Gitansh Raj Satija OS Assignment2 Question2

Code Explanation

335 common sh_task_info sys_sh_task_info
We add the above line in the syscalls_64.tbl. 335 was an unused sys call number hence we use that.
asmlinkage long sys_sh_task_info(int, char*);
We add this declaration to the syscalls.h file. It takes 2 arguments pid which is of int form and a char pointer which is for the filename. We print on the kernel logs using the printk function.

```
#include <linux/kernel.h>
#include <linux/string.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/sched.h>
#include <linux/uaccess.h>
#include <linux/linkage.h>
#include <linux/errno.h>
#include <linux/types.h>
#include <linux/list.h>
#include <linux/syscalls.h>
#include <asm/uaccess.h>
```

The following header files were used to implement the sys call.

```
SYSCALL_DEFINE2(sh_task_info, int, pid, char*, fname)
{
  struct task_struct *task;
  task = pid_task(find_vpid(pid), PIDTYPE_PID);
  if (task == NULL)
  {
    printk("Cannot find a process with that PID: %d\n", pid);
    return 1;
  }
  else
  {
    printk("PID: %d is the process  %s .\n", pid, task->comm);
    printk("vruntime is %lld\n", (task->se).vruntime);
    printk("Priority is %d\n", task->prio);
    printk("Static Priority is %d\n", task->static_prio);
    printk("Normal Priority is %d\n", task->normal_prio);
    char buf[256];
    loff_t pos;
    struct file *fp;
    mm_segment_t fs;
    long ch = strncpy_from_user(buf, fname, sizeof(buf));
    if (ch < 0 || ch == sizeof(buf)){
      printk("Error Code-14\n");
      return EFAULT;
    }
```

Since we are sending two input parameters to syscall we used SYSCALL_DEFINE2. The pid_task() function is used to find the task_struct corresponding to that pid. Then we print some of its field like PID, task name, vruntime, Priority and also static/normal priority. Then we create a file using filp_open() and append the data to be written on the file in a string and then write it. If success, syscall would return 0.

```
    fp = filp_open(buf, O_RDWR | O_CREAT, 0644);
    if (IS_ERR(fp))
    {
        int err=0;
        err=PTR_ERR(fp);
        printk("Error Code%d\n",err);
        return 1;
    }
    fs = get_fs();
    set_fs(KERNEL_DS);

    char str1[20];
    char str2[20];
    char str3[10];
    char str4[10];
    char str5[10];
    char b2[512]="";
    sprintf(str1, "%d", pid);
    sprintf(str2, "%lld",(task->se).vruntime);
    sprintf(str3, "%d", task->prio);
    sprintf(str4, "%d", task->static_prio);
    sprintf(str5, "%d", task->normal_prio);
    strcat(b2,"PID: ");
    strcat(b2,str1);
    strcat(b2,"\n");
    strcat(b2,"Process Name ");
    strcat(b2,task->comm);
    strcat(b2,"\n");
    strcat(b2,"Priority is ");
    strcat(b2,str3);
    strcat(b2,"\n");
    strcat(b2,"Static Priority is ");
    strcat(b2,str4);
    strcat(b2,"\n");
    strcat(b2,"Normal Priority is ");
    strcat(b2,str5);
    strcat(b2,"\n");
    strcat(b2,"vruntime is ");
    strcat(b2,str2);
    strcat(b2,"\n");
    pos=0;
    vfs_write(fp,b2, strlen(b2), &pos);
    filp_close(fp,NULL);
    set_fs(fs);
   }
  return 0;
}
```

