SESSION 5 – LINUX DISKS AND FILE SYSTMS

Course Writer: Dr. Ed. Danso Ansong, Dept of Computer Sc.
Contact Information: edansong@ug.edu.gh

UNIVERSITY OF GHANA
College of Education
School of Continuing and Distance Education

- The session is expected to expose students on the need to recognize basic disk components and to understand basic disk geometry as well as the file systems.

The key topics to be covered in the session are as follows:

- Partitioning
- Formatting
- Disk systems
- Windows and Unix file system structure

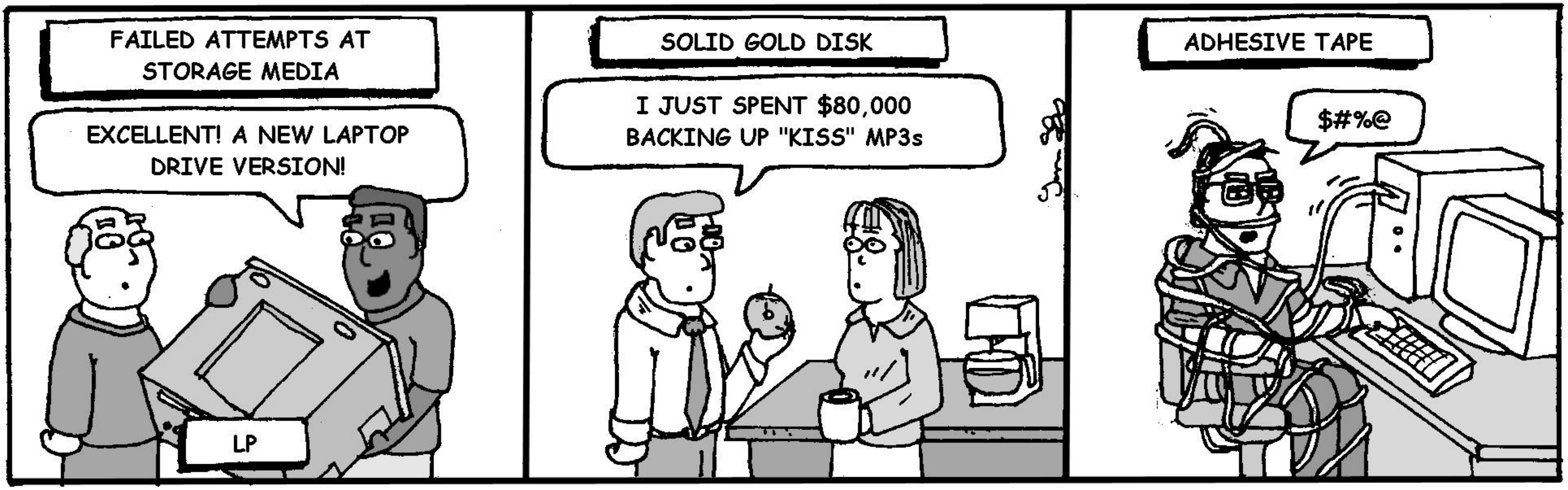
- Refer to the following reading material which is available on Sakai

RECOMMENDED TEXT

- Linux Administration Handbook, Second Edition [Pages 131-183].

At the end of the session, the student will be able to:

- Recognize basic disk components
- Understand basic disk geometry
- Understand how to create and manage file systems
- Recognize advantages and disadvantages of SCSI, IDE, and alternative disk systems
- Recognize similarities and differences in Windows and Unix file system structures

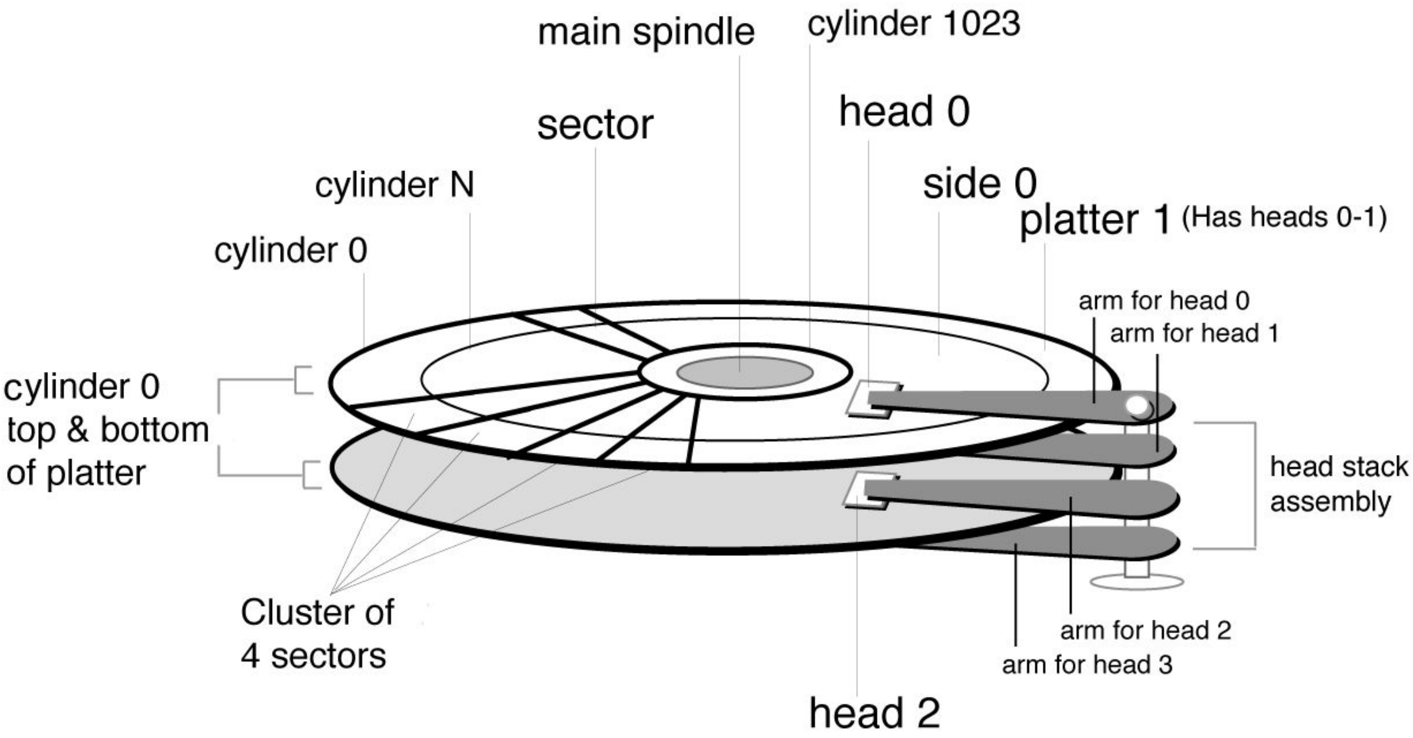

FAILED ATTEMPTS AT STORAGE MEDIA
EXCELLENT! A NEW LAPTOP DRIVE VERSION!
LP
SOLID GOLD DISK
I JUST SPENT $80,000 BACKING UP "KISS" MP3s
ADHESIVE TAPE
$#%@

Disk Subsystem

- Disk Drives (General)
  - Disk drives are one of the most important parts of a system.
  - Because they have moving parts, **disk drives break down**!
    - Some disk failures are survivable (soft errors, random bad blocks).
    - Other disk failures are catastrophic (head crash, electronics failures).
    - The System Administrator has to know how to determine what is wrong, and how to fix it.
  - Disks play a big part in overall system performance.
    - Access time is orders of magnitude slower than memory.
    - File system layout can cause performance problems.

Disk Subsystem

- Disk Drive Components
  - Disk drives contain several key components:
    - Magnetic media
    - Read/Write heads
    - Assorted logic
      - read/write,
      - positioning,
      - memory,
      - status,
      - bus interface)
    - Motor
    - Positioner


main spindle
cylinder 1023
head 0
sector
side 0
cylinder N
platter 1 (Has heads 0-1)
cylinder 0
arm for head 0
arm for head 1
cylinder 0
top & bottom
of platter
head stack
assembly
Cluster of
4 sectors
arm for head 2
arm for head 3
head 2

Disk Subsystem

- Disk Drive Terminology
  - Disk drives have a language all their own:
    - cylinders
    - tracks (or heads, or surfaces)
    - sectors
    - seek
    - crash
    - servo positioner
    - bad blocks
    - geometry
    - partition
    - format

Disk Drives 101

- A platter is coated with magnetic particles (oxides).
- By changing the orientation of these magnetic particles we can encode zero’s and ones and store them on the platter.
- The platter is mounted on a spindle that is driven by a motor.
- A read write head “flys” above the surface of the platter.
  - This head contains a small electromagnetic device that can determine the orientation of the particles on the platter.
- By passing a current through this magnetic device, we can also magnetize the particles to alter their orientation.

Disk Drives 101

- Because the platter is spinning, and the head’s position is somewhat “fixed”, we end up with a ring of information (cylinders) on the platter.
  - By creating many rings on the platter we increase the storage density of the platter.
  - A positioner moves the read/write heads from ring to ring on the platter(s).
  - By stacking many platters (hence more read/write heads) on the same spindle we increase the density of the disk.
- Each ring (cylinder) is broken up into uniform chunks (sectors).
- Each sector can hold some amount of data.
- The raw capacity of the disk drive is determined by :

