ELISA

SUMMIT

SEPTEMBER 7 - 8, 2022 | VIRTUAL

@PROJECTELISA

KERNEL TRACING

Shefali Sharma

$ whoami

- Senior year CSE student from India
- LFX Mentee at ELISA Medical Devices WG
- Email: sshefali021@gmail.com
- GitHub: Shefali321
- LinkedIn: https://www.linkedin.com/in/sharma-shefali/

Major Takeaways after the Session

- Strace for analyzing system calls and determining the resource usage made by a workload.
- Cscope for browsing C, C++ or Java codebases.
- Ftrace for analyzing kernel function calls.
- Perf for analyzing the performance of an application.

Acknowledgement

I would like to extend my heartfelt gratitude to my mentors **Ms. Shuah Khan**, **Mr. Milan Lakhani** for giving me this opportunity and for their stimulating suggestions and unending support.

I am also very grateful to all the members of Medical Devices WG **Ms. Kate Stewart**, **Ms. Nicole Pappler**, **Mr. Jason Smith** and **Mr. Jeffrey Osier-Mixon** for their constant support and guidance.

In the end I would also like to thank Linux Foundation for providing me this platform.

System Calls

System call is a programming interface for an application for requesting services (like filesystem access, memory) from the operating system.

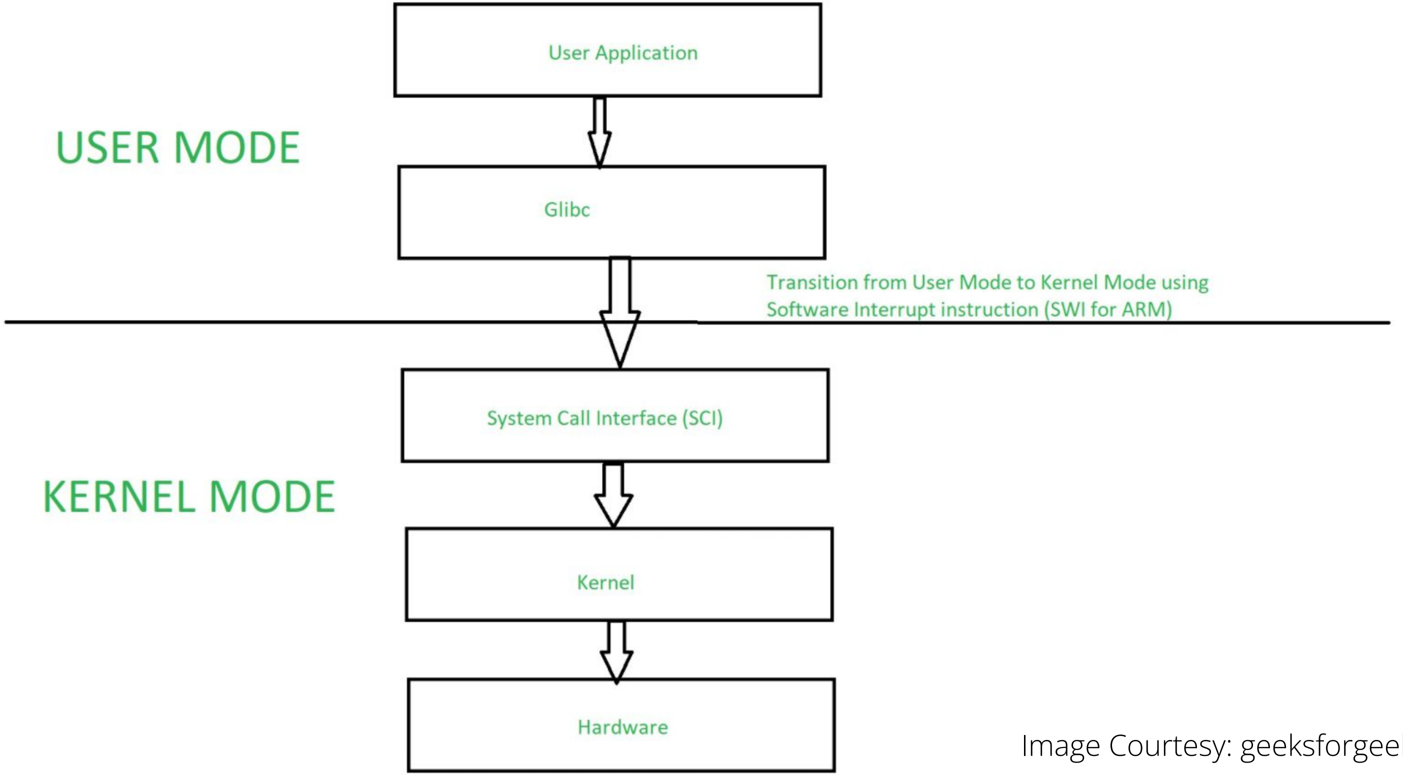

User Application
USER MODE
Glibc
Transition from User Mode to Kernel Mode using
Software Interrupt instruction (SWI for ARM)
System Call Interface (SCI)
KERNEL MODE
Kernel
Hardware

Image Courtesy: geeksforgeeks.org

Modes in an Operating System

User Mode: It has restricted control over the hardware. In order to use any system resources it has to issue system calls.

Kernel Mode / Privileged Mode: It has full control over the hardware. It can execute any instruction and access any memory location.

System Call Examples

- **create:** Creates a new file and assigns it a file descriptor.
- **pipe:** Creates an interprocess communication channel.
- **ioctl:** To control a device.
- **socket:** Creates an endpoint for communication over the network and returns a descriptor.
- **mmap:** Establishes a mapping between a process’s address space and a file.

Advantages of this approach

- System calls allow the kernel to carefully expose certain key pieces of functionality to user programs, such as accessing the file system, creating and destroying processes, communicating with other processes, and allocating more memory. Thus, they help to improve the security and safety of the systems.
- Results in less coupling between user space applications and operating system.

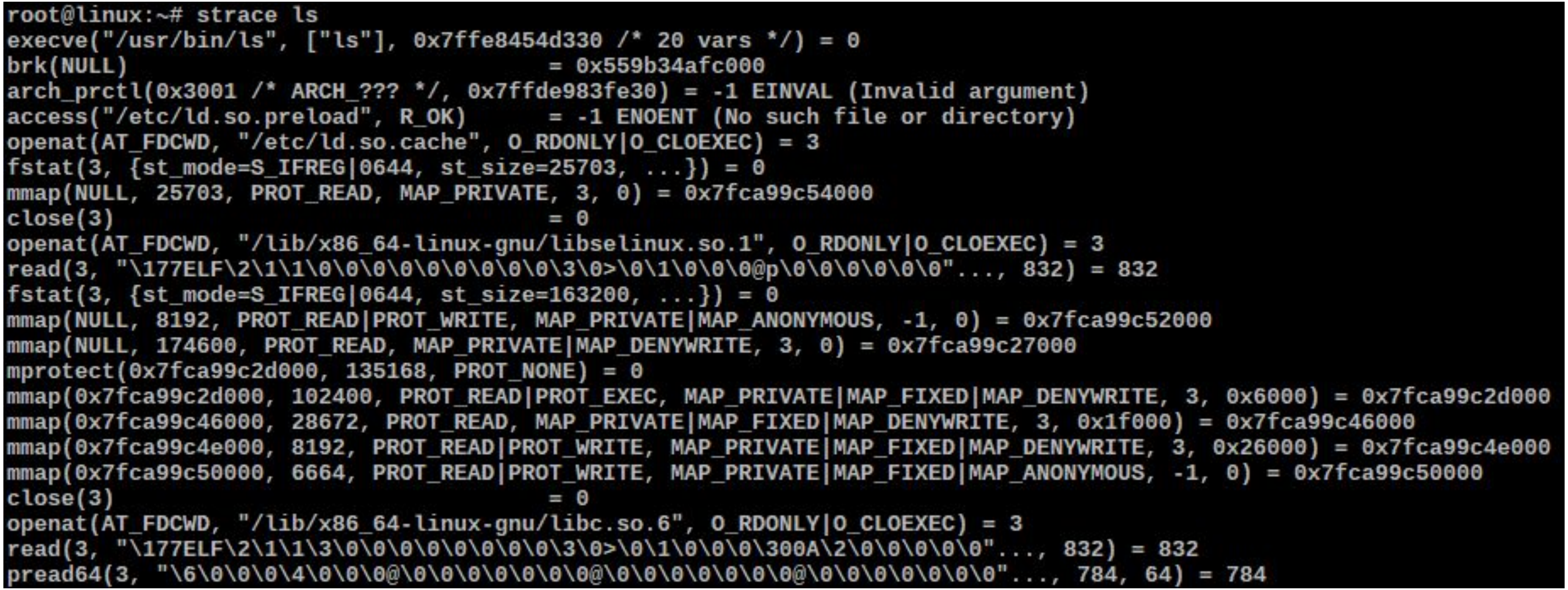
Strace

Strace enables us to track all the system calls made by an application. It lists all the system calls made by a process with the outputs of those calls. It is a very useful diagnostic, instructional, and debugging tool and can be used to discover the system resources in use by a workload.

Usage: **strace <command we want to trace>**

Strace Basic Usage

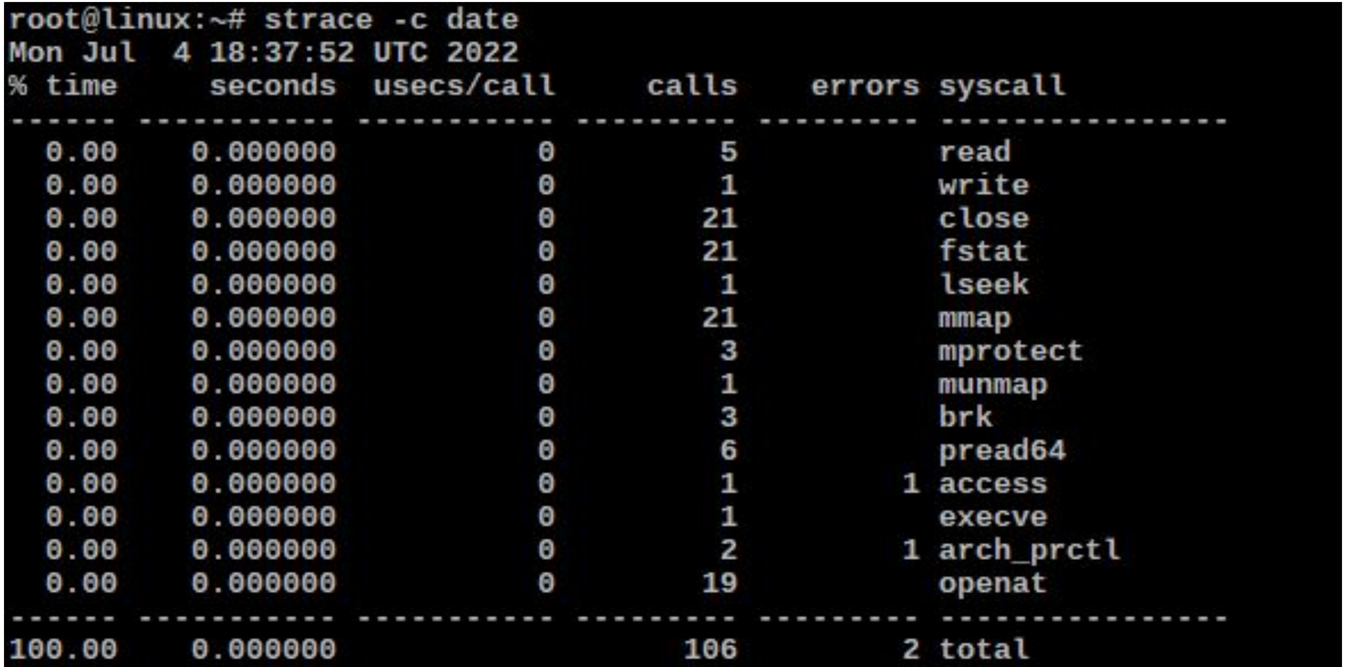
The following image shows strace ls output which shows system usage by “ls” command as it uses Linux System Calls to find and list information about the FILEs that reside under a directory.



```
root@linux:~# strace ls
execve("/usr/bin/ls", ["ls"], 0x7ffe8454d330 /* 20 vars */) = 0
brk(NULL)                               = 0x559b34afc000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffde983fe30) = -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=25703, ...}) = 0
mmap(NULL, 25703, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fca99c54000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libselinux.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0@p\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=163200, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fca99c52000
mmap(NULL, 174600, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fca99c27000
mprotect(0x7fca99c2d000, 135168, PROT_NONE) = 0
mmap(0x7fca99c2d000, 102400, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x6000) = 0x7fca99c2d000
mmap(0x7fca99c46000, 28672, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1f000) = 0x7fca99c46000
mmap(0x7fca99c4e000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x26000) = 0x7fca99c4e000
mmap(0x7fca99c50000, 6664, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fca99c50000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/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\0\300A\2\0\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
```

Gather Statistics

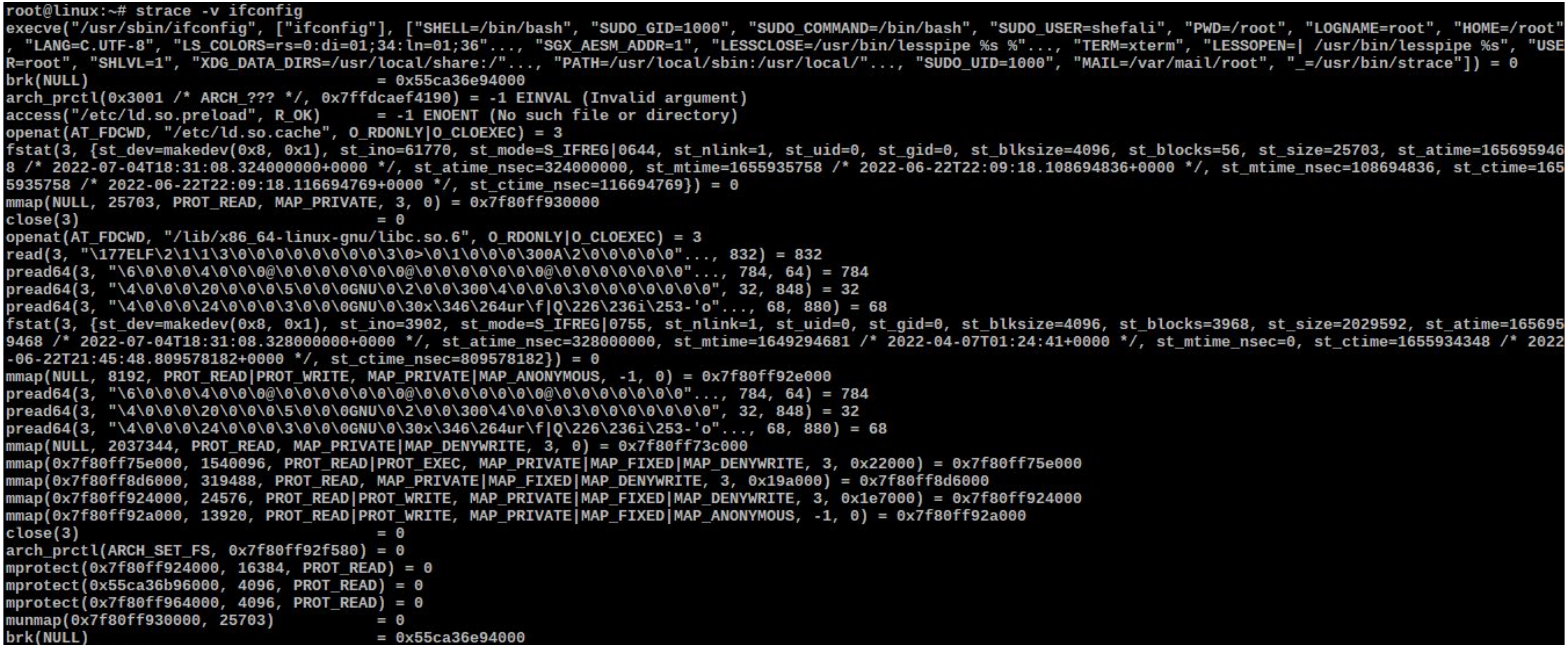
We can use the -c parameter to generate a report of the percentage of time spent in each system call, the total time in seconds, the microseconds per call, the total number of calls, the count of each system call that has failed with an error and the type of system call made - **strace -c <command>**



```
root@linux:~# strace -c date
Mon Jul  4 18:37:52 UTC 2022
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
  0.00    0.000000           0         5           read
  0.00    0.000000           0         1           write
  0.00    0.000000           0        21           close
  0.00    0.000000           0        21           fstat
  0.00    0.000000           0         1           lseek
  0.00    0.000000           0        21           mmap
  0.00    0.000000           0         3           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        19           openat
------ ----------- ----------- --------- --------- ----------------
100.00    0.000000                   106         2 total
```

