TATA ELXSI

Operating System Concepts

Learning & Development Team

Disclaimer

- This material is developed for in house training at **TATA ELXSI**.

- The training material therefore was solely developed for educational purposes. Currently, commercialization of the prototype does not exist, nor is the prototype available for general usage.

- **TATA ELXSI**, its staff, its students or any other participants can use it as a part of training. This material should not be found on any website.

- For the preparation of this material we have referred from the below mentioned links or books.

- Excerpts of these material has been taken where there is no copy right infringements.

Operating System Concepts

About Operating System and Kernels

- ✓ About Operating system and Kernels
- ✓ O.S vs. Kernels
- ✓ Multi – user
- ✓ Multi- processing O.S Real Time O.S
- ✓ Operating System components
- ✓ Overview of Process management sub system
- ✓ Overview of Memory management sub system
- ✓ Overview of File sub system
- ✓ Overview of Device drivers
- ✓ Overview of Signals and System calls

- ✓Process management sub system
- ✓Creating process
- ✓Process Id, process descriptor
- ✓Process contexts
- ✓Process state
- ✓Process scheduling
- ✓Threads vs. process
- ✓File sub system
- ✓Importance of File system
- ✓File representation in the system
- ✓File creation and maintenance
- ✓Block devices and file system

What is an Operating System?

- A program that acts as an intermediary between a user of a computer and the computer hardware.

- Operating system goals:
  - Execute user programs and make solving user problems easier.
  - Make the computer system convenient to use.

- Use the computer hardware in an efficient manner.

Computer System Components

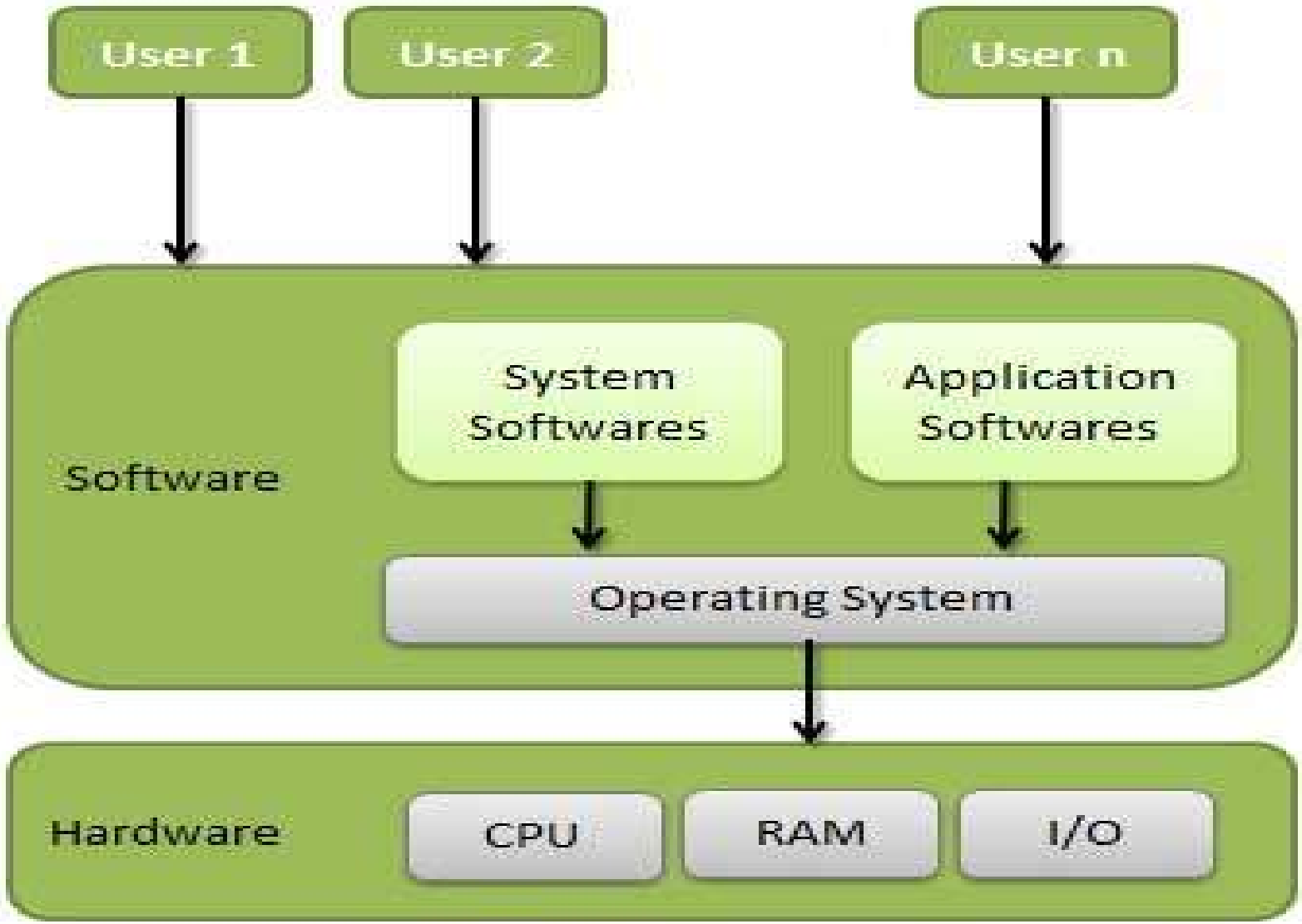
1. Hardware – provides basic computing resources (CPU, memory, I/O devices).

2. Operating system – controls and coordinates the use of the hardware among the various application programs for the various users.

3. Applications programs – Application program interface (API) is a set of routines, protocols, and tools for building software applications.

- - An API specifies how software components should interact

4. Users (people, machines, other computers).


User 1
User 2
User n
Software
System Softwares
Application Softwares
Operating System
Hardware
CPU
RAM
I/O

Operating System Definitions

- Resource allocator – manages and allocates resources.

