CSE231: OPERATING SYSTEMS
ASSIGNMENT-2 (PART-2)

RAHUL KHATOLIYA

2012965

CSAM" 2023

# DESCRIPTION OF CODE , IMPLEMENTATION , AND ERROR HANDLING :

A) SYSTEM CALL CODE (sh_task_info.c) :

# HEADERS USED :

```
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/sched.h>
#include <linux/errno.h>
#include <linux/syscalls.h>
#include <linux/module.h>
#include <linux/uaccess.h>
#include <linux/fs.h>
#include <linux/fcntl.h>
#include <linux/file.h>
```

Following the Syscall Macro convention , we have to provide **'N'** as the number of argument to be passed in the system call , and hence we have used the definition as **SYSTEM_DEFINE2** , Since in our case we have to pass 2 arguments , i.e **Process ID** and **filename/path** .

In first segment , we have checked whether the provided **PID** is negative or not , if yes then simple return the error_number corresponding to the invalid argument i.e **-EINVAL** using header **<errno.h>** which helps to handle errors and throw corresponding error code . Moreover , when used such negative values , using **Perror** we will print on the terminal " **sh_task_info failure: Invalid argument** " , and using **dmesg** command , we can see " **ERROR FOUND :**

NEGATIVE PID IS RESTRICTED " , which we have printed inside the kenel using printk() .

Going ahead , we have initialized buffer for the filename , so that we can use copy_from_user() , to fill out this buffer with the argument provided from user space for the corresponding filename/path , in which the information is to be written . Also we initialized pointers like task_struct and file , here we will iterate over all the process running currently , and will search for the process having ID as those provided by the arguments ,
And hence will log the same in kernel using printk() and using the file pointer we will use the filp_open() , to open the corresponding file name and write into it . Moreover we have provided flags such as O_WRONLY and O_TRUNC , which will insure that the file is opened for write only and the length will be truncated to 0 , if the file exist .

The file pointer thus used for opening the file , will be tested , whether the opening was successful or not , so using IS_ERR() , it will insure to throw corresponding error code -ENOENT , used when file or directory is missing . Else we will , simply fill out the buffer that we have initialized specially for the contents to be written in the file , using sprintf() , and finally writing the same to the file using kernel_write() .

At last we will simply close the file using filp_close() , and check if we have found the process corresponding to the provided PID or not , using an int variable , so if not found we will return the corresponding Error Code -ESRCH (responsible for throwing error when no process for given PID exist) and will log "ERROR : NO SUCH PROCESS EXIST" , and if exist then "SUCCESS : PROCESS FOUND" .

B) TEST PROGRAM CODE (test.c) :

# HEADERS USED :

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

Firstly we have created a Wrapper function for calling our **Syscall sh_task_info** , and since we had added the details of our system call at line number **440** of the **systable** , therefore we have provided the same number to call our required **Syscall** .

Inside Main method , we have initialized a **char** variable with **'y'** , and we have put the remaining code inside **do-while loop** so that as long as the user wants to use the implemented system call , she/he would be asked to continue or not , therefore until the value of this char variable is not other than **'y'** or **'Y'** , the loop will continue to execute .

Inside the loop , we have initialized two buffers , which would take care of the input for the **PID** and **FileName** , and since inside the **sh_task_info** , we have fully handled and returned the **error codes** corresponding to different types of errors , we have used simply the **perror()** , to print out the corresponding string to the error codes in the terminal , if found any , and otherwise we will simply print "**sh_task_info : SUCCESS**" . it must be noted that we had assigned a **long variable** to the returned value of our system call , and hence only when **negative values** are present , we would take **perror()** into consideration .

C) INPUTS AND EXPECTED OUTPUTS :

It must be noted that for PID argument , if it is negative , the buffer to be logged inside kernel will be " ERROR FOUND : NEGATIVE PID IS RESTRICTED " , and on the terminal the output will be " sh_task_info failure: Invalid argument ". Whereas if it's a number then on successful search , it will log " SUCCESS : PROCESS FOUND " on kernel and on the terminal it will show "sh_task_info : SUCCESS" , also if not found then it will log "NO SUCH PROCESS EXIST" and on terminal "sh_task_info failure: No such process" .

For filename argument given by user , if the file or directory is not present , then it will log " ERROR : CAN'T OPEN FILE <filename/path> " on the Kernel , and on the terminal " sh_task_info failure: No such file or directory " .

