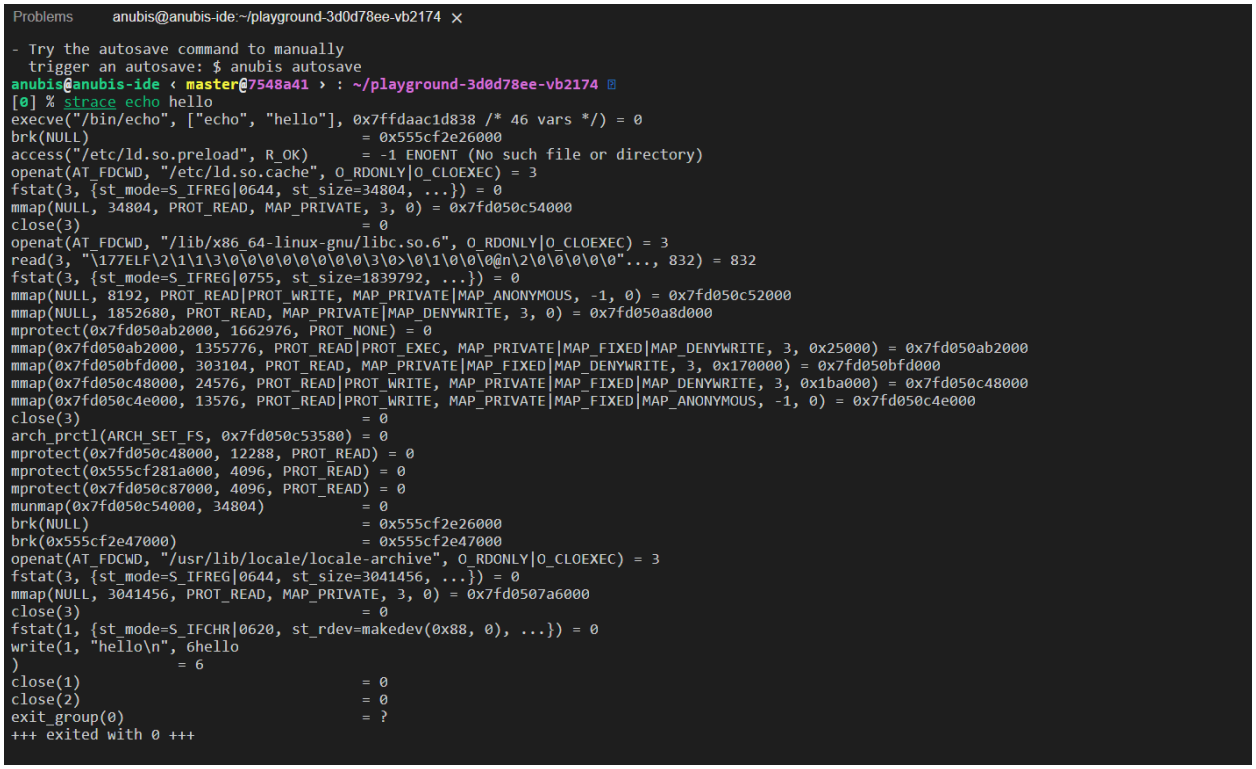
Major Project - OS

Team:
Swarna Swapna Ilamathy - si2150 - Anubis submission id : 1dbe14c2411724d6c759
Vikram Badhan - vb2174

