CSE451 13Sp Section 2 Notes

James Youngquist <jay@cs>

April 14, 2013

1 Linux Kernel

The following assumes you have your source unpacked in the directory LDIR.

1.1 Configuring / Compiling

The linux kernel as given does not *need* to be configured. But it's fun to look through all the options available to you. To run the configuration tool,

```
$ cd LDIR
$ make menuconfig
```

If you're using Qemu, you NEED to change a couple of options from modules to built-in. Modules requied for boot are stored in an initial RAM based filesystem (http://en.wikipedia.org/wiki/Initrd), which gives flexibility but adds unneeded complexity for our purposes. Thus, to run your kernel in Qemu, you must change the following options from modules to built-in:

```
Device Drivers
   -> Network device support
     -> Virtio network driver
    -> Ethernet driver support
      -> Intel PRO/1000*

File systems
  -> Second extended fs support
  -> Network File Systems
    -> NFS client support
       -> ALL suboptions
    -> Root file system on NFS
    -> NFS server support
      -> ALL suboptions

Networking support
  -> Networking options
    -> IP: kernel level autoconfiguration
      -> IP: DHCP support
      -> IP: BOOTP support
      -> IP: RARP support
```

which allows the kernel to boot without any external requirements.

After configuring, don't forget to run

```
$ make clean
$ make -j<# cores> bzImage
```

1.2 Relevant files

After you compile with MAKE BZIMAGE, several files are generated.

LDIR/.config This file is what MAKE MENUCONFIG changes. You can edit it directly if you like to change options.

LDIR/vmlinux An ELF "executable" that is the kernel. You can't actually run this, but you can examine it with OBJDUMP and READELF or load it into a GDB session.

LDIR/arch/x86/boot/bzImage The compressed kernel binary. This is the file you use to overwrite /BOOT/VMLINUZ-3.8.3-201.CSE451CUSTOM.FC18.X86_64

1.3 Introspection

The kernel provides several services that let you peak in on it's state.

1.3.1 printk

The simplest is using PRINTK. Any messages sent using PRINTK are displayed on the console and also stored in a kernel buffer so you can get to them later. The DMESG command will play back the kernel PRINTK buffer. For example

```
$ dmesg
....tons of messages scroll past your screen

# Send the output of dmesg through the 'less' pager
$ dmesg | less

# Search for a specific string
$ dmesg | grep some_string

# Save the output of dmesg to peruse at your leisure
$ dmesg > output_file.txt
```

1.3.2 /proc

The /PROC filesystem is a virtual file system that exports information primarily about processes. Each process has a PID (process ID), and a corresponding directory /PROC/PID that lets you examine state information about the process. For example, I have a BASH process running as process 8103,

```
# Show where the binary and shared libraries have been loaded into virtual memory
$ cat /proc/8103/maps

# Look at all the files it has open
$ ls -l /proc/8103/fd

# What limits have been placed on this process?
$ cat /proc/8103/limits
```

It's not inconceivably that you could export the execcount information here so it can be accessed by any language that can read a file, instead of being limited to the SYSCALL interface.

Proc on wikipedia – includes a nice list of what each file means.

1.3.3 /sys

The /SYS filesystem is a virtual file system that exports information primarily about devices and drivers from the kernel. It also lets you change parameters for the kernel at run time, either to tune performance, or enable/disable features (like IP forwarding).

It's not as pertinent to the kind of things we're doing in this course, but you could imagine having a "file" that you can write a 0 or 1 to, to toggle execcount instrumentation on a kernel-wide basis.

Sysfs on wikipedia

1.3.4 /sys/kernel/debug

The DEBUGFS is mounted as a subfolder of /SYS and is meant a free-for-all file system to use as you see fit for debugging the kernel. https://github.com/chadversary/debugfs-tutorial is a nice tutorial of how to do basic kernel-userspace communication via DEBUGFS. And http://www.linuxforu.com/2010/10/debugging-linux-kernel-with-debugfs/ is a bit longer of an explanation with links to further reading.