



# 《操作系统实验报告》

**学 院** 计算机学院

**专 业** 网络工程

**年级班别** 10级4班

**学 号** 3210006620

**学生姓名** 詹 妙 玲

**辅导教师** 申 建 芳

**成 绩**

2013年6月

**实验一 进程调度**

1. **实验目的**

用高级语言编写和调试一个进程调度程序，以加深对进程的概念及进程调度算法的理解。

1. **实验内容**
2. 设计一个有N个进程并发的进程调度程序。
3. 本实验采用了先来先服务以及段作业优先两种调度算法。
4. 本实验采用的是非抢占式调度，CPU分配给一个进程后，知道该进程运行完成后才释放CPU。
5. **实验运行截图**

1、主界面：



此时可以填写进程的各个参数，只有进程的各个参数都填好了以后，“提交”按钮才可用。提交完进程后，进程在就绪队列中排队：



2、选择调度算法；

当选择先来先服务算法时，进程按照提交时间先后在就绪队列中排好队，若选择短作业优先，进程按作业长度由小到大在就绪队列中排好队（如框中所示）：



1. 运行：

点击“运行”，可见位于就绪队列队首的进程被选择调度开始运行：



1. 运行完成

进程运行完成后，自动填到相应的完成记录列表中：





5、若希望用同一批数据进行模拟不同的算法分配，直接选定算法然后执行即可。若希望换一批数据进行模拟，可以点击“全部清空”，再重新输入一批数据。

1. **关键代码**

定义了一个结构体表示进程的PCB：

struct PCB{

int id;

String name;

int submit\_time;

int service\_time;

int source;

char status;

}；

选择先来先服务时将进程按提交时间先后在就绪队列中排队的代码：

void \_\_fastcall Tmianform::radio\_FCFSClick(TObject \*Sender)

{

int i,j,k;

//算出有多少个进程

for (i=0;ready[i+1].id; i++) {

;

}

//将进程按提交时间先后排序，并在就绪队列中显示

for(j=1;j<=i;j++){

for(k=1;ready[k].submit\_time!=j;k++)

{

;

}

this->ready\_queue->Cells[0][j] = ready[k].id;

this->ready\_queue->Cells[1][j] = ready[k].name;

this->ready\_queue->Cells[2][j] = ready[k].submit\_time;

this->ready\_queue->Cells[3][j] = ready[k].service\_time;

this->ready\_queue->Cells[4][j] = ready[k].source;

this->ready\_queue->Cells[5][j] = ready[k].status;

}

}

选择短进程优先时按照进程长短在就绪队列中排队的代码：

void \_\_fastcall Tmianform::radio\_SPFClick(TObject \*Sender)

{

int i,j,k,n;

PCB temp;

//算出有多少个进程

for (n=1;ready[n].id; n++) {

;

}

//用冒泡排序将进程按服务时间由短到长排序，并在就绪队列中显示

for(i=0;i<n-2;i++)

{

for(j=n-1;j>i;j--)

{

if(ready[j].service\_time<ready[j-1].service\_time)

{

temp=ready[j];

ready[j]=ready[j-1];

ready[j-1]=temp;

}

}

}

for(j=1;j<=n-1;j++)

{

this->ready\_queue->Cells[0][j] = ready[j].id;

this->ready\_queue->Cells[1][j] = ready[j].name;

this->ready\_queue->Cells[2][j] = ready[j].submit\_time;

this->ready\_queue->Cells[3][j] = ready[j].service\_time;

this->ready\_queue->Cells[4][j] = ready[j].source;

this->ready\_queue->Cells[5][j] = ready[j].status;

}

}

运行时代码：

void \_\_fastcall Tmianform::runClick(TObject \*Sender)

{

int i;

this->Label2->Caption = sum;

//运行的同时开始计时

this->Timer1->Enabled = TRUE;

