Linux Strace Command

Name	**:**	**Sajjad Ali**
CMSID	**:**	**023-19-0100**
Department	**:**	**Computer Science**
Semester	**:**	**BS IV Section B**
Course	**:**	**Operating System (Theory)**
Assignment	**:**	**Strace (#01)**
Submitted to	**:**	**Dr. Raheel Memon**

S trace

It is a diagnostic, debugging and instructional user space utility for Linux. It is used to monitor and temper with interactions processes and Linux kernel, which include system calls, signal deliveries and changes of process state.

It can be installed through this command on terminal:

$ sudo apt install strace

- We can check the commands in strace via

  $ strace -h

- We can trace any running processor via following command:

  $ sudo strace -c -p PID

  where PID is the processor Id of any running processor.

  Suppose we have run following command:

  $ sudo strace -c -p 2010

  The most probable output is the following type:

```
genius@192:~$ sudo strace -c -p 2010
strace: Process 2010 attached
^Cstrace: Process 2010 detached
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 34.43    0.001081           5       207       176 recvmsg
 24.17    0.000759           9        83           poll
 17.32    0.000544           9        59           ioctl
 10.67    0.000335           8        39           writev
  6.56    0.000206           5        40           getpid
  4.90    0.000154           5        29           write
  1.59    0.000050           4        12           read
  0.35    0.000011          11         1           restart_syscall
------ ----------- ----------- --------- --------- ----------------
100.00    0.003140                   470       176 total
```

Here syscall refers to the System Calls.

- We can also trace with respect to some predicate or condition.

  $ sudo strace -q -e trace=process df -h

```
genius@192:~$ sudo strace -q -e trace=process df -h
execve("/usr/bin/df", ["df", "-h"], 0x7ffc80957ba0 /* 25 vars */) = 0
arch_prctl(0x3001 /* ARCH_??? */, 0x7fff97c69860) = -1 EINVAL (Invalid argument)
arch_prctl(ARCH_SET_FS, 0x7f4dc21f9580) = 0
Filesystem      Size  Used Avail Use% Mounted on
udev            1.9G     0  1.9G   0% /dev
tmpfs           385M  2.0M  383M   1% /run
/dev/sda7        56G   25G   28G  48% /
tmpfs           1.9G  4.0K  1.9G   1% /dev/shm
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           1.9G     0  1.9G   0% /sys/fs/cgroup
/dev/loop1      128K  128K     0 100% /snap/anbox-installer/24
/dev/loop3       56M   56M     0 100% /snap/core18/1988
/dev/loop4      256K  256K     0 100% /snap/figlet/36
/dev/loop0       99M   99M     0 100% /snap/core/10823
/dev/loop5      163M  163M     0 100% /snap/gnome-3-28-1804/145
/dev/loop7       65M   65M     0 100% /snap/gtk-common-themes/1514
/dev/loop6      219M  219M     0 100% /snap/gnome-3-34-1804/66
/dev/loop10      32M   32M     0 100% /snap/snapd/11036
/dev/loop8       52M   52M     0 100% /snap/snap-store/518
/dev/loop11      33M   33M     0 100% /snap/snapd/11107
/dev/loop9      425M  425M     0 100% /snap/pycharm-community/226
/dev/loop2      100M  100M     0 100% /snap/core/10859
/dev/sda1        96M   33M   64M  35% /boot/efi
tmpfs           385M  100K  384M   1% /run/user/1000
exit_group(0)                           = ?
+++ exited with 0 +++
```

  Like this we can also trace for file, memory, network or signal just by replacing process with others.

- We can also save the result of trace in any file.

$ sudo strace -o filename df -h

```
genius@192:~/Downloads$ sudo strace -o file.txt df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            1.9G     0  1.9G   0% /dev
tmpfs           385M  2.0M  383M   1% /run
/dev/sda7        56G   25G   28G  48% /
tmpfs           1.9G  4.0K  1.9G   1% /dev/shm
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           1.9G     0  1.9G   0% /sys/fs/cgroup
/dev/loop1      128K  128K     0 100% /snap/anbox-installer/24
/dev/loop3       56M   56M     0 100% /snap/core18/1988
/dev/loop4      256K  256K     0 100% /snap/figlet/36
/dev/loop0       99M   99M     0 100% /snap/core/10823
/dev/loop5      163M  163M     0 100% /snap/gnome-3-28-1804/145
/dev/loop7       65M   65M     0 100% /snap/gtk-common-themes/1514
/dev/loop6      219M  219M     0 100% /snap/gnome-3-34-1804/66
/dev/loop10      32M   32M     0 100% /snap/snapd/11036
/dev/loop8       52M   52M     0 100% /snap/snap-store/518
/dev/loop11      33M   33M     0 100% /snap/snapd/11107
/dev/loop9      425M  425M     0 100% /snap/pycharm-community/226
/dev/loop2      100M  100M     0 100% /snap/core/10859
/dev/sda1        96M   33M   64M  35% /boot/efi
tmpfs           385M  100K  384M   1% /run/user/1000
```

- We can also trace the programs written in any language whether we have its source code or not.

