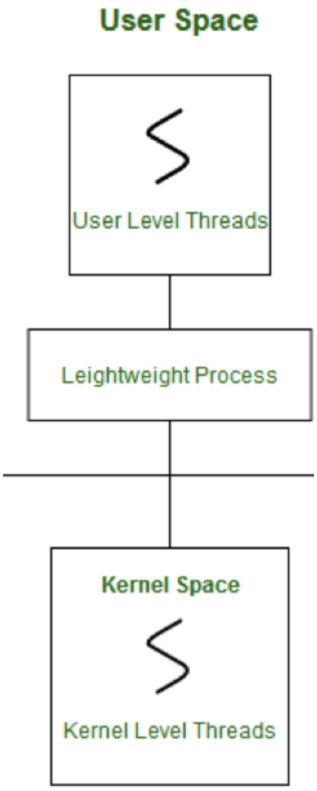
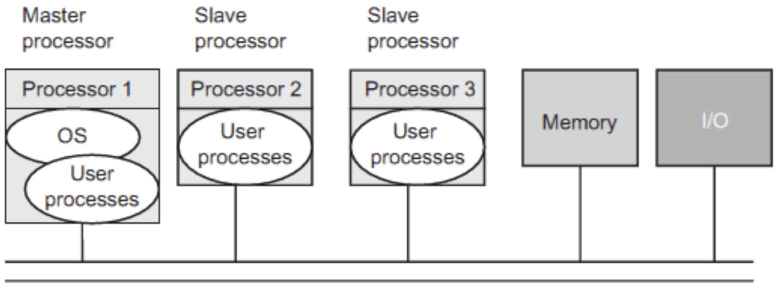
6 CPU Scheduling 2

- Thread Scheduling
    - Contention scope竞争范围
        - User threads are mapped to kernel threads.
            - 用户线程被映射到内核线程。
        - The thread models:
            - 线程模型:
            - • Many to One model
            - • One to One model
            - • Many to Many model.
        - The contention scope refers to the competition the User level threads to access the kernel resources.
            - 争用范围指的是用户级线程访问内核资源的竞争。
        - There are two possible contention scopes:
            - 有两种可能的争用范围:
            - Process Contention Scope PCS, a.k.a local contention scope.
                - 进程争用范围PCS，即本地争用范围。
            - System Contention Scope SCS, a.k.a global contention scope.
                - 系统争用范围SCS，又称全局争用范围。

