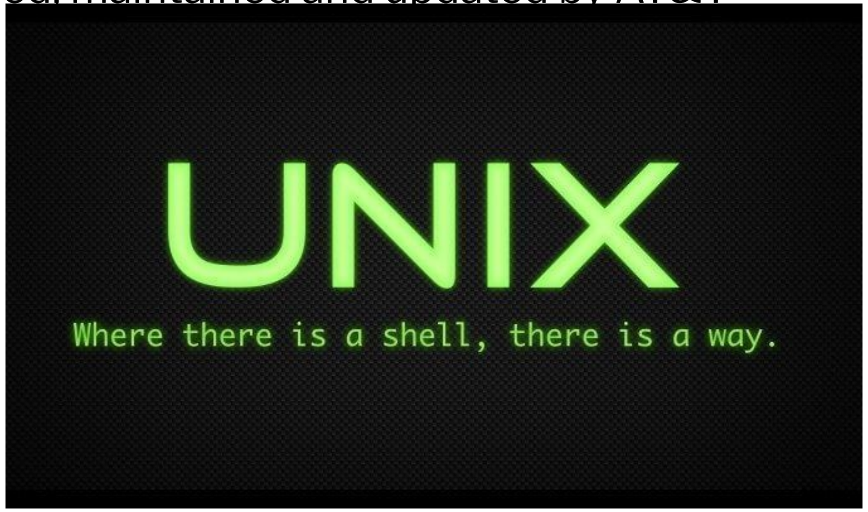
# INTRODUCTION TO UNIX

## INTRODUCTION TO UNIX

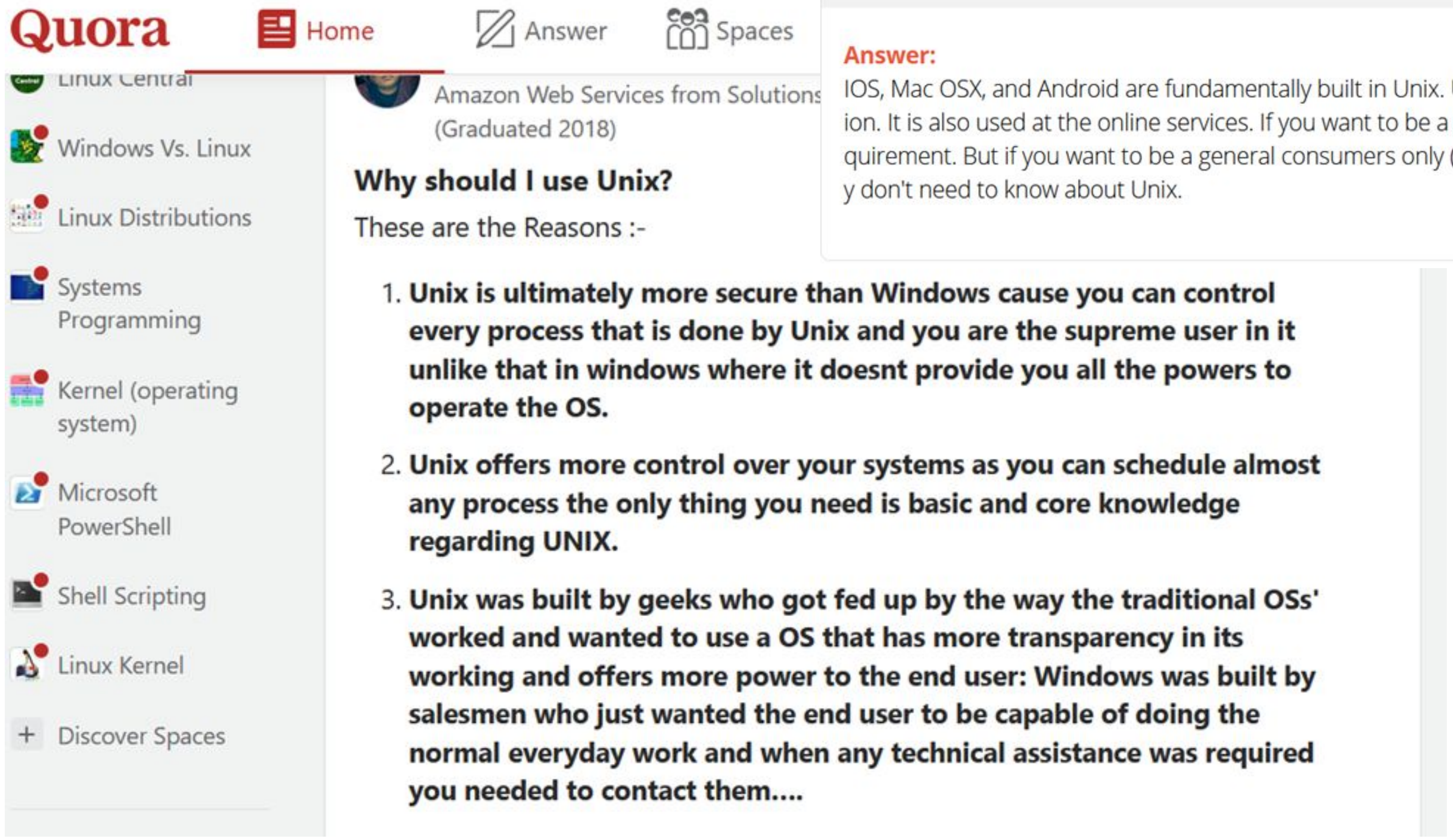
- Unix is a family of multitasking, multiuser computer operating systems that derive from the original AT&T Unix, developed in the 1970s at the Bell Labs research center by Ken Thompson, Dennis Ritchie, and others.
- Unix is a proprietary operating system commonly used in internet servers, workstations and PCs by Sun Microsystems, Intel, HP etc.
- The Unix OS is mainly used on large server systems, mainframes, expensive and high-end computer systems at big MNCs and institutions. Unix is being developed, maintained and updated by AT&T developers.
- Different versions of Unix are as follows:
  - AIS (IBM)
  - BSD (University of California)
  - HP – UX (HP)
  - Solaris (Sun Microsystems)



