Table of Contents

Strace little book

I like researching debugging techniques, so I decide to write this booklet to introduce strace. The following is the official definition of `strace`:

> strace is a diagnostic, debugging and instructional userspace utility for Linux. It is used to monitor and tamper with interactions between processes and the Linux kernel, which include system calls, signal deliveries, and changes of process state.
>
> 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.
>
> The operation of strace is made possible by the kernel feature known as ptrace.

In one word, `strace` helps you know a process's activities between user-space and kernel-space. Let's check a simple example to get first impression of `strace`:

```
# strace ls
execve("/usr/bin/ls", ["ls"], 0x7ffe7727eec0 /* 20 vars */) = 0
brk(NULL)                               = 0x560a32cda000
arch_prctl(0x3001 /* ARCH_??? */, 0x7fff167898f0) = -1 EINVAL (I
nvalid argument)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such fil
e or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=98317, ...}) = 0
mmap(NULL, 98317, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7face703c000
close(3)                                = 0
openat(AT_FDCWD, "/usr/lib/libcap.so.2", O_RDONLY|O_CLOEXEC) = 3
......
```

`strace` outputs all the syscalls' names, arguments, and return values. Very cool, isn't it? OK, let's begin our journey now.

Show help message

As other `*NIX` commands, `-h` option shows a concise help message of `strace`:

```
# strace -h
usage: strace [-CdffhiqrtttTvVwxxy] [-I n] [-e expr]...
              [-a column] [-o file] [-s strsize] [-P path]...
              -p pid... / [-D] [-E var=val]... [-u username] PRO
G [ARGS]
   or: strace -c[dfw] [-I n] [-e expr]... [-O overhead] [-S sort
by]
              -p pid... / [-D] [-E var=val]... [-u username] PRO
G [ARGS]

Output format:
  -a column      alignment COLUMN for printing syscall results (
default 40)
  -i             print instruction pointer at time of syscall
  -k             obtain stack trace between each syscall
......
```

`-V` shows version:

```
# strace -V
strace -- version 4.26
Copyright (c) 1991-2018 The strace developers <https://strace.io
>.
This is free software; see the source for copying conditions.  T
here is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICUL
AR PURPOSE.

Optional features enabled: stack-trace=libunwind stack-demangle
m32-mpers mx32-mpers
```

Redirect strace's output to file

You can use `-o` option to specify a file which saves `strace` 's output:

```
# strace -o log.txt ls
log.txt
# more log.txt
execve("/usr/bin/ls", ["ls"], 0x7ffe2e59b220 /* 20 vars */) = 0
brk(NULL)                               = 0x5599582f6000
arch_prctl(0x3001 /* ARCH_??? */, 0x7fffd38db780) = -1 EINVAL (I
nvalid argument)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such fil
e or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
......
```

Additionally, `-A` option can be used to append log to an existing file instead of truncate it:

```
# ls -lth log.txt
-rw-r--r-- 1 root root 3.6K Feb 28 10:07 log.txt
# strace -o log.txt ls
log.txt
# ls -lth log.txt
-rw-r--r-- 1 root root 3.6K Feb 28 10:08 log.txt
# strace -A -o log.txt ls
log.txt
# ls -lth log.txt
-rw-r--r-- 1 root root 7.2K Feb 28 10:08 log.txt
```

Attach to running processes

`strace` can attach and trace running processes. Check following simple example:

```
# cat dead_loop.c
#include <unistd.h>

int main(void)
{
        while (1)
        {
                sleep(1);
        }
        return 0;
}
```

Build and launch two processes:

```
# gcc dead_loop.c -o dead_loop
# ./dead_loop &
[1] 2194
# ./dead_loop &
[2] 2195
```

Use `-p` option to trace one process:

```
# strace -p 2194
strace: Process 2194 attached
restart_syscall(<... resuming interrupted nanosleep ...>) = 0
nanosleep({tv_sec=1, tv_nsec=0}, 0x7ffc037bf730) = 0
......
```

Use `-p` option to trace multiple processes:

```
# strace -p 2194,2195
strace: Process 2194 attached
strace: Process 2195 attached
[pid  2194] restart_syscall(<... resuming interrupted nanosleep
...> <unfinished ...>
[pid  2195] restart_syscall(<... resuming interrupted nanosleep
...> <unfinished ...>
[pid  2194] <... restart_syscall resumed> ) = 0
[pid  2194] nanosleep({tv_sec=1, tv_nsec=0},  <unfinished ...>
[pid  2195] <... restart_syscall resumed> ) = 0
......
```

`strace` can also utilize `pidof` command to trace processes:

```
# strace -p "`pidof dead_loop`"
strace: Process 2195 attached
strace: Process 2194 attached
[pid  2194] restart_syscall(<... resuming interrupted nanosleep
...> <unfinished ...>
[pid  2195] restart_syscall(<... resuming interrupted nanosleep
...>) = 0
[pid  2195] nanosleep({tv_sec=1, tv_nsec=0},  <unfinished ...>
[pid  2194] <... restart_syscall resumed> ) = 0
......
```

Or use `pgrep` :

```
# strace -p "`pgrep dead_loop`"
strace: Process 2194 attached
strace: Process 2195 attached
[pid  2195] restart_syscall(<... resuming interrupted nanosleep
...> <unfinished ...>
[pid  2194] restart_syscall(<... resuming interrupted nanosleep
...>) = 0
[pid  2194] nanosleep({tv_sec=1, tv_nsec=0},  <unfinished ...>
[pid  2195] <... restart_syscall resumed> ) = 0
[pid  2195] nanosleep({tv_sec=1, tv_nsec=0},  <unfinished ...>
......
```

