MIT-WPU

।। विश्वशान्तिर्ध्रुवं ध्रुवा ।।

Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

Operating Systems

School of Computer Engineering and technology

Operating Systems

Course Objectives:

1. <u>Knowledge</u>

To study functions of Operating Systems

To learn the basics of Unix Operating System

2. <u>Skills</u>

To design and implement algorithms of Operating Systems

To implement shell scripting

3. <u>Attitude</u>

To apply the knowledge of Operating Systems in solving real life problems

To design algorithms towards optimization of Operating System functions

Operating Systems

Course Outcomes: After completion of the course the students will be able to :-

1.Comprehend the functionalities of Operating Systems.
2.Comprehend and simulate the concepts of process and thread management.
3.Design and Implement the Process Synchronization concepts
4.Comprehend and Implement algorithms of Memory and I/O Management.
5.Ability to write basic shell scripts.

Module 1- Overview of Operating System

Overview of Operating Systems

Operating System objectives and its evolution. Operating System structure: Layered, Monolithic, Microkernel. Types of Operating Systems. Applications of Operating systems. Operating Systems protection and security.

Process Management

Process: Concept of a Process, Process States, Process Control - creation, new program execution, termination. Interposes communication(IPC). Examples of IPC. Threads: Differences between Threads and Processes. Concept of Threads, Concurrency. Multi- threading, Types of Threads. POSIX Threads functions. Scheduling: Concept of Scheduler, Scheduling Algorithms: FCFS, SJF, SRTN, Priority, Round Robin.

Module 3- Process Synchronization

Process Synchronization Tools: Concept of Mutual Exclusion, The Critical section Problem. Hardware Support for Synchronization. Semaphores, Mutex Locks, Monitors. Classical synchronization problems: Readers -Writers Problem and Producer Consumer problem. Synchronization within the kernel. Deadlock: Deadlock Characterization, Methods for handling Deadlocks, Deadlock Prevention, Deadlock Avoidance, Deadlock Detection and Recovery.

Module 4- Memory Management, I/O and File Management

Memory Management: Memory Partitioning: Fixed Partitioning,Dynamic Partitioning, Paging, segmentation ,Concept of virtual memory. Page Replacement Algorithms: FIFO, LRU, Optimal. Concept of Locality of Reference, Belady’s Anomaly. File Management: File Organization and Access, File Directories, File Sharing, Record Blocking. I/O Management: I/O Devices, Organization of the I/O Functions, I/O Buffering, Disk Scheduling- FCFS, SSTF

Module 5- Unix Operating System

Introduction to Unix Operating System. The Unix File System and Process Management. Comparison between Windows OS, Unix and Linux. Basics of shell scripting.

Module 1- Overview of Operating System

Overview of Operating Systems

Operating System objectives and its evolution.

Operating System structure: Layered, Monolithic, Microkernel.

Types of Operating Systems. Applications of Operating systems.

Operating Systems protection and security.

What is an Operating System?

- **Computer System**
  - Hardware + Software
- **Hardware**
  - Central Processing Unit (CPU), Memory and I/O devices
- **Software**
  - Systems and Applications
- **Operating System?**
  - A program that acts as an intermediary between a user and hardware
  - A program that controls the execution of application programs

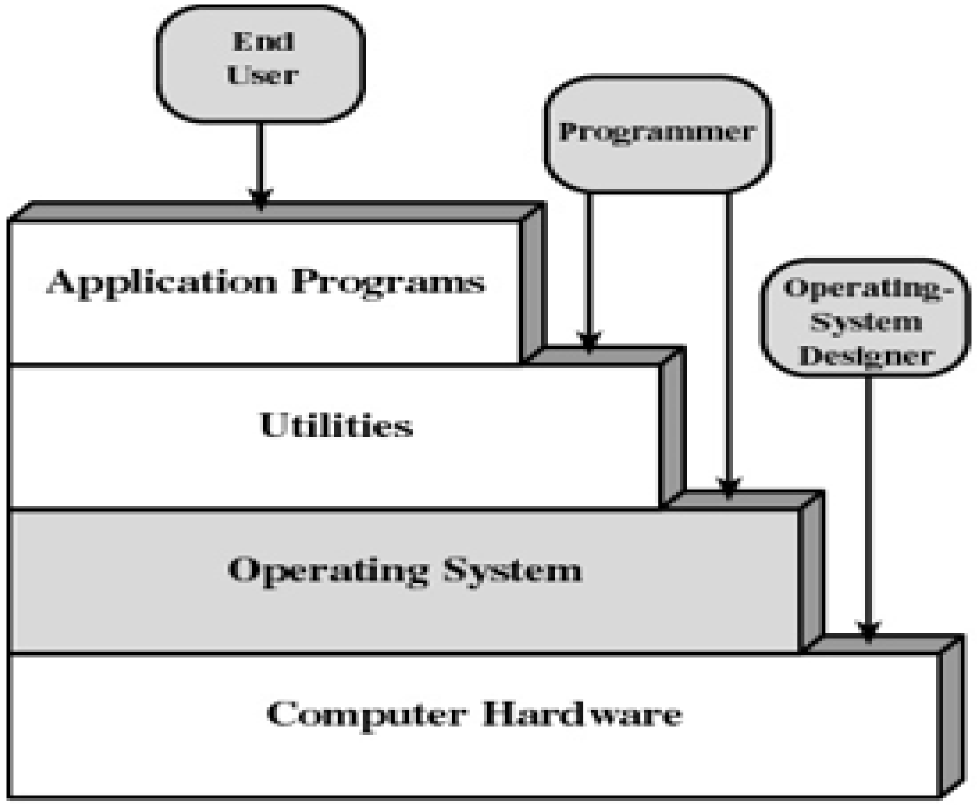
Main objectives of an OS:

- **Convenience** - To Make computer more convenient to use.
- **Efficiency** - To Allow resources to be used in an efficient manner.
- **Environment-** To execute user programs and make solving user problems easier and Permit effective application development

Operating System Definitions

- An **interface** between user & hardware
- A program that controls the execution of application programs
- Resource allocator – manages and allocates resources.
- Control program – controls the execution of user programs and operations of I/O devices .
- **Kernel** – the one program running at all times.

Layers of Computer System



Services Provided by the Operating System

- Program development
  - ➢ Editors and debuggers
- Program execution
- Access to I/O devices
- Protected access to files

Services Provided by the Operating System

- Error detection and response
    - ➢ internal and external hardware errors
    - ➢ memory error
    - ➢ device failure
- software errors
    - ➢ arithmetic overflow
    - ➢ access forbidden memory locations
- error recovery

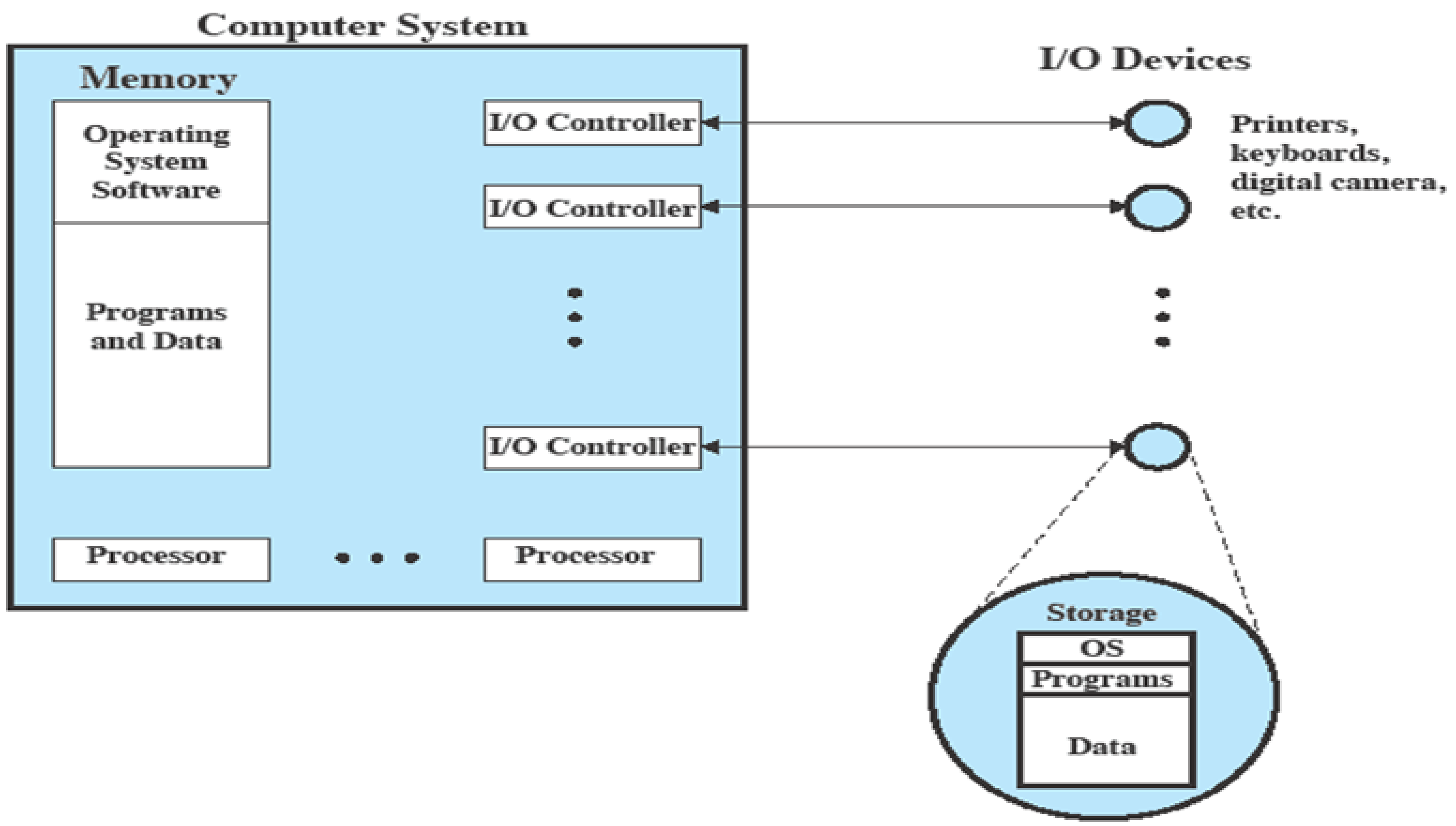
Services Provided by the Operating System

- Accounting
  - ➢ collect statistics
  - ➢ monitor performance
  - ➢ used to anticipate future enhancements
  - ➢ used for billing users

