BCA 306

LINUX ENVIRONMENT

UNIT – I

UNIX & LINUX:- Overview of UNIX and LINUX Architectures, UNIX Principles, GNU Project/FSF,GPL,Getting help in Linux with –help, whatis, man command, info command, simple commands like date,whoami, who, w, cal, bc ,hostname, name, concept of aliases etcLinux file system types ext2, ext3, ext4,Basic linux directory structure and the functions of different directories basic directory navigation commands like cd, mv, copy,rm,cat command , less command, runlevel (importance of /etc/inittab)[T1,T2,R1] [No. of Hrs: 11]

UNIT – II

Standard Input and Output, Redirecting input and Output, Using Pipes to connect processes, tee command, Linux File Security, permission types, examining permissions, changing permissions(symbolic method numeric method),default permissions and unmask Vi editor basics, three modes of vi editor, concept of inodes,inodes and directories,cp and inodes ,mv and inodes rm and inodes, symbolic links and hard links, mount and unmount command, creating archives, tar,gzip,gunzip,bzip2,bunzip2(basic usage of these commands)[T1,T2,R1]

[No. of Hrs: 11]

UNIT – III

Environment variables(HOME,LANG,SHELL,USER,DISPLAY,VISUAL),Local variables, concept of /etc/passwd, /etc/shadow, /etc/group, and su- command, special permissions(suid for an executable,sgid for an executable,sgid for a directory, sticky bit for a directory) tail, wc, sort, uniq, cut, tr, diff, aspell, basic shell scripts grep, sed, awk(basic usage) [T1,T2,R1]

[No. of Hrs: 11]

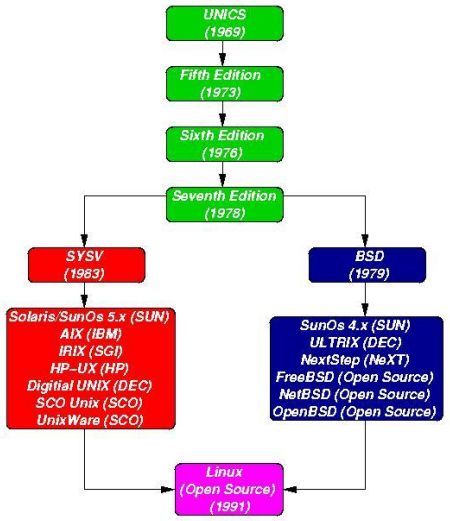
UNIT – IV

Process related commands(ps, top, pstree, nice, renice), Introduction to the linux Kernel, getting started with the kernel(obtaining the kernel source, installing the kernel source,using patches, the kernel source tree, building the kernel process management(process descriptor and the task structure, allocating the process descriptor, storing the process descriptor, process state, manipulating the current process state, process context, the process family tree, the Linux scheduling algorithm, overview of system calls, Introduction to kernel debuggers(in windows and linux)[T2]

UNIT I

OVERVIEW OF UNIX AND LINUX ARCHITECTURES

UNIX has been a popular OS for more than two decades because of its multi-user, multi-tasking environment, stability, portability and powerful networking capabilities.



In the late 1960s, researchers from General Electric, MIT and Bell Labs launched a joint project to develop an ambitious multi-user, multi-tasking OS for mainframe computers known as MULTICS (Multiplexed Information and Computing System). MULTICS failed (for some MULTICS enthusiasts "failed" is perhaps too strong a word to use here), but it did inspire Ken Thompson, who was a researcher at Bell Labs, to have a go at writing a simpler operating system himself. He wrote a simpler version of MULTICS on a PDP7 in assembler and called his attempt UNICS (Uniplexed Information and Computing System). Because memory and CPU power were at a premium in those days, UNICS (eventually shortened to UNIX) used short commands to minimize the space needed to store them and the time needed to decode them - hence the tradition of short UNIX commands we use today, e.g. ls, cp, rm, mv etc. Ken Thompson then teamed up with Dennis Ritchie, the author of the first C compiler in 1973. They rewrote the UNIX kernel in C - this was a big step forwards in terms of the system's portability - and released the Fifth Edition of UNIX to universities in 1974. The Seventh Edition, released in 1978, marked a split in UNIX development into two main branches: SYSV (System 5) and BSD (Berkeley Software Distribution). BSD arose from the University of California at Berkeley where Ken Thompson spent a sabbatical year. Its development was continued by students at Berkeley and other research institutions. SYSV was developed by AT&T and other commercial companies. UNIX flavors based on SYSV have traditionally been more conservative, but better

supported than BSD-based flavors.

Linux Architecture

- **Kernel:**The Linux kernel includes device driver support for a large number of PC hardware devices (graphics cards, network cards, hard disks etc.), advanced processor and memory management features, and support for many different types of file systems (including DOS floppies and the ISO9660 standard for CDROMs). In terms of the services that it provides to application programs and system utilities, the kernel implements most BSD and SYSV system calls, as well as the system calls described in the POSIX.1 specification.

The kernel (in raw binary form that is loaded directly into memory at system startup time) is typically found in the file /boot/vmlinuz, while the source files can usually be found in /usr/src/linux.The latest version of the Linux kernel sources can be downloaded from http://www.kernel.org.

- **Shells and GUIs** :Linux supports two forms of command input: through textual command line shells similar to those found on most UNIX systems (e.g. sh - the Bourne shell, bash - the Bourne again shell and csh - the C shell) and through graphical interfaces (GUIs) such as the KDE and GNOME window managers. If you are connecting remotely to a server your access will typically be through a command line shell.

- **System Utilities**: Virtually every system utility that you would expect to find on standard implementations of UNIX (including every system utility described in the POSIX.2 specification) has been ported to Linux. This includes commands such as ls, cp, grep, awk, sed, bc, wc, more, and so on. These system utilities are designed to be powerful tools that do a single task extremely well (e.g. grep finds text inside files while wc counts the number of words, lines and bytes inside a file). Users can often solve problems by interconnecting these tools instead of writing a large monolithic application program.

- **Application programs**: Linux distributions typically come with several useful application programs as standard. Examples include the emacs editor, xv (an image viewer), gcc (a C compiler), g++ (a C++ compiler), xfig (a drawing package), latex (a powerful typesetting language) and soffice (StarOffice, which is an MS-Office style clone that can read and write Word, Excel and PowerPoint files).

UNIX PRINCIPLES

Computer Terms

Memory contents are lost when the power goes, and memory is limited in size (usually only a few gigabytes in size). Data and programs are stored onto disk for later use and retrieval. A program is a set of instructions for the computer's CPU for manipulating data, including sending it to files, to the screen or to other bits of the computer.

The Kernel

In Unix (often spelt UNIX), the basic software controlling the hardware is know as the Kernel. The Kernel does all the difficult and nasty things like managing all the running processes (a name for a running program), the memory, the network connections, disks, tapes and virtually any bit of hardware on the computer.

Shells

A shell is a program which allows you to run other programs. Microsoft Windows has a shell called Explorer which allows you to run programs by clicking on them. Although there are graphical shells for UNIX, such as Gnome and KDE, they are not installed on the computing cluster.

Files

In UNIX, like in many environments, data are stored in things called files. A file contains data made up of a particular number of bytes. Many files in Unix are stored as text; often data in a program are translated into text for saving and loading. Other files, such as those your programs could write, or programs themselves, are encoded such that you can load them back into memory. Files have particular names, which can be hundreds of characters long, and are case sensitive, so that the file `Hello.txt' isn't the same as the file `hello.txt'. Conventionally files of a particular type have the same `suffix' or `extension', which is the bit at the end back to the dot.

Directories

Files are stored on disk (which may be a CDROM or a pretend disk in memory, like the /tmp file system). Files are organized in a `File System Hierarchy' or `Directory Tree' (it looks like a tree), which sounds a bit complex. A directory is a set of files with a name, which can also contain other directories. All the directories can be traced back to the root directory, `/'.

The Network File System (NFS) allows you to access your files on different computers.
Names of files are built up from the current working directory, or from root if an absolute name is given (starting with /). Directories are specified with slashes after them.

Permissions

Each file and each directory has an owner saying who has control of the file, and a group which can be several users, who also can have ac cess to the file. The important files on the computer are owned by the user `root' who is your local friend ly system administrator. Certain programs and files are restricted for use by root.

Processes

When you run a program, you load it into memory and it starts running as a `process'. Processes compete for the CPU time on the computer , according to their priority or `niceness'. Each process gets its own identification number (PID), which are recycled eventually. Processes are identified by this number in the `top' and `ps' process examination tools, and can be used to kill or stop the process with the `kill' command.

The X Window System

X is a system for displaying graphical applications on Unix (and other) systems. An X server runs on your computer, and receives requests to open windows, draw windows, accept input, and so on from a client application (which is the program you are running).

GNU PROJECT/FSF

GNU was launched by Richard Stallman (rms) in 1983, as an operating system which would be put together by people working together for the freedom of all software users to control their computing. rms remains the Chief GNUisance today. The primary and continuing goal of GNU is to offer a Unix-compatible system that would be 100% free software. Not 95% free, not 99.5%, but 100%. The name of the system, GNU, is a recursive acronym meaning GNU's Not Unix—a way of paying tribute to the technical ideas of Unix, while at the same time saying that GNU is something different. Technically, GNU is like Unix. But unlike Unix, GNU gives its users freedom. The GNU packages have been designed to work together so we could have a functioning GNU system. It has turned out that they also serve as a common “upstream” for many distros, so contributions to GNU packages help the free software community as a whole. Naturally, work on GNU is ongoing, with the goal to create a system that gives the greatest freedom to computer users. GNU packages include user-oriented applications, utilities, tools, libraries, even games—all the programs that an operating system can usefully offer to its users.

GPL

The **GNU General Public License** (**GNU GPL** or **GPL**) is the most widely used free software license, which guarantees end users(individuals, organizations, companies) the freedoms to use,

study, share (copy), and modify the software. Software that ensures that these rights are retained is called free software. The license was originally written by Richard Stallman of the Free Software Foundation (FSF) for the GNU project. The GPL grants the recipients of a computer program the rights of the Free Software Definition and uses copy left to ensure the freedoms are preserved whenever the work is distributed, even when the work is changed or added to. The GPL is a copy left license, which means that derived works can only be distributed under the same license terms. This is in distinction to permissive free software licenses, of which the BSD licenses are the standard examples. GPL was the first copy left license for general use.

GETTING HELP IN LINUX WITH

a**)** **–help**: Display information about built-in commands.

Syntax

help [-dms] [PATTERN ...]

Description

Displays brief summaries of shell builtin commands. If PATTERN is specified, gives detailed help on all commands matching PATTERN; otherwise the list of help topics is printed.
Options

-d	output short description for each topic.
-m	display usage in pseudo-manpage format.
-s	output only a short usage synopsis for each topic matching.

b) **whatis**,

whatis displays short manual page descriptions.