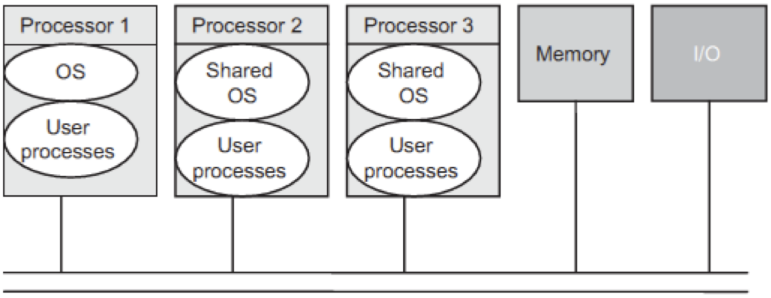


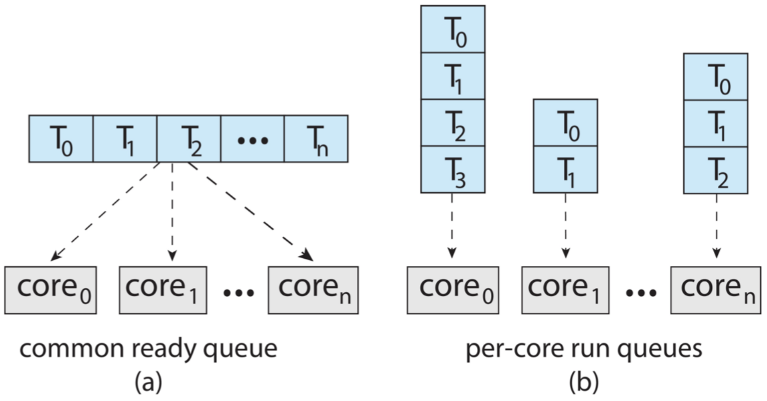
  - The basic levels to schedule threads:
    - Process Contention Scope (unbound threads) - competition for the CPU takes place among threads belonging to the same process. The thread library schedules the PCS thread to access there sources via available LWPs (priority as specified by the application developer during thread creation).
      - 进程争用范围(未绑定线程)——属于同一进程的线程之间发生对CPU的竞争。线程库调度PCS线程通过可用的lwp访问这些源(优先级由应用程序开发人员在线程创建期间指定)。
      - • It used many-to-many and many-to-one models.
        - •使用多对多和多对一模型。
    - o System Contention Scope (bound threads) - competition for the CPU takes place among all threads in the system. This scheme is used by the kernel to decide which kernel-level thread to schedule onto a CPU.
      - o系统争用范围(绑定线程)- CPU的竞争发生在系统中所有线程之间。内核使用此方案来决定将哪个内核级线程调度到CPU上。
      - • It used only a one-to-one model.
        - •它只使用了一对一模型。
- Multiple-Processor Scheduling
  - Approaches to Multiple-ProcessorScheduling
    - CPU scheduling more complex when multiple CPUs are available
      - 当多个CPU可用时，CPU调度更复杂
    - ❑We are focused on multiprocessor systems in which the processors are identical Homogeneous processors
      - 我们专注于多处理器系统，其中的处理器是相同的同质处理器
    - ❑ Asymmetric multiprocessing – only one processor accesses the system data structures, alleviating the need for data sharing
      - 不对称多处理——只有一个处理器访问系统数据结构，减少了数据共享的需要
    - ❑ Symmetric multiprocessing (SMP) – each processor is self-scheduling, all processes in common ready queue, or each has its own private queue of ready processes
      - 对称多处理(SMP)——每个处理器都是自调度的，所有进程在公共就绪队列中，或者每个进程都有自己私有的就绪进程队列
    - Asymmetric multiprocessing
      - Master – Slave Configuration主备配置
      - One processor as master and other processors in the system as slaves.
        - 一个处理器作为主处理器，系统中的其他处理器作为从处理器。

  - The master processor runs the OS and processes while slave processors run the processes only.
    - 主处理器运行操作系统和进程，而从处理器只运行进程。
  - The process scheduling is performed by the master processor.
    - 进程调度由主处理器执行。
  - The parallel processing is possible as a task can be broken down into sub-tasks and assigned to various processors.
    - 并行处理是可能的，因为一个任务可以被分解成子任务并分配给不同的处理器



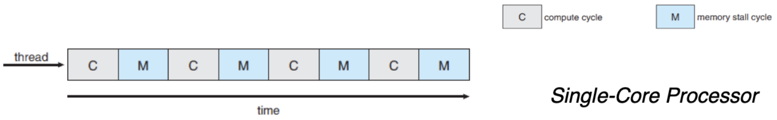
- Symmetric Configuration SMP
  - Any processor can access any device and can handle any interrupts generated on it.
    - 任何处理器都可以访问任何设备，并且可以处理在其上产生的任何中断。
  - Mutual exclusion must be enforced such that only one processor is allowed to execute the OS at one time.
    - 必须强制互斥，以便一次只允许一个处理器执行操作系统。
  - To prevent the concurrency of the processes many parts of the OS are independent of each other such as scheduler, file system call, etc.
    - 为了防止进程并发，操作系统的许多部分是相互独立的，例如调度器、文件系统调用等。



