SYSTEM ADMINISTRATION

CED17I034

D.JASWANTH REDDY

DEVICE DRIVERS ASSIGNMENT

Contents pg.no

1 Introduction

1.1 System administration is the field of work in which someone manages one or more systems, be they software, hardware, servers or workstations. Its goal is ensuring the systems are running efficiently and effectively.

System administration is typically done by information technology experts for or within an organization. Their job is to ensure that all related computer systems and services keep working.

A **system administrator**, or sysadmin, is a person responsible to maintain and operate a computer system or network for a company or other organization. System administrators are often members of an information technology department.It seems that most people agree that this job has to do with computers in some capacity, but that really is already where the definitions start to become fuzzy: System Administrators are in charge of “servers” as well as of personal computers, desktop machines, laptops and, increasingly, mobile devices. With the evolution of computing coming full circle, we find that the migration from the mainframe computer to the personal and independent computing device to the networked server back to the central Infrastructure as a Service model inherent in today’s Cloud Computing concepts places the System Administrator in the middle of it all.

So perhaps the main focus of a System Administrator’s job is then really the central connection between the independent systems, the network itself? Hmmm, System Administrators surely are involved in the deployment and installation of the computers in a datacenter (or is that the task of specific Data Center Technicians?) and connecting all the different components certainly involves a lot of cables. But then, don't we have Network Administrators, or is that a more specialized subcategory of System Administrators?

System Administrators seem to spend as much time typing cryptic commands into dark terminal windows as they do running cables and labelling hardware, and while they may no longer shuffle punch cards, they frequently do write programs to help them complete their tasks. System Administrators are known to get woken up in the middle of the night when things go "bump", and as a result they are also known to have a fondness for caffeinated drinks. They are able to kickstart a generator, assess the required cooling power for their server room, use duct tape in creative and unexpected ways, assemble servers out of mysterious looking parts, and may end up handling a circular saw or other heavy machinery when their Leatherman multi-tool cannot complete the task.

System Administrators plan, budget and design networks and backup or storage systems, add and delete users (well, user accounts, anyway), install and update software packages, draft policy documents, fight spam with one hand while rebuilding a corrupted revision control system with the other. They have access to all systems in the organization, may undergo retina- and fingerprint scanners to access "Mission Impossible"-style protected data centers and spend countless hours in front of a multitude of computer screens, typing away on oddly shaped keyboards consuming not entirely healthy amounts of coffee and energy drinks.

Well... in some places, a System Administrator might do all of this. In others, there might be different, more specialized people for the various tasks: there might be datacenter technicians and "SiteOps", Network Administrators, System Programmers, System Architects, Operators, Service Engineers, Storage Engineers, Site Reliability Engineers, Virtual Operations, Infrastructure Architects... the number of different job titles for things that might otherwise fall into the more general “SysAdmin” category seems endless.

The various areas of expertise included in the day to day routine such as system design, infrastructure architecture, system fault diagnosis, hardware benchmarking, and others eventually require experience in a number of related fields; a lot of them involve more than just a little bit of programming experience and in some cases complex infrastructure tools based on solid software engineering practices need to be built. This illustrates that just as it is hard to clearly define where System Administration begins, we often can’t quite put our finger on where it ends and another discipline becomes the primary job function.

Even if job descriptions and duties differ significantly, there are a lot of people who simply identify as System Administrator regardless of job title. We all have something in common: we manage and maintain computer and network systems.

1.2 The Profession of System Administration

Most professions have a fairly clear and formal definition as well as an obvious career path and background requirements. In fact, for a person to take on a particular line of work, it frequently is a requirement to undergo a specific, regulated education, to perform a certain amount of practical training (such as an apprenticeship), to subscribe to specific professional ethics and guidelines, and eventually to obtain a license to practice the given profession. Doctors, scientists and lawyers all have well-defined professions, as do engineers of most disciplines.

The profession of System Administration is incredibly interesting precisely because it eschews a fixed definition, a formal governing body, a static career path, or a licensing exam. It attracts and invites interesting people from all sorts of backgrounds; it allows phenomenal potential for job growth and it is – must be, due to the advances in the industry – fast-paced and continually evolving.

System Administrators may not need to be licensed by a central board anytime soon – and people will continue to argue about whether or not that is desirable or even possible in this field – but what a few years ago still was best described as “a job” certainly has grown up to become a career, a craft, a profession.

1.3 System Administration Education

System Administration is rarely taught as an academic discipline in large part due to its perceived “blue-collar” status: from the top of the scholarly ivory tower, it must be hard to distinguish where an entry-level job like “Tech Support” or the “Help Desk” ends and where a profession demanding a significant background in Computer Science and Engineering begins.

System Administrators are highly skilled information technology specialists shouldering significant responsibility in any organization. As we discussed in the previous sections, the variety of job descriptions and differences in what a System Administrator actually may be doing make it difficult to provide simple step-by-step instructions on how to enter this field. As a result, System Administration has long been a profession that is learned primarily by experience, where people grow into a position in order to fulfill the requirements of an organization rather than follow a career path well-defined by courses, degrees, and meaningful certifications.

1.4 Challenges in System Administration Education

Formal classes in System Administration as part of a Computer Science or Engineering curriculum are still uncommon, but in the past few years more and more institutions have recognized the industry's need for academic courses that adequately prepare students for the multitude of responsibilities within this field and have started to offer such classes (and in some cases complete degree programs). But the history of this profession brings with it a number of unique challenges when it comes to formally teaching its principles and practices.

To really understand and appreciate some of the most general aspects of System Administration, you need to be exposed to actual running systems. Practical experience is so integral to this profession that it cannot be separated from the theoretical knowledge and postponed until the student enters his or her first job or apprenticeship. But therein lies a significant hurdle to traditional teaching methods: students need to administer a system, to have superuser access, to have a chance to configure a system for a specific service, and to make the kind of spectacular mistakes that experienced System Administrators value (if only in hindsight).

This normally conflicts with the requirements of the IT department at your university: students would require access to a number of different OS when the school's system administrators strive for a certain level of homogeneity. In order to understand OS installation concepts, file system tuning, and other low-level principles, students need to perform these tasks themselves. Learning to debug network connectivity issues or being able to actually see the payload of captured network traffic requires access to raw sockets, which the staff responsible for the security of the campus network would certainly rather not provide.

1.5 The Future of System Administration

The System Administrator's job definition is diverse, and so is the rather significant change the profession has undergone over the years. Gone are the days of shuffling punch cards, of cycling magentic tapes, of connecting your dumb terminals using RS-232 serial connections, of stringing ribbon cables inside large computer cases... or are they?

Some things haven't really changed all that much. Technologies come and go, but certain principles have remained the same. Using fibre-optic cables rather than twisted-pair, using Infiniband or Fibre Channel instead of Ethernet or parallel SCSI, or using wireless networks instead of physically connected systems does not fundamentally change the day-to-day operations.

