学号 E21514033 专业 软件工程 姓名 何铭春

实验日期 教师签字 成绩

实验报告

【实验名称】 进程调度模拟实验

【实验目的】

1. 掌握进程控制块的作用和实现技术；

2. 熟悉操作系统的进程调度算法及实现方法。

【实验原理】

先来先服务的调度算法：

最简单的调度算法，既可以用于作业调度 ，也可以用于程序调度，当作业调度中采用该算法时，系统将按照作业到达的先后次序来进行调度，优先从后备队列中，选择一个或多个位于队列头部的作业，把他们调入内存，分配所需资源、创建进程，然后放入“就绪队列”,直到该进程运行到完成或发生某事件堵塞后，进程调度程序才将处理机分配给其他进程。

优先级算法：

根据进程创建时赋予的优先级进行调度，允许优先级高的进程打断优先级低的进程，让紧迫度高的程序先执行。

【实验内容】

用高级语言编写和调试一个 N个进程共行的进程调度程序。

进程调度算法：采用最高优先数优先的调度算法（即把处理机分配给优先数最高的进程）和先来先服务算法。

1.进程数据结构设计：

class process

{

public:

int ID;

int arriveTime;

int runTime;

char state;

int startTime;

int priority;

int runedTime;

int finish\_time;

process()

{

state='R';

runedTime=0;

}

};

对于先来先服务算法：

属性值：arriveTime用于记录提交时间，随机生成

startTime和runTime保存了开始执行时间和运行时间。

对于优先级算法：

Priority记录了随机生成的优先级

同样使用startTime和runTime，并且添加runedTime用于记录被打断时进程已经运行了多少。finish\_time用于记录程序运行结束时间，state记录进程是否运行结束，用于被打断进程的调用。

2.随机生成算法：

利用时间生成种子：

srand((unsigned)time(0));

//随机生成到达时间

int j;

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

{

srand(i+num);

j=rand()%200+1;

spring[i].arriveTime=j;

spring2[i].arriveTime=j;

}

//随机生成优先级

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

{

srand(34-i-num);

j=rand()%100+1;

spring[i].priority=j;

spring2[i].priority=j;

}

//随机生成运行时间

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

{

srand(11+i-num);

j=rand()%100+1;

spring[i].runTime=j;

spring2[i].runTime=j;

}

3先来先服务算法：

利用数组P按提交时间存储进程序号

//p[i]按到达顺序保存序号

int \*p=new int(pnum);

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

{

p[i]=i;

}

int temp;

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

{

for(int m=pnum-1;m>i;m--)

{

if(spring[p[m]].arriveTime<spring[p[m-1]].arriveTime)

{

temp=p[m];

p[m]=p[m-1];

p[m-1]=temp;

}

}

}

利用这个提交时间排列的P。

①做n（进程数）次循环

②按p[i]（i代表第几次）顺序比较当前时间轴时间finish\_time和这个进程到达时间。

③设置finish\_time和进程开始时间

4.抢占式优先级算法：

同样使用上面的P数组，来确定第一个到达的进程。

arriveCount：已经到达CPU队列的进程数。当所有进程到达时，控制跳出第一个循环，即不再需要判断下一个进程到达时间和优先级，只需要不断从等待队列中找未完成且优先级最高的进程运行。

finishTime：记录当前进程的完成时间，用于和下一个进程到达时间比较。

workTime：记录当前时间轴。

nowProcess：记录当前选中的进程

①选中第一个进程，相应变量赋值。

int temp;

int arriveCount=1;

int finishTime;

int nowProcess=0;

int workTime=pro2[p[0]].arriveTime;

finishTime=pro2[p[0]].arriveTime+pro2[p[0]].runTime;

②第一个循环，用于在进程未完全到达的情况下，判断下一个更替进程的时间点。

当所有进程到达时结束：

if (arriveCount==pnum)

{

break;

