**石 家 庄 铁 道 大 学**

**实 验 报 告**

|  |  |  |
| --- | --- | --- |
| **课程名称**： 操作系统A | **任课教师：** 陈娜 | **实验日期**： 2021.4.19 |
| **班 级**： 信1805-2 | **姓 名**： 宋泊然 | **学 号**： 20183553 |

1. 实验目的

通过对进程调度模拟算法的实验，了解到了优先级调度算法与循环轮转调度算法，体会到了两种调度算法区别，掌握了两种调度算法的编码实现，提高了对它们的理解。

1. 实验要求

进程调度是处理机管理的核心内容。本实验要求用高级语言编写模拟进程调度程序，以便加深理解有关进程控制快、进程队列等概念，并体会和了解优先数算法和时间片轮转算法 的具体实施办法。

1. 实验过程
2. 准备
3. 查阅相关资料

c++优先队列、制表符、优先级调度算法、循环轮转调度算法

1. 初步编写程序

准备一个优先队列存储等待区，一个队列存储完成区...

1. 准备测试数据

进程数：4

标识和时间：j1 20

标识和时间：j2 50

标识和时间：j3 40

标识和时间：j4 60

进程数：4

标识和时间：j1 30

标识和时间：j2 40

标识和时间：j3 40

标识和时间：j4 50

1. 上机调试
2. 主要流程和源代码
3. 主要流程：

选择调度算法

输入进程数以及每个进程的标识符和所需时间

开始执行调度算法

如果是优先级调度算法

每执行一次，优先数减1，CPU时间片数加1，进程还需时间数减1

执行一次后若进程还需时间不为0回到优先队列，否则进入完成队列

每次执行，将当前进程输出为R态，优先队列为W态，完成队列为F态

直到优先队列为空

如果是循环轮转调度算法

每执行一次，CPU时间片数加2，进程还需时间片数减2

每次执行前判断进程还需时间是否小于一个时间片

若大于，则执行一个时间片回到队列末尾

若小于，该进程运行结束，进入完成队列，退化为FCFS算法

每次执行，将当前进程输出为R态，优先队列为W态，完成队列为F态

直到W态队列为空

1. 源代码

#include <bits/stdc++.h>

using namespace std;

int n,m;

struct pcb

{

string name;

int prio,round,cputime,needtime;

string state;

bool operator < (const pcb & a) const

{

return prio < a.prio;

}

};

priority\_queue<pcb> q;

queue<pcb> qq;

queue<pcb> qr;

void print\_0(pcb p)

{

cout<<"name \t cputime \t needtime \t priority \t state"<<endl;

cout<<p.name<<" \t "<<p.cputime<<" \t\t "<<p.needtime<<" \t\t "<<p.prio<<" \t\t "<<p.state<<endl;

}

void print\_1(priority\_queue<pcb> q1)

{

while(!q1.empty())

{

pcb p = q1.top();

q1.pop();

if (p.state == "F" || p.state == "R") continue;

print\_0(p);

}

}

void print\_2(queue<pcb> q1)

{

while(!q1.empty())

{

pcb p = q1.front();

q1.pop();

print\_0(p);

}

}

void menu()

{

cout<<"请选择算法"<<endl;

cout<<"1:选择优先数调度算法"<<endl;

cout<<"2:选择循环轮转调度算法"<<endl;

cout<<"请输入数字："<<endl;

}

void read()

{

cout<<"请输入进程的数目：";

cin>>n;

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

{

cout<<"请输入第"<<i+1<<"个进程的信息"<<endl;

cout<<"请输入该进程的标识符:";

pcb pcb1;

cin>>pcb1.name;

cout<<"请输入进程需要的运行时间:";

cin>>pcb1.needtime;

pcb1.prio = 50 - pcb1.needtime;

pcb1.state = "W";

pcb1.cputime = 0;

pcb1.round = 2;

q.push(pcb1);

qr.push(pcb1);

cout<<endl;

}

cout<<"执行前就绪队列状态"<<endl;

print\_1(q);

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*开始执行\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

