

hzau-1

**实验1**

**进程调度**

姓 名：马家乐

学 号: 2019310220321

专业班级：信息1903班

课程名：操作系统实验

指导教师：倪福川

完成日期：2019.6.1

**一、实验目的**

**1. 目的**

进程调度是处理及管理的核心内容，本次实验要求用C语言编写和调试一个简单的进程调度程序。调度算法可以任意选择或自行设计，例如简单轮转法和优先数法等。通过本次实验可以加深各使用进程控制块进行进程调度和各种调度算法的理解及其实施方法。

**2. 要求**

1. 在微型计算机上设计进程控制块（PCB）结构，使其分别适用于简单轮转法和优先数调度算法。PCB通常包括以下信息：进程名、进程优先数（或轮转时间片）、进程所占用的CPU时间、进程的当前状态、当前队列指针等。根据调度算法的不同，PCB结构的内容可以作适当的增删。
2. 调度程序应包含2~3种不同的调度算法，运行时可任选一种，以利于各种算法的分析比较。基本要求是：优先数调度算法和简单循环轮转法。
3. 建立进程就绪队列，对各种不同的算法编制入链子程序，同时应具有显示或打印各进程的运行状态和参数的变化情况，便于观察各进程的调度过程。
4. 本次实验的上机时间为4学时。

**二、实验内容**

采用C语言编写程序，选用优先数调度算法或简单轮转法对五个进程进行调度，每个进程处于运行（Run）、就绪（Ready）和完成（Finish）三种状态之一，并假定起始状态为就绪状态。

**三、实验设计**

**a.优先数调度：**

**任务分析：**

进程就绪队列按优先数大小从高到低排列，链首进程首先投入运行。进程每执行一次，进程需要的时间片数减1、该进程的优先数减3。这样，该进程如果在一个时间片中没有完成，其优先数降低一级。接着仍是用该进程降低一级后的优先数与就绪队列中链首进程的优先数进行比较，如果仍是该进程的优先数高或相同，便让该进程继续执行；否则，调度就绪队列的链首进程投入运行。原运行过的进程按其现行优先数大小插入就绪队列，且改变它们对应的进程状态，一直到所有进程都运行完各自的时间片数。

**概要设计**：

定义一个进程控制块对应的数据结构

class PCB

{

public:

int id;//进程标识符

PCB \*next ;//连接指针

int priority\_num ;//优先数

int runtime ;//占用的CPU时间片数

int needtime ;//进程所需时间片数

char state;//进程状态

PCB()

{

id=0;

next=NULL;

priority\_num=0;

runtime=0;

needtime=0;

state='R';//R-就绪,r-运行,F-完成

}

};

定义一个运行、就绪、完成队列指针的集合的数据结构

class PCBS

{

public:

PCB \*run;//运行进程指针

PCB \*ready;//就绪队列头指针

PCB \*tail;//就绪队列尾指针

PCB \*finish;//完成队列指针

PCBS()

{

run=NULL;

ready=NULL;

tail=NULL;

finish=NULL;

}

};

定义功能函数模块：

void input\_readyqueue(PCBS \*pcbs)//插入进程至就绪队列

void Swap(PCB \* p1 , PCB \*p2)//交换两个进程的顺序

void Sort(PCBS \*pcbs)//将就绪队列按进程优先数从大到小排序

void print(PCBS \*pcbs)//打印运行、就绪、结束队列

void priority\_num\_algorithm(PCBS \*pcbs)//优先数调度算法

主函数流程：

建立就绪、运行、完成队列，插入进程至就绪队列；按照优先数调度算法进行进程调度。

程序框图：

开始

生成并按优先数大小排列进程控制块

链首进程投入运行

时间片到，进程时间片数减1，优先数减3

进程时间片数为0？

撤销该进程

从链首取一个进程投入运行

优先数大于链首进程？

运行进程推出，按优先数插入进程链

进程队列空？

结束

是

否

否

是

优先数调度算法流程图

**源代码：**

#include<iostream>

using namespace std;

class PCB

{

public:

int id;//进程标识符

PCB \*next ;//连接指针

int priority\_num ;//优先数

int runtime ;//占用的CPU时间片数

int needtime ;//进程所需时间片数

char state;//进程状态

PCB()