- Control program – controls the execution of user programs and operations of I/O devices .

Components of OS

- **The Bootloader:** The software that manages the boot process of your computer.
  - Ex: ntldr, BOOTMGR, LILO(Linux Loader) GNU GRUB (GNU GRand Unified Bootloader)

- **The kernel :** kernel is the core of Operating system and manages the CPU, memory, and peripheral devices.

- **Daemons:** These are background services (printing, sound, network services, etc) that either start up during boot, or after you login.

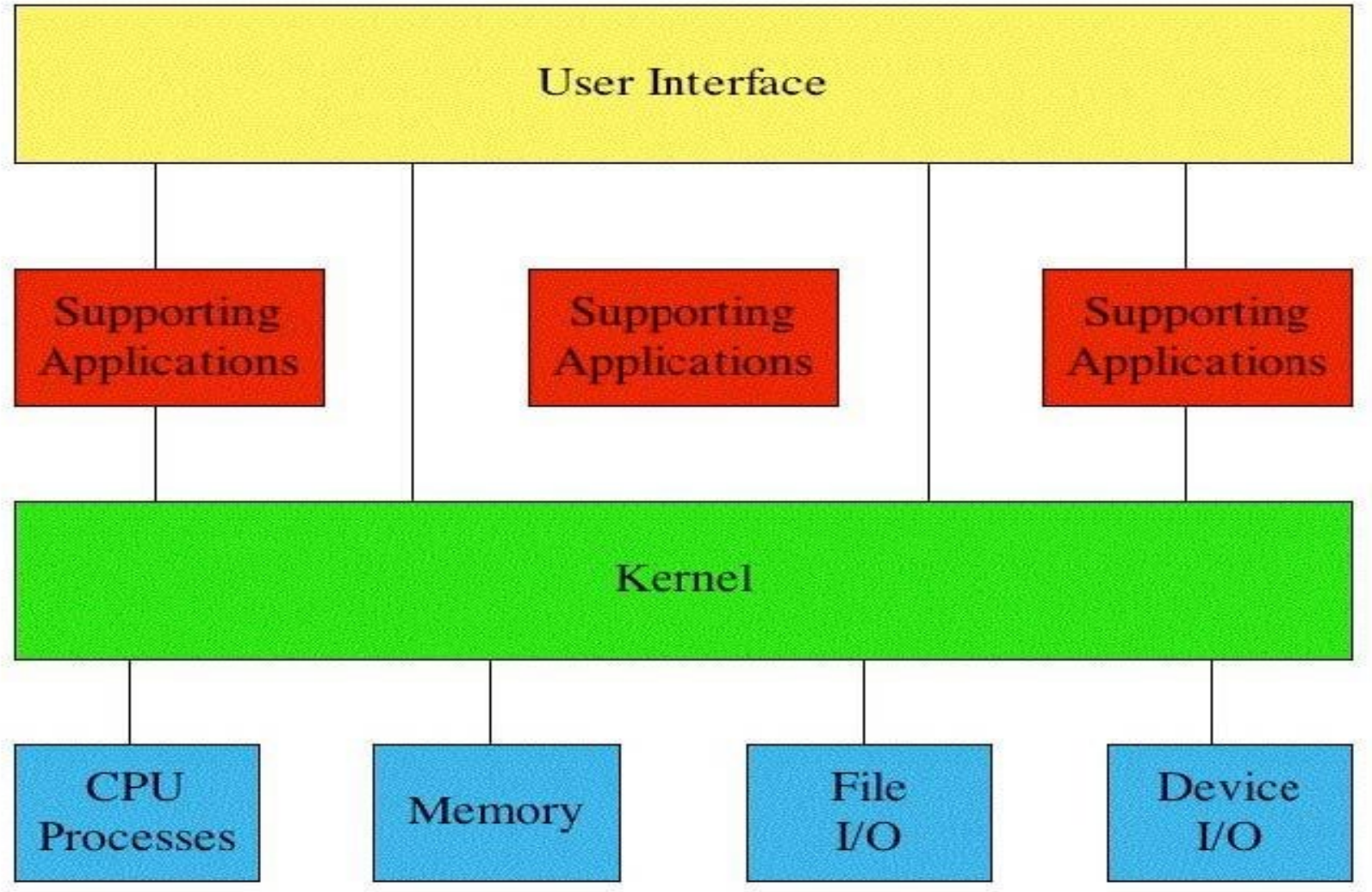
- **Shell**: Shell is command interpreter. Which consitute interface between kernel and user applications.

- **Applications:** An application is a type of software that allows you to perform specific tasks.

Kernel Definition

- Kernel as described above is the heart of OS which manages the core features of an OS.
- The kernel is the first program loads into memory during *booting* (i.e., system startup), and it remains there for the entire duration of the computer session because its services are required continuously.
- Thus it is important for it to be optimized as much as possible.
- Because of its critical nature, the kernel code is usually loaded into a *protected* area of memory, which prevents it from being overwritten by other.