}

int main(int argc, char const \*argv[])

{

menu();

cin>>m;

if (m == 1)

{

read();

while(!q.empty())

{

pcb p = q.top();

q.pop();

p.prio--;

p.cputime++;

p.needtime--;

p.state = "R";

cout<<"当前运行的进程是(即\"R\"态为)"<<endl;

print\_0(p);

cout<<"当前就绪队列的进程是(即\"W\"态为)"<<endl;

print\_1(q);

cout<<"当前运行结束的进程是(即\"F\"态为)"<<endl;

print\_2(qq);

cout<<endl;

if (p.needtime == 0)

{

p.state = "F";

qq.push(p);

}

if(p.state == "R")

{

p.state = "W";

q.push(p);

}

}

}

else if (m==2)

{

read();

while(!qr.empty())

{

pcb p = qr.front();

qr.pop();

pcb pp2;

bool flag = false;

if (p.needtime < 2) flag = true;

cout<<"当前运行的进程是(即\"R\"态为)"<<endl;

if (p.needtime < 2)

{

p.needtime = 0;

p.cputime++;

p.state = "R";

if (!qr.empty())

{

pp2 = qr.front();

qr.pop();

pp2.needtime--;

pp2.cputime++;

pp2.state = "R";

print\_0(p);

print\_0(pp2);

}

else

{

print\_0(p);

flag = false;

}

}

else

{

p.cputime += 2;

p.needtime -= 2;

p.state = "R";

print\_0(p);

}

cout<<"当前就绪队列的进程是(即\"W\"态为)"<<endl;

print\_2(qr);

cout<<"当前运行结束的进程是(即\"F\"态为)"<<endl;

print\_2(qq);

cout<<endl;

if (p.needtime <= 0)

{

p.state = "F";

qq.push(p);

}

if(p.state == "R")

{

p.state = "W";

qr.push(p);

}

if (flag)

{

if (pp2.needtime <= 0)

{

pp2.state = "F";

qq.push(pp2);

}

if(pp2.state == "R")

{

pp2.state = "W";

qr.push(pp2);

}

}

}

}

else

{

cout<<"输入有误请重试"<<endl;

return 0;

}

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*执行结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

print\_2(qq);

return 0;

