UNIT 5 I/O SYSTEM,LINUX & SHELL PROGRAMMING **10 Hrs.**

Mass Storage Structure - Disk Structure- Disk Scheduling - Disk Management - Swap Space Management - RAID Structure - Shell Operation Commends - Linux File Structure - File Management Operation - Internet Service - Telnet - FTP - Filters & Regular Expressions - Shell Programming - Variable, Arithmetic Operations, Control Structures, Handling Date, Time & System Information.

Max. 45 Hours

TEXT / REFERENCE BOOKS

1. Abraham Silberschatz, Peter Galvin and Gagne, "Operating System Concepts", 6th Edition, Addison Wesley, 2002.
2. Harvey M.Deitel, "Operating System", 2nd Edition, Addison Wesley, 2000.
3 Gary Nutt, "Operating System, A modern perspective", 2nd Edition, Addison Wesley, 2000.
4 Richard Peterson, "Linux : The Complete Reference", 6th Edition, Tata McGraw Hills, 2007.

Mass-Storage Systems

- Overview of Mass Storage Structure
- Disk Structure
- Disk Attachment
- Disk Scheduling
- Disk Management
- Swap-Space Management
- RAID Structure
- Stable-Storage Implementation

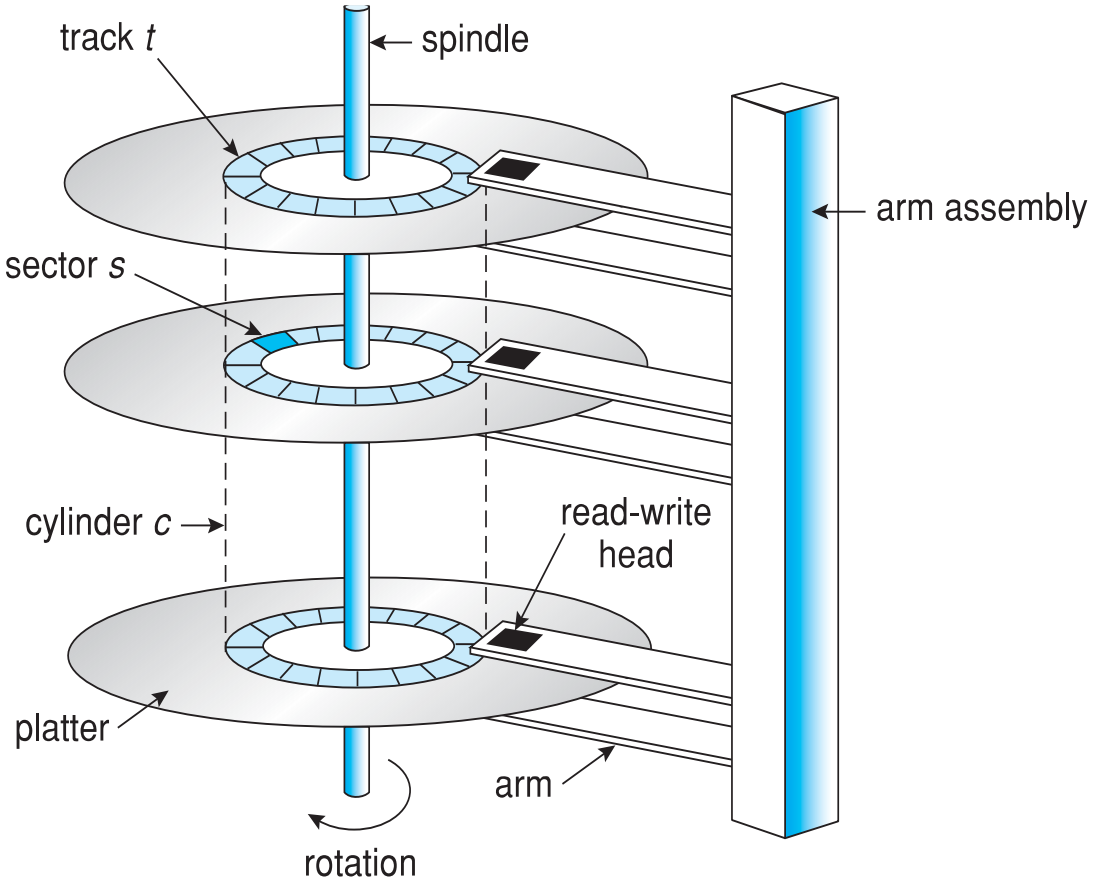
Objectives

- To describe the physical structure of secondary storage devices and its effects on the uses of the devices
- To explain the performance characteristics of mass-storage devices
- To evaluate disk scheduling algorithms
- To discuss operating-system services provided for mass storage, including RAID

Overview of Mass Storage Structure

- **Magnetic disks** provide bulk of secondary storage of modern computers
  - Drives rotate at 60 to 250 times per second
  - **Transfer rate** is rate at which data flow between drive and computer
  - **Positioning time** (**random-access time**) is time to move disk arm to desired cylinder (**seek time**) and time for desired sector to rotate under the disk head (**rotational latency**)
  - **Head crash** results from disk head making contact with the disk surface -- That's bad
- Disks can be removable
- Drive attached to computer via **I/O bus**
  - Busses vary, including **EIDE**, **ATA**, **SATA**, **USB**, **Fibre Channel**, **SCSI, SAS, Firewire**
  - **Host controller** in computer uses bus to talk to **disk controller** built into drive or storage array

Moving-head Disk Mechanism



Hard Disks

- Platters range from .85” to 14” (historically)
  - Commonly 3.5”, 2.5”, and 1.8”
- Range from 30GB to 3TB per drive
- Performance
  - Transfer Rate – theoretical – 6 Gb/sec
  - Effective Transfer Rate – real – 1Gb/sec
  - Seek time from 3ms to 12ms – 9ms common for desktop drives
  - Average seek time measured or calculated based on 1/3 of tracks
  - Latency based on spindle speed
    - 1 / (RPM / 60) = 60 / RPM
  - Average latency = ½ latency

Spindle [rpm]	Average latency [ms]
4200	7.14
5400	5.56
7200	4.17
10000	3
15000	2

(From Wikipedia)

Hard Disk Performance

- **Access Latency** = **Average access time** = average seek time + average latency
  - For fastest disk 3ms + 2ms = 5ms
  - For slow disk 9ms + 5.56ms = 14.56ms
- Average I/O time = average access time + (amount to transfer / transfer rate) + controller overhead
- For example to transfer a 4KB block on a 7200 RPM disk with a 5ms average seek time, 1Gb/sec transfer rate with a .1ms controller overhead =
  - 5ms + 4.17ms + 0.1ms + transfer time =
  - Transfer time = 4KB / 1Gb/s * 8Gb / GB * 1GB / 1024^2KB = 32 / (1024^2) = 0.031 ms
  - Average I/O time for 4KB block = 9.27ms + .031ms = 9.301ms

The First Commercial Disk Drive

1956
IBM RAMDAC computer included the IBM Model 350 disk storage system

5M (7 bit) characters
50 x 24” platters
Access time = < 1 second

Solid-State Disks

- Nonvolatile memory used like a hard drive
  - Many technology variations
- Can be more reliable than HDDs
- More expensive per MB
- Maybe have shorter life span
- Less capacity
- But much faster
- Busses can be too slow -> connect directly to PCI for example
- No moving parts, so no seek time or rotational latency

Magnetic Tape

- Was early secondary-storage medium
  - Evolved from open spools to cartridges
- Relatively permanent and holds large quantities of data
- Access time slow
- Random access ~1000 times slower than disk
- Mainly used for backup, storage of infrequently-used data, transfer medium between systems
- Kept in spool and wound or rewound past read-write head
- Once data under head, transfer rates comparable to disk
  - 140MB/sec and greater
- 200GB to 1.5TB typical storage
- Common technologies are LTO-{3,4,5} and T10000

Disk Structure

- Disk drives are addressed as large 1-dimensional arrays of **logical blocks**, where the logical block is the smallest unit of transfer
  - Low-level formatting creates **logical blocks** on physical media
- The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially
  - Sector 0 is the first sector of the first track on the outermost cylinder
  - Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost
  - Logical to physical address should be easy
    - Except for bad sectors
    - Non-constant # of sectors per track via constant angular velocity