}

先比较下一个到达时间和当前进程完成时间

下一个到达时间比较早：

A.优先级高抢占

if (pro2[p[arriveCount]].priority>pro2[p[nowProcess]].priority)

{

pro2[p[nowProcess]].runedTime=finishTime-pro2[p[arriveCount]].arriveTime;

pro2[p[nowProcess]].runedTime=pro2[p[nowProcess]].runTime-pro2[p[nowProcess]].runedTime;

finishTime=pro2[p[arriveCount]].arriveTime+pro2[p[arriveCount]].runTime;

nowProcess=arriveCount;

cout<<nowProcess+1<<"->";

arriveCount++;

}

B.优先级低不抢占

else

{

arriveCount++;

}

完成后才到达：

需要查看当前队列情况，也要分两种情况，当期队列有进程和无进程。

else

{

//先把当前的程序完成

workTime=finishTime;

pro2[p[nowProcess]].finish\_time=workTime;

pro2[p[nowProcess]].state='C';

temp=-1;

//先在队列找一个未完成的，赋值给temp

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

{

if (pro2[p[i]].state=='R')

{

temp=i;

break;

}

}

//上面找到了，再看看有没有优先级更高的且到达的

if (temp!=-1)

{

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

{

if (pro2[p[i]].state=='R' && pro2[p[temp]].priority<pro2[p[i]].priority)

{

temp=i;

}

}

}

//没有找到，那就设置一个绝对大于进程到达时间的finishTime，让下个进程到达时间做中断点

if (temp==-1)

{

finishTime=9999999;

}

else

{

nowProcess=temp;

cout<<nowProcess+1<<"->";

finishTime=workTime+pro2[p[nowProcess]].runTime-pro2[p[nowProcess]].runedTime;

}

}

}

③最后就是进程全部在队列，只需要每次找未完成且优先级最高的进程运行即可。

while(1)

{

nowProcess=-1;

//找个未完成的进程

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

{

if (pro2[p[i]].state=='R')

{

nowProcess=i;

}

}

//如果找不到，调度完成，结束

if (nowProcess==-1)

{

break;

}

//看看有没有优先级更高的

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

{

if (pro2[p[i]].state=='R' && pro2[p[i]].priority>pro2[p[nowProcess]].priority)

{

nowProcess=i;

}

}

cout<<nowProcess+1<<"->";

workTime=workTime+pro2[p[nowProcess]].runTime-pro2[p[nowProcess]].runedTime;

pro2[p[nowProcess]].state='C';

