Assignment 1

Computer Architecture and Operating Systems

Name: Anmol Gupta
Roll Number: 2018329

Extract the kernel source twice to get two copies, and rename one of them. Here we will be making changes to **`linux-3.16-rc-onlychanged`**.

```
anmolgupta@ubuntu:~$
anmolgupta@ubuntu:~$
anmolgupta@ubuntu:~$
anmolgupta@ubuntu:~$ ls
linux-3.16-rc2  linux-3.16-rc2.tar.gz  linux-3.16-rc-onlychanged
anmolgupta@ubuntu:~$ _
```

Create a folder **`taskinfo`**. This is where we will add the code for our system call.

```
linux-3.16-rc2  linux-3.16-rc2.tar.gz  linux-3.16-rc-onlychanged
anmolgupta@ubuntu:~$ cd linux-3.16-rc-onlychanged/
anmolgupta@ubuntu:~/linux-3.16-rc-onlychanged$ ls
arch     crypto         fs       ipc      lib          net              scripts   tools
block    Documentation  hello    Kbuild   MAINTAINERS  README           security  usr
COPYING  drivers        include  Kconfig  Makefile     REPORTING-BUGS   sound     virt
CREDITS  firmware       init     kernel   mm           samples          taskinfo
anmolgupta@ubuntu:~/linux-3.16-rc-onlychanged$ cd taskinfo
anmolgupta@ubuntu:~/linux-3.16-rc-onlychanged/taskinfo$ _
```

Create a file **`taskinfo.h`** and add the prototype of your system call here.

```
anmolgupta@ubuntu:~/linux-3.16-rc-onlychanged/taskinfo$ cat taskinfo.h
asmlinkage long sys_sh_task_info(long pid, char *filename);

anmolgupta@ubuntu:~/linux-3.16-rc-onlychanged/taskinfo$ _
```

Create a file **`list_task_info.c`** and write the system call in it.

```c
#include<linux/kernel.h>
#include<linux/init.h>
#include<linux/sched.h>
#include<linux/syscalls.h>
#include<linux/file.h>
#include<asm/uaccess.h>
#include<uapi/asm-generic/errno-base.h>
#include<linux/fs.h>
#include<linux/fcntl.h>

#include "taskinfo.h"

int write_to_file(struct file *file, char *data) {
        file->f_op->write(file, data, strlen(data), &file->f_pos);
        return 0;
}

asmlinkage long sys_sh_task_info(long pid, char *filename) {

        if(pid <= 0 || pid > 32768)
                return -EINVAL; // invalid argument

        struct task_struct *task;
        struct file *file;

        char data[400], temp[400];

        loff_t pos = 0;
        int fd;

        mm_segment_t old_fs = get_fs();
        set_fs(KERNEL_DS);

        fd = sys_open(filename, O_WRONLY|O_CREAT, 0644);
```

We include the header files necessary in the system call. E.g., sched.h contains details about all the processes in the system. We also include taskinfo.h which contains our system call prototype.

We use the write system call to write to the file. We supply the file descriptor, a character array and the length of data to be written.

The maximum value of PID can be 32768. We check the PID and return if the argument is invalid.

Create a pointer to the task_struct which we will use to get process details. Then create a file pointer which will represent an open file.

loff_t is used to the determine the current position in the file.

We increase the valid range of addresses for the buffer, by adding the kernel data space to it.

We open the file by specifying the filename and the mode (write only) in which we want to open it. We also create a file if it doesn't exist, and 0644 (owner can read and write) specifies the properties of the file if we *create* it.

System call code continued from the previous slide.

```
        for_each_process(task) {

                if(pid == (long) task->pid) {
                        printk("Process: %s\nPID_Number: %ld\nProcess State: %ld\nPriority: %ld\n",
task->comm, (long) task->pid, (long) task->state, (long) task->prio);


                        sprintf(temp, "Process %s\nPID_Number: %ld\nProcess State: %ld\nPriority: %l
d\n", task->comm, (long) task->pid, (long) task->state, (long) task->prio);


                        strcat(data, temp);

                        if(fd < 0)
                                return -EISDIR; // you cannot write to a directory


                        file = fget(fd);
                        write_to_file(file, data);

                }

        }


        set_fs(old_fs);
        return 0;
}
```

We use a macro defined in **sched**.h to iterate over all the processes.