Trace child processes

Check following code:

```
# cat fork.c
#include <sys/types.h>
#include <unistd.h>

int main(void)
{
        int pid = fork();
        if (pid < 0)
        {
                return 1;
        }
        else
        {
                while (1)
                {
                        sleep(1);
                }
        }
        return 0;
}
```

By default, `strace` won't trace child processes spawned by `fork`, `vfork` and `clone`:

```
# gcc fork.c -o fork
# strace ./fork
execve("./fork", ["./fork"], 0x7ffde8bab140 /* 21 vars */) = 0
brk(NULL)                                       = 0x556c0719c000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd59b05130) = -1 EINVAL (I
nvalid argument)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such fil
e or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
```

```
fstat(3, {st_mode=S_IFREG|0644, st_size=98317, ...}) = 0
mmap(NULL, 98317, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f045e508000
close(3)                                = 0
openat(AT_FDCWD, "/usr/lib/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0000C\2\0\
0\0\0\0"..., 832) = 832
lseek(3, 792, SEEK_SET)                 = 792
read(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\201\336\t\36\251c\324\23
3E\371SoK\5H\334"..., 68) = 68
fstat(3, {st_mode=S_IFREG|0755, st_size=2136840, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS
, -1, 0) = 0x7f045e506000
lseek(3, 792, SEEK_SET)                 = 792
read(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\201\336\t\36\251c\324\23
3E\371SoK\5H\334"..., 68) = 68
lseek(3, 864, SEEK_SET)                 = 864
read(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\
0\0\0\0\0", 32) = 32
mmap(NULL, 1848896, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0)
= 0x7f045e342000
mprotect(0x7f045e364000, 1671168, PROT_NONE) = 0
mmap(0x7f045e364000, 1355776, PROT_READ|PROT_EXEC, MAP_PRIVATE|M
AP_FIXED|MAP_DENYWRITE, 3, 0x22000) = 0x7f045e364000
mmap(0x7f045e4af000, 311296, PROT_READ, MAP_PRIVATE|MAP_FIXED|MA
P_DENYWRITE, 3, 0x16d000) = 0x7f045e4af000
mmap(0x7f045e4fc000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MA
P_FIXED|MAP_DENYWRITE, 3, 0x1b9000) = 0x7f045e4fc000
mmap(0x7f045e502000, 13888, PROT_READ|PROT_WRITE, MAP_PRIVATE|MA
P_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f045e502000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7f045e507500) = 0
mprotect(0x7f045e4fc000, 16384, PROT_READ) = 0
mprotect(0x556c0647d000, 4096, PROT_READ) = 0
mprotect(0x7f045e54a000, 4096, PROT_READ) = 0
munmap(0x7f045e508000, 98317)           = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_S
ETTID|SIGCHLD, child_tidptr=0x7f045e5077d0) = 5082
nanosleep({tv_sec=1, tv_nsec=0}, 0x7ffd59b050d0) = 0
nanosleep({tv_sec=1, tv_nsec=0}, 0x7ffd59b050d0) = 0
nanosleep({tv_sec=1, tv_nsec=0}, 0x7ffd59b050d0) = 0
```

```
nanosleep({tv_sec=1, tv_nsec=0}, ^Cstrace: Process 5081 detached
 <detached ...>
......
```

To trace child processes, `-f` option need to be specified:

```
# strace -f ./fork
execve("./fork", ["./fork"], 0x7ffcebee9288 /* 21 vars */) = 0
brk(NULL)                                = 0x55d21e20b000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd52e9e610) = -1 EINVAL (I
nvalid argument)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such fil
e or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=98317, ...}) = 0
mmap(NULL, 98317, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fd7e2fd8000
close(3)                                 = 0
openat(AT_FDCWD, "/usr/lib/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0000C\2\0\
0\0\0\0"..., 832) = 832
lseek(3, 792, SEEK_SET)                  = 792
read(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\201\336\t\36\251c\324\23
3E\371SoK\5H\334"..., 68) = 68
fstat(3, {st_mode=S_IFREG|0755, st_size=2136840, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS
, -1, 0) = 0x7fd7e2fd6000
lseek(3, 792, SEEK_SET)                  = 792
read(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\201\336\t\36\251c\324\23
3E\371SoK\5H\334"..., 68) = 68
lseek(3, 864, SEEK_SET)                  = 864
read(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\
0\0\0\0\0", 32) = 32
mmap(NULL, 1848896, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0)
= 0x7fd7e2e12000
mprotect(0x7fd7e2e34000, 1671168, PROT_NONE) = 0
mmap(0x7fd7e2e34000, 1355776, PROT_READ|PROT_EXEC, MAP_PRIVATE|M
AP_FIXED|MAP_DENYWRITE, 3, 0x22000) = 0x7fd7e2e34000
mmap(0x7fd7e2f7f000, 311296, PROT_READ, MAP_PRIVATE|MAP_FIXED|MA
P_DENYWRITE, 3, 0x16d000) = 0x7fd7e2f7f000
```

```
mmap(0x7fd7e2fcc000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MA
P_FIXED|MAP_DENYWRITE, 3, 0x1b9000) = 0x7fd7e2fcc000
mmap(0x7fd7e2fd2000, 13888, PROT_READ|PROT_WRITE, MAP_PRIVATE|MA
P_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fd7e2fd2000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7fd7e2fd7500) = 0
mprotect(0x7fd7e2fcc000, 16384, PROT_READ) = 0
mprotect(0x55d21d27e000, 4096, PROT_READ) = 0
mprotect(0x7fd7e301a000, 4096, PROT_READ) = 0
munmap(0x7fd7e2fd8000, 98317)           = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_S
ETTID|SIGCHLD, child_tidptr=0x7fd7e2fd77d0) = 5087
strace: Process 5087 attached
[pid  5086] nanosleep({tv_sec=1, tv_nsec=0},  <unfinished ...>
[pid  5087] nanosleep({tv_sec=1, tv_nsec=0},  <unfinished ...>
[pid  5086] <... nanosleep resumed> 0x7ffd52e9e5b0) = 0
[pid  5087] <... nanosleep resumed> 0x7ffd52e9e5b0) = 0
[pid  5087] nanosleep({tv_sec=1, tv_nsec=0},  <unfinished ...>
[pid  5086] nanosleep({tv_sec=1, tv_nsec=0},  <unfinished ...>
......
```

