并发、并行、串行？

并发：多个线程操作同一资源  
并行：多个线程操作多个资源  
串行：多个线程操作顺序操作单个资源

操作系统是什么？

1.操作系统（Operating System，简称 OS）是管理计算机硬件与软件资源的程序，是计算机的基石。

2.操作系统本质上是一个运行在计算机上的软件程序 ，用于管理计算机硬件和软件资源。 举例：运行在你电脑上的所有应用程序都通过操作系统来调用系统内存以及磁盘等等硬件。

3.操作系统存在屏蔽了硬件层的复杂性。 操作系统就像是硬件使用的负责人，统筹着各种相关事项。

4.操作系统的内核（Kernel）是操作系统的核心部分，它负责系统的内存管理，硬件设备的管理，文件系统的管理以及应用程序的管理。 内核是连接应用程序和硬件的桥梁，决定着系统的性能和稳定性。

用户态与系统态？

根据进程访问资源的特点，我们可以把进程在系统上的运行分为两个级别：

1.用户态(user mode) : 用户态运行的进程可以直接读取用户程序的数据。

2.系统态(kernel mode):可以简单的理解系统态运行的进程或程序几乎可以访问计算机的任何资源，不受限制。

说了用户态和系统态之后，那么什么是系统调用呢？

我们运行的程序基本都是运行在用户态，如果我们调用操作系统提供的系统态级别的子功能咋办呢？那就需要系统调用了！也就是说在我们运行的用户程序中，凡是与系统态级别的资源有关的操作（如文件管理、进程控制、内存管理等)，都必须通过系统调用方式向操作系统提出服务请求，并由操作系统代为完成。

系统调用按功能分以下五类

* 设备管理。完成设备的请求或释放，以及设备启动等功能。
* 文件管理。完成文件的读、写、创建及删除等功能。
* 进程控制。完成进程的创建、撤销、阻塞及唤醒等功能。
* 进程通信。完成进程之间的消息传递或信号传递等功能。
* 内存管理。完成内存的分配、回收以及获取作业占用内存区大小及地址等功能

进程和线程的联系与区别

* 进程（Process）是系统进行资源分配和调度的基本单位，线程（Thread）是CPU调度和分派的基本单位；
* 线程依赖于进程而存在，一个进程至少有一个线程；
* 进程有自己的独立地址空间，线程共享所属进程的地址空间；
* 进程是拥有系统资源的一个独立单位，而线程自己基本上不拥有系统资源，只拥有一点在运行中必不可少的资源(如程序计数器,一组寄存器和栈)，和其他线程共享本进程的相关资源如内存、I/O、cpu等；
* 在进程切换时，涉及到整个当前进程CPU环境的保存环境的设置以及新被调度运行的CPU环境的设置，而线程切换只需保存和设置少量的寄存器的内容，并不涉及存储器管理方面的操作，可见，进程切换的开销远大于线程切换的开销；
* 线程之间的通信更方便，同一进程下的线程共享全局变量等数据，而进程之间的通信需要以进程间通信(IPC)的方式进行；
* 多线程程序只要有一个线程崩溃，整个程序就崩溃了，但多进程程序中一个进程崩溃并不会对其它进程造成影响，因为进程有自己的独立地址空间，因此多进程更加健壮

同一进程中的线程可以共享哪些数据？

* 进程代码段
* 进程的公有数据（全局变量、静态变量...）
* 进程打开的文件描述符
* 进程的当前目录
* 信号处理器/信号处理函数：对收到的信号的处理方式
* 进程ID与进程组ID

线程独占哪些资源？

* 线程ID
* 一组寄存器的值
* 线程自身的栈（堆是共享的）
* 错误返回码：线程可能会产生不同的错误返回码，一个线程的错误返回码不应该被其它线程修改；
* 信号掩码/信号屏蔽字(Signal mask)：表示是否屏蔽/阻塞相应的信号（SIGKILL,SIGSTOP除外）

**进程有哪几种状态?**

一般把进程大致分为 5 种状态

* 创建状态(new) ：进程正在被创建，尚未到就绪状态。
* 就绪状态(ready) ：进程已处于准备运行状态，即进程获得了除了处理器之外的一切所需资源，一旦得到处理器资源(处理器分配的时间片)即可运行。
* 运行状态(running) ：进程正在处理器上上运行(单核 CPU 下任意时刻只有一个进程处于运行状态)。
* 阻塞状态(waiting) ：又称为等待状态，进程正在等待某一事件而暂停运行如等待某资源为可用或等待 IO 操作完成。即使处理器空闲，该进程也不能运行。
* 结束状态(terminated) ：进程正在从系统中消失。可能是进程正常结束或其他原因中断退出运行。