Disk Attachment

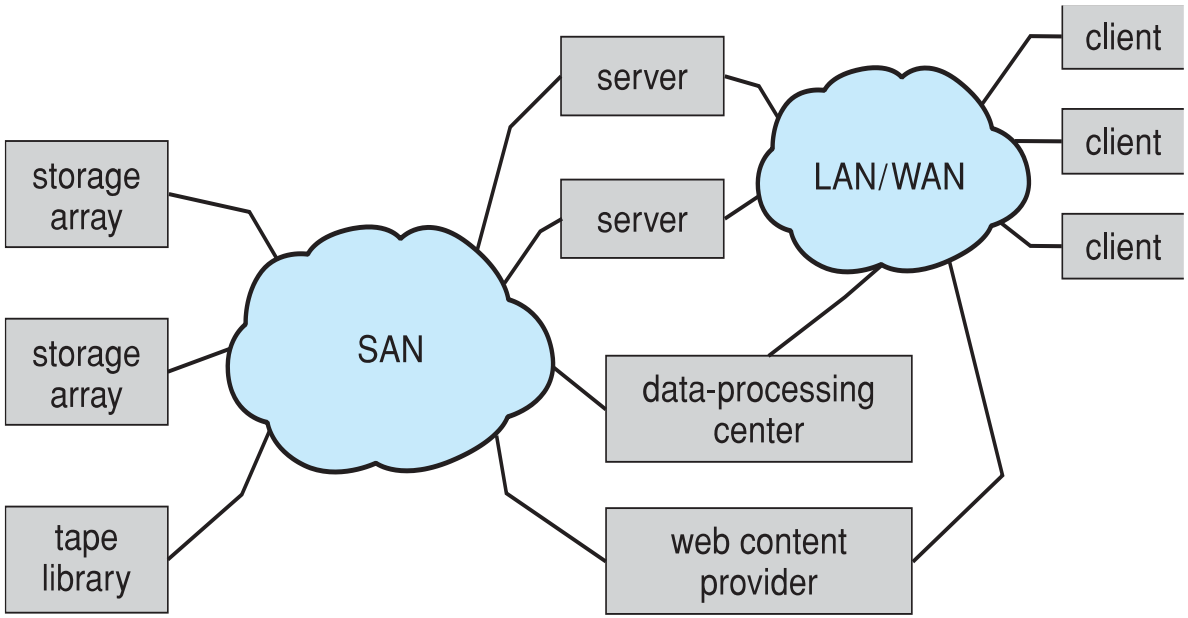
- Host-attached storage accessed through I/O ports talking to I/O busses
- SCSI itself is a bus, up to 16 devices on one cable, **SCSI initiator** requests operation and **SCSI targets** perform tasks
  - Each target can have up to 8 **logical units** (disks attached to device controller)
- FC is high-speed serial architecture
  - Can be switched fabric with 24-bit address space – the basis of **storage area networks (SANs)** in which many hosts attach to many storage units
- I/O directed to bus ID, device ID, logical unit (LUN)

Storage Array

- Can just attach disks, or arrays of disks
- Storage Array has controller(s), provides features to attached host(s)
  - Ports to connect hosts to array
  - Memory, controlling software (sometimes NVRAM, etc)
  - A few to thousands of disks
  - RAID, hot spares, hot swap (discussed later)
  - Shared storage -> more efficiency
  - Features found in some file systems
    - Snaphots, clones, thin provisioning, replication, deduplication, etc

Storage Area Network

- Common in large storage environments
- Multiple hosts attached to multiple storage arrays - flexible

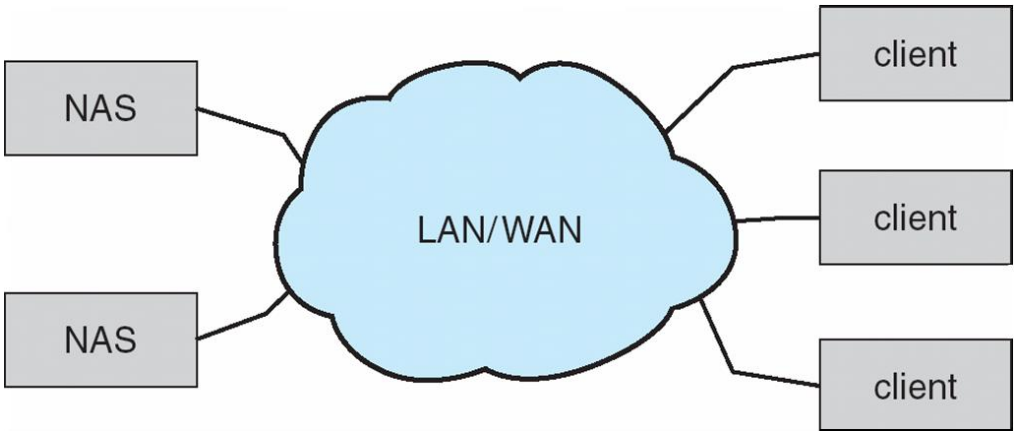


Storage Area Network (Cont.)

- SAN is one or more storage arrays
  - Connected to one or more Fibre Channel switches
- Hosts also attach to the switches
- Storage made available via **LUN Masking** from specific arrays to specific servers
- Easy to add or remove storage, add new host and allocate it storage
  - Over low-latency Fibre Channel fabric
- Why have separate storage networks and communications networks?
  - Consider iSCSI, FCOE

Network-Attached Storage

- Network-attached storage (**NAS**) is storage made available over a network rather than over a local connection (such as a bus)
  - Remotely attaching to file systems
- NFS and CIFS are common protocols
- Implemented via remote procedure calls (RPCs) between host and storage over typically TCP or UDP on IP network
- **iSCSI** protocol uses IP network to carry the SCSI protocol
  - Remotely attaching to devices (blocks)



Disk Scheduling

- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth
- Minimize seek time
- Seek time ≈ seek distance
- Disk **bandwidth** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer

Disk Scheduling (Cont.)

- There are many sources of disk I/O request
  - OS
  - System processes
  - Users processes
- I/O request includes input or output mode, disk address, memory address, number of sectors to transfer
- OS maintains queue of requests, per disk or device
- Idle disk can immediately work on I/O request, busy disk means work must queue
  - Optimization algorithms only make sense when a queue exists

Disk Scheduling (Cont.)

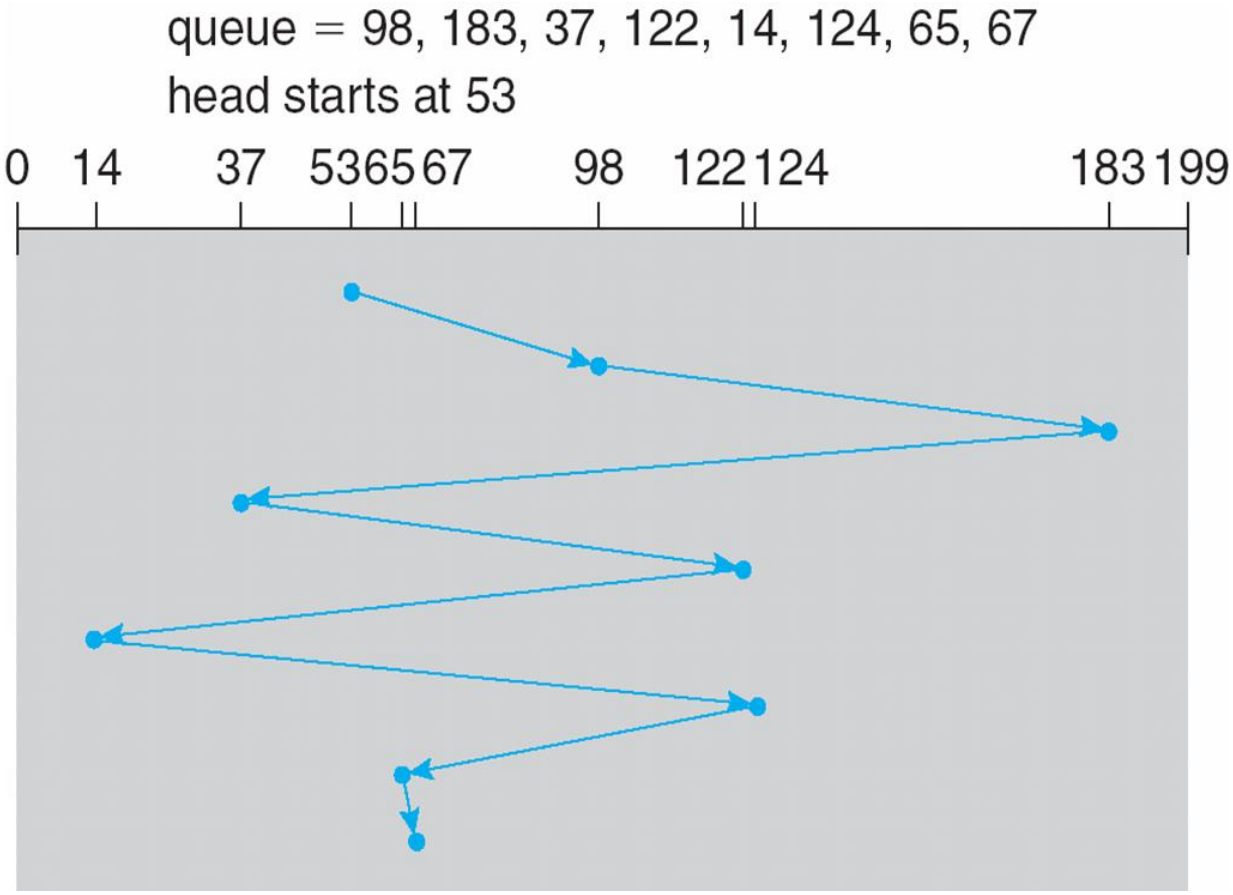
- Note that drive controllers have small buffers and can manage a queue of I/O requests (of varying “depth”)
- Several algorithms exist to schedule the servicing of disk I/O requests
- The analysis is true for one or many platters
- We illustrate scheduling algorithms with a request queue (0-199)

