OS Lab
Session 8: Kernel Programming

AUT – CEIT

Instructor: @MerajNouredini

Adding a system call to kernel

Defining our system call

- cd <downloaded-kernel-src>
- mkdir hello
- cd hello/
- Create a header file ‘helloCall.h’
- Include the following line in the header file:
  - asmlinkage long sys_hello<your name>(void);
  - asmlinkage tells the compiler to look at the CPU’s stack for the function parameters, and, long is generally used as a return type in kernel space for functions that return an int in user space.

- Now, let’s define our system call in ‘hello<your name>.c’.
- Write a Makefile in the same directory(i.e., hello/) with the following contents:
  - obj-y:=hello<your name>.o

Modifying necessary kernel files to integrate our system cal

- Add the new ‘hello’ directory to the kernel’s Makefile:
- nano <downloaded-kernel-src>/Makefile
  - core -y += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ hello/
- find and modify syscall_64.tbl
  - find -name syscall_64.tbl
  - In kernel 4.7.1, it is present in /arch/x86/entry/syscalls/syscall_64.tbl

- add this line at the end of 3## lines:
  - 3##<tab>common<tab>hello<your name><tab>sys_hello<your name>
- e.g:
  - 333   common   helloMeraj sys_helloMeraj
- find -name syscalls.h
  - in kernel 4: /include/linux/syscalls.h
- Add the following line to the end of the file (before the #endif)
  - asmlinkage long sys_hello<your name>(void);

recompile the kernel and reboot

- sudo make -j 4 && sudo make modules_install -j 4 && sudo make install -j 4
- sudo shutdown -r now

test

- To test the system call write a simple ‘test.c’
- If it runs successfully, then, it should give the corresponding prompt and you can now use ‘dmesg’ to check the kernel log and actually verify if the process information has been logged.

- Note: The system call implemented does not take care of privilege checks, does not return any error codes on failure and does not do anything particularly useful for the user. So, it is actually far from being a well-designed system call!

Introduction to modules

What exactly is a kernel module?

- Modules are pieces of code that can be loaded and unloaded into the kernel upon demand. They extend the functionality of the kernel without the need to reboot the system.

How Do Modules Get Into The Kernel?

- When the kernel needs a feature that is not resident in the kernel, the kernel module daemon kmod execs modprobe to load the module in. modprobe is passed a string in one of two forms:
  - A module name like softdog or ppp.
  - A more generic identifier like char-major-10-30.
  - If modprobe is handed a generic identifier, it first looks for that string in the file /etc/modprobe.conf

- Next, modprobe looks through the file /lib/modules/version/modules.dep, to see if other modules must be loaded before the requested module may be loaded.
- Lastly, modprobe uses insmod to first load any prerequisite modules into the kernel, and then the requested module.

loading module

- Example:
  - insmod /lib/modules/2.6.11/kernel/fs/fat/fat.ko
  - insmod /lib/modules/2.6.11/kernel/fs/msdos/msdos.ko

    or
  - modprobe msdos

let’s write a hello world kernel module

- Installing the linux headers:
  - apt-get install build-essential linux-headers-$(uname -r)
- Hello World Module Source Code
- Create Makefile to Compile Kernel Module
- Compile the module
  - make
- Insert or Remove the Sample Kernel Module
  - insmod hello.ko & rmmod hello.ko
- See the output: dmesg | tail -1

Questions?