Syntax: whatis [-dlhvV] [-r|-w] [-s list] [-m system[,...]] [-M path] [-L locale] [-C file] name ...

Each manual page has a short description available within it. **whatis** searches the manual page names and displays the manual page descriptions of any name matched.

name may contain wildcards (-w) or be a regular expression (-r). Using these options, it may be necessary to quote the name or escape (\) the special characters to stop theshell from interpreting them.

c) **man** command

On Linux and other Unix-like operating systems, **Man** is the interface used to view the system's reference manuals. **Man** is the system's manual viewer; it can be used to display manual pages, scroll up and down, search for occurrences of specific text, and other useful functions. Each argument given to **Man** is normally the name of a program, utility or function. The manual page associated with each of these arguments is then found and displayed. A section number, if provided, will direct **Man** to look only in that section of the manual.

Syntax

a) man [-C file] [-d] [-D] [--warnings[=warnings]] [-R encoding] [-L locale] [-m system[,...]] [-M

path] [-S list] [-e extension] [-i|---regex|- wildcard] --names-only] [-a] [-u]no-subpages] I] [- [[-- [- P pager] [-r prompt] [-7] [-E encoding]no-hyphenation] [--no-justification] [-p string] [- [-- t] [- T[device]] [-H[browser]] [-X[dpi]] [-Z] [[section] page ...] ...

b) man -k [apropos options] regexp ...

c) man -K [-w|-W] [-S list] [-i|-I] [--regex] [section] term ...

d) man -f [whatis options] page ...

e) man -l [-C file] [-d] [-D] [--warnings[=warnings]] [-R encoding] [-L locale] [-P pager] [-r prompt] [-7] [-E encoding] [-p string] [-t] [-T[device]] [-H[browser]] [-X[dpi]] [-Z] file ...

f) man -w|-W [-C file] [-d] [-D] page ...

g) man -c [-C file] [-d] [-D] page...

d) info command

Info reads documentation in the info format. Info is similar to man, with a more robust structure for linking pages together. Info pages are made using the text info tools, and can link with other pages, create menus and ease navigation in general. The default location of info documentation is */usr/share/info*.

syntax

Info[option]…[Menu-Item]

Options

-k, --apropos=STRING	look up STRING in all indices of all manuals.
-d, --directory=DIR	add DIR to INFOPATH.
--dribble=FILENAME	remember user keystrokes in FILENAME.
-f, --file=FILENAME	specify Info file to visit.
-h, --help	display this help and exit.
--index-search=STRING	go to node pointed by index entry STRING.

e) date

About date

Prints or sets the system time and date.

Syntax

date [OPTION]... [+FORMAT]

date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]

Options

-d, **--date**=STRING	display time described by string STRING, as opposed to the default, which is '**now**'
-f, **--file**=DATEFILE	like **--date**, but processed once for each line of file DATEFILE.
-I[TIMESPEC], **--iso-8601**[=TIMESPEC]	output date/time in ISO 8601 format. For values of TIMESPEC, use '**date**' for date only (the default), '**hours**', '**minutes**', '**seconds**', or '**ns**' for date and time to the indicated precision.
-r, **--reference**=FILE	display the last modification time of file FILE.
-R, **--rfc-2822**	output date and time in RFC 2822 format. Example: **Mon, 07 Aug 2006 12:34:56 -0600**

f) **whoami,**

Print effective userid. Would display the name of the current userid. For example, may list root if you're logged in as root.

Syntax

whoami

g) who

About who

Displays who is on the system.

Syntax

who [-a] [-b] [-d] [-H] [-l] [-m] [-nx] [-p] [-q] [-r] [-s] [-t] [-T] [-u] [am i] [file]

-a	Process /var/adm/utmp or the named file with -b, -d, -l, -p, -r, -t, -T, and -u options turned on.
-b	Indicate the time and date of the last reboot.
-d	Display all processes that have expired and not been respawned by init . The exit field appears for dead processes and contains the termination and exit values (as returned by **wait**), of the dead process. This can be useful in determining why a process terminated.
-H	Output column headings above the regular output.
-n	Take a numeric argument, x, which specifies the number of users to display per line. x must be at least x 1. The -n option may only be used with -q.

h) W

About w

Displays information about the users currently on the machine, and their processes. The header shows, in this order, the current time, how long the system has been running, how many users are currently logged on, and the system load averages for the past 1, 5, and 15

minutes.

Syntax

w [-husfVo] [user]

-h	Don't print the header.
-u	Ignores the username while figuring out the current process and cpu times. To demonstrate this, do a "su" and do a "w" and a "w -u".
-s	Use the short format. Don't print the login time, JCPU or PCPU times.
f	Toggle printing the from (remote hostname) field. The default as released is for the from field to not be printed, although your system administrator or distribution maintainer may have compiled a version in which the from field is shown by default.
-V	Display version information.
-o	Old style output. Prints blank space for idle times less than one minute.
user	Show information about the specified user only.

i) Cal

About cal

Calendar for the month and the year.

Description

cal originally appeared in version 6 of AT&T Unix. Since then there have been versions released for BSD, Linux, and other Unix variants. You should check your particular installation's manual for version-specific options. Listed below are the traditional syntax and options for Unix **cal**.

In general, if no options are given, **cal** displays the current month at the command line. It's a quick and convenient way to glance at the dates of the month, and can be useful as part of a login script.

Syntax

cal [options] [[[day] month] year]

Options

-1	Display a single month. This is the default.
-3	Display three months: last month, this month, and next month.
-s	Display the calendar using Sunday as the first day of the week.
-m	Display Monday as the first day of the week.
-j	Display dates of the Julian calendar.
-y	Display a calendar for the entire current year.

j) bc

About bc

bc is an arbitrary-precision calculator language.**bc** is a language that supports arbitrary-precision numbers, meaning that it delivers accurate results regardless of how large (or very very small) the numbers are.

It has an interactive mode, accepting input from the terminal and providing calculations on request. As a language, its syntax is similar to the C programming language. A standard math library is available using a command line option. If requested, the math library is defined before processing any files. **bc** starts by processing code from all the files listed on the command line in the order listed. After all files have been processed, **bc** reads from the standard input. All code is executed as it is read. Newer versions of **bc** contain several extensions beyond traditional **bc** implementations and the POSIX draft standard. Command-line options can cause these extensions to print a warning or to be rejected. This document describes the newer version of the **bc** language

Syntax

bc [-hlwsqv] [long-options] [file ...]

Options

-h, **--help**	Print a help message and exit.
-i, **--interactive**	Force interactive mode.
-l, **--mathlib**	Define the standard math library.

-w, **--warn**	Give warnings for extensions to POSIX **bc**.
-s, **--standard**	Process exactly the POSIX **bc** language.
-q, **--quiet**	Do not print the normal GNU **bc** welcome message.
-v, **--version**	Print the version number and copyright information, and exit.

k) Hostname

About hostname

The **hostname** command shows or sets the system hostname.

Syntax

hostname [-v] [-a|--alias] [-d|--domain] [-f|--fqdn|--long] [-A|--all-fqdns] [-i|--ip-address] [-I|--all-ip-addresses] [-s|--short] [-y|--yp|--nis]

hostname [-v] [-b|--boot] [-F|--file filename] [hostname]

hostname [-v] [-h|--help] [-V|--version]

Description

hostname is used to display the system's DNS name, and to display or set its hostname or NIS (Network Information Services) domain name.

When called without any arguments, **hostname** will display the name of the system as returned by the gethostname function.

When called with one argument or with the **--file** option, **hostname** will set the system's host name using the sethostname function. (Only the superuser can set the host name.)

The host name is usually set once at system startup in the script**/etc/init.d/hostname.sh** (normally by reading the contents of a file which contains the host name, e.g. **/etc/hostname**).

Options

-a, --alias	Display the alias name of the host (if used). This option is deprecated and should not be used anymore.
-A, --all-fqdns	Displays all FQDNs of the machine. This option enumerates all configured network addresses on all configured network interfaces, and translates them to DNS domain names. Addresses that cannot be translated (i.e. because they do not have an appropriate reverse DNS entry) are skipped. Note that different addresses may resolve to the same name, therefore the output may contain duplicate entries. Do not make any assumptions about the order of the output.
-b, --boot	Always set a hostname; this allows the file specified by -F to be non-existant or empty, in which case the default hostname localhost will be used if none is yet set.
-d, **--domain**	Display the name of the DNS domain. Don't use the command **domainname** to get the DNS domain name because it will show the NIS domain name and not the DNS domain name. Use **dnsdomainname** instead. See the warnings in section The FQDN, and avoid using this option if at all possible.
-f, **--fqdn**,**--long**	Display the FQDN (Fully Qualified Domain Name). A FQDN consists of a short host name and the DNS domain name. Unless you are using bind (Berkeley Internet Domain Name) or NIS for host lookups, you can change the FQDN and the DNS domain name (which is part of the FQDN) in the **/etc/hosts** file. See the warnings in section The FQDN, and avoid using this option if at all possible; use **hostname --all-fqdns**instead.

-F, **--file***filename*	Read the host name from the specified file. Comments (lines starting with a `#') are ignored.

l**) uname**

About uname

Print name of current system.

Syntax

uname [-a] [-i] [-m] [-n] [-p] [-r] [-s] [-v] [-X] [-S systemname]

-a	Print basic information currently available from the system.
-i	Print the name of the hardware implementation (platform).
-m	Print the machine hardware name (class). Use of this option is discouraged; use
	uname -p instead.
-n	Print the nodename (the nodename is the name by which the system is known to a communications network).
-p	Print the current host's ISA or processor type.
-r	Print the operating system release level.

-s	Print the name of the operating system. This is the default.
-v	Print the operating system version.

m) concept of aliases

About alias

alias instructs the shell to replace one string with another when executingcommands. It is used to customize the shell session interface. Using **alias**, frequently-used commands can be invoked using a different, preferred term; and complex or commonly-used options can be used as the defaults for a given command.

Aliases persist for the current session. They can be loaded at login time by modifying the shell's **.rc** file. The invocation and usage of **alias** differs depending on the shell; see your shell's documentation for details.

The examples below are common to many shells, including Unix.

Syntax

```
alias [name=['command']]
```

Name	Specifies the alias name.

Command	Specifies the command the **name** will be an alias for.
-a	Removes all alias definitions from the environment of the current shell session.

-t	Sets and lists tracked aliases.
-x	Sets or prints exported aliases. An exported alias is defined for scripts invoked by name.

LINUX FILE SYSTEM TYPES EXT2, EXT3, EXT4,

Ext2

Ext2 stands for second extended file system.

It was introduced in 1993. Developed by Rémy Card.

This was developed to overcome the limitation of the original ext file system. Ext2 does not have journaling feature.

