Linux（1）

- 用户空间与内核空间
  - 现在操作系统都是采用虚拟存储器，那么对32位操作系统而言，它的寻址空间（虚拟存储空间）为4G（2的32次方）。操作系统的核心是内核，独立于普通的应用程序，可以访问受保护的内存空间，也有访问底层硬件设备的所有权限。为了保证用户进程不能直接操作内核（kernel），保证内核的安全，操作系统将虚拟空间划分为两部分，一部分为内核空间，一部分为用户空间。针对Linux操作系统而言，将最高的1G字节（从虚拟地址0xC0000000到0xFFFFFFFF），供内核使用，称为内核空间，而将较低的3G字节（从虚拟地址0x00000000到0xBFFFFFFF），供各个进程使用，称为用户空间。
- 进程切换
  - 为了控制进程的执行，内核必须有能力挂起正在CPU上运行的进程，并恢复以前挂起的某个进程的执行。这种行为被称为进程切换。因此可以说，任何进程都是在操作系统内核的支持下运行的，是与内核紧密相关的。
  - 从一个进程的运行转到另一个进程上运行，这个过程中经过下面这些变化：
  - 保存处理机上下文，包括程序计数器和其他寄存器。
  - 更新PCB信息。
  - 把进程的PCB移入相应的队列，如就绪、在某事件阻塞等队列。
  - 选择另一个进程执行，并更新其PCB。
  - 更新内存管理的数据结构。
  - 恢复处理机上下文。
- 进程的阻塞
  - 正在执行的进程，由于期待的某些事件未发生，如请求系统资源失败、等待某种操作的完成、新数据尚未到达或无新工作做等，则由系统自动执行阻塞原语(Block)，使自己由运行状态变为阻塞状态。可见，进程的阻塞是进程自身的一种主动行为，也因此只有处于运行态的进程（获得CPU），才可能将其转为阻塞状态。当进程进入阻塞状态，是不占用CPU资源的。
- 文件描述符 fd
  - 文件描述符（File descriptor）是计算机科学中的一个术语，是一个用于表述指向文件的引用的抽象化概念。
  - 文件描述符在形式上是一个非负整数。实际上，它是一个索引值，指向内核为每一个进程所维护的该进程打开文件的记录表。当程序打开一个现有文件或者创建一个新文件时，内核向进程返回一个文件描述符。在程序设计中，一些涉及底层的程序编写往往会围绕着文件描述符展开。但是文件描述符这一概念往往只适用于UNIX、Linux这样的操作系统。
- 缓存 IO
  - 缓存 IO 又被称作标准 IO，大多数文件系统的默认 IO 操作都是缓存 IO。在 Linux 的缓存 IO 机制中，操作系统会将 IO 的数据缓存在文件系统的页缓存（ page cache ）中，也就是说，数据会先被拷贝到操作系统内核的缓冲区中，然后才会从操作系统内核的缓冲区拷贝到应用程序的地址空间。
    - 缓存 IO 的缺点：
      - 数据在传输过程中需要在应用程序地址空间和内核进行多次数据拷贝操作，这些数据拷贝操作所带来的 CPU 以及内存开销是非常大的。

- Linux IO模型
  - 网络IO的本质是socket的读取,socket在linux系统中被抽象为流,IO可以理解为对流的操作.对于一次IO访问,数据会先被拷到操作系统内核的缓冲区中,然后才会从操作系统内核的缓冲区拷贝到应用程序的地址空间,所以说当一个read操作发生时,它会经理两个阶段:
    - 第一阶段：等待数据准备 (Waiting for the data to be ready)。
    - 第二阶段：将数据从内核拷贝到进程中 (Copying the data from the kernel to the p
    - 对socket流而言:
      - 第一步：通常涉及等待网络上的数据分组到达，然后被复制到内核的某个缓冲区。
      - 第二步：把数据从内核缓冲区复制到应用进程缓冲区。
- 网络IO的模型大致有如下几种：
  - 同步阻塞模型需要在 IO 操作开始时阻塞应用程序。这意味着不可能同时重叠进行处理和 IO 操作。
  - 同步非阻塞模型允许处理和 IO 操作重叠进行，但是这需要应用程序根据重现的规则来检查 IO 操作的状态。
  - 这样就剩下异步非阻塞 IO 了，它允许处理和 IO 操作重叠进行，包括 IO 操作完成的通知。
  - 异步IO（asynchronous IO）
    - 如果这个进程正在用户态忙着做别的事（例如在计算两个矩阵的乘积），那就强行打断之，调用事先注册的信号处理函数，这个函数可以决定何时以及如何处理这个异步任务。由于信号处理函数是突然闯进来的，因此跟中断处理程序一样，有很多事情是不能做的，因此保险起见，一般是把事件 “登记” 一下放进队列，然后返回该进程原来在做的事。
    - 如果这个进程正在内核态忙着做别的事，例如以同步阻塞方式读写磁盘，那就只好把这个通知挂起来了，等到内核态的事情忙完了，快要回到用户态的时候，再触发信号通知。
    - 如果这个进程现在被挂起了，例如无事可做 sleep 了，那就把这个进程唤醒，下次有 CPU 空闲的时候，就会调度到这个进程，触发信号通知。
    -
  - 同步模型（synchronous IO）
    - 阻塞IO（bloking IO）
      - 我和女友点完餐后，不知道什么时候能做好，只好坐在餐厅里面等，直到做好，然后吃完才离开。女友本想还和我一起逛街的，但是不知道饭能什么时候做好，只好和我一起在餐厅等，而不能去逛街，直到吃完饭才能去逛街，中间等待做饭的时间浪费掉了。这就是典型的阻塞。
      - 当用户进程调用了recv()/recvfrom()这个系统调用，kernel就开始了IO的第一个阶段：准备数据（对于网络IO来说，很多时候数据在一开始还没有到达。比如，还没有收到一个完整的UDP包。这个时候kernel就要等待足够的数据到来）。这个过程需要等待，也就是说数据被拷贝到操作系统内核的缓冲区中是需要一个过程的。而在用户进程这边，整个进程会被阻塞（当然，是进程自己选择的阻塞）。第二个阶段：当kernel一直等到数据准备好了，它就会将数据从kernel中拷贝到用户内存，然后kernel返回结果，用户进程才解除block的状态，重新运行起来。