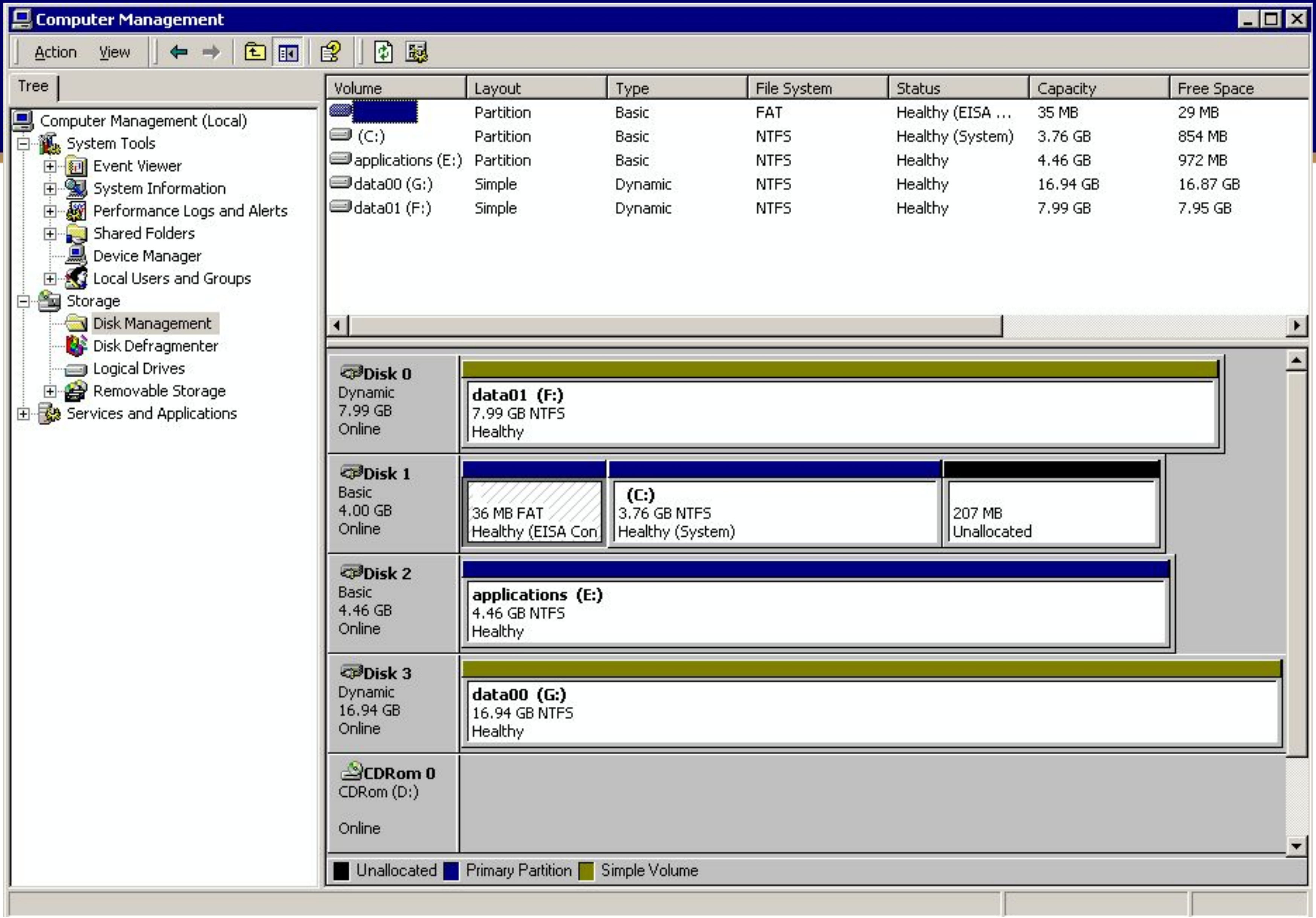
# cylinders X #heads/cylinder X #sectors/head X #bytes/sector

Disk Partitions

- Sometimes it is desirable to segment, or partition a physical disk into multiple logical disks.
  - Sometimes the OS buffering routines provide better performance if the disk is partitioned.
  - It is generally desirable to partition files such that system files, application files, and user files are on separate partitions.
  - A partition is a logical grouping of a number of physical cylinders into a logical disk drive.
  - Each operating system and hardware architecture has it's own method of partitioning a disk.

Disk Partitions

- Most operating systems for Intel x86 architectures use the fdisk program to partition the disk.
- Non-Intel architectures use another program to perform the same operation.
  - Once the disk has been partitioned, file systems are created on the logical disks.
  - On Windows boxes, this is done using the format program.
  - Non-Windows boxes typically use the mkfs, or newfs utilities to perform this operation.



Volume	Layout	Type	File System	Status	Capacity	Free Space
	Partition	Basic	FAT	Healthy (EISA ...	35 MB	29 MB
(C:)	Partition	Basic	NTFS	Healthy (System)	3.76 GB	854 MB
applications (E:)	Partition	Basic	NTFS	Healthy	4.46 GB	972 MB
data00 (G:)	Simple	Dynamic	NTFS	Healthy	16.94 GB	16.87 GB
data01 (F:)	Simple	Dynamic	NTFS	Healthy	7.99 GB	7.95 GB

Disk 0
Dynamic
7.99 GB
Online

data01 (F:)
7.99 GB NTFS
Healthy

Disk 1
Basic
4.00 GB
Online

36 MB FAT
Healthy (EISA Con

(C:)
3.76 GB NTFS
Healthy (System)

207 MB
Unallocated

Disk 2
Basic
4.46 GB
Online

applications (E:)
4.46 GB NTFS
Healthy

Disk 3
Dynamic
16.94 GB
Online

data00 (G:)
16.94 GB NTFS
Healthy

CDRom 0
CDRom (D:)

Online

■ Unallocated ■ Primary Partition ■ Simple Volume

Disk Subsystem

- Advanced Disk Geometry
  - O.K., so we have cylinders, heads, sectors, a positioner, and partitions. Now what? How does this relate to storing and retrieving the information we want?
  - Before we can store and retrieve data we have to come up with a way of uniquely identifying each cylinder/head/sector on the drive.
  - An Index mark on the spindle gives us a starting reference point.
  - The read/write surfaces are numbered 0 (zero) through H.
    - Head zero reads/writes data on platter 0.
    - Head 1 reads/writes data on platter 1, and so forth.
    - Cylinders are numbered 0 (zero) through C.
      - Some manufacturers use the innermost cylinder as 0.
      - Other manufacturers use the outermost cylinder as 0.

Disk Subsystem

- Starting on cylinder zero, sectors are numbered 0 (zero) through S.
  - Sector zero is the first sector after the index mark.
  - Sector zero is also the first sector on surface zero.

Any guess where sector “s” is located?

Disk Subsystem

- Disk Geometry
  - Sector S is the first sector on cylinder 0, surface 1.
  - **Why is sector s on the next surface, not the next cylinder??**
    - It is much faster to have electronics “turn off” one head, and enable the next head than it is to move the positioner to the next cylinder! (ns vs. ms).
  - Sector 2S is the first sector on cylinder 0,surface 2 and so on.
  - Sector HS is on cylinder 1, surface 0.
  - By continuing this numbering scheme, we can determine the “address” of each sector on the disk drive. As long as the hardware can keep up, we can count this way forever.

Disk Subsystem

- Disk Geometry
  - Unfortunately, early systems did not have the ability to count forever.
  - Early UNIX was designed to run on 16 bit hardware.
  - 16 bits allowed us to count to 65,536 sectors. At 512 bytes/sector, this only allowed us to access 32 Megabytes of data per disk drive.
  - By partitioning a large disk into several smaller logical disks, we had a way of addressing the entirety of the disk drive:
    - partition 0 was the first 32 Mb,
    - partition 1 was the second 32 Mb and so on.
  - Newer hardware is capable of addressing much larger disks, but partitioning is still in use.

Disk Interfaces

- At this point we have a basic understanding of the physical disk media.
- Now we need to examine how this media is connected to the system.
- There are several interfaces used between the system bus, and the disk drive.
  - Small Computer System Interconnect (SCSI)
  - Fibre Channel
  - Integrated Drive Electronics (IDE)
  - Firewire
  - USB
  - Parallel port

SCSI

- SCSI devices
  - SCSI has become one of the more popular bus adapters for PC, MacIntosh, and Workstation systems.
  - There are many flavors of SCSI (see table, next page).
  - You need to make sure that you don’t mix different types of SCSI devices together:
    - Performance might suffer!
    - Physical damage may result!
    - Errors will be difficult to track down

SCSI

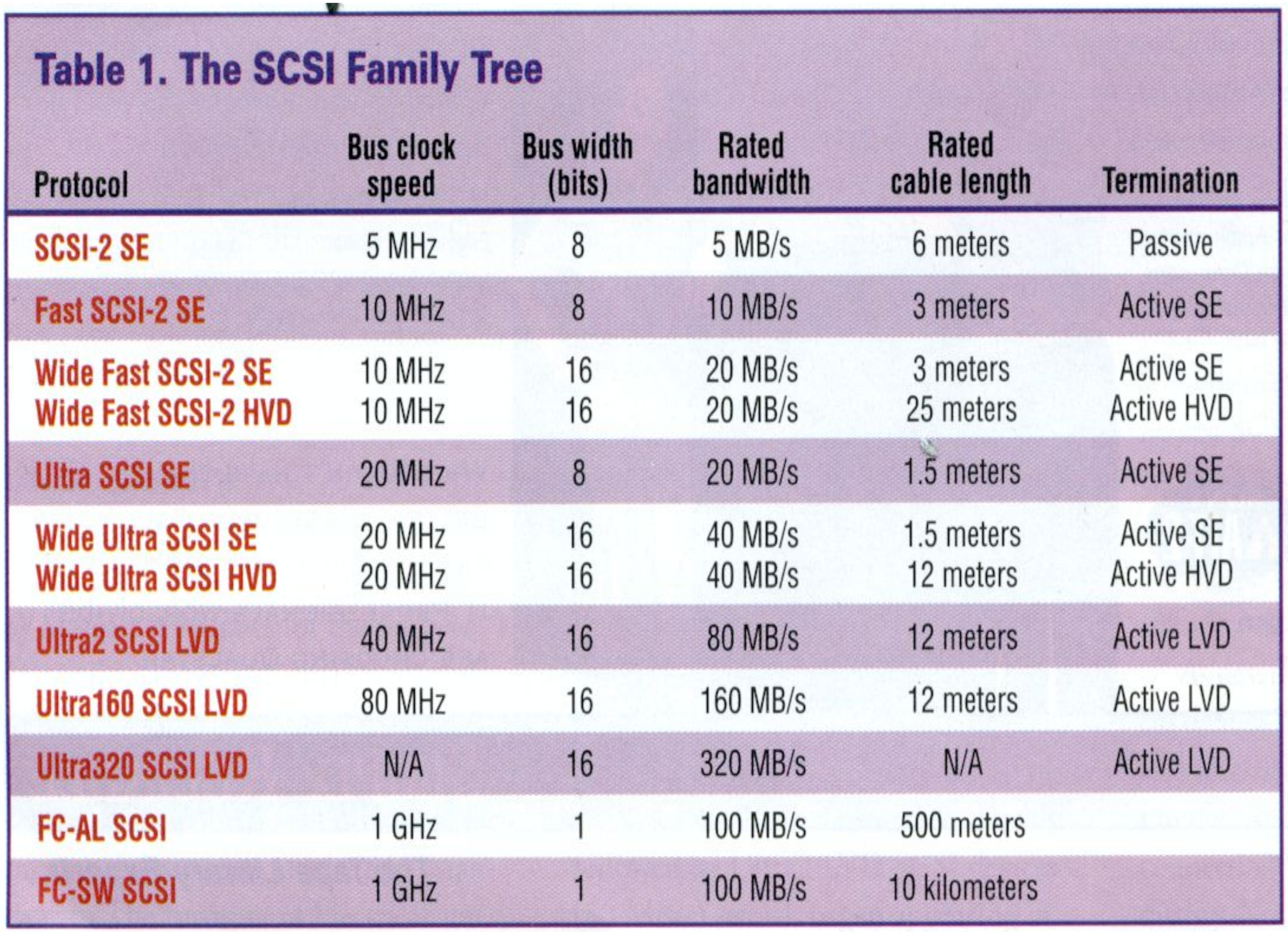


Table 1. The SCSI Family Tree

Protocol	Bus clock speed	Bus width (bits)	Rated bandwidth	Rated cable length	Termination
SCSI-2 SE	5 MHz	8	5 MB/s	6 meters	Passive
Fast SCSI-2 SE	10 MHz	8	10 MB/s	3 meters	Active SE
Wide Fast SCSI-2 SE	10 MHz	16	20 MB/s	3 meters	Active SE
Wide Fast SCSI-2 HVD	10 MHz	16	20 MB/s	25 meters	Active HVD
Ultra SCSI SE	20 MHz	8	20 MB/s	1.5 meters	Active SE
Wide Ultra SCSI SE	20 MHz	16	40 MB/s	1.5 meters	Active SE
Wide Ultra SCSI HVD	20 MHz	16	40 MB/s	12 meters	Active HVD
Ultra2 SCSI LVD	40 MHz	16	80 MB/s	12 meters	Active LVD
Ultra160 SCSI LVD	80 MHz	16	160 MB/s	12 meters	Active LVD
Ultra320 SCSI LVD	N/A	16	320 MB/s	N/A	Active LVD
FC-AL SCSI	1 GHz	1	100 MB/s	500 meters	
FC-SW SCSI	1 GHz	1	100 MB/s	10 kilometers	

