# LAB MANUAL
# FOR
# OS LAB

# Operating System

An Operating Sytem is a program that manages the computer hardware. It also provides a basisfor application programs and acts as an intermediary between a user of a computer and the computer hardware.

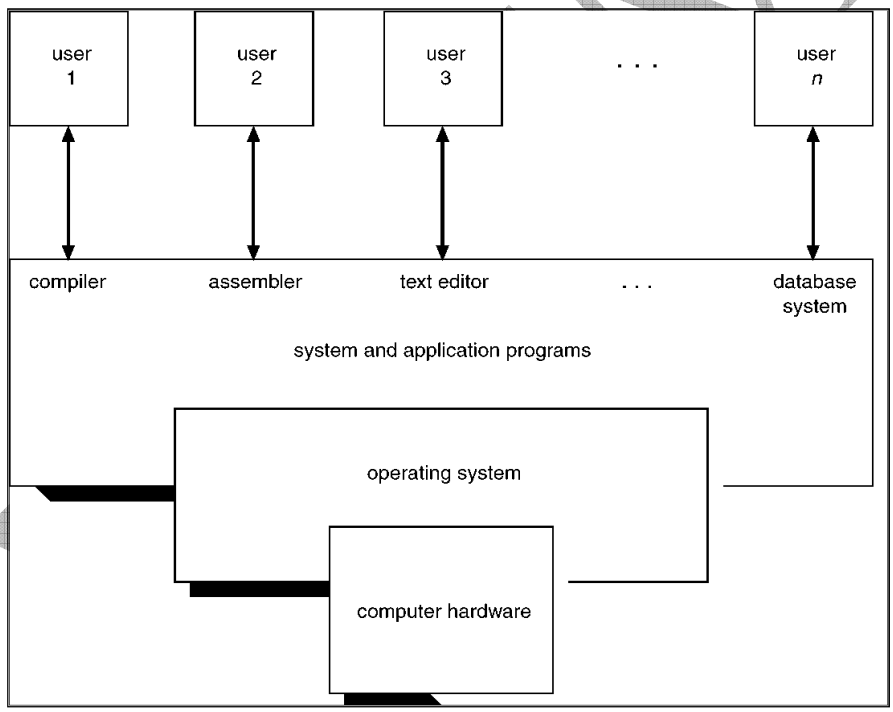
Goals of Operating system :

→Execute user programs and make solving user problems easier.
→Make the computer system convenient to use.
→Use the computer hardware in an efficient manner.

## Computer System Components

1. Hardware – provides basic computing resources (CPU, memory, I/O devices).
2. Operating system – controls and coordinates the use of the hardware among the various application programs for the various users.
3. Applications programs – define the ways in which the system resources are used to solve the computing problems of the users (compilers, database systems, video games, business programs).
4. Users (people, machines, other computers).

## Abstract View of System Components




## Operating System Definitions

**1. Resource allocator** – manages and allocates resources.
**2. Control program** – controls the execution of user programs and operations of I/O devices .
**3. Kernel** – the one program running at all times (all else being application programs).

# Windows 2000

The Microsoft Windows 2000 operating system is a 32-bit preemptive multitasking operating system for Intel Pentium & later microprocessors.

- Key goals for the system:
  - Portability
  - security
  - POSIX compliance
  - multiprocessor support
  - extensibility
  - international support
  - compatibility with MS-DOS and MS-Windows applications.
- Uses a micro-kernel architecture.
- Available in four versions, Professional, Server, Advanced Server, National Server.
- In 1996, more NT server licenses were sold than UNIX licenses

## History

- In 1988, Microsoft decided to develop a "new technology" (NT) portable operating system that supported both the OS/2 and POSIX APIs.
- Originally, NT was supposed to use the OS/2 API as its native environment but during development NT was changed to use the Win32 API, reflecting the popularity of Windows 3.0.

## Design Principles

- **Extensibility** — layered architecture.
  - Executive, which runs in protected mode, provides the basic system services.
  - On top of the executive, several server subsystems operate in user mode.
  - Modular structure allows additional environmental subsystems to be added without affecting the executive.

- **Portability** — 2000 can be moved from on hardware architecture to another with relatively few changes.
  - Written in C and C++.
  - Processor-dependent code is isolated in a dynamic link library (DLL) called the "hardware abstraction layer" (HAL).

- **Reliability** — 2000 uses hardware protection for virtual memory, and software protection mechanisms for operating system resources.

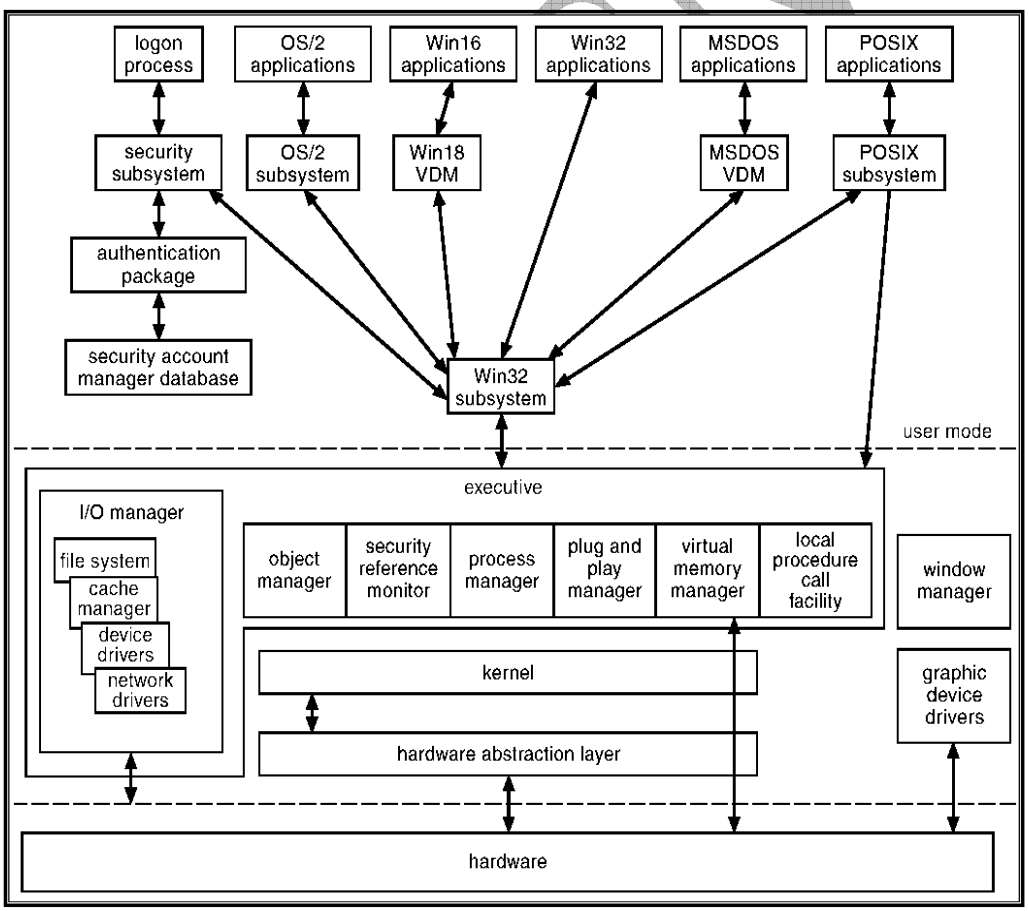
- **Compatibility** — applications that follow the IEEE 1003.1 (POSIX) standard can be complied to run on 2000 without changing the source code.

- **Performance** — 2000 subsystems can communicate with one another via high-performance message passing.
  - Preemption of low priority threads enables the system to respond quickly to external events.
  - Designed for symmetrical multiprocessing

- **International support** — supports different locales via the national language support (NLS) API.

