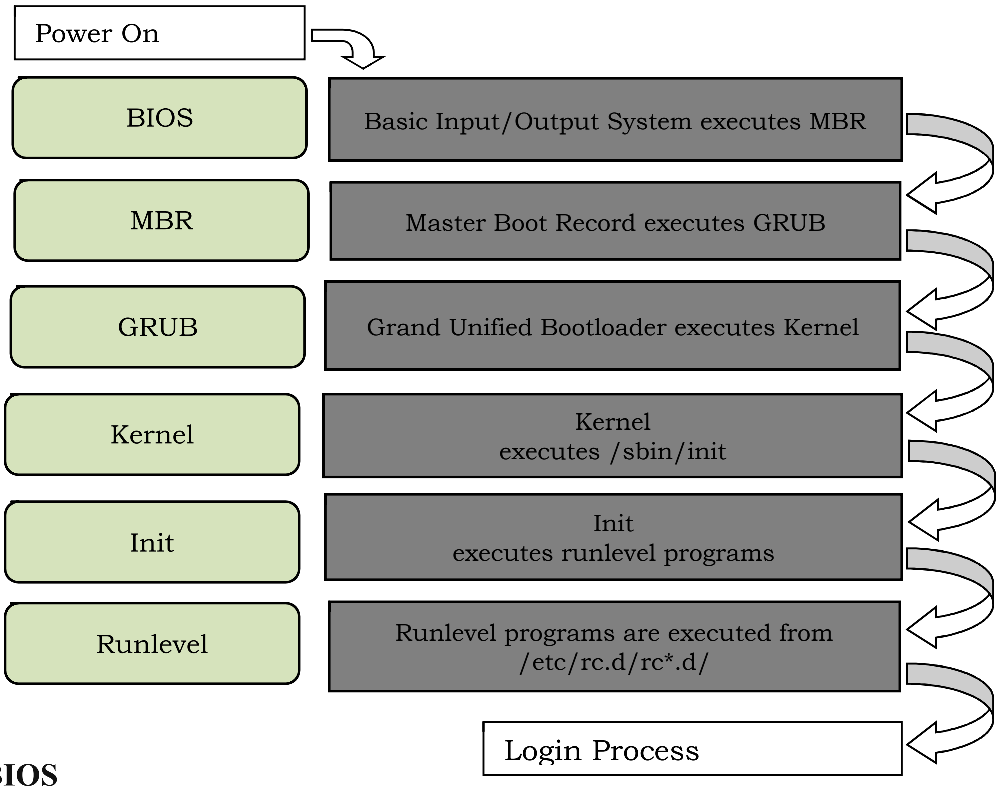
| Ex. No. 1a | BOOTING PROCESS OF LINUX | Date : |
|---|---|---|

Press the power button on your system, and after few moments you see the Linux login prompt. From the time you press the power button until the Linux login prompt appears, the following sequence occurs. The following are the 6 high level stages of a typical Linux boot process.


Power On
BIOS
Basic Input/Output System executes MBR
MBR
Master Boot Record executes GRUB
GRUB
Grand Unified Bootloader executes Kernel
Kernel
Kernel executes /sbin/init
Init
Init executes runlevel programs
Runlevel
Runlevel programs are executed from /etc/rc.d/rc*.d/
Login Process

**Step 1.BIOS**

- BIOS stands for Basic Input/Output System
- Performs some system integrity checks
- Searches, loads, and executes the boot loader program.
- It looks for boot loader in floppy, CD-ROMs, or hard drive. You can press a key (typically F12 or F2, but it depends on your system) during the BIOS startup to change the boot sequence.
- Once the boot loader program is detected and loaded into the memory, BIOS gives the control to it.
- So, in simple terms BIOS loads and executes the MBR boot loader.

**Step 2. MBR**

- MBR stands for Master Boot Record.
- It is located in the 1st sector of the bootable disk. Typically /dev/hda, or /dev/sda
- MBR is less than 512 bytes in size. This has three components 1) primary boot loader info in 1st 446 bytes 2) partition table info in next 64 bytes 3) mbr validation check in last 2 bytes.
- It contains information about GRUB (or LILO in old systems).
- So, in simple terms MBR loads and executes the GRUB boot loader.