//将就绪队列队首的进程移动到正在执行队列中（代表调度进CPU）

this->run\_id->Caption = this->ready\_queue->Cells[0][1];

this->run\_name->Caption = this->ready\_queue->Cells[1][1];

this->run\_service\_time->Caption = this->ready\_queue->Cells[3][1];

this->run\_source->Caption = this->ready\_queue->Cells[4][1];

this->run\_status->Caption = 'R';

this->run\_start\_time->Caption = this->Label2->Caption;

this->run\_rest\_time->Caption = this->run\_service\_time->Caption;

this->run\_finish\_time->Caption = 0;

sum--;

//就绪队列调整

for(i=2;this->ready\_queue->Cells[0][i]!="";i++)

{

this->ready\_queue->Cells[0][i-1] = this->ready\_queue->Cells[0][i];

this->ready\_queue->Cells[1][i-1] = this->ready\_queue->Cells[1][i];

this->ready\_queue->Cells[2][i-1] = this->ready\_queue->Cells[2][i];

this->ready\_queue->Cells[3][i-1] = this->ready\_queue->Cells[3][i];

this->ready\_queue->Cells[4][i-1] = this->ready\_queue->Cells[4][i];

this->ready\_queue->Cells[5][i-1] = this->ready\_queue->Cells[5][i];

}

this->ready\_queue->Cells[0][i-1] = "";

this->ready\_queue->Cells[1][i-1] = "";

this->ready\_queue->Cells[2][i-1] = "";

this->ready\_queue->Cells[3][i-1] = "";

this->ready\_queue->Cells[4][i-1] = "";

this->ready\_queue->Cells[5][i-1] = "";

}

void \_\_fastcall Tmianform::Timer1Timer(TObject \*Sender)

{

this->Label2->Caption = IntToStr(StrToInt(this->Label2->Caption)+1);

if(StrToInt(this->run\_start\_time->Caption)+StrToInt(this->run\_finish\_time->Caption) != StrToInt(this->run\_start\_time->Caption)+StrToInt(this->run\_service\_time->Caption))

{

this->run\_finish\_time->Caption = IntToStr(StrToInt(this->Label2->Caption)- StrToInt(this->run\_start\_time->Caption));

}

if(this->run\_rest\_time->Caption != 0)

{

this->run\_rest\_time->Caption = IntToStr(StrToInt(this->run\_service\_time->Caption)-StrToInt(this->run\_finish\_time->Caption));

}

if(this->run\_finish\_time->Caption == this->run\_service\_time->Caption)

{

if(this->radio\_FCFS->Checked == TRUE)

{

int i;

for(i=1; this->grid\_FCFS->Cells[0][i]!="";i++)

{

;

}

this->grid\_FCFS->Cells[0][i] = this->run\_id->Caption;

this->grid\_FCFS->Cells[1][i] = this->run\_name->Caption;

this->grid\_FCFS->Cells[2][i] = this->run\_start\_time->Caption;

this->grid\_FCFS->Cells[3][i] = this->run\_service\_time->Caption;

this->grid\_FCFS->Cells[4][i] = StrToInt(this->run\_start\_time->Caption) + StrToInt(this->run\_service\_time->Caption);

this->grid\_FCFS->Cells[5][i] = this->run\_source->Caption;

this->grid\_FCFS->Cells[6][i] = 'F';

this->grid\_FCFS->Cells[7][i] = IntToStr(StrToInt(this->grid\_FCFS->Cells[4][i]) - StrToInt(this->run\_id->Caption));

this->grid\_FCFS->Cells[8][i] = (float)(StrToInt(this->grid\_FCFS->Cells[7][i])) / (float)(StrToInt(this->run\_service\_time->Caption)) ;

}

else if(this->radio\_SPF->Checked == TRUE)

{

int i;

i = StrToInt(this->run\_id->Caption);

this->grid\_SPF->Cells[0][i] = this->run\_id->Caption;

this->grid\_SPF->Cells[1][i] = this->run\_name->Caption;

this->grid\_SPF->Cells[2][i] = this->run\_start\_time->Caption;

this->grid\_SPF->Cells[3][i] = this->run\_service\_time->Caption;

this->grid\_SPF->Cells[4][i] = StrToInt(this->run\_start\_time->Caption) + StrToInt(this->run\_service\_time->Caption);

this->grid\_SPF->Cells[5][i] = this->run\_source->Caption;

this->grid\_SPF->Cells[6][i] = 'F';

this->grid\_SPF->Cells[7][i] = IntToStr(StrToInt(this->grid\_SPF->Cells[4][i]) - StrToInt(this->run\_id->Caption));

this->grid\_SPF->Cells[8][i] = (float)(StrToInt(this->grid\_SPF->Cells[7][i])) / (float)(StrToInt(this->run\_service\_time->Caption)) ;