Verbose Mode

strace command when run in verbose mode gives more detailed information about the system calls - **strace -v <command>**.



```
root@linux:~# strace -v ifconfig
execve("/usr/sbin/ifconfig", ["ifconfig"], ["SHELL=/bin/bash", "SUDO_GID=1000", "SUDO_COMMAND=/bin/bash", "SUDO_USER=shefali", "PWD=/root", "LOGNAME=root", "HOME=/root"
, "LANG=C.UTF-8", "LS_COLORS=rs=0:di=01;34:ln=01;36"..., "SGX_AESM_ADDR=1", "LESSCLOSE=/usr/bin/lesspipe %s %"..., "TERM=xterm", "LESSOPEN=| /usr/bin/lesspipe %s", "USE
R=root", "SHLVL=1", "XDG_DATA_DIRS=/usr/local/share:/"..., "PATH=/usr/local/sbin:/usr/local/"..., "SUDO_UID=1000", "MAIL=/var/mail/root", "_=/usr/bin/strace"]) = 0
brk(NULL)                               = 0x55ca36e94000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffdcaef4190) = -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_dev=makedev(0x8, 0x1), st_ino=61770, st_mode=S_IFREG|0644, st_nlink=1, st_uid=0, st_gid=0, st_blksize=4096, st_blocks=56, st_size=25703, st_atime=165695946
8 /* 2022-07-04T18:31:08.324000000+0000 */, st_atime_nsec=324000000, st_mtime=1655935758 /* 2022-06-22T22:09:18.108694836+0000 */, st_mtime_nsec=108694836, st_ctime=165
5935758 /* 2022-06-22T22:09:18.116694769+0000 */, st_ctime_nsec=116694769}) = 0
mmap(NULL, 25703, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f80ff930000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/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\0\300A\2\0\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(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, 848) = 32
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\30x\346\264ur\f|Q\226\236i\253-'o"..., 68, 880) = 68
fstat(3, {st_dev=makedev(0x8, 0x1), st_ino=3902, st_mode=S_IFREG|0755, st_nlink=1, st_uid=0, st_gid=0, st_blksize=4096, st_blocks=3968, st_size=2029592, st_atime=165695
9468 /* 2022-07-04T18:31:08.328000000+0000 */, st_atime_nsec=328000000, st_mtime=1649294681 /* 2022-04-07T01:24:41+0000 */, st_mtime_nsec=0, st_ctime=1655934348 /* 2022
-06-22T21:45:48.809578182+0000 */, st_ctime_nsec=809578182}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f80ff92e000
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(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, 848) = 32
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\30x\346\264ur\f|Q\226\236i\253-'o"..., 68, 880) = 68
mmap(NULL, 2037344, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f80ff73c000
mmap(0x7f80ff75e000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x22000) = 0x7f80ff75e000
mmap(0x7f80ff8d6000, 319488, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19a000) = 0x7f80ff8d6000
mmap(0x7f80ff924000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7f80ff924000
mmap(0x7f80ff92a000, 13920, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f80ff92a000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7f80ff92f580) = 0
mprotect(0x7f80ff924000, 16384, PROT_READ) = 0
mprotect(0x55ca36b96000, 4096, PROT_READ) = 0
mprotect(0x7f80ff964000, 4096, PROT_READ) = 0
munmap(0x7f80ff930000, 25703)           = 0
brk(NULL)                               = 0x55ca36e94000
```

What problems can be solved using strace?

- To see how a process interacts with the kernel.
- To see why a process is failing or hanging.
- For reverse engineering a process.
- To find the files on which a program depends.
- To see what arguments are being passed to each system call.
- For troubleshooting various problems related to the environment/operating system.

Cscope

Cscope is a command line tool which is used for browsing C, C++ or Java codebases. It can be used for finding:

- All the references to a symbol
- Global definitions
- Functions called by a function
- Functions calling a function
- Text strings
- Regular Expressions
- Files including a file

Installation: **apt-get install cscope**

Finding which system call belongs to which subsystem using cscope

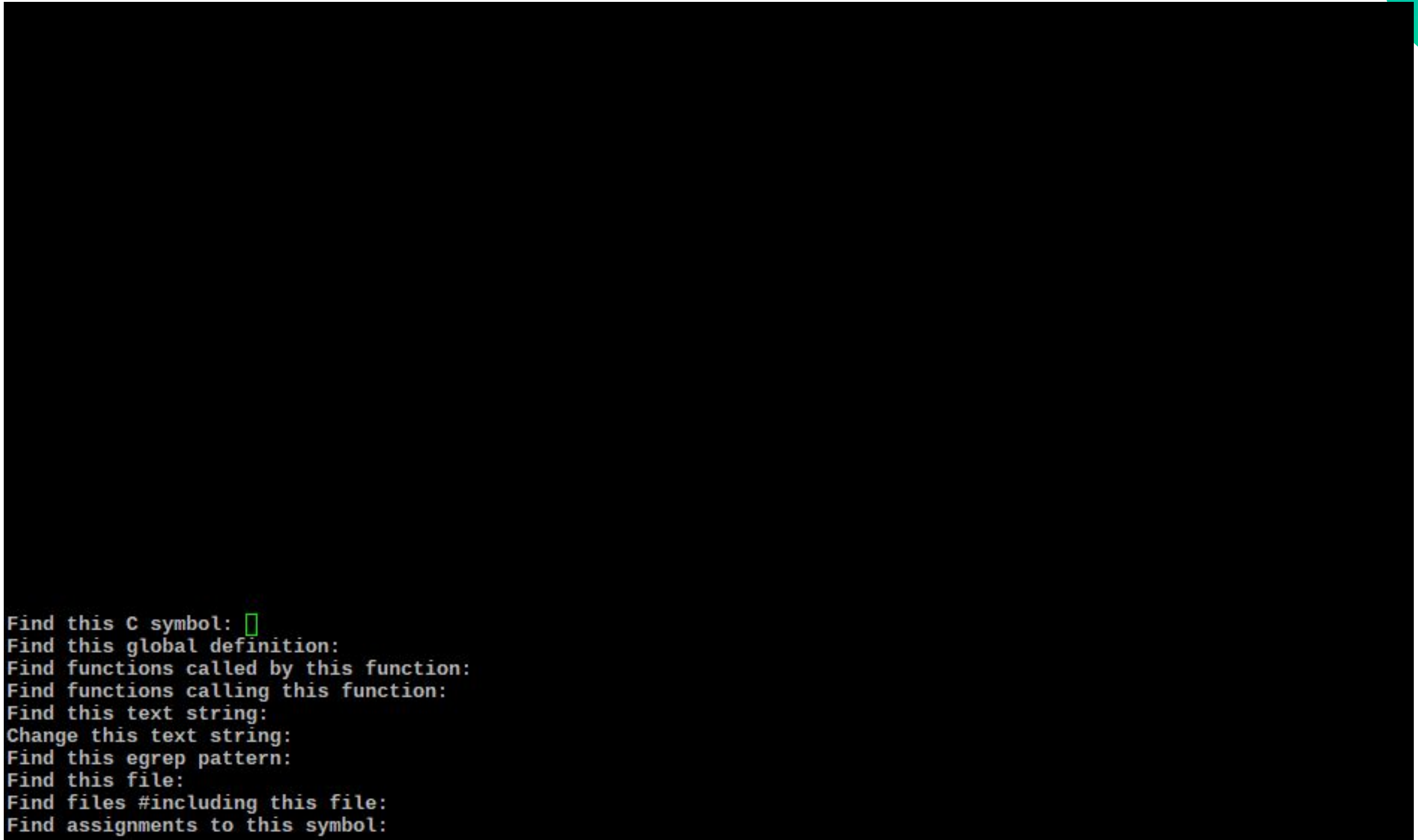
We can use cscope to find which system call belongs to which subsystem. This way we can find the kernel subsystems used by a process when it is executed. To use it navigate to the source code directory. Here we are analyzing the kernel source tree.

First let's checkout the latest Linux repository and build cscope database:

- **git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git linux**
- **cd linux**
- **cscope -R -p10 or cscope -d -p10**

Note: Run cscope -R to build the database (run it only once) and cscope -d -p10 to enter into the interactive mode of cscope. To get out of this mode press ctrl+d.

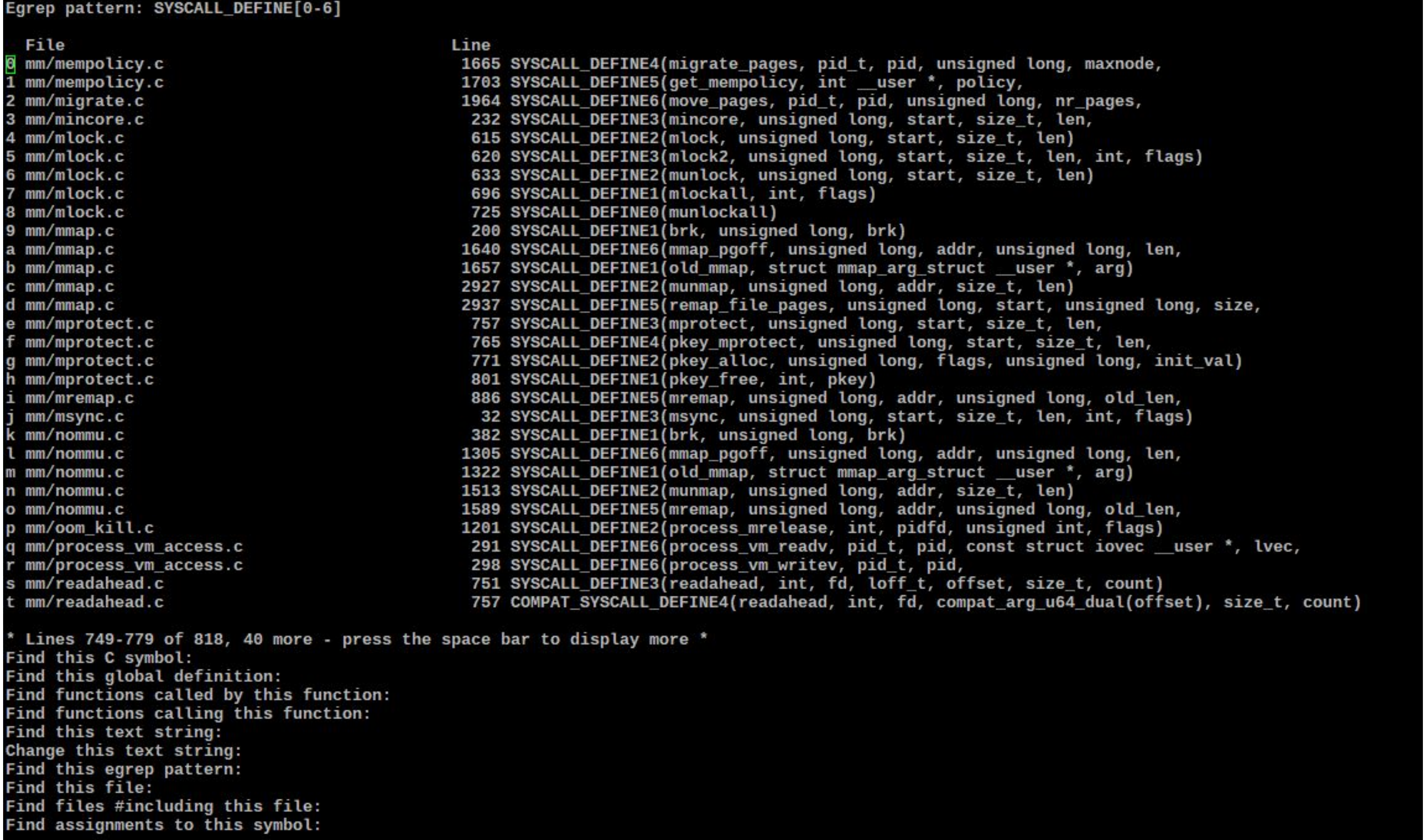
Cscope Console



```
Find this C symbol:
Find this global definition:
Find functions called by this function:
Find functions calling this function:
Find this text string:
Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this file:
Find assignments to this symbol:
```

Linux Syscalls

All the system calls are defined in the kernel using the **SYSCALL_DEFINE[0-6]** macro in their respective subsystem directory. We can search for this egrep pattern to find all the system calls and their subsystems (Press the Tab key to go back to the cscope options).



```
Egrep pattern: SYSCALL_DEFINE[0-6]

  File                                       Line
0 mm/mempolicy.c                             1665 SYSCALL_DEFINE4(migrate_pages, pid_t, pid, unsigned long, maxnode,
1 mm/mempolicy.c                             1703 SYSCALL_DEFINE5(get_mempolicy, int __user *, policy,
2 mm/migrate.c                               1964 SYSCALL_DEFINE6(move_pages, pid_t, pid, unsigned long, nr_pages,
3 mm/mincore.c                                232 SYSCALL_DEFINE3(mincore, unsigned long, start, size_t, len,
4 mm/mlock.c                                  615 SYSCALL_DEFINE2(mlock, unsigned long, start, size_t, len)
5 mm/mlock.c                                  620 SYSCALL_DEFINE3(mlock2, unsigned long, start, size_t, len, int, flags)
6 mm/mlock.c                                  633 SYSCALL_DEFINE2(munlock, unsigned long, start, size_t, len)
7 mm/mlock.c                                  696 SYSCALL_DEFINE1(mlockall, int, flags)
8 mm/mlock.c                                  725 SYSCALL_DEFINE0(munlockall)
9 mm/mmap.c                                   200 SYSCALL_DEFINE1(brk, unsigned long, brk)
a mm/mmap.c                                  1640 SYSCALL_DEFINE6(mmap_pgoff, unsigned long, addr, unsigned long, len,
b mm/mmap.c                                  1657 SYSCALL_DEFINE1(old_mmap, struct mmap_arg_struct __user *, arg)
c mm/mmap.c                                  2927 SYSCALL_DEFINE2(munmap, unsigned long, addr, size_t, len)
d mm/mmap.c                                  2937 SYSCALL_DEFINE5(remap_file_pages, unsigned long, start, unsigned long, size,
e mm/mprotect.c                               757 SYSCALL_DEFINE3(mprotect, unsigned long, start, size_t, len,
f mm/mprotect.c                               765 SYSCALL_DEFINE4(pkey_mprotect, unsigned long, start, size_t, len,
g mm/mprotect.c                               771 SYSCALL_DEFINE2(pkey_alloc, unsigned long, flags, unsigned long, init_val)
h mm/mprotect.c                               801 SYSCALL_DEFINE1(pkey_free, int, pkey)
i mm/mremap.c                                 886 SYSCALL_DEFINE5(mremap, unsigned long, addr, unsigned long, old_len,
j mm/msync.c                                   32 SYSCALL_DEFINE3(msync, unsigned long, start, size_t, len, int, flags)
k mm/nommu.c                                  382 SYSCALL_DEFINE1(brk, unsigned long, brk)
l mm/nommu.c                                 1305 SYSCALL_DEFINE6(mmap_pgoff, unsigned long, addr, unsigned long, len,
m mm/nommu.c                                 1322 SYSCALL_DEFINE1(old_mmap, struct mmap_arg_struct __user *, arg)
n mm/nommu.c                                 1513 SYSCALL_DEFINE2(munmap, unsigned long, addr, size_t, len)
o mm/nommu.c                                 1589 SYSCALL_DEFINE5(mremap, unsigned long, addr, unsigned long, old_len,
p mm/oom_kill.c                              1201 SYSCALL_DEFINE2(process_mrelease, int, pidfd, unsigned int, flags)
q mm/process_vm_access.c                      291 SYSCALL_DEFINE6(process_vm_readv, pid_t, pid, const struct iovec __user *, lvec,
r mm/process_vm_access.c                      298 SYSCALL_DEFINE6(process_vm_writev, pid_t, pid,
s mm/readahead.c                              751 SYSCALL_DEFINE3(readahead, int, fd, loff_t, offset, size_t, count)
t mm/readahead.c                              757 COMPAT_SYSCALL_DEFINE4(readahead, int, fd, compat_arg_u64_dual(offset), size_t, count)

* Lines 749-779 of 818, 40 more - press the space bar to display more *
Find this C symbol:
Find this global definition:
Find functions called by this function:
Find functions calling this function:
Find this text string:
Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this file:
Find assignments to this symbol:
```

stress-ng

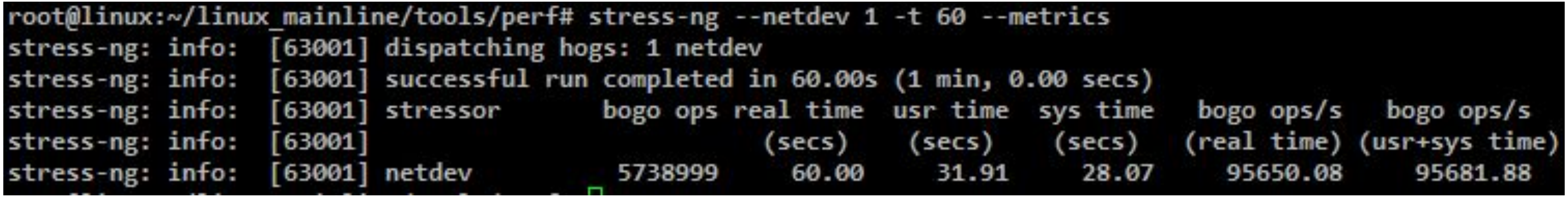
stress-ng is used for performing stress testing on the kernel. It allows you to exercise various physical subsystems of the computer, as well as interfaces of the OS kernel, using **stressors**. They are available for CPU, CPU cache, devices, I/O, interrupts, file system, memory, network, operating system, pipelines, schedulers, virtual machines.

