Rocky Enterprise Linux 9.2 Manual Pages on command 'strace.1'

C:\>man strace.1

STRACE(1) General Commands Manual STRACE(1)

NAME

strace - trace system calls and signals

SYNOPSIS

strace [-ACdffhikqqrtttTvVwxxyyzZ] [-I n] [-b execve] [-e expr]... [-a column]
[-o file] [-s strsize] [-X format] [-P path]... [-p pid]... [--seccomp-bpf]
{ -p pid | [-DDD] [-E var[=val]]... [-u username] command [args] }

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

DESCRIPTION

In the simplest case strace runs the specified command until it exits. It inter? cepts 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 adminis? trators, 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 ap? pended.

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

Signals are printed as signal symbol and decoded siginfo structure. An excerpt from stracing 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 and meanwhile another one is being called from a different thread/process then strace will try to preserve the order of those events and mark the ongoing call as being 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, {1130322148, 939977000}) = 0
[pid 28772] <... select resumed> )      = 1 (in [3])
```

Interruption of a (restartable) system call by a signal delivery is processed dif? ferently as kernel terminates the system call and also arranges its immediate reex? ecution after the signal handler completes.

```
read(0, 0x7ffff72cf5cf, 1)              = ? ERESTARTSYS (To be restarted)
--- SIGALRM ... ---
rt_sigreturn(0xe)                       = 0
read(0, "", 1)                          = 0
```

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

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

Here, the third argument of open(2) is decoded by breaking down the flag argument

into its three bitwise-OR constituents and printing the mode value in octal by tra?
dition. Where the traditional or native usage differs from ANSI or POSIX, the lat?
ter forms are preferred. In some cases, strace output is proven to be more read?
able than the source.

Structure pointers are dereferenced and the 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 dis?
played 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 ex?
ample that 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 sys?
tem 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.

Syscalls unknown to strace are printed raw, with the unknown system call number
printed in hexadecimal form and 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 charac?
ters 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(3) 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(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 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

General

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

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

where qualifier is one of trace, abbrev, verbose, raw, signal, read, write, fault, inject, status, or kvm, and value is a qualifier-depen? dent symbol or number. The default qualifier is trace. Using an ex? clamation mark negates the set of values. For example, -e open means literally -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 have the obvi? ous 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.

Startup

-E var=val

--env=var=val

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

-E var

--env=var Remove var from the inherited list of environment variables before passing it on to the command.

-p pid

--attach=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 many processes in addition to command (which is optional if at least one -p option is given). -p "`pidof PROG`" syntax is supported.

-u username

--user=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 effec? tive privileges.

Tracing

-b syscall

--detach-on=syscall

If specified syscall is reached, detach from traced process. Cur? rently, only execve(2) syscall is supported. This option is useful if you want to trace multi-threaded process and therefore require -f, but don't want to trace its (potentially very complex) children.

-D Run tracer process as a grandchild, not as the parent of the tracee. This reduces the visible effect of strace by keeping the tracee a di? rect child of the calling process.

-DD Run tracer process as tracee's grandchild in a separate process group. In addition to reduction of the visible effect of strace, it also avoids killing of strace with kill(2) issued to the whole process group.

-DDD Run tracer process as tracee's grandchild in a separate session ("true daemonisation"). In addition to reduction of the visible effect of strace, it also avoids killing of strace upon session termination.

-f Trace 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 -p PID -f will attach all threads of process PID if it is multi-threaded, not only thread with thread_id = PID.

-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. This is incompatible with -c, since no per-process counts are kept. One might want to consider using strace-log-merge(1) to obtain a com? bined strace log view.

-I interruptible

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

1 no signals are blocked;

2 fatal signals are blocked while decoding syscall (default);

3 fatal signals are always blocked (default if -o FILE PROG);

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

Filtering

-e trace=syscall_set

--trace=syscall_set

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

syscall Trace specific syscall, specified by its name (but see NOTES).

?value Question mark before the syscall qualification allows sup? pression of error in case no syscalls matched the qualifi? cation provided.

