**江苏科技大学**

**课程实验报告**

课 程： 操作系统原理与安全

课 题： 实验一·进程调度

学 院： 计算机学院

姓 名： 陈四贵

班 级： 1822107101

学 号： 182210710119

开发语言： Java

目 录

[一、 实验目的 1](#_Toc41664432)

[二、 实验内容 1](#_Toc41664433)

[1．优先权法、轮转法 1](#_Toc41664434)

[2．算法描述 1](#_Toc41664435)

[三、 流程图 1](#_Toc41664436)

[四、 实验要求 2](#_Toc41664437)

[五、 实验代码及其说明 2](#_Toc41664438)

[1．总体结构 2](#_Toc41664439)

[2．分支结构 2](#_Toc41664440)

[⑴ PCB类 2](#_Toc41664441)

[核心思想 2](#_Toc41664442)

[附·源代码 3](#_Toc41664443)

[⑵ Priority类 3](#_Toc41664444)

[核心思想 3](#_Toc41664445)

[附·源代码 3](#_Toc41664446)

[⑶ Rotation类 7](#_Toc41664447)

[核心思想 7](#_Toc41664448)

[附·源代码 7](#_Toc41664449)

[⑷主类中的main函数 13](#_Toc41664450)

[核心思想 13](#_Toc41664451)

[附·源代码 13](#_Toc41664452)

[六、 运行截图 14](#_Toc41664453)

[1.动态优先权法 14](#_Toc41664454)

[2.轮转法 16](#_Toc41664455)

[七、 实验心得 17](#_Toc41664456)

# 实验目的

多道程序设计中，经常是若干个进程同时处于就绪状态，必须依照某种策略来决定那个进程优先占有处理机。因而引起进程调度。本实验模拟在单处理机情况下的处理机调度问题，加深对进程调度的理解。

# 实验内容

## 1．优先权法、轮转法

简化假设

1）进程为计算型的（无 I/O）；

2）进程状态：ready、running、finish

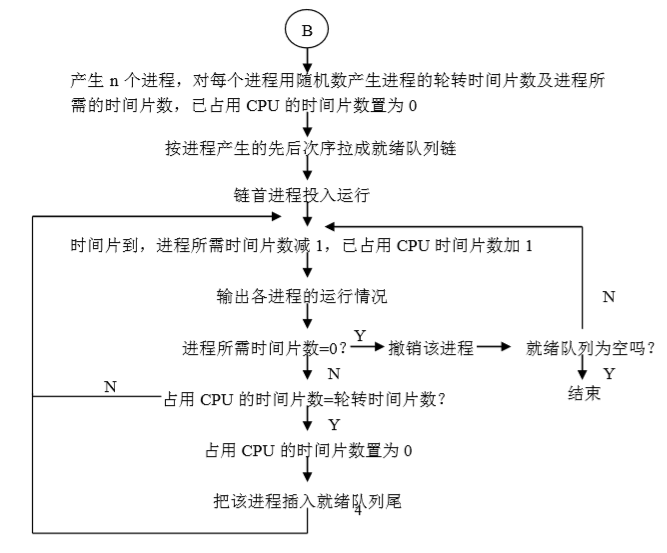
3）进程需要的 CPU 时间以时间片为单位确定。

## 2．算法描述

1）优先权法——动态优先权 当前运行进程用完时间片后，其优先权减去一个常数

2）轮转法。

# 流程图



# 实验要求

1．产生的各种随机数的取值范围加以限制，如所需的 CPU 时间限制在 1~20 之间。

2．进程数 n 不要太大通常取 4~8 个

3．使用动态数据结构

4．独立编程

5．至少三种调度算法

6．若有可能请在图形方式下，将 PCB 的调度用图形成动画显示。

# 实验代码及其说明

## 1．总体结构

整个实验采用面向对象的编程技术，按功能将各个部分模块化，分为Main类（初始化、启动类、主类）、PCB类、Priority类（优先权法调度类）、Rotation类（轮转法调度类）。其中，由于只使用了一个Java文件，PCB类、Priority类、Rotation类是作为内部类存在于Main类中的。

## 2．分支结构

### ⑴ PCB类

核心思想：通过参照linux系统中进程控制块PCB的结构，自定义了一个简化版的PCB结构体，去除了本实验中不需要用到的数据成员，仅保留了优先权、时间片、轮转次数等几个核心数据成员，用以模拟进程——创建一个对象即相当于创建一个进程。