SCSI

- SCSI bus systems can be very finicky.
  - **Termination is very important.**
    - There should be a terminator at each end of the bus.
    - Active (regulated, forced perfect) terminators are best.
  - **Cable length is very important.**
    - Specified cable lengths are maximums.
    - It is best to keep external cables as short as possible.
    - Internal cables often use a considerable amount of the cable budget.
  - **Don’t mix HVD and LVD devices!**
    - **High Voltage Differential**
    - **Low Voltage Differential**

SCSI

- Disk Formatting (SCSI)
  - SCSI disks are very different from their predecessors:
    - SMD disks had a fixed number of sectors/track.
      - **SCSI disks allow variable numbers of sectors/track.**
    - SMD disks had variable sector sizes.
      - **SCSI disks have a fixed sector size.**
    - SMD disks used data headers in each sector to keep track of control information. Therefore the sector size was usually set to 512 data bytes + some number of header bytes.
      - **SCSI disks do not have headers!**
        - » **That's a LIE**. **They do have headers, we just can't see them. The logic on the disk drive is the only thing that can see the headers.**

SCSI

- With the older disks, because of the fixed number of sectors/cylinder, the OS was in charge. The OS could always calculate exactly where the sector was because all of the operands were constants:
    - The system disk has 1024 cylinders, 19 heads, and 17 sectors/cylinder, and each sector can hold 512 data bytes.
        - The disk contains how many sectors?
            - » 1024*19*17 == 330752 sectors
        - The disk can store how many bytes of data?
            - » 330752 * 512 == 169,345,024 bytes

SCSI

- This user wants to read block zero of their file which the file system claims resides on block 14,560.
- How do we find the sector on the disk?
  - – 14,560 / (19*17) ==> Our sector is on the 45th cylinder
  - – The 45th cylinder starts at sector: 45*19*17 ==> 14,535
  - – We know our sector isn't the first one on cylinder 45, so we have to determine which surface it is on:
    - » 14,560 - 14,535 ==> Our sector is 25 sectors after the start of cylinder 45
    - » 45 / 17 > 2 Our sector is on surface 3
    - » 45 - (17 * 2) ==> Our sector is the 11th sector on surface 3
- Give me sector 11, on surface 3, cylinder 45 please

– NOTE: This is an oversimplification.

- In reality, the system counts from 0 for cylinder/head/sector!

SCSI

- With SCSI the variable number of sectors per cylinder tends to confuse the OS.
  - SCSI drives lie to the OS by telling it “I have C cylinders, H heads, and S sectors/track.
  - The OS uses this value when it fills in the file system tables.
  - The OS goes through the same computations to read/write the blocks on the disk.
- The SCSI disk hardware takes the numbers from the kernel, and does it’s own computations to determine where the block really resides on the disk!
- The operating systems still have the misconstrued notion that they are in charge!
  - In reality, most current OS’s just ask for file system block N, and let the disk drive do the calculations

SCSI

- Review - Disk Formatting (SCSI)
  - UNIX file systems do not understand the concept of variable numbers of sectors/track (or zone sectoring) on SCSI disks.
    - The file system wants to view the world as “the disk has X sectors/track, Y data heads, and Z cylinders”.
  - SCSI disk drives want the world to view them as N contiguous data blocks (1 cylinder, 1 head, N sectors).
  - In order to keep everybody happy, we tell the Operating System that the disk has some number of cylinders, some number of heads, and some number of sectors/track.
  - The disk drive promptly ignores this notation, and “does the right thing”

SCSI

- Disk Formatting (SCSI)
  - So how do we figure out these magic numbers?
    - First determine how many sectors the drive contains (alternately we can derive this from the drive capacity).
    - Factor this into prime numbers.
    - Make up #cylinder, #head, #sector entries based on these prime numbers and feed this information to the format program.
  - As long as the numbers of cylinders, heads, and sectors that we give the system generates a number equal to, or less than the number of sectors on the drive, we are in business.

SCSI

- It is generally best to tell the system numbers which make sense as far as performance goes:
  - Switching heads is fairly fast.
  - Switching cylinders is fairly slow.
    - Unfortunately, if you claim 1 cylinder, with a whole bunch of heads, the I/O performance is pretty bad.
    - It is better to try to find out how many cylinders the drive has, and use a # cylinders number that is close to that value.

SCSI

- The more sectors that go under one head in one revolution, the more data we can transfer without switching heads, or moving the heads to another cylinder.
  - On the other hand, the buffers are all finite sizes, so you lose if you tell the system too large a number for the number of sectors.
- Most vendors give a ncyl/nhead/nsect specification for their drives…use it whenever possible!
- When it isn’t possible, the following may be helpful:

SCSI

- Example: Assume we have a drive with 10,000 sectors.
  - Use the **/bin/factor** program to find the prime factors:
  - 10,000 ==> 2 X 2 X 2 X 2 X 5 X 5 X 5 X 5
  - Group these factors into C/H/S groupings:
  - 5 X 5 X 5 X 2 cylinders, 2 X 2 heads, 2 X 5 sectors
  - **250 cylinders, 4 heads, 10 sectors/track.**
- Example: Assume we have a 512 Megabyte drive:
  - determine the number of sectors by dividing the capacity by the sector size (512,000,000 / 512 ==> 1,000,000 sectors).
  - Factor this: 2X2X2X2X2X2X5X5X5X5X5X5
  - Group these factors into C/H/S groupings:
  - 5X5X5X5X2 Cylinders, 2X2X2 heads, 5X5X2X2 sectors
  - **1250 cylinders, 8 heads, 100 sectors/track**

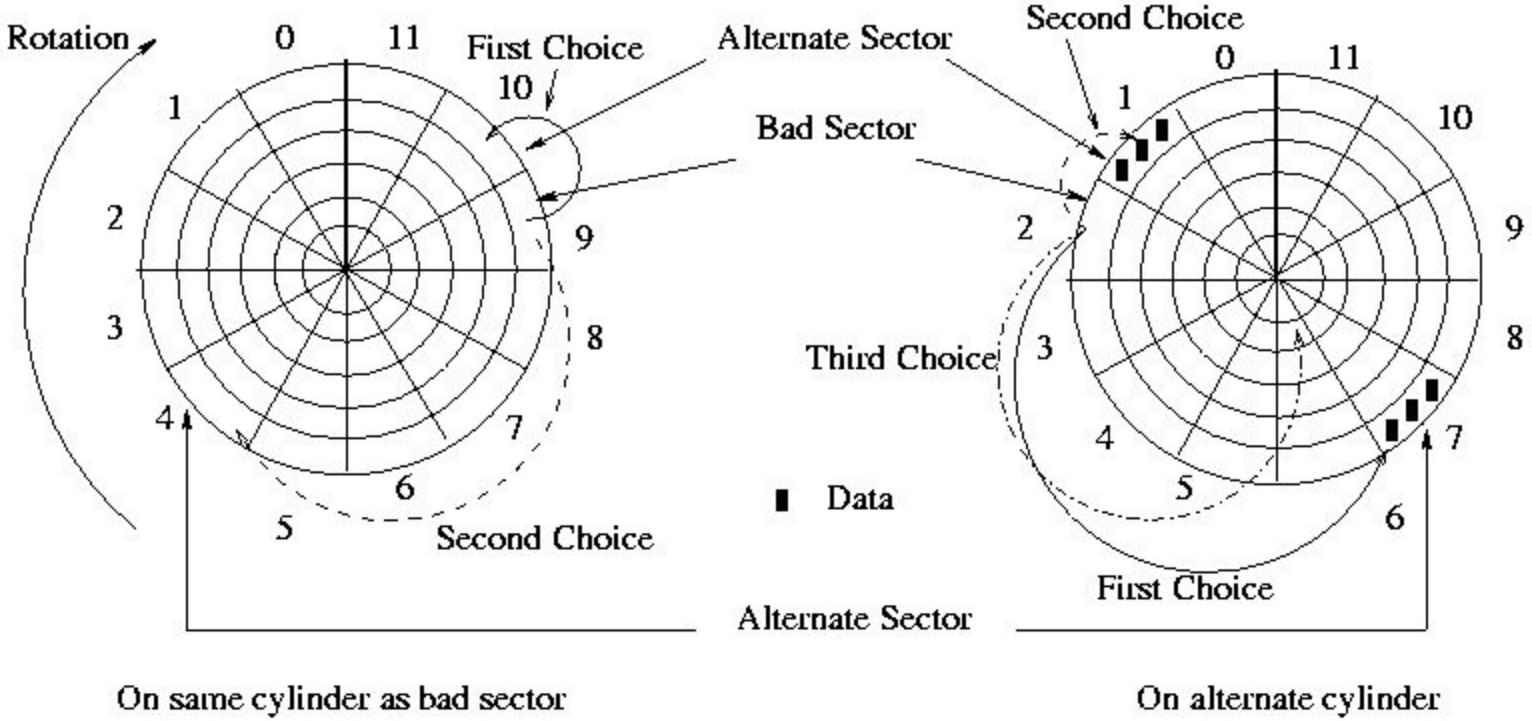
SCSI

- Disk Formatting (SCSI)
  - If SCSI disks were perfect, this scheme would work every time! Unfortunately, disks have defects when they leave the factory, and more defects appear over time.
  - In order to accommodate these defects, the concept of alternate sectors, alternate cylinders, and bad block forwarding was developed.
  - When the format program “numbers” the sectors from 0 through S-1. What happens if a sector is bad?
  - By allocating some number of “spare” sectors we can map bad sectors to good sectors!

SCSI

- Disk Formatting (SCSI)
  - Consider the following scenarios:
    - Platter with bad sector
    - Same platter with sector mapped to alternate sector on same cylinder.
    - Same platter with sector mapped to alternate cylinder.