Operating System

- Functions same way as ordinary computer software
  -It is a program that is executed to service requests coming
  - ➢ from applications: system calls
  - ➢ from the hardware: interrupts
- Operating system services the request, then relinquishes control of the processor to execute other programs

OS as Resource Manager



Operating System as Resource Manager

Kernel

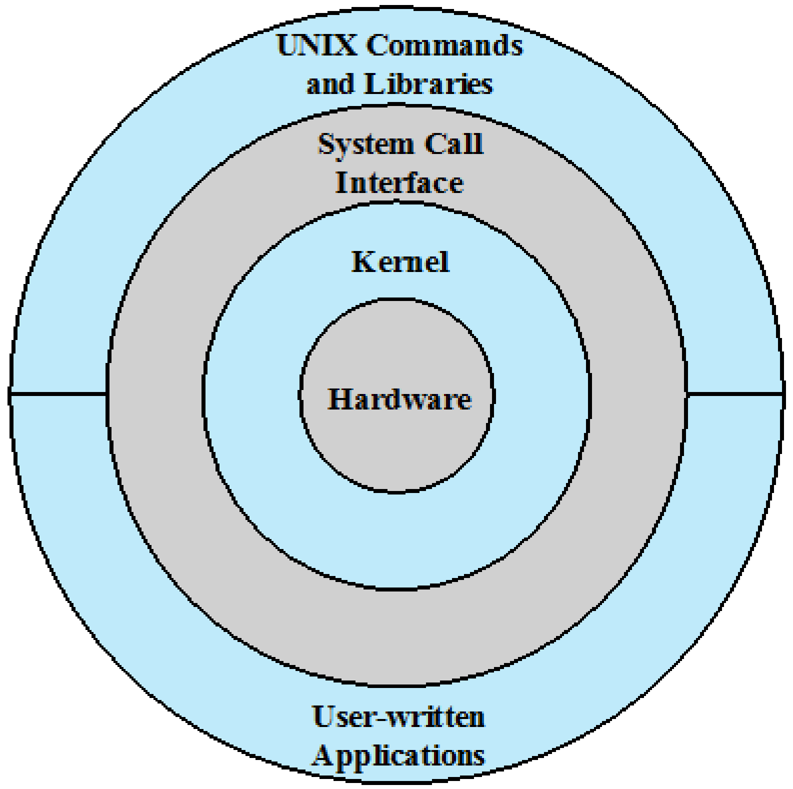
- Portion of operating system that is resident in main memory
- Contains a machine-independent part (code for system calls) and a machine-dependent part (device drivers)
- Maintains the OS state
- Executes in privileged/supervisor mode

Where OS are used?

- Desktop and server computers
    - DOS + Windows 95/98/ME
    - Windows NT/2000/XP
    - Free Unix variants: Linux, FreeBSD, NetBSD
    - Commercial Unix: Solaris, HP-UX
    - MacOS
- Game Consoles
    - Xbox
- Cars
- Digital Cameras

- Personal and Digital
    - PalmOS
    - Windows CE
    - Embedded Linux
- Mobile phones
    - Symbian OS
    - Windows Mobile
    - Android OS

OS Layers and Views

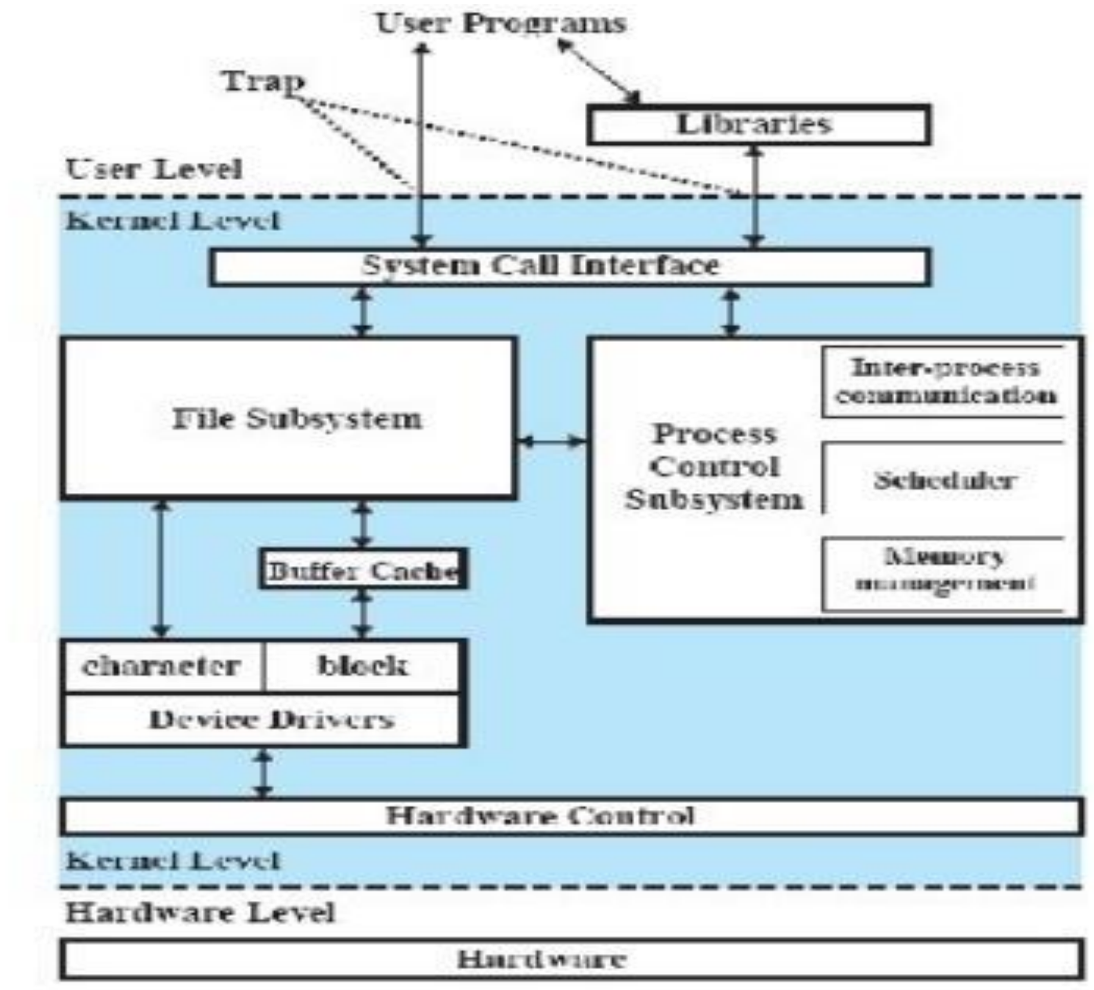


General UNIX Architecture

UNIX

- Hardware is surrounded by the operating-system software.
- Operating system **is often called the system kernel or kernel**
- Comes with a number of user services and interfaces
  - shell
  - C compiler
- The layer outside of this consists of user application and user interface to C compiler.
- User program can invoke OS services either directly or through library programs.
- The system call interface allows higher level software to gain access to specific kernel functions.

Traditional UNIX Kernel



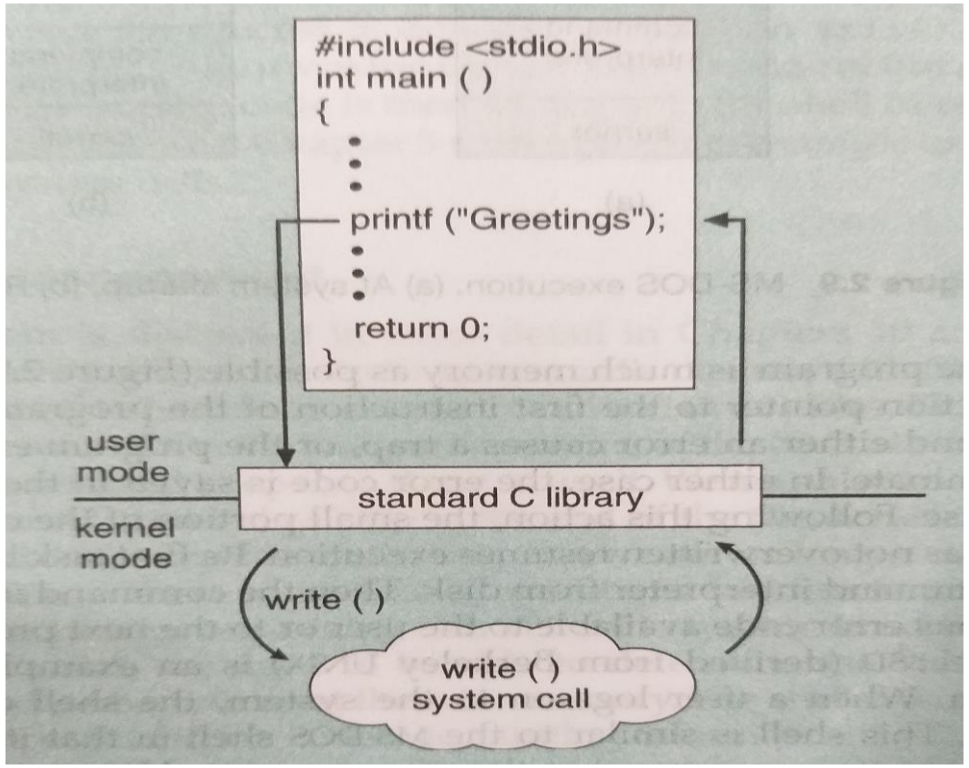
System Calls

- Programming interface to the services provided by the OS
- Interface between a running program and the operating system kernel.
- **Categories**
  - Process management
  - Memory management
  - File management
  - Device management
  - Communication
- Examples of System Calls of File Manipulations :

1.open()
2.read()
3.write()
4.close()

System call – print :Example

C program invoking printf() library call, which calls write() system call



```
#include <stdio.h>
int main ( )
{
    •
    •
    •
    printf ("Greetings");
    •
    •
    •
    return 0;
}
```

9/19/2022

Operating System Structure

9/19/2022

Operating System Structure

- Operating systems tend to be complex, as it provides many services
  - ➢ Support variety of hardware and software
  - ➢ Operating system architectures help manage this complexity
  - ➢ Organize operating system components
  - ➢ Specify privilege with which each component execute

- Modern O/S is large and complex and hence must be created carefully.
- It should not only function properly but should get modified easily.
- Common approach is to partition the task into small components.
- Each of the component is well defined portion of system, with well defined inputs, outputs and function