**进程间的通信IPC常见的的有哪几种方式呢?**

大概有 7 种常见的进程间的通信方式。

* **管道/匿名管道(Pipes)**：用于具有亲缘关系的父子进程间或者兄弟进程之间的通信。
* **有名管道(Names Pipes)** : 匿名管道由于没有名字，只能用于亲缘关系的进程间通信。为了克服这个缺点，提出了有名管道。有名管道严格遵循先进先出(first in first out)。有名管道以磁盘文件的方式存在，可以实现本机任意两个进程通信。
* **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；
* **消息队列(Message Queuing)**：消息队列是消息的链表,具有特定的格式,存放在内存中并由消息队列标识符标识。管道和消息队列的通信数据都是先进先出的原则。与管道（无名管道：只存在于内存中的文件；命名管道：存在于实际的磁盘介质或者文件系统）不同的是消息队列存放在内核中，只有在内核重启(即，操作系统重启)或者显式地删除一个消息队列时，该消息队列才会被真正的删除。消息队列可以实现消息的随机查询,消息不一定要以先进先出的次序读取,也可以按消息的类型读取.比 FIFO 更有优势。消息队列克服了信号承载信息量少，管道只能承载无格式字 节流以及缓冲区大小受限等缺点。
* **信号量(Semaphores)** ：信号量是一个计数器，用于多进程对共享数据的访问，信号量的意图在于进程间同步。这种通信方式主要用于解决与同步相关的问题并避免竞争条件。
* **共享内存(Shared memory)**：使得多个进程可以访问同一块内存空间，不同进程可以及时看到对方进程中对共享内存中数据的更新。这种方式需要依靠某种同步操作，如互斥锁和信号量等。可以说这是最有用的进程间通信方式。
* **套接字(Sockets)**: 此方法主要用于在客户端和服务器之间通过网络进行通信。套接字是支持 TCP/IP 的网络通信的基本操作单元，可以看做是不同主机之间的进程进行双向通信的端点，简单的说就是通信的两方的一种约定，用套接字中的相关函数来完成通信过程。

**线程间的同步的方式有哪些呢?**

线程同步是两个或多个共享关键资源的线程的并发执行。应该同步线程以避免关键的资源使用冲突。操作系统一般有下面三种线程同步的方式：

* **互斥量(Mutex)**：采用互斥对象机制，只有拥有互斥对象的线程才有访问公共资源的权限。因为互斥对象只有一个，所以可以保证公共资源不会被多个线程同时访问。比如 Java 中的 synchronized 关键词和各种 Lock 都是这种机制。
* **信号量(Semphares)** ：它允许同一时刻多个线程访问同一资源，但是需要控制同一时刻访问此资源的最大线程数量。
* **事件(Event)** :Wait/Notify：通过通知操作的方式来保持多线程同步，还可以方便的实现多线程优先级的比较操作。

**操作系统中进程的调度算法有哪些吗?**

为了确定首先执行哪个进程以及最后执行哪个进程以实现最大 CPU 利用率，计算机科学家已经定义了一些算法，它们是：

* **先到先服务(FCFS)调度算法** : 从就绪队列中选择一个最先进入该队列的进程为之分配资源，使它立即执行并一直执行到完成或发生某事件而被阻塞放弃占用 CPU 时再重新调度。
* **短作业优先(SJF)的调度算法** : 从就绪队列中选出一个估计运行时间最短的进程为之分配资源，使它立即执行并一直执行到完成或发生某事件而被阻塞放弃占用 CPU 时再重新调度。
* **时间片轮转调度算法** : 时间片轮转调度是一种最古老，最简单，最公平且使用最广的算法，又称 RR(Round robin)调度。每个进程被分配一个时间段，称作它的时间片，即该进程允许运行的时间。
* **多级反馈队列调度算法** ：前面介绍的几种进程调度的算法都有一定的局限性。如短进程优先的调度算法，仅照顾了短进程而忽略了长进程 。多级反馈队列调度算法既能使高优先级的作业得到响应又能使短作业（进程）迅速完成。，因而它是目前被公认的一种较好的进程调度算法，UNIX 操作系统采取的便是这种调度算法。
* **优先级调度** ： 为每个流程分配优先级，首先执行具有最高优先级的进程，依此类推。具有相同优先级的进程以 FCFS 方式执行。可以根据内存要求，时间要求或任何其他资源要求来确定优先级

