

**课程设计(综合实验)报告**

( 2019 – 2020 年度第 2 学期)

名 称： 操作系统综合实验

题 目： 处理机调度

院 系： 计算机系

班 级： 软件1802

学 号： 201809000121

学生姓名： 王毅涛

指导教师： 王德文

设计周数： 分散进行

成 绩：

日期： 年 月 日

**一、 综合实验的目的与要求**

在多道程序系统中，调度的实质是一种资源分配，处理机调度是对处理机资源进行分配。在多道批处理系统中，一个作业从提交到获得处理机执行，直至作业运行完毕，可能经历作业调度和进程调度。本实验模拟在单处理器情况下的处理器调度，帮助学生加深了解处理器调度的工作。

1.1设计一个按高响应比优先算法实现作业调度的程序

1.2设计一个按时间片轮转法实现进程调度的程序

**二、实验正文**

1.实验中采用的数据结构

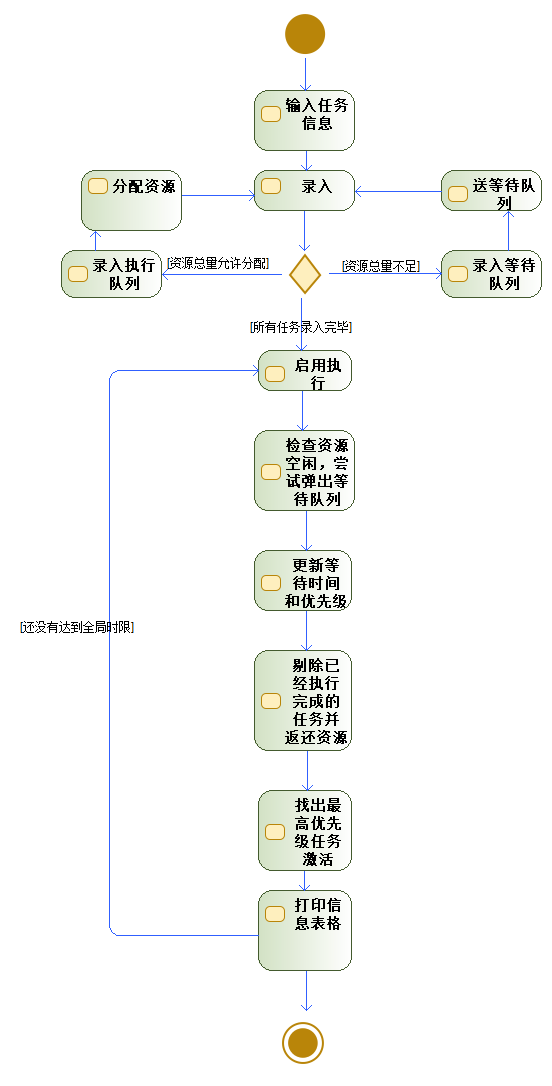
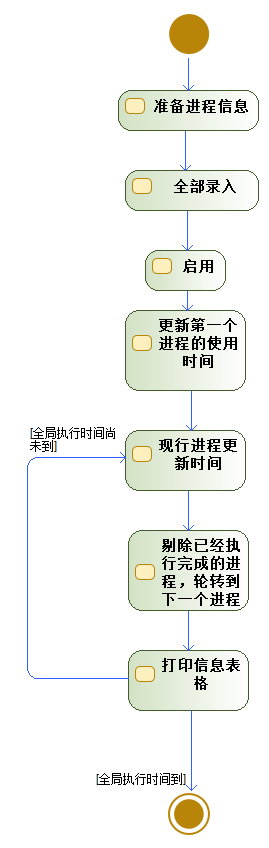
第一部分的JCB在Java中用类进行封装，封装了名称，大小，磁带机和打印机的占用量，预计使用时间，抵达时间，等待时间，开始执行的时刻，是否在执行的标志位，优先级数，以及指针。