Let's see a more complicated case:

```
#include <sys/types.h>
#include <unistd.h>
#include <omp.h>

int main(void)
{
    #pragma omp parallel num_threads(2)
    {
        sleep(30);
        int pid = fork();
        while (1)
        {
            sleep(1);
        }
    }
    return 0;
}
```

Above code will spawn `2` threads, and every thread will fork another child process after waiting `30` seconds:

```
# ./fork &
[1] 5239
# ps -T 5239
  PID  SPID TTY      STAT   TIME COMMAND
 5239  5239 pts/1    Sl     0:00 ./fork
 5239  5240 pts/1    Sl     0:00 ./fork
```

Use any `SPID` (`5239` or `5240` in this case), `strace` will trace all child processes of threads belong to current process:

```
# strace -p 5239 -f
strace: Process 5239 attached with 2 threads
[pid  5240] restart_syscall(<... resuming interrupted nanosleep
...> <unfinished ...>
[pid  5239] restart_syscall(<... resuming interrupted nanosleep
...> <unfinished ...>
[pid  5240] <... restart_syscall resumed> ) = 0
[pid  5239] <... restart_syscall resumed> ) = 0
[pid  5239] clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|C
LONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7f6dd132ced0) = 5245
[pid  5240] futex(0x7f6dd14f47a0, FUTEX_WAIT_PRIVATE, 2, NULL) =
 -1 EAGAIN (Resource temporarily unavailable)
[pid  5240] clone(strace: Process 5245 attached
child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|
SIGCHLD, child_tidptr=0x7f6dd132b9d0) = 5246
[pid  5240] futex(0x7f6dd14f47a0, FUTEX_WAKE_PRIVATE, 1) = 0
[pid  5240] nanosleep({tv_sec=1, tv_nsec=0},  <unfinished ...>
[pid  5239] futex(0x7f6dd14f47a0, FUTEX_WAKE_PRIVATE, 1) = 0
[pid  5239] nanosleep({tv_sec=1, tv_nsec=0}, strace: Process 524
6 attached
 <unfinished ...>
[pid  5246] set_robust_list(0x7f6dd132b9e0, 24) = 0
[pid  5246] futex(0x7f6dd14f47a0, FUTEX_WAKE_PRIVATE, 1) = 0
[pid  5246] nanosleep({tv_sec=1, tv_nsec=0},  <unfinished ...>
[pid  5245] set_robust_list(0x7f6dd132cee0, 24) = 0
[pid  5245] nanosleep({tv_sec=1, tv_nsec=0},  <unfinished ...>
......
```

Detach process on execve syscall

`-b syscall` option can be used to instruct `strace` to detach process when specified syscall is executed. However, currently only `execve` is supported. Check following program code (the code is from here):

```
# cat myecho.c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int j;

    for (j = 0; j < argc; j++)
       printf("argv[%d]: %s\n", j, argv[j]);

   exit(EXIT_SUCCESS);
}

# cat execve.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    char *newargv[] = { NULL, "hello", "world", NULL };
    char *newenviron[] = { NULL };

    if (argc != 2) {
        fprintf(stderr, "Usage: %s <file-to-exec>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    newargv[0] = argv[1];

    execve(argv[1], newargv, newenviron);
    perror("execve");   /* execve() returns only on error */
    exit(EXIT_FAILURE);
}
```

Compile them:

```
# gcc myecho.c -o myecho
# gcc execve.c -o execve
```

Observer the output of using `-b`:

```
# strace -b execve ./execve myecho
execve("./execve", ["./execve", "myecho"], 0x7ffe49f7edf8 /* 21
vars */) = 0
......
munmap(0x7f648b836000, 98317)           = 0
execve("myecho", ["myecho", "hello", "world"], 0x7ffeb1908d28 /*
 0 vars */strace: Process 8287 detached
 <detached ...>
argv[0]: myecho
argv[1]: hello
argv[2]: world
```

The log shows that `strace` detached the process once `execve` is executed. `-b` can be combined use with `-f` to ignore some insignificant child processes during debugging (please refer Trace child processes section).

Detach strace and tracee processes

`-D` option is used to detach `strace` and `tracee` processes. E.g.:

```
# strace -D ./dead_loop
strace: Process 19816 attached
......
```

Open another terminal to check the process relationship:

```
# ps -ef | grep dead
root     19816 19684  0 09:28 pts/0    00:00:00 ./dead_loop
root     19820     1  0 09:28 pts/0    00:00:00 strace -D ./dead
_loop
```

You can see now `dead_loop` is not child process of `strace`.