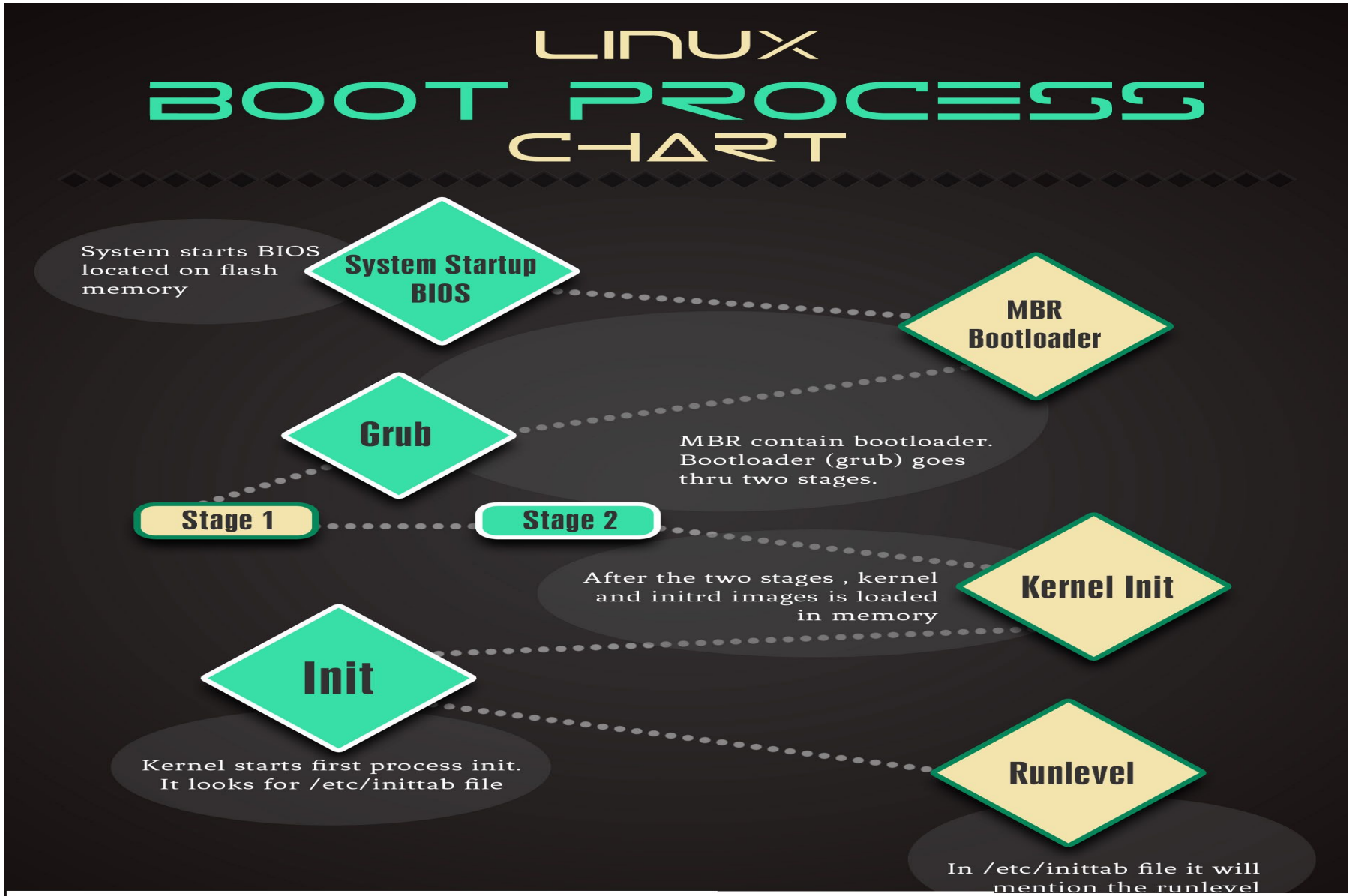
Kernel layout



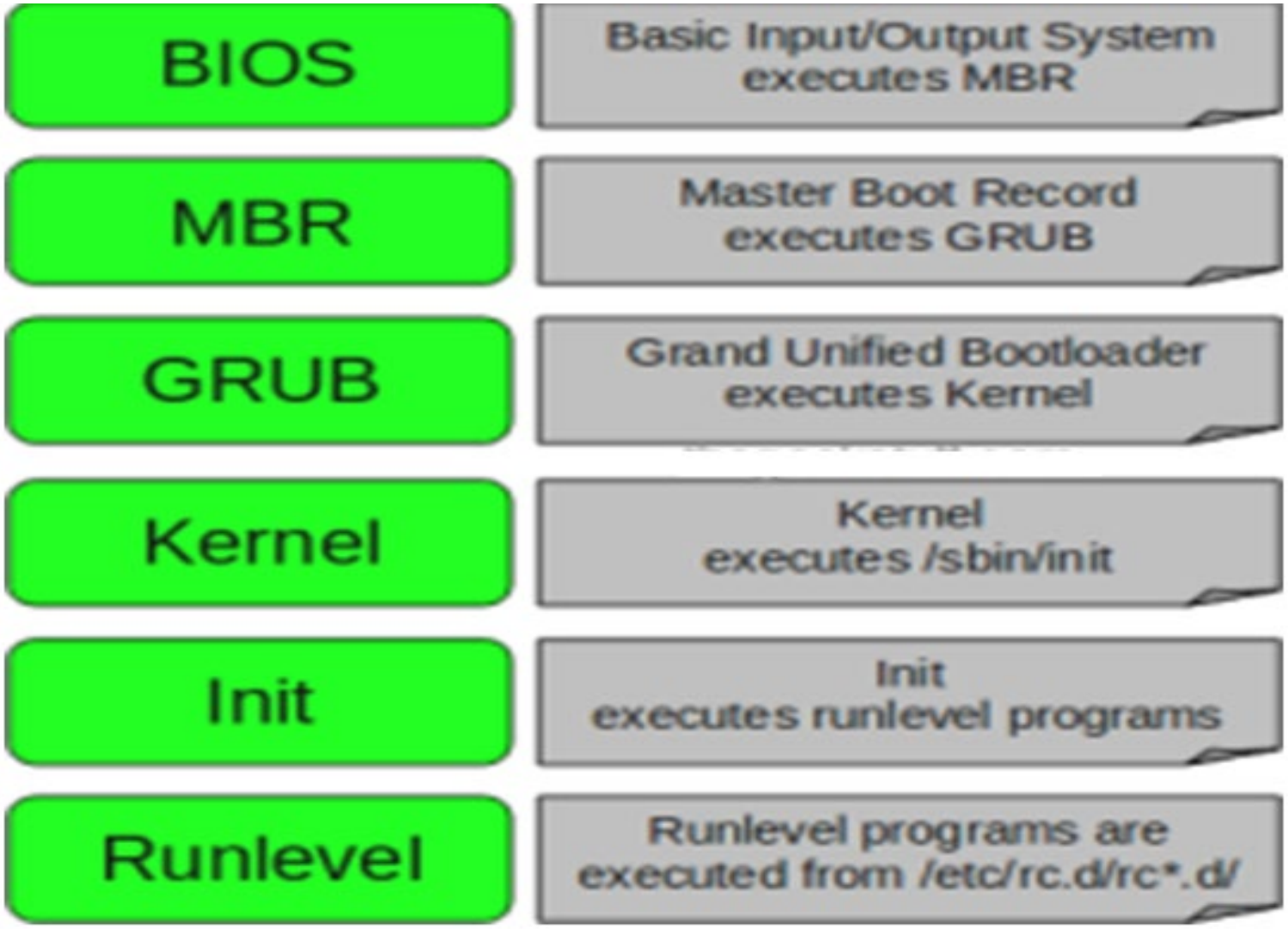
- When a computer *crashes*, it actually means the kernel has crashed.

