# 2020 面试题总结——操作系统篇

**1、进程和线程的区别。**

进程是具有一定功能的程序关于某个数据集合上的一次运行活动，进程是系统进行资源调度和分配的一个独立单位。进程是通过进程控制块 PCB 来控制的，主要包括：进程描述（PID、用户标识、进程组关系）、进程控制（状态、优先级、入口地址、队列指针）、资源和使用状况（存储空间、文件）、CPU 现场（进程不执行时保存寄存器值、指向页表的指针）

线程是进程的实体，是 CPU 调度和分派的基本单位，它是比进程更小的能独立运行的基本单位。一个进程可以有多个线程，多个线程也可以并发执行。

**2、进程同步的几种方式。**

主要分为：管道、系统 IPC（包括消息队列、信号量、共享存储）、SOCKET

管道主要分为：普通管道 PIPE 、流管道（s_pipe）、命名管道（name_pipe）

- 管道是一种半双工的通信方式，数据只能单项流动，并且只能在具有亲缘关系的进程间流动，进程的亲缘关系通常是父子进程。
- 命名管道也是半双工的通信方式，它允许无亲缘关系的进程间进行通信。
- 信号量是一个计数器，用来控制多个进程对资源的访问，它通常作为一种锁机制。
- 消息队列是消息的链表，存放在内核中并由消息队列标识符标识。
- 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。
- 共享内存就是映射一段能被其它进程访问的内存，这段共享内存由一个进程创建，但是多个进程可以访问。

**3、线程间同步的方式。**

- **互斥量 Synchronized/Lock：**采用互斥对象机制，只要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。
- **信号量 Semaphare：**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（信号），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。

