WRITE UP FOR QUES 2

SUYASHI SINGHAL
2019478
CSE

Description of my code and implementation of my system call

1) Description of my system call - logical and implementation details

- The name of my system call is “sh_task_info”. It takes two parameters as input. One is the process pid(integer) and the other is the name of the file to which the details of the process are written.
- I am using the function copy_from_user to copy the file name character pointer entered in user space into a character array in kernel space. The function access_ok() checks if the user pointer is accessible and valid. It returns -EFAULT (Bad address error) in case the user pointer is not accessible.
- I use the function find_get_pid() which takes the pid as parameter to get the struct pid. In the pid_task() function , I give the struct pid(obtained from find_get_pid()) and PIDTYPE_PID as the parameters. The function returns the task_struct structure which is then used to display the details of the process.
- In case the find_get_pid() returns NULL, it means the process does not exist. In this case, I print a kernel alert that the process does not exist in kernel log using printk(). I try to open/create the file given by the user and if I am successful at opening/creating the file given by user then I write an error message on the file too. I then return the -ESRCH which means that the process does not exist. I print the errors on the console using strerror(errno) in the test.c file since the errno is set when the error is returned by the system call.
- In case the process is a valid one i.e find_get_pid() does not return null, I print the details of the process on the kernel log. The details I print of the process are the process name, Id, Runtime(utime), Parent process name, Static priority, normal priority, RT_priority, Priority and Process state. I also print the user inputs i.e id and file name on the kernel log.
- After printing to the kernel log, I open/create the file with the filp_open() function. It takes the first parameter as file name/path. The second parameter are the flags. The flags used by me are O_WRONLY, O_CREAT and O_TRUNC. O_WRONLY writes to an already existing file, O_CREAT creates a new file if one isn’t present and O_TRUNC truncates the length of the file to 0 before overwriting.
- If we are successfully able to open the file, then I use kernel_write() to write into the file. Here the parameters are the file descriptor pointing to a struct file (f1 in

this case), the string to be written into the file (print), size of the string to be written(strlen(print)) and the offset(0 in this case such that it always writes from the beginning of the file). I then close the file using filp_close() which takes f1 and NULL as parameters.

- If we are not able to open the file then IS_ERR(f1) evaluates to true and we return PTR_ERR(f1) which sets the errno in test.c and hence allows us to print the error on the terminal itself.

2) ***Changes done in the kernel code to write the system call***

a) I have made a directory task_info in the linux directory which stores my system call and makefile. I added two files into the directory. The first is my system call i.e sh_task_info.c and the other is the Makefile. The makefile contains a single line i.e -
obj-y := sh_task_info.o

b) In the Makefile inside the linux directory I have made a change in one of the lines as follows -
core-y += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ task_info/

I added the task_info/ into the above line so that the files in the directory task_info can also be compiled during compilation.

c) *In the directory arch/x86/syscalls/ I have edited the file syscalls_64.tbl and added the following line at the end -*
548 64 task_info sys_sh_task_info

d) *In the directory include/linux/ I have edited the file syscalls.h and added the following line at the end -*
asmlinkage long sys_sh_task_info(int processid, char __user *fname);

e) *Finally after making all the changes I compiled the kernel, installed it and rebooted the virtual box to add my system call.*
I used the following commands -
sudo make -j 2
sudo make modeules_install install
sudo make install
shutdown -r now

Testing the system call

Description of my test.c

- *My test.c prompts the user to enter the pid of the process and the name of the file in which the details are to be written. The file is always made in the current working directory of the user. This is because I use getcwd() to get the current working directory and concatenate the name of the file to it.*
- *If the pid is negative, it prints error that a process cannot have a negative pid. The syscall is called and it returns -1. Then I print the error using strerror() which takes in the errno as the parameter. When the system call is called, it sets the errno in case of any error through it's return value.*
- *If the pid is non negative, the syscall is called normally and we get the output depending on whether the entered process exists or not and whether the entered file name is valid or not. If no error is there then the functions returns 0, else the function returns -1 and the corresponding error is printed on the terminal as well as the kernel log.*