附·源代码：

1. **public** **static** **class** PCB {
2. **public** **int** Pid;  //进程名
3. **public** **char** State;  //所处状态，A为ready,B为running,C为finish
4. **public** **int** ExeTime;  //时间片、运行时间
5. **public** **int** PriorityPower;  //动态产生的优先权
6. **public** **int** Frequency;   //动态产生的轮转次数
7. **public** **int** Counter;   //计数，记录已轮转次数
8. **public** PCB next;
10. **public** PCB(**int** pid){
11. Random random=**new** Random();
12. **this**.Pid=pid;
13. **this**.State='A';
14. **this**.ExeTime= (random.nextInt(5)+1);  //避免运行时间为0
15. **this**.PriorityPower=random.nextInt(30);
16. **this**.Frequency=random.nextInt(6)+1;
17. **this**.Counter=0;
18. **this**.next=**null**;
19. }
20. }

### ⑵ Priority类

核心思想：

①本类主要模拟动态优先权法调度进程，使用了数据结构链表，且构建了两个单链表——一个表示就绪“队列”，另一个表示完成队列。

②按照优先权的大小动态构建就绪队列（就绪队列并非数据结构中的队列，只是说明一个集合关系，实质是没有限制的单链表），链首的始终为优先权最高的进程，链尾的始终为优先权最低的进程。

③当链首进程运行所需CPU时间耗尽时，如果时间片未耗尽，将该进程按核减后的优先权插入到就绪队列中，反之则直接放入完成队列。

④由于存在两个有一定关系的链表，在打印进程调度信息时，应判断完成队列是否为空，若为空则不打印，反之则遍历打印。为实现这一功能，且保证插法尽可能一致，采用了带头指针的链表。

附·源代码：

1. **public** **static** **class** Priority {//优先权调度类
2. PCB first\_1 = **new** PCB(0);  //就绪队列队头，链首处于运行状态，其他为就绪
3. PCB first\_2= **new** PCB(0);   //完成队列
5. **public** Priority(**int** num) {
6. //创建就绪队列(实质是链表）
7. PCB p;
8. PCB q;
9. PCB temp;
10. **for** (**int** i = 1; i <= num; i++) {
11. p = first\_1;
12. q = **new** PCB(i);
13. **if** (p.next == **null**) {//头节点后没有，直接插
14. p.next = q;
15. } **else** {
16. p=p.next;
17. **if**(q.PriorityPower>=p.PriorityPower){//头插
18. q.next=p;
19. first\_1.next=q;
20. **continue**;
21. }
22. **while** (p.next != **null**) {//中间插
23. **if** (q.PriorityPower <= p.PriorityPower && q.PriorityPower > p.next.PriorityPower) {
24. q.next = p.next;
25. p.next = q;
26. **break**;
27. } **else** {
28. p = p.next;
29. }
30. }
31. **if** (p.next == **null**) {//尾插
32. p.next = q;
33. }
34. }
35. }
37. //交换完毕后，打印各进程的优先权，测试是否成功
38. p = first\_1;
39. **while** (p.next != **null**) {
40. p = p.next;
41. System.out.println("进程" + p.Pid + "的初始优先权为" + p.PriorityPower + "，进程所需CPU时间为" + p.ExeTime+"，状态:ready");
42. }
43. System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");
44. //模拟运行、调度
45. **long** startTime,endTime,last;
46. **while**(first\_1.next!=**null**) {
47. p = first\_1.next;
48. startTime = System.currentTimeMillis();
49. **do** {
50. endTime = System.currentTimeMillis();
51. last = endTime - startTime;
52. p.State = 'B';
53. } **while** (last < p.ExeTime);
55. q = p;
56. System.out.println("进程运行中，CPU正被占用。");
57. System.out.println("进程" + p.Pid + "的优先权为" + p.PriorityPower + "，进程所需CPU时间为" + p.ExeTime + "，状态：running。进程运行时间："+p.ExeTime);
58. **while** (q.next != **null**) {
59. q = q.next;
60. System.out.println("进程" + q.Pid + "的优先权为" + q.PriorityPower + "，进程所需CPU时间为" + q.ExeTime + "，状态：ready");
61. }
62. q=first\_2;
63. **if**(q.next==**null**);
64. **else**{
65. **while**(q.next!=**null**){
66. q=q.next;
67. System.out.println("进程" + q.Pid + "的优先权为" + q.PriorityPower + "，进程所需CPU时间为" + q.ExeTime + "，状态：finish");
68. }
69. }
70. System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");
72. p.PriorityPower -= 3;
73. p.ExeTime -= 1;
74. **if** (p.ExeTime <= 0) {//运行时间到，剔除出队列，插入完成队列
75. p.State = 'C';
76. first\_1.next = p.next;
77. p.next=**null**;
78. q=p;
79. //插入完成队列
80. p=first\_2;
81. **while**(p.next!=**null**){
82. p=p.next;
83. }
84. p.next=q;
85. } **else** {//时间未归零，插入就绪队列
86. first\_1.next = first\_1.next.next;
87. q = p;
88. q.next = **null**;
89. p = first\_1;
90. **if** (p.next == **null**) {//头节点后没有，直接插
91. p.next = q;
92. q.State = 'B';
93. } **else** {
94. p = p.next;
95. **if** (q.PriorityPower >= p.PriorityPower) {//头插
96. q.next = p;
97. first\_1.next = q;
98. } **else** {
99. **while** (p.next != **null**) {//中间插
100. **if** (q.PriorityPower <= p.PriorityPower && q.PriorityPower >= p.next.PriorityPower) {
101. q.next = p.next;
102. p.next = q;
103. q.State = 'A';
104. **break**;
105. } **else** {
106. p = p.next;
107. }
108. }
109. **if** (p.next == **null**) {//
110. q.next = **null**;
111. p.next = q;
112. q.State = 'A';
113. }
114. }
115. }
116. }
117. p = first\_1;q=first\_2;
118. System.out.println("进程运行完毕后，CPU调度下一进程前。");
119. **if** (p.next == **null**) {
120. **while**(q.next!=**null**){
121. q=q.next;
122. System.out.println("进程" + q.Pid + "的初始优先权为" + q.PriorityPower + "，进程所需CPU时间为" + q.ExeTime + "，状态：finish");
123. }
124. } **else** {
125. **while** (p.next != **null**) {
126. p = p.next;
127. System.out.println("进程" + p.Pid + "的初始优先权为" + p.PriorityPower + "，进程所需CPU时间为" + p.ExeTime + "，状态：ready");
128. }
129. **while**(q.next!=**null**){
130. q=q.next;
131. System.out.println("进程" + q.Pid + "的初始优先权为" + q.PriorityPower + "，进程所需CPU时间为" + q.ExeTime + "，状态：finish");
132. }
133. System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");
134. }
135. }
136. }
137. }

### ⑶ Rotation类

核心思想：基本同Priority类，只是在判断条件上有细微区别。

附·源代码：

1. **public** **static** **class** Rotation {//轮转法调度类
2. PCB p, q, first\_1, first\_2;
4. **public** Rotation(**int** num) {
5. first\_1 = **new** PCB(0);
6. first\_2 = **new** PCB(0);
7. **for** (**int** i = 1; i <= num; i++) {
8. q = **new** PCB(i);
9. p = first\_1;
10. **while** (p.next != **null**) {
11. p = p.next;
12. }
13. p.next = q;
14. }
15. //测试构成情况
16. p = first\_1;
18. **while** (p.next != **null**) {
19. p = p.next;
20. System.out.println("进程" + p.Pid + "的轮转次数为" + p.Frequency + "，进程所需CPU时间为" + p.ExeTime + "，状态:ready");
21. }
22. System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");
24. **long** startTime, endTime, last;
26. **while**(first\_1.next != **null**) {
27. p = first\_1.next;
28. startTime = System.currentTimeMillis();
29. **do** {
30. endTime = System.currentTimeMillis();
31. last = endTime - startTime;
32. } **while** (last < p.ExeTime);
33. p.State = 'B';
34. System.out.println("进程运行中，CPU正被占用。");
35. System.out.println("进程" + p.Pid + "的轮转次数为" + p.Frequency + "，进程所需CPU时间为" + p.ExeTime + "，状态：running。进程运行时间：" + p.ExeTime);
36. q=p;
37. **while** (q.next != **null**) {
38. q = q.next;
39. System.out.println("进程" + q.Pid + "的轮转次数为" + q.Frequency + "，进程所需CPU时间为" + q.ExeTime + "，状态：ready");
40. }
41. q = first\_2;
42. **if** (q.next == **null**) ;
43. **else** {
44. **while** (q.next != **null**) {
45. q = q.next;
46. System.out.println("进程" + q.Pid + "的轮转次数为" + q.PriorityPower + "，进程所需CPU时间为" + q.ExeTime + "，状态：finish");
47. }
48. }
49. System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");
51. p.ExeTime -= 1;
52. p.Counter++;
53. **if** (p.ExeTime <= 0) {//时间片到了，不能再继续运行
54. first\_1.next = first\_1.next.next;
55. p.next = **null**;
56. p.State = 'C';
57. q = first\_2;
58. **while** (q.next != **null**) {
59. q = q.next;
60. }
61. q.next = p;
62. **if** (first\_1.next == **null**) {
63. }
64. } **else** {//时间片未到
65. **if** (p.Frequency == p.Counter) {//占用时间片等于轮转次数，放至就绪队列队尾
66. first\_1.next = first\_1.next.next;
67. p.Counter = 0;
68. p.State = 'A';
69. p.next = **null**;
70. q = first\_1;
71. **while** (q.next != **null**) {
72. q = q.next;
73. }
74. q.next = p;
75. } **else** {//该进程继续运行
76. }
77. }
78. p = first\_1;
79. q = first\_2;
80. System.out.println("进程运行完毕后，CPU调度下一进程前。");
81. **if** (p.next == **null**) {//就绪队列为空
82. **while** (q.next != **null**) {
83. q = q.next;
84. System.out.println("进程" + q.Pid + "的轮转数为" + q.Frequency + "，占用时间片数为" + q.Counter + "，进程所需CPU时间为" + q.ExeTime + "，状态：finish");
85. }
86. } **else** {//就绪队列不为空
87. **while** (p.next != **null**) {
88. p = p.next;
89. System.out.println("进程" + p.Pid + "的轮转数为" + p.Frequency + "，占用时间片数为" + p.Counter + "，进程所需CPU时间为" + p.ExeTime + "，状态：ready");
90. }
91. **while** (q.next != **null**) {
92. q = q.next;
93. System.out.println("进程" + q.Pid + "的轮转数" + q.Frequency + "，占用时间片数为" + q.Counter + "，进程所需CPU时间为" + q.ExeTime + "，状态：finish");
94. }
95. System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");
96. }
97. }
98. }
99. }**public** **static** **class** Priority {//优先权调度类
100. PCB first\_1 = **new** PCB(0);  //就绪队列队头，链首处于运行状态，其他为就绪
101. PCB first\_2= **new** PCB(0);   //完成队列
103. **public** Priority(**int** num) {
104. //创建就绪队列(实质是链表）
105. PCB p;
106. PCB q;
107. PCB temp;
108. **for** (**int** i = 1; i <= num; i++) {
109. p = first\_1;
110. q = **new** PCB(i);
111. **if** (p.next == **null**) {//头节点后没有，直接插
112. p.next = q;
113. } **else** {