**Step 3. GRUB**

- GRUB stands for Grand Unified Bootloader.
- If you have multiple kernel images installed on your system, you can choose which one to be executed.
- GRUB displays a splash screen, waits for few seconds, if you don't enter anything, it loads the default kernel image as specified in the grub configuration file.
- GRUB has the knowledge of the filesystem (the older Linux loader LILO didn't understand filesystem).
- Grub configuration file is /boot/grub/grub.conf (/etc/grub.conf is a link to this). The following is sample grub.conf of CentOS.

```
#boot=/dev/sda
default=0
timeout=5
splashimage=(hd0,0)/boot/grub/splash.xpm.gz
hiddenmenu
title CentOS(2.6.18-194.el5PAE)
root(hd0,0)
kernel/boot/vmlinuz-2.6.18-194.el5PAE ro root=LABEL=/
initrd /boot/initrd-2.6.18-194.el5PAE.img
```

- As you notice from the above info, it contains kernel and initrd image.
- So, in simple terms GRUB just loads and executes Kernel and initrd images.

**Step 4. Kernel**

- Mounts the root file system as specified in the "root=" in grub.conf
- Kernel executes the /sbin/init program
- Since init was the 1st program to be executed by Linux Kernel, it has the process id (PID) of 1. Do a 'ps -ef | grep init' and check the pid.
- initrd stands for Initial RAM Disk.
- initrd is used by kernel as temporary root file system until kernel is booted and the real root file system is mounted. It also contains necessary drivers compiled inside, which helps it to access the hard drive partitions, and other hardware.

**Step 5. Init**

- Looks at the /etc/inittab file to decide the Linux run level.
- Following are the available run levels
    - 0 – halt
    - 1 – Single user mode
    - 2 – Multiuser, without NFS
    - 3 – Full multiuser mode
    - 4 – unused
    - 5 – X11
    - 6 – reboot

- Init identifies the default initlevel from /etc/inittab and uses that to load all appropriate program.
- Execute 'grep initdefault /etc/inittab' on your system to identify the default run level
- If you want to get into trouble, you can set the default run level to 0 or 6. Since you know what 0 and 6 means, probably you might not do that.
- Typically you would set the default run level to either 3 or 5.

**Step 6. Runlevel programs**

- When the Linux system is booting up, you might see various services getting started. For example, it might say "starting sendmail …. OK". Those are the runlevel programs, executed from the run level directory as defined by your run level.
- Depending on your default init level setting, the system will execute the programs from one of the following directories.
  - Run level 0 – /etc/rc.d/rc0.d/
  - Run level 1 – /etc/rc.d/rc1.d/
  - Run level 2 – /etc/rc.d/rc2.d/
  - Run level 3 – /etc/rc.d/rc3.d/
  - Run level 4 – /etc/rc.d/rc4.d/
  - Run level 5 – /etc/rc.d/rc5.d/
  - Run level 6 – /etc/rc.d/rc6.d/
- Please note that there are also symbolic links available for these directory under /etc directly. So, /etc/rc0.d is linked to /etc/rc.d/rc0.d.
- Under the /etc/rc.d/rc*.d/ directories, you would see programs that start with S and K.
- Programs starts with S are used during startup. S for startup.
- Programs starts with K are used during shutdown. K for kill.
- There are numbers right next to S and K in the program names. Those are the sequence number in which the programs should be started or killed.
- For example, S12syslog is to start the syslog deamon, which has the sequence number of 12. S80sendmail is to start the sendmail daemon, which has the sequence number of 80. So, syslog program will be started before sendmail.

**Login Process**

1. Users enter their username and password
2. The operating system confirms your name and password.
3. A "shell" is created for you based on your entry in the "/etc/passwd" file
4. You are "placed" in your "home"directory.
5. Start-up information is read from the file named "/etc/profile". This file is known as the system login file. When every user logs in, they read the information in this file.
6. Additional information is read from the file named ".profile" that is located in your "home" directory. This file is known as your personal login file.

| **Ex. No. 1b** | **BASIC LINUX COMMANDS** | **Date :** |
|---|---|---|

**a) Basics**

| | |
|---|---|
| 1. *echo* SRM | → to display the string SRM |
| 2. *clear* | → to clear the screen |
| 3. *date* | → to display the current date and time |
| 4. *cal* 2003 | → to display the calendar for the year 2003 |
| *cal* 6 2003 | → to display the calendar for the June-2003 |
| 5. *passwd* | → to change password |

