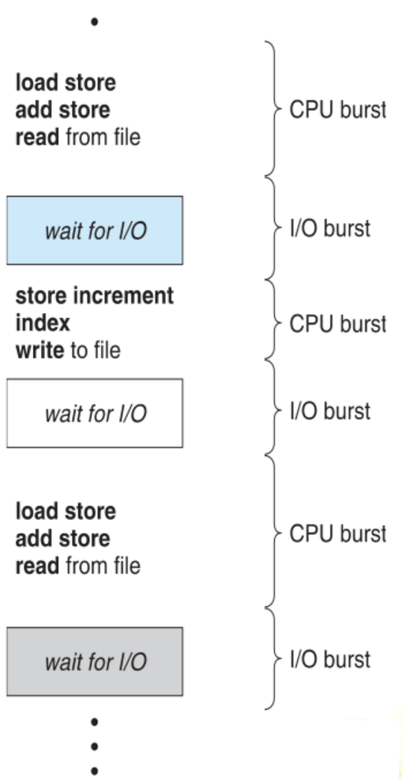
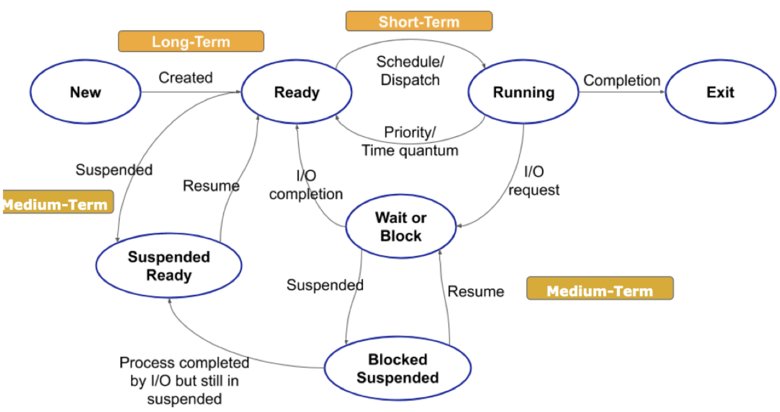
5 CPU Scheduling 1

- Basic Concepts
  - CPU–I/O Burst Cycle
    - Process execution consists of a cycle of CPU execution and I/O wait.
    - • Process execution begins with a CPU burst, followed by an I/O burst, then another CPU burst ... etc
    - • The duration of these CPU burst have been measured.
    - • An I/O-bound program would typically have many short CPUbursts, A CPU-bound program might have a few very long CPUbursts.
    - • this can help to select an appropriate CPU-scheduling algorithm.
    - 进程执行包括一个CPU执行周期和I/O等待周期。
    - 进程执行从CPU突发开始，接着是I/O突发，然后是另一个CPU突发…等
    - 这些CPU爆发的持续时间已经被测量。
    - 一个I/ o绑定的程序通常会有很多短的CPUbursts，一个cpu绑定的程序可能会有一些非常长的CPUbursts
      -

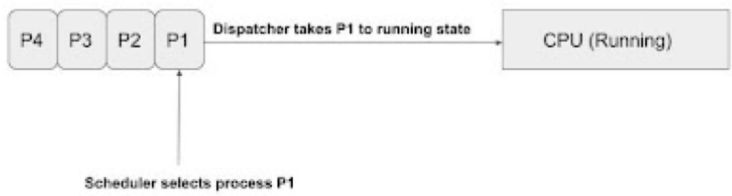


  - Types of Processes
    - I/O bound
      - • Has small bursts of CPU activity and then waits for I/O (eg. Word processor)
        - 有小的CPU活动爆发，然后等待I/O(例如。字处理器)

