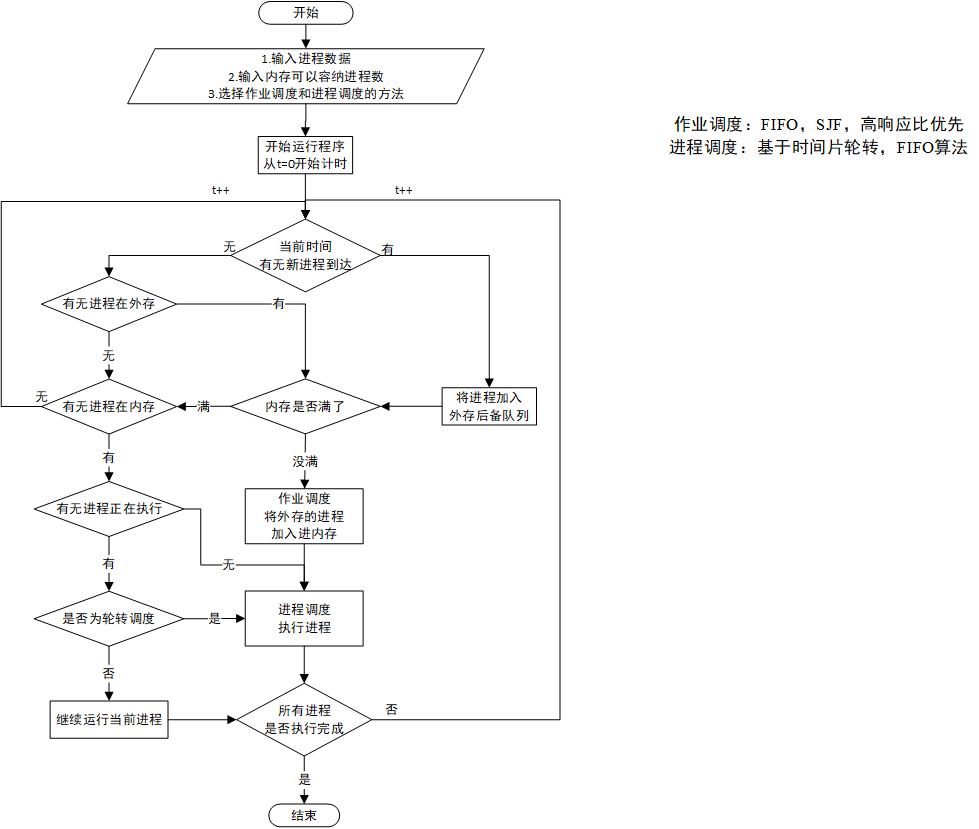
List<int> fpcb = new List<int>();//完成序列

用类似于数组的方式创建各种队列来存储进程的序号并且表明内存所在的位置。

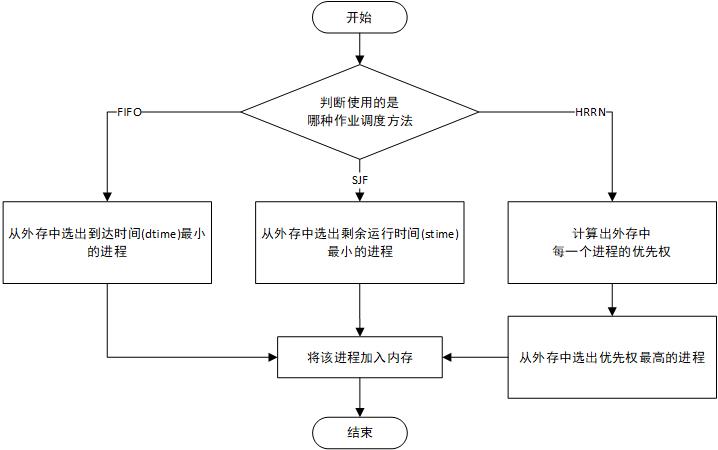


# 四．程序的主流程图以及各模块的代码

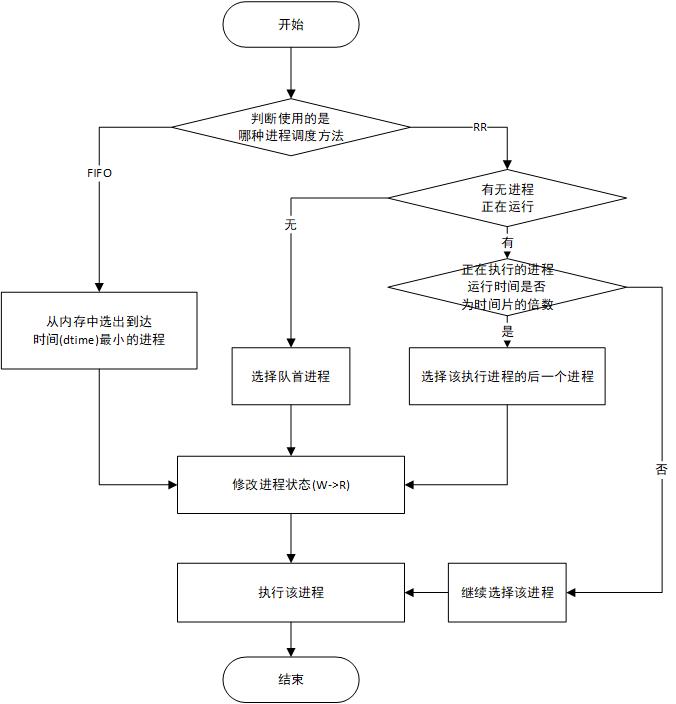
（1）主流程图如下：



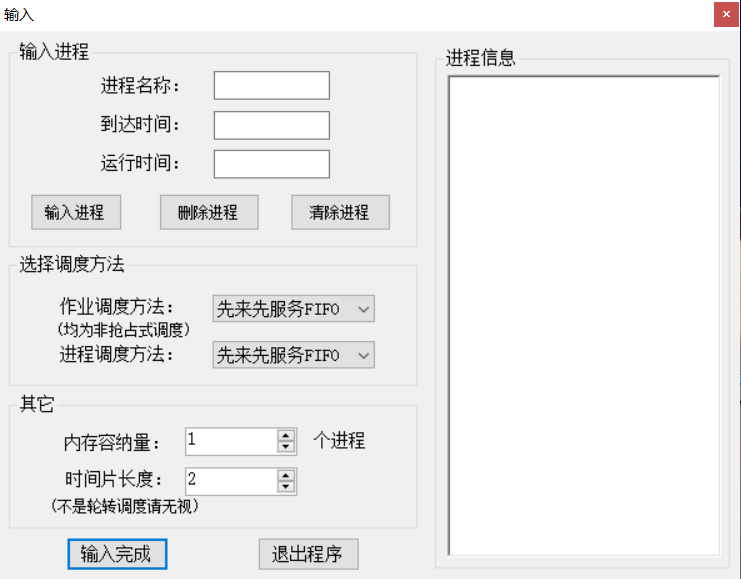
（2）作业调度流程图



（3）进程调度流程图



输入进程界面：



调度模拟界面：



调度界面代码：

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

namespace 进程调度模拟

{

public partial class RunForm : Form

{

public RunForm(string[] nname, string[] dtime, string[] ntime)

{

InitializeComponent();

NName=nname;

Dtime=dtime;

Ntime=ntime;

}

string[] NName; string[] Dtime; string[] Ntime;

int voywjd1,voywjd2,voywjd3,voywjd4,voywjd5,n;//n为进程数量

public int way\_work

{

get { return voywjd1; }

set { voywjd1 = value; }

}

public int way\_pcb

{

get { return voywjd2; }

set { voywjd2 = value; }

}

public int storagesize

{

get { return voywjd3; }

set { voywjd3 = value; }

}//内存大小

public int timelength

{

get { return voywjd4; }

set { voywjd4 = value; }

}//时间片

public int bool\_again

{

get { return voywjd5; }

set { voywjd5 = value; }

}//重来判断

int t = 0,stor;

List<Pcb> pcb = new List<Pcb>();

int irun = -1;

int n1;

private void RunForm\_Load(object sender, EventArgs e)

{

bool\_again = 0;

n = NName.Length;

n1 = n;

stor = storagesize;

switch (way\_work)

