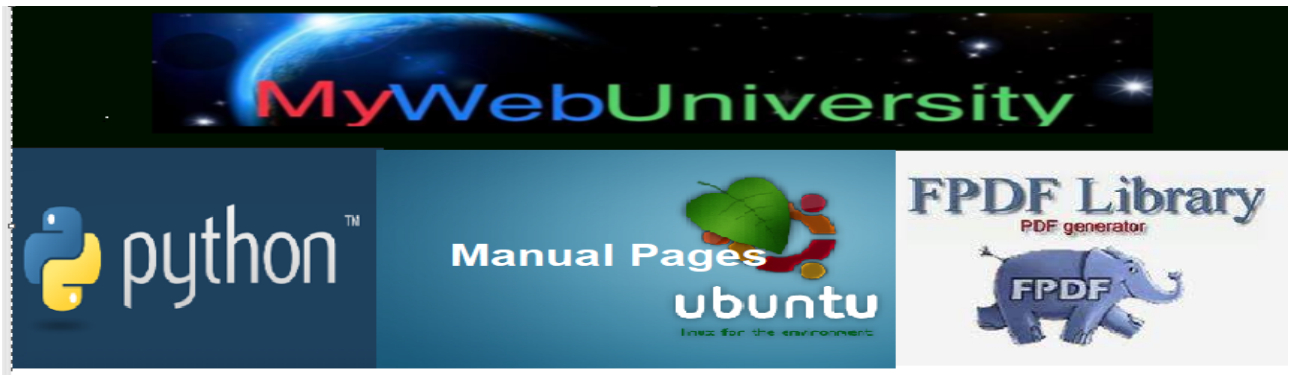

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'strace.1'

$ man strace.1

STRACE(1) General Commands Manual STRACE(1)

NAME

strace - trace system calls and signals

SYNOPSIS

strace [-ACdffhikqqrtttTvVwxxyyYzZ] [-a column] [-b execve] [-e expr]... [-I n] [-o file]
[-O overhead] [-p pid]... [-P path]... [-s strsize] [-S sortby] [-U columns]
[-X format] [--seccomp-bpf] [--stack-trace-frame-limit=limit] [--syscall-limit=limit]
[--tips[=format]] { -p pid | [-DDD] [-E var[=val]]... [-u username] command [args] }

strace -c [-dfwzZ] [-b execve] [-e expr]... [-I n] [-O overhead] [-p pid]... [-P path]...
[-S sortby] [-U columns] [--seccomp-bpf] [--syscall-limit=limit] [--tips[=format]] {
-p pid | [-DDD] [-E var[=val]]... [-u username] command [args] }

strace --tips[=format]

DESCRIPTION

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

strace is a useful diagnostic, instructional, and debugging tool. System administrators, diagnosticians, and troubleshooters will find it invaluable for solving problems with pro? grams for which source code is not readily available, as recompilation is not required for tracing. Students, hackers, and the overly-curious will discover that a great deal can be learned about a system and its system calls by tracing even ordinary programs. Programmers will find that since system calls and signals occur at the user/kernel interface, a close

examination of this boundary is very useful for bug isolation, sanity checking, and attempt?
ing to capture race conditions.

Each line in the trace contains the system call name, followed by its arguments in parenthe?
ses and its return value. An example from tracing the command "cat /dev/null" is:

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

Errors, typically indicated by 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 decoded siginfo structure. An excerpt from
tracing and interrupting the command "sleep 666" is:

```
sigsuspend([] <unfinished ...>
--- SIGINT {si_signo=SIGINT, si_code=SI_USER, si_pid=...} ---
+++ killed by SIGINT +++
```

If a system call is being executed while another is called from a different thread or
process, strace will attempt to preserve the order of these events and mark the ongoing call
as unfinished. When the call returns, it will be marked as resumed.

```
[pid 28772] select(4, [3], NULL, NULL, NULL <unfinished ...>
[pid 28779] clock_gettime(CLOCK_REALTIME, {tv_sec=1130322148, tv_nsec=3977000}) = 0
[pid 28772] <... select resumed> )      = 1 (in [3])
```

The interruption of a (restartable) system call by a signal delivery is handled differently,
as the kernel terminates the system call and arranges for its immediate re-execution after
the signal handler completes.

```
read(0, 0x7ffff72cf5cf, 1)              = ? ERESTARTSYS (To be restarted)
--- SIGALRM {si_signo=SIGALRM, si_code=SI_KERNEL} ---
rt_sigreturn({mask=[]})                 = 0
read(0, "", 1)                          = 0
```

Arguments are printed in symbolic form with passion. This example shows the shell perform?
ing ">>xyzzy" output redirection:

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

Here, the second and third arguments of open(2) are decoded by breaking down the flag argu?
ment into its three bitwise-OR constituents and printing the mode value in octal, following
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

code itself.

