PES University

Cloud Computing and Big Data

PROBLEM STATEMENT :

WHAT IS MY KERNEL DOING ?

TEAM DETAILS

PESU CCBD

PRIYA MOHATA
SRN : PES2UG19CS301
PESU EC CAMPUS

PRIYANSH JAIN
SRN : PES2UG19CS303
PESU EC CAMPUS

ARPIT KOGTA
SRN : PES2UG19CS065
PESU EC CAMPUS

SNEHIL JAIN
SRN : PES2UG19CS396
PESU EC CAMPUS

TABLE OF CONTENTS

PESU CCBD

EXECUTING AND TRACKING PERFORMANCE VIA STRACE FOR FIREFOX

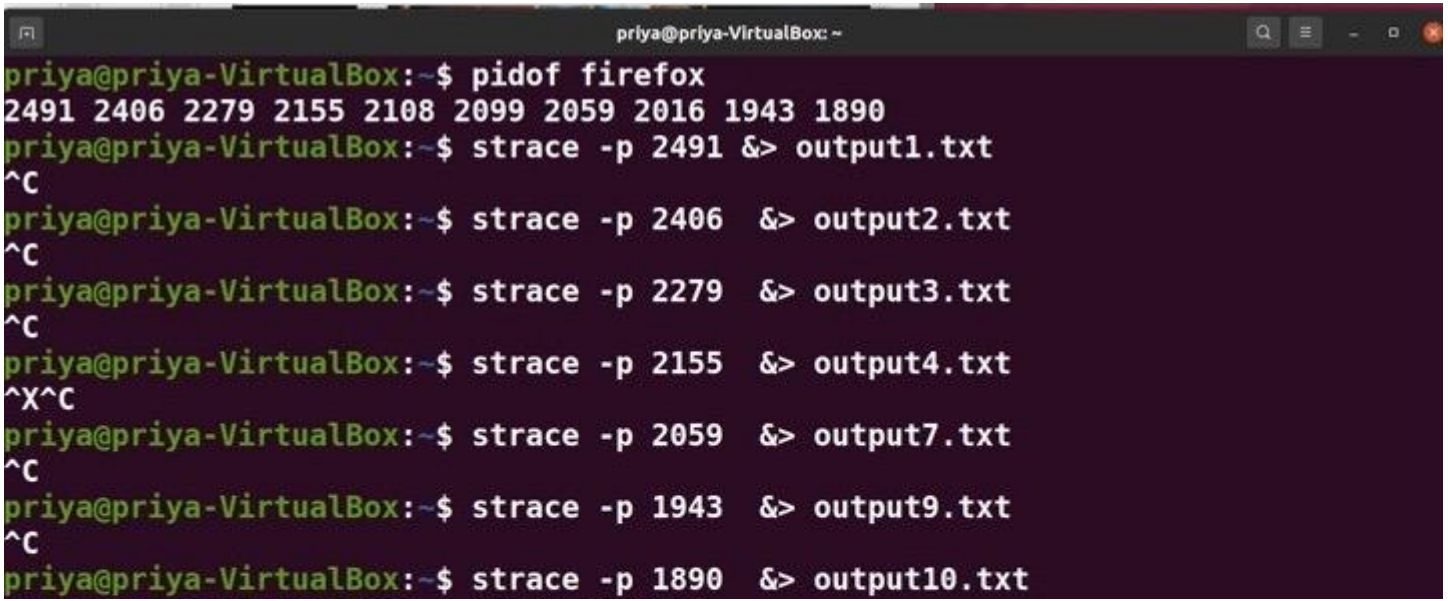
PESU CCBD

- **ABOUT STRACE** : Strace is a diagnostic, debugging and instructional user space utility for Linux. It is used to monitor and tamper with interactions between processes and the Linux kernel, which include system calls etc.
- We have used strace to track the system calls generated by the various tabs in Firefox.
- The websites which were running on the browser were : Youtube , Amazon , Gmail, Debian.org and Stack Overflow.
- COMMANDS USED :
    - To identify the process ids associated with Firefox - `pidof firefox`
    - To trace the system calls - `strace -p pid &> outputfilename`

OBSERVATIONS : Firefox used to support multithreading in the past , but in the recent times it uses multiprocessing and it also has additional processes running in parallel to provide security.

SCREENSHOTS SHOWING THE COMMANDS USED:

PESU CCBD



LINK FOR THE ALL THE SCREENSHOTS : SCREENSHOTS

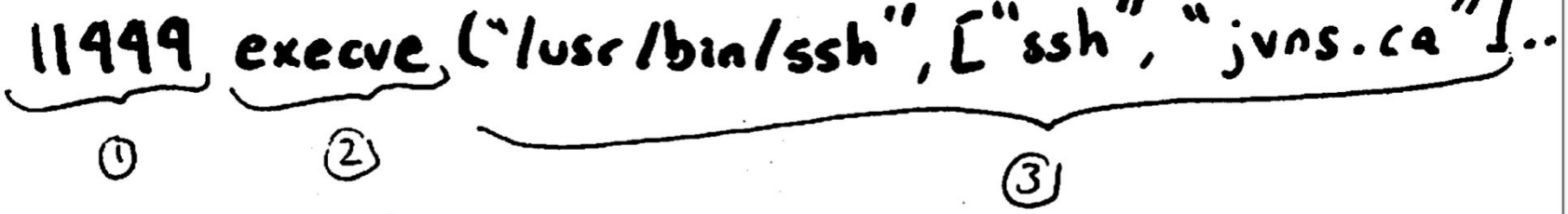
SAMPLE OUTPUT FOR STRACE -p PID

PESU CCBD

```
strace: Process 2491 attached
restart_syscall(<... resuming interrupted read ...>) = 1
read(8, "\372", 1)                      = 1
futex(0x7f4aceb592c4, FUTEX_WAKE_PRIVATE, 1) = 1
futex(0x7f4aceb59268, FUTEX_WAKE_PRIVATE, 1) = 1
write(9, "\372", 1)                     = 1
recvmsg(17, {msg_namelen=0}, 0)         = -1 EAGAIN (Resource temporarily unavailable)
poll([{fd=8, events=POLLIN}, {fd=17, events=POLLIN}, {fd=18, events=POLLIN}], 3, 0) = 1 ([{fd=8, revents=POLLIN}])
read(8, "\372", 1)                      = 1
recvmsg(17, {msg_namelen=0}, 0)         = -1 EAGAIN (Resource temporarily unavailable)
poll([{fd=8, events=POLLIN}, {fd=17, events=POLLIN}, {fd=18, events=POLLIN}], 3, 0) = 0 (Timeout)
recvmsg(17, {msg_namelen=0}, 0)         = -1 EAGAIN (Resource temporarily unavailable)
poll([{fd=8, events=POLLIN}, {fd=17, events=POLLIN}, {fd=18, events=POLLIN}], 3, 0) = 0 (Timeout)
recvmsg(17, {msg_namelen=0}, 0)         = -1 EAGAIN (Resource temporarily unavailable)
poll([{fd=8, events=POLLIN}, {fd=17, events=POLLIN}, {fd=18, events=POLLIN}], 3, 0) = 0 (Timeout)
recvmsg(17, {msg_namelen=0}, 0)         = -1 EAGAIN (Resource temporarily unavailable)
poll([{fd=8, events=POLLIN}, {fd=17, events=POLLIN}, {fd=18, events=POLLIN}], 3, -1) = 1 ([{fd=8, revents=POLLIN}])
read(8, "\372", 1)                      = 1
futex(0x7f4aceb592c0, FUTEX_WAKE_PRIVATE, 1) = 1
futex(0x7f4aceb59268, FUTEX_WAKE_PRIVATE, 1) = 1
write(9, "\372", 1)                     = 1
recvmsg(17, {msg_namelen=0}, 0)         = -1 EAGAIN (Resource temporarily unavailable)
poll([{fd=8, events=POLLIN}, {fd=17, events=POLLIN}, {fd=18, events=POLLIN}], 3, 0) = 1 ([{fd=8, revents=POLLIN}])
read(8, "\372", 1)                      = 1
recvmsg(17, {msg_namelen=0}, 0)         = -1 EAGAIN (Resource temporarily unavailable)
poll([{fd=8, events=POLLIN}, {fd=17, events=POLLIN}, {fd=18, events=POLLIN}], 3, 0) = 0 (Timeout)
recvmsg(17, {msg_namelen=0}, 0)         = -1 EAGAIN (Resource temporarily unavailable)
poll([{fd=8, events=POLLIN}, {fd=17, events=POLLIN}, {fd=18, events=POLLIN}], 3, 0) = 0 (Timeout)
recvmsg(17, {msg_namelen=0}, 0)         = -1 EAGAIN (Resource temporarily unavailable)
poll([{fd=8, events=POLLIN}, {fd=17, events=POLLIN}, {fd=18, events=POLLIN}], 3, 0) = 0 (Timeout)
recvmsg(17, {msg_namelen=0}, 0)         = -1 EAGAIN (Resource temporarily unavailable)
poll([{fd=8, events=POLLIN}, {fd=17, events=POLLIN}, {fd=18, events=POLLIN}], 3, -1) = 1 ([{fd=8, revents=POLLIN}])
```

LINK FOR THE ALL THE OUTPUT FILES :OUTPUT FILES

Output Explanation



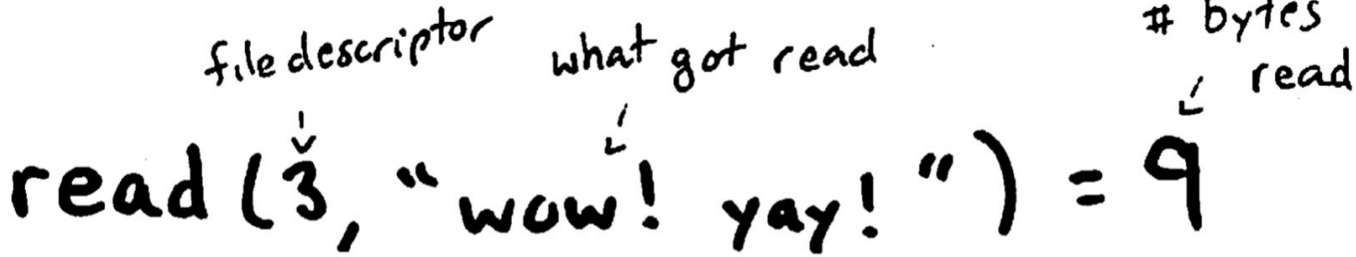
① The process ID

② The name of the system call (execve starts programs ☕)

③ The system call's arguments, in this case a program to start and the arguments to start it with.

④ (invisible, at the end) The return value.

Output Explanation



OBSERVATIONS AND GRAPHS - STRACE

PESU CCBD

- CSV files that were generated after cleaning the output of the strace command
- LINK : OUTPUT FILE FOR CLEANED DATA (CSV)
- SCREENSHOT OF THE CLEANED FILES
- OBSERVATIONS :
  - The most called system calls were - recvmsg ,poll
  - recvmsg() call is used to receive messages from a socket ,
  - poll() waits for one of a set of file descriptors to become ready to perform I/O
  - The least called system calls were - gettid , getpriority
  - gettid - returns the thread ID of the calling thread
  - getpriority - returns the highest priority (lowest numerical value) enjoyed by any of the specified processes

	A	B
1	System Call	Frequency
2	recvmsg	222346
3	poll	111168
4	futex	66801
5	read	44630
6	write	27617
7	mprotect	278
8	madvise	178
9	mmap	54
10	dup	40
11	close	38
12	getpid	34
13	memfd_create	32
14	fallocate	32
15	munmap	27
16	clone	6
17	getrandom	3
18	setpriority	2
19	getpriority	2
20	gettid	1

OBSERVATION - TIME FOR EACH SYSTEM CALL

PESU CCBD

Observations:

openat() had the largest system call time

opennat() returns a new file descriptor. On error, -1 is returned and *errno* is set to indicate the error.