**b) Working with Files**

| | |
|---|---|
| 1. *ls* | → list files in the present working directory |
| *ls –l* | → list files with detailed information (long list) |
| *ls –a* | → list all files including the hidden files |
| 2. *cat* f1 | → display the content of the file f1 |
| 3. *wc* f1 | → list no. of characters, words & lines of a file f1 |
| *wc –c* f1 | → list only no. of characters of file f1 |
| *wc –w* f1 | → list only no. of words of file f1 |
| *wc –l* f1 | → list only no. of lines of file f1 |
| 4. *cp* f1 f2 | → copy file f1 into f2 |
| 5. *mv* f1 f2 | → rename file f1 as f2 |
| 6. *rm* f1 | → remove the file f1 |
| 7. *head* –5 f1 | → list first 5 lines of the file f1 |
| *tail* –5 f1 | → list last 5 lines of the file f1 |

**c) Working with Directories**

| | |
|---|---|
| 1. *mkdir* elias | → to create the directory elias |
| 2. *cd* elias | → to change the directory as elias |
| 3. *rmdir* elias | → to remove the directory elias |
| 4. *pwd* | → to display the path of the present working directory |
| 5. *cd* | → to go to the home directory |
| *cd ..* | → to go to the parent directory |
| *cd -* | → to go to the previous working directory |
| *cd /* | → to go to the root directory |

**d) File name substitution**

1. *ls f?* → list files start with 'f' and followed by any one character
2. *ls* *.c → list files with extension 'c'
3. *ls* [gpy]et → list files whose first letter is any one of the character g, p or y and followed by the word et
4. *ls* [a-d,l-m]ring → list files whose first letter is any one of the character from a to d and l to m and followed by the word ring.

**e) I/O Redirection**

*1.* Input redirection
*wc –l* < ex1 → To find the number of lines of the file 'ex1'

*2.* Output redirection
*who* > f2 → the output of 'who' will be redirected to file f2
cat > f1 → to create a file. That is standard input data will be redirected to the file f1

*3. cat* >> f1 → to append more into the file f1

**f) Piping**

Syntax : Command1 | command2

Output of the command1 is transferred to the command2 as input. Finally output of the command2 will be displayed on the monitor.

ex. *cat* f1 | *more* → list the contents of file f1 screen by screen

*head* –6 f1 |*tail* –2 → prints the 5th & 6th lines of the file f1.

**g) Environment variables**

*1. echo $HOME* → display the path of the home directory

*2. echo $PS1* → display the prompt string $

*3. echo $PS2* → display the second prompt string ( > symbol by default )

*4. echo $LOGNAME* → login name

*5. echo $PATH* → list of pathname where the OS searches for an executable file

**h) File Permission**

-- chmod command is used to change the access permission of a file.

*Method-1*

Syntax : *chmod* [ugo] [+/-] [ rwxa ] filename

u : user, g : group, o : others
+ : Add permission - : Remove the permission
r : read, w : write, x : execute, a : all permissions

ex. *chmod* ug+rw f1
adding 'read & write' permissions of file f1 to both user and group members.

*Method-2*

Syntax : *chmod* octnum file1

The 3 digit octal number represents as follows

| | |
|---|---|
| • first digit | -- file permissions for the user |
| • second digit | -- file permissions for the group |
| • third digit | -- file permissions for others |

Each digit is specified as the sum of following
4 – read permission, 2 – write permission, 1 – execute permission

ex. *chmod* 754 f1

it change the file permission for the file as follows

- read, write & execute permissions for the user ie; 4+2+1 = 7
- read, & execute permissions for the group members ie; 4+0+1 = 5
- only read permission for others ie; 4+0+0 = 4

***Verified by***

| | |
|---|---|
| **Staff In-charge Sign :** | **Date :** |

| Ex. No. 2a | LINUX FILE SYSTEM | Date : |
|---|---|---|

**Linux File System**
Linux File System or any file system generally is a layer which is under the operating system that handles the positioning of your data on the storage, without it; the system cannot knows which file starts from where and ends where.