**4、什么是缓冲区溢出。有什么危害，其原因是什么。**

缓冲区溢出是指当计算机向缓冲区填充数据时超出了缓冲区本身的容量，溢出的数据覆盖在合法数据上。

危害有以下两点：

- 程序崩溃，导致拒绝的服务。
- 跳转并且执行一段恶意代码。

造成缓冲区溢出的主要原因是程序中没有仔细检查用户输入。

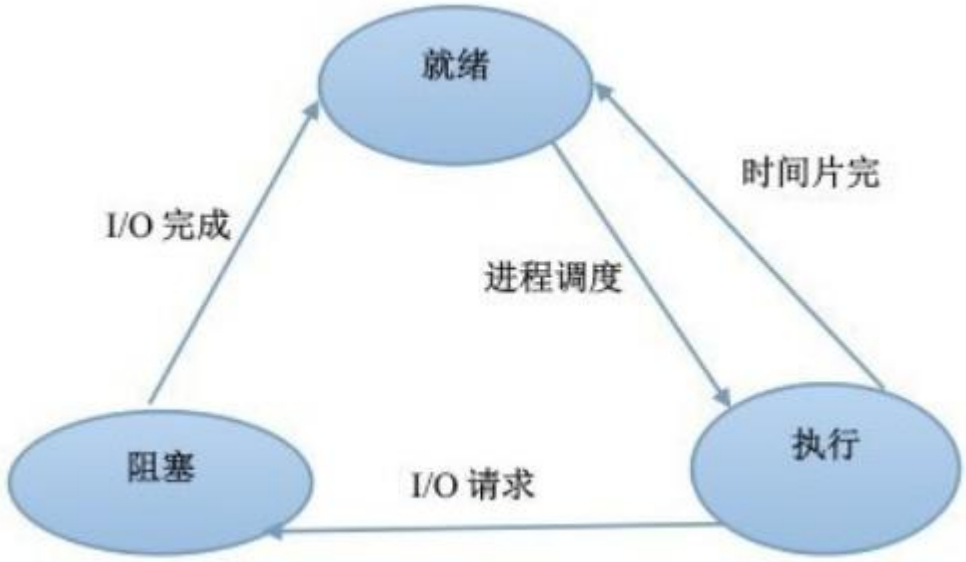
**5、进程中有哪几种状态。**

就绪状态：进程已获得除处理机以外的所需资源，等待分配处理机资源。

运行状态：占用处理机资源运行，处于此状态的进程数小于等于 CPU 数目。

阻塞状态：进程等待某种条件，在条件满足之前无法执行。

状态图如下图所示。




**6、分页和分段有什么区别。**

段式存储管理是一种符合用户视角地内存分配管理方案。在段式存储管理中，将程序的地址空间划分为若干段（segment），比如代码段、数据段、堆栈段。这样每个进程都有一个二维地址空间，相互独立，互不干扰。段式管理的优点是：没有内碎片（因为段大小可变，改变段大小来消除内碎片）。但段换入换出时，会产生外碎片（比如 5k 的段换 4k 的段，会产生 1k 的外碎片）。

页式存储管理方案是一种用户视角内存与物理内存相分离的内存分离管理方案。在页式存储管理中，将程序的逻辑地址划分为固定大小的页（page），而屋里内存划分为同样大小的帧，程序加载时，可以将任意一页放入内存中的任意一个帧，这些帧不必连续，从而实现了离散分离。页式存储管理的优点是：没有外碎片（因为页的大小固定），但会产生内碎片（一个页可能填充不满）。

两者的不同如下：

- **目的不同：**段是信息的逻辑单位，它是根据用户的需求划分的，因此段是对用户可见的；页是信息的物理单位，是为了管理主存的方便而划分的，对用户是透明的。
- **大小不同：**段的大小不固定，有它所完成的功能决定；页的大小是固定的，由系统决定。
- **地址空间不同：**段向用户提供二维地址空间；页向用户提供的是一维地址空间。
- **信息共享：**段是信息的逻辑单位，便于存储保护和信息的共享，页的保护和共享收到限制。
- **内存碎片：**页式存储管理的优点是没有外碎片，但是会产生内碎片。而段式管理的优点是没有内碎片，但会产生外碎片。

在分页系统中，允许将进程的每一页离散地存储在内存的任一物理块中，为了能在内存中找到每个页面对应的物理块，系统为每个进程建立了一张页面映射表，简称页表。页表的作用就是实现从页号到物理块号的地址映射。

**7、操作系统中进程调度策略有哪几种。**

操作系统中进程调度策略主要包括 FCFS（先来先服务）、优先级、时间片轮流、多级反馈等。具体分为以下几种，不同环境的调度算法目标不同，因此需要针对不同环境来讨论调度算法。

首先讨论**批处理系统**。批处理系统没有太多的用户操作，在该系统中，调度算法目标是保证吞吐量和周转时间（从提交到终止的时间）。

**先来先服务 first-come first-serverd（FCFS）：**非抢占式，FCFS 是一种最简单的调度算法，该算法即可用于作业调度，也可用于进程调度。当在作业调度中采用该算法时，每次调度都是从后备作业队列中选择一个或多个最先进入该队列的作业，将它们调入内存，为它们分配资源、创建线程，然后放入就绪队列。在进程调度中采用 FCFS 算法时，则每次调度是从就绪队列中选择一个最先进入该队列的进程，为之分配处理机，使之投入运行。该进程一直运行到完成或发生某事件而阻塞后才放弃处理机。

**短作业优先 shortest job first（SJF）：**非抢占式，是指对短作业或短进程优先调度的算法。它们可以分别用于作业调度和进程调度。它们可以分别用于作业调度和进程调度。短作业优先的调度算法是从后备队列中选择一个或若干个估计运行时间最短的作业，将它们调入内存运行。而短进程优先调度算法则是从就绪队列中选出一个估计运行时间最短的进程，将处理机分配给它，使它立即执行并一致执行到完成，或发生事件而被阻塞放弃处理机时再重新调度。