**什么是死锁**

多个进程可以竞争有限数量的资源。当一个进程申请资源时，如果这时没有可用资源，那么这个进程进入等待状态。有时，如果所申请的资源被其他等待进程占有，那么该等待进程有可能再也无法改变状态。这种情况称为 **死锁**。

**产生死锁的四个必要条件是什么?**

如果系统中以下四个条件同时成立，那么就能引起死锁：

* **互斥**：资源必须处于非共享模式，即一次只有一个进程可以使用。如果另一进程申请该资源，那么必须等待直到该资源被释放为止。
* **占有并等待**：一个进程至少应该占有一个资源，并等待另一资源，而该资源被其他进程所占有。
* **非抢占**：资源不能被抢占。只能在持有资源的进程完成任务后，该资源才会被释放。
* **循环等待**：有一组等待进程 {P0, P1,..., Pn}， P0 等待的资源被 P1 占有，P1 等待的资源被 P2 占有，......，Pn-1 等待的资源被 Pn 占有，Pn 等待的资源被 P0 占有。

注意，只有四个条件同时成立时，死锁才会出现。

解决死锁的方法可以从多个角度去分析，一般的情况下，有预防，避免，检测和解除四种。

* **预防**---是采用某种策略，限制并发进程对资源的请求，从而使得死锁的必要条件在系统执行的任何时间上都不满足。
* **避免**---则是系统在分配资源时，根据资源的使用情况提前做出预测，从而避免死锁的发生
* **检测**---是指系统设有专门的机构，当死锁发生时，该机构能够检测死锁的发生，并精确地确定与死锁有关的进程和资源。
* **解除**---是与检测相配套的一种措施，用于将进程从死锁状态下解脱出来。

-----------------------------------死锁的预防

* 破坏第一个条件 互斥条件：使得资源是可以同时访问的，这是种简单的方法，磁盘就可以用这种方法管理，但是我们要知道，有很多资源 往往是不能同时访问的 ，所以这种做法在大多数的场合是行不通的。
* 破坏第三个条件 非抢占 ：也就是说可以采用 剥夺式调度算法，但剥夺式调度方法目前一般仅适用于 主存资源 和 处理器资源 的分配，并不适用于所以的资源，会导致 资源利用率下降。

所以一般比较实用的 预防死锁的方法，是通过考虑破坏第二个条件和第四个条件。

* 静态分配策略可以破坏死锁产生的第二个条件（占有并等待）。所谓静态分配策略，就是指一个进程必须在执行前就申请到它所需要的全部资源，并且知道它所要的资源都得到满足之后才开始执行。进程要么占有所有的资源然后开始执行，要么不占有资源，不会出现占有一些资源等待一些资源的情况。 静态分配策略逻辑简单，实现也很容易，但这种策略 严重地降低了资源利用率，因为在每个进程所占有的资源中，有些资源是在比较靠后的执行时间里采用的，甚至有些资源是在额外的情况下才是用的，这样就可能造成了一个进程占有了一些 几乎不用的资源而使其他需要该资源的进程产生等待 的情况。
* 层次分配策略破坏了产生死锁的第四个条件(循环等待)。在层次分配策略下，所有的资源被分成了多个层次，一个进程得到某一次的一个资源后，它只能再申请较高一层的资源；当一个进程要释放某层的一个资源时，必须先释放所占用的较高层的资源，按这种策略，是不可能出现循环等待链的，因为那样的话，就出现了已经申请了较高层的资源，反而去申请了较低层的资源，不符合层次分配策略，证明略。

-----------------------------------死锁的避免

了解：上面提到的 破坏 死锁产生的四个必要条件之一就可以成功 预防系统发生死锁 ，但是会导致 低效的进程运行 和 资源使用率 。而死锁的避免相反，它的角度是允许系统中同时存在四个必要条件 ，只要掌握并发进程中与每个进程有关的资源动态申请情况，做出 明智和合理的选择 ，仍然可以避免死锁，因为四大条件仅仅是产生死锁的必要条件。

我们将系统的状态分为 **安全状态** 和 **不安全状态** ，每当在未申请者分配资源前先测试系统状态，若把系统资源分配给申请者会产生死锁，则拒绝分配，否则接受申请，并为它分配资源。如果操作系统能够保证所有的进程在有限的时间内得到需要的全部资源，则称系统处于安全状态，否则说系统是不安全的。很显然，系统处于安全状态则不会发生死锁，系统若处于不安全状态则可能发生死锁。