- If only a single program has crashed but the rest of the system remains in operation.

- The main tasks of the kernel are :
  - Process management
  - Device management
  - Memory management
  - Interrupt handling
  - I/O communication
  - File system...etc..


LINUX
BOOT PROCESS
CHART
System starts BIOS located on flash memory
System Startup BIOS
MBR Bootloader
Grub
MBR contain bootloader. Bootloader (grub) goes thru two stages.
Stage 1
Stage 2
After the two stages , kernel and initrd images is loaded in memory
Kernel Init
Init
Kernel starts first process init. It looks for /etc/inittab file
Runlevel
In /etc/inittab file it will mention the runlevel

Boot Process



The BIOS

- The *BIOS* (Basic Input/Output System).

- The BIOS is an independent program stored in a chip on the *motherboard* (the main circuit board of a computer)

- That is used during the booting process for such tasks as initializing the hardware and loading the kernel into memory.

Linux - kernel

- The term “Linux” to refer to the Linux kernel, but also the set of programs, tools, and services that are typically bundled together with the Linux kernel to provide all of the necessary components of a fully functional operating system.

- From smartphones to cars, supercomputers and home appliances, the Linux operating system is everywhere.

- List of Linux based OS:
  - Ubuntu
  - OpenSUSE
  - RedHat
  - CentOS
  - Android
  - uClinux

Categories of Kernels

- Kernels can be classified into four broad categories:
  - *monolithic kernels*,
  - *microkernels*,
  - *hybrid kernels* and
  - *exokernels*.
- Each has its own advantages.

Monolithic kernel

- Monolithic kernels, which have traditionally been used by Unix-like operating systems, contain all the operating system core functions and the *device drivers.*

- Modern monolithic kernels, such as those of Linux and FreeBSD, both of which fall into the category of Unix-like operating systems, feature the ability to load *modules* at runtime,

- Thereby allowing easy extension of the kernel's capabilities as required, while helping to minimize the amount of code running in kernel space.

Micro kernels

- The kernel provides only a small set of functions, such as some synchronization facilities, an elementary scheduler and inter process communication mechanisms.

- Important functions like memory management, device drivers, system call handlers etc., run on top of the micro kernel.
- These are all glued together.
- Examples of microkernel operating systems are:

  - **Mac OS X, MINIX and QNX.**

Micro kernels

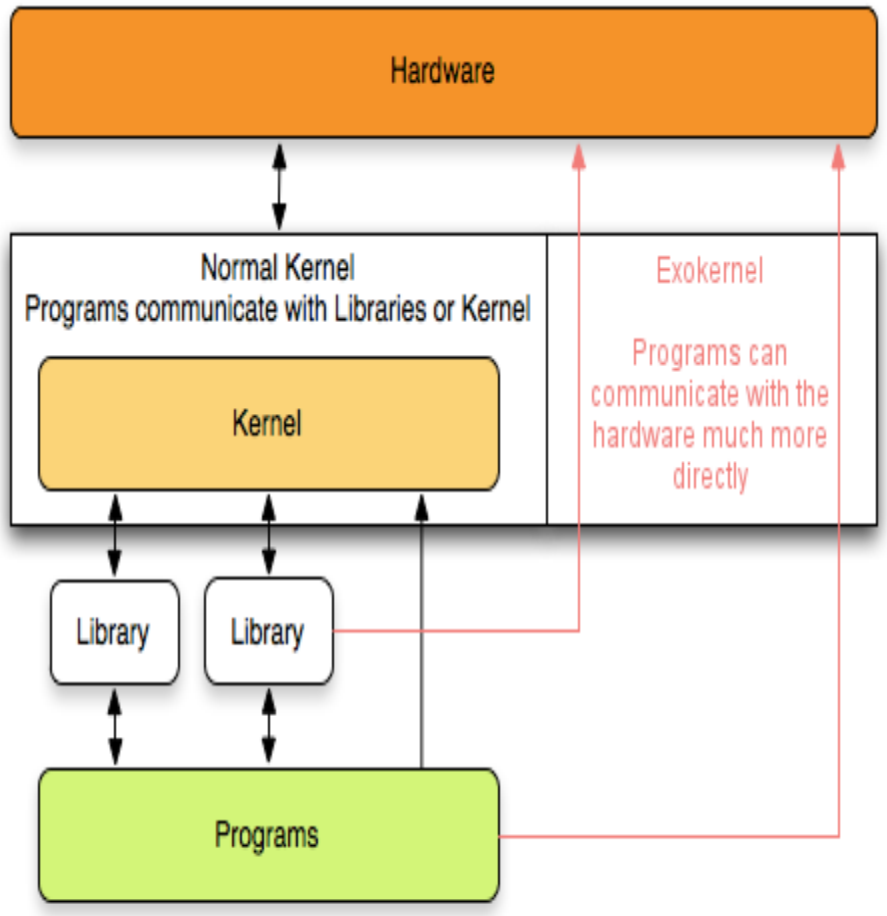
- Microkernels are easier to maintain than monolithic kernels.
- The large number of system calls and context switches might slow down the system because they typically generate more overhead than plain function calls.
- Microkernels generally underperform traditional designs, sometimes dramatically.