}

if(sum>0)

{

int i;

//将就绪队列队首的进程移动到正在执行队列中（代表调度进CPU）

this->run\_id->Caption = this->ready\_queue->Cells[0][1];

this->run\_name->Caption = this->ready\_queue->Cells[1][1];

this->run\_service\_time->Caption = this->ready\_queue->Cells[3][1];

this->run\_source->Caption = this->ready\_queue->Cells[4][1];

this->run\_status->Caption = 'R';

this->run\_start\_time->Caption = this->Label2->Caption;

this->run\_rest\_time->Caption = this->run\_service\_time->Caption;

this->run\_finish\_time->Caption = 0;

sum--;

if(sum>=0)

{

//就绪队列调整

for(i=2;this->ready\_queue->Cells[0][i]!="";i++)

{

this->ready\_queue->Cells[0][i-1] = this->ready\_queue->Cells[0][i];

this->ready\_queue->Cells[1][i-1] = this->ready\_queue->Cells[1][i];

this->ready\_queue->Cells[2][i-1] = this->ready\_queue->Cells[2][i];

this->ready\_queue->Cells[3][i-1] = this->ready\_queue->Cells[3][i];

this->ready\_queue->Cells[4][i-1] = this->ready\_queue->Cells[4][i];

this->ready\_queue->Cells[5][i-1] = this->ready\_queue->Cells[5][i];

}

this->ready\_queue->Cells[0][i-1] = "";

this->ready\_queue->Cells[1][i-1] = "";

this->ready\_queue->Cells[2][i-1] = "";

this->ready\_queue->Cells[3][i-1] = "";

this->ready\_queue->Cells[4][i-1] = "";

this->ready\_queue->Cells[5][i-1] = "";

}

else

{

;

}

}

else

{

this->Timer1->Enabled = FALSE;

this->Label2->Caption = 0;

this->run\_id->Caption = "";

this->run\_name->Caption = "";

this->run\_rest\_time->Caption = "";

this->run\_service\_time->Caption = "";

this->run\_source->Caption = "";

this->run\_start\_time->Caption = "";

this->run\_status->Caption = "";

this->run\_finish\_time->Caption = "";

if(MessageBox(NULL,L"进程已运行完成",L"消息",MB\_OK))

{

int q ;

for(q=1;ready[q].id != '\0';q++)

{

this->ready\_queue->Cells[0][q] = ready[q].id;

this->ready\_queue->Cells[1][q] = ready[q].name;

this->ready\_queue->Cells[2][q] = ready[q].submit\_time;

this->ready\_queue->Cells[3][q] = ready[q].service\_time;

this->ready\_queue->Cells[4][q] = ready[q].source;

this->ready\_queue->Cells[5][q] = ready[q].status;

}

sum = q-1;

}

}

}

}

**实验二 作业调度**

1. **实验目的**

用高级语言编写和调试一个或多个作业调度的模拟程序，了解作业调度在操作系统中的作用。

1. **实验内容**

模拟作业调度程序，分别采用时间片轮转法和静态优先权法。

1. **实验运行截图**

1、主界面



此时可以填写作业的各个参数，只有作业的各个参数都填好了以后，“提交”按钮才可用。提交完作业后，作业在就绪队列中排队：



2、选择调度算法；

当选择时间片轮转法时，进程按照提交时间先后在就绪队列中排好队，若选择优先级调度算法，进程按作优先级由小到大在就绪队列中排好队（如框中所示）：



1. 运行：

点击“运行”，可见位于就绪队列队首的进程被选择调度开始运行：



1. 运行完成

进程运行完成后，自动填到相应的完成记录列表中：





5、若希望用同一批数据进行模拟不同的算法分配，直接选定算法然后执行即可。若希望换一批数据进行模拟，可以点击“全部清空”，再重新输入一批数据。

1. **实验关键代码**

定义了一个结构体来表示PCB：

struct PCB{

int id;