## Windows 2000 Architecture

The architecture of Windows 2000 is Layered system of modules. The main layers are the HAL, kernel, executive all of which run in protected mode, and a large collection of subsystem that run in protected mode. User mode — collection of subsystems. The two categories : Environmental subsystems emulate different operating systems ; Protection subsystems provide security functions.



# System Components

## Kernel

The kernel of Windows 2000 provides the foundation for the executive and the subsystems.The Kernel is never paged out of memory; execution is never preempted. It has four main responsibilities:

- Thread scheduling
- Interrupt and exception handling
- Low-level processor synchronization
- Recovery after a power failure

The Kernel is object-oriented. An Object type in Windows 2000 is a system defined data type that has a set of attributes and a set of methods. The uses two sets of objects.
*Dispatcher objects* control dispatching and synchronization (events, mutants, mutexes, semaphores, threads and timers).
*Control objects* (asynchronous procedure calls, interrupts, power notify, power status, process and profile objects.)

### 1. Kernel Process & Threads :

The Kernel process has a virtual memory address space, information (such as a base priority), and an affinity for one or more processors.
Kernel threads are the unit of execution scheduled by the kernel's dispatcher.
Each thread has its own state, including a priority, processor affinity, and accounting information.
A thread can be one of six states: *ready, standby, running, waiting , transition,*and *terminated*.

### 2. Kernel — Scheduling

The dispatcher uses a 32-level priority scheme to determine the order of thread execution.
Priorities are divided into two classes..
The real-time class contains threads with priorities ranging from 16 to 31.
The variable class contains threads having priorities from 0 to 15.
The Characteristics of 2000's priority strategy is **T**rends to give very good response times to interactive threads that are using the mouse and windows , **E**nables I/O-bound threads to keep the I/O devices busy , **C**omplete-bound threads soak up the spare CPU cycles in the background.

Scheduling can occur when a thread enters the ready or wait state, when a thread terminates, or when an application changes a thread's priority or processor affinity.

Real-time threads are given preferential access to the CPU; but 2000 does not guarantee that a real-time thread will start to execute within any particular time limit. (This is known as *soft realtime*.)

### 3 Kernel — Trap Handling

The kernel provides trap handling when exceptions and interrupts are generated by hardware of software.Exceptions that cannot be handled by the trap handler are handled by the kernel's *exception dispatcher*.

The interrupt dispatcher in the kernel handles interrupts by calling either an interrupt service routine (such as in a device driver) or an internal kernel routine.
The kernel uses spin locks that reside in global memory to achieve multiprocessor mutual exclusion.

# Executive

## 1. Object Manager

Windows 2000 uses objects for all its services and entities; the object manger supervises the use of all the objects.

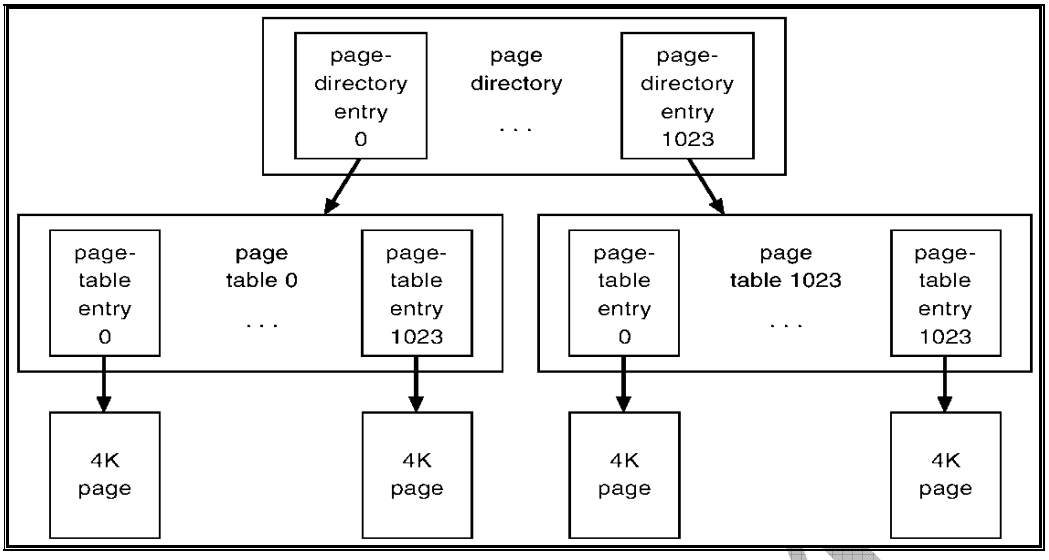
- Generates an object *handle,*
- Checks security.
- Keeps track of which processes are using each object.
- Objects are manipulated by a standard set of methods, namely create, open, close, delete, query name, parse and security.

The Windows 2000 executive allows any object to be given a name, which may be either permanent or temporary.Object names are structured like file path names in MS-DOS and UNIX.2000 implements a *symbolic link object*, which is similar to *symbolic links* in UNIX that allow multiple nicknames or aliases to refer to the same file.A process gets an object handle by creating an object by opening an existing one, by receiving a duplicated handle from another process, or by inheriting a handle from a parent process.Each object is protected by an access control list.

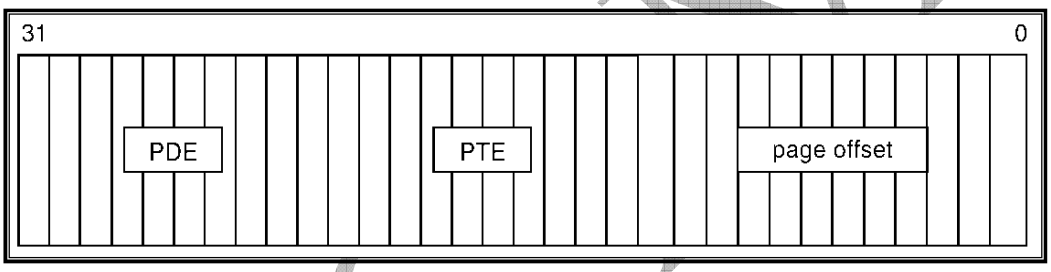
## 2. Virtual Memory Manager

The Virtual memory portion of Windows 2000 executive is the virtual memory Manager.The design of the VM manager assumes that the underlying hardware supports virtual to physical mapping a paging mechanism, transparent cache coherence on multiprocessor systems, and virtual addressing aliasing.
The VM manager in 2000 uses a page-based management scheme with a page size of 4 KB.
The 2000 VM manager uses a two step process to allocate memory :
The first step reserves a portion of the process's address space.
The second step commits the allocation by assigning space in the 2000 paging file.

The virtual address translation in 2000 uses several data structures.
Each process has a *page directory* that contains 1024 *page directory entries* of size 4 bytes.
Each page directory entry points to a *page table* which contains 1024 *page table entries* (PTEs) of size 4 bytes.
Each PTE points to a 4 KB *page frame* in physical memory.
A 10-bit integer can represent all the values form 0 to 1023, therefore, can select any entry in the page directory, or in a page table.
This property is used when translating a virtual address pointer to a bye address in physical memory.
A page can be in one of six states: valid, zeroed, free standby, modified and bad.




Virtual-Memory Layout




Virtual-to-Physical Address Translation

10 bits for page directory entry, 20 bits for page table entry, and 12 bits for byte offset in page.

## 3. Process Manager

- Provides services for creating, deleting, and using threads and processes.
- Issues such as parent/child relationships or process hierarchies are left to the particular environmental subsystem that owns the process.

## 4. Local Procedure Call Facility

The LPC passes requests and results between client and server processes within a single machine.In particular, it is used to request services from the various 2000 subsystems.When a LPC channel is created, one of three types of message passing techniques must be specified.

- First type is suitable for small messages, up to 256 bytes; port's message queue is used as intermediate storage, and the messages are copied from one process to the other.
- Second type avoids copying large messages by pointing to a shared memory section object created for the channel.
- Third method, called *quick* LPC was used by graphical display portions of the Win32 subsystem.

