<https://mp.weixin.qq.com/s/s8RxsKJrXuQlRsEQB2GYgw>

# 文件句柄？文件描述符？傻傻分不清楚

## 概述

在实际工作中会经常遇到一些bug，有些就需要用到文件句柄，文件描述符等概念，比如报错： too many open files, 如果你对相关知识一无所知，那么debug起来将会异常痛苦。在linux操作系统中，文件句柄（包括Socket句柄）、打开文件、文件指针、文件描述符的概念比较绕，而且windows的文件句柄又与此有何关联和区别？这一系列的问题是我们不得不面对的。

笔者通过翻阅相关资料，并采用了一些demo来验证相关观点。如果文中有理解偏差，欢迎指正。

这里先笼统的将一下笔者对上面的问题的一些理解：

句柄，熟悉Windows编程的人知道，**句柄是Windows用来标识被应用程序所建立或使用的对象的唯一整数**，windows使用各种各样的句柄标识诸如应用程序实例、窗口、控制、位图等。Windows的句柄有点像C语言中的文件句柄。更通俗的理解，句柄是一种指向指针的指针。

在linux系统中文件句柄（file handles）和文件描述符(file descriptor)是一个一一对应的关系（如果错误，欢迎指正），按照C语言的理解文件句柄是FILE\*（fopen()返回），而文件描述符是fd（int型，open()函数返回），FILE这个结构体中有一个字段是\_fileno，其就是指fd(文章末尾通过程序验证)，且FILE\*和fd可以通过C语言函数进行互相转换，故此笔者认为linux的文件句柄和文件描述符应该是一个一一对应的关系。文件指针即指FILE\*，即指文件句柄。打开文件（open files）包括文件句柄但不仅限于文件句柄，由于linux所有的事物都以文件的形式存在，要使用诸如共享内存、信号量、消息队列、内存映射等都会打开文件，但这些是不会占用文件句柄。

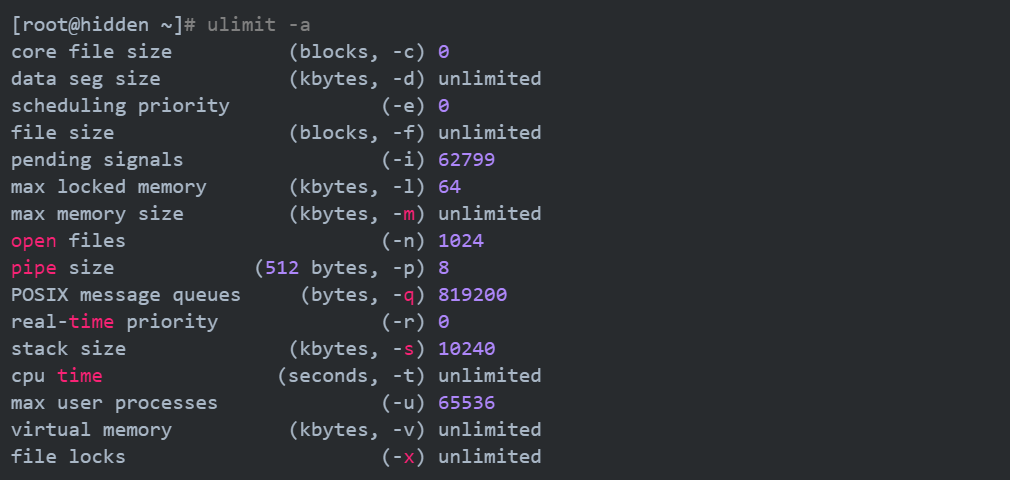
## ulimit

查看进程允许打开的最大文件句柄数：ulimit -n。设置进程能打开的最大文件句柄数：ulimit -n xxx。

ulimit在系统允许的情况下，提供对特定shell可利用的资源的控制（Provides control over the resources avaliable to the shell and to processes started by it, on systems that allow such control）。-H和-S选项设定指定资源的硬限制和软限制。硬限制设定之后不能再添加，而软限制则可以增加到硬限制规定的值。如果-H和-S选项都没有指定，则软限制和硬限制同时设定。限制值可以是指定资源的数值或者hard, soft, unlimited这些特殊值，其中hard代表当前硬限制, soft代表当前软件限制, unlimited代表不限制. 如果不指定限制值, 则打印指定资源的软限制值, 除非指定了-H选项.如果指定了不只一种资源, 则限制名和单位都会在限制值前显示出来.