98, 183, 37, 122, 14, 124, 65, 67

Head pointer 53

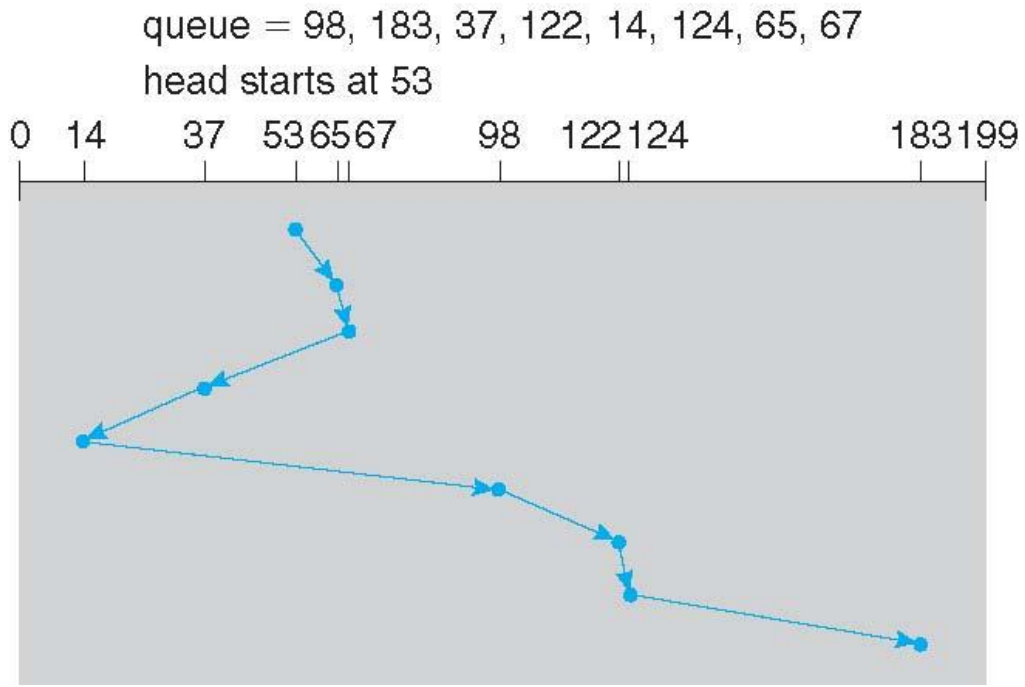
FCFS

Illustration shows total head movement of 640 cylinders



SSTF

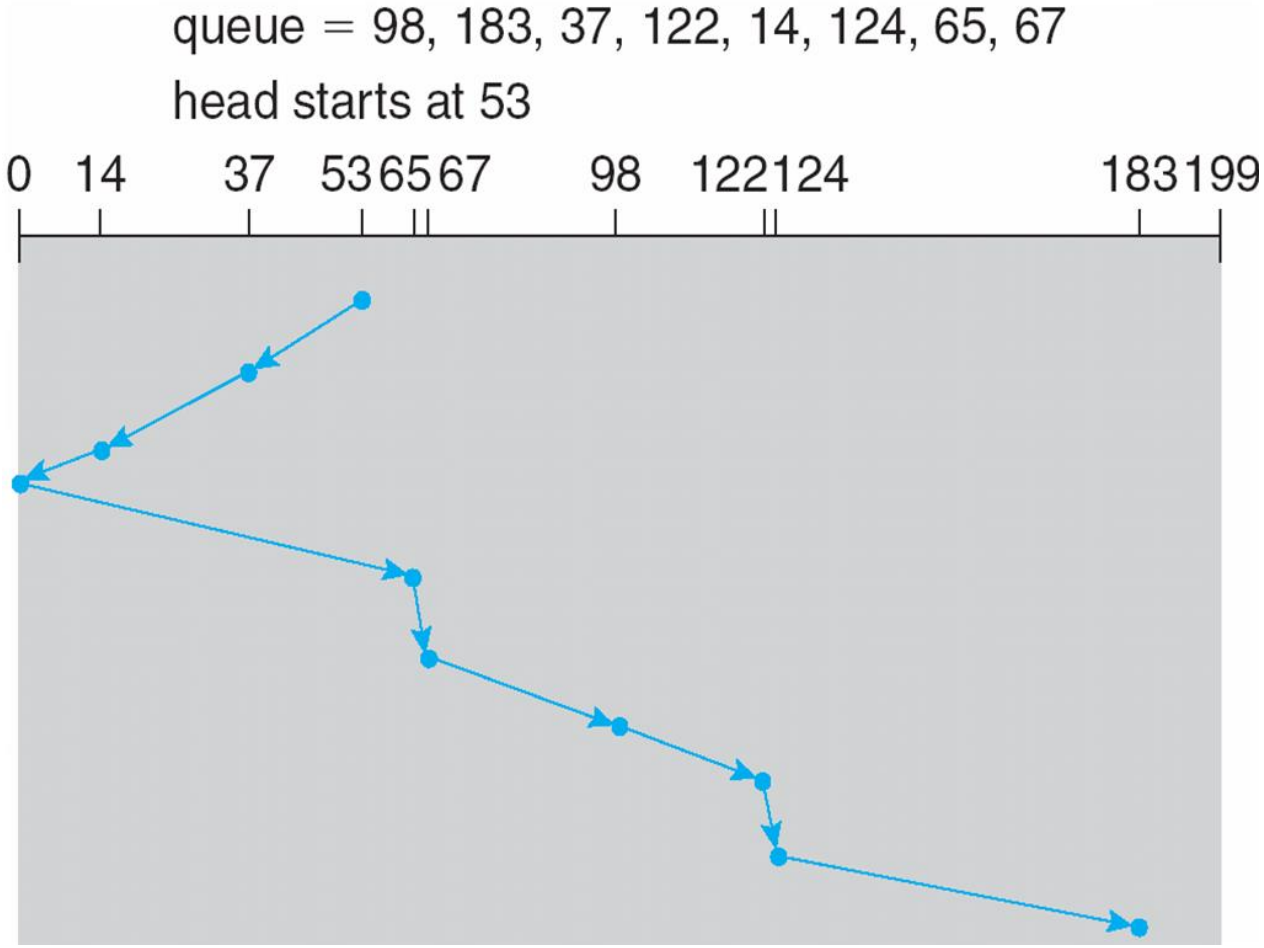
- Shortest Seek Time First selects the request with the minimum seek time from the current head position
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests
- Illustration shows total head movement of 236 cylinders



SCAN

- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.
- **SCAN algorithm** Sometimes called the **elevator algorithm**
- Illustration shows total head movement of 236 cylinders
- But note that if requests are uniformly dense, largest density at other end of disk and those wait the longest

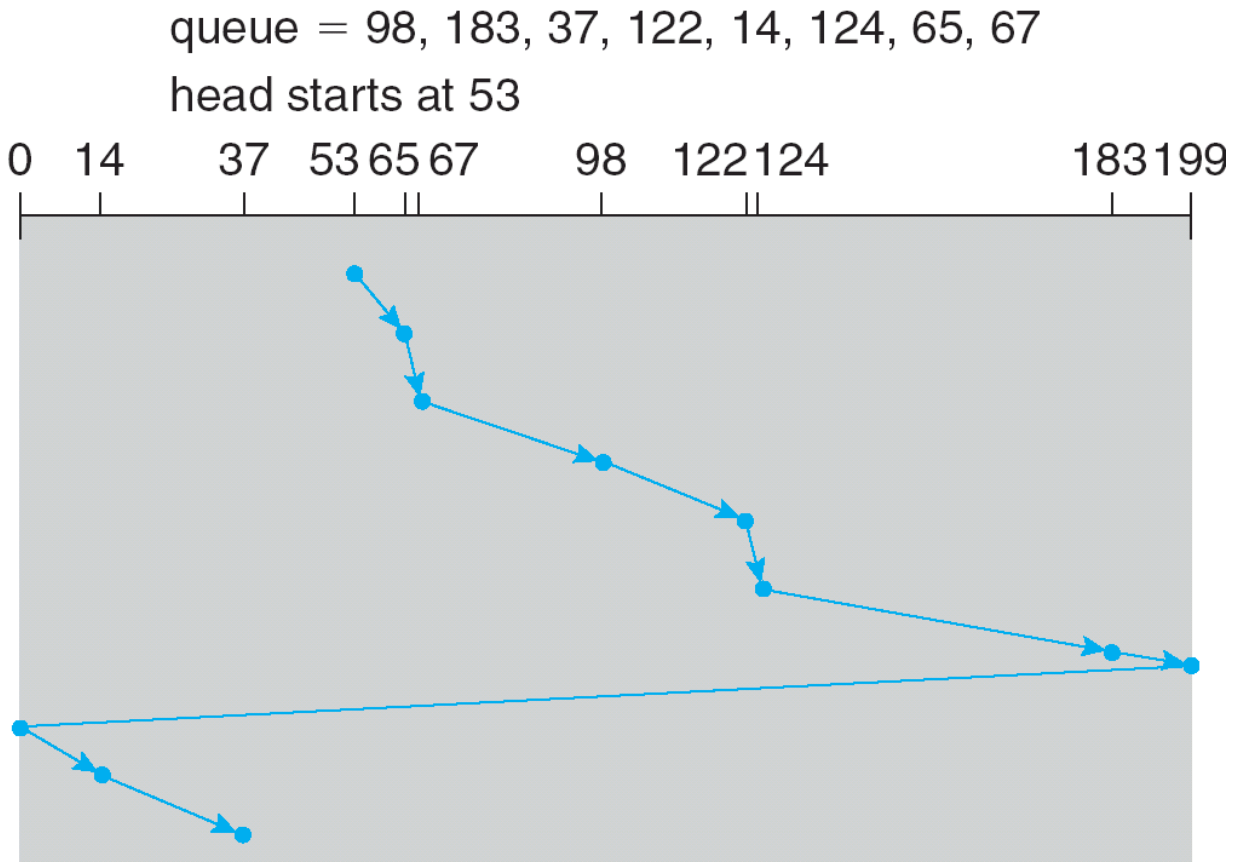
SCAN (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

C-SCAN

- Provides a more uniform wait time than SCAN
- The head moves from one end of the disk to the other, servicing requests as it goes
  - When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one
- Total number of cylinders?

