NAME

strace − trace system calls and signals

SYNOPSIS

strace [**−dffhiqrtttTvxx**] [**−a***column*] [**−e***expr*] ... [**−o** *file*] [**−p** *pid*] ... [**−s***strsize*] [**−u***username*] [**−E***var=val*] ... [**−E***var*] ... [*command* [*arg* ...]]

strace −c [**−e***expr*] ... [**−O***overhead*] [**−S***sortby*] [*command* [*arg* ...]]

DESCRIPTION

In the simplest case **strace** runs the specified *command* until it exits. It intercepts and records the system calls which are called by a process and the signals which are received by a process. The name of each system call, its arguments and its return value are printed on standard error or to the file specified with the **−o** option.

strace is a useful diagnostic, instructional, and debugging tool. System administrators, diagnosticians and trouble-shooters will find it invaluable for solving problems with programs for which the source is not readily available since they do not need to be recompiled in order to trace them. Students, hackers and the overly-curious will find that a great deal can be learned about a system and its system calls by tracing even ordinary programs. And programmers will find that since system calls and signals are events that happen at the user/kernel interface, a close examination of this boundary is very useful for bug isolation, sanity checking and attempting to capture race conditions.

Each line in the trace contains the system call name, followed by its arguments in parentheses and its return value. An example from stracing the command ‘‘cat /dev/null’’ is:

```
open("/dev/null", O_RDONLY) = 3
```

Errors (typically a return value of −1) have the errno symbol and error string appended.

```
open("/foo/bar", O_RDONLY) = -1 ENOENT (No such file or directory)
```

Signals are printed as a signal symbol and a signal string. An excerpt from stracing and interrupting the command ‘‘sleep 666’’ is:

```
sigsuspend([] <unfinished ...>
--- SIGINT (Interrupt) ---
+++ killed by SIGINT +++
```

Arguments are printed in symbolic form with a passion. This example shows the shell performing ‘‘>>xyzzy’’ output redirection:

```
open("xyzzy", O_WRONLY|O_APPEND|O_CREAT, 0666) = 3
```

Here the three argument form of open is decoded by breaking down the flag argument into its three bitwise-OR constituents and printing the mode value in octal by tradition. Where traditional or native usage differs from ANSI or POSIX, the latter forms are preferred. In some cases, **strace** output has proven to be more readable than the source.

Structure pointers are dereferenced and the members are displayed as appropriate. In all cases arguments are formatted in the most C-like fashion possible. For example, the essence of the command ‘‘ls −l /dev/null’’ is captured as:

```
lstat("/dev/null", {st_mode=S_IFCHR|0666, st_rdev=makedev(1, 3), ...}) = 0
```

Notice how the ‘struct stat’ argument is dereferenced and how each member is displayed symbolically. In particular, observe how the st_mode member is carefully decoded into a bitwise-OR of symbolic and

numeric values. Also notice in this example that the first argument to lstat is an input to the system call and the second argument is an output. Since output arguments are not modified if the system call fails, arguments may not always be dereferenced. For example, retrying the ‘‘ls −l’’ example with a non-existent file produces the following line:

```
lstat("/foo/bar", 0xb004) = -1 ENOENT (No such file or directory)
```

In this case the porch light is on but nobody is home.

Character pointers are dereferenced and printed as C strings. Non-printing characters in strings are normally represented by ordinary C escape codes. Only the first *strsize* (32 by default) bytes of strings are printed; longer strings have an ellipsis appended following the closing quote. Here is a line from ‘‘ls −l’’ where the **getpwuid** library routine is reading the password file:

```
read(3, "root::0:0:System Administrator:/"..., 1024) = 422
```

While structures are annotated using curly braces, simple pointers and arrays are printed using square brackets with commas separating elements. Here is an example from the command ‘‘id’’ on a system with supplementary group ids:

```
getgroups(32, [100, 0]) = 2
```

On the other hand, bit-sets are also shown using square brackets but set elements are separated only by a space. Here is the shell preparing to execute an external command:

```
sigprocmask(SIG_BLOCK, [CHLD TTOU], []) = 0
```

Here the second argument is a bit-set of two signals, SIGCHLD and SIGTTOU. In some cases the bit-set is so full that printing out the unset elements is more valuable. In that case, the bit-set is prefixed by a tilde like this:

```
sigprocmask(SIG_UNBLOCK, ~[], NULL) = 0
```

Here the second argument represents the full set of all signals.

OPTIONS

−c Count time, calls, and errors for each system call and report a summary on program exit. On Linux, this attempts to show system time (CPU time spent running in the kernel) independent of wall clock time.