第二部分的PCB在JAVA中用类封装，封装了名称，指针，要求运行时间，已经运行时间，状态字。

2.流程图

第一部分：左图

第二部分：右图



2.思路：

第一部分：在准备好实验参数后初始化，将JCB单元通过控制单元加入到控制单元的队列中，注意，此处由于有资源利用限制，所以我在默认的主队列外加了一个等待队列，并且默认先来的先分配资源（主存容量），如果后来的无法全部取得资源，则暂时挂入等待队列。主队列存放的是已经预先取得资源，但是不一定现行执行的任务。等待队列存放暂时无法满足资源需求的任务。客户端命令控制单元开始“执行”，控制单元完成如下操作，直到全局时间达到时限。遍历等待队列，如果有已经可以满足资源需求的任务就加入主队列；遍历主队列任务，根据当前时间更新所有的，正在执行（已经到预期执行时间）的任务的等待时间和优先级数；根据当前时间剔除已经执行完成（当前时间-到达时间==预期执行时间）的任务；遍历主队列找出优先级最高的，将其激活，也就是使之状态为正在运行，与此同时，将系统资源（磁带机打印机）真实地分配给它，如果没有足够的资源，就提示错误并且跳到下一阶段；打印主队列中所有的任务的状态。

在本次实验中，我将程序全局时限定为30个单位。

第二部分，在初始化完成所有参数后，将PCB单元通过控制单元的函数加入到主队列。主队列是一个带头节点的循环链表。客户端命令控制单元开始执行后，首先对第一个节点更新信息，然后进入以下循环，直至全局时间达到实现：更新当前进程的时间信息；检查当前进程有没有执行完成（局部运行时间也没有达到局部时限），并且将“当前进程”指针指向下一个进程块；打印队列信息。

运行流程见附录

**三、综合实验总结或结论**

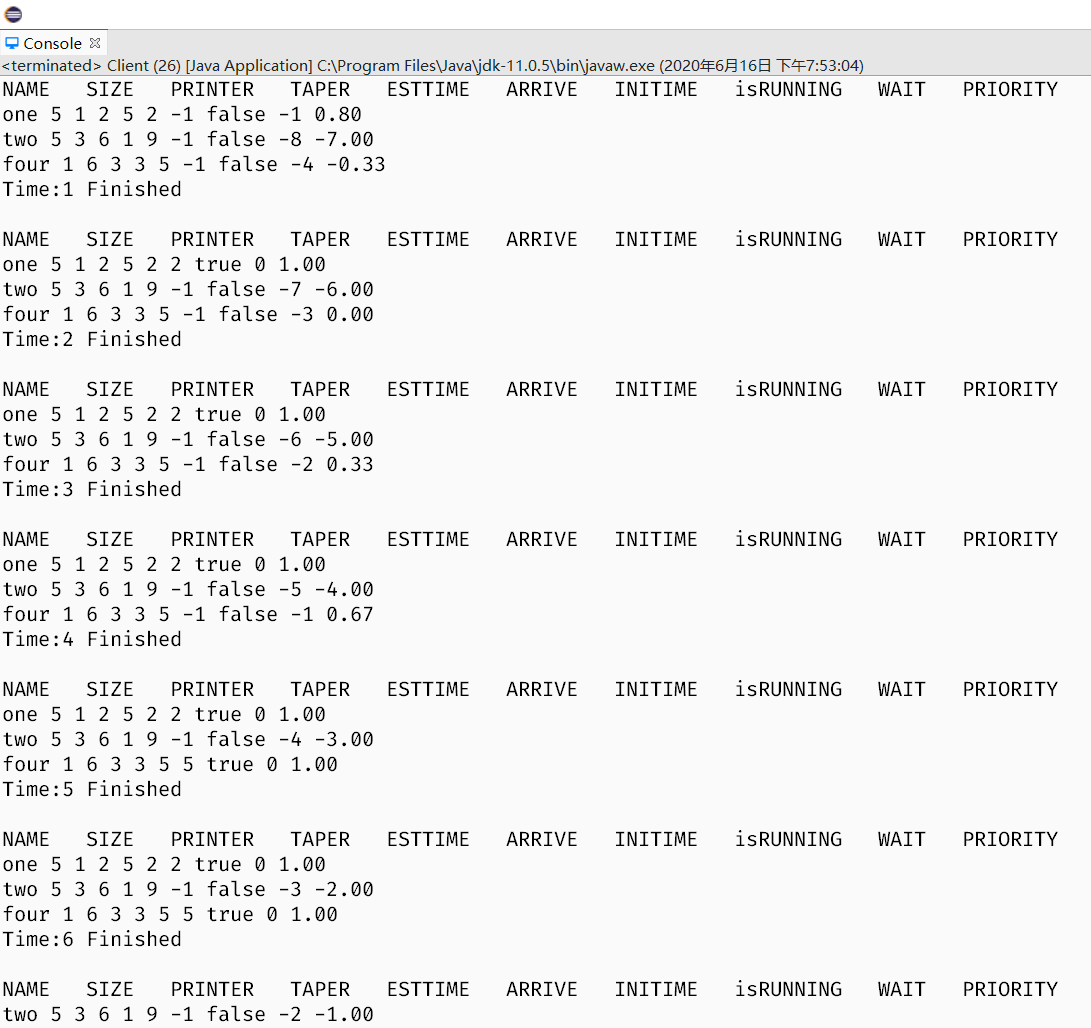
实验基本达到预期目的，总体来看，第一部分难度偏高，又缺少实验平台，无疑给调试增加了难度，好在我经过几次试验，找出来几个BUG。