Show statistics of system calls

`-c` option can be used to summarize the statistics of every system call:

```
# strace -c ls
project
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ------------
---
 20.92    0.000050          50         1           munmap
 19.25    0.000046          23         2           getdents64
 14.64    0.000035           5         6           close
 10.46    0.000025          25         1           write
 10.46    0.000025          12         2           ioctl
  9.21    0.000022           4         5           fstat
  9.21    0.000022           7         3           brk
  5.86    0.000014           3         4           openat
  0.00    0.000000           0         5           read
  0.00    0.000000           0         3           lseek
  0.00    0.000000           0        12           mmap
  0.00    0.000000           0         5           mprotect
  0.00    0.000000           0         1         1 access
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         2         1 arch_prctl
------ ----------- ----------- --------- --------- ------------
---
100.00    0.000239                    53         2 total
```

`strace` shows the time and error count of every system call. Similarly, `-C` option can print both regular output and statistics:

```
# strace -C ls
execve("/usr/bin/ls", ["ls"], 0x7ffe91857fa8 /* 21 vars */) = 0
brk(NULL)                               = 0x555bd69bc000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffc6ea19940) = -1 EINVAL (I
nvalid argument)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such fil
e or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=98317, ...}) = 0
mmap(NULL, 98317, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fdf81c6f000

......
close(1)                                = 0
close(2)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ------------
---
 26.89    0.000256          21        12           mmap
 13.97    0.000133          33         4           openat
 11.45    0.000109          21         5           fstat
  9.56    0.000091          15         6           close
  7.98    0.000076          15         5           mprotect
......
```

Notice that `-c` and `-C` options are mutually exclusive. Please notice the time here measured is system time of every system call. If wall time is required, it is `-w` option :

```
# strace -w -c ls
project
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ------------
---
 19.85    0.000297          24        12           mmap
 17.98    0.000270         269         1           execve
  9.32    0.000140          23         6           close
......
```

There is another `-S sortby` option to customize histogram output sorted by `calls`, `name` and `nothing` (default is sorted by `time`). E.g.:

```
# strace -c -S calls ls
project
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ------------
---
 27.66    0.000164          13        12           mmap
  8.26    0.000049           8         6           close
  7.76    0.000046           9         5           read
  5.40    0.000032           6         5           fstat
 13.15    0.000078          15         5           mprotect
  5.23    0.000031           7         4           openat
  5.23    0.000031          10         3           lseek
  4.05    0.000024           8         3           brk
  4.05    0.000024          12         2           ioctl
  1.69    0.000010           5         2         1 arch_prctl
  7.93    0.000047          23         2           getdents64
  4.55    0.000027          27         1           write
  5.06    0.000030          30         1           munmap
  0.00    0.000000           0         1         1 access
  0.00    0.000000           0         1           execve
------ ----------- ----------- --------- --------- ------------
---
100.00    0.000593                    53         2 total
```

Customize tracing event behavior

`-e expression` is the most complicated option which you use a qualifying expression to denote which events to trace or how to trace them. The format of expression is:

```
[qualifier=][!][?]value1[,[?]value2]...
```

Qualifier can be `trace` (or `t`), `abbrev` (or `a`), `verbose` (or `v`), `raw` (or `x`), `signal` / `signals` (or `s`), `read` / `reads` (or `r`), `write` / `writes` (or `w`), `fault`, `inject`, or `kvm`. If no qualifier is specified, `trace` is the used. E.g., `-e open` is equal to `-e trace=open` or `-e t=open`.

The value part can contain one or more values, and it depends on the qualifier. E.g., `-e t=open,write` means only output `open` and `write` system calls. `!` is used to filter out the specified system call. E.g.:

```
# strace -e trace=\!write ls
```

This won't trace `write` system call (The `!` need to be escaped in `bash` shell). `?` is used to suppress the error. Compare the following outputs:

```
# strace -e trace=op ls
strace: invalid system call 'op'
# strace -e trace=?op ls
project
+++ exited with 0 +++
```

"`@64`", "`@32`", or "`@x32`" suffixes can be used to specify system calls only for the `64-bit`, `32-bit`, or `32-on-64-bit` respectively.

The value can also be `all` or `none` which mean following all system calls or nothing:

```
# strace -e trace=all ls
execve("/usr/bin/ls", ["ls"], 0x7ffd5ae4ff10 /* 20 vars */) = 0
brk(NULL)                               = 0x55aae08a1000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffcdecdc3e0) = -1 EINVAL (I
nvalid argument)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such fil
e or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=98317, ...}) = 0
......
# strace -e trace=none ls
project
+++ exited with 0 +++
```

Values for trace qualifier

Below table describes the valid values for `trace` qualifier (Most are just copied from strace manual):

Value	Meaning
set	Trace the system calls in set. E.g., "strace -e write ls".
/regex	Trace all the system calls which match regular expression. E.g., "strace -e /wr* ls".
%file	Trace all the system calls which take a file name as an argument. It incules open, stat, etc.
%process	Trace all the system calls which involve process management.Such as fork, wait, etc.
%network(or %net)	Trace all the network related system calls.
%signal	Trace all signal related system calls.
%ipc	Trace all IPC related system calls.
%desc	Trace all file descriptor related system calls.
%memory	Trace all memory mapping related system calls.
%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 expression.
%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 expression.
%pure	Trace syscalls that always succeed and have no arguments. Currently, this list includes arc_gettls, getdtablesize, getegid, getegid32, geteuid, geteuid32, getgid, getgid32, getpagesize, getpgrp, getpid, getppid, get_thread_area (on architectures other than x86), gettid, get_tls, getuid, getuid32, getxgid, getxpid, getxuid, kern_features, and metag_get_tls.

