OS Assignment 1

Viresh Gupta (2016118)

May 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. state
2. cpu
3. prio
4. static_prio
5. normal_prio;
6. rt_priority;
7. se; /*sched entity*/
8. struct thread_struct thread;
9. pid;
10. tgid;
11. sigset_t blocked; (Printed signal numbers in this sigset)
12. sigset_t real_blocked; (Printed signal numbers in this sigset)
13. PID of the parent of the given process
14. The process name (as passed on by exec)

Implementation Details of these system calls

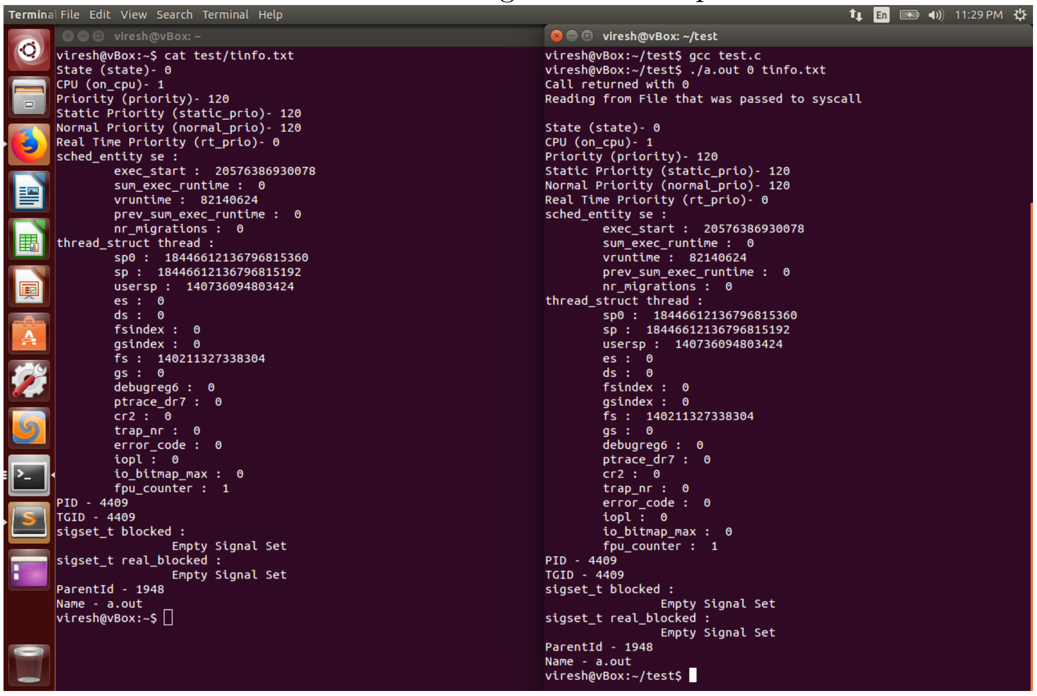
hello_world :