114. p=p.next;
115. **if**(q.PriorityPower>=p.PriorityPower){//头插
116. q.next=p;
117. first\_1.next=q;
118. **continue**;
119. }
120. **while** (p.next != **null**) {//中间插
121. **if** (q.PriorityPower <= p.PriorityPower && q.PriorityPower > p.next.PriorityPower) {
122. q.next = p.next;
123. p.next = q;
124. **break**;
125. } **else** {
126. p = p.next;
127. }
128. }
129. **if** (p.next == **null**) {//尾插
130. p.next = q;
131. }
132. }
133. }
135. //交换完毕后，打印各进程的优先权，测试是否成功
136. p = first\_1;
137. **while** (p.next != **null**) {
138. p = p.next;
139. System.out.println("进程" + p.Pid + "的初始优先权为" + p.PriorityPower + "，进程所需CPU时间为" + p.ExeTime+"，状态:ready");
140. }
141. System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");
142. //模拟运行、调度
143. **long** startTime,endTime,last;
144. **while**(first\_1.next!=**null**) {
145. p = first\_1.next;
146. startTime = System.currentTimeMillis();
147. **do** {
148. endTime = System.currentTimeMillis();
149. last = endTime - startTime;
150. p.State = 'B';
151. } **while** (last < p.ExeTime);
153. q = p;
154. System.out.println("进程运行中，CPU正被占用。");
155. System.out.println("进程" + p.Pid + "的优先权为" + p.PriorityPower + "，进程所需CPU时间为" + p.ExeTime + "，状态：running。进程运行时间："+p.ExeTime);
156. **while** (q.next != **null**) {
157. q = q.next;
158. System.out.println("进程" + q.Pid + "的优先权为" + q.PriorityPower + "，进程所需CPU时间为" + q.ExeTime + "，状态：ready");
159. }
160. q=first\_2;
161. **if**(q.next==**null**);
162. **else**{
163. **while**(q.next!=**null**){
164. q=q.next;
165. System.out.println("进程" + q.Pid + "的优先权为" + q.PriorityPower + "，进程所需CPU时间为" + q.ExeTime + "，状态：finish");
166. }
167. }
168. System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");
170. p.PriorityPower -= 3;
171. p.ExeTime -= 1;
172. **if** (p.ExeTime <= 0) {//运行时间到，剔除出队列，插入完成队列
173. p.State = 'C';
174. first\_1.next = p.next;
175. p.next=**null**;
176. q=p;
177. //插入完成队列
178. p=first\_2;
179. **while**(p.next!=**null**){
180. p=p.next;
181. }
182. p.next=q;
183. } **else** {//时间未归零，插入就绪队列
184. first\_1.next = first\_1.next.next;
185. q = p;
186. q.next = **null**;
187. p = first\_1;
188. **if** (p.next == **null**) {//头节点后没有，直接插
189. p.next = q;
190. q.State = 'B';
191. } **else** {
192. p = p.next;
193. **if** (q.PriorityPower >= p.PriorityPower) {//头插
194. q.next = p;
195. first\_1.next = q;
196. } **else** {
197. **while** (p.next != **null**) {//中间插
198. **if** (q.PriorityPower <= p.PriorityPower && q.PriorityPower >= p.next.PriorityPower) {
199. q.next = p.next;
200. p.next = q;
201. q.State = 'A';
202. **break**;
203. } **else** {
204. p = p.next;
205. }
206. }
207. **if** (p.next == **null**) {//
208. q.next = **null**;
209. p.next = q;
210. q.State = 'A';
211. }
212. }
213. }
214. }
215. p = first\_1;q=first\_2;
216. System.out.println("进程运行完毕后，CPU调度下一进程前。");
217. **if** (p.next == **null**) {
218. **while**(q.next!=**null**){
219. q=q.next;
220. System.out.println("进程" + q.Pid + "的初始优先权为" + q.PriorityPower + "，进程所需CPU时间为" + q.ExeTime + "，状态：finish");
221. }
222. } **else** {
223. **while** (p.next != **null**) {
224. p = p.next;
225. System.out.println("进程" + p.Pid + "的初始优先权为" + p.PriorityPower + "，进程所需CPU时间为" + p.ExeTime + "，状态：ready");
226. }
227. **while**(q.next!=**null**){
228. q=q.next;
229. System.out.println("进程" + q.Pid + "的初始优先权为" + q.PriorityPower + "，进程所需CPU时间为" + q.ExeTime + "，状态：finish");
230. }
231. System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");
232. }
233. }
234. }
235. }