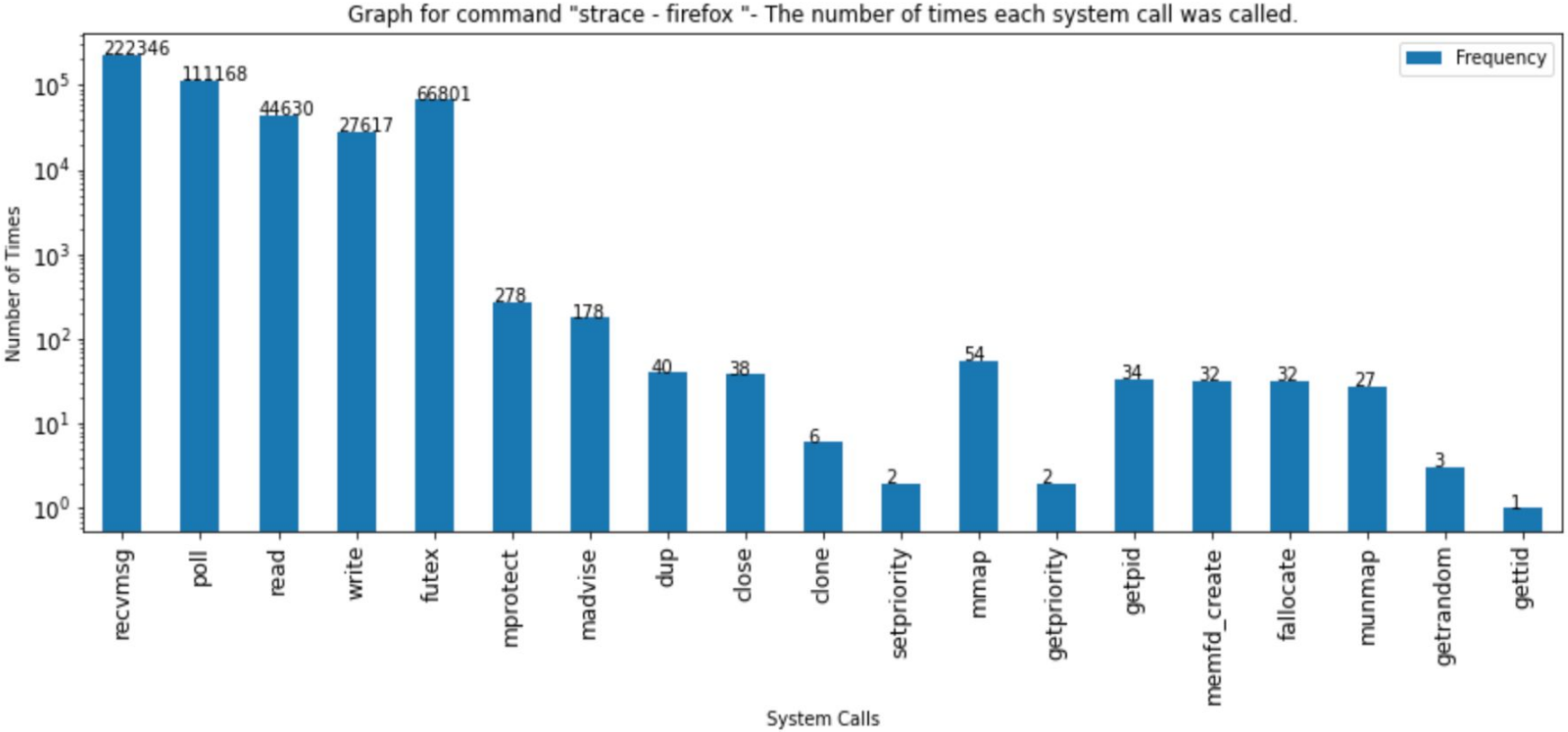
getegid() had the smallest system call time

getegid() function returns the effective group ID of the calling process.

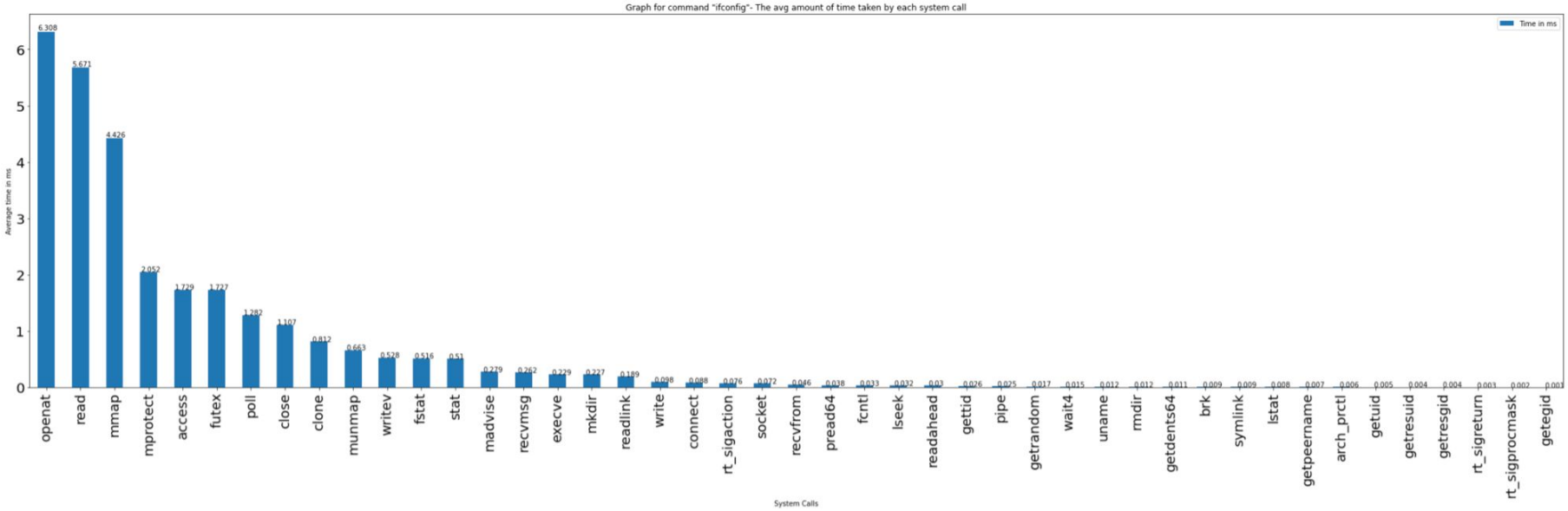
A	B	C	D	E
System-Calls	Time in ms		System-Calls	Time in ms
openat	6.308		pread64	0.038
read	5.671		fcntl	0.033
mmap	4.426		lseek	0.032
mprotect	2.052		readahead	0.03
access	1.729		gettid	0.026
futex	1.727		pipe	0.025
poll	1.282		getrandom	0.017
close	1.107		wait4	0.015
clone	0.812		uname	0.012
munmap	0.663		rmdir	0.012
writev	0.528		getdents64	0.011
fstat	0.516		brk	0.009
stat	0.51		symlink	0.009
madvise	0.279		lstat	0.008
recvmsg	0.262		getpeername	0.007
execve	0.229		arch_prctl	0.006
mkdir	0.227		getuid	0.005
readlink	0.189		getresuid	0.004
write	0.098		getresgid	0.004
connect	0.088		rt_sigreturn	0.003
rt_sigaction	0.076		rt_sigprocmask	0.002
socket	0.072		getegid	0.001
recvfrom	0.046			

GRAPH SHOWING THE VARIOUS OBSERVATIONS - STRACE

PESU CCBD



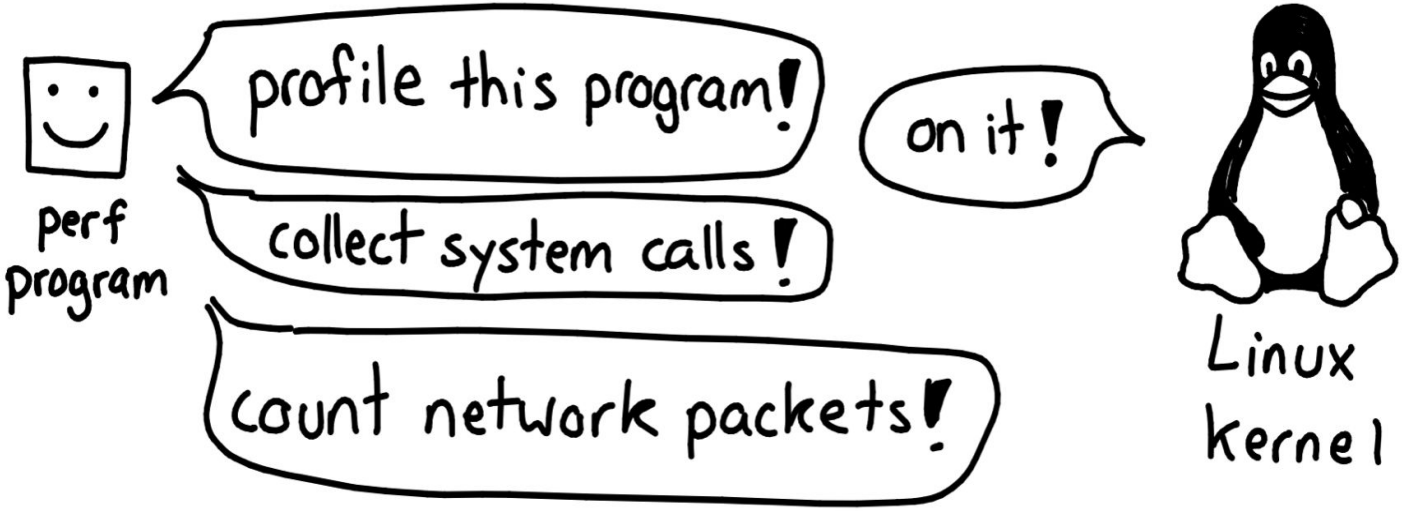
GRAPH - STRACE - FIREFOX



LINK FOR THE OUTPUT FILE : OUTPUT STRACE TIME VS NAME OF THE SYSTEM CALL

How perf works ?

- perf asks the kernel to collect information



EXECUTING AND TRACKING PERFORMANCE VIA PERF FOR FIREFOX

PESU CCBD

ABOUT PERF: It is a linux debugging tool which lets us trace system calls way faster than strace. It helps us trace almost any kernel event and can be used to profile our C, C++, Go, Ruby, JVM etc. programs.

- We have used perf to track the system calls generated by the various tabs in Firefox.
- We have opened tabs from diverse domains such as Flipkart, Google, Coursera and Govt of USA website.
- Commands Used:

  `pidof firefox`

  Two perf commands used-

  `perf trace -p pid -s`

  `perf stat 'syscalls:sys_enter_*' -p pid`

  (Note: both commands are used to trace system calls but the first command is about 5-6 times faster giving output thread wise while the second command gives a summarised stat of system calls for all threads running that process)
- Therefore concluded that perf trace command is way faster and used it in rest of our research.

SCREENSHOTS FOR PERF TRACE - FIREFOX

PESU CCBD

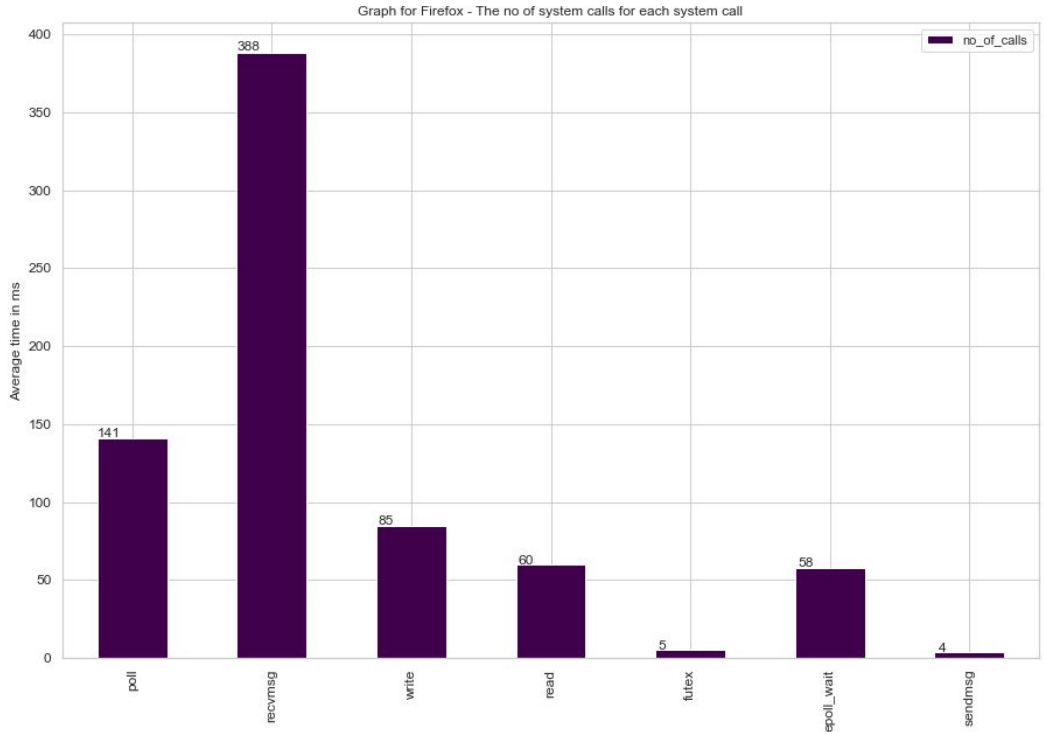
```
arpitkogta@ubuntu:~$ pidof firefox
11709 11655 11619 11600 11581 11543 11481 11434 11389
```

```
Summary of events:

Web Content (11709), 182 events, 82.7%

  syscall            calls  errors  total       min       avg       max       stddev
                                    (msec)    (msec)    (msec)    (msec)       (%)
  --------------- --------  ------ -------- --------- --------- ---------     ------
  poll                  29       0 43033.254     0.000  1483.905 18307.523     57.52%
  recvmsg               41      34     0.495     0.004     0.012     0.074     22.53%
  writev                 7       0     0.259     0.014     0.037     0.050     15.68%
  read                   6       0     0.044     0.005     0.007     0.012     13.70%
  write                  5       0     0.043     0.005     0.009     0.013     21.54%
  recvfrom               2       0     0.019     0.008     0.009     0.010     10.92%
  futex                  1       0     0.011     0.011     0.011     0.011      0.00%


IPC I/O Child (11714), 28 events, 12.7%

  syscall            calls  errors  total       min       avg       max       stddev
                                    (msec)    (msec)    (msec)    (msec)       (%)
  --------------- --------  ------ -------- --------- --------- ---------     ------
  epoll_wait             4       0 27937.488     0.000  6984.372 27936.509    100.00%
  recvmsg                4       2     0.040     0.006     0.010     0.015     24.25%
  write                  2       0     0.038     0.018     0.019     0.020      3.58%
  sendmsg                2       0     0.022     0.010     0.011     0.012      7.89%
  read                   2       0     0.014     0.007     0.007     0.007      5.19%


Timer (11721), 10 events, 4.5%

  syscall            calls  errors  total       min       avg       max       stddev
                                    (msec)    (msec)    (msec)    (msec)       (%)
  --------------- --------  ------ -------- --------- --------- ---------     ------
  futex                  4       1     0.408     0.000     0.102     0.396     96.03%
  write                  1       0     0.039     0.039     0.039     0.039      0.00%
```

```
arpitkogta@ubuntu:~$ sudo perf trace -p 11709 -s -o op1.txt
[sudo] password for arpitkogta:
^Carpitkogta@ubuntu:~$ sudo perf trace -p 11655 -s -o op2.txt
^Carpitkogta@ubuntu:~$ sudo perf trace -p 11619 -s -o op3.txt
^Carpitkogta@ubuntu:~$ sudo perf trace -p 11600 -s -o op4.txt
^Carpitkogta@ubuntu:~$ sudo perf trace -p 11581 -s -o op5.txt
^Carpitkogta@ubuntu:~$ sudo perf trace -p 11543 -s -o op6.txt
^Carpitkogta@ubuntu:~$ sudo perf trace -p 11543 -s -o op6.txt
```

