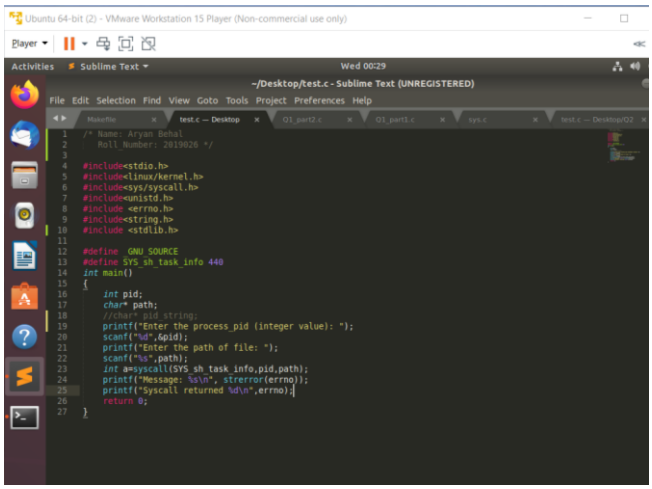
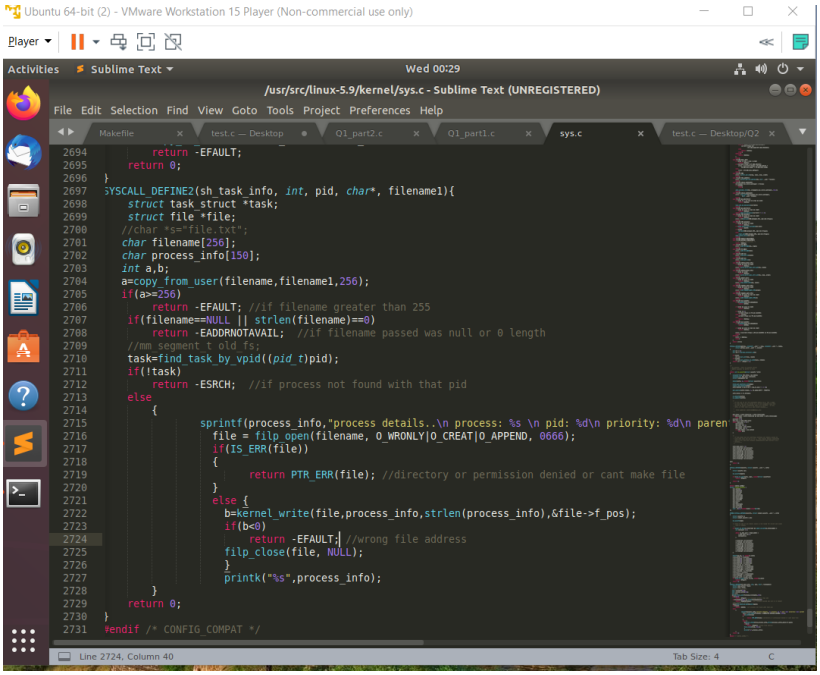
Aryan Behal <u>OS Assignment-2 Q2</u> 2019026

Kernel Version used: v5.9

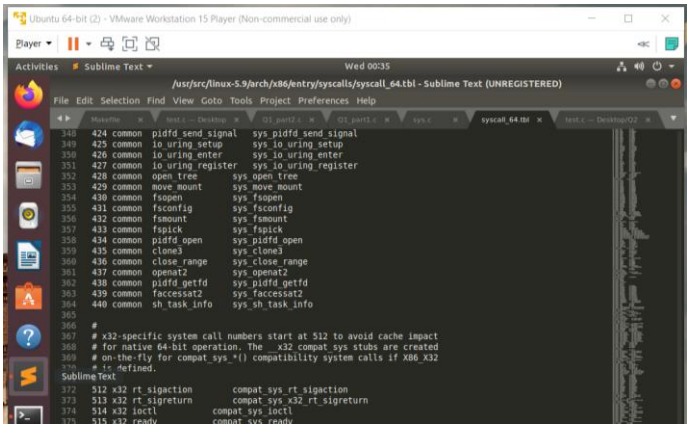


My test.c code



My syscall definition in kernel/sys.c by name SYSCALL_DEFINE2(sh_task_info, int, pid, char*, filename1)

- I have implemented my code for syscall in kernel/sys.c of linux-5.9 and have not made a special directory for a separate C code and Makefile for it.
- My syscall number is 440 for the new syscall I have added in syscall_64.tbl



My syscall_64.tbl

➔ <u>Code description</u>

- For the **test.c** code, I am asking the user to enter pid and file name (as absolute path or relative path) using scanf. Then I am calling the syscall with the parameters and printing success or failure using errno.
- I used functions:
  - For calling syscall – syscall (SYS_syscall_name, argument1, argument2,…)
    - I had only 2 arguments; pid and path so my sycall looked like:
      - syscall (SYS_sh_task_info, pid, path)
  - for handling errors- printf ("%s", strerror(errno)) where errno stores integer corresponding to each error handled.
- For the SYSCALL_DEFINE, I am simply using the 2 parameters passed to search for the process by pid and opening the file given by filename1 and then writing to that file and printing on kconsole using printk and returning from syscall.
- I used the following functions:
  - For copying filename1(user space pointer) to (filename)kernel space char array- **copy_from_user** (kernel_char_array, parameter passed, bytes_copied).
  - For searching process- **find_task_by_vpid** (pid_t pid)
    - It returns pointer to task_struct of process with given pid if present
  - For making a single string to write to file- **sprintf** (char* buf, "string")
  - For opening file- **filp_open** (path, modes, permissions)
    - Returns pointer to file struct, act as file descriptor.
    - I have opened file in write only, append (if existing) and create (if not existing) mode.
    - I have opened in 0666 permission. All can read and write.
  - For writing in file: **kernel_write (**struct file *descriptor, char* buf, buf_size, loff_t *pos (long offset-seek offset for where to read/write))
  - To kconsole: **printk** ("string")
- I have used the following structs:
  - Struct file
  - Struct task_struct

