

**计算机与信息 学院实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验课程： | 操作系统实验 | | | | |
| 实验编号： | 实验一 | | | | |
| 实验名称： | 进程状态迁移 | | | | |
| 实验人员： | 学号 | 18111207248 | | | |
| 姓名 | 吴钰 | | | |
| 班级 | 18计算机科学技术创新班 | | | |
| 实验日期： | 2020.4.12 | | | | |
| 实验室： |  | | | | |
|  |  | | | | |
| 实验评价： |  | | | | |
| 实验成绩： | |  | 评价日期： |  |
|  | 指导教师： | |  | | |

# 一、实验目的

## 1. 写一个程序描述进程状态迁移过程，理解进程概念、状态转换及其控制

## 二、实验要求

# 1) 提供导致进程状态变化的调用接口，包括创建、删除、调度、阻塞、时间到、挂 起、激活等。

# 2) 实现进程列表显示的接口。

# 3) 这里设计的进程是一个假设的对象实体，是由程序自己创建和删除，不是系统维 护的进程

# 三、设计

【定义所有抽象数据类型，自定义函数的功能详细描述（伪代码表示），以及主程序的流程图。】

# #include <cstdio>

# #include <iostream>

# #include <cstdlib>

# #include <queue>

# using namespace std;

# enum status{success,failed};

# enum Jstate{ready,run,block,death}; //进程的三个状态和消亡

# typedef struct PCB{

# int id;

# char name[20];

# Jstate state;

# int timebin;

# int grade;

# int key;//?

# }PCB;

# queue<PCB> q; //就绪队列

# PCB p\_run; //运行中的进程

# int p\_flag=0;// 为零表示没有正在执行的进程

# queue<PCB> Qblock; //阻塞队列

# queue<PCB> Qdeath;//消亡队列

# status addtoReady(PCB a)//向就绪队列添加进程

# {

# q.push(a);

# return success;

# 

# }

# status Init()//初始化队列

# {

# int n,i,num,max; //用来优先级入队

# PCB t,a[100];

# cout<<"输入进程个数：\n";

# cin>>n;

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

# {

# a[i].id=i;

# a[i].state=ready;

# printf("输入第%d个进程名称、运行的时间片、优先级数:\n",i+1);

# scanf("%s",a[i].name);

# cin>>a[i].timebin;

# cin>>a[i].grade;

# }

# cout<<"开始"<<endl;

# for(num=0;num<n;num++) //n个进程按照优先级大小排序入队

# {

# max=-100;

# for(i=num;i<n;i++)

# {

# if(max<a[i].grade)

# {

# max=a[i].grade;

# t=a[num];

# a[num]=a[i];

# a[i]=t;

# }

# }

# addtoReady(a[num]);

# }

# return success;

# }

# status wakeUp() //将阻塞队列中的进程唤醒

# {

# int flag=0;

# PCB p;

# 

# p=Qblock.front();//阻塞队列中第一个进程

# p.state=ready; //改状态

# printf("唤醒进程%d %s\n",p.id,p.name);

# addtoReady(p);

# Qblock.pop();

# return