{

case 1: { label\_work.Text += "先来先服务FIFO"; break; }

case 2: { label\_work.Text += "短作业优先SJF"; break; }

case 3: { label\_work.Text += "高响应比优先"; break; }

default: { break; }

}

switch (way\_pcb)

{

case 1: { label\_pcb.Text += "先来先服务FIFO"; break; }

case 2: { label\_pcb.Text += "时间片轮转RR"; break; }

default: { break; }

}

label\_storagesize.Text += storagesize + "个进程";

if (way\_pcb == 2) label\_timelength.Text += timelength;

else label\_timelength.Text = "";

for (int pcbi = 0; pcbi < n; pcbi++)//录入进程信息

{

Pcb p = new Pcb();

p.name = NName[pcbi];

p.state = "W";

p.super = 0;

p.dtime = Convert.ToInt32(Dtime[pcbi]);

p.ntime = Convert.ToInt32(Ntime[pcbi]);

p.stime = p.ntime;

p.super = 0;

p.rtime = 0;

p.ftime = -1;

pcb.Add(p);

noa.Add(pcbi);

AllTextBox.Text += "第" + (pcbi + 1) + "个进程\n进程名称： " + pcb[pcbi].name +

"\n到达时间：" + pcb[pcbi].dtime + "\n需要时间：" + pcb[pcbi].ntime + "\n-------------------\n";

}//录入进程信息

}

public class Pcb

{ /\* 定义进程控制块PCB \*/

public string name; //进程名

public string state; //状态

public int super; //优先级

public int dtime; //到达时间

public int ntime; //需要运行时间

public int stime; //剩下时间

public int rtime; //已用时间

public int ftime; //完成时刻

}; /\* 定义进程控制块PCB \*/

List<int> noa = new List<int>();//未创建序列

List<int> wca = new List<int>();//外存序列

List<int> nca = new List<int>();//内存序列

List<int> fpcb = new List<int>();//完成序列

double Mean\_turnaround\_time = -1;//平均周转时间

double Mean\_weighted\_turnaround\_time = -1;//平均带权周转时间

private void button\_nextsecond\_Click(object sender, EventArgs e)

{

rtb\_current.Text = "";

allevent\_rtb.Text += "t=" + t + "\n";

if (irun>-1)

if (pcb[irun].stime <= 0 || pcb[irun].rtime >= pcb[irun].ntime)//进程完成判断

{

pcb[irun].state = "F";

pcb[irun].ftime = t;

nca.Remove(irun);

fpcb.Add(irun);

allevent\_rtb.Text += "进程" + pcb[irun].name + "已完成\n"

+"进程" + pcb[irun].name + "从内存中移出\n";

rtb\_current.Text += "进程" + pcb[irun].name + "已完成\n"

+ "进程" + pcb[irun].name + "从内存中移出\n";

n1--;

stor++;

irun = -1;

}//进程完成判断

if (n1 == 0)

{

if (Mean\_turnaround\_time < 0)

{

Mean\_turnaround\_time = 0;

Mean\_weighted\_turnaround\_time = 0;

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

{

Mean\_turnaround\_time += pcb[i].ftime - pcb[i].dtime;

Mean\_weighted\_turnaround\_time += Convert.ToDouble(pcb[i].ftime - pcb[i].dtime)

/ Convert.ToDouble(pcb[i].ntime);

}

Mean\_turnaround\_time /= n;

Mean\_weighted\_turnaround\_time /= n;

}//计算平均周转时间和平均带权周转时间

allevent\_rtb.Text += "所有进程均已完成\n" +

"平均周转时间=" + Mean\_turnaround\_time +

"\n平均带权周转时间=" + Mean\_weighted\_turnaround\_time+"\n";

rtb\_current.Text += "所有进程均已完成\n" +

"平均周转时间=" + Mean\_turnaround\_time +

"\n平均带权周转时间=" + Mean\_weighted\_turnaround\_time + "\n";

MessageBox.Show("所有进程均已完成\n" +

"平均周转时间=" + Mean\_turnaround\_time +

"\n平均带权周转时间=" + Mean\_weighted\_turnaround\_time);