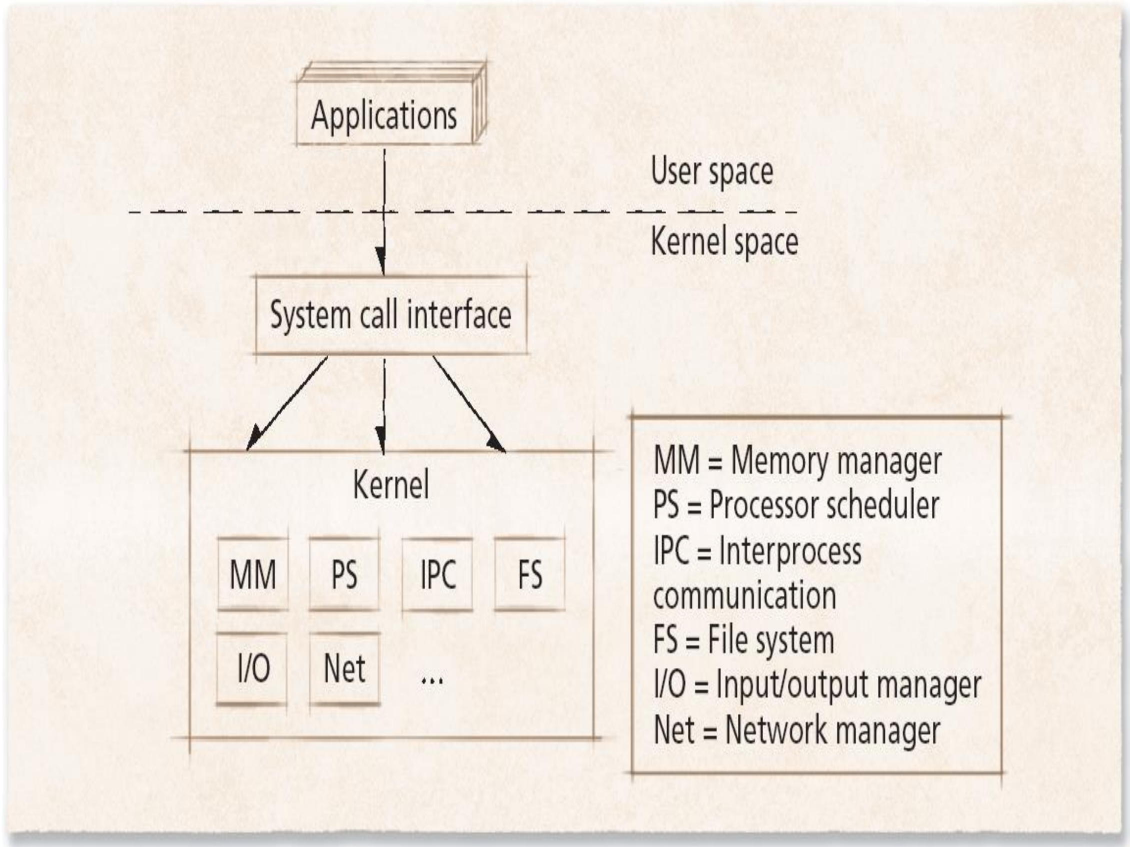
Operating System Structure

- Monolithic
- Layered Approach
- Microkernels

Monolithic Approach

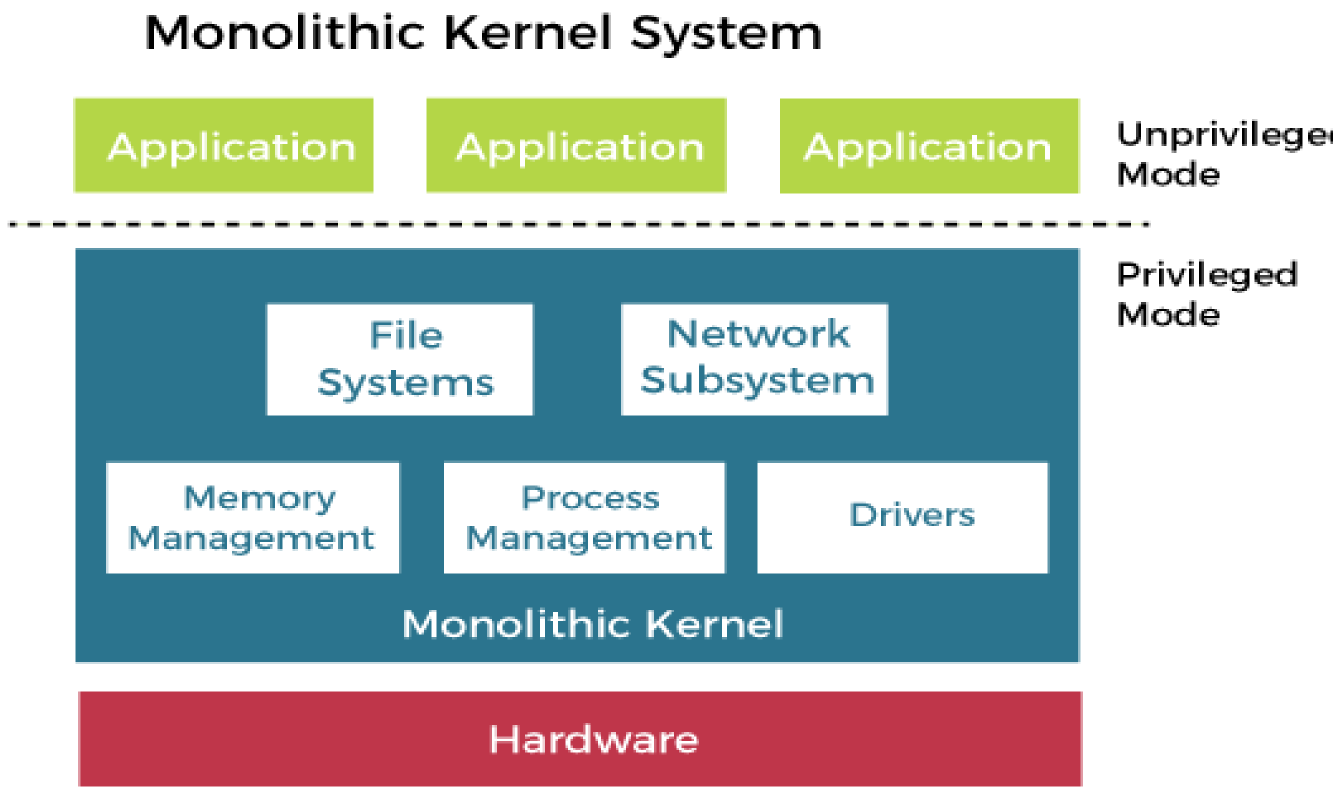
Monolithic operating system

- Every component contained in kernel
  - ➢Direct communication among all elements
  - ➢Highly efficient
  - ➢Problems:
    - complexity
    - new devices
    - emerging technologies enabling
    - protection



9/19/2022

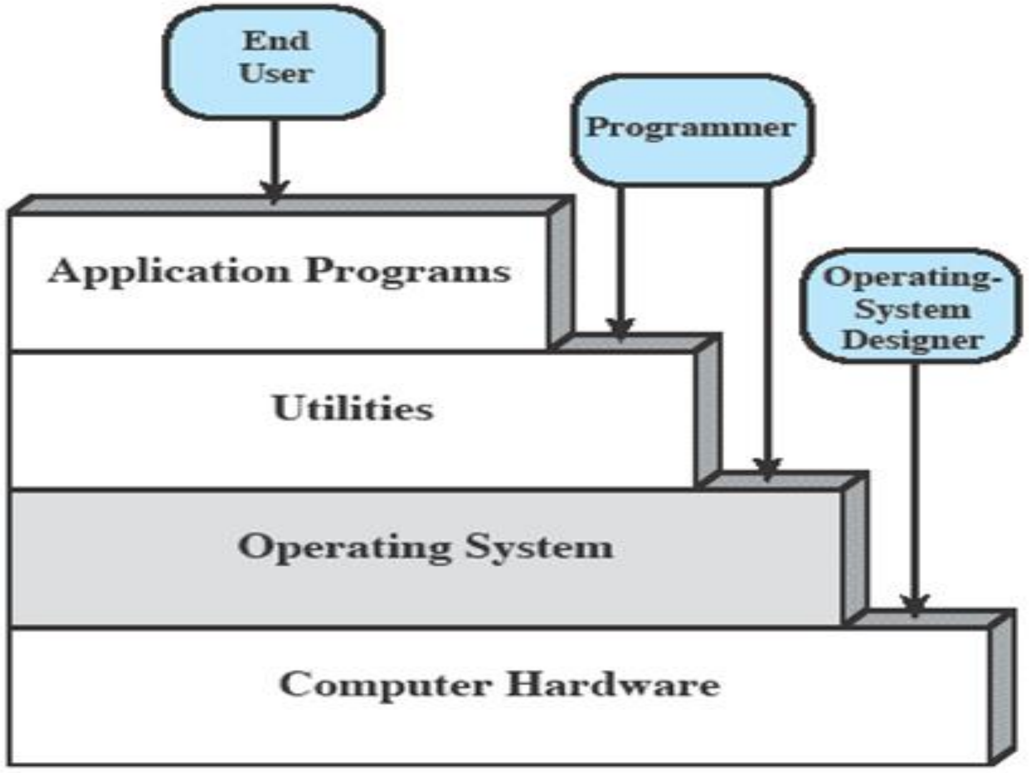
Monolithic Approach



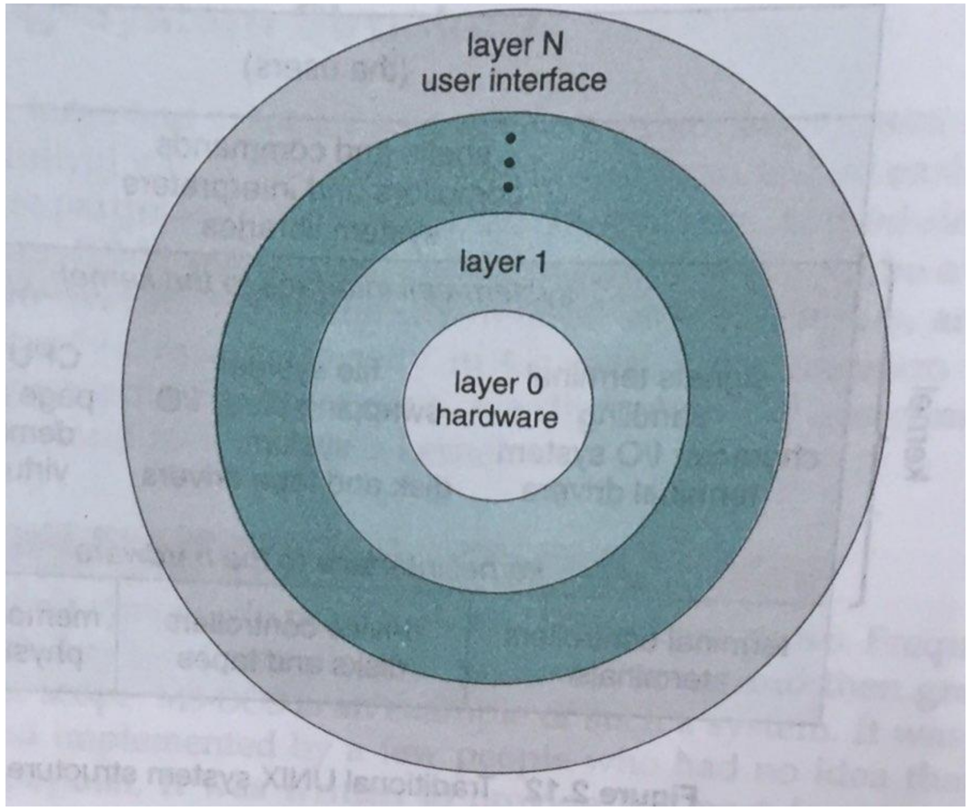
Layered Approach

- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- Typical O/S layer consists of data structures & set of operations which can be invoked by higher level layers.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers.
- Simplifies debugging and system verification, since layers use functions and services of only lower level layers.
- Each layer hides the existence of certain data structures, operations and hardware from higher level layers, since they do not have to know how these operations are implemented