On flash drives, usb drives, ext2 is recommended, as it doesn't need to do the over head of journaling.

Maximum individual file size can be from 16 GB to 2 TB Overall ext2 file system size can be from 2 TB to 32 TB

Ext3

Ext3 stands for third extended file system.

It was introduced in 2001. Developed by Stephen Tweedie. Starting from Linux Kernel 2.4.15 ext3 was available.

The main benefit of ext3 is that it allows journaling.

Journaling has a dedicated area in the file system, where all the changes are tracked. When the system crashes, the possibility of file system corruption is less because of journaling.

Maximum individual file size can be from 16 GB to 2 TB Overall ext3 file system size can be from 2 TB to 32 TB

There are three types of journaling available in ext3 file system. Journal – Metadata and content are saved in the journal.

Ordered – Only metadata is saved in the journal. Metadata are journaled only after writing the content to disk. This is the default.

Writeback – Only metadata is saved in the journal. Metadata might be journaled either before or after the content is written to the disk.

You can convert a ext2 file system to ext3 file system directly (without backup/restore).

Ext4

Ext4 stands for fourth extended file system. It was introduced in 2008.

Starting from Linux Kernel 2.6.19 ext4 was available.

Supports huge individual file size and overall file system size. Maximum individual file size can be from 16 GB to 16 TB

Overall maximum ext4 file system size is 1 EB (exabyte). 1 EB = 1024 PB (petabyte). 1 PB = 1024 TB (terabyte).

Directory can contain a maximum of 64,000 subdirectories (as opposed to 32,000 in ext3) You can also mount an existing ext3 fs as ext4 fs (without having to upgrade it).

Several other new features are introduced in ext4: multiblock allocation, delayed allocation, journal checksum. fast fsck, etc. All you need to know is that these new features have improved the performance and reliability of the filesystem when compared to ext3.

In ext4, you also have the option of turning the journaling feature "off".

BASIC LINUX DIRECTORY STRUCTURE AND THE FUNCTIONS OF DIFFERENT DIRECTORIES

Several major directories are associated with all modern Unix/Linux operating systems. These directories organize user files, drivers, kernels, logs, programs, utilities, and more into different categories. The standardization of the FHS makes it easier for users of other Unix-based operating systems to understand the basics of Linux. Every FHS starts with the root directory, also known by its label, the single forward slash (/). All of the other directories shown in Table are subdirectories of the root directory. Unless they are mounted separately, you can also find their files on the same partition as the root directory.

/	The root directory, the top-level directory in the FHS. All other directories are subdirectories of root, which is always mounted on some partition. All directories that are not mounted on a separate partition are included in the root directory?s partition.
/bin	Essential command line utilities. Should not be mounted separately; otherwise, it could be difficult to get to these utilities when using a rescue disk.
16MB	Includes Linux startup files, including the Linux kernel. Can be small; is /boot usually adequate for a typical modular kernel. If you use multiple kernels, such as for testing a kernel upgrade, increase the size of this partition accordingly.
/etc	Most basic configuration files.

/dev	Hardware and software device drivers for everything from floppy drives to terminals. Do not mount this directory on a separate partition.
/home	Home directories for almost every user.
/lib	Program libraries for the kernel and various command line utilities. Do not mount this directory on a separate partition.
/mnt	The mount point for removable media, including floppy drives, CD-ROMs, and Zip disks.
/opt	Applications such as WordPerfect or StarOffice.
/proc	Currently running kernel-related processes, including device assignments such as IRQ ports, I/O addresses, and DMA channels.
/root	The home directory of the root user.
/sbin	System administration commands. Don't mount this directory separately.
/tmp	Temporary files. By default, Red Hat Linux deletes all files in this directory periodically.
/usr	Small programs accessible to all users. Includes many system administration commands and utilities.
/var	Variable data, including log files and printer spools.

About dirname

Strip the last part of a filename.

Syntax

```
dirname [OPTION] NAME...
```

Description

dirname outputs each NAME with its last non-slash component and trailing slashes removed; if NAME contains no /'s, **dirname** outputs a single '**.**' (meaning the current directory).

-z, --zero	separate output with NUL rather than a newline.
--help	Display a help message and exit.
--version	Display version information and exit.

FUNCTIONS DIFFERENT DIRECTORIES BASIC DIRECTORY NAVIGATION COMMANDS LIKE

a) cd

About cd

Changes the shell's working directory.

Syntax

```
cd [-L|-P] directory
```

Options

-L	Force symbolic links to be followed.
-P	Use the physical directory structure without following symbolic links.

b) Mv

About mv

The **mv** command is used to move or rename files.

Syntax

```
mv ...[OPTION] [-T] SOURCE DEST

mv [OPTION]... SOURCE... DIRECTORY

mv -t DIRECTORY [OPTION]... SOURCE...
```

Description

mv renames file SOURCE to DEST, or moves the SOURCE file (or files) to DIRECTORY.

Options

--backup[=CONTROL]	Make a backup of each existing destination file, using the control method CONTROL. (See Backup Methods below for more about control methods.)
-b	Like **--backup**, but does not accept an argument; the default backup method is used.
-f, **--force**	Do not prompt before overwriting existing files.
-i, **--interactive**	Prompt before overwriting each existing destination file, regardless of the file's permissions. If the answer to the prompt is negative, the file is skipped.
-n, **--no-clobber**	Do not overwrite any existing file.

If you specify more than one of the obove options **-i**, **-f**, or **-n**, only the final option specified takes effect.

--strip-trailing-slashes	Remove any trailing slashes from each SOURCE argument.
-S, **--suffix**=SUFFIX	Use the suffix SUFFIX for all backup files. The default SUFFIX is "~".
-t, **--target-directory**=DIRECTORY	Move all SOURCE arguments into directory DIRECTORY.

-T, **--no-target-directory**	Treat DEST as a normal file, not as a directory.
-u, **--update**	Perform the move only if the SOURCE file is newer than the destination file, or the destination file does not already exist.
-v, **--verbose**	Operate verbosely.

--help	display a help message, and exit.
--version	output version information, and exit.

c) **Cp**

About cp

Copies files and directories.

Syntax

```
cp ...[OPTION] [-T] SOURCE DEST

cp [OPTION]... SOURCE... DIRECTORY

cp -t DIRECTORY [OPTION]... SOURCE...
```

Description

cp copies SOURCE to the destination DEST, or multiple SOURCE(s) to directory DIRECTORY.

Options

-a, **--archive**	same as **-dR --preserve=ALL**.
--attributes-only	don't copy the file data, just create a file with the same attributes.
--backup[**=CONTROL**]	make a backup of each existing destination file.
-b	like **--backup**, but does not accept an argument.
--copy-contents	When operating recursively, copy contents of special files.

-d	same as **--no-dereference --preserve=links**
-f, **--force**	if an existing destination file cannot be opened, remove it and try again (redundant if the **-n** option is used)
-i, **--interactive**	prompt before overwrite (overrides a previous **-n** option).
-H	follow command-line symbolic links in SOURCE.
-l, **--link**	Create hard links to files instead of copying them.

d) **Rm**

About rm

Deletes a file without confirmation (by default).

Syntax

rm [-f] [-i] [-R] [-r] [filenames | directory]

-f	Remove all files (whether write-protected or not) in a directory without prompting the user. In a write-protected directory, however, files are never removed (whatever their permissions are), but no messages are displayed. If the removal of a write-protected directory is attempted, this option will not suppress an error message.
-i	Interactive. With this option, rm prompts for confirmation before removing any files. It over- rides the -f option and remains in effect even if the standard input is not a terminal.
-R	Same as -r option.
filenames	A path of a filename to be removed.

e) **cat command**

About cat

cat stands for "catenate." It reads data from files, and outputs their contents. It is the simplest way to display the contents of a file at the command line.

Syntax

```
cat [OPTION]... [FILE]...
```

-A, **--show-all**	equivalent to **-vET**
-b, **--number-nonblank**	number nonempty output lines; overrides **-n**
-e	equivalent to **-vE**
-E, **--show-ends**	display "**$**" at end of each line
-n, **--number**	number all output lines
-s, **--squeeze-blank**	suppress repeated empty output lines
-t	equivalent to **-vT**
-T, **--show-tabs**	display TAB characters as **^I**
-v, **--show-nonprinting**	use ^ and **M-** notation, except for **LFD** and **TAB**
--help	display a help message and exit

--version	output version information and exit

f) **less command**

About less

less is a simple, feature-rich command-line file viewer.

Syntax

less [-[+]aABcCdeEfFgGiIJKLmMnNqQrRsSuUVwWX~] [-b space] [-h lines] [-j line] [-k keyfile] [-{oO} logfile] [-p pattern] [-P prompt] [-t tag] [-T tagsfile] [-x tab,...] [-y lines] [-[z] lines] [-# shift] [+[+]cmd] [--] [filename]...

Description

less is a program similar to more, but it has many more features. **less** does not have to read the entire input

file before starting, so with large input files it starts up faster than text editors

like vi. **less** uses **termcap** (or **terminfo** on some systems), so it can run on a variety of terminals. There is

even limited support for hardcopy terminals. On a hardcopy terminal, lines which should be printed at the

top of the screen are prefixed with a caret ("^").

Commands are based on both **more** and **vi**. Commands may be preceded by a decimalnumber, called *N* in

the descriptions below. The number is used by some commands, as indicated.

Commands

In the following descriptions, **^X** means **control-X**. **ESC** stands for the **ESCAPE** key; for example **ESC-**

v means the two character sequence "**ESCAPE**", then "**v**".

h, **H**	Help: display a summary of commands. If you forget all the other commands, remember this one.
SPACE, **^V**, **f**,**^F**	Scroll forward *N* lines, default one window (see option **z** below). If *N* is more than the screen size, only the final screenful is displayed. Warning: some systems use**^V** as a special literalization character.
Z	Like **SPACE**, but if *N* is specified, it becomes the new window size.
ESC-SPACE	Like **SPACE**, but scrolls a full screenful, even if it reaches end-of-file in the process.

ENTER,**RETURN**, **^N**,**e**, **^E**, **j**, **^J**	Scroll forward *N* lines, default 1. The entire *N* lines are displayed, even if *N* is more than the screen size.
d, **^D**	Scroll forward *N* lines, default one half of the screen size. If *N* specified is, it becomes the new default for subsequent **d** and **u** commands.
b, **^B**, **ESC-v**	Scroll backward *N* lines, default one window (see option -**z** below). If *N* is more than the screen size, only the final screenful is displayed.
W	Like **ESC-v**, but if *N* is specified, it becomes the new window size.
y, **^Y**, **^P**, **k**,**^K**	Scroll backward *N* lines, default 1. The entire *N* lines are displayed, even if *N* is more than the screen size. Warning: some systems use **^Y** as a special job control character.

g) Runlevels

A runlevel is a software configuration of the system which allows only a selected group of processes to exist. The processes spawned by **init** for each of these runlevels are defined in the **/etc/inittab** file. **Init** can be in one of eight runlevels: **0** through **6**, and**S** or **s**.