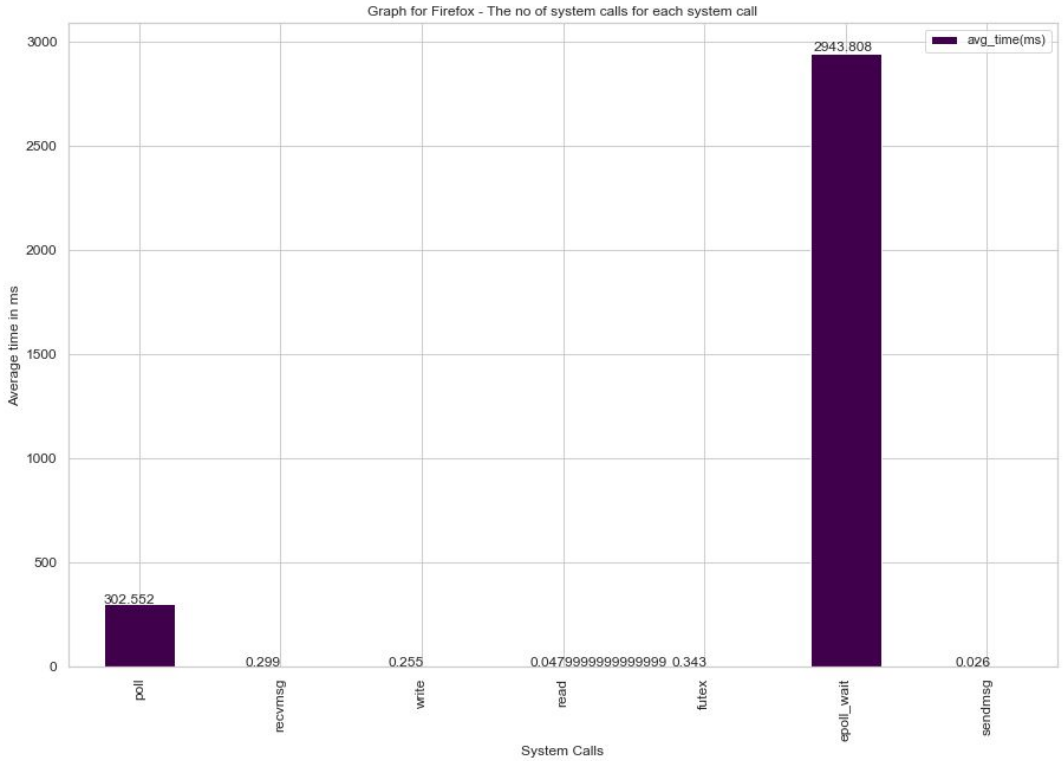
Link for all output files:O/P Files

GRAPH FOR PERF TRACE - FIREFOX



Link to O/P files

GRAPH FOR PERF TRACE - FIREFOX



EXECUTING AND TRACKING PERFORMANCE VIA PERF FOR WORD/EXCEL

PESU CCBD

- **Commands Used:**
  - `pidof soffice.bin`
  - `perf trace -p pid -s`
- We ran equivalent of word and excel, that is, libreoffice writer and libreoffice calc in the background and traced their system calls using the perf command.
- While tracing the system calls we also performed some computations like creating a table in writer and performing sum and product functions in calc.
- **Conclusions:** From the traced output data we could conclude that the epoll_wait and poll system call had the highest average completion time while the recvmsg system call and the poll system calls were the most frequently called. Also the poll system call had a higher average completion time in libreoffice writer than libreoffice calc.

Link to O/P Data Files

SCREENSHOTS FOR PERF TRACE - WORD/EXCEL

PESU CCBD

```
arpitkogta@ubuntu:~$ sudo perf trace -p 6574 -s -o output.txt
[sudo] password for arpitkogta:
Sorry, try again.
[sudo] password for arpitkogta:
arpitkogta@ubuntu:~$ sudo perf trace -p 9868 -s -o output1.txt
arpitkogta@ubuntu:~$ sudo perf trace -p 10122 -s -o output1.txt
```

```
arpitkogta@ubuntu:~/redis/redis-6.2.3$ pidof soffice.bin
6574
```

Summary of events:

soffice.bin (10122), 31975 events, 99.9%

syscall	calls	errors	total (msec)	min (msec)	avg (msec)	max (msec)	stddev (%)
poll	5690	0	45237.798	0.000	7.950	1427.568	7.21%
recvmsg	6711	4649	208.689	0.001	0.031	1.101	1.80%
writev	2062	0	96.707	0.002	0.047	1.674	3.19%
read	269	0	43.224	0.002	0.161	14.800	35.25%
brk	355	0	20.410	0.004	0.057	4.010	19.55%
stat	281	222	8.884	0.004	0.032	0.502	7.52%
openat	77	29	6.308	0.002	0.082	1.117	19.90%
pread64	80	0	5.032	0.001	0.063	1.872	42.25%
access	49	2	4.448	0.007	0.091	0.691	20.92%
mkdir	19	19	3.522	0.008	0.185	2.558	72.19%
lstat	8	0	2.561	0.007	0.320	1.661	66.35%
lseek	81	0	2.122	0.001	0.026	0.128	12.52%
futex	34	2	1.913	0.004	0.056	0.585	32.55%
mmap	42	0	1.636	0.008	0.039	0.231	15.33%
getdents64	8	0	1.616	0.006	0.202	1.343	80.93%
write	44	0	1.458	0.002	0.033	0.134	13.30%
clone	7	0	1.343	0.128	0.192	0.301	11.87%
close	48	0	0.861	0.002	0.018	0.173	22.77%
mprotect	15	0	0.739	0.016	0.049	0.103	15.13%
fstat	43	0	0.491	0.003	0.011	0.074	14.25%
fcntl	3	0	0.350	0.003	0.117	0.293	76.71%
clock_gettime	17	0	0.320	0.005	0.019	0.091	30.56%
munmap	3	0	0.274	0.047	0.091	0.116	24.19%
getuid	36	0	0.253	0.002	0.007	0.041	14.50%
getpid	5	0	0.043	0.008	0.009	0.010	5.88%

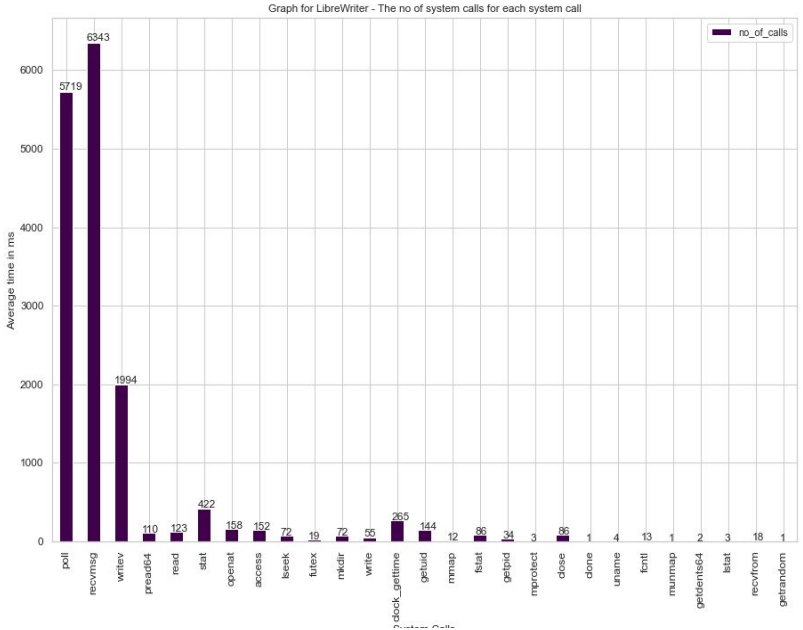
threaded-ml (10137), 18 events, 0.1%

syscall	calls	errors	total (msec)	min (msec)	avg (msec)	max (msec)	stddev (%)
futex	3	0	0.536	0.008	0.179	0.516	94.39%
poll	2	0	0.331	0.000	0.166	0.331	100.00%
read	3	1	0.104	0.011	0.035	0.059	40.27%
write	1	0	0.021	0.021	0.021	0.021	0.00%

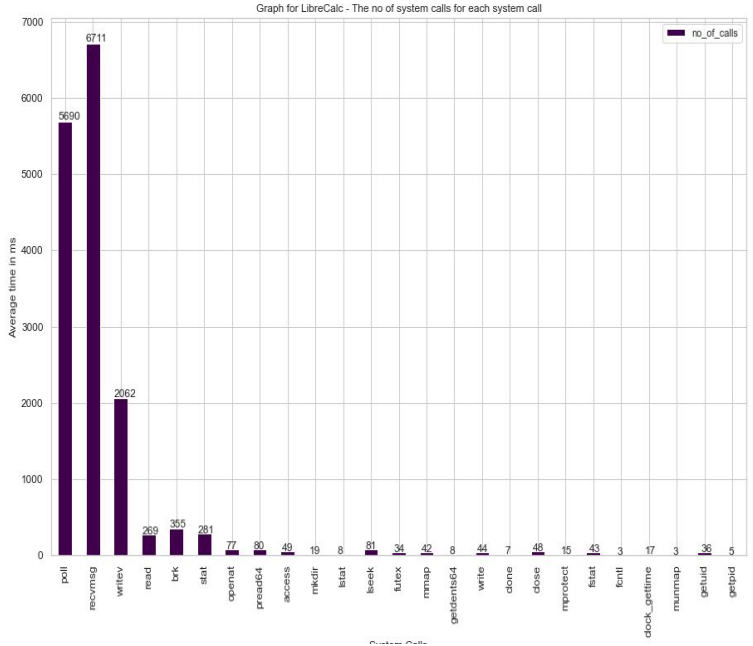
GRAPH FOR PERF TRACE - WORD/EXCEL

PESU CCBD

Word



Excel



EXECUTING AND TRACKING PERFORMANCE VIA PERF TRACE FOR REDIS

PESU CCBD

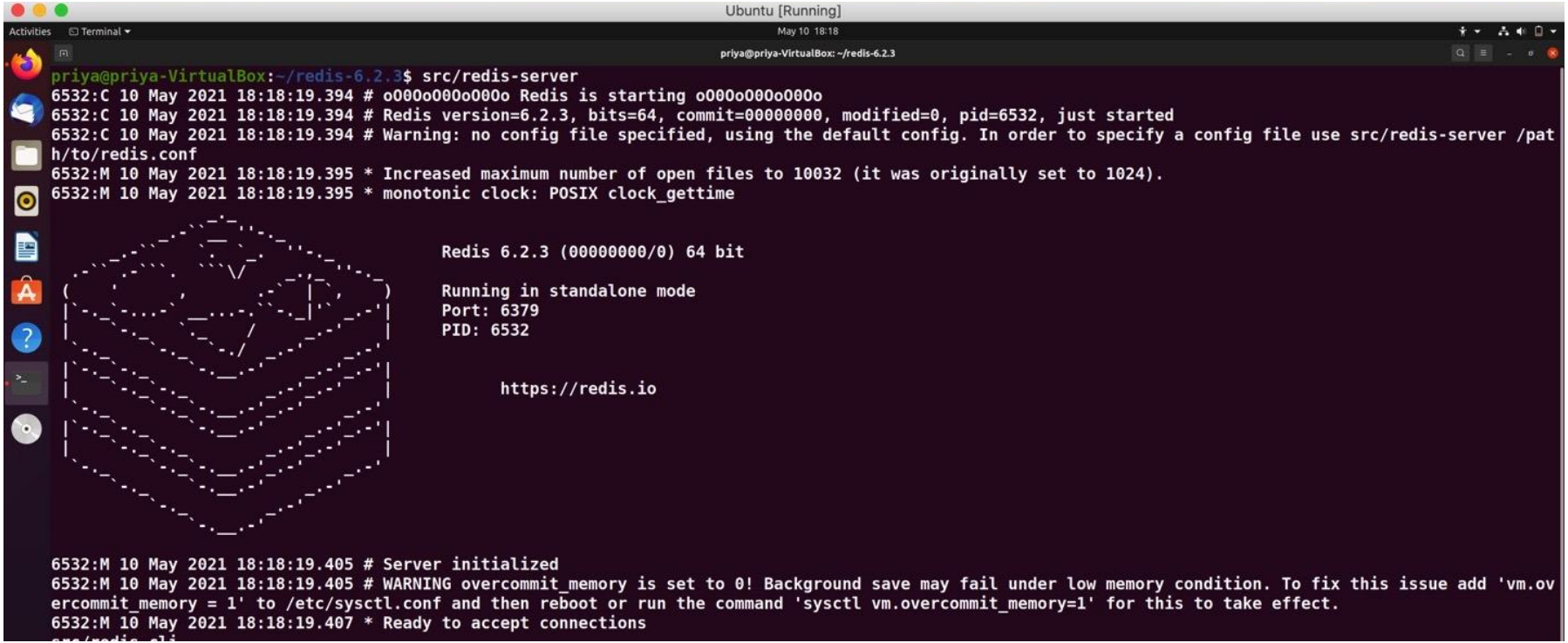
- **ABOUT REDIS** : Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker
- **COMMANDS USED** :
  - TO INSTALL REDIS