Structure pointers are dereferenced, and their members are displayed as appropriate. In most 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(0x1, 0x3), ...}) = 0

Notice how the struct stat argument is dereferenced and how each member is displayed symbol? ically. In particular, observe how the st_mode member is carefully decoded into a bitwise-OR of symbolic and numeric values. Also, note that in this example, the first argument to lstat(2) 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 derefer? enced. For example, retrying the "ls -l" example with a non-existent file produces the fol? lowing line:

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

In this case, the porch light is on but nobody is home. The pointer's value is displayed because the structure it points to was not populated due to the error.

System calls unknown to strace are printed in a raw format, with the hexadecimal system call number prefixed with "syscall_":

syscall_0xbad(0x1, 0x2, 0x3, 0x4, 0x5, 0x6) = -1 ENOSYS (Function not implemented)

Character pointers are dereferenced and printed as C strings. Non-printing characters in strings are normally represented by standard 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(3) library routine is reading the password file:

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

While structures are annotated using curly braces, pointers to basic types and arrays are printed using square brackets with commas separating the elements. Here is an example from the command id(1) 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 sepa? rated 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 it is more valuable to print the unset elements. 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

General

-e expr Modifies which events to trace or how to trace them by specifying a qualifying expression. The format of the expression is:

[qualifier=][!]value[,value]...

where qualifier is one of trace (or t), trace-fds (or trace-fd or fd or fds), abbrev (or a), verbose (or v), raw (or x), signal (or signals or s), read (or reads or r), write (or writes or w), fault, inject, status, quiet (or silent or silence or q), decode-fds (or decode-fd), decode-pids (or decode-pid), or kvm, 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, -e open is equivalent to -e trace=open, which in turn means trace only the open system call. By contrast, -e trace=!open means to trace every system call ex? cept open. In addition, the special values all and none may be used to trace every event or no events, respectively.

Note that some shells use the exclamation mark for history expansion even inside quoted arguments. In that case, the exclamation mark must be escaped with a backslash.

Startup

-E var=val

--env=var=val

Runs the command with the environment variable var=val set for execution.

-E var

--env=var Removes var from the inherited environment variables before executing the com? mand.

-p pid

--attach=pid

Attaches 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 processes, leaving them to con?

tinue running.

Multiple -p options can be used to attach to several processes in addition to the command, which is optional if at least one -p option is given.

A single -p option can accept multiple process IDs separated by a comma (?,?), space (? ?), tab, or newline. Consequently, syntaxes like -p "$(pidof PROG)" and -p "$(pgrep PROG)" are supported.

-u username

--user=username

Runs command with the user ID, group ID, and supplementary groups of username. This option is only useful when running as root, as it enables the correct exe? cution of setuid and/or setgid binaries. Unless this option is used, setuid and setgid programs are executed without their effective privileges.

-u UID:GID

--user=UID:GID

Alternative syntax where the program is started with exactly the given user and group IDs, and an empty list of supplementary groups. In this case, user and group name lookups are not performed.

--argv0=name

Sets the executed command's argv[0] to name. This is useful for tracing multi- call executables that interpret argv[0], such as busybox or kmod.

Tracing

-b syscall

--detach-on=syscall

Detaches from the traced process if the specified system call is reached. Cur? rently, only execve keyword is supported, which includes execve(2) and ex? ecveat(2) system calls. This option is useful for tracing a multi-threaded process with -f without also tracing its (potentially very complex) child processes.

-D

--daemonize

--daemonize=grandchild

Runs the tracer process as a grandchild of the tracee, not as its parent. This reduces the visible effect of strace by keeping the tracee a direct child of the

calling process.

-DD

--daemonize=pgroup

--daemonize=pgrp

Runs tracer process as tracee's grandchild in a separate process group. In ad? dition to reducing the visible effect of strace, this also prevents strace from being terminated by a kill(2) signal sent to the entire process group.

-DDD

--daemonize=session

Runs the tracer process as the tracee's grandchild in a separate session (known as "true daemonisation"). In addition to reduction of the visible effect of strace, this also prevents strace from being terminated upon session termina? tion.

-f

--follow-forks

Traces child processes as they are created by currently traced processes as a result of the fork(2), vfork(2) and clone(2) system calls. Note that if process PID is multi-threaded, using -f -p PID attaches to all of its threads, not just the one with thread_id = PID.

--output-separately

If the --output=filename option is in effect, the trace for each process is written to a separate filename.pid file, where pid is the process ID.

-ff

--follow-forks --output-separately

Combines the effects of --follow-forks and --output-separately options. This is incompatible with -c, since no per-process counts are kept.

Use strace-log-merge(1) to get a combined view of the log files.

-I interruptible

--interruptible=interruptible

Controls when strace can be interrupted by signals (such as pressing CTRL-C).

1, anywhere no signals are blocked;

2, waiting fatal signals are blocked while decoding system call (default);

3, never fatal signals are always blocked (default if -o FILE PROG);

4, never_tstp fatal signals and SIGTSTP (CTRL-Z) are always blocked (useful to make strace -o FILE PROG not stop on CTRL-Z, default if -D).

--syscall-limit=limit

Detaches all tracees after limit system calls have been captured. System calls filtered out via --trace, --trace-path or --status options are not considered when keeping track of the number of system calls that are captured.

