

# Introduction to UNIX

DR. DIPALI MEHER

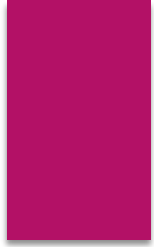
Prepared by : Dr.Dipali Meher

Source: The Design of Univ Operating System- Maurice Bach



# Introduction to UNIX/Linux Kernel

- ❑ System Structure, User Perspective, Assumptions about Hardware, Architecture of UNIX Operating System (TextBook-1: Chapter Topics: 1.2, 1.3, 1.5, 2.1)
- ❑ Concepts of Linux Programming- Files and the Filesystem, Processes, Users and Groups, Permissions, Signals, Interprocess Communication (TextBook-3: Chapter 1- relevant topics)



# What is Operating System?

An operating system is a **software** which acts as an **interface** between the end user and computer hardware.

Different types of Operating System in computer and other devices are: Batch Operating System, Multitasking/Time Sharing OS, Multiprocessing OS, Real Time OS, Distributed OS, Network OS & Mobile OS

# What is Kernel in Operating System?

- ✓ Kernel is **central** component of an operating system that manages operations of computer and hardware.
- ✓ It basically manages operations of **memory** and **CPU** time. It is core component of an operating system.
- ✓ Kernel acts as a **bridge** between applications and data processing performed at hardware level using inter-process communication and system calls.
- ✓ Generally, kernel is a **permanent resident** of the computer . It creates and terminates processes and responds to their for service.



# Objectives of Operating System

- 1. Convenient to use:** One of the objectives is to make the computer system more convenient to use in an efficient manner.
- 2. User Friendly:** To make the computer system more interactive with a more convenient interface for the users.
- 3. To provide easy access** to users for using resources by acting as an intermediary between the hardware and its users.
- 4. For managing the resources of a computer.**
- 5. Controls and Monitoring:** By keeping the track of who is using which resource, granting resource requests, and mediating conflicting requests from different programs and users.
- 6. Providing efficient and fair sharing** of **resources** between the users and programs.



# Functions of Operating System

- Device Management:** The operating system keeps track of all the devices. So, it is also called the Input / Output controller that decides which process gets the device, when, and for how much time.
- File Management:** It allocates and de-allocates the resources and also decides who gets the resource.
- Job Accounting:** It keeps the track of time and resources used by various jobs or users.
- Error-detecting Aids:** It contains methods that include the production of dumps, traces, error messages, and other debugging and error-detecting methods.



# Functions of Operating System

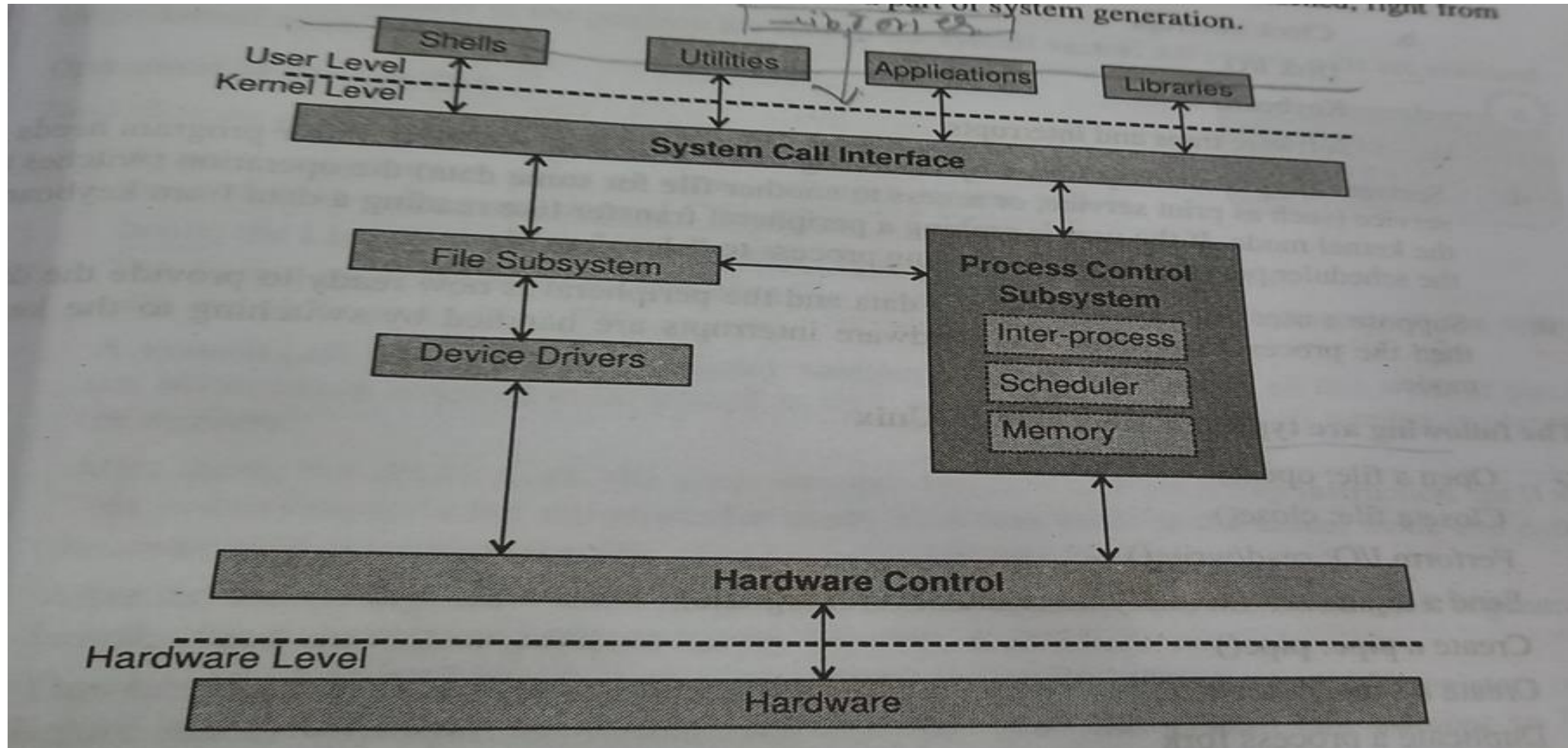
- Memory Management:** It keeps track of the primary memory, like what part of it is in use by whom, or what part is not in use, etc. and It also allocates the memory when a process or program requests it.
- Processor Management:** It allocates the processor to a process and then de-allocates the processor when it is no longer required or the job is done.
- Control on System Performance:** It records the delays between the request for a service and from the system.
- Security:** It prevents unauthorized access to programs and data by means of passwords or some kind of protection technique.