−d Show some debugging output of **strace** itself on the standard error.

−f Trace child processes as they are created by currently traced processes as a result of the **fork**(2) system call. The new process is attached to as soon as its pid is known (through the return value of **fork**(2) in the parent process). This means that such children may run uncontrolled for a while (especially in the case of a **vfork**(2)), until the parent is scheduled again to complete its (**v**)**fork**(2) call. If the parent process decides to **wait**(2) for a child that is currently being traced, it is suspended until an appropriate child process either terminates or incurs a signal that would cause it to terminate (as determined from the child’s current signal disposition).

−ff If the **−o** *filename* option is in effect, each processes trace is written to *filename.pid* where pid is the numeric process id of each process.

−F Attempt to follow **vfork**s. (On SunOS 4.x, this is accomplished with some dynamic linking trickery.) Otherwise, **vfork**s will not be followed even if **−f** has been given.

−h Print the help summary.

−i Print the instruction pointer at the time of the system call.

−q Suppress messages about attaching, detaching etc. This happens automatically when output is redirected to a file and the command is run directly instead of attaching.

−r Print a relative timestamp upon entry to each system call. This records the time difference between the beginning of successive system calls.

−t Prefix each line of the trace with the time of day.

−tt If given twice, the time printed will include the microseconds.

−ttt If given thrice, the time printed will include the microseconds and the leading portion will be printed as the number of seconds since the epoch.

−T Show the time spent in system calls. This records the time difference between the beginning and the end of each system call.

−v Print unabbreviated versions of environment, stat, termios, etc. calls. These structures are very common in calls and so the default behavior displays a reasonable subset of structure members. Use this option to get all of the gory details.

−V Print the version number of **strace**.

−x Print all non-ASCII strings in hexadecimal string format.

−xx Print all strings in hexadecimal string format.

−a *column* Align return values in a specific column (default column 40).

−e *expr* A qualifying expression which modifies which events to trace or how to trace them. The format of the expression is:

[*qualifier*=][**!**]*value1*[**,***value2*]...

where *qualifier* is one of **trace**, **abbrev**, **verbose**, **raw**, **signal**, **read**, or **write** and *value* is a qualifier-dependent symbol or number. The default qualifier is **trace**. Using an exclamation mark negates the set of values. For example, **−eopen** means literally **−e trace=open** which in turn means trace only the **open** system call. By contrast, **−etrace=!open** means to trace every system call except **open**. In addition, the special values **all** and **none** have the obvious meanings.

Note that some shells use the exclamation point for history expansion even inside quoted arguments. If so, you must escape the exclamation point with a backslash.

−e trace=*set* Trace only the specified set of system calls. The **−c** option is useful for determining which system calls might be useful to trace. For example, **trace=open,close,read,write** means to only trace those four system calls. Be careful when making inferences about the user/kernel boundary if only a subset of system calls are being monitored. The default is **trace=all**.

−e trace=file Trace all system calls which take a file name as an argument. You can think of this as an abbreviation for **−e trace=open,stat,chmod,unlink,**... which is useful to seeing what files the process is referencing. Furthermore, using the abbreviation will ensure that you don't accidentally forget to include a call like **lstat** in the list. Betchya woulda forgot that one.

−e trace=process
Trace all system calls which involve process management. This is useful for watching the fork, wait, and exec steps of a process.

−e trace=network
Trace all the network related system calls.

−e trace=signal
Trace all signal related system calls.

−e trace=ipc Trace all IPC related system calls.

−e abbrev=*set*
Abbreviate the output from printing each member of large structures. The default is **abbrev=all**. The **−v** option has the effect of **abbrev=none**.

−e verbose=*set*
Dereference structures for the specified set of system calls. The default is **verbose=all**.

−e raw=*set* Print raw, undecoded arguments for the specified set of system calls. This option has the effect of causing all arguments to be printed in hexadecimal. This is mostly useful if you don't trust the decoding or you need to know the actual numeric value of an argument.

−e signal=*set* Trace only the specified subset of signals. The default is **signal=all**. For example, **signal=!SIGIO** (or **signal=!io**) causes SIGIO signals not to be traced.

−e read=*set* Perform a full hexadecimal and ASCII dump of all the data read from file descriptors listed in the specified set. For example, to see all input activity on file descriptors 3 and 5 use **−e read=3,5**. Note that this is independent from the normal tracing of the **read**(2) system call which is controlled by the option **−e trace=read**.

−e write=*set* Perform a full hexadecimal and ASCII dump of all the data written to file descriptors listed in the specified set. For example, to see all output activity on file descriptors 3 and 5 use **−e write=3,5**. Note that this is independent from the normal tracing of the **write**(2) system call which is controlled by the option **−e trace=write**.