--kill-on-exit

Applies the PTRACE_O_EXITKILL ptrace option to all tracees, which sends a SIGKILL signal to a tracee if the tracer exits. This prevents tracees from be? ing left running after the tracer exits, as they will not be detached on cleanup. --kill-on-exit is not compatible with -p/--attach options.

Filtering

-e trace=syscall_set

-e t=syscall_set

--trace=syscall_set

Traces only the specified set of system calls. syscall_set is defined as [!]value[,value], and value can be one of the following:

syscall Traces specific system call, specified by its name (see syscalls(2) for a reference, but also see NOTES).

?value A question mark preceding the qualification suppresses errors if no matching system calls are found.

value@64 Limits the system call specification described by value to the 64-bit personality.

value@32 Limits the system call specification described by value to the 32-bit personality.

value@x32 Limits the system call specification described by value to the x32 personality.

all Traces all system calls.

/regex Traces only those system calls that match the regex. You can use POSIX Extended Regular Expression syntax (see regex(7)).

%file

file Traces all system calls that take a file name as an argument. You can think of this as an abbreviation for

--trace=open,stat,chmod,unlink,... which is useful to seeing what files the process is referencing. Furthermore, using the abbrevia? tion will ensure that you don't accidentally forget to include a call like newfstatat(2) in the list. The syntax without a preced? ing percent sign ("--trace=file") is deprecated.

%process

process Traces system calls associated with process lifecycle (creation, exec, termination). The syntax without a preceding percent sign ("--trace=process") is deprecated.

%net

%network

network Traces all the network related system calls. The syntax without a preceding percent sign ("--trace=network") is deprecated.

%signal

signal Traces all signal related system calls. The syntax without a pre? ceding percent sign ("--trace=signal") is deprecated.

%ipc

ipc Traces all IPC related system calls. The syntax without a preced? ing percent sign ("--trace=ipc") is deprecated.

%desc

desc Traces all file descriptor related system calls. The syntax with? out a preceding percent sign ("--trace=desc") is deprecated.

%memory

memory Traces all memory mapping related system calls. The syntax without a preceding percent sign ("--trace=memory") is deprecated.

%creds Traces system calls that read or modify user and group identifiers or capability sets.

%stat Traces stat system call variants.

%lstat Traces lstat system call variants.

%fstat Traces fstat, fstatat, and statx system call variants.

%%stat Traces system calls used for requesting file status (stat, lstat, fstat, fstatat, statx, and their variants).

%statfs Traces statfs, statfs64, statvfs, osf_statfs, and osf_statfs64 sys?

tem calls. The same effect can be achieved with --trace=/^(.*_)?statv?fs regular expression.

%fstatfs Traces fstatfs, fstatfs64, fstatvfs, osf_fstatfs, and osf_fstatfs64 system calls. The same effect can be achieved with --trace=/fs?tatv?fs regular expression.

%%statfs Traces system calls related to file system statistics (statfs-like, fstatfs-like, and ustat). The same effect can be achieved with --trace=/statv?fs|fsstat|ustat regular expression.

%clock Traces system calls that read or modify system clocks.

%pure Traces system calls that always succeed and have no arguments. Currently, this list includes arc_gettls(2), getdtablesize(2), getegid(2), getegid32(2), geteuid(2), geteuid32(2), getgid(2), get?gid32(2), getpagesize(2), getpgrp(2), getpid(2), getppid(2), get_thread_area(2) (on architectures other than x86), gettid(2), get_tls(2), getuid(2), getuid32(2), getxgid(2), getxpid(2), getxuid(2), kern_features(2), and metag_get_tls(2) 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-fd=set

-e trace-fds=set

-e fd=set

-e fds=set

--trace-fds=set

Traces only the system calls that operate on the specified subset of (non-nega?tive) file descriptors. Note that usage of this option also filters out all the system calls that do not operate on file descriptors at all.

This filter is combined with the --trace-path filter; a system call is traced if it matches either of them.

-e signal=set

-e signals=set

-e s=set

--signal=set

Traces 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 status=set

--status=set

Prints only system calls with the specified return status. The default is --status=all. When using the status qualifier, the chronological order of events may not be preserved. This is because strace must wait for a system call to complete before deciding whether to print it. If two system calls are exe? cuted by concurrent threads, strace will first print both the entry and exit of the first system call to exit, regardless of their respective entry time. The entry and exit of the second system call to exit will be printed afterwards. Here is an example when select(2) is called, but a different thread calls clock_gettime(2) before select(2) finishes:

```
[pid 28779] 1130322148.939977 clock_gettime(CLOCK_REALTIME, {1130322148, 939977000}) = 0
[pid 28772] 1130322148.438139 select(4, [3], NULL, NULL, NULL) = 1 (in [3])
```

set can include the following elements:

successful Traces system calls that returned without an error code. The -z option has the effect of --status=successful.

failed Traces system calls that returned with an error code. The -Z op? tion has the effect of --status=failed.

unfinished Traces system calls that did not return. This might happen, for example, due to an execve call in a different thread from the same thread group.

unavailable Traces system calls that returned but strace failed to fetch the error status.

detached Traces system calls for which strace detached before the return.