TASKS:-

1. Get Familiar with Linux strace

i) Executing the “strace echo hello” command shows us the list of system calls. The same can be seen in the screenshot below:-



```
Problems        anubis@anubis-ide:~/playground-3d0d78ee-vb2174 ×

- Try the autosave command to manually
  trigger an autosave: $ anubis autosave
anubis@anubis-ide ‹ master@7548a41 › : ~/playground-3d0d78ee-vb2174
[0] % strace echo hello
execve("/bin/echo", ["echo", "hello"], 0x7ffdaac1d838 /* 46 vars */) = 0
brk(NULL)                               = 0x555cf2e26000
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=34804, ...}) = 0
mmap(NULL, 34804, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fd050c54000
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@n\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1839792, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fd050c52000
mmap(NULL, 1852680, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fd050a8d000
mprotect(0x7fd050ab2000, 1662976, PROT_NONE) = 0
mmap(0x7fd050ab2000, 1355776, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x25000) = 0x7fd050ab2000
mmap(0x7fd050bfd000, 303104, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x170000) = 0x7fd050bfd000
mmap(0x7fd050c48000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1ba000) = 0x7fd050c48000
mmap(0x7fd050c4e000, 13576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fd050c4e000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7fd050c53580) = 0
mprotect(0x7fd050c48000, 12288, PROT_READ) = 0
mprotect(0x555cf281a000, 4096, PROT_READ) = 0
mprotect(0x7fd050c87000, 4096, PROT_READ) = 0
munmap(0x7fd050c54000, 34804)           = 0
brk(NULL)                               = 0x555cf2e26000
brk(0x555cf2e47000)                     = 0x555cf2e47000
openat(AT_FDCWD, "/usr/lib/locale/locale-archive", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=3041456, ...}) = 0
mmap(NULL, 3041456, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fd0507a6000
close(3)                                = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
write(1, "hello\n", 6hello
)                 = 6
close(1)                                = 0
close(2)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

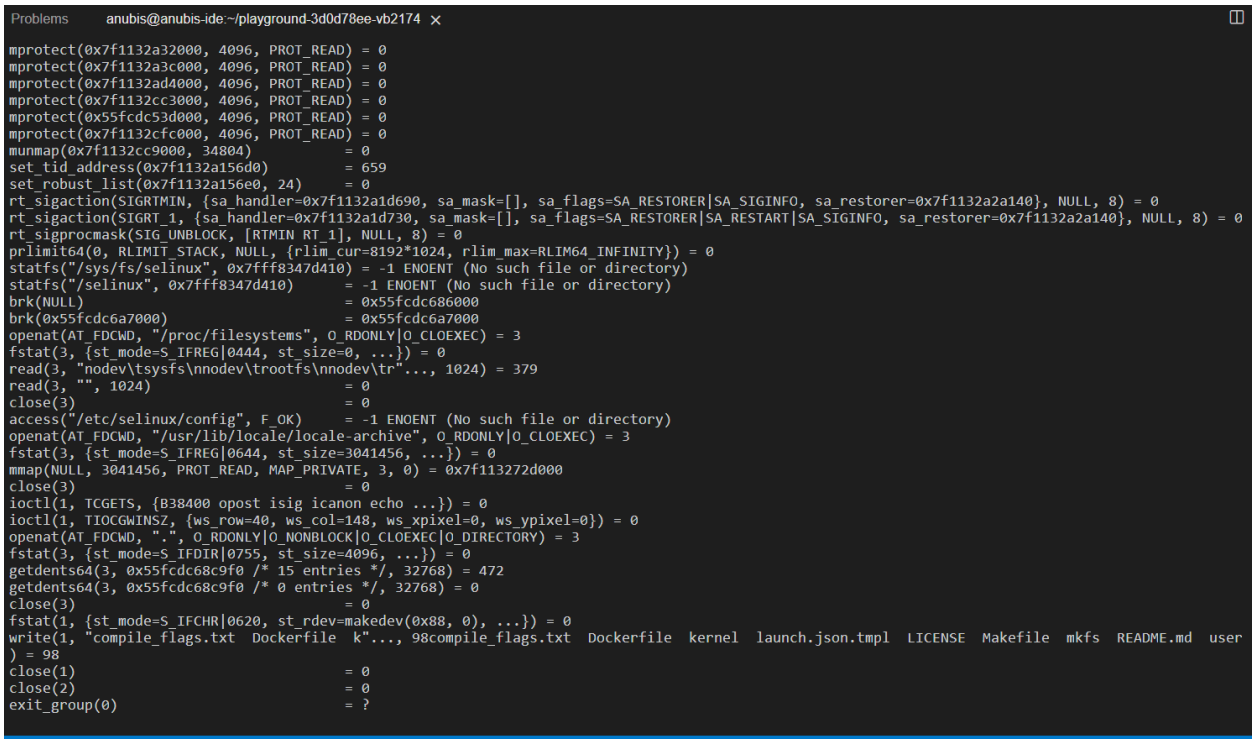
(ii) Executing the “strace ls” command shows us the following list of system calls:-

```
Problems        anubis@anubis-ide:~/playground-3d0d78ee-vb2174 ×

anubis@anubis-ide ‹ master@7548a41 › : ~/playground-3d0d78ee-vb2174
[0] % strace ls
execve("/bin/ls", ["ls"], 0x7ffc2d8226a0 /* 46 vars */) = 0
brk(NULL)                               = 0x55fcdc686000
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=34804, ...}) = 0
mmap(NULL, 34804, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f1132cc9000
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\3\0>\0\1\0\0\0\220}\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=166120, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f1132cc7000
mmap(NULL, 177672, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f1132c9b000
mmap(0x7f1132ca2000, 102400, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x7000) = 0x7f1132ca2000
mmap(0x7f1132cbb000, 32768, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x20000) = 0x7f1132cbb000
mmap(0x7f1132cc3000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x27000) = 0x7f1132cc3000
mmap(0x7f1132cc5000, 5640, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f1132cc5000
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@n\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1839792, ...}) = 0
mmap(NULL, 1852680, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f1132ad6000
mprotect(0x7f1132afb000, 1662976, PROT_NONE) = 0
mmap(0x7f1132afb000, 1355776, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x25000) = 0x7f1132afb000
mmap(0x7f1132c46000, 303104, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x170000) = 0x7f1132c46000
mmap(0x7f1132c91000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1ba000) = 0x7f1132c91000
mmap(0x7f1132c97000, 13576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f1132c97000
close(3)                                = 0
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/libpcre2-8.so.0", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\200\"\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=617128, ...}) = 0
mmap(NULL, 619432, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f1132a3e000
mmap(0x7f1132a40000, 438272, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7f1132a40000
mmap(0x7f1132aab000, 167936, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x6d000) = 0x7f1132aab000
mmap(0x7f1132ad4000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x95000) = 0x7f1132ad4000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libdl.so.2", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0000\21\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=18688, ...}) = 0
```

```
Problems        anubis@anubis-ide:~/playground-3d0d78ee-vb2174 ×

mmap(NULL, 20752, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f1132a38000
mmap(0x7f1132a39000, 8192, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1000) = 0x7f1132a39000
mmap(0x7f1132a3b000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x3000) = 0x7f1132a3b000
mmap(0x7f1132a3c000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x3000) = 0x7f1132a3c000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libpthread.so.0", O_RDONLY|O_CLOEXEC) = 3
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
fstat(3, {st_mode=S_IFREG|0755, st_size=149520, ...}) = 0
mmap(NULL, 136304, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f1132a16000
mmap(0x7f1132a1d000, 65536, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x7000) = 0x7f1132a1d000
mmap(0x7f1132a2d000, 20480, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x17000) = 0x7f1132a2d000
mmap(0x7f1132a32000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b000) = 0x7f1132a32000
mmap(0x7f1132a34000, 13424, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f1132a34000
close(3)                                = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f1132a14000
arch_prctl(ARCH_SET_FS, 0x7f1132a15400) = 0
mprotect(0x7f1132c91000, 12288, PROT_READ) = 0
mprotect(0x7f1132a32000, 4096, PROT_READ) = 0
mprotect(0x7f1132a3c000, 4096, PROT_READ) = 0
mprotect(0x7f1132ad4000, 4096, PROT_READ) = 0
mprotect(0x7f1132cc3000, 4096, PROT_READ) = 0
mprotect(0x55fcdc53d000, 4096, PROT_READ) = 0
mprotect(0x7f1132cfc000, 4096, PROT_READ) = 0
munmap(0x7f1132cc9000, 34804)           = 0
set_tid_address(0x7f1132a156d0)         = 659
set_robust_list(0x7f1132a156e0, 24)     = 0
rt_sigaction(SIGRTMIN, {sa_handler=0x7f1132a1d690, sa_mask=[], sa_flags=SA_RESTORER|SA_SIGINFO, sa_restorer=0x7f1132a2a140}, NULL, 8) = 0
rt_sigaction(SIGRT_1, {sa_handler=0x7f1132a1d730, sa_mask=[], sa_flags=SA_RESTORER|SA_RESTART|SA_SIGINFO, sa_restorer=0x7f1132a2a140}, NULL, 8) = 0
rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
statfs("/sys/fs/selinux", 0x7fff8347d410) = -1 ENOENT (No such file or directory)
statfs("/selinux", 0x7fff8347d410)      = -1 ENOENT (No such file or directory)
brk(NULL)                               = 0x55fcdc686000
brk(0x55fcdc6a7000)                     = 0x55fcdc6a7000
openat(AT_FDCWD, "/proc/filesystems", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0
read(3, "nodev\tsysfs\nnodev\trootfs\nnodev\tr"..., 1024) = 379
read(3, "", 1024)                       = 0
close(3)                                = 0
access("/etc/selinux/config", F_OK)     = -1 ENOENT (No such file or directory)
```



The above screenshots show us the list of system calls when we execute the “strace ls” command.
A system call is a programmatic means for a program to ask the kernel for a service, and strace is a useful tool for tracing the thin layer between user programs and the Linux kernel.

Given below are four amongst the many system calls that we got after running the strace command:-

a) “Nmap” system call:-
“nmap” stands for “Network Map” which is an open source tool which was basically designed for security auditing and network exploration. It was created to quickly scan big networks, but also works well with single hosts. Nmap analyzes raw IP packets in unique ways to figure out which host is currently using the network, which Operating system they are running, the number firewalls being used,etc.

b) “read’ system call:-
“read” system call is a type of call which is used to read from a file descriptor. It reads the input typed by the person on the keyboard and stores it in an array(buffer). It can only read 10 bytes at max, after that no character will be read.

c) “fstat” system call:-
“fstat” system call is a type of call which fetches us the status of a file which comprises all the useful information related to the file. The information can be about a file being a directory or a regular file, it may also tell us user Id of the file and it may also tell us other information like size, access, modification count of the file.

d) “openat” system call:-
“Openat” system call is a type of call which works the same way as “open” system call except for a few differences. In Open system call a pathname is specified and based on that the file is opened whereas an Openat system call allows you to specify a relative pathname that is interpreted relative to another directory represented by a file descriptor.

2. Building strace in xv6

i) Command : strace on

"strace on" command will set the trace flag to be on and while executing, since the trace flag is set, the syscall function will print the trace information of each process when each system call gets called by the process. The system call information is printed in the following format : pid, process name, system call name and return value of the system call.

