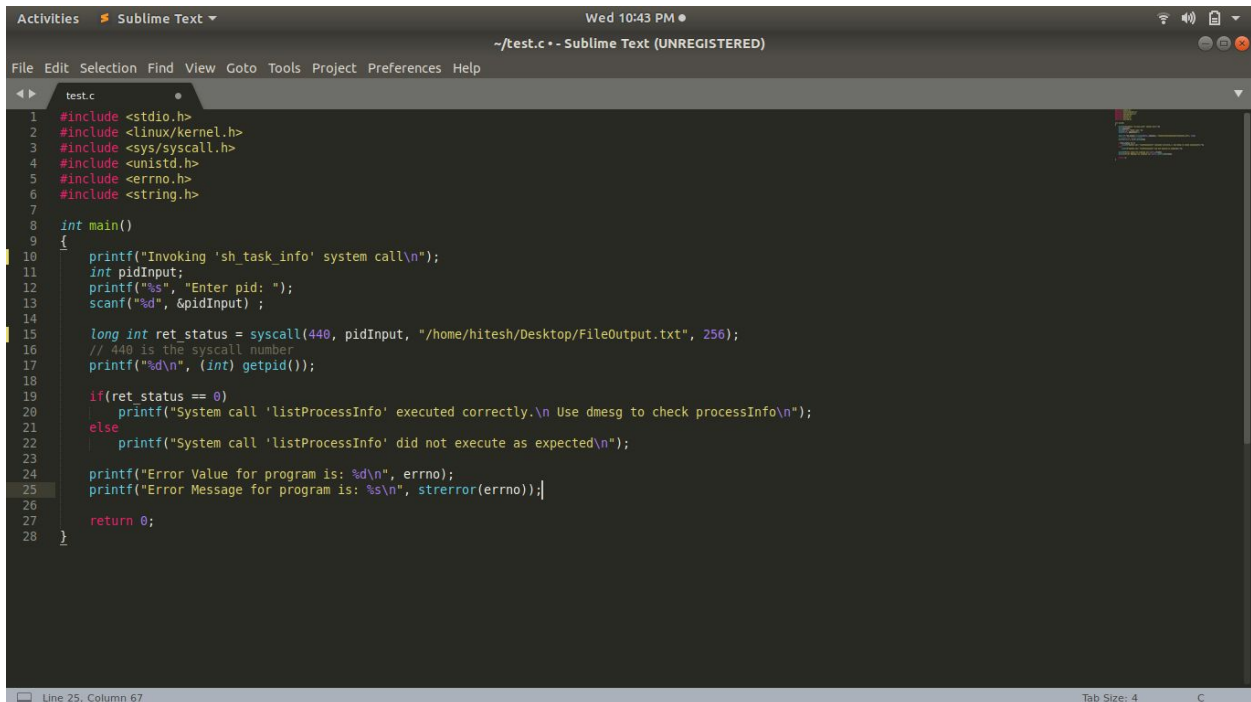


Test.c file

- First, We have taken a PID as input from the user (in the form of int) and output it on the console.
- Then we have called the syscall and saved its output.
- Then we have printed the PID of the entered process.
- Then we have printed messages on the console, based on whether the execution was a success or a failure.
- Then, we have printed the error message and number on the console.



```
1 #include <stdio.h>
2 #include <linux/kernel.h>
3 #include <sys/syscall.h>
4 #include <unistd.h>
5 #include <errno.h>
6 #include <string.h>
7
8 int main()
9 {
10     printf("Invoking 'sh_task_info' system call\n");
11     int pidInput;
12     printf("%s", "Enter pid: ");
13     scanf("%d", &pidInput);
14
15     long int ret_status = syscall(440, pidInput, "/home/hitesh/Desktop/FileOutput.txt", 256);
16     // 440 is the syscall number
17     printf("%d\n", (int) getpid());
18
19     if(ret_status == 0)
20         printf("System call 'listProcessInfo' executed correctly.\n Use dmesg to check processInfo\n");
21     else
22         printf("System call 'listProcessInfo' did not execute as expected\n");
23
24     printf("Error Value for program is: %d\n", errno);
25     printf("Error Message for program is: %s\n", strerror(errno));
26
27     return 0;
28 }
```