To know detailed information about which category a specific system call belongs to, you can refer sysent.h, sysent_shorthand_defs.h, and `syscallent.h` for your machine (E.g., `X86_64` file is here).

abbrev qualifier

`abbrev` qualifier is used to omit printing each member of large structures. The default value is `all`. The following example shows how to check environment variables when executing `execve` system call:

```
# strace -e trace=execve -e abbrev=none ls
execve("/usr/bin/ls", ["ls"], ["SHELL=/bin/bash", "PWD=/root", "
LOGNAME=root", "XDG_SESSION_TYPE=tty", "HOME=/root", "LANG=C", "
SSH_CONNECTION=10.218.195.134 65"..., "XDG_SESSION_CLASS=user",
"TERM=xterm", "USER=root", "SHLVL=1", "XDG_SESSION_ID=19", "XDG_
RUNTIME_DIR=/run/user/0", "SSH_CLIENT=10.218.195.134 65285 "...,
 "PATH=/root/.cargo/bin:/usr/local"..., "DBUS_SESSION_BUS_ADDRES
S=unix:pa"..., "HG=/usr/bin/hg", "MAIL=/var/spool/mail/root", "S
SH_TTY=/dev/pts/0", "_=/usr/bin/strace"]) = 0
+++ exited with 0 +++
```

Actually, `strace` has `-v` option which just leverages `abbrev` qualifier (code is here):

```
    ......
    case 'v':
        qualify("abbrev=none");
        break;
    ......
```

verbose qualifier

`verbose` qualifier is used to instruct whether deference structures of system calls or not. The default value is `all`. E.g., if you don't want to deference structures of `execve`, you can do this:

```
# strace -e trace=execve -e verbose=execve ls
execve("/usr/bin/ls", ["ls"], 0x7fffc8641bd0 /* 20 vars */) = 0
+++ exited with 0 +++
```

raw qualifier

`raw` qualifier is used to print undecoded arguments in hexadecimal format. You can compare the output of `access` without and with `raw` qualifier:

```
# strace -e access  ls
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such fil
e or directory)
+++ exited with 0 +++
# strace -e access -e raw=access ls
access(0x7f11091183a0, 0x4)             = -1 ENOENT (No such fil
e or directory)
+++ exited with 0 +++
```

signal qualifier

By default, `strace` will capture all signals. `signal` qualifier can be used to customize which signals should be traced. Check the following code:

```
# cat dead_loop.c
#include <unistd.h>

int main(void)
{
        while (1)
        {
                sleep(1);
        }
        return 0;
}
```

Build and run it in one terminal:

```
# gcc dead_loop.c -o dead_loop
# ./dead_loop
```

Use `strace` to track it in another terminal:

```
# strace -p `pidof dead_loop`
strace: Process 18085 attached
restart_syscall(<... resuming interrupted nanosleep ...>) = 0
nanosleep({tv_sec=1, tv_nsec=0}, 0x7ffd8492c6f0) = 0
......
```

Press `Ctrl+C` in first terminal to kill `dead_loop` process, you will find `SIGINT` related information is printed out in second terminal:

```
# strace -p `pidof dead_loop`
strace: Process 18085 attached
restart_syscall(<... resuming interrupted nanosleep ...>) = 0
nanosleep({tv_sec=1, tv_nsec=0}, 0x7ffd8492c6f0) = 0
......
nanosleep({tv_sec=1, tv_nsec=0}, {tv_sec=0, tv_nsec=825212664})
= ? ERESTART_RESTARTBLOCK (Interrupted by signal)
--- SIGINT {si_signo=SIGINT, si_code=SI_KERNEL} ---
+++ killed by SIGINT +++
```

If you ignore `SIGINT`, the `SIGINT` won't be showed:

```
# strace -p `pidof dead_loop` -e signal=\!int
strace: Process 18091 attached
restart_syscall(<... resuming interrupted nanosleep ...>) = 0
nanosleep({tv_sec=1, tv_nsec=0}, 0x7ffc92b62710) = 0
nanosleep({tv_sec=1, tv_nsec=0}, 0x7ffc92b62710) = 0
nanosleep({tv_sec=1, tv_nsec=0}, 0x7ffc92b62710) = 0
nanosleep({tv_sec=1, tv_nsec=0}, {tv_sec=0, tv_nsec=361293365})
= ? ERESTART_RESTARTBLOCK (Interrupted by signal)
```

read and write qualifiers

`read` / `write` qualifiers are used to perform a full hexadecimal and ASCII dump of all the data read/write from file descriptors. Check following example:

```
# strace -e read=3 ls
execve("/usr/bin/ls", ["ls"], 0x7ffd0fd7d750 /* 21 vars */) = 0
......
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0  \0\0\0\
0\0\0"..., 832) = 832
 | 00000  7f 45 4c 46 02 01 01 00  00 00 00 00 00 00 00 00  .ELF
............ |
 | 00010  03 00 3e 00 01 00 00 00  20 20 00 00 00 00 00 00  ..>.
....  ...... |
 | 00020  40 00 00 00 00 00 00 00  38 52 00 00 00 00 00 00  @...
....8R...... |
 | 00030  00 00 00 00 40 00 38 00  09 00 40 00 16 00 15 00  ....
@.8...@..... |
 | 00040  01 00 00 00 04 00 00 00  00 00 00 00 00 00 00 00  ....
............ |
 | 00050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ....
............ |
 | 00060  b0 14 00 00 00 00 00 00  b0 14 00 00 00 00 00 00  ....
............ |
......
```

inject qualifier

The definition of `inject` qualifier is like this:

```
-e inject=set[:error=errno|:retval=value][:signal=sig][:syscall=
syscall][:delay_enter=usecs][:delay_exit=usecs][:when=expr]
```

It is used to tamper a specific set of system calls.

`:error=errno` can set the wrong return value of a system call. E.g.:

```
# strace -e inject=read:error=1 ls
......
read(3, 0x7ffca47eb418, 832)            = -1 EPERM (Operation no
t permitted) (INJECTED)
close(3)                                = 0
......
```

`:retval=value` does the reverse thing: set the correct return value:

```
# strace -e inject=arch_prctl:retval=0 ls
......                                  = 0x561afd1f6000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffdcba37700) = 0 (INJECTED)
```

Please notice `:error=errno` and `:retval=value` are mutually exclusive.