# INTRODUCTION TO UNIX

- The Unix OS works primarily on Command Line Interface, though, recently, there have been developments for GUI on Unix systems.
- Unix is not free. Different flavors of Unix have different cost structures according to vendors.
- Unix is not as flexible as Linux. It has less compatibility with different types of hardware.
- Unix installation requires a strict and well-defined hardware machinery and works only on specific CPU machines. It is mostly used in big data servers around the world.

# WHY DO WE WANT TO LEARN UNIX?


Quora Home Answer Spaces

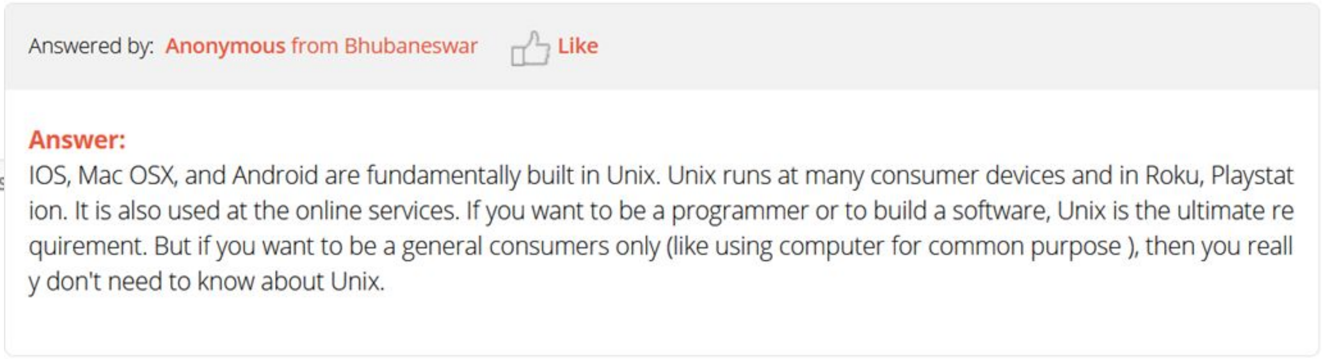
Linux Central
Windows Vs. Linux
Linux Distributions
Systems Programming
Kernel (operating system)
Microsoft PowerShell
Shell Scripting
Linux Kernel
+ Discover Spaces

Amazon Web Services from Solutions
(Graduated 2018)

