SYST13416
Introduction to Linux
Operating

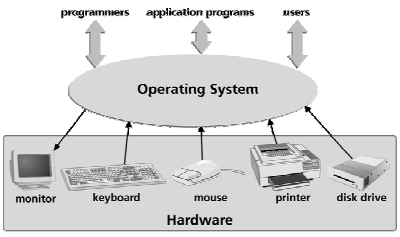
Linux Processes and Utilities

LINUX PROCESSES AND UTILITIES

Objectives

- Processes and utilities

- **Operating System manages the hardware and software resources of the system as various programs compete for the attention of the central processing unit (CPU) and demand memory, storage and input/output (I/O) bandwidth for their own purposes**
  - In a desktop computer, resources include such things as the processor, memory, disk space, etc.
  - On a cell phone, they include the keypad, the screen, the address book, the phone dialer, the battery and the network connection.
- **Operating System provides a stable, consistent way for applications to deal with the hardware without having to know all the details of the hardware.**
  - A consistent application program interface (API) allows a software developer to write an application on one computer and have a high level of confidence that it will run on another computer of the same type, even if the amount of memory or the quantity of storage is different on the two machines.



Recap: Operating System Tasks

The operating system tasks, in the most general sense, fall into six categories:

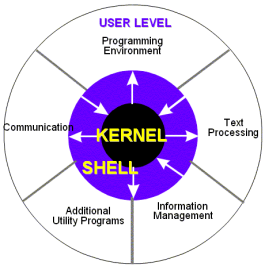
File system management
Process management
- Ensuring that each process and application receives enough of the processor's time to function properly.
- Using as many processor cycles for real work as is possible.

Memory storage and management
- Each process must have enough memory in which to execute, and it can neither run into the memory space of another process nor be run into by another process.
- The different types of memory in the system must be used properly so that each process can run most effectively.

Device management
Application interface
User interface

Understand Unix Utilities

- let you create and manage files
- run programs
- produce reports
- generally interact with the system
- offer a full range of services that let you monitor and maintain the system and recover from a wide range of errors



Recap: Classification of Unix Utilities

- File processing

  afio, awk, cat, cmp, comm, cp, cpio, cut, dd, diff, dump, fdformat, find, fmt, grep, groff, gzip, head, ispell, less, ln, lpr, ls, man, mkdir, mkfs, mv, od, paste, pr, pwd, rdev, restore, rm, rmdir, sed, sort, tail, tar, touch, uniq, wc

- System status

  chgrp, chmod, chown, date, df, du, file, finger, free, ed, quota, kill, ps, sleep, top, w, who

- Network

  ftp, ifconfig, netstat, ping, rcp, rlogin, rsh, rwho, showmount, telnet, wvdial

- Communications

  mail, mesg, pine, talk, wall, write

- Programming

  configure, gcc, make, patch

- Source code management

  ci, co, cvs, rcs, rlog

- Miscellaneous

  at, atq, atrm, batch, cal, crontab, expr, fsck, tee, tr, tty, xargs

- UNIX UTILITIES TO MANAGE RESOURCES

  Making a bootable Floppy Disk: **rdev, mkfs, fdformat, dd**
  Check hard disk usage: **df, du, rm, top**
  System status utilities: **free, ps, kill**
  Formatting text files utilities: **ispell, cmp, nroff, troff, groff**

- The *init* process initializes the system, starts another process to open **terminal lines**, and sets the **standard input** (*stdin*), **standard output** (*stdout*), and **standard error** (*stderr*), which are all associated with the *terminal*.
- At this point, a *login* appears at the *console*.
- The *kernel* controls and manages processes.
- A *process* consists of the **executable program**, its **data** and **stack pointer**, **registers**, and all the information needed for program to run.
- When you log in, the process running is normally a shell, called a **login shell**.
- When the command is entered at the prompt, the shell has the responsibility of finding the command either in its internal code (built-in) or out on the disk, and then arranging for the command to be executed. This is done with calls to the kernel, called system calls.
- A **system call** is a request for kernel services and only way a process can access the system's hardware.
- *Common internal commands:* alias, bg, cd, echo, exec, fg, jobs, pwd, set, shift, test, time umask, unset, wait, . (dot command)

Processes

- **A process is the UNIX system execution of a program.**
- When the system is started, the first process is called *init*.
- Each process has a **process identification number** (**PID**) associated with it.
- The process *init* has PID equal to 1.
- Initialization table: **/etc/inittab**
- User interaction: When you ask for something to be done, you "**spawn a new process**".

