课内讨论 处理机调度算法

姓名：叶倩琳

班级：软工1706

学号：201706061330

完成日期：2020年1月12日

# 一、讨论内容

选择一个处理机调度算法，分析算法的原理，并通过编程实现。

# 二、目的

在采用多道程序设计的系统中，往往有若干个进程同时处于就绪状态。当就绪进程个数大于处理机数时，就必须依照某种策略来决定哪些进程优先占用处理机。模拟在单处理机情况下的处理机调度，帮助学生加深了解处理机调度的工作。

# 三、要求

第一题：设计一个按优先数调度算法实现处理机调度的程序。

[提示]：

1. 假定系统有五个进程，每一个进程用一个进程控制块PCB来代表。进程控制的格式为：

进程名

指针

要求运行时间

优先数

状态

其中，进程名——作为进程的标识，假设五个进程的进程名分别为P1，P2，P3，P4，P5。

指针——按优先数的大小把五个进程连成队列，用指针指出下一个进程的进程控制块的首地址，最后一个进程中的指针为“0”。

要求运行时间——假设进程需要运行的单位时间数。

优先数——赋予进程的优先数，调度时总是选取优先数大的进程先执行。

状态——可假设有两种状态，“就绪”状态和“结束”状态。五个进程的初始状态都为“就绪”，用“R”表示，当一个进程运行结束后，它的状态为“结束”，用“E”表示。

1. 在每次运行你所设计的处理机调度程序之前，为每个进程任意确定它的“优先数”和“要求运行时间”。
2. 为了调度方便，把五个进程按给定的优先数从大到小连成队列。用一单元指出队首进程，用指针指出队列的连接情况例：

队首标志

k2

k1 P1 k2 P2 k3 P3 k4 P4 k5 P5

0 k4  k5 k3 k1

2 3 1 4 4

1 5 3 2 2

R R R R R

PCB1 PCB2 PCB3 PCB4 PCB5

1. 处理机调度总是选队首进程运行。采用动态改变优先数的办法，进程每运行一次优先数就减“1”。由于本实验是模拟处理机调度，所以，对被选中的进程并不实际的启动运行，而是执行：

优先数-1

要求运行时间-1

来模拟进程的一次运行。

提醒注意的是：在实际的系统中，当一个进程被选中运行时，必须恢复进程的现场，让它占有处理机运行，直到出现等待事件或运行结束。在这里省去了这些工作。

1. 进程运行一次后，若要求运行时间≠0，则再将它加入队列（按优先数大小插入，且置队首标志）；若要求运行时间=0，则把它的状态修改成“结束”（E），且退出队列。
2. 若“就绪”状态的进程队列不为空，则重复上面（4）和（5）的步骤，直到所有进程都成为“结束”状态。
3. 在所设计的程序中应有显示或打印语句，能显示或打印每次被选中进程的进程名以及运行一次后进程队列的变化。
4. 为五个进程任意确定一组“优先数”和“要求运行时间”，启动所设计的处理机调度程序，显示或打印逐次被选中进程的进程名以及进程控制块的动态变化过程。

# 四、实验分析

（1）进程优先数排列

* 1. 利用选择排序算法，根据进程的优先数，在每次单位时间运行结束后进行一次列队排序。同时进行排序的进程中不包含对已经结束运行的改动，即statue为E时该进程不再参与处理器的争夺。

（2）进程运行与系统运行结束判定

* 1. 进程每运行一次，该进程的优先数降低1，同时要求的运行时间减少1。
  2. 每次运行单位时间后，都对进程进行优先数排列，所以每次运行的进程都为队列的第一个未运行结束的进程。
  3. 进程要求运行时间减到0时则标记状态为E，表示结束运行不再抢夺队列的CPU抢占。同时每有一个进程结束运行时，记录结束个数，直到所有进程都结束运行时则退出系统程序。

