- **操作系统的五大功能，分别为：作业管理、文件管理、存储管理、输入输出设备管理、进程及处理机管理**
- 线程实际工作的单元。 一个进程管理一个或者多个线程。 当线程数量超过CPU数量时，CPU必须进行调度，就是分配**时间片**。（通常很短，Linux5ms-800ms）线程切换时要进行**上下文切换**。
- 主流操作系统里，一般进程是不能控制自己线程的执行程序的。
- corei7 超线程技术。
- 时间片由操作系统内核的调度程序分配给每个进程。

首先，内核会给每个进程分配相等的初始时间片，然后每个进程轮番地执行相应的时间，当所有进程都处于时间片耗尽的状态时，

内核会重新为每个进程计算并分配时间片，如此往复。

通常状况下，一个系统中所有的进程被分配到的时间片长短并不是相等的，尽管初始时间片基本相等（在Linux系统中，初始时间片也不相等，而是各自父进程的一半），

系统通过测量进程处于“睡眠”和“正在运行”状态的时间长短来计算每个进程的交互性，交互性和每个进程预设的静态优先级（Nice值）的叠加即是动态优先级，

动态优先级按比例缩放就是要分配给那个进程时间片的长短。一般地，为了获得较快的响应速度，

交互性强的进程（即趋向于IO消耗型）被分配到的时间片要长于交互性弱的（趋向于处理器消耗型）进程。

- 临界区问题：

- - 临界区: 一种代码段，在其中可能发生多个线程共同改变变量、读写文件等操作，其要求当一个线程进入临界区时，其他线程不能进入。从而避免出现同时读写的问题。(实际上，临界区只需保证可以有多个读者同时执行读取操作，或唯一写者执行写入操作)
- - 进入区: 判断线程能否进入临界区的代码段。
- - 退出区: 线程离开临界区后可能对其执行的某些操作。
- ◦ 剩余区: 线程完全退出临界区和退出区后的剩下全部代码。

- 临界区问题解决方法：

- 软件：解决临界区问题，主要在于保证资源的互斥访问，以及避免出现饥饿现象。

Peterson算法提供了一个合理的思路: 设置旗标数组flag标记请求进入临界区的线程，设置turn表示可以进入临界区的线程，在进入区进行双重判断，两个线程同时对turn赋值只会有一个保留下来，从而确保资源访问的互斥。 而在退出区，对flag旗标进行了false处理，从而保证了”前进”原则，避免了剩余区中的线程持续抢

占造成其他线程饥饿。

- 硬件：Peterson算法是基于软件的实现，而从硬件层面也可以解决此问题，硬件方面的处理主要在于线程修改共享资源时是原子地，即不可被中断。比如机器提供了能够原子执行的指令，那么我们可以通过简单的修改布尔变量来实现互斥，因为加锁的过程是原子的。而对于可能造成饥饿的问题，只需在退出区对等待列表进行一定处理，保证”前进”原则即可。

- 简单来说，无论硬件还是软件的实现，本质都是通过加锁。只是通过硬件的特性，可以提高效率，同时也简化了临界区的实现难度。

- 信号量

- 简单来说，信号量是某个实例可用资源的计数，初始为该实例可用资源的数量，而每当线程需要使用，则调用wait()方法减少信号量，释放资源时调用signal()方法增加信号量，故信号量为0表示所有资源都在被使用，线程使用资源的请求不被允许。

信号量主要分为计数信号量和二进制信号量，前者主要针对一个实例有多个资源的情况，值域不受限制，而后者信号量仅为0或1，也就是说线程之间访问该资源是互斥的，也可称作互斥锁。

同临界区问题的前提: 必须保证执行信号量操作wait(),signal()是原子地。

- 饥饿问题

- 在信号量大于0时从队列中取出哪个进程是需要讨论的问题，选择合适的调度方式很重要。FIFO(先进先出)可以解决，但如果LIFO(后进先出)调度则可能会造成部分进程无限期阻塞，也就是饥饿问题。