String name;

int start\_time;

int service\_time;

int finish\_time;

int end\_time;

int rest\_time;

int submit\_time;

int degree;

char status;

} ready[50];

时间片轮转法代码：

if(this->radio\_Timeround->Checked)

{

static int serv\_time = StrToInt(this->ComboBox\_timesize->Text);

static int count\_time = 0;

if(StrToInt(this->run\_rest\_time->Caption) > 0)

{

this->run\_finish\_time->Caption = StrToInt(this->run\_finish\_time->Caption)+1;

this->run\_rest\_time->Caption = StrToInt(this->run\_rest\_time->Caption)-1;

}

count\_time++;

if(count\_time != serv\_time)

{

if(this->run\_rest\_time->Caption == "0")

this->ready[StrToInt(this->run\_id->Caption)].end\_time = StrToInt(this->Label2->Caption);

}

else if(count\_time == serv\_time)

{

if(StrToInt(this->run\_rest\_time->Caption) != 0)

{

if(this->ready\_queue->Cells[0][1]!="")

{

//把服务从新排到服务队列的队尾

this->ready\_queue->Cells[0][sum+1] = this->run\_id->Caption;

this->ready\_queue->Cells[1][sum+1] = this->run\_name->Caption;

this->ready\_queue->Cells[2][sum+1] = this->run\_id->Caption;

this->ready\_queue->Cells[3][sum+1] = this->run\_service\_time->Caption;

this->ready\_queue->Cells[4][sum+1] = this->run\_source->Caption;

this->ready\_queue->Cells[5][sum+1] = 'W';

this->ready[StrToInt(this->run\_id->Caption)].finish\_time = StrToInt(this->run\_finish\_time->Caption);

this->ready[StrToInt(this->run\_id->Caption)].rest\_time = StrToInt(this->run\_rest\_time->Caption);

sum++;

}

}

else if(StrToInt(this->run\_rest\_time->Caption) == 0)

{ //还需运行时间为0时

//记录完毕时间

this->ready[StrToInt(this->run\_id->Caption)].end\_time = StrToInt(this->Label2->Caption);

//将正在运行作业放到已完成作业表中

int n = 1;

while(this->grid\_Timeround->Cells[0][n] != "") n++;

this->grid\_Timeround->Cells[0][n] = this->run\_id->Caption;

this->grid\_Timeround->Cells[1][n] = this->run\_name->Caption;

this->grid\_Timeround->Cells[2][n] = this->run\_start\_time->Caption;

this->grid\_Timeround->Cells[3][n] = this->run\_service\_time->Caption;

this->grid\_Timeround->Cells[4][n] = this->ready[StrToInt(this->run\_id->Caption)].end\_time;

this->grid\_Timeround->Cells[5][n] = this->run\_source->Caption;

this->grid\_Timeround->Cells[6][n] = 'F';

this->grid\_Timeround->Cells[7][n] = ready[StrToInt(this->run\_id->Caption)].end\_time-StrToInt(this->run\_id->Caption);

this->grid\_Timeround->Cells[8][n] = (float)(StrToInt(this->grid\_Timeround->Cells[7][n])) / (float)(StrToInt(this->run\_service\_time->Caption)) ;

this->run\_id->Caption = "";

}

if(this->ready\_queue->Cells[0][1]!="")

{

//服务首次进入正在执行队列时对开始时间进行赋值

if(ready[StrToInt(this->ready\_queue->Cells[0][1])].finish\_time == 0)

ready[StrToInt(this->ready\_queue->Cells[0][1])].start\_time = StrToInt(this->Label2->Caption);