# 流程图

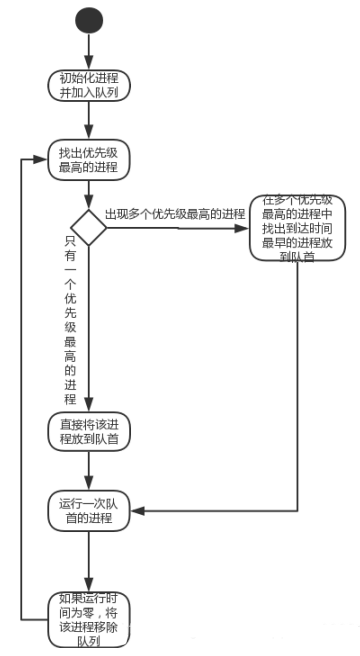


图1-1 优先级调度算法伪代码

# 六、伪代码

（1）定义全局变量：

1. **int** num;    //进程数
2. vector<string>pro;

（2）定义PCB结构体：

1. **struct** PCB{
2. string name;    //进程名
3. PCB\* ptr;   //进程指针
4. **int** time;   //要求运行时间
5. **int** priority;   //优先数
6. string statue;  //状态
7. }tem,pcb[20];

（3）打印进程状态信息：

1. **void** print(PCB\*p)  {
2. printf("|-进程名-|-要求运行时间-|-优先数-|-状态-|\n");
3. **for** i = 0 to num
4. cout <<"|-- "<< pcb[i].name <<" --|------ "<< pcb[i].time<<" ------|-- " << pcb[i].priority  <<" --|-- "<< pcb[i].statue<<"--|\n";
5. }

（4）按优先级排序：

1. **void** queue(PCB\* p) {
2. **for** i = 0 to  num - 1
3. **for** j = i+1 to num
4. **if** pcb[i].prioritypcb[j].priority&&pcb[j].statue!="E"
5. tem = pcb[i];
6. pcb[i] = pcb[j];
7. pcb[j] = tem;
8. }

（5）主函数：

1. **int** main() {
2. cout << "输入进程个数：";
3. cin >> num;
4. cout << "依次输入各个进程信息(进程名 要求运行时间 优先数)：" << endl;
5. PCB\* head=**new** PCB;
6. PCB HEAD;
7. head = &HEAD;
8. **for** i = 0 to num
9. cin >> pcb[i].name;
10. cin >> pcb[i].time;
11. cin >> pcb[i].priority;
12. pcb[i].statue = "R";
13. head = pcb;
14. cout << "\n       进程初始化状态\n";
15. queue(pcb);
16. print(pcb);
17. **int** index = 1,fi = 0;
18. **while** fi != num
19. **while** (head->time == 0)
20. head = head + 1;
21. printf("\n时间：第%d时刻。|当前运行进程为：",index);
22. cout << head->name << endl;
23. print(pcb); //输出当前进程队列信息
24. index++;
25. pro.push\_back(head->name); //进程运行记录
26. head->priority -= 1; //进程运行
27. head->time -= 1;
28. **if** head->time == 0
29. head->statue = "E";
30. fi++;
31. queue(pcb); //进行优先数排列
32. cout << "进程调度顺序：";
33. **For** i=0 to pro.size()
34. cout << pro[i] << " ";
35. system("pause");
36. **return** 0;
37. }

