Operating System Homework 1

Jinyan Xu, 3160101126, Information Security

1. Operating System Concept Chapter 1 Exercises: 1.14, 1.17, 1.19, 1.22.

Answer:

1.14) What is the purpose of interrupts? How does an interrupt differ from a trap? Can traps be generated intentionally by a user program? If so, for what purpose?

Interrupts are used to handle asynchronous events and to trap to supervisor-mode routines in the kernel.

An interrupt is the hardware mechanism that enables a device to notify the CPU, while a trap is a software-generated interrupt, caused either by an error or a user request.

Traps can be generated intentionally by a user program, the example is the implementation of system calls.

Traps are used to handle synchronous events, like errors, exceptions and user requests.

1.17) Some computer systems do not provide a privileged mode of operation in hardware. Is it possible to construct a secure operating system for these computer systems? Give arguments both that it is and that it is not possible.

Possible: In the absence of a privileged mode protecting system, it is necessary to ensure that every program does not act against system security. Therefore, a strong compiler is required to regulate the behavior of the program when generating the program, and the system should only run the programs generated by the compiler.

Impossible: Due to the lack of privileged mode, these may occur when programs at different hierarchy run on the system, a user program running away can wipe out the operating system by writing over it with data, and multiple programs are able to write a device at the same time, with possibly disastrous result.

1.19) Rank the following storage systems from slowest to fastest:

a. Hard-disk drives
b. Registers
c. Optical disk
d. Main memory
e. Nonvolatile memory
f. Magnetic tapes
g. Cache

Magnetic tapes, Optical disk, Hard-disk drives, Nonvolatile memory, Main memory, Cache, Registers.

1.22) Describe a mechanism for enforcing memory protection in order to prevent a program from modifying the memory associated with other programs.

When the program runs, the CPU can track the address of the program and make an offside check before reading instructions and writing data to avoid segment errors.

2. Install ubuntu 16.04 and capture a screenshot of the system information.

Answer:

```
phantom0308@phantom0308-VirtualBox:~$ uname -a
Linux phantom0308-VirtualBox 4.15.0-29-generic #31~16.04.1-Ubuntu SMP Wed Jul 18 08:54:04 UTC 2018 x86_64 x86_64 x86_64
 GNU/Linux
phantom0308@phantom0308-VirtualBox:~$ cat /proc/version
Linux version 4.15.0-29-generic (buildd@lcy01-amd64-024) (gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.10))
#31~16.04.1-Ubuntu SMP Wed Jul 18 08:54:04 UTC 2018
phantom0308@phantom0308-VirtualBox:~$
```

From the results of operations, I use VirtualBox to mount Ubuntu 16.04 and its kernel version is 4.15.

3. Compile and run the code demonstrated in the class to illustrate the concept of CPU/Memory virtualization and concurrency.

a) Compile and run the code, as shown in the OSTEP (chapter II). Show the results of these programs.

Answer:

cpu.c:

```
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master$ gcc cpu.c -o mycpu -lpthread
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master$ ./mycpu A & ./mycpu B & ./mycpu C &./mycpu D &
[1] 2212
[2] 2213
[3] 2214
[4] 2215
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master$ C
B
D
A
B
C
D
A
D
B
C
A
```

mem.c:

```
root@phantom0308-VirtualBox:/home/phantom0308/桌面/ostep-master# echo 0 > /proc/sys/kernel/randomize_va_space
root@phantom0308-VirtualBox:/home/phantom0308/桌面/ostep-master# exit
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master$ gcc mem.c -o mymem -lpthread
mem.c: In function ‘main’:
mem.c:11:58: warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]
     printf("(%d) memory address of p: %08x\n", getpid(), (unsigned) p);
                                                          ^
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master$ ./mymem & ./mymem &
[1] 2263
[2] 2264
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master$ (2263) memory address of p: 00603010
(2264) memory address of p: 00603010
(2264) p: 1
(2263) p: 1
(2263) p: 2
(2264) p: 2
(2264) p: 3
(2263) p: 3
```

threads.c:

```
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master$ gcc threads.c -o mythread -lpthread
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master$ ./mythread 100000
Initial value : 0
Final value : 200000
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master$ ./mythread 100000000
Initial value : 0
Final value : 198165208
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master$
```

b) Describe what you have learnt from running these programs.

Answer:

cpu.c: Using the technique for virtualizing the CPU, known as timesharing (multitasking), allows users to run as many concurrent processes as they would like.

mem.c: These two programs can run in the same address of memory because of the use of virtual memory, the running program thinks it is loaded into memory at a particular address and has a potentially very large address space. The reality is that physical memory is a shared resource, managed by the operating system.

threads.c: This problem is called concurrency, the reason for concurrency is the simultaneous operation of shared memory by multithreading.

c) Whether you can get the same results when running the second program (mem) as you have seen in the class (or in the book). And why?

Answer:

No, this is the initial result:

```
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master$ ./mymem & ./mymem &[1] 2360
[2] 2361
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master$ (2361) memory address of p: 024ae010
(2360) memory address of p: 00b9a010
(2361) p: 1
(2360) p: 1
(2361) p: 2
(2360) p: 2
(2360) p: 3
(2361) p: 3
```

As we can see, the memory of the two program is different, this is because the concept of the address space layout randomization (ASLR). In order to make this experiment successful, we need to make sure address-space randomization is disabled.

The ASLR in Linux is divided into 0, 1, 2 levels, users can use an integer parameter to control the level in “randomize_va_space” file, 0 means address-space randomization is disabled, while 1 & 2 is open.

I used the command: echo 0 > /proc/sys/kernel/randomize_va_space, then I got the result in 3.a.

From this experiment, I realize that ASLR is a good defense against certain kinds of security flaws by randomly arranging the address space positions of key data areas of a process, including the base of the executable and the positions of the stack, heap and libraries.