CST8177 – Linux II

More on file systems, Booting

Topics

- bind mounts
- Booting process and SysVinit
- Installation Disk rescue mode

Bind mounts

- A bind mount is used to mount a directory onto a mount point: man mount
- use the “bind” option for the mount command

mount –o bind /some/dir /anotherdir

  - now /some/dir and /anotherdir are the same directory
- Be careful with bind mounts, because they make it possible to form cycles in the file system
- e.g. dangerous: "mount –o bind /home /home/user/dir"
  - serious repercussions for
    - rm –rf /home/user # will remove all of /home
    - find /home/user # will never stop
    - any program that recursively descends directories

Bind mount examples

- make an inaccessible directory accessible:
  - mount –o bind /home/user/private/public /public
- make disk space in one file system available in another file system
  - suppose you have a large separate file system with lots of free space on /var, and root file system with /home is nearly full:
    - mkdir /var/local/home/{user1,user2}
    - move contents of /home/{user1,user2,...} to /var/local/home
    - mount –o bind /var/local/home /home
    - beware: new /home has same mount options as /var

Bind mount examples (cont'd)

- share directories across chroot environments
  - mount -o bind /dev /home/user/myroot/dev
  - chroot /home/user/myroot/dev
  - in the chroot-ed environment, /dev will be the same as the un-chroot-ed /dev

Booting

- http://teaching.idallen.com/cst8207/14f/notes/750_booting_and_grub.html
- page numbers for Fifth Edition Sobell:
  - Chapter 11: 424-431
  - Chapter 15: 551-552

Booting Sequence (CentOS)

- Power button pressed
- BIOS
- POST
- MBR : contains grub stage 1
- grub stage 1 : to find grub stage 2
- grub stage 2 : to launch kernel
- kernel running
- init process (PID 1) : consults inittab
- /etc/inittab
- /etc/rc.d/rc.sysinit
- /etc/rc.d/rc 3 : assuming default runlevel 3

/etc/inittab

- /etc/inittab contains records of the form
  - id:runlevels:action:process
  - id: identifies an entry
  - runlevels: the runlevels in which the action should be taken
  - action: the action that should be taken
  - process: the process to be executed
- Because CentOS 6.6 is migrating to a successor of sysVinit (upstartd, which will be replaced with systemd), only the `initdefault` action is present in our `/etc/inittab`

When booting

- Even in CentOS 6.6, with upstartd, when the system boots to runlevel 3, the following happens as it did with sysVinit

```
/etc/init.d/rc.sysinit
/etc/init.d/rc 3   #default runlevel 3
```

- The `sysinit` action now is invoked due to the upstartd `/etc/init/rcS.conf` file
- The `/etc/init.d/rc` script being called with argument `3` is due to the upstartd
  `/etc/init/rc.conf` file
- Under sysVinit, this was controlled by
  `/etc/inittab`

SysVinit scripts

- Even with upstartd, sysVinit is supported
- `/etc/init.d/*`
  - these are scripts for starting, stopping, restarting services
- `/etc/rc.d/rc.N.d/*  #where N is a runlevel`
  - these are symbolic links to service's script
  - begins with K means service should not be running in that runlevel: call it with "stop" argument
  - begins with S means service should be running in that runlevel: call it with "start" argument
- chkconfig maintains these scripts

chkconfig

- all `/etc/init.d/*` scripts manageable by chkconfig have two or more commented lines
- first tells chkconfig what runlevels, and start and stop priority
- runlevels is "-" if by default should not be started in any runlevel
- second is a description
- For example: `/etc/init.d/ntpd`

```
# chkconfig: - 58 74
# description: ntpd is the NTPv4 daemon. \
# The Network ....
```

/etc/rc.d/rcN.d/*

- The /etc/rc.d/rcN.d/ (N=0,1,2,3,4,5,6) directories contain symbolic links to scripts in `/etc/init.d`
- These links are maintained by `chkconfig` (links created or removed by commands like `chkconfig <service> on`)
- When entering a new runlevel
  - during boot as controlled by `/etc/inittab`
  - or by root running a `telinit <newlevel>` command (example `telinit 2` to enter runlevel 2)

The system will call scripts to stop services that should not run in that runlevel, and start services that should run in that runlevel