Input the user should give

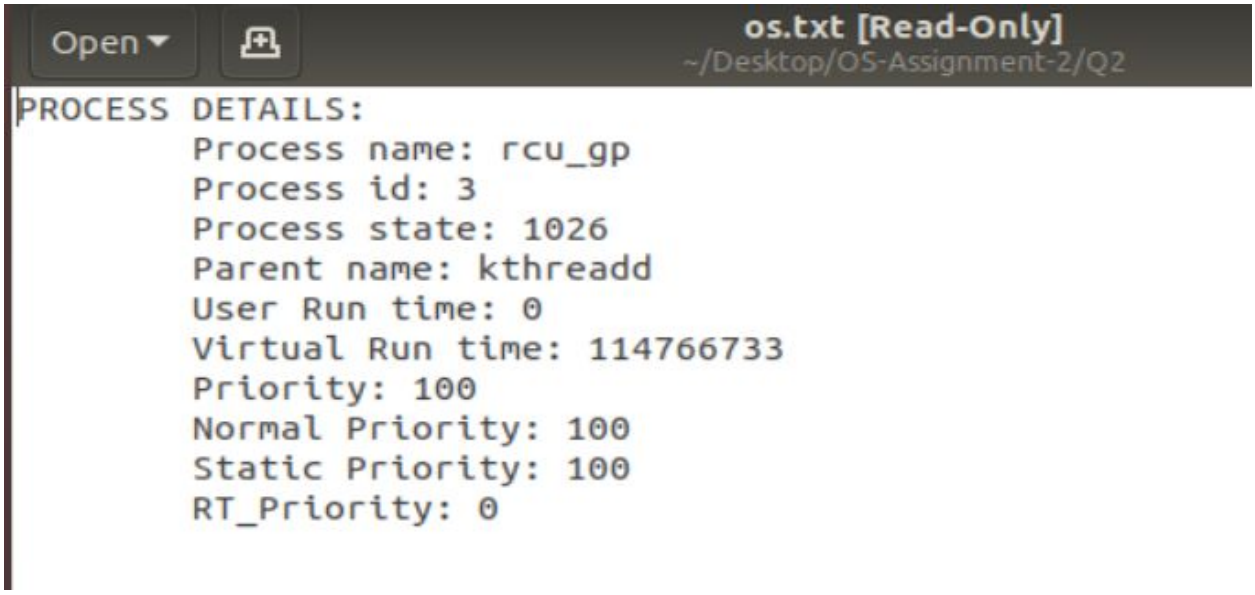
- *The user should give a valid pid and file name.*
- *In case the user gives an invalid pid(integer), then the error has been handled and we display "No such process" error using strerror().*
- *The user should not enter a string as input as pid is never a string. If he does that the test.c won't take any further input and the system call will return -1.*
- *In case the user enters a wrong file name e.g name of a directory or name of an invalid directory, then the corresponding errors will be displayed on the terminal.*

- *Hence the user has to enter an integer pid and file name. The rest is handled by the system calls which returns 0 if the values entered are valid. Else, it returns -1 and sets the errno for the corresponding error which we then display on the terminal.*

Error handling examples and expected output

a) ***When input without any errors are given*** *- In this case system call simply writes the details of the process in the kernel log as well as the input file(which is created in the current working directory if it does not already exist) Here the "os.txt" file did not exist earlier. It is created by the kernel when the system call is executed.*

```
root@suyashi-VirtualBox:~/Desktop/OS-Assignment-2/Q2# make run
gcc test.c
./a.out
Enter pid: 3
Enter file name : os.txt
System call sys_sh_task_info returned 0
root@suyashi-VirtualBox:~/Desktop/OS-Assignment-2/Q2# dmesg
[ 1050.799524] Process pid: 3
[ 1050.799525] File name: /home/suyashi912/Desktop/OS-Assignment-2/Q2/os.txt
[ 1050.799532] PROCESS DETAILS:
               Process name: rcu_gp
               Process id: 3
               Process state: 1026
               Parent name: kthreadd
               User Run time: 0
               Virtual Run time: 114766733
               Priority: 100
               Normal Priority: 100
               Static Priority: 100
               RT_Priority: 0
```



