《操作系统》 进程调度

下列哪些选项会导致中断？

A.从磁记录硬盘完成数据读

B.执行一条指令DIV 100 % 0

C.执行一条指令LOAD R1 100

D.完成相关数据在显示器的打印

请选择下列与系统调用相关的正确选项

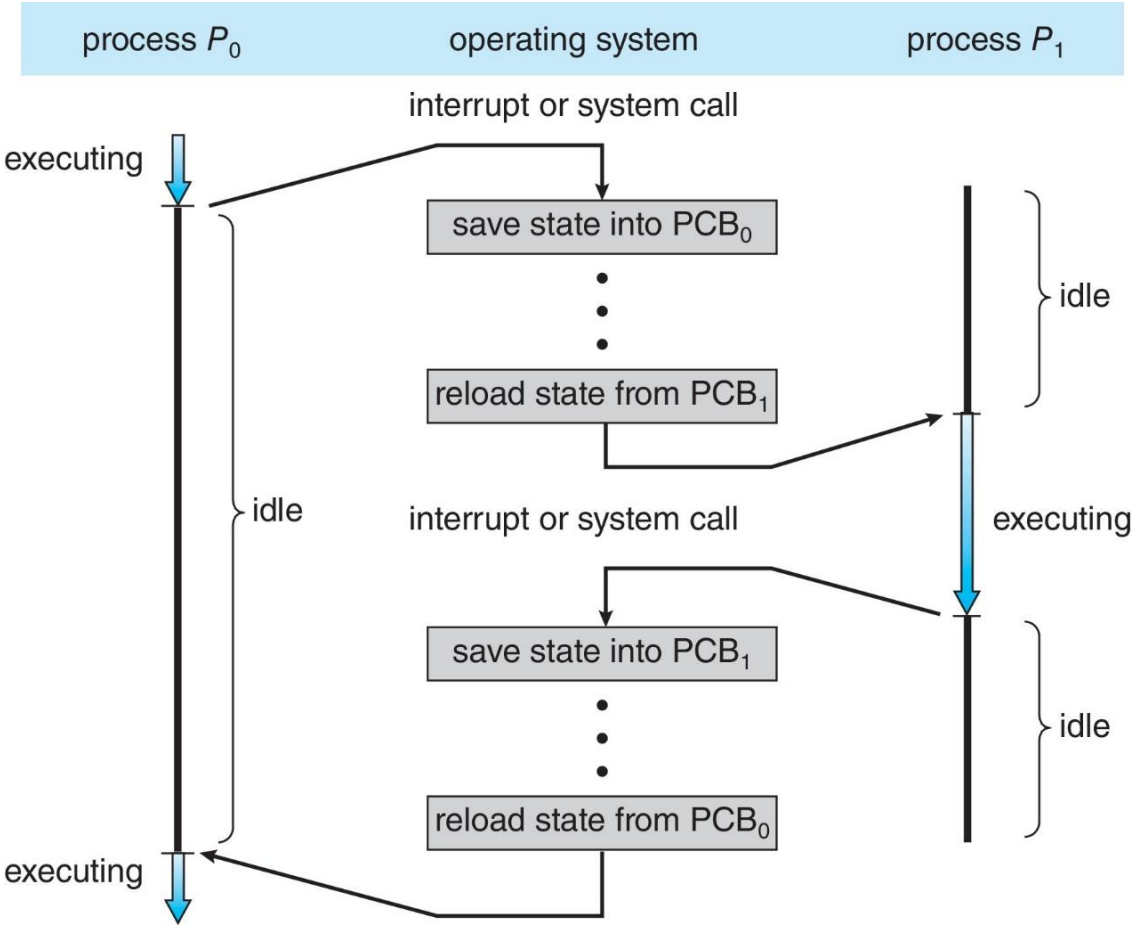
A.系统调用通常用C语言实现

B.系统调用通常用Java语言实现

C.系统调用有一个简单的功能

D.系统调用有一个复杂的功能

问题：以下哪些说法正确？



A. The saving state of PCB0 is the same as the reloading state of PCB0
PCB0的保存状态与PCB0的重新加载状态相同

B. The saving state of PCB1 is the same as the reloading state of PCB1
PCB1的保存状态与PCB1的重新加载状态相同

C. When P0 becomes idle, the corresponding process state should be changed
当P0变为空闲时，应改变相应的进程状态

D. The context switches have additional overhead for OS
上下文切换对操作系统有额外的开销

以下关于多线程的优点哪些是正确的?

A.能够减少响应时间

B.能够提高资源共享

C.能够减少创建开销

D.能够提高系统的可靠性

E.能够增强多核体系结构的可拓展性

F.能够减少上下文切换的开销

本节内容

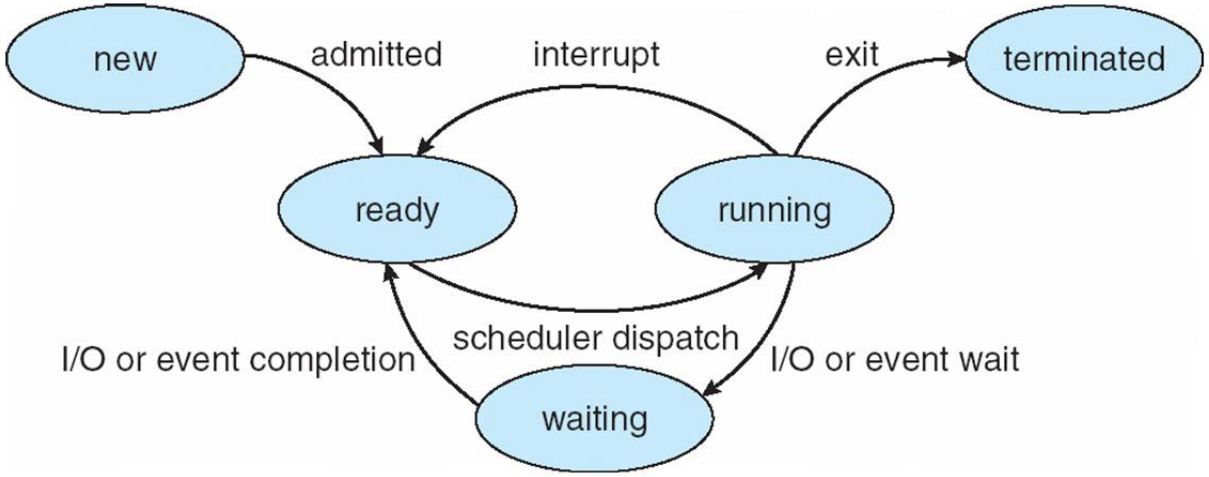
- 进程的状态
- CPU Scheduling：　处理器调度
  - 基本概念
  - 调度准则
  - 调度算法