## 5. I/O Manager

The I/O manager is responsible for

- file systems
- cache management
- device drivers
- network drivers

Keeps track of which installable file systems are loaded, and manages buffers for I/O requests.Works with VM Manager to provide memory-mapped file I/O.Controls the 2000 cache manager, which handles caching for the entire I/O system.Supports both synchronous and asynchronous operations, provides time outs for drivers, and has mechanisms for one driver to call another.

## 6. Security Reference Monitor

The object-oriented nature of 2000 enables the use of a uniform mechanism to perform runtime access validation and audit checks for every entity in the system. Whenever a process opens a handle to an object, the security reference monitor checks the process's security token and the object's access control list to see whether the process has the necessary rights.

## 7. Plug-and-Play Manager

The Plug-and-Play (PnP) manager is used to recognize and adapt to changes in the hardware configuration.When new devices are added (for example, PCI or USB), the PnP manager loads the appropriate driver.The manager also keeps track of the resources used by each device.

# LDAP(Lightweight Directory Access Protocol)

**Lightweight Directory Access Protocol (LDAP) is an established Internet standard that enables cross-network operating system interoperability between directory services that support it. In Windows 2000, LDAP is the primary way the Operating System accesses the Active Directory database. By using this open standard, Microsoft is enabling 3rd party vendors and other platforms (Unix) to be able to work in a Windows 2000 environment and use AD services.**

A lightweight protocol is any of a class of protocols designed for use on high-speed internetworks. High-Speed Transport Protocol (HSTP), Xpress Transfer Protocol (XTP), and Lightweight Directory Access Protocol (LDAP) are examples.

Lightweight protocols combine routing and transport services in a more streamlined fashion than do traditional network and transport layer protocols. This makes it possible to transmit more efficiently over high- speed networks, such as ATM or FDDI, and media, such as fiber-optic cable.

Lightweight protocols use various measures and refinements to streamline and speed up transmissions, such as using connection-oriented transmissions, such as (TCP/IP) and a fixed header and trailer size to save the overhead of transmitting a destination address with each packet.

Lightweight Directory Access Protocol (LDAP) is a subset of the X.500 protocol. LDAP clients are, therefore, smaller, faster, and easier to implement than are X.500 clients. LDAP is vendor-independent and works with, but does not require, X.500.

Contrary to X.500, LDAP supports TCP/IP, which is necessary for any type of Internet access. LDAP is an open protocol, and applications are independent of the of server platform hosting the directory.

The Active Directory is not an X.500 directory. Instead, it uses LDAP as the access protocol and supports the X.500 information model without requiring systems to host the entire X.500 overhead. The result is the high level of interoperability required for administering real-world, heterogeneous networks.

The Active Directory supports access via the LDAP protocol from any LDAP-enabled client. LDAP names are less intuitive than Internet names, but the complexity of LDAP naming is usually hidden within an application. LDAP names use the X.500 naming convention called "Attributed Naming."

An LDAP URL names the server holding Active Directory services and the Attributed Name of the object. For example:

```
LDAP://SomeServer.Myco.Com/CN=jamessmith,CN=Sys,CN=Product,CN
=Division,DC=myco,DC=domain-controller
```

LDAP C API (RFC 1823) is an informational RFC that is the de facto standard in C programming for LDAP applications.

By combining the best of the DNS and X.500 naming standards, LDAP, other key protocols and a rich set of APIs, the Active Directory allows a single point of administration for all resources, including: files, peripheral devices, host connections, databases, Web access, users, arbitrary other objects, services, and network resources.

# DNS(Domain Name System)

Windows 2000 includes many new DNS bells and whistles. The DNS server itself is much improved, with more features than ever that make it more functional and easier to manage. From a client perspective, Windows 2000 as an operating system is more dependent on DNS than any previous operating system from Microsoft. And then there's Active Directory....

## Active Directory

Active Directory is *the* major new feature of Windows 2000. It's a hierarchical database of information about all objects in the network: computers, printers, users, and so on. Both users and computers access the information in Active Directory. The Active Directory database is partitioned into *domains* for administrative purposes, and one or more *domain controllers* store information about particular domains. (Compare this to DNS's namespace, which is partitioned into zones, with one or more name servers authoritative for each zone.) The most important fact about Active Directory for our purposes is that it is integrated tightly with DNS. For more--much more--information about Active Directory, see *Windows 2000 Active Directory* by Alistair G. Lowe-Norris (O'Reilly).

## Active Directory Domain Names

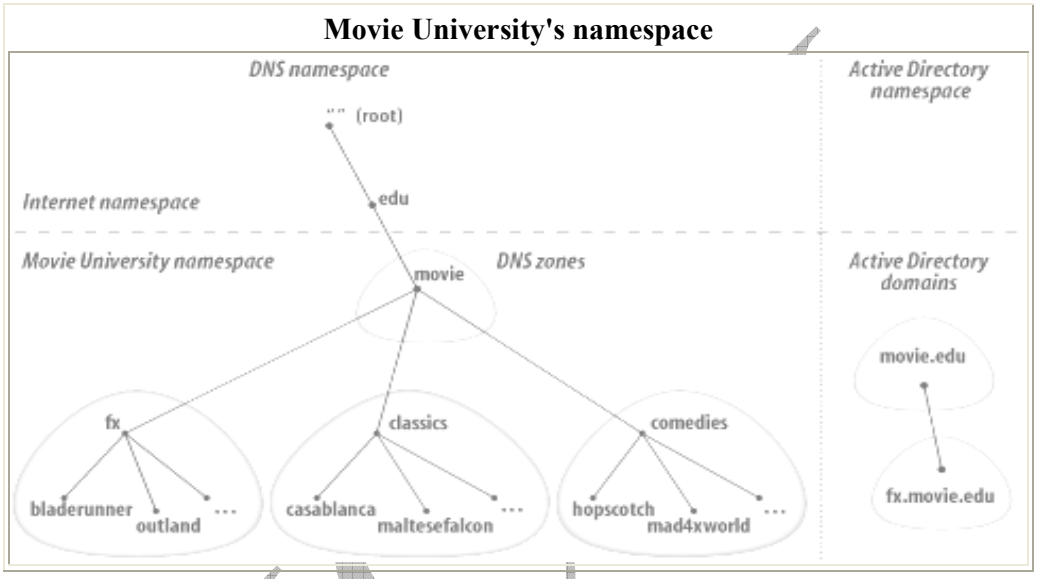
The most obvious connection between Active Directory and DNS is the naming of domains--Active Directory domains, that is. In the past, under Windows NT, domain names followed the NetBIOS host-naming rules: names consisted of a single label (i.e., no dots) and could contain letters, digits, and limited punctuation. Most Windows dialog boxes forced domain-name input to uppercase, so while they were case-insensitive, you usually saw domain names written in all uppercase. For example, Movie University's Windows NT domain name was *MOVIEU*.

With Windows 2000, all Active Directory domain names are DNS domain names, but--and this is important--not every DNS domain name is an Active Directory domain name.[1] So while an organization's Active Directory namespace resembles its DNS namespace, the two don't have to be and probably won't be identical. While it's beyond the scope of this book to give an exhaustive explanation of Active Directory namespace design, we can give you some examples to clarify the connection between the naming of Active Directory domains and DNS domains.

Consider Movie University. After reading this far, you're familiar with Movie U.'s DNS namespace: the apex (or top) of the namespace is *movie.edu*, and there are subdomains named *fx.movie.edu*, *classics.movie.edu*, and *comedies.movie.edu*.

Now let's talk about Movie U.'s Active Directory namespace. An organization's Active Directory domain names correspond to some of its DNS domain names, and the Active Directory domain at the top of an organization's domain tree usually corresponds to a subdomain of the apex of its DNS namespace. In Movie U.'s case, however, the root of the Active Directory domain tree is the same as the apex of the DNS namespace, *movie.edu*. Active Directory domain tree beside its DNS namespace.