Installation: **apt-get install stress-ng**

Netdev Stressor

The netdev stressor starts N workers that exercise various netdevice ioctl commands across all the available network devices. The ioctls exercised by this stressor are as follows: SIOCGIFCONF, SIOCGIFINDEX, SIOCGIFNAME, SIOCGIFFLAGS, SIOCGIFADDR, SIOCGIFNETMASK, SIOCGIFMETRIC, SIOCGIFMTU, SIOCGIFHWADDR, SIOCGIFMAP and SIOCGIFTXQLEN).

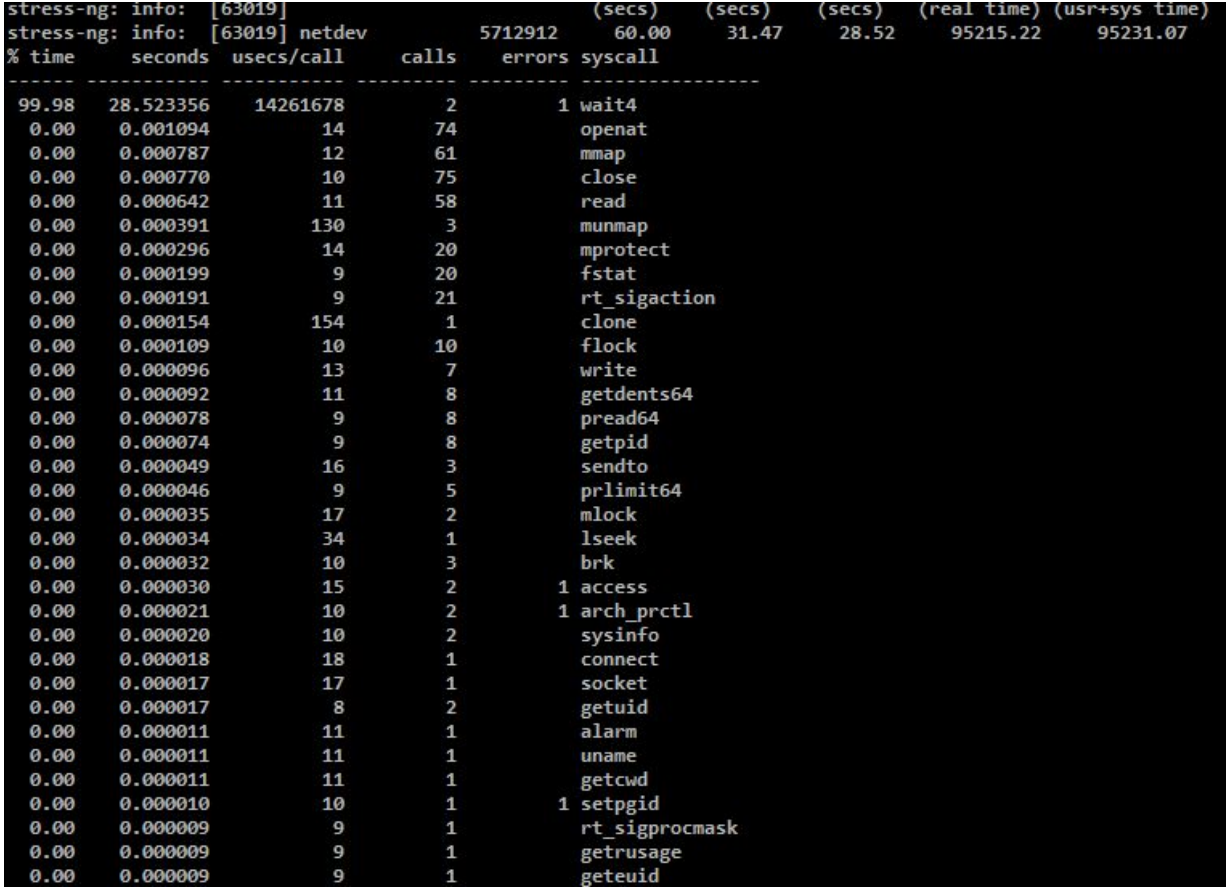
Usage: **stress-ng --netdev 1 -t 60 --metrics**



```
root@linux:~/linux_mainline/tools/perf# stress-ng --netdev 1 -t 60 --metrics
stress-ng: info:  [63001] dispatching hogs: 1 netdev
stress-ng: info:  [63001] successful run completed in 60.00s (1 min, 0.00 secs)
stress-ng: info:  [63001] stressor       bogo ops real time  usr time  sys time   bogo ops/s   bogo ops/s
stress-ng: info:  [63001]                           (secs)    (secs)    (secs)   (real time) (usr+sys time)
stress-ng: info:  [63001] netdev          5738999     60.00     31.91     28.07     95650.08     95681.88
```

Tracing stress-ng netdev stressor workload under Strace

Using strace to collect the trace for netdev stressor - **strace -c stress-ng --netdev 1 -t 60 --metrics**



```
stress-ng: info:  [63019]                          (secs)    (secs)    (secs)   (real time) (usr+sys time)
stress-ng: info:  [63019] netdev           5712912     60.00     31.47     28.52     95215.22     95231.07
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 99.98   28.523356    14261678         2         1 wait4
  0.00    0.001094          14        74           openat
  0.00    0.000787          12        61           mmap
  0.00    0.000770          10        75           close
  0.00    0.000642          11        58           read
  0.00    0.000391         130         3           munmap
  0.00    0.000296          14        20           mprotect
  0.00    0.000199           9        20           fstat
  0.00    0.000191           9        21           rt_sigaction
  0.00    0.000154         154         1           clone
  0.00    0.000109          10        10           flock
  0.00    0.000096          13         7           write
  0.00    0.000092          11         8           getdents64
  0.00    0.000078           9         8           pread64
  0.00    0.000074           9         8           getpid
  0.00    0.000049          16         3           sendto
  0.00    0.000046           9         5           prlimit64
  0.00    0.000035          17         2           mlock
  0.00    0.000034          34         1           lseek
  0.00    0.000032          10         3           brk
  0.00    0.000030          15         2         1 access
  0.00    0.000021          10         2         1 arch_prctl
  0.00    0.000020          10         2           sysinfo
  0.00    0.000018          18         1           connect
  0.00    0.000017          17         1           socket
  0.00    0.000017           8         2           getuid
  0.00    0.000011          11         1           alarm
  0.00    0.000011          11         1           uname
  0.00    0.000011          11         1           getcwd
  0.00    0.000010          10         1         1 setpgid
  0.00    0.000009           9         1           rt_sigprocmask
  0.00    0.000009           9         1           getrusage
  0.00    0.000009           9         1           geteuid
```

ELISA SUMMIT
SEPTEMBER 7 - 8, 2022 | VIRTUAL
@PROJECTELISA

Subsystem Usage by the Workload

System Call	Frequency	Linux Subsystem	System Call Entry Point (API)
openat	74	Filesystem	sys_openat()
close	75	Filesystem	sys_close()
read	58	Filesystem	sys_read()
fstat	20	Filesystem	sys_fstat()
flock	10	Filesystem	sys_flock()
write	7	Filesystem	sys_write()
getdents64	8	Filesystem	sys_getdents64()
pread64	8	Filesystem	sys_pread64()
lseek	1	Filesystem	sys_lseek()
access	2	Filesystem	sys_access()
getcwd	1	Filesystem	sys_getcwd()
execve	1	Filesystem	sys_execve()

mmap	61	Memory Mgmt.	sys_mmap()
munmap	3	Memory Mgmt.	sys_munmap()
mprotect	20	Memory Mgmt.	sys_mprotect()
mlock	2	Memory Mgmt.	sys_mlock()
brk	3	Memory Mgmt.	sys_brk()
rt_sigaction	21	Signal	sys_rt_sigaction()
rt_sigprocmask	1	Signal	sys_rt_sigprocmask()
sigaltstack	1	Signal	sys_sigaltstack()
rt_sigreturn	1	Signal	sys_rt_sigreturn()
getpid	8	Process Mgmt.	sys_getpid()
prlimit64	5	Process Mgmt.	sys_prlimit64()
arch_prctl	2	Process Mgmt.	sys_arch_prctl()
sysinfo	2	Process Mgmt.	sys_sysinfo()

getuid	2	Process Mgmt.	sys_getuid()
uname	1	Process Mgmt.	sys_uname()
setpgid	1	Process Mgmt.	sys_setpgid()
getrusage	1	Process Mgmt.	sys_getrusage()
geteuid	1	Process Mgmt.	sys_geteuid()
getppid	1	Process Mgmt.	sys_getppid()
sendto	3	Network	sys_sendto()
connect	1	Network	sys_connect()
socket	1	Network	sys_socket()
clone	1	Process Mgmt.	sys_clone()
set_tid_address	1	Process Mgmt.	sys_set_tid_address()
wait4	2	Time	sys_wait4()
alarm	1	Time	sys_alarm()
set_robust_list	1	Futex	sys_set_robust_list()

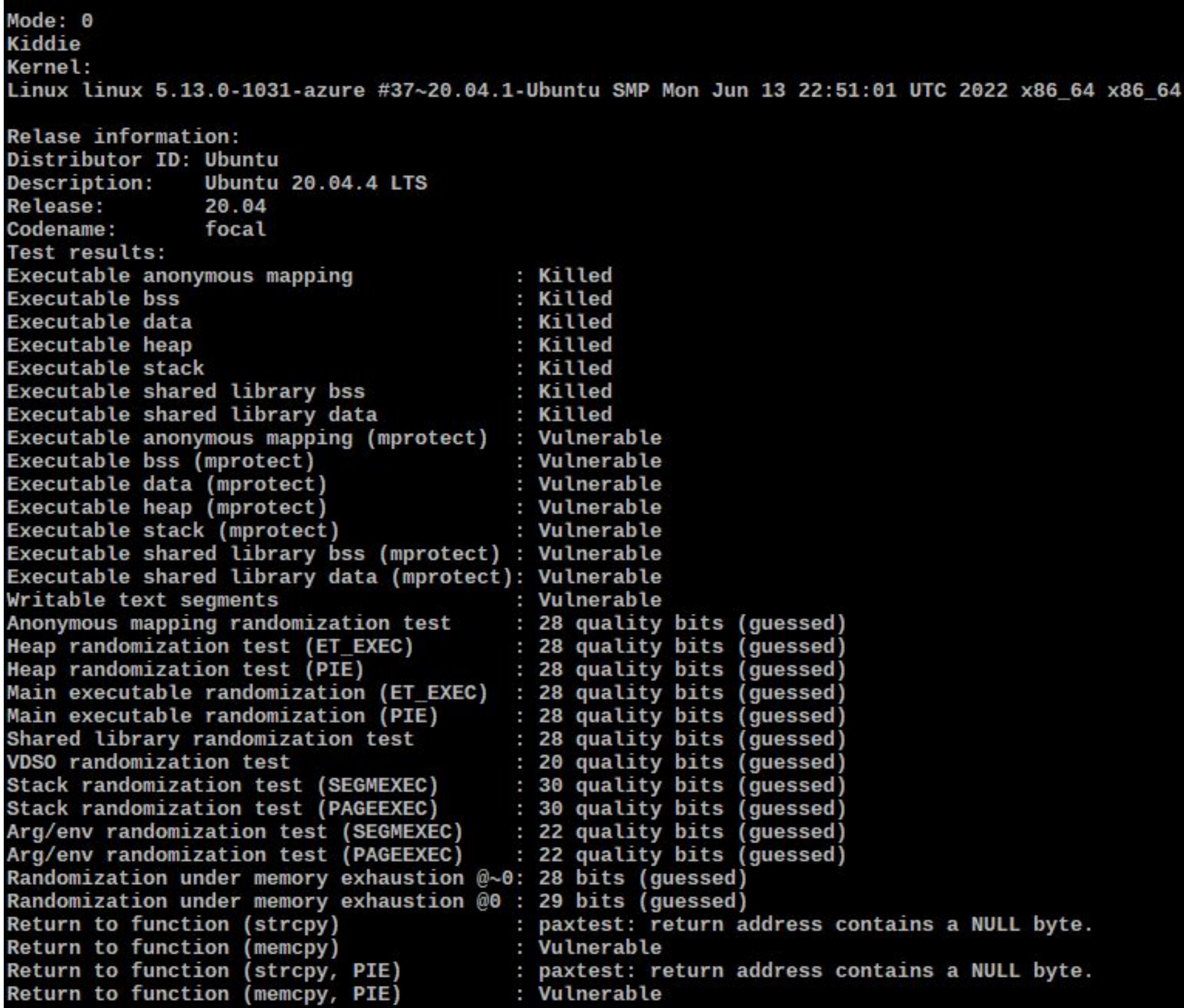
Paxtest

Paxtest is a program that tests buffer overflows in the kernel. It tests kernel enforcements over memory usage. Generally, execution in some memory segments makes buffer overflows possible. It runs a set of programs that attempt to subvert memory usage. It is used as a regression test suite for PaX, but might be useful to test other memory protection patches for the kernel.

