Linux系统
权限
用户
组
目录、文件
特殊权限：SUID，SGID，SBIT
设备
种类
字符设备(c)
按顺序访问，比如键盘
对字符设备发出读/写请求时，实际的硬件I/O一般就紧接着发生了，字符设备可以直接物理磁盘读取，不经过系统缓存
块设备(b)
无序访问
网络设备
各个目录
/bin目录
/etc目录
/lib目录
/var
...
文件系统
文件描述符与文件内容是分开存放的。通常我们会通过文件描述符来操作文件
capabilities
将之前与超级用户 root（UID=0）关联的特权细分为不同的功能组。使得普通用户可以做部分只要root才能做的工作。
Capabilites 作为线程（Linux 并不真正区分进程和线程）的属性存在，每个功能组都可以独立启用和禁用
将内核调用分门别类，具有相似功能的内核调用被分到同一组中
这样一来，权限检查的过程就变成了：在执行特权操作时，如果线程的有效身份不是 root，就去检查其是否具有该特权操作所对应的 capabilities，并以此为依据，决定是否可以执行特权操作。
pam
可插拔的鉴权模块。它通过提供一些动态连接库与一套统一的API，将系统提供的服务和该服务的认证方式分开，使得系统管理员可以灵活的根据需要给不同的服务配置不同的认证方式，而无需更改服务，同时也便于向系统添加新的认证。
用户在执行相应的命令时，会先使用pam模块进行权限校验
所有可用的pam模块在/etc/pam.d目录下
CPU
cpu队列
cpu维护着两个队列
活动队列
子主题
等待队列
执行完的进程在此队列。并且，内核设计者创建了一个叫做空闲任务的进程。在Linux 下就是第 0号进程。
当其它进程都处于不可运行状态时，调度器就从队列中取出空闲进程运行，显然，空闲进程永远处于就绪状态，且优先级最低
cpu管道
cpu亲和性
CPU亲和性是通过中断实现的，大致就是如果这个进程调度到其他CPU则触发中断，直到调度到指定CPU上才会执行
cpu亲和性是指有些线程想绑定在某一个核上面执行。这样减小了切换的消耗
CPU时间片
CPU维护着两个队列，一个是活动队列，一个是过期队列
当一个进程被挂起时，有两种情况
一种是它的时间片还没用完，此时回到活动队列等待再次唤醒
另一种是时间片用完了，这时它回导过期队列（并且CPU会给它重新分配时间片）。
当活动队列都空了的时候，可以直接将过期队列与活动队列调换，因为过期队列中的进程的时间片早就分配好了，所以可以直接工作
内存
存放运行的代码，全局变量，常量，栈，堆，调用的动态库
堆：地址从小到大，树形结构。栈：地址从大到小，栈底为最大地址
虚拟内存。我们使用的都是虚拟内存。
大小端。将高位放在低地址叫大端模式，反之叫小端模式
程序的某个函数使用小端还是大端，可以通过代码实现
计算机电路先处理低位字节，效率比较高，因为计算都是从低位开始的。所以，计算机的内部处理都是小端字节序
而在其他场景，比如网络传输和文件存储则使用大端序
子主题
swap
硬盘上交换分区的大小
cache
内存中读完缓存起来内容占的大小
buffer
内存中写完的东西缓存起来，这样快速响应请求，后面数据再定期刷到磁盘上
内存管理
在进程运行之前，不是装入全部页面，而是装入一个或零个页面，之后根据进程运行的需要，动态装入其他页面；当内存空间已满，而又需要装入新的页面时，则根据某种算法淘汰某个页面，以便装入新的页面。
FIFO，先进先出
总是选择最先加载到内存的一页调出
容易实现。把装入内存的那些页面的页号按进入的先后次序排好队列，每次总是调出队首的页。当系统维护一个所有当前在内存中的页面的链表，最老的页面在头上，最新的页面在表尾。当发生缺页时，淘汰表头的页面并把新调入的页面加到表尾
LRU，最近最少使用
置换未使用时间最长的页面
1. 新数据插入到链表头部；
2. 每当缓存命中（即缓存数据被访问），则将数据移到链表头部；
3. 当链表满的时候，将链表尾部的数据丢弃。
优点：当存在热点数据时，LRU的效率很好，因为最热数据都在表头
缺点：偶发性的、周期性的批量操作会导致LRU命中率急剧下降，缓存污染情况比较严重。
脏页
当经常被访问的数据缓存在内存中，则当这部分数据发生修改时，而磁盘内的数据并未发生修改，此时这部分数据就叫做脏页