The runlevel is changed by having a privileged user run **telinit**, which sends appropriate signals to **init**, telling it which runlevel to change to.Runlevels **0**, **1**, and **6** are reserved. Runlevel **0** is used to halt the system, runlevel **6** is used to reboot the system, and runlevel **1** is used to get the system down into single user mode. Runlevel **S** is not really meant to be used directly, but more for the scripts that are executed when entering runlevel **1**.Runlevels **7** - **9** are also valid, though not really documented. This is because "traditional" Unix variants don't use them. In case you're curious, runlevels **S** and **s** are in fact the same. Internally they are aliases for the same runlevel.

UNIT-II

STANDARD INPUT

Standard input, often abbreviated *stdin*, is the input data for a program in the absence of any command line arguments. It is by default any text entered from the keyboard. Thus if wc is typed in at the command line and the ENTER key is pressed without providing any arguments, any text typed in on all subsequent lines will be stored in memory until the wc command is executed by simultaneously pressing the CONTROL and *d* keys on a new, blank line. wc will then count the lines, words and characters in that stored text.Standard input can be *redirected* to come from any text file in place of the keyboard by using the input redirection operator, which is a leftward pointing angular bracket. Thus, to redirect the standard input for the command *sort* (which sorts lines of text in alphabetic order) from the keyboard to the file named *file3*, type:

```
sort < file3
```

The result is the same as using file3 as an argument, although the mechanism is different, i.e.,

```
sort file3
```

STANDARD OUTPUT

Most command line programs that display their results do so by sending their results to a facility called *standard output*. By default, standard output directs its contents to the display. To redirect standard output to a file, the ">" character is used like this:

```
[me@linuxbox me]$ ls > file_list.txt
```

In this example, the **ls** command is executed and the results are written in a file named file_list.txt. Since the output of **ls**was redirected to the file, no results appear on the display.

Each time the command above is repeated, file_list.txt is overwritten (from the beginning) with the output of the command**ls**. If you want the new results to be *appended* to the file instead, use ">>" like this:

[me@linuxbox me]$ ls >> file_list.txt

REDIRECTING INPUT AND OUTPUT

When the results are appended, the new results are added to the end of the file, thus making the file longer each time the command is repeated. If the file does not exist when you attempt to append the redirected output, the file will be created.Many commands can accept input from a facility called *standard input*. By default, standard input gets its contents from the keyboard, but like standard output, it can be redirected. To redirect standard input from a file instead of the keyboard, the "<" character is used like this:

[me@linuxbox me]$ **sort < file_list.txt**

In the above example we used the **sort** command to process the contents of file_list.txt. The results are output on the display since the standard output is not redirected in this example. We could redirect standard output to another file like this:

[me@linuxbox me]$ **sort < file_list.txt > sorted_file_list.txt**

As you can see, a command can have both its input and output redirected. Be aware that the order of the redirection does not matter. The only requirement is that the redirection operators (the "<" and ">") must appear after the other options and arguments in the command.

USING PIPES TO CONNECT PROCESSES

the most useful and powerful thing you can do with I/O redirection is to connect multiple commands together with what are called *pipes*. With pipes, the standard output of one command is fed into the standard input of another. Here is my absolute favorite:

[me@linuxbox me]$ **ls -l | less**

In this example, the output of the **ls** command is fed into **less**. By using this **"| less"** trick, you can make any command have scrolling output. I use this technique all the time.

By connecting commands together, you can acomplish amazing feats. Here are some examples

you'll want to try:

Command

ls -lt | head

du | sort –nr

find . -type f -print | wc –l

TEE COMMAND

About tee

What it does

Displays the 10 newest files in the current directory.

Displays a list of directories and how much space they consume, sorted from the largest to the smallest.

Displays the total number of files in the current working directory and all of its subdirectories.

Read from an input and write to a standard output and file.

Syntax

tee [OPTION]... [FILE]...

-a --append	Append to the given FILEs, do not overwrite.
-i	ignore interrupt signals.
--help	Display the help screen.
--version	Display the version.

LINUX FILE SECURITY

The Linux security model is based on the one used on UNIX systems, and is as rigid as the UNIX security model (and sometimes even more), which is already quite robust. On a Linux system, every file is owned by a user and a group user. There is also a third category of users, those that are not the user owner and don't belong to the group owning the file. For each category of users, read, write and execute permissions can be granted or denied. We already used the *long* option to list files using the **ls -l** command, though for other reasons. This command also displays file permissions for these three user categories; they are indicated by the nine characters that follow the first character, which is the file type indicator at the beginning of the file properties line. As seen in the examples below, the first three characters in this series of nine display access rights for the actual user that owns the file. The next three are for the group owner of the file, the last three for other users. The permissions are always in the same order: read, write, execute for the user, the group and the others.

PERMISSION TYPES

Permission Groups

Each file and directory has three user based permission groups:

- **owner** - The Owner permissions apply only the owner of the file or directory, they will not impact the actions of other users.
- **group** - The Group permissions apply only to the group that has been assigned to the file or directory, they will not effect the actions of other users.
- **all users** - The All Users permissions apply to all other users on the system, this is the permission group that you want to watch the most.

PERMISSION TYPES

Each file or directory has three basic permission types:

- **read** - The Read permission refers to a user's capability to read the contents of the file.
- **write** - The Write permissions refer to a user's capability to write or modify a file or directory.
- **execute** - The Execute permission affects a user's capability to execute a file or view the contents of a directory.

VIEWING THE PERMISSIONS

You can view the permissions by checking the file or directory permissions in your favorite GUI File

Manager (which I will not cover here) or by reviewing the output of the ***\"ls -l\"*** command while in the terminal and while working in the directory which contains the file or folder.

The permission in the command line is displayed as: ***_rwxrwxrwx 1 owner:group***

1. User rights/Permissions

    1. The first character that I marked with an underscore is the special permission flag that can vary.

    2. The following set of three characters (rwx) is for the owner permissions.

3. The second set of three characters (rwx) is for the Group permissions.

4. The third set of three characters (rwx) is for the All Users permissions.

2. Following that grouping since the integer/number displays the number of hardlinks to the file.

3. The last piece is the Owner and Group assignment formatted as Owner:Group.

MODIFYING THE PERMISSIONS

When in the command line, the permissions are edited by using the command ***chmod***. You can assign the permissions explicitly or by using a binary reference as described below.

Explicitly Defining Permissions

To explicity define permissions you will need to reference the Permission Group and Permission Types.

The Permission Groups used are:

- **u** - Owner
- **g** - Group
- **o** or **a** - All Users

The potential Assignment Operators are + (plus) and - (minus); these are used to tell the system whether to add or remove the specific permissions.

The Permission Types that are used are:

- **r** - Read
- **w** - Write
- **x** - Execute

h) permission types:

There are two parts to permissions. The first involves what someone is allowed to do with a

file, and the second involves who that "someone" can be.

What can be done

When controlling what can be done to a file or directory, there are three categories of actions: read, write, and execute. What is specifically allowed or disallowed can be different for files and directories, so we'll talk about both for each category.

Read

The "read" permission controls, well, who can read a file. If you don't have read permissions for a file you can't look inside and see its contents. The "read" permission for a directory controls whether or not you can see a list of the files in the directory. Note, however, that to do so you will also need "execute" permission for the directory.

Write

The "write" permission on a file controls whether or not you can change the file's contents. If you want to edit the text in an html file, for example, you need write permission before you can do so. The "write" permission on a directory controls whether or not you can add, delete, or rename files in that directory.

Execute

The "execute" permission for a file allows you to run that file from the command line. In order to run any command ("chown", "ls", "rm", etc.), you have to have execute permission for the file representing that command. If you try to run a command and get a "permission denied" error, it's because you don't have execute permission. The "execute" permission for a directory lets you perform an operation in that directory, or to change your working directory ("cd") to that directory.

i) EXAMINING PERMISSIONS

By examining permissions for each of the following files, identify if it is a file or directory, and describe the access allowed to the world, user, and group:

- a. -rwx---r-x

- b. drwx------
- c. -rwxrwxr--
- d. dr-x---r-x
- e. -rwx---rwx

j) CHANGING PERMISSIONS(SYMBOLIC METHOD NUMERIC METHOD),

Using Symbolic Modes With Chmod

In order to change the permissions of a file using symbolic permissions, use the command format:

```
chmod SYMBOLIC-MODE FILENAME
```

where SYMBOLIC-MODE is the symbolic representation of permissions (which we describe below) that you wish to apply to FILENAME.

The letters for user, group, and other are **u**,**g**, and **o** respectively. The letter **a** is used to mean all three of these categories.

Using Numeric Modes With Chmod

To set the permissions of a file or directory using numeric modes, simply use the format:

```
chmod OCTAL-MODE FILENAME
```

where OCTAL-MODE is the octal form of the permissions.

For example, to set the permissions of filename to -rw-r--r-- you could run the command:

```
chmod 644 filename
```

or to change permissions to -rwxrwxrwx you could use the command:

```
chmod 777 filename
```

Be careful when setting permissions to 777 as this means every single user account can

read, write, and execute that file.

DEFAULT PERMISSIONS AND UMASK

When a user creates a file, how does the system determine that file's initial permissions? This is done based on the user's **umask value**. The umask value specifies which permissions are **not** to be set. In Ubuntu, the default umask value for a normal user is 002, while the default for root is 022. You can find out the current umask value (or set it) using the **umask** command. If (as a normal user) you run the command:

```
umask
```

EDITOR BASICS

There are many ways to edit files in Unix and for me one of the best ways is using screen-oriented text editor **vi**. This editor enable you to edit lines in context with other lines in the file. Now a days you would find an improved version of vi editor which is called **VIM**. Here VIM stands for **Vi**ew **IM**proved.

The vi is generally considered the de facto standard in Unix editors because:

- It's usually available on all the flavors of Unix system.
- Its implementations are very similar across the board.
- It requires very few resources.
- It is more user friendly than any other editors like ed or ex.

Starting the vi Editor:

There are following way you can start using vi editor:

Command	Description
vi filename	Creates a new file if it already does not exist, otherwise opens existing file.
vi -R filename	Opens an existing file in read only mode.
view filename	Opens an existing file in read only mode.

THREE MODES OF VI EDITOR,

The vi editor has three modes, command mode, insert mode and command line mode.

Command mode: letters or sequence of letters interactively command vi. Commands are case sensitive. The ESC key can end a command.

Insert mode: Text is inserted. The ESC key ends insert mode and returns you to command mode. One can enter insert mode with the "i" (insert), "a" (insert after), "A" (insert at end of line), "o" (open new line after current line) or "O" (Open line above current line) commands.