C-SCAN (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

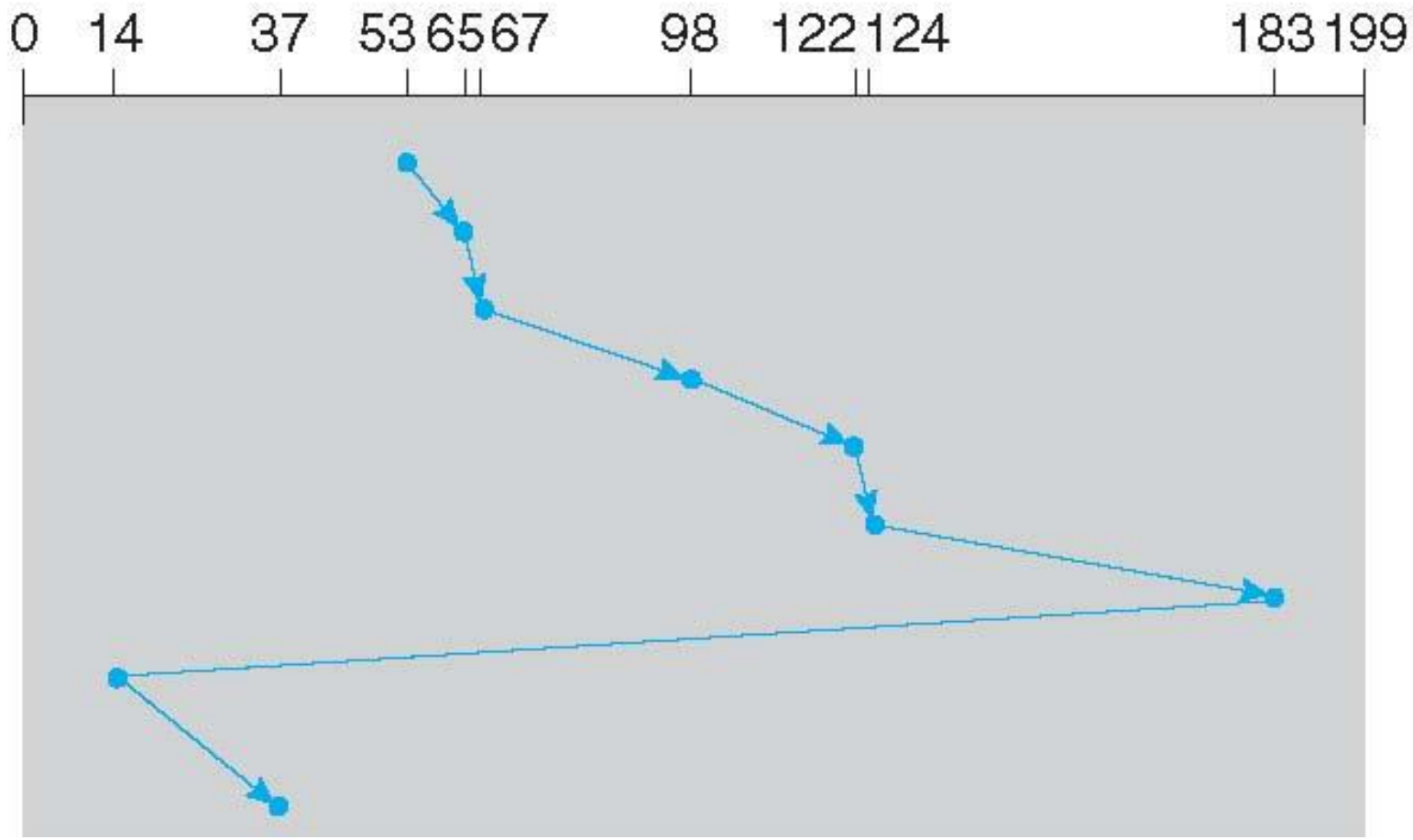
C-LOOK

- LOOK a version of SCAN, C-LOOK a version of C-SCAN
- Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk
- Total number of cylinders?

C-LOOK (Cont.)

queue = 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53



Selecting a Disk-Scheduling Algorithm

- SSTF is common and has a natural appeal
- SCAN and C-SCAN perform better for systems that place a heavy load on the disk
  - Less starvation
- Performance depends on the number and types of requests
- Requests for disk service can be influenced by the file-allocation method
  - And metadata layout
- The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary
- Either SSTF or LOOK is a reasonable choice for the default algorithm
- What about rotational latency?
  - Difficult for OS to calculate
- How does disk-based queueing effect OS queue ordering efforts?

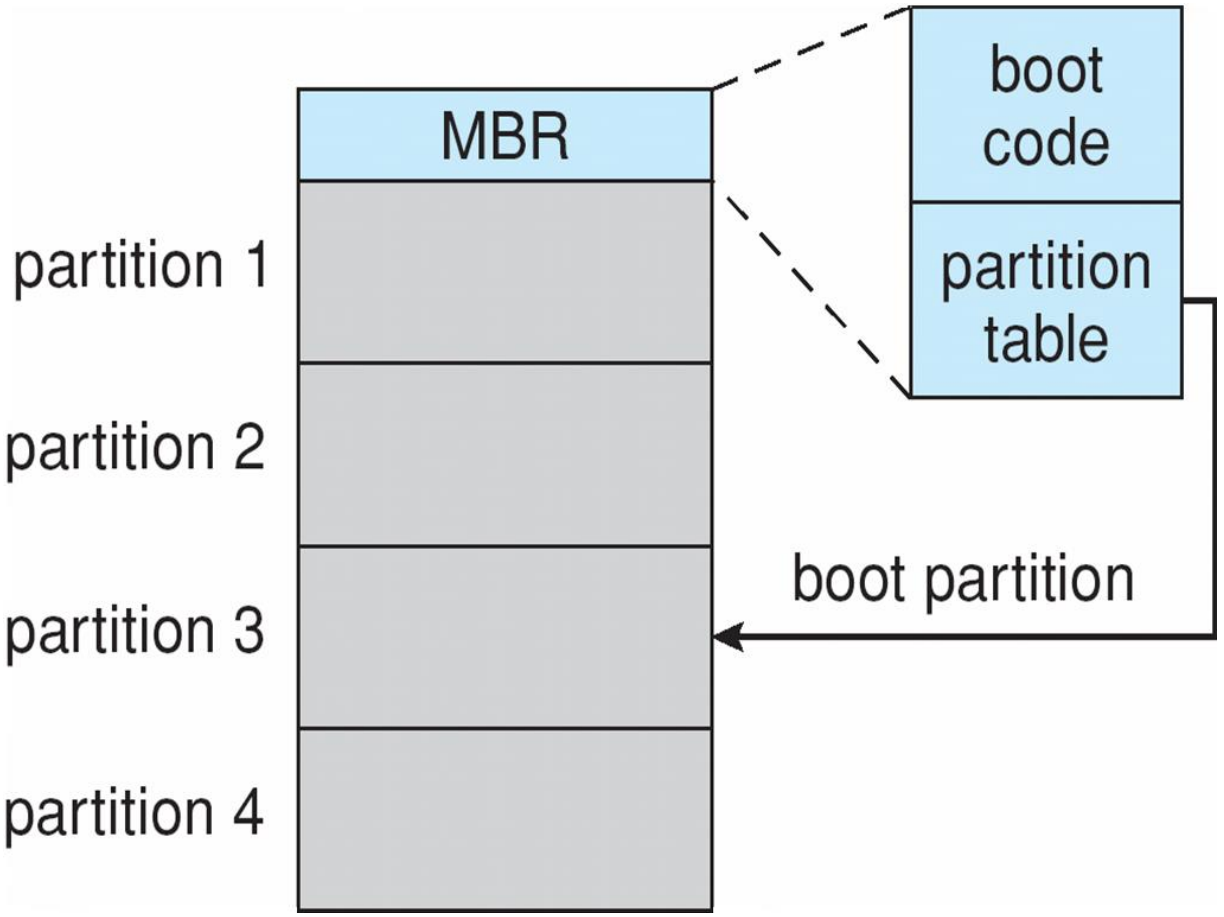
Disk Management

- **Low-level formatting**, or **physical formatting** — Dividing a disk into sectors that the disk controller can read and write
  - Each sector can hold header information, plus data, plus error correction code (**ECC**)
  - Usually 512 bytes of data but can be selectable
- To use a disk to hold files, the operating system still needs to record its own data structures on the disk
  - **Partition** the disk into one or more groups of cylinders, each treated as a logical disk
  - **Logical formatting** or "making a file system"
  - To increase efficiency most file systems group blocks into **clusters**
    - Disk I/O done in blocks
    - File I/O done in clusters

Disk Management (Cont.)