Installation: **apt-get install paxtest**

Usage: **paxtest <mode>**

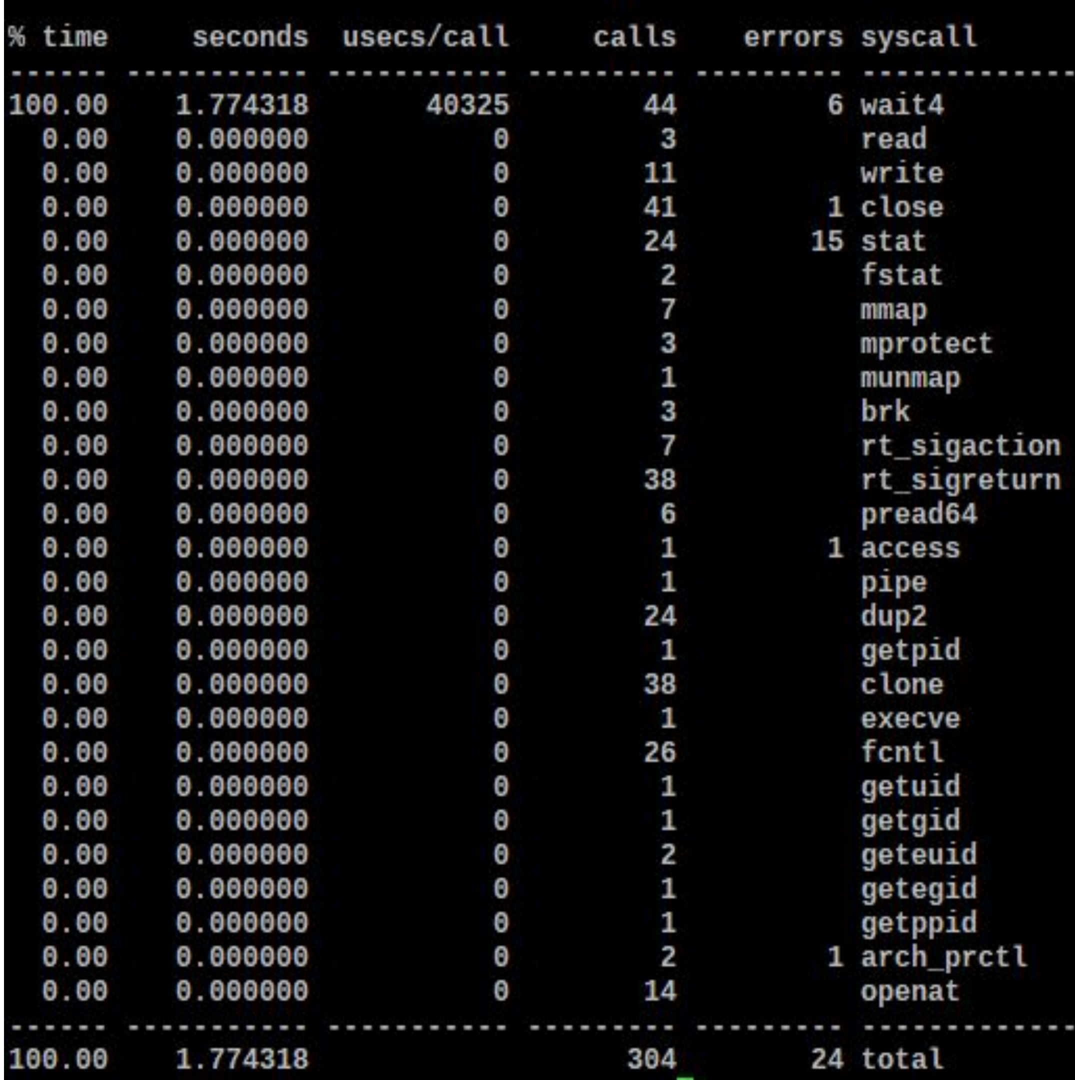
$ paxtest kiddie



```
Mode: 0
Kiddie
Kernel:
Linux linux 5.13.0-1031-azure #37~20.04.1-Ubuntu SMP Mon Jun 13 22:51:01 UTC 2022 x86_64 x86_64

Relase information:
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.4 LTS
Release:        20.04
Codename:       focal
Test results:
Executable anonymous mapping             : Killed
Executable bss                           : Killed
Executable data                          : Killed
Executable heap                          : Killed
Executable stack                         : Killed
Executable shared library bss            : Killed
Executable shared library data           : Killed
Executable anonymous mapping (mprotect)  : Vulnerable
Executable bss (mprotect)                : Vulnerable
Executable data (mprotect)               : Vulnerable
Executable heap (mprotect)               : Vulnerable
Executable stack (mprotect)              : Vulnerable
Executable shared library bss (mprotect) : Vulnerable
Executable shared library data (mprotect): Vulnerable
Writable text segments                   : Vulnerable
Anonymous mapping randomization test     : 28 quality bits (guessed)
Heap randomization test (ET_EXEC)        : 28 quality bits (guessed)
Heap randomization test (PIE)            : 28 quality bits (guessed)
Main executable randomization (ET_EXEC)  : 28 quality bits (guessed)
Main executable randomization (PIE)      : 28 quality bits (guessed)
Shared library randomization test        : 28 quality bits (guessed)
VDSO randomization test                  : 20 quality bits (guessed)
Stack randomization test (SEGMEXEC)      : 30 quality bits (guessed)
Stack randomization test (PAGEEXEC)      : 30 quality bits (guessed)
Arg/env randomization test (SEGMEXEC)    : 22 quality bits (guessed)
Arg/env randomization test (PAGEEXEC)    : 22 quality bits (guessed)
Randomization under memory exhaustion @~0: 28 bits (guessed)
Randomization under memory exhaustion @0 : 29 bits (guessed)
Return to function (strcpy)              : paxtest: return address contains a NULL byte.
Return to function (memcpy)              : Vulnerable
Return to function (strcpy, PIE)         : paxtest: return address contains a NULL byte.
Return to function (memcpy, PIE)         : Vulnerable
```

Tracing Paxtest workload under Strace

Using strace to collect the trace when we run paxtest under kiddie mode - **strace -c paxtest kiddie**.



```
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ------------
100.00    1.774318       40325        44         6 wait4
  0.00    0.000000           0         3           read
  0.00    0.000000           0        11           write
  0.00    0.000000           0        41         1 close
  0.00    0.000000           0        24        15 stat
  0.00    0.000000           0         2           fstat
  0.00    0.000000           0         7           mmap
  0.00    0.000000           0         3           mprotect
  0.00    0.000000           0         1           munmap
  0.00    0.000000           0         3           brk
  0.00    0.000000           0         7           rt_sigaction
  0.00    0.000000           0        38           rt_sigreturn
  0.00    0.000000           0         6           pread64
  0.00    0.000000           0         1         1 access
  0.00    0.000000           0         1           pipe
  0.00    0.000000           0        24           dup2
  0.00    0.000000           0         1           getpid
  0.00    0.000000           0        38           clone
  0.00    0.000000           0         1           execve
  0.00    0.000000           0        26           fcntl
  0.00    0.000000           0         1           getuid
  0.00    0.000000           0         1           getgid
  0.00    0.000000           0         2           geteuid
  0.00    0.000000           0         1           getegid
  0.00    0.000000           0         1           getppid
  0.00    0.000000           0         2         1 arch_prctl
  0.00    0.000000           0        14           openat
------ ----------- ----------- --------- --------- ------------
100.00    1.774318                   304        24 total
```

Subsystem Usage by the Workload

System Call	Frequency	Linux Subsystem	System Call Entry Point (API)
read	3	Filesystem	sys_read()
write	11	Filesystem	sys_write()
close	41	Filesystem	sys_close()
stat	24	Filesystem	sys_stat()
fstat	2	Filesystem	sys_fstat()
pread64	6	Filesystem	sys_pread64()
access	1	Filesystem	sys_access()
pipe	1	Filesystem	sys_pipe()
dup2	24	Filesystem	sys_dup2()
execve	1	Filesystem	sys_execve()
fcntl	26	Filesystem	sys_fcntl()
openat	14	Filesystem	sys_openat()

rt_sigaction	7	Signal	sys_rt_sigaction()
rt_sigreturn	38	Signal	sys_rt_sigreturn()
wait4	44	Time	sys_wait4()
mmap	7	Memory Mgmt.	sys_mmap()
mprotect	3	Memory Mgmt.	sys_mprotect()
munmap	1	Memory Mgmt.	sys_munmap()
brk	3	Memory Mgmt.	sys_brk()
getpid	1	Process Mgmt.	sys_getpid()
getuid	1	Process Mgmt.	sys_getuid()
getgid	1	Process Mgmt.	sys_getgid()
geteuid	2	Process Mgmt.	sys_geteuid()
getegid	1	Process Mgmt.	sys_getegid()
getppid	1	Process Mgmt.	sys_getppid()
arch_prctl	2	Process Mgmt.	sys_arch_prctl()
clone	38	Process Mgmt.	sys_clone()

Strace to determine Workload Usage

Thus, strace a very useful diagnostic, instructional, and debugging tool and can be used to discover the system resources in use by a workload. Once we discover and understand the workload needs, we can focus on them to avoid regressions and use it to evaluate safety considerations.

Strace to determine Workload Usage

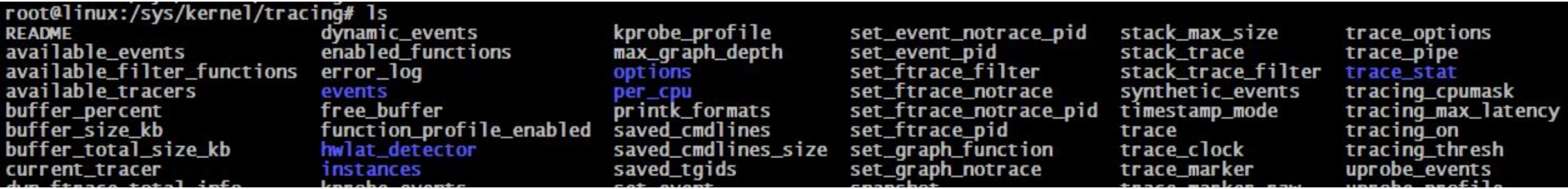
This method of tracing tells us the system calls invoked by the workload and doesn't include all the system calls that can be invoked. In addition to that this trace tells us just the code paths within these system calls that are invoked. As an example, if a workload opens a file and reads from it successfully, then the success path is the one that is traced. Any error paths in that system call will not be traced. If there is a workload that provides full coverage the method outlined here will trace and find all possible code paths. **The completeness of the system usage information depends on the completeness of coverage of a workload.**

Ftrace

- Ftrace is an internal tracer designed to help out developers and designers of systems to find what is going on inside the kernel.
- It allows you to trace kernel function calls.
- The API to interface with Ftrace is located in the Debugfs filesystem, which is typically mounted at /sys/kernel/debug in most distributions by default.

Important files of the Debugfs file system

- **trace:** It shows the output of an ftrace trace.
- **tracing_on:** To disable or enable recording to the ftrace buffer.
- **available_tracers:** The list of ftrace tracers that have been compiled into the kernel.
- **available_filter_functions:** All the kernel functions available for tracing.
- **available_events:** All the events available for tracing



```
root@linux:/sys/kernel/tracing# ls
README                      dynamic_events            kprobe_profile       set_event_notrace_pid   stack_max_size      trace_options
available_events            enabled_functions         max_graph_depth      set_event_pid           stack_trace         trace_pipe
available_filter_functions  error_log                 options              set_ftrace_filter       stack_trace_filter  trace_stat
available_tracers           events                    per_cpu              set_ftrace_notrace      synthetic_events    tracing_cpumask
buffer_percent              free_buffer               printk_formats       set_ftrace_notrace_pid  timestamp_mode      tracing_max_latency
buffer_size_kb              function_profile_enabled  saved_cmdlines       set_ftrace_pid          trace               tracing_on
buffer_total_size_kb        hwlat_detector            saved_cmdlines_size  set_graph_function      trace_clock         tracing_thresh
current_tracer              instances                 saved_tgids          set_graph_notrace       trace_marker        uprobe_events
dyn_ftrace_total_info       kprobe_events             set_event            snapshot                trace_marker_raw    uprobe_profile
```

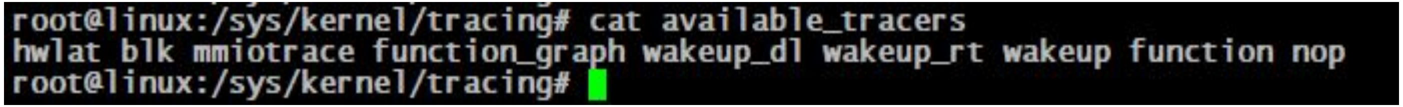
Available Tracers

Brief description of commonly used tracers:

- **nop:** No tracer enabled.
- **function:** It traces the entry of kernel functions.
- **function_graph:** It traces both the entry and exit of kernel functions and provides the ability to draw a graph of function calls similar to C source code.
- **blk:** The block tracer
- **mmiotrace:** It traces the interaction between drivers and hardware.

To set a tracer:

$ **echo <tracer> > current_tracer**



```
root@linux:/sys/kernel/tracing# cat available_tracers
hwlat blk mmiotrace function_graph wakeup_dl wakeup_rt wakeup function nop
root@linux:/sys/kernel/tracing#
```

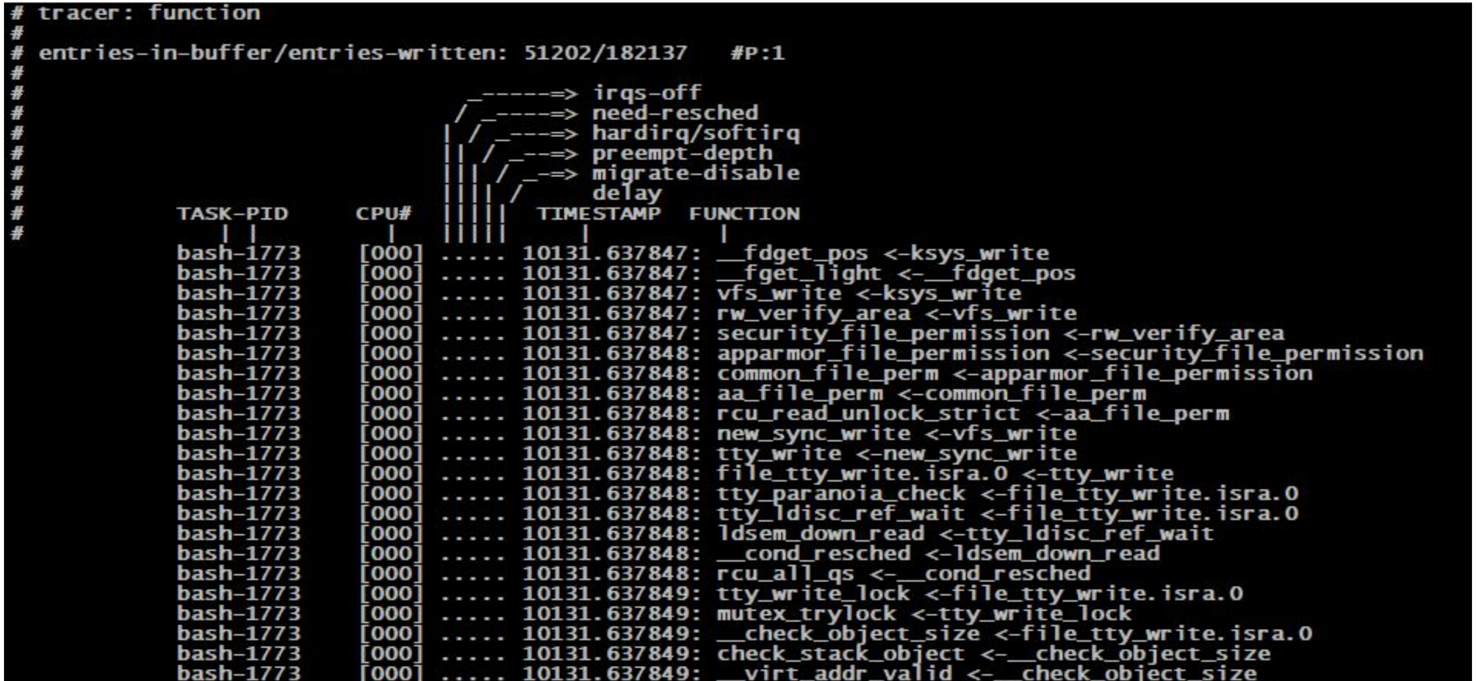
Gathering trace from the system

$ **echo 1 > tracing_on** # to turn on tracing

$ **echo function > current_tracer**

$ **echo 0 > tracing_on** # to turn off tracing

$ **less trace** # to display the trace data



```
# tracer: function
#
# entries-in-buffer/entries-written: 51202/182137   #P:1
#
#                                _-----=> irqs-off
#                               / _----=> need-resched
#                              | / _---=> hardirq/softirq
#                              || / _--=> preempt-depth
#                              ||| / _-=> migrate-disable
#                              |||| /     delay
#           TASK-PID     CPU#  |||||  TIMESTAMP  FUNCTION
#              | |         |   |||||     |         |
           bash-1773    [000] ..... 10131.637847: __fdget_pos <-ksys_write
           bash-1773    [000] ..... 10131.637847: __fget_light <-__fdget_pos
           bash-1773    [000] ..... 10131.637847: vfs_write <-ksys_write
           bash-1773    [000] ..... 10131.637847: rw_verify_area <-vfs_write
           bash-1773    [000] ..... 10131.637847: security_file_permission <-rw_verify_area
           bash-1773    [000] ..... 10131.637848: apparmor_file_permission <-security_file_permission
           bash-1773    [000] ..... 10131.637848: common_file_perm <-apparmor_file_permission
           bash-1773    [000] ..... 10131.637848: aa_file_perm <-common_file_perm
           bash-1773    [000] ..... 10131.637848: rcu_read_unlock_strict <-aa_file_perm
           bash-1773    [000] ..... 10131.637848: new_sync_write <-vfs_write
           bash-1773    [000] ..... 10131.637848: tty_write <-new_sync_write
           bash-1773    [000] ..... 10131.637848: file_tty_write.isra.0 <-tty_write
           bash-1773    [000] ..... 10131.637848: tty_paranoia_check <-file_tty_write.isra.0
           bash-1773    [000] ..... 10131.637848: tty_ldisc_ref_wait <-file_tty_write.isra.0
           bash-1773    [000] ..... 10131.637848: ldsem_down_read <-tty_ldisc_ref_wait
           bash-1773    [000] ..... 10131.637848: __cond_resched <-ldsem_down_read
           bash-1773    [000] ..... 10131.637848: rcu_all_qs <-__cond_resched
           bash-1773    [000] ..... 10131.637849: tty_write_lock <-file_tty_write.isra.0
           bash-1773    [000] ..... 10131.637849: mutex_trylock <-tty_write_lock
           bash-1773    [000] ..... 10131.637849: __check_object_size <-file_tty_write.isra.0
           bash-1773    [000] ..... 10131.637849: check_stack_object <-__check_object_size
           bash-1773    [000] ..... 10131.637849: __virt_addr_valid <-__check_object_size
```