Virtualization and Cloud Computing may seem to limit the need of an organization to hire their own System Administrators, but I believe that rather than a threat to the profession these technologies are simply one of the ways in which we make progress as information technology specialists.

Unix Introduction

2.1 What is Unix?

UNIX is an operating system which was first developed in the 1960s, and has been under constant development ever since. By operating system, we mean the suite of programs which make the computer work. It is a stable, multi-user, multi-tasking system for servers, desktops and laptops.

UNIX systems also have a graphical user interface (GUI) similar to Microsoft Windows which provides an easy to use environment. However, knowledge of UNIX is required for operations which aren't covered by a graphical program, or for when there is no windows interface available, for example, in a telnet session.

Years ago the name UNIX referred to a single operating system, but it is now used to refer to a family of operating systems that are offshoots of the original in terms of their user interfaces. Éric Lévénez (www.levenez.com/unix) lists names of over 270 UNIX flavors at the time of writing this book. Some of the members of this family are AIX, BSD, DYNIX,FreeBSD, HP-UX, Linux, MINIX, NetBSD, SCO, Solaris, OpenSolaris, SunOS, System V, XENIX, PC-BSD, OpenBSD, Mac OS X (Darwin), and XINU.

2.2 Types of Unix

There are many different versions of UNIX, although they share common similarities. The most popular varieties of UNIX are Sun Solaris, GNU/Linux, and MacOS X.

2.3 UNIX Software Architecture

The figure 1 shows a layered diagram for a UNIX-based computer system, identifying the system’s software components and their logical proximity to the user and hardware. We briefly describe each software layer from the bottom up.

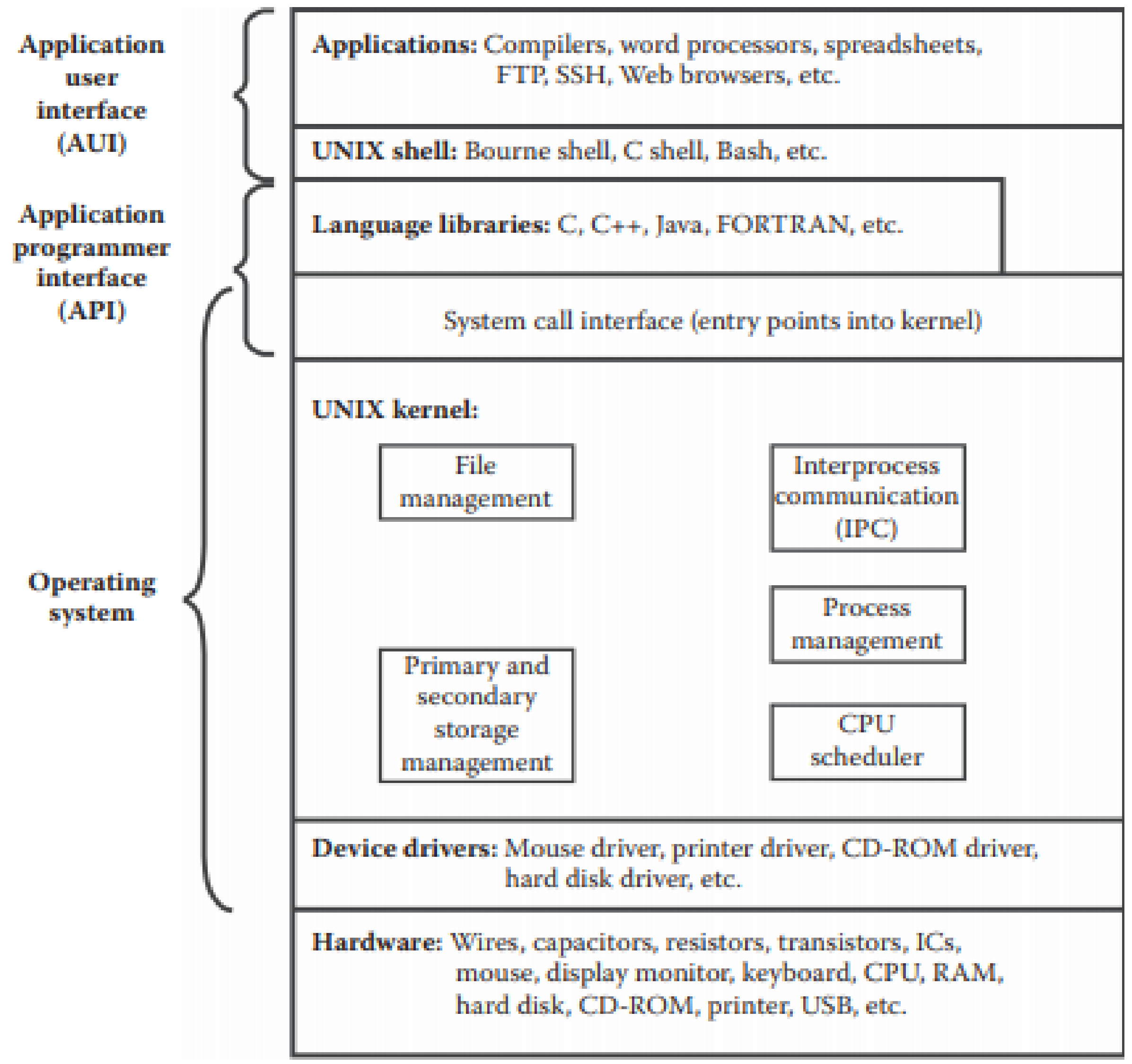

Application user interface (AUI)
Application programmer interface (API)
Operating system
Applications: Compilers, word processors, spreadsheets, FTP, SSH, Web browsers, etc.
UNIX shell: Bourne shell, C shell, Bash, etc.
Language libraries: C, C++, Java, FORTRAN, etc.
System call interface (entry points into kernel)
UNIX kernel:
File management
Interprocess communication (IPC)
Process management
Primary and secondary storage management
CPU scheduler
Device drivers: Mouse driver, printer driver, CD-ROM driver, hard disk driver, etc.
Hardware: Wires, capacitors, resistors, transistors, ICs, mouse, display monitor, keyboard, CPU, RAM, hard disk, CD-ROM, printer, USB, etc.

Figure 1

2.3.1 Device Driver Layer

The purpose of the device driver layer is to interact with various hardware devices. It contains a separate program for interacting with each device, including the hard disk driver, floppy disk driver, CD-ROM driver, keyboard driver, mouse driver, touchpad driver, and display driver. These programs execute on behalf of the UNIX kernel when a user command or application needs to perform a hardware-related operation such as a file read that translates to one or more disk reads. The user doesn't have direct access to these programs and therefore can't execute them as commands.