`:signal=sig` sets which signal will be delivered when entering a system call:

```
# strace -e inject=read:signal=SIGSEGV ls
......
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0  \0\0\0\
0\0\0"..., 832) = 832
--- SIGSEGV {si_signo=SIGSEGV, si_code=SI_KERNEL, si_addr=NULL}
---
+++ killed by SIGSEGV (core dumped) +++
Segmentation fault (core dumped)
```

`:delay_enter=usecs` / `:delay_exit=usecs` are used to set how many microseconds are delayed before entering/exiting system calls:

```
# strace -e inject=read:delay_enter=10:delay_exit=10 ls
```

`:syscall=syscall` is used to inject the specified system call, only "pure" system call is allowed now (please refer `Values for trace qualifier` section).

`:when=expr` is used to control the frequency of injection. By default every invocation will be injected. The format of the expression is one of the following:

Frequency	Meaning
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 invocations.
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:

```
# strace -e inject=read:error=1:when=2 ls
execve("/usr/bin/ls", ["ls"], 0x7fff64d98c70 /* 21 vars */) = 0
......
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\3\0>\0\1\0\0\0  \0\0\0\
0\0\0"..., 832) = 832
......
read(3, 0x7ffec8c113a8, 832)            = -1 EPERM (Operation no
t permitted) (INJECTED)
......
```

The `inject` takes effect when calling `read` in the second time.

There are two more tracing events: `fault=set[:error=errno][:when=expr]` is similar as `inject`, and `kvm=vcpu` is used to print exit reason of `kvm vcpu` (requires kernel version `4.16` or later).

Change environment variables' values

`-E` option is used to change environment variables' values. Actually, `strace` just calls `putenv()` function (code is here). Check following code:

```
# cat env_test.c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    char *p = NULL;
    if (p = getenv("HG"))
    {
        printf("HG enviromental variable value is %s\n", p);
    }
    else
    {
        printf("HG enviromental variable value is not set\n");
    }
    return 0;
}
```

Build and run it:

```
#  gcc env_test.c -o env_test
# ./env_test
HG enviromental variable value is /usr/bin/hg
```

Set `HG` to other value:

```
# strace -E "HG=git" ./env_test
......
write(1, "HG enviromental variable value i"..., 38HG enviromenta
l variable value is git
) = 38
......
```

Remove `HG`:

```
# strace -E "HG" ./env_test
......
write(1, "HG enviromental variable value i"..., 42HG enviromenta
l variable value is not set
) = 42
......
```

Run command as other user

`-u username` option can run command as other user, and this option only take effect when executing as `root`. Check following example:

```
# cat getuid.c
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>

int main(void)
{
    printf("uid is %d\n", getuid());
    return 0;
}
```

Build and run it as `root`:

```
# gcc getuid.c -o getuid
# strace ./getuid
execve("./getuid", ["./getuid"], 0x7ffce778f8a0 /* 12 vars */) =
 0
brk(NULL)                               = 0x563fe210a000
......
write(1, "uid is 0\n", 9uid is 0
)               = 9
exit_group(0)                           = ?
+++ exited with 0 +++
```

It shows `uid` is `0`. Run the program as another user:

```
# strace -u nan ./getuid
......
write(1, "uid is 1000\n", 12uid is 1000
)           = 12
exit_group(0)                           = ?
+++ exited with 0 +++
```

This time, `uid` is `1000`.

Set output format of naming constant

By default, `strace` will decode naming constant (equals to use `-X abbrev` option):

```
# strace ls
......
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such fil
e or directory)
......
```

If raw value is wanted, using `-X raw` option:

```
# strace -X raw ls
......
access("/etc/ld.so.preload", 0x4)       = -1 ENOENT (No such fil
e or directory)
......
```

`-X verbose` outputs both:

```
# strace -X verbose ls
......
access("/etc/ld.so.preload", 0x4 /* R_OK */) = -1 ENOENT (No suc
h file or directory)
```

Customize string output

`-x` option makes non-ASCII strings outputted in hexadecimal format (default is octal):

```
# strace -x ls
......
read(3, "\x7f\x45\x4c\x46\x02\x01\x01\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x03\x00\x3e\x00\x01\x00\x00\x00\x20\x20\x00\x00\x00\x0
0\x00\x00"..., 832) = 832
```

`-xx` makes all strings outputted in hexadecimal format:

```
# strace -xx ls
......
read(3, "\x7f\x45\x4c\x46\x02\x01\x01\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x03\x00\x3e\x00\x01\x00\x00\x00\x20\x20\x00\x00\x00\x0
0\x00\x00"..., 832) = 832
```

Furthermore, `-s strsize` changes the maximum print size of string (default value is `32`). Compare the following outputs:

(1) Using default value:

```
# strace ls
......
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0  \0\0\0\
0\0\0"..., 832) = 832
......
```

(2) Double the size:

```
# strace -s 64 ls
......
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0  \0\0\0\
0\0\0@\0\0\0\0\0\0\08R\0\0\0\0\0\0\0\0\0\0@\08\0\t\0@\0\26\0\25\
0"..., 832) = 832
......
```

Please note that filenames are not considered strings and are always printed in full.