We print some details about the required process to the kernel log. We also add it to the string **data** which we use to write to a file.

Check for unsuccessful opening of the file.

We change the limit of valid addresses back to the user address range.

We create a **`Makefile`** in the folder and add the following text to it. This is to make sure our system call is added to the kernel source code.

```
anmolgupta@ubuntu:~/linux-3.16-rc-onlychanged/taskinfo$ cat Makefile
obj-y:=list_task_info.o
anmolgupta@ubuntu:~/linux-3.16-rc-onlychanged/taskinfo$
```

We append the name of the directory **`taskinfo`** to this line in **`Makefile`** of the kernel source code. This tells the compiler where to look for our system call.

```
ifeq ($(KBUILD_EXTMOD),)
core-y          += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ hello/ taskinfo/
```

Inside the source code folder, we go to **`arch/x86/syscalls`** and add the following line to the end of the **`syscall_32.tbl`** file.

```
352     i386    sched_getattr           sys_sched_getattr
353     i386    renameat2               sys_renameat2
354     i386    hello                   sys_hello
355     i386    taskinfo                sys_sh_task_info
-- INSERT --
```

Inside the source code folder, we go to **`include/linux`** and add the new system call to the **`syscalls.h`** header file.

```
asmlinkage long sys_kcmp(pid_t pid1, pid_t pid2, int type,
                         unsigned long idx1, unsigned long idx2);
asmlinkage long sys_finit_module(int fd, const char __user *uargs, int flags);
asmlinkage long sys_sh_task_info(long pid, char *filename);
#endif
```

We finally run the commands **`make`**, **`make modules_install`** and **`make install`** to compile the kernel source and integrate our system call.

We create a **`test.c`** file to test our system call and we'll store the output in the file **`syscall_output`**.

```
anmolgupta@ubuntu:~$ ls
linux-3.16-rc2  linux-3.16-rc2.tar.gz  linux-3.16-rc-onlychanged  syscall_output  test.c
anmolgupta@ubuntu:~$ _
```

Add code in **`test.c`** to test your system call.

```c
#include<stdio.h>
#include<linux/kernel.h>
#include<sys/syscall.h>
#include<errno.h>
#include<unistd.h>

int main(int argc, char **argv) {
        int pid = atoi(argv[1]);
        long result = syscall(355, pid, argv[2]);
        if(result == 0)
                printf("System call sys_hello returned %ld\n", result);

        else {
                printf("Error :(\n");
                perror("Error ");
                printf("Error Number: %d\n", errno);

        }

        return 0;
}
```

We compile **`test.c`** and then run the executable **`a.out`** with two arguments--the **PID number of the process** and the **path of the file where process details have to be stored**. As expected, we see that the output gets written in the **`syscall_output`** file and is also printed to the kernel log.

```
anmolgupta@ubuntu:~$ gcc test.c
anmolgupta@ubuntu:~$ ./a.out 3 syscall_output
System call sys_hello returned 0
anmolgupta@ubuntu:~$ cat syscall_output
Process: ksoftirqd/0
PID Number: 3
Process State: 1
Priority: 120
anmolgupta@ubuntu:~$ _
```

We are getting the following details from the **`task_struct`**:

i. **process name** - init, kthread, etc.

ii. **process number** - unique process ID

iii. **process state** - either created, ready, running, blocked or terminated state

iv. **process priority** - used to carry out resource sharing between the processes

```
[   29.723879] systemd-logind[1169]: New session c1 of user anmolgupta.
[10130.104247] Process: ksoftirqd/0
[10130.104250] PID Number: 3
[10130.104252] Process State: 1
[10130.104254] Priority: 120
anmolgupta@ubuntu:~$ history 2
 1136  dmesg
 1137  history 2
anmolgupta@ubuntu:~$ _
```

We have used **`errno.h`** header file to display the error messages for both **invalid argument due to a wrong PID** and trying to **write to a directory**.

```
anmolgupta@ubuntu:~$ ./a.out 0 syscall_output
Error :(
Error : Invalid argument
Error Number: 22
anmolgupta@ubuntu:~$ ./a.out 1 syscall_output/
Error :(
Error : Is a directory
Error Number: 21
anmolgupta@ubuntu:~$
```