2.3.2 UNIX Kernel

The UNIX kernel layer contains the actual operating system. Some of the main functions of the UNIX kernel, listed in Figure 1, are described in this section. In addition, the kernel performs several other tasks for fair, orderly, and safe use of the computer system. These tasks include managing the CPU, printers, and other I/O devices. The kernel ensures that no user process takes over the CPU forever, that multiple files are not printed on a printer simultaneously, and that a user cannot terminate another user's process.

2.3.2.1 Process Management

This part of the kernel manages processes in terms of creating, suspending, resuming, and terminating them, and maintaining their states. It also provides various mechanisms for processes to communicate with each other and schedules the CPU to execute multiple processes simultaneously in a time-sharing system. Interprocess communication (IPC) is the key to the client–server-based software that is the foundation for Internet applications, including Web browsing (HTTP), file transfer (FTP), and remote login (SSH). The UNIX system provides three primary IPC mechanisms/channels:

- Pipe: Two or more related processes running on the same computer can use a pipe as an IPC channel. Typically, these processes have a parent–child or sibling relationship. A pipe is a temporary channel that resides in the main memory and is created by the kernel, usually on behalf of the parent process.

• Named pipe: A named pipe, also known as a FIFO, is a permanent communication channel that resides on the disk and can be used for IPC by two or more related or unrelated processes running on the same computer.

• BSD socket: A BSD socket is also a temporary channel that allows two or more processes in a network (or on the Internet) to communicate, although processes on the same computer can also use them. Sockets were originally a part of the BSD UNIX only, but they are now available on almost every UNIX system. Internet software such as Web browsers, File Transfer Protocol (FTP), Secure Shell (SSH), and electronic mailers are implemented by using sockets. AT&T UNIX has a similar mechanism called the Transport Layer Interface (TLI).

2.3.2.2 File Management

This part of the kernel manages files and directories, also known as folders. It performs all file-related tasks, including file creation and removal, directory creation and removal, setting access privileges on files and directories, and maintaining their attributes, such as file size. A file operation usually requires manipulation of a disk. In a multiuser system, a user must never be allowed to manipulate a disk directly because it contains files belonging to other users, and user access to a disk poses a security threat. Only the kernel must perform all file-related operations, such as file removal. Also, only the kernel must decide where and how much space to allocate to a file.

2.3.2.3 Main Memory Management

This part of the kernel allocates and deallocates RAM in an orderly manner so that each process has enough space to execute properly. It also ensures that part or all of the space allocated to a process does not belong to some other process. The space allocated to a process in the memory for its execution is known as its address space. The kernel ensures that no process accesses an area of memory that does not belong to its address space. The kernel maintains areas in the main memory that are free to be allocated to processes. The kernel code that performs this task is called the free space manager. When a program is to be loaded in the main memory, the free space manager allocates adequate space for it and the loader loads the program into this space. The kernel also records where all the processes reside in the memory so that, when a process tries to access main memory space that does not belong to it, the kernel can terminate the process and give a meaningful message to the user. When a process terminates, the kernel deallocates the space allocated to the process and puts it back in the free space pool so that it can be reused.

2.3.2.4 Disk Management

The kernel is also responsible for maintaining free and used disk space and for the orderly and fair allocation and deallocation of disk space. It decides where and how much space to allocate to a newly created file. The kernel code that performs this task is known as the disk storage manager. Also, the kernel performs disk scheduling, deciding which request to serve next when multiple requests for file read, write, and so on, arrive for the same disk.

2.3.3 System Call Interface

The system call interface layer contains entry points into the kernel code. Because the kernel manages all system resources, any user or application request that involves access to any system resource must be handled by the kernel code. But user processes must not be given open access to the kernel code for security reasons. So that user processes can invoke the execution of kernel code, UNIX provides several openings, or function calls, into the kernel, known as system calls. There are numerous system calls that deal with the manipulation of processes, files, and other system resources. These calls are well tested, and most of them have been used for several years, so their use poses much less of a security risk than if any user code were allowed to perform the task.

2.3.4 Language Libraries

A language library is a set of prewritten and pretested functions in a programming language available to programmers for use with the software that they develop. The availability and use of libraries saves time because programmers do not have to write these functions from scratch. This layer contains libraries for several languages, including C, C++, C#, Java, Perl, and Python. For the C language, for example, there are several libraries, including a string library (which contains functions for processing strings, such as a function for comparing two strings) and a math library (which contains functions for mathematical operations, such as finding the cosine of an angle).

As we stated earlier in this chapter, the libraries and system call interface form what is commonly known as the API. In other words, programmers who write software in a language such as C can use in their code the prewritten functions available in the various C libraries and system calls.

2.3.5 UNIX Shell

The UNIX shell is a program that starts running when you log on and interprets the commands that you enter. The most popular shells are the Bourne shell (sh), Bourne Again shell (bash), C shell (csh), TC shell (tcsh), and Korn shell (ksh). We show the usage of shell commands and shell scripts in Bourne and C shells.

2.3.6 Applications

The applications layer contains all the applications (tools, commands, and utilities) that are available for your use. A typical UNIX system contains hundreds of applications; we discuss the most useful and commonly used applications throughout this textbook. When an application that you're using needs to manipulate a system resource (e.g., reading a file), it needs to invoke some kernel code that performs the task. An application can find the appropriate kernel code to execute in one of two ways: (1) by using a proper library function and (2) by using a system call. Library calls constitute a higher-level interface to the kernel than system calls, which makes library calls a bit easier to use. However, all library calls eventually use system calls to begin execution of the appropriate kernel code. Therefore, the use of library calls in software results in slightly slower execution.

3 Storage Models

3.1 Introduction

This chapter deals primarily with how we store data. Virtually all computer systems require some way to store data permanently; even so-called “diskless” systems do require access to certain files in order to boot, run and be useful. Albeit stored remotely (or in memory), these bits reside on some sort of storage system.

Most frequently, data is stored on local hard disks, but over the last few years more and more of our files have moved “into the cloud”, where different providers offer easy access to large amounts of storage over the network. We have more and more computers depending on access to remote systems, shifting our traditional view of what constitutes a storage device.

As system administrators, we are responsible for all kinds of devices: we build systems running entirely without local storage just as we maintain the massive enterprise storage arrays that enable decentralized data replication and archival. We manage large numbers of computers with their own hard drives, using a variety of technologies to maximize throughput before the data even gets onto a network.

In order to be able to optimize our systems on this level, it is important for us to understand the principal concepts of how data is stored, the different storage models and disk interfaces. It is important to be aware of certain physical properties of our storage media, and the impact they, as well as certain historic limitations, have on how we utilize disks.

Available storage space is, despite rapidly falling prices for traditional hard disk drives, a scarce resource . The quote at the beginning of this chapter is rather apt: no matter how much disk space we make available to our users, we will eventually run out and need to expand. In order to accommodate the ever-growing need for storage space, we use technologies such as Logical Volume Management to combine multiple physical devices in a flexible manner to present a single storage container to the operating system. We use techniques such as RAID to increase capacity, resilience, or performance (pick two!), and separate data from one another within one storage device using partitions. Finally, before we can actually use the disk devices to install an operating system or any other software, we create a file system on top of these partitions.