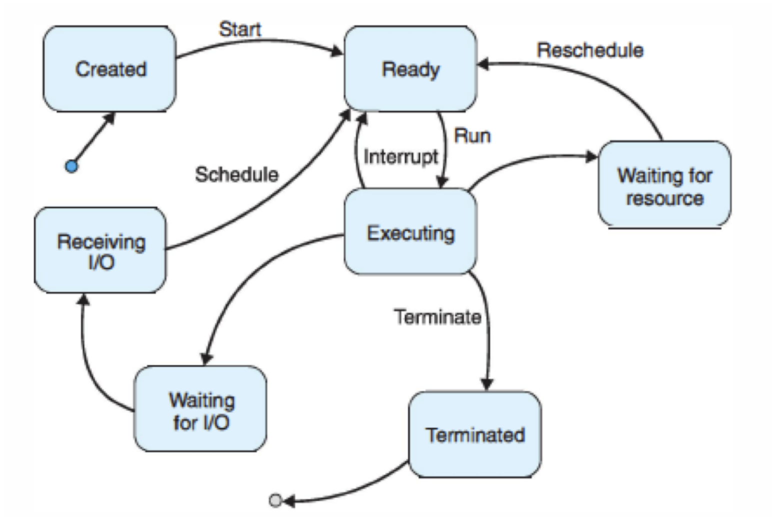
- • Affects user interaction (we want these processes to have highest priority)
    - •影响用户交互(我们希望这些进程具有最高优先级)
- CPU bound
  - Hardly any I/O, mostly CPU activity (eg. gcc, scientific modeling, 3Drendering, etc.)
    - 几乎没有I/O，主要是CPU活动(例如。gcc，科学建模，3d渲染等)
  - Useful to have long CPU bursts
    - 对于长时间的CPU爆发很有用
  - Could do with lower priorities
    - 可以降低优先级

- The CPU scheduler is the mechanism to select which process has to be executed next and allocates the CPU to that process.Schedulers are responsible for transferring a process from one state to the other.
  - CPU调度器是一种机制，用于选择下一步必须执行哪个进程，并将CPU分配给该进程。调度器负责将进程从一种状态转移到另一种状态。
- Basically, we have three types of schedulers i.e.
  - ❑ Long-Term Scheduler❑ Short-Term Scheduler❑ Medium-Term Scheduler
  - 

- Scheduler & Dispatcher
  - Schedulers are special system software that handles processscheduling in various ways.
  - Dispatcher, module of the operating system, removes process from the ready queue and sends it to the CPU to complete.
  - 调度器是以各种方式处理进程调度的特殊系统软件。
  - Dispatcher是操作系统的模块，它将进程从就绪队列中移除，并将其发送给CPU完成。
  -

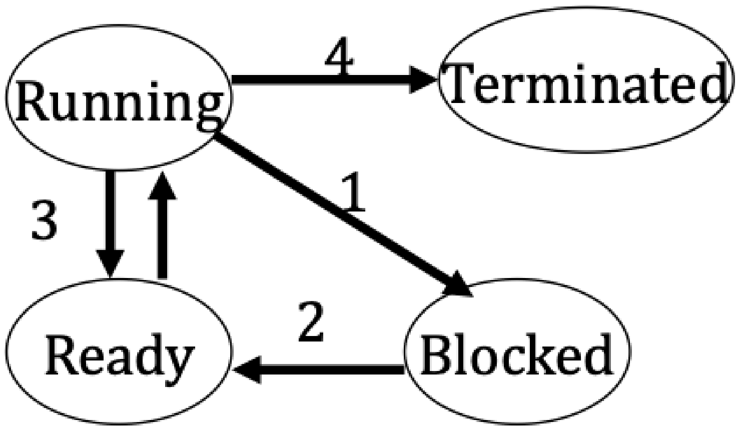


- Scheduling Policies
  - PREEMPTIVE SCHEDULING = the system may stop the execution of the running process and after that, the context switch may provide the processor to another process.
    - 抢占式调度=系统可能会停止正在运行的进程的执行，然后上下文切换可能会将处理器提供给另一个进程。
    - The interrupted process is put back into the ready queue and will be scheduled sometime in future, according to the scheduling policy.
      - 根据调度策略，被中断的进程被放回就绪队列，并将在将来的某个时候被调度。
  - NON-PREEMPTIVE SCHEDULING = when a process is assigned to the processor, it is allowed to execute to its completion, that is, a system cannot take away the processor from the process until it exits.
    - NON-PREEMPTIVE SCHEDULING =当一个进程被分配给处理器时，它被允许执行到完成，也就是说，系统不能从这个进程中拿走处理器，直到它退出。
    - Any other process which enters the queue has to wait until the current process finishes its CPU cycle.
      - 任何进入队列的其他进程都必须等待当前进程完成其CPU周期。
-