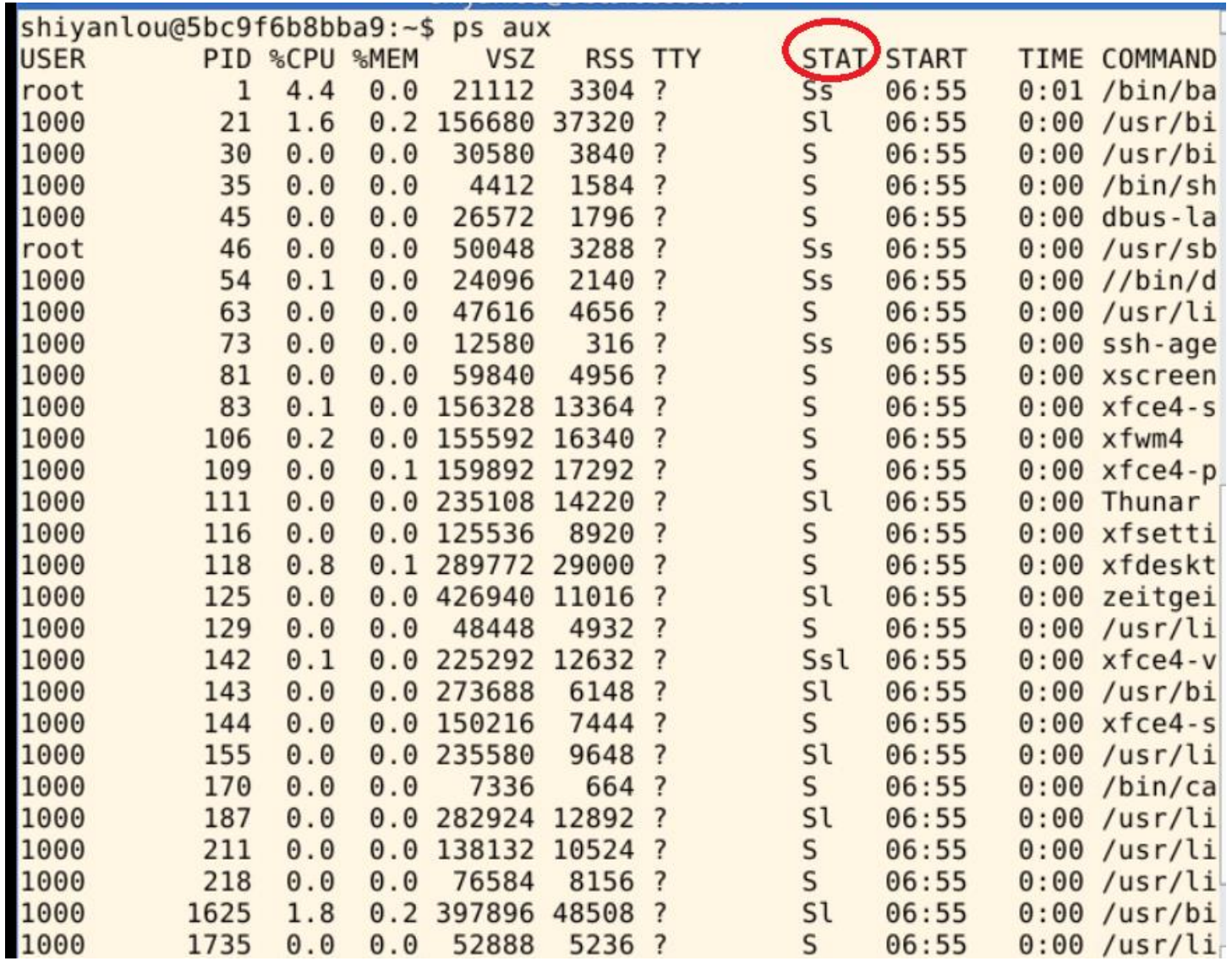
参考阅读：　Abraham Silberschatz等人, 操作系统概念　第五章

“进程”的状态

Process State 进程状态

- As a process executes, it changes *state*
  - **new**: The process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **ready**: The process is waiting to be assigned to a processor
  - **terminated**: The process has finished execution