- 忙等问题

当进程请求进入临界区而没有被允许时，如果此进程开始在进入区连续循环请求，则会消耗大量性能，浪费了部分CPU时间片。这种加锁方式称为自旋锁，即进程在等待锁时仍然在运行，此方法会造成忙等的性能浪费，但同时也比阻塞-唤醒机制效率更高，避免了阻塞到唤醒的上下文切换。

如果要克服忙等问题，可以在进入区增加当信号量小于0时，进程阻塞自己，进入等待队列；当临界区内的线程执行完毕后，唤醒等待队列中的进程。同时要保证等待队列调度的算法合理性，避免某进程无限期等待。

- 死锁问题

死锁问题就是多个进程无限等待某个事件，而该事件是由这些进程来产生的，这样就会造成”第二十二条军规”中的问题，进程之间互相等待，无法前进。在此不讨论死锁问题，只介绍可能出现的死锁情况。

- - 处理器解决方案

- 单个处理器: 单处理器时无须担心并行运算造成的同步问题，所以简单在wait()和signal()中禁止临界区中进程的中断即可

- 多处理器: 操作系统对于多处理器调度分为SMP和非SMP的情况(SMP为对称多处理，处理器各自控制各自的调度，而非SMP为某一个处理器作为中控，管理其他处理器的调度)。
- - 非对称多处理: 对于非对称多处理，由于有中央处理器来调度，可以简单使用自旋锁来进行忙等，系统来决策等待锁的进程的调度问题。
- - 对称多处理: 对于对称多处理系统，就要自行实现上述的信号量等待列表，以及等待锁时的阻塞——唤醒机制。

- 经典同步问题

- 读者-写者问题：仅进行读取操作的为读者，而读写操作均需要的为写者。仍然以刚才的买票问题为例，我们不难发现，当同时读取时，不会出现问题，当唯一写入时，也不会出现问题，但是当同时进行读写时，则发生了数据错误问题。

所以读者——写者问题应当保证同一时间写者的唯一性及读者要等待写者完成后再执行。

- 哲学家进餐问题（死锁问题）
- - 死锁问题预防：死锁预防 死锁避免 死锁检测 死锁恢复。

线程进程之间的通信

进程

同一主机

- unix进程间通信方式：管道PIPE，有名管道FIFO，信号Signal
- System V进程通信方式：包括信号量(Semaphore), 消息队列(Message Queue), 和共享内存(Shared Memory)

网络主机

- RPC: Remote Procedure Call 远程过程调用
- Socket: 当前最流行的网络通信方式, 基于TCP/IP协议的通信方式.

各自特点如下：

- 管道(PIPE)：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系(父子进程)的进程间使用。另外管道传送的是无格式的字节流，并且管道缓冲区的大小是有限的（管道缓冲区存在于内存中，在管道创建时，为缓冲区分配一个页面大小）。
- 有名管道 (FIFO)： 有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信。

- 信号(Signal)： 信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。
- 信号量(Semaphore)：信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。
- 消息队列(Message Queue)：消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。
- 共享内存(Shared Memory)：共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号量，配合使用，来实现进程间的同步和通信。
- 套接字(Socket)： 套解口也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同主机间的进程通信。

Linux 系统中线程的通信方式

- 锁机制：包括互斥锁、条件变量、读写锁

  互斥锁提供了以排他方式防止数据结构被并发修改的方法。

  读写锁允许多个线程同时读共享数据，而对写操作是互斥的。

  条件变量可以以原子的方式阻塞进程，直到某个特定条件为真为止。对条件的测试是在互斥锁的保护下进行的。条件变量始终与互斥锁一起使用。

- 信号量机制(Semaphore)：包括无名线程信号量和命名线程信号量
- 信号机制(Signal)：类似进程间的信号处理

线程间的通信目的主要是用于线程同步，所以线程没有像进程通信中的用于数据交换的通信机制。

线程与进程

- **进程在执行过程中拥有独立的内存单元，而多个线程共享内存**