

Finding Open Files with lsof

Sean Walberg

Files are ubiquitous in the UNIX® environment, leading to a common phrase: "Everything is a file." Not only is regular data accessed through files, but also network connections and, often, hardware. In some cases, an entry appears when you request a directory listing through ls. In other cases, such as Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) sockets, there is no directory listing. But behind the scenes, a file descriptor has been assigned to the application, regardless of the nature of the file, providing a common interface for applications to interact with the underlying operating system.

Because the list of open file descriptors of an application give a great deal of information about the application itself, being able to peer into this list is helpful. The utility to do so is called lsof, which means "list open files." This utility is available for nearly every UNIX flavor but, strangely, most vendors don't include it with the stock install of the operating system. For information about obtaining lsof, see the Resources section.

Introducing lsof

Simply typing lsof produces a lot of detail, as shown in Listing 1. Because lsof needs access to kernel memory and many files, it must be run as root to be fully effective.

Listing 1
Sample output of lsof

```
bash-3.00# lsof
COMMAND    PID    USER   FD     TYPE        DEVICE  SIZE/OFF      NODE NAME
sched      0     root   cwd    VDIR        136,8    1024          2 /
init       1     root   cwd    VDIR        136,8    1024          2 /
init       1     root   txt    VREG        136,8    49016        1655 /sbin/init
init       1     root   txt    VREG        136,8    51084        3185
/lib/libuutil.so.1
vi         2013  root   3u    VREG        136,8      0          8501
/var/tmp/ExXDaO7d
...
```

One open file is displayed per line and, unless you specify otherwise, all open files from all processes are displayed. The Command, PID, and User columns represent the name of a process, process identifier (PID), and owner's name, respectively. The Device, SIZE/OFF, Node, and Name columns refer to the file itself, specifying the name of the disk, size of the file, inode (the file's identification on the disk), and actual name of the file. Depending on the flavor of UNIX, the size of the file might also be reported as the current position the application is reading in the file (offset). Listing 1 came from a Sun Solaris 10 machine that has the ability to report this information, whereas Linux® can't.

The FD and Type columns are the most cryptic and provide more information about how the file is being used. The FD column represents the file descriptor, which is how the application sees the file. The Type column gives more description about what form the file takes. Looking specifically at the file descriptor column, there are three different values represented in Listing 1. The cwd value refers to the application's current working directory, which is the directory that the application was started from, unless it has changed the directory itself. The txt files are program code, such as the application binary itself or a shared library, as in the case of the init program in the listing. Finally, a number refers to the application's file descriptor, which is an integer returned upon opening the file. In the final line of the output from Listing 1, you can see that the user is editing /var/tmp/ExXDaO7d with vi, with file descriptor 3. The u means the file has been opened in read/write mode, rather than read-only (r) or write-only (w). As a bit of helpful trivia, each application is initially opened with three file descriptors, 0 through 2, for the standard input, output, and error streams, respectively. As such, most opened files from the application start at FD 3.

Finding Open Files with lsof

Sean Walberg

Compared to the FD column, the Type column is more straightforward. Depending on the operating system, you find files and directories called REG and DIR (in the case of Solaris, VREG and VDIR). Other possible values are CHR and BLK for character and block devices, or UNIX, FIFO, and IPv4 for UNIX domain sockets, first-in-first-out (FIFO) queues, and Internet Protocol (IP) sockets.

A Diversion Through The /Proc Directory

While not directly related to using lsof, a brief diversion to the /proc directory is in order. /proc is a directory in which the files are really just views of the kernel and process tree. The files and directories don't exist on disk, so when you're reading and writing to them, you're really getting information from the operating system itself. Most of the information relative to lsof is stored in directories named after the PID of the process, so /proc/1234 would have information on PID 1234.

Within each process directory in the /proc directory are files that can give an application a glimpse into the memory space of the process, a list of the file descriptors, a symbolic link to the file on disk, and other system information. The lsof utility uses this information and other knowledge about the kernel's internals to produce its output. Later on I'll tie the output of lsof to information in the /proc directory.

Common Usage

Earlier I showed you how simply running lsof without any parameters dumps information about every open file belonging to every process. The rest of this article focuses on using lsof to display only the information you need and how to interpret it properly.

Finding an Application's Open Files

One common use of lsof is to find the names, and possibly the numbers, of files an application has open. You might be trying to find out where a particular application is logging data to, or you might be tracking down a problem. As an example, UNIX limits the number of files a process can open. Often that number is high enough to not cause a problem, and the application can request more (up to a limit) if need be. If you suspect an application is running out of file descriptors, you can use lsof to count the number of files open to verify.