Inter-process communication

- **Two or more processes run at the same time. When two processes interact in some way, it is possible to arrange for one job to provide output to, or receive input from, another job.**
- **There are three important methods of doing this:**
  - A ***signal*** is a simple code that is sent from one process to another. The codes are identified by number. A signal indicates the status or initiates some action taken by the system. The signal is sent from an active process to another process via "electronic mailbox".
    1—terminal hangup
    2—terminal interrupt
    3—quit
    8—floating-point error
    9—user terminated process
    11—invalid memory reference
    13—pipe write error
    15—terminate process gracefully
  - A ***queue*** is a waiting line, consisting of jobs to be done. Any user can add a job to the queue and remove it from the queue. I.E. print queue - you queue up jobs to print. As jobs get printed, they are removed.
  - A ***pipe*** is a set of instructions within the computer's memory, analogous to a physical pipeline.

List of Commands

ps—report a snapshot of the current processes
nohup—run a command immune to hang-ups, with output to a non-tty
tty—print the file name of the terminal connected to standard output
stty—changes terminal settings
fg—send a job to the foreground
bg—send a job to the background
kill—send a signal to a process
jobs—view running or stopped jobs associated with your terminal
top—display Linux tasks
ln—make links between files
df—report file system disk space usage
du—estimate file space usage
free—display amount of free and used memory in the system
pstree—show parent-child process relationships
uptime—display the status of the system, how long the system has been up
sleep—deliberately slow down or pause command line
crontab—schedule a periodic process
at—schedule a process for just one execution

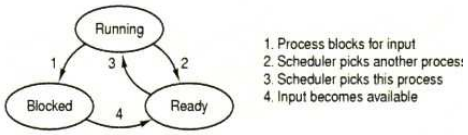
See manual pages for more information.

Some Terminology

- A ***job*** is a process that is not running in the foreground and is accessible only at the terminal with which it is associated, typically background or suspended processes.
- A ***daemon*** is a program that after being spawned, either at boot or by a command from a shell, disconnects itself from the terminal that started it and runs in the background.

Process States

- A process resides in various states: ready, running, waiting, swapped, and zombie.



- A systems must manage several (many) processes. The management of the processes is performed by a special process called the 'scheduler'. The scheduler in Linux has the process name 'init'. It is the init process that transfers processes from state to state.
- In Linux there are several utilities that allow us to view and manage processes.
- To have **ps** output extra information, you can use the u argument (on some systems the -l option). It reports the user running the process, the process ID, the percentage of the CPU the process has been using over the past minute, the percentage of the real memory, the virtual memory size in kilobytes, the physical memory used, the terminal it is connected to, the states, when the process was started, the about of CPU time used by process since it started, and the command name.


```
  PID TTY STAT TIME COMMAND
  520  p0 S    0:00 -bash
  545   ? S    3:59 /usr/X11R6/bin/ctwm -W
  548   ? S    0:10 xclock -geometry -0-0
 2159  pd SW   0:00 /usr/bin/vi lib/addresses
31956  p3 R    0:00 ps
```

Viewing Processes with **ps** Command

- A ***process*** is a running program. Each process on the system has a ***numeric process ID (PID)***. For a quick listing of the processes that you're running, just run the **ps** command on the command line. You should get a list like this:
  - **PID** The process ID.
  - **TTY** The terminal device where the process is running (don't worry about this for now).
  - **STAT** The process status; that is, what the process is doing at the given time and where its memory resides. For example, S means sleeping and R means running. *Check the ps(1) manual page for all the symbols.*
  - **TIME** The amount of CPU time (in minutes and seconds) that the process has used so far. In other words, this is the total amount of time that the process has spent running instructions on the processor.
  - **COMMAND** This one might seem obvious, but be aware that a process can change this field from its original value.
- If you discover that you have zombie processes (maybe you lost a remote connection, for example, and when you log back in and list the processes running, you discover that you have more shells running than the one window), you can determine which is your active session by displaying the PID of the current process ($$) and removing all the others:

```
ps –fu myid
echo $$
activePID
kill -9 zombiePID
```


Code	Process state
D	uninterruptible sleep (usually IO)
R	Runnable (On run queue)
S	Sleeping
T	Traced or stopped
W	Paging
X	Dead
Z	A defunct ("zombie") process

See manual pages for more information.

Process Attributes

- Each process has an environment with various attributes such as command-line arguments, user environment variables, file descriptors, working directory, file creation mask, controlling terminal (console), resource limitations, etc. Many of the attributes are shared with the parent process.
- The kernel also knows about and can report numerous other process attributes. Some examples include reporting the process ID of its parent, the real and effective user and group IDs, the time the process was invoked, and resource utilization, such as memory usage and total time spent executing the program. A real user or group ID is generally the user and group that initially started the program. The effective user or group ID is when a process is running with different permissions.
- To view the various process attributes, you can use the ps -o option. It is used for defining the output format. For example, to output the process IDs, parent PIDs, and the size of the processes in virtual memory of your own running processes, issue the command:
  `ps -o user,pid,ppid,vsz,comm.`
- The following table contains commonly used output format fields:

Field	Definition
user	Effective user ID of the process
pid	Process ID
ppid	Process ID of the parent
pcpu	Percentage of CPU time used
rss	Real memory size in kilobytes
pmem	Percentage of rss to physical memory
vsz	Kilobytes of the process in virtual memory
tty	Controlling terminal name
state	Process state
stime	Time started

- Review the ps manual to see what output formats are available.