### ⑷主类中的main函数

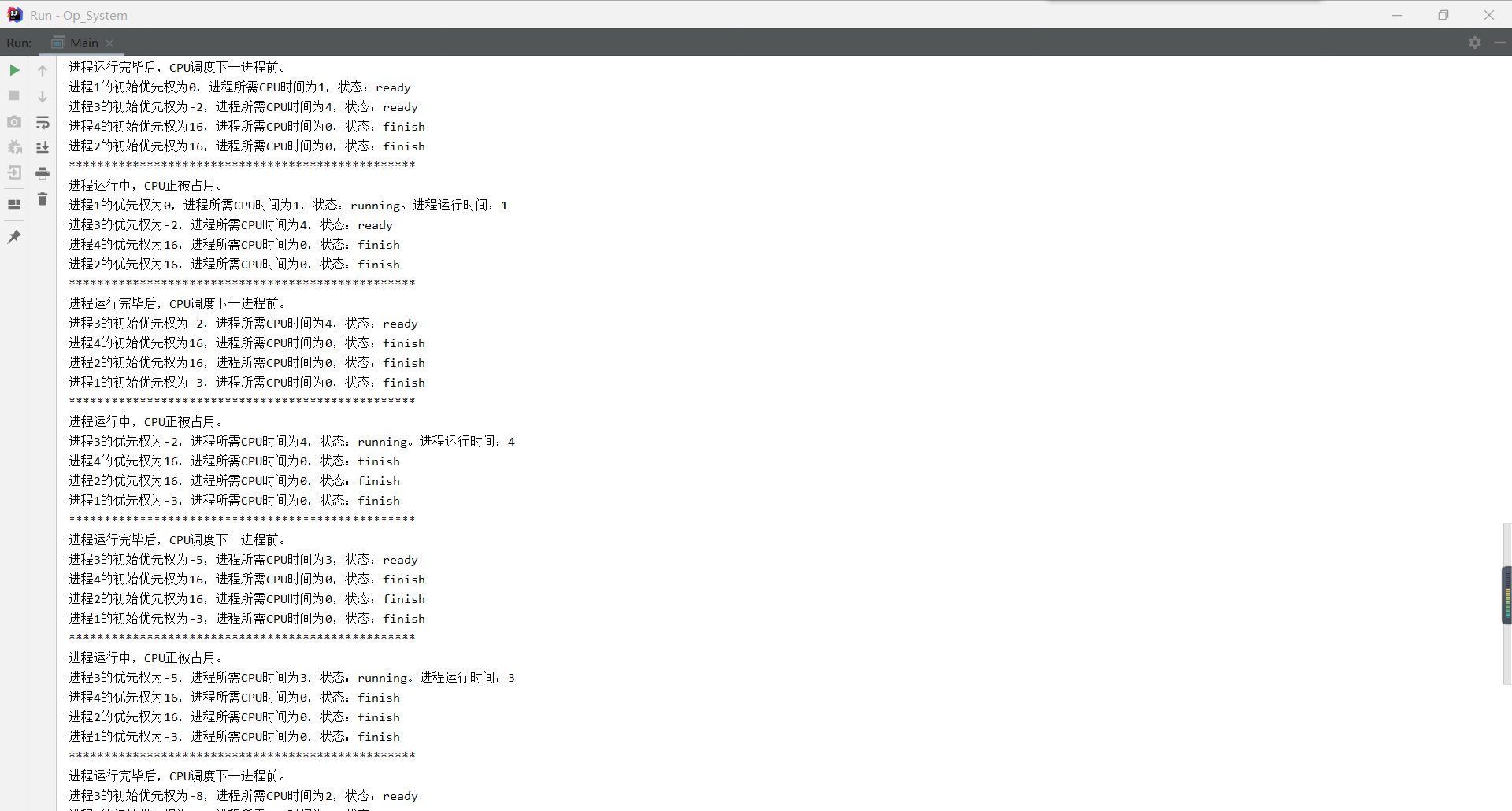