Layered Approach



Layers and Views of a Computer System



9/19/2022

Layered Approach

- Components are divided into layers
  – grouping similar components
- Each layer only interacts with:
  – the bottom layer - requesting services – to top layer - answering requests
- Higher level layer – Applications
- Lowest level layer – hardware
- Advantaged – good structure, well defined interface, ...
- Disadvantages – can be slow, may be difficult to define layers.

Difficulties with Layered Approach

- No clear definition (from functionality point of view) of the layers
- Memory management routines are above device drivers for disk space, it is obvious
- Logically backing store should be above the CPU scheduler, but it is below CPU scheduler, it is not obvious.
- Layered approach tends to be less effective

Microkernel Approach

➢ Kernel is modularized using microkernel approach.

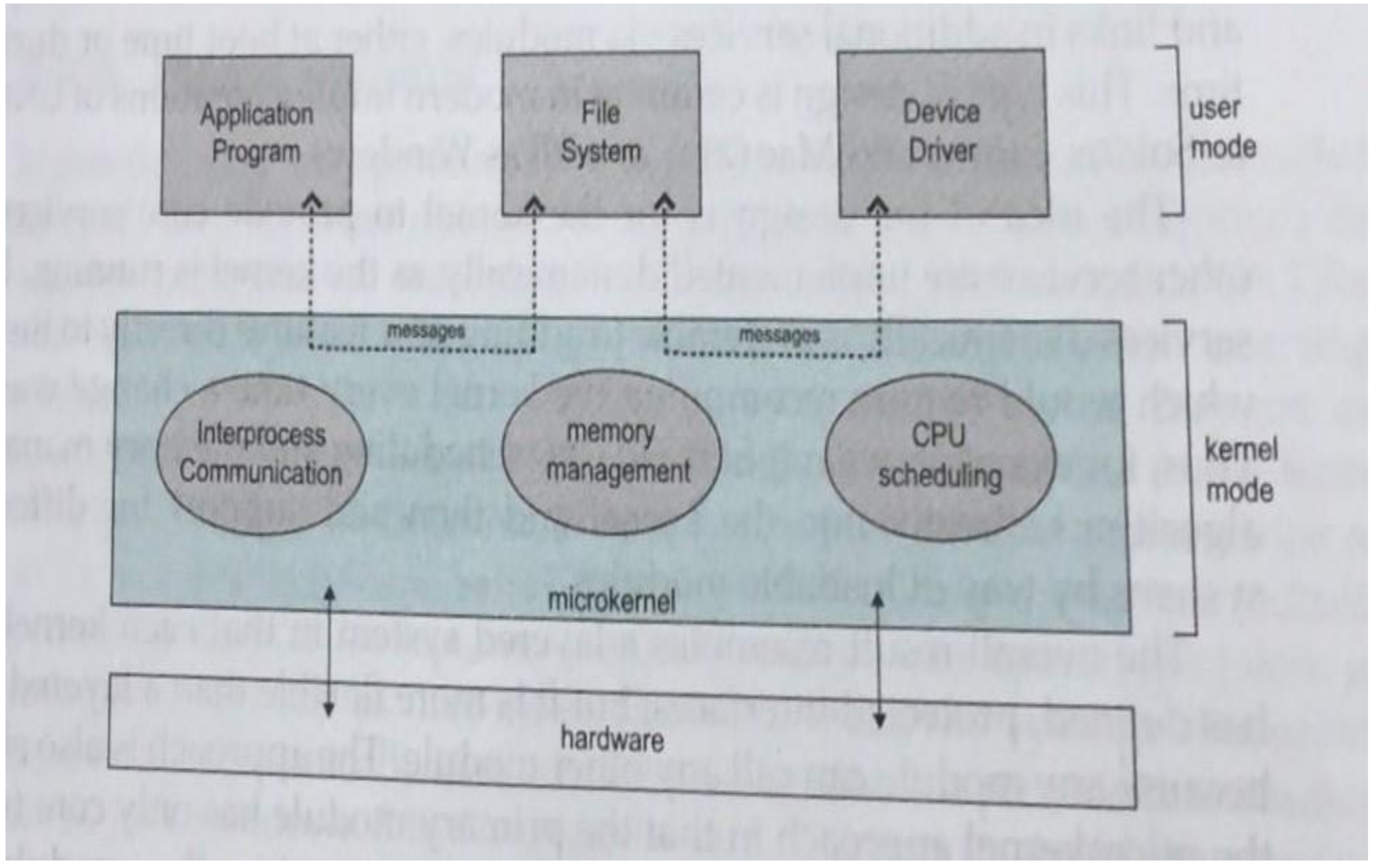
➢ It removes all **non essential components** from kernel and implements them as system and user level programs.

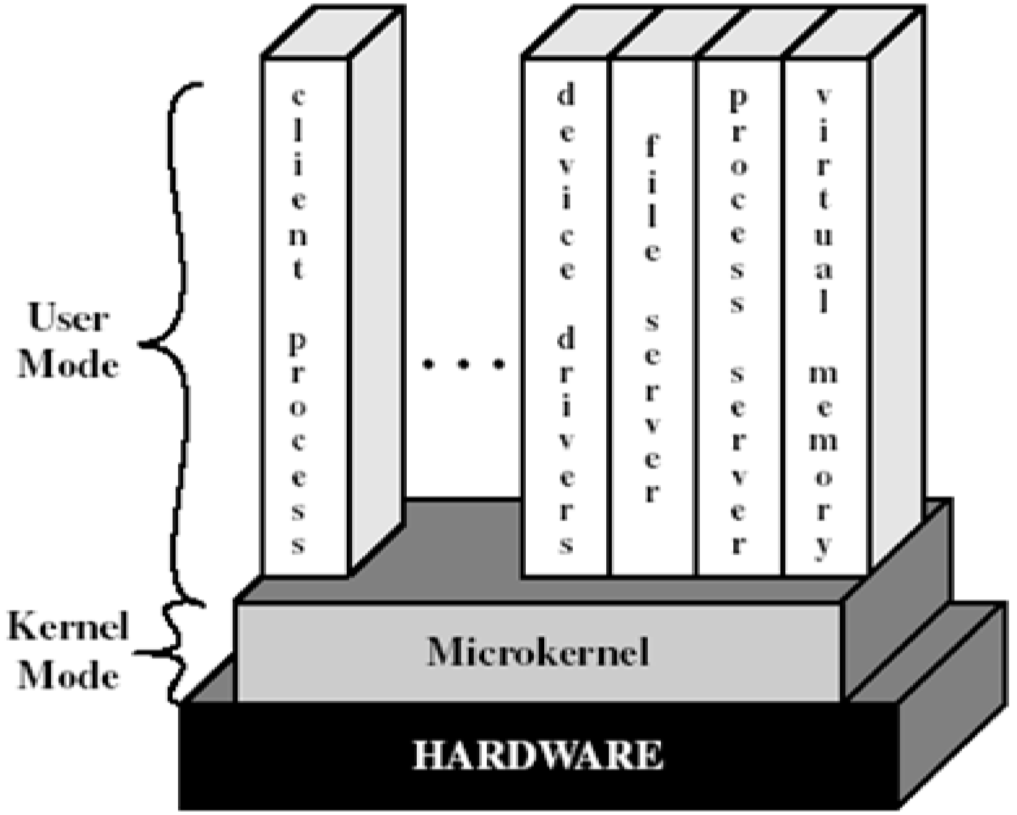
➢ It provides minimal process and memory management, in addition to communication facility.

➢ Other O/S services are provided by processes, called as servers that run in user mode and are treated like any other application by microkernel. (e.g diff file organizations can be implemented as one service) Device drivers, file systems, virtual memory manager, windowing system, security services

- Provides communication between client program and various services that are running in user space.
- Communication is provided by message passing.
- Clients and services communicate by exchanging messages with microkernel and not directly.

Microkernel Approach





Microkernel Design: Benefits

- Uniform Interface :Uniform Interface on requests made by process: Processes are not distinguished as user or kernel level, since services are provided by means of message passing
- Extendibility: Ease of extending O/S
- Flexibility :New services are added to the user space and no modification required at kernel level. Existing features can be subtracted to produce smaller and efficient implementation.
- Kernel level modification are very few.
- Portability O/S is easier to port from one H/W design to another
- Provides more security and reliability, since most services are running at user level.
- If service fails, rest of O/S remains unchanged
- Examples: Tru64Unix, Apple MacOs

OS Components / Functions

OS Components / Functions

- ➢Process Management and CPU scheduling
- ➢Memory Management
- ➢File Management
- ➢I/O System Management
- ➢System Security

Process Management

➢**A *process*** is a program in execution.

➢ A process needs certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task.

➢The operating system is responsible for the following activities in connection with process management.

- Process creation and deletion.
- Process suspension and resumption.
- Provision of mechanisms for:
- Process synchronization
- Process communication

Memory Management

- Memory is **a large array of words or bytes**, each with its own address.
- It is a **repository o**f quickly accessible data shared by the CPU and I/O devices.
- Main memory is a volatile storage device. It loses its contents in the case of system failure.
- The operating system is responsible for the following activities in connections with memory management:
  - Keep track of which parts of memory are currently being used and by whom.
  - Decide which processes to load when memory space becomes available.
  - Allocate and deallocate memory space as needed.