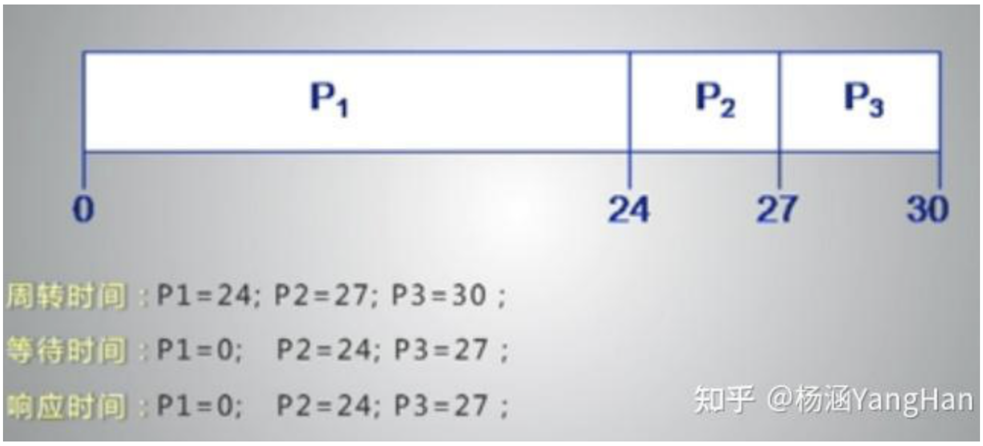
- Non-preemptive scheduling
  - the CPU is allocated to the process until it terminates.
  - the running process keeps the CPU until it voluntarily gives up the CPU

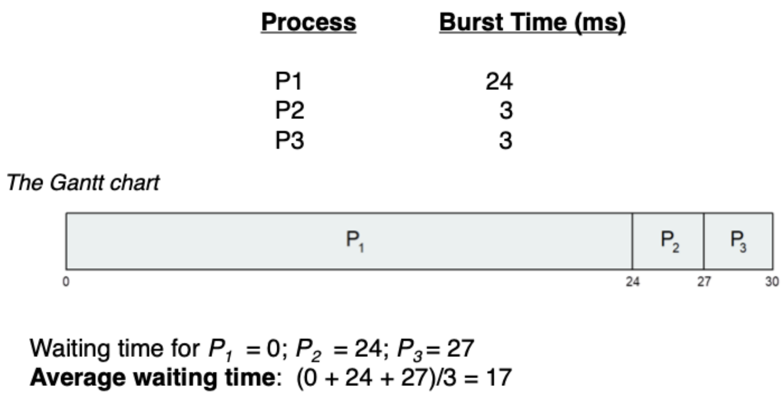
- CPU被分配给进程，直到进程终止。
- 正在运行的进程会一直占用CPU，直到它主动放弃CPU
- ☐ process exits Running
- ☐ switches to blocked state 3
- ☐ 1 and 4 only (no 3)
-



- Preemptive scheduling
  - the CPU is allocated to the processes for a specific time period
  - the running process can be interrupted and must release the CPU (can be forced to give up CPU)
  - CPU在特定的时间段内分配给进程
  - 正在运行的进程可以中断，必须释放CPU(可以强制放弃CPU)
- Dispatcher
  - Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
    - – switching context
    - – switching to user mode
    - – jumping to the proper location in the user program to restart that program
  - • Dispatch latency – time it takes for the dispatcher to stop one process and start another running.
  - Dispatcher is invoked during every process switch; hence it should be as fast as possible
  - Dispatcher模块将CPU的控制权交给由短期调度器选择的进程;这包括:
    - -切换上下文
    - —切换到用户模式
    - -跳转到用户程序中的适当位置以重新启动该程序
    - •调度延迟——调度程序停止一个进程并启动另一个进程所需的时间。
  - Dispatcher在每次进程切换期间被调用;因此，它应该尽可能快