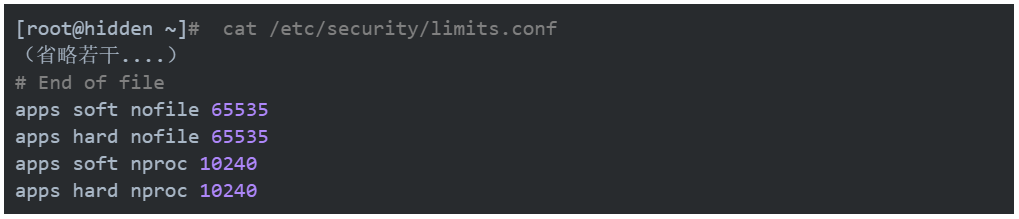
需要注意的是ulimit提供的是对特定shell可利用的资源的控制，而shell是与具体用户相关的。因此ulimit提供的是对单个用户的限制。包括以下项：



其中就有个“open files”的限制，默认是1024，也就是这个用户最大可以打开1024个文件。如果使用ulimit -n修改最大文件打开数，那么只对当前shell用户有用，同时也只对当前shell和这个shell fork出来的子shell生效，重启之后会重新恢复为默认值。

## limits.conf

limits.conf这个文件是在/etc/security/目录下，因此这个文件是出于安全考虑的。limits.conf文件是用于提供对系统中的用户所使用的资源进行控制和限制，对所有用户的资源设定限制是非常重要的，这可以防止用户发起针对处理器和内存数量等的拒绝服务攻击。这些限制必须在用户登录时限制。



其中含义如下：

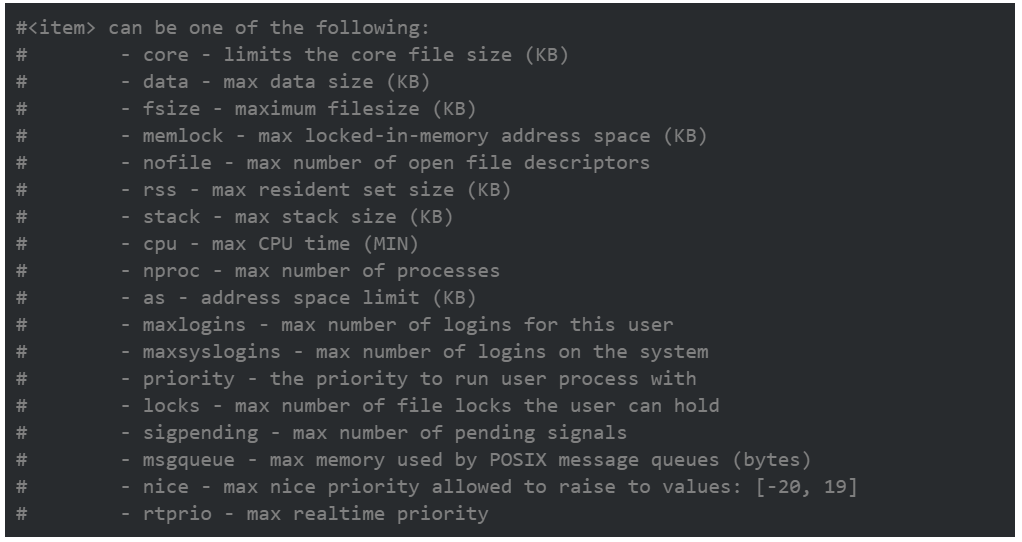
第一列表示域（domain）,可以使用用户名（root等），组名（以@开头）,通配置\*和%，%可以用于%group参数。

第二列表示类型(type),值可以是soft或者hard

第三列表示项目(item),值可以是core, data, fsize, memlock, nofile, rss, stack, cpu, nproc, as, maxlogins, maxsyslogins, priority, locks, msgqueue, nie, rtprio。其中nofile(Number of Open File）就是文件打开数。

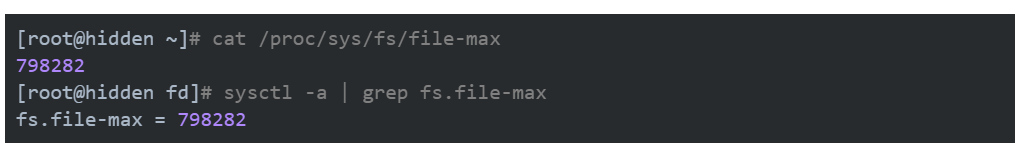
第四列表示值.