    ```
    $ wget https://download.redis.io/releases/redis-6.2.3.tar.gz
    $ tar xzf redis-6.2.3.tar.gz
    $ cd redis-6.2.3
    $ make
    ```
  - TO TRACK SYSTEM CALLS WHEN REDIS-SERVER WAS BEING CALLED :

    ```
    pidof redis-server
    sudo perf trace -s -p pid -o outputfilename
    ```

SCREENSHOTS - PERF TRACE REDIS

PESU CCBD



SCREENSHOTS - PERF TRACE REDIS

LINK FOR REDIS OUTPUT - Redis Output Name vs Number of System Calls
Redis Output Avg Time vs Name of the system call

OBSERVATIONS - PERF TRACE - REDIS

PESU CCBD

Observations:

epoll_wait() had the largest avg time.

The epoll_wait() system call waits for events on the epoll(7) instance referred to by the file descriptor *epfd*.

getpid() had the smallest avg time

getpid() returns the process ID (PID) of the calling process.

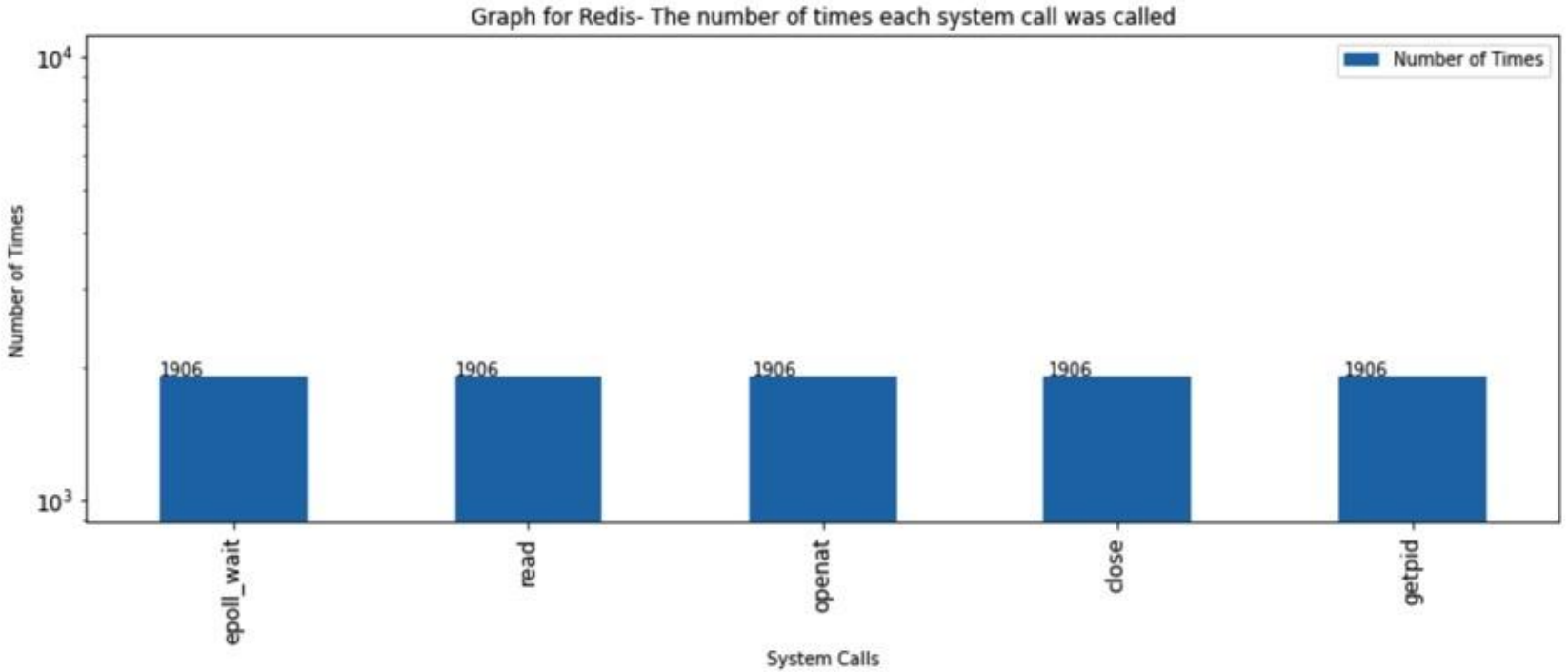
epoll_wait(), read() , opennat() , close(), getpid() all had equal number of system calls

	A	B	C	D
1	Serial Number	System-Calls	Average Time in ms	
2	1	epoll_wait	102.873	
3	2	read	0.294	
4	3	openat	0.144	
5	4	close	0.056	
6	5	getpid	0.042	
7				

	A	B	C
1	Serial Number	System-Calls	Number of Times
2	1	epoll_wait	1906
3	2	read	1906
4	3	openat	1906
5	4	close	1906
6	5	getpid	1906
7			
8			

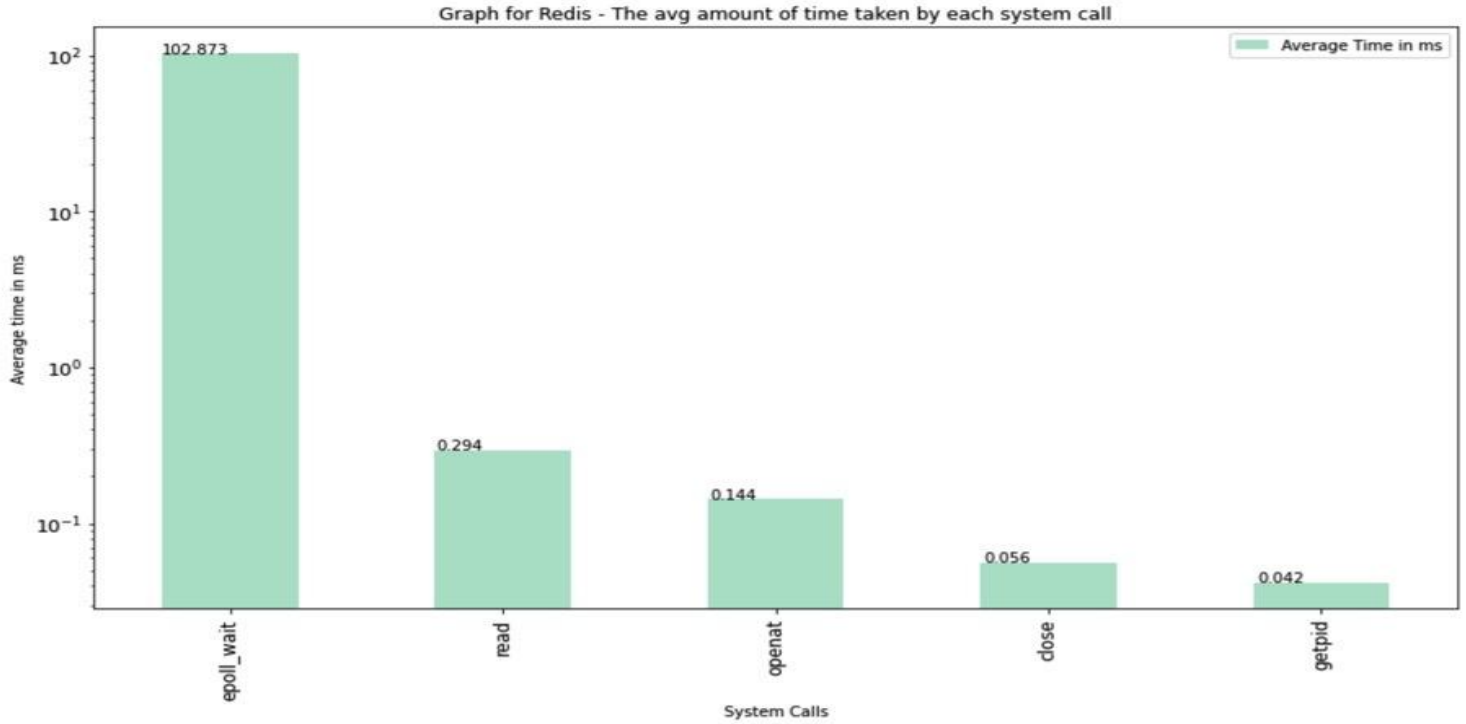
GRAPHS- PERF TRACE - REDIS

PESU CCBD



GRAPHS- PERF TRACE - REDIS

PESU CCBD



EXECUTING AND TRACKING PERFORMANCE VIA PERF TRACE FOR REDIS-BENCHMARK

PESU CCBD

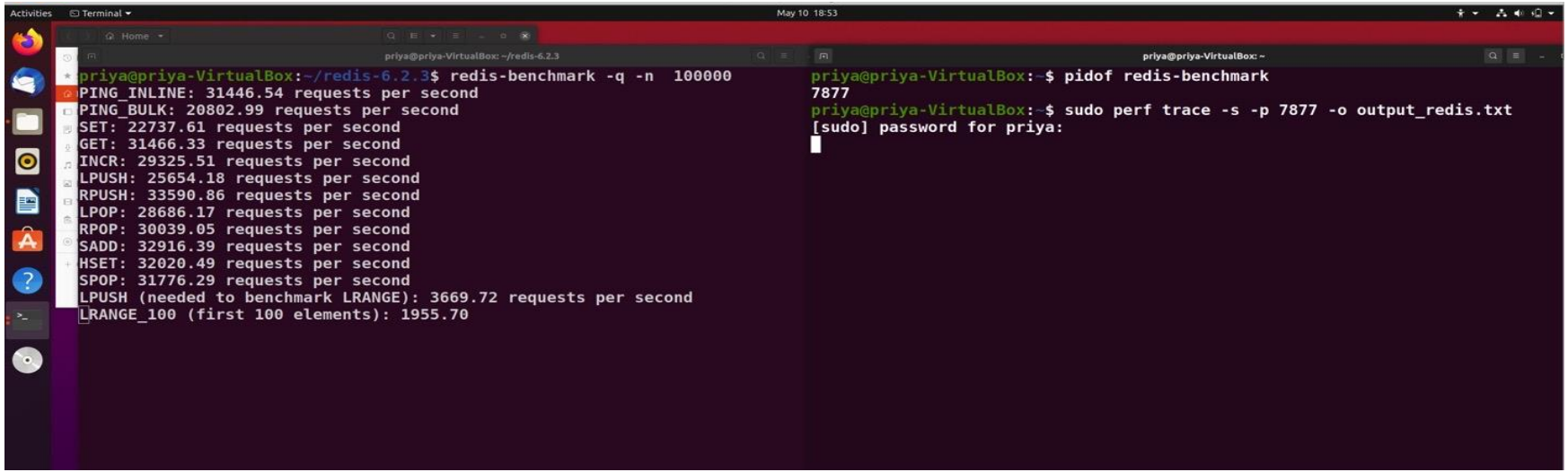
ABOUT REDIS-BENCHMARK :

The redis-benchmark program is a quick and useful way to get some figures and evaluate the performance of a Redis instance on a given hardware

COMMANDS USED :

`redis-benchmark -q -n noofrequests`

SCREENSHOTS PERF TRACE FOR REDIS-BENCHMARK



LINK FOR REDIS OUTPUT - [REDIS BENCHMARK AVG TIME VS NAME OF SYSTEM CALL]
[REDIS BENCHMARK NUMBER OF SYSTEM CALLS VS NAME]

OBSERVATIONS - PERF TRACE - REDIS

PESU CCBD

Observations:

connect() had the largest avg time

The connect() system call connects the socket referred to by the file descriptor *sockfd* to the address specified by *addr*

close() had the least avg time

close() closes a file descriptor

mmap() was called the least number of times.