Killing a process

- In Linux, life is brutal. If you no longer need a process, you destroy it! Killing a process involves stopping its execution. There are several ways
  - Enter Ctrl+C while a program is running
  - Enter kill PID for a particular process
  - Close the window in which a program runs
  - Exit from the shell in which a program runs
  - Exit GUI entirely
  - Log out from your session

Zombie Processes

- Occasionally, killing processes may not work. In extreme cases, supposedly killed processes may still linger. For example, an abrupt interruption of the system access might leave a process running, which ps will show when you log back on. Such processes are called zombies, that eat system resources. To see if a process is "zombified", use
  `ps –o pid,s,comm`

  where s gives a process state. If the value shown under S is Z, you've got a zombie. To kill zombies, try
  `kill PID`
- Normally, when a child process is killed, the parent process is told through a SIGCHILD signal. The parent can do some other task or restart a new child as needed. However, sometimes the parent process is killed. In this case the "parent of all processes", init, becomes the new PPID, parent process ID. You can see this by a process ID of 1 as the PPID of some other process.

Process signals

- If you have trouble killing a process, you can use the command
  `kill -9 PID`
- To see a list of signals you can send to a process, enter *(option is el, not a one)*
  `kill –l`
- A signal is a special value that can be sent to and from a command.

Zombies

When a process is killed, a ps listing may still show the process with a Z state. This is a zombie, or defunct process. The process is dead and not being used. In most cases, the process disappears in a few moments and the ps output will no longer show it. In rare instances, due to buggy software and other out-of-date utilities, you may find that some processes just won't die, even if you send a -SIGKILL signal to them. If a process is hung, first try sending a couple of -SIGTERM signals. If you wait and then verify that the process has not yet quit, try sending a -SIGKILL signal. If you find that the process stubbornly refuses to die, you may need to reboot the system.

Process Management

- The heart of managing the processor comes down to two related issues:
  - Ensuring that each process and application receives enough of the processor's time to function properly.
  - Using as many processor cycles for real work as is possible.
- The basic unit of software that the operating system deals with in scheduling the work done by the processor is either a process or a thread, depending on the operating system.
- It's tempting to think of a process as an application, but that gives an incomplete picture of how processes relate to the operating system and hardware. The application you see (word processor or spreadsheet or game) is, indeed, a process, but that application may cause several other processes to begin, for tasks like communications with other devices or other computers. There are also numerous processes that run without giving you direct evidence that they ever exist. For example, Windows XP and UNIX can have dozens of background processes running to handle the network, memory management, disk management, virus checking and so on.
- A process, then, is software that performs some action and can be controlled -- by a user, by other applications or by the operating system.
- It is processes, rather than applications, that the operating system controls and schedules for execution by the CPU. In a single-tasking system, the schedule is straightforward. The operating system allows the application to begin running, suspending the execution only long enough to deal with **interrupts** and user input.
- Interrupts are special signals sent by hardware or software to the CPU. It's as if some part of the computer suddenly raised its hand to ask for the CPU's attention in a lively meeting. Sometimes the operating system will schedule the priority of processes so that interrupts are masked -- that is, the operating system will ignore the interrupts from some sources so that a particular job can be finished as quickly as possible. There are some interrupts (such as those from error conditions or problems with memory) that are so important that they can't be ignored. These **non-maskable interrupts** (NMIs) must be dealt with immediately, regardless of the other tasks at hand.

- While interrupts add some complication to the execution of processes in a single-tasking system, the job of the operating system becomes much more complicated in a multi-tasking system. Now, the operating system must arrange the execution of applications so that you believe that there are several things happening at once. This is complicated because the CPU can only do one thing at a time. In order to give the appearance of lots of things happening at the same time, the operating system has to switch between different processes thousands of times a second. Here's how it happens:
  - A process occupies a certain amount of RAM. It also makes use of registers, stacks and queues within the CPU and operating-system memory space.
  - When two processes are multi-tasking, the operating system allots a certain number of CPU execution cycles to one program.
  - After that number of cycles, the operating system makes copies of all the registers, stacks and queues used by the processes, and notes the point at which the process paused in its execution.
  - It then loads all the registers, stacks and queues used by the second process and allows it a certain number of CPU cycles.
  - When those are complete, it makes copies of all the registers, stacks and queues used by the second program, and loads the first program.
- All of the information needed to keep track of a process when switching is kept in a data package called a process control block. The process control block typically contains:
  - An ID number that identifies the process
  - Pointers to the locations in the program and its data where processing last occurred
  - Register contents
  - States of various flags and switches
  - Pointers to the upper and lower bounds of the memory required for the process
  - A list of files opened by the process
  - The priority of the process
  - The status of all I/O devices needed by the process
- Each process has a status associated with it. Many processes consume no CPU time until they get some sort of input. For example, a process might be waiting on a keystroke from the user. While it is waiting for the keystroke, it uses no CPU time. While it is waiting, it is "suspended". When the keystroke arrives, the OS changes its status. When the status of the process changes, from pending to active, for example, or from suspended to running,