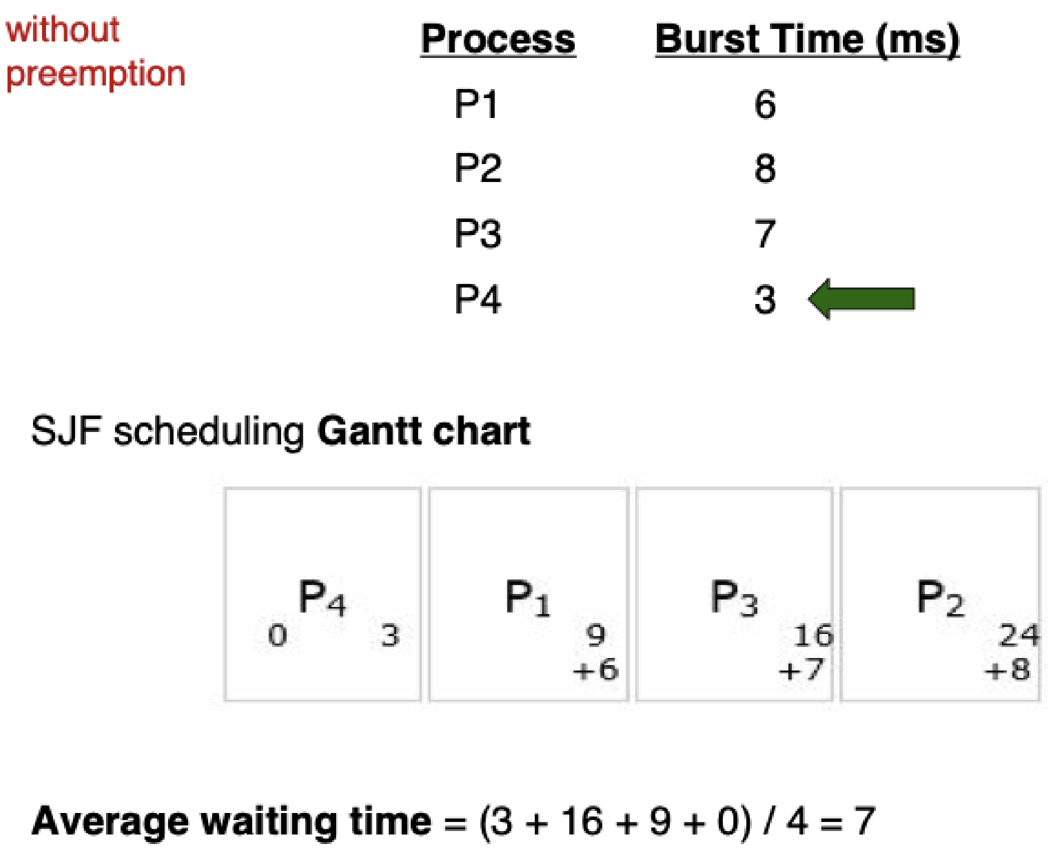
- Scheduling Criteria
  - Max CPU utilization – keep the CPU as busy as possibleMax Throughput – complete as many processes as
  - possible per unit timeFairness - give each process a fair share of CPU
  - Min Waiting time – process should not wait long in theready queue
  - Min Response time – CPU should respond immediately
  - 最大CPU利用率——让CPU尽可能忙碌。最大吞吐量——完成尽可能多的进程单位时间可能
  - 公平——给每个进程公平的CPU份额
  - 最小等待时间——进程不应该在就绪队列中等待很长时间
  - 最小响应时间- CPU应立即响应
- Scheduling Algorithms
  - 概念
    - Arrival Time (AT): Time at which the process arrives in the ready queue.
      - 到达时间(AT):进程到达就绪队列的时间。
    - Completion Time: Time at which process completes its execution.
      - 完成时间:流程完成执行的时间。
    - Burst Time: Time required by a process for CPU execution.
      - 突发时间:进程执行CPU所需的时间。
    - Turnaround Time (TT): the total amount of time spent by the process from coming in the ready state for the first time to its completion.
      - 周转时间(TT):流程从第一次进入就绪状态到完成所花费的总时间。
      - Turnaround time = Exit time - Arrival time.
        - 周转时间=退出时间-到达时间。
        - 没有到达时间就是运行完的时间
    - Waiting Time (WT): The total time spent by the process/thread in the readystate waiting for CPU.
      - 等待时间(WT):进程/线程在就绪状态等待CPU的总时间。
      - Waiting Time = Turn Around Time – Burst Time
        - 等待时间=周转时间-爆发时间
        - 没有到达时间的前提下
        - 等待时间=退出时间-爆发时间=等待的时间
        -



