

## Write up for Question 2

Steps followed for compiling the kernel and adding system call:

1. Download kernel source code using wget command
2. Extract the tar.gz file
3. Goto to directory kernel/ in the extracted folder
4. Edit file sys.c, ex : sudo nano sys.c
5. Add you syscall code in the file sys.c
6. Go to directory arch/x86/entry/syscalls/ in extracted folder
7. Add your system call to file syscall\_64.tbl  
Ex: 440    common    sample    sys\_sample
8. Edit config file so that only required things are compiled
9. Compile the kernel with sudo make -j(\$number of processors to use)
10. Install modules using : sudo make modules\_install install
11. Reboot system

Error handling done in system call :

1. Handling the case when creating task\_struct fails

```
if (task == NULL) {  
    printk(KERN_ALERT "Cannot find a process with pid %d\n",  
pid);  
    return -ESRCH;  
}
```

where task is the name of task\_struct pointer.

2. Handling case where strncpy\_from\_user() fails

```
if (val < 0 || val == n)  
    return -EFAULT;
```

where val is the value returned by `strncpy_from_user()`

3. error handling unable to open file i.e. `filp_open` fails

```
if (IS_ERR(ffile) || ffile == NULL) {
    printk(KERN_ALERT "ERROR: filp_open %ld\n",
PTR_ERR(ffile));
    return PTR_ERR(ffile);
}
```

`IS_ERR` is used to check if there is any error and `PTR_ERR()` is used to identify the error.

4. To be able to debug easily the syscall outputs the return values of every `kernel_write()` to the kernel console which can be viewed using the `dmesg` command.

Error handling doing in `test.c`:

1. Check if the arguments contain at least two inputs.

```
if (argc <= 2) {
    printf("Incorrect Arguments\n");
    return -1;
}
```

2. Identify if that the system call returns any error using `perror`.

```
perror("Console Output");
```

Prints "success" if no error is encountered

Short Description of how system calls is written and functions used (see more in comments) :

In the system call file all the information about the process is obtained using its pid in a *task\_struct* using *pid\_task()* and *find\_get\_pid()*. Then the fileName is first copied to kernel memory space and then its opened using *filp\_open()*.

Now the next step is to write the file from kernel, to achieve this *kernel\_write()* is uses a char \* buffer. To write desired text into a char \* buffer *sprintf* is used, which works as standard *printf* except it outputs the result to a buffer provided as input instead of stdout console.

Example Outputs:

Input format : ./a.out \$pid \$filename

1. Printing info about process with pid 1 to d.txt

```
zyrch@zyrch:~$ ./a.out 1 d.txt
Making system call with 1 d.txt
System call returned 0.
Console Output: Success
zyrch@zyrch:~$ sudo cat d.txt
=====Process Details=====
Pid : 1
Name: systemd
State: 1
Vruntime: 291333715
Time spent in user mode: 211795956
Priority: 120
zyrch@zyrch:~$ dmesg | tail
[11164.348787] writing Vruntime returned 20
[11164.348789] writing Time spent in user mode returned 35
[11164.348791] writing Priority returned 14
[11201.537946] writing Heading returned 34
[11201.537950] writing Pid returned 8
[11201.537952] writing Name returned 14
[11201.537953] writing State returned 9
[11201.537956] writing Vruntime returned 20
[11201.537957] writing Time spent in user mode returned 35
[11201.537959] writing Priority returned 14
zyrch@zyrch:~$
```

2. If a file already exists then it's truncated.

```
zyrch@zyrch:~$ sudo cat d.txt
Some random text
zyrch@zyrch:~$ ./a.out 1 d.txt
Making system call with 1 d.txt
System call returned 0.
Console Output: Success
zyrch@zyrch:~$ sudo cat d.txt
=====Process Details=====
Pid : 1
Name: systemd
State: 1
Vruntime: 292601582
Time spent in user mode: 211795956
Priority: 120
zyrch@zyrch:~$
```

3. If invalid pid is entered error is outputted accordingly

```
zyrch@zyrch:~$ ./a.out 342 d.txt
Making system call with 342 d.txt
System call returned -1.
Console Output: No such process
zyrch@zyrch:~$
```