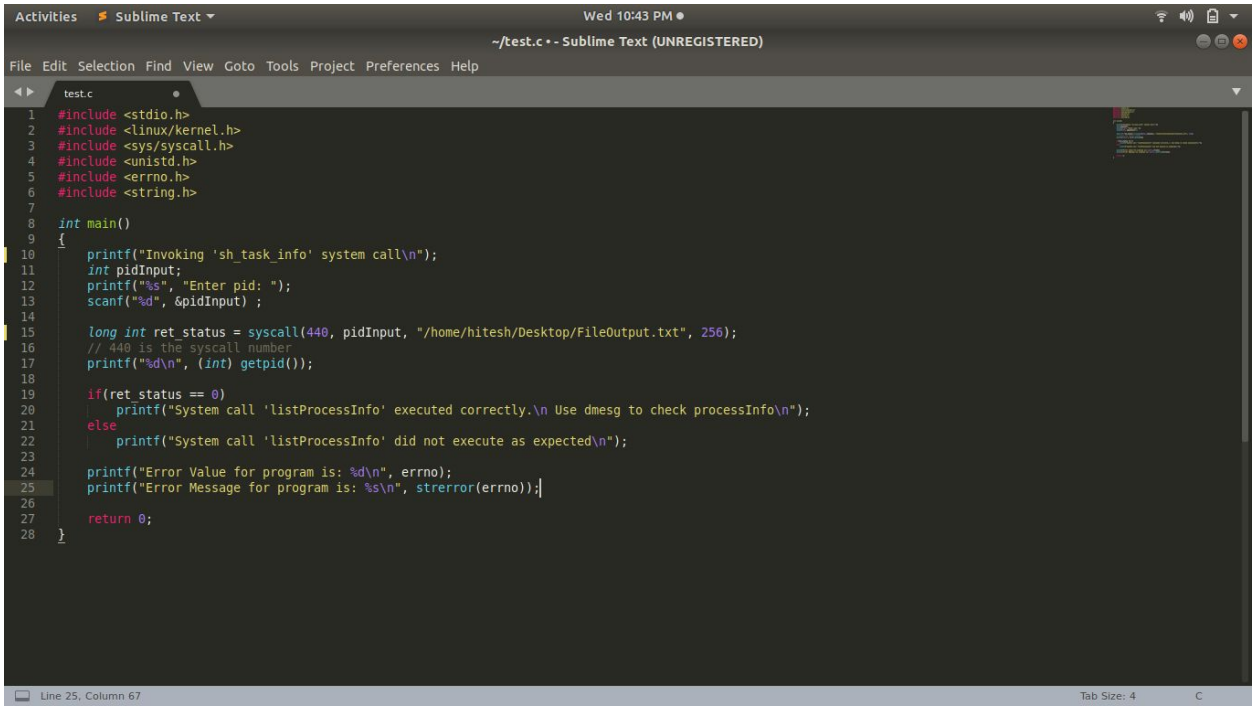
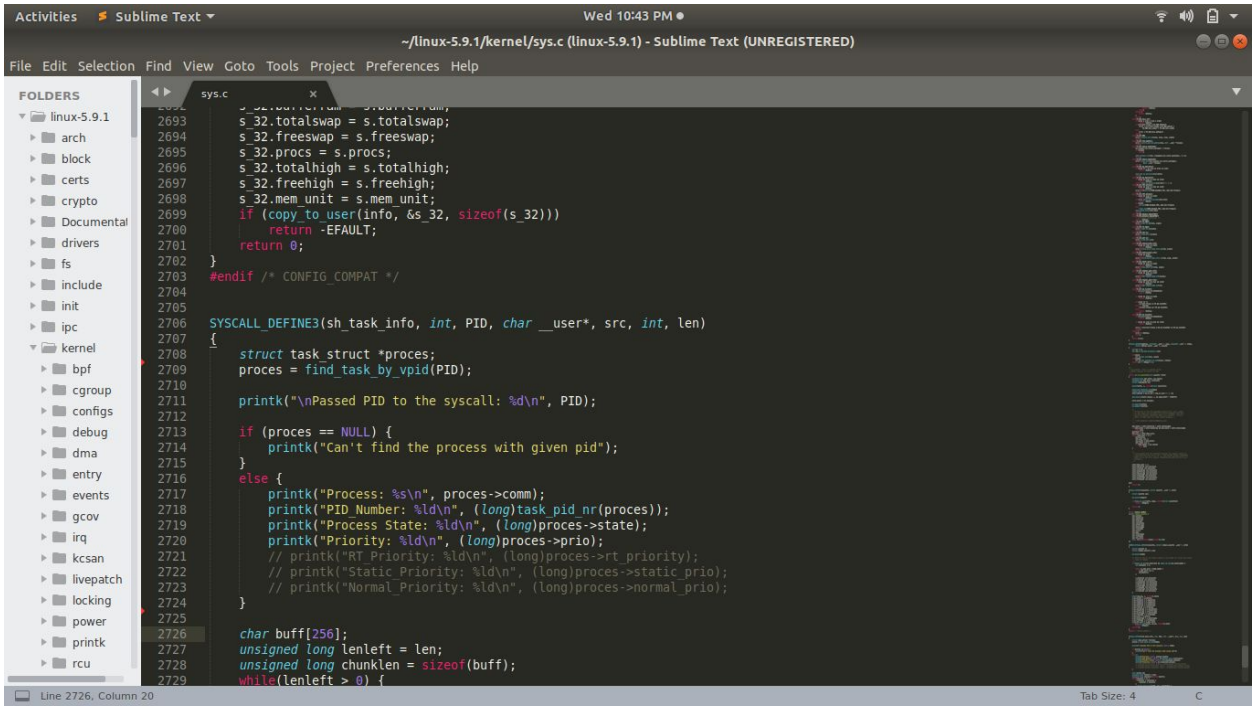
Test.c file

