

XI`AN TECHNOLOGICAL UNIVERSITY

实验报告

实验课程名称 计算机操作系统

专 业： 物联网工程

班 级： 16060616

姓 名： 田宇龙

学 号： 16060616107

实验学时：

指导教师：

成 绩：

年 月 日

西安工业大学实验报告

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 专业 | 物联网工程 | 班级 | 16 | 姓名 | 田宇龙 | 学号 | 16060616107 |
| 实验课程 | 计算机操作系统 | 指导教师 |  | 实验日期 |  | 同实验者 |  |
| 实验项目 |  | | | | | | |
| 实验设备及器材 |  | | | | | | |

1. **实验目的**

掌握动态分区分配的四种常用方法

利用代码实现四种分配方法

①首次适应算法

②循环首次适应算法

③最坏适应算法

④最佳适应算法

1. **实验环境**

计算机一台，编译环境 vs2013

1. **实现思想**

（1）首次适应算法

在首次适应算法中，是从已建立好的数组中顺序查找，直至找到第一个大小能满足要求的空闲分区为止，然后再按照作业大小，从该分区中划出一块内存空间分配给请求者，余下的空间令开辟一块新的地址，大小为原来的大小减去作业大小，若查找结束都不能找到一个满足要求的分区，则此次内存分配失败。

（2）循环首次适应算法

该算法是由首次适应算法演变而成，在为进程分配内存空间时，不再是每次都从第一个空间开始查找，而是从上次找到的空闲分区的下一个空闲分区开始查找，直至找到第一个能满足要求的空闲分区，从中划出一块与请求大小相等的内存空间分配给作业，为实现本算法，设置一个全局变量f，来控制循环查找，当f%N==0时，f=0；若查找结束都不能找到一个满足要求的分区，则此次内存分配失败。

（3）最坏适应算法

最坏适应分配算法是每次为作业分配内存时，扫描整个数组，总是把能满足条件的，又是最大的空闲分区分配给作业。

（4）最佳适应算法

最坏适应分配算法是每次为作业分配内存时，扫描整个数组，总是把能满足条件的，又是最小的空闲分区分配给作业。

1. **代码实现**

**结构体定义**：

typedef struct LNode

{

int startaddress; //初始地址

int size; //分区大小

int state; //分区状态

} LNode;

**全局变量**

LNode P[L] = { { 0, 128, 0 }, { 200, 256, 0 }, { 500, 512, 0 }, { 1500, 1600, 0 }, { 5000, 150, 0 } };

int N = 5; int f = 0;

**枚举：**

enum {

First\_fit=1,

CirFirst\_fit,

Worst\_fit,

Best\_fit,

End = 0,

};

**主菜单**

int Menu(){ //菜单选项

printf("-----菜单-----\n");

printf("1、首次适应算法\n");

printf("2、循环首次适应算法\n");

printf("3、最坏适应算法\n");

printf("4、最佳适应算法\n");

printf("0、退出\n");

printf("请输入选择：");

int choice = 0;

scanf("%d", &choice);

return choice;

}

**状态打印**

void print()

{

int i;

printf("起始地址 分区 状态\n");

for (i = 0; i<N; i++)

printf("%3d %8d %4d\n", P[i].startaddress, P[i].size, P[i].state);

}

**首次适应分配算法**

void First() {

int i, l = 0, m;

printf("\n输入请求分配分区的大小：");

scanf("%d", &m);

for (i = 0; i<N; i++)

{

if (P[i].size<m)

continue;

else if (P[i].size == m)

{

P[i].state = 1;

l = 1;

break;

}

else

{

P[N].startaddress = P[i].startaddress + m;

P[N].size = P[i].size - m;

P[i].size = m; P[i].state = 1;

l = 1; N++;

break;

}

}

if (l == 1 || i<N)

{

printf("地址成功分配\n\n");

printf("地址分配成功后的状态：\n");

print();

}

else

printf("没有可以分配的地址空间\n");

}

**循环首次适应算法**

void CirFirst(){

int l = 0, m, t = 0;

printf("\n输入请求分配分区的大小：");

scanf("%d", &m);

while (f<N)

{

if (P[f].size<m)

{

f = f + 1;

if (f%N == 0)