- Response time: Time at which the process gets the CPU for the first time.
    - 响应时间:进程第一次获得CPU的时间。
- First-Come, First-Served (FCFS) Scheduling
    - 

    - Processes are executed on first come, first served basis.
        - 流程按照先到先得的原则执行。
    - Poor in performance as average wait time is high.
        - 性能差，平均等待时间长。
- Shortest-Job-First (SJF) Scheduling
    - schedule process with the shortest burst time
        - 用最短的突发时间调度进程
    - the shortest burst time is scheduled first.
        - 首先安排最短的爆发时间。
    - Advantages– Minimizes average wait time and average response time•
        - 优点-最大限度地减少平均等待时间和平均响应时间
    - Disadvantages– Not practical : difficult to predict burst time
        - 缺点-不实用:难以预测爆发时间
    - May starve long jobs
        - 可能会让长期工作挨饿

- The real difficulty with the Shortest-Job-First SJF algorithm isknowing the length of the next CPU request.
    - 最短作业优先的SJF算法的真正困难在于知道下一个CPU请求的长度。
- - there is no way to know the exact length of process’s nextCPU burst.
    - 没有办法知道进程下一次cpu爆发的确切长度。
        - Computing an approximation of the length of the next CPU burst
            - 计算下一次CPU突发长度的近似值
- SJF cannot be implemented at the level of the short-term CPU scheduling
    - SJF不能在短期CPU调度级别上实现
        - 在长期作业调度中，可以通过历史数据对作业的执行时间进行预测，并且有更多的时间来计算和监测作业的执行时间。同时，在长期作业调度中，饥饿现象的影响也会降低
-



- Determining Length of Next CPU Burst
    -

Let t_n = actual length of the n^{th} burst;

τ_{n+1} = predicted value for the next CPU burst;

a = weighing factor, **$0 \le a \le 1$**.

The estimate of the next CPU burst period is:

$$\tau_{n+1} = a t_n + (1-a)\tau_n$$

Commonly, **a set to ½** -- determines the relative weight of recent and past history (τ_n)

- If **a=0**, then **recent history has no effect**

- If **a=1**, then only the **most recent CPU bursts matter**

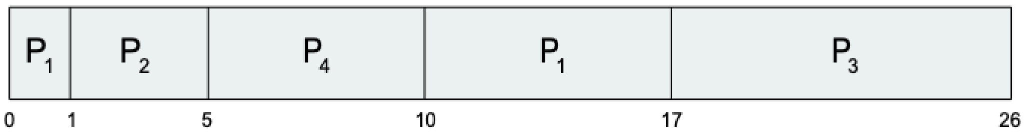
- $$\tau_{n+1} = \alpha\, t_n + (1-\alpha)at_{n-1} + (1-\alpha)(1-a)at_{n-2} + \ldots$$