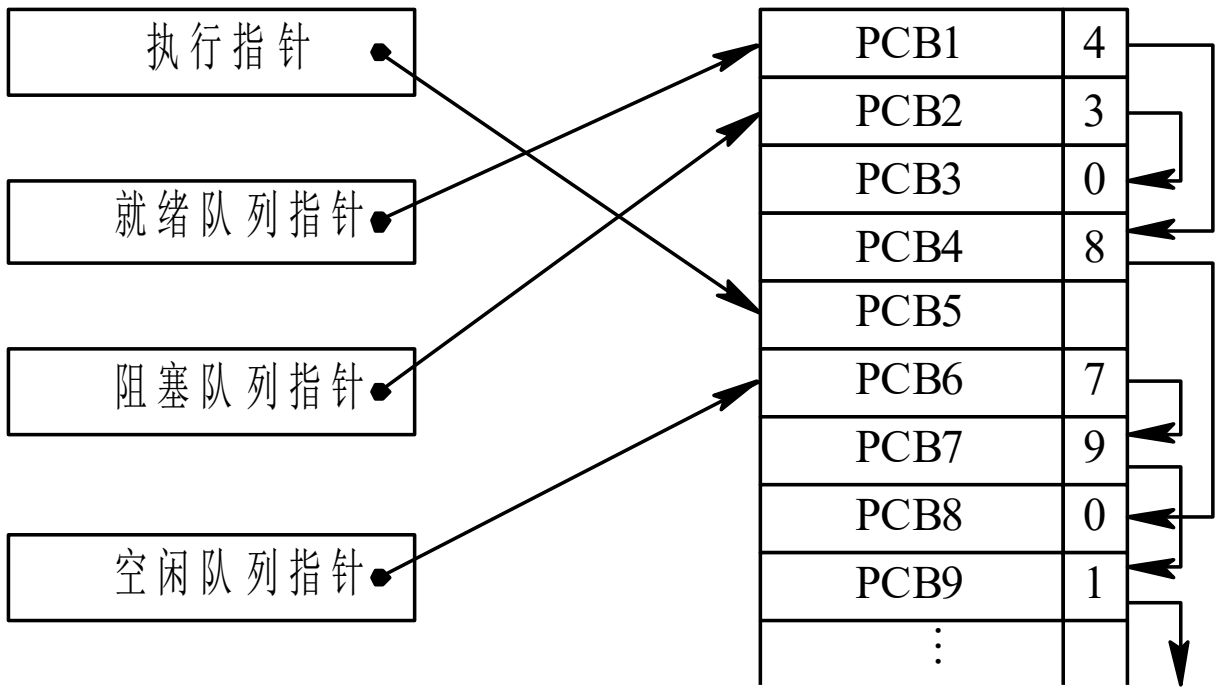




```
shiyanlou@5bc9f6b8bba9:~$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  4.4  0.0  21112  3304 ?        Ss   06:55   0:01 /bin/ba
1000        21  1.6  0.2 156680 37320 ?        Sl   06:55   0:00 /usr/bi
1000        30  0.0  0.0  30580  3840 ?        S    06:55   0:00 /usr/bi
1000        35  0.0  0.0   4412  1584 ?        S    06:55   0:00 /bin/sh
1000        45  0.0  0.0  26572  1796 ?        S    06:55   0:00 dbus-la
root        46  0.0  0.0  50048  3288 ?        Ss   06:55   0:00 /usr/sb
1000        54  0.1  0.0  24096  2140 ?        Ss   06:55   0:00 //bin/d
1000        63  0.0  0.0  47616  4656 ?        S    06:55   0:00 /usr/li
1000        73  0.0  0.0  12580   316 ?        Ss   06:55   0:00 ssh-age
1000        81  0.0  0.0  59840  4956 ?        S    06:55   0:00 xscreen
1000        83  0.1  0.0 156328 13364 ?        S    06:55   0:00 xfce4-s
1000       106  0.2  0.0 155592 16340 ?        S    06:55   0:00 xfwm4
1000       109  0.0  0.1 159892 17292 ?        S    06:55   0:00 xfce4-p
1000       111  0.0  0.0 235108 14220 ?        Sl   06:55   0:00 Thunar
1000       116  0.0  0.0 125536  8920 ?        S    06:55   0:00 xfsetti
1000       118  0.8  0.1 289772 29000 ?        S    06:55   0:00 xfdeskt
1000       125  0.0  0.0 426940 11016 ?        Sl   06:55   0:00 zeitgei
1000       129  0.0  0.0  48448  4932 ?        S    06:55   0:00 /usr/li
1000       142  0.1  0.0 225292 12632 ?        Ssl  06:55   0:00 xfce4-v
1000       143  0.0  0.0 273688  6148 ?        Sl   06:55   0:00 /usr/bi
1000       144  0.0  0.0 150216  7444 ?        S    06:55   0:00 xfce4-s
1000       155  0.0  0.0 235580  9648 ?        Sl   06:55   0:00 /usr/li
1000       170  0.0  0.0   7336   664 ?        S    06:55   0:00 /bin/ca
1000       187  0.0  0.0 282924 12892 ?        Sl   06:55   0:00 /usr/li
1000       211  0.0  0.0 138132 10524 ?        S    06:55   0:00 /usr/li
1000       218  0.0  0.0  76584  8156 ?        S    06:55   0:00 /usr/li
1000      1625  1.8  0.2 397896 48508 ?        Sl   06:55   0:00 /usr/bi
1000      1735  0.0  0.0  52888  5236 ?        S    06:55   0:00 /usr/li
```

- S 可中断睡眠 (受阻, 在等待某个条件的形成或接受到信号)
- s 某一个会话的leader进程
- l 线程加锁

OS对不同状态进程进行的管理



引起进程创建的原因

- 用户登录
- 作业调度
- 提供服务
- 应用请求

进程撤销的原因

- exit（0）：正常运行程序并退出程序;
- exit（1）：非正常运行导致退出程序;
- return（）：返回函数。在main函数中调用return和exit的现象就很模糊，多数情况下现象都是一致的。
- 强制撤销

进程阻塞

- **功能**
  - 停止进程的执行，变为阻塞。
- **阻塞的时机/事件**
  - 请求系统服务
    - （由于某种原因，OS不能立即满足进程的要求）
  - 启动某种操作
    - （进程启动某操作，阻塞等待该操作完成）
  - 新数据尚未到达
    - （A进程要获得B进程的中间结果，A进程等待）
  - 无新工作可作
    - （进程完成任务后，自我阻塞，等待新任务到达）

