组长：xxxx

课设1：进程状态转换的模拟

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 调研 | 设计 | 编码 | 测试 | 报告做成 | 对整个项目的贡献度 |
| xxx |  |  | √ |  |  | 40% |
| yyy |  | √ |  | √ |  | 20% |
| zzz |  |  |  |  | √ | 20% |
| bbb | √ |  |  |  |  | 20% |

**课设一：进程状态转换的模拟**

1. **课设目标：**

(1)了解 Linux进程状态的转化

(2)学习引入挂起状态进程转换的原理

(3)学会编写简单的引入挂起状态进程转换的方法

1. **课设内容：**

完成进程控制块结构体的编写,创建就绪队列、阻塞队列、运行队列、完成队列、挂起队列,初始化操作系统的原始进程：可静态设置也可动态添加,编写新建进程函数,系统开始调度,当前运行进程请求 I/O 事件,某进程 I/O 完成时间片到期其他,阻塞事件挂起,完成先来先服务算法

1. **预期结果**

实现写新建进程函数,系统开始调度,当前运行进程请求 I/O 事件,某进程 I/O 完成时间片到期其他,阻塞事件挂起,完成先来先服务算法并完成下面六态的转化

1）活动阻塞态挂起→静止阻塞态

2）静止阻塞态激活→活动阻塞态

3）静止阻塞态→静止就绪态

4）静止就绪态激活→活动就绪态

5）活动就绪态挂起→静止就绪态

6）运行态挂起→静止就绪态

# 背景描述

进程挂起（也称换出）是指在内存中的进程被暂时移出保存到外存中（如磁盘）的过程。当某个进程被挂起时，若被挂起的进程处于运行状态则停止执行；若被挂起的处于就绪状态则暂时不参加进程调度。引起进程挂起原因大致有以下三种。