- Shortest Remaining Time First (SRTF) Scheduling
    - If a new process arrives with a shorter burst time thanremaining of current process, then schedule new process
        - 如果新进程到达时的爆发时间比当前进程剩余时间短，则调度新进程
    - Further reduces average waiting time and average responsetime
        - 进一步减少平均等待时间和平均响应时间
    - Context Switch - the context of the process is saved in theProcess Control Block PCB when the process is removed from theexecution and the next process is scheduled.
        - 上下文切换——进程的上下文被保存在进程控制块PCB中，当进程被从执行中移除并计划下一个进程时。
    - This PCB is accessed on the next execution of this process.
        - 在此流程的下一次执行时访问此PCB
    -

Example

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Gantt chart



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

- Priority Scheduling
  - Each process is assigned a priority (an integer number).
    - 每个进程被分配一个优先级(一个整数)。
  - The CPU is allocated to the process with the highest priority (smallest integer = highest priority)
    - CPU分配给优先级最高的进程(最小整数=最高优先级)
  - Priorities may be:
    - Internal priorities based on criteria within OS. Ex: memory needs.
      - 基于操作系统标准的内部优先级。例如:内存需求。
    - External priorities based on criteria outside OS. Ex: assigned byadministrators.
      - 基于操作系统外部标准的外部优先级。例如:由管理员分配的。
  - !!! PROBLEM □ Starvation – low priority processes may never execute
    - ! !问题□饥饿-低优先级进程可能永远不会执行
  - SOLUTION □ Aging – as time progresses increase the priority of the
    - process Example: do priority = priority + 1 every 15 minutes
      - 解决方案□老化-随着时间的推移，增加的优先级
    - 大题画gantt chart

      SJF scheduling Gantt chart

  -

Each process has assigned a priority (integer number).

Process with highest priority is to be executed first and so on..

Process	**Burst Time**	**Priority**
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

P_2	P_5	P_1	P_3	P_4
0 1	6 +5	16 +10	18 +2	19 +1

Average waiting time = = (0 + 1 + 6 + 16 + 18) / 5 = 8.2

- Round Robin(RR) Scheduling
  - Each process gets a small unit of CPU time (time quantum ortime-slice), usually 10-100 milliseconds.
  - After this time has elapsed, the process is preempted and added to the end of the ready queue Ready queue is treated as a circular 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.
  - 每个进程获得一个小的CPU时间单位(时间量子或时间片)，通常是10-100毫秒。
  - 在此时间过后，该进程将被抢占并添加到就绪队列的末尾。就绪队列被视为循环队列
  - 如果就绪队列中有n个进程，时间量为q，那么每个进程一次最多获得1/n的CPU时间块(最多q个时间单位)。没有进程等待超过(n-1)q个时间单位。
  - q large - RR scheduling = FCFS scheduling
  - q small - q must be large with respect to context switch, otherwise overhead is too high对于上下文切换，Q必须很大，否则开销太高
  -

Round Robin (RR) Scheduling (Ex.1.)

Process	Burst Time
P1	24
P2	3
P3	3

The average wait time would be:

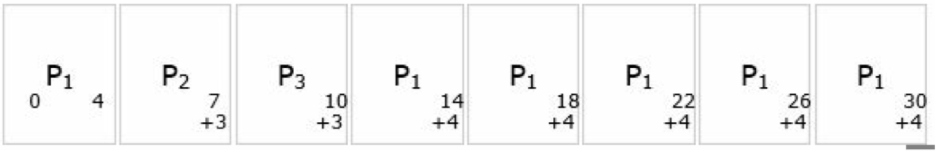
$P_{1_w} = 30 - 24 = 6$

$P_{2_w} = 7 - 3 = 4$

$P_{3_w} = 10 - 3 = 7$

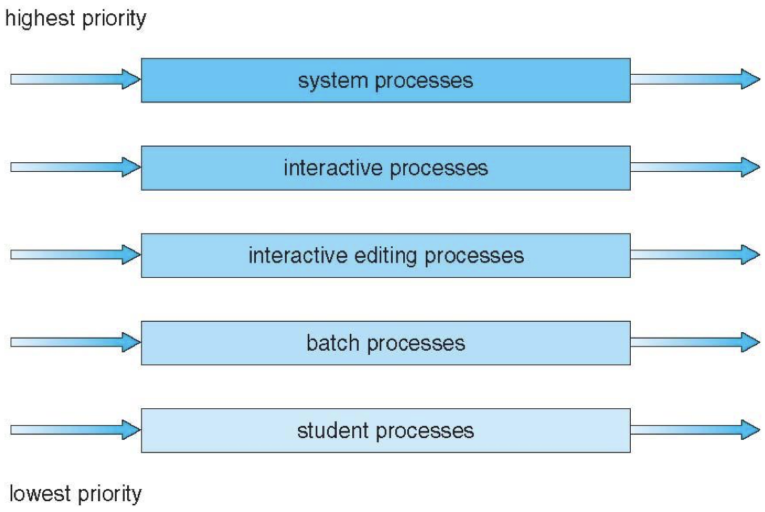
AWT = $(6 + 4 + 7) / 3 = 5.66$ms

Time quantum = 4ms



- Multiple-Level Queues Scheduling
  - Ready queue is partitioned into separate queues;
    - 将就绪队列划分为单独的队列;
    - e.g., two queues containing
      - 例如，两个队列包含
      - o foreground (interactive) processes . May have externally defined priority over background processes
        - 前台(交互)进程。可能对后台进程有外部定义的优先级
      - o background (batch) processes
        - 后台(批处理)进程
  - Process permanently associated to a given queue; no move to a different queue
    - 与给定队列永久关联的进程;没有移动到另一个队列
  - There are two types of scheduling in multi-level queue scheduling:
    - 在多级队列调度中有两种类型的调度:
    - • Scheduling among the queues.
      - •队列之间的调度。
    - • Scheduling between the processes of the selected queue.
      - •在所选队列的进程之间进行调度。
  - Must schedule among the queues too (not just processes):
    - 必须在队列之间进行调度(而不仅仅是进程):
    - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
      - 固定优先级调度;(即，先从前台服务，再从后台服务)。可能会饿死。

- Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes;
  - 时间片-每个队列获得一定数量的CPU时间，它可以在其进程之间调度;
  - ➢ 80% to foreground in RR, and 20% to background in FCFS
    - 在RR中，80%为前景，在FCFS中，20%为背景
- The various categories of processes can be:
  - Interactive processes
    - 互动的过程
  - • Non-interactive processes
    - •非交互式流程
  - • CPU-bound processes
    - •cpu绑定进程
  - • I/O-bound processes
    - •I/ o绑定进程
  - • Foreground processes
    - •前台进程
  - • Background processes
    - •后台进程