To specify a single process, you use the -p parameter, followed by the PID of the process. Because this brings up not only the files opened by the application but also the shared libraries and code, it is often worthwhile to filter those out. To do this, use the -d flag to filter on the FD column and the -a flag to indicate that both parameters must be met (AND). Without the -a flag, the default is to show files that match either of the parameters (OR). Listing 2 shows the files open for the sendmail process with the txt files filtered out.

Listing 2
Lsof output with a PID filter and txt file descriptors filtered out

```
sh-3.00# lsof -a -p 605 -d ^txt
COMMAND PID USER  FD  TYPE  DEVICE  SIZE/OFF      NODE NAME
sendmail 605 root   cwd  VDIR  136,8 1024 23554  /var/spool/mqueue
sendmail 605 root    0r  VCHR  13,2  6815752  /devices/pseudo/mm@0:null
sendmail 605 root    1w  VCHR  13,2  6815752  /devices/pseudo/mm@0:null
sendmail 605 root    2w  VCHR  13,2  6815752  /devices/pseudo/mm@0:null
sendmail 605 root    3r  DOOR  0t0    58      /var/run/name_service_door(door to
nscd[81]) (FA:->0x30002b156c0)
sendmail 605 root    4w  VCHR  21,0 11010052 /devices/pseudo/log@0:conslog->LOG
sendmail 605 root    5u  IPv4  0x300010ea640  0t0      TCP *:smtp (LISTEN)
sendmail 605 root    6u  IPv6  0x3000431c180  0t0      TCP *:smtp (LISTEN)
sendmail 605 root    7u  IPv4  0x300046d39c0  0t0      TCP *:submission
(LISTEN)
```

Finding Open Files with lsof

Sean Walberg

```
sendmail 605 root      8wW VREG 281,3 32 8778600 /var/run/sendmail.pid
```

Listing 2 specifies three parameters to lsof. The first is -a, which means that all parameters must hold true for a file to be displayed. The second parameter, -p 605, limits the output to PID 605, obtained from the ps command. The final parameter, -d ^txt, means that txt records are to be filtered out (the carat [^] means exclude).

The output of Listing 2 gives information about how the process behaves. The application's working directory is /var/spool/mqueue, as indicated by the cwd row. File descriptors 0, 1, and 2 are open to /dev/null (Solaris makes heavy use of symlinks, which is why the pseudo device is shown). FD 3 is a Solaris door (a high-speed Remote Procedure Call (RPC) interface), opened in read-only mode. FD 4 is where things get a bit more interesting, since it is a write-only handle to a character device, essentially /dev/log. From this, you can gather that the application logs to the UNIX syslog daemon, so /etc/syslog.conf dictates the location of the log files.

As a network application, sendmail listens on network ports. File descriptors 5, 6, and 7 tell you that the application is listening on the Simple Mail Transfer Protocol (SMTP) port in both IPv4 and IPv6 mode and on the submission port in IPv4 mode. The final file descriptor is write-only and refers to /var/run/sendmail.pid. The capital W in the FD column indicates that the application has a write lock on the whole file. This file is used to make sure that only one instance of the application is open at a time.

Finding an Open File's Application

In other cases, you have a file or directory and need to know what application owns the file (if the file is open, of course.) Listing 2 showed that /var/run/sendmail.pid was opened by the sendmail process. If you didn't know this, lsof could provide this information if given the file name. Listing 3 shows the output.

Listing 3 Lsof being asked for information about a file

```
bash-3.00# lsof /var/run/sendmail.pid
COMMAND PID USER  FD  TYPE DEVICE SIZE/OFF      NODE NAME
sendmail 605 root   8wW VREG 281,3      32 8778600 /var/run/sendmail.pid
```

As the output shows, /var/run/sendmail.pid is owned by PID 605, which is sendmail, and has been opened for writing with an exclusive lock. If for some reason you needed to get rid of the file, the intelligent thing to do would be to stop the process, rather than just deleting the file. Otherwise, the daemon might fail to start properly next time, or another instance might start up later and cause contention.

Sometimes you know only that a file is open at some part of the file system. When unmounting a file system, the operation fails if any files are open on the file system. You can use lsof to show all open files on a file system by specifying the name of the mount point. Listing 4 shows an attempt to unmount /export/home and then the use of lsof to find out what is using the file system.