Hybrid kernels

- Hybrid kernel is a kernel architecture based on a combination of microkernel and monolithic kernel.
- A hybrid kernel runs some services in the kernel space to reduce the performance overhead of a traditional microkernel, while still running kernel code as servers in the user space.
- A hybrid kernel design may keep the virtual files system and bus controllers inside the kernel and the file system drivers and storage drivers as user mode programs outside the kernel.
- Most modern operating systems use hybrid kernels, including Microsoft Windows NT, 2000 , XP, Windows Server 2003, Windows Vista, Windows Server 2008 and Windows 7.
- DragonFly BSD, a recent *fork* (i.e., variant) of FreeBSD, is the first non-Mach based BSD operating system to employ a hybrid kernel architecture.

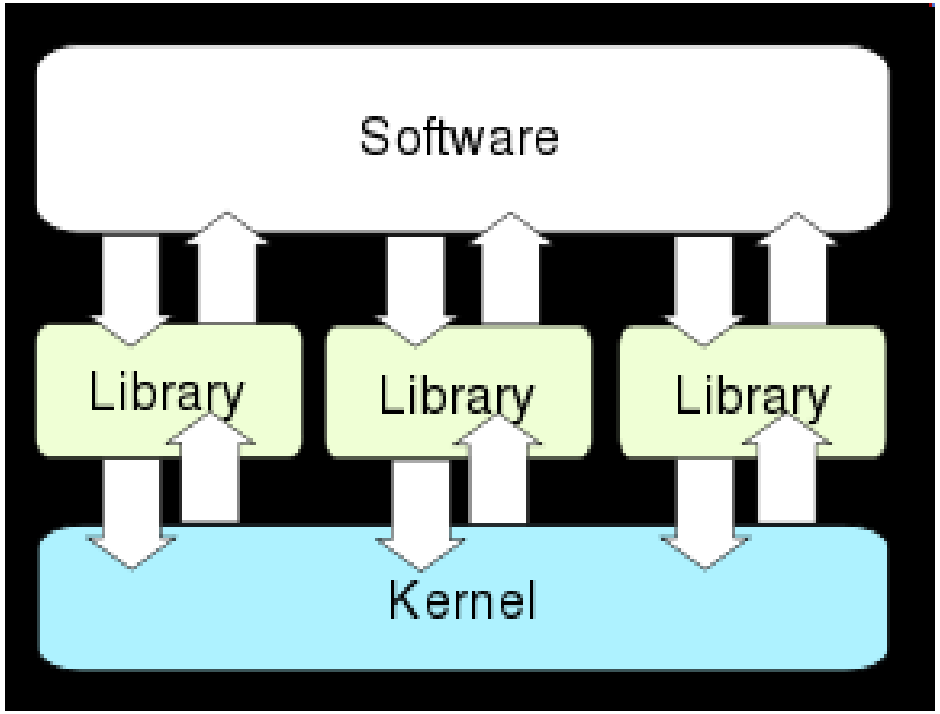
Exo kernels

- MIT developed two exokernel-based operating systems,
- They differ from the other types of kernels in that their functionality is limited to the protection and multiplexing of the raw hardware,
- Implemented Applications are called library operating systems ; they may request specific memory addresses, disk blocks, etc.



Exo kernels

- The kernel only ensures that the requested resource is free, and the application is allowed to access it.
- This low-level hardware access allows the programmer to implement custom abstractions, and omit unnecessary ones, most commonly to improve a program's performance.
- It also allows programmers to choose what level of abstraction they want, high, or low.



Exo kernels

- such as one for Linux and one for Microsoft Windows, thus making it possible to simultaneously run both Linux and Windows applications.

- using two kernels: **Aegis,** a proof of concept with limited support for storage, and **XOK,** which applied the exokernel concept more thoroughly.

Types of operating system

- Few types of Operating system:
    - Batch operating system
    - Time-sharing operating systems
    - Distributed operating System
    - Network operating System
    - Real Time operating System

Batch systems

- In this type of system, there is no direct interaction between user and the computer.
- The user has to submit a job (written on cards or tape) to a computer operator.
- Jobs are batched together by type of languages and requirement.
- Then a special program, the monitor, manages the execution of each program in the batch.
- The monitor is always in the main memory.

Multi Programming batch systems

- This type of operating system, picks and begins to execute one job from memory.
- Once this job needs an I/O operation operating system switches to another job (CPU and OS always busy).
- Jobs in the memory are always less than the number of jobs on disk(Job Pool).
- If several jobs are ready to run at the same time, then system chooses which one to run (CPU Scheduling).
- In Non-multiprogrammed system, there are moments when CPU sits idle and does not do any work.
- In Multiprogramming system, CPU will never be idle and keeps on processing.

Time-Sharing Systems

- Time-Sharing Systems are very similar to Multiprogramming batch systems.
- In fact time sharing systems are an extension of multiprogramming systems.
- In time sharing systems the prime focus is on minimizing the response time, while in multiprogramming the prime focus is to maximize the CPU usage.

Distributed Operating system

- The motivation behind developing distributed operating systems is the availability of powerful microprocessors and advances in communication technology.

- These advancements in technology have made it possible to design and develop distributed systems comprising of many computers that are inter connected by communication networks.

- Following are some advantages of this type of system.
  - As there are multiple systems involved, user at one site can utilize the resources of systems at other sites for resource-intensive tasks.
  - Fast processing.
  - Less load on the Host Machine.

Distributed Systems

- Distribute the computation among several physical processors.