（1）用户的请求。

（2）父进程的请求。

（3）操作系统的原因。

# 系统设计

图示

描述已自动生成

# 算法

。。。。。。

# 详细设计

1. ypedef struct node  
   {  
   char name[20]; //进程名  
   int id = 0; //进程ID  
   int priority = 0; //优先级  
   int ArriTime = 0; //进程到达时间  
   int serviceTime=0; //进程总需要时间  
   int needTime = 0; //还需要运行时间  
   int CPUTime = 0; //已用CPU时间  
   char state[20]; //进程状态 1.Ready 2.Run 3.Wait 4.Finish  
   int finishTime = 0; //完成时间  
   char blockCause[30]; //阻塞原因  
   int aroundTime=0; //周转时间  
   } PCB;  
   2） 创建就绪队列、阻塞队列、运行队列：采用队列的方式/其他方式也可以  
   思路：用队列的方式  
   代码：  
   queue processQueue, readyQueue, blockQueue, runQueue, finishQueue,hungQueue;//临时队列、就绪队列、阻塞队列、运行队列、完成队列、挂起队列  
   3）编写创建进程函数：  
   思路：分为手动输入还是自动输入，手动输入为用户自己创建，自动输入为调入我自己初始化的进程。  
   代码：  
   //初始化进程  
   void Init()  
   {  
   PCB p1, p2, p3,p4,p5;  
   strcpy(p1.name, “a”);  
   p1.id = 1;  
   p1.priority = 8;  
   p1.ArriTime = 2;  
   p1.needTime = 4;  
   p1.serviceTime=4;  
   processQueue.push(p1);  
   strcpy(p2.name, “b”);  
   p2.id = 2;  
   p2.priority = 4;  
   p2.ArriTime = 6;  
   p2.needTime = 4;  
   p2.serviceTime=4;  
   processQueue.push(p2);  
   strcpy(p3.name, “c”);  
   p3.id = 3;  
   p3.priority = 7;  
   p3.ArriTime = 3;  
   p3.needTime = 9;  
   p3.serviceTime=9;  
   processQueue.push(p3);  
   strcpy(p4.name, “d”);  
   p4.id = 4;  
   p4.priority = 1;  
   p4.ArriTime = 0;  
   p4.needTime = 1;  
   p4.serviceTime=1;  
   processQueue.push(p4);  
   strcpy(p5.name, “e”);  
   p5.id = 5;  
   p5.priority = 2;  
   p5.ArriTime = 2;  
   p5.needTime = 2;  
   p5.serviceTime=2;  
   processQueue.push(p5);  
   }
2. //添加进程  
   void addProcess()  
   {  
   int n;  
   cout << “请输入要添加的进程数：” << endl;  
   cin >> n;  
   for (int i = 0; i < n; i++)  
   {  
   PCB p;  
   cout << “请输入第”<<i+1<<“个进程名、进程优先级、进程到达时间、进程需要运行时间:” << endl;  
   cin >> p.name>>p.priority>> p.ArriTime>> p.serviceTime;  
   p.id=i+1;  
   p.needTime=p.serviceTime;  
   processQueue.push§;  
   }  
   }  
   2、 进程在其生命周期的都处于不断变换的过程，处理至少四种事件：（请文字写明思路并给出核心代码）
3. 设置调度算法  
   思路：设置菜单，让用户自己选择使用算法，设置调度算法。  
   代码：  
   int e;  
   cout << “请输入算法” << endl;  
   cout << “1.先来先服务 2.优先级抢占 3.时间片轮转” << endl;  
   cin >> e;  
   switch (e)  
   {  
   case 1:  
   FCFS();  
   break;  
   case 2:  
   SJF();  
   break;  
   case 3:  
   RR();  
   break;  
   }
4. 调度开始  
   思路：调度开始，程序先判断运行队列是否为空，如果为空，就将就绪队列的队头元素压入运行队列中，如果就绪队列为空，则判断阻塞队列是否为空，再判断挂起队列是否为空，若都为空则结束程序。  
   代码：  
   //系统开始调度  
   void beginDispatch()  
   {  
   if (runQueue.empty())  
   {  
   if (!readyQueue.empty())  
   {  
   strcpy(readyQueue.front().state,“Run”);  
   runQueue.push(readyQueue.front());  
   readyQueue.pop();  
   runQueue.front().CPUTime++;  
   runQueue.front().needTime–;  
   time++;  
   }  
   else if(!blockQueue.empty())  
   {  
   char chance=‘p’;  
   cout<<endl;  
   cout<<“就绪队列不为\*\*\*\*\*\*\*\*\*\*\*\*\*”<<endl;  
   cout<<“若结束序请按y,若不结束请按n"<<endl;  
   cin>>chance  
   if(chance=‘y’)  
   {  
   finish = true;  
   return;  
   }  
   }  
   else if(!hungQueue.empty())  
   {  
   char chance=‘p’;  
   cout<<endl;  
   cout<<"挂起队列不为空\*\*\*\*\*\*\*\*\*\*\*”<<endl;  
   cout<<"\*\*若结束程序请按y,若不结束请按n"<<endl;  
   cin>>chance;  
   if(chance==‘y’)  
   {  
   finish = true;  
   return;  
   }  
   }  
   else  
   {  
   finish = true;  
   return;  
   }  
   }  
   else if (!runQueue.empty())  
   {  
   if (runQueue.front().needTime == 0)  
   {  
   runQueue.front().finishTime = time;  
   strcpy(runQueue.front().state,“Finish”);  
   finishQueue.push(runQueue.front());  
   runQueue.pop();  
   if (!readyQueue.empty())  
   {  
   strcpy(readyQueue.front().state,“Run”);  
   runQueue.push(readyQueue.front());  
   readyQueue.pop();  
   }  
   }  
   //先来先服务  
   if (algorithm == 1)  
   {  
   time++;  
   runQueue.front().CPUTime++;  
   runQueue.front().needTime–;  
   }  
   //优先级  
   if (algorithm == 2)  
   {  
   time++;  
   runQueue.front().CPUTime++;  
   runQueue.front().needTime–;  
   }  
   //时间片轮转算法  
   if(algorithm == 3)  
   {  
   time++;  
   runQueue.front().CPUTime++;  
   runQueue.front().needTime–;  
   if (runQueue.front().needTime != 0)  
   {  
   if ((runQueue.front().CPUTime >= timeslice) &&(runQueue.front().CPUTime % timeslice == 0))  
   {  
   strcpy(runQueue.front().state,“Ready”);  
   readyQueue.push(runQueue.front());  
   runQueue.pop();  
   if (!readyQueue.empty())  
   {  
   strcpy(readyQueue.front().state,“Run”);  
   runQueue.push(readyQueue.front());  
   readyQueue.pop();  
   }  
   }  
   }  
   }  
   }
5. CPU运行时间  
   思路：用一个全局变量time计算CPU的运行时间，每运行一次time++；
6. I/O事件  
   思路：若用户在菜单键选择请求I/O事件，先判断运行队列是否为空，若为空则提示当前无进程运行，反之则将当前正在运行的进程压入阻塞队列，状态改为“wait”，阻塞原因改为“请求I/O事件”。  
   代码：  
   //当前运行进程请求 I/O 事件  
   void IORequest()  
   {  
   if (!runQueue.empty())  
   {  
   if (runQueue.front().needTime != 0)  
   {  
   strcpy(runQueue.front().blockCause, “请求 I/O 事件”);  
   strcpy(runQueue.front().state,“Wait”);  
   blockQueue.push(runQueue.front());  
   runQueue.pop();  
   }  
   }  
   else  
   {  
   cout << “当前无运行进程” << endl;  
   }  
   }
7. I/O完成  
   思路：若用户在菜单键选择I/O完成，先判断阻塞队列是否为空，若为空则提示当前无进程阻塞，反之用户输入I/O完成的队列的id，遍历一遍阻塞队列中id与输入id匹配的进程，则将当前正在运行的进程压入阻塞队列，状态改为“wait”，阻塞原因改为“请求I/O事件”。  
   代码：  
   //某进程I/O完成  
   void IOFinish()  
   {  
   //空  
   if (blockQueue.empty())  
   {  
   cout << “目前无阻塞进程” << endl;  
   }  
   //不空  
