# 核心代码（cpu.java）

**if**( jq.size() > 0 && clock.GetTime() >= jq.front.InTime)//若作业队列非空且该时刻有作业到达

{

Job\_Num++;//系统中的作业数目+1

JCB jcb = jq.front;//获取作业队列的头队列

System.***out***.println(clock.GetTime() + ":第" + Job\_Num + "个作业到达" + " 包含" + jcb.PRO\_Number + "个进程 ");//显示

**for**(**int** i=0;i<jcb.PRO\_Number;i++)//为每一个进程分配资源和创建pcb节点

{

System.***out***.println("进程号:" + Process\_Num + " 指令数:" + jcb.pro[i][2] + " 总页面数:" + jcb.pro[i][4]);

**try** {

PCB pcb = **this**.createpcb(jcb,i);//创建jcb的第i个进程

pcbtable.ReadyQueue.join(pcb);//创建该作业的每一个进程并将进程加入就绪队列

mmu.AddPagelist();//为新进程分配页表

**for**(**int** j=0;j<jcb.pro[i][4];j++)//为每个进程的每个页分配内存

{

**int** k = Memory.*Allocate*(Process\_Num , j);

//System.out.print(k + "|");

**if**(k == -1)//未分配成功

{

mmu.Pagelist[Process\_Num].Insert(j);//更新该进程的页表

}**else** {//分配成功

mmu.Pagelist[Process\_Num].Insert(j , k);//更新该进程的页表

/\*Memory.FreePageNum -= jcb.pro[i][4];

Memory.UsedPageFrameNum += jcb.pro[i][4]; \*/

}

}

Process\_Num++;//系统中进程数+1

} **catch** (Exception e) {

e.printStackTrace();

}

}

jq.quit();//将已经调度成进程的作业出队

}

//中级调度

//低级调度

**if**(pcbtable.RunQueue == **null** && pcbtable.ReadyQueue.size() == 0 && pcbtable.WaitQueue.size() == 0 && jq.size() ==0)//若所有队列都为空

{

IsEnd = **true**;//若所有进程已经执行完，更改标志位，退出循环

**int** a[] = Memory.*show*();//获取每一个页框中占用的进程号

**int** b[] = Memory.*show1*();//获取每一个页框中占用的进程的页号

inter.altermemory(a,b);//更新内存显示界面

//wat.update(0,0,0);

System.***out***.println("all job done!");//输出所有作业完成的信息

**break**;//退出大循环

}

**if**(pcbtable.WaitQueue.size() > 0)//如果等待队列非空，寻找可以唤醒的进程

{

**int** size = pcbtable.WaitQueue.size();//获取等待队列里节点的个数

PCB [] xy= **new** PCB[size];//存放不能被唤醒的进程

**int** num = 0;//记录有多少进程不能被唤醒

PCB p = pcbtable.WaitQueue.front();

**for**(**int** i=0;i<size;i++)//检查每一个节点是否可以被唤醒

{

**if**((p.WaitState == 0) && (clock.GetTime() -p.timeflag) >= p.instruction[p.PSW].Instr\_TotalTime)//如果该进程因I/O操作中断且I/O操作结束

{

p.PSW++;//运行完该条I/O指令

**if**(p.PSW > 200 && p.synchronousflag != -1)//如果该进程同步其他进程且该该进程已运行了200条指令

{

buy[p.Pro\_ID] = **true**;//修改该进程的同步标志位

}

p.WaitState = -1;//修改该进程的等待标志位为没有等待

System.***out***.print("{" + p.Pro\_ID + "stop wait}");

pcbtable.proinfo[p.Pro\_ID][2] = 1;//更新进程状态为就绪态

pcbtable.proinfo[p.Pro\_ID][3]++;//更新进程已运行的指令数

pcbtable.ReadyQueue.join(p);//加入就绪队列

}**else** **if**((p.WaitState == 1) && buy[p.bechargedflag] == **false**)//如果有被同步的进程且同步该进程的进程已经运行完200条指令

{

p.WaitState = -1;//修改该进程的等待标志位为没有等待

p.bechargedflag = -1;//修改该进程被同步标志位为不被其他进程同步

pcbtable.proinfo[p.Pro\_ID][2] = 1;//更新进程状态为就绪态

pcbtable.ReadyQueue.join(p);//加入就绪队列

}

**else** {//无法被唤醒，也就不需要出队

xy[num] = p;

num++;//记录不需要出队的进程数目

}

p = p.next;//检查下一个节点

}

pcbtable.WaitQueue.front = **null**;//赋值

pcbtable.WaitQueue.rear = **null**;//赋值

pcbtable.WaitQueue.Queue\_Size = 0;//重置等待队列

**for**(**int** j=0;j<num;j++)

{

pcbtable.WaitQueue.join(xy[j]);//将无法被唤醒的进程加入等待队列

}

}

**if**(pcbtable.RunQueue != **null**)//如果当前有进程在运行

{

**if**(pcbtable.RunQueue.bechargedflag != -1 && buy[pcbtable.RunQueue.bechargedflag])//如果该进程需要被同步，且同步它的那个进程还没执行完200条指令

{

pcbtable.proinfo[pcbtable.RunQueue.Pro\_ID][2] = 2;//记录进程状态为等待态

pcbtable.RunQueue.WaitState = 1;//更新等待标志位

//System.out.print("{" + pcbtable.RunQueue.Pro\_ID + "wait}");