{

f = 0; t = 1;

}

continue;

}

if (P[f].size == m && P[f].state != 1)

{

P[f].state = 1;

l = 1; f++;

break;

}

if (P[f].size>m && P[f].state != 1)

{

P[N].startaddress = P[f].startaddress + m;

P[N].size = P[f].size - m;

P[f].size = m;

P[f].state = 1;

l = 1; N++;

f++; break;

}

}

if (l == 1)

{

printf("地址成功分配\n\n");

printf("地址分配成功后的状态：\n");

print();

}

else

printf("没有可以分配的地址空间\n");

}

**最坏适应算法**

void Worst(){

int i, t = 0, l = 0, m;

int a[L];

printf("\n输入请求分配分区的大小：");

scanf("%d", &m);

for (i = 0; i<N; i++)

{

a[i] = 0;

if (P[i].size<m)

continue;

else if (P[i].size == m)

{

P[i].state = 1;

l = 1; break;

}

else

a[i] = P[i].size - m;

}

if (l == 0)

{

for (i = 0; i<N; i++)

{

if (a[i] != 0)

t = i;

}

for (i = 0; i<N; i++)

{

if (a[i] != 0 && a[i]>a[t])

t = i;

}

P[N].startaddress = P[t].startaddress + m;

P[N].size = P[t].size - m;

P[t].size = m; P[t].state = 1;

l = 1; N++;

}

if (l == 1 || i<N)

{

printf("地址成功分配\n\n");

printf("地址分配成功后的状态：\n");

print();

}

else

printf("没有可以分配的地址空间\n");

}

**最佳适应算法**

void Best(){

int i, t = 0, l = 0, m;

int a[L];

printf("\n输入请求分配分区的大小：");

scanf("%d", &m);

for (i = 0; i<N; i++)

{

a[i] = 0;

if (P[i].size<m)

continue;

else if (P[i].size == m)

{

P[i].state = 1;

l = 1;

break;

}

else

a[i] = P[i].size - m;

}

if (l == 0)

{

for (i = 0; i<N; i++)

{

if (a[i] != 0)

t = i;

}

for (i = 0; i<N; i++)

{

if (a[i] != 0 && a[i]<a[t])

t = i;

}

P[N].startaddress = P[t].startaddress + m;

P[N].size = P[t].size - m;

P[t].size = m; P[t].state = 1;

l = 1; N++;

}

if (l == 1 || i<N)

{

printf("地址成功分配\n\n");

printf("地址分配成功后的状态：\n");

print();

}

else

printf("没有可以分配的地址空间\n");

}

**主函数**

int main()

{

while (1)

{

int choice = Menu();

switch (choice)

{

case First\_fit:

printf("\n初始状态为：\n");

print();

First();

break;

case CirFirst\_fit:

printf("\n初始状态为：\n");

print();

CirFirst();

break;

case Worst\_fit:

printf("\n初始状态为：\n");

print();

Worst();

break;

case Best\_fit:

printf("\n初始状态为：\n");

print();

Best();

break;

case End:

printf("Goodbye!\n");

system("pause");

return 0;

default:

break;

}

}

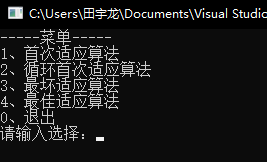
system("pause");

return 0;

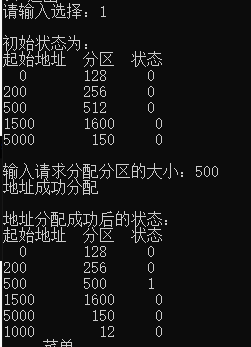
}

1. **程序运行截图**

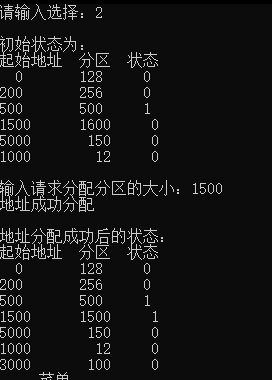
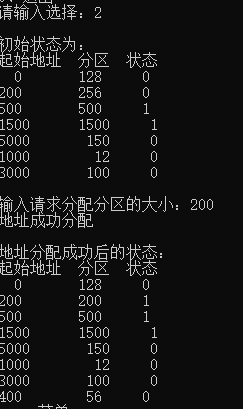
**主界面**