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

syscall@64 Trace syscall only for the 64-bit personality.

syscall@32 Trace syscall only for the 32-bit personality.

syscall@x32 Trace syscall only for the 32-on-64-bit personality.

%file

file Trace all system calls which take a file name as an argu? ment. 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. Further?

more, using the abbreviation will ensure that you don't accidentally forget to include a call like lstat(2) in the list. Betchya woulda forgot that one. The syntax without a preceding percent sign ("-e trace=file") is deprecated.

%process

process Trace all system calls which involve process management. This is useful for watching the fork, wait, and exec steps of a process. The syntax without a preceding percent sign ("-e trace=process") is deprecated.

%net

%network

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

%signal

signal Trace all signal related system calls. The syntax without a preceding percent sign ("-e trace=signal") is depre? cated.

%ipc

ipc Trace all IPC related system calls. The syntax without a preceding percent sign ("-e trace=ipc") is deprecated.

%desc

desc Trace all file descriptor related system calls. The syn? tax without a preceding percent sign ("-e trace=desc") is deprecated.

%memory

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

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

%stat Trace stat syscall variants.

%lstat Trace lstat syscall variants.

%fstat Trace fstat and fstatat syscall variants.

%%stat Trace syscalls used for requesting file status (stat, lstat, fstat, fstatat, statx, and their variants).

%statfs Trace statfs, statfs64, statvfs, osf_statfs, and osf_statfs64 system calls. The same effect can be achieved with -e trace=/^(.*_)?statv?fs regular expres? sion.

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

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

%pure Trace syscalls that always succeed and have no arguments. Currently, this list includes arc_gettls(2), getdtable? size(2), getegid(2), getegid32(2), geteuid(2), ge? teuid32(2), getgid(2), getgid32(2), getpagesize(2), getp? grp(2), getpid(2), getppid(2), get_thread_area(2) (on ar? chitectures other than x86), gettid(2), get_tls(2), ge? tuid(2), getuid32(2), getxgid(2), getxpid(2), getxuid(2), kern_features(2), and metag_get_tls(2) syscalls.

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 be? ing monitored. The default is trace=all.

-e signal=set

--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 status=set

--status=set

Print only system calls with the specified return status. The default is status=all. When using the status qualifier, because strace waits for system calls to return before deciding whether they should be printed or not, the traditional order of events may not be preserved anymore. If two system calls are executed 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 Trace system calls that returned without an error code. The -z option has the effect of status=successful.

failed Trace system calls that returned with an error code. The -Z option has the effect of status=failed.

unfinished Trace system calls that did not return. This might hap? pen, for example, due to an execve call in a neighbour thread.

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

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

-P path

--trace-path=path

Trace only system calls accessing path. Multiple -P options can be used to specify several paths.

-z Print only syscalls that returned without an error code.

-Z Print only syscalls that returned with an error code.

Output format

-a column

--columns=column

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

-e abbrev=syscall_set

--abbrev=syscall_set

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

-e verbose=syscall_set

--verbose=syscall_set

Dereference structures for the specified set of system calls. The syn? tax of the syscall_set specification is the same as in the -e trace op? tion. The default is verbose=all.

-e raw=syscall_set

--raw=syscall_set

Print raw, undecoded arguments for the specified set of system calls. The syntax of the syscall_set specification is the same as in the -e trace option. 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 argu? ment. See also -X raw option.

-e read=set

--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

--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.

-e kvm=vcpu

--kvm=vcpu Print the exit reason of kvm vcpu. Requires Linux kernel version 4.16.0 or higher.

-i

--instruction-pointer

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

-k

--stack-traces

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

-o filename

--output=filename

Write the trace output to the file filename rather than to stderr. filename.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. The latter is not compatible with -ff option currently.

-A

--output-append-mode

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

-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.

-qq If given twice, suppress messages about process exit status.

-r Print a relative timestamp upon entry to each system call. This records the time difference between the beginning of successive system calls. Note that since -r option uses the monotonic clock time for measuring time difference and not the wall clock time, its measurements can differ from the difference in time reported by the -t option.

-s strsize

--string-limit=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.