Command line mode: One enters this mode by typing ":" which puts the command line entry at the foot of the screen.

CONCEPT OF INODES,

The **inode (index node)** is a fundamental concept in the Linux and UNIX filesystem. Each object in the filesystem is represented by an inode. But what are the objects? Let us try to understand it in simple words. Each and every file under Linux (and UNIX) has following attributes:

=> File type (executable, block special etc) => Permissions (read, write etc)

=>
Owner
=> Group
=> File
Size

=> File access, change and modification time (remember UNIX or Linux never stores file creation time, this is favorite question asked in UNIX/Linux sys admin job interview)

=> File deletion time

=> Number of links (soft/hard)

=> Extended attribute such as append only or no one can delete file including root user (immutability)

=> Access Control List (ACLs)

All the above information stored in an inode. In short the inode identifies the file and its attributes (as above) . Each inode is identified by a unique inode number within the file system. Inode is also know as index number.

inode definition

An inode is a data structure on a traditional Unix-style file system such as UFS or ext3. An inode stores basic information about a regular file, directory, or other file system object.

Example:$ ls -i /etc/passwd

INODES AND DIRECTORIES

Inode Basics

An Inode number points to an Inode. An Inode is a data structure that stores the following information about a file :

Size of file
Device ID

User ID of the file

Group ID of the file

The file mode information and access privileges for owner, group and others File protection flags

The timestamps for file creation, modification etc link counter to determine the number of hard links Pointers to the blocks storing file's contents

Directory in linux

Linux stores data and programs in **files**. These are organized in directories. In a simple way, a directory is just a file that contains other files (or directories).

The part of the hard disk where you are authorised to save data is calle your **home directory**. Normally all the data you want will be saved in files and directories in your home directory. To find your home directory (if you need), type:

echo $HOME

The symbol ~ can also be used for your home directory.

There is a general directory called **/tmp** where every user can write files. But files in /tmp usually get removed (erased) when the system boots or periodically, so you should not store in /tmp data that you want to keep permanently.

Creating and Removing directories

To make a new directory do:

mkdir directory-name

To remove a directory that does **not** files inside do:
rmdir directory-name

Changing the working directory

To change ("enter") into a directory do:

cd directory-name

This assumes that the new directory is a subdirectory of the one you are currently working on. If that is not the case, you will have to type the name, for example:
cd /usr/local/share/bin

To go to your home directory do simply:
cd

Renaming directories

To change the name of a directory do:

mv directory-name new-name

p) Cp

About cp

Copies files and directories.

Syntax

```
cp [OPTION]... [-T] SOURCE DEST

cp [OPTION]... SOURCE... DIRECTORY

cp [OPTION]... -t DIRECTORY SOURCE...
```

Description

cp copies SOURCE to the destination DEST, or multiple SOURCE(s) to directory DIRECTORY.

q) inodes mv and inodes rm

About mv

The mv command is used to move or rename files.

Syntax

```
mv [OPTION]... [-T] SOURCE DEST

mv [OPTION]... SOURCE... DIRECTORY

mv [OPTION]... -t DIRECTORY SOURCE...
```

Description

mv renames file SOURCE to DEST, or moves the SOURCE file (or files) to DIRECTORY.

About rm

The rm command removes (deletes) files or directories.

Syntax

```
rm [OPTION]... FILE...
```

Description

rm removes each specified FILE. By default, it does not remove directories; seeRemoving Directories below for details.

The removal process unlinks a filename in a filesystem from data on the storage device, and marks that space as usable by future writes. In other words, removing files increases the amount of available space on your disk.

INODES,

The "inode" is sometimes referred to as an index node. But what is it? Basically, it is a file structure on a file system. More easily, it is a "database" of all file information except the file contents and the file name.

In a file system, inodes consist roughly of 1% of the total disk space, whether it is a whole storage unit (hard disk,thumb drive, etc.) or a partition on a storage unit. The inode space is used to "track" the files stored on the hard disk. The inode entries store metadata about each file, directory or object, but only points to these structures rather than storing the data. Each entry is 128 bytes in size. The metadata contained about each structure can include the following:

- Inode number
- Access Control List (ACL)
- Extended attribute
- Direct/indirect disk blocks
- Number of blocks
- File access, change and modification time
- File deletion time
- File generation number
- File size
- File type
- Group
- Number of links
- Owner
- Permissions
- Status flags

SYMBOLIC LINKS AND HARD LINKS

Soft link or symlink, a symbolic link is a Linux and Unixfile created with the ln command that links to another file using the path. Unlike a hard link, a symbolic link can link to any file on any computer. If you're more familiar with Microsoft Windows you can think of a symbolic link as a shortcut in Linux.

A hard link is a link file created with the Linux or Unix ln command that points to a file's inode.

MOUNT AND UMOUNT COMMAND,

About mount and umount

The mount command mounts a storage device or filesystem,

making it accessible and attaching it to an

existing directory structure.

The umount command "unmounts" a mounted filesystem, informing the system to complete any pending read or write operations, and safely detaching it.

Syntax: mount

```
mount [-lhV]

mount -a [-fFnrsvw] [-t vfstype] [-O optlist]

mount [-fnrsvw] [-o option[,option]...] device|dir

mount [-fnrsvw] [-t vfstype] [-o options] device|dir
```

Syntax: umount

```
umount [-hV]

umount -a [-dflnrv] [-t vfstype] [-O options]

umount [-dflnrv] {dir|device}...
```

CREATING ARCHIVES

- Connect to a shell or open a terminal/console on your Linux/Unix machine.
- To create an archive of a directory and its contents you would type the following and press enter:

o **tar -cvf name.tar /path/to/directory**

Substitute the name.tar with the name of the tar file you would like to create and substitute the directory name for the full path to the directory you would like to archive.

- To create an archive of certfain files you would type the following and press enter:

o **tar -cvf name.tar /path/to/file1 /path/to/file2 /path/to/file3**

Substitute the name.tar with the name of the tar file you would like to create and substitute the the various files for the full path to the files you would like to archive. Each file you would like included in the archive should be seperated by a space.

TAR

About tar

The tar program is used to create, modify, and access file archives of the tar format.

Syntax

```
--tar[-]A       --catenateconcatenate | c          --create | d      --  --diff
compare |       --delete | r --append | t --          list | --test-label | u --
update | x --extract --get [options] [pathname ...                    ]
```

Description

"tar" stands for tape archive. It is an archiving file format.

tar was originally developed in the early days of Unix for the purpose of backing up files to tape-based storage devices. It was later formalized as part of the POSIXstandard, and today is used to collect, distribute, and archive files, while preservingfile system attributes such as user and group permissions, access and modification dates, and directory structures.

GZIP,

gzip reduces the size of the named files using Lempel-Ziv coding (LZ77). Whenever possible, each file is replaced by one with the extension .gz, while keeping the sameownership modes, access and modification times. (The default extension is - gz forVMS, z for MSDOS, OS/2 FAT, Windows NT FAT and Atari.) If no files are specified, or if a file name is "-", the standard input is compressed to the standard output. gzip will only attempt to compress regular files. In particular, it will ignore symbolic links.

GUNZIP

gunzip takes a list of files on its command line and replaces each file whose name ends with .gz, -gz, .z, -z, or _z (ignoring case) and which begins with the correct

magic number with an uncompressed file without the original extension. gunzip also recognizes the special extensions .tgz and .taz as shorthands for .tar.gz and .tar.Zrespectively. When compressing, gzip uses

the .tgz extension if necessary instead of truncating a file with a .tar extension.

BZIP2

bzip2 expects a list of file names to accompany the command-line flags. Each file is replaced by a compressed version of itself, with the name "original_name.bz2". Each compressed file has the same modification date, permissions, and, when possible, ownership as the corresponding original, so that these properties can be correctly restored at decompression time. File name handling is naive in the sense that there is no mechanism for preserving original file names, permissions, ownerships or dates in filesystems which lack these concepts, or have serious file name length restrictions, such as MS-DOS.

BUNZIP2(BASIC USAGE OF THESE COMMANDS)

bunzip2 (or bzip2 -d) decompresses all specified files. Files which were not created by bzip2 will be detected and ignored, and a warning issued. bzip2 attempts to guess the filename for the decompressed file from that of the compressed file as follows:

filename.bz2	becomes	filename
filename.bz	becomes	filename
filename.tbz2	becomes	filename.tar
filename.tbz	becomes	filename.tar
anyothername	becomes	anyothername.out

UNIT – III

ENVIRONMENT VARIABLES

An environment variable is a named object that contains data used by one or more applications. In simple terms, it is a variable with a name and a value. The value of an environmental variable can for example be the location of all executable files in the file system, the default editor that should be used, or the system locale settings. Users new to Linux may often find this way of managing settings a bit unmanageable. However, an environment variable provides a simple way to share configuration settings between multiple applications and processes in Linux.

HOME Contains the path to the home directory of the current user. This variable can be used by applications to associate configuration files and such like with the user running it.

SHELL Contains the name of the running, interactive shell, i.e bash

VISUAL Contains the path to full-fledged editor that is used for more demanding tasks, such as editing mail; e.g., vi, **vim**, **emacs**, etc.

BROWSER Contains the path to the web browser. Helpful to set in an interactive shell configuration file so that it may be dynamically altered depending on the availability of a graphic environment, such as **X**

LANG contain the setting for every categories that are not directly set by a LC_* variable. LC_ALL is used to override every LC_* and LANG and LANGUAGE, it should not be set in a normal user environment, but can be useful when you are writing a script that depend on the precise output of a internationalized command.

USEWR This variable should have the same setting and purpose as LOGNAME(The name of the user. This is an easy way for a user to get own username. However, programs must not trust this variable because it can be set to an arbitrary value.

Both LOGNAME and USER should be set to the username.

Examples:

LOGNAME=tux

LOGNAME=sudhir

DISPLAY display/print/list environment variables.To list all environment

variables Execute: printenv.To print a specific environment variable

Execute: echo $Env_Var_Name

Example: echo $PATH

LOCAL VARIABLES

- By default all variables are global.
- Modifying a variable in a function changes it in the whole script.

- This can be result into problem. For example, create a shell script called fvar.sh: Commands :

local var=value

local varName

CONCEPT OF /ETC/PASSWD

The /etc/passwd contains one entry per line for each user (or user account) of the system. All fields are separated by a colon (:) symbol. Total seven fields as follows.

Generally, passwd file entry looks as follows (click to enlarge image):

(Fig.01: /etc/passwd file format - click to enlarge)

1. **Username**: It is used when user logs in. It should be between 1 and 32 characters in length.
2. **Password**: An x character indicates that encrypted password is stored in /etc/shadow file.
3. **User ID (UID)**: Each user must be assigned a user ID (UID). UID 0 (zero) is reserved

for root and UIDs 1-99 are reserved for other predefined accounts. Further UID 100-999 are reserved by system for administrative and system accounts/groups.