**最短剩余时间优先 shortest remaining time next（SRTN）：**最短作业优先的抢占式版本，按剩余运行时间的顺序进行调度。 当一个新的作业到达时，其

整个运行时间与当前进程的剩余时间作比较。如果新的进程需要的时间更少，则挂起当前进程，运行新的进程。否则新的进程等待。

然后讨论**交互式系统**，交互式系统有大量的用户交互操作，在该系统中调度算法的目标就是快速地进行响应。

**时间片轮转算法：**将所有就绪进程按照 FCFS 地原则排成一个队列，每次调度时，把 CPU 时间分配给队首进程，该进程可以执行一个时间片。当时间片用完时，由计时器发出时钟中断，调度程序边停止该进程的执行，并将它送往就绪队列的末尾，同时继续把 CPU 时间分配给队首的进程。时间片轮转算法的效率和时间片的大小有很大的关系。时间片过小，会导致进程切换太频繁。如果时间片过长，那么实时性就不能得到保证。

**优先级调度：**为每个进程分配一个优先级，按优先级进行调度。为了防止低优先级的进程永远得不到调度，可以随着时间的推移增加等待线程的优先级。

**多级反馈队列：**一个进程需要执行 100 个时间片，如果采用时间片轮转调度算法，那么需要交换 100 次。多级队列是为这种需要连续执行多个时间片的进程考虑，它设置了多个队列，每个队列时间片大小都不同，例如 1，2，4，8....。进程在第一个队列没执行完，就会转移到下一个队列。这种方式下，之前的进行就只需要交换 7 次。每个丢列优先权也不同，最上面的优先权最高。因此只有上一个队列没有进程在排休，才能调度当前队列上的进程。

**8、死锁的必要条件和处理方法。**

死锁的概念，在两个或者多个并发进程中，如果每个进程持有某个资源而又都等待别的进程释放它或他们现在保持的资源，在未改变这种状态之前都不能向前推进，称这一组进程产生了死锁。通俗地讲，死锁就是两个或多个进程被无限期地阻塞、相互等待的一种状态。

死锁的必要条件有四个。

- **互斥：**每个资源要么已经分配给一个进程，要么就是可用的。
- **占有和等待：**已经得到了某个资源的进程可以再请求新的资源。
- **不可抢占：**已经分配给一个进程的资源不能强制性地被抢占，它只能被占有它的进程显示地释放。
- **环路等待：**有两个或者两个以上的进程组成一条环路，该环路中的每个进程都在等待下一个进程所占有的资源。

主要有以下四种处理方法：

- **鸵鸟策略：**不才与任何措施，假装没有发生。
- **死锁检测与死锁恢复：**不试图阻止死锁，而是当检测到死锁发生时，采取措施进行恢复。每种类型一个资源的死锁检测算法是通过有向图是否存在环来实现，从一个节点出发进行深度优先搜索，对访问过的节点

进行标记，如果访问了已经标记的节点，就表示有向图存在环，也就是检测到了死锁的发生。还有一种是每种资源多个资源的死锁检测。死锁也可以通过抢占、回滚、杀死进程等方式恢复。

- **死锁预防：**在程序运行之前预防发生死锁。有多种方式，比如破坏互斥条件、破坏占有和等待条件、破坏不可抢占条件和破坏环路等待等。
- **死锁避免：**基本思想是动态地检测资源分配状态，以确保循环等待条件不成立，从而确保系统处于安全状态。所谓安全状态是指：如果系统能按某个顺序为每个进程分配资源（不超过其最大值），那么系统状态是安全的，换句话说，如果存在一个安全序列，那么系统就处于安全状态。资源分配图算法和银行家算法是两种经典地死锁避免的算法，其可以确保系统始终处于安全状态。

**9、Linux 中的文件描述符与打开文件之间的关系。**

在 Linux 系统中一切皆可以看成是文件，文件又可分为：普通文件、目录文件、链接文件和设备文件。文件描述符是内核为了高效管理已被打开的的文件所创建的索引，其是一个非负整数（通常是小整数），用于指代被打开的文件，所有执行 I/O 操作的系统调用都是通过文件描述符。程序刚刚启动的时候，0 是标准输入，1 是标准输出，2 是标准错误。如果此时去打开一个新的文件，它的文件描述符会是 3。POSIX 标准要求每次打开文件时（含 socket）必须使用当前进程中最小可用的文件描述符号码，因此，在网络通信过程中稍不注意就有可能造成串话。