-P path

--trace-path=path

Traces only system calls accessing path. Multiple -P options can be used to specify several paths. This filter is combined with the --trace-fds filter; a

system call is traced if it matches either option.

-z

--successful-only

Prints only system calls that returned without an error code.

-Z

--failed-only

Prints only system calls that returned with an error code.

Output format

-a column

--columns=column

Aligns return values in a specific column (default column 40).

-e abbrev=syscall_set

-e a=syscall_set

--abbrev=syscall_set

Abbreviates the output from printing each member of large structures. The syn? tax of the syscall_set specification is the same as in the --trace option. The default is --abbrev=all. The -v option has the effect of --abbrev=none.

-e verbose=syscall_set

-e v=syscall_set

--verbose=syscall_set

Dereferences structures for the specified set of system calls. The syntax of the syscall_set specification is the same as in the --trace option. The default is --verbose=all.

-e raw=syscall_set

-e x=syscall_set

--raw=syscall_set

Prints raw, undecoded arguments for the specified set of system calls. The syn? tax of the syscall_set specification is the same as in the --trace option. This option has the effect of causing all arguments to be printed in hexadecimal. This option is useful if the decoding is not trusted, or if the actual numeric value of an argument is needed. See also -X raw option.

-e read=set

-e reads=set

-e r=set

--read=set Performs a full hexadecimal and ASCII dump of all the data read from file de?

scriptors listed in the specified set. For example, to see all input activity on file descriptors 3 and 5 use --read=3,5. Note that this is independent from the normal tracing of the read(2) system call that is controlled by the option --trace=read.

-e write=set

-e writes=set

-e w=set

--write=set Performs a full hexadecimal and ASCII dump of all the data written to file de?

scriptors listed in the specified set. For example, to see all output activity on file descriptors 3 and 5 use --write=3,5. Note that this is independent from the normal tracing of the write(2) system call that is controlled by the option --trace=write.

-e quiet=set

-e silent=set

-e silence=set

-e q=set

--quiet=set

--silent=set

--silence=set

Suppresses various information messages. The default is --quiet=none. set can include the following elements:

attach Suppresses messages about attaching and detaching ("[Process NNNN attached]", "[Process NNNN detached]").

exit Suppress messages about process exits ("+++ exited with SSS +++").

path-resolution Suppress messages about resolution of paths provided via the -P option ("Requested path "..." resolved into "..."").

personality Suppress messages about process personality changes ("[Process PID=NNNN runs in PPP mode.]").

thread-execve

superseded Suppress messages about process being superseded by execve(2)

in another thread ("+++ superseded by execve in pid NNNN +++").

-e decode-fds=set

--decode-fds=set

Decodes various information associated with file descriptors. The default is --decode-fds=none. set can include the following elements:

path Prints file paths. Also enables printing of tracee's current working directory when AT_FDCWD constant is used.

socket Prints socket protocol-specific information.

dev Prints character/block device numbers.

eventfd Prints eventfd object details associated with eventfd file descriptors.

pidfd Prints PIDs associated with pidfd file descriptors.

signalfd Prints signal masks associated with signalfd file descriptors.

-e decode-pids=set

--decode-pids=set

Decodes various information associated with process IDs (and also thread IDs, process group IDs, and session IDs). The default is --decode-pids=none. set can include the following elements:

comm Prints command names associated with thread or process IDs.

pidns Prints thread, process, process group, and session IDs in strace's PID namespace if the tracee is in a different PID namespace.

-e kvm=vcpu|vcpu+

--kvm=vcpu|vcpu+

Prints the exit reason of kvm vcpu (=vcpu), or prints the whole kvm_run struct including the reason (=vcpu+). Requires Linux kernel version 4.16.0 or higher.

The vcpu+ output uses prefixes VCPU:CPU#< and VCPU:CPU#>. Lines starting with the former indicate the state of the structure before calling KVM_RUN, and those starting with the latter show the state after calling KVM_RUN.

-e namespace=new

--namespace=new

Prints the new namespaces entered by the tracee. The following system calls are supported: clone(2), clone3(2), setns(2), and unshare(2).

-i

--instruction-pointer

Prints the instruction pointer at the time of the system call.

-n

--syscall-number

Prints the system call number.

-N

--arg-names Prints the system call argument names.

-k

--stack-trace[=symbol]

Prints the execution stack trace of the traced processes after each system call.

--stack-trace-frame-limit=limit

Prints no more than this amount of stack trace frames when backtracing a system call (the default is 256). Use this option with the --stack-trace (or -k) op? tion.

-o filename

--output=filename

Writes the trace output to the file filename rather than to stderr. file? name.pid form is used if -ff option is supplied. If the argument begins with '|' or '!', 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. Piping output to a command is not currently compatible with the -ff option.

-A

--output-append-mode

Opens the file provided in the -o option in append mode.