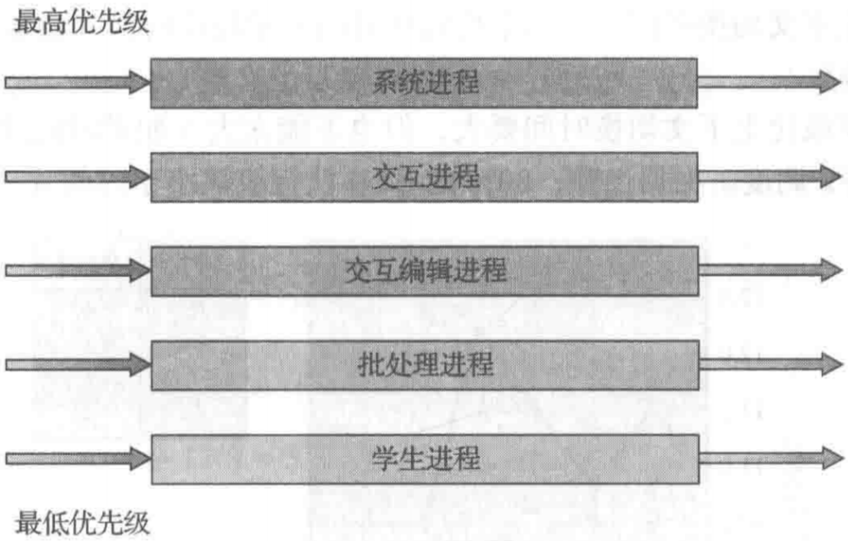
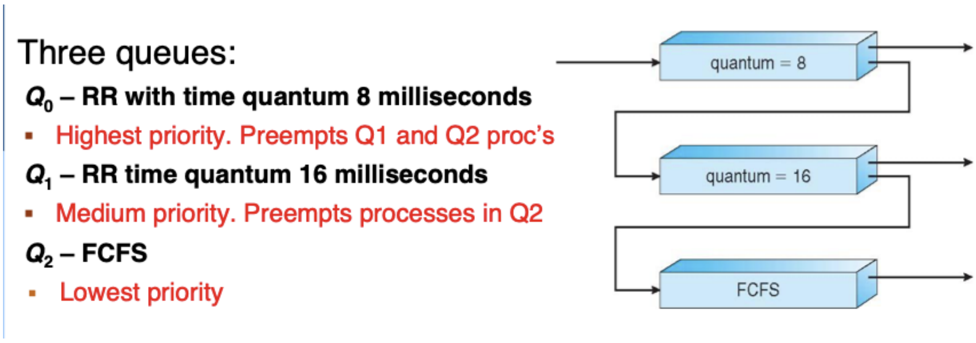


图 5-6 多级队列调度

- Multilevel Feedback Queue Scheduling
  - Multilevel feedback queues - automatically place processes into priority levels based on their CPU burst behavior.
    - 多级反馈队列——根据进程的CPU突发行为自动将其置于优先级级别。
  - • I/O-intensive processes will end up on higher priority queues and CPU-intensive processes will end up on low priority queues.
    - I/ o密集型进程将最终位于高优先级队列中，而cpu密集型进程将最终位于低优先级队列中。
  - A process can move between the various queuesA multilevel feedback queue uses two basic rules:
    - 一个进程可以在不同的队列之间移动。一个多级反馈队列使用两个基本规则:
    - 1. A new process gets placed in the highest priority queue.
    - 2. If a process does not finish its quantum, then it will stay at the same priority level otherwise it moves to the next lower priority level
      - 1. 一个新的进程被放置在最高优先级队列中。
      - 2. 如果一个进程没有完成它的量，那么它将保持在相同的优先级级别，否则它将移动到下一个较低的优先级级别
  - A process can move between the various queues;
    - 进程可以在不同的队列之间移动;
    - - aging can be implemented this way.
      - -老化可以这样实现。
  - • Multilevel-feedback-queue scheduler defined by the following parameters:
    - •由以下参数定义的多级反馈队列调度程序:
    - – number of queues
    - – scheduling algorithms for each queue
    - – method used to determine when to upgrade a process

- – method used to determine when to demote a process
- – method used to determine which queue a process will enter when that process needs service
  - -队列数量
  - -每个队列的调度算法
  - -用于确定何时升级进程的方法
  - -用于确定何时降级进程的方法
  - -用于确定进程需要服务时进入哪个队列的方法
  - 

  - Scheduling:
  - 1. A new job enters queue Q0 which is served first-come first served
    - 1. 一个新的作业进入队列Q0，该队列先到先得
    - ▪ When it gains CPU, job receives 8 milliseconds
      - ▪当它获得CPU时，作业接收8毫秒
    - ▪ If it does not finish in 8 milliseconds, job is moved to queue Q1
      - ▪如果没有在8毫秒内完成，作业将被移动到队列Q1
  - 2. At Q1 job is again served RR and receives 16 additional milliseconds
    - 2. 在Q1，作业再次为RR提供服务，并接收额外的16毫秒
    - ▪ If it still does not complete, it is preempted and moved to queue Q2
      - ▪如果它仍然没有完成，它将被抢占并移动到队列Q2