{

id=0;

next=NULL;

priority\_num=0;

runtime=0;

needtime=0;

state='R';//R-就绪,r-运行,F-完成

}

};

class PCBS

{

public:

PCB \*run;//运行进程指针

PCB \*ready;//就绪队列头指针

PCB \*tail;//就绪队列尾指针

PCB \*finish;//完成队列指针

PCBS()

{

run=NULL;

ready=NULL;

tail=NULL;

finish=NULL;

}

};

void input\_readyqueue(PCBS \*pcbs)//插入进程至就绪队列

{ int i;int p;int r;int n;

PCB \*pcb=new PCB();

cout<<"请输入进程标识符：";

cin>>i;

cout<<"输入格式为： （优先数，占用CPU时间片数，进程所需时间片数） : ";

cin>>p>>r>>n;

pcb->id=i;

pcb->priority\_num=p;

pcb->runtime=r;

pcb->needtime=n;

if(pcbs->ready == NULL && pcbs->tail == NULL)

{

pcbs->ready = pcb;

pcbs->tail = pcb;

}

else

{

pcbs->tail->next = pcb ;

pcbs->tail = pcb ;

}

}

void Swap(PCB \* p1 , PCB \*p2)//交换两个进程的顺序

{

int i,p,r,n;

i = p1->id;

p1->id = p2->id;

p2->id = i;

p = p1->priority\_num ;

p1->priority\_num = p2->priority\_num ;

p2->priority\_num = p ;

r = p1->runtime ;

p1->runtime = p2->runtime ;

p2->runtime = r ;

n = p1->needtime ;

p1->needtime = p2->needtime ;

p2->needtime = n ;

}

void Sort(PCBS \*pcbs)// 将就绪队列按进程优先数从大到小排序

{

PCB \*p , \*p1 ;

for(p=pcbs->ready; p != NULL ; p = p->next)

{

for(p1 = p->next ; p1 != NULL ; p1 = p1->next)

{

if(p->priority\_num < p1->priority\_num)

{

Swap(p , p1) ;

}

}

}

}

void print(PCBS \*pcbs)//打印运行、就绪、结束队列

{ PCB \*r,\*f;

r=pcbs->ready;f=pcbs->finish;

cout<<"-------------------------------------------------- \n";

cout<<"运行队列："<<endl;

if(pcbs->run != NULL)

{

cout<<"进程标识符："<<pcbs->run->id<<" 进程优先数："<<pcbs->run->priority\_num<<" 进程占用CPU时间片数："<<pcbs->run->runtime<<" 进程所需时间片数:"<<pcbs->run->needtime<<endl;

}

else

{

cout<<"运行队列为空"<<endl;

}

cout<<"就绪队列："<<endl;

if(r==NULL)

{

cout<<"就绪队列为空"<<endl;

}

else

{

while(r != NULL)

{

cout<<"进程标识符："<<r->id<<" 进程优先数："<<r->priority\_num<<" 进程占用CPU时间片数："<<r->runtime<<" 进程所需时间片数:"<<r->needtime<<endl;

r = r->next ;

}

}

cout<<"结束队列："<<endl;

if(f==NULL)

{

cout<<"就绪队列为空"<<endl;

}

else

{

while(f != NULL)

{

cout<<"进程标识符："<<f->id<<" 进程优先数："<<f->priority\_num<<" 进程占用CPU时间片数："<<f->runtime<<" 进程所需时间片数:"<<f->needtime<<endl;

f = f->next ;

}

}

}

void priority\_num\_algorithm(PCBS \*pcbs)//优先数调度算法

{

Sort(pcbs);

print(pcbs);

while(pcbs->ready != NULL)