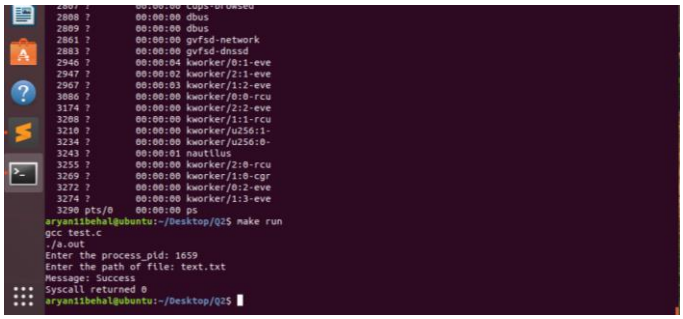
➔ <u>User Input</u>

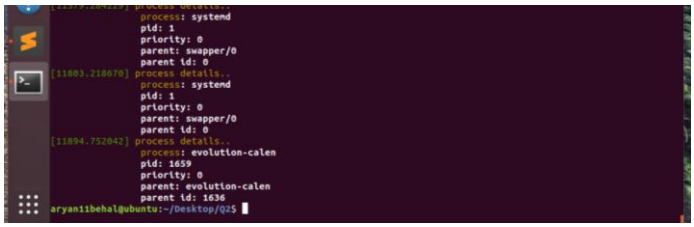
- For pid: any valid integer value (range of int data type in C)
    - Program will terminate if any other data type(like string) entered. This error not handled.

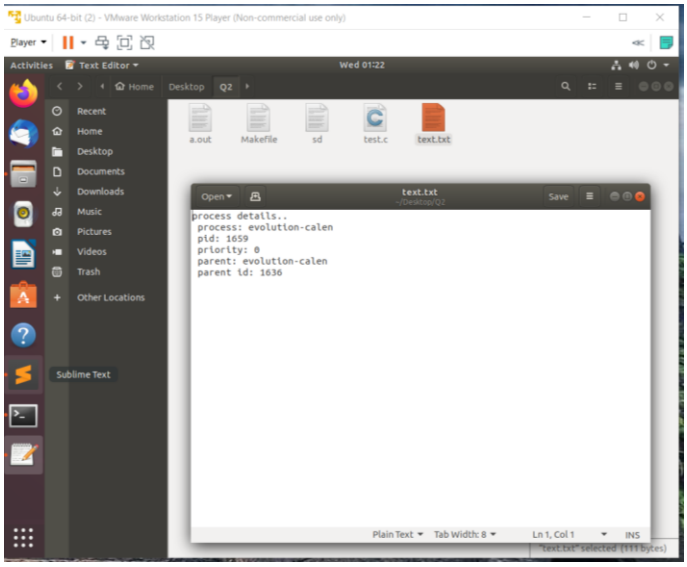
- For file name/path: any string (File path passed cannot be greater than 255 char)

➔ <u>Output</u>

- If user input is a valid input and process is found and file can be created at specified address, then we get a success message and errno gives 0 (success).
- My test.c file prints a success on getting success and an appropriate message when a failure is encountered. It also prints the error number which syscall returned.
- I have only used 5 fields of process; process name, pid, priority, parent name, parent pid.
- On writing dmesg to console, whatever printk printed can be seen at last. If some error is encountered nothing will be printed on kernel log.
- Also, a file gets created at specified path or relative path with process fields printed.

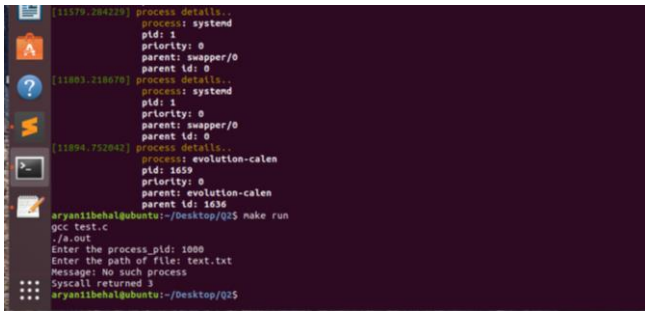






In case, there is an error like process is not found with the pid or given path a directory, a relevant message is printed indicating so by

strerror ().



- ➔ <u>Error handling –</u>
  - I used Linux system errors in <errno.h>.
  - If process not found with given pid (a valid integer value within range):
    - Return ESRCH: errno = 3
  - if at given address, file creation permission not given (using IS_ERR() and PTR_ERR())
    - Return EACCES: errno = 13
  - If given address is of already existing directory (using IS_ERR() and PTR_ERR())
    - Return EISDIR: errno = 21
  - If file address given > 255 words or no address passed
    - Return EFAULT: errno = 14 (Bad Address)
  - If file address of 0 length
    - Return EADDRNOTAVAIL: errno = 99 (can't assign this address)
  - if file too big
    - Return EFBIG: errno = 27
  - I have not dealt with error when user enters wrong pid as we were asked to deal with only 2 errors and it is user's responsibility to pass an integer and not a string to pid.

- ➔ <u>Sources:</u>
  - https://medium.com/anubhav-shrimal/adding-a-hello-world-system-call-to-linux-kernel-dad32875872
  - https://brennan.io/2016/11/14/kernel-dev-ep3/
  - https://www-numi.fnal.gov/offline_software/srt_public_context/WebDocs/Errors/unix_system_errors.html