-t Prefix each line of the trace with the wall clock time.

-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

--no-abbrev 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.

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

-xx Print all strings in hexadecimal string format.

-X format

--const-print-style=format

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

raw Raw number output, without decoding.

abbrev Output a named constant or a set of flags instead of the raw number if they are found. This is the default strace behav? iour.

verbose Output both the raw value and the decoded string (as a com? ment).

-y Print paths associated with file descriptor arguments.

-yy Print protocol specific information associated with socket file de? scriptors, and block/character device number associated with device file descriptors.

Statistics

-c

--summary-only

Count time, calls, and errors for each system call and report a summary on program exit, suppressing the regular output. This attempts to show system time (CPU time spent running in the kernel) 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 print regular output while processes are running.

-O overhead Set the overhead for tracing system calls to overhead. 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 sys? tem call time to the total produced using -c.

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

-S sortby

--summary-sort-by=sortby

Sort the output of the histogram printed by the -c option by the speci? fied criterion. Legal values are time (or time_total or total_time), calls (or count), errors (or error), name (or syscall or syscall_name), and nothing (or none); default is time.

-w

--summary-wall-clock

Summarise the time difference between the beginning and end of each system call. The default is to summarise the system time.

Tampering

-e inject=syscall_set[:error=errno|:retval=value][:sig? nal=sig][:syscall=syscall][:delay_enter=delay][:delay_exit=delay][:when=expr]

--inject=syscall_set[:error=errno|:retval=value][:sig? nal=sig][:syscall=syscall][:delay_enter=delay][:delay_exit=delay][:when=expr]

Perform syscall tampering for the specified set of syscalls. The syn? tax of the syscall_set specification is the same as in the -e trace op?

tion.

At least one of error, retval, signal, delay_enter, or delay_exit op?
tions has to be specified. error and retval are mutually exclusive.
If :error=errno option is specified, a fault is injected into a syscall
invocation: the syscall number is replaced by -1 which corresponds to
an invalid syscall (unless a syscall is specified with :syscall= op?
tion), and the error code is specified using a symbolic errno value
like ENOSYS or a numeric value within 1..4095 range.
If :retval=value option is specified, success injection is performed:
the syscall number is replaced by -1, but a bogus success value is re?
turned to the callee.
If :signal=sig option is specified with either a symbolic value like
SIGSEGV or a numeric value within 1..SIGRTMAX range, that signal is de?
livered on entering every syscall specified by the set.
If :delay_enter=delay or :delay_exit=delay options are specified, delay
injection is performed: the tracee is delayed by time period specified
by delay on entering or exiting the syscall, respectively. The format
of delay specification is described in section Time specification for?
mat description.
If :signal=sig option is specified without :error=errno, :retval=value
or :delay_{enter,exit}=usecs options, then only a signal sig is deliv?
ered without a syscall fault or delay injection. Conversely, :er?
ror=errno or :retval=value option without :delay_enter=delay, :de?
lay_exit=delay or :signal=sig options injects a fault without deliver?
ing a signal or injecting a delay, etc.
If both :error=errno or :retval=value and :signal=sig options are spec?
ified, then both a fault or success is injected and a signal is deliv?
ered.
if :syscall=syscall option is specified, the corresponding syscall with
no side effects is injected instead of -1. Currently, only "pure" (see
-e trace=%pure description) syscalls can be specified there.
Unless a :when=expr subexpression is specified, an injection is being
made into every invocation of each syscall from the set.

The format of the subexpression is one of the following:

first For every syscall from the set, perform an injection for the syscall invocation number first only.

first+ For every syscall from the set, perform injections for the syscall invocation number first and all subsequent invoca? tions.

first+step For every syscall from the set, perform injections for syscall invocations number first, first+step, first+step+step, and so on.

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

The valid range for numbers first and step is 1..65535.

An injection expression can contain only one error= or retval= specifi? cation, and only one signal= specification. If an injection expression contains multiple when= specifications, the last one takes precedence.

Accounting of syscalls that are subject to injection is done per syscall and per tracee.

Specification of syscall injection can be combined with other syscall filtering options, for example, -P /dev/urandom -e inject=file:er? ror=ENOENT.

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

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

Perform syscall fault injection for the specified set of syscalls.

This is equivalent to more generic -e inject= expression with default value of errno option set to ENOSYS.

Miscellaneous

-d

