

LINUX OPERATING SYSTEM ASSIGNMENT-3

Members of the group:

s06-1-5-055 (Kirti Reddy)

s06-1-5-066 (Swati)

We have to add a system call named “getprocessinfo” which will be stored in prinfo.c in the Kernel directory in the linux kernel.

1.) Extract the kernel from the linux-2.6.28.tar file using the command
tar -xvj /usr/src/KERNEL/linux-2.6.28.tar

2.) Open the following file:
usr/src/KERNEL/linux-2.6.28/arch/x86/include/asm/unistd_32.h

Scrolled to the bottom of the define statements and added a line:
#define __NR_getprocessinfo 333

333 is the system call number.

3.) Open the following file:
usr/src/KERNEL/linux-2.6.28/arch/x86/kernel/syscall_table_32.S

Scroll to the end of the file and add the following line:
.long sys_getprocessinfo

3.) To actually implement the system call, we created a new c prinfo.c file in the following directory:
/usr/src/KERNEL/linux-2.6.28/kernel/

4.) Edit the makefile present in the /usr/src/KERNEL/linux-2.6.28/kernel/ directory to add
obj-y = sched.o fork.o exec_domain.o panic.o ...

.....
..... pm_qos_params.o sched_clock.o **prinfo.o**

Notice the prinfo.o in the above line which has to be added so as to make the object file of prinfo.c while compiling the kernel.

5.) A header file named prinfo.h has to be included in the folder “linux-2.6.28/include/linux”. The c file prinfo.c has to include this header file.

6.) We then recompiled the kernel using the following commands:

```
cd /KERNEL/linux-2.6.28
make O=/home/name/build/kernel menuconfig
make O=/home/name/build/kernel
sudo make O=/home/name/build/kernel modules_install install
```

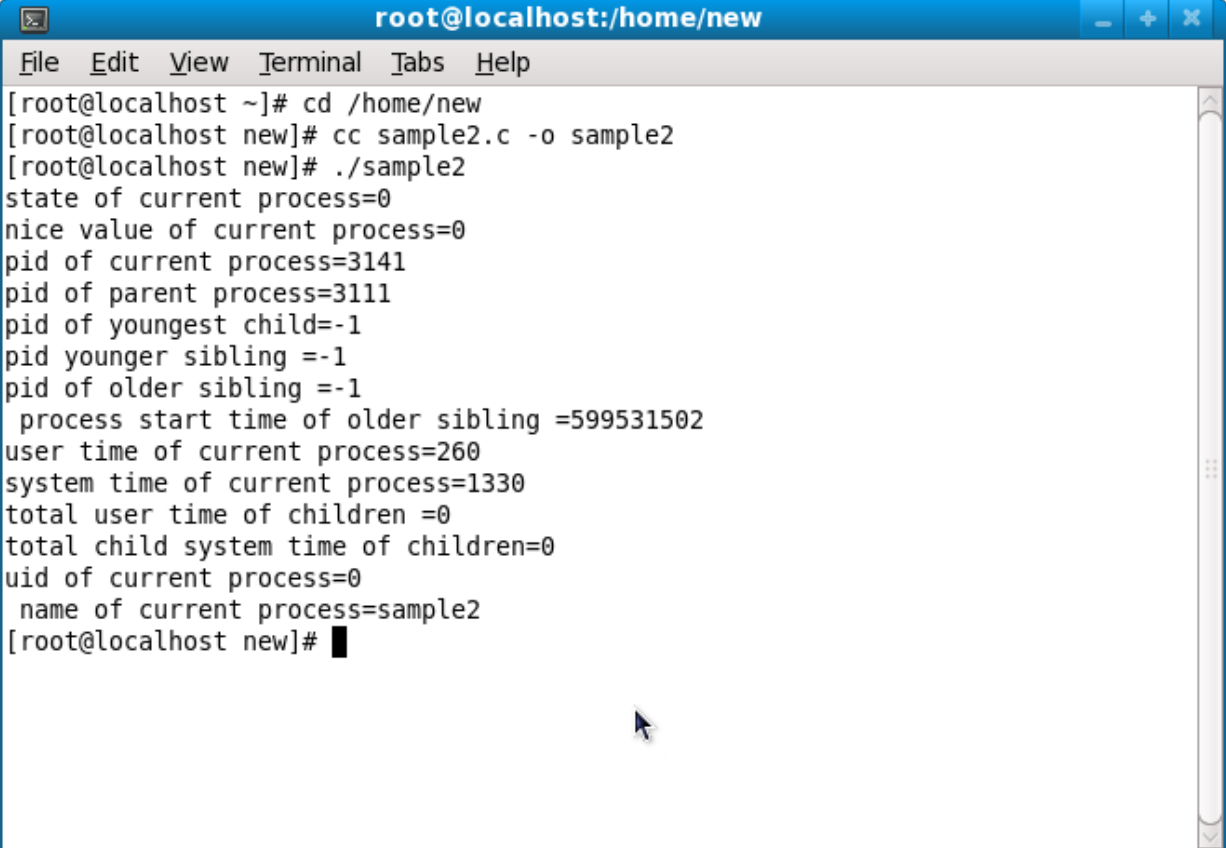
The compiled object files are now stored in /home/name/build/kernel directory. Make sure that the corresponding directory exists before running the make command, else create these directories or specify some existing directory.