- Raw disk access for apps that want to do their own block management, keep OS out of the way (databases for example)
- Boot block initializes system
  - The bootstrap is stored in ROM
  - **Bootstrap loader** program stored in boot blocks of boot partition
- Methods such as **sector sparing** used to handle bad blocks

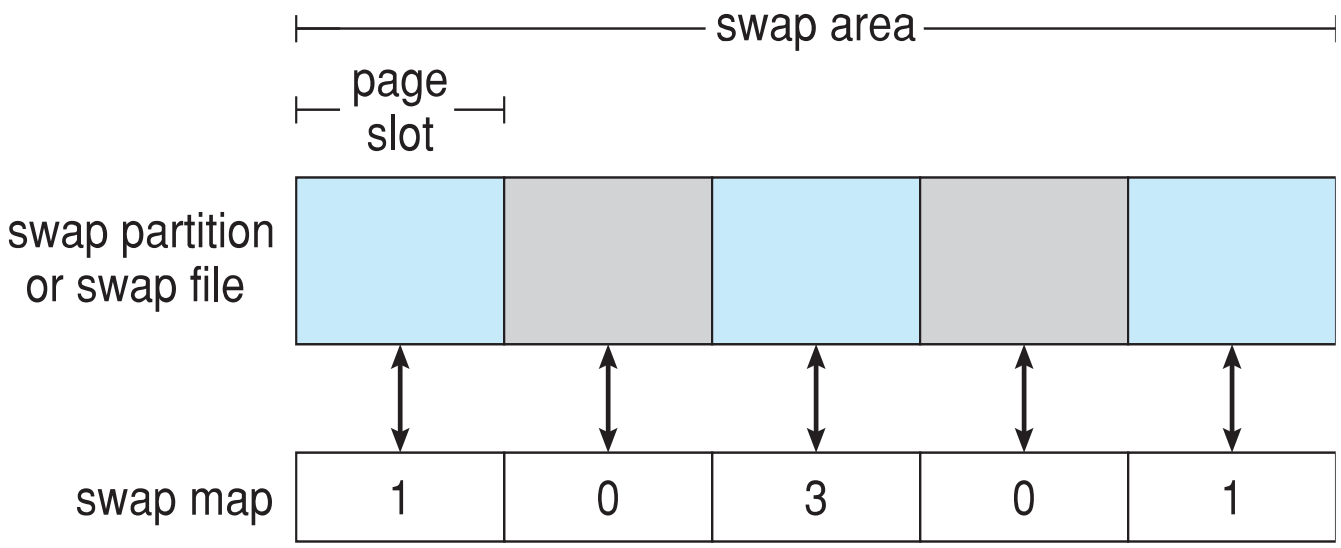
Booting from a Disk in Windows



Swap-Space Management

- Swap-space — Virtual memory uses disk space as an extension of main memory
  - Less common now due to memory capacity increases
- Swap-space can be carved out of the normal file system, or, more commonly, it can be in a separate disk partition (raw)
- Swap-space management
  - 4.3BSD allocates swap space when process starts; holds text segment (the program) and data segment
  - Kernel uses **swap maps** to track swap-space use
  - Solaris 2 allocates swap space only when a dirty page is forced out of physical memory, not when the virtual memory page is first created
    - File data written to swap space until write to file system requested
    - Other dirty pages go to swap space due to no other home
    - Text segment pages thrown out and reread from the file system as needed
- What if a system runs out of swap space?
- Some systems allow multiple swap spaces

Data Structures for Swapping on Linux Systems



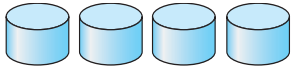
RAID Structure

- RAID – redundant array of inexpensive disks
  - multiple disk drives provides reliability via **redundancy**
- Increases the **mean time to failure**
- **Mean time to repair –** exposure time when another failure could cause data loss
- **Mean time to data loss** based on above factors
- If mirrored disks fail independently, consider disk with 1300,000 mean time to failure and 10 hour mean time to repair
  - Mean time to data loss is $100,000^2 / (2 * 10) = 500 * 10^6$ hours, or 57,000 years!
- Frequently combined with **NVRAM** to improve write performance
- Several improvements in disk-use techniques involve the use of multiple disks working cooperatively

RAID (Cont.)

- Disk **striping** uses a group of disks as one storage unit
- RAID is arranged into six different levels
- RAID schemes improve performance and improve the reliability of the storage system by storing redundant data
  - **Mirroring** or **shadowing** (**RAID 1**) keeps duplicate of each disk
  - Striped mirrors (**RAID 1+0**) or mirrored stripes (**RAID 0+1**) provides high performance and high reliability
  - **Block interleaved parity** (**RAID 4, 5, 6**) uses much less redundancy
- RAID within a storage array can still fail if the array fails, so automatic **replication** of the data between arrays is common
- Frequently, a small number of **hot-spare** disks are left unallocated, automatically replacing a failed disk and having data rebuilt onto them

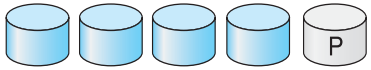
RAID Levels



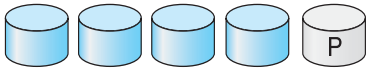
(a) RAID 0: non-redundant striping.

(b) RAID 1: mirrored disks.

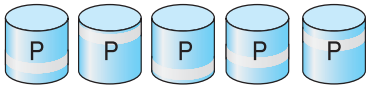
(c) RAID 2: memory-style error-correcting codes.



(d) RAID 3: bit-interleaved parity.



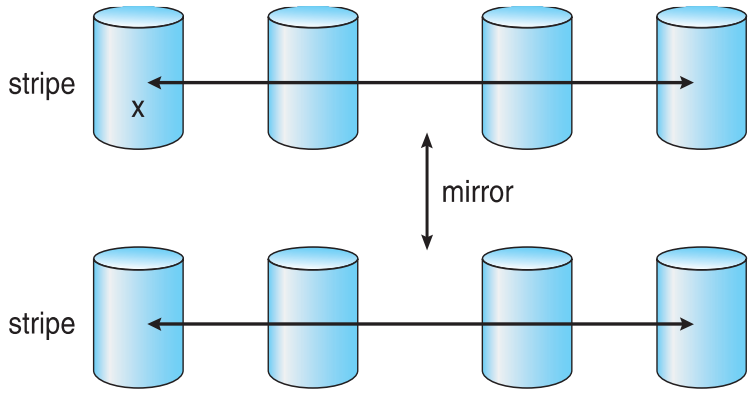
(e) RAID 4: block-interleaved parity.



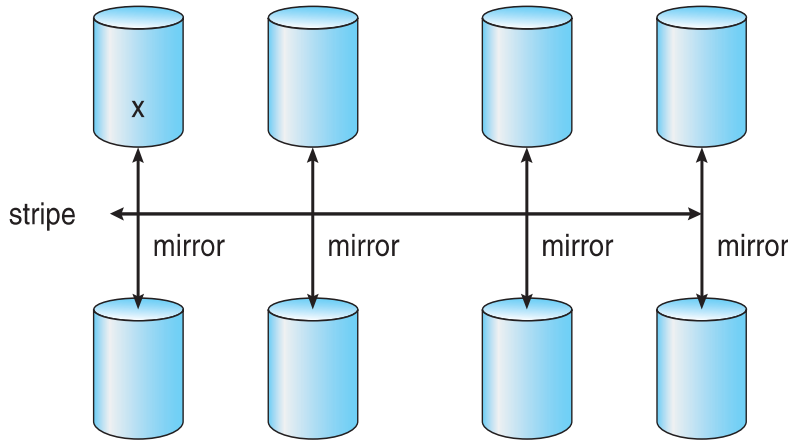
(f) RAID 5: block-interleaved distributed parity.

(g) RAID 6: P + Q redundancy.

RAID (0 + 1) and (1 + 0)



a) RAID 0 + 1 with a single disk failure.



b) RAID 1 + 0 with a single disk failure.

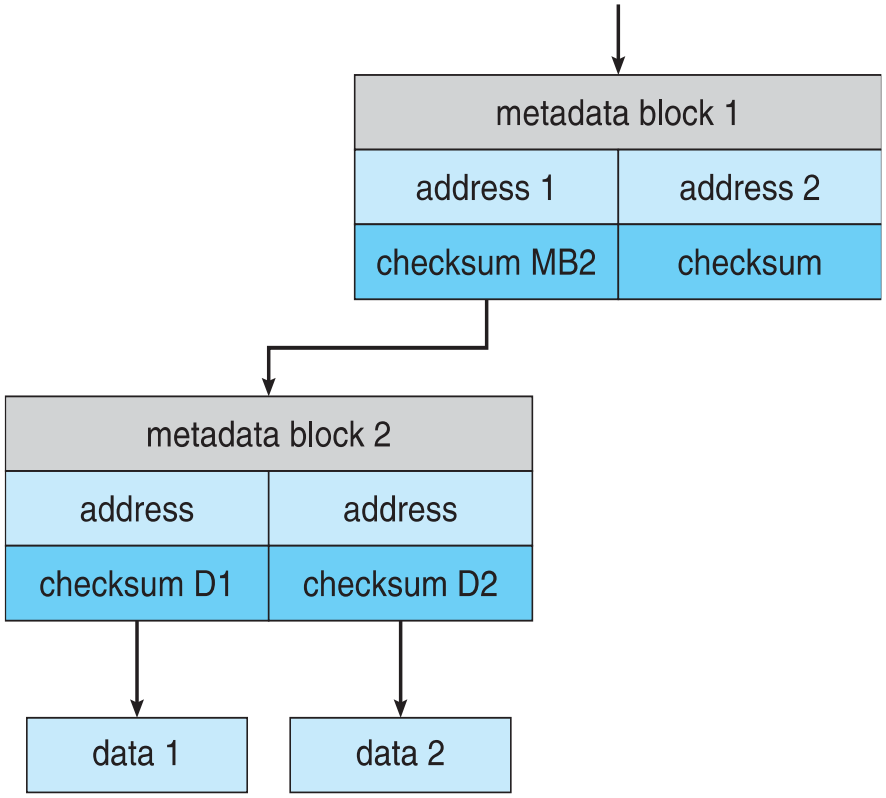
Other Features

- Regardless of where RAID implemented, other useful features can be added
- **Snapshot** is a view of file system before a set of changes take place (i.e. at a point in time)
  - More in Ch 12
- Replication is automatic duplication of writes between separate sites
  - For redundancy and disaster recovery
  - Can be synchronous or asynchronous
- Hot spare disk is unused, automatically used by RAID production if a disk fails to replace the failed disk and rebuild the RAID set if possible
  - Decreases mean time to repair