  Suppose hello.c program is written in c and prints hello world on the console. After compiling, we are tracing this program via:

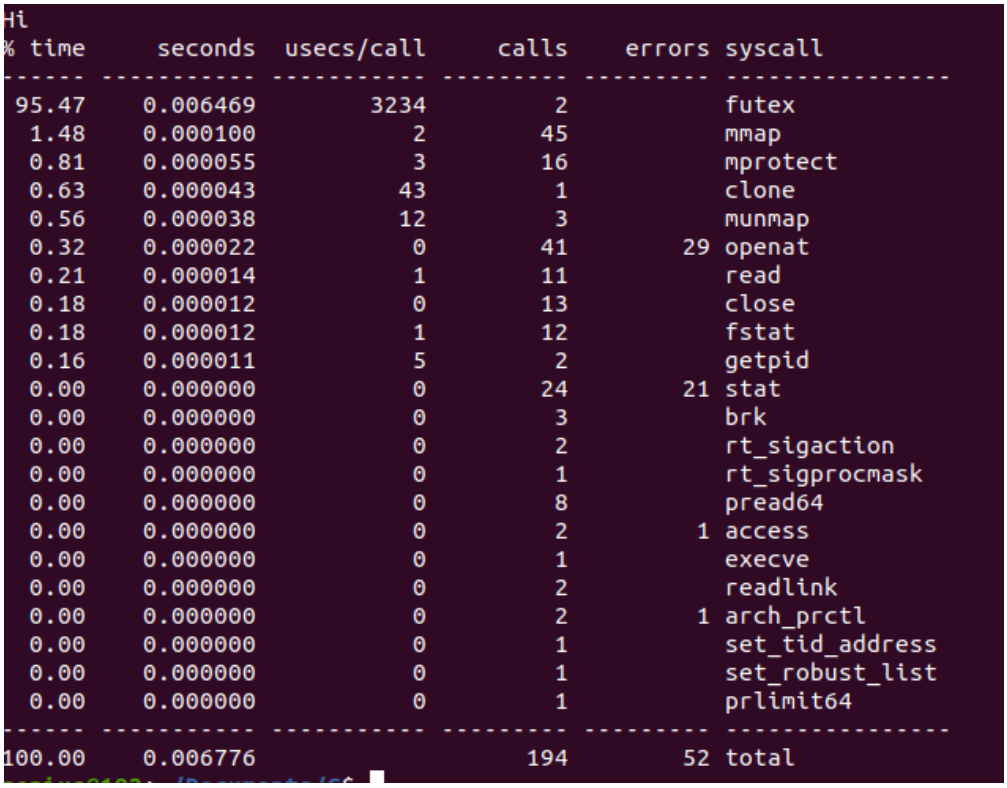
  $ strace -c ./hello

```
Hello World% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
  0.00    0.000000           0         1           read
  0.00    0.000000           0         1           write
  0.00    0.000000           0         2           close
  0.00    0.000000           0         3           fstat
  0.00    0.000000           0         7           mmap
  0.00    0.000000           0         4           mprotect
  0.00    0.000000           0         1           munmap
  0.00    0.000000           0         3           brk
  0.00    0.000000           0         6           pread64
  0.00    0.000000           0         1         1 access
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         2         1 arch_prctl
  0.00    0.000000           0         2           openat
------ ----------- ----------- --------- --------- ----------------
100.00    0.000000                    34         2 total
```

  Similarly tracing the java program named hi.java after compiling,

  $ strace -c java hi

h



```
Hi
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 95.47    0.006469        3234         2           futex
  1.48    0.000100           2        45           mmap
  0.81    0.000055           3        16           mprotect
  0.63    0.000043          43         1           clone
  0.56    0.000038          12         3           munmap
  0.32    0.000022           0        41        29 openat
  0.21    0.000014           1        11           read
  0.18    0.000012           0        13           close
  0.18    0.000012           1        12           fstat
  0.16    0.000011           5         2           getpid
  0.00    0.000000           0        24        21 stat
  0.00    0.000000           0         3           brk
  0.00    0.000000           0         2           rt_sigaction
  0.00    0.000000           0         1           rt_sigprocmask
  0.00    0.000000           0         8           pread64
  0.00    0.000000           0         2         1 access
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         2           readlink
  0.00    0.000000           0         2         1 arch_prctl
  0.00    0.000000           0         1           set_tid_address
  0.00    0.000000           0         1           set_robust_list
  0.00    0.000000           0         1           prlimit64
------ ----------- ----------- --------- --------- ----------------
100.00    0.006776                   194        52 total
```

We can see that java program makes more system calls than the same C based program.