success;

# }

# status RuntoBlock(PCB j) //从运行态变成阻塞态

# {

# j.state=block;

# Qblock.push(j);

# return success;

# }

# status RuntoDeath(PCB j) //进入消亡队列

# {

# j.state=death;

# Qdeath.push(j);

# return success;

# }

# status RunJob() //执行进程

# {

# 

# char S1;

# if(p\_flag==0)//无正在运行的进程

# {

# 

# if(q.size()==0)

# {

# cout<<"当前无就绪进程!"<<endl;

# if(p\_flag==0)

# {

# printf("运行结束...\n");

# return success;

# }

# }

# 

# PCB job=q.front(); //取就绪队列队头元素

# q.pop(); //队头元素出队

# if(job.timebin>0) //时间片未用完，继续执行

# {

# job.grade--;

# printf("%s开始进入执行状态...\n当前所需时间片：%d\n",job.name,job.timebin);

# job.timebin--;

# if(job.timebin<=0){

# cout<<"执行完成后该进程已消亡.."<<endl;

# RuntoDeath(job);

# p\_flag=0;//?????

# }

# if(job.timebin>0)

# {

# job.state=run; //修改状态

# p\_run=job; //运行中的进程

# p\_flag=1;}

# if(Qblock.size()!=0)//执行一次时间片后，看是否需要唤醒阻塞队列的进程

# {

# printf("是否唤醒阻塞的进程？（输入Y/N）\n");

# cin>>S1;

# if(S1=='Y')

# {

# wakeUp();

# return success;

# }

# }

# 

# }

# else if(job.timebin<=0&&job.state!=death)//时间片用完

# {

# cout<<"该进程时间片用完，进入消亡队列！"<<endl;

# RuntoDeath(job);//放到消亡队列

# }

# return success;

# }

# //已有正在执行的进程

# char S2;

# printf("是否阻塞（Y/N）ID:%d name:%s进程？\n",p\_run.id,p\_run.name);

# cin>>S2;

# if(S2=='Y')//将原来正在执行的进程阻塞

# {

# RuntoBlock(p\_run);

# printf("ID:%d name:%s 进入阻塞队列\n",p\_run.id,p\_run.name);

# //阻塞完之后再将就绪队列第一个进程调出执行

# if(q.size()==0)

# {

# cout<<"当前无就绪进程!"<<endl;

# if(p\_flag==0)

# {

# printf("运行结束...\n");

# return success;

# }

# else

# wakeUp();//

# }

# PCB job=q.front(); //取就绪队列队头元素

# q.pop(); //队头元素出队

# if(job.timebin>0) //时间片未用完，继续执行???

# {

# job.grade--;

# //job.timebin--;

# printf("%s开始进入执行状态...\n当前所需时间片：%d\n",job.name,job.timebin);

# job.timebin--;

# if(job.timebin<=0&&job.state!=death){

# cout<<"执行完成后该进程已消亡.."<<endl;

# RuntoDeath(job);

# p\_flag=0;//

# }

# if(job.timebin>0){

# job.state=run; //修改状态

# p\_run=job; //运行中的进程

# p\_flag=1;}

# 

# if(Qblock.size()!