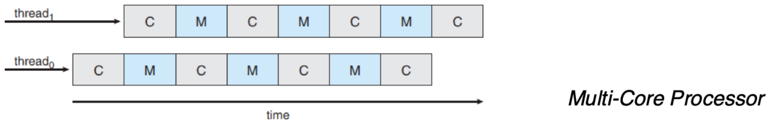


- Processor Affinity处理器亲和性
  - Processor affinity is the ability to direct a specific task, or process ,to use a specified core.
    - 处理器亲缘性是指示特定任务或进程使用指定核心的能力。
  - o The idea behind: if the process is directed to always use the same core it is possible that the process will run more efficiently because of the cache re-use.
    - o背后的想法:如果进程被引导到总是使用相同的核心，那么由于缓存重用，进程可能会更有效地运行。
    - • Note: If a process migrates from one CPU to another, the old instruction and address caches become invalid, and it will take time for caches on the newCPU to become 'populated'.
      - •注意:如果一个进程从一个CPU迁移到另一个CPU，旧的指令和地址缓存将失效，并且需要时间来填充新CPU上的缓存。
  - ➢Soft affinity – OSs try to keep a process running on the same processor but not guaranteeing it will do so.
    - 软关联-操作系统试图保持一个进程在同一处理器上运行，但不保证它会这样做。
  - ➢Hard affinity - allows a process to specify a subset of processors on which it may run.
    - 硬关联-允许进程指定它可以运行的处理器子集。
- Load Balancing
  - 保证所有处理器负载平衡,可以利用多个处理器有点,不让一个空闲,浪费资源
  - Load Balancing→a method of distributing work between the processors fairly in order to get optimal response time, resource utilization, and throughput.
    - 负载平衡——一种在处理器之间公平分配工作的方法，以获得最佳的响应时间、资源利用率和吞吐量。
    - ▪ Push migration = A system process periodically (e.g., every 200 ms)checks ready queues and moves (or push) processes to different queues (if need be).
      - Push迁移=系统进程定期(例如，每200毫秒)检查就绪队列，并将进程移动(或推送)到不同的队列(如果需要)。

        - 将进程从超载的推到push到空闲的处理器
      - • Pull migration = If scheduler finds there is no process in ready queue so it raids another processor’s run queue and transfers a process onto its own queue so it will have something to run (pulls a waiting task from a busy processor).
        - •拉迁移=如果调度程序发现就绪队列中没有进程，那么它会突袭另一个处理器的运行队列，并将一个进程转移到自己的队列中，这样它就有东西可以运行(从繁忙的处理器中拉出等待任务)。
        - 将空闲的处理器从一个忙的处理器上pull拉一个等待任务
  - Multicore Processors
    - A core executes one thread at a time. Hyper Threading allows multiple threads to run on each core of CPU
      - 一个核心一次执行一个线程。超线程允许多个线程在CPU的每个核心上运行
    - • Single-core processor spends time waiting for the data to become available(slowing or stopping of a process) = Memory stall.

      - •单核处理器花费时间等待数据可用(减慢或停止进程)=内存失速。花费太多时间切换和等待了
    - Solution!!! Multicore processor: to put multiple processor cores onto a single chip to run multiple kernel threads concurrently

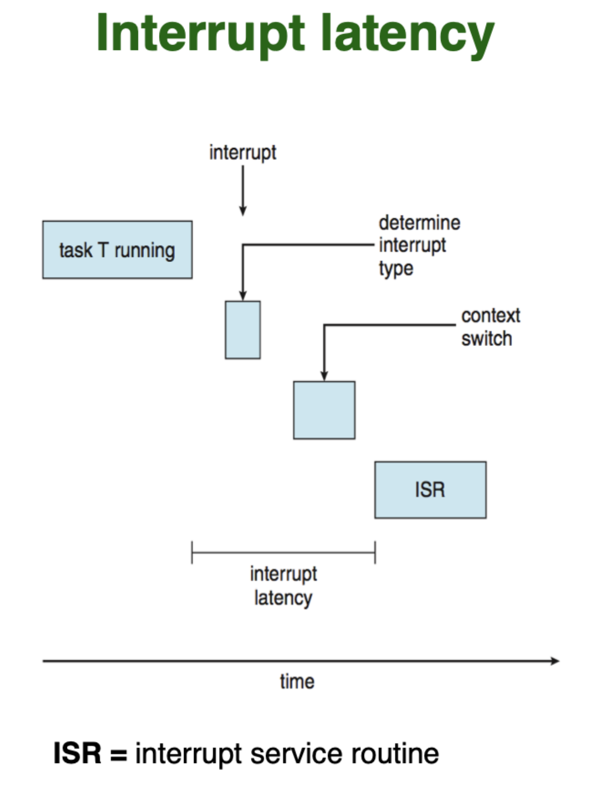
      

      - 解决方案! !多核处理器:将多个处理器内核放在单个芯片上，以并发地运行多个内核线程
    - How do we execute multiple threads on same core?
      - 我们如何在同一个核心上执行多个线程?
      - Techniques for multithreading:
        - 多线程技术:
        - ▪ Coarse-grained multithreading - switching between threads only when one thread blocks (long latency event such as a memory stall occurs).
          - 粗粒度多线程-只有当一个线程阻塞(长延迟事件，如内存失速发生)时才在线程之间切换。
        - ▪ Fine-grained multithreading - instructions “scheduling”among threads obeys a Round Robin policy.
          - ▪细粒度多线程-指令“调度”之间的线程遵循一个轮询策略。切换的成本很小,架构设计有线程切换的逻辑
- Real-Time CPU Scheduling