Note how the two diverge, though. For various administrative reasons, the folks over in *fx.movie.edu* need to run their own Active Directory domain. But everyone else at Movie U. is a part of the *movie.edu* Active Directory domain, even though individual hosts fall into different DNS domains.




## DNS as Location Broker

You may be wondering why Active Directory domain names are DNS domain names. The answer is that Windows 2000 systems (running in native mode) use DNS as a *location broker*; that is, to find services. Previous versions of Windows used NetBIOS to find domain controllers, but Windows 2000 hosts use DNS. Take the case of a Windows 2000 Professional host at Movie U. that's been joined to the *movie.edu* Active Directory domain. When this system boots up, it sends a series of DNS queries to its configured name server to find a domain controller for the *movie.edu* domain.

# Study of LINUX Operating System ( Kernel, Shell, Basic Commands, Pipe & Filter commands )

## What is Linux?

The primary author of Linux is Linus Torvalds. Since his original versions, it has been improved by countless numbers of people.
Linux is a freely distributed, multitasking, multiuser operating system that behaves like UNIX. The term "Linux" is actually somewhat vague. "Linux" is used in two ways: specifically to refer to the kernel itself –the heart of any version of Linux –and more generally to refer to any collection of applications that run on the kernel, usually referred to a distribution. The kernel's job is to provide the basic environment in which applications can run, including the basic interfaces with hardware.
Today, thousands of software developers are busy upgrading it, all the while, and extending support online.

Today, Linux is used for a variety of applications. This includes:

- File and Print Server
- E-mail Server
- Fax Server
- Internet gateway
- Firewall
- Database Server
- ISP Server
- Application Server
- Desktop OS

## Basic features of Linux

Linux systems excel in many areas, ranging from end user concerns such as stability, speed, and ease of use, to serious concerns such as development and networking.
The important features are listed here.

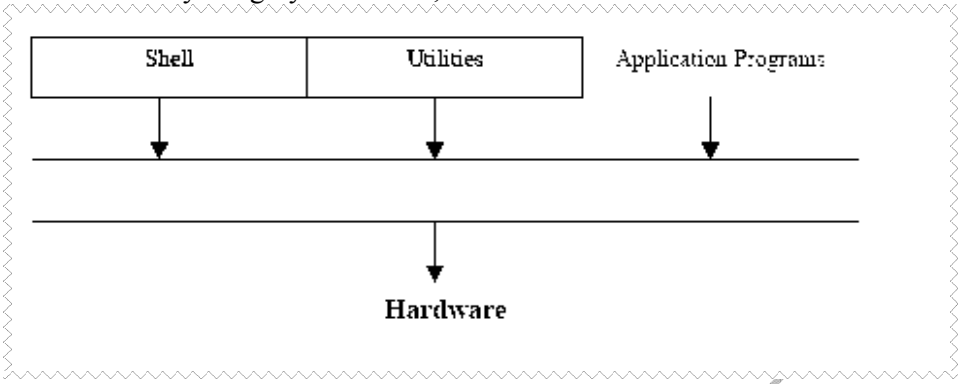
- Multi-programming
- High Speed
- POSIX compliance
- Dos emulator
- Data archiving utilities
- Time Sharing
- Virtual memory
- X concept
- CRON scheduler
- Licensing
- Multi-tasking
- Shared libraries
- Samba
- Office suits
- Web server
- Network information service(NIS)
- Support for programming languages
- Text processing and Word processing

## Linux architecture

### 1 . Linux kernel

The central nervous system of Linux is the kernel, the operating system code that runs the whole computer. The kernel is under constant development and is always available in both the latest stable release and the latest experimental release. Progress on development is very fast, and the recent 2.2-series kernels are simply amazing on all counts. The kernel design is modular, so that the actual OS code is very small, yet able to load whatever functionality it

needs when it needs it, and then free the memory afterwards. Because of this, the kernel remains small and fast yet highly extensible, in



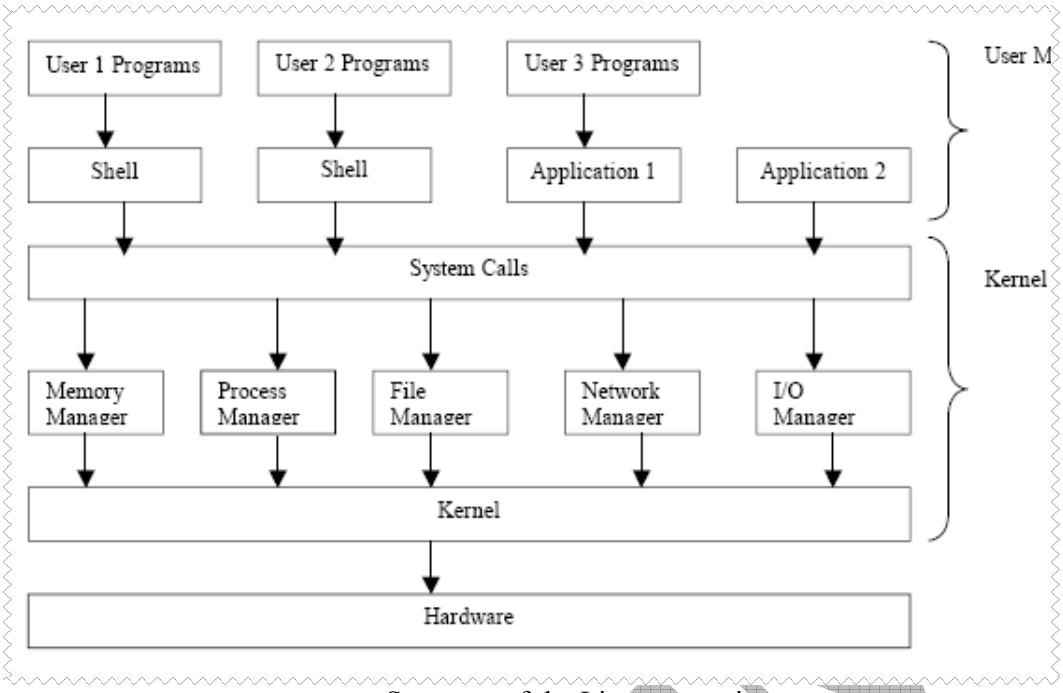

comparison to other operating systems which slow down the computer and waste memory by loading everything all the time, whether you need it or not.
The Linux kernel is the heart of Linux operating system and was originally developed for the Intel 80386 CPU's. Memory management is especially strong with the 80386 (compared to earlier CPU's). Linux kernel has the ability to have full access to the entire hardware capabilities of the machine. Actually there is no restriction on what a kernel module is allowed to do. Typically a kernel might implement a device driver, a file system or a networking protocol.
To support large memory requirements when only small amounts of physical RAM are available, Linux supports *swap space*. Swap space (discussed in detail in Book II) allows pages of memory to be written to a reserved area of a disk and treated as an extension of physical memory. By moving pages back and forth between the swap space and RAM, Linux can effectively behave as if it had much more physical RAM than it does. Besides this, Linux kernel support the following features:

- Has **memory protection** between processes, so that one program can't bring the whole system down.
- **Demand loads executables**: Linux only reads from disk those parts of a program that are actually used. More about this is discussed in the Session 2.

## 2. Linux Shell

A shell is a program that runs every time you log in. It is a program that lets you interact with the machine. Most users don't even notice the fact that they're running a shell. They just know that a prompt appears and that from there, they can read their mail and perform other tasks. This prompt is the shell that waits for you to tell what to do. Thus, Linux shell is a user program that the kernel runs when you log in. The shell acts as an interface between the user and the Linux operating system and helps in command interpretation for the kernel.

The figure shows that each user gets a copy of the shell where he is allowed to work. The shell interacts with the kernel through system calls. System calls are a set of routines that allow an application to access kernel services.
The structure of the Linux operating system is as follows:




Structure of the Linux operating system

## Shells available in Linux

We have several different shells available here:

**sh**

The first shell, historically, was sh, also known as the Bourne shell. It is good for writing shell scripts, but not so popular for interactive use.

**csh**

Also known as C-Shell, csh features a syntax somewhat like the C language. It allows (among other things) adding your own commands (aliasing), history substitution (reexecution of previously typed commands), and filename completion.

**tcsh**

This shell allows you to edit your command line while you're typing it, using emacslike commands. It has a number of other nifty features, but is otherwise compatible with csh.

**zsh**

The z shell has the best of the features of the Tcsh shell. It also has the capability to emulate all the features of the Korn shell and supports a large number of utilities and a detailed documentataion.

**bash**

Bash is an acronym for 'Bourne again shell'. It is an enhancement to the Bourne shell and is the default shell for most of the Linux systems. Compatible with *sh* for programming purposes, it has many of the good features of *csh* and *tcsh*: file name completion, job control, history substitution, emacs command-line editing, and many more.

**Pdksh**

Pdksh stands for public domain Korn shell and is an enhancement of the Korn shell. It has been written by several voluntary programmers. On Linux systems, ksh is the symbolic link to the *pdkh* shell.

## Basic Linux commands

❖ **The date command**

Linux maintains a system clock. As for now you can simply display the current date with the date command, which shows the
date and time for the nearest second as shown:

*$ date*
*Thur Nov 4 11:23:52 IST 1999*

The other format specifiers are as follows:
*D Specifies the day of the month*
*Y Indicates the last two digits of the year*
*H Indicates the hour*
*M Indicates the minute*
*S Indicates the second*
*T Indicates the time in hh::mm:ss format*

❖ **The who command**

Linux maintains an account of all the current users of the system. A list of all the users is displayed by the **who** command. The who command produces a three column output .This indicates the number of users of the system with their login names in the first column. The second column shows the device names of their respective terminals. The third column indicates the date and time of logging in.
The –H option prints the column headers:
*$ who –H*

❖ **The tty command**

Linux treats even terminals as files. It is therefore reasonable to expect a command which tells you the device name of the terminal you are using with the help of the *tty* command.
*$ tty*
*/dev/ttyp1*
*$*

❖ **The cal command**

This command is used for printing the calendar of any particular month or the entire year.Any calendar from the year 1 to 9999 can be displayed with this command:
*$ cal 2000*

❖ **The man command**

Linux systems maintain an online documentation about each command so that you can get to know the complete description about a particular command. The command:
*$ man*
will give a description of the man command in general.
The *man* command is used to view the description of other commands. For this purpose the *man* command is followed by the *commandname*. It has the following format:
***$ man** commandname*

❖ **The finger utility**

This utility is used to display the status of all the users currently logged on to the Linux system. The finger utility without any parameter, displays a single line output for each user currently

logged on . It displays the information like the user's login name, full name, terminal name, write status, idle time, login time, the machine's address where the user is currently logged in and the office number. The write permission is displayed along with the terminal name as an asterisk(*). If the asterisk appears after the terminal name, it means that the write permission is denied. The syntax of the finger command is as follows:
*finger [options] [user name]*

❖ **The chfn utility**

The chfn (change your finger information) utility is used to change the user's finger information. The chfn utility checks for the user's information from the /etc/passwd file and allows the user to chane information. The syntax for this utility is as follows:
*chfn [options] [username]*
The chfn asks for the password of the user to authenticate the user for changing the *finger*information. If you do not want to change any particular information, you can press *Enter* at the corresponding prompt and proceed with the remaining parameters.

❖ **The head command**

The head command is used to display the top few records of the file. The syntax of the command is as follows:
*$ head [option] file*
The option is as follows:
*count Display the first count lines of file*
The easiest way to use this command is to specify a filename without specifying the number of lines to be displayed. If this is the case, the first ten records of the file are displayed.
*$ head tmp.lst*
will display the first ten records of the file *tmp.lst*.

❖ **The tail command**

The tail command displays the end few records of the file. If no line count is given, the tail command displays the last ten lines of the file. We also have an option to specify a count and select that many lines from the end of file.
Consider an example:
*$ tail –3 emp.lst*

❖ **The mesg command**

The user has the option to allow or disallow other users to write on his terminal. The two options available with the mesg command are as follows:
*mesg y*
This option allows other users to write to your terminal
*mesg n*
This option disallows other users to write to your terminal
If you type the *mesg* command without any option, it displays the status of your terminal,say for example,
*$ mesg*
*is y*

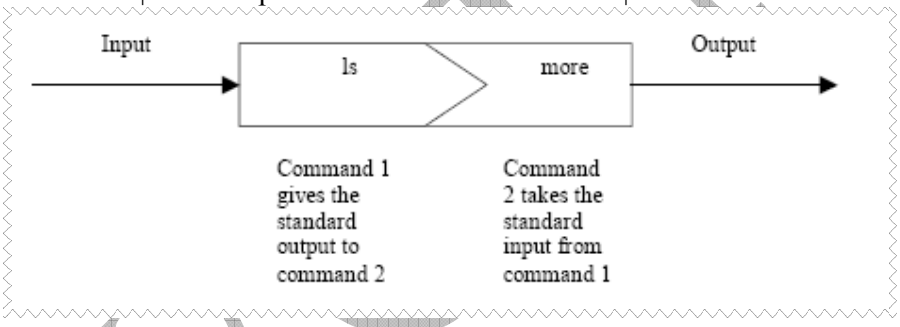
❖ **The wall command**

The wall command is used to write to all the users. The wall command sends the message to all the users who are currently logged on to the Linux system and have their *mesg* permission set to 'y'. The syntax of the command is as follows:

*wall Type in the above command at the command prompt. Press <Enter>.*
Now write the message you want to broadcast. *Press Ctrl <d>*. The message would be broadcasted to all the users currently logged on.

## Pipe Commands

If you have a series of commands in which the output of one command is the input of the next, you can pipe the output without saving it in temporary files:
*first_command | next_command*
For example, if you wanted to print out a sorted version of a file that contained a list of names and phone numbers, you could use a pipe (as well as input redirection):
*sort < my_phone_list | lpr*
Similarly, in order to display the contents of the current directory, a screen-full at a time,you can give the following commands:
*$ ls > myfile*
*$ more myfile*
Here, the listing of the directory is stored in the file, *myfile*, by the first command and this file is then used as input by the more command.
The above two steps can be combined and executed as a single command without creating a temporary file,
*$ ls | more*
The vertical bar (|) is the pipe character which indicates to the shell that the output of the command before the '|' is the input to the command after the '|'.




***Pipe***

## Filters Command

A filter is a program that takes its input from the standard input file, processes ( or filters)it and sends its output to the standard output file. Linux has a rich set of filters that can be used to work on data in an efficient way. Some examples of filters are:

- cat
- grep
- tr
- sort
- uniq
- sed
- awk

Some of the filters are discussed as follows:

### ❖ The grep command

The grep command searches a file for a specified pattern and displays it on the

screen. The syntax of this command is as follows:
*grep [options] regular expression filename[s]*

❖ **The tr command**

The *tr* command transliterates characters i.e. it copies the standard input to the standard output with substitution or deletion of specific characters. The command has the following syntax:
*tr [options] [string1] < file1*

The following is the list of options:
*c Complements the set of characters in string1 with respect to the universe of characters whose ASCII codes are 001 through 377 octal*
*d Deletes characters specified in string1*
*s Squeezes all repeated occurrences of a character to one character*

❖ **The sort command**

The sort command sorts lines of all the specified files together and writes the result on the standard output.
The general syntax of the command is as follows:
*sort [options][+pos1] [-pos2] filename[s]*
Say for example the command
*$sort myfile*
will sort the file considering each record of a file as one single field.