- Loosely coupled system – each processor has its own local memory; processors communicate with one another through various communications lines, such as high-speed buses or telephone lines.

- Advantages of distributed systems.
  - Resources Sharing
  - Computation speed up – load sharing
  - Reliability
  - Communications

Distributed Systems (cont)

- Requires networking infrastructure.
- Local area networks (LAN) or Wide area networks (WAN)
- May be either client-server or peer-to-peer systems.

Desktop Systems

- Personal computers – computer system dedicated to a single user.
- I/O devices – keyboards, mice, display screens, small printers.
- User convenience and responsiveness.
- Can adopt technology developed for larger operating system' often individuals have sole use of computer and do not need advanced CPU utilization of protection features.
- May run several different types of operating systems (Windows, MacOS, UNIX, Linux)

Parallel Systems

- Multiprocessor systems with more than one CPU in close communication.

- Tightly coupled system – processors share memory and a clock; communication usually takes place through the shared memory.

- Advantages of parallel system:
  - Increased throughput
  - Economical
  - Increased reliability
    - graceful degradation
    - fail-soft systems

Parallel Systems (Cont.)

- Symmetric multiprocessing (SMP)
  - Each processor runs and identical copy of the operating system.
  - Many processes can run at once without performance deterioration.
  - Most modern operating systems support SMP

- Asymmetric multiprocessing
  - Each processor is assigned a specific task; master processor schedules and allocated work to slave processors.
  - More common in extremely large systems

Clustered Systems

- Clustering allows two or more systems to share storage.
- Provides high reliability.
- Asymmetric clustering: one server runs the application while other servers standby.
- Symmetric clustering: all N hosts are running the application.

Real-Time Systems

- 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.

- Real-Time systems may be either hard or soft real-time.

Real-Time Systems (Cont.)

- Hard real-time:
  - Secondary storage limited or absent, data stored in short term memory, or read-only memory (ROM)
  - Conflicts with time-sharing systems, not supported by general-purpose operating systems.

- Soft real-time
  - Limited utility in industrial control of robotics
  - Useful in applications (multimedia, virtual reality) requiring advanced operating-system features.

Handheld Systems

- Personal Digital Assistants (PDAs)
- Cellular telephones
- Issues:
  - Limited memory
  - Slow processors
  - Small display screens.

Operating System components

Operating System components

- Introduction to Process management sub system
- Introduction to Memory management sub system
- Introduction to File sub system
- Introduction to Device drivers
- Signals and System calls

Process management sub system

- Process is program in execution.
- Process address space (memory) is divided into four segments :
  - The code (text) segment is made up of the compiled program code, read in from non-volatile storage when the program is launched.
  - The data segments is made up the global and static variables, allocated and initialized prior to executing the main.
  - The heap segment is used for the dynamic memory allocation, and is managed via calls to new, delete, malloc, free, etc.
  - The stack segment is used for local variables. Space on the stack is reserved for local variables when they are declared.

Memory management sub system

- Main Memory refers to a physical memory that is the internal memory to the computer.

- **Dynamic Loading:**
  - All the programs are loaded in the main memory for execution.
  - Sometimes complete program is loaded into the memory, but some times a certain part or routine of the program is loaded into the main memory
  - This mechanism is called Dynamic Loading.

Memory management sub system

- **Dynamic Linking**
  - At times one program is dependent on some other program.
  - In such a case, rather than loading all the dependent programs, CPU links the dependent programs to the main executing program when its required.
  - This mechanism is known as Dynamic Linking.

File sub system

- A file is a collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks.

- The internal representation of a file is given by an inode, which contains a description of the disk layout of the file data and other information such as the file owner, access permissions, and access times.

- The term inode is a contraction of the term index node and is commonly used in literature on the UNIX system.

- Every file has one inode.

Signals and System calls

- Signals are primitive IPC tool
- Are used as event notification sent to a process at any time
  - An event generates a signal
  - OS stops the process immediately
  - Signal handler executes and completes
  - The process resumes where it left off

System calls

- Most programming languages provides a system call interface

  - It serves as the link to system calls made available by the operating system

  - It intercepts function calls in the API and invokes the necessary system call within the operating system

  - Most of the details of the operating system interfaces are hidden from the programmer by the API

Device Drivers

- Device drivers are the kernel modules that control the operation of peripheral devices.

- Broadly there are three types of device drivers
  - Character device driver [keyboard, mouse, printer]
  - Block device driver [Tape drive, HDD, CDROM, ROM, RAM]
  - Network device drivers [HUB, Switches , Bridges]

- **Character Device:** Character device drivers normally perform I/O in a byte stream.

- **Block Device Drivers:** Devices that support a file system are known as block devices.

- **Network device drivers :** Receive and transmit data packets on hardware interface that connect to external systems, and provide a uniform interface that network protocols can access.

Process Management sub system

Process management sub system

- Creating process
- Process Id, process descriptor
- Process contexts
- Process state
- Process address space
- Process scheduling policies
- Threads vs process
- Multi threading and parallel programming

Process Creation

- Through appropriate system calls, such as fork or spawn, processes may create other processes.
- The process which creates other process, is termed the **parent** of the other process, while the created sub-process is termed its **child**.
- Each process is given an integer identifier, termed as process identifier, or PID. The parent PID (PPID) is also stored for each process.
- On a typical UNIX systems the first process created at system start-up time is init, which gives that process PID 1.
- Further Init launches all the system daemons and user logins, and becomes the ultimate parent of all other processes.

Process Creation

- Parent process creates children processes, which, in turn create other processes, forming a tree of processes.
- Resource sharing
  - Parent and children share all resources.
  - Children share subset of parent’s resources.
- Execution
  - Parent and children execute concurrently.
  - Parent waits until children terminate.
- Address space
  - Child duplicate of parent.
  - Child has a program loaded into it.