Print instruction pointer

`-i` option is used to print instruction pointer of calling system call. E.g.:

```
# strace -i ls
[00007f08c925a98b] execve("/usr/bin/ls", ["ls"], 0x7ffe81ee42c8
/* 21 vars */) = 0
[00007f53d35a026b] brk(NULL)            = 0x55fc72a56000
[00007f53d359f035] arch_prctl(0x3001 /* ARCH_??? */, 0x7ffc45b7e
080) = -1 EINVAL (Invalid argument)
[00007f53d35a0f3b] access("/etc/ld.so.preload", R_OK) = -1 ENOEN
T (No such file or directory)
[00007f53d35a1061] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY
|O_CLOEXEC) = 3
[00007f53d35a0e87] fstat(3, {st_mode=S_IFREG|0644, st_size=98309
, ...}) = 0
......
```

Handle signals

`-I interruptible` option is used to handle signals. The `interruptible`'s value can be `1` ~ `4`:

Value	Meaning
1	Don't block any signal
2	Block fatal signals while decoding syscall, and this is default behaviour.
3	Block fatal signals, and this is default behaviour when using '-o FILE PROG' option.
4	Block fatal signals and SIGTSTP (^Z), and this is useful when not want '-o FILE PROG' option to stop on ^Z.

The fatal signals include `SIGHUP`, `SIGINT`, `SIGQUIT`, `SIGPIPE` and `SIGTERM`. Check following example:

```
# strace -I 1 ./dead_loop
......
nanosleep({tv_sec=1, tv_nsec=0}, 0x7fff0a0d90f0) = 0
nanosleep({tv_sec=1, tv_nsec=0}, 0x7fff0a0d90f0) = 0
nanosleep({tv_sec=1, tv_nsec=0}, 0x7fff0a0d90f0) = 0
......
```

Open another terminal and input following command:

```
# kill -s INT `pidof strace`
```

You will find `strace` process is killed. If you run `strace` using "`-I 3`", you will find the `SIGINT` will not take effect when using above command:

```
 # kill -s INT `pidof strace`
```

Show file descriptor related information

`-y` option can be used to print file path associated with each file descriptor. Compare the following outputs:

a) Without `-y` option:

```
# strace ls
......
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=98309, ...}) = 0
......
```

b) With `-y` option:

```
# strace -y ls
......
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3</et
c/ld.so.cache>
fstat(3</etc/ld.so.cache>, {st_mode=S_IFREG|0644, st_size=98309,
 ...}) = 0
......
```

Furthermore, `-yy` option can be used to print protocol specific information associated with socket file descriptors, and block/character device number associated with device file descriptors. Compare following outputs:

a) Without `-yy` option:

```
# strace ip
......
socket(AF_NETLINK, SOCK_RAW|SOCK_CLOEXEC, NETLINK_ROUTE) = 3
setsockopt(3, SOL_SOCKET, SO_SNDBUF, [32768], 4) = 0
......

# strace ls
......
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x1), ...}
) = 0
......
```

b) With `-yy` option:

```
# strace -yy ip
......
socket(AF_NETLINK, SOCK_RAW|SOCK_CLOEXEC, NETLINK_ROUTE) = 3<NET
LINK:[29833]>
setsockopt(3<NETLINK:[29833]>, SOL_SOCKET, SO_SNDBUF, [32768], 4
) = 0
......

# strace -yy ls
......
fstat(1</dev/pts/1<char 136:1>>, {st_mode=S_IFCHR|0620, st_rdev=
makedev(0x88, 0x1), ...}) = 0
......
```

Show timestamp of system call

`-t` option shows the wall clock time of every system call:

```
# strace -t ls
13:20:19 execve("/usr/bin/ls", ["ls"], 0x7fff489a7c98 /* 21 vars
 */) = 0
13:20:19 brk(NULL)                      = 0x55cd60264000
......
```

`-tt` option will append microseconds:

```
# strace -tt ls
13:21:07.553038 execve("/usr/bin/ls", ["ls"], 0x7ffc5238f338 /*
21 vars */) = 0
13:21:07.553713 brk(NULL)               = 0x559753feb000
......
```

`-ttt` option prints seconds since epoch time:

```
# strace -ttt ls
1552540937.975302 execve("/usr/bin/ls", ["ls"], 0x7ffdef5c3918 /
* 21 vars */) = 0
1552540937.976413 brk(NULL)             = 0x55fe713d8000
......
```

`-r` option prints a relative timestamp upon entry to each system call. This records the time difference between the beginning of successive system calls:

```
# strace -r ls
0.000000 execve("/usr/bin/ls", ["ls"], 0x7ffc3ad8a5e8 /* 21 vars
 */) = 0
0.000623 brk(NULL)                      = 0x55cce063d000
0.000172 arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd8e8867d0) = -1
EINVAL (Invalid argument)
......
```

Show time spent in each system call

`-T` option is used to print time spent in each system call. E.g.:

```
# strace -T ls
execve("/usr/bin/ls", ["ls"], 0x7ffd141a4ee8 /* 20 vars */) = 0
<0.000354>
brk(NULL)                               = 0x55b222b07000 <0.0000
80>
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd1758c950) = -1 EINVAL (I
nvalid argument) <0.000051>
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such fil
e or directory) <0.000083>
......
```

Print stack trace of every system call

`-k` option is used to print stack trace of every system call. To enable this option, `ENABLE_STACKTRACE` is needed to build `strace` (code is here). E.g.:

```
# strace -k ls
execve("/usr/bin/ls", ["ls"], 0x7ffe9c8243b8 /* 21 vars */) = 0
 > /usr/lib/libc-2.28.so(execve+0xb) [0xc898b]
 > /usr/bin/strace(+0x0) [0xa25a4]
 > /usr/bin/strace(+0x0) [0xa40ad]
 > /usr/bin/strace(+0x0) [0x6f24c]
 > /usr/lib/libc-2.28.so(__libc_start_main+0xf3) [0x24223]
 > /usr/bin/strace(+0x0) [0x6fa7e]
brk(NULL)                               = 0x55d840fe5000
 > /usr/lib/ld-2.28.so(__brk+0xb) [0x1b26b]
 > /usr/lib/ld-2.28.so(_dl_sysdep_start+0x2f1) [0x19ef1]
 > /usr/lib/ld-2.28.so(_dl_start+0x288) [0x3088]
 > /usr/lib/ld-2.28.so() [0x2008]
......
```

Trace only system calls accessing path

`-P path` option is used to set only tracing system calls which access the specified `path`. E.g.:

```
# strace -P /etc/ld.so.preload ls
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such fil
e or directory)
+++ exited with 0 +++
```

Only `access` system call is tracked.