File Management

- ➢A file is a **collection of related information** defined by its creator. Commonly, files represent programs (both source and object forms) and data.
- ➢The operating system is responsible for the following activities in connections with file management:
  - ▪File creation and deletion.
  - ▪Directory creation and deletion.
  - ▪Support of primitives for manipulating files and directories.
  - ▪Mapping files onto secondary storage.
  - ▪File backup on stable (nonvolatile) storage media.

I/O System Management

- Control of Devices connected to Computer
- I/O devices vary widely in their function and speed, so different methods are needed to control them.
- Device drivers are required to provide an interface to I/O devices.
- Also the I/O system consists uses buffering to take care of speed difference between I/O devices and processor.

Protection

- If a computer system has multiple users and allows the concurrent execution of multiple processes, then access to data must be regulated.

- For that purpose, mechanisms ensure that files, memory, CPU and other resources can be operated on by only those processes that have gained proper authorization from the operating system.

- For eg. memory-addressing hardware ensures that a process can execute only within its own address space.

Protection

- Protection is any mechanism for controlling the access of processes or users-to the resources defined by a computer system.
- This mechanism must provide means to specify the controls to be imposed and means to enforce the controls.
- Unprotected resource cannot defend against use (or misuse) by an unauthorized or incompetent user.
- A protection-oriented system provides a means to distinguish between authorized and unauthorized usage

Security

- A system can have adequate protection but still be prone to failure and allows inappropriate access

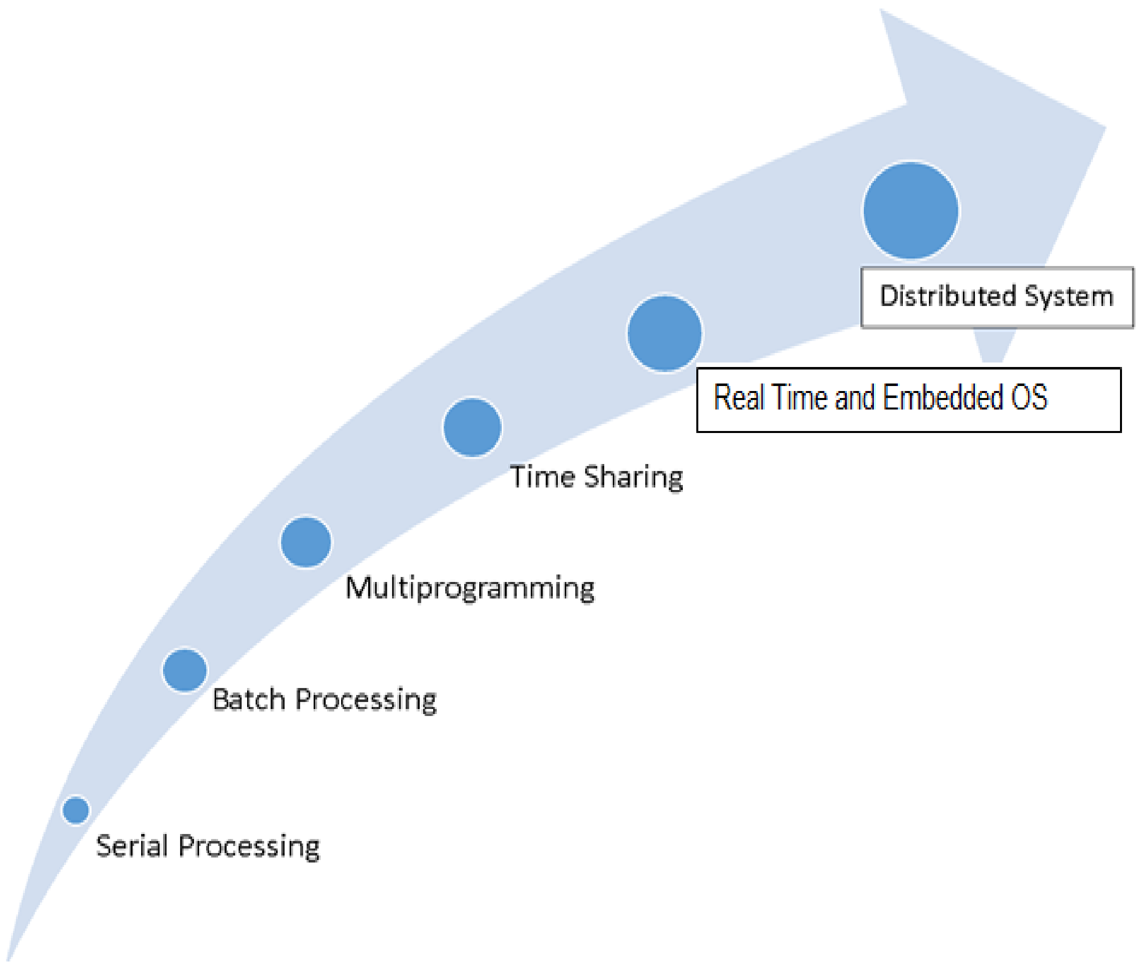
- Consider a user whose authentication information (her means of identifying herself to the system) is stolen

- Security is required to defend a system from external and internal attacks

  Such attacks include viruses and worms, denial-of service attacks, etc.

- Prevention of some of these attacks is considered an operating system function on some systems, while other systems leave the security aspects to some policy or additional software.

Evolution of Operating Systems



- **Reasons of OS Evolution**

➢A major OS will evolve over time for a number of reasons

➢Hardware upgrades

➢New services

➢Fixes/Bugs: mostly security fixes!

Serial Processing

- No operating system.
- Programmer interacted directly with computer hardware
- Program was loaded via card reader, etc & output on printer
- Manual scheduling & setup was required
- Setup included loading the compiler, source program, saving compiled program, and loading and linking.
- Windows 95 and Windows 98 are examples of operating systems which do the serial processing

Serial Processing

Problems:

1. Scheduling:

  - most installations used a hardcopy sign-up sheet to reserve computer time
  - time allocations could run short or long, resulting in wasted computer time

2. Setup time

Considerable amount of time was spent just on setting up the program to run

Simple Batch Systems

Simple batch system:

- Batch similar jobs together
- Good for large jobs needing very little interaction
- First rudimentary operating system
- Resident monitor
  - initial control in monitor
  - control transfers to job
  - when job completes control transfers back to monitor

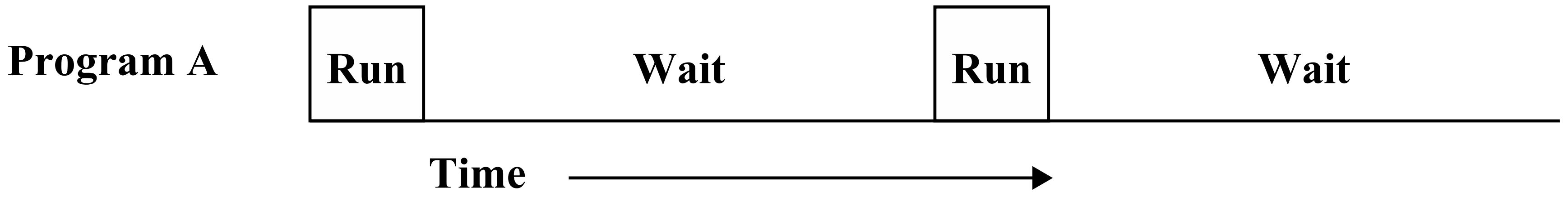
Early operating systems were custom-designed by each computer manufacturer to run on their hardware. Loaders, linkers, assemblers, computers and job sequencing software would form a rudimentary operating system, often called **resident monitor, monitor, supervisor or executive**

Simple Batch Systems

Problems/Overheads:

- Slow Performance – I/O and CPU could not overlap, card reader very slow
- Processor time alternates between execution of user programs and execution of the monitor
- Despite overhead, the simple batch system improves utilization of the computer

Uniprogramming



(a) Uniprogramming

Processor must wait for I/O instruction to complete before proceeding

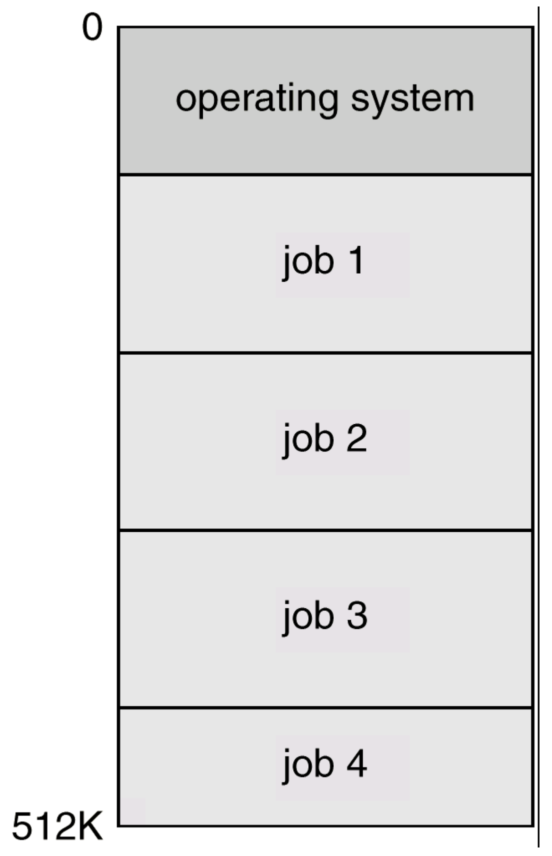
The processor spends a certain amount of time executing, until it reaches an I/O instruction; it must then wait until that I/O instruction concludes before proceeding

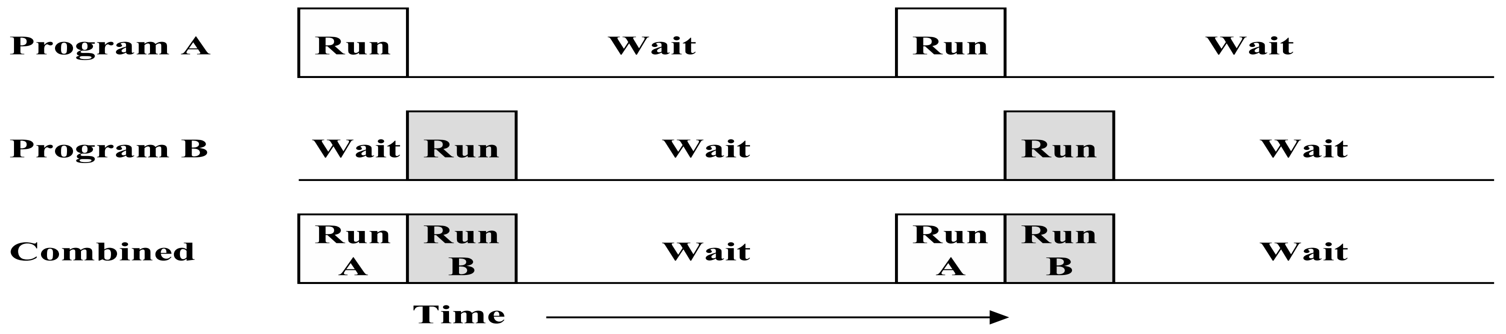
Multiprogramming

➢Several jobs are kept in main memory at the same time, and the CPU is multiplexed among them.