System administrators are expected to understand well all of these topics. Obviously, each one can (and does) easily fill many books; in this chapter, we will review the most important concepts underlying the different technologies from the bottom up to the file system level. At each point, we will compare and contrast traditional systems with recent developments, illustrating how the principles, even if applied differently, remain the same. For significantly deeper discussions and many more details, please see the chapter references, in particular the chapters on file systems in Silberschatz and McKusick et al.'s canonical paper on the Berkeley Fast File System

3.2 Storage Models

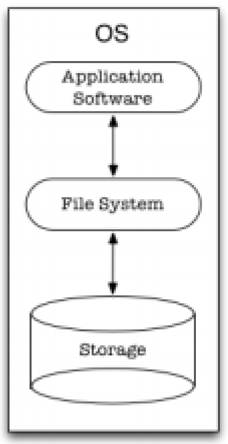
We distinguish different storage models by how the device in charge of keeping the bits in place interacts with the higher layers: by where raw block device access is made available, by where a file system is created to make available the disk space as a useful unit, by which means and protocols the operating system accesses the file system. Somewhat simplified, we identify as the main three components the storage device itself, i.e. the actual medium; the file system, providing access to the block level storage media to the operating system; and finally the application software. The operating system managing the file system and running the application software then acts as the agent making actual I/O possible.

3.2.1 Direct Attached Storage

The by far most common way to access storage is so simple that we rarely think about it as a storage model: hard drives are attached (commonly via a host bus adapter and a few cables) directly to the server, the operating system detects the block devices and maintains a file system on them and thus allows for access with the smallest level of indirection. The vast majority of hosts (laptops, desktop and server systems alike) all utilize this method. The term used nowadays – Direct Attached Storage (DAS) – was effectively created only after other approaches became popular enough to require a simple differentiating name.

Figure **3.1a** illustrates this model: all interfacing components are within the control of a single server's operating system (and frequently located within the same physical case) and multiple servers each have their own storage system. On the hardware level, the storage media may be attached using a variety of technologies, and we have seen a number of confusing standards come and go over the years. The best choice here depends on many factors, including the number of devices to be attached, driver support for the connecting interface in the OS, and performance or reliability considerations. We will touch upon all of these aspects throughout this chapter.

A server with a single hard drive (such as the one shown in Figure **3.1b** is perhaps the simplest application of this model. But it is not uncommon for servers to have multiple direct attached devices, the configuration of which then depends entirely on the next higher level of storage strategies. Individual disks can simply be mounted in different locations of the file system hierarchy



(a) Diagram

(b) A Maxtor IDE Drive

Figure 3.1

Alternatively, multiple direct attached disks can be combined to create a single logical storage unit through the use of a Logical Volume Manager (LVM) or a Redundant Array of Independent Disks (RAID). This allows for improved performance, increased amount of storage and/or redundancy.

Direct attached storage need not be physically located in the same case (or even rack) as the server using it. That is, we differentiate between internal storage (media attached inside the server with no immediate external exposure) and external storage (media attached to a server's interface ports, such as Fibre Channel, USB etc.) with cables the lengths of which depend on the technology used. External media allows us to have large amounts of storage housed in a separate enclosure with its own power supply, possibly located several feet away from the server. If a server using these disks suffers a hardware failure, it becomes significantly easier to move the data to another host: all you need to do is connect the cable to the new server.

Simple as this architecture is, it is also ubiquitous. The advantages of DAS should be obvious: since there is no network or other additional layer in between the operating system and the hardware, the possibility of failure on that level is eliminated. Likewise, a performance penalty due to network latency, for example, is impossible. As system administrators, we frequently need to carefully eliminate possible causes of failures, so the fewer layers of indirection we have between the operating system issuing I/O operations and the bits actually ending up on a storage medium, the better.

At the same time, there are some disadvantages. Since the storage media is, well, directly attached, it implies a certain isolation from other systems on the network. This is both an advantage as well as a drawback: on the one hand, each server requires certain data to be private or unique to its operating system; on the other hand, data on one machine cannot immediately be made available to other systems. This restriction is overcome with either one of the two storage models we will review next: Network Attached Storage (NAS) and Storage Area Networks (SANs).

DAS can easily become a shared resource by letting the operating system make available a local storage device over the network. In fact, all network file servers and appliances ultimately are managing direct attached storage on behalf of their clients; DAS becomes a building block of NAS. Likewise, physically separate storage enclosures can function as DAS if connected directly to a server or may be combined with others and connected to network or storage fabric, that is: they become part of a SAN.

3.2.2 Network Attached Storage

As the need for more and more data arises, we frequently want to be able to access certain data from multiple servers. An old and still very common example is to store all your users' data on shared disks that are made available to all clients over the network. When a user logs into hostA, she expects to find all her files in place just as when she logs into hostB. To make this magic happen, two things are required: (1) the host's file system has to know how to get the data from a central location and (2) the central storage has to be accessible over the network. As you can tell, this introduces a number of complex considerations, not the least of which are access control and performance.

For the moment, let us put aside these concerns, however, and look at the storage model from a purely architectural point of view: One host functions as the "file server", while multiple clients access the file system over the network. The file server may be a general purpose Unix system or a special network appliance – either way, it provides access to a number of disks or other storage media, which, within this system are effectively direct attached storage. In order for the clients to be able to use the server's file system remotely, they require support for (and have to be in agreement with) the protocols used . However, the clients do not require access to the storage media on the block level; in fact, they cannot gain such access.