**首次适应算法**



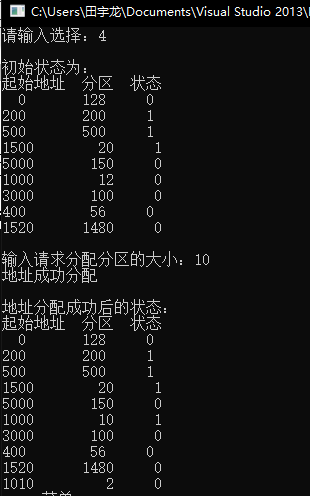
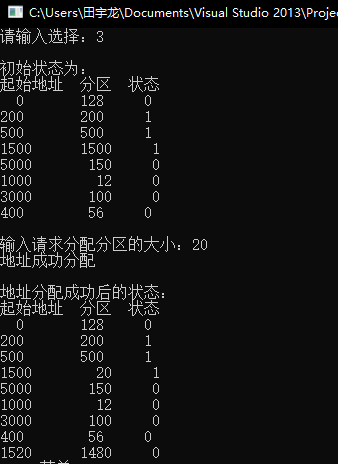
**循环首次适应算法**

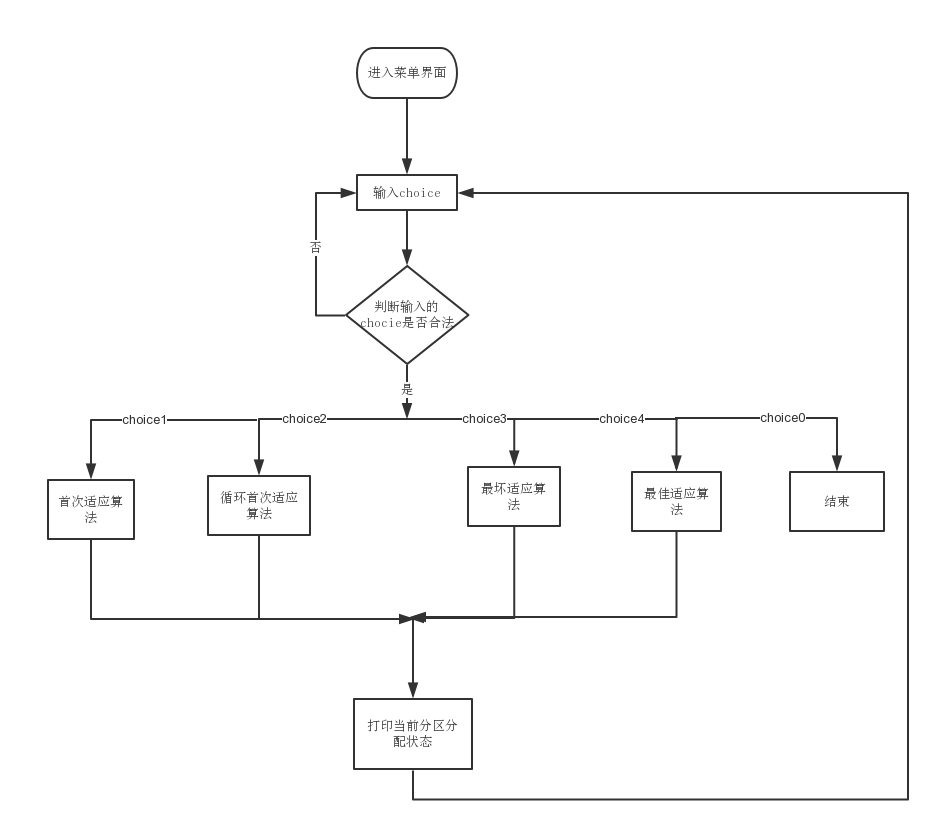
在第一次的分配作业大小为1500，则系统就会将地址为1500的分区进行分配，剩余分区令开辟空间。在第二次的分配作业大小为200，依照算法可得，地址为200的分区复合要求，则将其分配给作业

