实验1 处理器调度 实验报告

信息安全1501沈鑫楠

3150604028

## 1.实验题目

设计一个按优先数调度算法实现处理器调度的进程。

## 2.实验目的

在采用多道程序设计的系统中，往往有若干个进程同时处于就绪状态。当就绪状态进程 个数大于处理器数时，就必须依照某种策略来决定哪些进程优先占用处理器。本实验模拟在 单处理器情况下处理器调度，帮助学生加深了解处理器调度的工作。

## 3.程序中使用的数据结构及符号说明

### 3.1程序中使用的数据结构

由于处理器调度中需要把若干个进程进行调度，所以需要一个队列来实现这种调度。调度时首先从队列的头开始，所以需要一个头指针指向队列的首部。本程序中使用vector这种数据结构来进行进程调度的管理。每个进程有一个进程控制块PCB，在调度时应该以它为单位进行调度，所以应该把它设置为结构体。

### 3.2程序中的符号说明

1）enum Status{ R, E };//进程状态

2）struct PCB//进程控制块结构体

{

char name[5];//进程名

PCB \*next;//下一个执行进程的指针

int time;//执行时间

int sequence;//优先级

Status status;//进程状态

};

3）PCB \*head = &k2;//头指针

4）vector<PCB> q;//存储进程队列的向量

vector<PCB>::iterator pos1,pos2;//迭代器

## 4.程序执行流程图



## 5.源程序

#include <iostream>

#include <vector>

using namespace std;

enum Status{ R, E };//进程状态

struct PCB//进程控制块结构体

{

char name[5];//进程名

PCB \*next;//下一个执行进程的指针

int time;//执行时间

int sequence;//优先级

Status status;//进程状态

};

PCB k1 = { "P1", NULL, 2, 1, R }, k5 = { "P5", &k1, 4, 2, R }, k3 = { "P3", &k5, 1, 3, R }, k4 = { "P4", &k3, 2, 4, R }, k2 = { "P2", &k4, 3, 5, R };//5个进程的初始状态

PCB \*head = &k2;//头指针

vector<PCB> q;//存储进程队列的向量

vector<PCB>::iterator pos1,pos2;//迭代器

void pro(PCB \*&p)

{

cout << p->name << " selected.~~~" << endl;

p->time--;//时间-1

p->sequence--;//优先级-1

if (p->time <= 0)//时间<=0

{

p->status = E;//把标志置为E

q.erase(q.begin());//从队列中去除

}

else if (p->time>0)

{

//按优先级大小排序

for (pos1 = q.begin(); pos1 != q.end();++pos1)

for (pos2 = pos1 + 1; pos2 != q.end();++pos2)

if (pos1->sequence<pos2->sequence)

{

swap(\*pos1, \*pos2);

}

while (head->status!=R)

{

head = head->next;

}

}

}

int main()

{

PCB t;

PCB \*temp = head;

//打印初始各进程控制块

cout << "name sequence time status" << endl;

while (temp)

{

t = \*temp;

q.push\_back(t);//插入队列

cout << t.name << "\t" << t.sequence << "\t" << t.time << "\t" << t.status << endl;//打印

temp = temp->next;

}

cout << endl;

int i = 0;

temp =&q[0];//指向第一个要执行的进程

while (!q.empty())

{

pro(temp);//执行进程

//打印进程控制块

for (int j = 0; j < q.size();j++)

{

cout << q[j].name << "\t" << q[j].sequence << "\t" << q[j].time << "\t" << q[j].status << endl;

}

cout << endl;

if (q.size() == 0)//全部执行完毕退出

{

break;

}

}

return 0;

