Debugging Linux Kernel

Amir Modarresi

Prasanth Vivekanandan

Prof. Heechul Yun

Debugging Linux Kernel

- There are different ways to debug Linux kernel.
- The easiest way to debug kernel is using printk() function which you will see its usage in this lab.
- The other techniques is based on debuggers, which we will do it in this lab with GDB.

Debugging Linux Kernel with Printk()

- One easy way to debug the kernel is using printk() function. Printk() works nearly the same way as printf() in user mode. You can call this function anywhere in the kernel, even while a lock is held.
- The major difference between printf and printk is that, printk accepts loglevel. Linux uses loglevel to decide whether sends a message to output or not. The kernel shows all messages below the currently defined loglevel.
- In the next slide you see different values for loglevel

Debugging Linux Kernel with Printk()

Loglevel	Description
KERN_EMERG	An emergency condition; the system is probably dead
KERN_ALERT	A problem that requires immediate attention
KERN_CRIT	A critical condition
KERN_ERR	An error
KERN_WARNING	A warning
KERN_NOTICE	A normal, but perhaps noteworthy condition
KERN_INFO	An informational message
KERN_DEBUG	A debug message

If you don’t define the loglevel, kernel uses the default value which is currently KERN_WARNING

Debugging Linux Kernel with Printk()

- You can also identify loglevel by number, starting with <0> as the most important loglevel which is KERN_EMERG to <7> as the least critical loglevel, KERN_DEBUG.
- Examples:
  - Printk(KERN_WARNING “This is a warning message!\n”); or you can use
  - Printk(“<4> This is a warning message!\n);
  - Printk(KERN_ALERT “DEBUG: %s %d \n”, __SOMEVAR1__, __SOMEVAR2)
- Loglevels are defined in <linux/kernel.h>

The Log Buffer

- Kernel messages are stored in a circular buffer LOG_BUF_LEN which is configurable at the compile time with option CONFIG_LOG_BUF_SHIFT.
- Klogd is a daemon that retrieves the kernel messages from log buffer and sends them to the system log file via syslogd daemon.
- Syslogd provides two utilities for system logging and kernel message trapping. The syslogd daemon appends all the messages it receives to a file which is by default /var/log/messages.
- In order to see kernel messages you can use dmesg like
  - $ dmesg

Debugging with GDB

- You should be aware that debugging with gdb has some definite limitation, because a user space debugger peeks into the address space of a running kernel.
- You can examine the content of the kernel space
- But you can’t be able to
  - Set breakpoint or
  - Step through the kernel code

- There is another way to debug kernel with gdb using qemu which is explained in the next slide.

Debugging with gdb

- Run your KVM with the following command:
  - qemu-kvm –hda yourimage.qcow –m 512 redir tcp::[your_assigned_port]:22 –s –S
  - The first [s] is the lower case and the second one is uppercase
- This command let you connect into your booted machine over a port on the same machine.
- If in another terminal in the same machine you type the following command, you can connect to your virtual machine.
  - **> ssh -p[your-port] -l [username] localhost**

Debugging with gdb

- As an example, these commands connect the root user over the port 12345.

-
  ```
  amodarre@cycle1$ qemu-kvm -hda amir.qcow -m 512M -redir tcp:12345::22
  ```

-
  ```
  ...
  ```

-
  ```
  amodarre@cycle1$ ssh -p12345 -l root localhost
  ```
- Adding two options –s and –S causes the CPU freezes in the the beginning of the boot process and start gdb at port 1234. Boot process can be resume by pressing [c].

Debugging with gdb

- In order to start debugging, lunch gdb on the same machine that you have ran qemu-kvm.
- To debug the kernel, another file is necessary called vmlinux which is linux symbol table. If you have built the kernel correctly you should be able to find the file in the kernel folder.
- Copy vmlinux file to the host machine from the KVM using scp command.
- The following example shows the command to copy vmlinux file from kvm to your local machine at Download folder.
-
```
amir@eecs678-kvm:~$ scp /home/amir/kernel/linux-.6.32.60/vmlinux cycle2.eecs.ku.edu:~/Download/
```

Debugging with gdb

- After copying vmlinux file, you can launch gdb with the following format:
  - amir@eecs678-kvm:~$gdb <path to vmlinux>
- when you are inside the gdb run the following command to start debugging
  - (gdb) target remote:1234
- Now, you can set breakpoints and start debugging. If you press [c], gdb starts running.

Debugging with gdb

- If you don’t see anything in gdb, because you don’t have the code in your host machine. Copy the file that you want to debug to the host machine with SCP command to be able to see the code in gdb.

Debugging a kernel module

- All parts of the kernel are not loaded to the memory at the same time. Modules are loaded and unloaded based on their usage. if you need to debug such a module, you should acquire the symbol table related to the file.
- You can load a symbol table related to your module in gdb with add-symbol-file command. The format of the command is as below.
  - add-symbol-file FILE ADDR
  - where ADDR is the stating address of file’s text. You should be aware that, the file has already been loaded in the memory

Debugging a kernel module

- You can find the address of a file loaded in the memory from PROC file system in the /proc/module path. You should find the loading address of the module inside the file. Use that address in add-symbol-file command to load the symbol table in the gdb.

Conclusion

- In this lab, we learnt how to debug the Linux kernel. The easiest and fastest way is using printk function; however, gdb can be used as well.
- You learnt that gdb can’t be used the same way as we can use for debugging a simple application, because the operating system doesn’t let the user to debug a live kernel from user mode.
- Connecting remotely to the kernel is the way that is used for debugging a live kernel.