COMP150 Lab 4: The Boot Process

In this lab, you will get some experience with changing files related to the Linux boot process.

First up, as a good sysadmin you always want to be aware of what's happening when your servers are booting up. So, we need to modify the kernel options in the GRUB configuration to tell the kernel to show all the details during the boot process. Turn on your VM and log in like normal. Then:

1. Open a terminal. You will need to use sudo to get temporary root privileges.
2. Go to the GRUB configuration directory: `cd /boot/grub`
3. Edit the main GRUB configuration file, `grub.conf`. First copy the file to a backup, then edit it with your favorite editor.
4. Find the first `kernel` line, under the `CentOS (2.6.18-194.32.1.el5)` heading.
5. Go to the end of the line and remove the `rhgb` and `quiet` options.
6. Save the file and exit the editor.
7. **TAKE A SCREENSHOT** of the following command:
   `cat /boot/grub/grub.conf`
   a. You should see the `rhgb` and `quiet` options only in the second `kernel` entry, if you have one.
8. Reboot your VM: `shutdown -r now`

When the VM reboots, watch all of the information as it flies by on your screen. In particular, during the last phase of the boot process there will be a series of messages that should all end in `"[ OK ]"`. If you ever see a red `"[ FAILED ]"` message, then you should immediately follow up by looking at log files to determine what the actual problem is. After the VM finishes booting, log back in and open a terminal.

1. Run: `dmesg`
2. This prints out most of the messages from the last time the computer booted.

`dmesg` contains some very useful information about devices and drives connected to the computer. In particular, every hard disk and media drive (DVD, CD, usb, …) shows up on the system with a specific name. IDE drives show up as `"hd"` followed by a character starting at `"a"` first IDE drive 1. For example, `"hdc"` is the third IDE drive on the system. SCSI and SATA drives show up similarly except that they are `"sd"` followed by a character. The numbering of drives for `hd` and `sd` devices is separate. So, if you want to find all drives on your system, you should `grep` through the `dmesg` output and search for `hdX` and `sdX` where `X` is a normal alphabetic character. So:

1. Run `dmesg` and pipe it to `grep`.

2. You have to figure out what to `grep` for, so that a single command returns all lines of output that contain "`hd`" or "`sd`" followed by a character.
3. Hint: a single `grep` command is all that's needed, and the pattern contains three pieces (the stuff before the `d`, the `d`, and the stuff after the `d`).
4. You should get about 16 lines of output when done correctly.
5. **TAKE A SCREENSHOT** of your `dmesg`/`grep` command, showing the command itself and the output.

Next, let's change the hostname of your VM to something a little bit more interesting than "`localhost.localdomain`". The hostname is set via a configuration file in `/etc/sysconfig`. So:

1. In a terminal, become root: `su -`
2. Go to `/etc/sysconfig`.
3. Edit the `network` file.
4. There are three lines, and the last one sets the hostname.
5. So, edit the `HOSTNAME` line to be: `USERNAME.comp315`
    a. Obviously, change `USERNAME` to be your actually Tufts username.
    b. For example, my hostname line is now: `HOSTNAME=mhlepp.comp150`
6. As the file name (`/etc/sysconfig/network`) suggests, it is the `network` startup script that actually reads the file to set the hostname. In order for the change to take place, you need to stop and restart the `network` service (or reboot).
7. To stop the `network` service, run: `/etc/init.d/network stop`
8. To start the `network` service back, run: `/etc/init.d/network start`
9. Now, `exit` from your terminal window.
10. Open a new terminal and you should see that your shell prompt now includes `USERNAME@USERNAME` instead of `USERNAME@localhost`.

Note that the title of the new terminal window still says `USERNAME@localhost`. You have to log out completely and log back in (or reboot) for the change to take affect completely. Next, we're going to change the default run level. In your terminal, using sudo:

1. Edit the `inittab` file: `/etc/inittab`
2. Find the first non-comment line, which should look this: `id:5:initdefault:`
3. The `5` says that the default run level is `5` (multi-user with GUI login).
4. Change the `5` to `3` (multi-user with CLI login).
5. Reboot the VM: `shutdown -r now`
6. After the reboot completes, you will be at a command line login screen, rather than the GUI login screen. You should see a login prompt like `USERNAME login:`.
7. **TAKE A SCREENSHOT** of the CLI login screen.

This CLI environment is essentially the same as a single terminal window in the GUI environment. Log in with your normal user account and run a few commands to get a feel for it. Typically if you operate a machine that is used only for remote logins, you would set the default run level to 3 as no one should ever log in on the console directly unless something is broken. For now, though, we might as well change it back to run level 5. So, in the CLI, using sudo:

1. Edit `/etc/inittab`.
2. Change the default run level from `3` back to `5`.
3. Instead of rebooting again, use the `telinit` command to change the run level to 5 directly: `telinit 5`
4. Log back in like normal.