--color=when

Colorize the trace output. The default is --color=auto, which enables color only when the trace output stream is a terminal (by default stderr; see -o) that supports at least 8 colors. If the NO_COLOR environment variable is set, color is disabled unless --color=always is specified. The when parameter can be one of auto, always, or never. Custom color schemes can be specified with the STRACE_COLORS environment variable, for example: STRACE_COLORS="syscall=1;33:argname=39:argval=35:const=1;36:com? ment=36:punct=0:retval=32:error=31"

The format of this value is key=value pairs separated by : with semicolon-sepa? rated SGR codes as values.

The default palette uses standard ANSI colors (SGR 30?37) and bold attributes rather than bright colors (SGR 90?97), so that terminal color themes can adapt them for both dark and light backgrounds.

Recognized color keys are: syscall, argname, argval, const, comment, punct, ret? val, error.

-q

--quiet

--quiet=attach,personality

Suppresses messages about attaching, detaching, and personality changes. This happens automatically when output is redirected to a file and the command is run directly instead of attaching.

-qq

--quiet=attach,personality,exit

Suppresses messages about attaching, detaching, personality changes, and process exit status.

-qqq

--quiet=all Suppresses all suppressible messages (please refer to the --quiet option de? scription for the full list of suppressible messages).

-r

--relative-timestamps[=precision]

Prints a relative timestamp upon entry to each system call. This records the time difference between the beginning of successive system calls. precision can be one of s (for seconds), ms (milliseconds), us (microseconds), or ns (nanosec? onds), and allows setting the precision of time value being printed. Default is us (microseconds). Note that because the -r option uses the monotonic clock, its measurements may differ from the time differences reported by the -t option, which uses the wall clock.

-s strsize

--string-limit=strsize

Specifies the maximum string size to print (the default is 32). Note that file? names are not considered strings and are always printed in full.

--absolute-timestamps[=[[format:]format],[[precision:]precision]]

--timestamps[=[[format:]format],[[precision:]precision]]

Prefixes each line of the trace with the wall clock time in the specified format with the specified precision. format can be one of the following:

none No time stamp is printed. Can be used to override the previous setting.

time Wall clock time (strftime(3) format string is %T).

unix Number of seconds since the epoch (strftime(3) format string is %s).

precision can be one of s (for seconds), ms (milliseconds), us (microseconds), or ns (nanoseconds). Default arguments for the option are format:time,preci? sion:s.

-t

--absolute-timestamps

Prefixes each line of the trace with the wall clock time.

-tt

--absolute-timestamps=precision:us

Prints the wall clock time with microsecond precision.

-ttt

--absolute-timestamps=format:unix,precision:us

Prints the wall clock time as seconds since the epoch, with microsecond preci? sion.

-T

--syscall-times[=precision]

Shows the time spent in system calls. This records the time difference between the beginning and the end of each system call. precision can be one of s (for seconds), ms (milliseconds), us (microseconds), or ns (nanoseconds), and allows setting the precision of time value being printed. Default is us (microsec? onds).

-v

--no-abbrev Prints unabbreviated versions of environment, stat, termios, etc. calls. These structures are very common, so the default behavior is to display a reasonable subset of their members. Use this option to see all members in full detail.

--strings-in-hex[=option]

Controls the use of hexadecimal escape sequences when printing strings. This option alters the default escaping behavior.

Normally (when neither this option nor -x is used), strace introduces escape se? quences in two situations: to represent non-printable and non-ASCII characters (i.e., those with character codes less than 32 or greater than 127), or to dis? ambiguate output, for example, by escaping the quotation marks that enclose a string or the angle brackets used in file descriptor paths. When a character must be escaped, strace prioritizes symbolic C-standard sequences if one exists: ?\t? (tab), ?\n? (newline), ?\v? (vertical tab), ?\f? (form feed), and ?\r? (carriage return). For all other characters that require escaping, strace de? faults to using an octal representation of the character's byte value. This op? tion allows you to override this default behavior and use hexadecimal escapes instead of octal ones.

option can be one of the following:

none Hexadecimal numbers are not used in the output at all. When there is a need to emit an escape sequence, octal numbers are used.

non-ascii-chars Hexadecimal numbers are used instead of octal in the escape se? quences.

non-ascii Strings that contain non-ASCII characters are printed using es? cape sequences with hexadecimal numbers.

all All strings are printed using escape sequences with hexadecimal numbers.

When the option is supplied without an argument, all is assumed.

-x

--strings-in-hex=non-ascii

Prints all non-ASCII strings in hexadecimal string format.

-xx

--strings-in-hex[=all]

Prints all strings in hexadecimal string format.

-X format

--const-print-style=format

Sets the format for printing of named constants and flags. Supported format values are:

raw Raw number output, without decoding.

abbrev Outputs a named constant or a set of flags instead of the raw number if they are found. This is the default strace behaviour.

verbose Outputs both the raw value and the decoded string (as a comment).

-y

--decode-fds

--decode-fds=path

Prints paths associated with file descriptor arguments and with the AT_FDCWD constant.