−o *filename* Write the trace output to the file *filename* rather than to stderr. Use *filename.pid* if **−ff** is used. If the argument begins with '|' or with '!' then the rest of the argument is treated as a command and all output is piped to it. This is convenient for piping the debugging output to a program without affecting the redirections of executed programs.

−O *overhead* Set the overhead for tracing system calls to *overhead* microseconds. This is useful for overriding the default heuristic for guessing how much time is spent in mere measuring when timing system calls using the **−c** option. The accuracy of the heuristic can be gauged by timing a given program run without tracing (using **time**(1)) and comparing the accumulated system call time to the total produced using **−c**.

−p *pid* Attach to the process with the process ID *pid* and begin tracing. The trace may be terminated at any time by a keyboard interrupt signal (CTRL-C). **strace** will respond by detaching itself from the traced process(es) leaving it (them) to continue running. Multiple **−p** options can be used to attach to up to 32 processes in addition to *command* (which is optional if at least one **−p** option is given).

−s *strsize* Specify the maximum string size to print (the default is 32). Note that filenames are not considered strings and are always printed in full.

−S *sortby* Sort the output of the histogram printed by the **−c** option by the specified criterion. Legal values are **time**, **calls**, **name**, and **nothing** (default **time**).

−u *username* Run command with the user ID, group ID, and supplementary groups of *username*. This option is only useful when running as root and enables the correct execution of setuid and/or setgid binaries. Unless this option is used setuid and setgid programs

are executed without effective privileges.

−E *var=val* Run command with *var=val* in its list of environment variables.

−E *var* Remove *var* from the inherited list of environment variables before passing it on to the command.

SETUID INSTALLATION

If **strace** is installed setuid to root then the invoking user will be able to attach to and trace processes owned by any user. In addition setuid and setgid programs will be executed and traced with the correct effective privileges. Since only users trusted with full root privileges should be allowed to do these things, it only makes sense to install **strace** as setuid to root when the users who can execute it are restricted to those users who have this trust. For example, it makes sense to install a special version of **strace** with mode 'rwsr-xr--', user **root** and group **trace**, where members of the **trace** group are trusted users. If you do use this feature, please remember to install a non-setuid version of **strace** for ordinary lusers to use.

SEE ALSO

ptrace(2), **proc**(4), **time**(1), **trace**(1), **truss**(1)

NOTES

It is a pity that so much tracing clutter is produced by systems employing shared libraries.

It is instructive to think about system call inputs and outputs as data-flow across the user/kernel boundary. Because user-space and kernel-space are separate and address-protected, it is sometimes possible to make deductive inferences about process behavior using inputs and outputs as propositions.

In some cases, a system call will differ from the documented behavior or have a different name. For example, on System V-derived systems the true **time**(2) system call does not take an argument and the **stat** function is called **xstat** and takes an extra leading argument. These discrepancies are normal but idiosyncratic characteristics of the system call interface and are accounted for by C library wrapper functions.

On some platforms a process that has a system call trace applied to it with the **−p** option will receive a **SIGSTOP**. This signal may interrupt a system call that is not restartable. This may have an unpredictable effect on the process if the process takes no action to restart the system call.

BUGS

Programs that use the *setuid* bit do not have effective user ID privileges while being traced.

A traced process ignores SIGSTOP except on SVR4 platforms.

A traced process which tries to block SIGTRAP will be sent a SIGSTOP in an attempt to force continuation of tracing.

A traced process runs slowly.

Traced processes which are descended from *command* may be left running after an interrupt signal (CTRL-C).

On Linux, exciting as it would be, tracing the init process is forbidden.

The **−i** option is weakly supported.

HISTORY

strace The original **strace** was written by Paul Kranenburg for SunOS and was inspired by its trace utility. The SunOS version of **strace** was ported to Linux and enhanced by Branko Lankester, who also wrote the Linux kernel support. Even though Paul released **strace** 2.5 in 1992, Branko's work was based on Paul's **strace** 1.5 release from 1991. In 1993, Rick Sladkey merged **strace** 2.5 for SunOS and the second release of **strace** for Linux, added many of the features of **truss**(1) from SVR4, and produced an **strace** that worked on both platforms. In 1994 Rick ported **strace** to SVR4 and Solaris and wrote the automatic configuration support. In 1995 he ported **strace** to Irix and tired of writing about himself in the third person.

PROBLEMS

Problems with **strace** should be reported via the Debian Bug Tracking System, or to the **strace** mailing list at <strace-devel@lists.sourceforge.net>.