mmap() creates a new mapping in the virtual address space of the calling process

	A	B	C
1	System-Calls	Average Time in ms	
2	connect	0.129	
3	epoll_wait	0.04	
4	madvise	0.037	
5	write	0.033	
6	socket	0.026	
7	epoll_ctl	0.022	
8	read	0.022	
9	setsockopt	0.021	
10	fcntl	0.008	
11	close	0.007	
12			

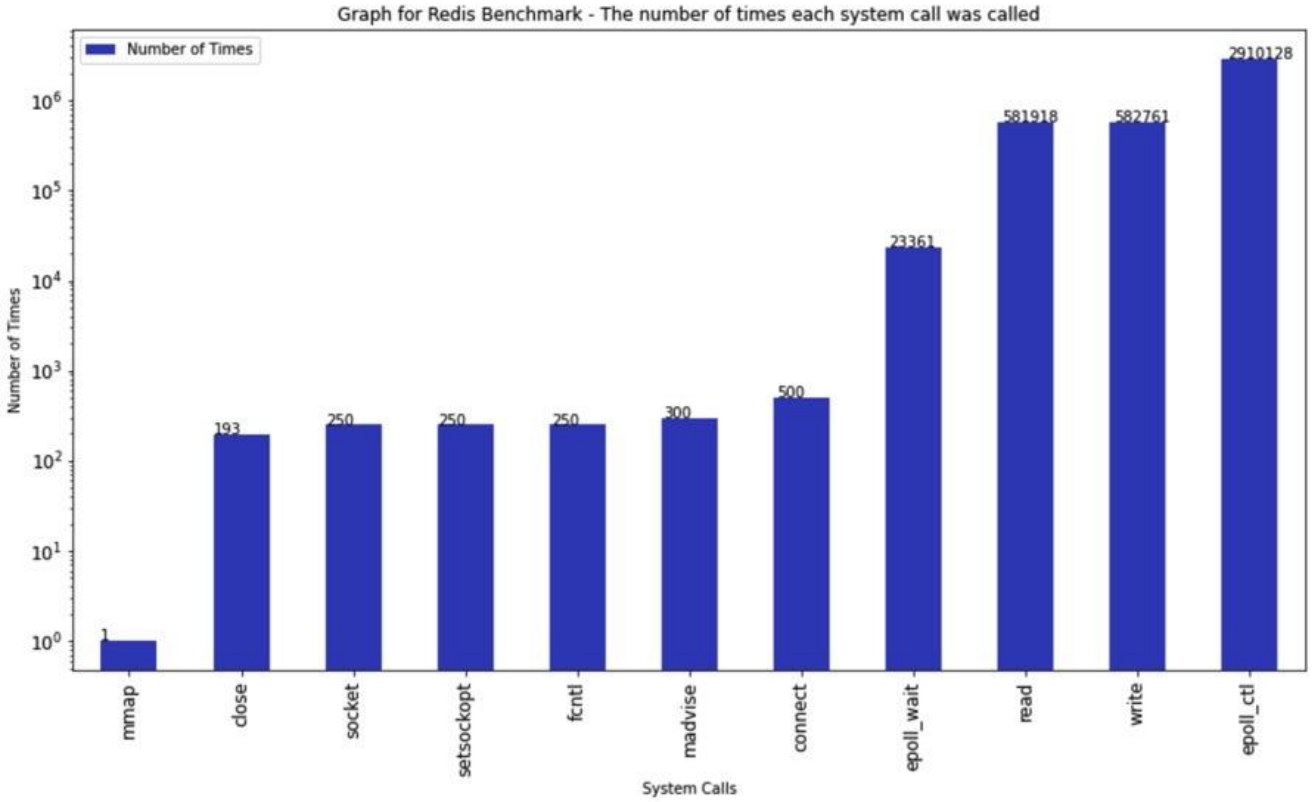
	A	B
1	System-Calls	Number of Times
2	mmap	1
3	close	193
4	socket	250
5	setsockopt	250
6	fcntl	250
7	madvise	300
8	connect	500
9	epoll_wait	23361
10	read	581918
11	write	582761

epoll_ctl() was called maximum number of times

This system call is used to add, modify, or remove entries in the interest list of the epoll(7) instance referred to by the file descriptor *epfd*

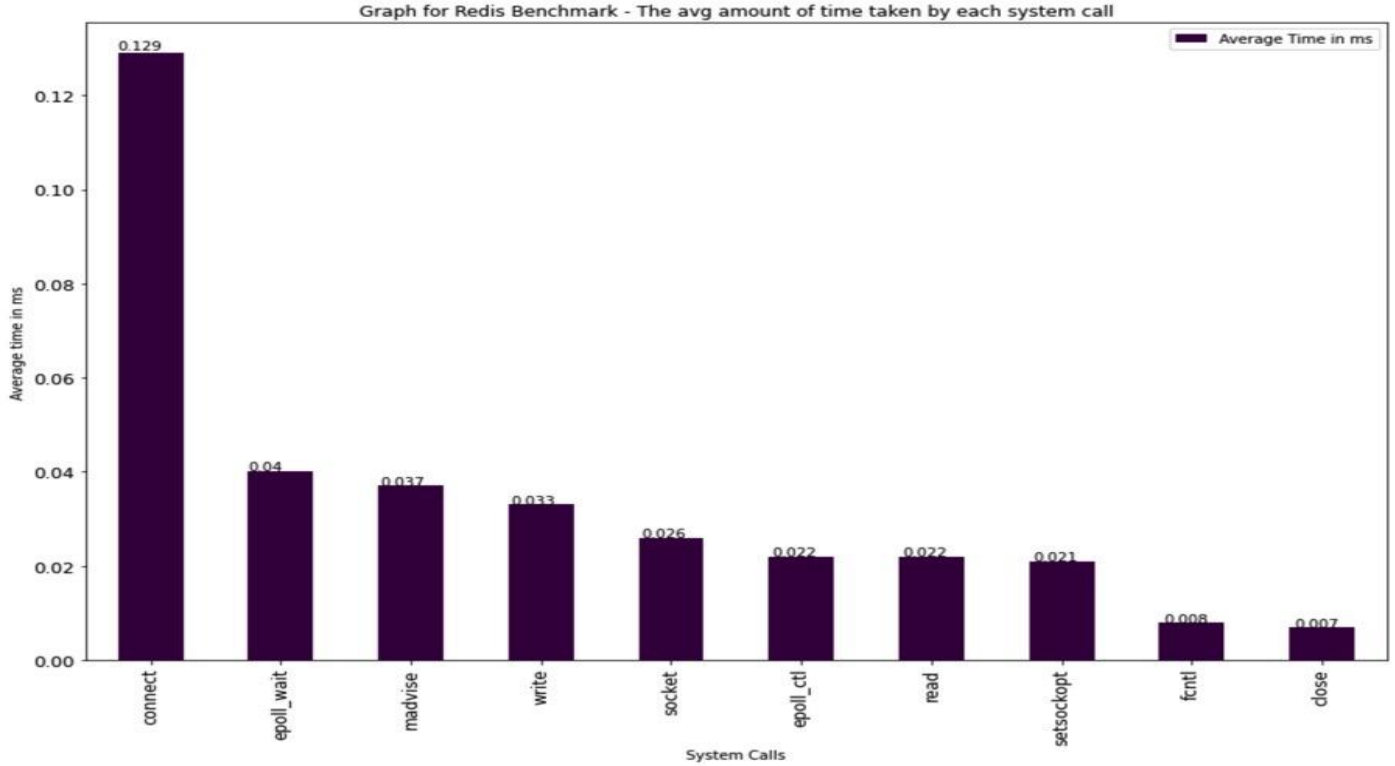
GRAPHS- PERF TRACE - REDIS BENCHMARK

PESU CCBD



GRAPHS- PERF TRACE - REDIS BENCHMARK

PESU CCBD



EXECUTING AND TRACKING PERFORMANCE VIA PERF TRACE FOR “LS”

PESU CCBD

ABOUT LS : It is used to list directory contents

COMMANDS USED :

```
ls
pidof ls
sudo perf trace -p pid -s -o outputfilename
```

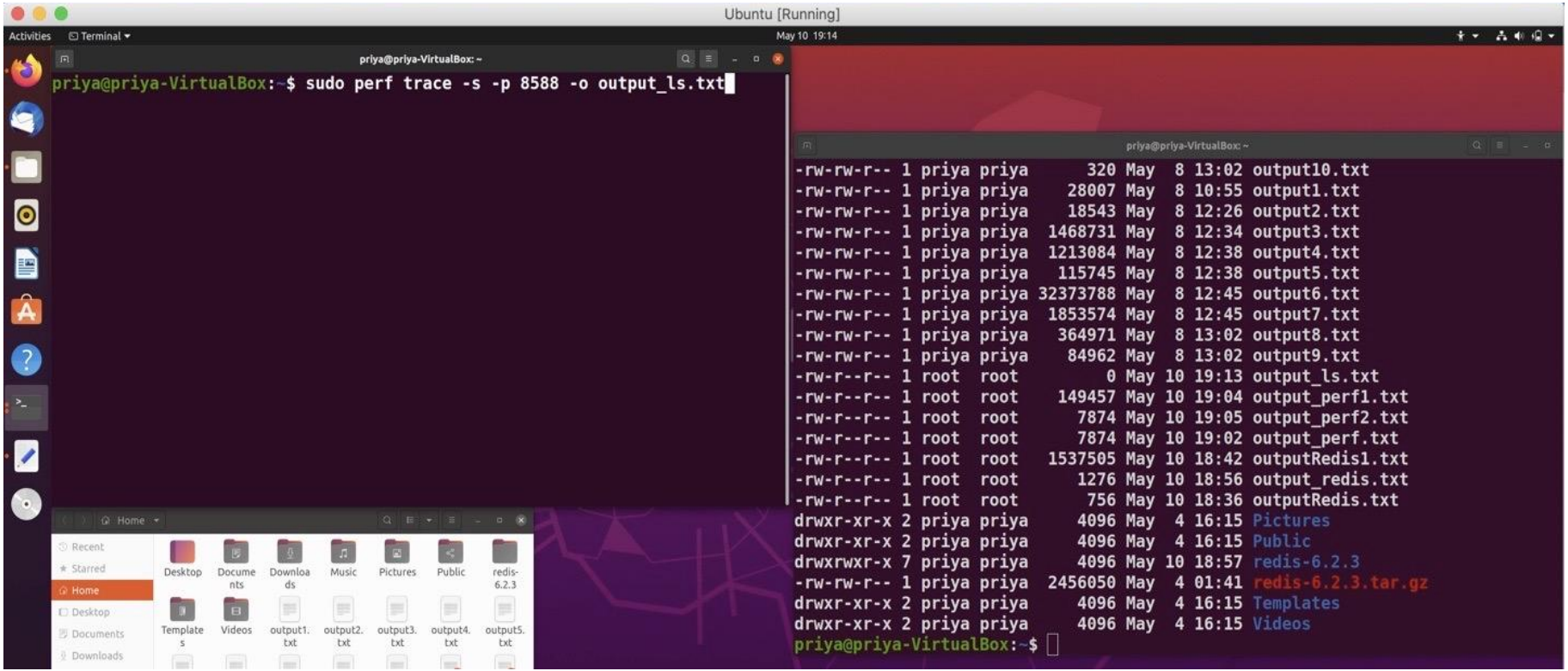
LINK FOR OUTPUT FILES:

LS OUTPUT NUMBER OF SYSTEM CALLS VS NAME OF THE SYSTEM CALL

LS OUTPUT NAME OF THE SYSTEM CALL VS AVG TIME

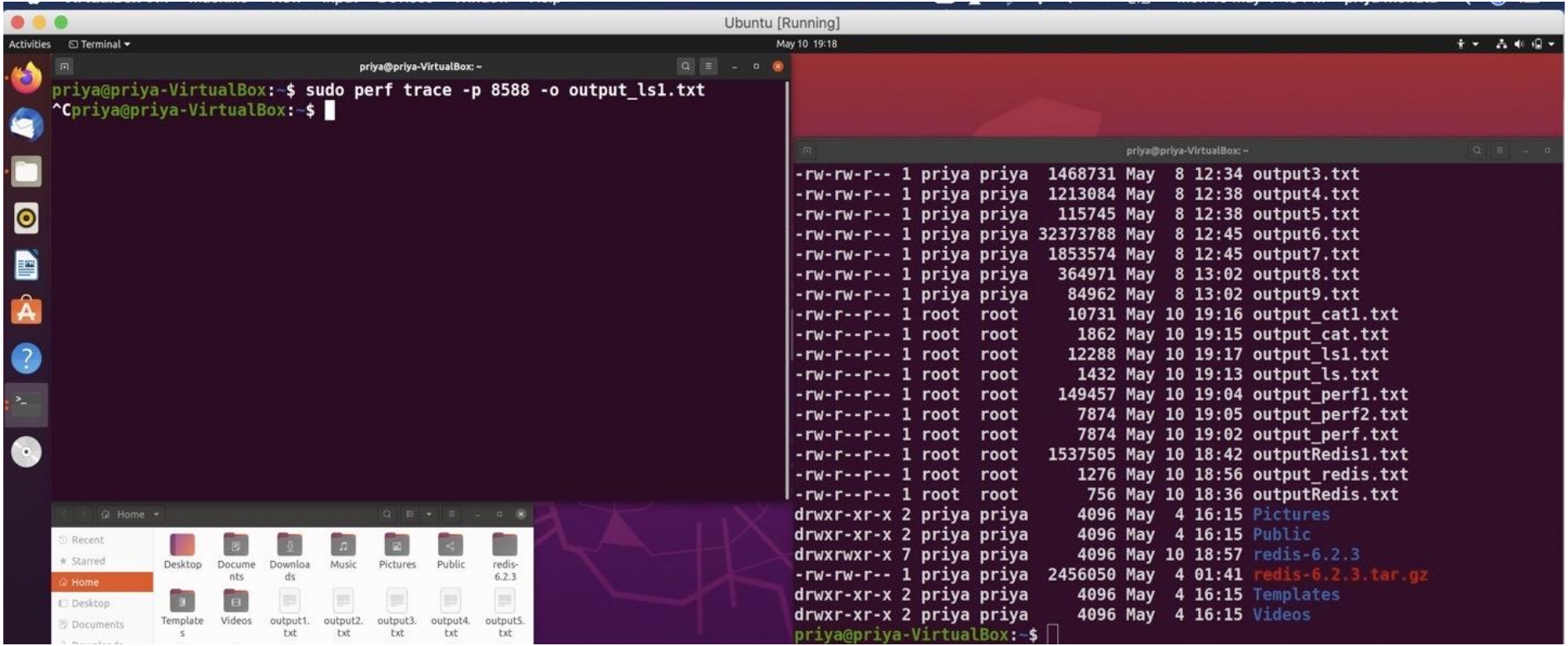
SCREENSHOTS - LS - PERF TRACE

PESU CCBD



SCREENSHOTS - LS - PERF TRACE

PESU CCBD



OBSERVATIONS - LS - PERF TRACE

PESU CCBD

rt_sigaction() was called maximum number of times.

