Name: _____ **Student ID #:** _____

Lab #3 Solutions: COMP 3000 (Operating Systems)
October 23, 2007

1 Part A

1.1 Virtual Machine Emulators

1. **[2]** By using the `lspci` command, determine what PCI bus hardware is being emulated. Specifically, what video card does `QEMU` report to the operating system?

   Cirrus Logic GD 5446

2. **[2]** Compare the output of the `lspci` on the emulated vs the real system. What is one piece of hardware available on the real system that is not available on the emulated system?

   USB Controller, Sound Card, SMBus controller, FireWire controller

3. **[2]** One thing you will notice is that `QEMU` is not preserving changes to the VM's filesystem. (Verify this by killing and restarting `QEMU`.) What option is being given to the `qemu` program to keep changes from being permanant?

   -snapshot

4. **[2]** What options can you give to `QEMU` to have it use a second "hard disk" (disk image)?

   -hdb, -hdc, or -hdd

1.2 The Linux Kernel

1. **[2]** What command line can be given to configure the kernel with a random assortment of configuration options?

   make randconfig

2. **[2]** Using the configuration command `make menuconfig`, browse through the menus until you find the option for *Magic SysRq Key*. What menu is the option contained in?

   Kernel Hacking

3. **[2]** Enable support for the ISA SoundBlaster 16 card under ALSA (Advanced Linux Sound Architecture) and rebuild the kernel using `make`. Where did you find the option for the SoundBlaster 16 PnP ISA card?

   Device Drivers → Sound → ALSA → ISA Devices → SoundBlaster 16 (PnP)

4. **[2]** After renaming the old kernel `vmlinuz` to `vmlinuz.old`, copy the newly built kernel into your top-level lab directory and boot it. What file did you have to copy?

   arch/i386/boot/bzImage

5. **[2]** While booting, the kernel will now detect a sound card on the emulated system. The command `dmesg` will display the debug messages output while the system is booting. What IRQ is the SoundBlaster device tied to?

   5

2 Part B

2.1 Virtual Machine Emulators

1. **[5]** This lab used `QEMU` as a virtual machine emulator to allow us to build, modify, and debug a Linux kernel without having to reboot the real machine. Very briefly, how does a virtual machine emulator work (i.e. what does it do)?

   A virtual machine emulates (simulates) the behavior of the CPU and some generic I/O devices.

2. **[10]** Using the documentation for `QEMU` (and the documentation for `GDB`) (both of which are available online), answer the following questions:

   (a) **[2]** What function does the kernel hang in when you shut down your emulated kernel? To get the kernel into the *hung* state, run the kernel in the emulator, log in as root, and then type `halt`. Wait for the *System halted* message.

   default_idle

   (b) **[4]** What option did you have to pass into `qemu` to enable gdb debugging of the running kernel?

   -s to enable remote gdb debugging.

   (c) **[4]** What command did you have to give to the gdb command line to debug the kernel running in `qemu`?

   target remote localhost:1234

3. **[5]** By reading *QEMU, a Fast and Portable Dynamic Translator* by Fabrice Bellard [1], determine why QEMU is capable of running quickly compared to other emulators like Bochs[2].

   It recompiles the code to the current hardware dynamically instead of always interpreting each assembly instruction.

2.2 The Linux Kernel

In this section, you will modify the Linux kernel. If you end up breaking the kernel beyond repair, you can always retrieve a fresh copy from the `/lab-data` directory.

1. **[4]** What filesystems are mounted in the virtual machine, and what is the purpose of each of these? (Hint: there are more than two!)

[1] http://www.usenix.org/publications/library/proceedings/usenix05/tech/freenix/bellard.html
[2] http://bochs.sourceforge.net/

/dev/hda1	**Root Filesystem**
proc	**A pseudo-filesystem containing information about the state of the kernel and running processes.**
sysfs	**A pseudo-filesystem containing information and configuration files for various pieces of hardware attached to the running system.**
devpts	**A pseudo-filesystem allowing the kernel to create pts entries as required rather than having several hundred device nodes sitting around.**
tmpfs	**A filesystem where all contents reside only in memory and are not writen back to a physical device.**

2. **[10]** If you run the `df` command in the VM, you will notice that there is not much free space. Expand the virtual machine disk image so that it is at least 1 GiB in total size and contains all of the same files, with the same "last modified" timestamps on all important files.

   One way to do this is to create a new disk image 1 GiB in size, mount and format it in the virtual machine, and then copy all of the files over. (Hint: cp has options that preserve attributes and only copy one file system.)

   There are several steps required in completing this question:

   (a) The first step is to create a hard drive image under the real system. There are a few ways of doing this, but the easist are either using `dd` or `qemu-img`.

   Using `dd`, we copy the contents of `/dev/zero` to `new.img` until we have 1G worth of data. We can use a shortcut, however, and just copy the last few bytes. Doing that, our command is `dd if=/dev/zero of=new.img bs=512 count=1 seek=2097151`. If you don't take the shortcut, the command is `dd if=/dev/zero of=new.img bs=512 count=2097152`. Of course, you can use a different blocksize (bs) and adjust count respectively.

   Using `qemu-img`, our command line resembles `qemu-img create new.img 1G` (you can add more options, but they are not requried).

   (b) The next step is to partition our new hard drive image. This can be either done inside or outside the emulated environment. In both cases we use `fdisk`.

   Outside of the emulated environment, fdisk gets confused because we are asking it to partition a file not a device. We therefore have to set the number of cylinders manually. We run `fdisk -C 130 new.img`

   Inside the emulated environment, the next step is to boot the emulator with the new file set to be represented as a second hard drive. We also have to make sure to remove the `-snapshot` option from the command line to ensure that our changes are actually saved to the real `new.img` file. The command line for qemu needs to be modifed to contain the additional option `-hdb new.img`. After this, we log in and run `fdisk /dev/hdb`.

   Regardless of where we ran fdisk from, we partition the harddrive using *n* (the new command). We allocate the entire drive to one partition. We make it primary partition 1 (although we could make it any of the other partitions but it would affect later commands). We run *w* (Write changes) before exiting `fdisk` to ensure our changes are written to disk.

(c) After we have partitioned the hard drive, we create a filesystem on the drive. We could do this outside the emulated environment, but it's even more messy than before, since the filesystem can't start at the beginning of the drive (because that's where the partition table is). So, from this point on we will do things inside the emulated environment.

We boot up the emulated environment (as before) and run `mkfs.ext3 /dev/hdb1` to create an ext3 filesystem. To create an ext2 filesystem, we'd use `mkfs.ext2 /dev/hdb1` (but ext2 does not have journalling and hence is rather outdated).

(d) We mount our newly created filesystem in the emulated environment. We could either update `/etc/fstab` to let the mount command know about it, or just run mount with all the options directly. Running mount directly is easier. The command line is `mnt /dev/hdb1 /mnt` to mount it to the location `/mnt` directory.

(e) Now, we copy over all the files to the new filesystem.

```
cp -a -x / /mnt
```

(f) We modify the qemu command line to refer to the new harddrive image by replacing `-hda drive.img` with `-hda new.img` and remove the reference to `-hdb`.

3. **[10]** Modify the system kernel to print *Bye for now.* after displaying the *System halted* message. In addition, have it print the system state by calling `show_state`. The files *arch/i386/kernel/reboot.c* and *kernel/printk.c* may be useful. What function did you modify? What process is running?

   - **machine_halt is modified**
   - **the halt process is running.**