文件描述符是有限制的。主要因为文件描述符是系统的一个重要资源，虽然说系统内存有多少就可以打开多少的文件描述符，但是在实际过程中内核是会做相应的处理的，一般是最大文件数会是系统内存的 10%（以 KB 来计算），这个称之为系统级限制。与此同时，内核为了不让某一个进程消耗掉所有的文件资源，其也会对单个进程最大打开文件数做默认值处理，一般默认值为 1024，这个称之为用户级限制。

此外，内核也会对所有打开的文件维护有一个系统级的描述符表格，成为打开文件表，并将表格中各条目称为打开文件句柄。

由于进程级文件描述符表的存在，不同的进程中会出现相同的我呢见描述符，它们可以指向同一个文件，也可能指向不同的文件。两个不同的文件描述符，若指向同一个打开文件句柄，将共享同一个文件偏移量。因此，如果通过其中一个文件描述符来修改文件偏移量（调用 read()、write()或 lseek()所致），那么从另一个描述符中也会观察到变化，无论这两个文件描述符是否属于不同进程，还是同一个进程，情况都是如此。

**10、进程间同步与互斥的区别。**

互斥：指某一个资源同时只允许一个访问者对其进行访问，具有唯一性和排他性。但是互斥无法限制访问者对资源的访问顺序，即访问是无序的。

同步：是指在互斥的基础上（大多数情况下），通过其它机制实现访问者对资源的有序访问。大多数情况下，同步已经实现了互斥，特别是所有写入资源的情况必定是互斥的。少数情况是指可以允许多个访问者同时访问资源。

同步体现的是一种协作性，互斥体现的是排他性。

**11、为什么要引入虚拟内存。**

为了更加有效地管理内存并且少出错，操作系统推出了虚拟内存（VM）。虚拟内存是硬件异常、硬件地址翻译、主存、磁盘文件和内核软件的完美交互，它为每个进程提供了一个大的、一致的和私有的地址空间。

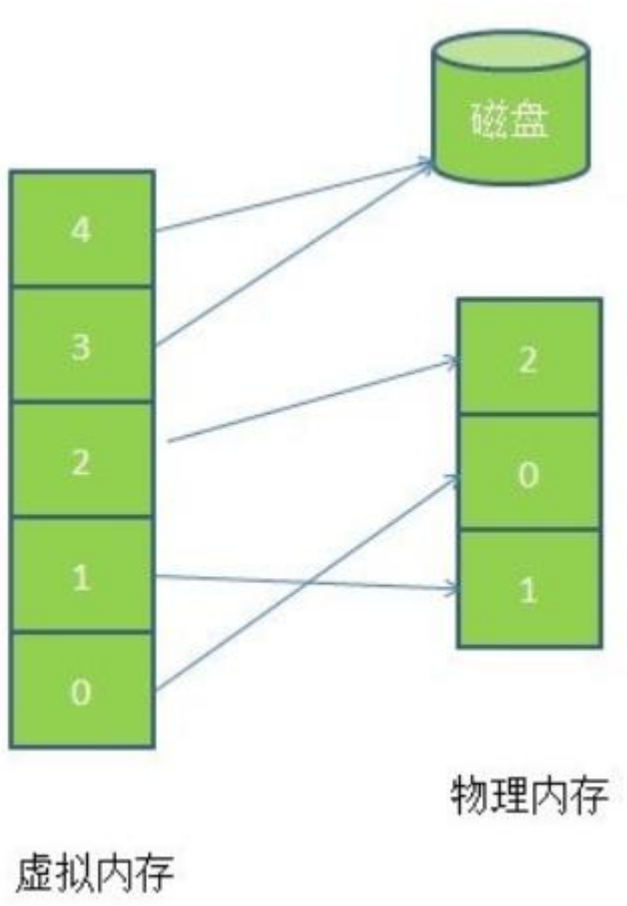
虚拟内存提供了三个重要的能力：

- 它将主存看成是一个存储在磁盘上的地址空间的高速缓存，在主存中只保存活动区域，并根据需要在磁盘和主存之间来回传送数据，通过这种方式，它高效地使用了主存。
- 它为每个进程提供了一致的地址空间，从而简化了内存管理。
- 它保护了每个进程的地址空间不被其他进程破坏。