   else  
   {  
   int id;  
   cout << “请输入I/O完成的进程ID” << endl;  
   cin >> id;  
   for (int i = 0; i < blockQueue.size(); i++)  
   {  
   if (blockQueue.front().id == id)  
   {  
   strcpy(blockQueue.front().blockCause, " ");  
   strcpy(blockQueue.front().state,“Ready”);  
   readyQueue.push(blockQueue.front());  
   blockQueue.pop();  
   if (algorithm == 1)//先来先服务  
   {  
   //排序  
   PCB temp[readyQueue.size()];  
   int n = 0;  
   while (!readyQueue.empty())  
   {  
   temp[n] = readyQueue.front();  
   readyQueue.pop();  
   n++;  
   }  
   sort(temp, temp + n, cmp1());  
   for (int i = 0; i < n; i++)  
   {  
   readyQueue.push(temp[i]);  
   }  
   }  
   if (algorithm == 2)//优先级  
   { if (algorithm == 1)//先来先服务  
   {  
   //排序  
   PCB temp[readyQueue.size()];  
   int n = 0;  
   while (!readyQueue.empty())  
   {  
   temp[n] = readyQueue.front();  
   readyQueue.pop();  
   n++;  
   }  
   sort(temp, temp + n, cmp1());  
   for (int i = 0; i < n; i++)  
   {  
   readyQueue.push(temp[i]);  
   }  
   //排序  
   PCB temp[readyQueue.size()];  
   int n = 0;  
   while (!readyQueue.empty())  
   {  
   temp[n] = readyQueue.front();  
   readyQueue.pop();  
   n++;  
   }  
   sort(temp, temp + n, cmp2());  
   for (int i = 0; i < n; i++)  
   {  
   readyQueue.push(temp[i]);  
   }  
   }  
   if (algorithm == 3)//时间片轮转  
   {  
   //排序  
   PCB temp[readyQueue.size()];  
   int n = 0;  
   while (!readyQueue.empty())  
   {  
   temp[n] = readyQueue.front();  
   readyQueue.pop();  
   n++;  
   }  
   for (int i = 0; i < n; i++)  
   {  
   readyQueue.push(temp[i]);  
   }  
   }  
   }  
   else  
   {  
   blockQueue.push(blockQueue.front());  
   blockQueue.pop();  
   }  
   }  
   }  
   3写函数查看当前系统状态：（请文字写明思路并给出核心代码）
8. 如何查看系统信息  
   思路：打印相应队列中的信息  
   代码：  
   //显示就绪队列  
   void displayReady()  
   {  
   cout << “----------------就绪队列----------------” << endl;  
   queue temp = readyQueue;  
   while (!temp.empty())  
   {  
   if(algorithm == 1)  
   {  
   cout << “进程名” <<" 进程ID"<< " 到达时间"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<< " 周转时间"<< " 状态"<< endl;  
   while (!temp.empty())  
   {  
   cout<<temp.front().name<<" “<<temp.front().id<<” “<< temp.front().ArriTime<<” “<< temp.front().serviceTime<<” “<<temp.front().needTime<<” “<<temp.front().CPUTime<<” “<<temp.front().CPUTime-temp.front().ArriTime<<” “<<temp.front().state<<endl;  
   temp.pop();  
   cout << endl;  
   }  
   }  
   if(algorithm == 2)  
   {  
   cout << “进程名” <<” 进程ID"<< " 优先级"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<< " 周转时间"<< " 状态"<< endl;  
   while (!temp.empty())  
   {  
   cout<<temp.front().name<<" “<<temp.front().id<<” “<< temp.front().priority<<” “<< temp.front().serviceTime<<” “<<temp.front().needTime<<” “<< temp.front().CPUTime<<” “<<temp.front().finishTime-temp.front().ArriTime<<” “<< temp.front().state<<endl;  
   temp.pop();  
   cout << endl;  
   }  
   }  
   if(algorithm == 3)  
   {  
   cout << “进程名” <<” 进程ID"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<< " 周转时间"<< " 状态"<< endl;  
   while (!temp.empty())  
   {  
   cout<<temp.front().name<<" “<<temp.front().id<<” “<< temp.front().serviceTime<<” “<<temp.front().needTime<<” “<<temp.front().CPUTime<<” “<<temp.front().finishTime-temp.front().ArriTime<<” "<<temp.front().state<<endl;  
   temp.pop();  
   cout << endl;  
   }  
   }  
   }
9. }  
   //显示运行队列  
   void displayRun()  
   {  
   cout << “----------------运行队列----------------” << endl;  
   queue temp = runQueue;  
   while (!temp.empty())  
   {  
   if(algorithm == 1)  
   {  
   cout << “进程名” <<" 进程ID"<< " 到达时间"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<<" 周转时间"<< " 状态"<< endl;  
   while (!temp.empty())  
   {  
   cout<<temp.front().name<<" “<<temp.front().id<<” “<< temp.front().ArriTime<<” “<< temp.front().serviceTime<<” “<<temp.front().needTime<<” “<<temp.front().CPUTime<<” “<<temp.front().CPUTime-temp.front().ArriTime<<” “<<temp.front().state<<endl;  
   temp.pop();  
   cout << endl;  
   }  
   }  
   if(algorithm == 2)  
   {  
   cout << “进程名” <<” 进程ID"<< " 优先级"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<< " 周转时间"<< " 状态"<< endl;  
   while (!temp.empty())  
   {  
   cout<<temp.front().name<<" “<<temp.front().id<<” “<< temp.front().priority<<” “<< temp.front().serviceTime<<” “<<temp.front().needTime<<” “<< temp.front().CPUTime<<” “<< temp.front().finishTime-temp.front().ArriTime<<” “<< temp.front().state<<endl;  
   temp.pop();  
   cout << endl;  
   }  
   }  
   if(algorithm == 3)  
   {  
   cout << “进程名” <<” 进程ID"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<< " 周转时间"<< " 状态"<< endl;  
   while (!temp.empty())  
   {  
   cout<<temp.front().name<<" “<<temp.front().id<<” “<< temp.front().serviceTime<<” “<<temp.front().needTime<<” “<<temp.front().CPUTime<<” “<<temp.front().finishTime-temp.front().ArriTime<<” “<<temp.front().state<<endl;  
   temp.pop();  
   cout << endl;  
   }  
   }  
   }  
   }  
   //显示阻塞队列  
   void displayBlock()  
   {  
   cout << “----------------阻塞队列----------------” << endl;  
   queue temp = blockQueue;  
   while (!temp.empty())  
   {  
   if(algorithm == 1)  
   {  
   cout << “进程名” <<” 进程ID"<< " 到达时间"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<<" 周转时间"<< " 状态"<< " 阻塞原因"<< endl;  
   while (!temp.empty())  
   {  