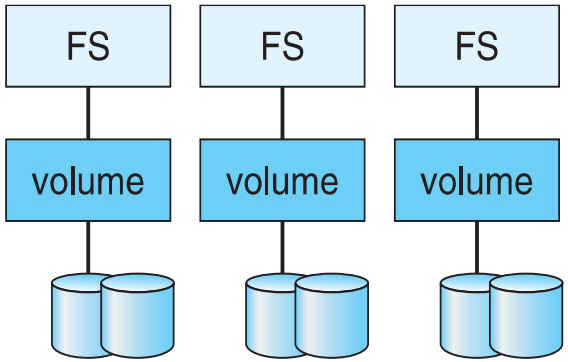
Extensions

- RAID alone does not prevent or detect data corruption or other errors, just disk failures
- Solaris ZFS adds **checksums** of all data and metadata
- Checksums kept with pointer to object, to detect if object is the right one and whether it changed
- Can detect and correct data and metadata corruption
- ZFS also removes volumes, partitions
  - Disks allocated in **pools**
  - Filesystems with a pool share that pool, use and release space like **`malloc()`** and **`free()`** memory allocate / release calls

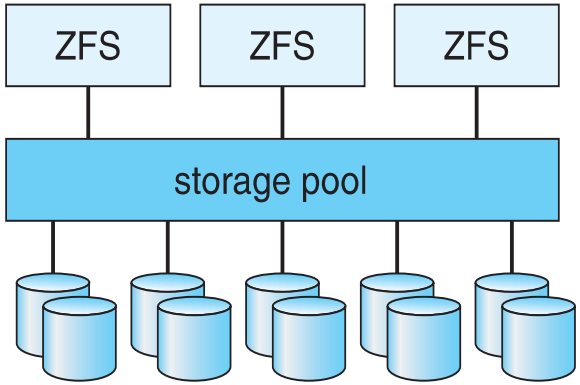
ZFS Checksums All Metadata and Data



Traditional and Pooled Storage



(a) Traditional volumes and file systems.



(b) ZFS and pooled storage.

Stable-Storage Implementation

- Write-ahead log scheme requires stable storage
- Stable storage means data is never lost (due to failure, etc)
- To implement stable storage:
  - Replicate information on more than one nonvolatile storage media with independent failure modes
  - Update information in a controlled manner to ensure that we can recover the stable data after any failure during data transfer or recovery
- Disk write has 1 of 3 outcomes
  1. **Successful completion -** The data were written correctly on disk
  2. **Partial failure -** A failure occurred in the midst of transfer, so only some of the sectors were written with the new data, and the sector being written during the failure may have been corrupted
  3. **Total failure -** The failure occurred before the disk write started, so the previous data values on the disk remain intact

Stable-Storage Implementation (Cont.)

- If failure occurs during block write, recovery procedure restores block to consistent state
  - System maintains 2 physical blocks per logical block and does the following:
    1. Write to 1st physical
    2. When successful, write to 2nd physical
    3. Declare complete only after second write completes successfully

  Systems frequently use NVRAM as one physical to accelerate

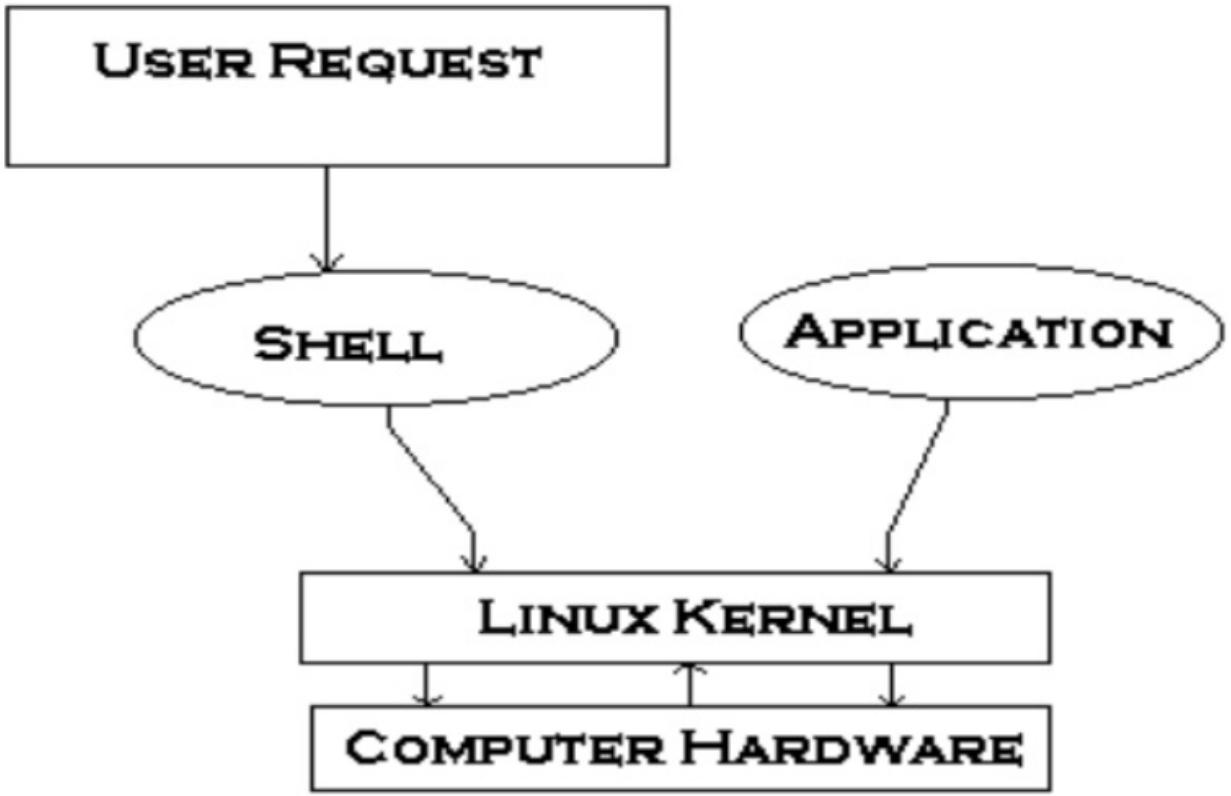
SHELL OPERATION COMMANDS

What Is Kernal ?

- Kernel is hart of Linux Os.
- It manages resource of Linux Os. Resources means facilities available in Linux. For e.g. Facility to store data, print data on printer, memory, file management etc .
- Kernel decides who will use this resource, for how long and when. It runs your programs (or set up to execute binary files).
- The kernel acts as an intermediary between the computer hardware and various programs/application/shell.

KERNAL

Kernal con.



KERNAL

- It's Memory resident portion of Linux. It performance following task :-
- I/O management
- Process management
- Device management
- File management
- Memory management

What Is a Shell?

- A *shell is a program that takes commands typed by the user and calls the operating system to run those commands.*
- A *shell is a program that acts as the interface between you and the Linux system, allowing you to enter* commands for the operating system to execute.
- Shell accepts your instruction or commands in English and translate it into computers native binary language

Why Use Shells?

- You can use shell scripts to automate administrative tasks.
- Encapsulate complex configuration details.
- Get at the full power of the operating system.
- The ability to combine commands allows you to create new commands
- Adding value to your operating system.

Kind of Shells

- *Bourne Shell*
- *C Shell*
- *Korn Shell*
- *Bash Shell*
- *Tcsh Shell*

Changing Your Default Shell

- Tip: To find all available shells in your system type following command:

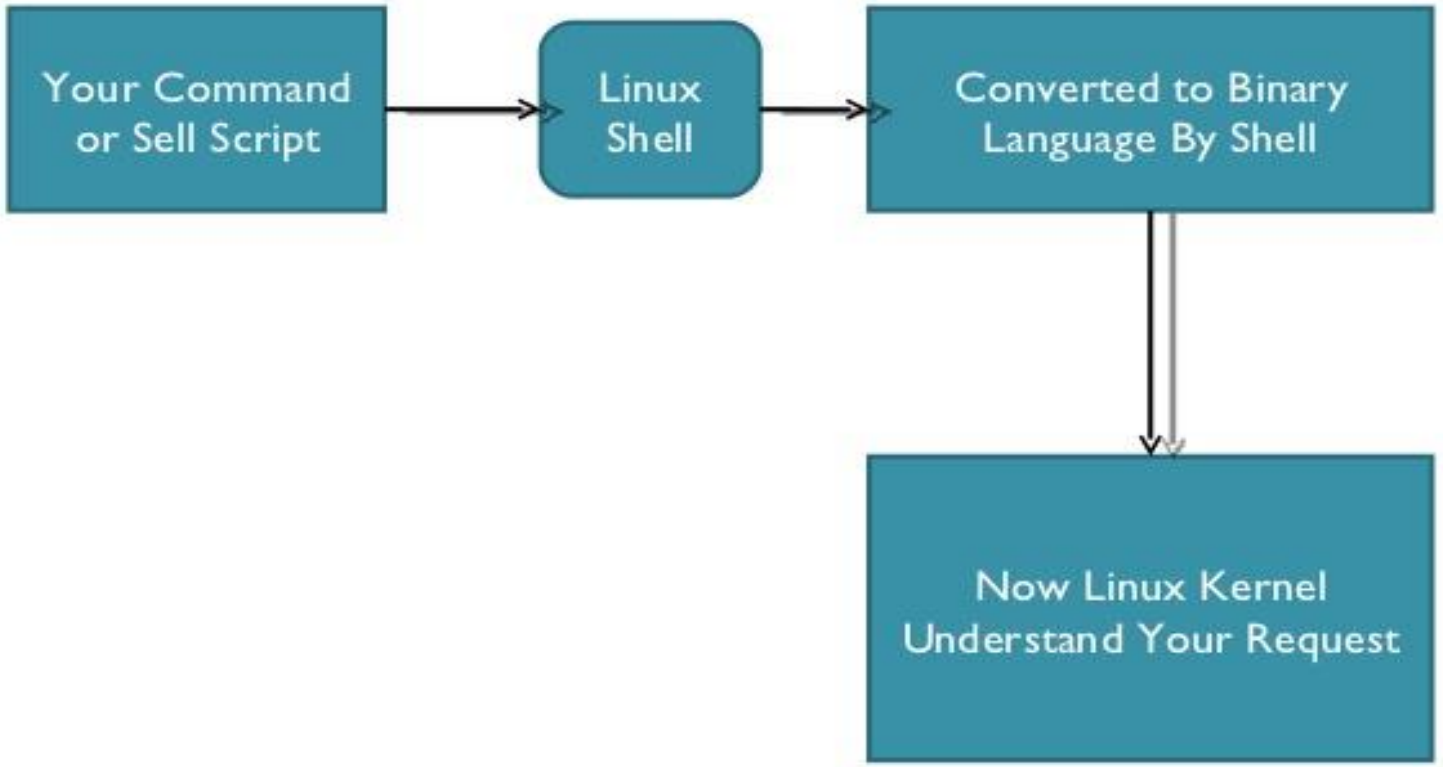
  $ cat /etc/shells

- The basic Syntax :

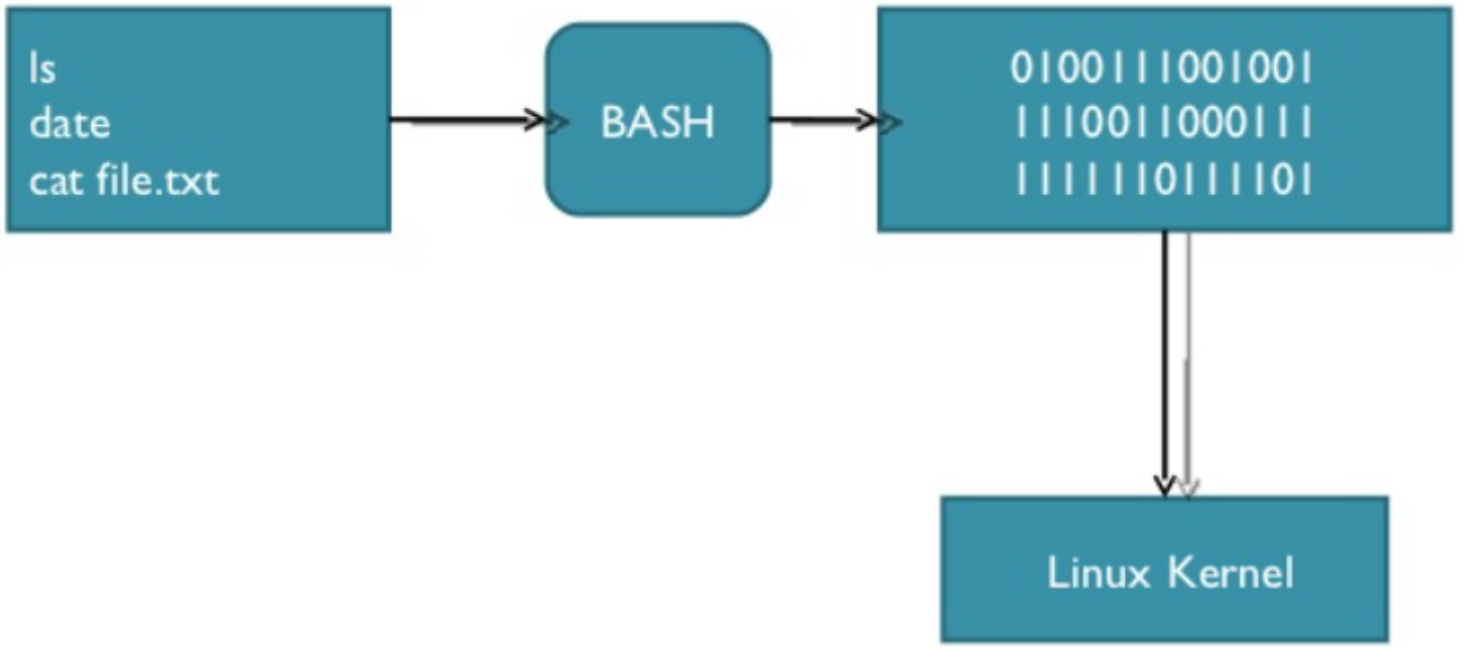
chsh *username new_default_shell*

- The administrator can change your default shell.

This is what Shell Does for US



Example



Shell is an command language interpreter that executes commands read from the standard input device (keyboard) or from a file.

The Shell as a Programming Language

- Now that we've seen some basic shell operations, it's time to move on to scripts.
- There are two ways of writing shell programs.
  1. You can type a sequence of commands and allow the shell to execute them interactively.
  2. You can store those commands in a file that you can then invoke as a program(shell script).

Shell Scripting

- *Shell script is a series of command(s) stored in a plain text file.*
- *A shell script is similar to a batch file in MS-DOS, but is much more powerful.*

Why to Write Shell Script ?

- Shell script can take input from user, file and output them on screen.
- Useful to create our own commands. Save lots of time.
- To automate some task of day today life.
- System Administration part can be also automated.

Practical examples where shell scripting actively used:

1. Monitoring your Linux system.
2. Data backup and creating snapshots.
3. Find out what processes are eating up your system resources.
4. Find out available and free memory.
5. Find out all logged in users and what they are doing.
6. Find out if all necessary network services are running or not.

Create a script

- As discussed earlier shell scripts stored in plain text file, generally one command per line.
  - vi myscript.sh
- Make sure you use .bash or .sh file extension for each script. This ensures easy identification of shell script.

Setup executable permission

- Once script is created, you need to setup executable permission on a script. Why?
  - Without executable permission, running a script is almost impossible.
  - Besides executable permission, script must have a read permission.
- Syntax to setup executable permission:
  - $ chmod +x your-script-name.
  - $ chmod 755 your-script-name.

Run a script (execute a script)

- Now your script is ready with proper executable permission on it. Next, test script by running it.
  - bash your-script-name
  - sh your-script-name
  - ./your-script-name
- Examples
  - $ bash bar
  - $ sh bar
  - $ ./bar

Example

```
$ vi first
```

```
#
# My first shell script
#
clear
echo "This is my First
script"
```

```
$ chmod 755 first


$ ./first
```

Variables in Shell

- In Linux (Shell), there are two types of variable:
  - **System variables** - Created and maintained by Linux itself. This type of variable defined in CAPITAL LETTERS.
  - **User defined variables (UDV)** - Created and maintained by user. This type of variable defined in lower letters.

User defined variables (UDV)

- To define UDV use following syntax:
  - variable name=value
  - $ no=10
- **Rules for Naming variable name**
  - Variable name must begin with Alphanumeric character or underscore character (_), followed by one or more Alphanumeric character.
  - Don't put spaces on either side of the equal sign when assigning value to variable.
  - Variables are case-sensitive.
  - You can define NULL variable
  - Do not use **?,*** etc, to name your variable names.

Print or access value of UDV

- To print or access UDV use following syntax :
  - $variablename.
- Examples:
  - $vech=Bus
  - $ n=10
  - $ echo $vech
  - $ echo $n

Cont…

- Don't try
  - **$ echo vech**
  - it will print vech instead its value 'Bus'.
  - **$ echo n**
  - it will print n instead its value '10'.
- You must *use $ followed by variable name.*

Shell Arithmetic

- *Syntax:*
  - expr op1 math-operator op2

- *Examples:*
  - $ expr 1 + 3
  - $ expr 2 – 1
  - $ expr 10 / 2
  - $ expr 20 % 3
  - $ expr 10 * 3
  - $ echo `expr 6 + 3`

The read Statement

- Use to get input (data from user) from keyboard and store (data) to variable.
- *Syntax:*
  - read variable1, variable2,...variableN

```
echo "Your first name please:"
read fname

echo "Hello $fname, Lets be friend!"
```

Shorthand

Shorthand	Meaning
$ ls *	will show all files
$ ls a*	will show all files whose first name is starting with letter 'a'
$ ls *.c	will show all files having extension .c
$ ls ut*.c	will show all files having extension .c but file name must begin with 'ut'.
$ ls ?	will show all files whose names are 1 character long
$ ls fo?	will show all files whose names are 3 character long and file name begin with fo
$ ls [abc]*	will show all files beginning with letters a,b,c

if condition

Syntax:

if condition

then

command1 if condition is true or if exit status of condition is 0 (zero)

fi

Math- ematical Operator in Shell Script	Meaning	Normal Arithmetical/ Mathematical Statements
-eq	is equal to	5 == 6
-ne	is not equal to	5 != 6
-lt	is less than	5 < 6
-le	is less than or equal to	5 <= 6
-gt	is greater than	5 > 6
-ge	is greater than or equal to	5 >= 6

Example

- $ vim myscript.sh

```
read choice

if [ $choice -gt 0 ]; then
echo "$choice number is positive"
else
echo "$ choice number is negative"
fi
```

- **$#** Stores the number of **command**-line arguments that were passed to the **shell** program.
- **$?** Stores the exit value of the last **command** that was executed.

Loops in Shell Scripts

- Bash supports:

2. for loop.
3. while loop.