➢There must be enough memory to hold the OS (resident monitor) and one user program

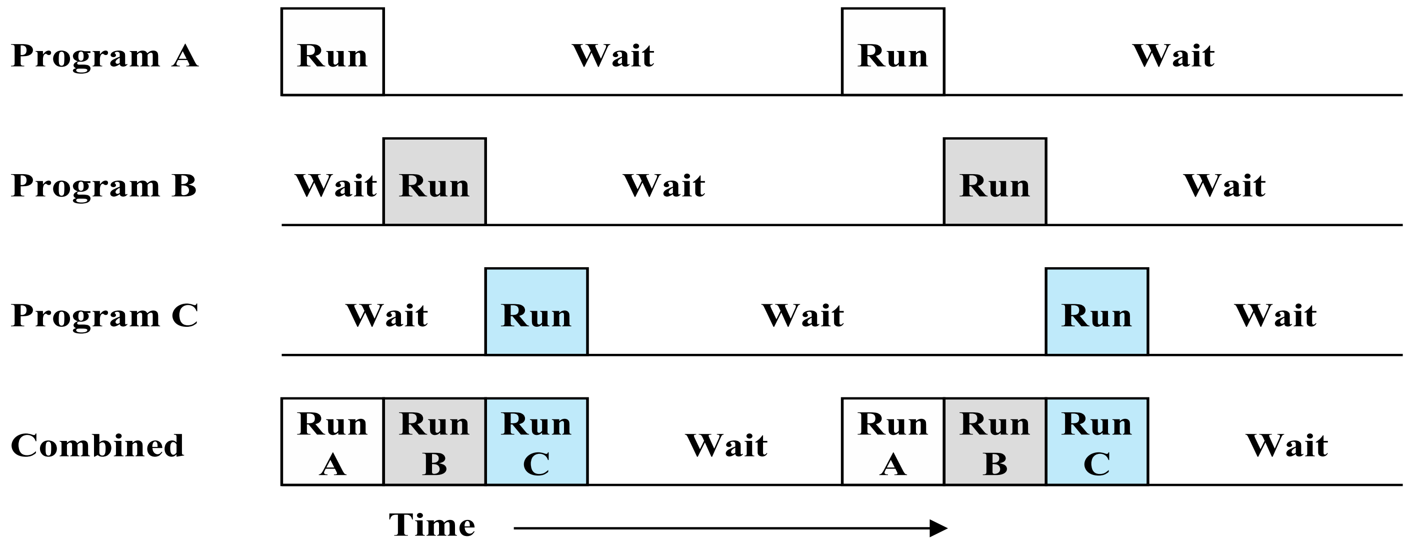
➢When one job needs to wait for I/O, the processor can switch to the other job, which is likely not waiting for I/O





(b) Multiprogramming with two programs

Multiprogramming



(c) Multiprogramming with three programs

Multiprogramming also known as multitasking
Memory is expanded to hold three, four, or more programs and switch among all of them

Time-Sharing Systems

Time Sharing systems:

- ➢Can be used to handle multiple interactive jobs
- ➢Processor time is shared among multiple users
- ➢Multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in a short burst or quantum of computation.

UNIX and LINUX are examples of Time-sharing operating systems

Distributed OS

Distributed system is collection of loosely coupled processors interconnected by a communications network.

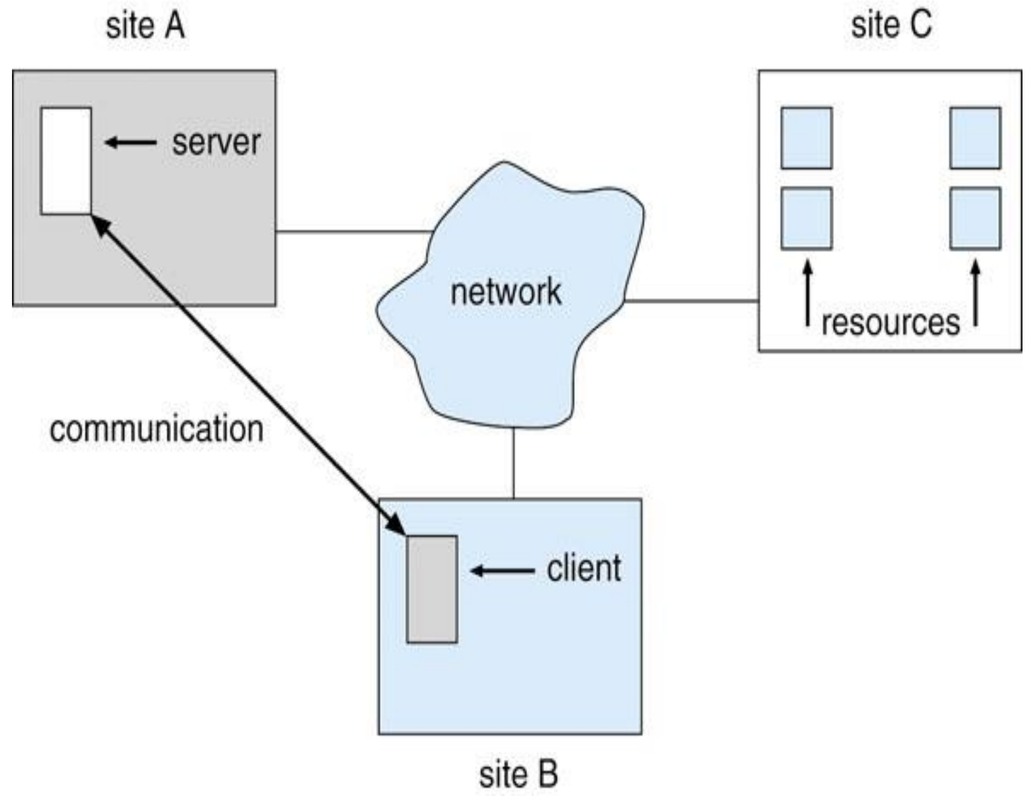
- ➢ Processors variously called *nodes, computers, machines, hosts.*
- ➢ Gives the impression there is a single operating system controlling the network.
- ➢ Users not aware of multiplicity of machines Access to remote resources similar to access to local ones.

- ➢ AIX operating system for IBM RS/6000 computers.
- ➢ Solaris operating system for SUN multiprocessor workstations.

Advantages :

- ▪ Resource Sharing
- ▪ Reliability
- ▪ Communication
- ▪ Computational Speed up

Distributed OS



Real-Time & Embedded OS

- Rigid time requirements are placed on operation of a processor or flow of data
- Often used as a control device in a dedicated application such as controlling scientific experiments, medical imaging systems, industrial control systems, and some display systems.
- Well-defined fixed-time constraints and processing must be done within define constraints.
- Real-Time systems may be either *hard* or *soft* real-time.
- Example: washing machine, aeroplane

Introduction to Linux OS

- A fully-networked 32/64-Bit Unix-like Operating System
- Multi-user, Multitasking, Multiprocessor
- Runs on multiple platforms
- Coexists with other Operating Systems

Linux: A bit about where did it come from?

- **Created by: Linus Torvalds**

  with assistance from programmers around the world

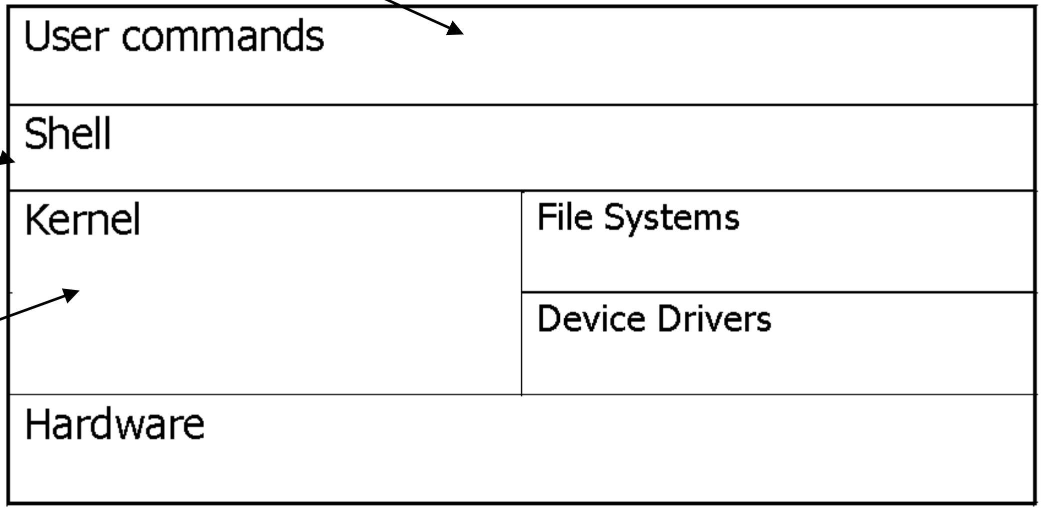
  - ➢ first posted on Internet in 1991
  - ➢ Linux 1.0 in 1994; 2.2 in 1999
  - ➢ Today used on million of computers