{

pcbs->run = pcbs->ready ;

pcbs->run->state='r';

pcbs->ready = pcbs->ready->next ;

pcbs->run->next = NULL;

print(pcbs);

pcbs->run->runtime = pcbs->run->runtime+1;

pcbs->run->priority\_num = pcbs->run->priority\_num-3 ;

pcbs->run->needtime = pcbs->run->needtime-1 ;

if(pcbs->run->needtime == 0)

{

if(pcbs->finish == NULL)

{

pcbs->finish = pcbs->run ;

pcbs->finish->state='F';

pcbs->finish->next = NULL ;

}

else

{

PCB \*f=pcbs->finish;

pcbs->finish=pcbs->run;

pcbs->finish->state='F';

pcbs->finish->next=f;

}

}

else

{

if(pcbs->ready != NULL && pcbs->tail != NULL)

{

PCB \*r=pcbs->ready;

pcbs->ready=pcbs->run;

pcbs->ready->state='R';

pcbs->ready->next=r;

Sort(pcbs);

}

else

{

pcbs->ready = pcbs->run ;

pcbs->tail = pcbs->run ;

pcbs->ready->state='R';

}

}

}

pcbs->run = NULL ;

print(pcbs);

}

int main()

{

PCBS \*pcbs=new PCBS();

int n;

cout<<"请输入要处理的进程数目： ";

cin>>n;

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

{

input\_readyqueue(pcbs);

}

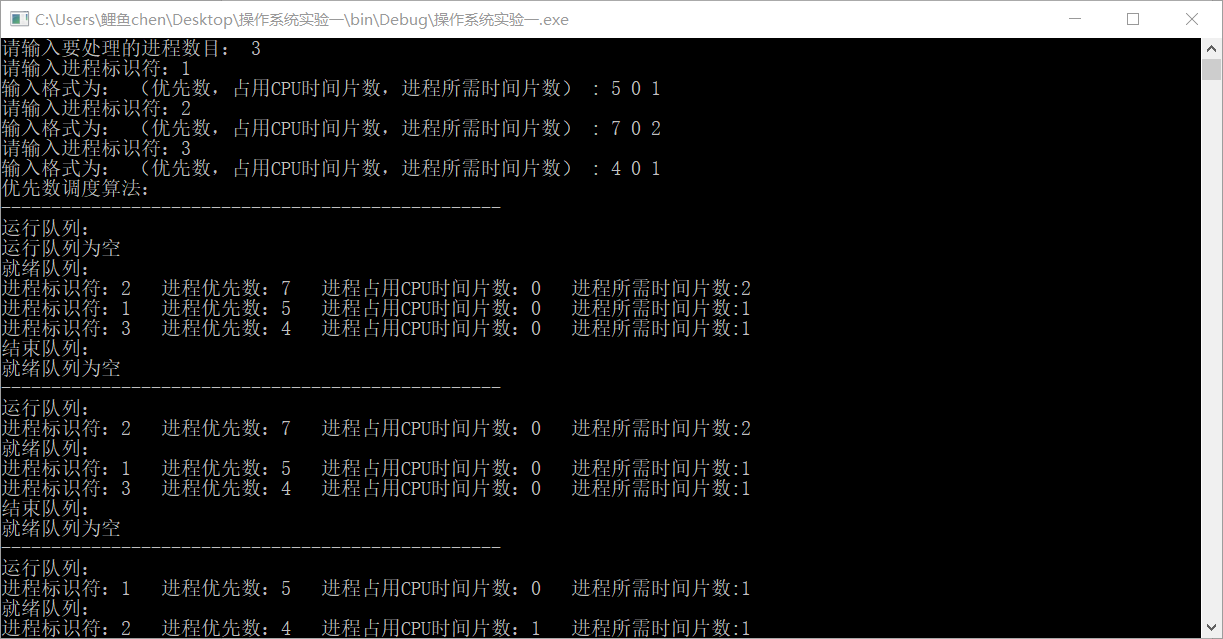
cout<<"优先数调度算法："<<endl;