- 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.



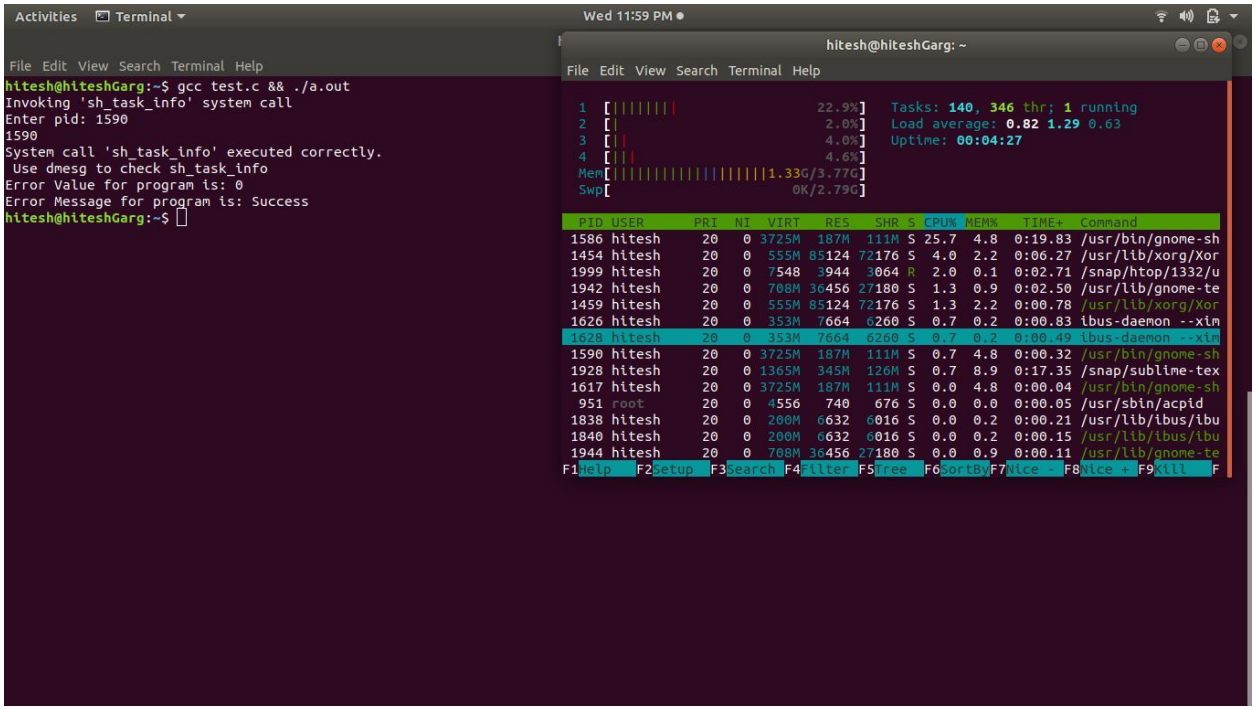
System Call

- **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