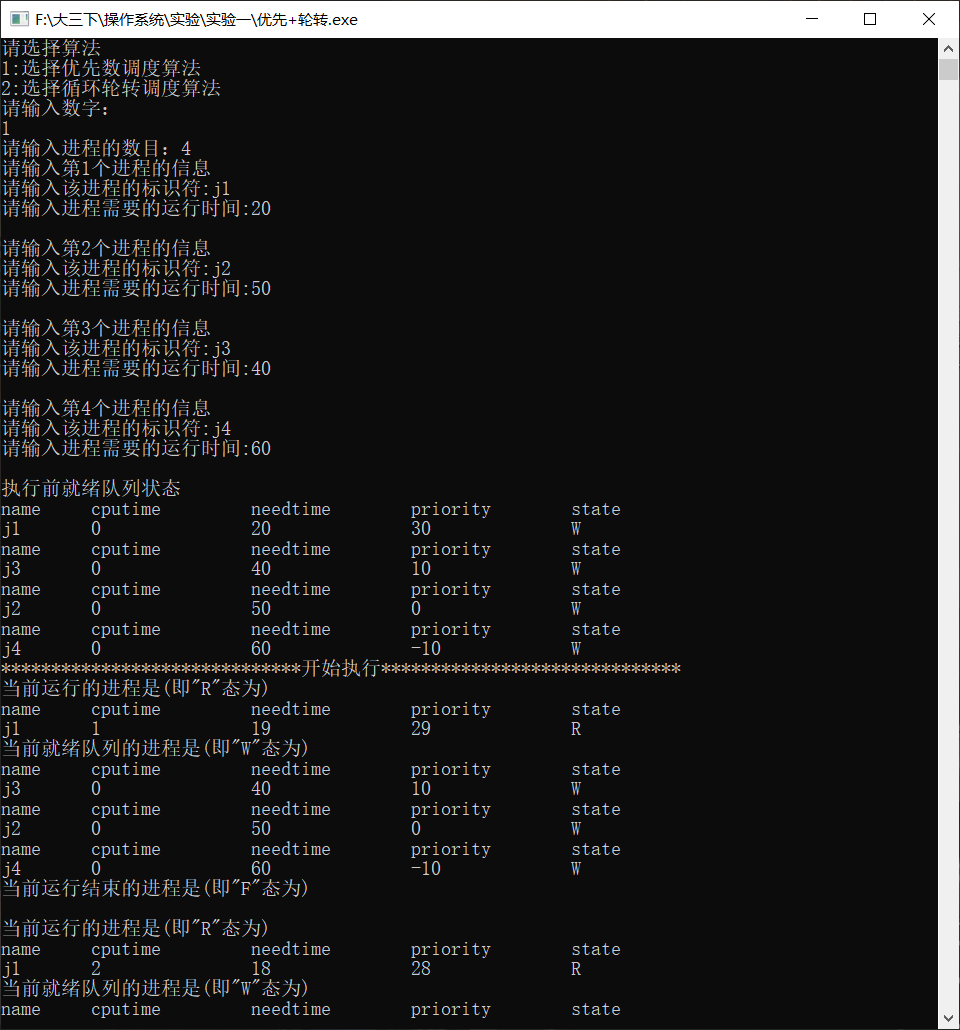
}

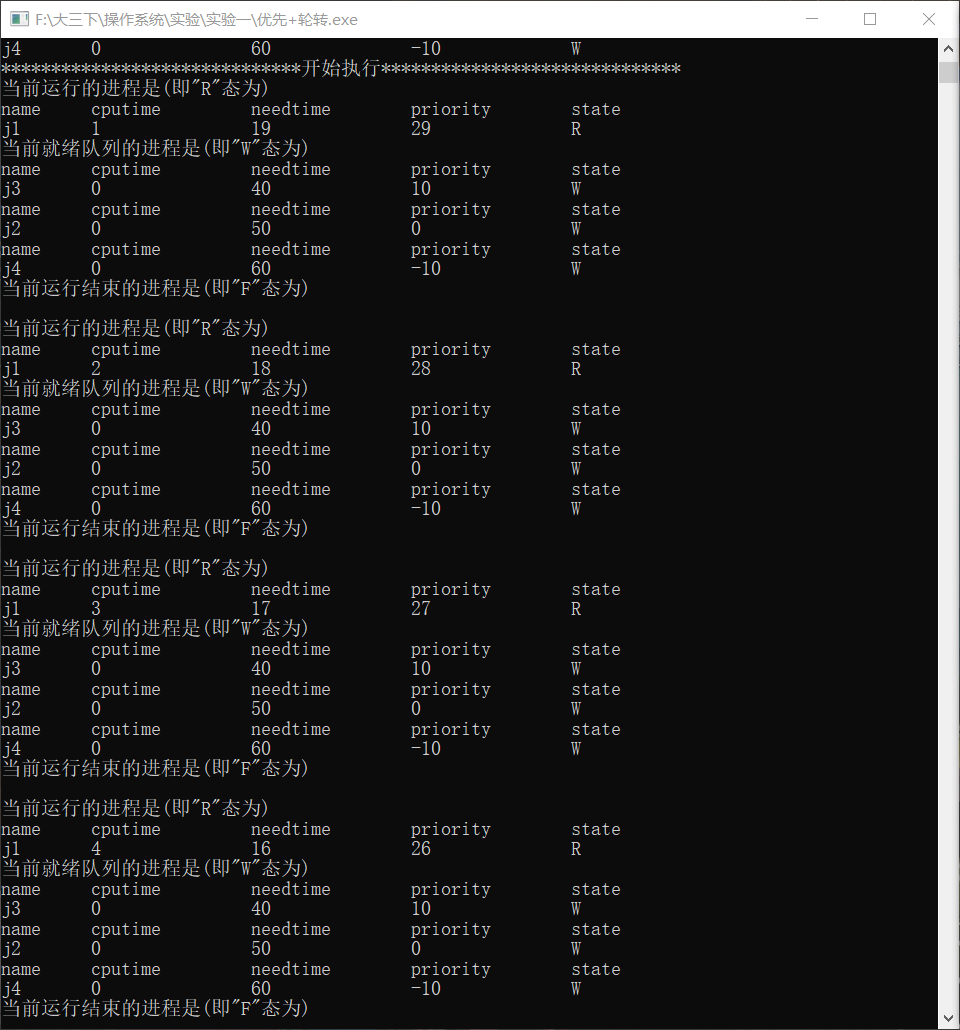