SCSI


Rotation
0
11
First Choice
10
1
2
9
3
8
4
7
6
5
Second Choice
Alternate Sector
Bad Sector
Second Choice
0
11
1
10
2
9
Third Choice
3
8
4
7
5
6
First Choice
▮ Data
Alternate Sector
On same cylinder as bad sector
On alternate cylinder

SCSI

- Disk Formatting (SCSI)
  - Most versions of UNIX have a file which contains a database of disk models and corresponding format information for these disk drives. Under Solaris this file is **/etc/format.dat**
  - The format.dat file contains a lot more than just the C/H/S information on the drives:
    - Other information includes:
      - the type of controller,
      - the cache control flags for the drive,
      - the number of alternate sectors,
      - the rotational speed of the drive,
      - the number of bytes per track,
      - zone sectoring information, and
      - typical partition table entries for this drive.

```
# Data file for the 'format' program.  This file defines the known
# disks, disk types, and partition maps.
#
# This is the list of Sun supported disks for embedded SCSI.
#
disk_type = "SUN2.1G" \
          : ctlr = SCSI : fmt_time = 4 \
          : ncyl = 2733 : acyl = 2 : pcyl = 3500 : nhead = 19 : nsect = 80 \
          : rpm  = 5400 : bpt = 44823

disk_type = "SUN4.2G" \
          : ctlr = SCSI : fmt_time = 4 \
          : ncyl = 3880 : acyl = 2 : pcyl = 3500 : nhead = 16 : nsect = 135 \
          : rpm  = 7200

#
# This is the list of partition tables for the embedded SCSI controllers.
#
partition = "SUN2.1G" \
          : disk = "SUN2.1G" : ctlr = SCSI \
          : 0 = 0, 62320 : 1 = 41, 197600 : 2 = 0, 4154160 : 6 = 171, 3894240

partition = "SUN4.2G" \
          : disk = "SUN2.9G" : ctlr = SCSI \
          : 0 = 0, 390852 : 1 = 180, 781704 : 2 = 0, 11367972 : 6 = 454, 1019541
```

SCSI

- Disk Formatting (SCSI)
  - Once the information is in the format.dat file we use the format command to format the disk.
  - The format program issues a “format unit” directive to the SCSI disk.
  - The SCSI disk electronics format the media, and map out bad spots.
  - Once the sectors are all numbered and verified, the format command writes a label on the disk. This label contains the geometry information, and other information required by the operating system (rotational speed, capacity, partitioning...).

IDE

- The IDE interface, a mainstay of the PC market, has found its way into the workstation market because of its ease of configuration and low price.
  - The IDE interface automatically takes care of all of the details of formatting the drive, and mapping out bad sectors.
  - The SCSI interface requires more attention from the operator during format/repair operations, but the data throughput performance makes it the logical choice for high-performance servers and workstations.
- The system administrator needs to be aware of the interface capabilities when recommending systems for particular applications!
- The administrator also needs to be aware of the different operation/maintenance procedures for both SCSI and IDE drives

IDE

- Early IDE disks were relatively slow, at 1 megabyte per second.
- Current IDE standards (ATA33 and ATA66) have produced IDE disk systems that transfer at rates of 33 megabytes per second, and 66 megabytes/second, with 100-megabyte-per-second IDE systems (ATA100) just around the corner.

Fibre Channel

- Although current SCSI subsystems can transfer data at 160+ megabytes per second, this was not always the case.
- Designers wanted a faster channel between the host and the disk farm than SCSI could provide.
- One technology that provided a high-speed channel already existed: Fibre Channel.
  - Because it uses fiber optic cabling, Fibre Channel is capable of driving signals over a long distance at high speeds.
  - Fibre Channel technology provides connections capable of delivering a gigabyte of data per second, compared to a high-end SCSI that can provide two to four hundred megabytes of data per second.

Fibre Channel

- To provide maximum I/O subsystem performance, many RAID arrays include a fiber channel host interface.
  - This interface provides a 25-megabit-per-second (or faster) interface between the disk array and the file server.
  - High-end RAID arrays may include a 100-megabit-per-second Fiber Channel interface, or a Fiber Channel Arbitrated Loop (FCAL) interface.

Disk Subsystem

- Load Balancing Disks
  - Disks are one of the biggest performance bottlenecks in any computer system!
    - Reliable high-speed disks **do not** exist!
    - We have to make the best of the drives available.
    - We can increase disk I/O throughput by adding I/O adapters.
      - The system can overlap operations on all I/O adapters.
      - If we split the file systems up the right way we can help the system:
        - » Put swap and /tmp on separate disks,
        - » Put /usr and /opt on separate disks.

Disk Subsystem

- Load Balancing Disks
  - We can increase disk I/O throughput by adding spindles:
    - The system can overlap operations on multiple drives.
    - Command queuing on the drives improves throughput.
    - If we split the file systems up the right way we can help the system:
      - Put swap and /tmp on separate disks,
      - Put /usr and /opt on separate disks.

Disk Subsystem

- We can increase disk I/O throughput by using technologies such as Redundant Arrays of Inexpensive Disks (RAID).
  - RAID defines several types of disk interfaces.
    - » Some of these are for designed for high reliability applications,
    - » some are designed for high throughput applications.

Disk Subsystem

- Load Balancing Disks
  - Raid Level 0 implements a striped disk array. The data is written across multiple disk drives. This improves throughput by spreading the I/O load across several channels and drives. A striped disk **does not** provide any additional data redundancy or fault tolerance over standard disk systems.
  - Raid Level 1 implements a mirrored disk array. The data is written to two disks. This ensures that the data is available even if one disk drive fails. A mirrored array **does not** provide any performance benefits.
  - Raid Level 0 + 1 implements a mirrored striped array. This allows for data redundancy and improved throughput.

Disk Subsystem

- Load Balancing Disks
  - Raid Level 2 interleaves data sectors across many disks. A sophisticated error correction facility provides some degree of fault tolerance. Raid 2 is not found in use very often.
  - Raid Level 3 interleaves data sectors across many disks much like Raid Level 2. Raid Level 3 uses a separate “parity” disk per array for error correction. If a disk fails, the parity disk can regenerate that portion of the data on the fly.
  - Raid Level 4 implements a striped array with a parity disk. This level is not used very often.
  - Raid Level 5. Raid Level 5 interleaves data sectors across many disks much like Raid Level 3, but instead of a single parity disk, the parity information is also interleaved across multiple disks.

Disk Subsystems

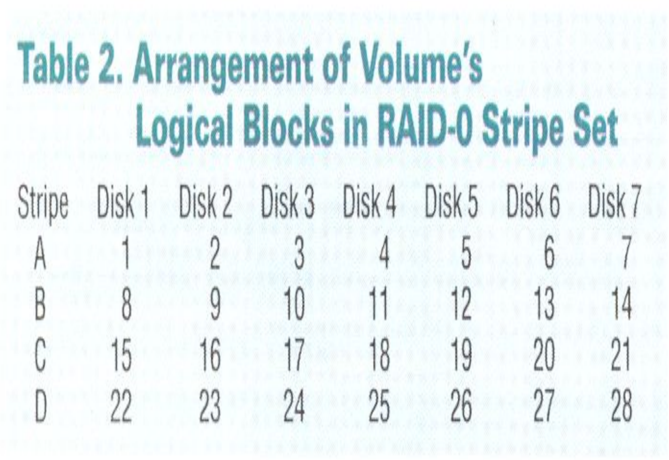


Table 2. Arrangement of Volume's Logical Blocks in RAID-0 Stripe Set

Stripe	Disk 1	Disk 2	Disk 3	Disk 4	Disk 5	Disk 6	Disk 7
A	1	2	3	4	5	6	7
B	8	9	10	11	12	13	14
C	15	16	17	18	19	20	21
D	22	23	24	25	26	27	28

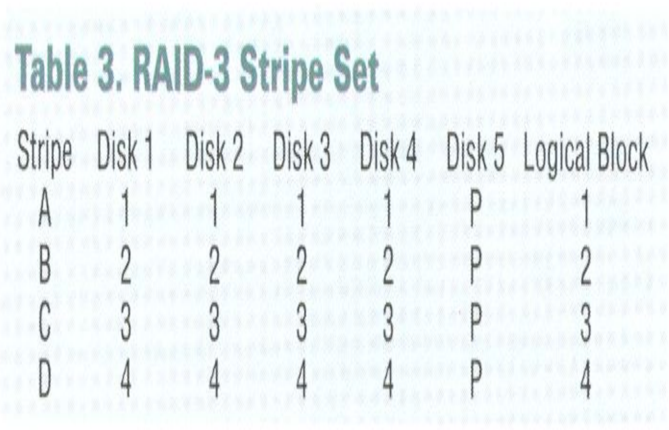


Table 3. RAID-3 Stripe Set

Stripe	Disk 1	Disk 2	Disk 3	Disk 4	Disk 5	Logical Block
A	1	1	1	1	P	1
B	2	2	2	2	P	2
C	3	3	3	3	P	3
D	4	4	4	4	P	4

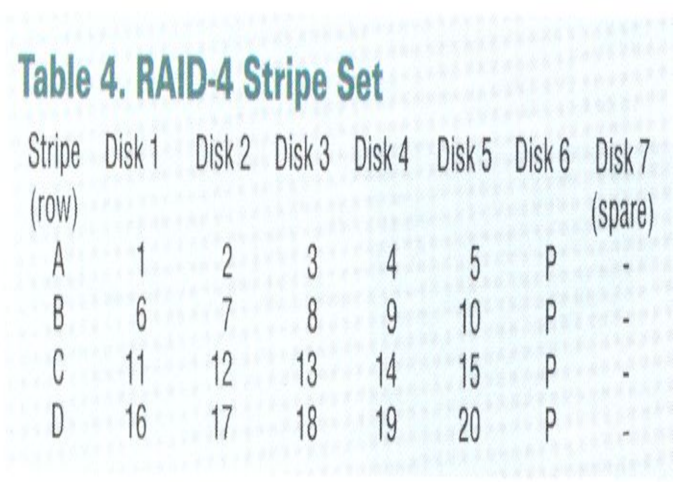


Table 4. RAID-4 Stripe Set

Stripe (row)	Disk 1	Disk 2	Disk 3	Disk 4	Disk 5	Disk 6	Disk 7 (spare)
A	1	2	3	4	5	P	-
B	6	7	8	9	10	P	-
C	11	12	13	14	15	P	-
D	16	17	18	19	20	P	-

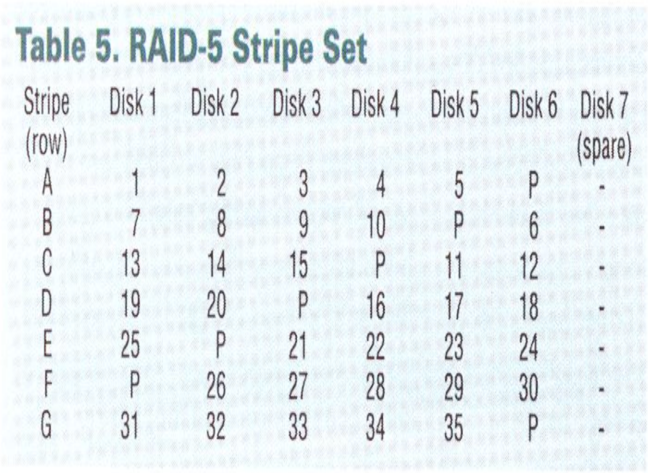


Table 5. RAID-5 Stripe Set

Stripe (row)	Disk 1	Disk 2	Disk 3	Disk 4	Disk 5	Disk 6	Disk 7 (spare)
A	1	2	3	4	5	P	-
B	7	8	9	10	P	6	-
C	13	14	15	P	11	12	-
D	19	20	P	16	17	18	-
E	25	P	21	22	23	24	-
F	P	26	27	28	29	30	-
G	31	32	33	34	35	P	-

Disk Subsystem

- Load Balancing Disks
  - Hardware solutions aren’t the only way to improve disk throughput and/or reliability.
    - Many Vendors offer software products that will treat ordinary SCSI controllers and disks as though they were an array of disks.
      - The Windows volume manager allows volume sets, RAID volumes, and extended storage.
    - Many vendors offer software that will migrate files between disk drives to improve throughput.
    - Many vendors offer products that will defragment and migrate files within a file system in an attempt to improve throughput.