**最坏适应算法： 最佳适应算法：**

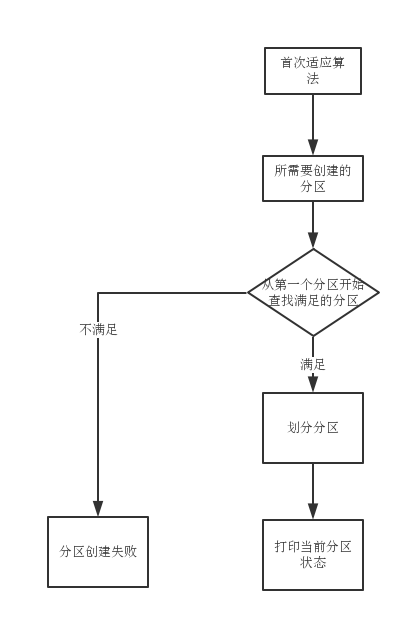
**从最大的分区开始分配 从最小的分区开始分配**



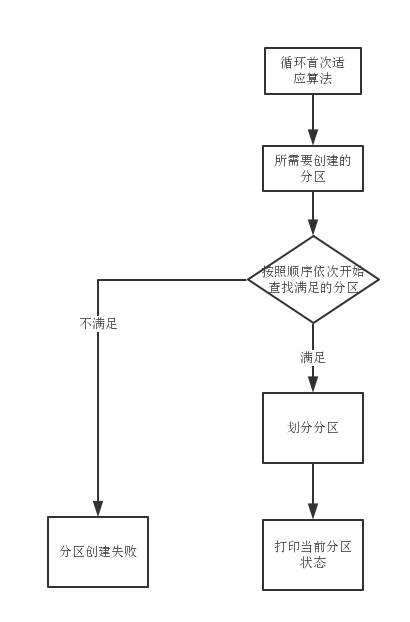
1. **程序流程图**

****

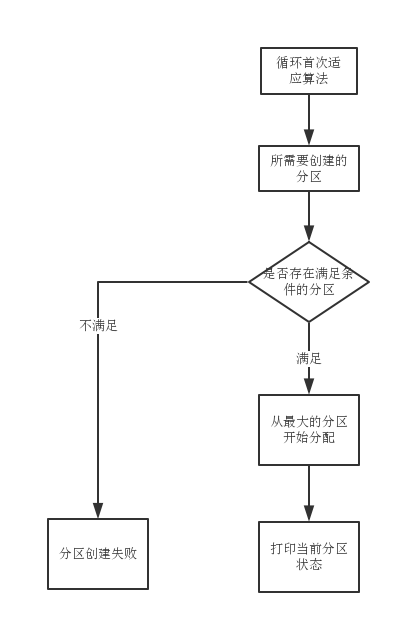
**首次适应算法**



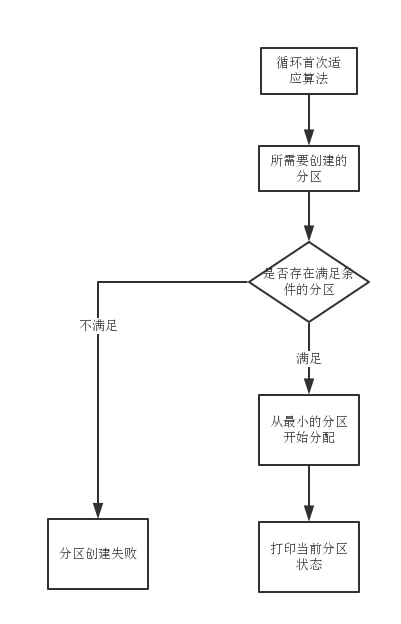
**循环首次适应算法**

****

**最坏适应算法**

****

**最佳适应算法**

****

西安工业大学实验报告

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 专业 | 物联网工程 | 班级 | 16060616 | 姓名 | 田宇龙 | 学号 | 16060616107 |
| 实验课程 | 计算机操作系统 | 指导教师 |  | 实验日期 |  | 同实验者 |  |
| 实验项目 |  | | | | | | |
| 实验设备及器材 |  | | | | | | |
|  |  | | | | | | |
|  |  | | | | | | |