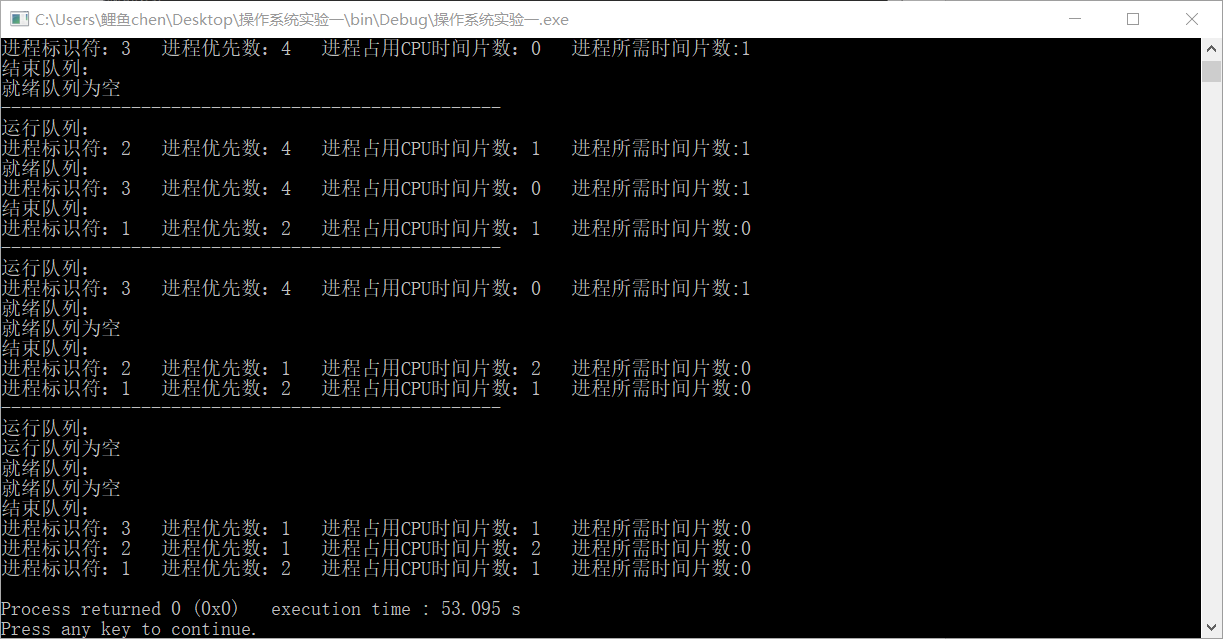
priority\_num\_algorithm(pcbs);

return 0 ;

}

**输入输出示例：**





b.时间片轮转调度：

**任务分析：**

进程就绪队列按各进程进入的先后顺序排列。进程每次所需处理机的轮转式按其重要程度记入进程控制块中的轮转时间片数记录项。进程执行时，每运行一个时间片，进程还需要的时间片数减1，运行进程占用处理机的时间片数加1，然后比较占用CPU的时间片数是否与该进程的轮转时间片数相等，若相等则说明已达到轮转时间，应将现运行的进程排列就绪队列的末尾，调度队列上的首进程运行，且改变它们的进程状态，直至所有进程完成各自的时间片。

**概要设计：**

定义一个进程控制块对应的数据结构

class PCB

{

public:

int id;//进程标识符

PCB \*next ;//连接指针

int time\_slice ;//轮转时间片

int runtime ;//占用的CPU时间片数

int needtime ;//进程所需时间片数

char state;//进程状态

PCB()

{

id=0;

next=NULL;

time\_slice=0;

runtime=0;

needtime=0;

state='R';//R-就绪,r-运行,F-完成

}

};

定义一个运行、就绪、完成队列指针的集合

class PCBS

{

public:

PCB \*run;//运行进程指针

PCB \*ready;//就绪队列头指针

PCB \*tail;//就绪队列尾指针

PCB \*finish;//完成队列指针

PCBS()

{

run=NULL;

ready=NULL;

tail=NULL;

finish=NULL;

}

};

