The American University in Cairo
School of Sciences and Engineering
Department of Computer Science and Engineering

CSCE 345 / 3401 – Operating Systems
Prof. Amr El-Kadi

Project II

Due 13th of May, 11:59 PM

In this project, you will study the internal process data structures (task_struct), different system process queues, system call interface provided by the Linux operating system and how user programs communicate with the operating system kernel via this interface. Your task is to 1) access process queues and report on who is listening on which communication ports, and 2) incorporate a new system call into the kernel that would ignite data collection from relevant process queues, and 3) write a simple application that tests the use of your newly incorporated system call.

Useful Linux and C Resources

To help you start with Linux, we have compiled a list of Linux resources that would help take you from novice stage to Kernel expert (as needed). The list is the work of the very large community of people who have worked with the Linux kernel.

For Starters

- Kernel Newbies: Intended to help new kernel hackers
- Linux Questions: A civil forum asking Linux questions
- Ubuntu Support Forums: From the best-known consumer distribution of Linux, including a section for absolute beginners

Kernel News

- Linux Weekly News (LWN.net)

Kernel Hacking

- Here is where you can get any kernel you want: Linux Kernel Archives at kernel.org
- The Linux Kernel Documentation Project
- The Linux Kernel by David A Rusling
- Linux Kernel Internals by Tigran Aivazian
- The Linux Kernel Hackers' Guide: HTML

Linux Kernel Debuggers

- KDB: built-in kernel debugger with multiprocessor support
- Kgdb: kernel debugger used through serial line

Source Navigators

The source navigators provide browse-able kernel code, making it easier to read than your ssh window or emacs. However, the kernel is not necessarily the same version as the kernel you will be working with, so watch out for subtle differences.

- Linux Source Navigator
- Another Linux Source Navigator

C Language Resources

As Linux is written mostly in C, if you are unfamiliar with the language, we provide you with few links that will jump-start your C capabilities in no time.

- C programming.com
- ANSI C On Unix Systems
- Common C Problems
- C Frequently Asked Questions
- This includes a tutorial and full reference to C: Programming In C: Unix System Calls and Subroutines in C

Step Zero: Building Your Kernel

Before getting into the requirements of a kernel project, you must familiarize yourself with the task of building the binary for a kernel from its source code and booting the machine with the newly built kernel. Depending on the Linux distribution you have installed (I personally prefer Ubuntu), this activity comprises the following tasks with specifics being distribution dependent:

- Make sure that you have obtained and installed the kernel source code for the Linux distribution. If the source code package has been previously installed on your machine, the corresponding files might be available under /usr/src/linux or /usr/src/linux-2.x (where the suffix corresponds to the kernel version number). If the package has not been installed earlier, it can be downloaded from the provider of your Linux distribution or from http://www.kernel.org.
- Learn how to configure, compile, and install the kernel binary in accordance with your Linux Distribution. Typical commands for building the kernel (after entering the directory where the kernel source code is stored) include (for an explanation of each step, refer to your distribution documentation):

```
make xconfig
make dep
make bzImage
```

- Add a new entry to the set of bootable kernels supported by the system. Most probably the boot loader you will be using is grub. You want to make sure that you add a new entry to grub's configuration files that represents your newly generated kernel image. Note that care should be given to having at least one working kernel image to boot to in emergencies!

Getting Started

A user-mode procedure call is performed by passing arguments to the called procedure either on the stack or through registers, saving the current state and the value of the program counter, and jumping to the beginning of the code corresponding to the called procedure. The process continues to have the same privileges as before.

System calls appear as procedure calls to user programs, but result in a change in execution context and privileges. In Linux on the Intel 386 architecture, a system call is accomplished by storing the system call number into the EAX register, storing arguments to the system call in other hardware registers, and executing a trap instruction (which is the INT 0x80 assembly instruction). After the trap is executed, the system call number is used to index into a table of code pointers to obtain the starting address for the handler code implementing the system call. The process then jumps to that address and the privileges of the process are switched from user to kernel mode implicitly.

The system call numbers for recent versions of the Linux kernel are listed in /usr/src/linux-2.x/include/asm-i386/unistd.h. The list of pointers to system call handlers is typically stored in the file /usr/src/linux-2.x/arch/i386/kernel/entry.S under the heading ENTRY(sys call table). (The keyword .long denotes that the entry will occupy the same number of bytes as a data value of type long.)

Your System Call

Implement a new system call that returns the returns collected information on processes listening to communications ports. Pick a suitable signature for your system call and give it an unused number. The system call should log each of its invocations to the system log (check the above resources for further details.)

Primary Task

Typically, system administrators and experienced users may wish to know which process is listening to which communication port. There are a lot of system utilities that would report on that such as netstat, fuser, lsof, etc. In addition, under Linux, the /proc file system provides such information under **/proc/$pid/ file system** which includes a directory for each running process at /proc/PID, containing information about that process including the processes name that opened port. Your job is to instrument the Linux kernel with code that will access process queues and report on who is listening on which communication port.

You should divide your tasks into smaller ones. Start by getting acquainted with the kernel data structures and how Linux treats communication ports, how they are linked to the resources used by task_struct. Try to report on such findings by instrumenting the kernel with proper printk calls to report on findings. Try and save such findings in a buffer. Add a simple system call to Linux and upon success decide on how the saved buffer content would be transferred back to the user space. Adjust your code and system call to ignite the data collection and reporting back. In essence, work on your project step by step and then add more functionality.

User-space Memory Access

- Validating read or write access to user memory block:

```
access_ok(type, addr, size)
```

  Returns true (non-zero) if access is allowed, otherwise returns false (0). type should either be VERIFY_READ or VERIFY_WRITE. addr is a pointer to the first byte of the memory block to be tested. size is the size, in bytes, of the memory block to be tested. For Example:

```
if (!access_ok(VERIFY_READ, ptr, sizeof(double)))
   return -1;
```

- Copying a memory block from user space to kernel space:

```
copy_from_user(to_ptr, from_ptr, size)
```

  Copies size bytes from user space memory block beginning at from_ptr to kernel space memory block beginning at to_ptr. Returns number of bytes that could not be copied. On success, this will be 0.

- Copying a memory block from kernel space to user space:

```
copy_to_user(to_ptr, from_ptr, size)
```

  Copies size bytes from kernel space memory block beginning at from_ptr to user space memory block beginning at to_ptr. Returns number of bytes that could not be copied. On success, this will be 0.

What You Should Turn In

A detailed technical report is needed to document your project and the roles played by each member of the group. Along with the sources, send the report to the GTA on or before the due date. You will need to schedule an appointment with the GTA for a demo and questioning (refer to your GTA for further instructions).