//把当前队列首个服务进入正在执行队列

this->run\_id->Caption = this->ready\_queue->Cells[0][1];

this->run\_name->Caption = this->ready\_queue->Cells[1][1];

this->run\_start\_time->Caption = this->ready[StrToInt(this->ready\_queue->Cells[0][1])].start\_time;

this->run\_finish\_time->Caption = this->ready[StrToInt(this->ready\_queue->Cells[0][1])].finish\_time;

this->run\_rest\_time->Caption = this->ready[StrToInt(this->ready\_queue->Cells[0][1])].rest\_time;

this->run\_source->Caption = this->ready\_queue->Cells[4][1];

this->run\_status->Caption = 'W';

this->run\_service\_time->Caption = this->ready[StrToInt(this->ready\_queue->Cells[0][1])].service\_time;

sum--;

//就绪队列调整

int i;

for(i=2;this->ready\_queue->Cells[0][i]!="";i++)

{

this->ready\_queue->Cells[0][i-1] = this->ready\_queue->Cells[0][i];

this->ready\_queue->Cells[1][i-1] = this->ready\_queue->Cells[1][i];

this->ready\_queue->Cells[2][i-1] = this->ready\_queue->Cells[2][i];

this->ready\_queue->Cells[3][i-1] = this->ready\_queue->Cells[3][i];

this->ready\_queue->Cells[4][i-1] = this->ready\_queue->Cells[4][i];

this->ready\_queue->Cells[5][i-1] = this->ready\_queue->Cells[5][i];

}

this->ready\_queue->Cells[0][i-1] = "";

this->ready\_queue->Cells[1][i-1] = "";

this->ready\_queue->Cells[2][i-1] = "";

this->ready\_queue->Cells[3][i-1] = "";

this->ready\_queue->Cells[4][i-1] = "";

this->ready\_queue->Cells[5][i-1] = "";

}

else

{

this->run\_finish\_time->Caption = StrToInt(this->run\_finish\_time->Caption)+1;

this->run\_rest\_time->Caption = StrToInt(this->run\_rest\_time->Caption)-1;

}

//总数sum为0时停止计时

if(this->run\_id->Caption == "")

{

this->Timer1->Enabled = FALSE;

this->Label2->Caption = 0;

this->run\_id->Caption = "";

this->run\_name->Caption = "";

this->run\_rest\_time->Caption = "";

this->run\_service\_time->Caption = "";

this->run\_source->Caption = "";

this->run\_start\_time->Caption = "";

this->run\_status->Caption = "";

this->run\_finish\_time->Caption = "";

if(MessageBox(NULL,L"进程已运行完成",L"消息",MB\_OK))

{

int q ;

for(q=1;ready[q].id != '\0';q++)

{

//重置数据记录时间

this->ready[q].start\_time = 0;

this->ready[q].finish\_time = 0;

this->ready[q].rest\_time = ready[q].service\_time;

this->ready[q].end\_time = 0;

//重置等候列表

this->ready\_queue->Cells[0][q] = ready[q].id;

this->ready\_queue->Cells[1][q] = ready[q].name;

this->ready\_queue->Cells[2][q] = ready[q].submit\_time;

this->ready\_queue->Cells[3][q] = ready[q].service\_time;

this->ready\_queue->Cells[4][q] = ready[q].degree;

this->ready\_queue->Cells[5][q] = ready[q].status;

}

sum = q-1;

}

}

count\_time = 0;

}

}

静态优先权调度算法代码：

else if(this->radio\_Staticrank->Checked)

{

if(StrToInt(this->run\_start\_time->Caption)+StrToInt(this->run\_finish\_time->Caption) != StrToInt(this->run\_start\_time->Caption)+StrToInt(this->run\_service\_time->Caption))

{

this->run\_finish\_time->Caption = IntToStr(StrToInt(this->Label2->Caption)- StrToInt(this->run\_start\_time->Caption));

}

if(this->run\_rest\_time->Caption != 0)

{

this->run\_rest\_time->Caption = IntToStr(StrToInt(this->run\_service\_time->Caption)-StrToInt(this->run\_finish\_time->Caption));

}

if(this->run\_finish\_time->Caption == this->run\_service\_time->Caption)

{

if(this->radio\_Timeround->Checked == TRUE)