In the output of `ps`, any processes that have a name enclosed in brackets is a kernel process and not a normal system process. Sometimes you want to see those kernel processes alone and sometimes you want to see all the normal processes without the kernel processes cluttering up the `ps` output. In your terminal:

1. Run: `ps -ef`
2. Most of the kernel processes are near the top of the output, so scroll up to take a look.
3. The kernel processes all have a `[` (open bracket) in the name, so narrow down the `ps` output: `ps -ef | grep '\['`
4. Note that the backslash (`\`) before the open bracket in the `grep` pattern is necessary to tell `grep` to look for an actual open bracket, rather than a list of items.
5. Unfortunately, that command still includes a few other non-kernel processes like `init` and `avahi-daemon`. It also includes that pesky extra `grep` command.
6. Look at the output, and devise a way to extend the `grep` pattern so that it only includes actual kernel processes, and not `init` or any other processes that have brackets in the argument list. Also get rid of the extra `grep` command.
7. Hint: look at the column of data before the command name.

Now, take that command and make it into a script named `psk.sh`. Create the `psk.sh` script and put your `ps`/`grep` command in there so that you don't have to type out all that `grep` pattern every time you want to perform this task. Be sure that your script prints out ALL of the kernel processes and NONE of the non-kernel processes (including `init`).

Next, add an optional command line argument, `-v`, to your `psk.sh` script that causes the output to be the opposite. That is, if the command line argument is given when the script is run, it should print out ALL non-kernel processes (including `init`) and NONE of the kernel processes. Note that this should be a very simple change. The script should print out a usage line if more than one argument is given. Here are some sample runs (the … is not part of the actual output, I just shortened the output a bit here).

```
$ ./psk.sh blah blah
usage: ./psk.sh [-v]
$ ./psk.sh
root         2     1  0 14:47 ?        00:00:00 [migration/0]
root         3     1  0 14:47 ?        00:00:00 [ksoftirqd/0]
root         4     1  0 14:47 ?        00:00:00 [watchdog/0]
root         5     1  0 14:47 ?        00:00:00 [events/0]
root         6     1  0 14:47 ?        00:00:00 [khelper]
root         7     1  0 14:47 ?        00:00:00 [kthread]
root        10     7  0 14:47 ?        00:00:00 [kblockd/0]
root        11     7  0 14:47 ?        00:00:00 [kacpid]
root        48     7  0 14:47 ?        00:00:00 [cqueue/0]
root        51     7  0 14:47 ?        00:00:00 [khubd]
…
$ ./psk.sh -v
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 14:47 ?        00:00:01 init [5]
root       385     1  0 14:48 ?        00:00:00 /sbin/udevd -d
root      1520     1  0 14:48 ?        00:00:00 mcstransd
root      1762     1  0 14:48 ?        00:00:00 /sbin/dhclient -1 -q -lf   /var/
lib/dhclient/dhclient-eth0.leases -pf /var/run/dhclient-eth0.pid eth0
root      1814     1  0 14:48 ?        00:00:00 auditd
root      1816  1814  0 14:48 ?        00:00:00 /sbin/audispd
root      1838     1  0 14:48 ?        00:00:00 /usr/sbin/restorecond
root      1852     1  0 14:48 ?        00:00:00 syslogd -m 0
root      1855     1  0 14:48 ?        00:00:00 klogd -x
…
```

Once you've got it tested and working, copy and paste the script file into your submission document and **TAKE A SCREENSHOT** of the output of the script using some example arguments.

Next, let's assume you've actually forgotten your root password and you have not yet setup sudo. The only way to recover the password is to boot the computer into single user mode. To do that:

1. Reboot the computer.
2. When GRUB comes up, hit any key to stop it from automatically booting the default kernel.
3. The default kernel will be highlighted, so hit the '`a`' key to modify the kernel options.
4. Add a space at end of the line and then add the word "`single`".
5. Hit `enter` to boot with the modified option.
6. Note that this is not a permanent change. It only applies to this boot.
7. Linux will boot like normal except that will stop a shell prompt (already logged in as root) before it runs many of the startup scripts.
8. At this prompt, you can run commands like normal, and in particular you can use the `passwd` command to change the root password if necessary.
9. To see how few non-kernel processes are running, go to the directory with your `psk.sh` script (probably `/home/USERNAME`, and maybe in a subdirectory if you put in there).