Entering a runlevel

- When entering a new runlevel, the system needs to stop the services that should not be running in that runlevel, and start the services that should be running in that runlevel
- To do this, the system calls the scripts in that runlevel's directory,

`/etc/rc<lev>.d/`, where `<lev>` is a runlevel

  - Scripts whose names begin with K are called with a stop argument (if that service is running)
  - Scripts whose names begin with S are called with a start argument (if that service is not running)

Example of entering a runlevel

- Upon entering runlevel 3 (for example):
  - each `/etc/rc3.d/K*` script is called with "stop" (if that service is running)
  - each `/etc/rc3.d/S*` script is called with "start" (if that service is not running)
  - The ordering of the scripts being called is given by the chkconfig priority, which is a number in the symlink-ed name of each script
  - These numbers in the link names put the scripts in a certain order
    - chkconfig created the link with this number in the link name because of those commented lines in the script itself (we talked about those a few slides ago)

Example service: sshd

- example `/etc/rc3.d/S55sshd`
  - sshd is configured to run in runlevel 3
    - otherwise, there would be a `K25sshd` script there instead (why 25?)
  - 55 is the priority of starting the sshd service when entering that run level
- This `S55sshd` script is a symlink to `/etc/init.d/sshd`
- Again, the `chkconfig` command creates and removes these links when we use it to enable or disable a service for a runlevel

service – run a System V init script

- `service SCRIPT COMMAND [OPTIONS]`
- `SCRIPT` is `/etc/init.d/SCRIPT`
- `COMMAND` is an argument to the script
  - `start`
  - `stop`
  - `restart`
  - *etc*
  - `start` and `stop` must be recognized by `SCRIPT`
- Example: `service ntpd start`
  - same effect as `/etc/init.d/ntpd start`
- Example: `service ntpd stop`
  - same effect as `/etc/init.d/ntpd stop`

Installation DVD for rescue mode / Live CD

- There are dangers associated with doing file system operations on "system directories" that might be used in system operation.
- For example, many programs will use the shared libraries in `/usr/lib`, which disappear if we move `/usr` as we did earlier when we had to run `/usr1/bin/rsync`
- Also, there may come a time when the system won't boot properly: MBR corrupted, bad entry in `/etc/fstab`, inconsistent `/` file system

Rescue Mode

- When you boot with a CD/DVD into rescue mode, you are running a different Linux system installation (from the CD)
- However, because the rescue Linux system is running on your hardware, it can access the hard disks you have attached (where your "real" Linux system installation resides)
- Your "real" Linux installation is not running in rescue mode – it might even be broken
- The rescue system can let you make changes/repairs to that "real" Linux system which isn't even running

linux rescue

- To boot into rescue mode
  - ensure BIOS boot order is set for booting from CD/DVD before Hard Drive (even in VMware – F2 to enter setup)
  - insert the installation DVD into drive (or the iso image into the virtual DVD drive)
  - boot the system
  - type "linux rescue" at the prompt, or select the "Rescue" menu item
  - Linux will run "from" the DVD (Live CD), not from your file systems (your system is not running)
  - It will offer to search for and mount your Linux file systems on /mnt/sysimage

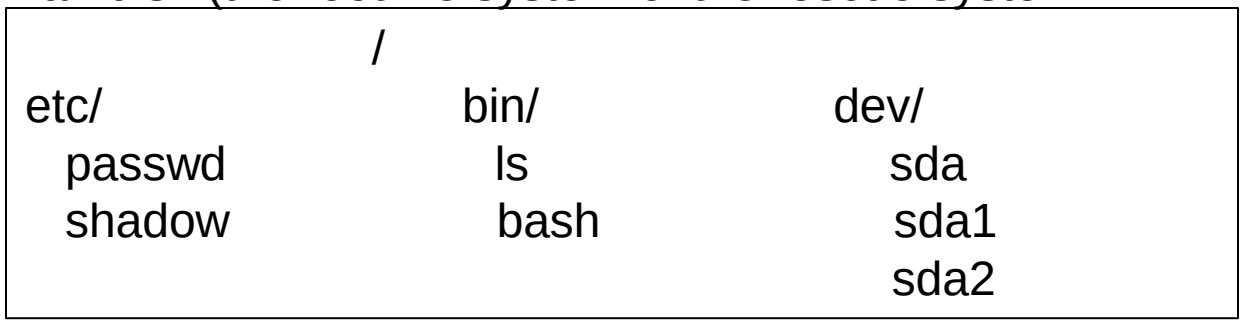
linux rescue (cont'd)