Linux offers many file systems types like:

- **Ext**: an old one and no longer used due to limitations.
- **Ext2**: first Linux file system that allows 2 terabytes of data allowed.
- **Ext3**: came from Ext2, but with upgrades and backward compatibility.
- **Ext4**: faster and allow large files with significant speed.**(Best Linux File System)** It is a very good option for SSD disks and you notice when you try to install any Linux distro that this one is the default file system that Linux suggests.
- **JFS**: old file system made by IBM. It works very well with small and big files, but it failed and files corrupted after long time use, reports say.
- **XFS**: old file system and works slowly with small files.
- **Btrfs:** made by Oracle. It is not stable as Ext in some distros, but you can say that it is a replacement for it if you have to. It has a good performance.

**File System Structure**
The following table provides a short overview of the most important higher-level directories you find on a Linux system

| Directory | Contents |
|---|---|
| `/` | Root directory—the starting point of the directory tree. |
| `/bin` | Essential binary files. Binary Executable files |
| `/boot` | Static files of the boot loader. |
| `/dev` | Files needed to access host-specific devices. |
| `/etc` | Host-specific system configuration files. |
| `/lib` | Essential shared libraries and kernel modules. |
| `/media` | Mount points for removable media. |
| `/mnt` | Mount point for temporarily mounting a file system. |
| `/opt` | Add-on application software packages. |
| `/root` | Home directory for the superuser `root`. |
| `/sbin` | Essential system binaries. |
| `/srv` | Data for services provided by the system. |
| `/tmp` | Temporary files. |
| `/usr` | Secondary hierarchy with read-only data. |
| `/var` | Variable data such as log files |

| Ex. No. 2b | EDITORS AND FILTERS | Date : |
|---|---|---|

## VI EDITOR

- vi fname → to open the file fname

- There are two types of mode in vi editor
  Escape mode – used to give commands – to switch to escape mode, press <Esc> key
  Command mode – used to edit the text – to switch to command mode, press any one the following inserting text command

**a) Inserting Text**

| | |
|---|---|
| **i** | → insert text before the cursor |
| **a** | → append text after the cursor |
| **I** | → insert text at the beginning of the line |
| **A** | → append text to the end of the line |
| **r** | → replace character under the cursor with the next character typed |
| **R** | → Overwrite characters until the end of the line |
| **o** | → (small o) open new line after the current line to type text |
| **O** | → (capital O) open new line before the current line to type text |

**b) Cursor movements**

| | |
|---|---|
| **h** | → left |
| **j** | → down |
| **k** | →up |
| **l** | → right |

(The arrow keys usually work also)

| | |
|---|---|
| ^**F** | →forward one screen |
| ^**B** | →back one screen |
| ^**D** | →down half screen |
| ^**U** | →up half screen |

(^ indicates control key; case does not matter)

**0** → (zero) beginning of line
**$** → end of line

**c) Deleting text**

Note : (n) indicates a number, and is optional

| | | |
|---|---|---|
| **dd** | → deletes current line | |
| (n)**dd** | → deletes (n) line(s) | ex. 5dd → deletes 5 lines |
| (n)**dw** | → deletes (n) word(s) | |
| **D** | → deletes from cursor to end of line | |
| **x** | → deletes current character | |
| (n)**x** | → deletes (n) character(s) | |
| **X** | → deletes previous character | |

**d) Saving files**

| | |
|---|---|
| :w | → to save & resume editing (write & resume) |
| :wq | → to save & exit (write & quit) |
| :q! | → quit without save |

**e) Cut, Copy and Paste**

| | |
|---|---|
| yy | → copies current line |
| (n) yy | → copies (n) lines from the current line. ex. 4yy copies 4 lines. |
| p | → paste deleted or yanked (copied) lines after the cursor |

## FILTERS

### 1. cut

- Used to cut characters or fileds from a file/input

Syntax : **cut** **-c**chars filename
**-f**fieldnos filename

- By default, tab is the filed separator(delimiter). If the fileds of the files are separated by any other character, we need to specify explicitly by **–d** option

**cut** **-d**delimitchar **-f**fileds filname

### 2. paste

- Paste files vertically. That is nth line of first file and nth line of second file are pasted as the nth line of result