# Functions of OS

- Implementing UI
- Sharing H/W among Users
- Allowing Users to Share Data
- Resource Scheduling & management
- Error Recovery
- Data Organization
- Handling N/W communications
- Facilitating i/p and o/p



## Architecture of the UNIX





## Architecture of the UNIX

All functions of operating system are run by the kernel i.e the kernel runs the show

There are 2 OS requirements

- 1) functions for process management – allocation of resources( CPU, memory and all services needed by processes)
- 2) functions for file management- handling all the files required by processes , communication with device and regulating transmission of data to and from peripherals.

## Architecture of the UNIX

Kernel Handles following operations

- 1) Scheduling of user and other processes
- 2) Memory allocation
- 3) Swapping between memory and disk
- 4) Data movement from and to peripherals
- 5) Receiving all services requests from all process and honors them

Kernel tell everything to OS how to run and what to run when to run

Example: when anew disk is attached, right from its formatting, mounting it with file system and unmounting it.

# User Mode & Kernel Mode

At any time only one process is active at CPU. This can be user process or system routine(chmod) that is providing service.

CPU always engage a process that is runnable.

Scheduler will choose runnable process and give its control to CPU to execute it.

The process enters into running state from runnable state.

Switching from kernel mode to user mode.

In Unix-like operating systems, the chmod command is used to change the access mode of a file. The name is an abbreviation of change mode.

# User Mode & Kernel Mode

Switching from kernel mode to user mode:

- 1) Scheduler allocates a user process a slice of time( 0.1 sec)and then system clock interrupts. So currently running process stopped and another process which is runnable is selected. This switching is done in kernel mode. The current process priority is revalued and (made lower)
- 1) Priorities are as follows:
  - 1) Hardware errors
  - 2) Clock interrupts
  - 3) DISK I/Q
  - 4) Keyboard
  - 5) Software traps and interrupts

# User Mode & Kernel Mode

Switching from kernel mode to user mode:

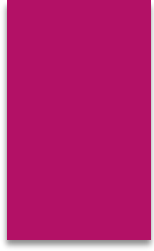
2) Services provided by Kernel- I user wants service form kernal( printing a file, access to some another data)

Example if the user us seeking a peripheral transfer like reading a data from keyboard then scheduler puts the currently running process to sleep mode.