Listing 4 Using lsof to find out who is using a file system

```
bash-3.00# umount /export/home
umount: /export/home busy
bash-3.00# lsof /export/home
COMMAND PID USER  FD  TYPE DEVICE SIZE/OFF      NODE NAME
```

Finding Open Files with lsof

Sean Walberg

```
bash 1943 root cwd VDIR 136,7 1024 4 /export/home/sean
bash 2970 sean cwd VDIR 136,7 1024 4 /export/home/sean
ct 3030 sean cwd VDIR 136,7 1024 4 /export/home/sean
ct 3030 sean 1w VREG 136,7 0 25 /export/home/sean/output
```

In this example, a user (sean) is doing some work in his home directory. There are two instances of bash (a shell) running, with the current directory being sean's home directory. There is also an application named ct that is running out of the same directory and has its standard output (file descriptor 1) redirected to a file called output. To successfully unmount /export/home, these processes must be stopped, hopefully after giving the user a call to make sure it's all right.

This example shows how the current working directory of an application is important, because it still holds a file resource and can prevent a file system from being unmounted. This is why most daemons (background processes) change their directory to the root, or a service-specific directory, such as /var/spool/mqueue in the case of sendmail, to prevent the daemon from preventing an unrelated file system from being unmounted. Had sendmail been started from /export/home/sean and not changed its directory to /var/spool/mqueue, it would have had to be stopped before unmounting /export/home.

If you are interested in the open files under a directory that is not a mount point, you must specify the name of the directory with +d or +D, depending on if you need to recurse into subdirectories (+D), or not (+d). For instance, to see all open files under /export/home/sean, use lsof +D /export/home/sean. This is a subtle difference from the previous example, where the directory in question was a mount point, and is just a limitation of the way lsof and the kernel interact. It also brings up a potential problem in that lsof /export/home is different from lsof /export/home/ (note the trailing slash). The first works because it refers to the mount point. The second doesn't produce any output because it refers to the directory. You might run into this if you use tab completion in your shell, which helpfully adds the trailing slash. In this case, you either have to remove the slash or specify the directory with +D. The first is the preferred method, because it is faster than specifying an arbitrary directory.

Uncommon Usage

The previous section explored the fundamental usage of lsof, namely to show the relationship between open files and the processes that own them. This is helpful for the times when you want to do some heavy-handed work on the system without stomping on someone's important document. You can also use the same techniques to perform some UNIX wizardry.

Recovering a Deleted File

A common thing that happens when a UNIX machine is broken into is that the log files are deleted to cover the attacker's tracks. Administrative error can also cause important files to be inadvertently deleted, such as accidentally removing the active transaction log of a database while cleaning up the old logs. Sometimes these files can be recovered, and lsof can help you.

When a process opens a file, it exists on disk even if deleted, as long as the process holds the file open. This means that the process doesn't know the file has been deleted; it can still read and write to the file descriptor it was granted when the file was opened. If you're not that process, the file is invisible because the directory entries have been removed.

Recall that in A diversion through the /proc directory section you had access file descriptors of a process by looking in the appropriate directory. You later discovered that lsof shows the file descriptor of a process and the associated file name. See where I'm going here?

If only it were that easy. When you pass lsof the name of a file, such as in lsof /file//deleted, it first uses the stat() system call to get information about the file, which is, unfortunately, gone. Depending

Finding Open Files with lsof

Sean Walberg

on the operating system, lsof might be able to grab the name of the file from kernel memory. Listing 5 shows a Linux system where an Apache log has accidentally been deleted, and I am using the grep tool to find out if anyone has it open.

Listing 5 Using lsof on Linux to look for a deleted file

```
# lsof | grep error_log
Httpd 2452 root 2w REG 33,2 499 3090660 /var/log/httpd/error_log
(deleted)
Httpd 2452 root 7w REG 33,2 499 3090660 /var/log/httpd/error_log
(deleted)
... more httpd processes ...
```

From this, you can see that PID 2452 has the file opened on file descriptors 2 (standard error) and 7. Thus, the data is available by looking at /proc/2452/fd/7, as shown in Listing 6.

Listing 6 Looking at a deleted file through /proc

```
# cat /proc/2452/fd/7
[Sun Apr 30 04:02:48 2006] [notice] Digest: generating secret for digest
authentication
[Sun Apr 30 04:02:48 2006] [notice] Digest: done
[Sun Apr 30 04:02:48 2006] [notice] LDAP: Built with OpenLDAP LDAP SDK
```

Linux was nice in that it saved the name of the file and even told us it was deleted. This is a handy thing to look for when investigating a compromised system because attackers often delete logs to hide their tracks. Solaris doesn't offer this information. However, knowing that error_log is used by the httpd daemon and that I can find the PID with the ps command, I can look at all the open files for the daemon.