```
$ strace on
$ echo hello
TRACE: pid = 3 | process name = sh | syscall = exec
hTRACE: pid = 3 | process name = echo | syscall = write | return val = 1
eTRACE: pid = 3 | process name = echo | syscall = write | return val = 1
lTRACE: pid = 3 | process name = echo | syscall = write | return val = 1
lTRACE: pid = 3 | process name = echo | syscall = write | return val = 1
oTRACE: pid = 3 | process name = echo | syscall = write | return val = 1

TRACE: pid = 3 | process name = echo | syscall = write | return val = 1
TRACE: pid = 3 | process name = echo | syscall = exit
```

ii) Command : strace off

"strace off" command will unset the trace flag and only if the flag is set, the syscall function in syscall.c will write the trace information. Unsetting the flag will switch off the trace.

```
$ strace on
$ echo hello
TRACE: pid = 3 | process name = sh | syscall = exec
hTRACE: pid = 3 | process name = echo | syscall = write | return val = 1
eTRACE: pid = 3 | process name = echo | syscall = write | return val = 1
lTRACE: pid = 3 | process name = echo | syscall = write | return val = 1
lTRACE: pid = 3 | process name = echo | syscall = write | return val = 1
oTRACE: pid = 3 | process name = echo | syscall = write | return val = 1

TRACE: pid = 3 | process name = echo | syscall = write | return val = 1
TRACE: pid = 3 | process name = echo | syscall = exit
$ strace off
$ echo hello
hello
```

iii) Command strace run

“strace run” command will provide trace information only for the command given along with the strace run command. In this example, strace run echo hi, strace information will be shown only for echo hi

```
$ strace run echo hi
TRACE: pid = 2 | process name = sh | syscall = exec
hTRACE: pid = 2 | process name = echo | syscall = write | return val = 1
iTRACE: pid = 2 | process name = echo | syscall = write | return val = 1

TRACE: pid = 2 | process name = echo | syscall = write | return val = 1
TRACE: pid = 2 | process name = echo | syscall = exit
```

The next command won’t have any strace information as shown below.

```
$ echo hi
hi
$
```

v) Trace child process

To show how the strace information will be when we fork the child process, I have forked two child process and shown how a child process is forked from a parent process and the system calls involved in that.