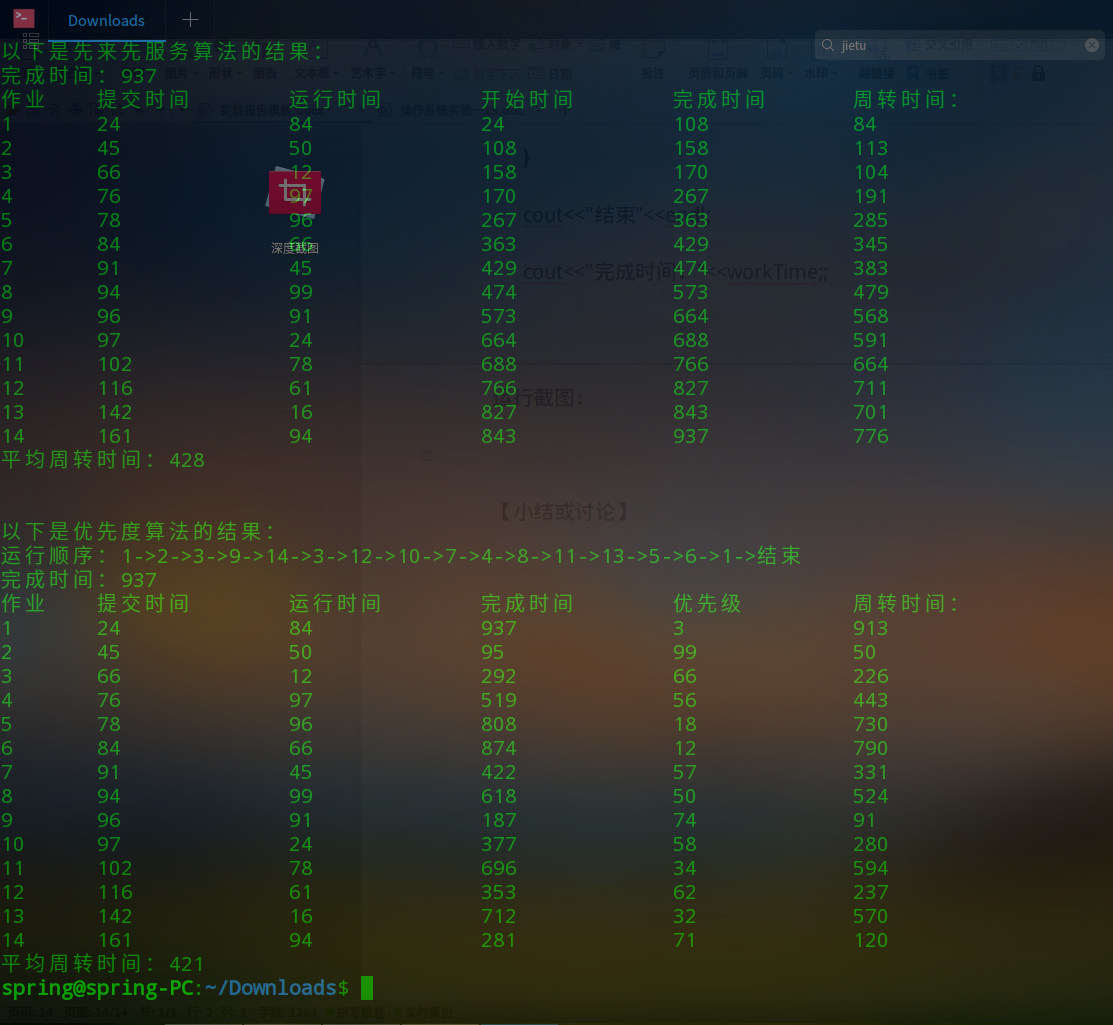
pro2[p[nowProcess]].finish\_time=workTime;

}

cout<<"结束"<<endl;

cout<<"完成时间："<<workTime;;

运行截图：



【小结或讨论】

1.不论哪种调度方式，在不计调度时间的前提下，总完成时间相等。

2.当优先级高的进程又运行时间短时，优先级算法的平均周转时间要小于先来先服务算法。

3.采用了条件判断的抢占式算法，判断下次中断时间。

附录：源代码

#include <iostream>

#include <stdlib.h>

#include <ctime>

using namespace std;

/\* run this program using the console pauser or add your own getch, system("pause") or input loop \*/

class process

{

public:

int ID;

int arriveTime;

int runTime;

char state;

int startTime;

int priority;

int runedTime;

int finish\_time;

process()

{

state='R';

runedTime=0;

}

};

int FCFS(process\* pro,int \*p,int pnum)

{

int finish\_time=0;

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

{

if(finish\_time<pro[p[i]].arriveTime)

{

finish\_time=pro[p[i]].arriveTime;

finish\_time=finish\_time+pro[p[i]].runTime;

pro[p[i]].startTime=pro[p[i]].arriveTime;

pro[p[i]].state='C';

}

else

{

pro[p[i]].startTime=finish\_time;

pro[p[i]].state='C';

finish\_time=finish\_time+pro[p[i]].runTime;

}

}

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

{

pro[p[i]].finish\_time=pro[p[i]].startTime+pro[p[i]].runTime;

}

cout<<"完成时间："<<finish\_time<<endl;

return 0;

}

int PSA(process\* pro2,int \*p,int pnum)