General Checklist for Adding a Disk

- Regardless of the disk interface in use, the general procedure for adding a new disk to a system, is very similar.

  1. Identify the target address the new drive will use.
  2. Shut the system down using the *shutdown* command.
  3. Set the drive address jumpers, and install the new drive on the system.
  4. Boot the system as required to cause the system to find new hardware.
  5. If necessary, use the *format* and/or *fdisk* utilities to format and partition the new drive.
  6. Bind sectors in a partition into a file system.
  7. Check file system integrity.
  8. Add the new file systems to the system tables.
  9. “Mount” the new file systems.
  10. Bring the system down.
  11. Use *boot* to restart the system.

Adding an SCSI Disk to the System

- As long as the SCSI or IDE adapter does not already have its full complement of devices, connecting a new disk is pretty simple. Depending on the type of SCSI adapter, it may be possible to connect 7, or 15, devices to a single SCSI adapter.
  - Each device must have a unique SCSI identification (ID), which is also known as the SCSI address, or unit number. Some systems automatically select the SCSI ID when you plug the drive into the system chassis. This ensures that you do not connect two devices with the same address.
- On other systems the operator needs to determine which SCSI IDs are already in use, and then set the SCSI ID of the new device to one of the unused values.
  - This is typically accomplished by installing small jumpers on the drive. If the drive is mounted in a case, the SCSI is sometimes selected via a push-button switch on the case. Some OSs only allow certain types of devices to use specific SCSI IDs, whereas other systems have no such restrictions.

Adding an IDE Disk to the System

- IDE adapters allow the connection of two devices to the system. Most PC motherboards include two IDE channels, which facilitate the connection of four IDE devices (two per channel). IDE disks usually require less setup than their SCSI counterparts. If there is only one device connected to the IDE channel, install the MASTER jumper on the drive.
- If two devices are installed, one should be the MASTER (install the master jumper), and the other should be the SLAVE (install the slave jumper). Once the drive jumpers have been set, adding it to the system is as simple as plugging it into the IDE cable, and connecting a power cable to the drive's power connector.
- ***NOTE:* Setting the master/slave jumpers incorrectly, or attempting to connect more than two devices to the IDE bus will cause problems communicating with the IDE subsystem.**

NAS AND SAN

- ***Network-attached Storage***
  - Another type of storage system currently in use is the Network Attached Storage (or NAS) system.
  - The NAS system is a special-purpose computer that runs a stripped-down OS.
    - This computer is outfitted with several disk controllers, and many disk drives, typically providing hundreds of gigabytes of storage for a local facility.
    - The NAS storage system can typically provide network-based file sharing for UNIX and Windows clients.
  - A typical NAS storage system contains one or more high-speed network interfaces.
    - Data requests and delivery are carried out over the communications network.
    - It is not unusual for a NAS system to provide data to the client machines at a rate of 15 to 45 megabytes per second over a 100- (or 1000-) megabit-per-second network.

NAS AND SAN

- ***Storage Area Networks***
- A more recent entry in the storage systems arena is the storage area network, or SAN.
  - A SAN is designed to provide multiple paths between end nodes (client workstations) and storage elements.
  - A SAN relies heavily on high-speed fiber connections between storage components.
    - These data paths are independent of the normal network connections, and are used exclusively for disk-farm to client-system data transfers.
  - The system’s network interfaces are used for normal data communication traffic, such as web browsing, e-mail, and log-in sessions

Utilities

- Once a disk is connected to the system, it has to be “formatted” before it can be put into use.
- The task of formatting the disk drive is carried out using a hardware-based *format* utility, or an operating-system-based *format* utility.
  - The hardware-based utilities are often referred to as a “low-level” formatter, as they provide a hardware-level format to disk drive(s).
  - The low-level formatter knows nothing of partitions. These formatters number all sectors on the drive under the assumption that the entire disk drive is one large partition.

Utilities

- The software-based utilities allow the operator to partition the disk into multiple logical disk drives.
- The sectors on each logical partition might be independently numbered, but generally the formatter just creates a table that delineates how to split the existing “low-level” format into multiple partitions.
- Some OS provide multiple tools for partitioning and formatting disk drives. Other OS combine all of these functions into a single utility.

Windows File systems

- **Windows Disk Administrator Utility**
  - The Windows 2000 Disk Administrator utility is a GUI-based tool that allows the operator to format the disk, partition disks, write labels to disks, and assign volume labels and drive letters to system disks.
  - The Disk Administrator also allows the operator to create software-based RAID storage systems known as extended partitions or volumes.
  - In addition to allowing the operator to partition and format system disks, the Disk Administrator is the tool that creates the Windows file systems and all of the requisite file system structures on the disk.

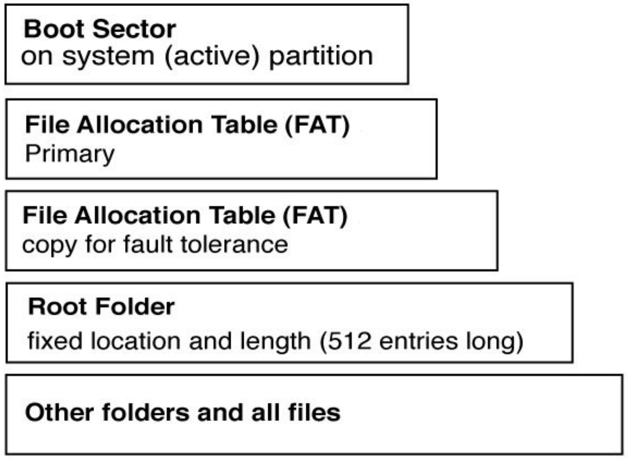
Windows File systems

- **Windows Command-line Disk Utilities**
  - The Disk Administrator utility contains all of the tools required to manage a disk on a Windows system.
  - However, sometimes it is not possible, or desirable, to use the GUI-based Disk Administrator utility.
  - In addition to the Disk Administrator, the command-line tools *fdisk.exe* and *format.exe* allow the administrator to partition and format disks.
    - The *fdisk* tool is used to partition disks, and to select the type of partition.
    - *fdisk* allows the operator to select which partition will be "active."
      - The active partition is the one that contains the OS you want to boot.
  - The *format* utility numbers all of the sectors on the partition, maps out any bad spots, and generates all of the necessary file system structures.
  - The *format* utility uses the following generic syntax.

```
format volume [/fs:file-system] [/v:label] [/q]
   [/a:unitsize] [/f:size] [/t:tracks /n:sectors] [/c] [/x]
   [/1] [/4] [/8]
```

Windows File systems

- Windows offers several file systems. Under the most recent commercial versions of windows there are three file systems available: FAT 16, FAT 32, and NTFS.
  - ***FAT***
  - The File Allocation Table (FAT) file system is the “old standard” file system for Windows installations.
  - The FAT file system suffers from a lack of feature bloat, but it is still the backbone of the consumer-oriented versions of Windows.
  - There are several versions of the FAT file system available under various versions of Windows.
    - The most current versions are FAT 16 and FAT 32.
      - FAT 16 is used for backward compatibility with older consumer versions of Windows.
      - FAT 32 is the latest version of the FAT file system,
    - The on-disk structure of the FAT 16 and FAT 32 file systems is similar, but just different enough to make them incompatible.
    - The FAT file system stores two copies of the file allocation table within the volume.
      - If one copy of the table becomes corrupted, in theory the system can automatically recover using the second copy of the information.



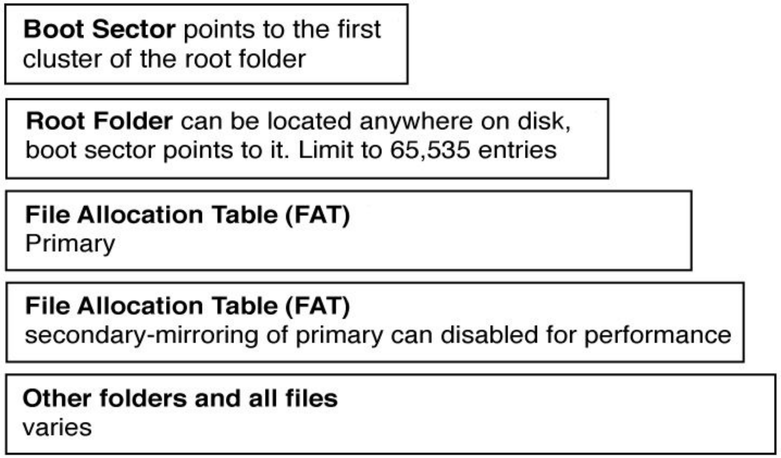
Boot Sector
on system (active) partition

File Allocation Table (FAT)
Primary

File Allocation Table (FAT)
copy for fault tolerance

Root Folder
fixed location and length (512 entries long)

Other folders and all files



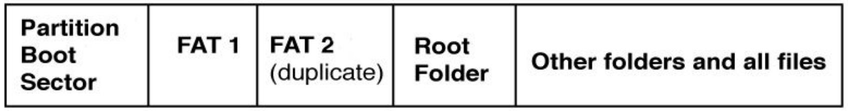
Boot Sector points to the first cluster of the root folder

Root Folder can be located anywhere on disk, boot sector points to it. Limit to 65,535 entries

File Allocation Table (FAT)
Primary

File Allocation Table (FAT)
secondary-mirroring of primary can disabled for performance

Other folders and all files
varies



Partition Boot Sector	**FAT 1**	**FAT 2** (duplicate)	**Root Folder**	**Other folders and all files**

Windows File systems

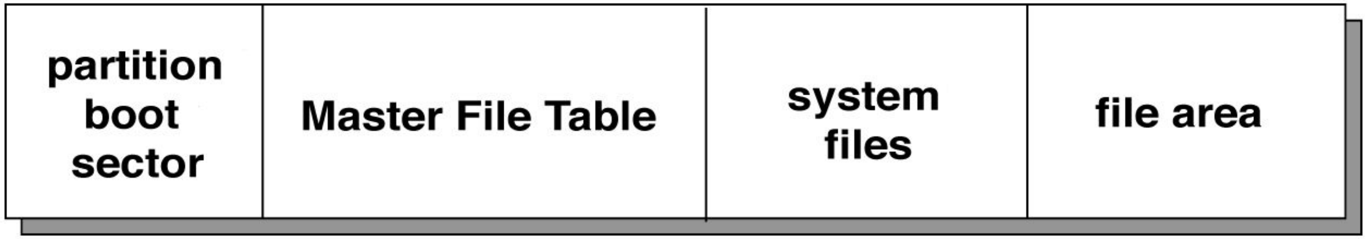
- The FAT 16 file system can support partitions up to 4 gigabytes (4,096 MB).
- Each file or folder stored in a FAT 16 file system is provided with a 32-byte entry in the file allocation table.
- The file allocation table is the master index of what files are stored on the partition, and which sectors/clusters are used to store this file.
- Using this information, it is also possible to determine which sectors should be “free” and available to store future files.
- The file allocation table is a packed list of 32-bit entries with a one-to-one mapping to file system clusters.
- The file allocation table entries contain the following information.
  - Name 8.3 format *filename.extension*
  - Attribute 8 bits Only four bits are currently defined: read-only, system file, archive, hidden
  - Create time 24 bits
  - Create date 16 bits
  - Last access date 16 bits
  - Last modified time 16 bits
  - Last modified date 16 bits
  - Starting cluster number 16 bits Cluster contains a pointer to the next cluster, or the value 0xFFFF if this is the last cluster used by the file
  - File size 32 bits

Windows File systems

- The FAT 32 file system supports partitions up to 2,047 gigabytes, and files up to 4 gigabytes, but the most current versions of Windows can only create partitions up to 32 GB.
- The file allocation table entries are very similar to the FAT 16 table entries, but many fields have been expanded in size.
- The primary operational difference is that the cluster number entry is 4 bytes (32 bits) on a FAT 32 file system, and a FAT 32 file system requires at least 65,527 clusters.
- FAT 16 and FAT 32 file systems do not scale well.
  - As the number of entries in the file allocation table increases, the amount of work required locating a file increases.
  - In addition, the table does not provide storage for attributes such as file ownership, or access modes.
  - The FAT file system provides no means of securing files from users that might use the system.