    - 所以，同步阻塞(blocking IO)的特点就是在IO执行的两个阶段都被block了。
- 非阻塞IO（non-blocking IO）
    - 我女友不甘心白白在这等，又想去逛商场，又担心饭好了。所以我们逛一会，回来询问服务员饭好了没有，来来回回好多次，饭都还没吃都快累死了啦。这就是非阻塞。需要不断的询问，是否准备好了。
    - 非阻塞的recvform系统调用调用之后，进程并没有被阻塞，内核马上返回给进程，如果数据还没准备好，此时会返回一个error。进程在返回之后，可以干点别的事情，然后再发起recvform系统调用。重复上面的过程，循环往复的进行recvform系统调用。这个过程通常被称之为轮询。轮询检查内核数据，直到数据准备好，再拷贝数据到进程，进行数据处理。需要注意，拷贝数据整个过程，进程仍然是属于阻塞的状态。
    - 当用户进程发出read操作时，如果kernel中的数据还没有准备好，那么它并不会block用户进程，而是立刻返回一个error。从用户进程角度讲，它发起一个read操作后，并不需要等待，而是马上就得到了一个结果。用户进程判断结果是一个error时，它就知道数据还没有准备好，于是它可以再次发送read操作。一旦kernel中的数据准备好了，并且又再次收到了用户进程的system call，那么它马上就将数据拷贝到了用户内存，然后返回。
    - 所以，nonblocking IO的特点是用户进程需要不断的主动询问kernel数据好了没有。
- 优点：能够在等待任务完成的时间里干其他活了（包括提交其他任务，也就是“后台” 可以有多个任务在同时执行）。
- 缺点：任务完成的响应延迟增大了，因为每过一段时间才去轮询一次read操作，而任务可能在两次轮询之间的任意时间完成。这会导致整体数据吞吐量的降低。
- 多路复用IO（multiplexing IO）
    - 与第二个方案差不多，餐厅安装了电子屏幕用来显示点餐的状态，这样我和女友逛街一会，回来就不用去询问服务员了，直接看电子屏幕就可以了。这样每个人的餐是否好了，都直接看电子屏幕就可以了，这就是典型的IO多路复用。
    - 由于同步非阻塞方式需要不断主动轮询，轮询占据了很大一部分过程，轮询会消耗大量的CPU时间，而 “后台” 可能有多个任务在同时进行，人们就想到了循环查询多个任务的完成状态，只要有任何一个任务完成，就去处理它。如果轮询不是进程的用户态，而是有人帮忙就好了。那么这就是所谓的“IO 多路复用”。UNIX/Linux 下的 select、poll、epoll 就是干这个的（epoll 比 poll、select 效率高，做的事情是一样的）。
    - IO多路复用有两个特别的系统调用select、poll、epoll函数。select调用是内核级别的，select轮询相对非阻塞的轮询的区别在于—前者可以等待多个socket，能实现同时对多个IO端口进行监听，当其中任何一个socket的数据准好了，就能返回进行可读，然后进程再进行recvform系统调用，将数据由内核拷贝到用户进程，当然这个过程是阻塞的。select或poll调用之后，会阻塞进程，与同步阻塞(blocking IO)不同在于，此时的select不是等到socket数据全部到达再处理, 而是有了一部分数据就会调用用户进程来处理。如何知道有一部分数据到达了呢？监视的事情交给了内核，内核负责数据到达的处理。也可以理解为”非阻塞”吧。
    - I/O复用模型会用到select、poll、epoll函数，这几个函数也会使进程阻塞，但是和阻塞I/O所不同的的，这两个函数可以同时阻塞多个I/O操作。而且可以同时对多个读操作，多个写操作的I/O函数进行检测，直到有数据可读或可写时（注意不是全部数据可读或可写），才真正调用I/O操作函数。

