Operating System Design & Implementation

TA: 周昆霖 吳崧銘

Lab 7: Creating a kernel thread and hiding a process

Objective:

In this Lab, you will learn

* Task structure
* Kernel thread creation
* Task structure modifications to hide your process from ps/top

Experiment:

* 7-1: create a kernel thread
* 7-2: hide a process

Exp. 7-1 create a kernel thread

* You are requested to create a kernel thread by revising /usr/src/linux-2.6.32-60/init/main.c

1. Create a kernel thread and invoke the following function

printk(“hello\n”);

<hint>

1. Use kernel\_thread(int (\*fn)(void \*), void \*arg, unsigned long flags) to create a new kernel thread . It receives the parameters: the address of the kernel function to be executed (fn), the argument to be passed to that function (arg), and a set of clone flags (flags).

Ex:

kernel\_thread(my\_thread , NULL , CLONE\_KERNEL);

Flag CLONE\_KERNEL: share the file system, file and signals with parent process.

1. Implement your kernel function my\_thread:

*static int my\_thread(void \* n){*

*daemonize(“my\_thread”);*

*printk(“hello\n”);*

*return 0;*

*}*

Notable daemonize() function is generally used to convert ordinary processes into kernel threads.

iii. Be sure to add *static int my\_thread(void \*);* to the top line of

main.c

1. Your kernel thread pid should be 2

<hint>

1. Please refer to /usr/src/linux-2.6.32-60/init/main.c for how the first process: init (pid = 1) started.

Exp. 7-2 hide a process

* Modify the process ***task\_struct*** in your kernel

thread to hide your process from ps/top

1. Compile and run the program (loop.c).
2. Implement a kernel thread in kernel module yourthread.c

<hint>

* 1. Implement static int yourthread(void \* data){

While(1){

Step1 : traverse all the process

Step2 : if the process’s name = “loop” then change

this process’s pid to other number

…

set\_current\_state(TASK\_INTERRUPTIBLE);

schedule\_timeout(10\*HZ);

}

}

* 1. Q: Why changing pid hide the process information for ps/top??

A: The kernel derives the process descriptor pointer

from PID . If we change the pid, kernel won’t be able to

find the correct process descriptor.

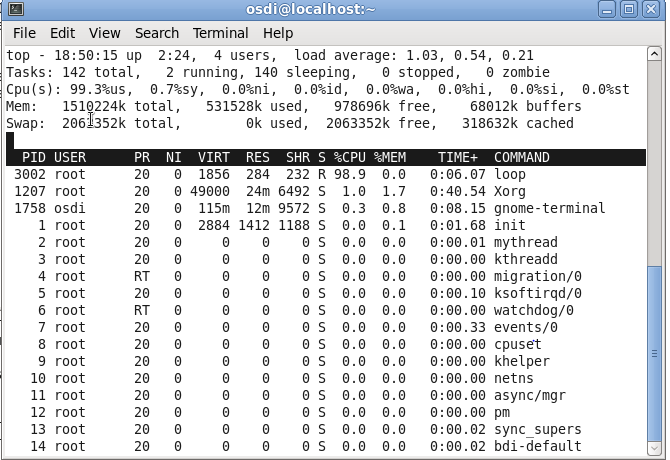
* 1. set\_current\_state(TASK\_INTERRUPTIBLE) will force the

process to be in the sleep state.

* 1. schedule\_timeout(10\*HZ) is to wake up the thread after 10\*HZ;

Notable HZ= timer interrupts /second

The result will be:



After loading module:

