Operating Systems - Winter 2018
Assignment 1

Date of Submission : February 17, 2018

A Simple System Call

Suhavi 2016099
Riya Sinha 2016079

Description
It is required of us to print all the fields of task_struct associated with a particular PID.

Functions Used

1. for_each_process : To iterate over all processes
2.The functions to get the various fields of task-struct:

```
proc->comm
(long)task_pid_nr(proc)
(long)proc->state
(long)proc->prio
(long)proc->rt_priority
(long)proc->static_prio
(long)proc->normal_prio
(long)proc->on_cpu
(unsigned int)proc->se.on_rq
proc->blocked
proc->real_blocked
proc->thread.sp
```

3. Sprintf: To output to a text file.

Logic/Process:

The code iterates for the Process ids of all process and compares that to the pid input by the user.

The details for the same get printed as the output using the functions described above.
sprintf has been used to output the same set ofinfo to a file.
fd>=0 implies the user input is correct by all means and the output gets printed to the file passed as an argument in the test file. (test.c acc to our code)
fd<0 is where the error handling comes in.

Implementation Details

Changes to the Source Code

1. First, we created a new directory in the linux source code called sh_task_info. (name as was required)
2. In this directory we created two files, a sh_task_info.h header file and another sh_task_info.c file which is the file that contains the system call.
3. Then we created the Makefile through which the binary for the .c file would be created.
4. Next, we change the Makefile of the source code. Here, we give the link to the new directory.
5. Next, in the arch directory we go to x86 folder and inside this there is the syscalls folder. In this folder we have the syscall_64.tbl file and here we add our system call.
6. Next we go to the include folder and inside that we go to the linux folder and to the syscalls.h header file. Here we give the link to our system call function that we wrote in sh_task_info.c
7. Finally we compile the kernel and update the grub.
8. To test the code we create a test file through which we see if we can conclude if the file was correctly executed or not as it returns 0 or -1, correct and incorrect implementation respectively.
9. The output is printed to a text file as well.

Input by User
In the test file
(318,pid_no.,filename) arguments are passed and the file compiled and run to get output.

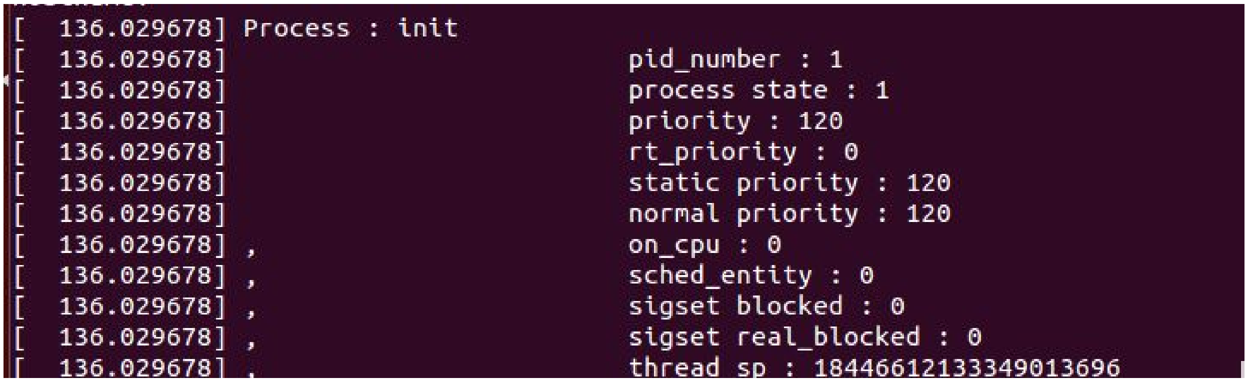
Error Handling
Error handling is done by errno which is an inbuilt variable in c.

Expected Output and Interpretation

The process name : proc->comm
The process ID : (long)task_pid_nr(proc)
Process State: (long)proc->state
Process priority: (long)proc->prio
Rt_priority: (long)proc->rt_priority
Process Static Priority: (long)proc->static_prio
Process Normal Priority: (long)proc->normal_prio
CPU / on-cpu: (long)proc->on_cpu

Ready Queue: (unsigned int)proc->se.on_rq
Stigset_blocked: proc->blocked
Stigset_real blocked: proc->real_blocked
Thread: proc->thread.sp

A terminal display of the output is as below:



```
[  136.029678] Process : init
[  136.029678]                          pid_number : 1
[  136.029678]                          process state : 1
[  136.029678]                          priority : 120
[  136.029678]                          rt_priority : 0
[  136.029678]                          static priority : 120
[  136.029678]                          normal priority : 120
[  136.029678] ,                        on_cpu : 0
[  136.029678] ,                        sched_entity : 0
[  136.029678] ,                        sigset blocked : 0
[  136.029678] ,                        sigset real_blocked : 0
[  136.029678] ,                        thread sp : 18446612133349013696
```

Sources:
medium.com
LKD by Robert Love
Elixir
Source Pages
Documents on errno