--debug Show 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. Usage of multiple in? stances of -F option is still equivalent to a single -f, and it is ig? nored at all if used along with one or more instances of -f option.

-h

--help Print the help summary.

--seccomp-bpf

Enable (experimental) usage of seccomp-bpf (see seccomp(2)) to have ptrace(2)-stops only when system calls that are being traced occur in the traced processes. Implies the -f option. An attempt to rely on seccomp-bpf to filter system calls may fail for various reasons, e.g. there are too many system calls to filter, the seccomp API is not available, or strace itself is being traced. --seccomp-bpf is also in? effective on processes attached using -p. In cases when seccomp-bpf filter setup failed, strace proceeds as usual and stops traced pro? cesses on every system call.

-V

--version Print the version number of strace.

Time specification format description

Time values can be specified as a decimal floating point number (in a format ac? cepted by strtod(3)), optionally followed by one of the following suffices that specify the unit of time: s (seconds), ms (milliseconds), us (microseconds), or ns (nanoseconds). If no suffix is specified, the value is interpreted as microsec? onds.

The described format is used for -O, -e inject=delay_enter, and -e inject=de? lay_exit options.

DIAGNOSTICS

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

When using -p without a command, the exit status of strace is zero unless no pro? cesses has been attached or there was an unexpected error in doing the 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 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 fea? ture, please remember to install a regular non-setuid version of strace for ordi? nary users to use.

MULTIPLE PERSONALITIES SUPPORT

On some architectures, strace supports decoding of syscalls for processes that use different ABI rather than the one strace uses. Specifically, in addition to decod? ing native ABI, strace can decode the following ABIs on the following architec? tures:

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 relies on ability to generate and parse structure def? initions during the build time. Please refer to the output of the strace -V com? mand in order to figure out what support is available in your strace build ("non-

native" refers to an ABI that differs from the ABI strace has):

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 bina?
ries.

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

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

If the output contains neither m32-mpers nor no-m32-mpers, then decoding of non-na?
tive 32-bit binaries is not implemented at all or not applicable.

Likewise, if the output contains neither mx32-mpers nor no-mx32-mpers, then decod?
ing of non-native 32-on-64-bit binaries is not implemented at all or not applica?
ble.

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 ad?
dress-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, the faccessat(2) system call does not have flags ar?
gument, and the setrlimit(2) library function uses prlimit64(2) system call on mod?
ern (2.6.38+) kernels. These discrepancies are normal but idiosyncratic character?
istics of the system call interface and are accounted for by C library wrapper
functions.

Some system calls have different names in different architectures and personali?
ties. 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 syscall
and xtensa_fadvise64_64(2) Xtensa syscall are filtered and printed as fad?
vise64_64(2).

On x32, syscalls that are intended to be used by 64-bit processes and not x32 ones

(for example, readv(2), that has syscall number 19 on x86_64, with its x32 counter? part has syscall number 515), but called with __X32_SYSCALL_BIT flag being set, are designated with #64 suffix.

On some platforms a process that is attached to with the -p option may observe a spurious EINTR return from the current system call that is not restartable. (Ide? ally, all system calls should be restarted on strace attach, making the attach in? visible to the traced process, but a few system calls aren't. Arguably, every in? stance of such behavior is a kernel bug.) This may have an unpredictable effect on the process if the process takes no action to restart the system call.

As strace executes the specified command directly and does not employ a shell for that, scripts without shebang that usually run just fine when invoked by shell fail to execute with ENOEXEC error. 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 be? ing traced.

A traced process runs slowly.

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

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 re? leased 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 re? lease 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.

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, the burden of strace maintainership 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 syscalls decoders on Linux; strace development migrated to git during that period. Since 2009, strace is actively maintained by Dmitry Levin. strace gained support for AArch64, ARC, AVR32, Blackfin, Meta, Nios II, OpenSISC 1000, RISC-V, Tile/TileGx, Xtensa architectures since that time. In 2012, unmaintained and apparently 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 traces printing was added. In 2016, syscall fault injection 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 ?mailto:strace-devel@lists.strace.io?.

SEE ALSO

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

strace Home Page ?https://strace.io/?

AUTHORS

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