A QUICK TOUR THROUGH UNIX

Ken Wong
Washington University, St. Louis
kenw@arl.wustl.edu
www.arl.wustl.edu/~kenw

HISTORY OF UNIX

- Original Goals: Designed by programmers, for sophisticated software developers
  - Simple, elegant, consistent (e.g., X* has same meaning everywhere it is used)
  - Parsimony, power, flexibility (Combine components)
  - Terse (e.g., avoid multiple prompts within a single command)
- MIT MULTICS (64?)
- Bell Laboratories/AT&T
  - V1 (69), V6 (76), [Parent Unix], V7 (78) [First portable Unix]
  - System III (82), System V (83)
- Berkeley Software Distribution
  - 1BSD (77), 4.3BSD (86), 4.4BSD (93), FreeBSD (93)
- Sun Microsystems: SunOS (84), Solaris (92)
- Microkernels: MINIX (87), Mach (86)
- Linux V0.01 (91)

UNIX INTERFACES

<table>
<thead>
<tr>
<th>Users</th>
<th>Utility Programs (shell, cc, vi, make, GUI)</th>
<th>Standard Library (open, close, read, write, fork)</th>
<th>Operating System (kernel mode) (process, memory, I/O)</th>
<th>Hardware (CPU, memory, disks, terminals)</th>
</tr>
</thead>
</table>

User Interface
Library Interface
System Call Interface

THE SHELL

- A Command Sequence
  ```
  grep -i 'cs.*422' 422.email* > x1  # extract student info
  grep -i -v subj < x1 > x2          # delete extra lines
  cut -f 2,3 x2 > x3                # retain columns 2 and 3
  sort x3 > 422roster.txt           # sort by last name
  ```
- Alternative Commands
  ```
  grep -i 'cs.*422' 422.email* | grep -i -v subj |
  cut -f 2,3 | sort > 422roster.txt &
  ```
- Shell Script
  ```
  grep -i 'cs422' $* | grep -i -v subj |
  cut -f 2,3 | sort > 422roster.txt
  ```
- Usage: mkroster 422.email* &
UNIX SHELL

- A shell is a command line interpreter
  - **Primary Purpose**: Translate command lines typed at a terminal
  - Input to a shell is from the terminal (interactive shell) or a file
  - The shell is determined by a user's entry in the /etc/passwd file
- **Some Shells**
  - Bourne Shell (sh)
    - What I use mostly for writing shell scripts
  - C-Shell (csh)
    - Typically, the default command line interpreter
  - TC-Shell (tcsh)
    - C-shell with file name completion and command line editing
  - Bash Shell (bash)
- **Shell Script**: A file containing shell commands

SYSTEM CALLS (1)

- **See Section 2 of the Manual Pages** (e.g., man 2 wait)
- **Process Management**
  - *pid = fork()*: Create a child process identical to the parent
  - *pid = wait(&status)*: Wait for a child to terminate
  - *ret = exec(path, arg0, ...)*: Replace a process image
  - *exit(status)*: Terminate process and return status
- **File Management**
  - *fd = open(path, how, ...)*: Open a file for reading, writing or both
  - *ret = close(fd)*: Close a file
  - *m = read(fd, buf, n)*: Read n bytes from a file into a buffer
  - *m = write(fd, buf, n)*: Write n bytes to a file from a buffer
  - *offset = lseek(fd, offset, whence)*: Move file pointer
  - *ret = stat(path, &stats)*: Get file status information

SYSTEM CALLS (2)

- **Directory and File System Management**
  - *ret = mkdir(path, mode)*: Create a new directory
  - *ret = rmdir(path)*: Remove an empty directory
  - *ret = link(path1, path2)*: Create new file path2 pointing to path1
  - *ret = unlink(path)*: Remove directory entry
  - *ret = mount(special, path, flag)*: Mount a file system
- **Miscellaneous**
  - *ret = chdir(path)*: Change working directory
  - *ret = chmod(path, mode)*: Change a file's access permissions
  - *ret = kill(pid, signal)*: Send a signal to a process
  - *ret = gettimeofday(&timeval, 0)*: Get time of day