1. 遇到的主要问题和解决方法

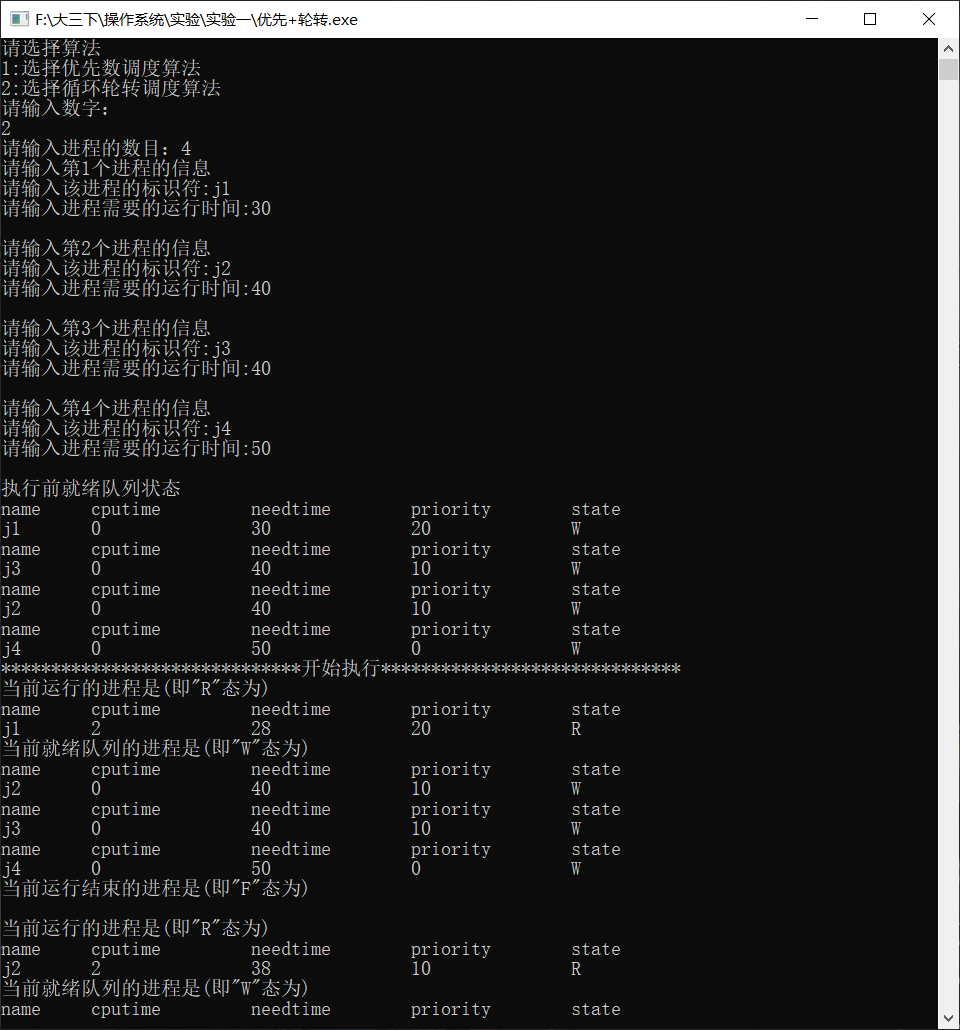
1首先自己的数据结构有些忘了，不熟悉了，需要去复习，也方便为下面的实验坐下基础。