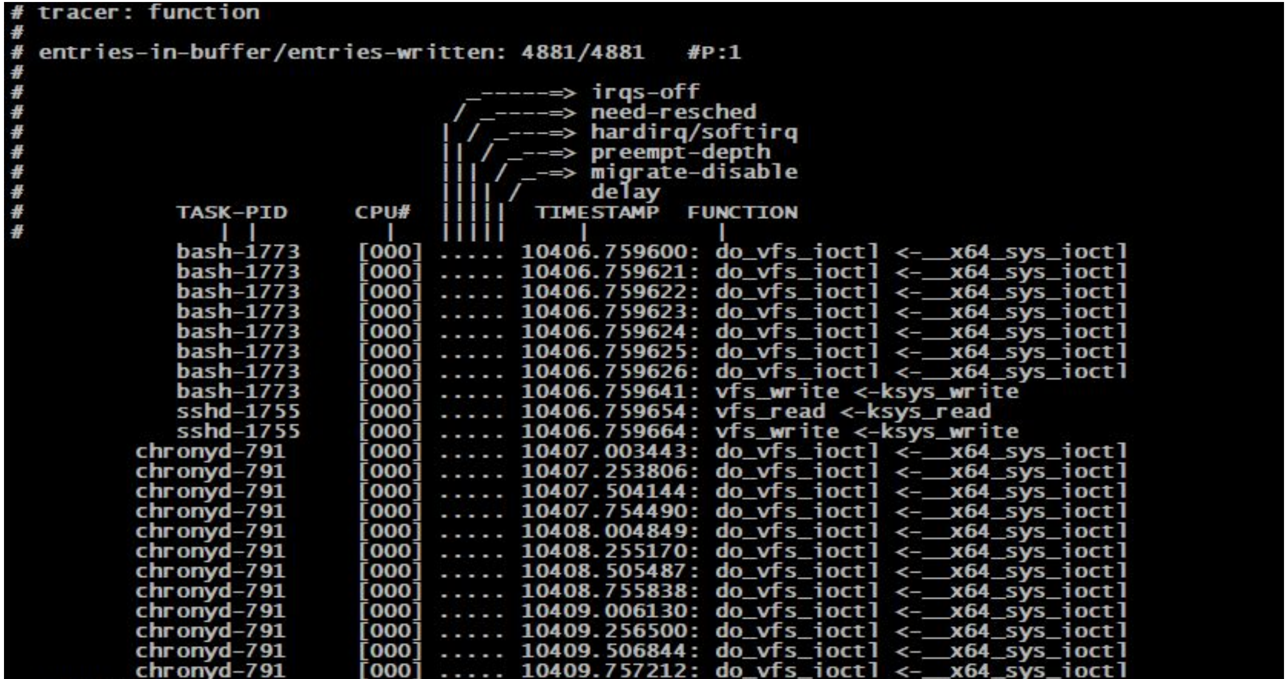
Function filter

If you want to trace only a particular function then specify it in the **set_ftrace_filter** file.

$ **echo *vfs* > set_ftrace_filter**

If you want to trace functions only of a specific module then:

$ **echo '*:mod:i2c_core' > set_ftrace_filter**



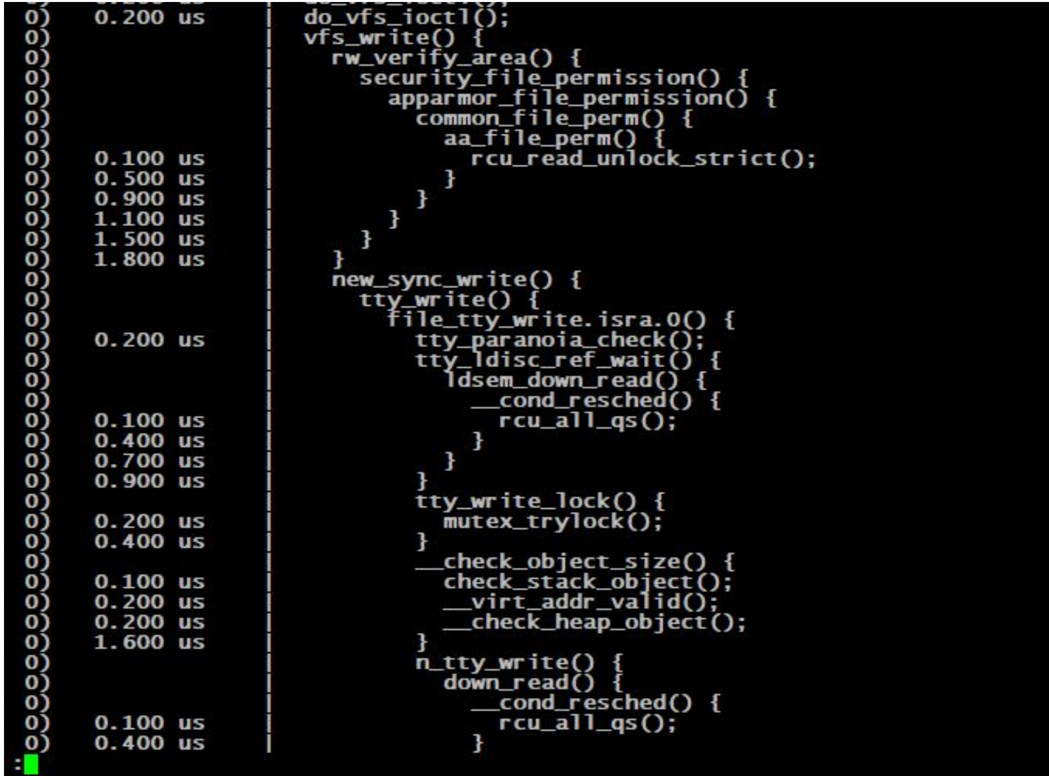
```
# tracer: function
#
# entries-in-buffer/entries-written: 4881/4881   #P:1
#
#                                _-----=> irqs-off
#                               / _----=> need-resched
#                              | / _---=> hardirq/softirq
#                              || / _--=> preempt-depth
#                              ||| / _-=> migrate-disable
#                              |||| /     delay
#           TASK-PID     CPU#  |||||  TIMESTAMP  FUNCTION
#              | |         |   |||||     |         |
           bash-1773    [000] ..... 10406.759600: do_vfs_ioctl <-__x64_sys_ioctl
           bash-1773    [000] ..... 10406.759621: do_vfs_ioctl <-__x64_sys_ioctl
           bash-1773    [000] ..... 10406.759622: do_vfs_ioctl <-__x64_sys_ioctl
           bash-1773    [000] ..... 10406.759623: do_vfs_ioctl <-__x64_sys_ioctl
           bash-1773    [000] ..... 10406.759624: do_vfs_ioctl <-__x64_sys_ioctl
           bash-1773    [000] ..... 10406.759625: do_vfs_ioctl <-__x64_sys_ioctl
           bash-1773    [000] ..... 10406.759626: do_vfs_ioctl <-__x64_sys_ioctl
           bash-1773    [000] ..... 10406.759641: vfs_write <-ksys_write
           sshd-1755    [000] ..... 10406.759654: vfs_read <-ksys_read
           sshd-1755    [000] ..... 10406.759664: vfs_write <-ksys_write
        chronyd-791     [000] ..... 10407.003443: do_vfs_ioctl <-__x64_sys_ioctl
        chronyd-791     [000] ..... 10407.253806: do_vfs_ioctl <-__x64_sys_ioctl
        chronyd-791     [000] ..... 10407.504144: do_vfs_ioctl <-__x64_sys_ioctl
        chronyd-791     [000] ..... 10407.754490: do_vfs_ioctl <-__x64_sys_ioctl
        chronyd-791     [000] ..... 10408.004849: do_vfs_ioctl <-__x64_sys_ioctl
        chronyd-791     [000] ..... 10408.255170: do_vfs_ioctl <-__x64_sys_ioctl
        chronyd-791     [000] ..... 10408.505487: do_vfs_ioctl <-__x64_sys_ioctl
        chronyd-791     [000] ..... 10408.755838: do_vfs_ioctl <-__x64_sys_ioctl
        chronyd-791     [000] ..... 10409.006130: do_vfs_ioctl <-__x64_sys_ioctl
        chronyd-791     [000] ..... 10409.256500: do_vfs_ioctl <-__x64_sys_ioctl
        chronyd-791     [000] ..... 10409.506844: do_vfs_ioctl <-__x64_sys_ioctl
        chronyd-791     [000] ..... 10409.757212: do_vfs_ioctl <-__x64_sys_ioctl
```

Function graph filter

Functions listed in the **set_graph_function** file will cause the function graph tracer to only trace these functions and the function they call.

$ **echo *vfs* > set_graph_function**

$ **echo function_graph > current_tracer**

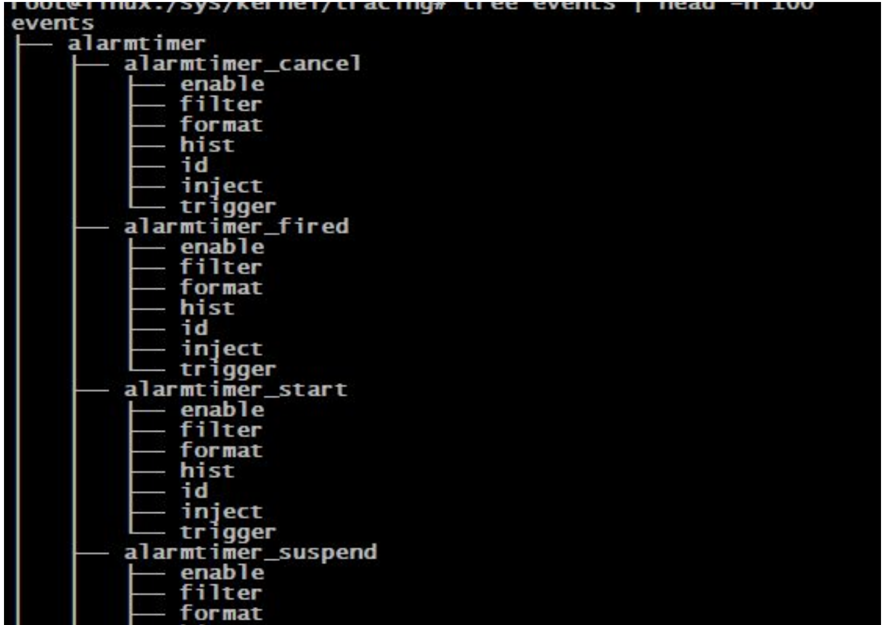


```
 0)   0.200 us    |      do_vfs_ioctl();
 0)               |      vfs_write() {
 0)               |        rw_verify_area() {
 0)               |          security_file_permission() {
 0)               |            apparmor_file_permission() {
 0)               |              common_file_perm() {
 0)               |                aa_file_perm() {
 0)   0.100 us    |                  rcu_read_unlock_strict();
 0)   0.500 us    |                }
 0)   0.900 us    |              }
 0)   1.100 us    |            }
 0)   1.500 us    |          }
 0)   1.800 us    |        }
 0)               |        new_sync_write() {
 0)               |          tty_write() {
 0)               |            file_tty_write.isra.0() {
 0)   0.200 us    |              tty_paranoia_check();
 0)               |              tty_ldisc_ref_wait() {
 0)               |                ldsem_down_read() {
 0)               |                  __cond_resched() {
 0)   0.100 us    |                    rcu_all_qs();
 0)   0.400 us    |                  }
 0)   0.700 us    |                }
 0)   0.900 us    |              }
 0)               |              tty_write_lock() {
 0)   0.200 us    |                mutex_trylock();
 0)   0.400 us    |              }
 0)               |              __check_object_size() {
 0)   0.100 us    |                check_stack_object();
 0)   0.200 us    |                __virt_addr_valid();
 0)   0.200 us    |                __check_heap_object();
 0)   1.600 us    |              }
 0)               |              n_tty_write() {
 0)               |                down_read() {
 0)               |                  __cond_resched() {
 0)   0.100 us    |                    rcu_all_qs();
 0)   0.400 us    |                  }
:
```

Event tracing

Throughout the kernel is hundreds of static event points that can be enabled via the tracefs file system to see what is going on in certain parts of the kernel. These events / tracepoints are very useful for retrieving the state of the kernel at a particular instant.

$ **tree events**



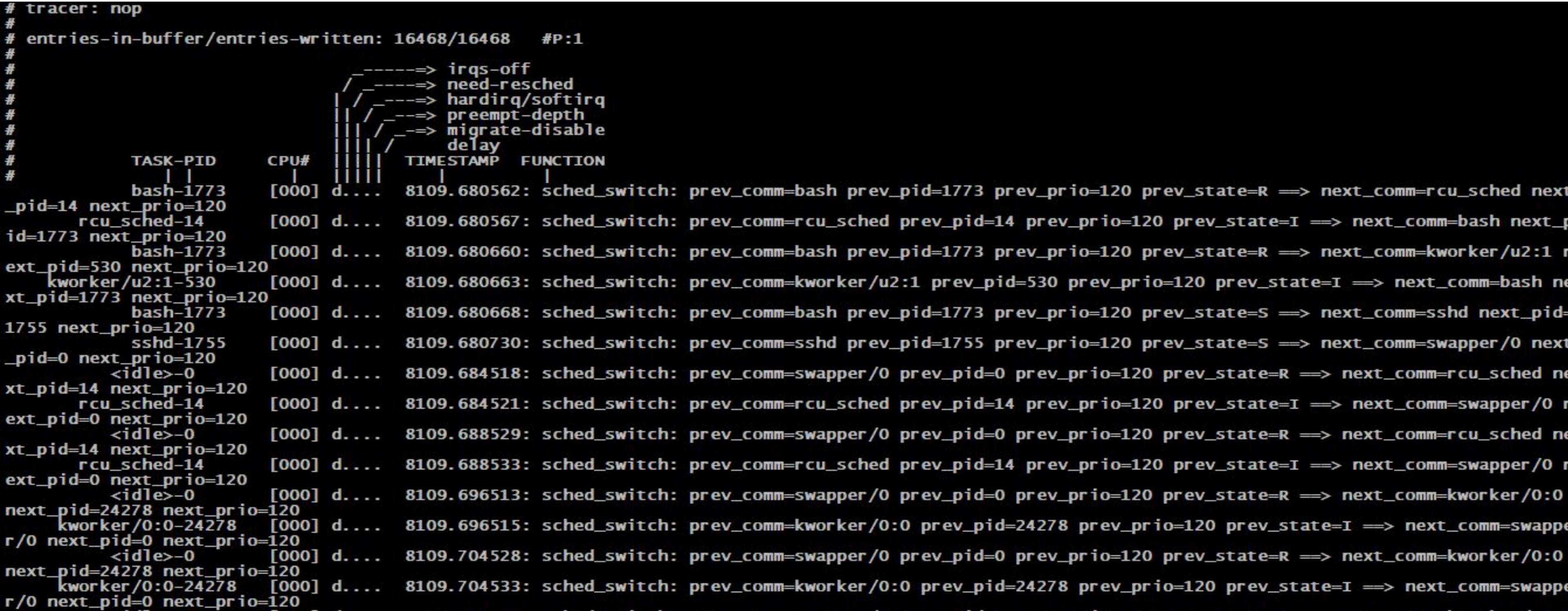
```
root@linux:/sys/kernel/tracing# tree events | head -n 100
events
├── alarmtimer
│   ├── alarmtimer_cancel
│   │   ├── enable
│   │   ├── filter
│   │   ├── format
│   │   ├── hist
│   │   ├── id
│   │   ├── inject
│   │   └── trigger
│   ├── alarmtimer_fired
│   │   ├── enable
│   │   ├── filter
│   │   ├── format
│   │   ├── hist
│   │   ├── id
│   │   ├── inject
│   │   └── trigger
│   ├── alarmtimer_start
│   │   ├── enable
│   │   ├── filter
│   │   ├── format
│   │   ├── hist
│   │   ├── id
│   │   ├── inject
│   │   └── trigger
│   ├── alarmtimer_suspend
│   │   ├── enable
│   │   ├── filter
│   │   ├── format
```