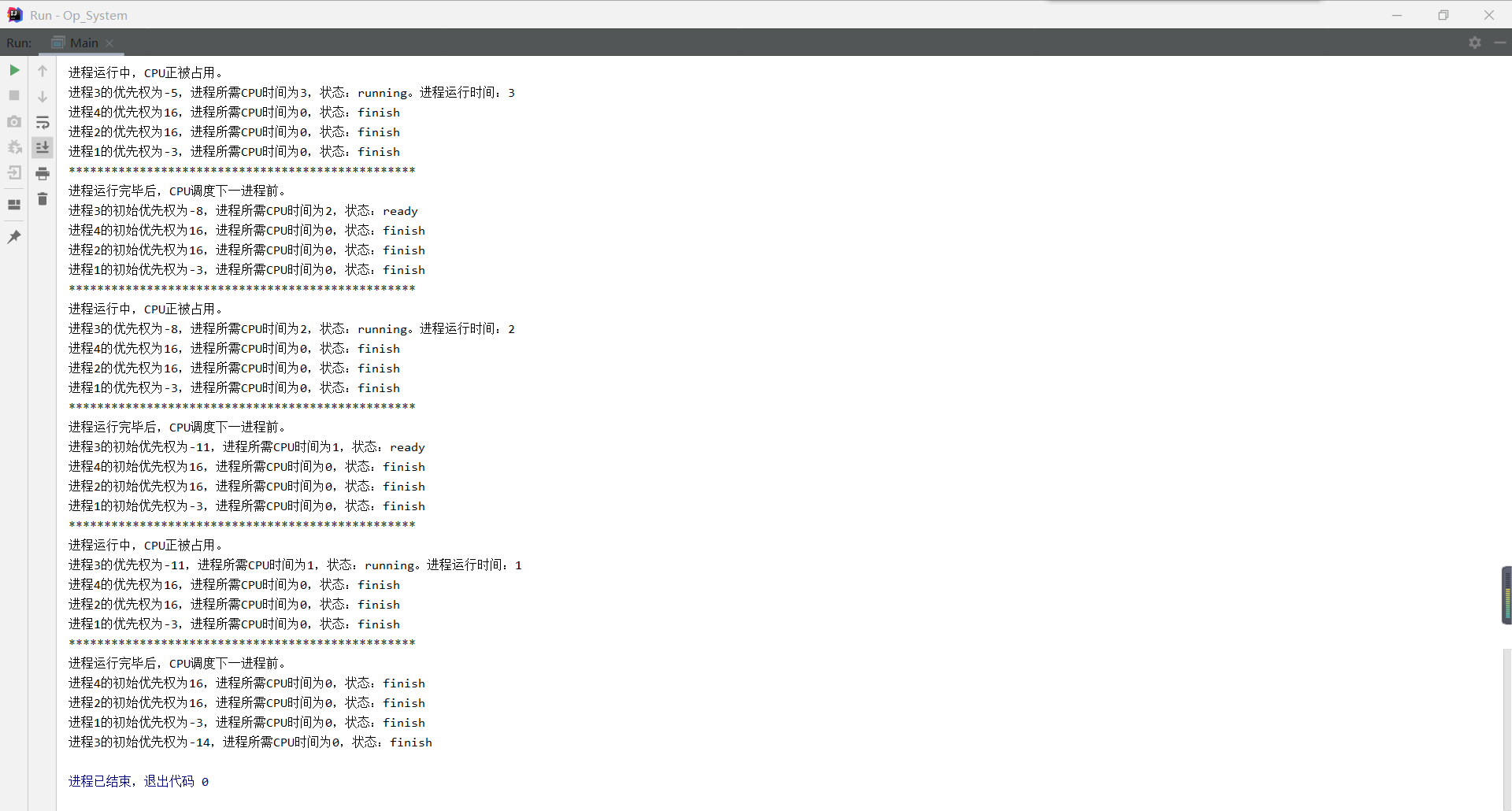
核心思想：初始化相关数据，启动调度方法。