The **sigaction**() system call is used to change the action taken by a process on receipt of a specific signal

fcntl() was called minimum number of times

fcntl() performs one of the operations described below on the open file descriptor *fd*

pselect6() had the maximum average time

pselect() allow a program to monitor multiple file descriptors, waiting until one or more of the file descriptors become "ready" for some class of I/O operation

close() had the minimum average time

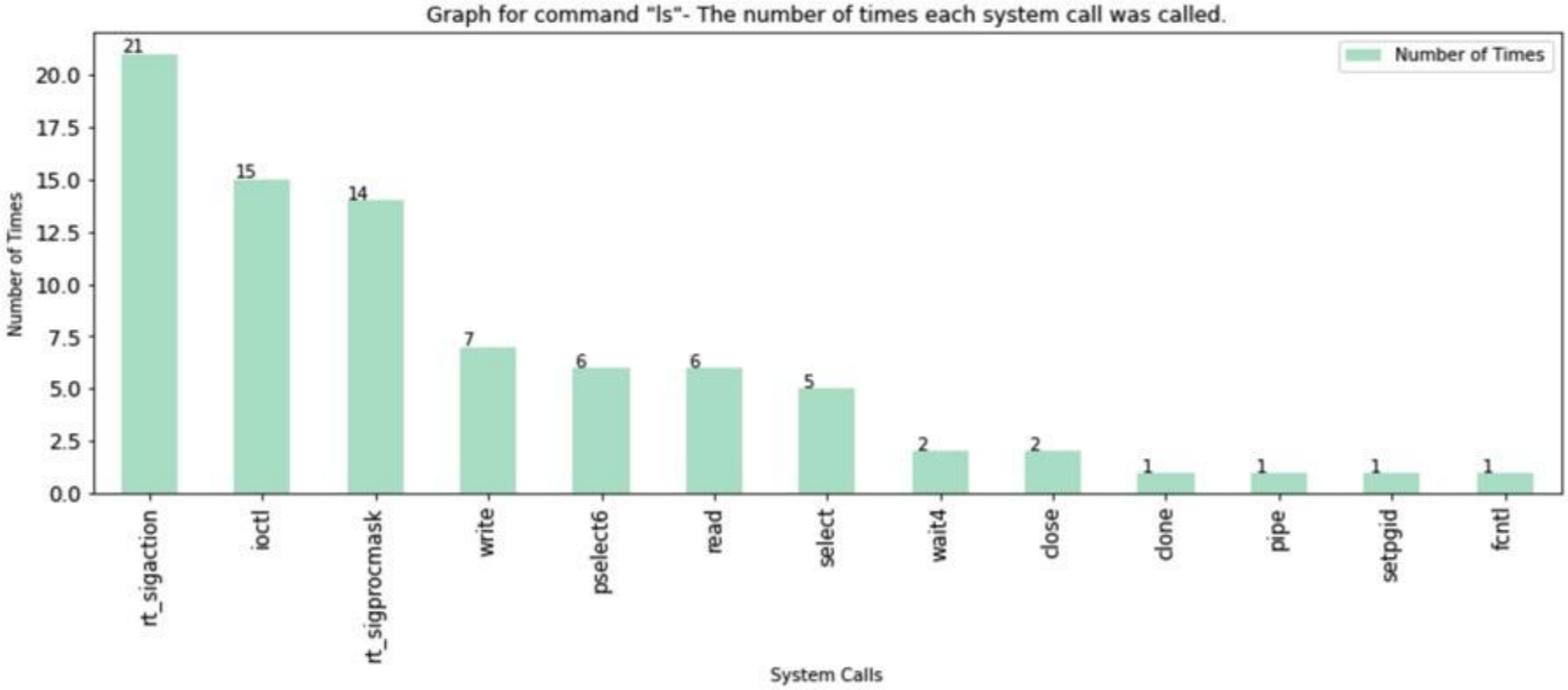
close() closes a file descriptor

	A	B
1	System-Calls	Number of Times
2	rt_sigaction	21
3	ioctl	15
4	rt_sigprocmask	14
5	write	7
6	pselect6	6
7	read	6
8	select	5
9	wait4	2
10	close	2
11	clone	1
12	pipe	1
13	setpgid	1
14	fcntl	1

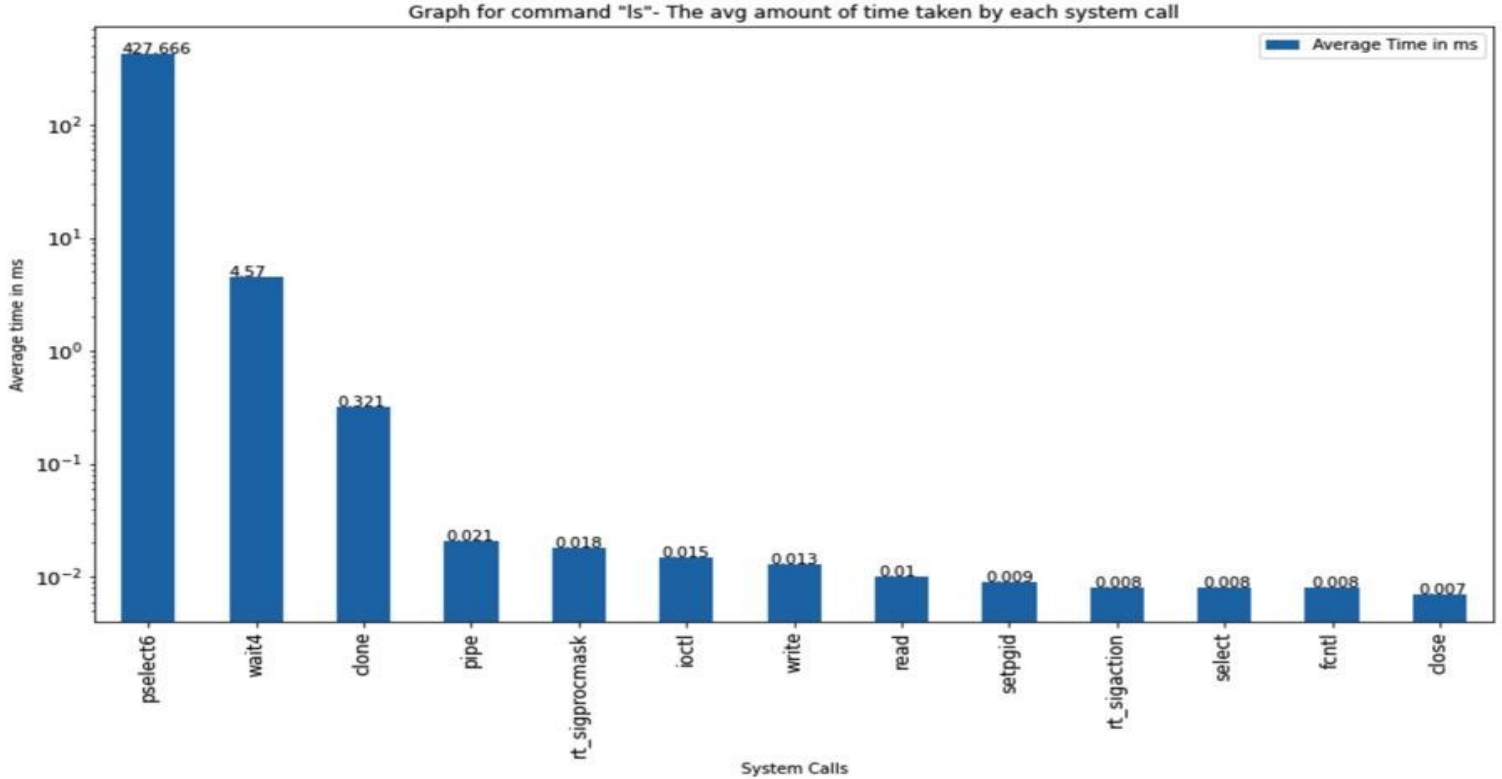
	A	B
1	System-Calls	Average Time in ms
2	pselect6	427.666
3	wait4	4.57
4	clone	0.321
5	pipe	0.021
6	rt_sigprocmask	0.018
7	ioctl	0.015
8	write	0.013
9	read	0.01
10	setpgid	0.009
11	rt_sigaction	0.008
12	select	0.008
13	fcntl	0.008
14	close	0.007

GRAPHS- LS - PERF TRACE

PESU CCBD



GRAPHS- LS - PERF TRACE



EXECUTING AND TRACKING PERFORMANCE VIA PERF TRACE FOR “CAT”

PESU CCBD

ABOUT CAT : It is used to display file contents

COMMANDS USED :

```
cat filename
pidof cat
sudo perf trace -p pid -s -o outputfilename
```

LINK FOR THE OUTPUT FILES :

CAT OUTPUT NUMBER OF SYSTEM CALLS VS NAME OF THE SYSTEM CALL

CAT OUTPUT NAME OF THE SYSTEM CALLS VS NUMBER OF SYSTEM CALLS

OBSERVATIONS - CAT - PERF TRACE

PESU CCBD

rt_sigaction() was called maximum number of times.

The **sigaction**() system call is used to change the action taken by a process on receipt of a specific signal

fcntl() was called minimum number of times

fcntl() performs one of the operations described below on the open file descriptor *fd*

wait4() had the maximum average time

wait4() wait for process to change state

pselect6() had the minimum average time

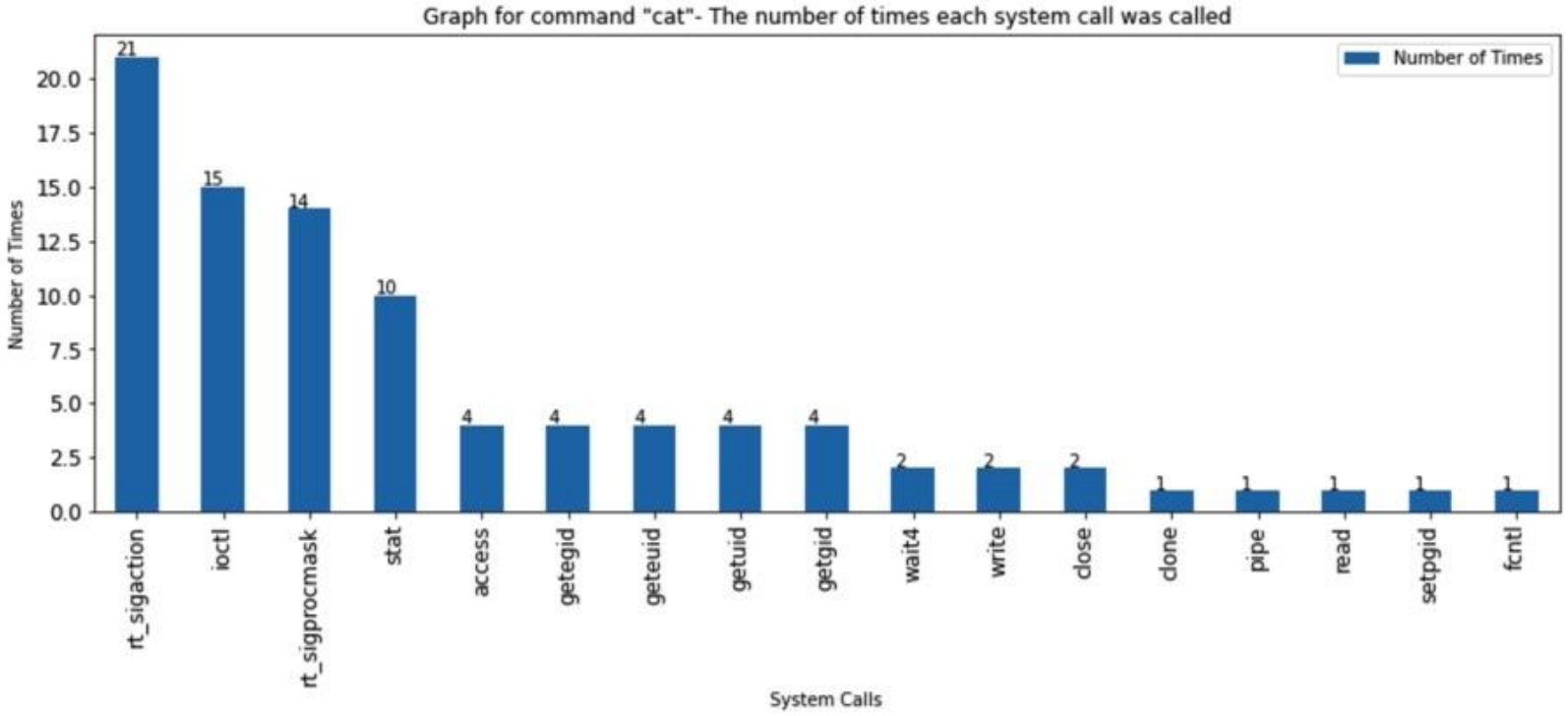
pselect() allow a program to monitor multiple file descriptors, waiting until one or more of the file descriptors become "ready" for some class of I/O operation