**Why should I use Unix?**

These are the Reasons :-

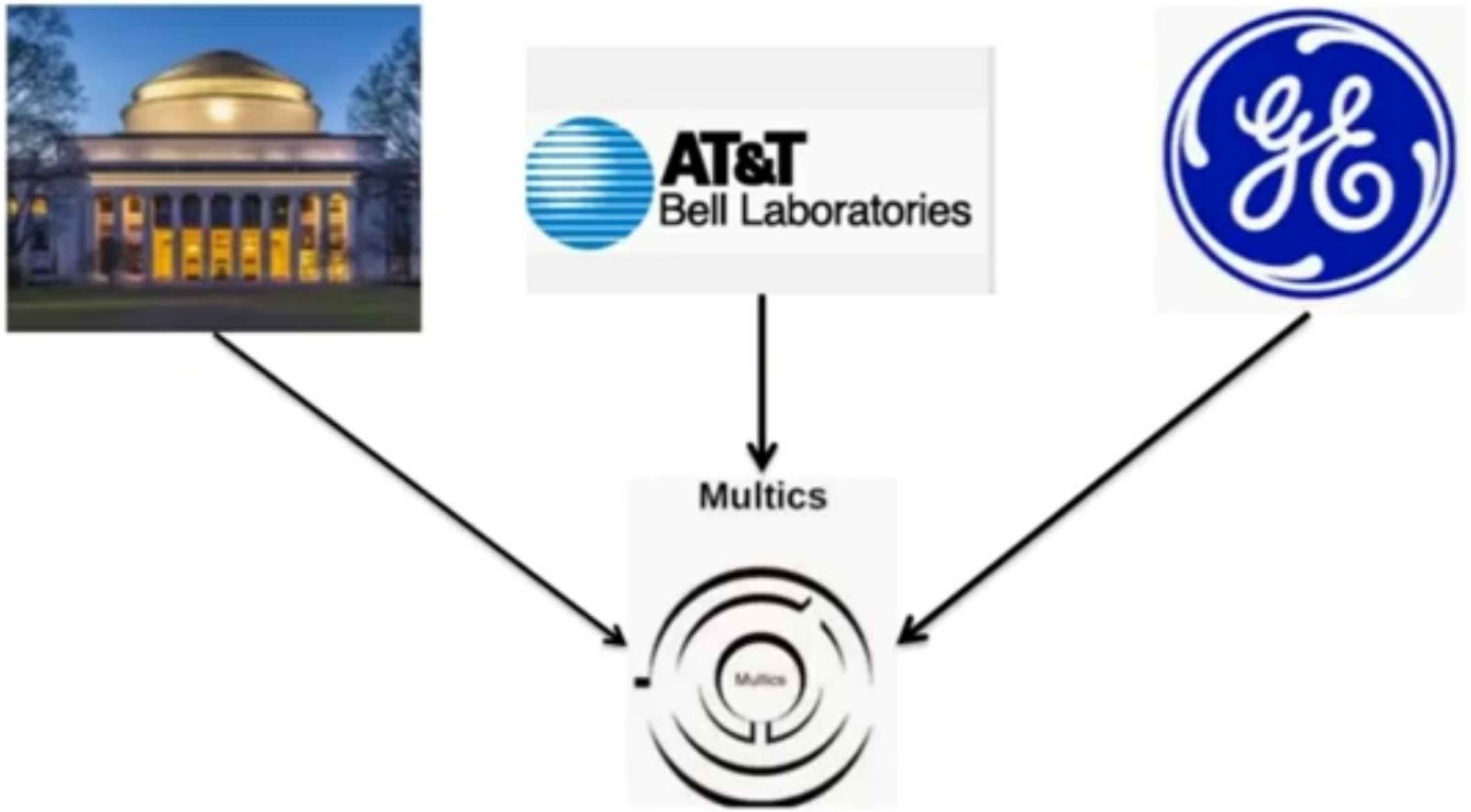
1. **Unix is ultimately more secure than Windows cause you can control every process that is done by Unix and you are the supreme user in it unlike that in windows where it doesnt provide you all the powers to operate the OS.**
2. **Unix offers more control over your systems as you can schedule almost any process the only thing you need is basic and core knowledge regarding UNIX.**
3. **Unix was built by geeks who got fed up by the way the traditional OSs' worked and wanted to use a OS that has more transparency in its working and offers more power to the end user: Windows was built by salesmen who just wanted the end user to be capable of doing the normal everyday work and when any technical assistance was required you needed to contact them....**