4. **Group ID (GID)**: The primary group ID (stored in /etc/group file)

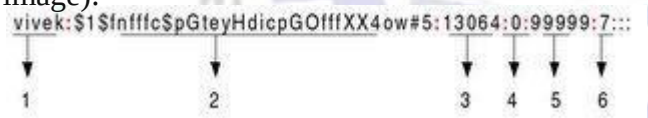
5. **User ID Info**: The comment field. It allow you to add extra information about the users such as user's full name, phone number etc. This field use by finger command.

6. **Home directory**: The absolute path to the directory the user will be in when they log in. If this directory does not exists then users directory becomes /

7. **Command/shell**: The absolute path of a command or shell (/bin/bash). Typically, this is a shell. Please note that it does not have to be a shell.

/ETC/SHADOW

A. **/etc/shadow** file stores actual password in encrypted format for user's account with additional properties related to user password i.e. it stores secure user account information. All fields are separated by a colon (:) symbol. It contains one entry per line for each user listed in /etc/passwd file Generally, shadow file entry looks as follows (click to enlarge image):



(Fig.01: /etc/shadow file fields)

1. User name : It is your login name

2. Password: It your encrypted password. The password should be minimum 6-8 characters long including special characters/digits

3. Last password change (last changed): Days since Jan 1, 1970 that password was last changed

4. Minimum: The minimum number of days required between password changes i.e. the number of days left before the user is allowed to change his/her password

5. Maximum: The maximum number of days the password is valid (after that user is forced to change his/her password)

6. Warn : The number of days before password is to expire that user is warned that

his/her password must be changed

7. Inactive : The number of days after password expires that account is disabled

8. Expire : days since Jan 1, 1970 that account is disabled i.e. an absolute date specifying when the login may no longer be used

/ETC/GROUP

/etc/group is a text file which defines the groups to which users belong under Linux and UNIX operating system. Under Unix / Linux multiple users can be categorized into groups. Unix file system permissions are organized into three classes, user, group, and others. The use of groups allows additional abilities to be delegated in an organized fashion, such as access to disks, printers, and other peripherals.

This method, amongst others, also enables the Superuser to delegate some administrative tasks to normal users.It stores group information or defines the user groups i.e. it defines the groups to which users belong. There is one entry per line, and each line has the following format (all fields are separated by a colon (:)

cdrom:x:24:vivek,student13,raj

———— – – ————

| | | |

| | | |

1 2 3 4

Where,

1. **group_name**: It is the name of group. If you run ls -l command, you will see this name printed in the group field.

2. **Password**: Generally password is not used, hence it is empty/blank. It can store encrypted password. This is useful to implement privileged groups.

3. **Group ID (GID)**: Each user must be assigned a group ID. You can see this number in your /etc/passwd file.

4. **Group List**: It is a list of user names of users who are members of the group. The user names, must be separated by commas.

SU- COMMAND

The *su* (short for *substitute user*) command makes it possible to change a login session's *owner* (i.e., the user who originally created that session by logging on to the system) without the owner having to first log out of that session. Although su can be used to change the ownership of a session to any user, it is most commonly employed to change the ownership from an ordinary user to the *root*(i.e., administrative) user, thereby providing access to all parts of and all commands on the computer or system. For this reason, it is often referred to (although somewhat inaccurately) as the *super user* command. It is also sometimes called the *switch user* command.

SPECIAL PERMISSIONS(SUID FOR AN EXECUTABLE,SGID FOR AN EXECUTABLE,SGID FOR A DIRECTORY,STICKY BIT FOR A DIRECTORY)

SUID (Set User ID)

When a SUID bit is set on a command then that command always executes with the User ID of its own user owner (who created it) instead of the user who is executing it.EXAMPLE: The binary

of ***passwd*** command has SUID permission set on it, that is why, when unpriviledged users execute this command, it always executes with the UID of "root" and changes their password in /etc/shadow (which is only readable or writable by root).

To set SUID on a program, run:

[kh@server ~]$ chmod u+s "/path/to/command/binary"

SGID (Set Group ID)(on command binary)

When SGID permission is set on any command, then that command runs with the Group ID of

group owner of the command's binary instead of GID of the user who is executing it. To set SGID on a program, run:

```
[kh@server ~]$ chmod g+s "/path/to/command/binary"
```

SGID (Set Group ID)(on directories)

When SGID permission is set on a directory, then all the new (future) files created under that directory will have the same group owner as that of the parent directory. Moreover subdirectories (created in future) will also have SGID bit on them. Example: If we set SGID on a directory, for example:

on */tmp/test* with group owner as "john", now if another user "mike" creates any file in */tmp/test* directory then the user owner of this file will be "mike" but group owner will be "john" because of SGID on parent directory. To set SGID on a directory, run:

```
[kh@server ~]$ chmod g+s /path/to/directory
```

STICKY BIT

The new files created under the directory having Sticky Bit on it can be only deleted by root or the user who created that file. No other user can delete that file even if they have write permission on the parent directory. EXAMPLE: **/tmp** directory is having Sticky Bit permission on it, that is why the content under this can be only deleted by root or the user owner of the content/file. To set Sticky Bit on a directory, run:

```
[kh@server ~]$ chmod o+t /path/to/directory
```

TAIL

tail - output the last part of files

command

tail [*OPTION*]... [*FILE*]...

DESCRIPTION

Print the last 10 lines of each FILE to standard output. With more than one FILE, precede each with a header giving the file name. With no FILE, or when FILE is -, read standard input. Mandatory arguments to long options are mandatory for short options too.

--retry :>keep trying to open a file even if it is inaccessible when tail starts or if it becomes inaccessible later -- useful only with **-f-c, --bytes=*N:***>output the last N bytes

-f, --follow[={name|descriptor}]:>output appended data as the file grows; **-f**, **--follow**, and **--** **follow**=*descriptor* are equivalent

-F:>same as **--follow**=*name* **--retry**

-n, --lines=*N:*>output the last N lines, instead of the last 10

--max-unchanged-stats=*N:*>with **--follow**=*name*, reopen a FILE which has not changed size after N (default 5) iterations to see if it has been unlinked or renamed (this is the usual case of rotated log files) **--pid=*PID:***>with **-f**, terminate after process ID, PID dies

-q, --quiet, --silent:>never output headers giving file names

-s, --sleep-interval=*S:*>with **-f**, sleep for approximately S seconds (default 1.0) between iterations. **-v, --verbose:**>always output headers giving file names

--help:>display t

WC

Short for word count, wc displays a count of lines, words, and characters in a file.

Syntax

wc [-c | -m | -C] [-l] [-w] [file ...]

-c	Count bytes.
-m	Count characters.
-C	Same as -m.

-l	Count lines.
-w	Count words delimited by white space characters or new line characters. Delimiting characters are Extended Unix Code (EUC) characters from any code set defined by iswspace()
file	Name of file to word count.

SORT

About sort

Sorts the lines in a text file.

Syntax

sort [options]... [file]

-b	Ignores spaces at beginning of the line.
-c	Check whether input is sorted; do not sort
-d	Uses dictionary sort order and ignores the punctuation.
-f	Ignores caps
-g	Ccompare according to general numerical value
-i	Ignores nonprinting control characters.

-k	Start a key at POS1, end it at POS2 (origin 1)
-m	Merges two or more input files into one sorted output.
-M	Treats the first three letters in the line as a month (such as may.)

UNIQ

About uniq

Report or filter out repeated lines in a file.

Syntax

uniq [-c | -d | -u] [-f fields] [-s char] [-n] [+m] [input_file [output_file]]

-c Precede	each output line with a count of the number of times the line occurred in the input.

-d	Suppress the writing of lines that are not repeated in the input.
-u	Suppress the writing of lines that are repeated in the input.
-f fields	Ignore the first fields fields on each input line when doing comparisons, where fields is a positive decimal integer. A field is the maximal string matched by the basic regular expression: [[:blank:]]*[^[:blank:]]* If fields specifies more fields than appear on an input line, a null string will be used for comparison.
-s char	Ignore the first chars characters when doing comparisons, where chars is a positive decimal integer. If specified in conjunction with the -f option, the first chars characters after the first fields fields will be ignored. If chars specifies more characters than remain on an input line, a null string will be used for comparison.

CUT

About cut

Remove or "cut out" sections of each line of a file or files.

Syntax

cut OPTION... [FILE]...

Options

-b, **--bytes**=LIST	Select only the bytes specified in LIST.
-c, **--characters**=LIST	Select only the characters specified in LIST
-d, **--delimiter**=DELIM	use DELIM instead of a tab for the field delimiter
-f, **--fields**=LIST	select only these fields; also print any line that contains no delimiter character, unless the **-s** option is specified

n	This option is ignored, but is included for compatibility reasons.
--complement	complement the set of selected bytes, characters or fields.
-s, **--only-delimited**	do not print lines not containing delimiters.

TR

About tr

The **tr** command automatically translates (substitutes) sets of characters.

Syntax

tr [-Ccsu] string1 string2

In this form, the characters in the string *string1* are translated into the characters in*string2* where the first character in *string1* is translated into the first character in*string2* and so on.

DIFF

About diff

Displays two files and prints the lines that are different.

Syntax

```
diff [OPTION]... FILES
```

Options

--normal	output a normal diff (This is the default).
-q, **--brief**	report only when files differ.
-s, **--report-identical-files**	report when two files are the same.
-c, **-C** NUM, **--context**[=NUM]	output NUM (default 3) lines of copied context.
-u, **-U** NUM, **--unified**[=NUM]	output NUM (default 3) lines of unified context.

ASPELL

About

aspell

aspell is an interactive spell checker. It will scan your files (or anything from standard input), check for

misspellings, and allow you to correct the words interactively.

Syntax

```
aspell [options] <command>
```

Commands

<command> can be one of the following:

usage, -?	Send a brief Aspell Utility usage message to standard output. This is a short summary listing more common spell-check commands and options.
Help	Send a detailed Aspell Utility help message to standard output. This is a complete list showing all commands, options, filters and dictionaries.
version, -v	Print version number of Aspell Library and Utility to standard output.
check <file>, -c <file>	Spell-check a single file.

pipe, -a	Run Aspell in ispell -a compatibility mode.

BASIC SHELL SCRIPTS GREP

About grep

grep prints lines of input matching a specified pattern.

Syntax

```
grep [OPTIONS] PATTERN [FILE...]
```

Description

grep searches the named input FILEs (or standard input if no files are named, or if a single dash ("-") is given as the file name) for lines containing a match to the given PATTERN. By default, **grep** prints the matching lines.

In addition, three variant programs egrep, fgrep and **rgrep** are available:

- **egrep** is the same as **grep -E**.
- **fgrep** is the same as **grep -F**.
- **rgrep** is the same as **grep -r**.