Windows File systems

- The New Technology File System (NTFS) is a newer file system provided with the commercial versions of Windows.
- The NTFS file system provides the structures required for file security and ownership, as well as correcting the scalability problems found in FAT file systems.
- In addition, the NTFS file systems includes transaction logging to provide increased reliability and performance.



partition boot sector	Master File Table	system files	file area

Windows File systems

- When a partition is formatted for NTFS, the following support structures and metadata required for the NTFS file system are created.
- The first 16 records (roughly 1 megabyte) of the MFT are reserved for the NTFS metadata.
  - **Boot sector: Sectors 0 through 15. The boot sector contains the BIOS parameter block.**
    - **BIOS parameter block: Contains information about the volume layout and file system structures.**
    - **Code: Describes how to locate and load startup files (typically *NTLDR.EXE*).**
  - **System files: Include entries for the following information.**
    - **MFT Record 0, Master File Table: Contains information about every file and folder stored on the partition.**
    - **MFT Record 1, Master File Table # 2: Backup copy of the MFT.**
    - **MFT Record 2, Log File: List of recent transactions. Used to provide recoverability.**
    - **MFT Record 3, Volume Name: Name of this partition**
    - **MFT Record 4, Attributes: Table of file and folder attributes, including security descriptors and the following file type attributes. • Standard information: Time stamps, link counts, and the like**
  - **Attribute list: List of the attribute records that do not reside in the MFT.**
  - **Filename: Up to 255 unicode characters, and an 8.3 representation.**
  - **Security descriptor: File access permissions.**
  - **Data: Data stored in the file.**

Windows File systems

- **Data: Data stored in the file.**
- **Index root: Used to implement folders.**
- **Index allocation: Used to implement folders.**
- **Volume information: Volume name and version number.**
- **Bitmap: Map of clusters used by this file.**
- **Extended attribute information: Not used by NT; provided for OS/2 compatibility.**
- **Extended attributes: Not used by NT; provided for OS/2 compatibility.**
- **MFT Record 5, Root Filename Index: Name of the root folder ($.).**
- **MFT Record 6, Cluster Bitmap: Bitmap that represents which clusters are used and which are free for use**
- **MFT Record 7, Partition Boot Sector: Bootstrap code for this partition, if the partition is marked as bootable. A copy of this information is located elsewhere on the partition.**
- **MFT Record 8, Bad Cluster File: List of the bad clusters on this partition.**
- **MFT Record 9, Quota Table: Disk usage information for each user. Currently unused.**
- **MFT Record 10, Uppercase Table: Table to provide lowercase to uppercase unicode character mapping.**
- **MFT Records 11 – 15: Six records reserved for future use.**

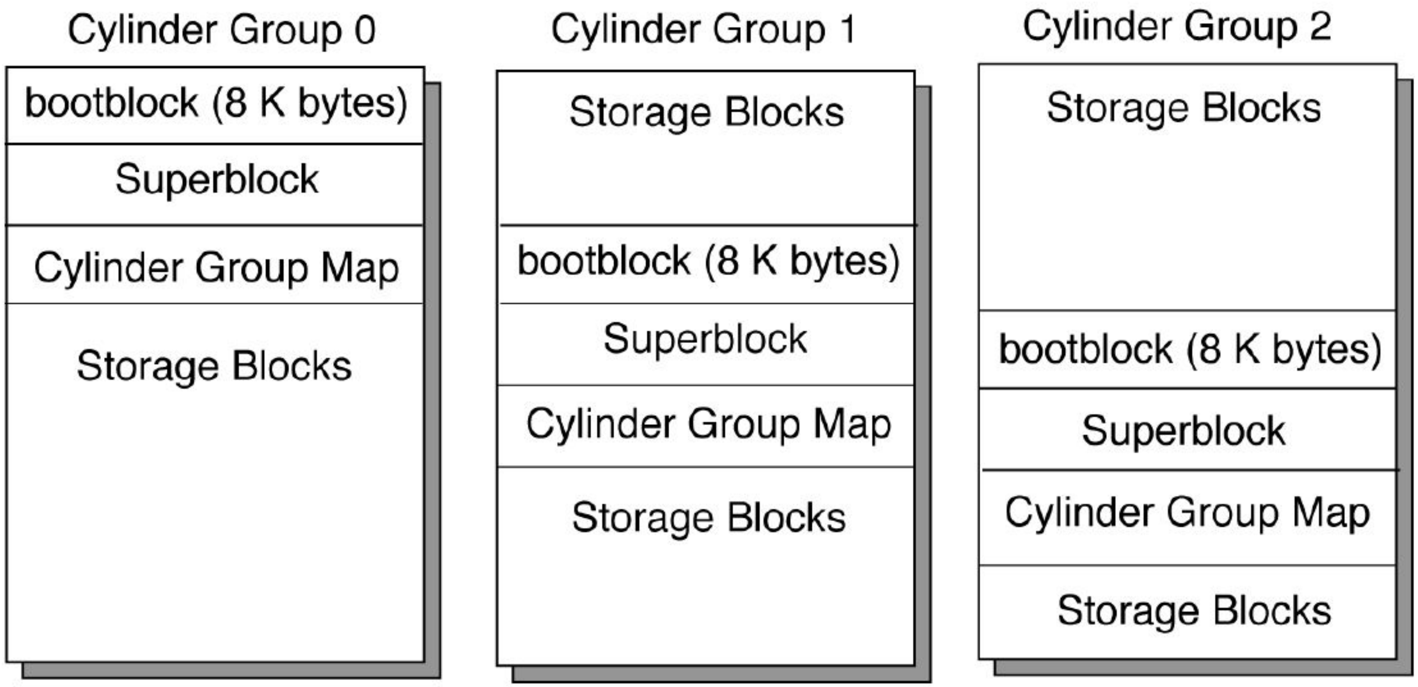
- It should be obvious that the NTFS stores much more information about a file than the FAT file system. This extra information allows the NTFS file system to support security, as well as provide improved system performance and file lookup speed.

UNIX File systems

- Due to the large number of UNIX variants, there are also large numbers of file systems supported under UNIX.
- The first five file system types listed—*ufs*, *hsfs*, *pcfs*, *nfs*, and *tmpfs*—are the most commonly used under most UNIX variants.
  - A workstation on a network may employ many types of file systems concurrently.
  - Some OS accomplish this by abstracting the file system driver out of the kernel, and into the file system mount code, or as loadable kernel modules.
  - Other OS accomplish this by making file system drivers an integral portion of the OS kernel code.

UNIX File systems

- ***ufs File Systems***
  - The *ufs* file system is the standard UNIX file system based on the Berkeley Fast File system**.**
  - Most other UNIX file systems rely on the structures provided by the UFS file system.
  - The current UFS file system implements several modifications to early UNIX file systems to improve disk subsystem performance and reliability.
  - Many system utilities will work only on *ufs* file systems. The most notable of these utilities are the *ufsdump* and *ufsrestore* commands.

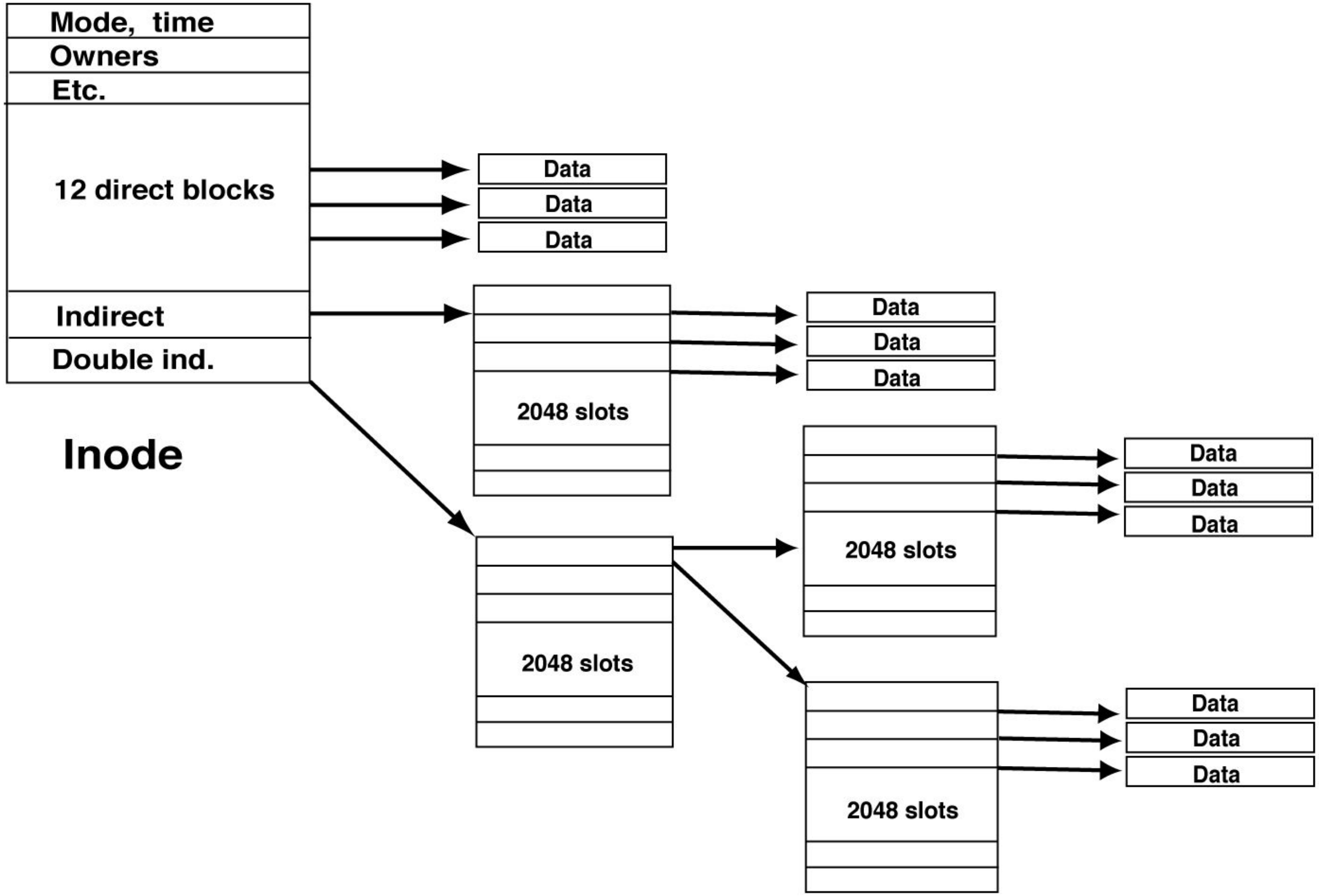

Cylinder Group 0
bootblock (8 K bytes)
Superblock
Cylinder Group Map
Storage Blocks
Cylinder Group 1
Storage Blocks
bootblock (8 K bytes)
Superblock
Cylinder Group Map
Storage Blocks
Cylinder Group 2
Storage Blocks
bootblock (8 K bytes)
Superblock
Cylinder Group Map
Storage Blocks