Answered by: Anonymous from Bhubaneswar Like

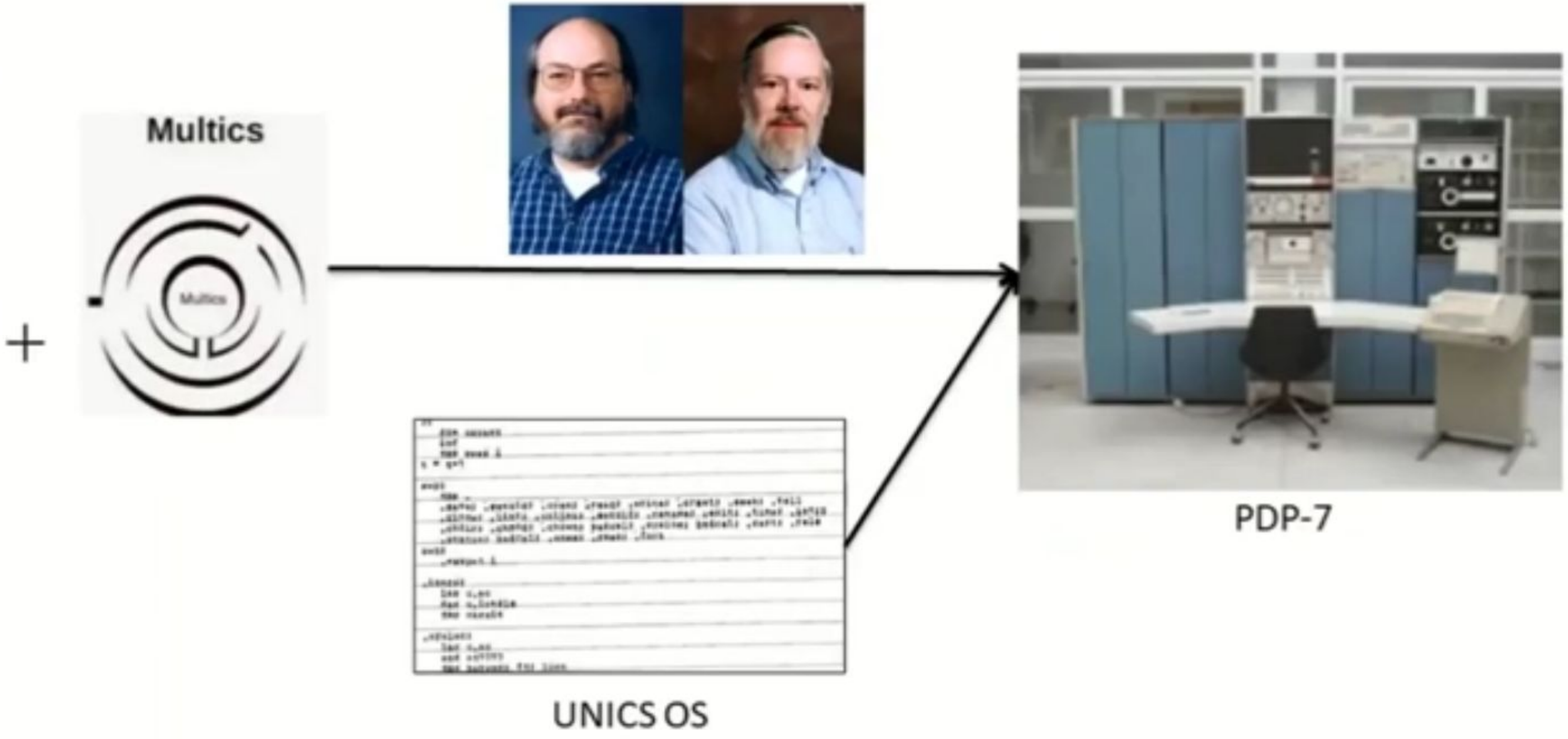
**Answer:**

IOS, Mac OSX, and Android are fundamentally built in Unix. Unix runs at many consumer devices and in Roku, Playstat ion. It is also used at the online services. If you want to be a programmer or to build a software, Unix is the ultimate re quirement. But if you want to be a general consumers only (like using computer for common purpose ), then you reall y don't need to know about Unix.