Direct invocation as either **egrep** or **fgrep** is deprecated, but is provided to allow historical applications that rely on them to run unmodified.

General Options

--help	Print a help message briefly summarizing command-line options, and

	exit.
V, **--version**	Print the version number of **grep**, and exit.

SED

Sort for Stream Editor sed allows you to use pre-recorded commands to make changes to text.

Syntax

sed [-n] [-e script] ... [-f script_file] ... [file ...]

-n	Suppress the default output.
-e script	script is an edit command for sed . See USAGE below for more information on the format of script. If there is just one -e option and no -f options, the flag -e may be omitted.
-f script_file	Take the script from script_file. script_file consists of editing commands, one per line.
commands	The sed commands to perform.
File	A path name of a file whose contents will be read and edited. If multiple file operands are specified, the named files will be read in the order specified and the concatenation will be edited. If no file operands are specified, the standard input will be used

AWK(BASIC USAGE)

About awk

Short for "Aho, Weinberger, and Kernighan," **AWK** is an interpreted programming language which focuses on processing text.

AWK was developed in the 1970s at Bell Labs by Alfred Aho, Peter Weinberger, andBrian Kernighan. It was designed to execute complex pattern-matching operations on streams of textual data. It makes heavy use of strings, associative arrays, and regular expressions, and is immensely useful for parsing system data and generating automatic reports.
AWK is a direct predecessor of Perl, and is still very useful in modern systems.

TheGNU free software project distributes an open-source version of AWK called gawk.
Syntax

```
awk [ -F fs ] [ -v var=value ] [ 'prog' | -f progfile ] [ file ... ]
```

Arguments

-F *fs*	Sets the input field separator to the regular expression *fs*.
-v *var=value*	Assigns the value *value* to the variable *var* before executing the awk program.
'prog'	An **awk** program.
-f *progfile*	Specify a file, *progfile*, which contains the **awk** program to be executed.
file ...	A file to be processed by the specified **awk** program.

UNIT – IV

PROCESS RELATED COMMANDS(PS, TOP, PSTREE, NICE, RENICE)

The Linux terminal has a number of useful commands that can display running processes, kill them, and change their priority level. This post lists the classic, traditional commands, as well as some more useful, modern ones. Many of the commands here perform a single function and can be combined — that's the Unix philosophy of designing programs. Other programs, like htop, provide a friendly interface on top of the commands.

Top

The top **command is the traditional way to view your system's resource usage and see the processes that are taking up the most system resources. Top displays a list of processes, with the ones using the most CPU at the top.**

ps

The **ps** command lists running processes. This may be too many processes to read at one time, so you can pipe the output through the **less** command to scroll through them at your own pace: Press **q** to exit when you're done. The following command lists all processes running on your system:

ps -A

pstree

The **pstree** command is another way of visualizing processes. It displays them in tree format. So, for example, your X server and graphical environment would appear under the display manager that spawned them.

kill

The **kill** command can kill a process, given its process ID. You can get this information from the **ps - A**, **top** or **pgrep** commands. Technically speaking, the kill command can send any signal to a process. You can use **kill -KILL** or **kill -9** instead to kill a stubborn process

kill PID

renice

The **renice** command changes the nice value of an already running process. The nice value

determines what priority the process runs with. A value of **-19** is very high priority, while a value of **19** is very low priority. A value of **0** is the default priority. The renice command requires a process's PID. The following command makes a process run with very low priority:

renice 19 *PID*

INTRODUCTION TO THE LINUX KERNEL

A kernel is the core of an operating system. The operating system is all of the programs that manages the hardware and allows users to run applications on a computer. The kernel controls the hardware and applications. Applications do not communicate with the hardware directly; instead they go to the kernel.
In summary, software runs on the kernel and the kernel operates the hardware. Without a kernel, a computer is a useless object.
There are many reasons for a user to want to make their own kernel. Many users may want to make a kernel that only contains the code needed to run on their system. For instance, my kernel contains drivers for FireWire devices, but my computer lacks these ports. When the system boots up, time and RAM space is wasted on drivers for devices that my system does not have installed. If I wanted to streamline my kernel, I could make my own kernel that does not have FireWire drivers. As for another reason, a user may own a device with a special piece of hardware, but the kernel that came with their latest version of Ubuntu lacks the needed driver. The Linux kernel is also a preemptive multitasking kernel. This means that the kernel will pause some tasks to ensure that every application gets a chance to use the CPU. For instance, if an application is running but is waiting for some data, the kernel will put that application on hold and allow another program to use the newly freed CPU resources until the data arrives. Otherwise, the system would be wasting resources for tasks that are waiting for data or another program to execute.
The kernel will force programs to wait for the CPU or stop using the CPU. The Linux kernel makes devices appear as files in the folder /dev. USB ports, for instance, are located in /dev/bus/usb. The hard-drive partitions are seen in /dev/disk/by-label.

GETTING STARTED WITH THE KERNEL

All of the major Linux distributions (Craftworks, Debian, Slackware, Red Hat etcetera) include the kernel sources in them. Usually the Linux kernel that got installed on your Linux system was built from those sources. They are kept onftp://ftp.cs.helsinki.fi and all of the other web sites shadow them. This makes the Helsinki web site the most up to date, but sites like MIT and Sunsite are never very far behind. If you do not have access to the web, there are many CD ROM vendors who offer snapshots of the world's major web sites at a very reasonable cost. Some even offer a subscription service with quarterly or even monthly updates. Your local Linux User Group is also a good source of sources.

The Linux kernel sources have a very simple numbering system. Any even number kernel (for example 2.0.30) is a stable, released, kernel and any odd numbered kernel (for example 2.1.42 is

a development kernel. This book is based on the stable 2.0.30 source tree. Development kernels have all of the latest features and support all of the latest devices. Although they can be unstable, which may not be exactly what you want it, is important that the Linux community tries the latest kernels. That way they are tested for the whole community.

Remember that it is *always* worth backing up your system thoroughly if you do try out non-production kernels. Changes to the kernel sources are distributed as patch files. The patch utility is used to apply a series of edits to a set of source files. So, for example, if you have the 2.0.29 kernel source tree and you wanted to move to the 2.0.30 source tree, you would obtain the 2.0.30 patch file and apply the patches (edits) to that source tree:

```
$ cd /usr/src/linux

$ patch -p1 < patch-2.0.30
```

OBTAINING THE KERNEL SOURCE

"Source tree" is just a term for the directory which contains the source code you will be compiling.

When a kernel source package is installed, the source tarball is placed in the /usr/src directory. To work in this directory, you must be the either the root user or a member of the src group. To avoid doing things as root whenever possible, add yourself to src. Only the root user can add users to groups, so become root and do the following:
bash:~# adduser my_username src Adding user my_username to group src...

Done.

Of course, substitute your own user name for "my_username".

```
bash:~$ groups

my_username src
```

using patches

Copy the patch that you generated in Step 2 to the directory containing the file to which you want to apply the patch. Now, type in your shell:

patch <patch_file

where patch file is the filename of the patch file. Note that the filename of the file to be patched must be the same.If you want to apply a patch to a whole directory tree, add a -p1 to

your patch command:

patch -p1 <patch_file

the kernel source tree

The kernel source tree is divided into a number of directories, most of which contain many more subdirectories. The directories in the root of the source tree, along with their descriptions, are listed

Directory	Description
Arch	Architecture-specific source
Crypto	Crypto API
Documentation	Kernel source documentation
Drivers	Device drivers
Fs	The VFS and the individual file systems
Include	Kernel headers
Init	Kernel boot and initialization
Ipc	Interprocess communication code
Kernel	Core subsystems, such as the scheduler
Lib	Helper routines
Mm	Memory management subsystem and the VM
Net	Networking subsystem
Scripts	Scripts used to build the kernel

Directory **Description**

Security	Linux Security Module
Sound	Sound subsystem
Usr	Early user-space code (called initramfs)

Building The Kernel Process Management

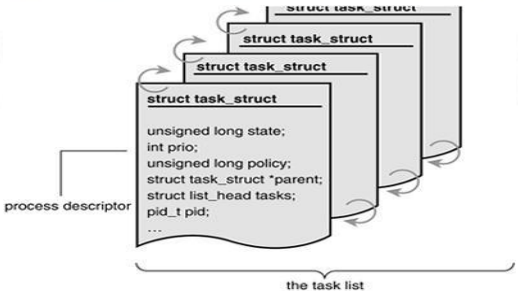
The Process Management subsystem controls the creation, termination, accounting, and scheduling of processes. It also oversees process state transitions and the switching between privileged and non-privileged modes of execution. The Process Management subsystem also facilitates and manages the complex task of the creation of child processes. A simple definition of a process is that it is an executing program. It is an entity that requires system resources, and it has a finite lifetime. It has the capability to create other processes via the system call interface. In short, it is an electronic representation of a user's or programmer's desire to accomplish some useful piece of work. A process may appear to the user as if it is the only job running in the machine. This "sleight of hand" is only an illusion. At any one time a processor is only executing a single process.

Process Descriptor and the Task Structure

The kernel stores the list of processes in a circular doubly linked list called the *task list*. Each element in the task list is a *process descriptor* of the type struct task_struct, which is defined in <Linux/sched.h>. The process descriptor contains all the information about a specific process.

The task_struct is a relatively large data structure, at around 1.7 kilobytes on a 32-bit machine. This size, however, is quite small considering that the structure contains all the information that the kernel has and needs about a process. The process descriptor contains the data that describes the executing program open files, the process's address space, pending signals, the process's state, and much more

The process descriptor and task list.

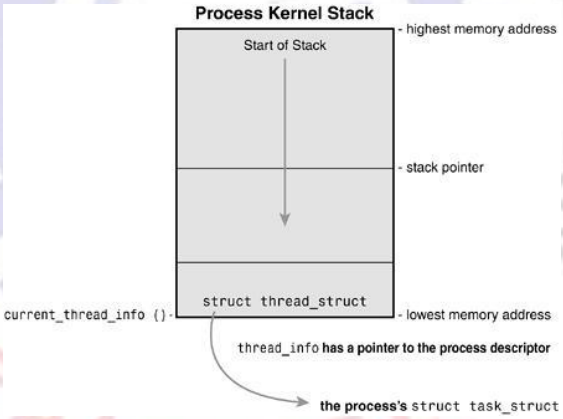


PROCESS DESCRIPTOR AND THE TASK STRUCTURE

The task_struct structure is allocated via the *slab allocator* to provide object reuse and cache coloring

.Prior to the 2.6 kernel series, struct task_struct was stored at the end of the kernel stack of each process. This allowed architectures with few registers, such as x86, to calculate the location of the process descriptor via the *stack pointer* without using an extra register to store the location. With the process descriptor now dynamically created via the slab allocator, a new structure, struct thread_info, was created that again lives at the bottom of the stack (for stacks that grow down) and at the top of the stack (for stacks that grow up) See Figure . The new structure also makes it rather easy to calculate offsets of its values for use in assembly code.