}

## 6.程序运行初值和运行结果

### 6.1进程控制块的初始状态

P1

0

2

1

R

P2

k4

3

5

R

P3

k5

1

3

R

P4

k3

2

4

R

P5

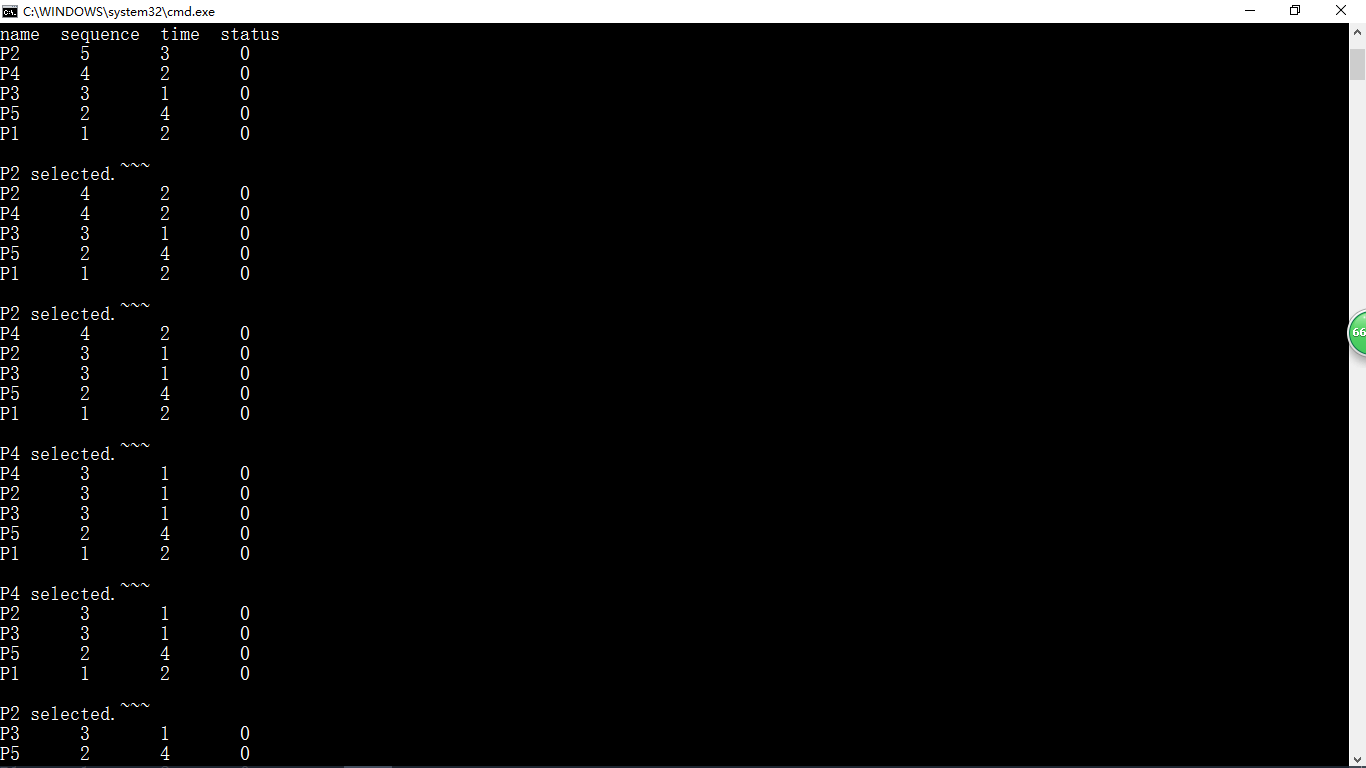
k1

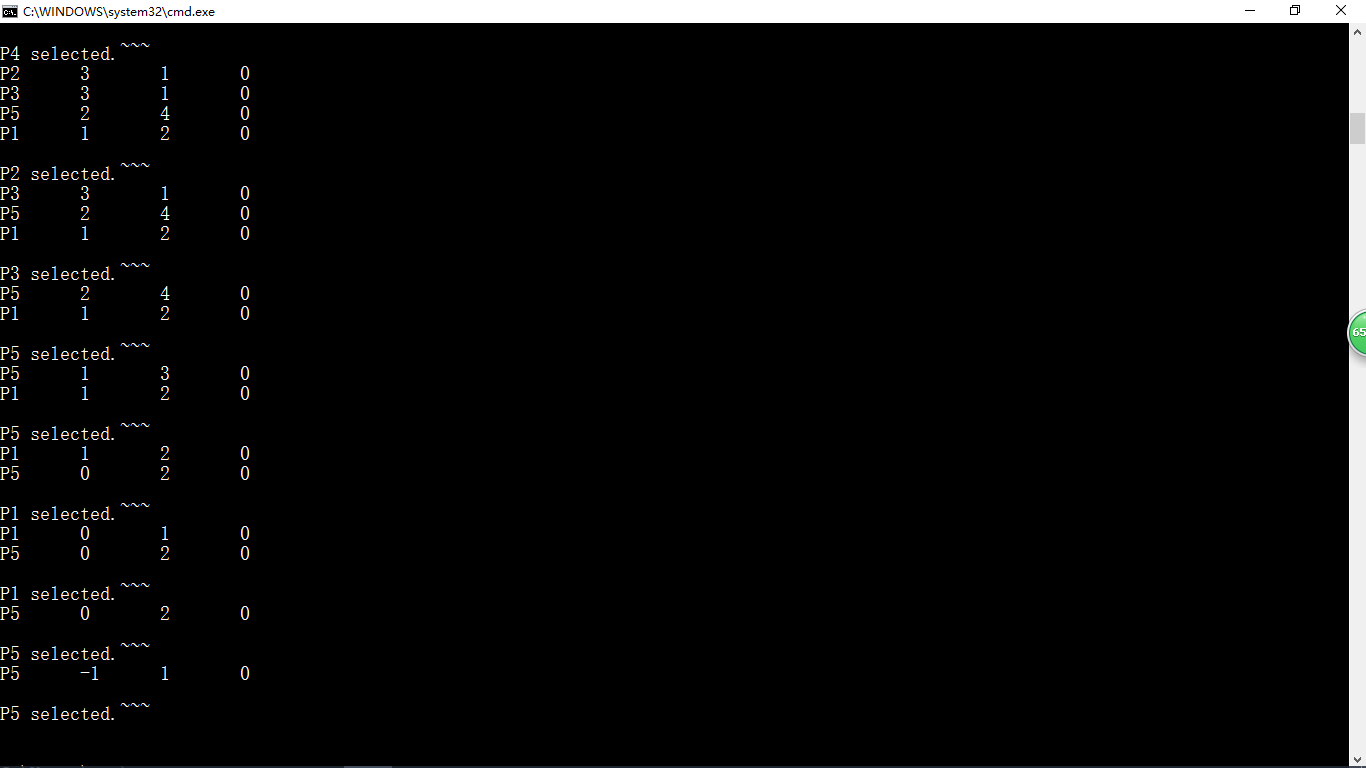
4

2

R

### 6.2进程控制块的运行状态





#### 6.2.1选中进程P2

P1

0

2

1

R

P2

k4

2

4

R

P3

k5

1

3

R

P4

k3

2

4

R

P5

k1

4

2

R

#### 6.2.2选中进程P2

P1

0

2

1

R

P2

k3

1

3

R

P3

k5

1

3

R

P4

k2

2

4

R

P5

k1

4

2

R

#### 6.2.3选中进程P4

P1

0

2

1

R

P2

k3

1

3

R

P3

k5

1

3

R

P4

k2

1

3

R

P5

k1

4

2

R

#### 6.2.4选中进程P4

P1

0

2

1

R

P2

k3

1

3

R

P3

k5

1

3

R

P4

k2

0

2

E

P5

k1

4

2

R

#### 6.2.5选中进程P2

P1

0

2

1

R

P2

k3

0

2

E

P3

k5

1

3

R

P4

k2

0

2

E

P5

k1

4

2

R