4.4BSD KERNEL

<table>
<thead>
<tr>
<th>System Calls</th>
<th>Interrupt/Traps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terminals</td>
<td>Sockets</td>
</tr>
<tr>
<td>Raw tty</td>
<td>Cooked tty</td>
</tr>
<tr>
<td>Line Disciplines</td>
<td>Routing Buffers</td>
</tr>
<tr>
<td>Character</td>
<td>Network Devices</td>
</tr>
<tr>
<td>Device Drivers</td>
<td>Hardware</td>
</tr>
</tbody>
</table>

- **Block Device**: Can seek to a byte offset within a block device
- **Character Devices**
  - Raw tty: Each key stroke is used by application (e.g., editor)
  - Cooked tty with line discipline: Line-oriented (can edit up to end of line)
THE FILE SYSTEM (1)

- Powerful, elegant file system from small number of mechanisms

- Types of Files
  - Ordinary File: An unstructured sequence of bytes
  - Directory: A set (grouping) of files (even other directories)
  - Special File: An I/O device (Normally in /dev/)

THE FILE SYSTEM (2)

- Root File System: The root directory "/"

- File Names
  - Absolute Path: File name begins with "/"
  - Relative Path: Any file name NOT beginning with "/

- Removable File System: A file system that can be mounted

- Link (Hard and Symbolic): A directory entry that points to another file

FILE ATTRIBUTES

dors> ls -l /etc/passwd
-rw-r--r-- 1 root wheel 256 Jul 16 13:27 /etc/passwd
dors> df /
Filesystem 512-blocks Used Avail Capacity Mounted on
/dev/sda 294224 79542 199970 28% /
dors> ls -l /dev/sda
brw-r----- 1 root operator 4, 0 Jan 5 1999 /dev/sda

dors> ls -l /bin/mount
-rw-r-xr-x 1 root root 55420 Oct 4 2000 mount

- File Mode

- Number of Links, Owner Name and Group Name

- Number of Bytes in the File

- Date and Time File Was Last Modified

- Pathname

USER IDENTIFICATION

- User ID (UID)
  - A numeric value between 0 and 65,535 identifies each user (in /etc/passwd)

- Group ID (GID)
  - A user's group ID is stored in /etc/passwd file
  - Group IDs are mapped in the /etc/group file to group names
  - A user can belong to multiple groups

- Each file has:
  - The UID and GID of its owner (by default, the creator)
  - Read/write/execute-search permissions for owner, group, and all other users
    - Represented by 9 bits

- Each process has the UID and GID of its owner
  - Can be modified by the setuid and setgid bits
  - The effective UID is the UID of the executable file when setuid bit is on
I/O DEVICES

- All I/O devices look like files
  - Can be accessed with the same `read/write` calls
  - Each device has a special file path name (e.g., `/dev/sd0a`, `/dev/tty0`, `/dev/lp`)
- Block Special File (usually used for disks)
  - A sequence of numbered blocks
  - A block can be accessed directly (i.e., without reading preceding blocks)
- Character Special File
  - Character stream devices (keyboard, printer, mouse)
  - Must access characters sequentially (not directly)
- There is a device driver for handling each special file

INPUT AND OUTPUT

- File Descriptor
  - Small non-negative integer (index into file descriptor table)
  - Every successful file open returns a file descriptor
  - All shells (sh, csh, tcsh, ...) open: 0) stdin, 1) stdout, 2) stderr
- All shells open 3 file descriptors
  - Standard input (stdin), Standard output (stdout), Standard error (stderr)
  - All 3 are normally connected to the terminal (e.g., `ls`
- `stdin`, `stdout`, and `stderr` can be redirected
  - `ls > outfile`

BUFFERED AND UNBUFFERED I/O (1)

`fread(buf, sizeof(char), nitems=40, stdout)`

**User Buffer**

**Cooked**

**Raw**

**I/O Library**

**User Space**

**Kernel Buffers**

**Kernel Space**

**Device Drivers**

BUFFERED AND UNBUFFERED I/O (2)

- Buffered (Added functionality; e.g., concept of a line)

  ```c
  #include <stdio.h>
  int main (void) {
      char line[N];
      while ( (fgets (line,N,stdin)) != EOF) {
          if (fputs (line) == EOF) { ... error ... } }
          return 0;
      }
  }
  ```

- Unbuffered

  ```c
  #include <unistd.h>
  int main (void) {
      int n;  char buf[N];
      while ( (n = read (STDIN_FILENO, buf, N)) > 0) {
          if (write (STDOUT_FILENO, buf, n) != n) ... error ...
          if (n < 0) ... error ...
      return 0;
  }
  ```