Process creation: fork()

- The fork() system call is the basic way to create a new process.
- fork() returns zero to child process and PID of child to parent.
- On error fork returns -1
- Child process returns **SIGCHILD** to parent process when it terminates, till parent acknowledges the CHILD remains in **zombie state**
- If parent terminates before child , then child becomes orphan process.
- In UNIX/LINUX init process becomes parent to orphan process.

Process creation: fork() - example

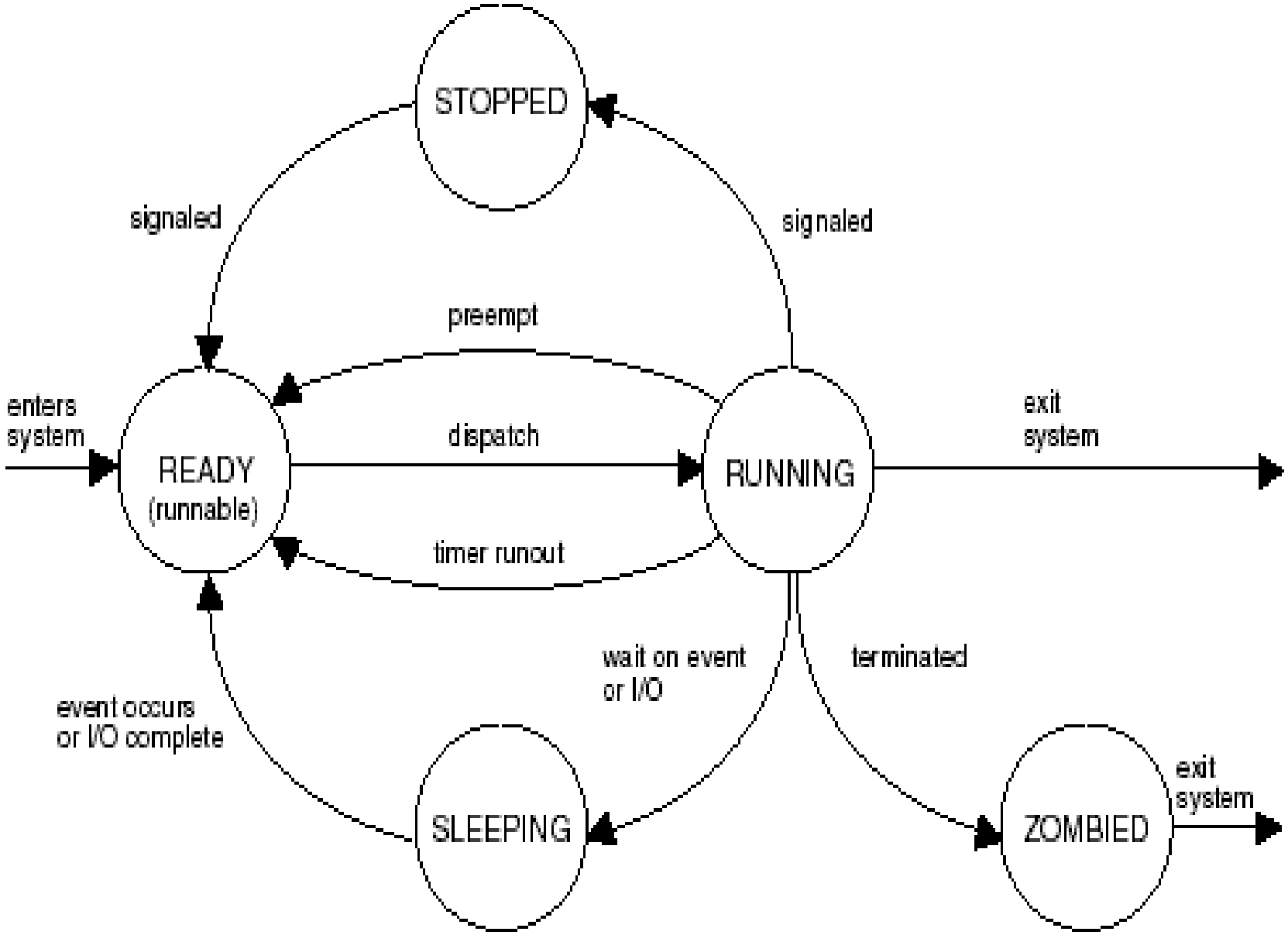
```
main() {
        int pid,fd;
        if((pid =fork())== 0) {
                printf(" Child process message printed\n");
                printf("pid is %d\n",getpid());
                printf("Parent process pid is %d\n",getppid());
                }
}
```

Process State

- As a process executes it changes state according to its circumstances.

- Standard Unix/Linux processes have the following states:
  - Ready
  - Running
  - Wait / Sleep
  - stopped
  - zombie

Process State



Process scheduling

- The act of determining which process in the ready state should be moved to the running state is known as Process Scheduling.

- The prime aim of the process scheduling system is to keep the CPU busy all the time and to deliver minimum response time for all programs.

- Schedulers fell into one of the two general categories :
  - **Non pre-emptive scheduling.** When the currently executing process gives up the CPU voluntarily.

  - **Pre-emptive scheduling.** When the operating system decides to favour another process, pre-empting the currently executing process

CPU Scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
- CPU scheduling decisions may take place when a process:
  - 1. Switches from running to waiting state.
  - 2. Switches from running to ready state.
  - 3. Switches from waiting to ready.
  - 4. Terminates.

- Scheduling under 1 and 4 is non preemptive.
- All other scheduling is preemptive.

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program

- Dispatch latency – time it takes for the dispatcher to stop one process and start another running.

Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particular process
- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

Processor Scheduling

- Have a separate run queue for each processor
- Each processor only selects processes from its own queue to run
- Yes, it’s possible for one processor to be idle while others have jobs waiting in their run queues
- Periodically, the queues are rebalanced: if one processor’s run queue is too long, some processes are moved from it to another processor’s queue.