pcbtable.WaitQueue.join(pcbtable.RunQueue);//加入等待队列

pcbtable.RunQueue = **null**;//运行态指针改为空指针

**if**(pcbtable.ReadyQueue.size() > 0)//如果就绪队列非空

{

pcbtable.RunQueue = pcbtable.ReadyQueue.front();//将就绪队列队头置为运行态

pcbtable.ReadyQueue.quit();//出队

PC = pcbtable.RunQueue.PSW;//PC改为该进程的PSW标志位

pcbtable.RunQueue.timeflag = clock.GetTime();//设置开始运行时间

pcbtable.proinfo[pcbtable.RunQueue.Pro\_ID][2] = 0;//修改进程状态信息为运行态

}

}

**else**{

**switch**(pcbtable.RunQueue.instruction[PC].Instr\_State) {//判断是当前运行态进程要运行哪种指令

**case** 0://系统调用指令

pcbtable.RunQueue.instruction[PC].Instr\_RunTime += 10;//指令运行10ms

**if**(pcbtable.RunQueue.instruction[PC].Instr\_RunTime >= pcbtable.RunQueue.instruction[PC].Instr\_TotalTime)//当前指令运行完毕

{

PC++;//PC自增，指向该进程下一条指令;

**if**(pcbtable.RunQueue.synchronousflag != -1 && PC > 200)//如果该进程需要同步其他进程且已经运行完200条指令

{

buy[pcbtable.RunQueue.Pro\_ID] = **false**;//修改同步标志位

}

pcbtable.proinfo[pcbtable.RunQueue.Pro\_ID][3]++;//已经运行的指令+1

**if**(PC == pcbtable.RunQueue.Pro\_InstrNum)//如果该进程所有指令全部运行完毕

{

pcbtable.proinfo[pcbtable.RunQueue.Pro\_ID][2] = 3;//修改进程状态为完成态

pcbtable.FinishQueue.join(pcbtable.RunQueue);//将该进程加入完成队列

Memory.*recover*(pcbtable.RunQueue.Pro\_ID);//回收其资源

pcbtable.RunQueue = **null**;//设置空值

}**else** {

**this**.Run\_to\_Ready();//进程切换

}

}

**break**;

**case** 1://用户态计算指令

**if**(pcbtable.RunQueue.instruction[PC].Need\_Data == 1)//如果该进程需要访问数据

{

**int** o;

**if**(pcbtable.RunQueue.Pro\_InstrNum % 256 == 0)

{

o = pcbtable.RunQueue.Pro\_InstrNum / 256;

}**else** {

o = (**int**)(pcbtable.RunQueue.Pro\_InstrNum / 256) + 1;

}

**for**(**int** y=o;y<pcbtable.RunQueue.PageNum;y++)//该指令需要存放数据段的页面

{

**int** pageframeid = mmu.StartJob(pcbtable.RunQueue.Pro\_ID , y);//MMU开始工作

System.***out***.print("<需要数据页面，mmu工作一次，逻辑地址为：" + Integer.*toString*(y) + "000,");

**switch** (pageframeid){

**case** -1:

mmu.OutDealing();//越界中断处理

System.***out***.println("产生越界中断>");

**break**;

**case** -2:

**int** back = mmu.LostDealing(pcbtable.RunQueue.Pro\_ID , y);//缺页中断处理，将缺页调入内存

System.***out***.println("产生缺页，缺页处理后对应的物理地址为：" + Integer.*toString*(back) + "000>");

**break**;

**default**:

System.***out***.println("对应的物理地址为：" + Integer.*toString*(pageframeid) + "000>");

**break**;

}

}

}

pcbtable.RunQueue.instruction[PC].Instr\_RunTime += 10;//指令运行10ms

**if**(pcbtable.RunQueue.instruction[PC].Instr\_RunTime >= pcbtable.RunQueue.instruction[PC].Instr\_TotalTime)//当前指令运行完毕

{

//System.out.println("\*\*\*");

PC++;//PC自增，指向该进程下一条指令;

**if**(pcbtable.RunQueue.synchronousflag != -1 && PC > 200)//如果该进程需要同步其他进程且已经运行完200条指令

{

buy[pcbtable.RunQueue.Pro\_ID] = **false**;//修改同步标志位

}

pcbtable.proinfo[pcbtable.RunQueue.Pro\_ID][3]++;//已经运行的指令+1

**if**(PC == pcbtable.RunQueue.Pro\_InstrNum)//如果该进程所有指令全部运行完毕

{

pcbtable.proinfo[pcbtable.RunQueue.Pro\_ID][2] = 3;//已经运行的指令+1

pcbtable.FinishQueue.join(pcbtable.RunQueue);//将该进程加入完成队列

Memory.*recover*(pcbtable.RunQueue.Pro\_ID);//回收其资源

pcbtable.RunQueue = **null**;//设置空值

}

**else** **if**(clock.GetTime() - pcbtable.RunQueue.timeflag > 1000)//时间片到

{

**this**.Run\_to\_Ready();//进程切换

}

}

**break**;//主要包括了作业调度流程以及低级调度流程，低级调度由于篇幅所限，以系统调用指令和用户态指令为例