- 实时cpu调度
- Characteristics of a RTOS
    - A real-time operating system RTOS are deadline driven.实时操作系统RTOS是期限驱动的。
        - Examples: the patient monitoring in a hospital intensive-care unit, the autopilot in an aircraft, radar systems, robot control in an automated factory, etc.
            - 例如:医院重症监护病房的病人监护、飞机上的自动驾驶仪、雷达系统、自动化工厂中的机器人控制等等。
        - ❑Hard RTOS – is one that must meet its deadline; otherwise, it will cause unacceptable damage or a fatal error to the system.
            - 硬的RTOS——必须满足它的最后期限;否则，将对系统造成不可接受的损害或致命的错误。
        - ❑Soft RTOS – an associated deadline that is desirable but not necessary; it still makes sense to schedule and complete the task even if it has passed its deadline.
            - 3、软RTOS——相关的截止日期是理想的，但不是必需的;即使已经超过了截止日期，安排和完成任务仍然是有意义的。优先于其他非关键进程
    - The timing constraints are in the form of period and deadline.
        - 时间约束以期限和截止日期的形式存在。
    - The period is the amount of time between iterations of a regularly repeated task.
        - 周期是定期重复任务的迭代之间的时间量。
    - The deadline is a constraint of the maximum time limit within which the operation must be complete.
        - 截止日期是操作必须完成的最大时间限制的约束。
        - ❑Aperiodic tasks (random time) has irregular arrival times and either soft or hard deadlines.
            - 3、非周期性任务(随机时间)有不规律的到达时间和软期限或硬期限。
        - ❑Periodic tasks (repeated tasks), the requirement may be stated as “once per period T” or “exactly T units apart.”
            - 3、周期任务(重复任务)，要求可以表述为“每周期T次”或“正好间隔T个单位”。
- Issues in Real-time Scheduling
    - The major challenges for an RTOS is to schedule the real-time tasks.RTOS面临的主要挑战是调度实时任务。
    - Two types of latencies may delay the processing (performance):
        - 两种类型的延迟可能会延迟处理(性能):
        - 1. Interrupt latency – aka interrupt response time is the time elapsed between the last instruction executed on the current interrupted task and start of the

interrupt handler.

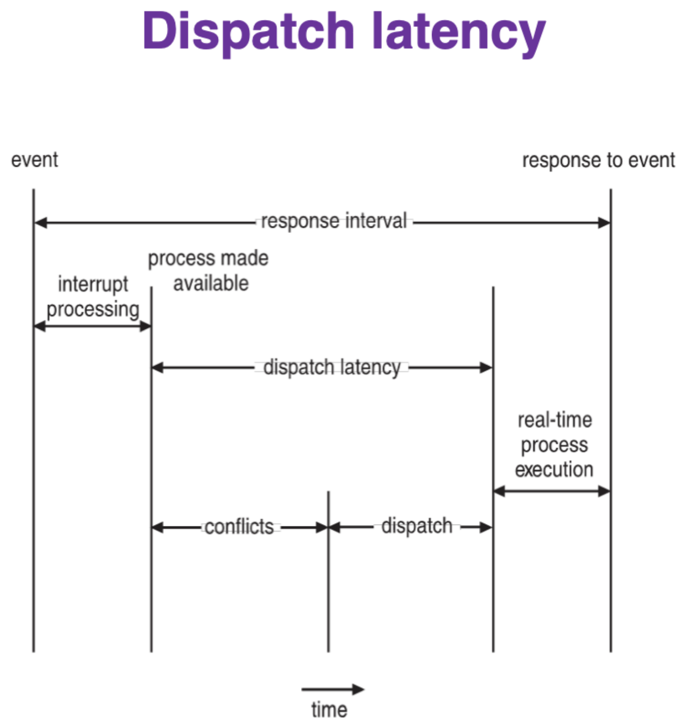
- 1. 中断延迟-又名中断响应时间是在当前中断任务上执行的最后一条指令和中断处理程序开始之间经过的时间。cpu收到中断到中断处理程序开始的时间
- An interrupt is a signal emitted by a device attached to a computer or from a program within the computer. It requires the operating system to stop and figure out what to do next.
    - 中断是由连接到计算机上的设备或计算机内的程序发出的信号。它需要操作系统停下来思考下一步该做什么。