The Linux System

User commands includes executable programs and scripts

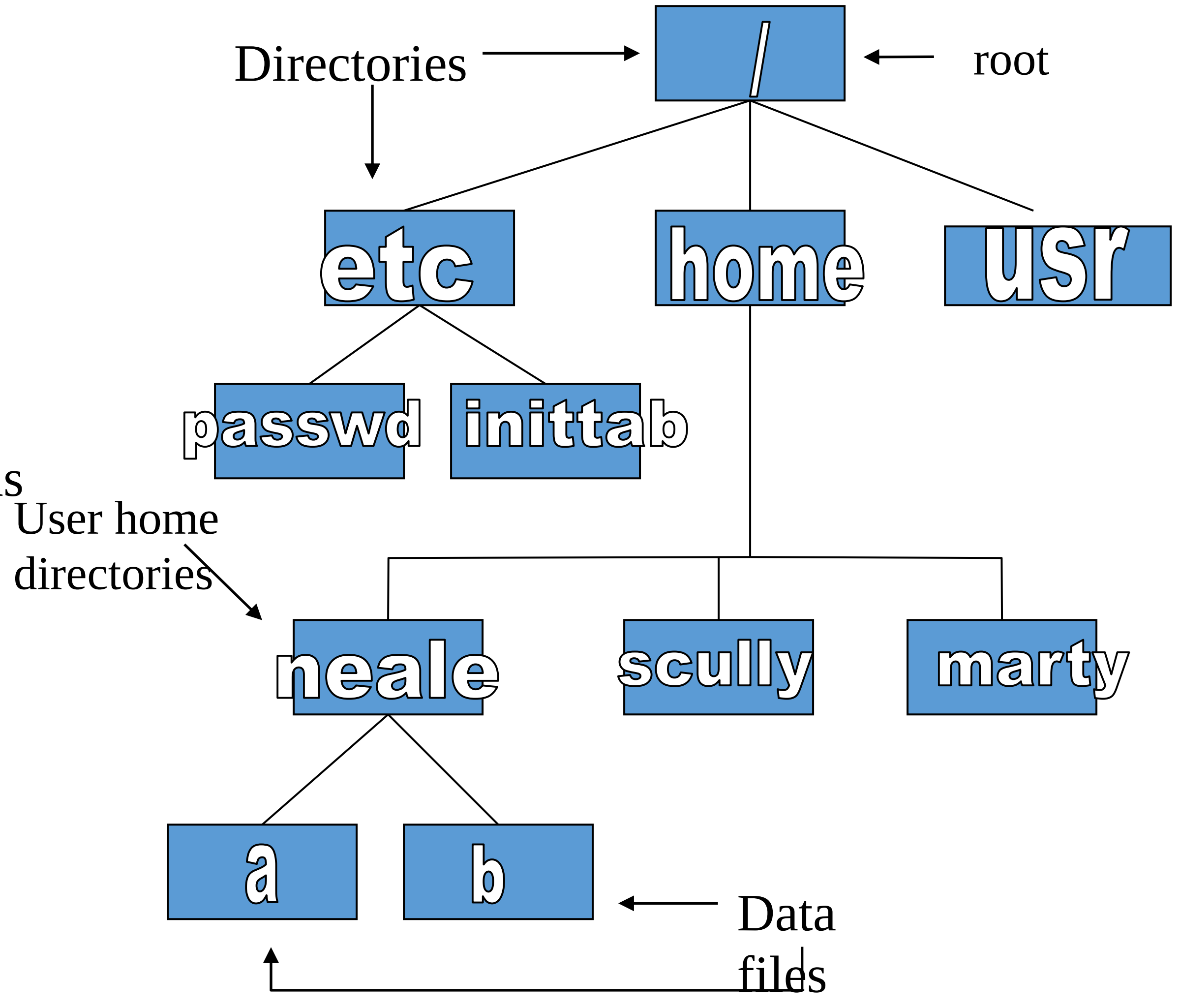
The shell interprets user commands. It is responsible for finding the commands and starting their execution. Several different shells are available. Bash is popular,

The kernel manages the hardware resources for the rest of the system.



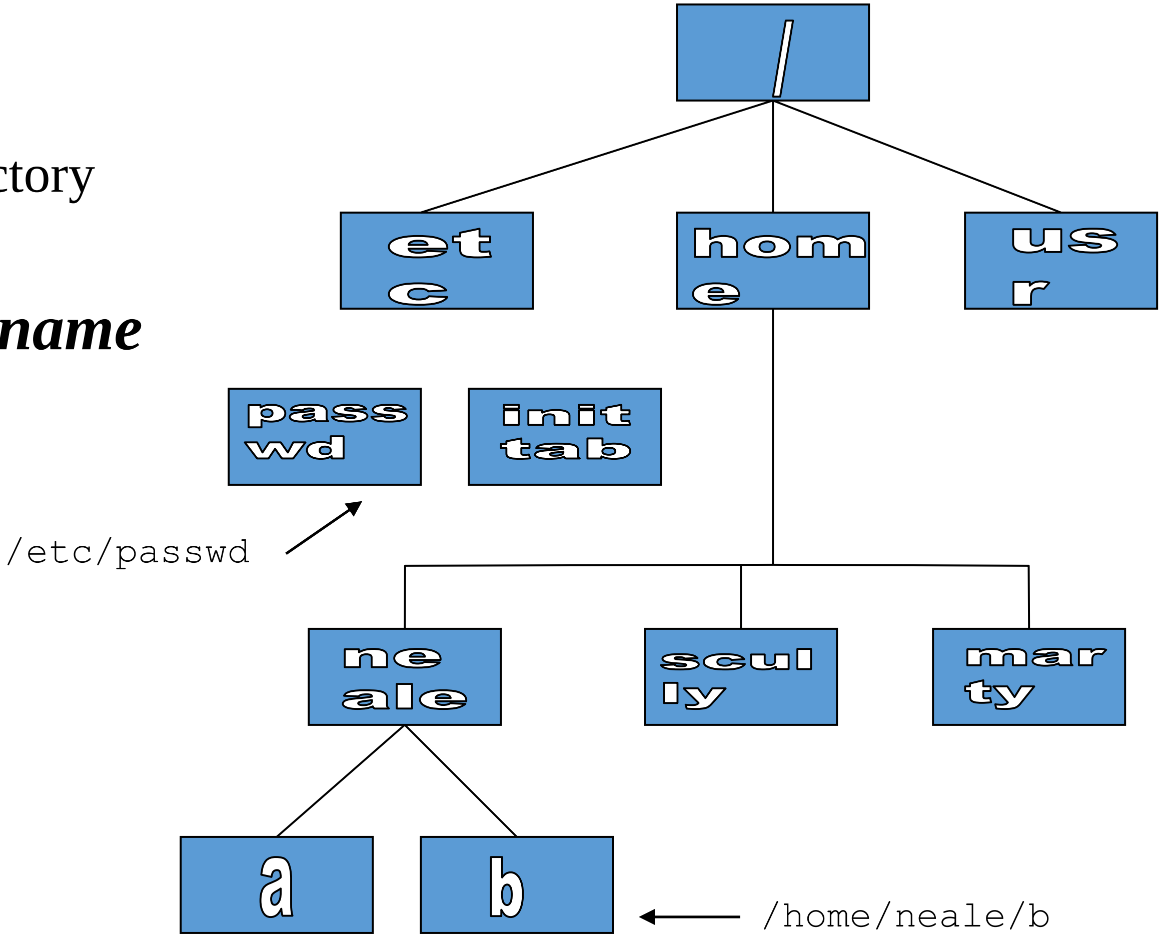
Linux File System Basics

- Linux files are stored in a single rooted, hierarchical file system
  - Data files are stored in directories (folders)
  - Directories may be nested as deep as needed



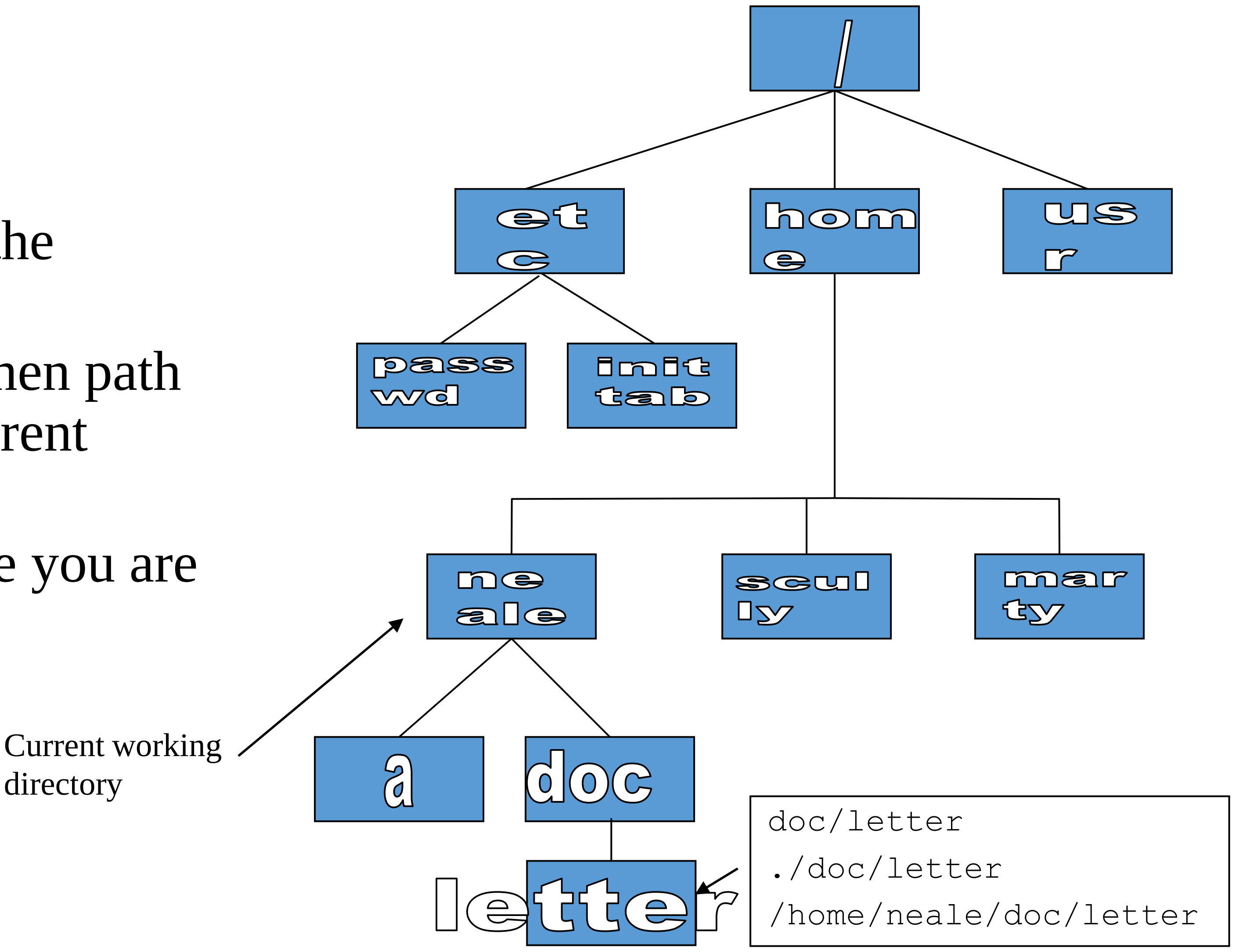
Naming Files

- **Files are named by**
  - naming each containing directory
  - starting at the root
- **This is known as the *pathname***