## 1.1 HISTORY OF UNIX

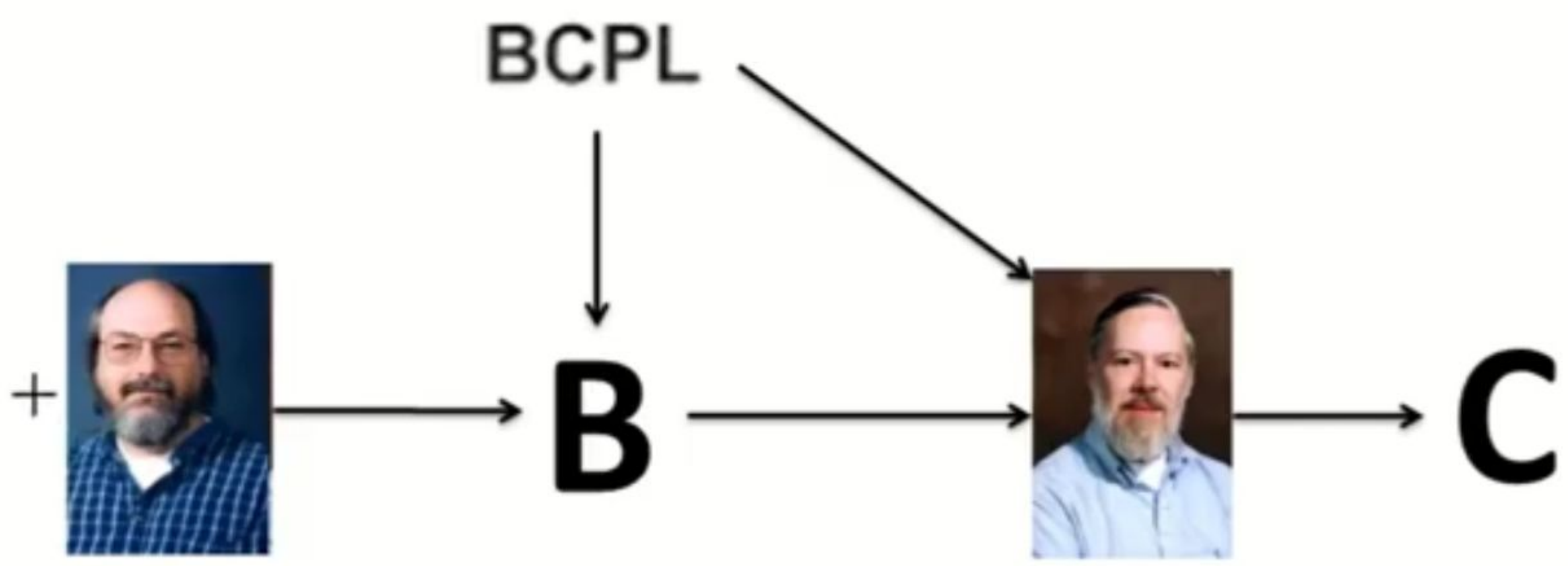


In 1965 MIT,GE,AT&T Bell **Laboratories** worked together in a project called **MULTICS(Multiplexed Information & Computing System)** to develop a multi user operating system. But, it is an failed attempt



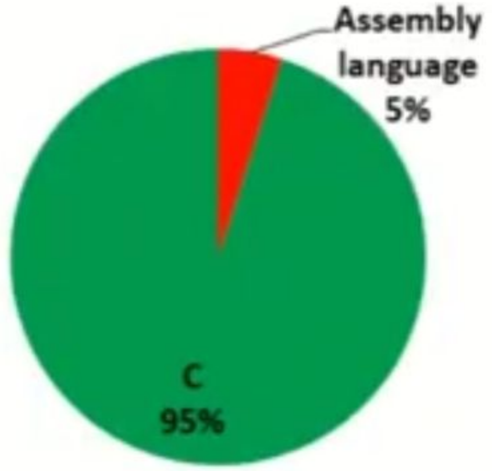
[illegible]

On the basis of ideas acquired on **MULTICS**, **Ken Thompson & Dennis Ritchie** developed an operating system called **UNICS (Uniplexed Information and Computing System)** on machine **PDP-7**,But,it is not portable



UNIX

- To achieve portability, Thompson developed B language ,but did  not yield expected results
- At same time many researchers showed interested on UNIX Project(In 1970 UNICS became UNIX)
- In 1973 **Dennis Ritchie**  developed high level programming language called **C**



- **Dennis Ritchie** Completely rewrote the entire **UNIX** using **C**.
- Around 95% of UNIX system was written in **C** and remaining was written in **Assembly language**
- The details of UNIX implementation in C was made available through public paper.
- Later **on Ken Thompson & Dennis Ritchie** was awarded with **ACM Turing award**

$\longrightarrow$ UNIX®

+

- **University of California at** Berkeley(UCB) was interested in the UNIX operating system
- The team at Berkeley was responsible for many important developments and utilities
- Berkeley filled gaps that existed in AT&T's UNIX and released their own version of UNIX called **BSD-UNIX**

$\longrightarrow$

## Many variants to UNIX?

Unix was available to universities for educational purposes and licenses were given to commercial institutions as well.

With the growing popularity of microprocessors, other companies ported the UNIX system to new machines, but its simplicity and clarity tempted many developers to enhance it in their own way, resulting in several variants of the basic system.

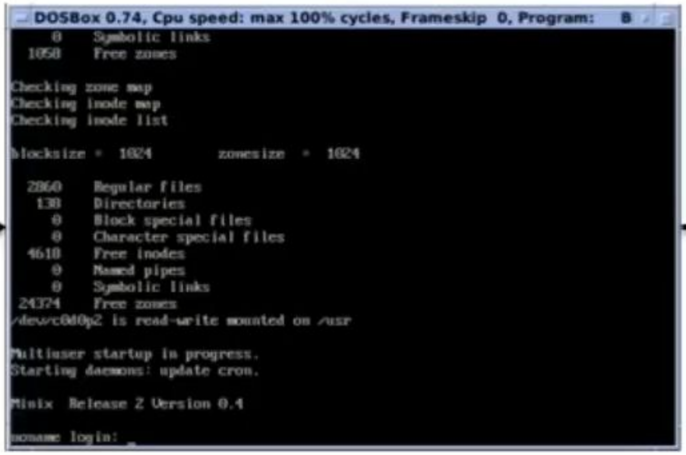
- From Mid-1970's there we have many variants of the UNIX System
- One of the reasons is AT& T & BSD giving their products for free of cost

## Any standards?

- **IEEE standards** board has given set of rules called **POSIX(Portable Operating System)**
- **AT& T** given standards called **UNIX International(UI)**
- However, still there exist a large number of UNIX variants in the market

- Earlier many variants to UNIX are commercial
- Researcher **Richard Stallman** introduced the society called **Free software foundation** to create **GNU(Genuinely Not UNIX)** - GNU is a Unix-like operating system.