System Call

- **Arguments:** (440, pidInput, "/home/hitesh/Desktop/FileOutput.txt", 100)
- The first argument is the system call number. The second argument is the inputted PID. The third argument is the address of the text file where the output will be saved. The fourth argument is the length limit (256) of the address string.
- **Code:**
- First, we have defined a task_struct and found the task_struct corresponding to the given pid.
- Then, we printed the details of the on the kernel log if pid exists.
- Then, we have copied the given address in a buffer variable (char buff[]).
- Then, we declared variables related to the file opening and closing and initialized them.
- Then, we opened the file and wrote the details of the process in it.
- Then, we closed the file.

```
2692 struct task_struct *proc;
2693 $ 32.totalswap = s.totalswap;
2694 $ 32.freeswap = s.freeswap;
2695 $ 32.procs = s.procs;
2696 $ 32.totalhigh = s.totalhigh;
2697 $ 32.freehigh = s.freehigh;
2698 $ 32.mem unit = s.mem unit;
2699 if (copy_to_user(info, &s_32, sizeof(s_32)))
2700     return -EFAULT;
2701 return 0;
2702 }
2703 #endif /* CONFIG_COMPAT */
2704
2705 SYSCALL_DEFINE3(sh_task_info, int, PID, char __user*, src, int, len)
2706 {
2707     struct task_struct *proce;
2708     proce = find_task_by_vpid(PID);
2709
2710     printk("\nPassed PID to the syscall: %d\n", PID);
2711
2712     if (proce == NULL) {
2713         printk("Can't find the process with given pid");
2714     }
2715     else {
2716         printk("Process: %s\n", proce->comm);
2717         printk("PID Number: %ld\n", (long)task_pid_nr(proce));
2718         printk("Process State: %ld\n", (long)proce->state);
2719         printk("Priority: %ld\n", (long)proce->prio);
2720         // printk("RT Priority: %ld\n", (long)proce->rt_priority);
2721         // printk("Static Priority: %ld\n", (long)proce->static_prio);
2722         // printk("Normal Priority: %ld\n", (long)proce->normal_prio);
2723     }
2724
2725     char buff[256];
2726     unsigned long lenleft = len;
2727     unsigned long chunklen = sizeof(buff);
2728     while(lenleft > 0) {
2729         ...
2730     }
2731 }
```

Inputs

- The only input that is to be given is a valid pid from the console.

```
hitesh@hiteshGarg:~$ gcc test.c && ./a.out
Invoking 'sh_task_info' system call
Enter pid: 1590
System call 'sh_task_info' executed correctly.
Use dmesg to check sh_task_info
Error Value for program is: 0
Error Message for program is: Success
hitesh@hiteshGarg:~$
```

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
1586	hitesh	20	0	3725M	187M	111M	S	25.7	4.8	0:19.83	/usr/bin/gnome-sh
1454	hitesh	20	0	555M	85124	72176	S	4.0	2.2	0:06.27	/usr/lib/xorg/Xor
1999	hitesh	20	0	7548	3944	3064	R	2.0	0.1	0:02.71	/snap/htop/1332/u
1942	hitesh	20	0	708M	36456	27180	S	1.3	0.9	0:02.50	/usr/lib/gnome-te
1459	hitesh	20	0	555M	85124	72176	S	1.3	2.2	0:00.78	/usr/lib/xorg/Xor
1626	hitesh	20	0	353M	7664	6260	S	0.7	0.2	0:00.83	ibus-daemon --xin
1628	hitesh	20	0	353M	7664	6260	S	0.7	0.2	0:00.49	ibus-daemon --xin
1590	hitesh	20	0	3725M	187M	111M	S	0.7	4.8	0:00.32	/usr/bin/gnome-sh
1928	hitesh	20	0	1365M	345M	126M	S	0.7	8.9	0:17.35	/snap/sublime-tex
1617	hitesh	20	0	3725M	187M	111M	S	0.0	4.8	0:00.04	/usr/bin/gnome-sh
951	root	20	0	4556	740	676	S	0.0	0.0	0:00.05	/usr/sbin/acpid
1838	hitesh	20	0	200M	6632	6016	S	0.0	0.2	0:00.21	/usr/lib/ibus/ibu
1840	hitesh	20	0	200M	6632	6016	S	0.0	0.2	0:00.15	/usr/lib/ibus/ibu
1944	hitesh	20	0	708M	36456	27180	S	0.0	0.9	0:00.11	/usr/lib/gnome-te

Output

- The output should be a message saying “invoking sh_task_info system call”.
- Then, we should print the inputted pid

- Then, a message showing the success or failure of the system call.
- Then, the value of the error number and its corresponding message.
- Watch the text file output at the provided address in the test.c file.
- You can also view the kernel log by using dmesg in the terminal.