定义几个功能函数模块：

void input\_readyqueue(PCBS \*pcbs)//插入进程至就绪队列

void print(PCBS \*pcbs)//打印运行、就绪、完成队列

void time\_slice\_algorithm(PCBS \*pcbs)//时间片轮转调度算法

主函数流程：

建立就绪、运行、完成队列，插入进程至就绪队列；按照时间片轮转调度算法进行进程调度。

程序框图：

开始

生成并按进入次序排列进程控制块链

链首进程投入运行

时间片到，进程时间片数减1，占用CPU时间加1

进程时间片数为0？

撤销该进程

从链首取一个进程投入运行

占用CPU时间已到轮转时间？

运行进程推出，排到运行链尾部

否

否

是 是

进程队列空？

结束

否

是

时间片轮转调度算法流程图

**源代码：**

#include<iostream>

using namespace std;

class PCB

{

public:

int id;//进程标识符

PCB \*next ;//连接指针

int time\_slice ;//轮转时间片

int runtime ;//占用的CPU时间片数

int needtime ;//进程所需时间片数

char state;//进程状态

PCB()

{

id=0;

next=NULL;

time\_slice=0;

runtime=0;

needtime=0;

state='R';//R-就绪,r-运行,F-完成

}

};

class PCBS

{

public:

PCB \*run;//运行进程指针

PCB \*ready;//就绪队列头指针

PCB \*tail;//就绪队列尾指针

PCB \*finish;//完成队列指针

PCBS()

{

run=NULL;

ready=NULL;

tail=NULL;

finish=NULL;

}

};

void input\_readyqueue(PCBS \*pcbs)//插入进程至就绪队列

{ int i;int t;int r;int n;

PCB \*pcb=new PCB();

cout<<"请输入进程标识符：";

cin>>i;

cout<<"输入格式为： （轮转时间片，占用CPU时间片数，进程所需时间片数） : ";

cin>>t>>r>>n;

pcb->id=i;

pcb->time\_slice=t;

pcb->runtime=r;

pcb->needtime=n;

if(pcbs->ready == NULL && pcbs->tail == NULL)

{

pcbs->ready = pcb;

pcbs->tail = pcb;

}

else

{

pcbs->tail->next = pcb ;

pcbs->tail = pcb ;

}

}

void print(PCBS \*pcbs)//打印运行、就绪、完成队列

{ PCB \*r,\*f;

r=pcbs->ready;f=pcbs->finish;

cout<<"-------------------------------------------------- \n";

cout<<"运行队列："<<endl;

if(pcbs->run != NULL)

{

cout<<"进程标识符："<<pcbs->run->id<<" 轮转时间片："<<pcbs->run->time\_slice<<" 进程占用CPU时间片数："<<pcbs->run->runtime<<" 进程所需时间片数:"<<pcbs->run->needtime<<endl;

}

else

{

cout<<"运行队列为空"<<endl;

}

cout<<"就绪队列："<<endl;

if(r==NULL)

{

cout<<"就绪队列为空"<<endl;

}

else

{

while(r != NULL)

{

cout<<"进程标识符："<<r->id<<" 轮转时间片："<<r->time\_slice<<" 进程占用CPU时间片数："<<r->runtime<<" 进程所需时间片数:"<<r->needtime<<endl;

r = r->next ;

}

}

cout<<"结束队列："<<endl;

if(f==NULL)

{

cout<<"就绪队列为空"<<endl;

}

else

{

while(f != NULL)

{

cout<<"进程标识符："<<f->id<<" 轮转时间片："<<f->time\_slice<<" 进程占用CPU时间片数："<<f->runtime<<" 进程所需时间片数:"<<f->needtime<<endl;

f = f->next ;

}

}

}

void time\_slice\_algorithm(PCBS \*pcbs)//时间片轮转调度算法

{ print(pcbs);

while(pcbs->ready != NULL)

{

pcbs->run = pcbs->ready ;

pcbs->run->state='r';

pcbs->ready = pcbs->ready->next ;

pcbs->run->next = NULL;

print(pcbs);

pcbs->run->runtime = pcbs->run->runtime+1;

pcbs->run->needtime = pcbs->run->needtime-1 ;

if(pcbs->run->needtime == 0)

{

if(pcbs->finish == NULL)