}//判断所有进程是否全部完成

else

{

allevent\_rtb.Text += "内存可容纳" + stor + "个进程\n";

rtb\_current.Text += "内存可容纳" + stor + "个进程\n";

int bool\_nmw;

label\_time.Text = "Time=" + t;

button\_nextsecond.Text = "Time++";

bool\_nmw = noa\_moveto\_wca();

if (bool\_nmw == 1)

{//有新到进程

if (stor > 0) { job\_scheduling(); pcb\_scheduling(); }//内存未满情况，则要执行作业调度和进程调度

else if (irun == -1) { pcb\_scheduling(); }//内存满了，且无进程执行，要进行进程调度

else if (irun >= 0)

{

if (way\_pcb == 2) { pcb\_scheduling(); }

else perform\_pcb(irun);//内存满了，有进程执行，继续执行该进程

}

}//有新进程到达

else if (bool\_nmw == 0)//无新到进程

{

if (wca.Count > 0 && stor > 0) { job\_scheduling(); }//外存是否还有进程

if (nca.Count == 0)//判断有无进程在内存，无则直接结束判断，有则进行下一步判断

{

allevent\_rtb.Text += "当前无进程执行\n";

rtb\_current.Text += "当前无进程执行\n";

}

else if (irun == -1) pcb\_scheduling();//当无进程在执行时，说明可能上一时间有进程完成，进行进程调度

else if (irun >= 0)

{

if (way\_pcb == 2) { pcb\_scheduling(); }

else perform\_pcb(irun);

}

}//无新进程到达

}

array\_appear();

if(irun > -1)

{

pcb[irun].rtime++;

pcb[irun].stime--;

}

t++;

//jt();

allevent\_rtb.Text += "-----------------------------\n";

}//执行按键

private void allevent\_rtb\_TextChanged(object sender, EventArgs e)

{

allevent\_rtb.SelectionStart = allevent\_rtb.Text.Length;

allevent\_rtb.ScrollToCaret();

}//事件框信息改变时，自动下拉到最底下

private void button1\_Click(object sender, EventArgs e)

{

this.Close();

}

private void button\_return\_Click(object sender, EventArgs e)

{

bool\_again = 1;

this.Close();

}

void perform\_pcb(int ip)

{

irun = ip;

allevent\_rtb.Text += "正在执行 " + pcb[ip].name + "\n";

rtb\_current.Text += "正在执行 " + pcb[ip].name + "\n";

}//执行进程

void array\_appear()

{

rtb\_fpcb.Text = "";

rtb\_nca.Text = "";

rtb\_wca.Text = "";

rtb\_noa.Text = "";

AllTextBox.Text = "";

for (int i = 0; i < noa.Count; i++)

rtb\_noa.Text += pcb[noa[i]].name + "\n";

for (int i = 0; i < wca.Count; i++)

rtb\_wca.Text += pcb[wca[i]].name + "\n";

for (int i = 0; i < nca.Count; i++)

rtb\_nca.Text += pcb[nca[i]].name + "\n";

for (int i = 0; i < fpcb.Count; i++)

rtb\_fpcb.Text += pcb[fpcb[i]].name + "\n";

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

{

AllTextBox.Text += "第" + (i + 1) + "个进程\n进程名称：" + pcb[i].name +

"\n到达时间：" + pcb[i].dtime +

"\n需要时间：" + pcb[i].ntime +

"\n已运行时间：" + pcb[i].rtime +

"\n进程状态：" + pcb[i].state;

if (pcb[i].state == "F") AllTextBox.Text += "\n进程完成时刻：" + pcb[i].ftime;

AllTextBox.Text+="\n-------------------\n";

}

}//显示进程在哪个堆栈

int noa\_moveto\_wca()

{

int t1 = 0;//用来记录是否有新进度到达

for(int i = 0; i < noa.Count; i++)

{

if (pcb[noa[i]].dtime <= t)

{

allevent\_rtb.Text += "进程" + pcb[noa[i]].name + "移到外存后备队列\n";

rtb\_current.Text += "进程" + pcb[noa[i]].name + "移到外存后备队列\n";

wca.Add(noa[i]);

noa.RemoveAt(i);

i--;

t1 = 1;