内存交互
当内存不足时，页面也可以移出内存并放入磁盘中，这个过程叫内存交换
内存结构
用户态跟内核态中的使用的内存地址都是虚拟地址
内存结构从低到高分别是
存放二进制文件、存放初始化的静态变量、存放为初始化的静态变量、堆、文件映射区、栈
再往上就是内核空间
虚拟空间一切二，一部分用来放内核的东西，称为内核空间，一部分用来放进程的东西，称为用户空
用户空间在低地址
子主题
内核
安全
cgroup
命令
grep
grep filename
-c，只输出匹配个数
-n，输出匹配行号跟内容
-v，输出不匹配的内容
sed
sed [参数] '匹配条件 动作' filename
sed '1a hello' xxx。在第一行追加hello
awk
子主题
打印第三行的第一列内容
awk 'NR==3{print $0}'
内核中的附加程序
进程
系统分配资源的最小单位
进程在内存中有三部分组成：数据段（放运行中产生的各种数据）、堆栈段（存放变量）和代码段（存放代码）
操作系统对进程的控制和管理通过PCB
PCB通常是系统内存占用区中的一个连续存区，它存放着操作系统用于描述进程情况及控制进程运行所需的全部信息
上下午切换
操作系统对把CPU控制权在不同进程之间交换执行的机制成为上下文切换
上下文切换主要干两件事情，一是切换进程空间，也即虚拟内存；二是切换寄存器和CPU上下文。
多进程
调用fork()创建子进程
进程状态
R，可执行与running
S，可终端休眠
D，不可中断休眠
kill -9也杀不掉
T，暂停状态
Z，退出状态，僵尸进程
子进程退出时，父进程回收子进程异常。此时只有杀死父进程或重启才能消灭僵尸进程
孤儿进程
父进程退出，但是子进程并没有退出。会被init进程收养
线程
系统可以调度的最小单位
同一进程中的多个线程将共享该进程中的全部系统资源，如虚拟地址空间，文件描述符和信号处理等等
但同一进程中的多个线程有各自的调用栈和线程本地存储
协程
不被操作系统内核所管理，而完全是由程序所控制
协程之间的切换不需要涉及任何系统调用或任何阻塞调用
协程只在一个线程中执行，并且一个线程中只有一个协程，所以协程中不存在同时写变量冲突。不需要加锁等操作
调度
子主题
select
工作模式
将已连接的 Socket 都放到一个文件描述符集合，然后调用 select 函数将文件描述符集合拷贝到内核里，让内核来检查是否有网络事件产生，检查的方式很粗暴，就是通过遍历文件描述符集合的方式，当检查到有事件产生后，将此 Socket 标记为可读或可写， 接着再把整个文件描述符集合拷贝回用户态里，然后用户态还需要再通过遍历的方法找到可读或可写的 Socket，然后再对其处理。
全量拷贝了两次，遍历了两次
epoll
产生的原因
因为linux一切皆文件。所以可以说linux一直大量的与fd进行交互
介绍
改进的poll。多路复用IO接口select/poll的增强版本
它无须遍历整个被侦听的描述符集，只要遍历那些被内核IO事件异步唤醒而加入Ready队列的描述符集合就行了
工作原理
子主题
为什么快?
多路复用IO。只使用一个进程来维护多个 Socket
1、功能分离
不需要像 select/poll 每次操作时都传入整个 socket 集合，只需要传入一个待检测的 socket，减少了内核和用户空间大量的数据拷贝和内存分配。
使用的是红黑树。是一种自平衡二叉查找树
2、就绪事件列表
内核维护一个“就绪事件列表”，引用收到数据的socket，避免遍历
当sock2和sock3收到数据时。中断程序会给eventpoll的“就绪事件列表”添加收到数据socket的引用。当进程被唤醒后，只要获取rdlist的内容，就能够知道哪些socket收到数据
就绪事件列表应是一种双向链表，能够快速插入和删除的数据。
我们熟悉的select/poll/epoll 内核提供给用户态的多路复用系统调用，进程可以通过一个系统调用函数从内核中获取多个事件
硬件
kernel
系统调用
shell
库函数
用户进程
kernel
系统调用子系统
进程管理子系统
内存管理子系统
文件管理子系统
网络管理子系统
设备管理子系统