It must be noted that for PID , if its alphanumeric or alphabetic or any special character , the logged value will be " NO SUCH PROCESS EXIST " and on terminal it will be "sh_task_info failure: No such process" .

NOTE :

In Case when both PID and filename/path are buggy , then it will throw out the error handling corresponding to PID only , and would result in logging of "NO SUCH PROCESS EXIST" and on terminal "sh_task_info failure: No such process" .

D) SOURCE CODES :

(sh_task_info.c) :

```
#include <linux/kernel.h>
```

```
#include <linux/init.h>
#include <linux/sched.h>
#include <linux/errno.h>
#include <linux/syscalls.h>
#include <linux/module.h>
#include <linux/uaccess.h>
#include <linux/fs.h>
#include <linux/fcntl.h>
#include <linux/file.h>


SYSCALL_DEFINE2(sh_task_info, int, pi, const char*, filePath)
{
        if(pi<0){
                printk("ERROR FOUND : NEGATIVE PID IS RESTRICTED ");
                return -EINVAL;
        }
        else{
                char fileName[512];
                copy_from_user(fileName,filePath,512);
                struct file *fptr;
                struct task_struct* task;
                int t=0;
                char Contents[600];

                for_each_process(task){
                        if((int)task->pid==pi){
                                t=1;
                                printk(
                                        "\n PROCESS INFO : \n \
                                        PROCESS ID = %d\n \
                                        PROCESS NAME = %s\n \
                                        PROCESS STATE = %ld\n \
                                        PRIORITY = %ld\n \
                                        EXECUTION TIME (utime) = %ld (ns) \n \
                                        PROCESS VRUNTIME = %ld (ns)\n \
                                        TOTAL TIME (stime) = %ld (ns) \n",
                                        (int)task->pid,
                                        task->comm, \
                                        (long)task->state, \
                                        (long)task->prio, \
                                        (long)task->utime, \
                                        (long)task->se.vruntime, \
                                        (long)task->stime\
                                        );
```

```
                      printk("SUCCESS : PROCESS FOUND");
                      fptr=filp_open(fileName,O_WRONLY |
O_TRUNC,777);

                  if(IS_ERR(fptr)){
                      printk("ERROR : CAN'T OPEN FILE %s",fileName);
                      return -ENOENT;
                  }
                  else{
                      sprintf(Contents,"\n PROCESS INFO : \n \
                          PROCESS ID = %d\n \
                          PROCESS NAME = %s\n \
                          PROCESS STATE = %ld\n \
                          PRIORITY = %ld\n \
                          EXECUTION TIME (utime) = %ld (ns) \n \
                          PROCESS VRUNTIME = %ld (ns)\n \
                          TOTAL TIME (stime) = %ld (ns) \n",
                          (int)task->pid,
                          task->comm, \
                          (long)task->state, \
                          (long)task->prio, \
                          (long)task->utime, \
                          (long)task->se.vruntime, \
                          (long)task->stime\
                      );

                      kernel_write(fptr,Contents,strlen(Contents),0);
                      filp_close(fptr,NULL);
              }
      }
      }
          if(t==0){
          printk(KERN_INFO "NO SUCH PROCESS EXIST \n");
          return -ESRCH;
          /*NO PRINTING INSIDE FILE*/
      }
      }
      return 0;
  }
```

(test.c) :

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

#define __NR_sh_task_info 440

long sh_task_info_syscall(char* fileName,int pi)
{
    return syscall(__NR_sh_task_info,pi,fileName);
}

int main()
{
    char Continue='y';
    do{
        printf("\n ENTER PROCESS ID : ");
        char* pidArg=(char*)malloc(20*sizeof(char));
        scanf("%s",pidArg);
        int pid=atoi(pidArg);

        printf("\n ENTER FILENAME/PATH : ");
        char* fileName = (char*) malloc(512*sizeof(char));
        scanf("%s", fileName);
        long activity;
        activity = sh_task_info_syscall(fileName,pid);
        if(activity < 0)
        {
            perror(" sh_task_info failure");
        }
        else
        {
            printf(" sh_task_info : SUCCESS");
        }
        printf("\n Want to test more (y) ?");
        scanf(" %c",&Continue);

    }while(Continue=='y' || Continue=='Y');
    return 0;
}
```