进程阻塞

参数

- 阻塞原因
- 不同原因构建有不同的阻塞队列。

进程阻塞的实现

- 停止运行
- 将PCB “运行态”改“阻塞态”
- 插入相应原因的阻塞队列
- 转调度程序

进程唤醒

功能

- 唤醒处于阻塞队列当中的某个进程。

引起唤醒的时机/事件

- 系统服务由不满足到满足
- I/O完成
- 新数据到达
- 进程提出新请求（服务）

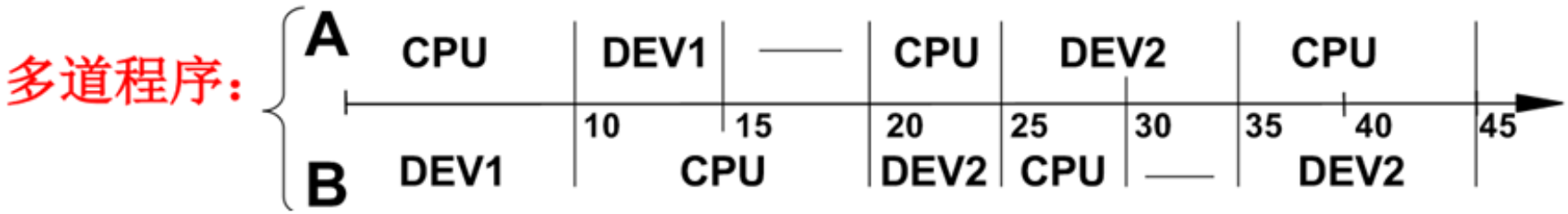
参数

- 被唤醒进程的标识

处理器调度：基本概念

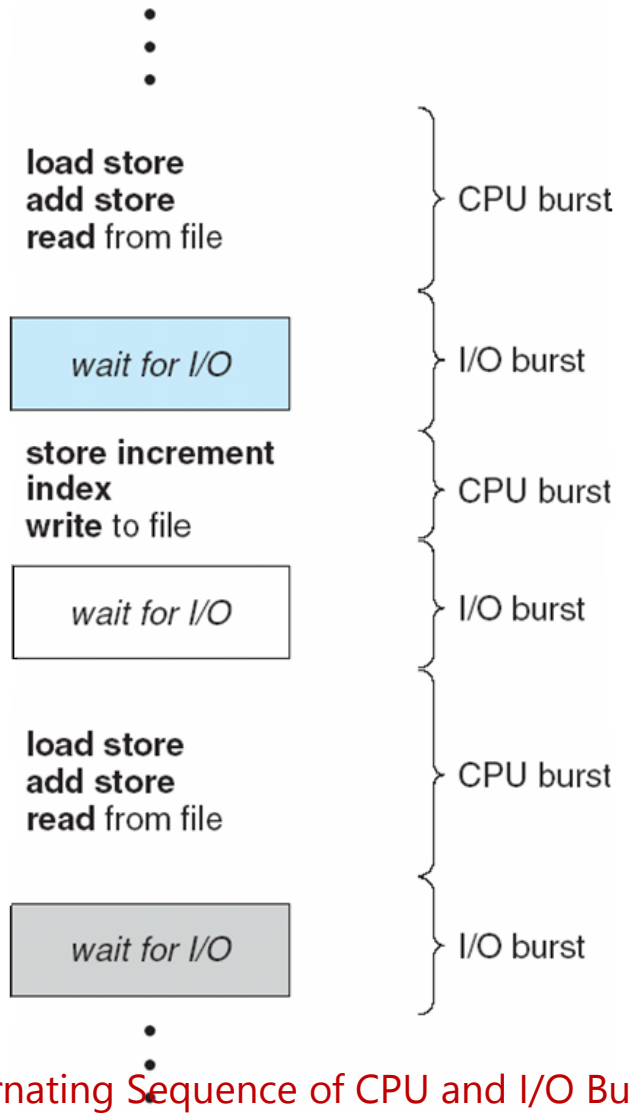
基本概念

- **CPU 调度的目的：**
  - 最大化CPU利用率
- 某个进程处于wait状态，一般是因为等待某些 I/O 请求的响应



基本概念

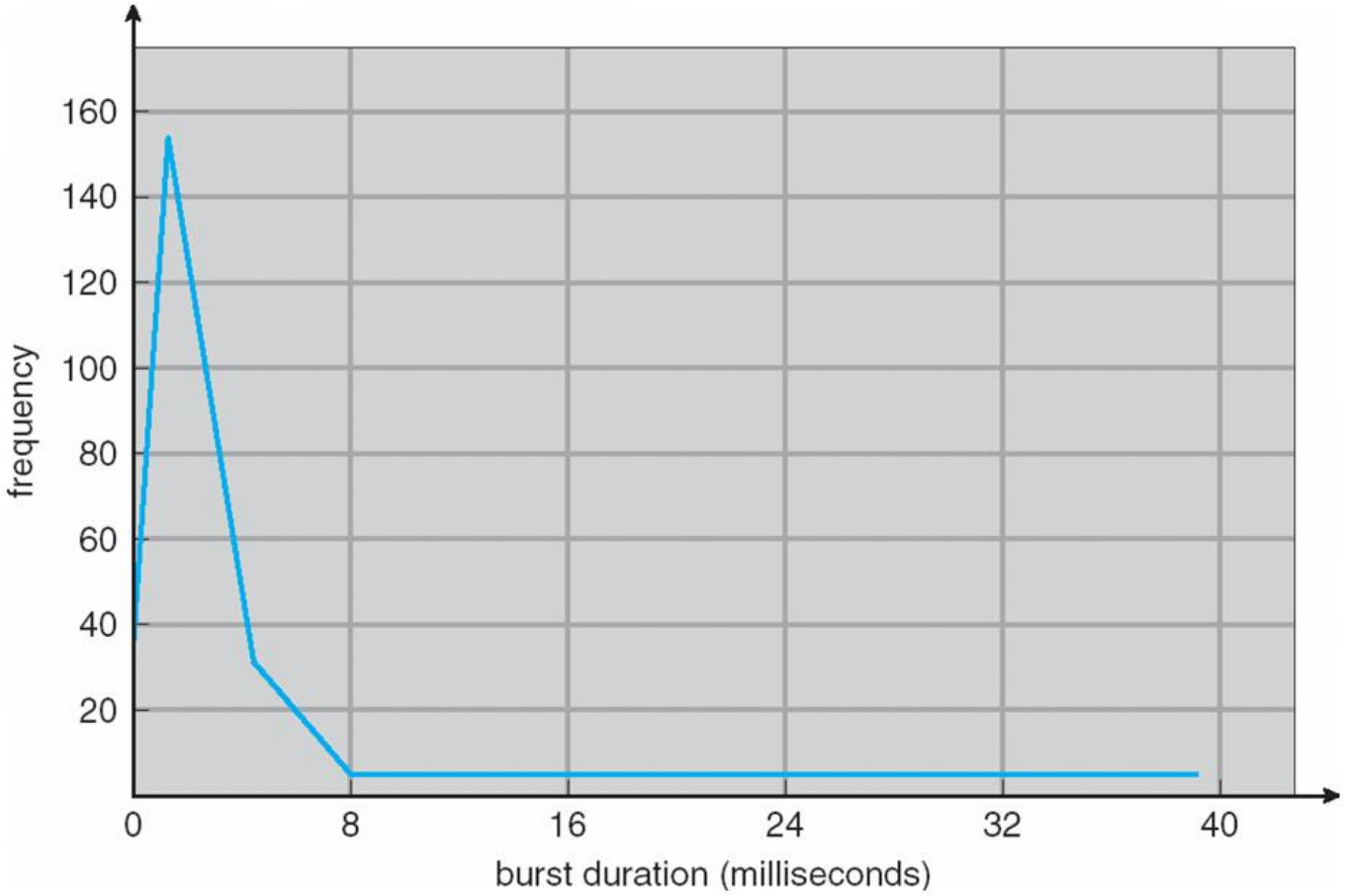
- CPU–I/O Burst Cycle
  - Process execution consists of a cycle of CPU execution and I/O wait
- **CPU burst （CPU区间）**
- **I/O burst (I/O 区间）**



Alternating Sequence of CPU and I/O Bursts

CPU-burst （区间）时间曲线图

- 大部分CPU区间都是短时间
- 仅少量CPU区间时间较长
- **An I/O-bound program**：typically has many short CPU bursts.
- **A CPU-bound program**：might have a few long CPU bursts.



CPU Scheduler

- 调度程序：在合适的时候、选择一个就绪进程运行

  调度的目标

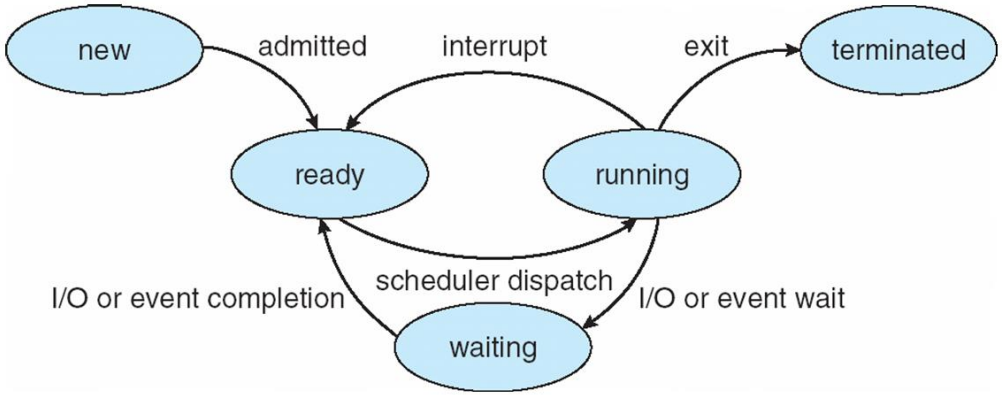
  调度的时机

  调度的策略

- 分派程序（dispatcher)： 把CPU控制权交给被调度的程序:
  - switching context 上下文切换
  - switching to user mode 从系统模式切换回用户模式
  - 跳转到用户程序合适的位置