-yy

--decode-fds=all

Prints all available information associated with file descriptors: protocol-specific information associated with socket file descriptors, block/character device number associated with device file descriptors, and PIDs associated with pidfd file descriptors.

--pidns-translation

--decode-pids=pidns

If strace and tracee are in different PID namespaces, print PIDs in strace's namespace, too.

-Y

--decode-pids=comm

Prints command names for PIDs.

--always-show-pid

Shows PID prefix also for the process started by strace. Implied when -f and -o are both specified.

Statistics

-c

--summary-only

Counts time, calls, and errors for each system call and report a summary on program exit, suppressing the regular output. This shows system time (CPU time spent in the kernel), which is independent of wall clock time. If -c is used

with -f, only aggregate totals for all traced processes are kept.

-C

--summary Like -c, but also prints the regular output while processes are running.

-O overhead

--summary-syscall-overhead=overhead

Sets the overhead for tracing system calls to overhead. This is useful for overriding the default heuristic, which estimates the time spent in the measure? ment process itself when timing system calls with 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 pro? duced using -c.

The format of overhead specification is described in section Time specification format description.

-S sortby

--summary-sort-by=sortby

Sorts the output of the histogram printed by the -c option by the specified cri? terion. Valid values are time (or time-percent or time-total or total-time), min-time (or shortest or time-min), max-time (or longest or time-max), avg-time (or time-avg), calls (or count), errors (or error), name (or syscall or syscall-name), and nothing (or none); default is time.

-U columns

--summary-columns=columns

Configures the set and order of columns shown in the call summary. The columns argument is a comma-separated list containing one or more of the following val? ues:

time-percent (or time) Percentage of cumulative time consumed by a specific system call.

total-time (or time-total) Total system (or wall clock, if -w option is provided) time consumed by a specific system call.

min-time (or shortest or time-min) Minimum observed call duration.

max-time (or longest or time-max) Maximum observed call duration.

avg-time (or time-avg) Average call duration.

calls (or count) Call count.

errors (or error) Error count.

name (or syscall or syscall-name) System call name.

The default value is time-percent,total-time,avg-time,calls,errors,name. If the name field is not supplied explicitly, it is added as the last column.

-w

--summary-wall-clock

Summarizes the wall clock time for each system call, measured from its beginning to its end. The default is to summarize the system time.

Tampering

--inject=syscall_set[:error=errno|:retval=value][:signal=sig][:syscall=syscall][:delay_en?ter=delay][:delay_exit=delay][:poke_enter=@argN=DATAN,@argM=DATAM...][:poke_exit=@argN=DATAN,@argM=DATAM...][:when=expr]

Performs system call tampering for the specified set of system calls.

The syntax of the syscall_set specification is the same as in the --trace op?tion.

At least one of error, retval, signal, delay_enter, delay_exit, poke_enter, or poke_exit action options must be specified. error and retval are mutually ex?clusive.

If the error=errno option is specified, a fault is injected into the system call. This is achieved by replacing the system call number with -1 (represent?ing an invalid system call) and setting the error code to the specified errno.

This behavior of replacing the syscall number with -1 can be overridden using the syscall= option. The errno can be a symbolic name like ENOSYS or a numeric value in the range 1..4095.

If the retval=value option is specified, a success value is injected. The sys?tem call number is replaced as with the error= option, but instead of an error, the specified success value is returned to the caller process.

If the signal=sig option is specified with either a symbolic value like SIGSEGV or a numeric value within 1..SIGRTMAX range, that signal is delivered on enter?ing every system call specified by the syscall_set.

If the delay_enter=delay or delay_exit=delay options are specified, delay injec?tion is performed: the tracee is delayed by time period specified by delay on

entering or exiting the system call, respectively. The format of delay specifi?
cation is described in section Time specification format description.

If the poke_enter=@argN=DATAN,@argM=DATAM... or poke_exit=@argN=DATAN,@argM=DATAM... options are specified, tracee's memory at locations, pointed to by system call arguments argN and argM (going from arg1 to arg7) is overwritten by data DATAN and DATAM (specified in hexadecimal format; for example poke_enter=@arg1=0000DEAD0000BEEF). The poke_enter option modifies memory on system call enter, while poke_exit does so on system call exit.

The injection actions are independent. For example, specifying only signal= de?
livers a signal without altering the system call's outcome or delaying it. Sim?
ilarly, specifying only error= injects a system call fault without adding a sig?
nal or delay.

If the signal=sig option is specified together with error=errno or retval=value, then both injection of a fault or success and signal delivery are performed.

If the syscall=syscall option is specified, the given syscall is injected in?
stead of the default -1. The specified syscall must have no side effects; cur?
rently, only system calls from the %pure set are supported.

Unless the when=expr subexpression is specified, an injection is being made into every invocation of each system call from the syscall_set.