{

int temp;

int arriveCount=1;

int finishTime;

int nowProcess=0;

int workTime=pro2[p[0]].arriveTime;

finishTime=pro2[p[0]].arriveTime+pro2[p[0]].runTime;

cout<<"运行顺序："<<nowProcess+1<<"->";

while(1)

{

if (arriveCount==pnum)

{

break;

}

if (pro2[p[arriveCount]].arriveTime<finishTime)

{

workTime=pro2[p[arriveCount]].arriveTime;

if (pro2[p[arriveCount]].priority>pro2[p[nowProcess]].priority)

{

pro2[p[nowProcess]].runedTime=finishTime-pro2[p[arriveCount]].arriveTime;

pro2[p[nowProcess]].runedTime=pro2[p[nowProcess]].runTime-pro2[p[nowProcess]].runedTime;

finishTime=pro2[p[arriveCount]].arriveTime+pro2[p[arriveCount]].runTime;

nowProcess=arriveCount;

cout<<nowProcess+1<<"->";

arriveCount++;

}

else

{

arriveCount++;

}

}

else

{

workTime=finishTime;

pro2[p[nowProcess]].finish\_time=workTime;

pro2[p[nowProcess]].state='C';

temp=-1;

//先在队列找一个未完成的，赋值给temp

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

{

if (pro2[p[i]].state=='R')

{

temp=i;

break;

}

}

//上面找到了，再看看有没有优先级更高的且到达的

if (temp!=-1)

{

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

{

if (pro2[p[i]].state=='R' && pro2[p[temp]].priority<pro2[p[i]].priority)

{

temp=i;

}

}

}

//没有找到，那就设置一个绝对大于进程到达时间的finishTime，让下个进程到达时间做中断点

if (temp==-1)

{

finishTime=9999999;

}

else

{

nowProcess=temp;

cout<<nowProcess+1<<"->";

finishTime=workTime+pro2[p[nowProcess]].runTime-pro2[p[nowProcess]].runedTime;

}

}

}

pro2[p[nowProcess]].finish\_time=finishTime;

pro2[p[nowProcess]].state='C';

workTime=finishTime;

nowProcess=-1;

while(1)

{

nowProcess=-1;

//找个未完成的进程

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

{

if (pro2[p[i]].state=='R')

{

nowProcess=i;

}

}

//如果找不到，调度完成，结束

if (nowProcess==-1)

{

break;

}

//看看有没有优先级更高的

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

{

if (pro2[p[i]].state=='R' && pro2[p[i]].priority>pro2[p[nowProcess]].priority)

{

nowProcess=i;

}

}

cout<<nowProcess+1<<"->";

workTime=workTime+pro2[p[nowProcess]].runTime-pro2[p[nowProcess]].runedTime;

pro2[p[nowProcess]].state='C';

pro2[p[nowProcess]].finish\_time=workTime;

}

cout<<"结束"<<endl;

cout<<"完成时间："<<workTime;

return 0;

}

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

{

//由运行时间随机生成一个数，用于做种子

cout<<"请输入进程数："<<endl;

int pnum;

cin>>pnum;

srand((unsigned)time(0));

int num=rand()%10+1;

//为ID赋值

process \*spring=new process[pnum];

process \*spring2=new process[pnum];

cout<<"下面是两种算法比较"<<endl;

//按顺序生成ID

for(int m=0;m<pnum;m++)

{

spring[m].ID=m;

spring2[m].ID=m;

}

//随机生成到达时间

int j;

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

{

srand(i+num);

j=rand()%200+1;

spring[i].arriveTime=j;

spring2[i].arriveTime=j;

}

//随机生成优先级

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

{

srand(34-i-num);

j=rand()%100+1;

spring[i].priority=j;

spring2[i].priority=j;

}

//随机生成运行时间

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

{

srand(11+i-num);

j=rand()%100+1;

spring[i].runTime=j;

spring2[i].runTime=j;

}

//以上为进程类赋值

//p[i]按到达顺序保存序号

int \*p=new int(pnum);