❖ **The uniq command**

There is often a problem of duplicate entries due to faulty data entry. The sort command suppresses them with the –u option. In addition to this there is a special tool to handle these records called the *uniq* command. The command is most useful when placed in pipelines.

❖ **The sed Command**

The sed command provides a stream editor.
The syntax of the command is as follows:
*sed [ -n ] Script [ File ... ]*
*sed [ -n ] [ -e Script ] ... [ -f ScriptFile ] ... [ File ... ]*
The **sed** command modifies lines from the specified *File* parameter according to an edit script and writes them to standard output. The **sed** command includes many features for selecting lines to be modified and making changes only to the selected lines.

**Alphabetical LINUX Command List:**

**addbib** - create or extend a bibliographic database
**apropos** - locate commands by keyword lookup
**ar** - create library archives, and add or extract files
**at** - execute a command or script at a specified time
**awk** - pattern scanning and processing language
**banner** - display a string in large letters
**basename** - display portions of path names and filenames
**batch** - runs jobs when the system load level permits
**biff** - give notice of incoming mail messages
**cal** - display a calendar
**calendar** - a simple reminder service
**cancel** - cancel requests to a printer
**cat** - concatenate and display
**cb** - a simple C program beautifier
**cc** - C compiler
**cd** - change working directory
**checknr** - check nroff and troff input files; report possible errors
**checkeq** - checks documents formatted with memoranda macros
**chgrp** - change the group ownership of a file
**chmod** - change the permissions mode of a file
**clear** - clear the terminal screen
**cmp** - perform a byte-by-byte comparison of two files
**colcrt** - filter nroff output for a terminal without overstrike capability
**comm** - selects or rejects lines common to two sorted files
**compress** - compress files (see uncompress also)
**cp** - copy files
**cpio** - copy file archives in and out
**cpp** - the C language preprocessor
**csh** - a shell with a C-like syntax and advanced interactive features
**ctags** - create a tags file for use with ex and vi
**cut** - Writes selected bytes, characters, or fields from each line of a file.

# System Administration Basics

So far in this book, you've seen how to use Linux for many different tasks. However, there are some issues we haven't dealt with because they are used rarely, or only by a single administrator (who may be the only user). This chapter looks at simple system administration tasks, including the following:

- Starting and shutting down the system properly
- Managing the disk partitions
- Making backups
- gzip, compress, and tar
- Message of the day
- Emergency boot floppies

Of course, we can't cover everything you need to know to run a system efficiently. Instead, we will look at the basic information and utilities and leave you to experiment. For more details, check the documentation files with your Linux operating system. Better yet, consider purchasing a good UNIX system administration book, such as Linux System Administrator's Survival Guide (Sams Publishing, 1995). Much of the information in a UNIX book will be applicable to Linux.

## The root Account

The root login, as you probably know, has no limitations at all. It can do anything anywhere, access any files it wants, and control any processes. This power has its price, though: Any mistake can be disastrous, sometimes resulting in damage to the entire operating system.

A mystique has built up in the UNIX community about the root login, because it holds unlimited power over the system. The tendency to want to use this superuser login is overwhelming for many. However, a simple rm command in the wrong place can spell many hours of trouble.

For this reason, the root account should be employed only for limited system use, and then only when its power is necessary (such as when rebuilding a kernel, installing new software, or setting up new file systems). As a general rule, you should not use the root account for routine tasks.

Naturally, many people use root for their daily Linux sessions, ignoring any advice because they think they won't make mistakes. In truth, everyone makes a mistake occasionally. Check with any UNIX system administrator and you'll find that accidents happen with the root account. (I have managed to delete entire file systems more than once while trying to do two things at the same time.) Although many people will ignore the rule about using root only when necessary, most of them eventually find out why this rule is important!

# Starting and Stopping the System

There are several ways of booting the Linux operating system, as well as a few ways to safely shut it down. Some were mentioned earlier in this book. Because Linux can be installed in many different ways, there is no single "right" method of booting the operating system, so we must look at both hard-disk-based and floppy-disk-based boot procedures.

## Booting from a Floppy

A boot floppy, as its name implies, is a floppy disk that boots the Linux kernel. A boot floppy has the root partition installed on the floppy itself instead of the hard drive (although both may co-exist). Without the root partition, Linux would be unable to find the hard drives for the rest of the operating system.

You can create Linux boot floppies with the setup routine included in most distributions of the operating system. Check the documentation or information files that came with your Linux distribution, if there are any. Alternatively, most Linux setup utilities have a menu-driven interface that prompts you for a boot floppy setup when you rebuild or reconfigure the kernel. You should use this procedure to make a boot floppy, which is also useful for emergencies.

In most cases, a boot floppy is used only in emergencies when your system won't start up normally. The boot floppy enables you to load Linux, and then mount the hard drives that are causing the problem to check for damage. Luckily, this is not required very often. If you haven't used LILO to choose the partition to boot or set your boot sequence to Linux by default, you may need the boot floppy to start up Linux. In this case, the boot floppy is much like a DOS boot floppy.

You can create a boot floppy from scratch by copying over the kernel image from the hard drive. The kernel image is usually in the file vmlinuz, vmlinux, Image, or /etc/Image, depending on the distribution of Linux. The Red Hat distribution uses vmlinuz, which is a compressed kernel (hence the z in the name). Compressed kernels uncompress themselves as they are loaded into memory at boot time. The vmlinuz image expands to vmlinux. (Compressed kernels take up less disk space; that's why they are used.)

After you have identified the kernel, you can set the root device in the kernel image to point to the root partition on either the floppy or hard drive. In this case, we want the floppy. The root partition is set with the rdev command, whose format is as follows:

```
rdev kernelname device
```

where kernelname is the name of the kernel image, and device is the name of the Linux root partition. To set a floppy boot device with the file vmlinuz, the command would be

```
rdev vmlinuz /dev/fd0
```

for the first floppy on the system. You can set other parameters with rdev as well if you want to change system defaults during boot. Check the rdev man page for the rdev help file for complete information.

As a final step in creating the boot floppy, copy the kernel image to the floppy disk. You should use a preformatted diskette (format with DOS if necessary) to allow the Linux routines to identify the type of diskette and its density. To copy the vmlinuz kernel to the first floppy drive, use this command:

```
cp vmlinuz /dev/fd0
```

The floppy should now be ready to boot the system. You might not be able to boot the system without the floppy if you changed the location of the root partition. You can change the root partition back to the hard drive with the rdev command after completing the boot floppy, which enables you to boot from either. This can be useful when you have diskettes for several different boot configurations. You can also create the boot floppy from the Linux setup program.

## Using LILO To Boot

LILO is a program that resides in the boot sector of your hard drive and allows Linux to be booted from the hard disk either after you tell it to or after a default number of seconds has elapsed.

LILO can also be used with other operating systems such as OS/2 and DOS. If you have LILO set to autoboot Linux, you must interrupt the process by pressing the Ctrl, Alt, or Shift keys when the bootup is started if you want to boot into another operating system. This displays a boot prompt that enables you to specify another operating system.

If LILO is set to allow a given time before it boots into Linux, you can use the Ctrl-Alt-Shift sequence to interrupt the boot process before the timer expires and Linux starts loading. Finally, if LILO is set to not autoboot into Linux, but to wait for

explicit instructions, you must press Enter to boot Linux or type the name of the other operating system.

Some Linux distributions have a configuration file in the directory /etc/lilo that can be edited to provide boot information, while other versions of Linux configure LILO during the installation process. If the latter is the case, you can change the settings with the setup utility. Some versions of Linux use the configuration file /etc/lilo.conf instead of /etc/lilo.

## Shutting Down Linux

You can't just turn off the power switch! This can cause damage to the file system, sometimes irreversibly. Because Linux keeps many files open at once, as well as several processes, they must all be closed down properly before you cycle the power to the unit.