关于第三列的详细解释如下：

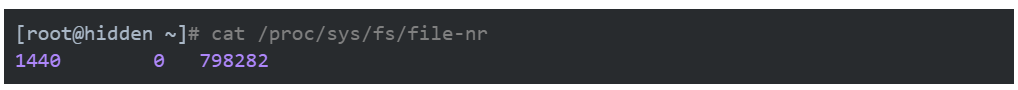


limits.conf与ulimit的区别在于前者是针对所有用户的，而且在任何shell都是生效的，即与shell无关，而后者只是针对特定用户的当前shell的设定。在修改最大文件打开数时，最好使用limits.conf文件来修改，通过这个文件，可以定义用户，资源类型，软硬限制等。也可修改/etc/profile文件加上ulimit的设置语句来是的全局生效。  
当达到上限时，会报错：too many open files或者遇上Socket/File: Cannot open so many files等。

## file-max & file-nr



该文件指定了可以分配的文件句柄的最大数目（系统全局的可用句柄数目. The value in file-max denotes the maximum number of file handles that the Linux kernel will allocate）。如果用户得到的错误消息诸如“由于打开文件数已经达到了最大值”之类，那么说明他们不能打开更多文件，则可能需要增加该值。可将这个值设置成任意多个文件，并且能通过将一个新数字值写入该文件来更改该值。这个参数的默认值和内存大小有关系，可以使用公式：file-max ≈ 内存大小/ 10k.



关于file-nr参数的解释如下：   
Historically, the three values in file-nr denoted the number of allocated file handles, the number of allocated but unused file handles, and the maximum number of file handles. Linux 2.6 always reports 0 as the number of free file handles – this is not an error, it just means that the number of allocated file handles exactly matches the number of used file handles.

这三个值分别指：系统已经分配出去的句柄数、已经分配但是还没有使用的句柄数以及系统最大的句柄数(和file-max一样)。



lsof是列出系统所占用的资源（list open files），但是这些资源不一定会占用句柄。比如共享内存、信号量、消息队列、内存映射等，虽然占用了这些资源，但不占用句柄。   
如果出了某些故障，使用lsof | wc -l的结果，这个时候可以通过file-nr粗略的估算一下。

查看硬盘信息：df -m   
查看内存信息：free -m   
查看CPU信息：cat /proc/cpuinfo   
查看内核所能打开的线程数：cat /proc/sys/kernel/threads-max

## 为什么有限制？

为什么Linux内核对文件句柄数、线程和进程的最大打开数进行了限制？以及如果我们把它调的太大，会产生什么样的后果？

原因1 - 资源问题：the operating system needs memory to manage each open file, and memory is a limited resource - especially on embedded systems.   
原因2 - 安全问题：if there were no limits, a userland software would be able to create files endlessly until the server goes down.

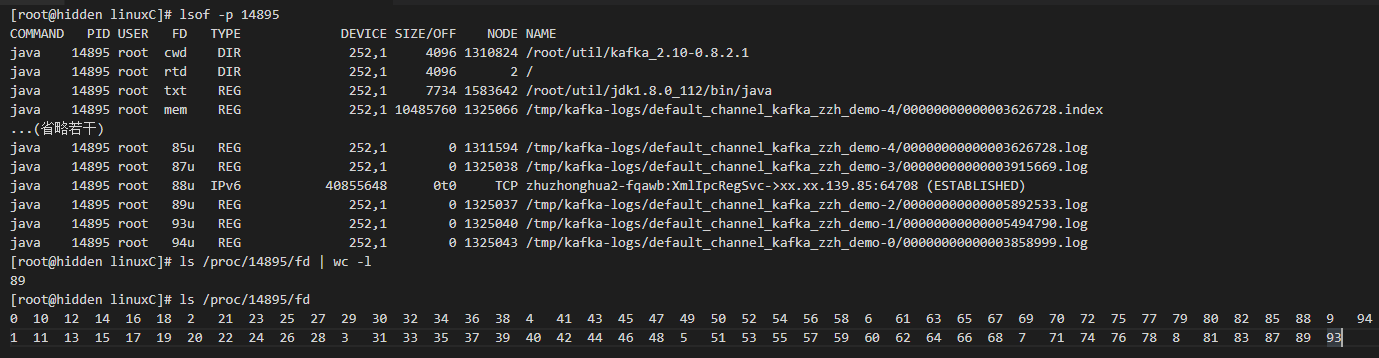
What’s more? If the file descriptors are tcp sockets, etc, then you risk using up a large amount for the socket buffers and other kernel objects, this memory is not going to be swappable.