}

}

if (t1 == 1) return 1;//有则返回1

else return 0;//没有进度到达则返回0

}//判断是否有新的进程到达并将到达的进程移入外存

void job\_scheduling()//作业调度

{

if (wca.Count <= 0) return;

int ito=0;

switch (way\_work)//选出应该调进内存的进程

{

case 1: {//FIFO

for(int i = 0; i < wca.Count; i++)//选出到达时间最小的进程

{

if (pcb[wca[ito]].dtime > pcb[wca[i]].dtime)

{

ito = i;

}

}

break;

}//FIFO

case 2: {//SJF

for (int i = 0; i < wca.Count; i++)//选出剩余时间最少的进程

{

if (pcb[wca[ito]].stime > pcb[wca[i]].stime)

{

ito = i;

}

}

break;

}//SJF

case 3: {//HNNR

for (int i = 0; i < wca.Count; i++)//选出响应比最高的进程

{

pcb[wca[i]].super = (t - pcb[wca[i]].dtime + pcb[wca[i]].stime) / pcb[wca[i]].stime;

if(pcb[wca[ito]].super == pcb[wca[i]].super)

{

if (pcb[wca[ito]].stime > pcb[wca[i]].stime)

{

ito = i;

}

}

else if (pcb[wca[ito]].super < pcb[wca[i]].super)

{

ito = i;

}

}

break;

}//HNNR

}

allevent\_rtb.Text += "进程" + pcb[wca[ito]].name + "移入内存\n";

rtb\_current.Text += "进程" + pcb[wca[ito]].name + "移入内存\n";

nca.Add(wca[ito]);

wca.RemoveAt(ito);

stor--;

if (stor > 0 && wca.Count!=0) job\_scheduling();

}//作业调度

void pcb\_scheduling()//进程调度

{

int ito=0;

if (nca.Count == 0) return;

switch (way\_pcb)//选出需要执行的进程

{

case 1: {//FIFO

for (int i = 0; i < nca.Count; i++)

{

if (pcb[nca[ito]].dtime > pcb[nca[i]].dtime)

{

ito = i;

}

}

break;

}//FIFO

case 2: {//RR

if (nca.Count < 1) return;

else if (nca.Count == 1)

{

pcb[nca[0]].state = "R";

perform\_pcb(nca[0]);

return;

}

else if (nca.Count > 1)

{

if (irun == -1)

{

pcb[nca[0]].state = "R";

perform\_pcb(nca[0]);

return; }

else if (irun > -1) {if (pcb[nca[0]].rtime % timelength == 0)

{

pcb[nca[0]].state = "W";

allevent\_rtb.Text += "进程" + pcb[irun].name + "已暂停";

rtb\_current.Text += "进程" + pcb[irun].name + "已暂停";

nca.Add(nca[0]); nca.RemoveAt(0); perform\_pcb(nca[0]); return; } }}

break;

}//RR

}//选出需要执行的进程

if (irun != nca[ito] && irun >= 0)

{

pcb[irun].state = "W";

allevent\_rtb.Text += "进程" + pcb[irun].name + "已暂停";

rtb\_current.Text += "进程" + pcb[irun].name + "已暂停";

}//将原来运行的进程停止

pcb[nca[ito]].state = "R";

perform\_pcb(nca[ito]);