Event Tracing

$ **cd /sys/kernel/tracing**

$ **echo sched:sched_switch > set_event**

$ **cat trace**



```
# tracer: nop
#
# entries-in-buffer/entries-written: 16468/16468   #P:1
#
#                                _------=> irqs-off
#                               / _-----=> need-resched
#                              | / _----=> hardirq/softirq
#                              || / _---=> preempt-depth
#                              ||| / _--=> migrate-disable
#                              |||| /     delay
#           TASK-PID     CPU#  |||||  TIMESTAMP  FUNCTION
#              | |         |   |||||     |         |
            bash-1773    [000] d....  8109.680562: sched_switch: prev_comm=bash prev_pid=1773 prev_prio=120 prev_state=R ==> next_comm=rcu_sched next
_pid=14 next_prio=120
       rcu_sched-14      [000] d....  8109.680567: sched_switch: prev_comm=rcu_sched prev_pid=14 prev_prio=120 prev_state=I ==> next_comm=bash next_p
id=1773 next_prio=120
            bash-1773    [000] d....  8109.680660: sched_switch: prev_comm=bash prev_pid=1773 prev_prio=120 prev_state=R ==> next_comm=kworker/u2:1 n
ext_pid=530 next_prio=120
    kworker/u2:1-530     [000] d....  8109.680663: sched_switch: prev_comm=kworker/u2:1 prev_pid=530 prev_prio=120 prev_state=I ==> next_comm=bash ne
xt_pid=1773 next_prio=120
            bash-1773    [000] d....  8109.680668: sched_switch: prev_comm=bash prev_pid=1773 prev_prio=120 prev_state=S ==> next_comm=sshd next_pid=
1755 next_prio=120
            sshd-1755    [000] d....  8109.680730: sched_switch: prev_comm=sshd prev_pid=1755 prev_prio=120 prev_state=S ==> next_comm=swapper/0 next
_pid=0 next_prio=120
          <idle>-0       [000] d....  8109.684518: sched_switch: prev_comm=swapper/0 prev_pid=0 prev_prio=120 prev_state=R ==> next_comm=rcu_sched ne
xt_pid=14 next_prio=120
       rcu_sched-14      [000] d....  8109.684521: sched_switch: prev_comm=rcu_sched prev_pid=14 prev_prio=120 prev_state=I ==> next_comm=swapper/0 n
ext_pid=0 next_prio=120
          <idle>-0       [000] d....  8109.688529: sched_switch: prev_comm=swapper/0 prev_pid=0 prev_prio=120 prev_state=R ==> next_comm=rcu_sched ne
xt_pid=14 next_prio=120
       rcu_sched-14      [000] d....  8109.688533: sched_switch: prev_comm=rcu_sched prev_pid=14 prev_prio=120 prev_state=I ==> next_comm=swapper/0 n
ext_pid=0 next_prio=120
          <idle>-0       [000] d....  8109.696513: sched_switch: prev_comm=swapper/0 prev_pid=0 prev_prio=120 prev_state=R ==> next_comm=kworker/0:0
next_pid=24278 next_prio=120
     kworker/0:0-24278   [000] d....  8109.696515: sched_switch: prev_comm=kworker/0:0 prev_pid=24278 prev_prio=120 prev_state=I ==> next_comm=swappe
r/0 next_pid=0 next_prio=120
          <idle>-0       [000] d....  8109.704528: sched_switch: prev_comm=swapper/0 prev_pid=0 prev_prio=120 prev_state=R ==> next_comm=kworker/0:0
next_pid=24278 next_prio=120
     kworker/0:0-24278   [000] d....  8109.704533: sched_switch: prev_comm=kworker/0:0 prev_pid=24278 prev_prio=120 prev_state=I ==> next_comm=swappe
r/0 next_pid=0 next_prio=120
```

Trace-cmd

Trace-cmd is a command line tool that provides a layer of abstraction over ftrace. This tool is more user-friendly and easy to use.

Tracing the Functions called by a Process using Trace-cmd

To record the trace in trace.data file

```
$ trace-cmd record -p function_graph -F <command you want to trace>
```

To read the data from trace.data file

```
$ trace-cmd report
```

To filter functions

```
$ trace-cmd record -g <function> -F <command you want to trace>
```

To filter events

```
$ trace-cmd record -e <event> -F <command you want to trace>
```

Ftrace Uses

- For collecting data from the system.
- To understand how kernel works and how it handles various events.
- For debugging the kernel.
- For detecting the sources of latency.
- Very useful for tracing embedded systems in busybox environment.

Perf

Perf is an analysis tool based on Linux 2.6+ systems, which abstracts the CPU hardware difference in performance measurement in Linux, and provides a simple command line interface. Perf is based on the **perf_events** interface exported by the kernel. It is very useful for profiling the system and finding performance bottlenecks in an application.

Perf Installation

To install the latest version of perf you have to clone the latest Linux mainline repository and then build the perf tool.

- **apt-get install flex bison yacc**
- **apt-get install libelf-dev systemtap-sdt-dev libaudit-dev libslang2-dev libperl-dev libdw-dev**
- **git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git linux**
- **cd linux**
- **make -j3 all**
- **cd tools/perf**
- **make**

Note: It is not necessary to compile the kernel to install the latest version of perf. But if the version of perf matches with the kernel then it gives more accurate information on the subsystem usage.

perf stat

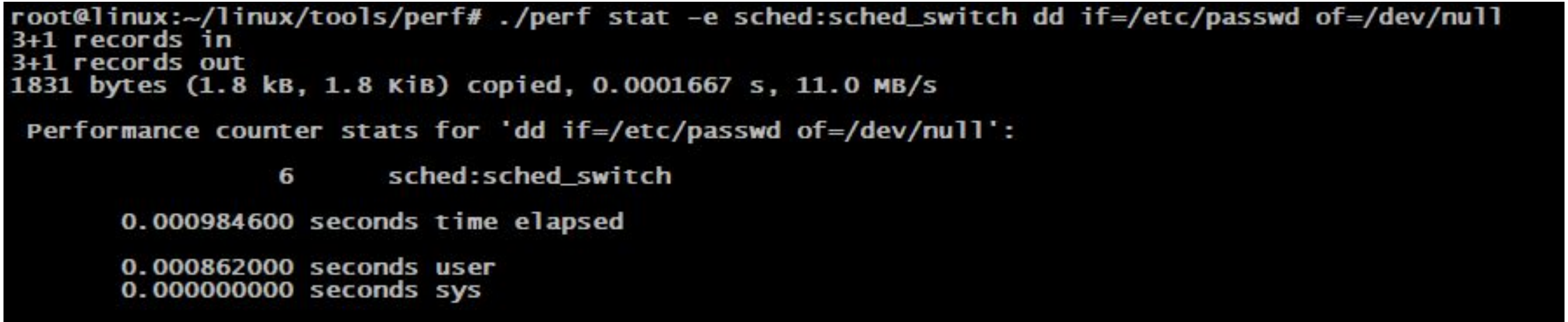
The perf stat command can generate a report of various hardware (eg cache-misses), software (eg page faults) and tracepoint (eg sched:sched_stat_runtime, syscalls:sys_enter_socket) events.

Usage:

$ **perf stat** <command you want to analyze>

$ **perf list** # To list all the events available for a system

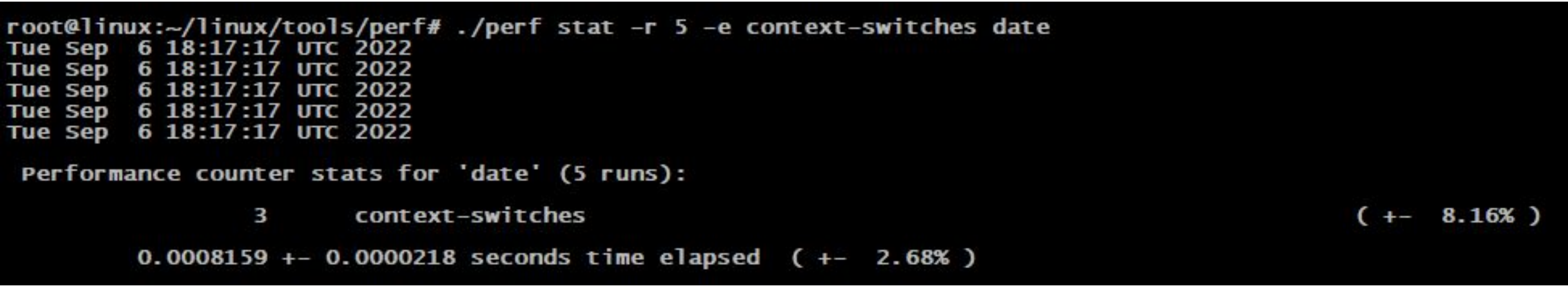
$ **perf stat -e <event you want to measure> <command you want to analyze>**



```
root@linux:~/linux/tools/perf# ./perf stat -e sched:sched_switch dd if=/etc/passwd of=/dev/null
3+1 records in
3+1 records out
1831 bytes (1.8 kB, 1.8 KiB) copied, 0.0001667 s, 11.0 MB/s

 Performance counter stats for 'dd if=/etc/passwd of=/dev/null':

                 6      sched:sched_switch

       0.000984600 seconds time elapsed

       0.000862000 seconds user
       0.000000000 seconds sys
```

Repeated Measurement

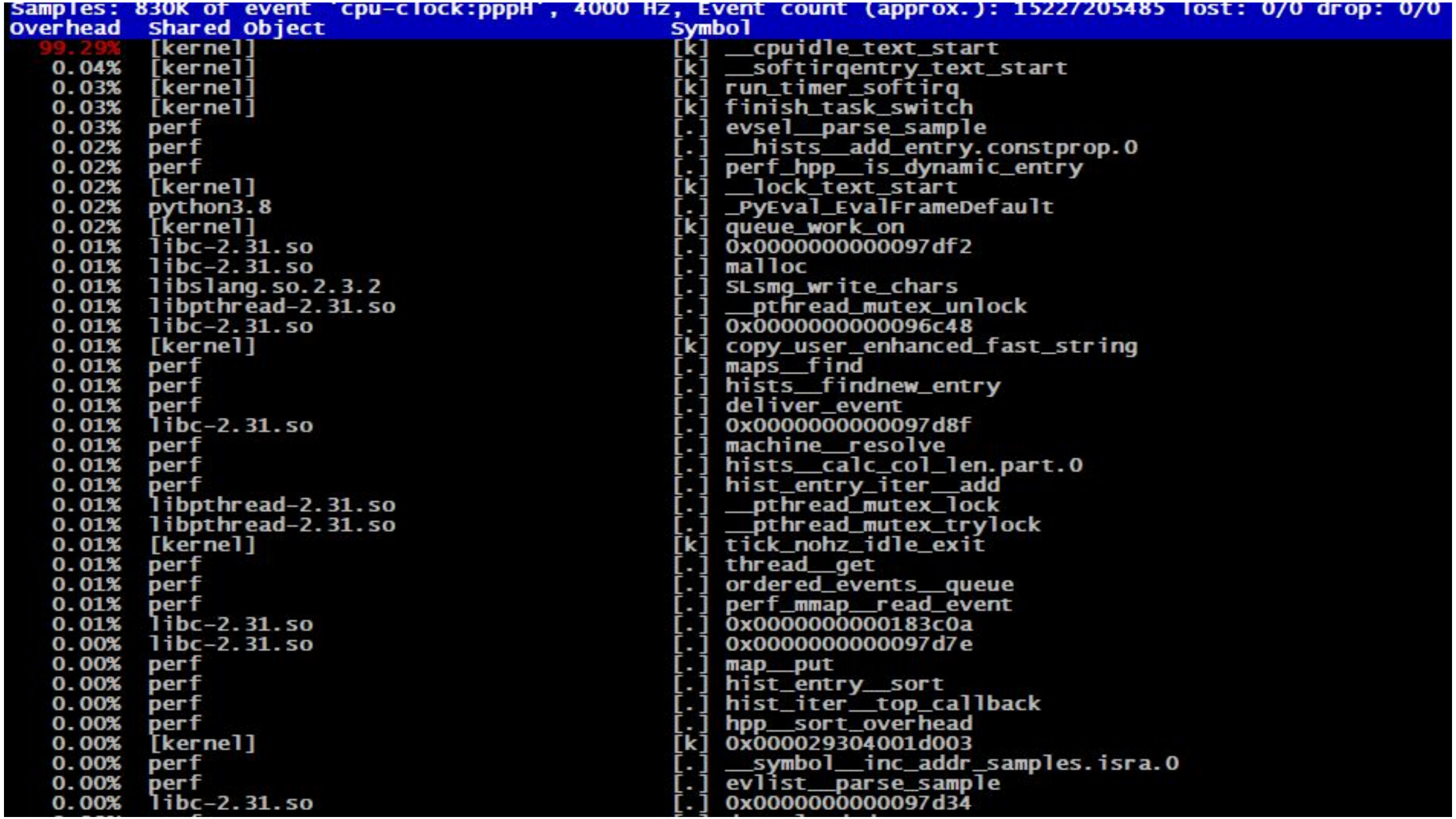
We can use -r option to test the same workload multiple times and get for each count, the standard deviation from the mean.



```
root@linux:~/linux/tools/perf# ./perf stat -r 5 -e context-switches date
Tue Sep  6 18:17:17 UTC 2022
Tue Sep  6 18:17:17 UTC 2022
Tue Sep  6 18:17:17 UTC 2022
Tue Sep  6 18:17:17 UTC 2022
Tue Sep  6 18:17:17 UTC 2022

 Performance counter stats for 'date' (5 runs):

                 3      context-switches                                                ( +-  8.16% )

         0.0008159 +- 0.0000218 seconds time elapsed  ( +-  2.68% )
```

