OS Assignment 1

Baani Leen (2016234) | Viresh Gupta (2016118)

February 11, 2018

Syscalls implemented (Name : Number)-

hello_world : 318
sh_task_info : 319

Logical Details of these system calls

hello_world :
This system call is simply to test the functionality of the kernel logging. It has the following signature :

hello_world*(void);*

It takes in no arguments and simply prints a test message to the kernel log. This message can be viewed using *dmesg* command.
No errors thrown.

sh_task_info :
This system call is to write the details of process identified by the given pid and write it out to the given file name. Additionally this information is also printed out on to the kernel log.
If the pid argument is equal to 0, then information about the process who is calling this system call is returned.

It has the following signature :

sh_task_info*(int* ***pid****, char ** ***filename****);*

It takes in two arguments. The *pid* identifies the process whose information is to be gathered and *filename* is the name of the file where the information

has to stored.
It can throw ESRCH (when the given PID is not found or is invalid), EINVAL (when the given filename is invalide or not sufficient permissions with the process), EFAULT (miscellaneous).
For a given pid, the following information is printed and written to file:

1. PID of the given process
2. PID of the parent of the given process
3. Group ID of the given process (different from PID in case of threads)
4. Total CPU time in user mode
5. Total CPU time in system mode
6. It's state (uninterruptible / interruptible / running / zombie etc)
7. Its total running time
8. It's assigned time slice
9. It's scheduling policy (FIFO / RR / NORMAL / CFS)
10. The number of context switches it has had (Sum of voluntary and involuntary switches)
11. The process name (as passed on by exec)

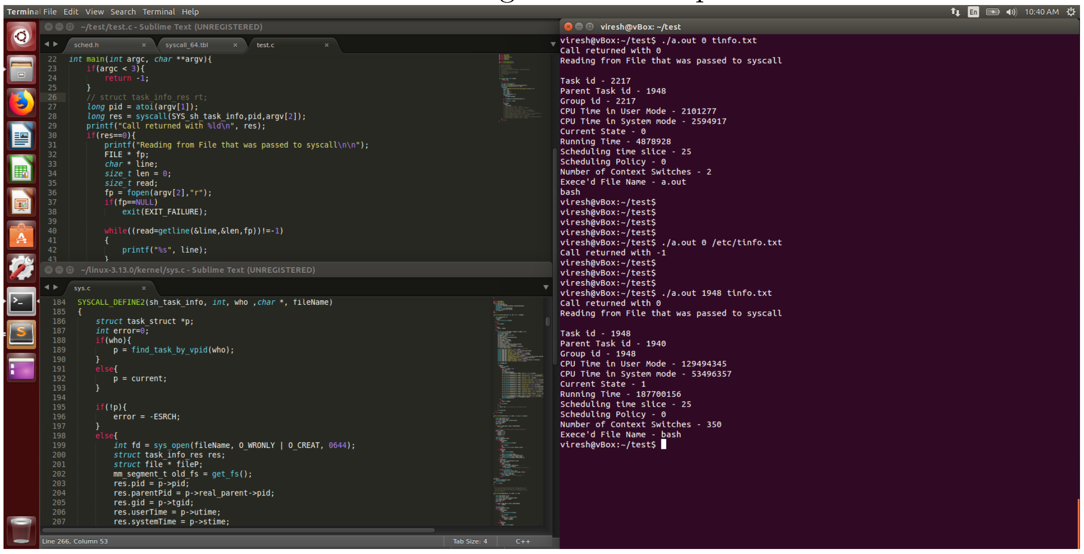
Implementation Details of these system calls

hello_world :
This system call is implemented via the printk function. It uses logging level of KERN_INFO.

sh_task_info :
This system call is implemented using a variety of techniques.
Firstly the given process id is used to find the corresponding task_struct using the function find_task_by_vpid(int).
Then the appropriate information from this task struct is stored into a struct format (Redundant work, but was done in case we had to return these values into user space).
Now these values are printed on the kernel log using printk().
After the above is completed, a file is opened in the user space using sys_open().

Next the memory segment is switched from user space to kernel space. This is done so that the information can be printed from kernel memory into file. All the information from the struct is then dumped into a single string and this string is then written to the file opened above (using vfs_write()).