Syntax : **paste** file1 file2

| | |
|---|---|
| **-d**dchar | option is used to paste the lines using the delimiting character *dchar* |
| **-s** | option is used paste the lines of the file in a single line |

### 3. tr

- Used to translate characters from standard input

Syntax : **tr** char1 char2 < filename

It translates char1 into char2 in file filename

- Octal representation characters can also be used

| Octal value | Character |
|---|---|
| '\7' | Bell |
| '\10' | Backspace |
| '\11' | Tab |
| '\12' | Newline |
| '\33' | Escape |

Ex. tr : '\11' < f1 *translates all* **:** *into tab of f ile f1*

| | |
|---|---|
| **-s** | Option translate multiple occurrences of a character by single character. |
| **-d** | Option is to delete a character |

**4. grep**

- Used to search one or more files for a particular pattern.

Syntax : **grep** pattern filename(s)

- Lines that contain the *pattern* in the file(s) get displayed
- pattern can be any regular expressions
- More than one files can be searched for a pattern

| | |
|---|---|
| **-v** | option displays the lines that do not contain the *pattern* |
| **-l** | list only name of the files that contain the *pattern* |
| **-n** | displays also the line number along with the lines that matches the *pattern* |

**5. sort**

- Used to sort the file in order

Syntax : **sort** filename

- Sorts the data as text by default
- Sorts by the first filed by default

| | |
|---|---|
| **-r** | option sorts the file in descending order |
| **-u** | eliminates duplicate lines |
| **-o** filename | writes sorted data into the file *fname* |
| **-t**dchar | sorts the file in which fileds are separated by *dchar* |
| **-n** | sorts the data as number |
| **+1n** | skip first filed and sort the file by second filed numerically |

**6. Uniq**

- Displays unique lines of a sorted file

Syntax : **uniq** filename

| | |
|---|---|
| **-d** | option displays only the duplicate lines |
| **-c** | displays unique lines with no. of occurrences. |

**7. cmp**

- Used to compare two files

Syntax : **cmp** f1 f2

compare two files f1 & f2 and prints the line of first difference .

**8. diff**

- Used to differentiate two files

Syntax : **diff** f1 f2
compare two files f1 & f2 and prints all the lines that are differed between f1 & f2.

**9. comm**

- Used to compare two sorted files
Syntax : **comm** file1 file2

Three columns of output will be displayed.
First column displays the lines that are unique to file1
Second column displays the lines that are unique to file2
Third column displays the lines that are appears in both the files

| | |
|---|---|
| -1 | option suppress first column |
| -2 | option suppress second column |
| -3 | option suppress third column |
| -12 | option display only third column |
| -13 | option display only second column |
| -23 | option display only first column |

**Q1.** Write a command to cut 5 to 8 characters of the file *f1*.
$

**Q2.** Write a command to display user-id of all the users in your system.
$

**Q3.** Write a command to paste all the lines of the file *f1* into single line
$

**Q4.** Write a command to cut the first field of file *f1* and second field of file *f2* and paste into the file *f3*.
$

**Q5.** Write a command to change all small case letters to capitals of file *f2*.
$

**Q6.** Write a command to replace all *tab* character in the file *f2* by **:**
**$**

**Q7.** Write a command to check whether the user j*udith* is available in your system or not. (use grep)
$

**Q8.** Write a command to display the lines of the file *f1* starts with SRM.

$

**Q9.** Write a command to display the name of the files in the directory */etc/init.d* that contains the pattern *grep*.

$

**Q10.** Write a command to display the names of nologin users. (Hint: the command *nologin* is specified in the last filed of the file /etc/passwd for nologin users)

$

**Q11.** Write a command to sort the file /etc/passwd in descending order

$

**Q12.** Write a command to sort the file /etc/passwd by user-id numerically. (Hint : user-id is in 3rd field)

$

**Q13.** Write a command to sort the file *f2* and write the output into the file *f22*. Also eliminate duplicate lines.

$

**Q14.** Write a command to display the unique lines of the sorted file *f21*. Also display the number of occurrences of each line.