   cout<<temp.front().name<<" “<<temp.front().id<<” “<< temp.front().ArriTime<<” “<< temp.front().serviceTime<<” “<<temp.front().needTime<<” “<<temp.front().CPUTime<<” “<<temp.front().CPUTime-temp.front().ArriTime<<” “<<temp.front().state<<” “<<temp.front().blockCause<<endl;  
   temp.pop();  
   cout << endl;  
   }  
   }  
   if(algorithm == 2)  
   {  
   cout << “进程名” <<” 进程ID"<< " 优先级"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<<" 周转时间"<< " 状态"<< " 阻塞原因"<< endl;  
   while (!temp.empty())  
   {  
   cout<<temp.front().name<<" “<<temp.front().id<<” “<< temp.front().priority<<” “<< temp.front().serviceTime<<” “<<temp.front().needTime<<” “<< temp.front().CPUTime<<” “<< temp.front().finishTime-temp.front().ArriTime<<” “<< temp.front().state<<” “<<temp.front().blockCause<<endl;  
   temp.pop();  
   cout << endl;  
   }  
   }  
   if(algorithm == 3)  
   {  
   cout << “进程名” <<” 进程ID"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<<" 周转时间"<< " 状态"<< " 阻塞原因"<< endl;  
   while (!temp.empty())  
   {  
   cout<<temp.front().name<<" “<<temp.front().id<<” “<< temp.front().serviceTime<<” “<<temp.front().needTime<<” “<<temp.front().CPUTime<<” “<<temp.front().finishTime-temp.front().ArriTime<<” “<<temp.front().state<<” “<<temp.front().blockCause<<endl;  
   temp.pop();  
   cout << endl;  
   }  
   }  
   }  
   }  
   //显示完成队列  
   void displayFinish()  
   {  
   cout << “----------------完成队列----------------” << endl;  
   queue temp = finishQueue;  
   while (!temp.empty())  
   {  
   if(algorithm == 1)  
   {  
   cout << “进程名” <<” 进程ID"<< " 到达时间"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<< " 完成时间"<< " 周转时间"<< " 状态"<< endl;  
   while (!temp.empty())  
   {  
   cout<<temp.front().name<<" “<<temp.front().id<<” “<< temp.front().ArriTime<<” “<< temp.front().serviceTime<<” “<<temp.front().needTime<<” “<<temp.front().CPUTime<<” “<< temp.front().finishTime<<” “<< temp.front().finishTime-temp.front().ArriTime<<” “<<temp.front().state<<endl;  
   temp.pop();  
   cout << endl;  
   }  
   }  
   if(algorithm == 2)  
   {  
   cout << “进程名” <<” 进程ID"<< " 优先级"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<< " 周转时间"<< " 完成时间"<< " 状态"<< endl;  
   while (!temp.empty())  
   {  
   cout<<temp.front().name<<" “<<temp.front().id<<” “<< temp.front().priority<<” “<< temp.front().serviceTime<<” “<<temp.front().needTime<<” “<< temp.front().CPUTime<<” “<< temp.front().finishTime-temp.front().ArriTime<<” “<< temp.front().finishTime<<” “<< temp.front().state<<endl;  
   temp.pop();  
   cout << endl;  
   }  
   }  
   if(algorithm == 3)  
   {  
   cout << “进程名” <<” 进程ID"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<< " 周转时间"<< " 完成时间"<< " 状态"<< endl;  
   while (!temp.empty())  
   {  
   cout<<temp.front().name<<" “<<temp.front().id<<” “<< temp.front().serviceTime<<” “<<temp.front().needTime<<” “<<temp.front().CPUTime<<” “<<temp.front().finishTime-temp.front().ArriTime<<” “<< temp.front().finishTime<<” "<<temp.front().state<<endl;  
   temp.pop();  
   cout << endl;  
   }  
   }  
   }
10. }  
    //显示挂起队列  
    void displayHung()  
    {  
    cout << “----------------挂起队列----------------” << endl;  
    queue temp = hungQueue;  
    while (!temp.empty())  
    {  
    if(algorithm == 1)  
    {  
    cout << “进程名” <<" 进程ID"<< " 到达时间"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<<" 周转时间"<< " 状态"<< " 挂起原因"<< endl;  
    while (!temp.empty())  
    {  
    cout<<temp.front().name<<" “<<temp.front().id<<” “<< temp.front().ArriTime<<” “<< temp.front().serviceTime<<” “<<temp.front().needTime<<” “<<temp.front().CPUTime<<” “<<temp.front().CPUTime-temp.front().ArriTime<<” “<<temp.front().state<<” “<<temp.front().blockCause<<endl;  
    temp.pop();  
    cout << endl;  
    }  
    }  
    if(algorithm == 2)  
    {  
    cout << “进程名” <<” 进程ID"<< " 优先级"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<<" 周转时间"<< " 状态"<< " 挂起原因"<< endl;  
    while (!temp.empty())  
    {  
    cout<<temp.front().name<<" “<<temp.front().id<<” “<< temp.front().priority<<” “<< temp.front().serviceTime<<” “<<temp.front().needTime<<” “<< temp.front().CPUTime<<” “<< temp.front().finishTime-temp.front().ArriTime<<” “<< temp.front().state<<” “<<temp.front().blockCause<<endl;  
    temp.pop();  
    cout << endl;  
    }  
    }  
    if(algorithm == 3)  
    {  
    cout << “进程名” <<” 进程ID"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<<" 周转时间"<< " 状态"<< " 挂起原因"<< endl;  
    while (!temp.empty())  
    {  
    cout<<temp.front().name<<" “<<temp.front().id<<” “<< temp.front().serviceTime<<” “<<temp.front().needTime<<” “<<temp.front().CPUTime<<” “<<temp.front().finishTime-temp.front().ArriTime<<” “<<temp.front().state<<” "<<temp.front().blockCause<<endl;  
    temp.pop();  
    cout << endl;  
    }  
    }  
    }  
    }  
    2) 周转时间的计算。  
    思路：周转事件=完成时间-到达时间。  
    代码：  
    temp.front().finishTime-temp.front().ArriTime
11. 3、 需要实现的调度算法：（请文字写明思路并给出核心代码）
12. 先来先服务调度算法  
    思路：详细过程在见开始调度algorithm == 1。  
    代码：  
    void FCFS()  
    {  
    algorithm = 1;  
    PCB temp[processQueue.size()];  
    int n = 0;  
    while (!processQueue.empty())  
    {  
    temp[n] = processQueue.front();  
    processQueue.pop();  
    n++;  
    }  
    sort(temp, temp + n, cmp1());  
    for (int i = 0; i < n; i++)  
    {  
    strcpy(temp[i].state,“Ready”);  
    readyQueue.push(temp[i]);  
    }  
    }