Interrupt latency

ISR = interrupt service routine

- 2. Dispatch latency – time it takes for the dispatcher to stop one process and start another running. To keep dispatch latency low is to provide preemptive kernels.
    - 2. 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。为了保持较低的调度延迟，需要提供抢占式内核



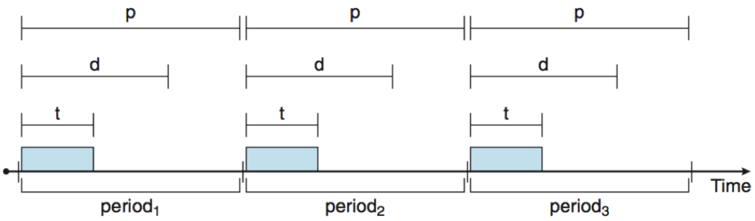
Dispatch latency

      - 从停止一个进程到启动另一个进程需要的时间
- Real-Time CPU Scheduling
  - The RTOS schedules all tasks according to the deadline information and ensures that all deadlines are met.
    - RTOS根据截止日期信息调度所有任务，并确保所有任务都在截止日期前完成。
  - Static scheduling. A schedule is prepared before execution of the application begins.
    - 静态调度。计划是在应用程序开始执行之前准备好的。
  - Priority-based scheduling. The priority assigned to the tasks depends on how quickly a task has to respond to the event.
    - 基于优先级的调度。分配给任务的优先级取决于任务对事件的响应速度。
  - Dynamic scheduling. There is complete knowledge of tasks set, but new arrivals are not known. Therefore, the schedule changes over the time.
    - 动态调度。对任务集有完整的了解，但不知道新到达的任务。因此，时间表会随着时间的推移而变化。
- The RTOS schedules all tasks according to the deadline information and ensures that all deadlines are met.
  - RTOS根据截止日期信息调度所有任务，并确保所有任务都在截止日期前完成。
- Static scheduling. A schedule is prepared before execution of the application begins.
  - 静态调度。计划是在应用程序开始执行之前准备好的。
- Priority-based scheduling. The priority assigned to the tasks depends on how quickly a task has to respond to the event.
  - 基于优先级的调度。分配给任务的优先级取决于任务对事件的响应速度。

- Dynamic scheduling. There is complete knowledge of tasks set ,but new arrivals are not known. Therefore, the schedule changes over the time.
    - 动态调度。对任务集有完整的了解，但不知道新到达的任务。因此，时间表会随着时间的推移而变化。
    - Rate-Monotonic Scheduling
        - Rate-Monotonic调度
        - ➢Missed Deadlines with Rate Monotonic Scheduling
            - (四)错过期限，率单调调度
    - ▪ Earliest-Deadline-First Scheduling
        - ▪最早截止日期优先调度
    - ▪ Proportional Share Scheduling
        - ▪比例共享调度
- Characteristics of processes
    - Processes are considered periodic (repeated tasks).
        - 流程被认为是周期性的(重复的任务)。
    - A periodic process has:
        - 周期过程有:
        - - processing time t,
            - -处理时间t;
        - - deadline d by which it must be serviced by the CPU, and
            - - CPU必须服务的截止日期，以及
            - 截止期限为d
        - - period p.
            - -周期p。

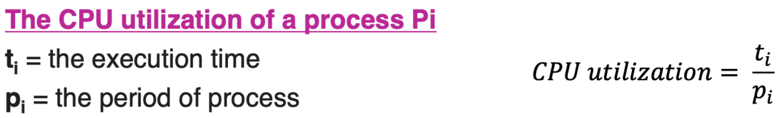
                $$0 \leq t \leq d \leq p$$

        - Rate of a periodic task is 1/p
            - 周期任务的速率为1/p
        - A process may have to announce its deadline requirements to the scheduler.
            - 进程可能必须向调度器宣布其截止日期要求。



- Rate Monotonic Scheduling (RMS)

- it is a static priority-based preemptive scheduling algorithm.The task with the shortest period will always preempt the executing task.
  - 它是一种基于静态优先级的抢占调度算法。周期最短的任务总是抢占正在执行的任务。
- The shortest period = the highest priority;
  - 最短的周期=最高的优先级;