#### 6.2.6选中进程P3

P1

0

2

1

R

P2

k3

0

2

E

P3

k5

0

2

E

P4

k2

0

2

E

P5

k1

4

2

R

#### 6.2.7选中进程P5

P1

0

2

1

R

P2

k3

0

2

E

P3

k5

0

2

E

P4

k2

0

2

E

P5

k1

3

1

R

#### 6.2.8 选中进程P5

P1

0

2

1

R

P2

k3

0

2

E

P3

k5

0

2

E

P4

k2

0

2

E

P5

k1

2

0

R

#### 6.2.9选中进程P1

P1

k5

1

0

R

P2

k3

0

2

E

P3

k5

0

2

E

P4

k2

0

2

E

P5

0

2

0

R

#### 6.2.10选中进程P1

P1

k5

0

-1

E

P2

k3

0

2

E

P3

k5

0

2

E

P4

k2

0

2

E

P5

0

2

0

R

#### 6.2.11选中进程P5

P1

k5

0

-1

E

P2

k3

0

2

E

P3

k5

0

2

E

P4

k2

0

2

E

P5

0

1

-1

R

#### 6.2.12 选中进程P5

P1

k5

0

-1

E

P2

k3

0

2

E

P3

k5

0

2

E

P4

k2

0

2

E

P5

0

0

-2

E

### 6.3结果分析

运行结果正确