        - 上面的图和blocking IO的图其实并没有太大的不同，事实上，还更差一些。因为这里需要使用两个system call (select 和 recvfrom)，而blocking IO只调用了一个system call (recvfrom)。但是，用select的优势在于它可以同时处理多个connection。
        - 所以，如果处理的连接数不是很高的话，使用select/epoll的web server不一定比使用multi-threading + blocking IO的web server性能更好，可能延迟还更大。（select/epoll的优势并不是对于单个连接能处理得更快，而是在于能处理更多的连接。）
        - 在I/O编程过程中，当需要同时处理多个客户端接入请求时，可以利用多线程或者I/O多路复用技术进行处理。I/O多路复用技术通过把多个I/O的阻塞复用到同一个select的阻塞上，从而使得系统在单线程的情况下可以同时处理多个客户端请求。与传统的多线程/多进程模型比，I/O多路复用的最大优势是系统开销小，系统不需要创建新的额外进程或者线程，也不需要维护这些进程和线程的运行，降底了系统的维护工作量，节省了系统资源，I/O多路复用的主要应用场景如下：
        - //服务器需要同时处理多个处于监听状态或者多个连接状态的套接字。 //服务器需要同时处理多种网络协议的套接字
        - 此处仍然不太清楚的，强烈建议大家在细究《聊聊同步、异步、阻塞与非阻塞》中讲同步与异步的根本性区别，同步是需要主动等待消息通知，而异步则是被动接收消息通知，通过回调、通知、状态等方式来被动获取消息。IO多路复用在阻塞到select阶段时，用户进程是主动等待并调用select函数获取数据就绪状态消息，并且其进程状态为阻塞。所以，把IO多路复用归为同步阻塞模式。
    - 信号驱动式IO（signal-driven IO）
-
- sort命令
    - -b：忽略每行前面开始出的空格字符；
    - -c：检查文件是否已经按照顺序排序；
    - -d：排序时，处理英文字母、数字及空格字符外，忽略其他的字符；
    - -f：排序时，将小写字母视为大写字母；
    - -i：排序时，除了040至176之间的ASCII字符外，忽略其他的字符；
    - -m：将几个排序号的文件进行合并；
    - -M：将前面3个字母依照月份的缩写进行排序；
    - -n：依照数值的大小排序；
    - -o<输出文件>：将排序后的结果存入制定的文件；
    - -r：以相反的顺序来排序；
    - -t<分隔字符>：指定排序时所用的栏位分隔字符；
    - +<起始栏位>-<结束栏位>：以指定的栏位来排序，范围由起始栏位到结束栏位的前一栏位
- awk命令
    - awk [options] 'script' var=value file(s)
    - awk [options] -f scriptfile var=value file(s)
    - 模式可以是以下任意一个：
    - /正则表达式/：使用通配符的扩展集。
    - 关系表达式：使用运算符进行操作，可以是字符串或数字的比较测试。
    - 模式匹配表达式：用运算符~（匹配）和~!（不匹配）。

    - BEGIN语句块、pattern语句块、END语句块
- Linux的I/O模型介绍以及同步异步阻塞非阻塞的区别（超级重要）
    -
- 文件系统的理解（EXT4，XFS，BTRFS）
- 文件处理grep,awk,sed这三个命令必知必会
- IO复用的三种方法（select,poll,epoll）深入理解，包括三者区别，内部原理实现?
- Epoll的ET模式和LT模式（ET的非阻塞）
- 查询进程占用CPU的命令（注意要了解到used，buf，cache代表意义）
- linux的其他常见命令（kill，find，cp等等）
- shell脚本用法
- 硬连接和软连接的区别
    - 硬连接指通过索引节点来进行连接。在Linux的文件系统中，保存在磁盘分区中的文件不管是什么类型都给它分配一个编号，称为索引节点号(Inode Index)。在Linux中，多个文件名指向同一索引节点是存在的。比如：A是B的硬链接（A和B都是文件名），则A的目录项中的inode节点号与B的目录项中的inode节点号相同，即一个inode节点对应两个不同的文件名，两个文件名指向同一个文件，A和B对文件系统来说是完全平等的。删除其中任何一个都不会影响另外一个的访问。
    - 另外一种连接称之为符号连接（Symbolic Link），也叫软连接。软链接文件有类似于Windows的快捷方式。它实际上是一个特殊的文件。在符号连接中，文件实际上是一个文本文件，其中包含的有另一文件的位置信息。比如：A是B的软链接（A和B都是文件名），A的目录项中的inode节点号与B的目录项中的inode节点号不相同，A和B指向的是两个不同的inode，继而指向两块不同的数据块。但是A的数据块中存放的只是B的路径名（可以根据这个找到B的目录项）。A和B之间是“主从”关系，如果B被删除了，A仍然存在（因为两个是不同的文件），但指向的是一个无效的链接。
- 文件权限怎么看（rwx）
- 文件的三种时间（mtime, atime，ctime），分别在什么时候会改变
- Linux监控网络带宽的命令，查看特定进程的占用网络资源情况命令