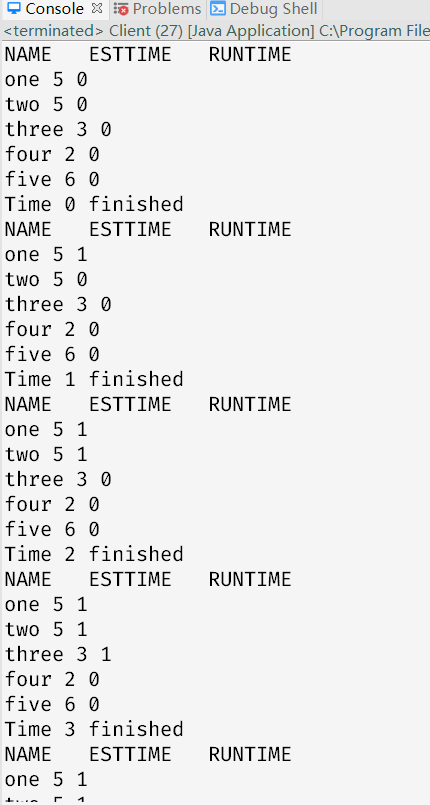
**四、参考文献**

[1] 汤小丹, 梁红兵.《计算机操作系统》. 西安电子科技大学出版社, 第四版. 2014-05

**附录（设计流程图、程序、表格、数据等）**

以下为运行实况：(已经达到运行时限的任务或者进程直接剔除出主队列，不再另外打印)

HRRN:

RR:

以下为代码：

JCB:

1. **package** oneHRRN;
3. **public** **class** JCB {
4. String name;
5. **int** size;
6. **int** pReq;//需要的打印机
7. **int** tReq;//需要的磁带机
8. **int** estTime;//预计使用时间
9. **int** arriveTime;//应当抵达时间
10. **int** waitTime;//等待时间（需要更新）
11. **int** iniTime;//开始执行的时刻
12. **boolean** isRunning;//是不是在执行
13. JCB next;//下一个的指针
14. **float** priority;//优先级需要实时更新
15. **public** JCB() {
16. **super**();
17. next=**null**;
18. }
19. **public** JCB(String name, **int** size, **int** pReq, **int** tReq, **int** estTime, **int** arriveTime) {
20. **super**();
21. **this**.name = name;
22. **this**.size = size;
23. **this**.pReq = pReq;
24. **this**.tReq = tReq;
25. **this**.estTime = estTime;
26. **this**.arriveTime = arriveTime;
27. **this**.priority=(**float**) 1;
28. **this**.isRunning=**false**;
29. **this**.waitTime=0;
30. **this**.iniTime=-1;
31. **this**.next=**null**;
32. }
33. **public** **void** setNext(JCB next) {
34. **this**.next = next;
35. }
37. **public** **int** getIniTime() {
38. **return** iniTime;
39. }
40. **public** **void** setIniTime(**int** iniTime) {
41. **this**.iniTime = iniTime;
42. }
43. **public** **void** updatePrior() {
44. **this**.priority=(**float**) (1+(**float**)(**this**.waitTime)/(**float**)(**this**.estTime));
45. }
46. **public** **void** updatewaitTime(**int** currentTime) {
47. **this**.waitTime=currentTime-**this**.arriveTime;
48. }
49. **public** **boolean** isRunning() {
50. **return** isRunning;
51. }
52. **public** **void** setRunning(**boolean** isRunning) {
53. **this**.isRunning = isRunning;
54. }


58. }

高优先级调度：

1. **public** **class** Conductor {
2. **int** printer;
3. **int** tape;
4. **int** volumn;
5. **int** curVolumn;
6. //JCB executing;
7. JCB queue;//加队列时，放不下的就塞到这里，有任务退出再尝试加入主队列
8. JCB head;
9. JCB display;
10. **public** Conductor(**int** printer, **int** tape, **int** volumn) {
11. **super**();
12. **this**.printer = printer;
13. **this**.tape = tape;
14. **this**.volumn = volumn;
15. **this**.curVolumn=0;
16. **this**.head=**new** JCB();
17. **this**.queue=**new** JCB();
18. }
19. **public** **void** addQueue(JCB adder) {//在末尾添加，并且清空该JCB的NEXT
20. JCB block=**this**.queue;
21. **while** (block.next!=**null**) {block=block.next;}
22. block.next=adder;
23. adder.next=**null**;
24. }
25. **public** **void** popQueue() {
26. JCB block=**this**.queue;
27. **while** (block.next!=**null**) {
28. **if** (**this**.appendAllowed(block.next)){
29. **this**.append(block.next);
30. JCB tmp=block.next.next;
31. block.next=tmp;
33. //return;
34. }
35. **else**
36. {block=block.next;}
37. }
38. }
39. **public** **boolean** appendAllowed(JCB appex) {
40. **if** (**this**.curVolumn+appex.size>**this**.volumn)