```
$ strace on
$ test_trace_child
TRACE: pid = 9 | process name = sh | syscall = exec
TRACE: pid = 9 | process name = test_trace_chil | syscall = fork | return val = 10
Executing Child 1
TRACE: pid = 9 | process name = test_trace_chil | syscall = fork | return val = 11
PTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
Executing Child 2
aTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
rTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
eTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
nTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
tTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
 TRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
WTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
aTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
iTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
tTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
iTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
nTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
gTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1

TRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
TRACE: pid = 9 | process name = test_trace_chil | syscall = wait | return val = 10
TRACE: pid = 9 | process name = test_trace_chil | syscall = wait | return val = 11
CTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
hTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
iTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
```

```
dTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
rTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
eTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
nTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
 TRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
cTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
oTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
mTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
pTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
lTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
eTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
tTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
eTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
dTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1

TRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
PTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
aTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
rTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
eTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
nTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
tTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
 TRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
ETRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
xTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
eTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
cTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
uTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
```

```
tTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
iTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
nTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
gTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1

TRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
PTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
aTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
rTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
eTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
nTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
tTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
 TRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
eTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
xTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
iTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
tTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
iTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
nTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
gTRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
.TRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1

TRACE: pid = 9 | process name = test_trace_chil | syscall = write | return val = 1
TRACE: pid = 9 | process name = test_trace_chil | syscall = exit
```

---

3. Building option for strace

To implement the options for trace, we have implemented a system call to set the flag in proc.c for proc->option for the specified option. Based on the flag information from proc->option

i) Option: -e <system call name>

For option e, we have set the flag proc->option as 1 and also set filter name in the proc->flag in the kernel space. So, we can access the flag name and option set and process based on that.
For this, only the system call mentioned in the command will be shown in the trace information.

```
$ strace -e write
$ echo hello
TRACE: pid = 12 | process name = sh | syscall = exec
hTRACE: pid = 12 | process name = echo | syscall = write | return val = 1
eTRACE: pid = 12 | process name = echo | syscall = write | return val = 1
lTRACE: pid = 12 | process name = echo | syscall = write | return val = 1
lTRACE: pid = 12 | process name = echo | syscall = write | return val = 1
oTRACE: pid = 12 | process name = echo | syscall = write | return val = 1

TRACE: pid = 12 | process name = echo | syscall = write | return val = 1
$ echo hello
TRACE: pid = 13 | process name = sh | syscall = exec
hTRACE: pid = 13 | process name = echo | syscall = write | return val = 1
eTRACE: pid = 13 | process name = echo | syscall = write | return val = 1
lTRACE: pid = 13 | process name = echo | syscall = write | return val = 1
lTRACE: pid = 13 | process name = echo | syscall = write | return val = 1
oTRACE: pid = 13 | process name = echo | syscall = write | return val = 1

TRACE: pid = 13 | process name = echo | syscall = write | return val = 1
TRACE: pid = 13 | process name = echo | syscall = exit
```

ii) Option: -s

Set the proc->option for 2 and based on that only the successful system call information will be printed.

```
$ strace -s
$ echo hi
TRACE: pid = 15 | process name = sh | syscall = exec
hTRACE: pid = 15 | process name = echo | syscall = write | return val = 1
iTRACE: pid = 15 | process name = echo | syscall = write | return val = 1

TRACE: pid = 15 | process name = echo | syscall = write | return val = 1
```

iii) Option: -f

Set the proc->option for 2 and based on that only the failed system call information will be printed.

```
$ strace -f
$ cat temp
TRACE: pid = 22 | process name = cat | syscall = open | return val = -1
```

iv) Option runs only once

From the below screenshot you can see that the strace option command runs only once. The next time you execute a command, all the system calls are shown, not just the failed system calls.