{

pcbs->finish = pcbs->run ;

pcbs->finish->state='F';

pcbs->finish->next = NULL ;

}

else

{

PCB \*f=pcbs->finish;

pcbs->finish=pcbs->run;

pcbs->finish->state='F';

pcbs->finish->next=f;

}

}

else

{

if(pcbs->run->time\_slice!=pcbs->run->runtime)

{

if(pcbs->ready != NULL && pcbs->tail != NULL)

{

PCB \*r=pcbs->ready;

pcbs->ready=pcbs->run;

pcbs->ready->state='R';

pcbs->ready->next=r;

}

else

{

pcbs->ready = pcbs->run ;

pcbs->tail = pcbs->run ;

pcbs->ready->state='R';

}

}

else

{ pcbs->run->runtime=0;

if(pcbs->ready == NULL && pcbs->tail == NULL)

{

pcbs->ready = pcbs->run;

pcbs->tail = pcbs->run;

pcbs->ready->state='R';

}

else

{

pcbs->tail->next = pcbs->run ;

pcbs->tail = pcbs->run ;

pcbs->tail->state='R';

}

}

}

}

pcbs->run = NULL ;

print(pcbs);

}

int main()

{

PCBS \*pcbs=new PCBS();

int n;

cout<<"请输入要处理的进程数目： ";

cin>>n;

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

{

input\_readyqueue(pcbs);

}

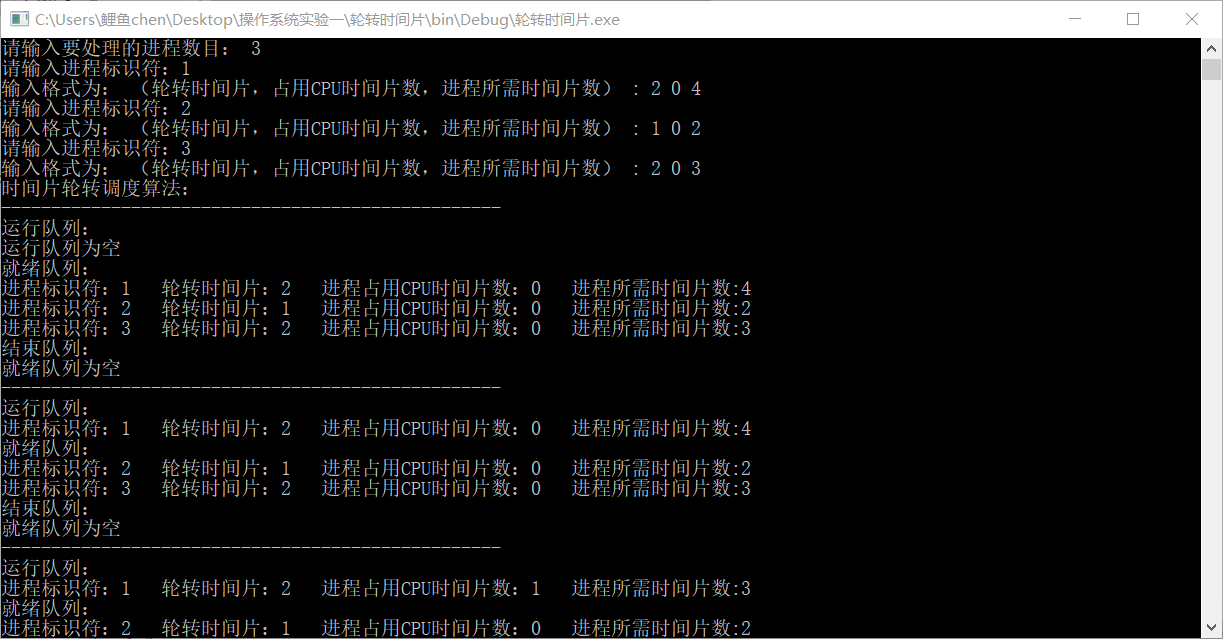
cout<<"时间片轮转调度算法："<<endl;