41. {**return** **false**;}
42. **else** {**return** **true**;}
43. }
44. **public** **boolean** append(JCB appex) {
45. **if** (**this**.curVolumn+appex.size>**this**.volumn) {**this**.addQueue(appex);**return** **false**;}
46. **else** {
47. **this**.curVolumn+=appex.size;
48. JCB block=**this**.head;
49. **while** (block.next!=**null**) {block=block.next;}
50. block.next=appex;
51. appex.next=**null**;
52. **return** **true**;
53. }
54. }
55. **public** **void** execute() {
56. //挨个遍历，找出优先级最高的，然后检查资源有没有齐全(两个都要满足)，没有就等待释放，有就分配资源
57. //加入的进程一概设定其已经在运作,设定起始时间
58. //释放意味着时间到，将其排除出队列，资源还给系统
59. //执行的最后，遍历更新所有节点的WAIT和PRIOIRTY
60. //假设只允许执行30秒，也就是计数30次循环
61. **for** (**int** curTime=1;curTime<=30;curTime++) {
62. **this**.popQueue();
63. **this**.updateAllWaitAndPrior(curTime);
64. **this**.sweep(curTime);
65. JCB blc=**this**.seekHigh();
66. **this**.activate(curTime, blc);
67. **this**.printme(curTime);
69. }
70. }
71. **public** **void** sweep(**int** curTime) {
72. //清扫已经达到运行时间的任务
73. JCB block=**this**.head;
74. **while** (block.next!=**null**) {
75. **if** ((curTime-block.next.iniTime>=block.next.estTime)&&(block.next.isRunning==**true**)) {
76. **this**.printer+=block.next.pReq;
77. **this**.tape+=block.next.tReq;
78. **this**.curVolumn-=block.next.size;
79. block.next=block.next.next;
80. //资源归还后剔除
81. }
82. //          else if ((curTime-block.next.iniTime==block.next.estTime)&&(block.next.isRunning==true)&&(block.next.next==null))
83. //          {block.next=null;}
84. **else** {block=block.next;;}
86. }
87. }
88. **public** **void** updateAllWaitAndPrior(**int** curTime) {//等待时间有可能设为负数，那么就代表没有入场
89. //要求没有执行的任务
90. JCB block=**this**.head.next;
91. **while** (block!=**null**) {
92. //          if (block.arriveTime==curTime) {
93. //              block.iniTime=curTime;
94. //              block.setRunning(true);
95. //          }else {;}
96. //要不要激活由EXECUTION函数决定
97. **if** (block.isRunning) {block=block.next;**continue**;}
98. **else** {;}
99. block.updatewaitTime(curTime);
100. block.updatePrior();
101. block=block.next;
103. }
104. }
105. **public** **void** printme(**int** curTime) {
106. //System.out.println();
107. System.out.println("NAME   SIZE   PRINTER   TAPER   ESTTIME   ARRIVE   INITIME   isRUNNING   WAIT   PRIORITY");
108. JCB block=**this**.head.next;
109. **while** (block!=**null**) {
110. System.out.printf("%s %d %d %d %d %d %d %b %d %.2f",block.name,block.size,block.pReq,
111. block.tReq,block.estTime,block.arriveTime,
112. block.iniTime,block.isRunning,block.waitTime,block.priority);
113. System.out.println();
114. block=block.next;
115. }
116. System.out.println("Time:"+curTime+" Finished"+"\n");