    The CPU utilization of a process Pi

    t_i = the execution time

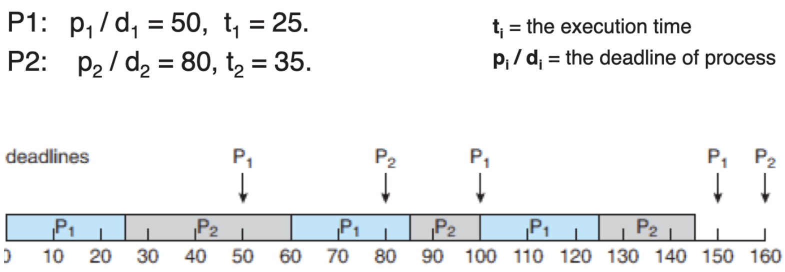
    p_i = the period of process

    $$CPU\ utilization = \frac{t_i}{p_i}$$

- To meet all deadlines in the system, the following must be satisfied:

  $$\sum_i \frac{t_i}{p_i} \leq 1$$

- Earliest-Deadline-First Scheduling(EDF)
  - The scheduling criterion is based on the deadline of the processes.
    - 调度标准基于流程的截止日期。
  - The processes / tasks need not be periodic.
    - 进程/任务不需要是周期性的。
  - Dynamically assigns priorities according to deadline.
    - 根据截止日期动态分配优先级。
    - • the earlier the deadline = the higher the priority;
      - •截止日期越早=优先级越高;

        P1: $p_1 / d_1 = 50$, $t_1 = 25$.

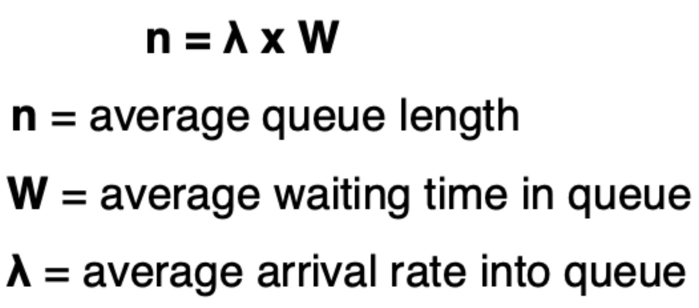
        P2: $p_2 / d_2 = 80$, $t_2 = 35$.

        t_i = the execution time

        p_i / d_i = the deadline of process

- Proportional Share Scheduling
  - Scheduling that pre-allocates certain amount of CPU time to each of the processes.
    - 为每个进程预先分配一定数量的CPU时间的调度。
  - Fair-share scheduler
    - 公平调度器
    - • Guarantee that each process obtain a certain percentage of CPU time
      - •保证每个进程获得一定百分比的CPU时间
    - • Not optimized for turnaround or response time
      - •没有优化周转时间或响应时间
  - o T shares are allocated among all processes in the system

            - 在系统中所有进程之间分配T个共享
        - o An application receives N shares where N < T
            - o应用程序获得N股，其中N <T
        - o This ensures each application will receive N / T of the total processor time
            - 这确保每个应用程序将接收到总处理器时间的N / T
            - Example: T = 100 shares is to be divided among three processes, A, B, and C.A is assigned 50 shares, B is assigned 15 shares, and C is assigned 20 shares.
                - 例如:T = 100股将分配给A、B和C三个进程。A分配50股， B分配15股， C分配20股。
            - 准入控制
                - 剩下100-50-15-20 = 15,所以如果有一个30的,不能让他进去
- Algorithm Evaluation
    - Deterministic Modeling
        - Takes a particular predetermined workload and defines theperformance of each algorithm for that workload.
        - What algorithm can provide the minimum average waiting time?
        - 获取一个特定的预定工作负载，并为该工作负载定义每个算法的性能。
        - 什么算法可以提供最小的平均等待时间?
        - 比较各个模型的平均等待时间
    - Queueing Models
        - If we define a queue for the CPU and a queue for each I/Odevice, we can test the various scheduling algorithms usingqueueing theory.
        - o Little ’ s formula – processes leaving queue must equal processes arriving, thus:
        - 如果我们为CPU定义一个队列，为每个I/Odevice定义一个队列，我们可以使用队列理论测试各种调度算法。
        - o利特尔公式-离开队列的进程必须等于到达队列的进程，因此:
        -



$$\mathbf{n} = \boldsymbol{\lambda} \times \mathbf{W}$$

n = average queue length

W = average waiting time in queue

λ = average arrival rate into queue

        - if on average 7 processes arrive per second, and normally 14processes in queue, then average wait time per process = 2 seconds
        - 如果平均每秒有7个进程到达，并且通常有14个进程在队列中，那么每个进程的平均等待时间= 2秒