Suppress attach/detach/exit status messages

`-q` option is used to suppress attach/detach process message. Compare following outputs:

a) Without `-q` option:

```
# strace -p 444
strace: Process 444 attached
restart_syscall(<... resuming interrupted nanosleep ...>) = 0
nanosleep({tv_sec=1, tv_nsec=0}, 0x7ffc026d31c0) = 0
nanosleep({tv_sec=1, tv_nsec=0}, 0x7ffc026d31c0) = 0
nanosleep({tv_sec=1, tv_nsec=0}, ^Cstrace: Process 444 detached
 <detached ...>
```

b) With `-q` option:

```
# strace -q -p 444
restart_syscall(<... resuming interrupted nanosleep ...>) = 0
nanosleep({tv_sec=1, tv_nsec=0}, 0x7ffc026d31c0) = 0
nanosleep({tv_sec=1, tv_nsec=0}, 0x7ffc026d31c0) = 0
nanosleep({tv_sec=1, tv_nsec=0}, ^C <detached ...>
```

`-qq` option will also suppress process exit status message. Compare following outputs:

a) Without `-qq` option:

```
# strace ls
......
exit_group(0)                           = ?
+++ exited with 0 +++
```

b) With `-qq` option:

```
# strace -qq ls
......
exit_group(0)                           = ?
```

Display strace's debug information

`-d` option can be used to show `strace` 's own debug information. Check following simple program:

```
# cat fork.c
#include <sys/types.h>
#include <unistd.h>

int main(void)
{
        int pid = fork();
        while (1)
        {
            sleep(1);
        }
        return 0;
}
```

Build and use `strace` 's `-d` option to track it:

```
# gcc fork.c -o fork
# strace -d ./fork
strace: ptrace_setoptions = 0x51
strace: PTRACE_GET_SYSCALL_INFO: Input/output error
strace: PTRACE_GET_SYSCALL_INFO does not work
strace: new tcb for pid 583, active tcbs:1
strace: [wait(0x80137f) = 583] WIFSTOPPED,sig=SIGSTOP,EVENT_STOP
 (128)
strace: pid 583 has TCB_STARTUP, initializing it
strace: [wait(0x80057f) = 583] WIFSTOPPED,sig=SIGTRAP,EVENT_STOP
 (128)
strace: [wait(0x00127f) = 583] WIFSTOPPED,sig=SIGCONT
strace: [wait(0x00857f) = 583] WIFSTOPPED,sig=133
execve("./fork", ["./fork"], 0x7ffcc9f68b48 /* 21 vars */strace:
 [wait(0x04057f) = 583] WIFSTOPPED,sig=SIGTRAP,EVENT_EXEC (4)
......
strace: [wait(0x00857f) = 583] WIFSTOPPED,sig=133
clone(strace: [wait(0x00857f) = 583] WIFSTOPPED,sig=133
......
```

The debug information is interleaved with normal output. Open another terminal to check the process relationship, and this can help knowing debug information better:

```
# pstree -p `pidof strace`
strace(580)---fork(583)---fork(584)
```

Besides child processes' exit status, `strace` cares about events such as forking process, exit process, etc (please refer related code).

Set the overhead for tracing system calls

`-O` option is used to set the overhead for tracing system calls, and the unit is microseconds (us). The following is excerpted from manual:

> 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.

Check the source code:

```
    ......
    ts_mul(&dtv, &overhead, counts[i].calls);
    ts_sub(&counts[i].time, &counts[i].time, &dtv);
    ......
```

The overhead time is subtracted from the total consuming time, and this option just makes sure the measure of time spent on system calls to be more accurate.

Align return value column

By default, the alignment column for results is `40` :

```
# strace ls
execve("/usr/bin/ls", ["ls"], 0x7ffca1e1f070 /* 21 vars */) = 0
brk(NULL)                               = 0x55c6d34e2000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd28e4cd10) = -1 EINVAL (I
nvalid argument)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such fil
e or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=98317, ...}) = 0
mmap(NULL, 98317, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f6ebc2ae000
close(3)                                = 0
openat(AT_FDCWD, "/usr/lib/libcap.so.2", O_RDONLY|O_CLOEXEC) = 3
......
```

`-a` option can be used to adjust the alignment:

```
# strace -a 80 ls
execve("/usr/bin/ls", ["ls"], 0x7fffe93ef0d0 /* 21 vars */)
                = 0
brk(NULL)
                = 0x562a6bfa1000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd332954c0)
                = -1 EINVAL (Invalid argument)
access("/etc/ld.so.preload", R_OK)
                = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC)
                = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=98317, ...})
                = 0
mmap(NULL, 98317, PROT_READ, MAP_PRIVATE, 3, 0)
                = 0x7f2f8fd22000
close(3)
                = 0
......
```

Tune this parameter according to your screen or your habit.

The end

This manual refers `strace`'s manual and code. If it helps you, please give it a star in github. :-)