{

int i;

for(i=1; this->grid\_Timeround->Cells[0][i]!="";i++)

{

;

}

this->grid\_Timeround->Cells[0][i] = this->run\_id->Caption;

this->grid\_Timeround->Cells[1][i] = this->run\_name->Caption;

this->grid\_Timeround->Cells[2][i] = this->run\_start\_time->Caption;

this->grid\_Timeround->Cells[3][i] = this->run\_service\_time->Caption;

this->grid\_Timeround->Cells[4][i] = StrToInt(this->run\_start\_time->Caption) + StrToInt(this->run\_service\_time->Caption);

this->grid\_Timeround->Cells[5][i] = this->run\_source->Caption;

this->grid\_Timeround->Cells[6][i] = 'F';

this->grid\_Timeround->Cells[7][i] = IntToStr(StrToInt(this->grid\_Timeround->Cells[4][i]) - StrToInt(this->run\_id->Caption));

this->grid\_Timeround->Cells[8][i] = (float)(StrToInt(this->grid\_Timeround->Cells[7][i])) / (float)(StrToInt(this->run\_service\_time->Caption)) ;

}

else if(this->radio\_Staticrank->Checked == TRUE)

{

int i;

i = StrToInt(this->run\_id->Caption);

this->grid\_Staticrank->Cells[0][i] = this->run\_id->Caption;

this->grid\_Staticrank->Cells[1][i] = this->run\_name->Caption;

this->grid\_Staticrank->Cells[2][i] = this->run\_start\_time->Caption;

this->grid\_Staticrank->Cells[3][i] = this->run\_service\_time->Caption;

this->grid\_Staticrank->Cells[4][i] = StrToInt(this->run\_start\_time->Caption) + StrToInt(this->run\_service\_time->Caption);

this->grid\_Staticrank->Cells[5][i] = this->run\_source->Caption;

this->grid\_Staticrank->Cells[6][i] = 'F';

this->grid\_Staticrank->Cells[7][i] = IntToStr(StrToInt(this->grid\_Staticrank->Cells[4][i]) - StrToInt(this->run\_id->Caption));

this->grid\_Staticrank->Cells[8][i] = (float)(StrToInt(this->grid\_Staticrank->Cells[7][i])) / (float)(StrToInt(this->run\_service\_time->Caption)) ;

}

if(sum>0)

{

int i;

//将就绪队列队首的进程移动到正在执行队列中（代表调度进CPU）

this->run\_id->Caption = this->ready\_queue->Cells[0][1];

this->run\_name->Caption = this->ready\_queue->Cells[1][1];

this->run\_service\_time->Caption = this->ready\_queue->Cells[3][1];

this->run\_source->Caption = this->ready\_queue->Cells[4][1];

this->run\_status->Caption = 'R';

this->run\_start\_time->Caption = this->Label2->Caption;

this->run\_rest\_time->Caption = this->run\_service\_time->Caption;

this->run\_finish\_time->Caption = 0;

sum--;

if(sum>=0)

{

//就绪队列调整

for(i=2;this->ready\_queue->Cells[0][i]!="";i++)

{

this->ready\_queue->Cells[0][i-1] = this->ready\_queue->Cells[0][i];

this->ready\_queue->Cells[1][i-1] = this->ready\_queue->Cells[1][i];

this->ready\_queue->Cells[2][i-1] = this->ready\_queue->Cells[2][i];

this->ready\_queue->Cells[3][i-1] = this->ready\_queue->Cells[3][i];

this->ready\_queue->Cells[4][i-1] = this->ready\_queue->Cells[4][i];

this->ready\_queue->Cells[5][i-1] = this->ready\_queue->Cells[5][i];

}

this->ready\_queue->Cells[0][i-1] = "";

this->ready\_queue->Cells[1][i-1] = "";

this->ready\_queue->Cells[2][i-1] = "";

this->ready\_queue->Cells[3][i-1] = "";

this->ready\_queue->Cells[4][i-1] = "";

this->ready\_queue->Cells[5][i-1] = "";

}

else

{

;