1. **实验目的**

掌握进程调度的五种基本算法

采用合适自己的编程语言，实现操作系统作用进程调度常用的五种算法

**二．实现思想**

先来先服务调度算法：采用先来先服务（FCFS）调度算法，即按作业到达的先后次序进行调度。总是首先调度在系统中等待时间最长的作业

高响应比优先调度算法：为每个作业设置一个优先权(响应比)，调度之前先计算各作业的优先权，优先数高者优先调度。RP (响应比)＝ 作业周转时间 / 作业运行时间=1+作业等待时间/作业运行时间

时间片轮转：每个进程被分配一个时间段，称作它的时间片，即该进程允许运行的时间。如果在时间片结束时进程还在运行，则CPU将被剥夺并分配给另一个进程。如果进程在时间片结束前阻塞或结束，则CPU当即进行切换。调度程序所要做的就是维护一张就绪进程列表，当进程用完它的时间片后，它被移到队列的末尾

短进程优先调度算法：优先调度要求运行时间最短的作业

高优先权优先调度算法：

（1）当该算法用于作业调度时，系统从后备作业队列中选择若干个优先级最高的，且系统能满足资源要求的作业装入内存运行。

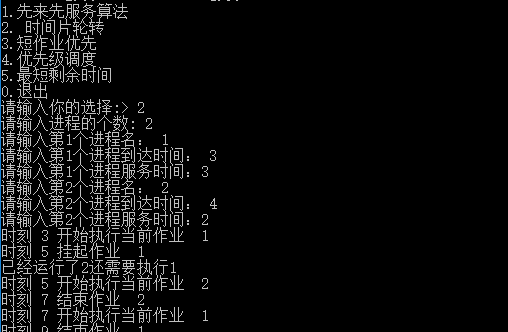
（2）当该算法用于进程调度时，将把处理机分配给就绪进程队列中优先级最高的进程。

**实验界面及实验结果运行图：**

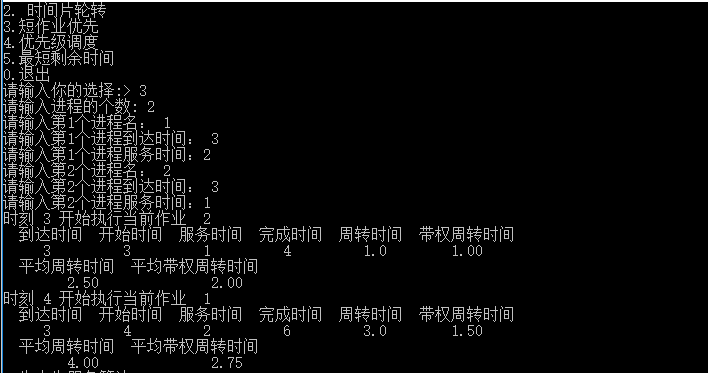
**先来先服务**

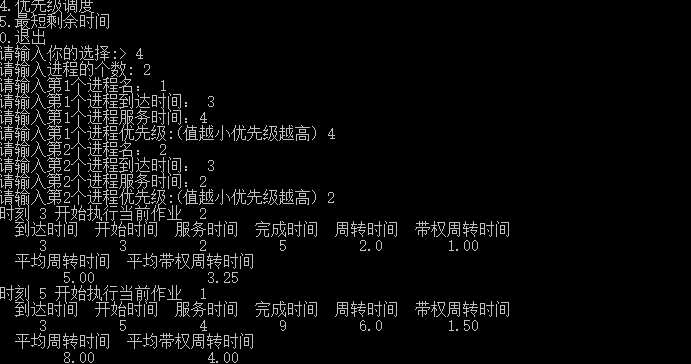


**时间片轮转：**

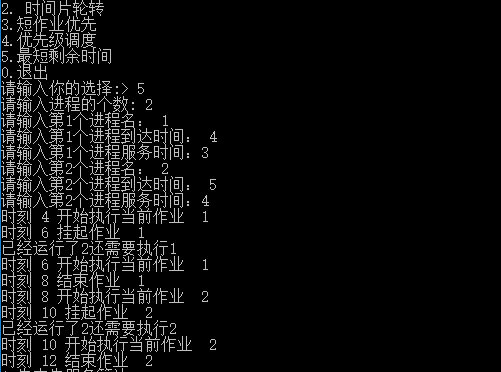


**短作业优先：**

****

**高优先级调度：**

**最短剩余时间：**

****

**程序源代码：**

#include<iostream>

using namespace std;