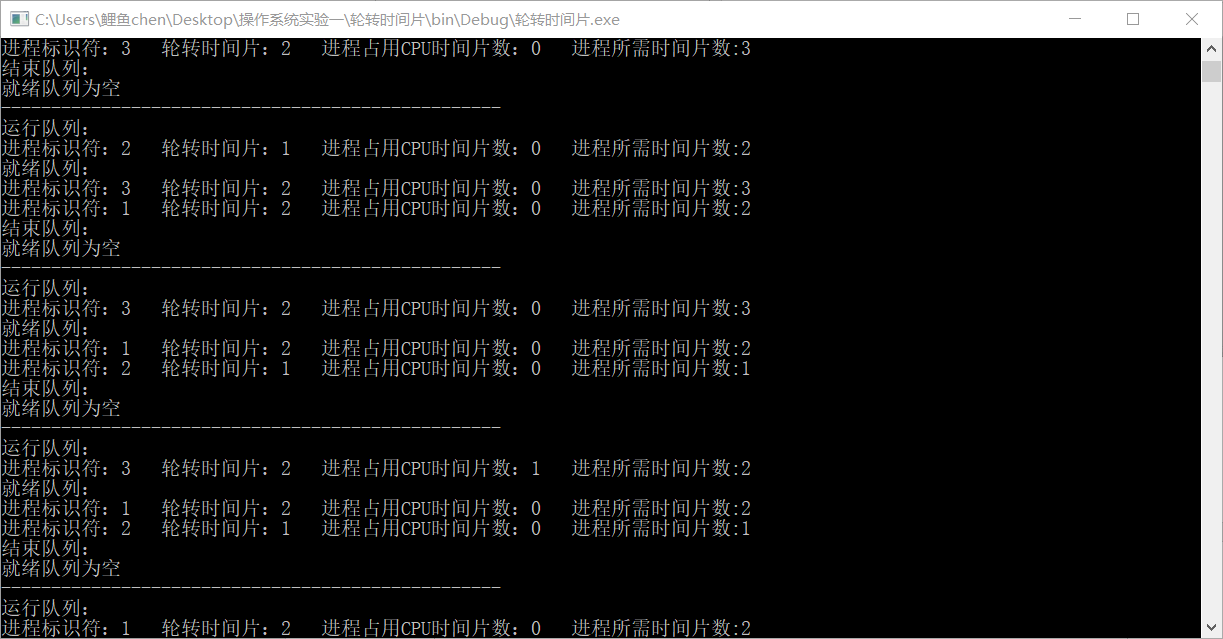
time\_slice\_algorithm(pcbs);

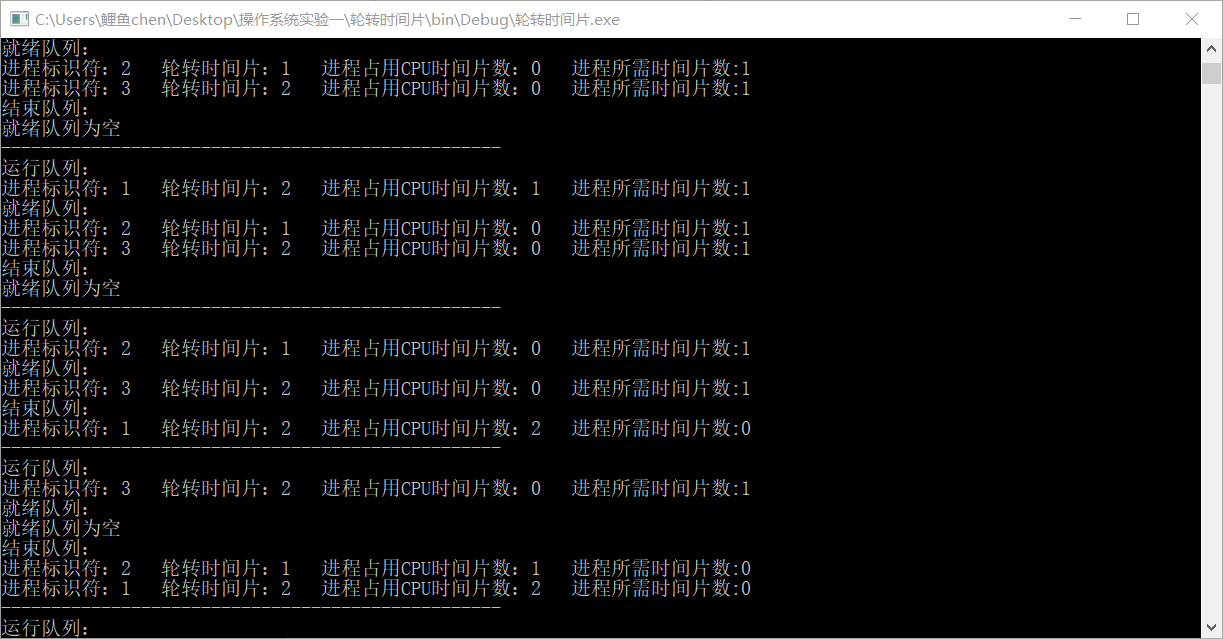
return 0 ;