}

}

else

{

this->Timer1->Enabled = FALSE;

this->Label2->Caption = 0;

this->run\_id->Caption = "";

this->run\_name->Caption = "";

this->run\_rest\_time->Caption = "";

this->run\_service\_time->Caption = "";

this->run\_source->Caption = "";

this->run\_start\_time->Caption = "";

this->run\_status->Caption = "";

this->run\_finish\_time->Caption = "";

if(MessageBox(NULL,L"进程已运行完成",L"消息",MB\_OK))

{

int q ;

for(q=1;ready[q].id != '\0';q++)

{

this->ready\_queue->Cells[0][q] = ready[q].id;

this->ready\_queue->Cells[1][q] = ready[q].name;

this->ready\_queue->Cells[2][q] = ready[q].submit\_time;

this->ready\_queue->Cells[3][q] = ready[q].service\_time;

this->ready\_queue->Cells[4][q] = ready[q].degree;

this->ready\_queue->Cells[5][q] = ready[q].status;

}

sum = q-1;

}

}

}

}

**实验三 存储管理**

1. **实验目的**

通过编写和调试存储管理的模拟程序以加深对存储管理方案的理解。熟悉虚存管理的各种页面淘汰算法。

1. **实验内容**

设计一个有固定式分区分配的存储管理方案，模拟实现分区的分配与回收过程。

本实验采用了首次适应算法、循环首次适应算法、最佳适应算法三种算法。

1. **实验运行截图**

1、主界面：



此时可以随机生成一批作业和空闲分区（只有在生成了分区和作业以及选定了算法后，“开始分配”按钮才可用）：



2、选择算法并运行：

当选择最佳适应算法时，空闲分区会在每次分配过后重新按照剩余空间由小到大排好序：



3、回收资源：

点击“回收全部资源”，将分配出去的空间收回，此时，被回收资源的作业状态改为”已完成”。



4、二次分配：

回收完资源后，剩余未被分配的作业就可以进行分配了：



可见，剩余一个作业未被分配，由于没有空闲分区可以容纳下这个作业。

5、若希望用同一批数据进行模拟不同的算法分配，直接选定算法然后执行即可。若希望换一批数据进行模拟，可以点击清空，再进行一次随机生成数据操作。

1. **关键代码**

实现空闲分区分配的代码：

void \_\_fastcall Tmianform::Button3Click(TObject \*Sender)

{

this->Button3->Enabled = FALSE;

int i,j,k,temp=1;

if(this->RadioButton1->Checked == TRUE)

{

for(i=1;i<=10;i++)

{

for(j=1;(j<=10)&&(StrToInt(this->StringGrid\_memory->Cells[2][j]) < rand3[i]);j++)

{

;

}

if(j>10)

{

this->StringGrid\_work->Cells[2][i] = "未分配";

this->StringGrid\_work->Cells[3][i] = "N";

}

else if(StrToInt(this->StringGrid\_memory->Cells[2][j]) >= rand3[i])

{

this->StringGrid\_memory->Cells[2][j] = StrToInt(this->StringGrid\_memory->Cells[2][j]) - rand3[i];

this->StringGrid\_memory->Cells[4][j] = this->StringGrid\_memory->Cells[4][j]+"|"+i;

this->StringGrid\_work->Cells[2][i] = j;

this->StringGrid\_work->Cells[3][i] = "Y";

}

}

}

else if(this->RadioButton2->Checked == TRUE)

{ int sum;

for(i=1;i<=10;i++)

{

for(j=temp,sum=1;sum<10;j++)

{

k=j%10;

if(k==0) k=k+10;

sum++;

if(StrToInt(this->StringGrid\_memory->Cells[2][k]) >= rand3[i])

{

break;

}

}

if(sum==10)

{

this->StringGrid\_work->Cells[2][i] = "未分配";

this->StringGrid\_work->Cells[3][i] = "N";

}

else if(StrToInt(this->StringGrid\_memory->Cells[2][k])>=rand3[i])