Listing 7 Looking for deleted files in Solaris

```
# lsof -a -p 8663 -d ^txt
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
httpd 8663 nobody cwd VDIR 136,8 1024 2 /
httpd 8663 nobody 0r VCHR 13,2 6815752
/devices/pseudo/mm@0:null
httpd 8663 nobody 1w VCHR 13,2 6815752
/devices/pseudo/mm@0:null
httpd 8663 nobody 2w VREG 136,8 185 145465 / (/dev/dsk/c0t0d0s0)
httpd 8663 nobody 4r DOOR 0t0 58 /var/run/name_service_door (door to
nscd[81]) (FA:->0x30002b156c0)
httpd 8663 nobody 15w VREG 136,8 185 145465 /
(/dev/dsk/c0t0d0s0)
httpd 8663 nobody 16u IPv4 0x300046d27c0 0t0 TCP *:80 (LISTEN)
httpd 8663 nobody 17w VREG 136,8 0 145466
/var/apache/logs/access_log
httpd 8663 nobody 18w VREG 281,3 0 9518013 /var/run (swap)
```

I use the -a and -d parameters to filter my output to exclude code segments, which I know aren't what I'm looking for. A look at the Name column shows that two of the files (FDs 2 and 15) have the disk

Finding Open Files with lsof

Sean Walberg

name instead of a file name and that they're of type VREG (regular file). In Solaris, a deleted file shows up as the name of the disk that the file resides on. This is your clue that the FD refers to a deleted file. Indeed, a look at `/proc/8663/fd/15` gives me the data I'm looking for.

When you can view the data through the file descriptor, you can copy it into a file with I/O redirection, such as in `cat /proc/8663/fd/15 > /tmp/error_log`. At this point, you can stop the daemon (which eliminates the FD and, hence, the deleted file), copy the temporary file to where it is expected, and then restart the daemon.

This method of recovering deleted files is handy for many applications, especially log files and databases. As you can see, some operating systems (and versions of lsof) make it easier than others to find the data.

Finding Network Connections

Network connections are also files, which means that lsof can also get information about them. You saw an example of this in Listing 2. It assumed you already knew the PID, which is not always the case. If you know only the port, use the `-i` parameter to search using socket information. Listing 8 shows a search for TCP port 25.

Listing 8 Looking for the process listening on port 25

```
# lsof -i :25
COMMAND PID USER  FD  TYPE          DEVICE  SIZE/OFF  NODE NAME
sendmail 605 root   5u  IPv4 0x300010ea640    0t0  TCP *:smtp (LISTEN)
sendmail 605 root   6u  IPv6 0x3000431c180    0t0  TCP *:smtp (LISTEN)
```

The lsof utility expects that you will pass it something in the form of `protocol:@ip:port`, where the protocol is TCP or UDP (and optionally prefixed by 4 or 6 to refer to the version of IP), the IP is a resolvable name or IP address, and the port is a number or name (out of `/etc/services`) representing the service. One or more elements (port, IP, protocol) are required. In Listing 8, `:25` refers to port 25. The output shows that process 605 is listening on port 25 using both IPv6 and IPv4. If you're not interested in IPv4, you can change the filter to `6:25` to refer to an IPv6 socket listening on port 25, or simply `6` to mean all IPv6 connections.

In addition to seeing what daemons are listening, lsof can also spy on what connections are happening, again using the `-i` parameter. Listing 9 shows a search for all connections to or from 192.168.1.10.

Listing 9 Searching for active connections

```
# lsof -i @192.168.1.10
COMMAND PID USER  FD  TYPE          DEVICE  SIZE/OFF  NODE NAME
Sshd    1934 root   6u  IPv6 0x300046d21c0 0t1303608  TCP sun:ssh->linux:40379
                                     (ESTABLISHED)
Sshd    1937 root   4u  IPv6 0x300046d21c0 0t1303608  TCP sun:ssh->linux:40379
                                     (ESTABLISHED)
```

In this example, there are two IPv6 connections between sun and linux. A closer look shows that the connections are owned by two separate processes, but they're the same because both the hosts are the same and so are the ports (ssh and 40379). This is caused by the connection coming into a master process that forks off a handler, passing it the socket. You can also see that the machine

Finding Open Files with lsof

Sean Walberg

called sun is using port 22 (ssh), while linux has a port of 40379. This indicates that sun is the recipient of the connection because it is the well-known port associated with the service. 40379 is the source, or ephemeral port, and is only significant for this connection.

Because, at least in UNIX, a socket is just another file, lsof can get detailed information about the connections and find out who is responsible for them.

Summary

UNIX makes heavy use of files. As a systems administrator, lsof lets you peer into kernel memory to find out how the system is currently using these files. The simplest uses of lsof tell you which process has what files open and which files are opened by what process. This is particularly helpful for gleaning information about how an application works or for making sure a file isn't being used before doing something that could cause data corruption. More advanced uses of lsof help you find deleted files and get more information about network connections. This is a powerful tool -- the places it can be used are endless.