- In 1991 **LINUS Torvalds** and his students used **GNU** to develop operating System **MINIX** later on its is called **LINUX**


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program:

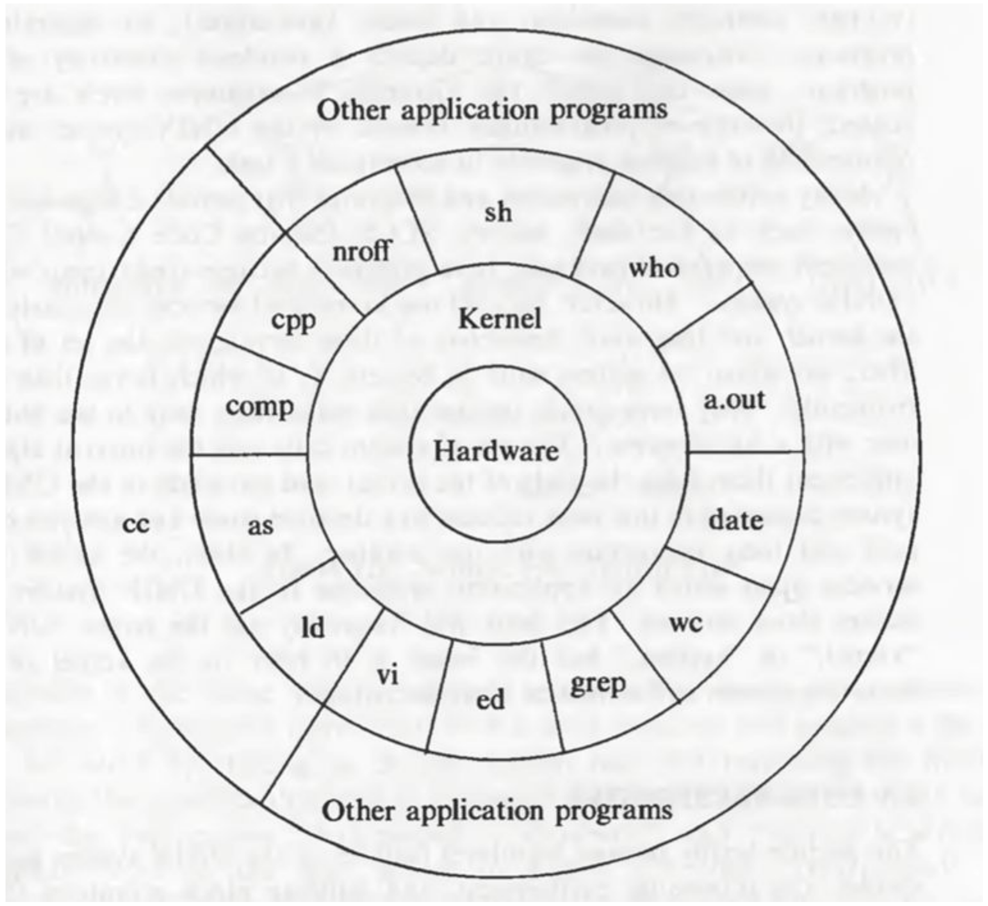
```
    0    Symbolic links
 1058    Free zones

Checking zone map
Checking inode map
Checking inode list

blocksize =  1024        zonesize  =  1024

 2860    Regular files
  138    Directories
    0    Block special files
    0    Character special files
 4618    Free inodes
    0    Named pipes
    0    Symbolic links
24374    Free zones
/dev/c0d0p2 is read-write mounted on /usr

Multiuser startup in progress.
Starting daemons: update cron.

Minix  Release 2 Version 0.4

noname login: _
```

- **LINUX** operating system is available under **GPL(General Public License)**

- Different flavors of **LINUX** operating System

archlinux
debian
fedora
foresight linux
gentoo linux
Mandriva
linux Mint
kubuntu
openSUSE
PCLinuxOS
redhat
sabayon
slackware
slax
ubuntu
xubuntu

# UNIX SYSTEM STRUCTURE



# UNIX SYSTEM STRUCTURE

- The hardware at the center of the diagram provides the operating system with basic services.
- The operating system interacts directly with the hardware, providing common services to programs and isolating them from hardware characteristics and it is called kernel, isolating it from user programs.
- Programs are independent of the underlying hardware, it is easy to move them between UNIX systems running on different hardware, if the programs do not make assumptions about the underlying hardware.
- Programs such as the shell and editors (ed and vi) shown in the outer layers interact with the kernel by invoking a well defined set of system calls .
- The system calls instruct the kernel to do various operations for the calling program and exchange data between the kernel and the program.