b) **When negative pid is given as the input** - In this case an error message is displayed as given below. Here the error message is displayed in kernel log, terminal as well as the file. Since the file “os.txt” had already been created earlier, it’s contents are overwritten.

```
root@suyashi-VirtualBox:~/Desktop/OS-Assignment-2/Q2# make run
gcc test.c
./a.out
Enter pid: -15
Enter file name : os.txt
Process pid cannot be negative
System call sys_sh_task_info returned -1
Error : No such process
root@suyashi-VirtualBox:~/Desktop/OS-Assignment-2/Q2# dmesg
[ 1261.109724] The process does not exist : -15

root@suyashi-VirtualBox:~/Desktop/OS-Assignment-2/Q2#
```

Open ▾ os.txt [Read-Only]
~/Desktop/OS-Assignment-2/Q2

```
This is an incorrect process pid :-15
The process does not exist.
```

In this case our system call returns -ESRCH(error number 3) which stands for "No such process"

c) **When a non negative pid is given but the process does not exist** - Error is printed on the terminal, kernel log as well as on the file(if the file name is valid).

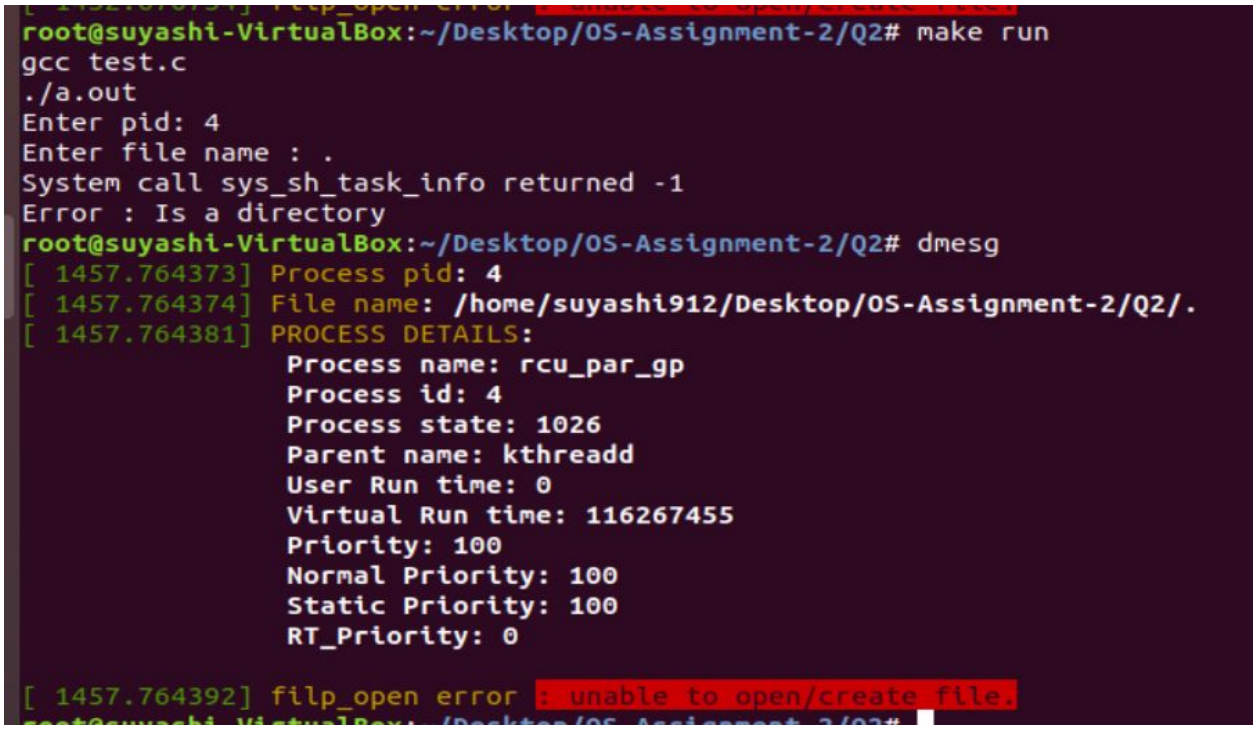
```
root@suyashi-VirtualBox:~/Desktop/OS-Assignment-2/Q2# make run
gcc test.c
./a.out
Enter pid: 242425
Enter file name : os.txt
System call sys_sh_task_info returned -1
Error : No such process
root@suyashi-VirtualBox:~/Desktop/OS-Assignment-2/Q2# dmesg
[ 1364.566467] The process does not exist : 242425
```