=0)//执行一次时间片后，看是否需要唤醒阻塞队列的进程

# {

# printf("是否唤醒阻塞的进程？（输入Y/N）\n");

# cin>>S1;

# if(S1=='Y')

# {

# wakeUp();

# return success;

# }

# }

# else if(job.timebin<=0&&job.state!=death)//时间片用完

# {

# cout<<"时间片已用完，进入消亡队列！"<<endl;

# RuntoDeath(job);//放到消亡队列

# }

# return success;

# }

# }

# 

# p\_run.timebin--;

# if(p\_run.timebin==0&&p\_run.state!=death)//消亡

# {

# printf("ID:%d name:%s 时间片已用完,进入消亡队列\n",p\_run.id,p\_run.name);

# RuntoDeath(p\_run);

# p\_flag=0;

# }

# return success;

# 

# }

# status Show(queue<PCB> temp)

# {

# cout<<temp.size()<<endl;

# PCB p;

# while(temp.size()!=0)

# {

# p=temp.front();

# if(p.state==death)

# printf("进程%d\t进程名:%s\t运行时间片：%d\t优先级数：%d\n",p.id,p.name,p.timebin,p.grade);

# temp.pop();

# }

# cout<<endl;

# return success;

# }

# int main()

# {

# int x;

# Init();

# do

# {

# cout<<"请选择操作：（1：执行进程 2：输出就绪队列 3：输出阻塞队列 4：输出消亡队列 5：结束） "<<endl;

# cin>>x;

# switch(x)

# {

# case 1: RunJob();break;

# case 2: Show(q);break;

# case 3: Show(Qblock);break;

# case 4: Show(Qdeath);break;

# case 5: return 0;break;

# }

# if(q.size()==0&&Qblock.size()==0&&p\_flag==0)

# {

# cout<<"所有进程均结束"<<endl;

# break;

# }

# }while(1);

# cout<<"就绪队列进程："<<endl;

# Show(q);

# cout<<"阻塞队列进程："<<endl;

# Show(Qblock);

# cout<<"消亡队列进程："<<endl;

# Show(Qdeath);

# return 0;

# }

# 四、调试分析和思考

【(1) 调试过程中遇到的问题是如何解决的以及对设计与实现中关键点的回顾讨论和分析；(2) 算法的时空分析(包括基本操作和其他算法的时间复杂度和空间复杂度的分析)和改进设想；(3) 经验和体会等。】

遇到的问题：

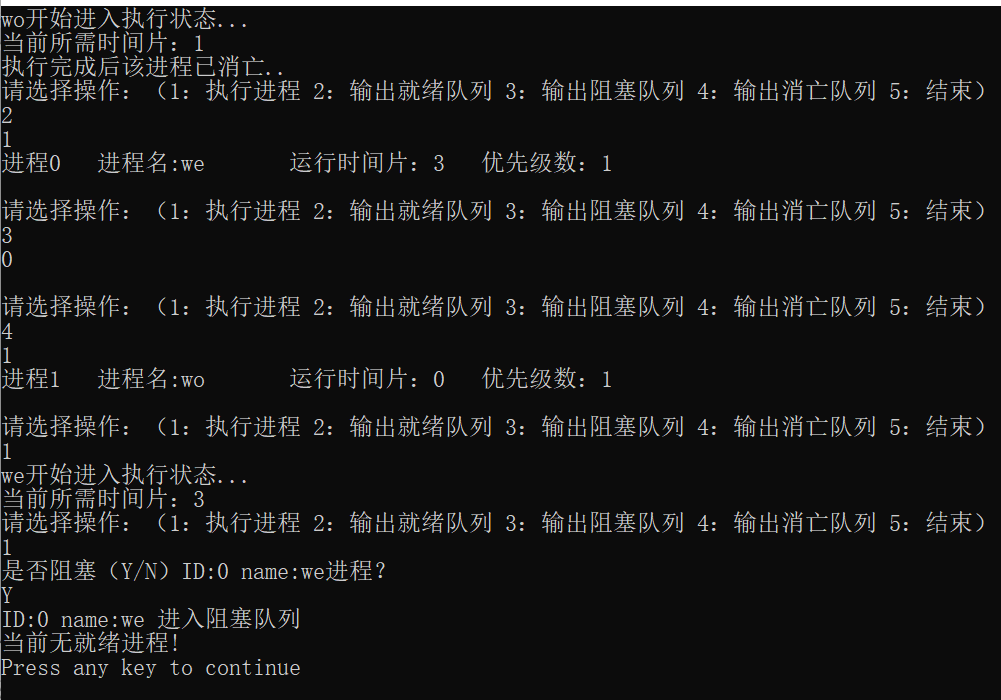
1. 只剩下一个进程时，怎么反复调度一个进程执行而不是让他进入阻塞状态
2. 如何实现挂起和激活操作
3. 简化程序的执行时的状态，我在执行函数里写的过于复杂

4、我采用的方法是在每次选择操作时，判断当前是否有正在执行的进程，如果有则用户判断是否应当阻塞当前进程。也可以更改为，每次执行完以后，该进程直接进入阻塞或就绪队列，每次操作时默认没有正在运行的程序

# 五、测试数据与结果

【列出你的测试结果，明确输入和输出数据。测试数据应该完整和严格，可以直接贴结果图。】

# C:\Users\wuyu\Documents\Tencent Files\1124274751\FileRecv\MobileFile\Image\]WD5NR[{{T$W0W70OZ]`8NF.png

****