# 编码

#include <iostream>

#include <queue>

#include <algorithm>

#include <cstdio>

#include <cstring>

#include <cstdlib>

using namespace std;

const int MAXN = 1000; //假设能够容纳的进程最多的个数

typedef struct node

{

char name[20]; //进程名

int id = 0; //进程ID

int priority = 0; //优先级

int ArriTime = 0; //进程到达时间

int serviceTime=0; //进程总需要时间

int needTime = 0; //还需要运行时间

int CPUTime = 0; //已用CPU时间

char state[20]; //进程状态 1.Ready 2.Run 3.Wait 4.Finish

int finishTime = 0; //完成时间

char blockCause[30]; //阻塞原因

int aroundTime=0; //周转时间

} PCB;

queue<PCB> processQueue, readyQueue, blockQueue, runQueue, finishQueue,hungQueue;

int xtime = 0; //系统时间

int algorithm = 0; //算法选择标志

int timeslice = 0; //时间片大小

bool finish; //是否完成

struct cmp1

{

bool operator()(const PCB &p1, const PCB &p2)

{

return p1.ArriTime < p2.ArriTime; //到达时间从小到大排序

}

};

struct cmp2

{

bool operator()(const PCB &p1, const PCB &p2)

{

return p1.priority > p2.priority; //优先级从大到小排序

}

};

//初始化进程

void Init()

{

PCB p1, p2, p3,p4,p5;

strcpy(p1.name, "a");

p1.id = 1;

p1.priority = 8;

p1.ArriTime = 2;

p1.needTime = 4;

p1.serviceTime=4;

processQueue.push(p1);

strcpy(p2.name, "b");

p2.id = 2;

p2.priority = 4;

p2.ArriTime = 6;

p2.needTime = 4;

p2.serviceTime=4;

processQueue.push(p2);

strcpy(p3.name, "c");

p3.id = 3;

p3.priority = 7;

p3.ArriTime = 3;

p3.needTime = 9;

p3.serviceTime=9;

processQueue.push(p3);

strcpy(p4.name, "d");

p4.id = 4;

p4.priority = 1;

p4.ArriTime = 0;

p4.needTime = 1;

p4.serviceTime=1;

processQueue.push(p4);

strcpy(p5.name, "e");

p5.id = 5;

p5.priority = 2;

p5.ArriTime = 2;

p5.needTime = 2;

p5.serviceTime=2;

processQueue.push(p5);

}

//添加进程

void addProcess()

{

int n;

cout << "请输入要添加的进程数：" << endl;

cin >> n;

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

{

PCB p;

cout << "请输入第"<<i+1<<"个进程名、进程优先级、进程到达时间、进程需要运行时间:" << endl;

cin >> p.name>>p.priority>> p.ArriTime>> p.serviceTime;

p.id=i+1;

p.needTime=p.serviceTime;

processQueue.push(p);

}

}

//系统开始调度

void beginDispatch()

{

if (runQueue.empty())

{

if (!readyQueue.empty())

{

strcpy(readyQueue.front().state,"Run");

runQueue.push(readyQueue.front());

readyQueue.pop();

runQueue.front().CPUTime++;

runQueue.front().needTime--;

xtime++;

}

else if(!blockQueue.empty())

{

char chance='p';

cout<<endl;

cout<<"\*\*\*\*\*\*\*\*就绪队列不为空\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*若结束程序请按y,若不结束请按n\*\*\*\*\*\*"<<endl;

cin>>chance;

if(chance=='y')

{

finish = true;

return;

}

}

else if(!hungQueue.empty())

{

char chance='p';

cout<<endl;

cout<<"\*\*\*\*\*\*\*\*挂起队列不为空\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*若结束程序请按y,若不结束请按n\*\*\*\*\*\*"<<endl;

cin>>chance;

if(chance=='y')

{

finish = true;

return;

}

}

else

{

finish = true;

return;

}

}

else if (!runQueue.empty())

{

if (runQueue.front().needTime == 0)

{

runQueue.front().finishTime = xtime;

strcpy(runQueue.front().state,"Finish");

finishQueue.push(runQueue.front());

runQueue.pop();

if (!readyQueue.empty())

{

strcpy(readyQueue.front().state,"Run");

runQueue.push(readyQueue.front());

readyQueue.pop();

}

}

//先来先服务

if (algorithm == 1)

{

xtime++;

runQueue.front().CPUTime++;

runQueue.front().needTime--;

}

//优先级

if (algorithm == 2)

{

xtime++;

runQueue.front().CPUTime++;

runQueue.front().needTime--;

}

//时间片轮转算法

if(algorithm == 3)

{

xtime++;

runQueue.front().CPUTime++;

runQueue.front().needTime--;

if (runQueue.front().needTime != 0)

{

if ((runQueue.front().CPUTime >= timeslice) && (runQueue.front().CPUTime % timeslice == 0))

{

strcpy(runQueue.front().state,"Ready");

readyQueue.push(runQueue.front());

runQueue.pop();

if (!readyQueue.empty())

{

strcpy(readyQueue.front().state,"Run");

runQueue.push(readyQueue.front());

readyQueue.pop();

}

}

}

}

}

}

//当前运行进程请求 I/O 事件

void IORequest()

{

if (!runQueue.empty())

{

if (runQueue.front().needTime != 0)

{

strcpy(runQueue.front().blockCause, "请求 I/O 事件");

strcpy(runQueue.front().state,"Wait");

blockQueue.push(runQueue.front());

runQueue.pop();

}

}

else

{

cout << "当前无运行进程" << endl;

}

}

//某进程I/O完成

void IOFinish()

{

//空

if (blockQueue.empty())

{

cout << "目前无阻塞进程" << endl;

}

//不空

else

{

int id;

cout << "请输入I/O完成的进程ID" << endl;

cin >> id;

for (int i = 0; i < blockQueue.size(); i++)

{

if (blockQueue.front().id == id)

{

strcpy(blockQueue.front().blockCause, " ");

strcpy(blockQueue.front().state,"Ready");

readyQueue.push(blockQueue.front());

blockQueue.pop();

if (algorithm == 1)//先来先服务

{

//排序

PCB temp[readyQueue.size()];

int n = 0;

while (!readyQueue.empty())

{

temp[n] = readyQueue.front();

readyQueue.pop();

n++;

}

sort(temp, temp + n, cmp1());

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

{

readyQueue.push(temp[i]);

}

}

if (algorithm == 2)//优先级

{ if (algorithm == 1)//先来先服务

{

//排序

PCB temp[readyQueue.size()];

int n = 0;

while (!readyQueue.empty())

{

temp[n] = readyQueue.front();

readyQueue.pop();

n++;

}

sort(temp, temp + n, cmp1());

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

{

readyQueue.push(temp[i]);

