

strace

Strace is a tool for tracing system calls and signals [4]. It intercepts and records the system calls made by a running process. strace can print a record of each system call, its arguments, and its return value. You can use strace on programs for which you do not have the source since using strace does not require recompilation. It is often useful in instances where a program freezes or otherwise fails to work and offers few clues as to the problem. It is also a great for instructional purposes.

Basics

It is easiest to explain with an example. The simplest example is perhaps

```
bash$ strace -o strace-echo-output.txt echo "Hello there"
Hello World
bash$
```

This example traced the execution of echo "Hello there", and printed the resulting trace to strace-echo-output.txt. As you can see, the program runs normally, the only difference is that it runs a little slower under strace and at the end you have a trace file. Trace files tend to be fairly large. Even for this simple example, strace-echo-output.txt was 48 lines long for me. The file contains lines such as

```
open("/opt/gnome2/lib/libc.so.6", O_RDONLY) = -1 ENOENT (No such file or directory)
stat64("/opt/gnome2/lib", {st_mode=S_IFDIR|0755, st_size=20480, ...}) = 0
open("/etc/ld.so.cache", O_RDONLY) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=65928, ...}) = 0
old_mmap(NULL, 65928, PROT_READ, MAP_PRIVATE, 3, 0) = 0xbf596000
close(3) = 0
open("/lib/tls/libc.so.6", O_RDONLY) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0`ht\000"... , 512) = 512
```

That may look a little intimidating at first, but it is not so scary once you learn the basic format and realize that these files are usually meant to be searched, not read. Each line starts with a system call name, is followed by its arguments in parenthesis and then has the return value at the end of the line. Errors (which typically have a return value of -1) have the symbolic error name (such as ENOENT in the first line in the example above) as well as a more informative error string appended.

You can usually follow an strace output file using common sense without ever looking up the meanings of most of the system calls since the important ones tend to have names with pretty obvious meanings (e.g. open, read, write, close). However, if you are curious, you can look up the other system calls in the man pages, such as read(2) . For some of the system calls, you may have to try slightly different names, such as mmap instead of old_mmap and stat instead of stat64. (If you are unfamiliar with man, you would use the command man 2 mmap to see the mmap(2) man page.)

The interesting part of the strace output file is often near the end. This case is no different. Near the end of the file we see the line

```
write(1, "Hello World\n", 12) = 12
```

This is the line corresponding to the system call that caused Hello World to be printed on the screen. Note that the first argument, 1, is the file descriptor of the file to write to. Since 1 corresponds to standard output (see stdout(3) for more information), this message is written to the terminal instead of being written to some file on the disk or some pipe.

Examples

Permissions Problem

An example that demonstrates permissions failure is strace -o strace-less-output.txt less /etc/sudoers. This command, when run as a normal user instead of root, will result in a "Permission denied" error

strace

message on the screen since only the root user has permissions to read that file. In the strace output file are the following two lines:

```
open("/etc/sudoers", O_RDONLY|O_LARGEFILE) = -1 EACCES (Permission denied)
write(2, "/etc/sudoers: Permission denied\n", 32) = 32
```

This shows the the open command resulted in an EACCES error, and that the less program then printed an error message to stderr (2 is the file descriptor for standard error, see stderr(3) for more information). You should note that, unlike less, there are many programs that silently fail when they do not have permissions to read the files that they try to open. This is one of the things that makes strace so handy.

Searching \$PATH

Another common problem is when a file does not exist (an ENOENT error) and the program fails silently. However, most of the times that this error occurs in an strace output it is innocuous as the example `strace -o strace-which-output.txt` which `jhbuild` demonstrates. This command will try to find the location of the `jhbuild` executable. The relevant portion of the strace output file is the following lines

```
stat64("/opt/gnome2/bin/jhbuild", 0xbfeda98c) = -1 ENOENT (No such file or directory)
stat64("/usr/kerberos/bin/jhbuild", 0xbfeda98c) = -1 ENOENT (No such file or directory)
stat64("/usr/local/bin/jhbuild", 0xbfeda98c) = -1 ENOENT (No such file or directory)
stat64("/usr/bin/jhbuild", 0xbfeda98c) = -1 ENOENT (No such file or directory)
stat64("/bin/jhbuild", 0xbfeda98c) = -1 ENOENT (No such file or directory)
stat64("/usr/X11R6/bin/jhbuild", 0xbfeda98c) = -1 ENOENT (No such file or directory)
stat64("/home/newren/bin/jhbuild", {st_mode=S_IFREG|0755, st_size=153, ...}) = 0
```

Note that there were several "No such file or directory" errors. In this case, it was simply because the which command was walking through my PATH, looking for an executable named `jhbuild`. Since it was in a directory near the end of my PATH, it first had to look in several wrong directories before it found it.

Finding a Relevant Configuration File

As a final example, let me provide a case where I used strace to solve an actual problem I had. I had a workstation, which had been configured by a very busy system administrator. It was set up to print to any of dozens of printers at the school. My laptop, however, was set up by me and had not been configured to print to any of those printers. Adding each printer manually and trying to get all the configuration options right (especially since the closest of the printers was on the opposite side of the building) was a royal pain. I knew I should just be able to copy some kind of configuration file from the workstation to my laptop and everything would work, but I was not very familiar with CUPS, so I did not know the name of the configuration file. I tried searching for files in the `/etc/cups` directory (and a few other places) that contained one of the printer names, but did not find any. So, I ran `strace -o strace-lpq-output.txt lpq`, and then searched for files that were opened. I found

```
open("/etc/cups/client.conf", O_RDONLY) = 3
```

Upon closer examination of that file, I noticed a line that read

```
ServerName print.math.utah.edu
```

Apparently, the printer names were not showing up in any of the configuration files simply because a print server was being used. By adding this line to the same file on my laptop, I was suddenly connected to all the printers.

strace

Learning more

There are many other options you can pass to strace to fine tune its behavior. The options range from restricting the tracing to a subset of system calls to timing how long various things take. You can read more about these options in the `strace(1)` man pages, but let me just point out a few of the more common options. It is not very uncommon for a program to have a shell script wrapper. In those cases, one usually wants to trace the real application and not just the wrapper shell script. This is where the `-f` and `-ff` options come in handy. The `-f` option causes strace to trace child process created by the `fork(2)` system call. Since this can often result in very long traces, the `-ff` splits the traces into separate files according to process. The `-p` option allows one to attach to an already existing process by its pid number. Finally, the `-s` with a specified (integer) length allows one to adjust the maximum length of strings to print.

If you are using one of the BSD variants instead of linux, the equivalent tool is called `ktrace`. Note that the arguments that `ktrace` takes are different than the ones that `strace` takes so you will need to read its man page. You will also need to use `kdump` since the output of `ktrace` is not human readable.

Due to an unfortunate abbreviation, many users mistakenly think that strace will provide stack traces. This is not the case; a stack trace is very different from a system call trace. If a stack trace (also known as a backtrace) is what you want, `gdb` is the program to use.