perf top

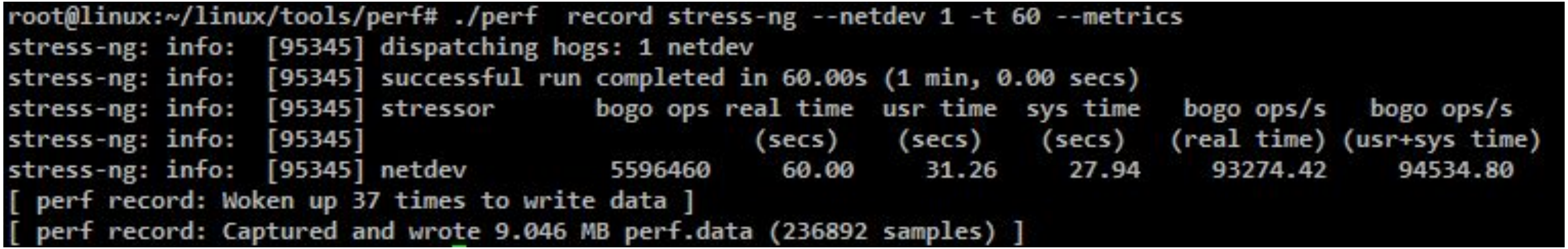
For system wide live profiling, shows you how much CPU time each specific function uses.



```
Samples: 830K of event 'cpu-clock:pppH', 4000 Hz, Event count (approx.): 15227205485 lost: 0/0 drop: 0/0
Overhead  Shared Object                        Symbol
  99.29%  [kernel]                             [k] __cpuidle_text_start
   0.04%  [kernel]                             [k] __softirqentry_text_start
   0.03%  [kernel]                             [k] run_timer_softirq
   0.03%  [kernel]                             [k] finish_task_switch
   0.03%  perf                                 [.] evsel__parse_sample
   0.02%  perf                                 [.] __hists__add_entry.constprop.0
   0.02%  perf                                 [.] perf_hpp__is_dynamic_entry
   0.02%  [kernel]                             [k] __lock_text_start
   0.02%  python3.8                            [.] _PyEval_EvalFrameDefault
   0.02%  [kernel]                             [k] queue_work_on
   0.01%  libc-2.31.so                         [.] 0x0000000000097df2
   0.01%  libc-2.31.so                         [.] malloc
   0.01%  libslang.so.2.3.2                    [.] SLsmg_write_chars
   0.01%  libpthread-2.31.so                   [.] __pthread_mutex_unlock
   0.01%  libc-2.31.so                         [.] 0x0000000000096c48
   0.01%  [kernel]                             [k] copy_user_enhanced_fast_string
   0.01%  perf                                 [.] maps__find
   0.01%  perf                                 [.] hists__findnew_entry
   0.01%  perf                                 [.] deliver_event
   0.01%  libc-2.31.so                         [.] 0x0000000000097d8f
   0.01%  perf                                 [.] machine__resolve
   0.01%  perf                                 [.] hists__calc_col_len.part.0
   0.01%  perf                                 [.] hist_entry_iter__add
   0.01%  libpthread-2.31.so                   [.] __pthread_mutex_lock
   0.01%  libpthread-2.31.so                   [.] __pthread_mutex_trylock
   0.01%  [kernel]                             [k] tick_nohz_idle_exit
   0.01%  perf                                 [.] thread__get
   0.01%  perf                                 [.] ordered_events__queue
   0.01%  perf                                 [.] perf_mmap__read_event
   0.01%  libc-2.31.so                         [.] 0x0000000000183c0a
   0.00%  libc-2.31.so                         [.] 0x0000000000097d7e
   0.00%  perf                                 [.] map__put
   0.00%  perf                                 [.] hist_entry__sort
   0.00%  perf                                 [.] hist_iter__top_callback
   0.00%  perf                                 [.] hpp__sort_overhead
   0.00%  [kernel]                             [k] 0x000029304001d003
   0.00%  perf                                 [.] __symbol__inc_addr_samples.isra.0
   0.00%  perf                                 [.] evlist__parse_sample
   0.00%  libc-2.31.so                         [.] 0x0000000000097d34
```

perf record

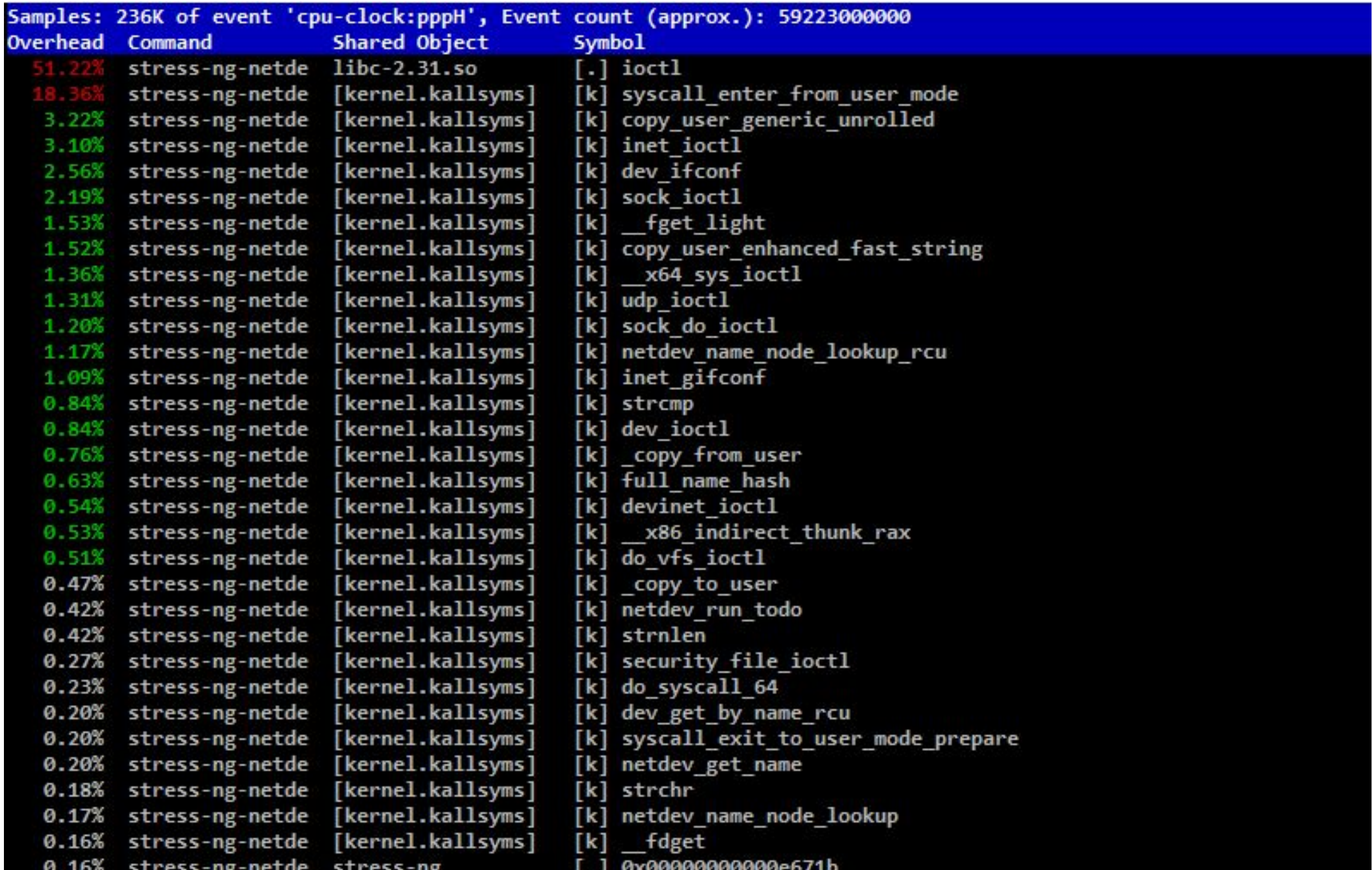
The perf record command records the events and information associated with a process. This command stores the profiling data in the perf.data file in the same directory.



```
root@linux:~/linux/tools/perf# ./perf  record stress-ng --netdev 1 -t 60 --metrics
stress-ng: info:  [95345] dispatching hogs: 1 netdev
stress-ng: info:  [95345] successful run completed in 60.00s (1 min, 0.00 secs)
stress-ng: info:  [95345] stressor       bogo ops real time  usr time  sys time   bogo ops/s   bogo ops/s
stress-ng: info:  [95345]                           (secs)    (secs)    (secs)   (real time) (usr+sys time)
stress-ng: info:  [95345] netdev          5596460     60.00     31.26     27.94     93274.42     94534.80
[ perf record: Woken up 37 times to write data ]
[ perf record: Captured and wrote 9.046 MB perf.data (236892 samples) ]
```

perf report

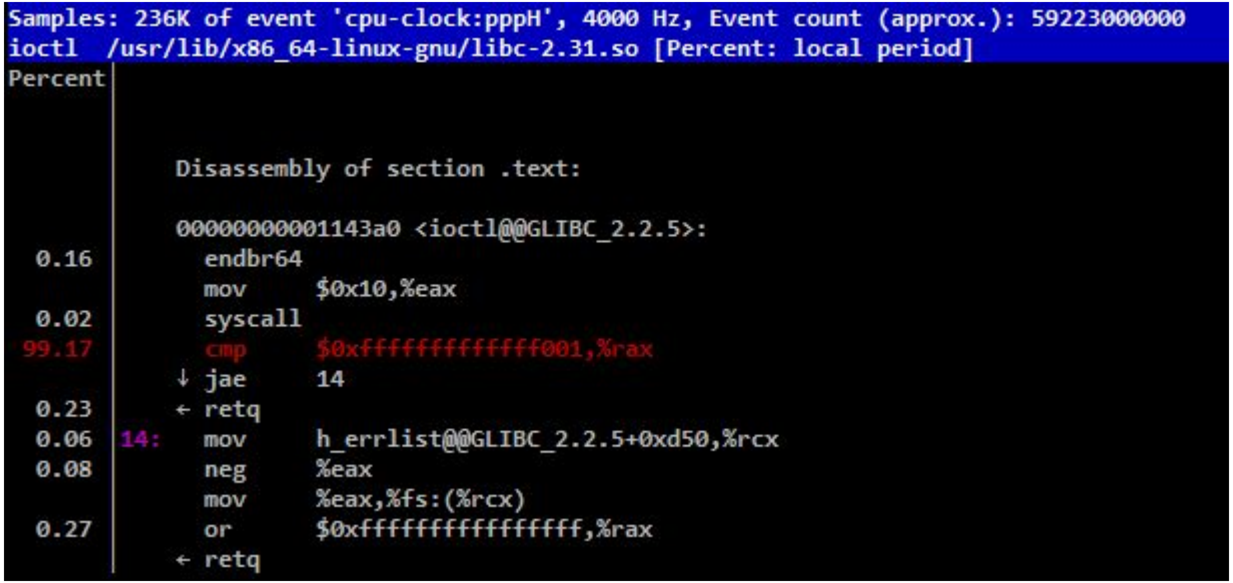
The perf report command helps us to read the perf.data file and view the final report



```
Samples: 236K of event 'cpu-clock:pppH', Event count (approx.): 59223000000
Overhead  Command          Shared Object       Symbol
  51.22%  stress-ng-netde  libc-2.31.so        [.] ioctl
  18.36%  stress-ng-netde  [kernel.kallsyms]   [k] syscall_enter_from_user_mode
   3.22%  stress-ng-netde  [kernel.kallsyms]   [k] copy_user_generic_unrolled
   3.10%  stress-ng-netde  [kernel.kallsyms]   [k] inet_ioctl
   2.56%  stress-ng-netde  [kernel.kallsyms]   [k] dev_ifconf
   2.19%  stress-ng-netde  [kernel.kallsyms]   [k] sock_ioctl
   1.53%  stress-ng-netde  [kernel.kallsyms]   [k] __fget_light
   1.52%  stress-ng-netde  [kernel.kallsyms]   [k] copy_user_enhanced_fast_string
   1.36%  stress-ng-netde  [kernel.kallsyms]   [k] __x64_sys_ioctl
   1.31%  stress-ng-netde  [kernel.kallsyms]   [k] udp_ioctl
   1.20%  stress-ng-netde  [kernel.kallsyms]   [k] sock_do_ioctl
   1.17%  stress-ng-netde  [kernel.kallsyms]   [k] netdev_name_node_lookup_rcu
   1.09%  stress-ng-netde  [kernel.kallsyms]   [k] inet_gifconf
   0.84%  stress-ng-netde  [kernel.kallsyms]   [k] strcmp
   0.84%  stress-ng-netde  [kernel.kallsyms]   [k] dev_ioctl
   0.76%  stress-ng-netde  [kernel.kallsyms]   [k] _copy_from_user
   0.63%  stress-ng-netde  [kernel.kallsyms]   [k] full_name_hash
   0.54%  stress-ng-netde  [kernel.kallsyms]   [k] devinet_ioctl
   0.53%  stress-ng-netde  [kernel.kallsyms]   [k] __x86_indirect_thunk_rax
   0.51%  stress-ng-netde  [kernel.kallsyms]   [k] do_vfs_ioctl
   0.47%  stress-ng-netde  [kernel.kallsyms]   [k] _copy_to_user
   0.42%  stress-ng-netde  [kernel.kallsyms]   [k] netdev_run_todo
   0.42%  stress-ng-netde  [kernel.kallsyms]   [k] strnlen
   0.27%  stress-ng-netde  [kernel.kallsyms]   [k] security_file_ioctl
   0.23%  stress-ng-netde  [kernel.kallsyms]   [k] do_syscall_64
   0.20%  stress-ng-netde  [kernel.kallsyms]   [k] dev_get_by_name_rcu
   0.20%  stress-ng-netde  [kernel.kallsyms]   [k] syscall_exit_to_user_mode_prepare
   0.20%  stress-ng-netde  [kernel.kallsyms]   [k] netdev_get_name
   0.18%  stress-ng-netde  [kernel.kallsyms]   [k] strchr
   0.17%  stress-ng-netde  [kernel.kallsyms]   [k] netdev_name_node_lookup
   0.16%  stress-ng-netde  [kernel.kallsyms]   [k] __fdget
   0.16%  stress-ng-netde  stress-ng           [.] 0x00000000000e671b
```

perf annotate

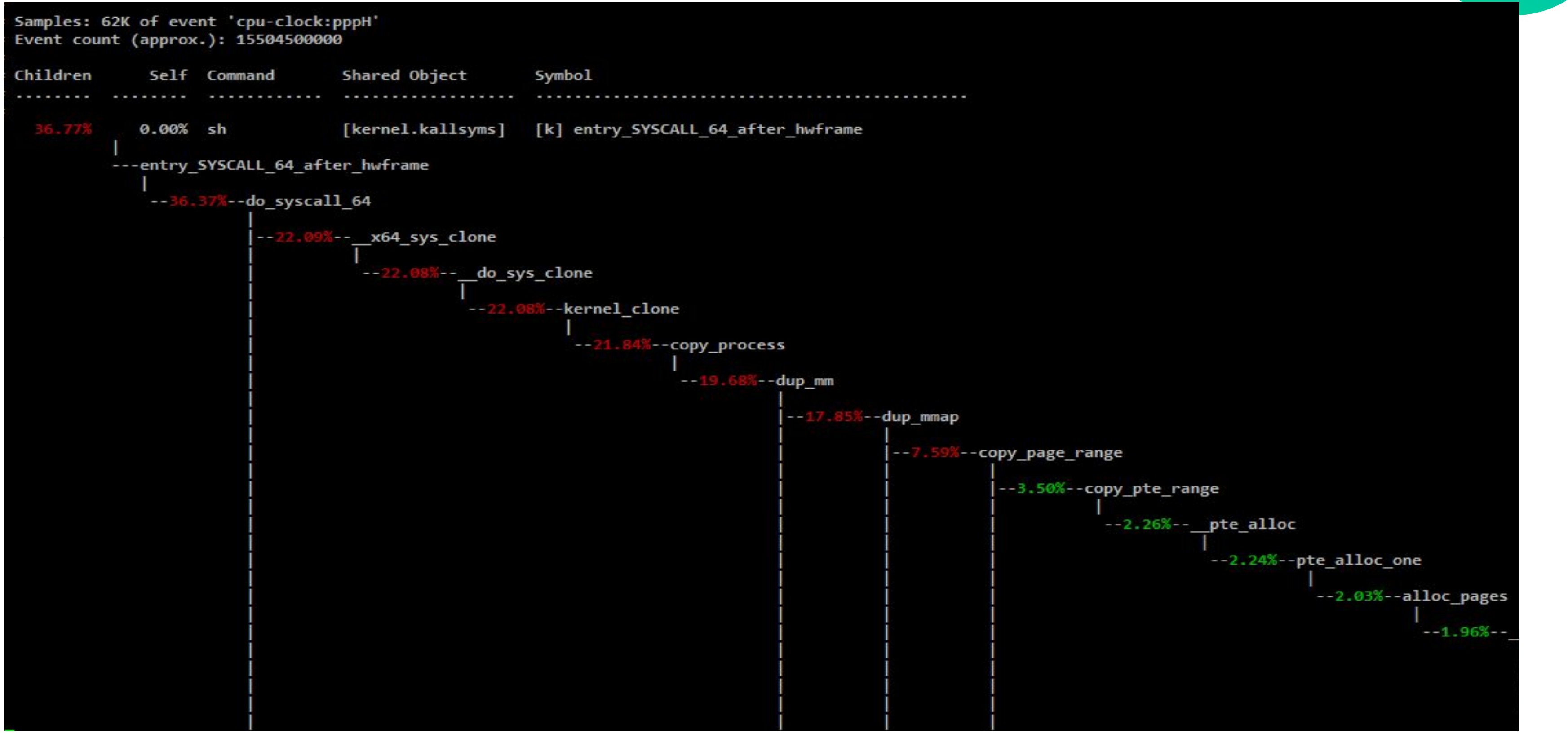
Reads perf.data and displays annotated code.



```
Samples: 236K of event 'cpu-clock:pppH', 4000 Hz, Event count (approx.): 59223000000
ioctl  /usr/lib/x86_64-linux-gnu/libc-2.31.so [Percent: local period]
Percent│
       │
       │
       │
       │     Disassembly of section .text:
       │
       │     00000000001143a0 <ioctl@@GLIBC_2.2.5>:
  0.16 │       endbr64
       │       mov     $0x10,%eax
  0.02 │       syscall
 99.17 │       cmp     $0xfffffffffffff001,%rax
       │     ↓ jae     14
  0.23 │     ← retq
  0.06 │14:    mov     h_errlist@@GLIBC_2.2.5+0xd50,%rcx
  0.08 │       neg     %eax
       │       mov     %eax,%fs:(%rcx)
  0.27 │       or      $0xffffffffffffffff,%rax
       │     ← retq
```