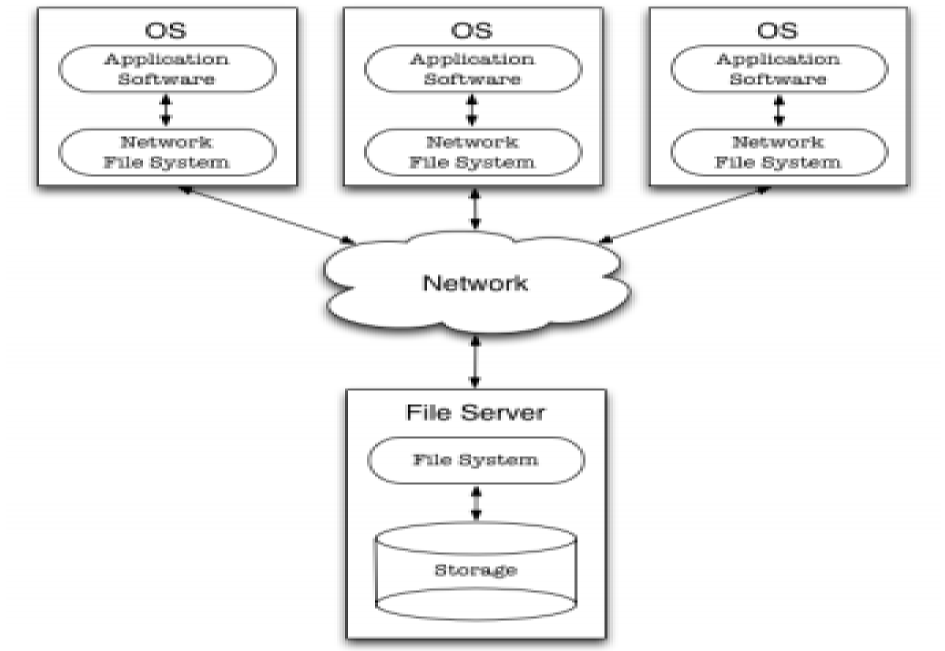


Figure 3.2 Three hosts using Network Attached Storage, or NAS

From the clients' perspective, the job of managing storage has become simpler: I/O operations are performed on the file system much as they would be on a local file system, with the complexity of how to shuffle the data over the network being handled in the protocol in question. This model is illustrated in Figure 3.2, albeit in a somewhat simplified manner: even though the file system is created on the file server, the clients still require support for the network file system that brokers the transaction performed locally with the file server.

In contrast to DAS, a dedicated file server generally contains significantly more and larger disks; RAID or LVM may likewise be considered a requirement in this solution, so as to ensure both performance and failover. Given the additional overhead of transferring data over the network, it comes as no surprise that a certain performance penalty (mainly due to network speed or congestion) is incurred. Careful tuning of the operating system and in particular the network stack, the TCP window size, and the buffer cache can help minimize this cost.

The benefits of using a central file server for data storage are immediate and obvious: data is no longer restricted to a single physical or virtual host and can be accessed (simultaneously) by multiple clients. By pooling larger resources in a dedicated NAS device, more storage becomes available. Other than the performance impact we mentioned above, the distinct disadvantage lies in the fact that the data becomes unavailable if the network connection suffers a disruption. In many environments, the network connection can be considered sufficiently reliable and persistent to alleviate this concern. However, such solutions are less suitable for mobile clients, such as laptops or mobile devices, which frequently may disconnect from and reconnect to different networks. Recent developments in the area of Cloud Storage have provided a number of solutions , but it should be noted that mitigation can also be found in certain older network file systems and protocols: the Andrew File System (AFS), for example, uses a local caching mechanism that lets it cope with the temporary loss of connectivity without blocking.

While network attached storage is most frequently used for large, shared partitions or data resources, it is possible to boot and run a server entirely without any direct attached storage. In this case, the entire file system, operating system kernel and user data may reside on the network. We touch on this special setup in future chapters.

3.2.3 Storage Area Networks

Network Attached Storage (NAS) allows multiple clients to access the same file system over the network, but that means it requires all clients to use specifically this file system. The NAS file server manages and handles the creation of the file systems on the storage media and allows for shared access, overcoming many limitations of direct attached storage. At the same time, however, and especially as we scale up, our requirements with respect to

(a) Huawei Tecal RH 2288H V2

(b) NetApp FAS 3050c

Figure 3.3: NAS and SAN Hardware

NAS and SAN enterprise hardware. On the left, Huawei storage servers with 24 hard drives each and built-in hardware RAID controllers; on the right, a NetApp Fabric Attached Storage device, also known as a “filer” with disk enclosures commonly referred to as “shelves”.

storage size, data availability, data redundancy, and performance, it becomes desirable to allow different clients to access large chunks of storage on a block level. To accomplish this, we build high performance networks specifically dedicated to the management of data storage: Storage Area Networks.

In these dedicated networks, central storage media is accessed using high performance interfaces and protocols such as Fibre Channel or iSCSI, making the exposed devices appear local to the clients. As you can tell, the boundaries between these storage models are not rigid: a single storage device connected via Fibre Channel to a single host (i.e. an example of DAS) is indistinguishable (to the client) from a dedicated storage device made available over a Storage Area Network (SAN). In fact, today’s network file servers frequently manage storage made available to them over a SAN to export a file system to the clients as NAS.

Figure 3.4 illustrates how the storage volumes managed within a SAN can be accessed by one host as if it was direct attached storage while other parts are made available via a file server as NAS to different clients. In order for the

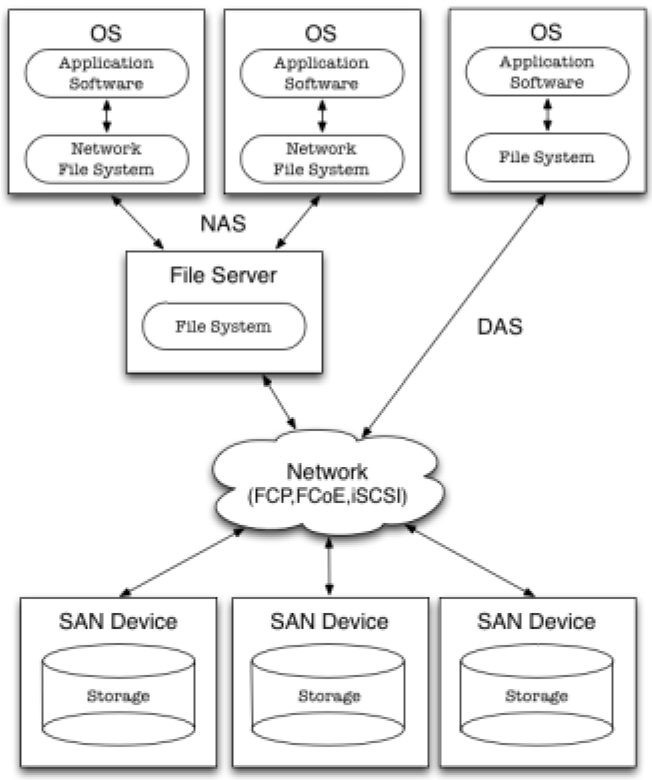


Figure 3.4: A SAN providing access to three devices; one host accesses parts of the available storage as if it was DAS, while a file server manages other parts as NAS for two clients.

different consumers in a SAN to be able to independently address the distinct storage units, each is identified by a unique Logical Unit Number (LUN). The system administrator combines the individual disks via RAID, for example, into separate volumes; assigning to each storage unit an independent LUN allows for correct identification by the clients and prevents access of data by unauthorized servers, an important security mechanism. Fibre Channel switches used in SANs allow further partitioning of the fabric by LUNs and subdivision into SAN Zones, allowing sets of clients specifically access to “their” storage device only In this storage model the clients – the computers, file servers or other devices directly attached to the SAN – are managing the volumes on a block level.That is, they need to create a logical structure on top of the block devices (as which the SAN units appear), and they control all aspects of the I/O operations down to the protocol. With this low-level access, clients can treat the storage like any other device. In particular, they can boot off SAN attached devices, they can partition the volumes, create different file systems for different purposes on them and export them via other protocols.