## UNIX SYSTEM STRUCTURE

- Commands and private user programs may also exist in this layer as indicated by the program whose name is a. out, the standard name for executable files produced by the C compiler.
- Other application programs can be build on top of lower-level programs, hence the existence of the outermost layer.
- For example, the standard C compiler, cc, is in the outermost layer of the figure: it invokes a C preprocessor, two-pass compiler, assembler, and loader link-editor, all separate lower-level programs.
- Many application subsystems and programs provide a high-level view of the system such as the shell, editors, SCCS (Source Code Control System), and document preparation packages.

## WHY UNIX BECAME SO POPULAR?

- The system is written in a high-level language, making it easy to read, understand, change, and move to other machines.
- It has a simple user interface that has the power to provide the services that users want.
- It provides primitives that permit complex programs to be built from simpler programs.
- It uses a hierarchical file system that allows easy maintenance and efficient implementation.
- It uses a consistent format for files, the byte stream, making application programs easier to write.
- It provides a simple, consistent interface to peripheral devices.
- It is a multi-user, multi-process system; each user can execute several processes simultaneously.
- It hides the machine architecture from the user, making it easier to write programs that run on different hardware implementations.

# BUILDING BLOCK PRIMITIVES

- UNIX system is to provide operating system primitives that enable users to write small, modular programs that can be used as building blocks to build more complex programs.
- One such primitive visible to shell users is the capability to **redirect I/O**.
- Processes conventionally have access to three files: they read from their **standard input file**, write to their **standard output file**, and write error messages to their **standard error file**.
- Processes executing at a terminal typically use the terminal for these three files, but each may be "redirected" independently.
- For instance, the command line
  - ls
    - lists all files in the current directory on the standard output.

## BUILDING BLOCK PRIMITIVES

- But the command line
  - ls > output
    - redirects the standard output to the file called "output" in the current directory, using the creat system call mentioned above.
- Similarly, the command line
  - mail mjb < letter
    - opens the file "letter" for its standard input and mails its contents to the user named "mjb."
- Processes can redirect input and output simultaneously, as in
  - wc –l < a.txt > count.txt 2> error.txt

# BUILDING BLOCK PRIMITIVES

- The second building block primitive is the **pipe,** a mechanism that allows a stream of data to be passed between reader and writer processes.
- Processes can redirect their standard output to a pipe to be read by other processes that have redirected their standard input to come from the pipe.
- For example, the program grep searches a set of files (parameters to grep) for a given pattern:
  - grep main a.c b.c c.c
    - searches the three files a.c, b.c, and c.c for lines containing the string "main”

## BUILDING BLOCK PRIMITIVES

- The program wc with the option -l counts the number of lines in the standard input file. The command line
  - grep main a.c b.c c.c | wc -l
    - counts the number of lines in the files that contain the string "main"; the output from grep is "piped" directly into the wc command
- The use of pipes frequently makes it unnecessary to create temporary files.

## OPERATING SYSTEM SERVICES

- **Controlling the execution of processes** by allowing their creation, termination or suspension, and communication.
- **Scheduling processes fairly for execution of the CPU**. Processes share the CPU in a time-shared manner: the CPU executes a process, the kernel suspends it when its time quantum elapses, and the kernel schedules another process to execute. The kernel later reschedules the suspended process.
- **Allocating main memory for an executing process.** The kernel allows processes to share portions of their address space under certain conditions, but protects the private address space of a process from outside tampering. If the system runs low on free memory, the kernel frees memory by writing a process temporarily to secondary memory, called a swap device. If the kernel writes entire processes to a swap device, the implementation of the UNIX system is called a swapping system; if it writes pages of memory to a swap device, it is called a paging system.

## OPERATING SYSTEM SERVICES

- **Allocating secondary memory for efficient storage and retrieval of user data**. This service constitutes the file system. The kernel allocates secondary storage for user files, reclaims unused storage, structures the file system in a well understood manner, and protects user files from illegal access.
- **Allowing processes controlled access to peripheral devices** such as terminals, tape drives, disk drives, and network devices.

## ASSUMPTIONS ABOUT HARDWARE

- The execution of user processes on UNIX systems is divided into two levels: **user and kernel.**
- When a process executes a system call, the execution mode of the process changes from user mode to kernel mode : the operating system executes and attempts to service the user request, returning an error code if it fails.
- Even if the user makes no explicit requests for operating system services, the operating system still does bookkeeping operations that relate for the user process, handling interrupts, scheduling processes, managing memory, and so on.
- Many machine architectures (and their operating systems) support more levels than these two, but the two modes, user and kernel, are sufficient for UNIX systems.

