Controlling Processes and the File System
Portions courtesy Ellen Liu

Signals
- Process-level interrupt requests
- Dozens of them, use “kill -l” to list them
- They can be sent
  - among processes as a means to communicate
  - by terminal to kill, interrupt, suspend processes
  - by kernel when encountering e.g., division by zero
  - by kernel to notify, e.g., data arrived on an I/O channel

Common Signals
<table>
<thead>
<tr>
<th>#</th>
<th>Name Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>INT  Interrupt (when type ctrl-C)</td>
</tr>
<tr>
<td>3</td>
<td>QUIT     Quit</td>
</tr>
<tr>
<td>9</td>
<td>KILL     Kill</td>
</tr>
<tr>
<td>11</td>
<td>SEGV   Segmentation fault</td>
</tr>
<tr>
<td>15</td>
<td>TERM  Software termination</td>
</tr>
</tbody>
</table>

The Kill Command: Send Signals
- Can send any signals to a process by process owner or the superuser
  - $kill 8021 8021 is the PID
- Default is the SIGTERM, i.e., kill -TERM
- SIGTERM may not always terminate a process, kill -9 8081 sends SIGKILL
  - SIGTERM may be blocked by a process
  - SIGKILL is a signal that can’t be blocked by processes

Outline
- Controlling processes (2)
  - Signals and the kill command
  - Process monitoring: states, niceness, memory, ps, top, uptime
- The Filesystem
  - Pathnames
  - Mounting and unmounting filesystems
  - File tree organization
  - File types

Upon Receiving a Signal
- A process can “catch it”, i.e., designate a signal handler routine to handle it
  - Handler is called. Upon completion, resume (continue) process execution
- A process can also request to block (and then unblock) or ignore signals.
- Otherwise, kernel takes default actions on behalf of the process
  - Generate core dump, or terminate the process
  - Core dump: a process’ memory image, for debugging
Process States

- **Runnable**: The process can be executed
- **Sleeping**: The process is waiting for some resources
- **Zombie**: terminated but not reaped by its parent
- **Stopped**: The process is suspended (not allowed to execute) or traced

Use the "ps" command to view a process’ state

Nice and Renice: Scheduling Priority

- Kernel does process scheduling: which one do I run next among the Runnable processes?
- Process “niceness” affects the scheduling priority
  - A high nice value means a low priority
  - A low nice value means a high priority
  - In Linux, the range is [-20, 19]

Owner of a process can increase its nice value but cannot lower it

$nice +19 ./myjob10 starts myjob10, and sets it to the lowest priority

The ps Command: Monitor Processes

- Sysadmin’s main tool for monitoring processes
- Shows a process’
  - PID, PPID, UID,
  - control terminal, priority,
  - memory consumption,
  - CPU time used,
  - current status
- a: all processes, x: even those without terminal, u: user oriented output format

Memory Consumed by a Process

- %MEM: % of physical (real) memory consumed
- VSZ: total amount of virtual memory allocated to the process
- RSS: Resident set size (portion of VSZ, i.e., number of pages that are currently in real memory)

Virtual memory -> physical memory + some disk space

Managed by pages

Other Commands

- ps gives only a one-time snapshot of the system
- top: provides a regularly updated summary of active processes and their resource consumption
  - By default, every 10 second
- uptime: show the up time, the number of users, the load averages (average numbers of runnable processes) over 1, 5, and 15-minute intervals

Read their man pages
Outline

• Controlling processes (2)
  – Signals and the `kill` command
  – Process monitoring: states, niceness, memory, `ps`, 
    `top`, `uptime`

• The Filesystem
  – Pathnames
  – Mounting and unmounting filesystems
  – File tree organization
  – File types

The Filesystem

• Represent and organize the system’s storage 
  resources, as well as other types of objects – e.g., 
  processes, audio devices, serial ports ...

• Four main components
  – A namespace: name and organize things in a hierarchy
  – An API: system calls to navigate/manipulate objects
  – A security model: scheme to protect/hide/share objects
  – An implementation: software that ties logical model to the 
    hardware

Pathnames

• The filesystem is a single unified hierarchy that starts 
  at the directory `/`, and continues downward through 
  subdirectories
  – `/`: the root directory

• Pathname: the list of directories that must be 
  traversed to locate a file plus that file’s filename
  – Absolute paths: start from root. E.g., `/tmp/foo`
  – Relative paths: start from current directory. E.g., `cse311/A1`
  – Terms pathnamen, filename, path are interchangeable

Pathnames (cont’d)

• Filesystem can be arbitrarily deep

• Each pathname must be <= 255 characters
  – For longer ones, `cd` to an intermediate directory first, then 
    use a relative pathname

• Filenames
  – Must not contain slash `/` character
  – Spaces are permitted, though not recommended. E.g.,
    `$less ”My excellent file.txt”`

Mounting A filesystem

• Smaller filesystems – each consists of one directory 
  and its subdirectories and files

• Smaller filesystems are attached to the tree with the 
  "mount" command
  – Mount maps a directory in the tree (called mounting 
    point) to the root of the newly attached filesystem
    `$mount /dev/sda4 /users`
    install the filesystem
    stored on the disk partition `/dev/sda4` under the path `/users`
  – To see the filesystem content, use `ls /users`
Umoutning A Filesystem

• Filesystems are detached with the “umount” command
  - E.g., `umount /users`
  - E.g.2, `umount /mnt/usb` if to umount a USB key
device if it was mounted to `/mnt/usb`
• The filesystem can not be busy, i.e., no open files or
processes with current directories located there

Organization of the File Tree

• Every distribution or flavor has slight difference
• Root filesystem: root directory and a small set of files
  and subdirectories
  - `/bin`: core OS commands
  - `/boot`: kernel and files needed to load the kernel
  - `/dev`: entries for devices, e.g., disks, printers, ...
  - `/etc`: critical startup and configuration files
  - `/home`: default home directories for users
  - `/tmp`: temporary files

More Standard Directories

- `/lib`: libraries, and parts of the C compiler
- `/mnt`: temporary mount points for removable media
- `/proc`: information about all running processes
- `/root`: home directory of the superuser
- `/usr/bin`: most commands and executables
- `/usr/include`: header files for C compiler
- `/usr/lib`: more libraries
- `/usr/sbin`: less essential commands for sysadmins
- `/var`: log files, accounting info; change rapidly
  - ...

File Types (7 of them)

• Regular files
• Directories
• Character device files
• Block device files
• Local domain sockets
• Named pipes
• Symbolic links

File Types (cont’d)

• Regular files: a series of bytes. Text, data,
  executable, libraries, etc.
• Directories: "." refers to it self, ".." refers to its
  parent directory. `cd ..` go to parent dir
• Device files: used for hardware, peripherals.
  - Characterized by two numbers: major and minor device
    numbers. Major device number identifies a device driver.
    Minor tells the driver the actual unit.
    E.g., the first serial port `/dev/tty0` has 4,0

File Types (even more)

• Local domain socket: for connections between
  processes in local host
  - A filesystem object, not a network port
  - Also called UNIX domain socket
• Named pipes: similar to above. Both for IPC (inter-
  process communication)
• Symbolic links: also called “soft links”