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).
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:



```
viresh@vBox:~$ cat test/tinfo.txt
State (state)- 0
CPU (on_cpu)- 1
Priority (priority)- 120
Static Priority (static_prio)- 120
Normal Priority (normal_prio)- 120
Real Time Priority (rt_prio)- 0
sched_entity se :
        exec_start :  20576386930078
        sum_exec_runtime :  0
        vruntime :  82140624
        prev_sum_exec_runtime :  0
        nr_migrations :  0
thread_struct thread :
        sp0 :  18446612136796815360
        sp :  18446612136796815192
        usersp :  140736094803424
        es :  0
        ds :  0
        fsindex :  0
        gsindex :  0
        fs :  140211327338304
        gs :  0
        debugreg6 :  0
        ptrace_dr7 :  0
        cr2 :  0
        trap_nr :  0
        error_code :  0
        iopl :  0
        io_bitmap_max :  0
        fpu_counter :  1
PID - 4409
TGID - 4409
sigset_t blocked :
                Empty Signal Set
sigset_t real_blocked :
                Empty Signal Set
ParentId - 1948
Name - a.out
viresh@vBox:~$
```

```
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

State (state)- 0
CPU (on_cpu)- 1
Priority (priority)- 120
Static Priority (static_prio)- 120
Normal Priority (normal_prio)- 120
Real Time Priority (rt_prio)- 0
sched_entity se :
        exec_start :  20576386930078
        sum_exec_runtime :  0
        vruntime :  82140624
        prev_sum_exec_runtime :  0
        nr_migrations :  0
thread_struct thread :
        sp0 :  18446612136796815360
        sp :  18446612136796815192
        usersp :  140736094803424
        es :  0
        ds :  0
        fsindex :  0
        gsindex :  0
        fs :  140211327338304
        gs :  0
        debugreg6 :  0
        ptrace_dr7 :  0
        cr2 :  0
        trap_nr :  0
        error_code :  0
        iopl :  0
        io_bitmap_max :  0
        fpu_counter :  1
PID - 4409
TGID - 4409
sigset_t blocked :
                Empty Signal Set
sigset_t real_blocked :
                Empty Signal Set
ParentId - 1948
Name - a.out
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):

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

  printk(KERN_INFO "State (state)- %ld\n", p->state);
  printk(KERN_INFO "CPU (on_cpu)- %d\n", p->on_cpu);
  printk(KERN_INFO "Priority (prio)- %d\n", p->prio);
  printk(KERN_INFO "Static Priority (static_prio)- %d\n", p->static_prio);
  printk(KERN_INFO "Normal Priority (normal_prio)- %d\n", p->normal_prio);
  printk(KERN_INFO "Real Time Priority (rt_priority)- %d\n", p->rt_priority);

  printk(KERN_INFO "sched_entity se :  \n");
  printk(KERN_INFO "\texec_start :  %llu\n", (p->se).exec_start);
  printk(KERN_INFO "\tsum_exec_runtime :  %llu\n",
    (p->se).sum_exec_runtime);
  printk(KERN_INFO "\tvruntime :  %llu\n", (p->se).vruntime);
  printk(KERN_INFO "\tprev_sum_exec_runtime :  %llu\n", (p->se).prev_sum_exec_runt
  printk(KERN_INFO "\tnr_migrations :  %llu\n", (p->se).nr_migrations);
  printk(KERN_INFO "thread_struct thread :  \n");
  printk(KERN_INFO "\tsp0 :  %lu\n", (p->thread).sp0);
  printk(KERN_INFO "\tsp :  %lu\n", (p->thread).sp);
  printk(KERN_INFO "\tusersp :  %lu\n", (p->thread).usersp);
  printk(KERN_INFO "\tes :  %hu\n", (p->thread).es);
  printk(KERN_INFO "\tds :  %hu\n", (p->thread).ds);
  printk(KERN_INFO "\tfsindex :  %hu\n", (p->thread).fsindex);
  printk(KERN_INFO "\tgsindex :  %hu\n", (p->thread).gsindex);
  printk(KERN_INFO "\tfs :  %lu\n", (p->thread).fs);
  printk(KERN_INFO "\tgs :  %lu\n", (p->thread).gs);
  printk(KERN_INFO "\tdebugreg6 : %lu\n", (p->thread).debugreg6);
  printk(KERN_INFO "\tptrace_dr7 : %lu\n", (p->thread).ptrace_dr7);
  printk(KERN_INFO "\tcr2 :  %lu\n", (p->thread).cr2);
  printk(KERN_INFO "\ttrap_nr :  %lu\n", (p->thread).trap_nr);
  printk(KERN_INFO "\terror_code : %lu\n", (p->thread).error_code);
  printk(KERN_INFO "\tiopl :  %lu\n", (p->thread).iopl);
  printk(KERN_INFO "\tio_bitmap_max : %u\n", (p->thread).io_bitmap_max);
  printk(KERN_INFO "\tfpu_counter : %u\n", (p->thread).fpu_counter);

  printk(KERN_INFO "PID - %d\n", p->pid);
  printk(KERN_INFO "TGID - %d\n", p->tgid);

  printk(KERN_INFO "sigset_t blocked :  \n\t");
  cnt=0;
  for(sig=1; sig<_NSIG; sig++){
    if(sigismember(&(p->blocked),sig)){
      cnt++;
      printk(KERN_INFO "%d ",sig);
    }
  }
  if(cnt==0){
    printk(KERN_INFO "\tEmpty Signal Set");
  }
  printk(KERN_INFO "\n");