Perf Call Graph

The call graph format allows us to collect CPU stack traces to see which functions are calling which functions in the performance profile - **./perf record --call-graph dwarf <command>**

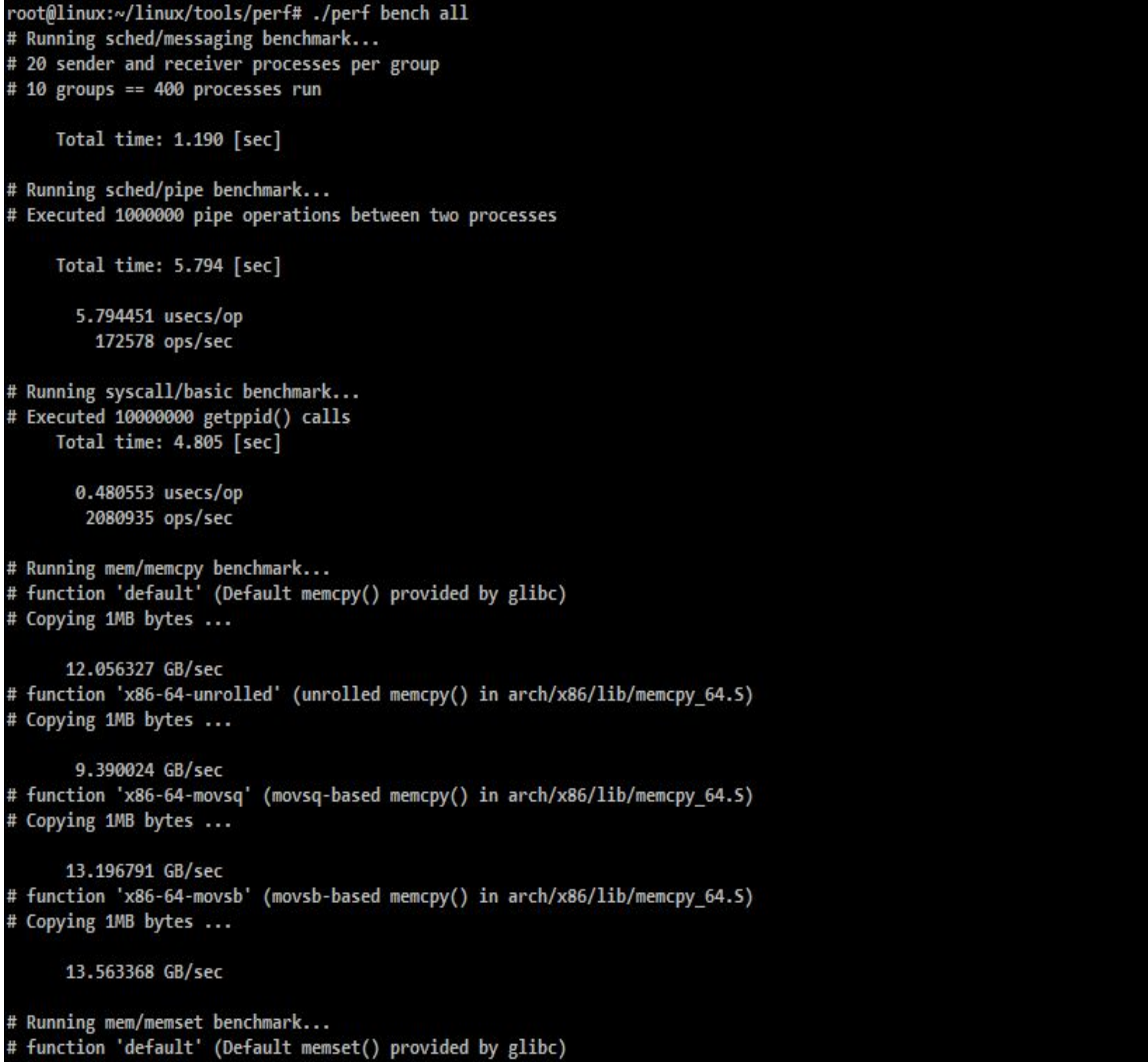


```
Samples: 62K of event 'cpu-clock:pppH'
Event count (approx.): 15504500000

Children      Self  Command       Shared Object       Symbol
--------  --------  ------------  ------------------  ---------------------------------------------

  36.77%     0.00%  sh            [kernel.kallsyms]   [k] entry_SYSCALL_64_after_hwframe
          |
          ---entry_SYSCALL_64_after_hwframe
             |
              --36.37%--do_syscall_64
                        |
                        |--22.09%--__x64_sys_clone
                        |          |
                        |           --22.08%--__do_sys_clone
                        |                     |
                        |                      --22.08%--kernel_clone
                        |                                |
                        |                                 --21.84%--copy_process
                        |                                           |
                        |                                            --19.68%--dup_mm
                        |                                                      |
                        |                                                      |--17.85%--dup_mmap
                        |                                                      |          |
                        |                                                      |          |--7.59%--copy_page_range
                        |                                                      |          |          |
                        |                                                      |          |          |--3.50%--copy_pte_range
                        |                                                      |          |          |          |
                        |                                                      |          |          |           --2.26%--__pte_alloc
                        |                                                      |          |          |                     |
                        |                                                      |          |          |                      --2.24%--pte_alloc_one
                        |                                                      |          |          |                                |
                        |                                                      |          |          |                                 --2.03%--alloc_pages
                        |                                                      |          |          |                                           |
                        |                                                      |          |          |                                            --1.96%--_
                        |                                                      |          |          |
                        |                                                      |          |          |
                        |                                                      |          |          |
                        |                                                      |          |          |
```

perf bench (all) workload

The perf bench command contains multiple multithreaded microkernel benchmarks for executing different subsystems in the Linux kernel and system calls. This allows us to easily measure the impact of changes, which can help mitigate performance regressions. It also acts as a common benchmarking framework, enabling developers to easily create test cases, integrate transparently, and use performance-rich tooling subsystems.

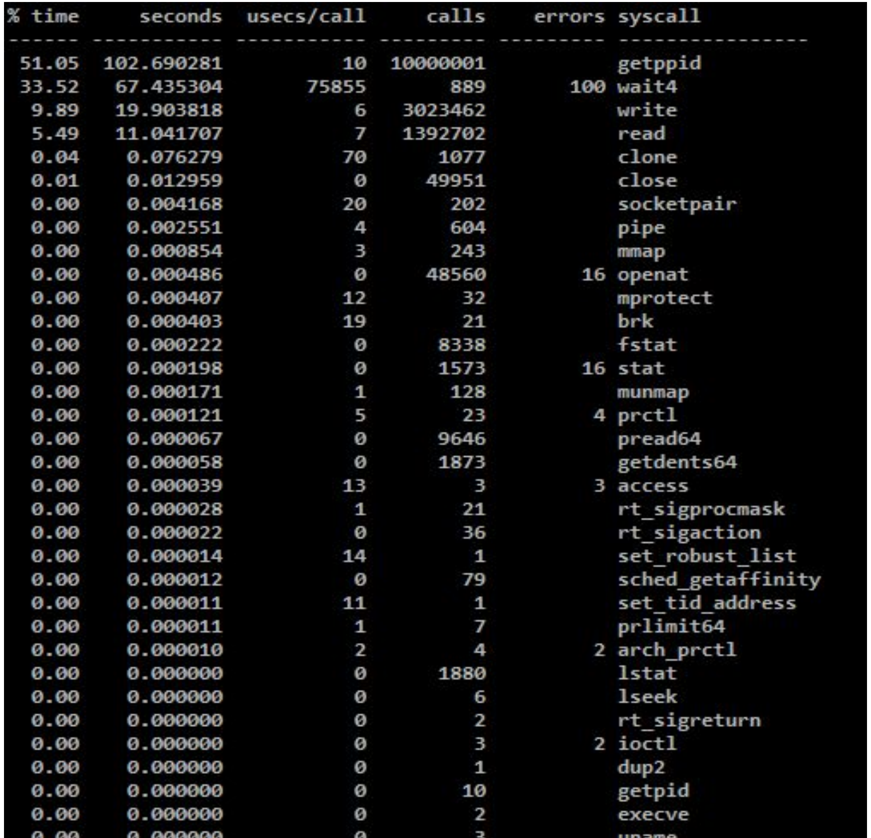
$ ./perf bench all



```
root@linux:~/linux/tools/perf# ./perf bench all
# Running sched/messaging benchmark...
# 20 sender and receiver processes per group
# 10 groups == 400 processes run

     Total time: 1.190 [sec]

# Running sched/pipe benchmark...
# Executed 1000000 pipe operations between two processes

     Total time: 5.794 [sec]

       5.794451 usecs/op
         172578 ops/sec

# Running syscall/basic benchmark...
# Executed 10000000 getppid() calls
     Total time: 4.805 [sec]

       0.480553 usecs/op
        2080935 ops/sec

# Running mem/memcpy benchmark...
# function 'default' (Default memcpy() provided by glibc)
# Copying 1MB bytes ...

      12.056327 GB/sec
# function 'x86-64-unrolled' (unrolled memcpy() in arch/x86/lib/memcpy_64.S)
# Copying 1MB bytes ...

       9.390024 GB/sec
# function 'x86-64-movsq' (movsq-based memcpy() in arch/x86/lib/memcpy_64.S)
# Copying 1MB bytes ...

      13.196791 GB/sec
# function 'x86-64-movsb' (movsb-based memcpy() in arch/x86/lib/memcpy_64.S)
# Copying 1MB bytes ...

      13.563368 GB/sec

# Running mem/memset benchmark...
# function 'default' (Default memset() provided by glibc)
```

Tracing perf bench all Workload under strace

Gathering system call statistics under the perf bench (all) workload - **strace -c ./perf bench all**



```
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 51.05  102.690281          10  10000001           getppid
 33.52   67.435304       75855       889       100 wait4
  9.89   19.903818           6   3023462           write
  5.49   11.041707           7   1392702           read
  0.04    0.076279          70      1077           clone
  0.01    0.012959           0     49951           close
  0.00    0.004168          20       202           socketpair
  0.00    0.002551           4       604           pipe
  0.00    0.000854           3       243           mmap
  0.00    0.000486           0     48560        16 openat
  0.00    0.000407          12        32           mprotect
  0.00    0.000403          19        21           brk
  0.00    0.000222           0      8338           fstat
  0.00    0.000198           0      1573        16 stat
  0.00    0.000171           1       128           munmap
  0.00    0.000121           5        23         4 prctl
  0.00    0.000067           0      9646           pread64
  0.00    0.000058           0      1873           getdents64
  0.00    0.000039          13         3         3 access
  0.00    0.000028           1        21           rt_sigprocmask
  0.00    0.000022           0        36           rt_sigaction
  0.00    0.000014          14         1           set_robust_list
  0.00    0.000012           0        79           sched_getaffinity
  0.00    0.000011          11         1           set_tid_address
  0.00    0.000011           1         7           prlimit64
  0.00    0.000010           2         4         2 arch_prctl
  0.00    0.000000           0      1880           lstat
  0.00    0.000000           0         6           lseek
  0.00    0.000000           0         2           rt_sigreturn
  0.00    0.000000           0         3         2 ioctl
  0.00    0.000000           0         1           dup2
  0.00    0.000000           0        10           getpid
  0.00    0.000000           0         2           execve
  0.00    0.000000           0         3           uname
```

Subsystem Usage by the Workload

System Call	Frequency	Linux Subsystem	System Call Entry Point (API)
getppid	10000001	Process Mgmt	sys_getpid()
clone	1077	Process Mgmt.	sys_clone()
prctl	23	Process Mgmt.	sys_prctl()
prlimit64	7	Process Mgmt.	sys_prlimit64()
getpid	10	Process Mgmt.	sys_getpid()
uname	3	Process Mgmt.	sys_uname()
sysinfo	1	Process Mgmt.	sys_sysinfo()
getuid	1	Process Mgmt.	sys_getuid()
getgid	1	Process Mgmt.	sys_getgid()
geteuid	1	Process Mgmt.	sys_geteuid()
getegid	1	Process Mgmt.	sys_getegid
getpgrp	1	Process Mgmt.	sys_getpgrp()

write	3023462	Filesystem	sys_write()
read	1392702	Filesystem	sys_read()
close	49951	Filesystem	sys_close()
pipe	604	Filesystem	sys_pipe()
openat	48560	Filesystem	sys_opennat()
fstat	8338	Filesystem	sys_fstat()
stat	1573	Filesystem	sys_stat()
pread64	9646	Filesystem	sys_pread64()
getdents64	1873	Filesystem	sys_getdents64()
access	3	Filesystem	sys_access()
lstat	1880	Filesystem	sys_lstat()
lseek	6	Filesystem	sys_lseek()
ioctl	3	Filesystem	sys_ioctl()

dup2	1	Filesystem	sys_dup2()
execve	2	Filesystem	sys_execve()
fcntl	8779	Filesystem	sys_fcntl()
statfs	1	Filesystem	sys_statfs()
epoll_create	2	Filesystem	sys_epoll_create()
epoll_ctl	64	Filesystem	sys_epoll_ctl()
newfstatat	8318	Filesystem	sys_newfstatat()
eventfd2	192	Filesystem	sys_eventfd2()
mmap	243	Memory Mgmt.	sys_mmap()
mprotect	32	Memory Mgmt.	sys_mprotect()
brk	21	Memory Mgmt.	sys_brk()
munmap	128	Memory Mgmt.	sys_munmap()
set_mempolicy	156	Memory Mgmt.	sys_set_mempolicy()

set_tid_address	1	Process Mgmt.	sys_set_tid_address()
set_robust_list	1	Futex	sys_set_robust_list()
futex	341	Futex	sys_futex()
sched_getaffinity	79	Scheduler	sys_sched_getaffinity()
sched_setaffinity	223	Scheduler	sys_sched_setaffinity()
socketpair	202	Network	sys_socketpair()
rt_sigprocmask	21	Signal	sys_rt_sigprocmask()
rt_sigaction	36	Signal	sys_rt_sigaction()
rt_sigreturn	2	Signal	sys_rt_sigreturn()
wait4	889	Time	sys_wait4()
clock_nanosleep	37	Time	sys_clock_nanosleep()
capget	4	Capability	sys_capget()

Conclusion

- Understanding system resources necessary to build and run a workload is important.
- Linux tracing and strace can be used to discover the system resources in use by a workload. The completeness of the system usage information depends on the completeness of coverage of a workload.
- Performance and security of the operating system can be analyzed with the help of tools like ftrace, perf, stress-ng, paxtest.
- Once we discover and understand the workload needs, we can focus on them to avoid regressions and use it to evaluate safety considerations.

My Mentorship Experience

- My journey in this Mentorship Program was a life changing experience, it motivated me to delve deep into the world of Linux kernel and interact with luminaries of this field.
- It has groomed me professionally by providing me expert guidance through my scholarly mentors.
- I got to learn very powerful tools and techniques for tracing and analyzing the kernel.
- I learned about STPA Analysis for analyzing the safety of applications.
- I wrote a White Paper on my findings which is available at: https://github.com/elisa-tech/ELISA-White-Papers/blob/master/Processes/Discovering_Linux_kernel_subsystems_used_by_a_workload.md

ELISA SUMMIT

SEPTEMBER 7 - 8, 2022 | VIRTUAL

@PROJECTELISA

Questions?

ELISA

SUMMIT

SEPTEMBER 7 - 8, 2022 | VIRTUAL

@PROJECTELISA

Thank you

Licensing of Summit Results

All work created during the summit is licensed under *Creative Commons Attribution 4.0 International (CC-BY-4.0)* [https://creativecommons.org/licenses/by/4.0/] by default, or under another suitable open-source license, e.g., GPL-2.0 for kernel code contributions.

You are free to:

- **Share** — copy and redistribute the material in any medium or format
- **Adapt** — remix, transform, and build upon the material for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.