A screenshot of all the above along with the output:


Terminal File Edit View Search Terminal Help
~/test/test.c - Sublime Text (UNREGISTERED)
sched.h
syscall_64.tbl
test.c
~/linux-3.13.0/kernel/sys.c - Sublime Text (UNREGISTERED)
sys.c
Line 266, Column 53
Tab Size: 4
C++
viresh@vBox: ~/test

```c
int main(int argc, char **argv){
    if(argc < 3){
        return -1;
    }
    // struct task_info_res rt;
    long pid = atoi(argv[1]);
    long res = syscall(SYS_sh_task_info,pid,argv[2]);
    printf("Call returned with %ld\n", res);
    if(res==0){
        printf("Reading from File that was passed to syscall\n\n");
        FILE * fp;
        char * line;
        size_t len = 0;
        size_t read;
        fp = fopen(argv[2],"r");
        if(fp==NULL)
            exit(EXIT_FAILURE);

        while((read=getline(&line,&len,fp))!=-1)
        {
            printf("%s", line);
        }
```

```c
SYSCALL_DEFINE2(sh_task_info, int, who ,char *, fileName)
{
    struct task_struct *p;
    int error=0;
    if(who){
        p = find_task_by_vpid(who);
    }
    else{
        p = current;
    }

    if(!p){
        error = -ESRCH;
    }
    else{
        int fd = sys_open(fileName, O_WRONLY | O_CREAT, 0644);
        struct task_info_res res;
        struct file * fileP;
        mm_segment_t old_fs = get_fs();
        res.pid = p->pid;
        res.parentPid = p->real_parent->pid;
        res.gid = p->tgid;
        res.userTime = p->utime;
        res.systemTime = p->stime;
```

```
viresh@vBox:~/test$ ./a.out 0 tinfo.txt
Call returned with 0
Reading from File that was passed to syscall

Task id - 2217
Parent Task id - 1948
Group id - 2217
CPU Time in User Mode - 2101277
CPU Time in System mode - 2594917
Current State - 0
Running Time - 4878928
Scheduling time slice - 25
Scheduling Policy - 0
Number of Context Switches - 2
Exece'd File Name - a.out
bash
viresh@vBox:~/test$
viresh@vBox:~/test$
viresh@vBox:~/test$
viresh@vBox:~/test$
viresh@vBox:~/test$ ./a.out 0 /etc/tinfo.txt
Call returned with -1
viresh@vBox:~/test$
viresh@vBox:~/test$
viresh@vBox:~/test$
viresh@vBox:~/test$ ./a.out 1948 tinfo.txt
Call returned with 0
Reading from File that was passed to syscall

Task id - 1948
Parent Task id - 1940
Group id - 1948
CPU Time in User Mode - 129494345
CPU Time in System mode - 53496357
Current State - 1
Running Time - 187700156
Scheduling time slice - 25
Scheduling Policy - 0
Number of Context Switches - 350
Exece'd File Name - bash
viresh@vBox:~/test$
```

How to use the patch ?

The patch was created using (from the directory containing the modified kernel src) :

```
diff -rcNP OriginalKernel/linux-3.13.0/ linux-3.13.0/ > patch.diff
```

Thus the patch can be applied as follows (from the root of the kernel source tree) :

```
patch -p2 -R < patch.diff
```

How to use the given test file ?

Once booted up with the custom kernel, we just need to compile the file using gcc and run it with appropriate arguments.

An example run:

```
viresh@vBox:~/test$ uname -r
3.13.11-ckt39-custom
viresh@vBox:~/test$ ls
test.c
viresh@vBox:~/test$ gcc test.c
viresh@vBox:~/test$ ./a.out 0 tinfo.txt
Call returned with 0
Reading from File that was passed to syscall

Task id - 10780
Parent Task id - 10702
Group id - 10780
CPU Time in User Mode - 2122716
CPU Time in System mode - 2420158
Current State - 0
Running Time - 4607344
Scheduling time slice - 25
Scheduling Policy - 0
Number of Context Switches - 0
Exece'd File Name - a.out
viresh@vBox:~/test$ ls
a.out  test.c  tinfo.txt
viresh@vBox:~/test$
```

The arguments in the test file:
./a.out <pid> <filepath>

Code Files :

Code for hello_world syscall (inside src/kernel/sys.c) :

```
SYSCALL_DEFINE0(hello_world)
{
    printk( KERN_INFO "Hey there !. Hello World. VKernel");
    return 100;
}
```