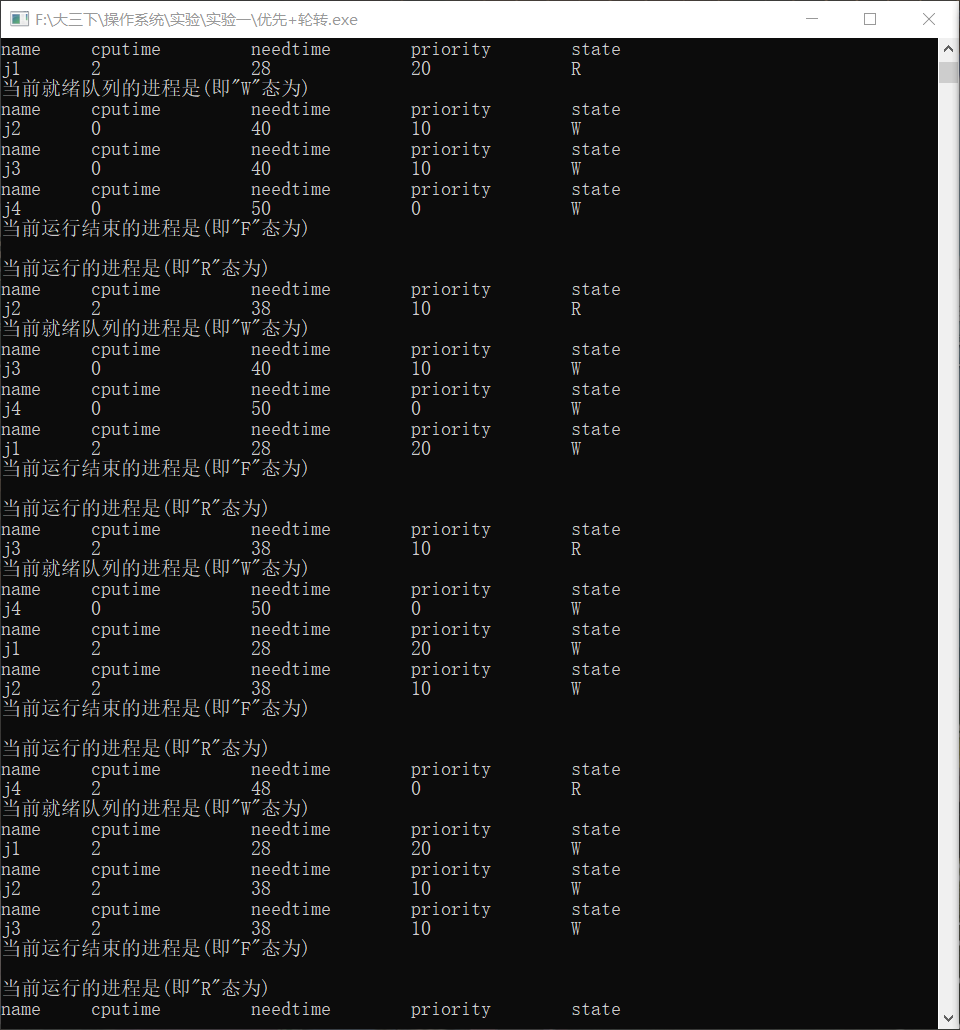
2出现了死循环的情况，是条件没有掌握好，虽然值差了一点，但是结果却很大差别

1. 实验结果









1. 实验总结

通过本次实验，我了解了两个算法之间的差别：轮转算法：优点：公平，适应快，适用于分时操作系统，缺点：高频率的进程切换，需要一定的开销，并且不区分任务紧急程度。并且不会导致饥饿现象。优先级调度算法：优点：使用于实时操作系统，可灵活调整对作业/进程的偏好程度。缺点：源源不断有高优先级到来，可能会导致届现象。并且复习了数据结构。同时也产生了一些问题：如果时间片太大或者太小会对轮转算法有何影响？而对于优先级调度算法来说，各类型的进程如何设置优先级？如何调整优先级呢？