《操作系统》课下作业 (OS-HW1)

中国人民大学 信息学院 崔冠宇 2018202147

P172, 3.2 Suppose that four interleaved processes are running in a system having start addresses 4050, 3200, 5000 and 6700. The traces of the individual processes are as follows:

Process P1	Process P2	Process P3	Process P4
4050	3200	5000	6700
4051	3201	5001	6701
4052	3202	5002	6702
4053	3203	5003	<I/O>
4054	3204	5004	
4055	3205	5005	
4056	3206	5006	
4057	<I/O>	5007	
4058		5008	
4059		5009	
4060		5010	

Find the interleaved traces of the processes. Assume that the dispatcher is invoked after 5 instructions or for interrupts and the dispatcher cycle has 4 instructions.

解. (题目大意: 设有四个进程交替运行在一个系统上, 它们的起始地址分别为 4050, 3200, 5000 和 6700. 单个进程的轨迹 (**trace**) 如上表所示. 写出这些进程的组合轨迹. 假定分派器 (**dispatcher**) 在5条指令或中断 (**interrupt**) 后被唤醒, 且分派器周期为4条指令.)

假定从 **Process P1** 开始运行, 分派器地址为 100 ∼ 103, 分派器从小到大轮流指派. 列表如下:

序号	地址	进程
1	4050	
2	4051	
3	4052	**P1**
4	4053	
5	4054	
——超时——		
6	100	
7	101	**分派器**
8	102	
9	103	
10	3200	
11	3201	
12	3202	**P2**
13	3203	
14	3204	
——超时——		

序号	地址	进程
15	100	
16	101	**分派器**
17	102	
18	103	
19	5000	
20	5001	
21	5002	**P3**
22	5003	
23	5004	
——超时——		
24	100	
25	101	**分派器**
26	102	
27	103	
28	6700	**P4**
29	6701	

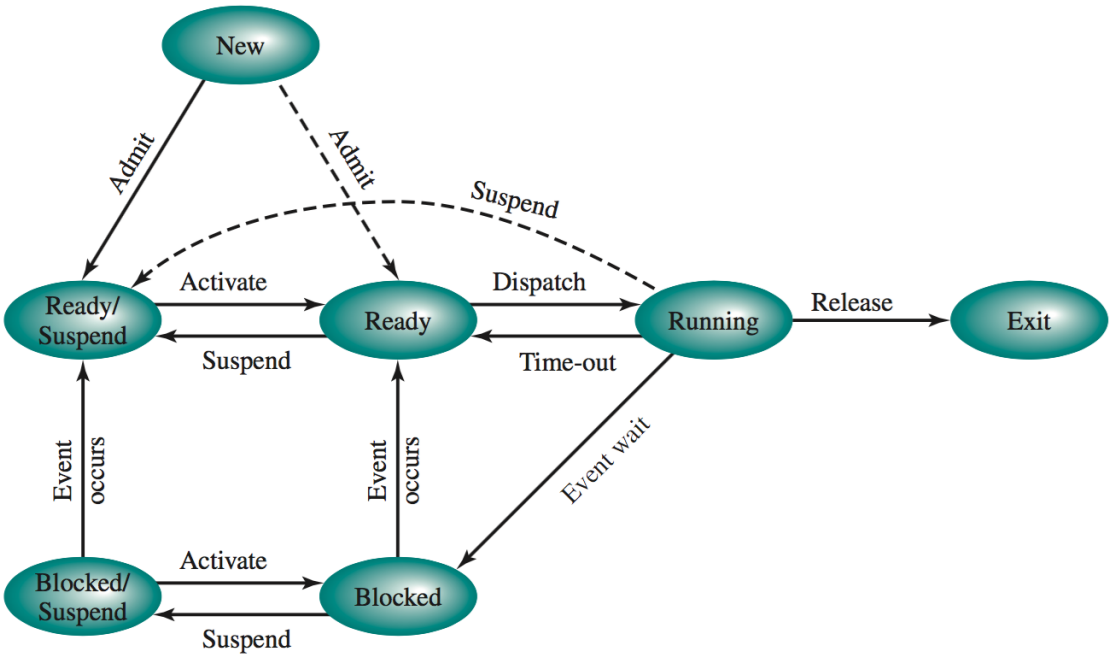
序号	地址	进程
30	6702	**P4**
——I/O 请求——		
31	100	
32	101	**分派器**
33	102	
34	103	
35	4055	
36	4056	
37	4057	**P1**
38	4058	
39	4059	
——超时——		
40	100	
41	101	**分派器**
42	102	
43	103	

序号	地址	进程
44	3205	**P2**
45	3206	
——I/O 请求——		
46	100	**分派器**
47	101	
48	102	
49	103	
50	5005	**P3**
51	5006	
52	5007	
53	5008	
54	5009	
——超时——		

序号	地址	进程
55	100	**分派器**
56	101	
57	102	
58	103	
59	4060	**P1**
——**P1** 结束——		
60	100	**分派器**
61	101	
62	102	
63	103	
64	3200	**P3**
——**P3** 结束——		
——结束——		

P173, 3.3 Figure 3.9b contains seven states. In principle, one could draw a transition between any two states, for a total of 42 different transitions.