#define MAX\_PEOC\_ID 65536

#define TIME\_SLICE 2

//进程控制块 ，结构体定义

typedef struct PCB

{

char name[10];//进程名 id

char state; //进程状态 W/R

int ArriveTime; //进程到达时间

int StartTime; //进程开始时间

int FinshTime;//进程结束时间

int ServiceTime;//进程服务时间

float WholeTime; //周转时间

float Weight\_WholeTime;//带权周转时间

double Average\_WholeTime;//平均周转时间

double Average\_Weight\_WholeTime;//带权平均周转时间

int RunTime;//已经占用CPU时间

int NeedTime;//还要占用CPU时间

int Prio;//优先级

struct PCB \*next;

}pcb;

//全局变量

double Sum\_WholeTime=0,Sum\_Weight\_WholeTime=0;

int time=0;

int Proc\_Num=0;

pcb \*head = NULL;

pcb \*tail=NULL;

void FCFS\_RunProccess(pcb \*proc)

{

proc->StartTime=time;

cout<<"时刻 "<<time<<" 开始执行当前作业 "<<proc->name<<endl;

time+=proc->ServiceTime;

proc->state='R';

proc->FinshTime=time;

proc->WholeTime=proc->FinshTime-proc->ArriveTime;

proc->Weight\_WholeTime=proc->WholeTime/proc->ServiceTime;

Sum\_WholeTime+=proc->WholeTime;

Sum\_Weight\_WholeTime+=proc->Weight\_WholeTime;

proc->Average\_WholeTime=Sum\_WholeTime/Proc\_Num;

proc->Average\_Weight\_WholeTime=Sum\_Weight\_WholeTime/Proc\_Num;

printf(" 到达时间 开始时间 服务时间 完成时间 周转时间 带权周转时间\n");

printf("%6d %6d %6d %6d %8.1f %8.2f\n",

proc->ArriveTime,proc->StartTime,proc->ServiceTime,

proc->FinshTime,proc->WholeTime,proc->Weight\_WholeTime);

//printf("共 %d 个进程\n",Proc\_Num);

printf(" 平均周转时间 平均带权周转时间 \n");

printf(" %10.2f %10.2f\n",proc->Average\_WholeTime,

proc->Average\_Weight\_WholeTime);

}

//先来先服务算法

void FCFS()

{

pcb\* cur\_proc=head;

pcb\*new\_proc=NULL;

while(cur\_proc)

{

if(cur\_proc->state == 'W')

{

new\_proc=cur\_proc;

FCFS\_RunProccess(new\_proc);

}

cur\_proc=cur\_proc->next;

head=cur\_proc;

free (new\_proc);

new\_proc=NULL;

}

}

//进程创建

void FCFS\_CreateProccess()

{

cout<<"请输入进程的个数: ";

cin>>Proc\_Num;

if(Proc\_Num > MAX\_PEOC\_ID)

{

cout<<"sorry I can't give you PCB \n";

return;

}

for(int i=1;i<=Proc\_Num;++i)

{

pcb\*new\_proc=NULL;

if((new\_proc=(pcb\*)malloc(sizeof(pcb))) == NULL)

{

perror("malloc");

return;

}

cout<<"请输入第"<<i<<"个进程名： ";

cin>>new\_proc->name;

cout<<"请输入第"<<i<<"个进程到达时间： ";

cin>>new\_proc->ArriveTime;

cout<<"请输入第"<<i<<"个进程服务时间：";

cin>>new\_proc->ServiceTime;

new\_proc->next=NULL;

if(head == NULL)

{

new\_proc->next=head;

head=new\_proc;

tail=head;

time=new\_proc->ArriveTime;

}

else

{

if(head->ArriveTime>new\_proc->ArriveTime)

{

new\_proc->next=head;

head=new\_proc;

}

else

{

pcb\* cur\_proc=head;

while(cur\_proc->next != NULL && cur\_proc->next->ArriveTime<new\_proc->ArriveTime)

{

cur\_proc=cur\_proc->next;

}

if(cur\_proc->next==NULL)

{

tail=new\_proc;