虚拟内存的大小有两点限制条件。

- 虚拟内存<=内存+外存容量之和。
- 虚拟内存<=计算机地址位数所能容纳的（2^计算机位数）。

虚拟内存允许执行进程不必完全在内存中。虚拟内存的基本思想是：每个进程拥有独立的地址空间，这个空间被分为大小相等的多个块，称为页（page），每个页都是一段连续的地址。这些页被映射到物理内存（页表），但并不是所有的页都必须在内存中才能运行程序。当程序引用到一部分在物理内存中的地址空间时，由硬件立刻进行必要的映射；当程序引用到一部分不在物理内存中的地址空间时，由操作系统负责将缺失的部分装入物理内存并重新执行失败的命令。这样，**对于进程而言，逻辑上似乎有很大的内存空间，实际上其中一部分对应物理内存上的一块（称为帧，通常页和帧大小相等），还有一些没有加载在内存中的对应在硬盘上。**如下图所示。




从上图可知，虚拟内存实际比物理内存要大，当访问虚拟内存时，会访问 MMU（内存管理单元）去匹配对应的物理地址（比如上图的 0，1，2）。如果虚拟内存的页并不存在于物理内存中（如上图的 3，4），会产生缺页中断，从硬盘中取得缺的页放入内存，如果内存已满，还会根据某种算法将硬盘中的页换出来。

**12、页面置换算法。**

**FIFO 先进先出算法：**在操作系统中经常被用到，比如作业调度。

**LRU 最近最少使用算法：**根据使用时间到现在的长短来判断。

**LFU 最少使用次数算法：**根据使用次数来判断。

**OPT 最优置换算法：**理论的最优，当然，这是理论情况。就是要保证置换出去的是不再被使用的页，或者是在实际内存中最晚使用的算法。

**13、颠簸。**

颠簸本质上是指**频繁的页调度行为**，具体来讲，进程发生缺页中断，这时，必须置换某一页。然而，其他所有的页都在使用，它置换一个页，但又立即再次需要这个页。因此，会不断产生缺页中断，导致整个系统的效率急剧下降，这种现象称之为颠簸（抖动）。

内存颠簸的解决策略包括：

- 如果是因为页面替换策略失误，可以修改替换算法来解决这个问题。
- 因为是运行的程序太多，造成程序无法同时将所有频繁访问的页面调入内存，则要降低程序的数量。
- 否则，还剩下两个办法：终止该进程或增加物理内存容量。

**14、fork()函数。**

在 Linux 系统中，创建子线程的方法是使用系统调用 fork()函数。fork()函数用于从一个已存在的进程内创建一个新的进程，新的进程称为“子进程”，相应地称创建子进程的进程为“父进程”。使用 fork()函数得到的子进程是父进程的复制品，子进程完全复制了父进程的资源，包括进程上下文，代码区、数据区、堆区、栈区、内存信息、打开文件的文件描述符、信号处理函数、进程优先级、进程组号、当前工作目录、根目录等信息。而子进程与父进程的区别有进程号、资源使用情况和计时器等。

由于复制父进程的资源需要大量的操作，十分浪费时间与系统资源，因此 Linux 内核采取了写时拷贝技术（只有进程空间的各段内容要发生变化时，才会将父进程的内容复制给子进程）来提高效率。

由于子进程几乎对父进程完全复制，因此父子进程会同时运行同一个程序。因此我们需要某种方式来区分父子进程。区分父子进程常见的方法是查看 fork()函数的返回值或者区分父子进程的 PID。

父子进程的运行先后顺序是完全随机的（取决于系统的调度）。

多线程与 fork()的协作性很差。fork()一般不会在多线程程序中调用，因为 **Linux 的 fork()只克隆当前线程的 thread of control，不克隆其他线程。fork()之后，除了当前线程之外，其他线程都消失了。**也就是不能一下子 fork()出一个和父进程一样的多线程子进程。