- **a.** List all of the possible transitions and give an example of what could cause each transition.
- **b.** List all of the impossible transitions and explain why.



解. (题目大意: 图3.9b 含有七个状态. 原则上, 如果在任意两状态之间画出一个状态转换, 则总共有42种不同的转换. **a.** 列出所有可能的状态转换, 并给出一个能导致这种转换的实例. **b.** 列出所有不可能的状态转换并解释原因.)

a. 可能的状态转换列表如下:

转换	实例
New → Ready/Suspend	新建进程完毕, 但部分资源在外存中, 尚未加载入内存.
New → Ready	新建进程完毕, 且除CPU资源外均已准备完毕.
Ready/Suspend → Ready	处于就绪/挂起态但优先级较高的进程可能被载入内存.
Ready → Running	就绪的进程被分派器选中运行.
Running → Ready	达到允许进程不中断的最大时间、被系统抢占等.
Running → Blocked	需要等待某些事件, 如 I/O 等.
Blocked/Suspend → Blocked	可能是该挂起进程优先级较高, 而等待的事件很快将发生.
Ready → Ready/Suspend Running → Ready/Suspend Blocked → Blocked/Suspend	为腾出空间有可能被挂起.
Blocked → Ready Blocked/Suspend → Ready/Suspend	所等待的事件(如 I/O)已发生.
* → Exit(6种)	进程正常结束、超时、I/O失败、父进程请求等.

b. 不可能的状态转换列表如下:

转换	原因
* → New (6种)	进程新建状态, 不能回头.
New → Running, Blocked, Blocked/Suspend	未就绪不能直接运行, 也不会被阻塞和挂起.
Exit → * (5种, Exit → New 已包括)	进程退出状态, 不能继续.
Ready/Suspend → Running	未加载到内存中, 无法直接运行.
Ready/Suspend → Blocked, Blocked/Suspend	被挂起的进程未运行, 不会遇到阻塞事件.
Ready → Blocked	未运行, 不会遇到阻塞事件, 也不会被挂起.
Running → Blocked/Suspend	跳步, 阻塞和挂起应该分两步进行.
Blocked → Ready/Suspend	跳步, 事件发生和挂起不能同时进行.
Blocked → Running	跳步, 应首先变为就绪态才可能继续.
Blocked/Suspend → Ready, Running	跳步, 事件发生和激活才能进入就绪态.

P173, 3.5 Consider the state transition diagram of Figure 3.9b. Suppose it is time for the OS to dispatch a process and there are processes in both the Ready state and the Ready/Suspend state, and at least one process in the Ready/Suspend state has higher scheduling priority than any of the processes in the Ready state. Two extreme policies are as follows: (1) Always dispatch from a process in the Ready state, to minimize swapping, and (2) always give preference to the highest-priority process, even though that may mean swapping when swapping is not necessary. Suggest an intermediate policy that tries to

balance the concerns of priority and performance.

解. (题目大意: 考虑如图3.9(b)所示的状态转换图. 假设操作系统分派正在分配进程, 现在有些进程处于就绪态 (**Ready**) 和就绪/挂起态 (**Ready/Suspend**)) 的进程, 并且至少一个处于就绪/挂起态的进程拥有比其他所有处于就绪态的进程更高的优先级. 两种极端的策略如下: (1) 总是从就绪态的进程开始分派, 来减少交换 (**Swapping**), 以及 (2) 总是倾向于最高优先级的进程, 尽管这可能意味着不必要的交换. 试提出一种折中的方案, 尽量平衡优先级和性能考量.)

一种可行的方案: 适当降低就绪/挂起态进程的优先级, 使处于就绪/挂起态的部分进程的优先级低于处于就绪态的较高优先级的进程. 这样一来, 只有处于就绪/挂起态的进程的优先级相对较高时, 才会被分派器选中, 从而平衡了优先级与性能的考量.

P175, 3.12 You have executed the following C program:

```
1  main ()
2  { int pid;
3  pid = fork ();
4  printf ("%d \n", pid);
5  }
```

What are the possible outputs, assuming the fork succeeded?

解. (题目大意: 运行上面的 C 程序. 可能的输出结果是什么, 假定 fork 成功?)

程序将会输出 0 和一个正数. 在我的环境下是正数在前, 0 在后, 但不保证一定如此.