Storage area networks are frequently labeled an “enterprise solution” due to their significant performance advantages and distributed nature. Especially when used in a switched fabric, additional resources can easily be made available to all or a subset of clients. These networks utilize the Small Computer System Interface (SCSI) protocol for communications between the different devices; in order to build a network on top of this, an additional protocol layer – the Fibre Channel Protocol (FCP) being the most common one – is required.

SANs overcome their restriction to a local area network by further encapsulation of the protocol: Fibre Channel over Ethernet (FCoE) or iSCSI, for example, allow connecting switched SAN components across a Wide Area Network (or WAN). But the concept of network attached storage devices facilitating access to a larger storage area network becomes less accurate when end users require access to their data from anywhere on the Internet. Cloud storage solutions have been developed to address these needs. However, as we take a closer look at these technologies, it is important to remember that at the end of the day, somewhere a system administrator is in charge of making available the actual physical storage devices underlying these solutions. Much like a file server may provide NAS to its clients over a SAN, so do cloud storage solutions provide access on “enterprise scale” (and at this size the use of these words finally seems apt) based on the foundation of the technologies we discussed up to here.

3.2.4 Cloud Storage

In the previous sections we have looked at storage models that ranged from the very simple and very local to a more abstracted and distributed approach, to a solution that allows access across even a Wide Area Network (WAN). At each step, we have introduced additional layers of indirection with the added benefit of being able to accommodate larger requirements: more clients, more disk space, increased redundancy, etc.

We also have come full circle from direct attached storage providing block level access, to distributed file systems, and then back around to block-level access over a dedicated storage network. But this restricts access to clients on this specific network. As more and more (especially smaller or mid-sized) companies are moving away from maintaining their own infrastructure towards a model of Infrastructure as a Service (IaaS) and Cloud Computing, the storage requirements change significantly, and we enter the area of Cloud Storage.

The term “cloud storage” still has a number of conflicting or surprisingly different meanings. On the one hand, we have commercial services offering file hosting or file storage services; common well-known providers currently include Dropbox, Google Drive, Apple’s iCloud and Microsoft’s SkyDrive. These services offer customers a way to not only store their files, but to access them from different devices and locations: they effectively provide network attached storage over the largest of WANs, the Internet.

On the other hand we have companies in need of more flexible storage solutions than can be provided with the existing models. Especially the increased use of virtualization technologies demands faster and more flexible access to reliable, persistent yet relocatable storage devices. In order to meet these requirements, storage units are rapidly allocated from large storage area networks spanning entire data centers.

Since the different interpretations of the meaning of “cloud storage” yield significantly different requirements, the implementations naturally vary, and there are no current industry standards defining an architecture. As such, we are forced to treat each product independently as a black box; system administrators and architects may choose to use any number of combinations of the previously discussed models to provide the storage foundation upon which the final solution is built.

We define three distinct categories within this storage model: (1) services that provide file system level access as in the case of file hosting services such as those mentioned above; (2) services that provide access on the object level, hiding file system implementation details from the client and providing for easier abstraction into an API and commonly accessed via web services such as Amazon’s Simple Storage Service (S3), or Windows Azure’s Blob Storage; and (3) services that offer clients access on the block level, allowing them to create file systems and partitions as they see fit (examples include Amazon’s Elastic Block Store (EBS) and OpenStack’s Cinder service).

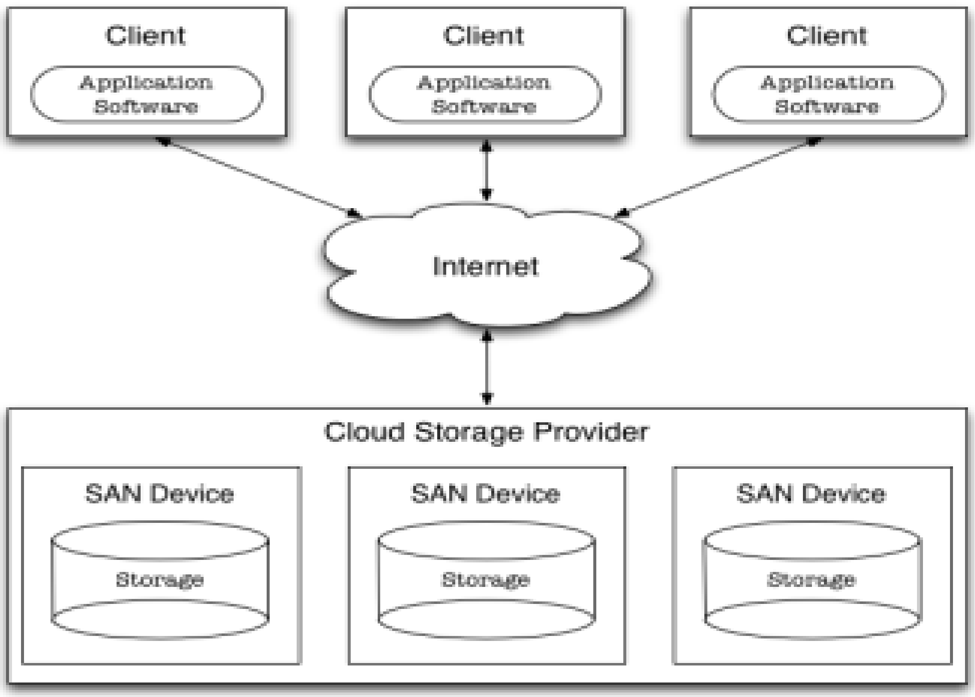


Figure 3.5: A possible cloud storage model: an internal SAN is made available over the Internet to multiple clients. In this example, the storage provider effectively functions as a NAS server, though it should generally be treated as a black box.

as Amazon's Simple Storage Service (S3), or Windows Azure's Blob Storage; and (3) services that offer clients access on the block level, allowing them to create file systems and partitions as they see fit (examples include Amazon's Elastic Block Store (EBS) and OpenStack's Cinder service).

All of these categories have one thing in common, however. In order to provide the ability of accessing storage units in a programmatic way – a fundamental requirement to enable the flexibility needed in demanding environments – they rely on a clearly specified API. Multiple distributed resources are combined to present a large storage pool, from which units are allocated, de-allocated, re-allocated, relocated, and duplicated, all by way of higher-level programs using well-defined interfaces to the lower-level storage systems.

Customers of cloud storage solution providers reap a wealth of benefits, including: their infrastructure is simplified through the elimination of storage components; storage units are almost immediately made available as needed and can grow or shrink according to immediate or predicted usage patterns;applications and entire OS images can easily be deployed, imported or exported as virtual appliances.