The Current Directory

- One directory is designated the *current working directory*
  - if you omit the leading / then path name is relative to the current working directory
  - Use <u>pwd</u> to find out where you are



```
doc/letter
./doc/letter
/home/neale/doc/letter
```

Special File Names

❑**Some file names are special**:

/ The root directory (not to be confused with the root user)

. The current directory

.. The parent (previous) directory

~ My home directory

Special Files

/home - all users' home directories are stored here

/bin, /usr/bin - system commands

/sbin, /usr/sbin - commands used by sys admins

/etc - all sorts of configuration files

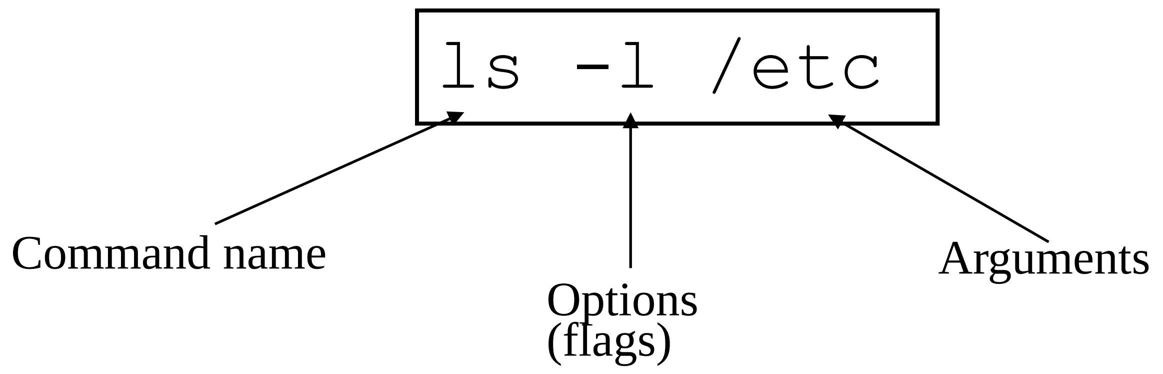
/var - logs, spool directories etc.

/dev - device files

/proc - special system files

Linux Command Basics

- To execute a command, type its name and arguments at the command line


```
ls -l /etc
```

Standard Files

- **UNIX concept of “standard files”**
  - standard input (where a command gets its input) - default is the terminal
  - standard output (where a command writes it output) - default is the terminal
  - standard error (where a command writes error messages) - default is the terminal

Redirecting Output

- The output of a command may be sent (piped) to a file:

```
ls -l >output
```

“>” is used to specify the output file

Redirecting Input

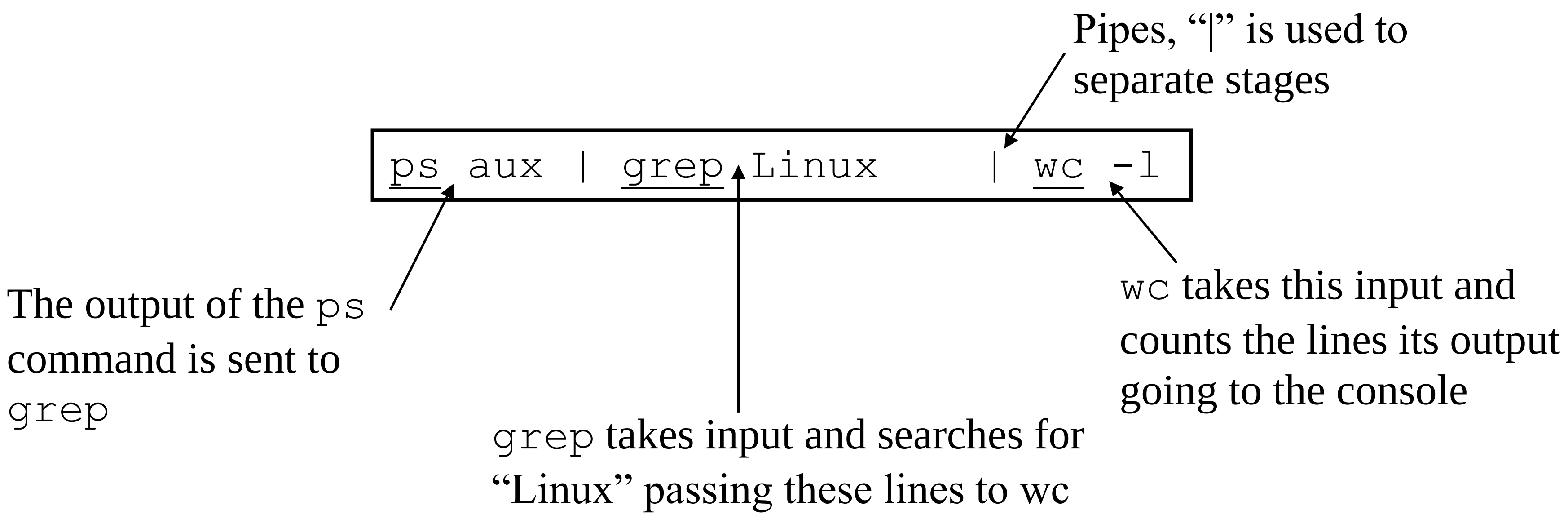
- The input of a command may come (be piped) from a file:

```
wc <input
```

“<” is used to specify the input file

Connecting commands: Pipes

- Not as powerful as CMS Pipes but the same principle
- The output of one command can become the input of another:



```
ps aux | grep Linux    | wc -l
```

Pipes, “|” is used to separate stages

The output of the `ps` command is sent to `grep`

`grep` takes input and searches for “Linux” passing these lines to wc

`wc` takes this input and counts the lines its output going to the console

Basic Commands

pwd print (display) the working directory

cd <*dir*> change the current working directory to *dir*

ls list the files in the current working directory

Ls -l list the files in the current working directory in long format

File related Commands

cp *<fromfile> <tofile>*

Copy from the <fromfile> to the <tofile>

mv *<fromfile> <tofile>*

Move/rename the <fromfile> to the <tofile>

rm <file>

Remove the file named <file>

mkdir *<newdir>*

Make a new directory called <newdir>

rmdir *<dir>*

Remove an (empty) directory

More Commands

who

List who is currently logged on to the system

whoami

Report what user you are logged on as

ps

List your processes on the system

ps aux

List all the processes on the system

echo *"A string to be echoed"*

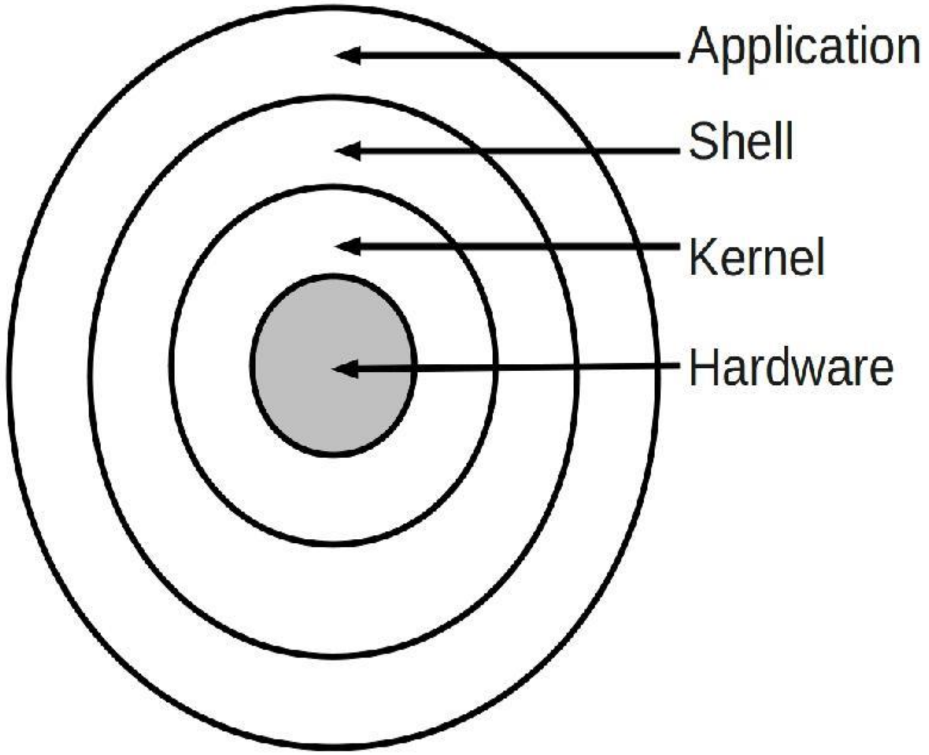
Echo a string (or list of arguments) to the terminal

Linux Shells

- An interface between the Linux system and the user
- Used to call commands and programs
- An interpreter
- Powerful programming language
  - “Shell scripts”
- Many available (bsh; ksh; csh; bash; tcsh)

What is Shell ?

- The **“Shell”** is simply ***another program*** on top of the kernel which provides a basic human-OS interface.
    - It is a command interpreter
    - Enables users to run services provided by the UNIX OS
- Functionality:
    - Command Interpreter
    - Interface
    - Programming language



Most Commonly Used Shells

/bin/csh	-- C shell
/bin/tcsh	-- Enhanced C Shell
/bin/sh	-- The Bourne Shell / POSIX shell
/bin/ksh	-- Korn shell
/bin/bash	-- Korn shell clone, from GNU

All Linux versions use the Bash shell (Bourne Again Shell) as the default shell

➢To find all available shells in your system type following command

```
$ cat /etc/shells
```

($ means **terminal prompt**)

➢To Check default Shell of your system

```
$ echo  $SHELL
```

(SHELL is a pre-defined variable)

What is Shell Script ?

- A shell script is a series of command(s) stored in plain text file.
- **Why to Write Shell Script ?**
  - Shell script can take input from user or file and output them on screen.
  - Useful to create our own commands. Save lots of time.
  - To automate some task of daily life.
  - System Administration part can be also automated.

Practical examples where shell scripting actively used:

1.Monitoring your Linux system.

2.Data backup.

3.Find out what processes are eating up your system resources.

4.Find out available and free memory.

5.Find out all logged in users and what they are doing.

6.Find out if all necessary network services are running or not.

And many more…..

References

1. William Stallings, Operating System: Internals and Design Principles, Prentice Hall, ISBN-10: 0-13-380591-3, ISBN-13: 978-0-13-380591-8, 8th Edition

2. Abraham Silberschatz, Peter Baer Galvin and Greg Gagne, Operating System Concepts, WILEY,ISBN 978-1-118-06333-0, 9th Edition