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

{

p[i]=i;

}

int temp;

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

{

for(int m=pnum-1;m>i;m--)

{

if(spring[p[m]].arriveTime<spring[p[m-1]].arriveTime)

{

temp=p[m];

p[m]=p[m-1];

p[m-1]=temp;

}

}

}

cout<<"请选择程序功能："<<endl;

cout<<"1.先来先服务算法"<<endl;

cout<<"2.优先级算法"<<endl;

cout<<"3.两种算法同步比较"<<endl;

int chooseFor;

cin>>chooseFor;

if (chooseFor==1)

{

int sum=0;

cout<<"以下是先来先服务算法的结果："<<endl;

FCFS(spring,p,pnum);

cout<<"作业 提交时间 运行时间 开始时间 完成时间 周转时间："<<endl;

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

{

cout<<i+1<<'\t'<<spring[p[i]].arriveTime<<'\t'<<'\t'<<spring[p[i]].runTime<<'\t'<<'\t'<<spring[p[i]].startTime

<<'\t'<<'\t'<<spring[p[i]].finish\_time<<'\t'<<'\t'<<spring[p[i]].finish\_time-spring[p[i]].arriveTime<<endl;

}

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

{

sum=sum+spring[p[i]].finish\_time-spring[p[i]].arriveTime;

}

cout<<"平均周转时间："<<sum/pnum<<endl;

}

if (chooseFor==2)

{

cout<<endl;

cout<<"以下是优先度算法的结果："<<endl;

PSA(spring2,p,pnum);

cout<<'\n'<<"作业 提交时间 运行时间 完成时间 优先级 周转时间："<<endl;

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

{

cout<<i+1<<'\t'<<spring2[p[i]].arriveTime<<'\t'<<'\t'<<spring2[p[i]].runTime<<'\t'<<'\t'<<spring2[p[i]].finish\_time<<'\t'<<'\t'<<spring2[p[i]].priority<<'\t'<<'\t'<<spring2[p[i]].finish\_time-spring2[p[i]].arriveTime<<endl;

}

int sum=0;

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

{

sum=sum+spring2[p[i]].finish\_time-spring2[p[i]].arriveTime;

}

cout<<"平均周转时间："<<sum/pnum<<endl;

}

if (chooseFor==3)

{

int sum=0;

cout<<"以下是先来先服务算法的结果："<<endl;

FCFS(spring,p,pnum);

cout<<"作业 提交时间 运行时间 开始时间 完成时间 周转时间："<<endl;

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

{

cout<<i+1<<'\t'<<spring[p[i]].arriveTime<<'\t'<<'\t'<<spring[p[i]].runTime<<'\t'<<'\t'<<spring[p[i]].startTime

<<'\t'<<'\t'<<spring[p[i]].finish\_time<<'\t'<<'\t'<<spring[p[i]].finish\_time-spring[p[i]].arriveTime<<endl;

}

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

{

sum=sum+spring[p[i]].finish\_time-spring[p[i]].arriveTime;

}

cout<<"平均周转时间："<<sum/pnum<<endl;

cout<<endl;

cout<<endl;

cout<<"以下是优先度算法的结果："<<endl;

PSA(spring2,p,pnum);

cout<<'\n'<<"作业 提交时间 运行时间 完成时间 优先级 周转时间："<<endl;

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

{

cout<<i+1<<'\t'<<spring2[p[i]].arriveTime<<'\t'<<'\t'<<spring2[p[i]].runTime<<'\t'<<'\t'<<spring2[p[i]].finish\_time<<'\t'<<'\t'<<spring2[p[i]].priority<<'\t'<<'\t'<<spring2[p[i]].finish\_time-spring2[p[i]].arriveTime<<endl;

}

sum=0;

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

{

sum=sum+spring2[p[i]].finish\_time-spring2[p[i]].arriveTime;

}

cout<<"平均周转时间："<<sum/pnum<<endl;

}

return 0;

}