最主要的是资源问题，为防止某一单一进程打开过多文件描述符而耗尽系统资源，对进程打开文件数做了限制。

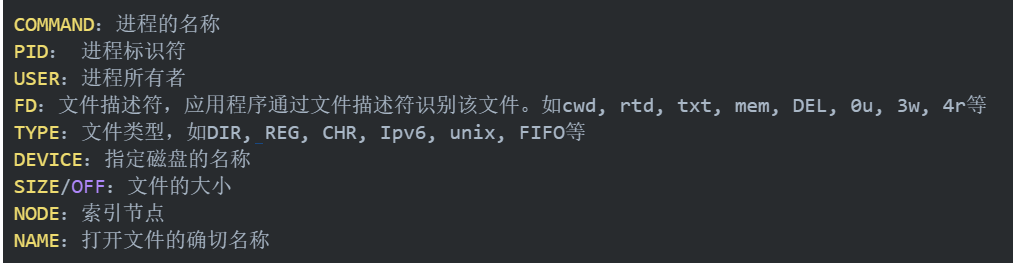
## lsof

lsof(list open files)是一个列出当前系统打开文件的工具。在linux环境下，任何事物都以文件的形式存在，通过文件不仅仅可以访问常规数据，还可以访问网络连接和硬件。所以如TCP和UDP等，系统在后台都为该应用程序分配了一个文件描述符，无论这个文件的本质如何，该文件描述符为应用程序与基础操作系统之间的交互提供了通用接口。因为应用程序打开文件的描述符列表提供了大量关于这个应用程序本身的信息，因此通过lsof工具能够查看这个列表对系统检测以及排错将是很有帮助的。

在终端下输入lsof即可显示系统打开的文件，因为lsof需要访问核心内存和各种文件，所以必须以root身份运行它才能够充分地发挥其功能。



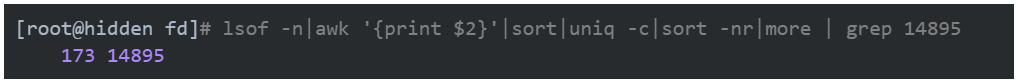
lsof输出各列信息的意义如下：



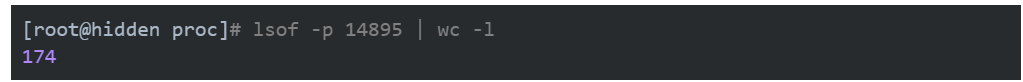
FD列中的文件描述符cwd表示应用程序的当前工作目录，这是该应用程序启动的目录，除非它本身对这个目录进行更改；txt类型的文件是程序代码，如应用程序二进制文件本身或共享库，如上列表中显示的、sbin/init程序；数值表示应用程序的文件描述符，这是打开该文件时返回的一个整数，如“lsof -p 14895”命令解析出来的最后一行的文件描述符为94，u表示该文件被打开处于读写模式，而不是只读r或只写w模式，同时还有大写的W表示该应用程序具有对整个文件的写锁。该文件描述符用于确保每次只能打开一个应用程序实例。初始打开每个应用程序时，都有三个文件描述符：0、1、2，分别表示标准输入、标准输出、错误流。所以大多数应用程序所打开的文件的FD都是从3开始的。

TYPE列比较直观。文件和目录分别为REG和DIR。而CHR和BLK分别表示字符和块设备。或者unix, FIFO, Ipv6分表表示UNIX域套接字，FIFO队列和IP套接字。