```
$ strace -t
$ cat temp
TRACE: pid = 3 | process name = cat | syscall = open | return val = -1
cat: cannot open temp
$ cat temp
TRACE: pid = 4 | process name = sh | syscall = exec
TRACE: pid = 4 | process name = cat | syscall = open | return val = -1
cTRACE: pid = 4 | process name = cat | syscall = write | return val = 1
aTRACE: pid = 4 | process name = cat | syscall = write | return val = 1
tTRACE: pid = 4 | process name = cat | syscall = write | return val = 1
:TRACE: pid = 4 | process name = cat | syscall = write | return val = 1
 TRACE: pid = 4 | process name = cat | syscall = write | return val = 1
cTRACE: pid = 4 | process name = cat | syscall = write | return val = 1
aTRACE: pid = 4 | process name = cat | syscall = write | return val = 1
nTRACE: pid = 4 | process name = cat | syscall = write | return val = 1
nTRACE: pid = 4 | process name = cat | syscall = write | return val = 1
oTRACE: pid = 4 | process name = cat | syscall = write | return val = 1
tTRACE: pid = 4 | process name = cat | syscall = write | return val = 1
 TRACE: pid = 4 | process name = cat | syscall = write | return val = 1
oTRACE: pid = 4 | process name = cat | syscall = write | return val = 1
pTRACE: pid = 4 | process name = cat | syscall = write | return val = 1
eTRACE: pid = 4 | process name = cat | syscall = write | return val = 1
nTRACE: pid = 4 | process name = cat | syscall = write | return val = 1
 TRACE: pid = 4 | process name = cat | syscall = write | return val = 1
tTRACE: pid = 4 | process name = cat | syscall = write | return val = 1
eTRACE: pid = 4 | process name = cat | syscall = write | return val = 1
```

---

5. Applications of Strace

i) executed a memory leak command to see trace information on xv6

```
$ trace_unexpected
TRACE: pid = 3 | process name = sh | syscall = exec
mTRACE: pid = 3 | process name = trace_unexpecte | syscall = write | return val = 1
eTRACE: pid = 3 | process name = trace_unexpecte | syscall = write | return val = 1
mTRACE: pid = 3 | process name = trace_unexpecte | syscall = write | return val = 1
 TRACE: pid = 3 | process name = trace_unexpecte | syscall = write | return val = 1
tTRACE: pid = 3 | process name = trace_unexpecte | syscall = write | return val = 1
eTRACE: pid = 3 | process name = trace_unexpecte | syscall = write | return val = 1
sTRACE: pid = 3 | process name = trace_unexpecte | syscall = write | return val = 1
tTRACE: pid = 3 | process name = trace_unexpecte | syscall = write | return val = 1

TRACE: pid = 3 | process name = trace_unexpecte | syscall = write | return val = 1
TRACE: pid = 3 | process name = trace_unexpecte | syscall = getpid | return val = 3
TRACE: pid = 3 | process name = trace_unexpecte | syscall = fork | return val = 4

allocuvm out of memory
allocuvm out of memory
couldn't allocate mem?!!
TRACE: pid = 3 | process name = trace_unexpecte | syscall = wait | return val = 4
```

ii) memory leak on linux

```
anubis@anubis-ide ‹ master@7133487 ●● › : ~/final-project-371525eb-si2150
[0] % strace ./test
execve("./test", ["./test"], 0x7ffc983d5140 /* 46 vars */) = 0
brk(NULL)                               = 0x55a83259c000
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=34804, ...}) = 0
mmap(NULL, 34804, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f9a6a23a000
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@n\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1839792, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f9a6a238000
mmap(NULL, 1852680, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f9a6a073000
mprotect(0x7f9a6a098000, 1662976, PROT_NONE) = 0
mmap(0x7f9a6a098000, 1355776, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x25000) = 0x7f9a6a098000
mmap(0x7f9a6a1e3000, 303104, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x170000) = 0x7f9a6a1e3000
mmap(0x7f9a6a22e000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1ba000) = 0x7f9a6a22e000
mmap(0x7f9a6a234000, 13576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f9a6a234000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7f9a6a239540) = 0
mprotect(0x7f9a6a22e000, 12288, PROT_READ) = 0
mprotect(0x55a830d10000, 4096, PROT_READ) = 0
mprotect(0x7f9a6a26d000, 4096, PROT_READ) = 0
munmap(0x7f9a6a23a000, 34804)           = 0
--- SIGSEGV {si_signo=SIGSEGV, si_code=SEGV_MAPERR, si_addr=0x1} ---
+++ killed by SIGSEGV (core dumped) +++
[1]    47440 segmentation fault  strace ./test
```

iii) As we already know that strace is a diagnostic tool which is used to debug a program when we do not have access to the source code of that program. In operating systems like Linux, strace function is already inbuilt for the user to take advantage of as opposed to XV6 where there is no strace function present. When we run strace on Linux Operating system, we are able to view information about the various system calls running on the OS in much greater detail.

For example:- Using the “strace -c pwd” command we can get a summary of the system calls. It shows us the various system calls on the OS ordered by the time spent on each one of them, the number of times each call was made and the percentage of time each used in running the command.

And so, we notice that the Linux strace provides us with additional information about the various system calls in our Operating System and so it is quite better at debugging the problem.