119. }
120. **public** **void** activate(**int** Time,JCB block) {
121. **if** (block==**null**) {**return**;}
122. **if** (**this**.tape-block.tReq>=0&&**this**.printer-block.pReq>=0) {
123. block.isRunning=**true**;
124. block.iniTime=Time;
125. **this**.tape-=block.tReq;
126. **this**.printer-=block.pReq;
127. }
128. **else** {
129. System.out.println("Resource Lacking,activation suspended");
130. }
132. }
133. **public** JCB seekHigh() {
134. //找未开始执行的
135. JCB block =**this**.head.next;
136. String name = **null**;
137. **float** prio=-114;
138. **while** (block!=**null**) {
139. **if** (block.priority>prio&&block.isRunning==**false**) {name=block.name;prio=block.priority;}
140. **else** {;}
141. block=block.next;
142. }
143. **if** (prio<1) {**return** **null**;}
144. **else** {
145. block=**this**.head.next;
146. **while**(block!=**null**) {
147. **if** (block.name.equals(name)&&block.priority==prio&&block.isRunning==**false**) {
148. **return** block;
149. }**else** {block=block.next;}
150. }
152. **return** **null**;
153. }
155. }
156. }

PCB：

1. **public** **class** PCB {
2. String name;
3. PCB next;
4. **int** estTime;//要求运行
5. **int** runTime;//已经运行
6. String status;//R就绪，E结束，一开始都是R
7. **public** PCB(String name, **int** estTime) {
8. **super**();
9. **this**.name = name;
10. //this.next = next;
11. **this**.estTime = estTime;
12. **this**.runTime=0;
13. **this**.status = "R";
14. }
15. **public** PCB() {
16. **super**();
17. }
19. **public** PCB getNext() {
20. **return** next;
21. }
22. **public** **void** setNext(PCB next) {
23. **this**.next = next;
24. }
25. **public** **void** update() {
26. **if** (**this**.status.equals("R")&&(**this**.estTime>**this**.runTime)) {
27. **this**.runTime+=1;
28. **if** (**this**.runTime>=**this**.estTime) {
29. **this**.status="E";
30. }
31. **else** {;}
32. }
33. **else** {**return**;}
34. }
35. **public** **void** printme() {
36. System.out.printf("%s %d %d %s",**this**.next.name,**this**.next.estTime,**this**.next.runTime,**this**.status);
37. System.out.println();
38. }
39. }

轮转调度：

1. **package** oneRR;

4. **public** **class** Conductor {
5. PCB head;
6. PCB running;
7. **public** Conductor() {
8. **this**.head=**new** PCB();
9. **this**.running=**new** PCB();
10. head.next=head;
11. running=head;
12. }
13. **public** **void** execute() {
14. //我们假设执行时间30S
15. //一开始先排好循环链表
16. //然后在客户端操作加好循环链表
17. //一开始先转移到第一个PCB，对他做UPDATE
18. //然后尝试对它清扫，转移到下一个PCB
19. //打印
20. //如果要扫除，先判断是不是和RUNNING重合，是的话必须迁移RUNNING
21. //最后要在客户端放一个序列打印
22. **this**.updateRunning();
23. //      this.running.update();
24. **this**.printme(0);
25. **for** (**int** i=1;i<=30;i++) {
26. **this**.running.update();
27. **this**.sweep();
28. **this**.printme(i);
29. }
30. }
32. **public** **boolean** append(PCB block) {
33. **if** (head.next==head) {
34. head.setNext(block);
35. block.setNext(head);
36. **return** **true**;
37. }
38. **else** {
39. PCB tmp=head;
40. **while** (tmp.next!=head) {
42. tmp=tmp.next;
43. }
44. tmp.next=block;
45. block.next=head;
46. **return** **true**;
47. }
48. }
49. **public** **void** updateRunning() {//只能把RUNNING往后推一格
50. **if** (head.next==head) {
51. **return**;//没有可用节点
52. }
53. **else** **if** (head==running) {
54. running=head.next;//对付一开始虽然已经添加完毕但RUNNING还在HEAD上
55. }
56. **else** {
57. PCB tmp=running.next;
58. **if** (tmp==head)
59. {
60. tmp=tmp.next;
61. **this**.running=tmp;
62. }
63. **else**{running=tmp;}
64. }
65. }
66. **public** **void** sweep() {//把已经运行完成的进程赶出队列,并且无论如何也要把RUNNING后推
67. **if** (running.status.equals("E")) {
68. PCB tmp=**this**.head;
69. **if** (tmp.next==tmp) {**return**;}**else** {;}
70. **while** (tmp.next!=running) {
71. tmp=tmp.next;
72. }
73. **this**.updateRunning();
74. tmp.next=tmp.next.next;
75. }
76. **else** {
77. **this**.updateRunning();
78. }
79. }
80. **public** **void** printme(**int** curTime) {
81. System.out.println("NAME   ESTTIME   RUNTIME");
82. //System.out.printf("%s %d %d %d %d %d %d %b %d %.2f");
83. PCB tmp=**this**.head;
84. **while** (tmp.next!=**this**.head) {
85. System.out.printf("%s %d %d",tmp.next.name,tmp.next.estTime,tmp.next.runTime);
86. System.out.println();
87. tmp=tmp.next;
88. }
89. System.out.println("Time "+curTime+" finished");
90. }




96. }