Of course these benefits carry a cost. As usual, any time we add layers of abstraction we also run the risk of increasing, possibly exponentially, the number of ways in which a system can fail. Cloud storage is no exception: by relying on abstracted storage containers from a third-party provider, we remove the ability to troubleshoot a system end-to-end; by outsourcing data storage, we invite a number of security concerns regarding data safety and privacy; by accessing files over the Internet, we may increase latency and decrease throughput; the cloud service provider may become a single point of failure for our systems, one that is entirely outside our control.

3.2.5 Storage Model Considerations

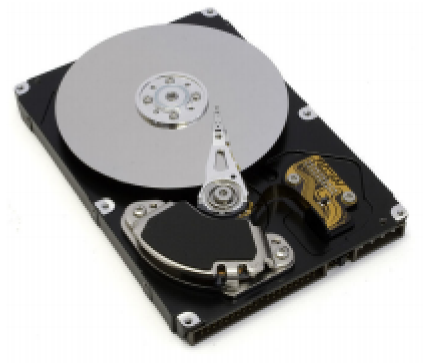
As we have seen in the previous sections, the larger we grow our storage requirements, the more complex the architecture grows. It is important to keep this in mind: even though added layers of abstraction and indirection help us scale our infrastructure, the added complexity has potentially exponential costs. The more moving parts a system has, the more likely it is to break, and the more spectacular its failure will be.

A single bad hard drive is easy to replace; rebuilding the storage array underneath hundreds of clients much less so. The more clients we have, the more important it is to build our storage solution for redundancy as well as reliability and resilience.

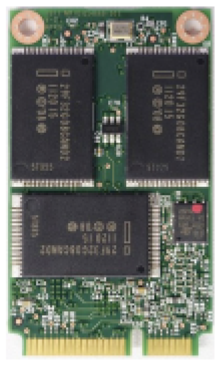
System administrators need to understand all of the storage models we discussed, as they are intertwined: DAS must eventually underlie any storage solution, since the bits do have to be stored somewhere after all; the concepts of NAS permeate any infrastructure spanning more than just a few work stations, and SANs and cloud storage combine DAS and NAS in different ways to make storage available over complex networks.

At each layer, we introduce security risks, of which we need to be aware: any time bits are transferred over a network, we need to consider the integrity and privacy of the files: who has access, who should have access, how is the access granted, how are clients authenticated, and so on. NAS and SAN solutions tend to ignore many of these implications and work under the assumption that the network, over which the devices are accessed, are “secure”; access controls are implemented on a higher layer such as the implementation of the file system. Oftentimes, access to the network in question implies.

access to the shared storage, even though layer-2 security mechanisms such as IPsec may be combined with or integrated into the solution. Cloud storage, on the other hand, has to directly address the problem of transmitting data and providing access over untrusted networks and thus usually relies on application layer protocols such as Transport Layer Security (TLS)/Secure Sockets Layer (SSL).



(a) Open Hard Disc Drive



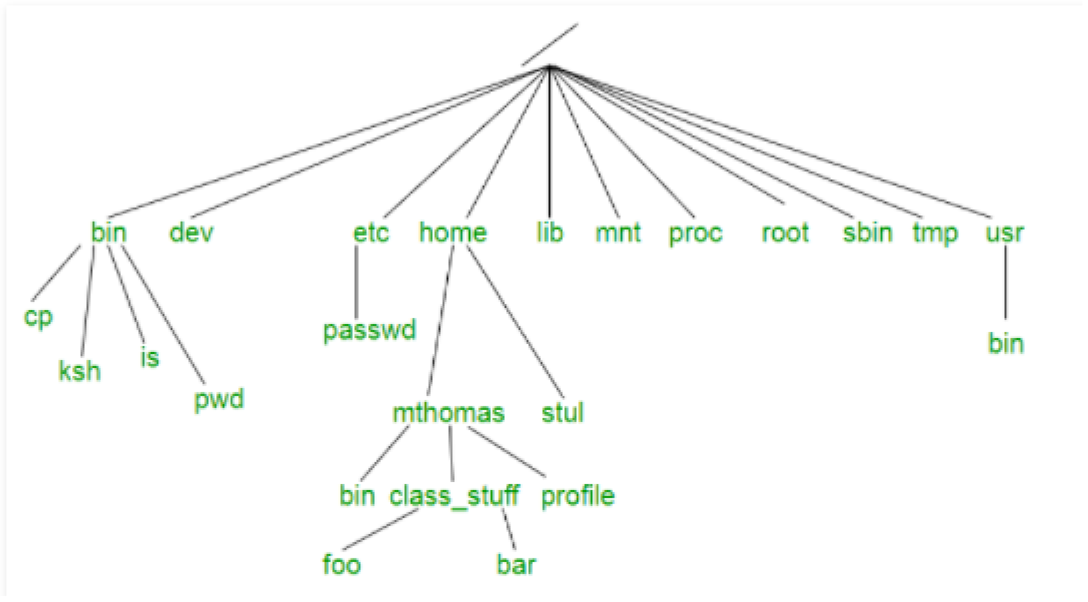
(b) SSD Mini PCIe Card

Figure 3.6: An open PATA (or IDE) hard drive (left) and a Solid State Drive (right). The HDD shows the rotating disk platters, the read-write head with its motor, the disk controller and the recognizable connector socket.

4 Unix File System

Unix file system is a logical method of **organizing and storing** large amounts of information in a way that makes it easy to manage. A file is a smallest unit in which the information is stored. The Unix file system has several important features. All data in Unix is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the file system.

Files in the Unix System are organized into a multi-level hierarchy structure known as a directory tree. At the very top of the file system is a directory called “root” which is represented by a “/”. All other files are “descendants” of root.



4.1 Directories or Files and their description –

- **/ :** The slash / character alone denotes the root of the filesystem tree.
- **/bin :** Stands for "binaries" and contains certain fundamental utilities, such as ls or cp, which are generally needed by all users.
- **/boot :** Contains all the files that are required for a successful booting process.
- **/dev :** Stands for "devices". Contains file representations of peripheral devices and pseudo-devices.
- **/etc :** Contains system-wide configuration files and system databases. Originally also contained "dangerous maintenance utilities" such as init,but these have typically been moved to /sbin or elsewhere.