UNIX File systems

- The superblock provides the structure of the UFS file system.
  - The superblock stores much of the information about the file system. Such information includes the following.
    - **Size and status of the file system**
    - **Label: file system name and volume name**
    - **Size of the file system logical block**
    - **Date and time of the last update**
    - **Cylinder group size**
    - **Number of data blocks per cylinder group**
    - **Summary data block: contains summary information about the partition, number of *inodes*, directories, fragments, and storage blocks on the file system**
    - **File system state information: clean, dirty, stable, active**
    - **Path name to last mount point**

UNIX File systems

- Superblocks are located at the beginning of the partition, and are replicated in each cylinder group. Each superblock replica is offset from the beginning of the cylinder group by a different amount. The amount of offset is calculated to provide a superblock on each platter of the drive.
- The *inode* structure is stored in the summary information block of the superblock. The *inode* is a 128-byte structure that contains the following.
    - **Type of file: regular, directory, block special, character special, symbolic link, FIFI/named pipe, or socket**
    - **Mode of the file: read/write/execute permissions for this file**
    - **User ID of the file owner**
    - **Group ID of the file owner**
    - **Number of bytes in the file**
    - **Array of 15 disk block addresses used by this file**
    - **Date and time file was last accessed**
    - **Date and time file was last modified**
    - **Date and time file was created**


Mode, time
Owners
Etc.
12 direct blocks
Indirect
Double ind.
Inode
Data
Data
Data
2048 slots
Data
Data
Data
2048 slots
Data
Data
Data
2048 slots
2048 slots
Data
Data
Data

UNIX File systems

- The array of 15 disk blocks stored in the *inode* point to the data blocks that store the file's content. The first 12 of these blocks are called “direct” blocks. The direct blocks point directly to the first 12 data blocks of the file. If the file is larger than 12 data blocks, the 13th disk block points to an “indirect” data block. The indirect block contains block addresses of other blocks used by this file.
- The 14th disk block points to a double indirect block. The double indirect block points to blocks that contain addresses of additional indirect data blocks for this file. Similarly, the 15th disk block points to a triple indirect block. Figure 8-18 shows how these data blocks, indirect blocks, and double indirect blocks are used to keep track of the data portion of a file.

UNIX File systems

- ***Journaling File Systems***
- Journaling file systems are not a file system per se, but a method of protecting the system against lost data and corruption of information.
  - A journaled file system allocates some portion of the partition’s storage space as a transaction journal.
  - Every file system operation is logged to the journal.
  - If a problem such as a system crash or power failure occurs, the system can rebuild the file system transactions by reading the journal and replaying the transactions that were missed due to the problem.
- Several OS provide journaling file systems. IRIX, Solaris, Windows, AIX, HP/UX, and Linux all provide some form of journaled file system.
- Although each of these OS provides this feature, each OS accomplishes this via different means.

Linux File Systems

- The Linux OS currently provides support for 15 file systems, with more on the way!
  - Many of these file systems are similar to the “UNIX” file systems previously listed.
  - The remainder of the Linux file systems are different enough to warrant their own coverage.
  - As in UNIX, the Linux file systems are grouped into a single hierarchical file system tree structure.
  - Linux file system drivers provide the means of mounting the different file systems on the tree such that they may be accessed using standard utilities.

Linux File Systems

- The file systems currently supported by Linux include the following.
  - **Minix:**
  - **Ext:**
  - **Ext2:**
  - **Ext3**
  - **Xia**
  - **Umsdos**
  - **Msdos**
  - **Vfat**
  - **Proc**
  - **Smb**
  - **Ncp**
  - **Iso9660**
  - **Sysv**
  - **Hpfs**
  - **Affs**
  - **Ufs**

Creating UNIX File Systems

- Except in special cases, a file system must be created on a partition before the partition can be used under UNIX.
- The act of creating a file system puts a framework on the partition that enables UNIX to store files there.
- The file system framework provides index information for the disk blocks initialized and mapped when the disk was formatted.
- Creation of a new file system under most variants of UNIX can be accomplished by invoking the *newfs* utility.
  - The *newfs* utility allows you to create file systems with default (or custom) file system parameters.
  - In effect, *newfs* is just a user-friendly front end to the *mkfs* program.
    - The *mkfs* command creates the file system superblocks, divides the file system into several cylinder groups, and takes care of creating all file system substructures required by the OS.

Working with the newfs Command

- On most UNIX systems, the default type of file system created by *newfs* is a *ufs* file system.
  - Two notable exceptions are Linux, which defaults to an EXT2 file system, and Irix, which defaults to an XFS file system.
- The syntax for invoking the *newfs* command:

```
# newfs [mkfs options /dev/rdsk/c#t#d#s#
```

  - Invoking *newfs* with the *-N* option causes *newfs* to display the basic parameters of the file system without actually creating it.
  - The *newfs* command will display information about the new file system as it operates.
    - The first line printed by *newfs* describes the basic disk geometry.
- The second line of output from *newfs* describes the *ufs* file system created in this partition.
  - Cylinder groups are a concept introduced in the Berkeley UNIX Fast File system. Cylinder groups provide a method by which file blocks are allocated close together on the disk to improve performance.

Working with the newfs Command

***TIP:* Varying the size of the cylinder groups is one way to tune the performance of a *ufs* file system.**

- The final two lines of *newfs* output list the locations of the superblock backups.
  - The superblock is the head of the file index node or *inode* information used by the *ufs* file system.
  - The *inodes* are “used” by routines in the Solaris kernel to allocate, read, write, and delete files.
  - Backup copies of these superblocks are created in the event a system crash, power failure, or other problem leaves the main copy of the superblock damaged.

How newfs Connects to mkfs

- As previously mentioned, the *newfs* command is a front end to *mkfs*.
  - The *newfs* command invokes *mkfs* to perform the actual creation of the file system on the target device.
  - The *mkfs* command requires a long list of parameters.
    - For most applications, a detailed knowledge of the *mkfs* parameters is unnecessary; *newfs* makes proper choices for the parameters it passes to *mkfs*.
    - It is possible to build a file system tuned to store a large quantity of small, frequently changing files, such as those found in the Usenet net-news system. Such a file system would require more index nodes or *inodes* than are provided by the default options used by *newfs*.

***TIP:* Use of the *-v* option flag with *newfs* will allow the administrator to see which parameters it passes to *mkfs*.**

- Many file system parameters may be altered by using an option flag to *newfs* instead of manually invoking *mkfs*.

Mounting File Systems

- File systems can be mounted via several methods:
  - manual mounting by invoking the */etc/mount* command,
  - mounting the file system at boot time via the */etc/[v]fstab* file,
  - or mounting via Volume Manager.
- **General Procedure for Mounting a File System**
  - The general procedure for mounting a file system is as follows.
  - 1. Format the disk drive.
  - 2. Partition the disk drive.
  - 3. *newfs* the partition.
  - 4. Create a mount point for the file system.
  - 5. Determine which method will be used to mount the file system (manual mount, *[v]fstab* mount, Automounter, or Volume Manager).
  - 6. Mount the file system.

Mounting File Systems

- Options are given as a comma-separated list following the *-o* flag (e.g., *mount -o rw,quota*).
  - Some of the more common options for the mount command are.
    - *Quota* - Activates the user disk quota system when the file system is mounted
    - *Ro* - Mounts the file system read-only
    - *-F fstype* - Mounts a file system of *fstype*
    - *-a* - Mounts all file systems with “mount at boot” (in the *vfstab* file) set to *yes*
    - *-o* - Uses specific *fstype* options when mounting file system
    - *-p* - Provides a list of mounted file systems in *vfstab* format
    - *-v* - Provides verbose output when mounting file system(s)
    - *-O* - Overlays the file system on an existing mount point
    - *-m* - Mounts file system without making an entry in */etc/mnttab*
    - *Nosuid* - Disallows *setUID* execution for files in this file system

Mounting File Systems

- **Mounting via the fstab**
  - The *[v]fstab* file is used to mount file systems at boot time.
    - The “v” is bracketed, as some versions of UNIX call this file */etc/fstab* (primarily BSD UNIX), other versions of UNIX call this file */etc/vfstab* (primarily System V UNIX), and a few versions of UNIX call this file */etc/checktab* (HP/UX).
  - This file is read by the */etc/mountall* command when it is run as part of the system boot sequence.
  - A quick way to add items to [*v]fstab* is to use the *- p* option to the *mount* command.
  - It is also possible to add new file systems by editing the *[v]fstab* file and entering the required information manually.
  - The *[v]fstab* file format may contain minor modifications on different variants of UNIX,.
    - A tab or spaces separate the fields of this file.
    - The following is a typical Solaris *[v]fstab* file.

Mounting File Systems

```
#device        device         mount     FS   fsck mount     mount
#to mount      to fsck        pointtype pass at boot   options
/proc          -         /proc     proc -    no   -
fd        -         /dev/fd   fd   -    no   -
swap      -         /tmp tmpfs     -    yes  -
/dev/dsk/c0t3d0s0   /dev/rdsk/c0t3d0s0 / ufs 1    no   -
/dev/dsk/c0t3d0s6   /dev/rdsk/c0t3d0s6 /usr  ufs  2    no   -
/dev/dsk/c0t3d0s5   /dev/rdsk/c0t3d0s5  /opt ufs  5    yes  -
/dev/dsk/c0t3d0s1   -         -    swap -    no   -
/dev/dsk/c1t3d0s1   -         -    swap -    no   -
```

- Device to mount - Name of device to be mounted.
- Device to *fsck* - Raw device to be checked by the *fsck* utility.
- Mount point - Directory where the device should be added to the UNIX file tree.
- FS type - File system type.
- *fsck* pass - Number indicates the order which the file system will be checked.
- Mount at boot - Yes to cause file system to mount at boot.
- Mount options - Options to be passed to the *mount* command.

