The Linux Boot Sequence

- A good understanding of how Linux boots will be useful as the course progresses. Many of the activities that you'll be doing over the course of the semester require you to modify system files, and it's entirely likely that at some point you'll find yourself sitting in front of a workstation that no longer works.
  - It's a little ironic, but a good understanding of how Linux boots will often allow you to avoid actually rebooting the workstation. Instead, you can invoke the necessary startup actions manually.
- The material in this lecture is drawn from Chaps. 6 & 7 of the *Linux System Administrator's Guide* and from Chap. 6 of *Linux Unleashed*.
- Let's begin at the beginning. When a PC (or clone) powers up, the hardware transfers control to a program called the BIOS, which resides in ROM.
- The BIOS is not too bright, but it knows enough to choose a disk drive and read the very first sector, called the boot sector. The boot sector contains, among other things, a slightly more intelligent program whose job is to load the operating system kernel into memory.
- In Linux, this program is called LILO (the Linux Loader). LILO reads the Linux kernel into memory and starts it.
  - If your computer is set up to run multiple operating systems (Linux and Windows, for example), this is where you get to choose which one to run.
- The Linux kernel performs a sequence of actions as it starts:
  - It scans the hardware configuration and initialises device drivers.
  - It mounts the root file system and performs a consistency check. If any damage is found, it is repaired if possible and the kernel is restarted.
  - Once these preparations are completed, the kernel starts a user process, `/sbin/init`, which is responsible for starting everything else.
- The `init` program has process id 0, the root of the process tree. All processes running on the computer are descended from `init`. `init` remains in existence until the computer is halted, and we'll see that it has overall responsibility for the state of the system.

  When it is started, `init` looks for the file `/etc/inittab` for instructions on how to proceed. Here's a fairly simple example from a workstation in the Network Lab.

```
# inittab

# This file describes actions to be taken by the init process to bring
# the system to a given run-level.

# Set the default run level, so init can learn it at boot time.

id:5:initdefault:

# Actions to take at system boot, prior to moving to a specific run
# level.

si::sysinit:/etc/rc.d/rc.sysinit

# Actions to take on entering run level k. init will execute the command
# and wait for completion.

l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6

# For every run level, execute update once at entry to flush dirty disk blocks
# blocks back to the file system. ‘once’ implies init will not wait for
# completion.

ud::once:/sbin/update

# Trap CTRL-ALT-DEL and do a clean shutdown and reboot, allowing processes
# 3 seconds (-t3) to shut down between initial notification (SIGTERM) and
# hard kill (SIGKILL).

ca::ctrlaltdel:/sbin/shutdown -t3 -r now

# When our UPS tells us power has failed, assume we have a few minutes
# of power left.  Schedule a shutdown for 2 minutes from now.

pf::powerfail:/sbin/shutdown -f -h +2 "Power Failure; System Shutting Down"

# If power was restored before the shutdown kicked in, cancel it.

pr:12345:powerokwait:/sbin/shutdown -c "Power Restored; Shutdown Cancelled"

# Run gettys on virtual consoles 1 -- 6 in standard runlevels

1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6

# Run our preferred X display manager on the remaining virtual console
# (VC 7) in runlevel 5.

x:5:respawn:/etc/X11/prefdm -nodaemon
```

Lines in `inittab` have the form

id : *runlevel(s)* : *action* : *command*

- The *id* is just a convenient identifier, with no particular meaning to `init`.
- The *command* is a command to be executed, either a program or a shell script.
- A full explanation of *runlevel* and *action* will be easier after a bit more explanation.

- The first thing that `init` looks for in `inittab` is a line with the keyword '`sysinit`' in the *action* field. This is the command that should be executed when the system is booting.
  - In the RedHat Linux distribution, the command associated with this line is the script `/etc/rc.d/rc.sysinit`.
  - `rc.sysinit` does many things, among them
    - enable virtual memory and paging;
    - mount other local file systems;
    - clean up system status files;
    - establish the initial execution path (the `PATH` environment variable).
- When `rc.sysinit` finishes, `init` looks for another line in `inittab` with the keyword '`initdefault`' in the *action* field. This entry has no associated *command*; its purpose is to specify the default run level which should be entered to complete the boot process.
- This is a good point to explain the concept of a run level. By convention in a unix system, a *run level* is an integer between 0 and 6 which specifies the overall system state (usually called a mode).

  Here's the conventional set of run levels:

0: Halt the system; setting the run level to 0 will cause Linux to do an orderly shutdown and halt the computer.

1: Single-user mode; in this state only the kernel and a bare minimum of essential system services are running. This state is used for system maintenance, upgrade, and repair which cannot be performed during normal system operation. (Usually because the system state will not be consistent, and thus unsafe, while the work is in progress. Also, excluding other users avoids unnecessary system activity.)

2: Multi-user mode without NFS or X windows; in this state the system is capable of supporting multiple users, but does not mount remote file systems and provides only a command line login interface.

3, 4: Multi-user mode without X windows; as mode 2, but remote file systems are now available.

5: Full multi-user mode; as modes 3 & 4, but also provides a login via the X display manager.

6: Reboot; the system is halted and then an immediate reboot is automatically started.

These run levels are just conventions; while all unix systems follow the same general startup sequence, the specific mapping of run levels to system modes will vary from one to the next.