  printk(KERN_INFO "sigset_t real_blocked :  \n\t");
  cnt=0;
  for(sig=1; sig<_NSIG; sig++){
    if(sigismember(&(p->real_blocked),sig)){
      cnt++;
      printk(KERN_INFO "%d ",sig);
    }
  }
  if(cnt==0){
    printk(KERN_INFO "\tEmpty Signal Set");
  }
  printk(KERN_INFO "\n");

  printk(KERN_INFO "ParentId - %d\n", p->real_parent->pid);
  printk(KERN_INFO "Name - %s\n", p->comm);

  if(!p){
    error = -ESRCH;
  }
  else{
    int fd = sys_open(fileName, O_WRONLY | O_CREAT, 0644);
    struct file * fileP;
    mm_segment_t old_fs = get_fs();
    set_fs(KERNEL_DS);
    if(fd>=0){
      fileP = fget(fd);
      if(fileP){
        const int MSIZE = 3000;
        loff_t pos = 0;
        int cx=0;
        char tempHolder[MSIZE];
     cx += snprintf(tempHolder+cx, MSIZE, "State (state)- %ld\n", p->state);
     cx += snprintf(tempHolder+cx, MSIZE, "CPU (on_cpu)- %d\n", p->on_cpu);
     cx += snprintf(tempHolder+cx, MSIZE, "Priority (priority)- %d\n", p->prio);
     cx += snprintf(tempHolder+cx, MSIZE, "Static Priority (static_prio)- %d\n",
          p->static_prio);
     cx += snprintf(tempHolder+cx, MSIZE, "Normal Priority (normal_prio)- %d\n",
          p->normal_prio);
     cx += snprintf(tempHolder+cx, MSIZE, "Real Time Priority (rt_prio)- %d\n",
          p->rt_priority);

     cx += snprintf(tempHolder+cx, MSIZE, "sched_entity se :  \n");
     cx += snprintf(tempHolder+cx, MSIZE, "\texec_start :  %llu\n",
          (p->se).exec_start);
     cx += snprintf(tempHolder+cx, MSIZE, "\tsum_exec_runtime :  %llu\n",
          (p->se).sum_exec_runtime);
```

```
cx += snprintf(tempHolder+cx, MSIZE, "\tvruntime : %llu\n",
     (p->se).vruntime);
cx += snprintf(tempHolder+cx, MSIZE, "\tprev_sum_exec_runtime :
      %llu\n", (p->se).prev_sum_exec_runtime);
cx += snprintf(tempHolder+cx, MSIZE, "\tnr_migrations : %llu\n",
     (p->se).nr_migrations);


cx += snprintf(tempHolder+cx, MSIZE, "thread_struct thread : \n");
cx += snprintf(tempHolder+cx, MSIZE, "\tsp0 : %lu\n", (p->thread).sp0);
cx += snprintf(tempHolder+cx, MSIZE, "\tsp : %lu\n", (p->thread).sp);
cx += snprintf(tempHolder+cx, MSIZE, "\tusersp : %lu\n", (p->thread).usersp)
cx += snprintf(tempHolder+cx, MSIZE, "\tes : %hu\n", (p->thread).es);
cx += snprintf(tempHolder+cx, MSIZE, "\tds : %hu\n", (p->thread).ds);
 cx += snprintf(tempHolder+cx, MSIZE, "\tfsindex : %hu\n",
     (p->thread).fsindex);
 cx += snprintf(tempHolder+cx, MSIZE, "\tgsindex : %hu\n",
     (p->thread).gsindex);
cx += snprintf(tempHolder+cx, MSIZE, "\tfs : %lu\n", (p->thread).fs);
cx += snprintf(tempHolder+cx, MSIZE, "\tgs : %lu\n", (p->thread).gs);
cx += snprintf(tempHolder+cx, MSIZE, "\tdebugreg6 : %lu\n",
     (p->thread).debugreg6);
cx += snprintf(tempHolder+cx, MSIZE, "\tptrace_dr7 : %lu\n",
     (p->thread).ptrace_dr7);
cx += snprintf(tempHolder+cx, MSIZE, "\tcr2 : %lu\n", (p->thread).cr2);
 cx += snprintf(tempHolder+cx, MSIZE, "\ttrap_nr : %lu\n",
     (p->thread).trap_nr);
cx += snprintf(tempHolder+cx, MSIZE, "\terror_code : %lu\n",
     (p->thread).error_code);
   cx += snprintf(tempHolder+cx, MSIZE, "\tiopl : %lu\n",
     (p->thread).iopl);
cx += snprintf(tempHolder+cx, MSIZE, "\tio_bitmap_max : %u\n",
     (p->thread).io_bitmap_max);
cx += snprintf(tempHolder+cx, MSIZE, "\tfpu_counter : %u\n",
     (p->thread).fpu_counter);

 cx += snprintf(tempHolder+cx, MSIZE, "PID - %d\n", p->pid);
cx += snprintf(tempHolder+cx, MSIZE, "TGID - %d\n", p->tgid);

cx += snprintf(tempHolder+cx, MSIZE, "sigset_t blocked : \n\t");
   cnt=0;
```

```
      for(sig=1; sig<_NSIG; sig++){
        if(sigismember(&(p->blocked),sig)){
          cnt++;
          cx += snprintf(tempHolder+cx, MSIZE, "%d ",sig);
        }
      }
      if(cnt==0){
     cx += snprintf(tempHolder+cx, MSIZE, "\tEmpty Signal Set");
      }

    cx += snprintf(tempHolder+cx, MSIZE, "\nsigset_t real_blocked :  \n\t");
      cnt=0;
      for(sig=1; sig<_NSIG; sig++){
        if(sigismember(&(p->real_blocked),sig)){
          cnt++;
          cx += snprintf(tempHolder+cx, MSIZE, "%d ",sig);
        }
      }
      if(cnt==0){
     cx += snprintf(tempHolder+cx, MSIZE, "\tEmpty Signal Set");
      }

     cx += snprintf(tempHolder+cx, MSIZE, "\nParentId - %d\n",
        p->real_parent->pid);
    cx += snprintf(tempHolder+cx, MSIZE, "Name - %s\n", p->comm);

      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>
#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;
}
```