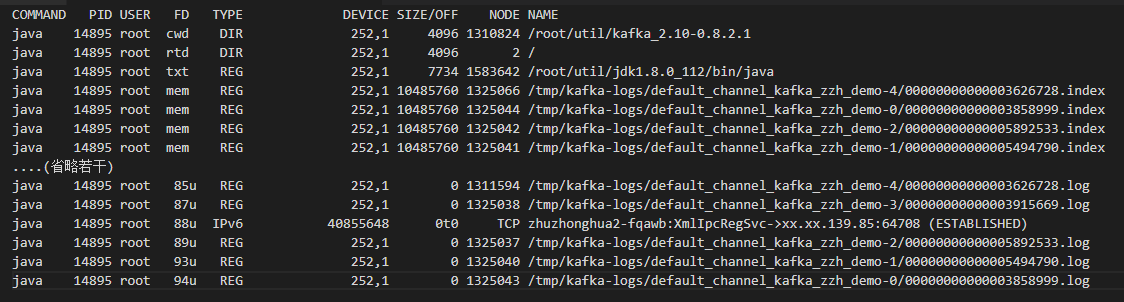
查看当前进程打开了多少文件：lsof -n|awk ‘{print $2}’|sort|uniq -c|sort -nr|more | grep [PID]



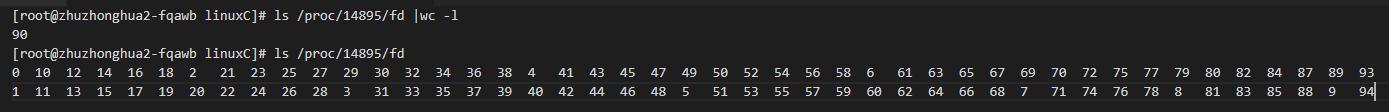
第一列是句柄数，第二列是进程号PID.



这里多了一个是由于：



多了“COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME”这一行。  
而文件描述符的个数为90：



## 文件描述符（file descriptor）

对于linux而言，所有对设备和文件的操作都使用文件描述符来进行的。文件描述符是一个非负的整数，它是一个索引值，指向内核中每个进程打开文件的记录表。当打开一个现存文件或创建一个新文件时，内核就向进程返回一个文件描述符；当需要读写文件时，也需要把文件描述符作为参数传递给相应的函数。   
通常，一个进程启动时，都会打开3个文件：标准输入、标准输出和标准出错处理。这3个文件分别对应文件描述符为0、1和2（宏STD\_FILENO、STDOUT\_FILENO和STDERR\_FILENO）。

每一个文件描述符会与一个打开文件相对应，同时，不同的文件描述符也会指向同一个文件。相同的文件可以被不同的进程打开也可以在同一个进程中被多次打开。系统为每一个进程维护了一个文件描述符表，该表的值都是从0开始的，所以在不同的进程中你会看到相同的文件描述符，这种情况下相同文件描述符有可能指向同一个文件，也有可能指向不同的文件。具体情况要具体分析，要理解具体其概况如何，需要查看由内核维护的3个数据结构。

1. 进程级的文件描述符表
2. 系统级的打开文件描述符表
3. 文件系统的i-node表

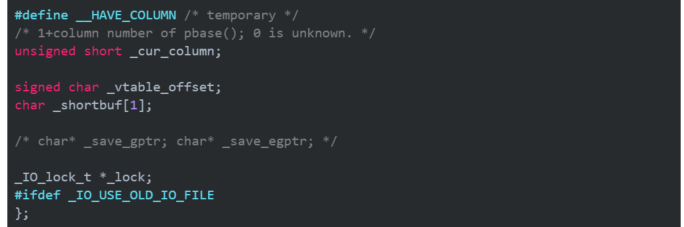
由于进程级文件描述符表的存在，不同的进程中会出现相同的文件描述符，它们可能指向同一个文件，也可能指向不同的文件。两个不同的文件描述符，若指向同一个打开文件句柄，将共享同一文件偏移量。因此，如果通过其中一个文件描述符来修改文件偏移量，那么从另一个文件描述符中也会观察到变化，无论这两个文件描述符是否属于不同进程，还是同一个进程，情况都是如此。

## 文件句柄 vs 文件描述符

文件句柄也称为文件指针(FILE \*)：C语言中使用文件指针做为I/O的句柄。文件指针指向进程用户区中的一个被称为FILE结构的数据结构。FILE结构包括一个缓冲区和一个文件描述符。而文件描述符是文件描述符表的一个索引，因此从某种意义上说文件指针就是句柄的句柄（在Windows系统上，文件描述符被称作文件句柄）。

C语言中FILE结构体的定义：





这个\_IO\_FILE结构体中的“int \_fileno”就是fd，即文件描述符。这个可以通过程序验证：



编译：g++ fileno.cpp -o fileno.out   
执行+输出：



查看test文件：