10. Run `psk.sh -v` to see the 4-6 non-kernel processes running in single user mode (run level1).
11. When you're done, reboot the VM: `shutdown -r now`

Now let's turn our attention to startup scripts. Recall that `chkconfig` is used to change which scripts are run for each run level. In a terminal:

1. Run: `sudo /sbin/chkconfig --list`
2. That will give you the list of all services on the system and which ones are turned `on` or `off` for each run level.
3. To narrow it down to just the ones that are `on` for some run level, run:
   `sudo /sbin/chkconfig --list | grep ":on"`
4. There are about 45 services in that list. All of them are `on` for run level 5, and nearly all of them are `on` for run level 3.

Many of these services are useful in a broad range of environments, but some of them aren't really necessary on our VMs right now. So, let's turn a few of them `off`. First:

1. Run: `sudo /sbin/chkconfig cups off`
2. That turns the "`cups`" service (printing) `off` for all run levels.
3. Repeat the above command for these services:
   a. `autofs`: automatically mount remote file systems when accessed
   b. `avahi-daemon`: an Apple zero-conf daemon
   c. `cpuspeed`: scale CPU speed with load
   d. `hplip`: another printing service
   e. `isdn`: ISDN networking
   f. `netfs`: networked file systems
   g. `portmap`: service that other network services rely on
   h. `sshd`: remote login daemon
4. Remember that this doesn't stop those services right now. Instead, it sets the system configuration to NOT start those services during the next boot.
5. Check that all of those services are indeed `off`:
   `sudo /sbin/chkconfig --list | grep ":on"`
6. If you look closely, you'll see that the `cpuspeed` service is still `on` for run level 1. So, you should always double check that `chkconfig` has done exactly what you want.
7. To turn it `off` specifically for run level 1, run:
   `sudo /sbin/chkconfig --level 1 cpuspeed off`

8. **TAKE A SCREENSHOT** of the output of:
   `sudo /sbin/chkconfig --list | grep ":on"`
9. There should be 36 services left that are `on` for some run level.

To be sure that the system still functions correctly after turning these services `off`, you should reboot your VM:

1. Run: sudo `/sbin/shutdown -r now`

The last step is to write a script that maintains a log file of each time your VM boots successfully. First, write a script named `bootlog.sh` that adds a single line to the file `/home/USERNAME/boot.log`. The line should contain the current date and time as well as message stating that the local machine booted. The line should be appended on the end of the file (it should not overwrite the file each time!). You'll need the `date` command, the `$HOSTNAME` environment variable and the `>>` I/O redirection operator. The script should not output anything to the screen. Put the script in your `/home/USERNAME/bin` directory. Here is an example run:

```
$ bootlog.sh
$ cat boot.log
Tue Feb 15 19:06:57 EST 2011 wisemanc.comp315 booted!
$
```

Once you've got it tested and working, copy and paste the script file into your submission document and **TAKE A SCREENSHOT** of the output of the script and the contents of the boot.log file.

Next, add it to the `rc.local` script so that it is run each time your VM boots. Edit `/etc/rc.d/rc.local` and add a line that calls your script to the bottom:

1. Run: `sudo nano /etc/rc.d/rc.local (or your favorite editor)`
2. Add this line to bottom of the file: `/home/USERNAME/bin/bootlog.sh`
3. **TAKE A SCREENSHOT** of the output of: `cat /etc/rc.d/rc.local`.
4. Now, reboot your VM and you should see a new entry at the bottom of your `boot.log` file that tells you when the VM finished booting.

Extra credit: If you want to learn more about the startup script system and `chkconfig` you can complete the following extra steps to earn some extra credit. It will require some digging on your own to figure out exactly how to make it all work.

Expand your `bootlog.sh` script to accept one command line argument: `start` or `stop`. Any other argument should result in a usage message and the script exiting. The same goes if no arguments are given, so either `start` or `stop` must be given on the command line. If the argument is `start`, output the message like before. If it `stop`, simply change the output to

say "shutting down" instead of "booted", but the rest (date and machine name) should stay the same.

Next, you'll add the script to the actual startup script system. Copy the script to the `/etc/init.d` directory. To have the script fit in, you'll have to add some extra information in comments to the top of your script so that `chkconfig` knows what to do with it. Look at the `chkconfig` documentation and other scripts in `/etc/init.d` to see how to do this. You should configure the script so that its default run levels are 3, 4, and 5, it starts with sequence number 98, and it stops with sequence number 2. Then, use `chkconfig` (again, look at the documentation) to add the script to the system and enable it for the default run levels. **TAKE A SCREENSHOT** of the output of `chkconfig --list | grep bootlog`. You should see `bootlog.sh` turned `on` for levels 3, 4, and 5. Also, copy and paste the script file into your submission document. Make sure it all works by rebooting your VM. You should see a shutting down line followed by a booted line added to your `boot.log` file after the reboot completes. **TAKE A SCREENSHOT** of the file showing the additional lines.