The terminal window shows the following commands and output:

```
hitesh@hiteshGarg:~$ gcc test.c && ./a.out
Invoking 'sh_task_info' system call
Enter pld: 1590
1590
System call 'sh_task_info' executed correctly.
Use dmesg to check sh_task_info
Error Value for program is: 0
Error Message for program is: Success
hitesh@hiteshGarg:~$ dmesg | tail
[ 251.332525] Passed PID to the syscall: 1590
[ 251.332534] Process: gdbus
[ 251.332537] PID_Number: 1590
[ 251.332539] Process State: 1
[ 251.332542] Priority: 120
[ 251.332545] FileName is : /home/hitesh/Desktop/FileOutput.txt
[ 251.332547] My module is loaded
[ 251.332550] Opening file for writing
hitesh@hiteshGarg:~$
```

Overlaid on the terminal is a window showing system statistics and a process list:

```
Tasks: 161, 509 thr; 4 running
Load average: 2.98 3.12 1.64
Uptime: 00:09:03
Mem[ 2.55G/3.77G]
Swap[ 3.27M/2.79G]
```

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
2254	hitesh	20	0	19.0G	358M	112M	S	74.4	9.3	3:29.95	/opt/google/chrom
1609	hitesh	9	-11	2334M	20140	15584	S	11.3	0.5	0:26.56	/usr/bin/pulseaud
2570	hitesh	20	0	19.0G	358M	112M	R	11.3	9.3	0:20.82	/opt/google/chrom
2509	hitesh	20	0	19.0G	358M	112M	R	10.6	9.3	0:17.21	/opt/google/chrom
2557	hitesh	20	0	19.0G	358M	112M	S	9.3	9.3	0:19.02	/opt/google/chrom
2312	hitesh	20	0	19.0G	358M	112M	R	8.0	9.3	0:16.79	/opt/google/chrom
2323	hitesh	20	0	19.0G	358M	112M	S	5.3	9.3	0:17.11	/opt/google/chrom
2522	hitesh	20	0	1019M	63180	51252	S	5.3	1.6	0:11.18	/opt/google/chrom
2545	hitesh	20	0	19.0G	358M	112M	S	4.7	9.3	0:08.76	/opt/google/chrom
1586	hitesh	20	0	3727M	197M	114M	S	4.0	5.1	0:49.32	/usr/bin/gnome-sh
2186	hitesh	20	0	1092M	314M	133M	S	4.0	8.1	0:49.70	/usr/bin/google-c
2593	hitesh	20	0	19.0G	358M	112M	R	4.0	9.3	0:08.27	/opt/google/chrom
2228	hitesh	20	0	542M	87432	64312	S	3.3	2.2	0:09.53	/opt/google/chrom
1454	hitesh	20	0	576M	104M	92196	S	2.7	2.7	0:27.95	/usr/lib/xorg/Xor

The text editor window, titled 'FileOutput.txt', contains the following text:

```
Process: gdbus
PID_Number: 1590
Process State: 1
Priority: 120
Normal_Priority: 120
: 120
```

Error-values

- Value 0: Success

- Value 1: Operation not permitted
- Value 2: No such file or directory
- Value 3: No such process
- Value 4: Interrupted system call
- Value 5: I/O error
- Value 6: No such device or address
- Value 7: Arguments list too long
- Value 8: Exec format error
- Value 9: Bad file number
- Value 10: No child processes
- Value 11: Try again
- Value 12: Out of memory
- Value 13: Permission denied

```
hitesh@hiteshGarg:~$ gcc test.c && ./a.out
Invoking 'sh_task_info' system call
Enter pid: 1590
1590
System call 'sh_task_info' executed correctly.
Use dmesg to check sh_task_info
Error Value for program is: 0
Error Message for program is: Success
hitesh@hiteshGarg:~$ dmesg | tail
```

Implementation

- First, update your kernel to the given version using an online tutorial.
- Then, write your c code of the system call in /kernel/sys.c file of the Linux directory.
- Then, add your system call entry in \arch\x86\entry\syscalls\syscall_64.tbl file of the Linux directory.

/* .entry_64.o.cmd	357	433	common	fspick	sys_fspick
/* .entry_64_compat	358	434	common	pidfd_open	sys_pidfd_open
/* .modules.order.crr	359	435	common	clone3	sys_clone3
/* .syscall_32.o.cmd	360	436	common	close_range	sys_close_range
/* .syscall_64.o.cmd	361	437	common	openat2	sys_openat2
/* .syscall_x32.o.cmc	362	438	common	pidfd_getfd	sys_pidfd_getfd
/* .thunk_64.o.cmd	363	439	common	faccessat2	sys_faccessat2
/* calling.h	364	440	common	sh_task_info	sys_sh_task_info
/* common.c	365				
/* entry_32.S	366				
/* entry_64.S	367				
	368				
	369				
	370				
	371				