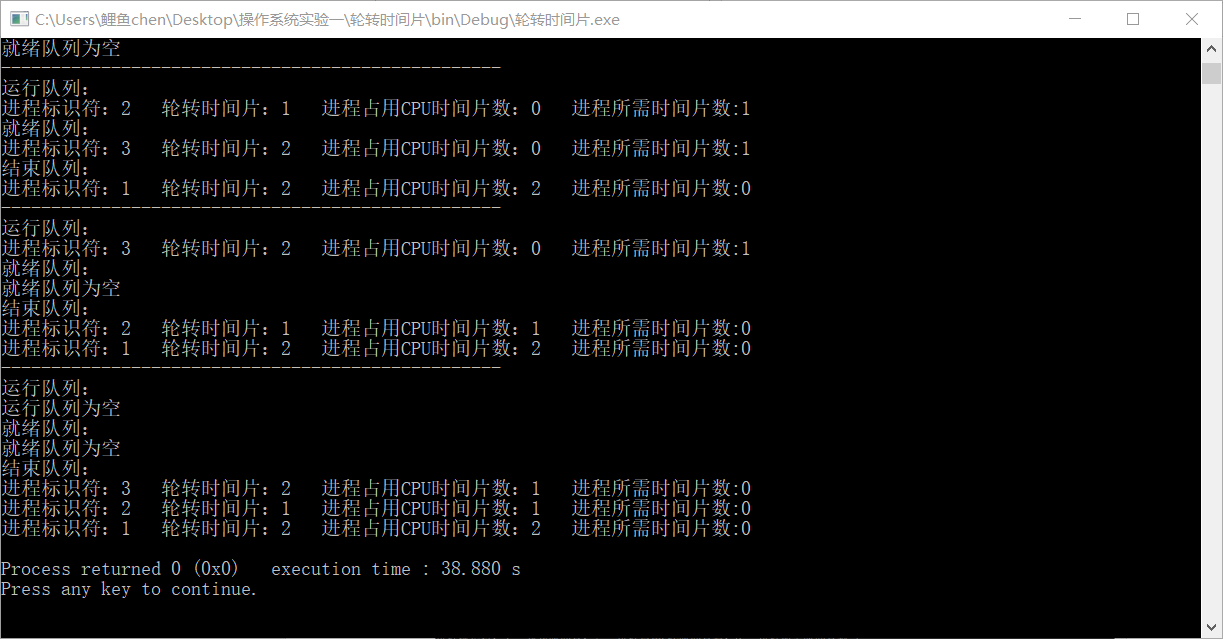
}

**输入输出示例：**









**四、实验心得**

这是操作系统课程的第一次实验，我好久没写代码了，刚开始有点生疏，慢慢找回了状态，这次实验令我受益匪浅，通过写代码的过程更加清晰地了解了两种进程调度算法的思想和用法，对算法的了解加深的同时也锻炼了写代码的能力，希望接下来的实验代码我能写的越来越顺利，越来越漂亮！