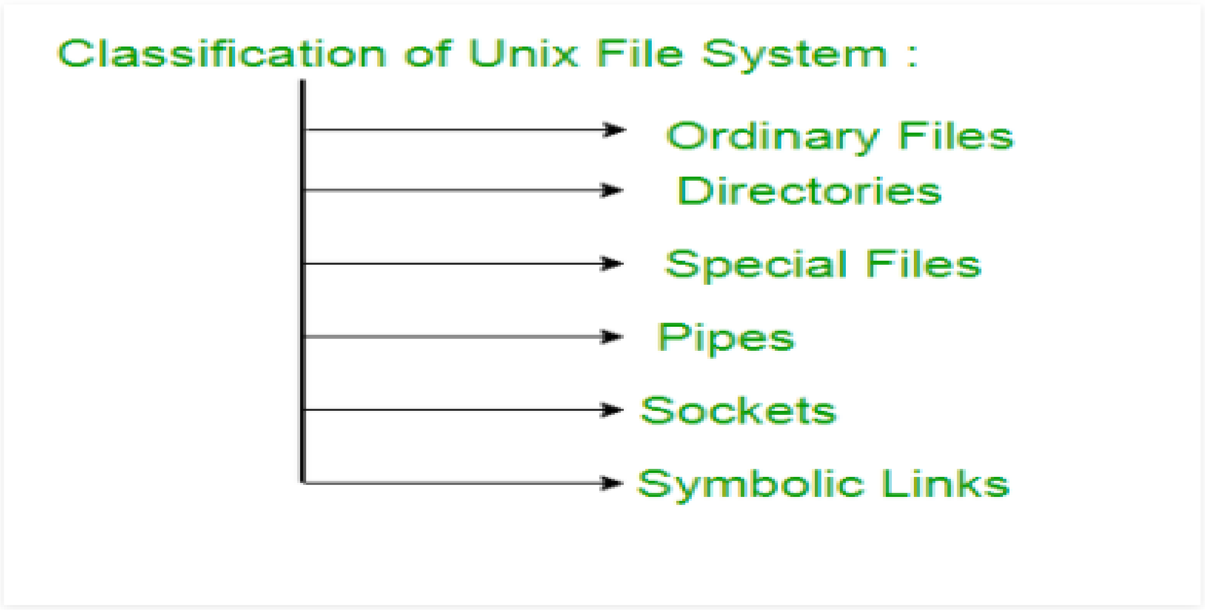
- **/home :** Contains the home directories for the users.
- **/lib :** Contains system libraries, and some critical files such as kernel modules or device drivers.
- **/media :** Default mount point for removable devices, such as USB sticks, media players, etc.
- **/mnt :** Stands for "mount". Contains filesystem mount points. These are used, for example, if the system uses multiple hard disks or hard disk partitions. It is also often used for remote (network) filesystems, CD-ROM/DVD drives, and so on.
- **/proc :** procfs virtual filesystem showing information about processes as files.
- **/root :** The home directory for the superuser "root" – that is, the system administrator. This account's home directory is usually on the initial filesystem, and hence not in /home (which may be a mount point for another filesystem) in case specific maintenance needs to be performed, during which other filesystems are not available. Such a case could occur, for example, if a hard disk drive suffers physical failures and cannot be properly mounted.
- **/tmp :** A place for temporary files. Many systems clear this directory upon startup; it might have tmpfs mounted atop it, in which case its contents do not survive a reboot, or it might be explicitly cleared by a startup script at boot time.
- **/usr :** Originally the directory holding user home directories,its use has changed. It now holds executables, libraries, and shared

resources that are not system critical, like the X Window System, KDE, Perl, etc. However, on some Unix systems, some user accounts may still have a home directory that is a direct subdirectory of /usr, such as the default as in Minix. (on modern systems, these user accounts are often related to server or system use, and not directly used by a person).

- **/usr/bin :** This directory stores all binary programs distributed with the operating system not residing in /bin, /sbin or (rarely) /etc.
- **/usr/include :** Stores the development headers used throughout the system. Header files are mostly used by the **#include** directive in C/C++ programming language.
- **/usr/lib :** Stores the required libraries and data files for programs stored within /usr or elsewhere.
- **/var :** A short for “variable.” A place for files that may change often – especially in size, for example email sent to users on the system, or process-ID lock files.
- **/var/log :** Contains system log files.
- **/var/mail :** The place where all the incoming mails are stored. Users (other than root) can access their own mail only. Often, this directory is a symbolic link to /var/spool/mail.
- **/var/spool :** Spool directory. Contains print jobs, mail spools and other queued tasks.

- **/var/tmp :** A place for temporary files which should be preserved between system reboots.

4.2 Types of Unix files – The UNIX files system contains several different types of files :



1. Ordinary files – An ordinary file is a file on the system that contains data, text, or program instructions.

- Used to store your information, such as some text you have written or an image you have drawn. This is the type of file that you usually work with.
- Always located within/under a directory file.

- Do not contain other files.
- In long-format output of ls -l, this type of file is specified by the "-" symbol.

2. Directories – Directories store both special and ordinary files. For users familiar with Windows or Mac OS, UNIX directories are equivalent to folders. A directory file contains an entry for every file and subdirectory that it houses. If you have 10 files in a directory, there will be 10 entries in the directory. Each entry has two components.

(1) The Filename

(2) A unique identification number for the file or directory (called the inode number)

Branching points in the hierarchical tree.
Used to organize groups of files.
May contain ordinary files, special files or other directories.
Never contain "real" information which you would work with (such as text). Basically, just used for organizing files.
All files are descendants of the root directory, (named /) located at the top of the tree.
In long-format output of ls –l , this type of file is specified by the "d" symbol.
3. Special Files – Used to represent a real physical device such as a printer, tape drive or terminal, used for Input/Output (I/O) operations.
Device or special files are used for device Input/Output(I/O) on UNIX and Linux systems. They appear in a file system just like an ordinary file or a directory.

On UNIX systems there are two flavors of special files for each device, character special files and block special files :

- When a character special file is used for device Input/Output(I/O), data is transferred one character at a time. This type of access is called raw device access.
- When a block special file is used for device Input/Output(I/O), data is transferred in large fixed-size blocks. This type of access is called block device access.

For terminal devices, it’s one character at a time. For disk devices though, raw access means reading or writing in whole chunks of data – blocks, which are native to your disk.

- In long-format output of ls -l, character special files are marked by the “c” symbol.
- In long-format output of ls -l, block special files are marked by the “b” symbol.

4. Pipes – UNIX allows you to link commands together using a pipe. The pipe acts as a temporary file which only exists to hold data from one command until it is read by another.A Unix pipe provides a one-way flow of data.The output or result of the first command sequence is used as the input to the second command sequence. To make a pipe, put a vertical bar (|) on the command line between two commands.For example: **who | wc -l**
In long-format output of ls –l , named pipes are marked by the “p” symbol.

5. Sockets – A Unix socket (or Inter-process communication socket) is a special file which allows for advanced inter-process communication. A Unix Socket is used in a client-server application framework. In essence, it is a stream of data, very similar to network stream (and network sockets), but all the transactions are local to the filesystem.
In long-format output of ls -l, Unix sockets are marked by the "s" symbol.
6. Symbolic Link – Symbolic link is used for referencing some other file of the file system.Symbolic link is also known as Soft link. It contains a text form of the path to the file it references. To an end user, symbolic link will appear to have its own name, but when you try reading or writing data to this file, it will instead reference these operations to the file it points to. If we delete the soft link itself , the data file would still be there.If we delete the source file or move it to a different location, the symbolic file will not function properly.
In long-format output of ls –l , Symbolic links are marked by the "l" symbol (that's a lower case L).