}

new\_proc->next=cur\_proc->next;

cur\_proc->next=new\_proc;

}

if(new\_proc->ArriveTime < time)

{

time=new\_proc->ArriveTime;

}

}

new\_proc->StartTime=0;

new\_proc->FinshTime=0;

new\_proc->WholeTime=0;

new\_proc->Weight\_WholeTime=0;

new\_proc->Average\_Weight\_WholeTime=0;

new\_proc->Average\_WholeTime=0;

new\_proc->state= 'W';

new\_proc->RunTime=0;

new\_proc->NeedTime=0;

}

}

//短作业优先创建

void ShortJobsCreateProccess()

{

cout<<"请输入进程的个数: ";

cin>>Proc\_Num;

if(Proc\_Num > MAX\_PEOC\_ID)

{

cout<<"sorry I can't give you PCB \n";

return;

}

for(int i=1;i<=Proc\_Num;++i)

{

pcb\*new\_proc=NULL;

if((new\_proc=(pcb\*)malloc(sizeof(pcb))) == NULL)

{

perror("malloc");

return;

}

cout<<"请输入第"<<i<<"个进程名： ";

cin>>new\_proc->name;

cout<<"请输入第"<<i<<"个进程到达时间： ";

cin>>new\_proc->ArriveTime;

cout<<"请输入第"<<i<<"个进程服务时间：";

cin>>new\_proc->ServiceTime;

new\_proc->next=NULL;

if(head == NULL)

{

new\_proc->next=head;

head=new\_proc;

tail=head;

time=new\_proc->ArriveTime;

}

else

{

if(head->ServiceTime>new\_proc->ServiceTime)

{

new\_proc->next=head;

head=new\_proc;

FCFS\_CreateProccess }

else

{

pcb\* cur\_proc=head;

while(cur\_proc->next != NULL && cur\_proc->next->ServiceTime<new\_proc->ServiceTime)

{

cur\_proc=cur\_proc->next;

}

if(cur\_proc->next==NULL)

{

tail=new\_proc;

}

new\_proc->next=cur\_proc->next;

cur\_proc->next=new\_proc;

}

if(new\_proc->ArriveTime < time)

{

time=new\_proc->ArriveTime;

}

}

new\_proc->StartTime=0;

new\_proc->FinshTime=0;

new\_proc->WholeTime=0;

new\_proc->Weight\_WholeTime=0;

new\_proc->Average\_Weight\_WholeTime=0;

new\_proc->Average\_WholeTime=0;

new\_proc->state= 'W';

new\_proc->RunTime=0;

new\_proc->NeedTime=0;

}

}

void PrioCreateProccess()

{

cout<<"请输入进程的个数: ";

cin>>Proc\_Num;

if(Proc\_Num > MAX\_PEOC\_ID)

{

cout<<"sorry I can't give you PCB \n";

return;

}

for(int i=1;i<=Proc\_Num;++i)

{

pcb\*new\_proc=NULL;

if((new\_proc=(pcb\*)malloc(sizeof(pcb))) == NULL)

{

perror("malloc");

return;

}

cout<<"请输入第"<<i<<"个进程名： ";

cin>>new\_proc->name;

cout<<"请输入第"<<i<<"个进程到达时间： ";

cin>>new\_proc->ArriveTime;

cout<<"请输入第"<<i<<"个进程服务时间：";

cin>>new\_proc->ServiceTime;

cout<<"请输入第"<<i<<"个进程优先级:(值越小优先级越高) ";

cin>>new\_proc->Prio;

new\_proc->next=NULL;

if(head == NULL)

{

new\_proc->next=head;

head=new\_proc;

tail=head;

time=new\_proc->ArriveTime;

}

else

{

if(head->Prio>new\_proc->Prio)

{

new\_proc->next=head;

head=new\_proc;

}

else

{

pcb\* cur\_proc=head;

while(cur\_proc->next != NULL && cur\_proc->next->Prio<new\_proc->Prio)

{

cur\_proc=cur\_proc->next;

}

if(cur\_proc->next==NULL)

{

tail=new\_proc;

}

new\_proc->next=cur\_proc->next;

cur\_proc->next=new\_proc;

}

if(new\_proc->ArriveTime < time)

{

time=new\_proc->ArriveTime;