❋ While we're explaining, might as well deal with the remainder of the *action* labels.

sysinit	Specifies the command to be performed at boot time.
initdefault	Specifies the default run level; no associated command.
powerfail	Specifies the command to be executed if a power failure is detected.
ctrlaltdel	Specifies the command to be executed if the user tries to force a reboot by pressing the ctl-alt-delete key sequence.
once	Specifies that the command is to be started exactly once; `init` will not wait for completion.
wait	Specifies that the command is to be executed exactly once, and that `init` should wait for it to complete before proceeding. (This is the usual behaviour, also applied to the commands specified in the sysinit, ctrlaltdel, and powerfail lines.)
respawn	Start the command and monitor it; if it ever terminates, immediately execute the command again. This behaviour is used for programs which monitor the console and other ports for login attempts.

There are other actions in addition to those listed here. Consult the `man` page for `inittab`.

- Once `init` knows the default run level, it looks for a line which specifies that run level in the *action* field, and executes the associated command.
  - In a Linux system, this will be `/etc/rc.d/rc`, which is executed with the run level as the single parameter.
- `rc` looks for scripts in a directory with the name `/etc/rc.d/rc`*N*`.d`, where *N* is replaced by the desired run level.
  - So, if the system is to go to run level 5, `rc` looks for a directory `/etc/rc.d/rc5.d`.
- In the directories `/etc/rc.d/rc`*N*`.d` are scripts with names of the form `[S|K]`*ddcommand*.
  - The initial `S` or `K` specifies whether the script is used to start (`S`) or kill (`K`) a particular service or set of services. When entering a run level, the `rc` script will execute the kill scripts first, followed by the start scripts.
  - The two digits *dd* specify a number between 00 and 99. This number determines the order of execution within the start and kill groups.
  - The *command* portion of the file name is chosen to indicate which system or group of systems is manipulated by the script.

- If you use '`ls -F`' or '`ls -l`' to list one of the `rc`*N*`.d` directories, you'll see that these scripts are actually symbolic links to scripts in `/etc/rc.d/init.d`.
- Typically, each script can be called with one of four parameters: start, stop, status, and restart; some have additional parameters.

- What sorts of functions are controlled by the scripts in `/etc/rc.d`?
  - File sharing with NFS (for other unix systems) or SAMBA (for DOS systems).
  - Internet service dæmons and protocol dæmons.
  - The `cron` dæmon, which can run a job at a specified time (handy for scheduling backups, accounting, *etc*.).
  - Printing and logging dæmons.
  - The Network Information Service (NIS), formerly known as the yellow pages (YP). This is a set of databases maintained by a central server which can be queried over the network.
  - Remote procedure call and remote execution dæmons.

- There's one more thing to notice in `inittab`. Down at the bottom of the file are a set of lines with numbers in the *id* position. These lines control the program which is started up to listen for login attempts.
  - For each port where a user can log in to the computer — typically the console and perhaps serial ports connected to terminals — there must be a program which listens for login attempts.
  - Linux uses a program called `mingetty`; other unix systems have slightly different names. This program is responsible for listening for activity at the port and presenting a login prompt when it senses activity. Once you've authenticated yourself with a login id and a password, mingetty starts up a command shell for you to begin work.

- You'll notice that there are seven such entries in `inittab`. Six of them run `mingetty` and the final one (which is used only for run level 5) runs a program called `prefdm`.

- Let's deal first with why there are seven entries when there's only one console terminal on a workstation.
  - Linux provides a facility called a virtual console; the workstations in the network lab are configured for seven.

- Each provides an entirely independent port, just as if there were seven separate terminals attached to the computer.
- To change from one to another, use the key combination 'alt-F*n*', replacing *n* with 1 – 7. It's a little harder to actually escape from the X login screen which is presented on virtual console 7; you'll need to type 'ctl-alt-F*n*' to get to one of the other virtual consoles.

- Virtual consoles 1 – 6 provide what's known as a command line login.
  - Once you've authenticated yourself to the system, a command shell is started for you. A unix command shell is the analog of the DOS command shell, but bears about the same relationship as modern man does to a flatworm.
  - There are many different command shells available; the one that is started for you is determined by your entry in the `/etc/passwd` file. This first shell is called the *login shell*.
  - Once the login shell is started, it will search for an initialisation file. The name varies from shell to shell; for `tcsh`, the file is called `.tcshrc`, for `bash`, `.bashrc`. (Notice the leading '.' in these file names; they are hidden files, to cut down on the clutter in your home directory. As you'll see, unix has *lots* of these.) If there's no initialisation file in your home directory, a default file will be used, usually found in `/etc`.
  - The shell initialisation file is executed every time you start a new command shell (which you'll do often in unix). But the login shell also executes a second initialisation file, called `.login`. Again, it will look for this in your home directory; there may or may not be a system default.
  - When the login shell finishes executing the shell initialisation file and the *.login* file, it will present you with a prompt and you can get down to work.

  Virtual console 7 is different; it provides a GUI login interface, courtesy of the X Window System display manager.
  - This login screen will look generally familiar — there are spaces to type your id and password, and some option menus.
  - The screen is presented courtesy of the display manager, of which Linux provides several to choose from. The one running in the Network Lab is called `gdm`, the Gnome Display Manager.
  - To explain any further what's going on here, we need to take a step back and talk about the X Window System.