.



Inputs

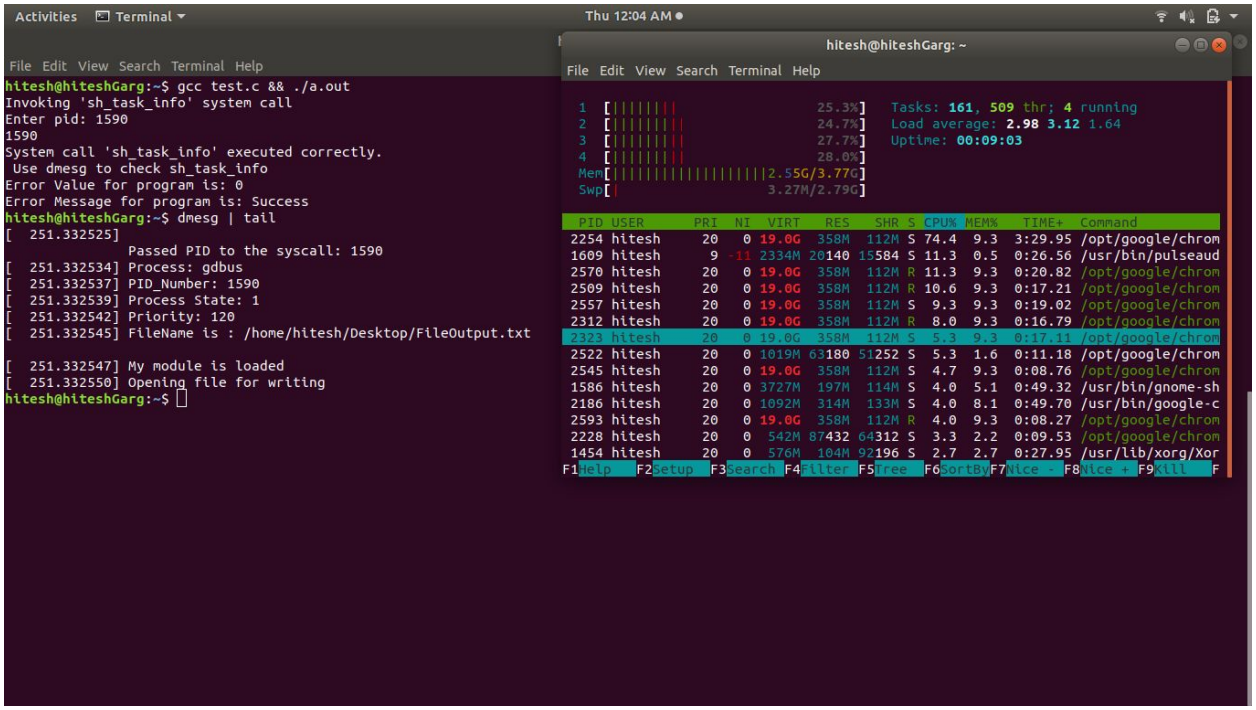
- The only input that is to be given is a valid pid from the console.