the information in the process control block must be used like the data in any other program to direct execution of the task-switching portion of the operating system.

- This process swapping happens without direct user interference, and each process gets enough CPU cycles to accomplish its task in a reasonable amount of time. Trouble can come, though, if the user tries to have too many processes functioning at the same time. The operating system itself requires some CPU cycles to perform the saving and swapping of all the registers, queues and stacks of the application processes. If enough processes are started, and if the operating system hasn't been carefully designed, the system can begin to use the vast majority of its available CPU cycles to swap between processes rather than run processes. When this happens, it's called thrashing, and it usually requires some sort of direct user intervention to stop processes and bring order back to the system.
- One way that operating-system designers reduce the chance of thrashing is by reducing the need for new processes to perform various tasks. Some operating systems allow for a "**process-lite**", called a **thread**, that can deal with all the CPU-intensive work of a normal process, but generally does not deal with the various types of I/O and does not establish structures requiring the extensive process control block of a regular process. A process may start many threads or other processes, but a thread cannot start a process.
- So far, all the scheduling we've discussed has concerned a single CPU. In a system with two or more CPUs, the operating system must divide the workload among the CPUs, trying to balance the demands of the required processes with the available cycles on the different CPUs. Asymmetric operating systems use one CPU for their own needs and divide application processes among the remaining CPUs. Symmetric operating systems divide themselves among the various CPUs, balancing demand versus CPU availability even when the operating system itself is all that's running.
- Even if the operating system is the only software with execution needs, the CPU is not the only resource to be scheduled. Memory management is the next crucial step in making sure that all processes run smoothly.

Display Linux tasks with **top** command

Send a job to the background with **bg** command

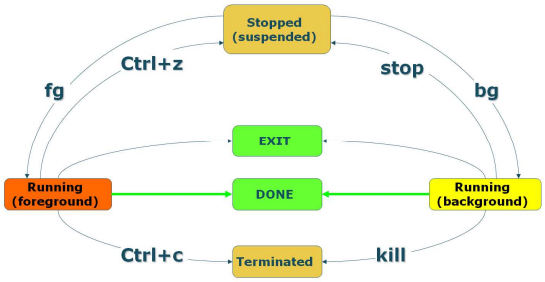
Send a job to the foreground with **fg** command

View running or stopped processes with **jobs** command

- View running or stopped jobs associated with your terminal

Job Control

- Commands can run in foreground (command at the prompt and <Enter>), or background (ending command with &). Background processes have lower priority.
- The Unix command-line shell can be used to run programs in the background so you can run multiple programs at a time. It also can be used to suspend commands and restart suspended commands. To tell the shell to run a given command in the background, simply append an ampersand to the end of the command line. For example, to run sleep in the background for 60 seconds:
  `sleep 60 &`
- The sleep tool basically waits a set amount of time and then exits successfully. By using the ampersand, the next command-line shell prompt is displayed and is displayed and is usable immediately for running other commands.



- A job is a process that is not running in the foreground and is accessible only at the terminal with which it is associated (typically background or suspended processes).
- Most shells have a built-in jobs command that can be used to show running shell jobs. A command running in the background is not waiting for console input, but may possibly be outputting text to the console. A command running in the foreground can have full interaction.

- You can move a job from the background to the foreground by running the built-in fg command. If you have multiple jobs, you can set the job number as the fg argument:
  `fg %1`
- Once you bring a command to the foreground, the shell's prompt does not display until the process is ended, and you can't run another command until then.
- The shell also enables you to suspend a currently running foreground process by pressing the suspend keystroke combination Ctrl+Z by default.

Monitoring Disk and Memory Resources

Report file system disk space usage with **df** command

- The df command displays file system free space
- In the following example, the –m option modifies the output to list sizes in Mb, rather than the default Kb.
  `df –m`

Estimate file space usage with **du** command

- The du command displays disk usage of directories and files
- In the following examples, the –h option modifies the output to display human readable units
  `du –h somefile.txt`
  `du –h /some/directory`

Display amount of memory with **free** command

- The command free display the amount of free and used memory in the system
  `free`

Contents