}

}

new\_proc->StartTime=0;

new\_proc->FinshTime=0;

new\_proc->WholeTime=0;

new\_proc->Weight\_WholeTime=0;

new\_proc->Average\_Weight\_WholeTime=0;

new\_proc->Average\_WholeTime=0;

new\_proc->state= 'W';

new\_proc->RunTime=0;

new\_proc->NeedTime=0;

}

}

//时间片

void RR\_RunProccess(PCB \*proc)

{

proc->StartTime=time;

cout<<"时刻 "<<time<<" 开始执行当前作业 "<<proc->name<<endl;

proc->RunTime+=TIME\_SLICE;

time+=TIME\_SLICE;

proc->NeedTime=proc->ServiceTime-proc->RunTime;

if(proc->NeedTime <=0)

{

cout<<"时刻 "<<time<<" 结束作业 "<<proc->name<<endl;

head=proc->next;

free(proc);

proc=NULL;

if(head==NULL)

{

tail=head;

}

}

else

{

cout<<"时刻 "<<time<<" 挂起作业 "<<proc->name<<endl;

cout<<"已经运行了"<<proc->RunTime<<"还需要执行"<<proc->NeedTime<<endl;

if(proc->next != NULL)

{

head=proc->next;

proc->next=NULL;

tail->next=proc;

tail=proc;

}

}

}

void RoundRobin()

{

pcb\* cur\_proc=head;

pcb\* new\_proc=NULL;

while(cur\_proc)

{

if(cur\_proc->state == 'W')

{

new\_proc=cur\_proc;

RR\_RunProccess(new\_proc);

}

cur\_proc=head;

}

}

void Sort\_By\_NeedTime()

{

pcb\* prev=head;

pcb\*cur=head->next;

prev->next=NULL;

pcb\*nxt=NULL;

if( cur && head->NeedTime <= cur->NeedTime)

{

cur->next=head;

head=cur;

}

else if(cur)

{

while(cur && prev->next != NULL && prev->next->NeedTime <= cur->NeedTime)

{

prev=prev->next;

}

nxt=cur->next;

cur->next=prev->next;

prev->next=cur;

cur=nxt;

}

}

void ShortNeed\_Time\_RunProccess(pcb \*proc)

{

proc->StartTime=time;

cout<<"时刻 "<<time<<" 开始执行当前作业 "<<proc->name<<endl;

proc->RunTime+=TIME\_SLICE;

time+=TIME\_SLICE;

proc->NeedTime=proc->ServiceTime-proc->RunTime;

if(proc->NeedTime <=0)

{

cout<<"时刻 "<<time<<" 结束作业 "<<proc->name<<endl;

head=proc->next;

free(proc);

proc=NULL;

if(head==NULL)

{

tail=head;

}

}

else

{

cout<<"时刻 "<<time<<" 挂起作业 "<<proc->name<<endl;

cout<<"已经运行了"<<proc->RunTime<<"还需要执行"<<proc->NeedTime<<endl;

Sort\_By\_NeedTime();

}

}

void ShortNeed\_Time()

{

pcb\* cur\_proc=head;

pcb\* new\_proc=NULL;

while(cur\_proc)

{

if(cur\_proc->state == 'W')

{

new\_proc=cur\_proc;

ShortNeed\_Time\_RunProccess(new\_proc);

}

cur\_proc=head;

}

}

void main()

{

int select=1;

while(select)

{

cout<<"1. 先来先服务算法 \n";

cout<<"2.时间片轮转 \n";

cout<<"3.短作业优先 \n";

cout<<"4.优先级调度\n";

cout<<"5.\*\*\*\*\*\*\*\* 最短剩余时间\n";

cout<<"\*0.\*\*\*\*\*\*\*\*\*\*退出\n";

cout<<"请输入你的选择:> ";

cin>>select;

switch(select)

{

case 1:

FCFS\_CreateProccess();

FCFS();

break;

case 2:

FCFS\_CreateProccess();

RoundRobin();

break;

case 3:

ShortJobsCreateProccess();

FCFS();

break;

case 4:

PrioCreateProccess();

FCFS();

break;

case 5:

ShortJobsCreateProccess();

ShortNeed\_Time();

break;

default:

break;

}

}

}