The format of the subexpression is:

first[..last][+[step]]

Number first stands for the first invocation number in the range, number last stands for the last invocation number in the range, and step stands for the step between two consecutive invocations. The following combinations are useful:

first Injects into invocation number first only for each system call in the syscall_set.

first..last Injects into invocations from first through last (inclusive) for each system call in the syscall_set.

first+ Injects into every invocation, starting with number first, for each system call in the syscall_set.

first+step Injects into invocations number first, first+step, first+step+step, and so on, for each system call in the syscall_set.

first..last+step Same as the previous, but consider only invocations with num?
bers up to last (inclusive).

For example, to fail each third and subsequent chdir system calls with ENOENT, use --inject=chdir:error=ENOENT:when=3+.

The valid range for numbers first and step is 1..65535, and for number last is 1..65534.

An injection expression can contain at most one fault or return value specifica? tion (i.e., either error= or retval=) and at most one signal= specification. If an injection expression contains multiple when= specifications, the last one takes precedence.

Accounting of system calls that are subject to injection is done per system call and per tracee.

Specification of system call injection can be combined with other system call filtering options, for example, -P /dev/urandom --inject=file:error=ENOENT.

-e inject=args

This is equivalent to --inject=args.

--fault=syscall_set[:error=errno][:when=expr]

Performs system call fault injection for the specified set of system calls.

This is a shortcut for the more general --inject= option, using a default errno of ENOSYS.

-e fault=args

This is equivalent to --fault=args.

Miscellaneous

-d

--debug Shows some debugging output of strace itself on the standard error.

-F This option is deprecated. It is retained for backward compatibility only and may be removed in future releases. Using multiple -F options is equivalent to a single -f. This option is ignored entirely if used in conjunction with one or more -f options.

-h

--help Prints the help summary.

--seccomp-bpf

Attempts to use seccomp-bpf (see seccomp(2)) to cause the kernel to stop the

tracee only for the system calls that are being traced.

This option has no effect unless -f/--follow-forks is also specified. --sec?

comp-bpf is not compatible with --syscall-limit and -b/--detach-on options. It

is also not applicable to processes attached using -p/--attach option.

An attempt to enable system calls filtering using seccomp-bpf may fail for vari?

ous reasons, e.g. there are too many system calls to filter, the seccomp API is

not available, or strace itself is being traced. If the seccomp-bpf filter

setup fails, strace proceeds as usual, stopping traced processes on every system

call.

When --seccomp-bpf is activated and -p/--attach option is not used,

--kill-on-exit option is activated as well.

Note that in cases when the tracee has another seccomp filter that returns an

action value with a precedence greater than SECCOMP_RET_TRACE, strace --sec?

comp-bpf will not be notified. That is, if another seccomp filter, for example,

disables the system call or kills the tracee, then strace --seccomp-bpf will not

be aware of that system call invocation at all.

--tips[=[[id:]id],[[format:]format]]

Shows strace tips, tricks, and tweaks before exit. The id can be a non-negative

integer to print a specific tip (note: these IDs are not guaranteed to be sta?

ble). It can also be random (the default), in which case a random tip is

printed. format can be one of the following:

none No tip is printed. Can be used to override the previous setting.

compact Prints the tip just big enough to contain all the text.

full Prints the tip in its full glory.

Default is id:random,format:compact.

-V

--version Prints the version number of strace and the list of enabled optional features.

Multiple instances of this option beyond specific threshold tend to increase der

Strauss awareness.

Time specification format description

Time values are specified as a decimal floating point number (in a format accepted by str?

tod(3)), optionally followed by a suffix to indicate the unit of time: s (seconds), ms (mil?

liseconds), us (microseconds), or ns (nanoseconds). If no suffix is specified, the value

defaults to microseconds.

The described format is used for -O, --inject=delay_enter, and --inject=delay_exit options.

DIAGNOSTICS

When command exits, strace exits with the same exit status. If command is terminated by a signal, strace terminates itself with the same signal, so that strace can be used as a wrap? per process transparent to the invoking parent process. Note that the parent-child rela? tionship (signal stop notifications, the getppid(2) value, etc) between the traced process and its parent is not preserved unless -D is used.

When using -p without a command, the exit status of strace is zero unless no processes have been attached or an unexpected error occurred during tracing.

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 these capabilities should only be granted to users with full root privileges, installing strace as setuid to root is only ap? propriate when its use is restricted to such trusted users. For example, a special version of strace could be installed with mode 'rwsr-x---', user root, and group trace. In this configuration, only trusted users who are members of the trace group could execute it. If you use this feature, remember to also install a regular, non-setuid version of strace for ordinary users.

MULTIPLE PERSONALITIES SUPPORT

On some architectures, strace can decode system calls for processes that use a different Ap? plication Binary Interface (ABI) from the one strace uses. Specifically, in addition to de? coding native ABI, strace can decode the following ABIs on the following architectures:

??

? Architecture ? ABIs supported ?

??

? x86_64 ? i386, x32 [1]; i386 [2] ?

??

? AArch64 ? ARM 32-bit EABI ?

??

? PowerPC 64-bit [3] ? PowerPC 32-bit ?

??

? s390x ? s390 ?

??

? SPARC 64-bit ? SPARC 32-bit ?

??

