CSC 256/456: Operating Systems

QEMU and the Linux Kernel

John Criswell
University of Rochester

Outline

- Useful tools
- Compiling the Linux Kernel
- QEMU
- Linux Kernel Details

Useful Tools

screen

- Virtual console
- Allows you to leave a session running after logout
- screen : Starts a new screen
- screen -ls : Lists active screens
- screen -d -r : Re-connects to existing default screen
- screen -d -r <screen_name> : Re-connects to existing screen

cscope

- Source code browser
- cscope -R in top-level source code directory
- Available at http://cscope.sourceforge.net
- Allows you to find:
  - C structures by name
  - Functions
  - Functions called by/calling other functions
    - Doesn't handle indirect function calls

Revision Control Tools

- Tracks change that you make
- Permits you to revert to previous version
- Helps merge changes made by multiple developers
- Well-known tools
  - git
  - Subversion (svn)

Linux Kernel

Unpacking the Linux Kernel

- xzcat linux-2.6.32.63.tar.xz | tar -xvf -

Kernel Compile Overview

- make mrproper
- make menuconfig
- make

Configuring the Linux Kernel

- make mrproper: Required step
- make menuconfig : Configures the Linux kernel
- Load alternative configuration file csc256.config
- Exit and save the new configuration

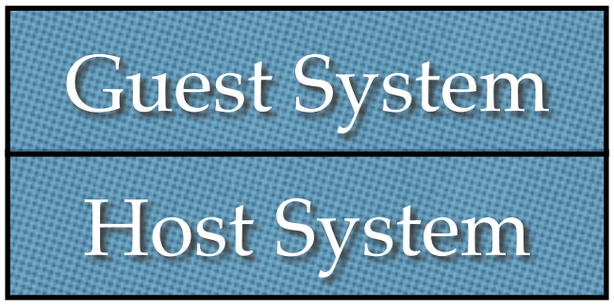
Compiling the Linux Kernel

- make
  - Compiles the Linux kernel
  - Use the -j option to compile files in parallel
- Kernel located in arch/x86_64/boot/bzImage
- Initial compile may take up to 2 hours
- Subsequent compiles should be much shorter (~2 mins)
- Writing header files will increase compile time

QEMU

About QEMU

- A processor emulator
- Emulates code for target CPU on host CPU



Other Potential VMs

- Emulators (i.e., slow but useful for debugging)
  - Bochs
- Fast solutions
  - KVM (part of Linux)
  - VMWare
  - VirtualBox
  - Hyper-V (Windows)

Why QEMU?

- Allow you to change the kernel
  - Privileged operation
- Isolate kernel changes from the real machine
- Make programming and debugging easier

Contents QEMU Package

- QEMU: QEMU executables and related files
- openwrt-x86-generic-combined-ext2.img: File containing hard disk image of OpenWRT Linux distribution

Quick Start

- Get the files you need at
  - http://www.cs.rochester.edu/courses/256/fall2014/QDGL/index.html
- Approximate space requirements
  - QEMU Image: 53 MB
  - Kernel source and object code: 4 GB
  - Everything: About 4 GB of disk space

Fire It Up!

- Start QEMU and default Debian installation
  - cd install
  - ./bin/qemu-system-x86_64 -m 64M -L ./qemu/share/qemu/ -curses -hda openwrt-x86-generic-combined-ext2.img
- No password
- Use halt command to shutdown
- Kill qemu from different window

Start Up a New Kernel

- ./bin/qemu-system-x86_64 -m 64M -L ./qemu/share/qemu/ -curses -hda openwrt-x86-generic-combined-ext2.img -kernel ~/bzImage -append "root=/dev/sda2 rootfstype=ext2”
- When QEMU boots, use uname -a to check if you booted the correct kernel image

Getting Files on to the VM

- Networking doesn't work (sorry!)
- No compiler, either (keeps image small)
- Make files look like hard disks
- Add -hdb <filename> to QEMU command line
- Boot the kernel
- dd if=/dev/sdb of=<FileName> bs=<FileSize> count=1

Mini Homework #1

- Build the Linux kernel
- Boot the QEMU image with the Linux kernel
- Report output of the procedure
  - uname -a
  - recipe from boot scripts

How to Train Your Linux Kernel

Adding a System Call

- Implement a function for your new system call

```
#include <linux/kernel.h>

asmlinkage long newsyscall (struct prinfo * p) {
    printk (“CSC256: Hi!\n”);
    return 0;
}
```

Adding a New System Call

- Add the function pointer in arch/x86/kernel/syscall_table_32.S
  - .long newsyscall /* 181 */
- Add the system call number in include/asm-x86/unistd_64.h
  - #define __NR_newsyscall 181
- Replace getpmsg with your system call in ./arch/x86/include/asm/unistd_64.h

```
#define __NR_newsyscall                          181

__SYSCALL(__NR_newsyscall, newsyscall)
```

Using Your New System Call

- Run the following program within QEMU

```
#include <linux/unistd.h>
#include <sys/syscall.h>

int main (int argc, char ** argv) {
    syscall (181, NULL);
    return 0;
}
```

Credits

- Previous semester project setup
  - Konstantinos Menychtas for previous QEMU setup
  - Hongzhou Zhao
  - Zhuan Chen
  - Li Lu
- Slides
  - Li Lu