There are a few ways to shut the Linux system down, but the formal method is to use the shutdown command. The syntax for shutdown is

```
shutdown [minutes] [warning]
```

where minutes is the number of minutes to wait before shutting the system down and warning is an optional message displayed for all users currently logged in. Some versions of shutdown allow the word now instead of a time, while others require either no argument or the number 0 to shut the system down immediately without waiting. You can have shutdown reboot the system after the shutdown by adding the argument -r (for reboot).

Using shutdown is best if you have other users on your system, because it gives them a warning that they should log out, and it prevents loss of information. It can also be used to automate a shut-down much later (such as at midnight), with messages just before that time warning any users still logged in.

If you can't wait and want to shut the system down immediately, use the halt command or the "three-finger salute" of Ctrl-Alt-Delete. This immediately shuts down all the processes and halts the system as quickly as possible. Then the power can be shut off.

---

Some Linux distributions don't support Ctrl-Alt-Delete, and a couple of older distributions use it to halt the system immediately without terminating processes properly. This can cause damage. Check the documentation or man pages for information.

---

## Creating a New File System

To create a file system on a floppy (so it can be mounted), you should use the utility mke2fs or the command mkdev fs, depending on the version of Linux. To use mke2fs, for example, issue the command

```
mke2fs /dev/fd0 1440
```

to create a floppy file system on a 1.44MB 3.5-inch diskette.

## Unmounting File Systems

To detach a mounted file system from your Linux file system, use the umount command with the name of the device. For example, to unmount a floppy in /dev/fd0, issue the command

```
umount /dev/fd0
```

and the floppy will be removed from the mounted point. Be sure to type umount instead of unmount!

If you want to remove the current floppy and replace it with another, you can't simply swap them. The current floppy must be unmounted, and then the new one must be mounted. Failure to follow this process can result in corruption or erroneous directory listings.

## Checking File Systems

Every now and again a file might get corrupted or a file system's inode table might get out of sync with the disk's contents. For these reasons, it is a good idea to check the file system at regular intervals. Several utilities can check file systems, depending on the version of Linux. The utility fsck is available for some systems, while the utility e2fsck is designed for Linux's ext2fs file system. Many Linux versions include other utilities such as xfsck and efsfck for different file systems. In many cases, the fsck command is linked to the individual file system versions.

To use e2fsck to check a file system, issue the command with the device name and the options a (automatically correct errors) and v (verbose output):

```
e2fsck -av /dev/hda1
```

This command checks and repairs any problems on the /dev/hda1 (or whatever device driver you specify) partition. If any corrections have been made to a partition, you should reboot the machine as soon as possible to allow the system to resync its tables.

Whenever possible, it is a good idea to unmount the file system before checking it, because this can prevent problems with open files. Of course, you can't unmount the primary root partition while running from it, so you can boot from a boot floppy that contains the check utilities, and start them from the floppy.

# Backups

The three rules of system administration are back up, back up, and back up. This might sound silly and trite, but a backup can save you whenever you do something silly to the file system, or when problems occur. With UNIX, most backups are made to a tape device using tar, although many Linux users don't have tape units available and have to resort to floppies.

Backups are made with the tar utility, as I mentioned earlier. The procedure is exactly the same as I showed you earlier. To back up the entire system on floppy, the command is

```
tar -cvfbk /dev/fd0 1440 4 /
```

To back up to a high-capacity tape device larger than the file system (and hence not needing a capacity limit) called /dev/rct0, the command is

```
tar -cvfk /dev/rct0 20 /
```

In many cases, you won't want to back up the entire system, because it's easier to reinstall off a CD-ROM. However, you should back up your user files by either backing up the entire /usr directory or specifically backing up your own home directory.

To restore a backup, you use the tar command again:

```
tar -xvf /dev/rct0
```

This recovers all files from the tape device /dev/rct0. You can explicitly restore specific files if you need to.

Several commercial products offer automated backups, although you can do this quite easily with the cron command.

# Setting Up Your System

You can perform several little tasks to tweak or optimize your Linux system, although in many cases they are dependent on the version you are running and other applications coexisting. We can look at a few of the miscellaneous tasks here.

## Setting the System Name

The system name is contained in a file called /etc/HOSTNAME. It is simply the name the system calls itself for identification, which is especially useful if you are networking your Linux machine with others. You can call the system anything you want.

To set your system name (also called a host name), you can either edit the system files (which should be followed by a reboot to make the changes effective) or use the hostname command. The command

```
hostname hellfire
```

sets the machine's name to hellfire.

## Using a Maintenance Disk

Every system should have a maintenance disk that enables you to check the root file system, recover from certain disk problems, and solve simple problems (such as forgetting your root password). The emergency disks, also called the boot/root floppies, are created with the setup program in most distributions of Linux when the configuration is changed.

You can usually create an emergency boot disk from the CD-ROM that the system came on, as well as obtain the necessary files from FTP sites.

After you have booted your machine with the emergency disk, you can mount the disk partitions with the mount command.

## Forgetting the root Password

This is an embarrassing and annoying problem, but luckily one easily fixed with Linux. (If only other UNIX systems were so easy!) To recover from a problem with the root password, use a boot floppy and boot the system. Mount the root partition, and edit the /etc/passwd file to remove any password for root; then, reboot from the hard disk.

After the system has booted, you can set a password again.

This points out one major security problem with Linux: Anyone with a boot floppy can get unrestricted access to your system!

### Setting the Login Message

If you have more than one user on the system, you can display information about the system, its maintenance, or changes in a file called /etc/motd (message of the day). The contents of this file are displayed whenever someone logs in.

To change the /etc/motd file, use any text editor and save the contents as ASCII. You can make the contents as long as you want, but readers usually appreciate brevity. The /etc/motd file is useful for informing users of downtime, backups, or new additions. You can also use it to give a more personal feel to your system.

## Summary

System administration is not a complicated subject, unless you want to get into the nitty-gritty of your operating system and its configuration. For most Linux users who use the operating system for their personal experimentation, the administration steps explained in this chapter should be sufficient for most purposes. If you want to get into more detail, check out a good UNIX system administration book.

# SHELL PROGRAMMING

## Ques 1 : To add two no.

```
echo Enter the 1st no.
read a
echo Enter the 2nd no.
read b
c='expr $a + $b'
echo Addition of two no. is $c
```

## Ques 2 : To find Odd or Even no.

```
a=2
b=0
echo Enter the no.
read x
b=’expr $x % $a’
if[$b –eq 0]
then
echo Given no. is Even
else
echo Given no. is Odd
```

## Ques 3 : To reverse a no.

```
echo Enter the no.
read x
while[$x –gt 0]
do
c=’expr $x % 10’
echo $c
x=’expr $x/10’
done
```

## Ques 4 : To print the fibonni cii series

```
count=1
f1=1
f2=1
k=0
echo $f1
echo $f2
while[$count –le 10]
do
k=’expr $f1 + $f2’
echo $k
f1=$f2
f2=$k
count=’expr $count + 1’
done
```

## Ques 5 : To find average of two numbers

```
echo Enter the 1st no.
read a
echo Enter the 2nd no.
read b
c=’expr $a + $b’
d=$c/2
echo The average of two no. is $d
```

# AWK programming

Awk is essentially a stream editor, like sed. You can pipe text to it, and it can manipulate it on a line-by-line basis. [or it can read from a file]. It is also a programming language. That basically means it can do anything sed can do, and a lot more. (But you might have to type more :-)

Unlike sed, it has the ability to remember context, do comparisons, and most things another full programming language can do. For example, it isn't just limited to single lines. It can JOIN multiple lines, if you do things right.