调度的时机：什么时候进行调度？

1. Switches from running to waiting state
   例如： 发生外部中断请求，系统调用等
   被动调用
2. Switches from running to ready state
   例如：**Terminates（from running to exit)**
3. Switches from waiting to ready
   例如： I/O完成，中断返回时。 要切换为用户态



在CPU调度中，以下哪些被包含在调度延迟中？

A.加载进程的PCB信息

B.在用户态和内核态之间切换

C.保存进程的PCB信息

D.读取程序计数器（PC）寄存器信息

调度的目标/准则

- **响应速度快：响应时间短（交互性的评价指标）**
  - **response time**: amount of time it takes from when a request was submitted until the first response is produced。
- **处理时间短：周转时间 （批处理系统 的指标）**
  - **Turn around time**: the interval from the time of submission of a process to the time of completion is the turnaround time
- **等待时间：**
  - **Waiting time ：** amount of time a process has been waiting in the ready queue
- **吞吐率：**
  - **Throughput ：** # of processes that complete their execution per time unit
- **利用率：**
  - **CPU utilization ：** keep the CPU as busy as possible
- **公平：每个进程获得合理的处理器份额**

处理器调度：调度算法

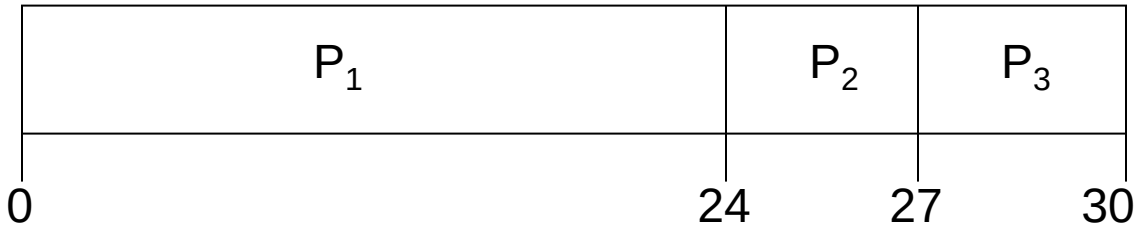
CPU Scheduling Algorithms

- First-Come, First-Served (FCFS) Scheduling
- Shortest-Job-First (SJF) Scheduling
- Priority Scheduling (PS)
- Round-Robin Scheduling (RR)
- Multilevel Queue Scheduling
- Multilevel Feedback Queue Scheduling

First-Come, First-Served (FCFS) Scheduling

Process	CPU Burst	*Arrival* Time
P_1	24	0
P_2	3	1
P_3	3	2

- The **Gantt Chart** for the schedule is:



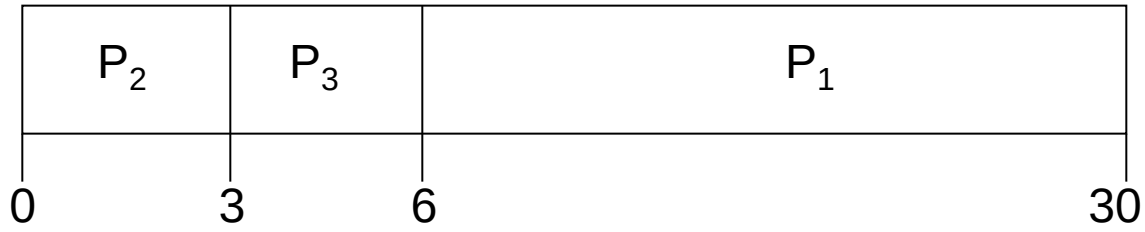
- Waiting time for P_1 = 0; P_2 = 24-1=23; P_3 = 27-2=25
  Average waiting time: (0 + 23 + 25)/3 = 16

- Turnaround time for P_1 = 24; P_2 = 27-1=26; P_3 = 30-2=28
  Average turnaround time: (24 + 26 + 28)/3 = 26

FCFS Scheduling (Cont.)

Process	CPU Burst	*Arrival* Time
P_1	24	2
P_2	3	0
P_3	3	1

- The Gantt chart for the schedule is:

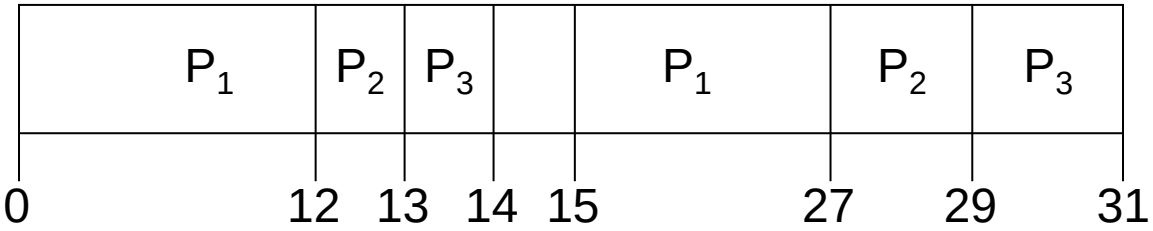


- Waiting time for P1 = 4; P2 = 0; P3 = 2
- Average waiting time: (4 + 0 + 2)/3 = 2
- Much better than previous case

FCFS Scheduling (Cont.)

Process	CPU Burst	I/O Burst	CPU Burst	*Arrival* Time
P_1	12	3	12	0
P_2	1	2	2	1
P_3	1	2	2	2

- The **Gantt Chart** for the schedule is:



- Waiting time for
  - P_1 = 15-12-3=0
  - P_2 = (12-1)+(27-13-2)=23
  - P_3 = (13-2)+(29-14-2)=24
- Turnaround time for P_1 = 27; P_2 = 29-1=28; P_3 = 31-2=29
- CPU utilization 30/31 = 96.77%

Shortest-Job-First (SJF) Scheduling

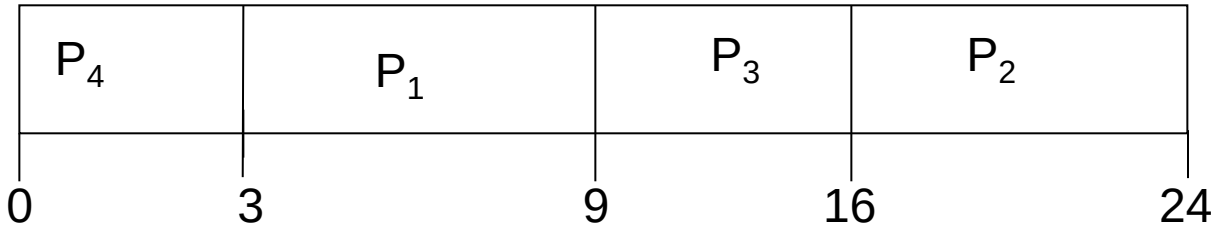
- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time

- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user

Example of SJF （短作业优先）

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



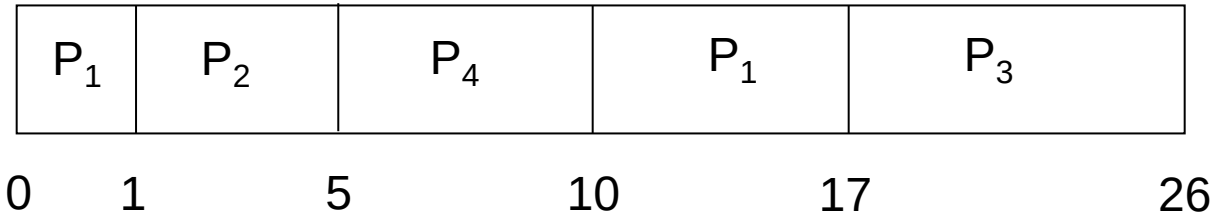
- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

短作业优先(抢占式）举例

- We now add the concepts of varying arrival times and preemption to the analysis

Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- *Preemptive* SJF Gantt Chart



- Average waiting time = [(10-1)+(1-1)+(17-2)+(5-3)]/4 = 26/4 = 6.5

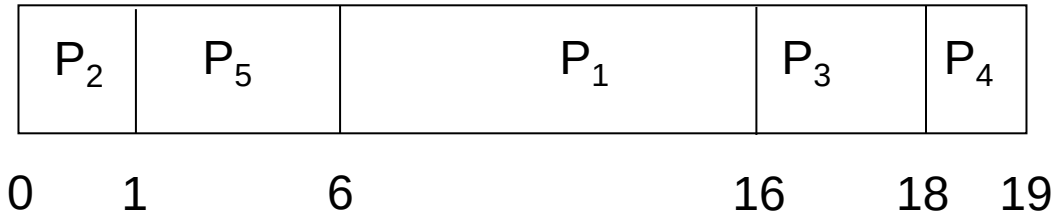
优先级调度： Priority Scheduling

- 优先数： A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority
- （some systems: smallest integer ≡ highest priority)
  - Preemptive
  - Nonpreemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem: **Starvation** – low priority processes may never execute
- Solution: **Aging** – as time progresses increase the priority of the process

优先级调度举例

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Priority scheduling Gantt Chart



- Average waiting time = 8.2 msec

课堂练习

- 某系统正在执行三个进程P1、P2和P3，各进程的计算（CPU）时间和I/O时间比例如下所示：

进程	计算时间	I/O时间
P1	90%	10%
P2	50%	50%
P3	15%	85%

为提高系统资源利用率，合理的进程优先级设置应为：

A.P3 > P2 > P1

B.P2 > P1 > P3

C.P1 > P2 > P3

D.P3 > P1 > P2

问题：如何考虑静态优先级？

- 考虑静态优先级的时候，
- 下列因素
  - “基于进程所需的资源多少”，
  - “基于程序运行时间的长短”
  - “基于进程的类型[IO/CPU,前台/后台,核心/用户]”
- 如何影响静态优先级确定的？
- 譬如：进程所需的资源多（或少），就分配较低（或较高）的静态优先级！
- 你的理由何在？

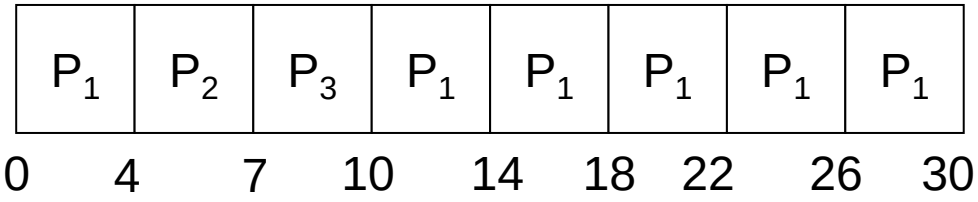
轮转式调度 Round Robin (RR)

- Round Robin (RR) is similar to FCFS scheduling, but preemption is added to switch between processes.
- Each process gets a small unit of CPU time (**time quantum** q), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are *n* processes in the ready queue and the time quantum is *q*, then each process gets 1/*n* of the CPU time in chunks of at most *q* time units at once. No process waits more than (*n*-1)*q* time units before next execution.
- Timer interrupts every quantum to schedule next process
- Performance
  - *q* large $\Rightarrow$ FIFO
  - *q* small $\Rightarrow$ *q* must be large with respect to context switch, otherwise overhead is too high

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

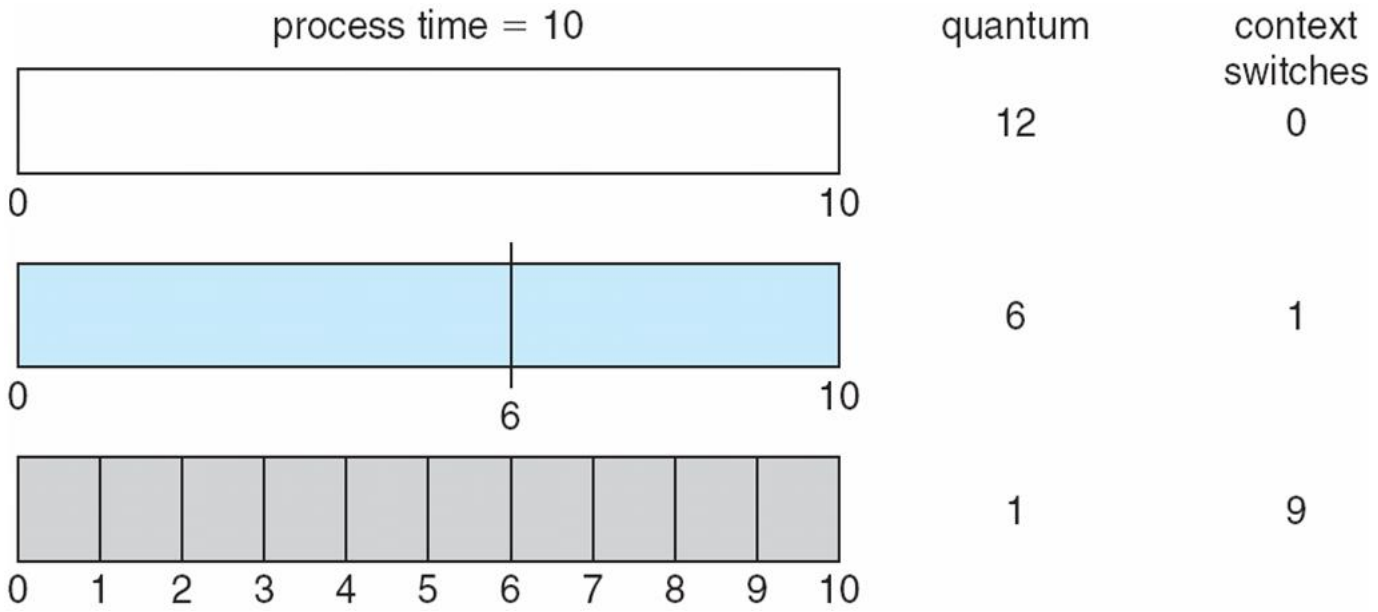
- The Gantt chart is:



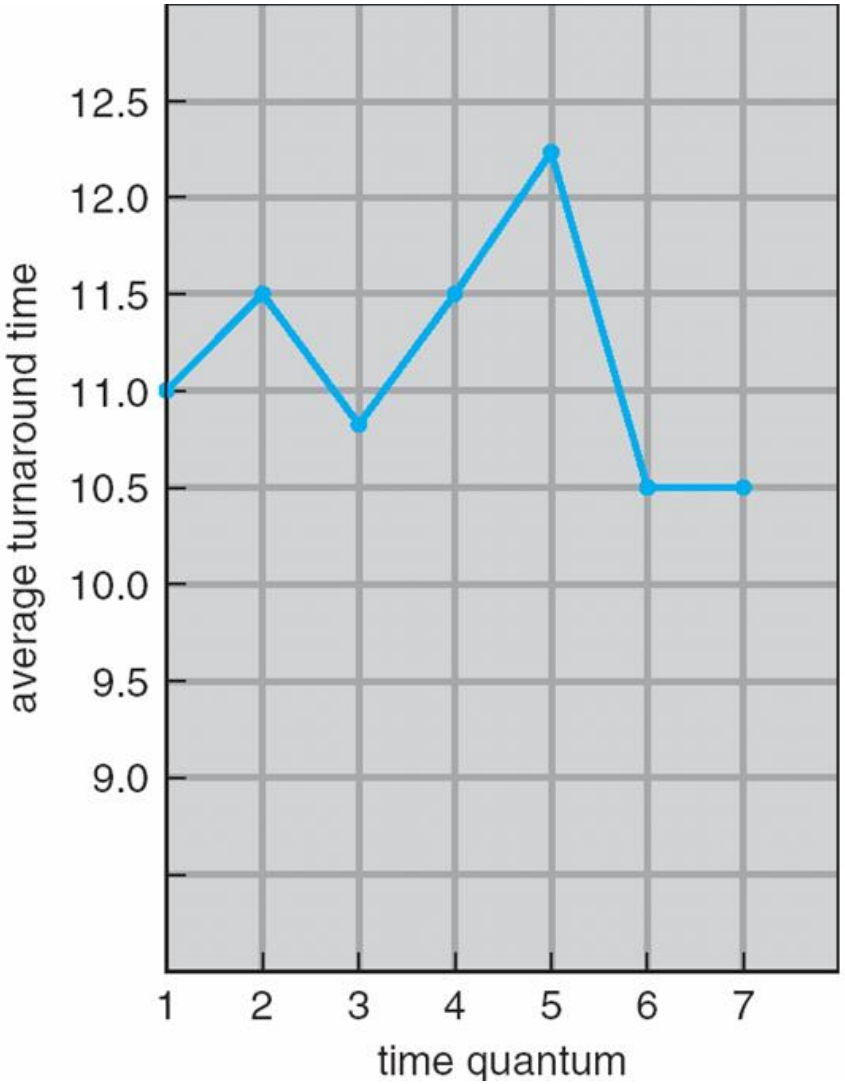
- Typically, higher average turnaround than SJF, but better *response*
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec
- What's the number of context switch? 上下文切换了几次?

时间片与上下文切换时间的关系

- Context switching: the process switch not caused by a voluntary yielding of CPU from the running process



Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

A rule of thumb :

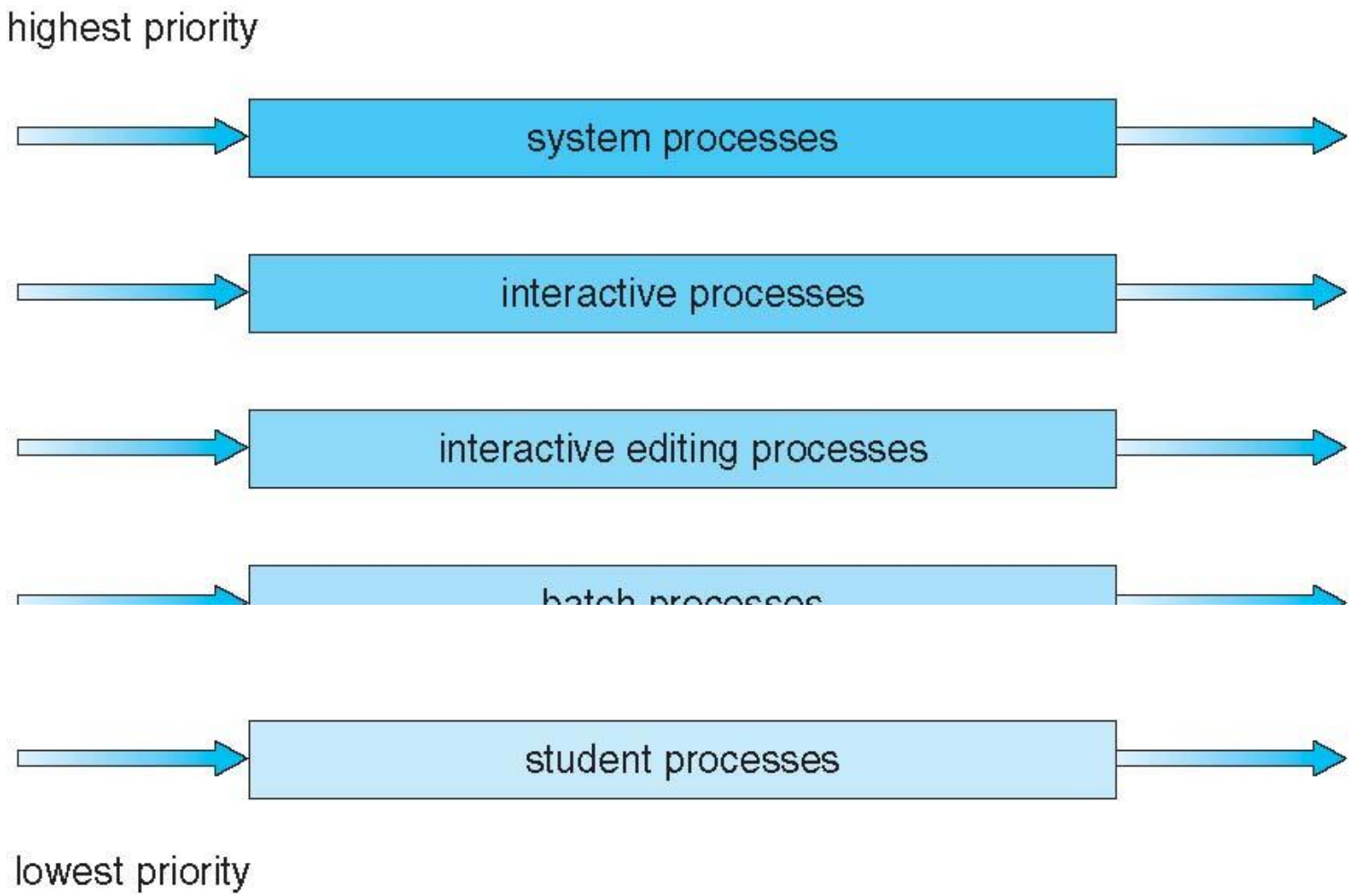
80% of CPU bursts should be shorter than q