Input/output

Here are the sample inputs shown. After correct input and implementation teh file containing data is created and the data is also printed in the kernel logs which can be seen by using the command dmesg |tail . In case of an error like invalid pid error message is stored in the kernel log and file is not created. In case of error in creating a file, data is printed in kernel log along with error message however file is not created.

```
git@ubuntu:~/Desktop/Ques2$ make run
gcc test.c
./a.out
Enter pid value:1
Enter file path:/home/git/Desktop/abc.txt
/home/git/Desktop/abc.txt
System call sys_sh_task_info returned 0. Check Kernel Logs
git@ubuntu:~/Desktop/Ques2$ make run
gcc test.c
./a.out
Enter pid value:50
Enter file path:/home/git/Desktop/def.txt
/home/git/Desktop/def.txt
System call sys_sh_task_info returned 1. Check Kernel Logs
git@ubuntu:~/Desktop/Ques2$ make run
gcc test.c
./a.out
Enter pid value:108
Enter file path:/home/git/Desktop
/home/git/Desktop
System call sys_sh_task_info returned 1. Check Kernel Logs
```

```
git@ubuntu:~/Desktop/Ques2$ dmesg |tail
[ 3918.405154] Priority is 120
[ 3918.405154] Static Priority is 120
[ 3918.405155] Normal Priority is 120
[ 3971.928653] Cannot find a process with that PID: 50
[ 4010.331411] PID: 108 is the process  irq/39-pciehp .
[ 4010.331413] vruntime is 0
[ 4010.331414] Priority is 49
[ 4010.331415] Static Priority is 120
[ 4010.331415] Normal Priority is 49
[ 4010.331429] Error Code-21
```

Error Handling:

Syscall returns 1 in case of error. Error handling has been done in both the syscall implementation and the test.c file. If a wrong(invalid) pid is entered i.e. a process with that pid does not exist then we get a corresponding message. In this case file is not created.

```
if (task == NULL)
{
  printk("Cannot find a process with that PID: %d\n", pid);
  return 1;
}
```

In the case of a bad address entry (null or long path (assumption: 256 bytes is max path limit for filename). The EFAULT flag is raised and error code 14 is returned which signifies bad address.

```
if (ch < 0 || ch == sizeof(buf)){
  printk("Error Code-14\n");
  return EFAULT;
}
```

In case there is some problem in opening the file i.e. file is a directory/ write permission denied or any other such error , we use IS_ERR to handle that. PTR_ERR generates the error code which is printed in the kernel log. In this case kernel log also prints the pid data.

```
if (IS_ERR(fp))
{
    int err=0;
    err=PTR_ERR(fp);
    printk("Error Code%d\n",err);
    return 1;
}
```

Please note the error message corresponding to the error code:-

```
#define EPERM  1 /* Operation not permitted */
#define ENOENT  2 /* No such file or directory */
#define ESRCH  3 /* No such process */
#define EINTR  4 /* Interrupted system call */
#define EIO  5 /* I/O error */
#define ENXIO  6 /* No such device or address */
#define E2BIG  7 /* Argument list too long */
#define ENOEXEC  8 /* Exec format error */
#define EBADF  9 /* Bad file number */
#define ECHILD  10 /* No child processes */
#define EAGAIN  11 /* Try again */
#define ENOMEM  12 /* Out of memory */
#define EACCES  13 /* Permission denied */
#define EFAULT  14 /* Bad address */
#define ENOTBLK  15 /* Block device required */
#define EBUSY  16 /* Device or resource busy */
#define EEXIST  17 /* File exists */
```

```
#define EXDEV  18 /* Cross-device link */
#define ENODEV  19 /* No such device */
#define ENOTDIR  20 /* Not a directory */
#define EISDIR  21 /* Is a directory */
#define EINVAL  22 /* Invalid argument */
#define ENFILE  23 /* File table overflow */
#define EMFILE  24 /* Too many open files */
#define ENOTTY  25 /* Not a typewriter */
#define ETXTBSY  26 /* Text file busy */
#define EFBIG  27 /* File too large */
#define ENOSPC  28 /* No space left on device */
#define ESPIPE  29 /* Illegal seek */
#define EROFS  30 /* Read-only file system */
#define EMLINK  31 /* Too many links */
#define EPIPE  32 /* Broken pipe */
#define EDOM  33 /* Math argument out of domain of func */
#define ERANGE  34 /* Math result not representable */
```

Apart from this, error handling has also been done in test.c for handling errors relating to calls like scanf() and printf(). Relevant error messages are generated using perror functionality.