$

**Q15.** Write a command to display the lines that are common to the files *f1* and *f2*.

$

***Verified by***

| | |
|---|---|
| **Staff In-charge Sign :** | **Date :** |

| Ex. No. 3a | COMPILATION OF C PROGRAM | Date : |
|---|---|---|

**Compilation of C Program**

Step 1 : Open the terminal and edit your program using vi editor/gedit editor and save with extension ".c"

```
Ex.    vi  test.c
       (or)
       gedit text.c
```

Step 2 : Compile your program using gcc compiler

```
Ex.    gcc  test.c   → Output file will be "a.out"
       (or)
       gcc –o test text.c  → Output file will be "test"
```

Step 3 : Correct the errors if any and run the program

```
Ex.    ./a.out
       or
       ./test
```

Optional Step : In order to avoid **./** prefix each time a program is to be executed, insert the following as the last line in the file **.profile**

```
export PATH=.:$PATH
```

This Step needs only to be done once.

**Debug C Programs using gdb debugger**

Step 1 : Compile C program with debugging option –g

```
Ex.    gcc –g test.c
```

Step 2 : Launch gdb. You will get gdb prompt

```
Ex.    gdb a.out
```

Step 3 : Step break points inside C program

```
Ex.    (gdb) b 10
```

Break points set up at line number 10. We can have any number of break points

Step 4 : Run the program inside gdb

```
Ex.    (gdb) r
```

Step 5 : Print variable to get the intermediate values of the variables at break point

```
Ex.    (gdb) p i        → Prints the value of the variable 'i'
```

Step 6 : Continue or stepping over the program using the following gdb commands

c → continue till the next break
n → Execute the next line. Treats function as single statement
s → Similar to 'n' but executes function statements line by line
l → List the program statements

Step 7 : Quit the debugger

```
(gdb) q
```

| Ex. No. 3b | PROCESS CREATION | Date : |
|---|---|---|

**Syntax for process creation**

int fork();

Returns 0 in child process and child process ID in parent process.

**Other Related Functions**

| | |
|---|---|
| int getpid() | → returns the current process ID |
| int getppid() | → returns the parent process ID |
| wait() | → makes a process wait for other process to complete |

**Virtual fork**

vfork() function is similar to fork but both processes shares the same address space.

**Q1. Find the output of the following program**

```
#include <stdio.h>
#include<unistd.h>

int main()
{
  int a=5,b=10,pid;

  printf("Before fork a=%d b=%d \n",a,b);
  pid=fork();

  if(pid==0)
  {
    a++;
    b++;
    printf("In child a=%d b=%d \n",a,b);
  }
  else
  {
    a++;
    b++;
    printf("In Parent a=%d b=%d \n",a,b);
  }
  return 0;
}
```

**Output :-**

**Q2. Rewrite the program in Q1 using vfork() and write the output**

**Q3. Calculate the number of times the text "SRMIST" is printed.**

```
#include <stdio.h>
#include<unistd.h>

int main()
{
      fork();
      fork();
      fork();
      printf("SRMIST\n");
      return 0;
}
```

**Output :**

**Q4. Complete the following program as described below :**
The child process calculates the sum of odd numbers and the parent process calculate the sum of even numbers up to the number ‘n’. Ensure the Parent process waits for the child process to finish.

```
#include <stdio.h>
#include<unistd.h>

int main()
{
  int pid,n,oddsum=0,evensum=0;

  printf("Enter the value of n : ",a);
  scanf("%d",&n);
  pid=fork();
  // Complete the program















  return 0;
}
```

**Sample Output :**

```
Enter the value of n        : 10
Sum of odd numbers          : 25
Sum of even numbers         : 30
```

**Q5. How many child processes are created for the following code?**
Hint : Check with small values of ‘n’.

```
for (i=0; i<n; i++)
        fork();
```

**Output :**

**Q6. Write a program to print the Child process ID and Parent process ID in both Child and Parent processes**

```
#include <stdio.h>
#include<unistd.h>
int main()
{









return 0;
}
```

**Sample Output:**

```
In Child Process
Parent Process ID       :     18
Child Process ID        :     20

In Parent Process
Parent Process ID       :     18
Child Process ID        :     20
```

**Q7. How many child processes are created for the following code?**

```
#include <stdio.h>
#include<unistd.h>

int main()
{
      fork();
      fork()&&fork()||fork();
      fork();
      printf("Yes");
      return 0;
}
```

**Output :**

***Verified by***

| **Staff In-charge Sign :** | **Date :** |
|---|---|