COMP421
Unix Environment for Programmers

Lecture 02: System overview

Jeff Wiegley, Ph.D.
Computer Science

jeffw@csun.edu

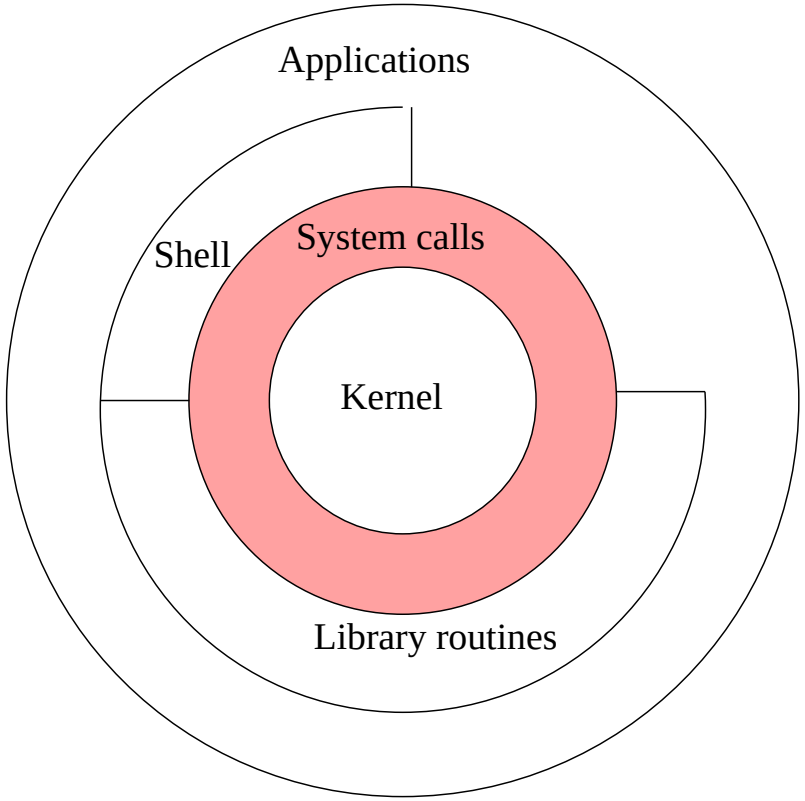
08/29/2005

Unix

"Unix" has come to mean two different compenents simultaneously:

1. Operating System (Kernel): Responsible for protecting and allocating resources in a fair manner.
   - File input and output.
   - Network resources.
   - Memory protection.
   - Interprocess communication.
2. A collection of applications and utilities that provide productivity.
   - emacs, vi, nano,
   - gcc, g++, javac
   - X-Windows, gnome, kde
   - gimp
   - mozilla/firefox/opera/konqueror
   - LaTeX 2_ε, troff, nroff, sed, awk, grep, find, cut, tail, xargs, etc.

Basic operating system architecture:



Kernel:

- Makes authorization decisions and enforces security policies.
- Controls hardware
- Internal state cannot be seen or modified by normal users.
- All access to kernel must be done through requests using only System calls.
- Device drivers operate in kernel mode
- Creates and schedules processes. (Every running program is a process.)
- Handles memory paging.

Technically "Linux" only refers to the Linux kernel (as provided by linux-2.6.13.tar.bz2 for example).

GNU/Linux or Linux/GNU refers to the everything including the kernel.

Library routines:

- Provides basic features common to most applications
  - printf, scanf
- Built on top of system calls as an added layer of abstraction and functionality.
- Reduces memory consumption by centralizing a single copy of needed code.

Shell:

The "Shell" is the interactive application that a person is presented with upon logging on to the system.

- The shell reads in typed commands and executes the commands entered to provide flexible productivity.
- The shell is more than just a command executor.
  - Grown into a programming language. Has conditional branching, loops and variables.
  - limited high-level capabilities
    - No object oriented features.
    - Limited variable scope.
    - No dynamic memory allocation.
  - excellent "logic" capabilities.
    - If this command fails do this, otherwise do that...
  - Excellent tool for automatic information or task processing.

Applications:

Applications are everything else the user uses (and to a large extent the shell is just another application).

Unix Applications (sort of) fall into two broad classifications;

1. Utilities: Small programs that do some type of limited processing but do it very efficiently and very flexibly.
   - sed, grep, awk, find, cut, tail, head, xargs, dd, wc

   These are sometimes called "filters" because their input and output can be chained together through *I/O redirection* to produce very complicated and accurate results very, very quickly.
2. Applications: Major, single purpose, stand alone programs.
   - gimp, mozilla, evolution, LaTeX 2_ε, Matlab
   - X Windows, kde, gnome

File system:

The Unix kernel treats almost every resource as a file and presents it to the use as a file.

Files are, of course, files that can be read and written using a *file descriptor*.

But memory network communication are also "files".

Even the input and output from programs is treated as a file. (Which is why you can "pipe" the output from one program as input to another program.)

Files and Directories:

Unix doesn't have `C:/>` or `D:/>` drives because drives are not presented to the user as discrete seperate filesystems.

Instead Unix "mounts" drives under a single filesystem.

The top most directory is simply called "/" or the "root" filesystem.

Entries in the filesystem can be one of:

- A plain file (text, data, programs)
- A socket (provides communications between independent processes).
- Symbolic link
- block device (provides communications to hardware)
- character device (also communication with hardware)
- directory. (like a folder, only non-graphical)
- mount point. (any directory is a mount point candidate)

Typical filesystem:

- `/bin` Read only hard disk mount, supplies basic user programs.
- `/boot` Minimal filesystem, items necessary for booting.
- `/dev` Contains block and character files for devices.
- `/etc` Read only mount, system configuration items.
- `/home` read/write mount containing all user files beneath it.
- `/lib` Read only mount, contains shared libraries.
- `/mnt` General place to mount additional filesystems.
- `/proc` A pseudo filesystem presenting kernel internal/info.
- `/root` Files owned by root
- `/sbin` Files statically linked for disaster recovery.
- `/tmp` Read/Write mount for providing scratch space.
- `/usr` Read Only mount for system wide data
- `/var` Read/Write mount for system wide data.

For example this lecture file is:

```
/home/jeffw/Northridge/Semesters/Current/COMP421/Lectures/lecture02.tex
```

All of the data can be read on the hard drive at `/dev/hda` since the kernel represents the first harddrive as that file. (`/dev/hdb` is the second IDE drive). Sounds scary except the permissions for `/dev/hda` are:

```
brw-rw---- 1 root disk 3, 0 2005-08-22 14:38 /dev/hda
```

which means only the root user can access that file (and therefor the harddrive directly) [the "`b`" means it's a "block special file".

Quick nomenclature:

`/home/jeffw/Northridge/Semesters/Current/COMP421/Lectures/lecture02.tex`

Is known as a *Path* or *pathname*. This particular one is an *absolute path* since its reference goes all the way to the root.

The *current working directory* is the directory that a *relative path* would be relative to when given.

`Lectures/lecture02.tex`

Is a relative path. And depending on what my current working directory is might be COMP421 lectures or COMP598EA lectures.

The current working directory is not specific to an interactive shell. Running programs have a current working directory that relative paths are suffixed to when issuing commands such as

`open(somefilename,O_RDONLY).`