那么如何保证系统保持在安全状态呢？通过算法，其中最具有代表性的 避免死锁算法 就是 **Dijkstra 的银行家算法**，银行家算法用一句话表达就是：当一个进程申请使用资源的时候，银行家算法 通过先 试探 分配给该进程资源，然后通过 安全性算法 判断分配后系统是否处于安全状态，若不安全则试探分配作废，让该进程继续等待，若能够进入到安全的状态，则就 真的分配资源给该进程。

死锁的避免(银行家算法)改善解决了 资源使用率低的问题 ，但是它要不断地检测每个进程对各类资源的占用和申请情况，以及做 安全性检查 ，需要花费较多的时间。

-----------------------------------死锁的检测

了解：对资源的分配加以限制可以 **预防和避免** 死锁的发生，但是都不利于各进程对系统资源的充分共享。解决死锁问题的另一条途径是 **死锁检测和解除**  (这里突然联想到了乐观锁和悲观锁，感觉死锁的检测和解除就像是 乐观锁 ，分配资源时不去提前管会不会发生死锁了，等到真的死锁出现了再来解决嘛，而 死锁的预防和避免 更像是悲观锁，总是觉得死锁会出现，所以在分配资源的时候就很谨慎)。

具体：这种方法对资源的分配不加以任何限制，也不采取死锁避免措施，但系统 定时地运行一个 “**死锁检测**” 的程序，判断系统内是否出现死锁，如果检测到系统发生了死锁，再采取措施去解除它。

🡪死锁检测步骤

知道了死锁检测的原理，我们可以利用下列步骤编写一个 死锁检测 程序，检测系统是否产生了死锁。

1. 如果进程-资源分配图中**无环路**，则此时系统没有发生死锁
2. 如果进程-资源分配图中**有环路**，且**每个资源类仅有一个资源**，则系统中已经发生了死锁。
3. 如果进程-资源分配图中**有环路**，且涉及到的**资源类有多个资源**，此时系统未必会发生死锁。如果能在进程-资源分配图中找出一个 **既不阻塞又非独立的进程** ，该进程能够在有限的时间内归还占有的资源，也就是把边给消除掉了，重复此过程，直到能在有限的时间内 **消除所有的边** ，则不会发生死锁，否则会发生死锁。(消除边的过程类似于 拓扑排序)

-----------------------------------死锁的解除

当死锁检测程序检测到存在死锁发生时，应设法让其解除，让系统从死锁状态中恢复过来，常用的解除死锁的方法有以下四种：

* **立即结束所有进程的执行，重新启动操作系统** ：这种方法简单，但以前所在的工作全部作废，损失很大。
* **撤销涉及死锁的所有进程，解除死锁后继续运行** ：这种方法能彻底打破死锁的循环等待条件，但将付出很大代价，例如有些进程可能已经计算了很长时间，由于被撤销而使产生的部分结果也被消除了，再重新执行时还要再次进行计算。
* **逐个撤销涉及死锁的进程，回收其资源直至死锁解除。**
* **抢占资源** ：从涉及死锁的一个或几个进程中抢占资源，把夺得的资源再分配给涉及死锁的进程直至死锁解除。

============================内存管理=========================

**操作系统的内存管理主要是做什么？**

操作系统的内存管理主要负责内存的分配与回收（malloc 函数：申请内存，free 函数：释放内存），另外地址转换也就是将逻辑地址转换成相应的物理地址等功能也是操作系统内存管理做的事情。

**操作系统的内存管理机制了解吗？内存管理有哪几种方式?**

简单分为**连续分配管理方式**和**非连续分配管理方式**这两种。连续分配管理方式是指为一个用户程序分配一个连续的内存空间，常见的如 **块式管理** 。同样地，非连续分配管理方式允许一个程序使用的内存分布在离散或者说不相邻的内存中，常见的如**页式管理** 和 **段式管理**。