# ASSUMPTIONS ABOUT HARDWARE

- The differences between the two modes are
- Processes in user mode can access their own instructions and data but not kernel instructions and data (or those of other processes) .
- Processes in kernel mode, however, can access kernel and user addresses.
  - For example, the virtual address space of a process may be divided between addresses that are accessible only in kernel mode and addresses that are accessible in either mode.
- Some machine instructions are privileged and result in an error when executed in user mode.
  - For example, a machine may contain an instruction that manipulates the processor status register
- Although the system executes in one of two modes, the kernel runs on behalf of a user process.
- The kernel is not a separate set of processes that run in parallel to user processes, but it is part of each user process.

## INTERRUPTS AND EXCEPTIONS

- The UNIX system allows devices such as I/O peripherals or the system clock to interrupt the CPU asynchronously.
- On receipt of an interrupt, the kernel saves its current context (a frozen image of what the processor was doing), determines the cause of the interrupt, and services the interrupt.
- After servicing the interrupt, it restores its interrupted context and begins execution as if nothing had happened.
- The hardware usually prioritizes devices according to the order that interrupts should be handled: When the kernel services an interrupt, it blocks out lower priority interrupts but services higher priority interrupts.

## INTERRUPTS AND EXCEPTIONS

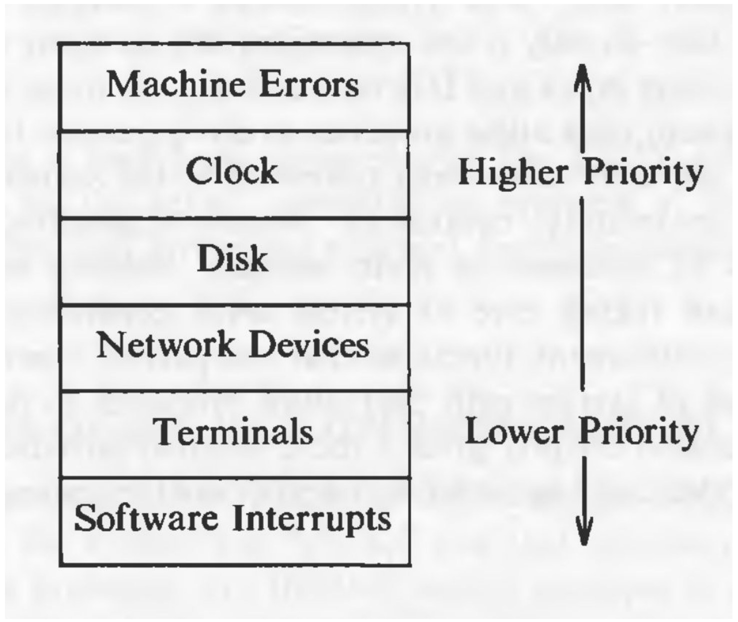
- An exception condition refers to unexpected events caused by a process, such as addressing illegal memory, executing privileged instructions, dividing by zero, and so on.
- They are distinct from interrupts, which are caused by events that are external to a process.
- Exceptions happen "in the middle" of the execution of an instruction, and the system attempts to restart the instruction after handling the exception; interrupts are considered to happen between the execution of two instruction, and the system continues with the next instruction after servicing the interrupt.

## PROCESSOR EXECUTION LEVELS

- The kernel to prevent occurrence of interrupts when it is doing some important work.
  - For instance, the kernel may not want to receive a disk interrupt while manipulating linked lists, because handling the interrupt could corrupt the pointers
- In such situations, if the interrupts are not prevented, the kernel itself might get corrupt. Computers typically have a set of privileged instructions that set the processor execution level in the **processor status word.**
- Setting the processor execution level to certain values masks off interrupts from the level and lower levels, allowing only higher-level interrupts.

## PROCESSOR EXECUTION LEVELS

- The below figure shows a sample set of execution levels.



- If the kernel masks out disk interrupts, all interrupts except for clock interrupts and machine error interrupts are prevented. If it masks out software interrupts, all other interrupts may occur.

## MEMORY MANAGEMENT

- The kernel permanently resides in the main memory.
- When compiling a program, the compiler generates a set of addresses in the program that represent addresses of variables and data structures of the addresses of instructions such as functions.
- The compiler generates the virtual addresses as if no other program will execute simultaneously on the physical machine.
- When the program is run on the machine, the kernel allocates space in main memory for it, but the virtual addresses generated by the compiler need not be identical to the physical addresses that they occupy in the machine.
- The kernel coordinates with the machine hardware to set up a virtual to physical address translation that maps the compiler-generated addresses to the physical machine addresses.

## THANK YOU