}

sort(temp, temp + n, cmp2());

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

{

readyQueue.push(temp[i]);

}

}

if (algorithm == 3)//时间片轮转

{

//排序

PCB temp[readyQueue.size()];

int n = 0;

while (!readyQueue.empty())

{

temp[n] = readyQueue.front();

readyQueue.pop();

n++;

}

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

{

readyQueue.push(temp[i]);

}

}

}

else

{

blockQueue.push(blockQueue.front());

blockQueue.pop();

}

}

}

}

}

//时间片到期

void timesliceDaoqi()

{

if (!runQueue.empty())

{

if (runQueue.front().needTime != 0)

{

strcpy(runQueue.front().state,"Ready");

readyQueue.push(runQueue.front());

runQueue.pop();

}

}

else

{

cout << "当前无运行进程" << endl;

}

}

//其他阻塞事件

void ElseZC()

{

if (!runQueue.empty())

{

if (runQueue.front().needTime != 0)

{

strcpy(runQueue.front().blockCause, "其他阻塞事件");

strcpy(runQueue.front().state,"Wait");

blockQueue.push(runQueue.front());

runQueue.pop();

}

}

else

{

cout << "当前无运行进程" << endl;

}

}

void ElseZCFinish()

{

//空

if (blockQueue.empty())

{

cout << "目前无阻塞进程" << endl;

}

//不空

else

{

int id;

cout << "请输入其他阻塞事件完成的进程ID" << endl;

cin >> id;

for (int i = 0; i < blockQueue.size(); i++)

{

if (blockQueue.front().id == id)

{

strcpy(blockQueue.front().blockCause, " ");

strcpy(blockQueue.front().state,"Ready");

readyQueue.push(blockQueue.front());

blockQueue.pop();

if (algorithm == 1)//先来先服务

{

//排序

PCB temp[readyQueue.size()];

int n = 0;

while (!readyQueue.empty())

{

temp[n] = readyQueue.front();

readyQueue.pop();

n++;

}

sort(temp, temp + n, cmp1());

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

{

readyQueue.push(temp[i]);

}

}

if (algorithm == 2)//优先级

{ if (algorithm == 1)//先来先服务

{

//排序

PCB temp[readyQueue.size()];

int n = 0;

while (!readyQueue.empty())

{

temp[n] = readyQueue.front();

readyQueue.pop();

n++;

}

sort(temp, temp + n, cmp1());

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

{

readyQueue.push(temp[i]);

}

//排序

int m = 0;

while (!readyQueue.empty())

{

temp[m] = readyQueue.front();

readyQueue.pop();

n++;

}

sort(temp, temp + m, cmp2());

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

{

readyQueue.push(temp[i]);

}

}

if (algorithm == 3)//时间片轮转

{

//排序

PCB temp[readyQueue.size()];

int n = 0;

while (!readyQueue.empty())

{

temp[n] = readyQueue.front();

readyQueue.pop();

n++;

}

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

{

readyQueue.push(temp[i]);

}

}

}

else

{

blockQueue.push(blockQueue.front());

blockQueue.pop();

}

}

}

}

}

//挂起

void Hung()

{

if (!runQueue.empty())

{

if (runQueue.front().needTime != 0)

{

strcpy(runQueue.front().blockCause, "请求挂起");

strcpy(runQueue.front().state,"Hung");

hungQueue.push(runQueue.front());

runQueue.pop();

}

}

else

{

cout << "当前无运行进程" << endl;

}

}

//某进程挂起完成

void HungFinish()

{

//空

if (hungQueue.empty())

{

cout << "目前无挂起进程" << endl;

}

//不空

else

{

int id;

cout << "请输入挂起完成的进程ID" << endl;

cin >> id;

for (int i = 0; i < hungQueue.size(); i++)

{

if (hungQueue.front().id == id)

{

strcpy(hungQueue.front().blockCause, " ");

strcpy(hungQueue.front().state,"Ready");

readyQueue.push(hungQueue.front());

hungQueue.pop();

if (algorithm == 1)//先来先服务

{

//排序

PCB temp[readyQueue.size()];

int n = 0;

while (!readyQueue.empty())

{

temp[n] = readyQueue.front();

readyQueue.pop();

n++;

}

sort(temp, temp + n, cmp1());

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

{

readyQueue.push(temp[i]);

}

}

if (algorithm == 2)//优先级

{ if (algorithm == 1)//先来先服务

{

//排序

PCB temp[readyQueue.size()];

int n = 0;

while (!readyQueue.empty())

{

temp[n] = readyQueue.front();

readyQueue.pop();

n++;

}

sort(temp, temp + n, cmp1());

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

{

readyQueue.push(temp[i]);

}

//排序

int m = 0;

while (!readyQueue.empty())

{

temp[m] = readyQueue.front();

readyQueue.pop();

n++;

}

sort(temp, temp + m, cmp2());

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

{

readyQueue.push(temp[i]);

}

}

if (algorithm == 3)//时间片轮转

{

//排序

PCB temp[readyQueue.size()];

int n = 0;

while (!readyQueue.empty())

{

temp[n] = readyQueue.front();

readyQueue.pop();

n++;

}

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

{

readyQueue.push(temp[i]);

}

}

}

else

{

hungQueue.push(hungQueue.front());

hungQueue.pop();

}

}

}

}

}

//显示就绪队列

void displayReady()

{

cout << "----------------就绪队列----------------" << endl;

queue<PCB> temp = readyQueue;

while (!temp.empty())

{

if(algorithm == 1)

{

cout << "进程名" <<" 进程ID"<< " 到达时间"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<< " 周转时间"<< " 状态"<< endl;

while (!temp.empty())

{

cout<<temp.front().name<<" "<<temp.front().id<<" "<< temp.front().ArriTime<<" "<< temp.front().serviceTime<<" "<<temp.front().needTime<<" "<<temp.front().CPUTime<<" "<<temp.front().CPUTime-temp.front().ArriTime<<" "<<temp.front().state<<endl;

temp.pop();

cout << endl;

}

}

if(algorithm == 2)