Mounting File Systems

- UNIX file systems (*ufs*) are the only file systems on which the file system checker (*fsck*) operates.
- If the *fsck* pass field contains a minus sign (–) or a zero (0), no file system integrity checking is performed.
- A file system with an *fsck* pass of one (1) indicates that the file system is to be checked sequentially (in the order it is listed in the [*v]fstab* file).
- Note that the / (*root*) file system is always checked first.
- File systems with an *fsck* pass number greater than one (1) are checked in parallel (simultaneously).
- For systems with a large number of disks spanning multiple disk controllers or disk busses, parallel file system checking is generally faster than sequential checking.
- For efficiency, use *fsck* on file systems of similar size on different disks simultaneously.

- Once the appropriate information has been entered in the [*v]fstab* file, you can mount all file systems by invoking the following command.

```
# /etc/mount -a
```

Mounting File Systems

- The Volume Manager provides users with a means of mounting/unmounting removable media without granting system users root privileges typically required to mount/unmount a file system.
- Under Linux, you can add the “user” option to the list of options for a file system. This allows any user to mount/unmount the file system (an operation that usually requires root access).
  - For example, the following *fstab* entry would allow any user to mount the */jaz* file system located on the removable disk */dev/sda1*. The *nosuid* option disallows execution of *suid* programs from this medium.

```
/dev/sda1 /jaz ext2 defaults,user,exec,nosuid,noauto 0 0
```

- *usermount* is a graphical tool developed by mdejonge@wins.uva.nl.
  - The *usermount* tool is available for several UNIX distributions that will allow users to manage removable media, such as floppy disks or zip disks.
  - When the tool is invoked, it scans */etc/fstab* for all file systems that have been configured to allow users to mount and unmount them.
  - The file system can be mounted or unmounted by pressing the toggle button labeled Mount. *usermount* is available at *http://www.cwi.nl/~mdejonge/software/usermount/*.

Creating and Mounting

- ***Identifying Mounted File Systems***
  - Connecting a file system to the tree structure is called “mounting” the file system.
  - UNIX provides several ways for the administrator to identify which file systems are currently mounted on the system. Three of the most common methods are:
    - invoke */etc/mount* command invoked with no arguments,
    - Invoke the *df* command, and
    - examination of the */etc/mnttab* file
  - **The df Command**
  - Invoking the df command with no argument results in the display of information for all mounted file systems.

```
glenn% df
/       (/dev/dsk/c0t3d0s0):11690 blocks  9292 files
/usr    (/dev/dsk/c0t3d0s6):786776 blocks 322309 files
/tmp    (swap):         218816 blocks 3288 files
/opt    (/dev/dsk/c0t3d0s5):91236 blocks  73801 files
```

Creating and Mounting

- When *df* is invoked with a directory argument, the output format changes, as shown in the following.

```
glenn% df /
File system  kbytes       used avail    capacity Mounted on
/dev/dsk/c0t3d0s0 20664       14819    3785 80%      /
```

- The most complete *df* listing is obtained by using the *-g* option

```
glenn% df -g /
/    (/dev/dsk/c0t3d0s0):  8192 block size  1024 frag size
41328 total blocks   11690 free blocks  7570 available 11520 total files
9292 free files      8388632 filesys id
ufs fstype        0x00000004 flag  255 file name length
```

- ***NOTE:* It is important to remember the differences among the three block sizes discussed in this chapter. A disk block is the basic unit of storage on the physical medium. This unit is often 512 bytes long, but not always. An allocation block is the basic unit of space in a file system. It is either 8,192 or 4,096 bytes in length. A fragment, often 1,024 bytes in length, is a portion of an allocation block.**

Creating and Mounting

- **The /etc/mnttab File**
- Another way to determine which file systems are mounted is by examination of the */etc/mnttab* file.
  - This file is created and maintained by the */etc/mount* command.
  - Every time a file system is mounted, an entry is added to the *mnttab* file. When a file system is unmounted, the entry is removed from the *mnttab* file.

```
# cat /etc/mnttab
/dev/dsk/c0t3d0s0 / ufs rw,suid,dev=800018 869622929
/dev/dsk/c0t3d0s6 /usr ufs rw,suid,dev=80001e 869622929
/proc /proc proc rw,suid,dev=2740000 869622929
fd /dev/fd fd rw,suid,dev=27c0000 869622929
/dev/dsk/c0t1d0s6 /var ufs rw,suid,dev=80000e 869622929
swap /tmp tmpfs dev=0 869622932
```

Creating and Mounting

- **The mount Command**
- The *mount* command allows the administrator to view which file systems are mounted, as well as providing a method of mounting file systems. When invoked without arguments, the *mount* command lists mounted file systems by their mount points, showing the device mounted at each mount point, the mount options used, and the time the file system was mounted.

```
glenn% /etc/mount
/ on /dev/dsk/c0t3d0s0 read/write/setuid on Sat Apr 1 1:23:45 2000
/usr on /dev/dsk/c0t3d0s6 read/write/setuid on Sat Apr 1 1:23:45 2000
/proc on /proc read/write/setuid on Sat Apr 1 1:23:45 2000
/dev/fd on fd read/write/setuid on Sat Apr 1 1:23:45 2000
/tmp on swap read/write on Sat Apr 1 1:23:45 2000
/opt on /dev/dsk/c0t3d0s5 setuid/read/write on Sat Apr 1 1:23:45 2000
```

Creating and Mounting

- ***Unmounting a File System***
- The complement to mounting a file system is to unmount it using the */etc/umount* command.
  - Exercise great care when unmounting file systems!
  - Some file systems are required for proper system operation.
  - Other file systems may be unmounted while allowing the system to continue to operate, but the system will not perform as expected.
  - The syntax for the *umount* command follows.

```
# umount mount_point
```

or

```
# umount device
```

Creating and Mounting

- The first form of the *umount* command unmounts the file system referred to by "mount point."
- For example, *umount /scratch* would unmount the */scratch* file system.
- The second form of the *umount* command unmounts the file system on the referenced disk partition.
- For example, if the */scratch* file system is on the *c0t2d0s1* disk partition, the operator could unmount it with *umount /dev/dsk/c0t2d0s1*.
- If you are uncertain about the effect unmounting a file system may have on the system, it is best to bring the system to the single-user state before invoking the *umount* command.
- In the case of a file server, unmounting a file system may also have an effect on *nfs* client machines on which the file system is mounted.

Creating and Mounting

- ***Volume Manager***
- Adding entries to the Solaris *vfstab* file works well for hard disks but is not suitable for removable media such as floppies and CD-ROMs.
- These devices tend to be mounted and unmounted much more frequently than hard disks, and the user performing this “mount” operation may not have the root privileges required to mount a normal file system.
- To handle such situations, Solaris uses the Volume Manager.
- Other OSs handle this situation by providing utilities under the desktop manager toolkit (such as KDE under Linux), *mediad* under Irix, and *autodiskmount* under MacOS X.
- Under Solaris, automatic detection of the media is limited to the CD-ROM device. The Solaris File Manager accomplishes this with a combination of a daemon (*vold*) and a configuration file (*/etc/vold.conf*), which specifies the actions to be taken for various removable devices and file system types.
- For floppy disks, Volume Manager is unable to detect the presence of new disks and must be informed that a new disk is present. This is accomplished by invoking the */bin/volcheck* command.
- Volume Manager always mounts floppy disks under the */floppy* directory it controls. CD-ROMs are likewise mounted under the */cdrom* directory.

Creating and Mounting

- ***Configuring Volume Manager***
- Changing the actions taken and devices under control of Volume Manager is a simple matter of modifying the */etc/vold.conf* file. #

```
# Volume Daemon Configuration file
#
# Database to use (must be first)
db db_mem.so
# Labels supported
label dos label_dos.so floppy
label cdrom label_cdrom.so cdrom
label sun label_sun.so floppy
# Devices to use
use cdrom drive /dev/dsk/c0t6 dev_cdrom.so cdrom0
use floppy drive /dev/diskette dev_floppy.so floppy0
# Actions
insert /vol*/dev/diskette[0-9]/* user=root /usr/sbin/rmmount
insert /vol*/dev/dsk/* user=root /usr/sbin/rmmount
eject /vol*/dev/diskette[0-9]/* user=root /usr/sbin/rmmount
eject /vol*/dev/dsk/* user=root /usr/sbin/rmmount
notify /vol*/rdsk/* group=tty /usr/lib/vold/volmissing -c
# List of file system types unsafe to eject
unsafe ufs hsfs pcfs
```

Creating and Mounting

- **The rmmount.conf Configuration File**
- The */usr/sbin/rmmount* command has its own configuration file named */etc/rmmount.conf*.
  - Although not often modified, this file allows the specification of additional actions to occur when a disk is mounted.
  - A common use of this feature is to allow CD-ROMs mounted by Volume Manager to be automatically shared, or made accessible to other workstations on the network via NFS.
  - To accomplish this, a *share* line is added to the bottom of the */etc/rmmount.conf* file as follows.

```
share cdrom*
```

- This line would share any CD-ROM mounted by Volume Manager without any restrictions. To control access the administrator can add options to the *share* line in a form similar to the *share* command.

Creating and Mounting

- Mounting Non-ufs File Systems with Volume Manager
- Volume Manager is also able to handle file system types other than *ufs*.
- For instance, inserting an MS-DOS-formatted floppy and using File Manager or the *volcheck* command results in the disk being mounted under the */floppy* directory on a mount point that bears the floppy volume name.
- Starting and Stopping Volume Manager
- UNIX makes it possible to disable Volume Manager and work directly with the CD-ROM and/or floppy drive.
- To disable Volume Manager, invoke the */etc/init.d/volmgt* script with the *stop* option.
- To restart Volume Manager, invoke the */etc/init.d/volmgt* script with the *start* option.

Summary

- This chapter focused on the mass storage subsystem, one of the most critical subsystems.
- Topics included how to install a new disk drive, and formatting and partitioning a disk drive.
- Local file systems and their relationships to the UNIX file tree constitute a basic building block of a UNIX system.
- Understanding how file systems are created, mounted, checked, and tuned enables the administrator to effectively manage corporate disk space.
- More importantly, understanding how to mount, create, and unmount file systems on an active system can result in minimization of down-time for corporate users.

Summary

- The administrator should be able to Recognize basic disk components.
- The administrator should understand basic disk geometry.
- The administrator should understand partitioning and formatting.
- The administrator should recognize advantages and disadvantages of SCSI, IDE, and alternative disk systems.
- The administrator should recognize similarities and differences in popular Windows and UNIX file system structures.
- The administrator should understand how to create a file system.
- The administrator should understand how to make a file system available.

- Unix And Linux System Administration Handbook, 5th Edition By Evi Nemeth, Garth Snyder, Trent R. Hein, Ben Whaley, Dan Mackin. Released September 2017. Publisher(s): Addison-Wesley Professional. ISBN: 9780134278308
- Practice Of System And Network Administration, The: Devops And Other Best Practices For Enterprise IT, Volume 1, By Thomas A. Limoncelli, Strata R. Chalup, Christina J. Hogan. Released November 2016. Publisher(s): Addison-Wesley Professional. ISBN: 9780133415087
- Essential System Administration, Third Edition bv Æleen Frisch, Published by O'Reilly Media, Inc. (20(0-596-00343-9