The process descriptor and kernel stack.



STORING THE PROCESS DESCRIPTOR

The system identifies processes by a unique *process identification* value or *PID*. The PID is a numerical value that is represented by the opaque type pid_t, which is typically an int. Because of backward compatibility with earlier Unix and Linux versions, however, the default maximum value is only 32,768 (that of a short int), although the value can optionally be increased to the full range afforded the type. The kernel stores this value as pid inside each process descriptor.

This maximum value is important because it is essentially the maximum number of processes that may exist concurrently on the system. Although 32,768 might be sufficient for a desktop system, large servers may require many more processes. The lower the value, the sooner the values will wrap around, destroying the useful notion that higher values indicate later run

processes than lower values. If the system is willing to break compatibility with old applications, the administrator may increase the maximum value via/proc/sys/kernel/pid_max.

PROCESS STATE

The state field of the process descriptor describes the current condition of the process Each process on the system is in exactly one of five different states. This value is represented by one of five flags:

- **TASK_RUNNING** The process is runnable; it is either currently running or on a run queue waiting to run is the only possible state for a process executing in user-space; it can also apply to a process in kernel-space that is actively running.

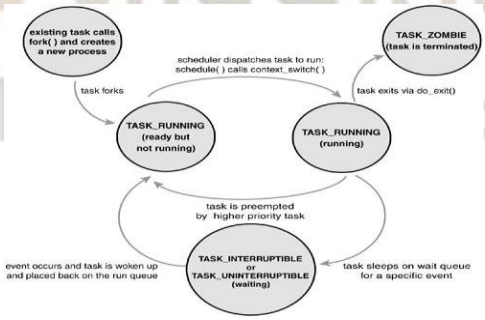
- **TASK_INTERRUPTIBLE** The process is sleeping (that is, it is blocked), waiting for some condition to exist. When this condition exists, the kernel sets the process's state toTASK_RUNNING. The process also awakes prematurely and becomes runnable if it receives a signal.

- **TASK_UNINTERRUPTIBLE** This state is identical to TASK_INTERRUPTIBLE except that it does *not* wake up and become runnable if it receives a signal. This is used in situations where the process must wait without interruption or when the event is expected to occur quite quickly. Because the task does not respond to signals in this state,TASK_UNINTERRUPTIBLE is less often used than TASK_INTERRUPTIBLE

- **TASK_ZOMBIE** The task has terminated, but its parent has not yet issued a wait4() system call. The task's process descriptor must remain in case the parent wants to access it. If the parent calls wait4(), the process descriptor is deallocated.

- **TASK_STOPPED** Process execution has stopped; the task is not running nor is it eligible to run. This occurs if the task receives the SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU signal or if it receives *any* signal while it is being debugged.

Figure 3.3. Flow chart of process states.



MANIPULATING THE CURRENT PROCESS STATE

Kernel code often needs to change a process's state. The preferred mechanism is using

```
set_task_state(task, state);    /* set task 'task' to state 'state' */
```

This function sets the given task to the given state. If applicable, it also provides a memory barrier to force ordering on other processors (this is only needed on SMP systems). Otherwise, it is equivalent to

```
task->state = state;
```

The method set_current_state(state) is synonymous to set_task_state(current, state).

PROCESS CONTEXT

One of the most important parts of a process is the executing program code. This code is read in from an *executable file* and executed within the program's address space. Normal program execution occurs in *user-space*. When a program executes a system call or triggers an exception, it enters *kernel-space*. At this point, the kernel is said to be "executing on behalf of the process" and is in *process context*. When in process context, the current macro is valid .Upon exiting the kernel, the process resumes execution in user-space, unless a higher-priority process has become runnable in the interim, in which case the scheduler is invoked to select the higher priority process. System calls and exception handlers are well-defined interfaces into the kernel. A process can begin executing in kernel-space only through one of these interfaces *all* access to the kernel is through these interfaces.

THE PROCESS FAMILY TREE

A distinct hierarchy exists between processes in UNIX systems, and Linux is no exception. All processes are descendents of the init process, whose PID is one. The kernel starts init in the last step of the boot process. The init process, in turn, reads the system *init scripts* and executes more programs, eventually completing the boot process.

Every process on the system has exactly one parent. Likewise, every process has zero or more children. Processes that are all direct children of the same parent are called *siblings*. The relationship between processes is stored in the process descriptor. Each task_struct has a pointer to the parent's task_struct, named parent, and a list of children, named children. Consequently, given the current process, it is possible to obtain the process descriptor of its parent with the following code:

```
struct task_struct *my_parent = current->parent;
```

THE LINUX SCHEDULING ALGORITHM

The Linux scheduling algorithm works by dividing the CPU time into *epochs*. In a single epoch, every process has a specified time quantum whose duration is computed when the epoch begins. In general, different processes have different time quantum durations. The time quantum value is the maximum CPU time portion assigned to the process in that epoch. When a process has exhausted its time quantum, it is preempted and replaced by another runnable process. Of course, a process can be selected several times from the scheduler in the same epoch, as long as its quantum has not been exhausted--for instance, if it suspends itself to wait for I/O, it preserves some of its time quantum and can be selected again during the same epoch. The epoch ends when all runnable processes have exhausted their quantum; in this case, the scheduler algorithm recomputes the time-quantum durations of all processes and a new epoch begins.

Each process has a *base time quantum*: it is the time-quantum value assigned by the scheduler to the process if it has exhausted its quantum in the previous epoch. The users can change the base time quantum of their processes by using the nice( ) and setpriority( ) system calls (see the section "System Calls Related to Scheduling" later in this chapter). A new process always inherits the base time quantum of its parent.

The INIT_TASK macro sets the value of the base time quantum of process 0 (*swapper*) to DEF_PRIORITY; that macro is defined as follows:

```
#define DEF_PRIORITY (20*HZ/100)
```

Since HZ, which denotes the frequency of timer interrupts, is set to 100 for IBM PCs (see the section "Programmable Interval Timer" in Chapter 5), the value of DEF_PRIORITY is 20 ticks, that is, about 210 ms.

Users rarely change the base time quantum of their processes, so DEF_PRIORITY also denotes the base time quantum of most processes in the system.

In order to select a process to run, the Linux scheduler must consider the priority of each process. Actually, there are two kinds of priority:

Static priority

> This kind is assigned by the users to real-time processes and ranges from 1 to 99. It is never changed by the scheduler.

Dynamic priority

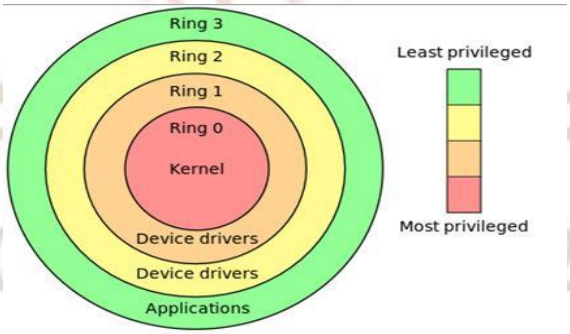
This kind applies only to conventional processes; it is essentially the sum of the base time quantum (which is therefore also called the *base priority* of the process) and of the number of ticks of CPU time left to the process before its quantum expires in the current epoch.

OVERVIEW OF SYSTEM CALLS

A system call is very much like an ordinary function call, but is implemented in a very special way. A process executes a special machine instruction (in the user space) to make a system call. The instruction execution performs two actions at one stroke. First, the instruction execution flips the processor from the user-operating mode to the kernel-operating mode. Second, it alters the current program execution flow, and the processor (in the same process context) starts executing instructions from an operating system routine called *system call handler*. That is, the (system call) instruction execution teleports the process from its private address space to the kernel space, and the process becomes "effectively" a kernel process for the duration of the system call, and we say the kernel is being executed on behalf of the process. Depending on the type of the call, the system call handler invokes the appropriate operating system service routines. When the execution of the system call handler is complete, the kernel executes another special instruction and the execution control returns to the original program from where the system call originated, and the processor automatically reverts to the user-operating mode. The system call makes an address space switch. The process, while in the kernel space, can reference entities from its private address space; for example, to read input values of the system call parameters.

INTRODUCTION TO KERNEL DEBUGGERS (IN WINDOWS AND LINUX)

Before trying to debug the kernel, we must first understand a few things. We must know what the Rings in computer security are. Let's take a look at the picture :



On the picture above, we can see four protection rings, which are mechanisms to protect data and functionality from faults and malicious behavior. Each protection ring provides access to certain resources within the computer system, which is generally hardware-enforced. The most

privileged ring is the ring 0 (kernel mode) and the least privileged ring is the ring 3 (user mode). Ring 0 has direct access to the hardware, such as CPU and memory.

There are special gates between the outer rings to access the inner ring's resources. Correctly limiting access between rings can improve security by preventing programs from one ring or privilege level from misusing resources intended for programs in another. Despite the picture above showing four rings, only two are being used: ring 0 and ring 3 are for kernel and user mode. If we would like to use protections rings successfully, the operating system must closely work with the underlying hardware. But it's often the case that operating systems are designed to work on different hardware, so the operating system can only use a limited number of rings: in most Windows systems, only 2 rings are used. When the process or a thread is being run by the system, that process/thread has direct access to the privileged functions like accessing real memory, modifying descriptor tables, disabling interrupts, etc… When we would like to use kernel mode under Windows/Linux, we need to perform a system call into kernel mode where the system call is executed and after that, the control is returned to the user space. The real purpose of the kernel and user mode is to provide protection against system corruption. Let's say we've just written a program that tries to access some non-existent memory address like 0×00000000. In such cases, since the program is executing in the user mode, only the program will crash; the system will be left unaffected. Now imagine what would happen if we wouldn't have different protections rings: one program like this can endanger the stability of the whole system, in which case the whole system can crash. I'm not saying this can't happen, because we all know the Window's blue screen of death, but this is a rare occurrence that is the result of a bug in the Windows system. This is because the program misbehaves and a fault/exception is generated in user mode, which doesn't affect the stability of the system. But if a fault/exception is generated in the kernel mode, the whole operating system can become unstable and crash the computer.

When debugging, we must also be aware of the fact that all loaded modules the program needs are still in user mode; so kernel32.dll, ntdll.dll and other DLLs are loaded in user-mode. Those DLLs are just gateways into the kernel-mode, but they also provide the error handling and parameter verification. This is because the kernel must receive a valid function call that can't endanger the stability of the system.

References:

Sumitabha Das, "Unix Concepts and Application", TMH

Robert Love, "Linux Kernel Development", Pearson Education

Sumitabha Das, "Your Unix The Ultimate Guide", TMH