	A	B
1	System-Calls	Number of Times
2	rt_sigaction	21
3	ioctl	15
4	rt_sigprocmask	14
5	stat	10
6	access	4
7	getegid	4
8	geteuid	4
9	getuid	4
10	getgid	4
11	wait4	2
12	write	2
13	close	2
14	clone	1
15	pipe	1
16	read	1
17	setpgid	1
18	fcntl	1
19		

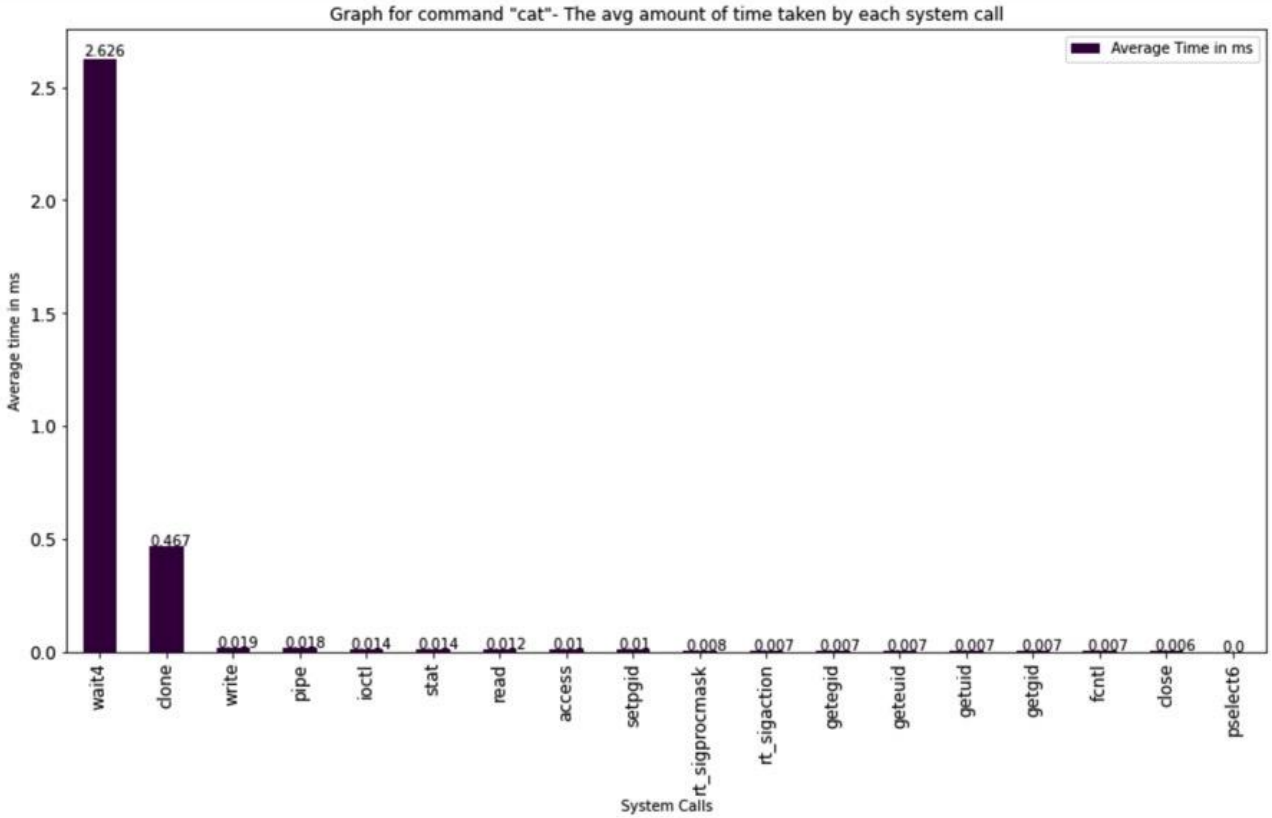
	A	B
1	System-Calls	Average Time in ms
2	wait4	2.626
3	clone	0.467
4	write	0.019
5	pipe	0.018
6	ioctl	0.014
7	stat	0.014
8	read	0.012
9	access	0.01
10	setpgid	0.01
11	rt_sigprocmask	0.008
12	rt_sigaction	0.007
13	getegid	0.007
14	geteuid	0.007
15	getuid	0.007
16	getgid	0.007
17	fcntl	0.007
18	close	0.006
19	pselect6	0
20		

GRAPHS- CAT - PERF TRACE

PESU CCBD



GRAPHS- CAT - PERF TRACE



EXECUTING AND TRACKING PERFORMANCE VIA PERF TRACE FOR IFCONFIG COMMAND

PESU CCBD

ABOUT IFCONFIG :

ifconfig stands for "interface configuration." It is used to view and change the configuration of the network interfaces on your system.

COMMANDS USED:

```
ifconfig
pidof ifconfig
sudo perf trace -s -p pid -o outputfilename
```

LINK FOR THE OUTPUT FILES :

IFCONFIG OUTPUT FOR AVG TIME VS NAME OF THE SYSTEM CALL

IFCONFIG OUTPUT FOR NAME OF THE SYSTEM CALL VS NUMBER OF TIMES SYSTEM CALL WAS CALLED

OBSERVATIONS FOR PERF TRACE FOR IFCONFIG COMMAND

PESU CCBD

ioctl() was called maximum number of times

The **ioctl**() system call manipulates the underlying device parameters of special files

setpgid() was called minimum number of times

setpgid() sets the PGID of the process specified by *pid* to *pgid*

execve() had the maximum average time

execve() executes the program referred to by pathname

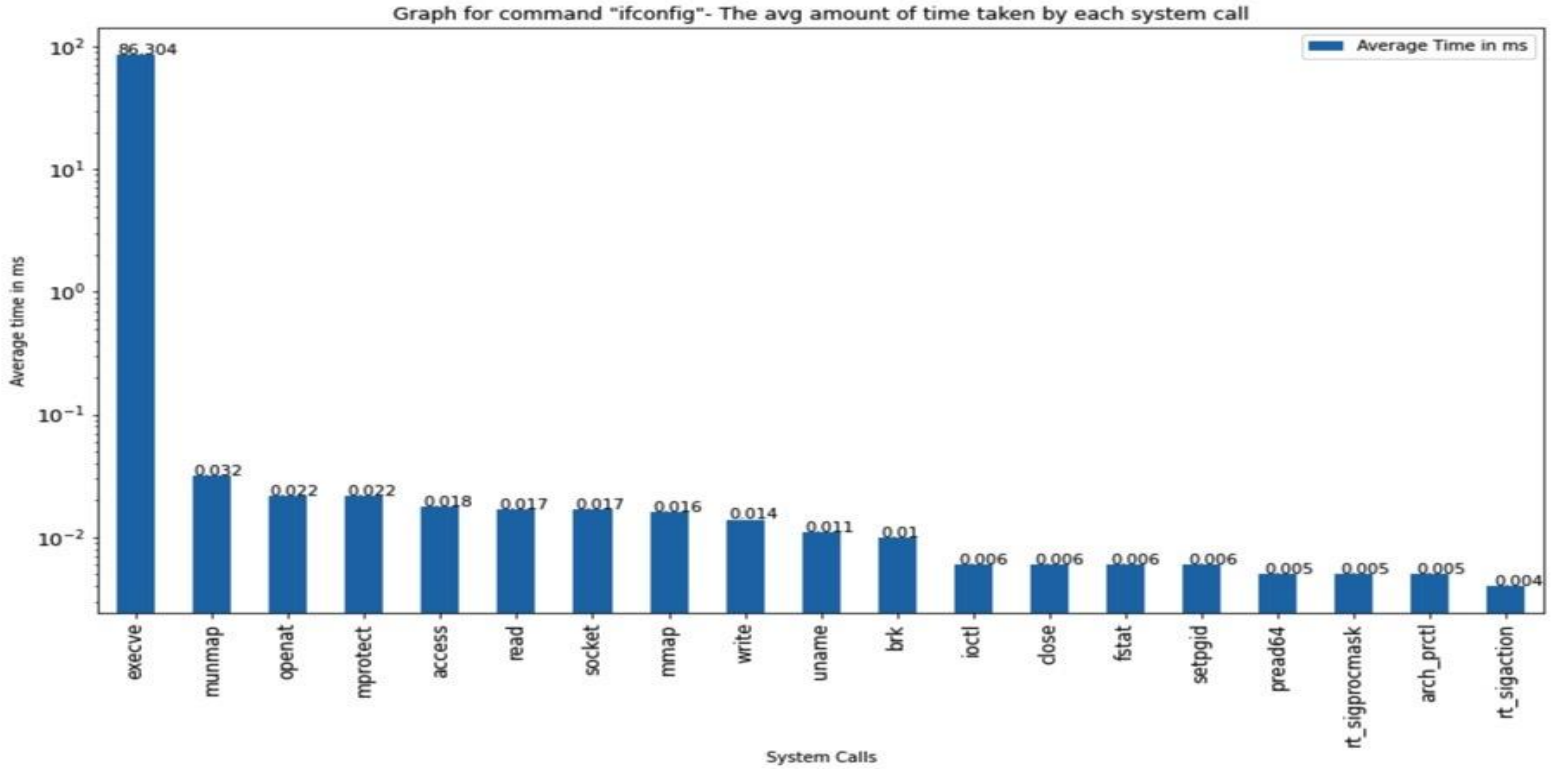
rt_sigaction() had the minimum average time

The **sigaction**() system call is used to change the action taken by a process on receipt of a specific signal

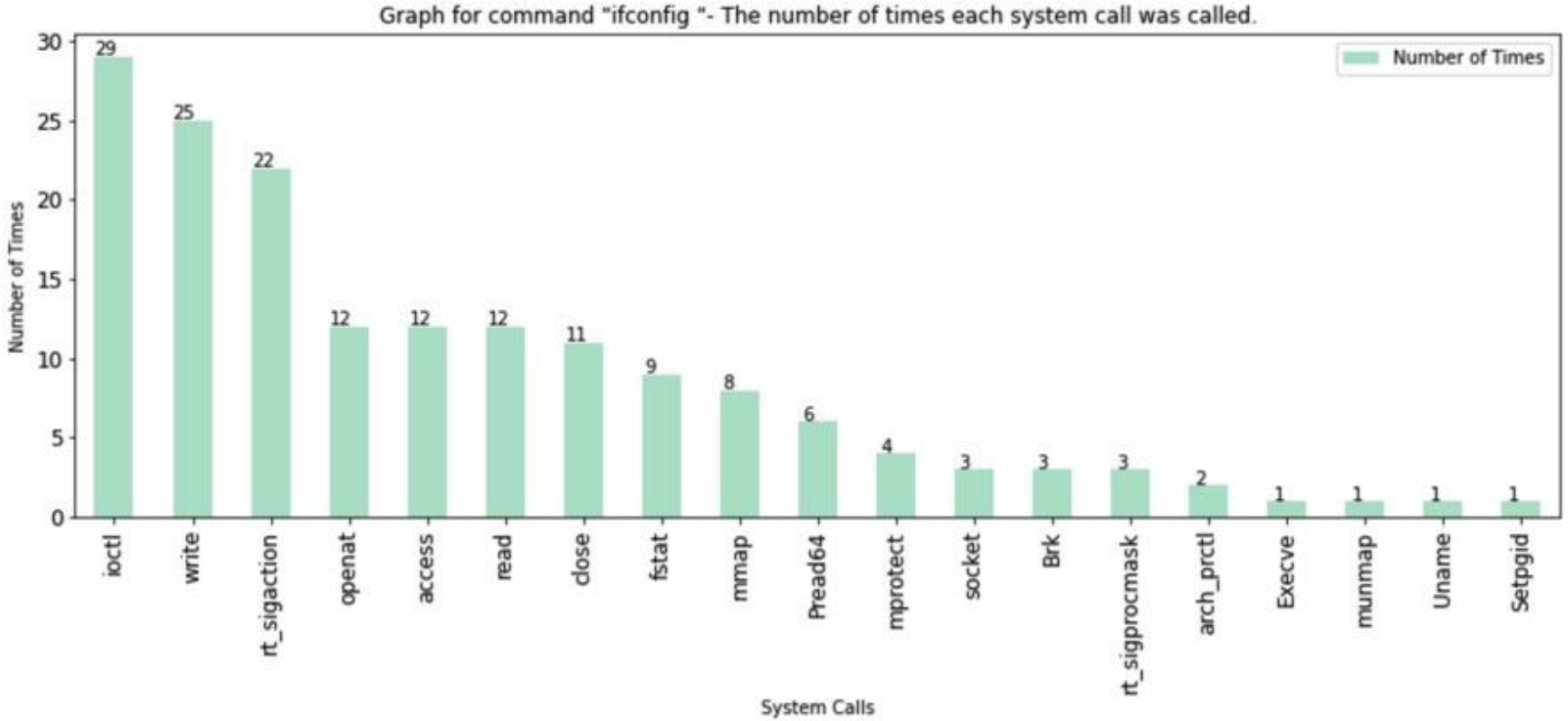
	A	B	C
1	System-Calls	Average Time in ms	
2	execve	86.304	
3	munmap	0.032	
4	openat	0.022	
5	mprotect	0.022	
6	access	0.018	
7	read	0.017	
8	socket	0.017	
9	mmap	0.016	
10	write	0.014	
11	uname	0.011	
12	brk	0.01	
13	ioctl	0.006	
14	close	0.006	
15	fstat	0.006	
16	setpgid	0.006	
17	pread64	0.005	
18	rt_sigprocmask	0.005	
19	arch_prctl	0.005	
20	rt_sigaction	0.004	