Open ▾ os.txt [Read-Only]
~/Desktop/OS-Assignment-2/Q2

```
This is an incorrect process pid :242425
The process does not exist.
```

In this case our system call returns -ESRCH(error number 3) which stands for "No such process"

d) **When a directory name is specified instead of a file name** - Error is displayed on the terminal and kernel log



In this case our system call returns -EISDIR (error number 21) which stands for "Is a directory."

e) **When a wrong file name is given as input by the user** - e.g "/file/hello" - error will be displayed on terminal and kernel log

```
root@suyashi-VirtualBox:~/Desktop/OS-Assignment-2/Q2# make run
gcc test.c
./a.out
Enter pid: 6
Enter file name : /file/hello
System call sys_sh_task_info returned -1
Error : No such file or directory
root@suyashi-VirtualBox:~/Desktop/OS-Assignment-2/Q2# dmesg
[ 1576.640851] Process pid: 6
[ 1576.640852] File name: /home/suyashi912/Desktop/OS-Assignment-2/Q2//file/hel
lo
[ 1576.640860] PROCESS DETAILS:
                Process name: kworker/0:0H
                Process id: 6
                Process state: 1026
                Parent name: kthreadd
                User Run time: 0
                Virtual Run time: 1228697439
                Priority: 100
                Normal Priority: 100
                Static Priority: 100
                RT_Priority: 0

[ 1576.640873] filp_open error : unable to open/create file.
```

In this case our system call returns -ENOENT(error number 2) which stands for "No such file or directory."

Error handling in code

1) **Error handling in copy_from-user()**
   I have handled the possible errors in copy_from_user(). Using access_ok() function I check if the pointer from the user space is valid. If not, it returns -EFAULT i.e Bad address error. This will be printed on the terminal if such an error occurs. Also if copy_from_user() fails, then -EFAULT is returned.

```
    temp[0] = '\0';
    if(access_ok( fname, strlen(fname)+2) == 0)
        return -EFAULT;

    copy = copy_from_user(u, fname, sizeof(u));
    u[strlen(u)]='\0';

    if(copy!=0)
    {
        printk(KERN_ALERT "copy_from_user error : unable to copy from user space.");
        return -EFAULT;
    }
```

In this case our system call returns -EFAULT(error number 14) which stands for "Bad address".

2) ***Error handling for wrong pid number***

```
if(p_id == NULL)
{
    printk(KERN_ALERT "The process does not exist : %d",processid);
    strncpy(er, "This is an incorrect process pid :", sizeof(er)-1);
    sprintf(er1, "%d", processid);
    strncat(er, er1, strlen(er1)+2);
    strncat(er, "\nThe process does not exist.", sizeof(er)-1);
    er[strlen(er)]='\0';
    size = strlen(er);
    printk("\n");
    f1 = filp_open(u, O_WRONLY|O_CREAT|O_TRUNC, 0644);
    if(IS_ERR(f1))
    {
        printk(KERN_ALERT "filp_open error : unable to open/create file");
    }
    else
    {
        write = kernel_write(f1, (void *) er, size, 0);
        filp_close(f1, NULL);
    }

    return -ESRCH;
}
```

3) ***Error handling for wrong file name***

```
if(IS_ERR(f1))
{
    printk(KERN_ALERT "filp_open error : unable to open/create file.\n");
    return PTR_ERR(f1);
}
```

Generation of diff

- *I have generated the diff of my linux directory with the original linux directory.*
- *I have two directories in /usr/src - **mylinux** which stores the modified code of linux-5.9.1 in which I have added my system call and **origLinux** which stores the original code of linux-5.9.1*
- *Before generating diff I used the command **sudo make distclean** in mylinux/linux-5.9.1/ directory.*
- *After that I generated the diff using the command -*
  diff -Naur origLinux/linux-5.9.1 mylinux/linux-5.9.1/ > diff.txt