课堂练习

- 下列调度算法中，不可能导致饥饿现象的是:

A. 时间片轮转
B. 抢占式短作业优先
C. 静态优先级调度
D. 非抢占式短作业优先

Multilevel Queue Scheduling

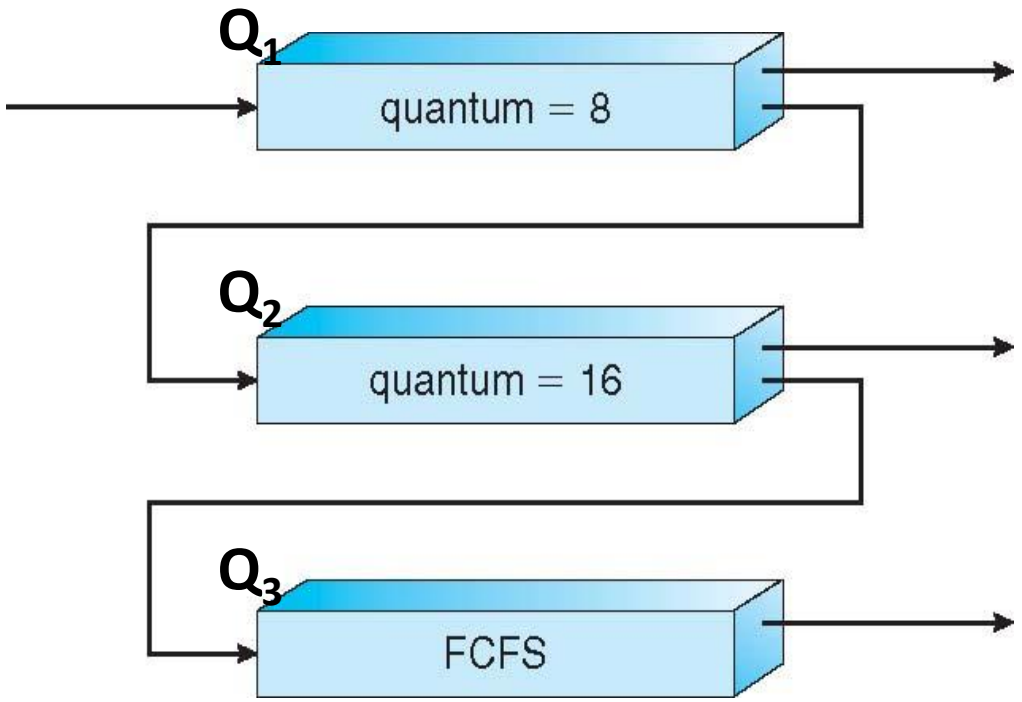


多级反馈队列 Multilevel Feedback Queue

- 进程可在不同的队列之间挪动：
- 需要确定以下参数
  - 队列数目
  - 每个队列的调度算法
  - 升降级的条件
  - 进程在需要服务时，进入哪个队列

Example of Multilevel Feedback Queue

- Three queues:
  - Q_1 – RR with time quantum 8 milliseconds
  - Q_2 – RR with time quantum 16 milliseconds
  - Q_3 – FCFS



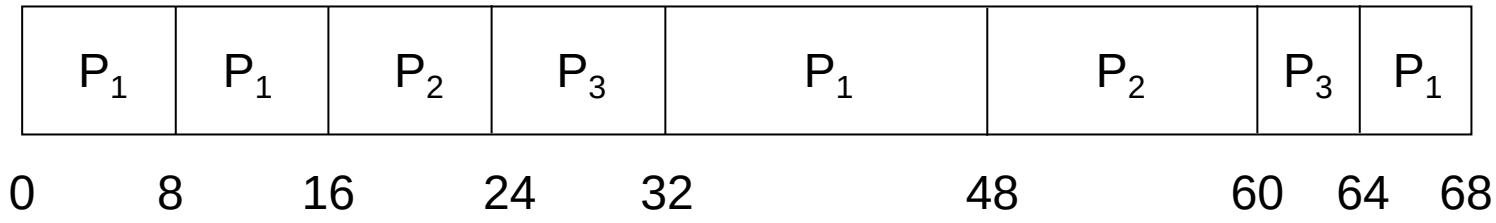
Multilevel Feedback Queue

- Scheduling
  - A new job enters queue Q_1 which is served FCFS/RR
    - When it gains CPU, job receives 8 milliseconds
    - If it does not finish in 8 milliseconds, job is moved to queue Q_2
  - At Q_2 job is again served FCFS/RR and receives 16 additional milliseconds
    - If it still does not complete, it is preempted and moved to queue Q_3
  - If a process does not use up its quantum in the current level, it will keep its current queuing level and be put into the end of the queue. Then, it can still get the same amount of quantum (not remaining quantum) next time when it is picked.

Example of Using Multilevel Feedback Queue

Process	*Arrival* Time	Burst Time
P_1	0	36
P_2	16	20
P_3	20	12

- The Gantt chart is:



课堂练习

- 下列选项中，降低进程优先级的合理时机是:

A.进程的时间片用完
B.进程从就绪态转为运行态
C.进程长期处于就绪队列中
D.进程刚完成I/O，进入就绪队列

讨论

- 考虑动态优先级的时候
- 下列因素
  - “当使用CPU超过一定时长时”,
  - “当进行I/O操作后”
  - “当进程等待超过一定时长时”
- 如何影响动态优先数确定的?
- 譬如：当使用CPU超过一定时长时，就减少（或增加）的动态优先级？！
- 你的理由何在?

下一节

- Linux 调度机制
- 进程之间的通信与同步