	A	B
1	System-Calls	Number of Times
2	ioctl	29
3	write	25
4	rt_sigaction	22
5	openat	12
6	access	12
7	read	12
8	close	11
9	fstat	9
10	mmap	8
11	Pread64	6
12	mprotect	4
13	socket	3
14	Brk	3
15	rt_sigprocmask	3
16	arch_prctl	2
17	Execve	1
18	munmap	1
19	Uname	1
20	Setpgid	1

SCREENSHOTS FOR PERF TRACE FOR IFCONFIG COMMAND



SCREENSHOTS FOR PERF TRACE FOR IFCONFIG COMMAND



HOW MANY THREADS ARE USED BY EACH PROCESS?

PESU CCBD

COMMANDS USED :

- ps -eLf
- top

Meaning of the options :

-e => To list all the processes

-L => Show threads, possibly with LWP and NLWP columns.

-f => Do full-format listing

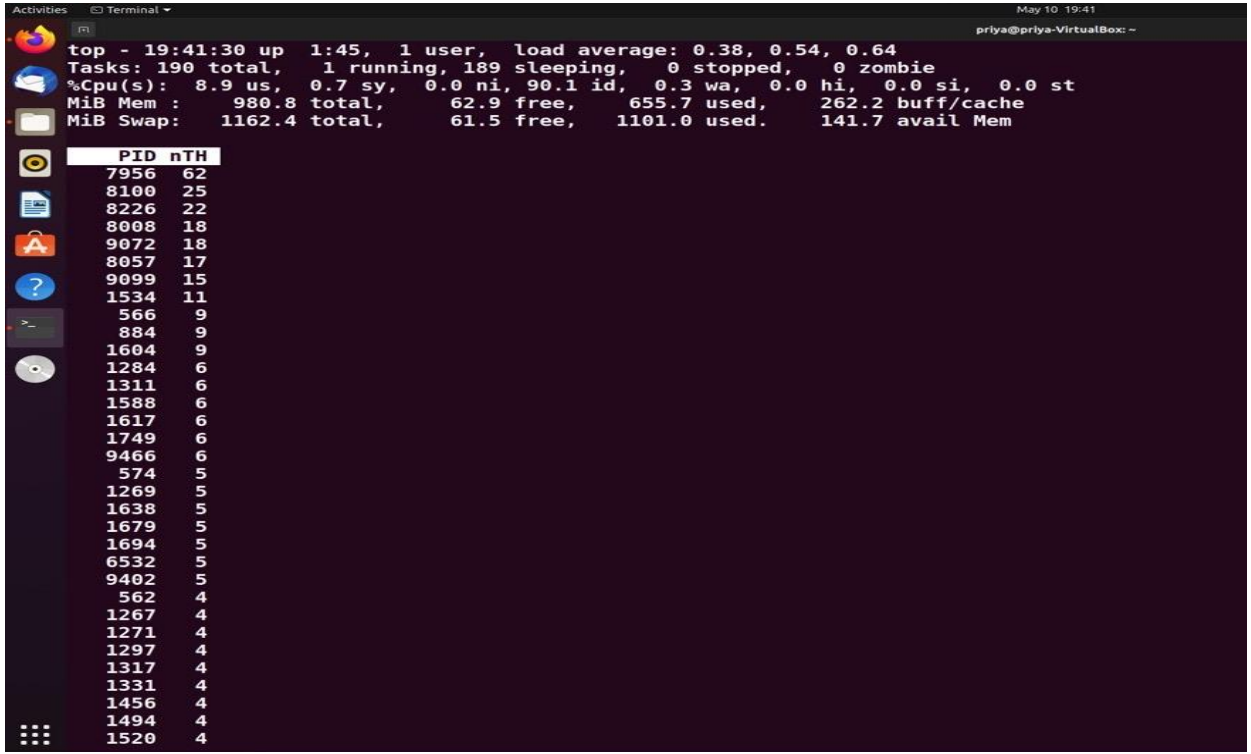
LINK FOR THE OUTPUT FILES :

Output file of ps -eLF

HOW MANY THREADS ARE USED BY EACH PROCESS?

PESU CCBD

OUTPUT SCREENSHOTS : COMMAND - TOP



HOW MANY THREADS ARE USED BY EACH PROCESS?

PESU CCBD

OUTPUT SCREENSHOTS : COMMAND - ps -eLf

```
priya@priya-VirtualBox:~$ ps -eLf
UID          PID    PPID     LWP  C NLWP STIME TTY          TIME CMD
root           1       0       1  1    1 18:07 ?        00:00:02 /sbin/init splash
root           2       0       2  0    1 18:07 ?        00:00:00 [kthreadd]
root           3       2       3  0    1 18:07 ?        00:00:00 [rcu_gp]
root           4       2       4  0    1 18:07 ?        00:00:00 [rcu_par_gp]
root           5       2       5  0    1 18:07 ?        00:00:00 [kworker/0:0-events]
root           6       2       6  0    1 18:07 ?        00:00:00 [kworker/0:0H-kblockd]
root           7       2       7  0    1 18:07 ?        00:00:00 [kworker/0:1-events]
root           8       2       8  0    1 18:07 ?        00:00:00 [kworker/u2:0-events_unbound]
root           9       2       9  0    1 18:07 ?        00:00:00 [mm_percpu_wq]
root          10       2      10  0    1 18:07 ?        00:00:00 [ksoftirqd/0]
root          11       2      11  0    1 18:07 ?        00:00:01 [rcu_sched]
root          12       2      12  0    1 18:07 ?        00:00:00 [migration/0]
root          13       2      13  0    1 18:07 ?        00:00:00 [idle_inject/0]
root          14       2      14  0    1 18:07 ?        00:00:00 [cpuhp/0]
root          15       2      15  0    1 18:07 ?        00:00:00 [kdevtmpfs]
root          16       2      16  0    1 18:07 ?        00:00:00 [netns]
root          17       2      17  0    1 18:07 ?        00:00:00 [rcu_tasks_kthre]
root          18       2      18  0    1 18:07 ?        00:00:00 [rcu_tasks_rude_]
root          19       2      19  0    1 18:07 ?        00:00:00 [rcu_tasks_trace]
root          20       2      20  0    1 18:07 ?        00:00:00 [kauditd]
root          21       2      21  0    1 18:07 ?        00:00:00 [khungtaskd]
root          22       2      22  0    1 18:07 ?        00:00:00 [oom_reaper]
root          23       2      23  0    1 18:07 ?        00:00:00 [writeback]
root          24       2      24  0    1 18:07 ?        00:00:00 [kcompactd0]
root          25       2      25  0    1 18:07 ?        00:00:00 [ksmd]
root          26       2      26  0    1 18:07 ?        00:00:00 [khugepaged]
root          72       2      72  0    1 18:07 ?        00:00:00 [kintegrityd]
root          73       2      73  0    1 18:07 ?        00:00:00 [kblockd]
root          74       2      74  0    1 18:07 ?        00:00:00 [blkcg_punt_bio]
root          75       2      75  0    1 18:07 ?        00:00:00 [tpm_dev_wq]
root          76       2      76  0    1 18:07 ?        00:00:00 [ata_sff]
root          77       2      77  0    1 18:07 ?        00:00:00 [md]
root          78       2      78  0    1 18:07 ?        00:00:00 [edac-poller]
root          79       2      79  0    1 18:07 ?        00:00:00 [devfreq_wq]
root          80       2      80  0    1 18:07 ?        00:00:00 [watchdogd]
root          81       2      81  0    1 18:07 ?        00:00:00 [kworker/u2:1-ext4-rsv-conversion]
root          82       2      82  0    1 18:07 ?        00:00:00 [pm_wq]
root          84       2      84  2    1 18:07 ?        00:00:06 [kswapd0]
```

HOW MANY THREADS ARE USED BY EACH PROCESS?

PESU CCBD

GRAPH: COMMAND - ps -eLf

Stats data provided by command:

- UID:determines users
- PID:Unique id for each process
- PPID:Determines parent of a process by showing id.
- LWP: Light weight process(Kernel threads attached to a process)
- NLWP:No. of kernel threads attached to a process.

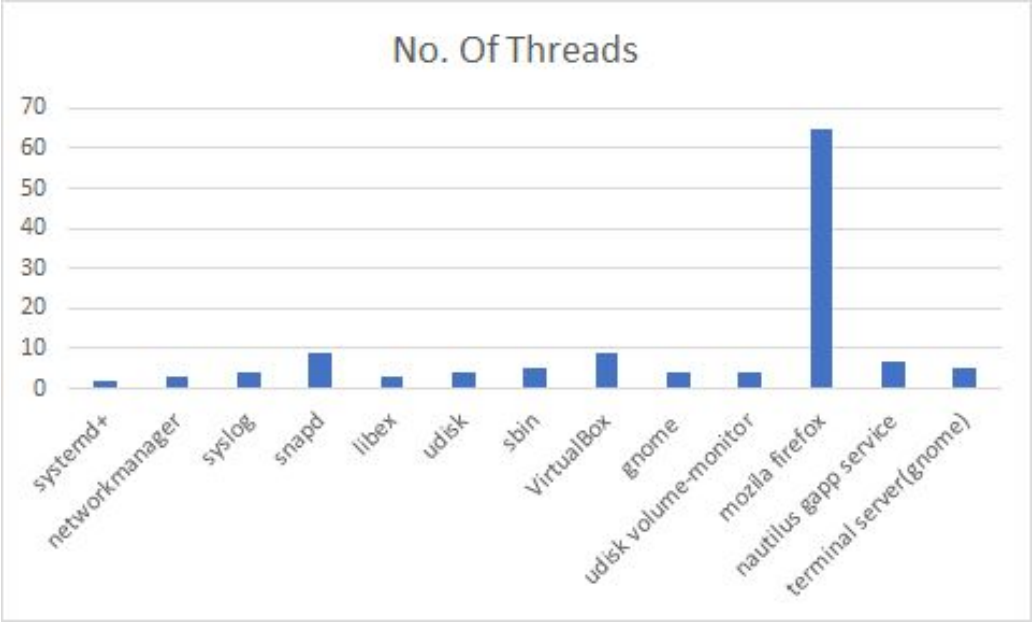
Single threaded processes occupy one line each in the output. Multi-threaded processes occupy as many lines in the output as the number of threads in the relevant process.

This a observation we derived based on the output file attached in slide 41.

NUMBER OF THREADS USED BY HEAVY WEIGHT PROCESS

PESU CCBD

The NLP column of the text file gave no. of light weight threads(kernel level) attached to a process. A short summary of which is represent as a graph below.



SYNOPSIS OF OUR RESEARCH

PESU CCBD

- From our research on the topic we learnt following things:
- perf vs strace
  - perf is better than strace as :
  - It can tell you about both user programs and programs in kernel.
  - It gives statistics of system call in more explicit manner i.e more organised.
  - “perf trace -s” was one way solution to most of our observations.
- Some popular system calls for different scenarios of problem statement were
  - For firefox browser popular system calls were recvmsg,poll,read,write,futex.
  - For word/excel browser popular system calls were recvmsg,poll,read,write.
  - For redis_benchmark browser popular system calls were openat,epoll_wait,read,dose.
- For study of Threads for each process-
  - Each process has unique pid,ppid and LWP’s.The no. of threads attached to the process is proportional to weight of process.
  - We can know multiprogramming of process if same process occurs more than one time.

REFERENCES

PESU CCBD

1. How fast is Redis? – Redis,
2. Redis Tutorial
3. 4 commands to check thread count per process (threads vs processes) in Linux
4. Linux man pages
5. Threads for Processes

PES University

Cloud Computing and Big Data

THANK YOU!