3) When user process wants data and peripheral device is now ready to provide the data then the process interrupts. The hardware interrupts are handled by switching to the kernel mode

# System Calls in UNIX

- ☐ Open a file: `open()`
- ☐ Close a file: `close()`
- ☐ Perform I/O: `read/write()`
- ☐ Send a signal `kill`
- ☐ Create a pipe: `pipe()`
- ☐ Create a socket: `socket()`
- ☐ Duplicate a process `fork`
- ☐ Overlay a process `exec`
- ☐ Terminate a process `exit`



# Concepts of Linux Programming- Files

File System: a large group of files and directories

User files and directories are on different file system than system files and directories

UNIX imposes some restrictions on file names.

Filename upto 256 characters long with any alphanumeric character except ‘

Extensions

.h--: Header file

.C→ C source file

.F→ Fortran file

.p→ Pascal file

.s→ Assembler source file

./a.out→ file contains machine language code



# Concepts of Linux Programming- Files and File Systems

Linux follows everything is a file philosophy.

To access a file it should be must opened for read/write purpose or both  
File descriptor is used to reference a open file which datatype is int.

**Regular file** → bytes of data organized into a linear array called a byte stream

File is referenced by inode not by its name. Inode is a numeric value. As i-number or ino. An inode stores metadata associated with a file, such as

- length
- owner,
- Type
- modification timestamp
- location files data



**Special file**→ kernel objects that are represented as files.

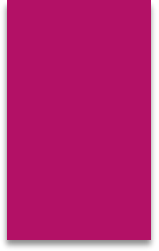
Linux provides system call to create special file

- Block device files
- character device files.
- Names pipes
- Unix domain sockets

**Names Pipes**→ they are in IPC (Interprocess Communication) mechanism that provides a communication channel over a file descriptor accessed via a special file.

Regular pipes→ they are the method used to pipe the output of one program into the input of another program, They are created by system call.

**Sockets**→ They are final type of special file. They are advanced form of IPC that allows for communication between two different process on same or different machine.



# Concepts of Linux Programming- Signals

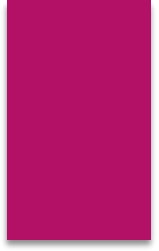
The interrupt is a signal emitted by hardware or software when a process or an event needs immediate attention.

Signals are software generated **interrupts** that are sent to a process when an event happens.

They are **synchronous** or **asynchronous**.

Signals can be posted to a process when the system detects a software event, such as user entering an interrupt or stop or kill request from another process.

Or can also come directly from OS kernel when a hardware event such as a bus error or an illegal instruction is encountered.



## Actions taken for signals

- 1) The signal is discarded after begin received
- 2) The process is terminated after the signal is received
- 3) A core file is written, then the process is terminated
- 4) Stop the process after the signal is received.

# Concepts of Linux Programming- Processes

Processes are the object code in execution.

Process consists of data, resources, state and a virtualized computer.

An active running program is a process.

Process means executable file. The executable file format contains multiple sections of code and data. The sections are:

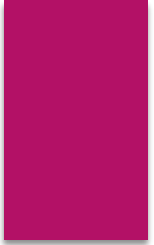
**DATA: initialized data such as c variable with values assigned**

**BSS: uninitialized global data**

**TEXT:- executable code and read-only data**

**Absolute section:-non relocatable symbols**

**Undefined section**



A process is associated with various system resources managed by kernel. A process requests and manipulated resources only through system calls.

Resources such as

- timers

- pending signals

- open files

- network connections

- hardware

- mechanism



Through virtual memory and paging, the kernel allows many processes to co-exist on the system each operating in a different address space.

The kernel manages this virtualization through hardware support provided by modern processors.

Kernel supports both pre-emptive multitasking and non-preemptive multitasking.

Terms related to Kernel System:

**wait system calls:** This system call blocks the calling process until one of its child process exists( or an error occurs). It returns a status code(integer pointer) telling that how child process existed.

**Process termination: two ways 1) and 2) are normal**

**1) executing program calls exit function**

**2) main function returns**

**3) abnormally- process response to signal. SIGTERM signal I sent by kill command.**

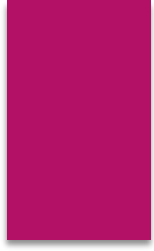
**Every process has exit code they is returned by it to its parent process.**

**Signals: Mechanism used for communication between processes. Special message sent to process. Signals are asynchronous .**

**Threads: Each process consists of one or more threads of execution. It is an activity within a process.**

**Users and Groups:Authorization in linux is provided bu users and groups with uid and gid.**





# Assumptions about Hardware

As we know execution of user processes on UNIX is divided into two level user mode and kernel mode. In system call: execution mode of the process changes from user mode to kernel mode.

When OS attempts user requests the mode is differentiated as follows:

- i) Processes in user mode can access their own instructions and data but not kernel instructions. But processes in kernel mode can access the kernel and user addresses.
- ii) Some machine instructions are privileged and result in an error when executed in user mode.