- The Live CD Linux system can see your hard drives, and this is how you can repair or alter what is on those hard drives

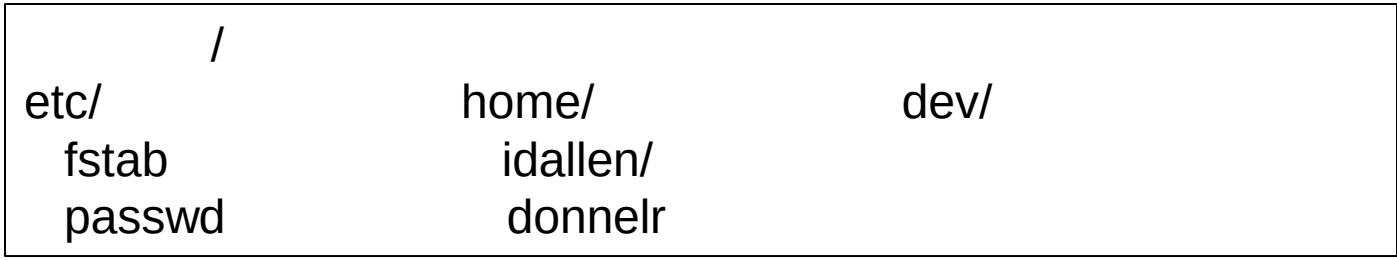
- You need to remember that a Live CD Linux system is running from its own root filesystem (like dual boot?), so this means
  - the users are different `/etc/passwd` `/etc/shadow`, etc (or should we say all of `/etc`) are different
  - the services running, firewalling, and so on, are different

Rescue mode / Live CD

ramdisk (the root file system of the rescue system



/dev/sda1 (your "real" root file system)



linux rescue example 1

- Fix /etc/fstab
  - mount /dev/sda1 /mnt/sysimage (if it isn't already mounted – the rescue boot process probably offered to mount this for you)
  - vi /mnt/sysimage/etc/fstab
    - fix the problem
    - save and quit
  - exit (to reboot)

linux rescue example 2

- fix MBR
  - # our root file system is mounted on /mnt/sysimage
  - chroot /mnt/sysimage
  - # now / is our root file system!
  - # our boot filesystem is mounted on /boot
  - grub-install /dev/sda
- Whoa! That chroot thing was neat
  - chroot runs a program or interactive shell using the named directory as the root directory
  - Default program is ${SHELL} -i
  - This simulates running off our system's root file system without going through its boot process

chroot

- That `chroot` command did something very special, so let's be sure we understand what it did
- `chroot /some/dir` gives us a shell process where the `/some/dir` is `/` for that shell process
- In that shell process, any commands you run from its prompt and those resulting processes will work with that changed "root"
- They will use the `/bin, /lib/, /etc...` in the changed root

rescue chroot /mnt/sysimage

- When we are running in rescue mode, and our "real" root file system is mounted on

`/mnt/sysimage`

then the shell prompt we get from

`chroot /mnt/sysimage`

will "use" (because that's what it sees) our "real" `/bin, /lib/, /etc`... (our "real" root file system that resides on our disk)

- We can even start services from that chroot-ed prompt – they will run with our "real" root file system binaries(`/bin`) libraries (`/lib`) and configuration (`/etc`), but on the rescue kernel

rescue /dev

- The `/dev` directory on modern Linux systems contains the device nodes, and these are managed by `udev` at boot time
- When booting in rescue mode, `udev` puts device nodes for *your* hardware (disks, partitions, etc) into `/dev`
- Your "real" (non-rescue) root file system contains an empty `/dev` directory (it looks full to you because `udev` populates it when you boot your real system!

/mnt/sysimage and /dev

- When the rescue system mounts your "real" root file system on `/mnt/sysimage`, it first creates a bind mount from

`/dev` to `/mnt/sysimage/dev`

so that when you do

```
chroot /mnt/sysimage
```

the shell you get will see a populated `/dev` instead of the empty directory

- This is a good reason to know about bind mounts!