{

temp=k+1;

this->StringGrid\_memory->Cells[2][k] = StrToInt(this->StringGrid\_memory->Cells[2][k]) - rand3[i];

this->StringGrid\_memory->Cells[4][k] = this->StringGrid\_memory->Cells[4][k]+"|"+i;

this->StringGrid\_work->Cells[2][i] = k;

this->StringGrid\_work->Cells[3][i] = "Y";

}

}

}

else if(this->RadioButton3->Checked == TRUE)

{

for(i=1;i<=10;i++)

{

for(j=1;j<=10&&memory[j].rest\_size<rand3[i];j++)

{

;

}

if(j>10)

{

this->StringGrid\_work->Cells[2][i] = "未分配";

this->StringGrid\_work->Cells[3][i] = "N"; ;

}

else if(memory[j].rest\_size>=rand3[i])

{

this->StringGrid\_memory->Cells[2][j] = StrToInt(this->StringGrid\_memory->Cells[2][j]) - rand3[i];

this->StringGrid\_memory->Cells[4][j] = this->StringGrid\_memory->Cells[4][j]+"|"+i;

this->memory[j].rest\_size = StrToInt(this->StringGrid\_memory->Cells[2][j]);

this->memory[j].work = this->StringGrid\_memory->Cells[4][j];

this->StringGrid\_work->Cells[2][i] = this->memory[j].id;

this->StringGrid\_work->Cells[3][i] = "Y";

bubblesort(memory,10);

for(int z=1;z<=10;z++)

{

this->StringGrid\_memory->Cells[0][z] = memory[z].id;

this->StringGrid\_memory->Cells[1][z] = memory[z].original\_size;

this->StringGrid\_memory->Cells[2][z] = memory[z].rest\_size;

this->StringGrid\_memory->Cells[3][z] = memory[z].head\_address;

this->StringGrid\_memory->Cells[4][z] = memory[z].work;

}

}

}

}

else

{

MessageBox(NULL,L"请选择算法",L"Tips",MB\_OK);

}

}

回收资源代码：

void \_\_fastcall Tmianform::Button5Click(TObject \*Sender)

{

this->Button4->Enabled = TRUE;

this->Button3->Enabled = TRUE;

int i,j,k;

for(j=1;j<=10;j++)

{

this->StringGrid\_memory->Cells[0][j] = j;

this->StringGrid\_memory->Cells[1][j] = rand1[j];

this->StringGrid\_memory->Cells[2][j] = rand1[j];

this->StringGrid\_memory->Cells[3][j] = rand2[j];

this->StringGrid\_memory->Cells[4][j] = "";

this->memory[j].id = j;

this->memory[j].original\_size = rand1[j];

this->memory[j].rest\_size = rand1[j];

this->memory[j].head\_address = rand2[j];

this->memory[j].work = this->StringGrid\_work->Cells[4][j];

}

for(k=1;k<=10;k++)

{

this->StringGrid\_work->Cells[0][k] = k;

this->StringGrid\_work->Cells[1][k] = rand3[k];

this->StringGrid\_work->Cells[2][k] = "";

if(this->StringGrid\_work->Cells[3][k] == "Y")

{

this->StringGrid\_work->Cells[3][k] = "已完成";

}

else

;

}

}

随机生成空闲分区代码：

void \_\_fastcall Tmianform::Button1Click(TObject \*Sender)

{

this->Button5->Enabled = TRUE;

int x=1,y=100,i,j;

srand((unsigned)time(NULL));

for(i=1;i<=10;i++)

{

rand1[i] = rand()%(y-x+1)+x;

rand2[i] = rand()%(y-x+1)+x;

}

for(j=1;j<=10;j++)

{

this->StringGrid\_memory->Cells[0][j] = j;

this->StringGrid\_memory->Cells[1][j] = rand1[j];

this->StringGrid\_memory->Cells[2][j] = rand1[j];

this->StringGrid\_memory->Cells[3][j] = rand2[j];

this->memory[j].id = j;

this->memory[j].original\_size = rand1[j];

this->memory[j].rest\_size = rand1[j];

this->memory[j].head\_address = rand2[j];

this->memory[j].work = this->StringGrid\_work->Cells[4][j];

}

}