Code for sh_task_info syscall (inside src/kernel/sys.c):

```
#ifndef TASK_INFO_H
#define TASK_INFO_H
struct task_info_res{
    int pid,parentPid,gid;
    unsigned long userTime,systemTime, numContextSwitches;
    long state;
    unsigned long long totalRunTime;
    unsigned int time_slice, policy;
    char name[16];
};
#endif

SYSCALL_DEFINE2(sh_task_info, int, who ,char *, fileName)
{
    struct task_struct *p;
    int error=0;
    if(who){
        p = find_task_by_vpid(who);
    }
    else{
        p = current;
    }

    if(!p){
        error = -ESRCH;
    }
    else{
```

```
   int fd = sys_open(fileName, O_WRONLY | O_CREAT, 0644);
   struct task_info_res res;
   struct file * fileP;
   mm_segment_t old_fs = get_fs();
   res.pid = p->pid;
   res.parentPid = p->real_parent->pid;
   res.gid = p->tgid;
   res.userTime = p->utime;
   res.systemTime = p->stime;
   res.state = p->state;
   res.policy = p->policy;
   res.totalRunTime  = (p->se).sum_exec_runtime;
   res.time_slice = (p->rt).time_slice;
   res.numContextSwitches = (p->nvcsw + p->nivcsw);
   strcpy(res.name,p->comm);

   printk( KERN_INFO "Task id - %d\n", p->pid);
printk( KERN_INFO "Parent Task id - %d\n", p->real_parent->pid);
   printk( KERN_INFO "Group id - %d\n", p->tgid);
printk( KERN_INFO "CPU Time in user mode - %lu\n", p->utime);
printk( KERN_INFO "CPU Time in System mode - %lu\n", p->stime);
   printk( KERN_INFO "Current State - %ld\n", p->state);
printk( KERN_INFO "Scheduled Average Running Time - %llu\n",
     (p->se).sum_exec_runtime);
printk( KERN_INFO "Scheduling time slice - %u\n", (p->rt).time_slice);
  printk( KERN_INFO "Scheduling Policy - %u\n", p->policy);
   printk( KERN_INFO "Number of Context Switches - %lu\n",
     (p->nvcsw + p->nivcsw));
   printk( KERN_INFO "Exece'd File Name - %s\n", p->comm);

   set_fs(KERNEL_DS);

   if(fd>=0){
       fileP = fget(fd);
       if(fileP){
           const int MSIZE = 1000;
           loff_t pos = 0;
           int cx=0;
           char tempHolder[MSIZE];
      cx += snprintf(tempHolder+cx, MSIZE, "Task id - %d\n", res.pid);
      cx += snprintf(tempHolder+cx, MSIZE, "Parent Task id - %d\n",
                res.parentPid);
            cx += snprintf(tempHolder+cx, MSIZE, "Group id - %d\n", res.gid);
            cx += snprintf(tempHolder+cx, MSIZE, "CPU Time in User Mode -
                %lu\n", res.userTime);
            cx += snprintf(tempHolder+cx, MSIZE, "CPU Time in System mode
                - %lu\n", res.systemTime);
            cx += snprintf(tempHolder+cx, MSIZE, "Current State - %ld\n",
                res.state);
            cx += snprintf(tempHolder+cx, MSIZE, "Running Time - %llu\n",
                res.totalRunTime);
            cx += snprintf(tempHolder+cx, MSIZE, "Scheduling time slice -
                %u\n", (res.time_slice));
            cx += snprintf(tempHolder+cx, MSIZE, "Scheduling Policy - %d\n",
                res.policy);
            cx += snprintf(tempHolder+cx, MSIZE, "Number of Context Switches
                - %lu\n", (res.numContextSwitches));
            cx += snprintf(tempHolder+cx, MSIZE, "Exece'd File Name - %s\n",
                res.name);
            vfs_write(fileP,tempHolder, strlen(tempHolder),&pos);
                filp_close(fileP,NULL);
             }
             else{
                 error = -EINVAL;
             }
             sys_close(fd);
        }
        else{
            error = fd; // Not accesible from username (EFAULT)
        }

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

Code for testing the syscall:

```
#define _GNU_SOURCE
#include <unistd.h>
#include <sys/syscall.h>
```

```c
#include <stdio.h>
#include <stdlib.h>

#define SYS_hello_world 318
#define SYS_sh_task_info 319

int main(int argc, char **argv){
    if(argc < 3){
        return -1;
    }
    long pid = atoi(argv[1]);
    long res = syscall(SYS_sh_task_info,pid,argv[2]);
    printf("Call returned with %ld\n", res);
    if(res==0){
      printf("Reading from File that was passed to syscall\n\n");
        FILE * fp;
        char * line;
        size_t len = 0;
        size_t read;
        fp = fopen(argv[2],"r");
        if(fp==NULL)
            exit(EXIT_FAILURE);

        while((read=getline(&line,&len,fp))!=-1)
        {
            printf("%s", line);
        }
        fclose(fp);
        if(line)
            free(line);
    }
    return 0;
}
```