IMPLEMENTATION OF A NEW SYSTEM CALL

04.11.2017

AMMANAMANCHI SAI KARTHIK

B150310CS

B BATCH

10

KONGARI SAI MANOJ

B150610CS

B BATCH

35

TANGELLA MAHESH KUMAR

B150588CS

B BATCH

62

Abstract

This system call helps us to know the FILE NAME and PID of the runnable,terminated,unrunnable processes along with their total count .

Along with this ,the user also get to know the information(file name,pid) of the high priority ,low priority,and standard priority of the processes along with their count.

Directory for new system call:

1. Create a new directory, say ‘process_info’ and change to this directory.

2. Define the new system call in ‘process_info.c’ which contains the code for the new system call.

3. Write a make file in the same directory(i.e, process_info/) contains:

```
obj-y:=process_info.o
```

This ensures that process_info.c is compiled and included in the kernel source code.

Linking system call with kernel:

Now add the ‘process_info’ directory to the kernel make file.

The above link specifies compiler that source files of our new system call can be found in the /process_info directory.

Altering syscall_64.tbl:

To figure out where this file is present, we use the find command on the terminal from the linux-4.13.10 directory.

find -name syscall_64.tbl ### Should show the file's location

In kernel-4.13.10, it is present in /arch/x86/entry/syscalls/syscall_64.tbl

Now, edit the file to include the new system call number and it’s entry point.

Altering syscalls.h:

We use the find command to look for where the syscalls.h file is present.

```
find –name syscalls.h
```

In kernel-4.13.10, this is present in /include/linux/syscalls.h.

Add the following line at the end of the file(before the #endif) as shown:

```
asmlinkage long sys_process_info(void)
```

Recompile and Reboot:

To integrate the system call and to be able to actually use it, we will need to recompile the kernel.

```
sudo make -j 4 && sudo make modules_install -j 4 && sudo make install -j 4
```

Once this is done, we restart the system.

Testing the system call:

To test the system call, we write a simple ‘test.c’ function. Compile and execute this program. If it runs successfully, then, it should give the corresponding prompt and we can use ‘dmesg’ to check the kernel log.

Functionality:

This system call helps us to know the FILE NAME and PID of the runnable,terminated,unrunnable processes along with their total count .

Along with this ,the user also get to know the information(file name,pid) of the high priority ,low priority,and standard priority of the processes along with their count.