Assignment 1 Answers

Part 1:

1) The “task_struct” is defined in linux_source/include/linux/sched.h on line 637

2) The size of a task_struct is 7040 bytes.

3) Each platform has its own system call table in linux_source/arch/<platform name>/. For example, the desktop processor system call tables are defined in linux_source/arch/x86/. The 32 bit architecture system call table is defined in linux_source/arch/x86/um/sys_call_table_32.c and the 64 bit architecture system call table is defined in linux_source/arch/x86/um/sys_call_table_64.c.

4) “do_syscall_64” is called from in assembly from line 175 of linux_source/arch/x86/entry/entry_64.S.

5) Linux 5.3 figures out if it is running in a virtual machine by using the following process [1]:
    a. setup_arch is running in linux_source/arch/x86/kernel.setup.c
    b. setup_arch calls a function called “init_hypervisor_platform”, which is defined in linux_source/arch/x86/kernel/cpu/hypervisor.c
    c. init_hypervisor_platform then calls another function in the same file called “detect_hypervisor_vendor”
    d. detect_hypervisor_vendor loops through a hard-coded global array of “hypervisor_x86” structures (found in linux_source/x86/include/asm/hypervisor.h), which contain the hypervisor name and pointers to important functions; one of these functions is called “detect”, which is responsible for determining if the kernel is executing in its virtual machine environment. The loop calls the detect function of each hypervisor_x86 structure. If the detect function succeeds, then the pointer to the hypervisor is saved and will be returned once the loop exits.
    e. When the detect method of a VM struct is called, the function would get the signature of the CPU, and check if it matches a specific string associated with VM. If the signature matches the string, then the function would return true, which means that Linux is in a VM. Otherwise it would return false.

6) “current” is defined as a macro in the x86 architecture. Its definition is in linux_source/arch/x86/include/asm/current.h.

7) The purpose of the “current” macro is to make easier and cleaner calls to the “get_current()” function. It is defined this way because “get_current()” is a very common operation (searching through the x86 architectures code shows it called in

many different places) and the macro makes it cleaner and easier to read without any overhead (since all occurrences of “current” are just replaced with “get_current()” at compile time).

8) “module_init” defines which function is called at time of module insertion and boot, whereas “module_exit” defines which function is called when a module is unloaded.

9) The two system calls which handle module loading are “init_module” and “finit_module” [2]. The main difference between those functions is that “init_module” loads its module from a given buffer, whereas “finit_module” loads its module from a file descriptor [3].

10) The “delete_module” system call is used to unload kernel modules [2]. The five distinct error codes which this function can return are [4]:
    1) EBUSY: The module is in the process of being loaded or unloaded
    2) EFAULT: The pointer argument to the module name contains an invalid address
    3) ENOENT: No module with the given name argument is loaded
    4) EPERM: The process making this system call does not have the required privilege level, or the kernel does not allow module unloading
    5) EWOULDBLOCK: There are other running modules which depend on the module to be removed; or O_NONBLOCK was set in the flags argument, O_TRUNC was not set in the flags argument, and reference count inside the module is greater than zero.

11) According to the “module_state” enum in linux_source/include/linux/module.h, a module can have 4 states:
    1) MODULE_STATE_LIVE
    2) MODULE_STATE_COMING
    3) MODULE_STATE_GOING
    4) MODULE_STATE_UNFORMED

3.1):

The call stack for do_one_initcall is:

1) entry_SYSCALL_64
2) do_syscall_64
3) __do_sys_finit_module
4) load_module
5) do_init_module
6) do_one_initcall

3.2)

All the module parameters are stored at /sys/module/<module name>/parameters/. The module creates these files at startup, and then insmod writes the supplied arguments into them. You can also modify those parameters at runtime if you make them writable in the permissions [5].

3.3.1)

The “insmod” operation is killed by the kernel, and the “dmesg” log is filled with information about the crash:

```
[  238.372438] BUG: kernel NULL pointer dereference, address: 0000000000000000
[  238.372465] #PF: supervisor write access in kernel mode
[  238.372479] #PF: error_code(0x0002) - not-present page
…
```

Followed by system information (CPU, specs, etc.), callstack, registers, etc.

3.3.2)

The “lsmod” command shows the module in its list, and it shows the module has been used once:

```
Module                  Size  Used by
broken_module          16384  1
```

3.3.3)

When we try to unload the module, we get an error saying the module is in use.

Build instructions:

Each module directory contains a makefile. To build our modules, you first need to ensure that the "dir" folder defined in each of those directories points to correct header path in your machine.

Then you can simply do "make" in each folder to build each module. The "simple_module" and "broken_module" have "make load" and "make unload" commands to make loading and unloading easier, but "simple_args" does not because it uses parameters and hence needs to be loaded manually.

The 2 flags used by "simple_args" are:

1) print_greeting - a boolean value flag. Ex. sudo insmod simple_args.ko print_greeting=0

2) message - the string which the module will print. Ex. sudo insmod simple_args.ko message='"This is my message"' (NOTE: both the single and double quotes are required!)

Bibliography

[1] M. Gebai, "Detecting virtualization: How to know if you're running inside a virtual machine," 13 June 2013. [Online]. Available: http://linuxmogeb.blogspot.com/2013/06/detecting-virtualization.html.

[2] M. Kerrisk, "syscalls(2) - Linux manual page," 2 August 2019. [Online]. Available: http://man7.org/linux/man-pages/man2/syscalls.2.html.

[3] M. Kerrisk, "init_module(2) - Linux manual page," 2 August 2019. [Online]. Available: http://man7.org/linux/man-pages/man2/init_module.2.html.

[4] M. Kerrisk, "delete_module(2) - Linux manual page," 2 August 2019. [Online]. Available: http://man7.org/linux/man-pages/man2/delete_module.2.html.

[5] The Linux Foundation, "The Kernel Newbie Corner: Everything You Wanted To Know About Module Parameters," 15 July 2009. [Online]. Available: https://www.linux.com/tutorials/kernel-newbie-corner-everything-you-wanted-know-about-module-parameters/.