7) After rebooting the computer from the new boot image of linux kernel 2.6.28, we created a user program "sample2.c" which calls the syscall "getprocessinfo()" added by us to the kernel.

When we compiled and ran the program

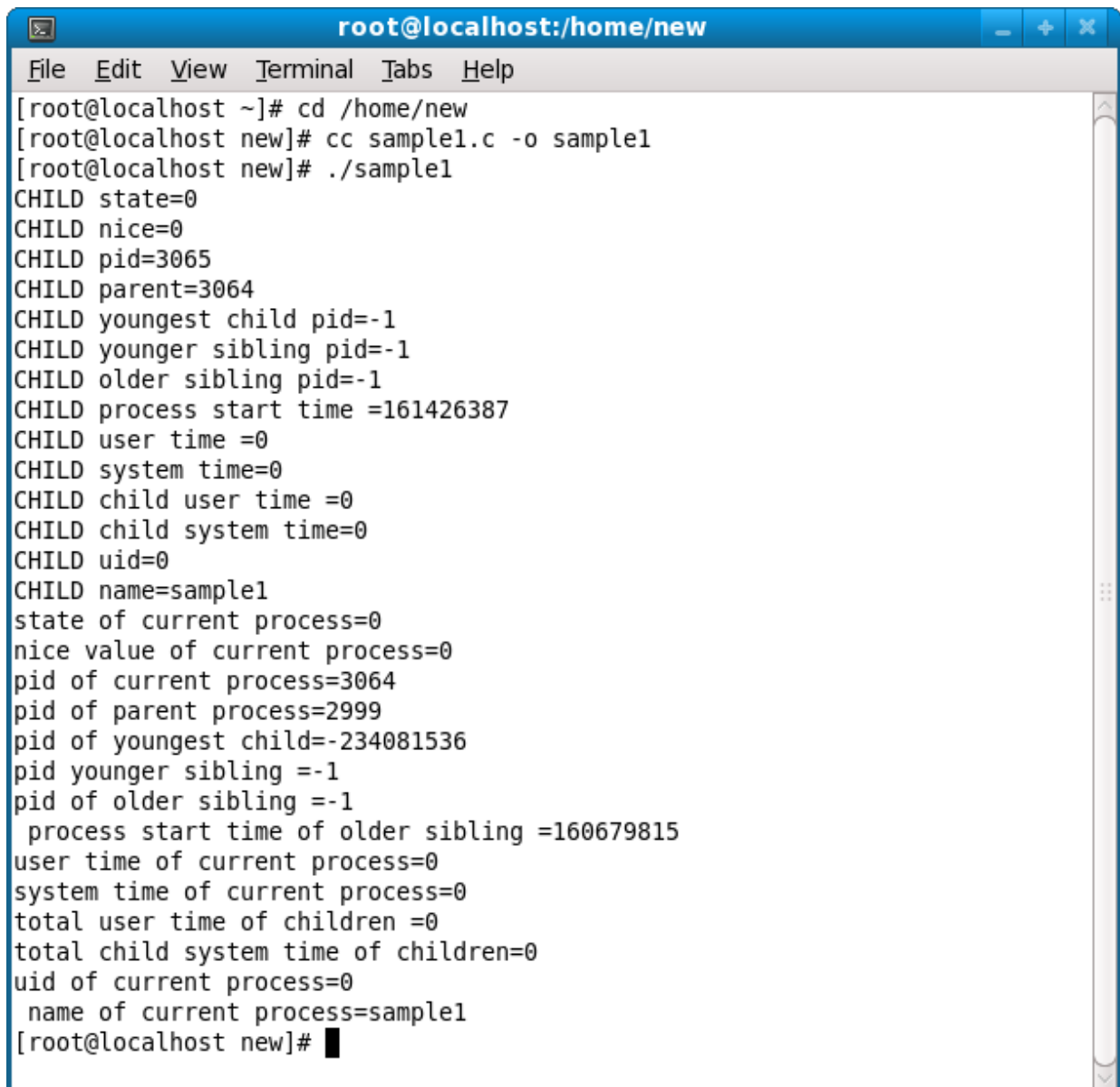
We got the following output

If the process does not have a child or a sibling process, it returns -1 in the corresponding field as shown in the screenshot:

A screenshot of a terminal window titled "root@localhost:/home/new". The terminal shows the execution of a program called "sample2". The output of the program is as follows:

```
[root@localhost ~]# cd /home/new
[root@localhost new]# cc sample2.c -o sample2
[root@localhost new]# ./sample2
state of current process=0
nice value of current process=0
pid of current process=3141
pid of parent process=3111
pid of youngest child=-1
pid younger sibling =-1
pid of older sibling =-1
process start time of older sibling =599531502
user time of current process=260
system time of current process=1330
total user time of children =0
total child system time of children=0
uid of current process=0
name of current process=sample2
[root@localhost new]#
```

Next we create a child process using CLONE in another user program sample2.c and we got the following output:

A terminal window titled 'root@localhost:/home/new' with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows the execution of a C program 'sample1.c' which prints detailed process information. The output includes fields like CHILD state, nice, pid, parent, youngest child, siblings, start times, and user/system times for the current process and its children. The prompt returns to '[root@localhost new]#'.

```
root@localhost:/home/new
File Edit View Terminal Tabs Help
[root@localhost ~]# cd /home/new
[root@localhost new]# cc sample1.c -o sample1
[root@localhost new]# ./sample1
CHILD state=0
CHILD nice=0
CHILD pid=3065
CHILD parent=3064
CHILD youngest child pid=-1
CHILD younger sibling pid=-1
CHILD older sibling pid=-1
CHILD process start time =161426387
CHILD user time =0
CHILD system time=0
CHILD child user time =0
CHILD child system time=0
CHILD uid=0
CHILD name=sample1
state of current process=0
nice value of current process=0
pid of current process=3064
pid of parent process=2999
pid of youngest child=-234081536
pid younger sibling =-1
pid of older sibling =-1
  process start time of older sibling =160679815
user time of current process=0
system time of current process=0
total user time of children =0
total child system time of children=0
uid of current process=0
  name of current process=sample1
[root@localhost new]#
```

For reference, the user test program sample.c, the user space header file prinfo.h and the makefile is present in the directory USER.

Also the kernel space file prinfo.c and the kernel space header file prinfo.h is present in the directory KERNEL.

The comments have been provided in the programs to explain the function of each line and the value being returned.