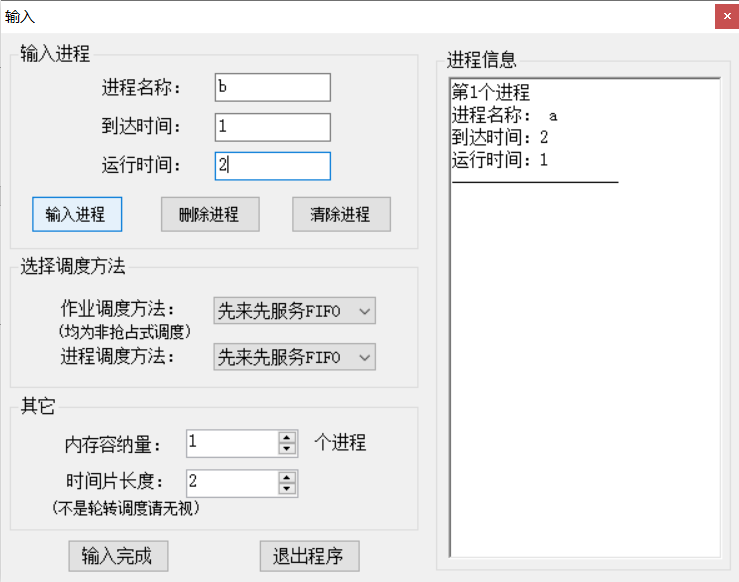
}//进程调度

}

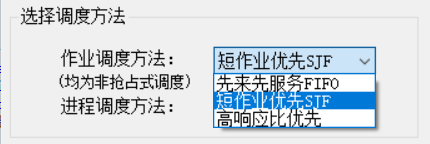
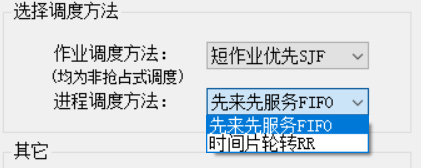
}

# 五．程序运行及清单

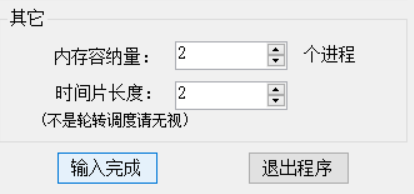
进入系统后，输入进程的属性后点击输入进程按钮：



选择调度方式：

输入内存大小，如果是RR调度则需要输入时间片长度：



点击输入完成后便进入调度模拟界面：



点击开始后便可以从0开始查看每个进程的状况以及事件：



按下Time++后查看之后每一时刻的状况：



# 六．使用说明书

（1）打开系统后进入“输入”界面输入进程名，到达时间和需要运行的时间，点击“输入进程”将进程输入，点击“删除进程”可以删除上次输入的进程，点击“清楚进程”可以删除所有进程。

（2）输入完所有进程后选择调度方法，输入内存大小，时间片大小后，点击“输入完成”便可打开调度模拟界面，点击“退出程序”可以退出本程序。

（3）输入完成后，进入调度模拟界面，只需点击“开始”按键，之后一直按“Time++”按键便可查看每一时刻的事件及进程状况。

（4）当所有进程完成后将会输出平均周转时间和平均带权周转时间。

# 七．心得体会

本次操作系统课程设计选择了进程调度模拟，因为在之前已经有设计过实验的基础上，相对其它课设的工作量少了一些，但也并不容易，进程调度的可视化需要我对进程调度的每一个模块的细节都需要理解透彻，其中最难的是对进程调度可视化的一个整体的架构，需要画好仔细的流程图，而且即使有了一个清晰的思路，编写时也会出现各种错误，编写时的错误常常是不可预的，需要不断寻找错误，修改流程图，添加细节，来完成程序。但因为时间及技术的原因只能按照要求实现课设要求的5种调度方式，比较基础。这次课设运用的是我本学期才课外学习的C#语言，与数据库的课设一起是第一次的真正实操项目。我觉得这次课设对我最大的帮助不是使我对调度方式的理解有多深，而是使我对编写C#语言与界面有了一个全新的认知，更重要的是让我对编写一个程序的逻辑严密性有了一个较大的锻炼。

# 八.参考文献

**[1]** 计算机操作系统（第四版）， 汤小丹，梁红兵等 ，西安电子科技大学出版社， 2014

**[2]** C#程序开发教程，罗福强，李瑶，人民邮电出版社出版发行，2017

**[3]** C#从入门到精通（第3版），赵洛育，清华大学出版社，2016

# 广东工业大学本科生课程设计（论文）评议表

（指导教师、评阅教师用表）

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 学院 | 计算机学院 | | | 专 业 | 计算机科学与技术 | | |
| 学号 | 3117008753 | | | 姓 名 | 吴缺 | 班 级 | 17计算机7班 |
| 课程设计（论文）题目 | | | 可视化模拟实现进程调度算法 | | | | |
| **指导教师评语**（在学生工作态度、综合运用知识能力、应用文献资料能力、设计（实验）能力、计算能力、外文应用能力、计算机应用能力、技术经济分析能力、图纸（或插图）质量、设计（论文）质量、规范化程度、应用价值及创新点等方面提出评价意见） | | | | | | | |
| 建议成绩 | |  | | | | | |