了解：

* 块式管理 ： 远古时代的计算机操系统的内存管理方式。将内存分为几个固定大小的块，每个块中只包含一个进程。如果程序运行需要内存的话，操作系统就分配给它一块，如果程序运行只需要很小的空间的话，分配的这块内存很大一部分几乎被浪费了。这些在每个块中未被利用的空间，我们称之为碎片。
* 页式管理 ：把主存分为大小相等且固定的一页一页的形式，页较小，相对相比于块式管理的划分力度更大，提高了内存利用率，减少了碎片。页式管理通过页表对应逻辑地址和物理地址。
* 段式管理 ： 页式管理虽然提高了内存利用率，但是页式管理其中的页实际并无任何实际意义。 段式管理把主存分为一段段的，段是有实际意义的，每个段定义了一组逻辑信息，例如,有主程序段 MAIN、子程序段 X、数据段 D 及栈段 S 等。 段式管理通过段表对应逻辑地址和物理地址。
* 段页式管理机制： 段页式管理机制结合了段式管理和页式管理的优点。简单来说段页式管理机制就是把主存先分成若干段，每个段又分成若干页，也就是说 **段页式管理机制** 中段与段之间以及段的内部的都是离散的。

简单来说：页是物理单位，段是逻辑单位。分页可以有效提高内存利用率，分段可以更好满足用户需求。

页表管理机制中有两个很重要的概念：快表和多级页表，这两个东西分别解决了页表管理中很重要的两个问题。介绍下？

首先，在分页内存管理中，很重要的两点是：1.虚拟地址到物理地址的转换要快。2.解决虚拟地址空间大，页表也会很大的问题。

快表--为了解决虚拟地址到物理地址的转换速度，操作系统在 **页表方案** 基础之上引入了 **快表** 来**加速虚拟地址到物理地址的转换**。

（了解：我们可以把快表理解为一种特殊的高速缓冲存储器（Cache），其中的内容是页表的一部分或者全部内容。作为页表的 Cache，它的作用与页表相似，但是提高了访问速率。由于采用页表做地址转换，读写内存数据时 CPU 要访问两次主存。有了快表，有时只要访问一次高速缓冲存储器，一次主存，这样可加速查找并提高指令执行速度。）

多级页表--引入多级页表的主要目的是为了避免把全部页表一直放在内存中占用过多空间，特别是那些根本就不需要的页表就不需要保留在内存中。

总结：为了提高内存的空间性能，提出了多级页表的概念；但是提到空间性能是以浪费时间性能为基础的，因此为了补充损失的时间性能，提出了快表（TLB）的概念。

**分页机制和分段机制有哪些共同点和区别呢？**

共同点 ：

* 分页机制和分段机制都是为了提高内存利用率，减少内存碎片。
* 页和段都是离散存储的，所以两者都是离散分配内存的方式。但是，每个页和段中的内存是连续的。

区别 ：

* 页的大小是固定的，由操作系统决定；而段的大小不固定，取决于我们当前运行的程序。
* 分页仅仅是为了满足操作系统内存管理的需求，而段是逻辑信息的单位，在程序中可以体现为代码段，数据段，能够更好满足用户的需要。

什么是**逻辑地址和物理地址？**

我们编程一般只可能和逻辑地址打交道，比如在 C 语言中，指针里面存储的数值就可以理解成为内存里的一个地址，这个地址也就是我们说的逻辑地址，**逻辑地址由操作系统决定**。物理地址指的是真实物理内存中地址，更具体一点来说就是内存地址寄存器中的地址。**物理地址是内存单元真正的地址。**

**什么是虚拟内存(Virtual Memory)?**

这个在我们平时使用电脑特别是 Windows 系统的时候太常见了。很多时候我们使用点开了很多占内存的软件，这些软件占用的内存可能已经远远超出了我们电脑本身具有的物理内存。为什么可以这样呢？

正是因为 虚拟内存 的存在，通过 **虚拟内存** 可以让程序可以拥有超过系统物理内存大小的可用内存空间。另外，虚拟内存为每个进程提供了一个一致的、私有的地址空间，它让每个进程产生了一种自己在独享主存的错觉（每个进程拥有一片连续完整的内存空间）。这样会更加有效地管理内存并减少出错。

虚拟内存是计算机系统内存管理的一种技术，我们可以手动设置自己电脑的虚拟内存。不要单纯认为虚拟内存只是“使用硬盘空间来扩展内存“的技术。虚拟内存的重要意义是它定义了一个连续的虚拟地址空间，并且 把内存扩展到硬盘空间。

**虚拟内存技术的实现呢？**

虚拟内存的实现需要建立在离散分配的内存管理方式的基础上。 虚拟内存的实现有以下三种方式：

* 请求分页存储管理
* 请求分段存储管理
* 请求段页式存储管理