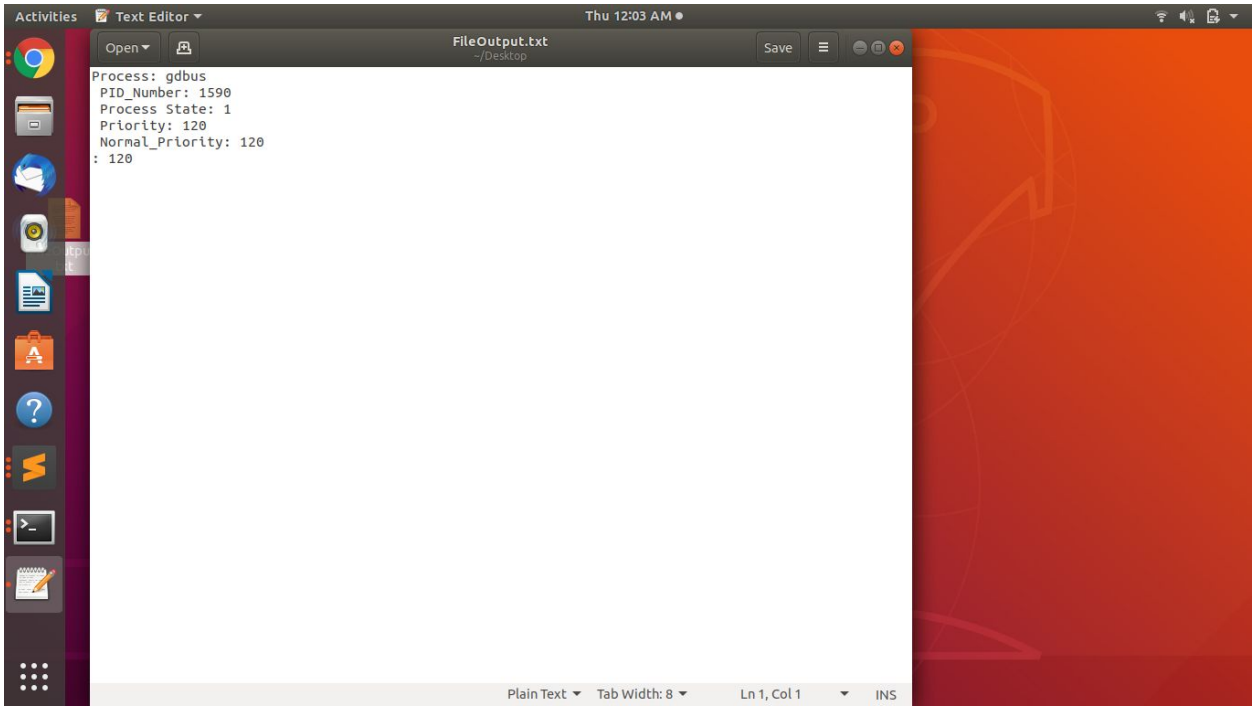


Output

- 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.





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
/* .entry_64_compat
/* .modules.order.cm
/* .syscall_32.o.cmd
/* .syscall_64.o.cmd
/* .syscall_x32.o.cmd
/* .thunk_64.o.cmd
/* calling.h
/* common.c
/* entry_32.S
/* entry_64.S
```

```
357  433 common  fspick          sys_fspick
358  434 common  pidfd_open      sys_pidfd_open
359  435 common  clone3          sys_clone3
360  436 common  close_range     sys_close_range
361  437 common  openat2         sys_openat2
362  438 common  pidfd_getfd     sys_pidfd_getfd
363  439 common  faccessat2      sys_faccessat2
364  440 common  sh_task_info        sys_sh_task_info
365
366  #
367  # x32-specific system call numbers start at 512 to avoid cach
368  # for native 64-bit operation. The __x32_compat_sys stubs are
369  # on-the-fly for compat_sys_*() compatibility system calls if
370  # is defined.
371  #
```