附·源代码：

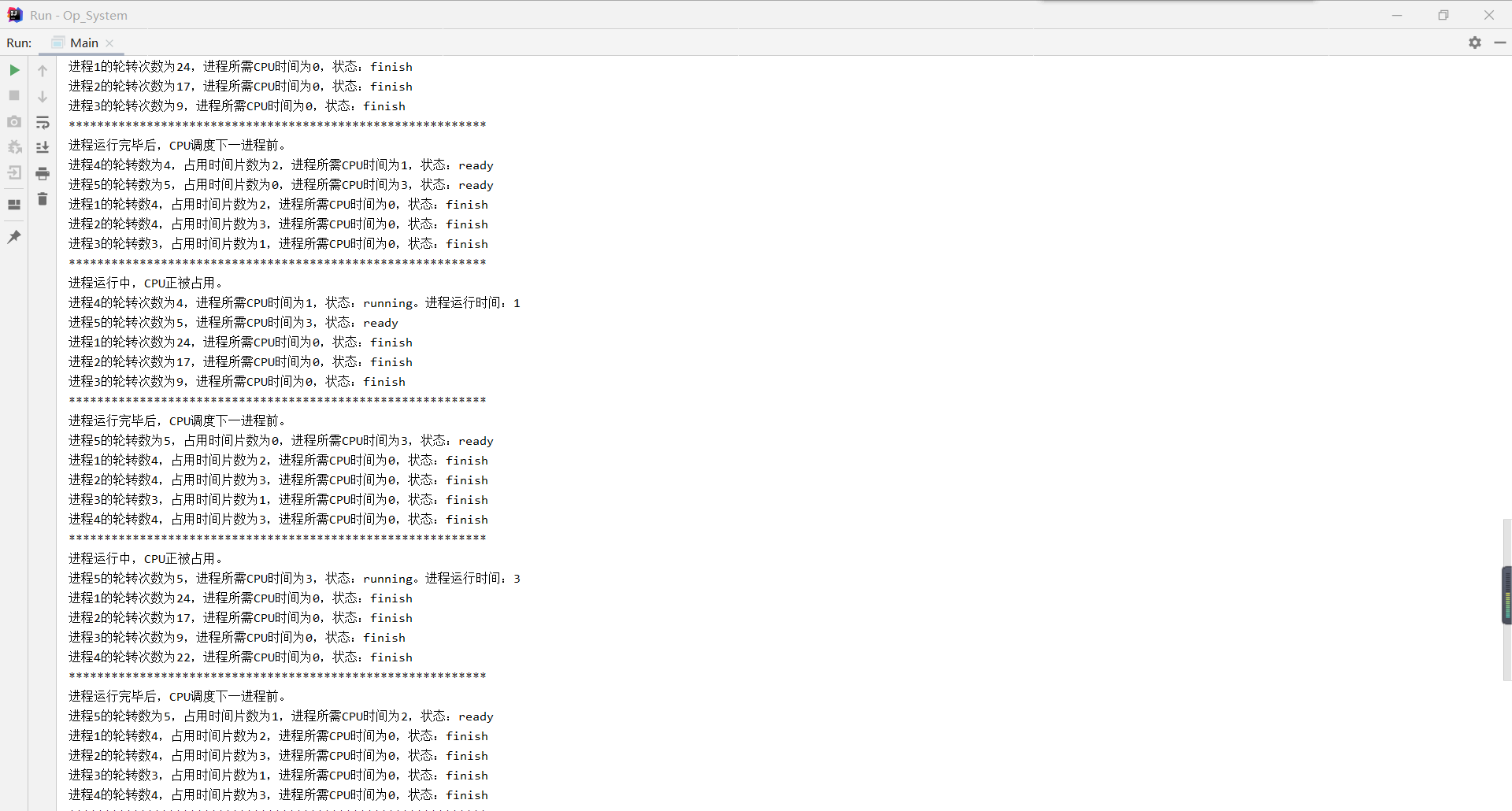
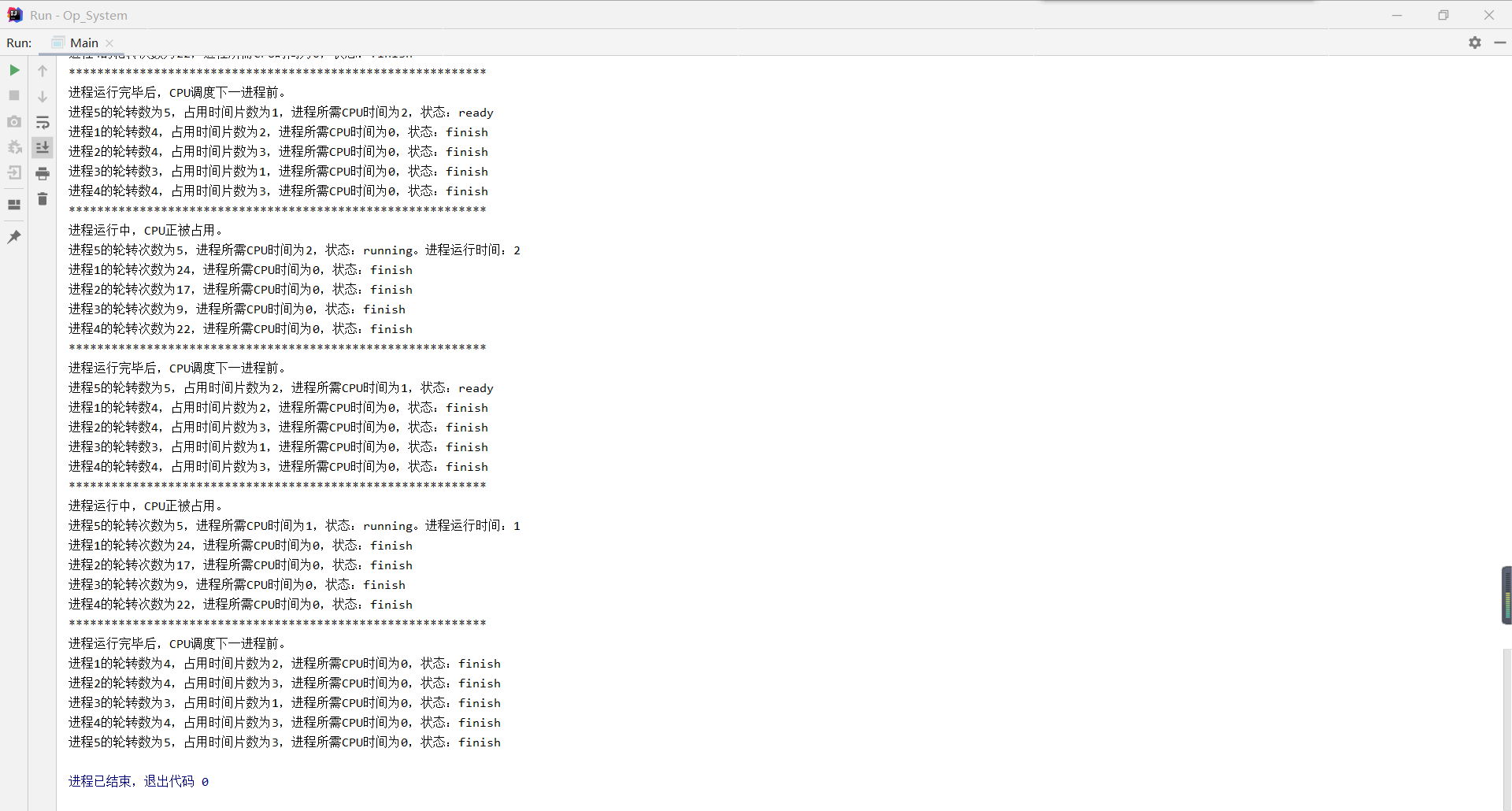
1. **public** **static** **void** main(String[] args){
2. System.out.print("请您输入数量在4~8的进程数：");
3. Scanner scanner=**new** Scanner(System.in);
4. **int** num= scanner.nextInt();
5. **while**(num<4||num>8){
6. System.out.print("您输入的数据范围不对，请输入数量在4~8的进程数：");
7. scanner=**new** Scanner(System.in);
8. num=scanner.nextInt();
9. }
10. **new** Priority(num);
11. **new** Rotation(num);
12. }

# 运行截图

## 1.动态优先权法



### 2.轮转法



# 实验心得

①本实验流程中的就绪队列并不是数据结构中的队列，而可以是数据结构中的单链表。对动态优先权调度法而言，在进程本轮调度完毕，时间片却未结束前，应该将其按照优先权的比对插入“就绪队列”，而不是机械地插入队尾，能在该数据结构的中间部分操作，应为不受限制的单链表；轮转调度法同。

②进程调度是一个值得研究的方面。对单核计算机而言，同一时刻只能运行一个进程，如果采用静态的、一成不变的顺序调度，若某正在运行的进程所需时间较长，易造成资源分配效率低下、系统卡顿等问题。因此我们设计操作系统时有必要根据进程的使用率、使用时长编写合适的动态调度法来调度进程，使最需要紧迫调度的进程优先调度，达到资源利用最大化、优化用户体验的目的。