{

cout << "进程名" <<" 进程ID"<< " 优先级"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<< " 周转时间"<< " 状态"<< endl;

while (!temp.empty())

{

cout<<temp.front().name<<" "<<temp.front().id<<" "<< temp.front().priority<<" "<< temp.front().serviceTime<<" "<<temp.front().needTime<<" "<< temp.front().CPUTime<<" "<<temp.front().finishTime-temp.front().ArriTime<<" "<< temp.front().state<<endl;

temp.pop();

cout << endl;

}

}

if(algorithm == 3)

{

cout << "进程名" <<" 进程ID"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<< " 周转时间"<< " 状态"<< endl;

while (!temp.empty())

{

cout<<temp.front().name<<" "<<temp.front().id<<" "<< temp.front().serviceTime<<" "<<temp.front().needTime<<" "<<temp.front().CPUTime<<" "<<temp.front().finishTime-temp.front().ArriTime<<" "<<temp.front().state<<endl;

temp.pop();

cout << endl;

}

}

}

}

//显示运行队列

void displayRun()

{

cout << "----------------运行队列----------------" << endl;

queue<PCB> temp = runQueue;

while (!temp.empty())

{

if(algorithm == 1)

{

cout << "进程名" <<" 进程ID"<< " 到达时间"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<<" 周转时间"<< " 状态"<< endl;

while (!temp.empty())

{

cout<<temp.front().name<<" "<<temp.front().id<<" "<< temp.front().ArriTime<<" "<< temp.front().serviceTime<<" "<<temp.front().needTime<<" "<<temp.front().CPUTime<<" "<<temp.front().CPUTime-temp.front().ArriTime<<" "<<temp.front().state<<endl;

temp.pop();

cout << endl;

}

}

if(algorithm == 2)

{

cout << "进程名" <<" 进程ID"<< " 优先级"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<< " 周转时间"<< " 状态"<< endl;

while (!temp.empty())

{

cout<<temp.front().name<<" "<<temp.front().id<<" "<< temp.front().priority<<" "<< temp.front().serviceTime<<" "<<temp.front().needTime<<" "<< temp.front().CPUTime<<" "<< temp.front().finishTime-temp.front().ArriTime<<" "<< temp.front().state<<endl;

temp.pop();

cout << endl;

}

}

if(algorithm == 3)

{

cout << "进程名" <<" 进程ID"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<< " 周转时间"<< " 状态"<< endl;

while (!temp.empty())

{

cout<<temp.front().name<<" "<<temp.front().id<<" "<< temp.front().serviceTime<<" "<<temp.front().needTime<<" "<<temp.front().CPUTime<<" "<<temp.front().finishTime-temp.front().ArriTime<<" "<<temp.front().state<<endl;

temp.pop();

cout << endl;

}

}

}

}

//显示阻塞队列

void displayBlock()

{

cout << "----------------阻塞队列----------------" << endl;

queue<PCB> temp = blockQueue;

while (!temp.empty())

{

if(algorithm == 1)

{

cout << "进程名" <<" 进程ID"<< " 到达时间"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<<" 周转时间"<< " 状态"<< " 阻塞原因"<< endl;

while (!temp.empty())

{

cout<<temp.front().name<<" "<<temp.front().id<<" "<< temp.front().ArriTime<<" "<< temp.front().serviceTime<<" "<<temp.front().needTime<<" "<<temp.front().CPUTime<<" "<<temp.front().CPUTime-temp.front().ArriTime<<" "<<temp.front().state<<" "<<temp.front().blockCause<<endl;

temp.pop();

cout << endl;

}

}

if(algorithm == 2)

{

cout << "进程名" <<" 进程ID"<< " 优先级"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<<" 周转时间"<< " 状态"<< " 阻塞原因"<< endl;

while (!temp.empty())

{

cout<<temp.front().name<<" "<<temp.front().id<<" "<< temp.front().priority<<" "<< temp.front().serviceTime<<" "<<temp.front().needTime<<" "<< temp.front().CPUTime<<" "<< temp.front().finishTime-temp.front().ArriTime<<" "<< temp.front().state<<" "<<temp.front().blockCause<<endl;

temp.pop();

cout << endl;

}

}

if(algorithm == 3)

{

cout << "进程名" <<" 进程ID"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<<" 周转时间"<< " 状态"<< " 阻塞原因"<< endl;

while (!temp.empty())

{

cout<<temp.front().name<<" "<<temp.front().id<<" "<< temp.front().serviceTime<<" "<<temp.front().needTime<<" "<<temp.front().CPUTime<<" "<<temp.front().finishTime-temp.front().ArriTime<<" "<<temp.front().state<<" "<<temp.front().blockCause<<endl;

temp.pop();

cout << endl;

}

}

}

}

//显示完成队列

void displayFinish()

{

cout << "----------------完成队列----------------" << endl;

queue<PCB> temp = finishQueue;

while (!temp.empty())

{

if(algorithm == 1)

{

cout << "进程名" <<" 进程ID"<< " 到达时间"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<< " 完成时间"<< " 周转时间"<< " 状态"<< endl;

while (!temp.empty())

{

cout<<temp.front().name<<" "<<temp.front().id<<" "<< temp.front().ArriTime<<" "<< temp.front().serviceTime<<" "<<temp.front().needTime<<" "<<temp.front().CPUTime<<" "<< temp.front().finishTime<<" "<< temp.front().finishTime-temp.front().ArriTime<<" "<<temp.front().state<<endl;

temp.pop();

cout << endl;

}

}

if(algorithm == 2)

{

cout << "进程名" <<" 进程ID"<< " 优先级"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<< " 周转时间"<< " 完成时间"<< " 状态"<< endl;

while (!temp.empty())

{

cout<<temp.front().name<<" "<<temp.front().id<<" "<< temp.front().priority<<" "<< temp.front().serviceTime<<" "<<temp.front().needTime<<" "<< temp.front().CPUTime<<" "<< temp.front().finishTime-temp.front().ArriTime<<" "<< temp.front().finishTime<<" "<< temp.front().state<<endl;

temp.pop();

cout << endl;

}

}

if(algorithm == 3)