? TILE 64-bit ? TILE 32-bit ?

??

[1] When strace is built as an x86_64 application

[2] When strace is built as an x32 application

[3] Big endian only

This support is optional and depends on the ability to generate and parse structure defini? tions at build time. Refer to the output of the strace -V command to determine which ABIs are supported by your strace build. In this context, "non-native" refers to an ABI that differs from the one strace is using:

m32-mpers strace can trace and properly decode non-native 32-bit binaries.

no-m32-mpers strace can trace, but cannot properly decode non-native 32-bit binaries.

mx32-mpers strace can trace and properly decode non-native 32-on-64-bit binaries.

no-mx32-mpers strace can trace, but cannot properly decode non-native 32-on-64-bit bina? ries.

If the output contains neither m32-mpers nor no-m32-mpers, it means that support for decod? ing non-native 32-bit binaries is not applicable to the architecture.

Likewise, if the output contains neither mx32-mpers nor no-mx32-mpers, it means that support for decoding non-native 32-on-64-bit binaries is not applicable to the architecture.

NOTES

Systems that use shared libraries often produce a large amount of tracing output when load? ing them.

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-pro? tected, 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, the underlying faccessat(2) system call does not have a flags argument, and the setrlimit(2) library function is implemented using prlimit64(2) system call on mod? ern (2.6.38+) kernels. These discrepancies are normal characteristics of the system call

interface and are handled by C library wrapper functions.

Some system calls have different names in different architectures and personalities. In these cases, system call filtering and printing uses the names that match corresponding __NR_* kernel macros of the tracee's architecture and personality. There are two exceptions from this general rule: arm_fadvise64_64(2) ARM system call and xtensa_fadvise64_64(2) Xtensa system call are filtered and printed as fadvise64_64(2).

On the x32 ABI, some system calls are intended for 64-bit processes but can be invoked from x32 by setting the __X32_SYSCALL_BIT flag. When this occurs, strace designates these calls with a #64 suffix. An example is readv(2), which is syscall number 19 on x86_64, whereas its distinct x32 counterpart is syscall number 515.

On some platforms, a process attached with the -p option may receive a spurious EINTR error from a non-restartable system call. This can have an unpredictable effect on the process if it does not attempt to restart the call. Ideally, all system calls should be restarted on strace attach, making the attach invisible to the traced process, but a few system calls aren't. Arguably, every instance of such behavior is a kernel bug.

Since strace executes the specified command directly without a shell, scripts that lack a shebang line (e.g., #!/bin/sh) will fail with an ENOEXEC error, even if a shell could run them correctly. It is advisable to manually supply a shell as a command with the script as its argument.

BUGS

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

A traced process runs more slowly than a non-traced one. The performance impact can be mit? igated by using the --seccomp-bpf option.

When tracing a command, its descendant processes may be left running after strace is termi? nated by an interrupt signal (such as CTRL-C). This can be prevented by using the --kill-on-exit option, or by using --seccomp-bpf option in a way that implies --kill-on-exit.

A traced process can use the CLONE_UNTRACED flag with the clone system call to create a child process that is not traced by strace. This breaks a guarantee of the --seccomp-bpf option, as this untraced child may be left with an active seccomp filter after strace termi? nates.

HISTORY

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 took on the project. He merged strace 2.5 for SunOS with the second release of strace for Linux, added many features from SVR4's truss(1), and produced a ver? sion of strace that worked on both platforms. In 1994 Rick ported strace to SVR4 and So? laris and wrote the automatic configuration support. In 1995 he ported strace to Irix (and became tired of writing about himself in the third person).

Beginning with 1996, strace was maintained by Wichert Akkerman. During his tenure, strace development migrated to CVS; ports to FreeBSD and many architectures on Linux (including ARM, IA-64, MIPS, PA-RISC, PowerPC, s390, SPARC) were introduced.

In 2002, responsibility for strace maintenance was transferred to Roland McGrath. Since then, strace gained support for several new Linux architectures (AMD64, s390x, SuperH), bi-architecture support for some of them, and received numerous additions and improvements in system calls decoders on Linux; strace development migrated to Git during that period.

Since 2009, strace has been actively maintained by Dmitry Levin. During this period, strace has gained support for the AArch64, ARC, AVR32, Blackfin, C-SKY, LoongArch, Meta, Nios II, OpenRISC 1000, RISC-V, Tile/TileGx, and Xtensa architectures. In 2012, unmaintained and ap? parently broken support for non-Linux operating systems was removed. Also, in 2012 strace gained support for path tracing and file descriptor path decoding. In 2014, support for stack trace printing was added. In 2016, system call tampering was implemented.

For the additional information, please refer to the NEWS file and strace repository commit log.

REPORTING BUGS

Problems with strace should be reported to the strace mailing list.

SEE ALSO

strace-log-merge(1), ltrace(1), perf-trace(1), trace-cmd(1), time(1), ptrace(2), seccomp(2), syscall(2), proc(5), signal(7)

strace Home Page

AUTHORS

The complete list of strace contributors can be found in the CREDITS file.