The simplest form of awk is a one-liner:

```
awk '{ do-something-here }'
```

The "do-something-here" can be a single print statement, or something much more complicated. It gets somewhat 'C'-like. Simple example:

```
awk '{print $1,$3}'
```

will print out the first and third columns, where columns are defined as "Things between whitespace". (whitespace==tab or space) Complicated example:

```
awk '{  if ( $1 = "start") {
                start=1;
                print "started";
                if ( $2 != "" ){
                        print "Additional args:",$2,$3,$4,$5
                }
                continue;
        }
        if ( $1 = "end") {
                print "End of section";
                printf ("Summary: %d,%d,%d (first, second, equal)\n",
                        firstcol, secondcol, tied);
                firstcol=0;
                secondcol=0;
                tied=0;
                start=0;
        }
        if ( start >0) {
                if ( $1 > $2 ) {
                        firstcol= firstcol+1
                }else
                if ( $2 > $1 ) {
                        secondcol= secondcol+1
                }else
                        tied=$tied+1
        }
}'
```

AWK's main goal in life is to manipulate its input on a line by line basis. An awk program usually goes through some version of
Process a line. Move on.
Process a line. Move on.
Process a line. ...

If what you want to do does not fit into that model, then awk may not be a good fit for what you want to do.

The general syntax used by all awk programing can be described as:

PATTERN {COMMAND(S)}

What this means is,

"For each line of input, go look and see if the PATTERN is present. If it is present, run the stuff between {}"

[If there is no pattern specified, the command gets called for EVERY line]

A specific example:

```
awk '/#/ {print "Got a comment in the line"}' /etc/hosts
```

will print out "Got a comment" for every line that contains at least one '#', **anywhere in the line**, in /etc/hosts

The '//' bit in the pattern is one way to specify matching. There are also other wasy to specify if a line matches. For example,

```
 $1 == "#" {print "got a lone, leading hash"}
```

will match lines that the first column is a single '#'. The '==' means an EXACT MATCH of the ENTIRE column1.

On the other hand, if you want a partial match of a particular column, use the '~' operator

```
  $1 ~ /#/ {print "got a hash, SOMEWHERE in column 1"}
```

**NOTE THAT THE FIRST COLUMN CAN BE AFTER WHITESPACE.**

Input of "# comment" will get matched
Input of " # comment" will ALSO get matched

If you specifically wanted to match "a line that begins with exactly # and a space" you should use

```
  /^# /  {do something}
```

## *Math tweaks*

```
  +, -, /, *, sin(), cos(), tan(), atan(), sqrt(), rand(), srand()
```

## *String manipulation*

index() will tell you if and where a string occurs in a substring.

match() is similar, but works for regular expressions.

sprintf() gives you a way to format output, and do conversions along the way. This should be familiar to anyone who has used printf() in C. For example,

```
   newstring=sprintf("one is a number %d, two is a string %s\n", one,
two);
   print  newstring
```

"%d" says "print the variable matching me, as a decimal number"
"%s" says "print the variable matching me, as a string"

So if you wanted to join two strings with no gaps, ONE way would be to use

```
    newstring=sprintf("%s%s", one,  two)
```

length() just gives you an easy way to count characters in a string, if you need that.

## *System level functions*

system() lets you call potentially ANY executable on the system. The target executable can be either in your $PATH, or you can specify it by absolute path.

For example, the painful

```
    system("rm -rf $HOME");
```

or

```
    system("/bin/kill 1")
```

If you want to do more complex things, you'll probably end up doing something like

```
   sysstring=sprintf("somecommand %s %s", arg1,  arg2);
   system(sysstring)
```

close() is an important function that is often overlooked. This is probably because there isn't an explicit open() call, so people don't expect to need a close() call. And for most purposes, you dont. But you DO NEED IT, if you are dealing with more than one output file.

Awk gives you the capability to open random files on the fly. For example

```
  /^file/       { print $3 >> $2 }
```

should take the line "file output here-is-a-word", open the file 'output', and print 'here-is-a-word' to it.

AWK is "smart", in that it keeps track of what files you open, and KEEPS them open. It assumes if you opened a file once, you are likely to do it again. Unfortunately, this means if you open LOTS of files, you may run out of file descriptors. So, when you

know you are done with a file, close it. So, to improve the above example, you might use something along the lines of:

```
/^file/       { if ( $2 !=  oldfile ) { close( oldfile) };
                print $3 >> $2 ;  oldfile = $2; }
```

## *Array concepts*

I previously didn't have a reason to use arrays, but since someone recently emailed me an example of when you could use arrays , I feel inclined to share it with folks.

We have previously covered variables, as a name that holds a value for you. Arrays are an extension of variables. Arrays are variables that hold more than one value. How can it hold more than one value? Because it says "take a number".

If you want to hold three numbers, you could say

```
value1="one"; value2="two"; value3="three";
```

OR, you could use

```
values[1]="one"; values[2]="two"; values[3]="three";
```

You must always have a value in the brackets[] when using a variable as an array type. You can pick any name for an array variable name, but from then on, that name can ONLY be used as an array. You CANNOT meaningfully do

```
values[1]="one";
values="newvalue";
```

You CAN reassign values, just like normal variables, however. So the following IS valid:

```
values[1]="1";
print values[1];
values[1]="one";
print values[1];
```

The really interesting this is that unlike some other languages, you dont have to just use numbers. The [1],[2],[3] above are actually treated as ["1"], ["2"], ["3"]. Which means you can also use other strings as identifiers, and treat the array almost as a single column database. This formal name for this is an "associative array".

```
numbers["one"]=1;
numbers["two"]=2;
print numbers["one"];
value="two";
print numbers[value];
value=$1;
if(numbers[value] = ""){ print "no such number"; }
```

## Straight output

Sometimes, you just want to use awk as a formatter, and dump the output stright to the user. The following script takes a list of users as its argument, and uses awk to dump information about them out of /etc/passwd.

**Note:** observe where I unquote the awk expression, so that the shell does expansion of $1, rather than awk.

```
#!/bin/sh

while [ "$1" != "" ] ; do
        awk -F: '$1 == "'$1'" { print $1,$3} ' /etc/passwd
        shift
done
```

## Setting a shell variable from awk output

Sometimes you just want to use awk as a quick way to set a value for a variable. Using the passwd theme, we have a way to grab the shell for a user, and see if it is in the list of official shells.

Again, be aware of how I unquote the awk expression, so that the shell does expansion of $1, rather than awk.

```
#!/bin/sh

user="$1"
if [ "$user" ="" ] ; then echo ERROR: need a username ; exit ; fi

usershell=`awk -F: '$1 == "'$1'" { print $7} ' /etc/passwd`
grep -l $usershell /etc/shells
if [ $? -ne 0 ] ; then
        echo ERROR: shell $usershell for user $user not in
/etc/shells
fi
```

Other alternatives:

```
# See "man regex"
usershell=`awk -F: '/^'$1':/ { print $7} ' /etc/passwd`

#Only modern awks take -v. You may have to use "nawk" or "gawk"
usershell=`awk -F: -v user=$1 '$1 == user { print $7} ' /etc/passwd`
```

The explaination of the extra methods above, is left as an excercise to the reader :-)

## In a pipe-line

Sometimes, you just want to put awk in as a filter for data, either in a larger program, or just a quickie one-liner from your shell prompt. Here's a quickie to look at the "Referrer" field of weblogs, and see what sites link to your top page many different types of web browsers come to look at your site.

```
#!/bin/sh

grep -h ' /index.html' $* | awk -F\" '{print $4}' | sort -u
```

## *AWK summary*

All the features I have mentioned, make AWK a fairly decent language. Its main drawback is that it is so line-oriented. It would be kind of nice to have a proceedural language with all the power of AWK. Which is why perl was invented.

Unfortunately, Larry then decided to go waaaaay beyond the simple concept, by throwing in the kitchen sink, AND destroying the cleanness of the AWK language syntax, all in the name of reducing the number of keystrokes needed to accomplish something. The extra functions in perl are good. The syntax, however, is disgusting to programmers capable of touch-typing more than 20 words a minute. Having the shortest number of characters to implement something, should not be the judge of value for a language.