{

cout << "进程名" <<" 进程ID"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<< " 周转时间"<< " 完成时间"<< " 状态"<< endl;

while (!temp.empty())

{

cout<<temp.front().name<<" "<<temp.front().id<<" "<< temp.front().serviceTime<<" "<<temp.front().needTime<<" "<<temp.front().CPUTime<<" "<<temp.front().finishTime-temp.front().ArriTime<<" "<< temp.front().finishTime<<" "<<temp.front().state<<endl;

temp.pop();

cout << endl;

}

}

}

}

//显示挂起队列

void displayHung()

{

cout << "----------------挂起队列----------------" << endl;

queue<PCB> temp = hungQueue;

while (!temp.empty())

{

if(algorithm == 1)

{

cout << "进程名" <<" 进程ID"<< " 到达时间"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<<" 周转时间"<< " 状态"<< " 挂起原因"<< endl;

while (!temp.empty())

{

cout<<temp.front().name<<" "<<temp.front().id<<" "<< temp.front().ArriTime<<" "<< temp.front().serviceTime<<" "<<temp.front().needTime<<" "<<temp.front().CPUTime<<" "<<temp.front().CPUTime-temp.front().ArriTime<<" "<<temp.front().state<<" "<<temp.front().blockCause<<endl;

temp.pop();

cout << endl;

}

}

if(algorithm == 2)

{

cout << "进程名" <<" 进程ID"<< " 优先级"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<<" 周转时间"<< " 状态"<< " 挂起原因"<< endl;

while (!temp.empty())

{

cout<<temp.front().name<<" "<<temp.front().id<<" "<< temp.front().priority<<" "<< temp.front().serviceTime<<" "<<temp.front().needTime<<" "<< temp.front().CPUTime<<" "<< temp.front().finishTime-temp.front().ArriTime<<" "<< temp.front().state<<" "<<temp.front().blockCause<<endl;

temp.pop();

cout << endl;

}

}

if(algorithm == 3)

{

cout << "进程名" <<" 进程ID"<< " 服务时间"<< " 需要时间" << " 已用CPU时间"<<" 周转时间"<< " 状态"<< " 挂起原因"<< endl;

while (!temp.empty())

{

cout<<temp.front().name<<" "<<temp.front().id<<" "<< temp.front().serviceTime<<" "<<temp.front().needTime<<" "<<temp.front().CPUTime<<" "<<temp.front().finishTime-temp.front().ArriTime<<" "<<temp.front().state<<" "<<temp.front().blockCause<<endl;

temp.pop();

cout << endl;

}

}

}

}

//先来先服务

void FCFS()

{

algorithm = 1;

PCB temp[processQueue.size()];

int n = 0;

while (!processQueue.empty())

{

temp[n] = processQueue.front();

processQueue.pop();

n++;

}

sort(temp, temp + n, cmp1());

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

{

strcpy(temp[i].state,"Ready");

readyQueue.push(temp[i]);

}

}

//优先级调度算法

void SJF()

{

algorithm = 2;

PCB temp[processQueue.size()];

int n = 0;

while (!processQueue.empty())

{

temp[n] = processQueue.front();

processQueue.pop();

n++;

}

sort(temp, temp + n, cmp2());

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

{

strcpy(temp[i].state,"Ready");

readyQueue.push(temp[i]);

}

}

//时间片轮转

void RR()

{

cout << "请输入时间片大小" << endl;

cin >> timeslice;

algorithm = 3;

readyQueue = processQueue;

for (int i = 0; i < readyQueue.size(); i++)

{

strcpy(readyQueue.front().state,"Ready");

readyQueue.push(readyQueue.front());

readyQueue.pop();

}

}

int main()

{

cout<<"\*\*\*\*\*\*\*\*进程状态转换系统\*\*\*\*\*\*\*\*"<<endl;

int chance;

cout << "1:自动添加进程 2:手动添加进程" << endl;

cin>>chance;

switch(chance)

{

case 1:

Init();

break;

case 2:

addProcess();

break;

}

int e;

cout << "请输入算法" << endl;

cout << "1.先来先服务 2.优先级抢占 3.时间片轮转" << endl;

cin >> e;

switch (e)

{

case 1:

FCFS();

break;

case 2:

SJF();

break;

case 3:

RR();

break;

}

while (true)

{

int e;

cout << "请输入操作" << endl;

cout << "1.开始调度 2.IO请求 3.IO完成 4.时间片到期 5.其他阻塞事件 6.其他阻塞事件完成 7.挂起 8.挂起完成 0.退出" << endl;

cin >> e;

switch (e)

{

case 1:

beginDispatch();

break;

case 2:

IORequest();

break;

case 3:

IOFinish();

break;

case 4:

timesliceDaoqi();

break;

case 5:

ElseZC();

break;

case 6:

ElseZCFinish();

break;

case 7:

Hung();

break;

case 8:

HungFinish();

break;

case 0:

exit(0);

}

displayReady();

displayRun();

displayBlock();

displayFinish();

displayHung();

cout << "当前系统CPU时间：" << xtime << endl;

if (finish)

{

cout << "当前进程全部完成" << endl;

cout << "当前时间：" << xtime << endl;

break;

}

system("pause");

}

system("pause");

return 0;

}

# 测试

测试表：

|  |  |  |  |
| --- | --- | --- | --- |
| Test case | Description | Test result | Reference\* |
| 1 | 创建进程 | OK | 1-1 |
| 2 | 进程调度 | OK | 2-1 |
| 3 | 请求i/o | OK | 3-1 |
| 4 | 进程阻塞 | OK | 4-1 |
| 5 | 挂起 | OK | 5-1 |
| 6 | 完成调度 | OK | 6-1 |

\*Reference:

运行截图

1-1

图片包含 日程表

描述已自动生成

2-1

图片包含 日程表

描述已自动生成

图片包含 文本

描述已自动生成

3-1

日程表

描述已自动生成

4-1

电脑的屏幕截图

描述已自动生成

5-1

图形用户界面

描述已自动生成

6-1

屏幕上有字

描述已自动生成

# 心得体会