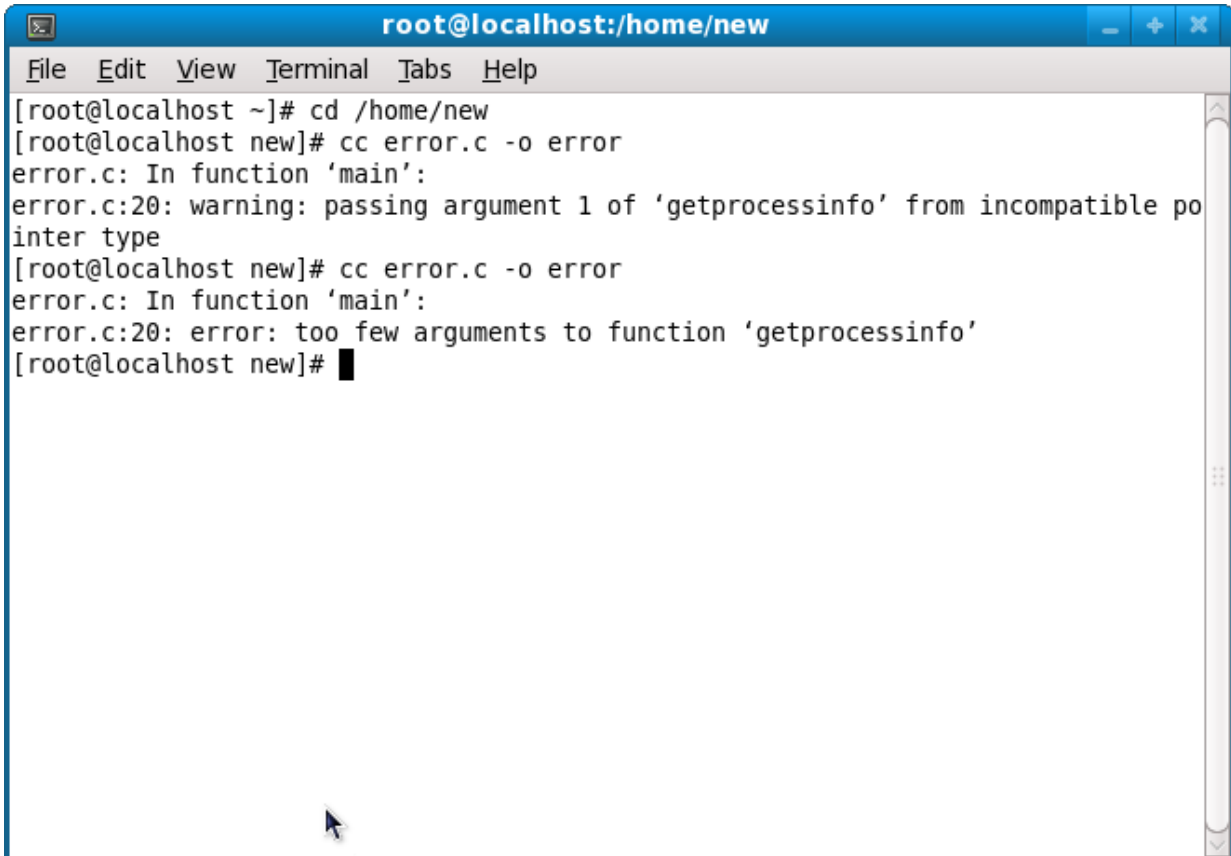
To run the user test program, enter into the directory USER through terminal and type

make

./sample

TEST RESULTS:

- 1.) If the pointer argument to the system call was null or invalid, the user program gave errors while compilation. The screen shot is shown below:



```
root@localhost:/home/new
File Edit View Terminal Tabs Help
[root@localhost ~]# cd /home/new
[root@localhost new]# cc error.c -o error
error.c: In function 'main':
error.c:20: warning: passing argument 1 of 'getprocessinfo' from incompatible po
inter type
[root@localhost new]# cc error.c -o error
error.c: In function 'main':
error.c:20: error: too few arguments to function 'getprocessinfo'
[root@localhost new]#
```

- 2.) The following observations were made when the user program was run several times:

- a) STATE: Since the current process is always in running state, so it always returned 0 which signifies RUNNING state.
- b) NICE: By default, the nice value is 0. Unless this nice value is changed using a system call, it always returned 0.
- c) PID: Different values were returned every time since a different pid is assigned to every process each time it is run.
- d) PARENT_PID: Different values were returned every time since a different pid is assigned each time it is run.
- e) YOUNGEST_CHILD_PID: Different values were returned every time since a different pid is assigned each time it is run. In case no child exists, it returns -1.
- f) YOUNGER_SIBLING_PID: Different values were returned every time since a different pid is assigned each time it is run. In case no sibling exists, it returns -1.

- g) YOUNGER_SIBLING_PID: Different values were returned every time since a different pid is assigned each time it is run. In case no sibling exists, it returns -1.
- h) STAT_TIME: Different values were returned every time. An important point to note is that the value returned is always greater than the value returned when run previously.
- i) USER_TIME: This value varied some of the times.
- i) SYS_TIME: This value varied some of the times.
- i) CUTIME: This value varied some of the times.
- i) CETIME: This value varied some of the times.
- i) UID: It returned the user id. So for a particular user, the value was always same. For example, when we signed in as root, the value was always 0 and when we signed in as another user, the value returned was 500.
- i) COMM: since it returned the name of the current process, for a particular program, it always returned the same value. For example, if the program is sample.c, it returns sample.