- **Note** that in each and every loop:
  - First, the variable used in loop condition must be initialized, then execution of the loop begins.
  - A test (condition) is made at the beginning of each iteration.
  - The body of loop ends with a statement that modifies the value of the test (condition) variable.

for Loop

Syntax:

Syntax:

Syntax:

for { variable name } in { list }

do

execute one for each item in the list until the list is not finished and repeat all statement between do and done

done

for Loop

- *Syntax:*

```
for (( expr1; expr2; expr3 ))
  do
    repeat all statements between
    do and done until expr2 is TRUE
Done
```

Example

```
for ((  i = 0 ;  i <= 5;  i++  ))
do
  echo "Welcome $i times"
done
```

Nesting of for Loop

- $ vi nestedfor.sh

```
for (( i = 1; i <= 5; i++ ))
do

    for (( j = 1 ; j <= 5; j++ ))
    do
        echo -n "$i "
    done

  echo “”
don
```

while loop

- *Syntax:*

while [condition]

do

command1 command2 command3 .. ....

done

Example

```
i=1
while [ $i -le 10 ]
 do
   echo "$n * $i = `expr $i \* $n`"
   i=`expr $i + 1`
 done
```

The case Statement

- *Syntax:*

```
case $variable-name in
  pattern1) command…..;;
  pattern2) command…..;;
  pattern N) command….;;
  .
 *) command  ;;
esac
```

Example

```
read var
 case $var in
  1) echo “One”;;
  2) echo “Two”;;
  3) echo “Three”;;
  4) echo “Four”;;
  *) echo "Sorry, it is bigger than Four";;
esac
```

Example

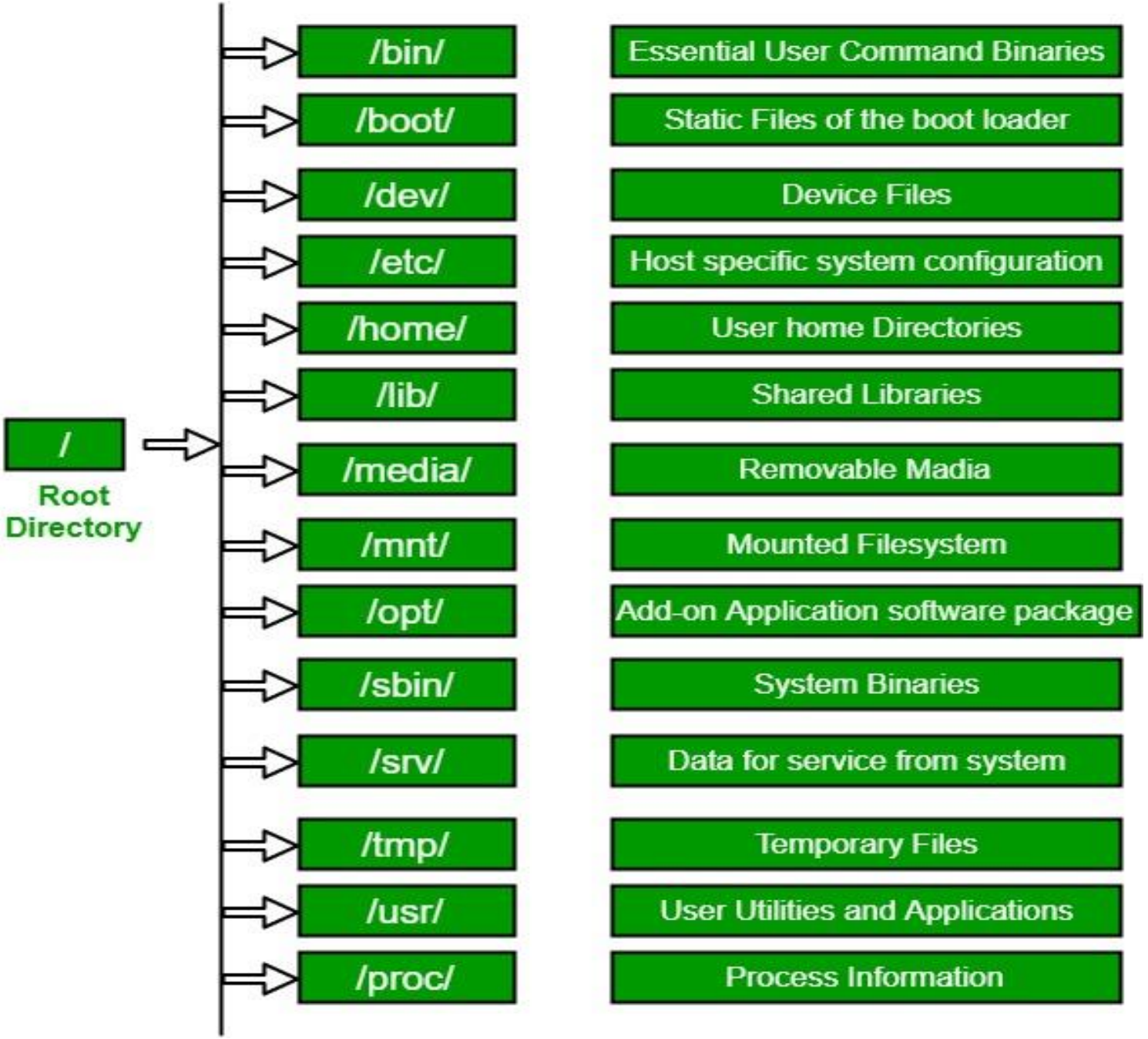
```
while :
do
    clear
    echo "--------------------------------"
    echo " Main Menu "
    echo "--------------------------------"
    echo "[1] Show Todays date/time"
    echo "[2] Show files in current directory“
    echo "[3] Show calendar"
    echo "[4] Start editor to write letters"
    echo "[5] Exit/Stop"
    echo "========================"
    echo -n "Enter your menu choice [1-5]: "
    read yourch
```

Cont...

```
case $yourch in
    1) echo "Today is `date` , press a key. . ." ; read ;;
    2) echo "Files in `pwd`" ; ls -l ; echo "Press a key. . ." ; read ;;
    3) cal ; echo "Press a key. . ." ; read ;;
    4) vi ;;
    5) exit 0 ;;
    *) echo "Opps!!! Please select choice 1,2,3,4, or 5";
    echo "Press a key. . ." ; read ;;
esca
done
```

LINUX FILE STRUCTURE

- The Linux File Hierarchy Structure or the Filesystem Hierarchy Standard (FHS) defines the directory structure and directory contents in Unix-like operating systems.
- It is maintained by the Linux Foundation.
- In the FHS, all files and directories appear under the root directory /, even if they are stored on different physical or virtual devices.
- Some of these directories only exist on a particular system if certain subsystems, such as the X Window System, are installed.
- Most of these directories exist in all UNIX operating systems and are generally used in much the same way; however, the descriptions here are those used specifically for the FHS, and are not considered authoritative for platforms other than Linux.


/
Root Directory
/bin/
Essential User Command Binaries
/boot/
Static Files of the boot loader
/dev/
Device Files
/etc/
Host specific system configuration
/home/
User home Directories
/lib/
Shared Libraries
/media/
Removable Madia
/mnt/
Mounted Filesystem
/opt/
Add-on Application software package
/sbin/
System Binaries
/srv/
Data for service from system
/tmp/
Temporary Files
/usr/
User Utilities and Applications
/proc/
Process Information

Sr.No.	Directory & Description
1	**/** This is the root directory which should contain only the directories needed at the top level of the file structure
2	**/bin** This is where the executable files are located. These files are available to all users
3	**/dev** These are device drivers
4	**/etc** Supervisor directory commands, configuration files, disk configuration files, valid user lists, groups, ethernet, hosts, where to send critical messages

5	**/lib** Contains shared library files and sometimes other kernel-related files
6	**/boot** Contains files for booting the system
7	**/home** Contains the home directory for users and other accounts
8	**/mnt** Used to mount other temporary file systems, such as **cdrom** and **floppy** for the **CD-ROM** drive and **floppy diskette drive**, respectively
9	**/proc** Contains all processes marked as a file by **process number** or other information that is dynamic to the system

10	**/tmp** Holds temporary files used between system boots
11	**/usr** Used for miscellaneous purposes, and can be used by many users. Includes administrative commands, shared files, library files, and others
12	**/var** Typically contains variable-length files such as log and print files and any other type of file that may contain a variable amount of data
13	**/sbin** Contains binary (executable) files, usually for system administration. For example, ***fdisk*** and ***ifconfig*** utlities
14	**/kernel** Contains kernel files

File System in Linux

- Linux supports many different file systems, but common choices for the system disk include the ext family (such as ext2 and ext3), XFS, JFS and ReiserFS.
- The ext3 or third extended file system is a journaled file system and is the default file system for many popular Linux distributions .
- It is an upgrade of its predecessor ext2 file system and among other things it has added the journouling feature.
- A journaling file system is a file system that logs changes to a journal (usually a circular log in a dedicated area) before committing them to the main file system. Such file systems are less likely to become corrupted in the event of power failure or system crash.

Directory Structure

Unix uses a hierarchical file system structure, much like an upside-down tree, with root (/) at the base of the file system and all other directories spreading from there.

A Unix filesystem is a collection of files and directories that has the following properties –

- It has a root directory (/) that contains other files and directories.
- Each file or directory is uniquely identified by its name, the directory in which it resides, and a unique identifier, typically called an **inode**.
- By convention, the root directory has an **inode** number of **2** and the **lost+found** directory has an **inode** number of **3**. Inode numbers **0** and **1** are not used. File inode numbers can be seen by specifying the **-i option** to **ls command**.
- It is self-contained. There are no dependencies between one filesystem and another.