# 实验结果

本程序测试过程中加入了5个进程，程序运行稳定且结果正确。

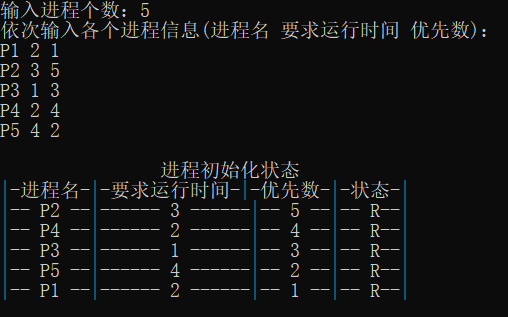


图2-1 进程初始化

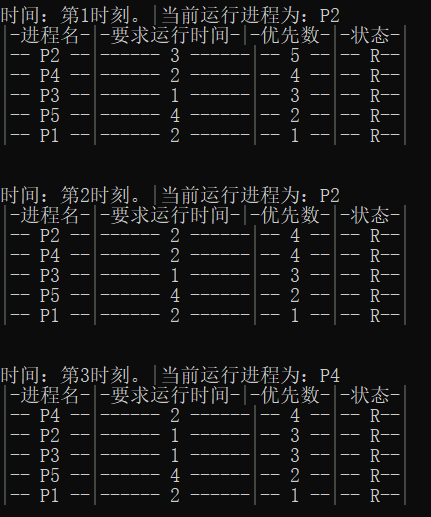


图2-2 1-3时刻进程调度情况

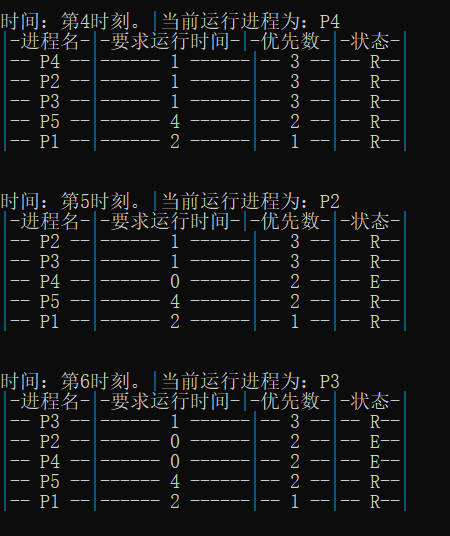


图2-3 第4-6时刻进程调度情况

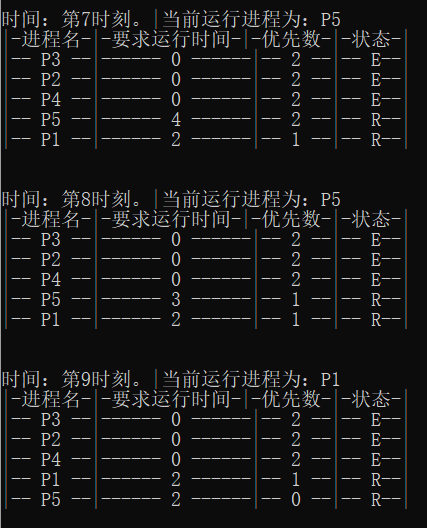


图2-4 第7-9时刻进程调度情况

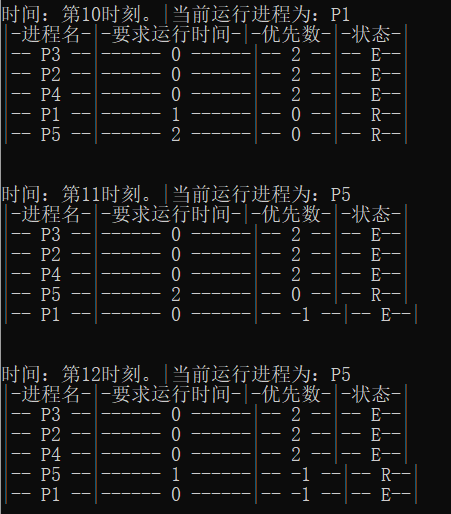


图2-5 第9-12时刻进程调度情况



图2-6 进程调度顺序

# 八、实验体会

本课内讨论我选择的是模拟优先级调度算法实现处理机调度的程序，实验中假设所有创建的进程都已在内存（等待运行的队列）中，每次都将优先级最高的进程放到队首，每运行一次进程就使优先级减一，再进行优先权比较，如果优先级相同则比较到达时间，将优先级最高且到达时间最早的进程放到队首。若出现优先权更高的进程，则立即停止当前执行，并将处理机分配给新到的优先权最高的进程。

综上，通过本次讨论，我对调度问题有了更进一步的理解。调度是为了公平地将处理器分配给多个进程使用、提高响应时间、吞吐量、处理器的效率和低开销。优先级调度算法总是选择具有较高优先级的进程运行，按优先级递减的顺序维护就绪队列，但低优先级的进程可能饿死，所以我们还可以考虑动态优先级：允许进程随着时间和执行历史的变化而动态地修改优先级。除此，调度策略还有FCFS、RR、SPN、SPT、HRRN等，我对他们的调度思想（抢占模式以及选择函数）都进行了了解，并能简单实现。