Scheduling Algorithms

- We'll discuss four major scheduling algorithms here which are following :

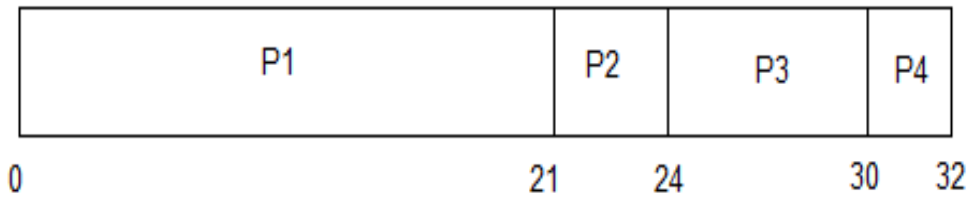
- First Come First Serve(FCFS) Scheduling
- Shortest-Job-First(SJF) Scheduling
- Priority Scheduling
- Round Robin(RR) Scheduling

First-Come, First-Served (FCFS) Scheduling

- Jobs are executed on first come, first serve basis.
- Easy to understand and implement.
- Poor in performance as average wait time is high.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The average waiting time will be = (0 + 21 + 24 + 30)/4 = 18.75 ms



This is the GANTT chart for the above processes

Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.

- Two schemes:
  - Non preemptive – once CPU given to the process it cannot be preempted until completes its CPU burst.
  - preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF).

- SJF is optimal – gives minimum average waiting time for a given set of processes.

Example of Non-Preemptive SJF

Process	Burst Time
P1	7
P2	5
P3	1
P4	4

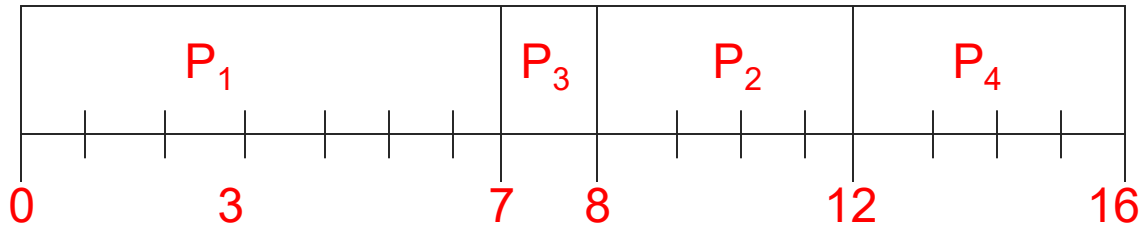
SJF (non-preemptive)

- Average waiting time = ???

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

SJF (non-preemptive)

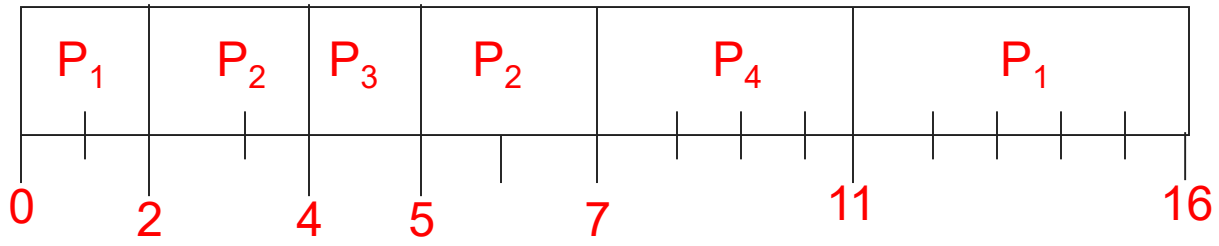


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

Example of Preemptive SJF

Process	Arrival Time	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

- SJF (preemptive)



- Average waiting time = (9 + 1 + 0 +2)/4 = 3

Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority).
- SJF is a priority scheduling where priority is the predicted next CPU burst time.
- Problem ≡ Starvation – low priority processes may never execute.
- Solution ≡ Aging – as time progresses increase the priority of the process.

Priority Scheduling: Example

Process	Duration	Priority	Arrival Time
P1	6	4	0
P2	8	1	0
P3	7	3	0
P4	3	2	0

P2 waiting time: 0
P4 waiting time: 8
P3 waiting time: 11
P1 waiting time: 18

The average waiting time (AWT):
(0+8+11+18)/4 = 9.25
(worse than SJF)

16

Round Robin (RR)

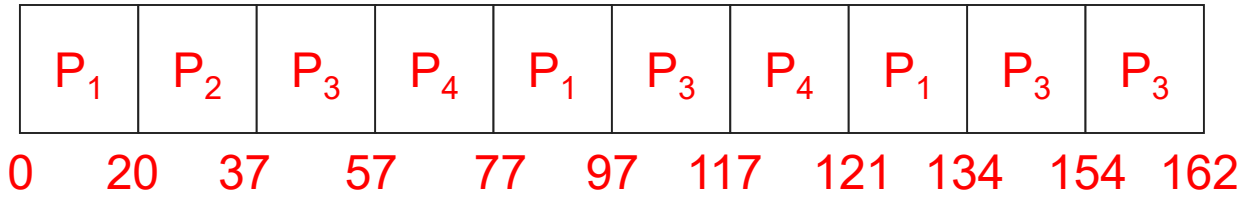
- Each process gets a small unit of CPU time (time quantum), usually 5-600 milliseconds.
- After this time has elapsed, the process is preempted and added to the end of the ready queue.

- If there are n processes in the ready queue and the time quantum is q, then each process gets 1/n of the CPU time in chunks of at most q time units at once.
- No process waits more than (n-1)q time units.
- Performance
  - q large $\Rightarrow$ FIFO
  - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high.

Example of RR with Time Quantum = 20

Process	Burst Time
P1	53
P2	17
P3	68
P4	24

The Gantt chart is:



- Typically, higher average turnaround than SJF, but better response.

Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available.
- Homogeneous processors within a multiprocessor.
- Load sharing
- Asymmetric multiprocessing – only one processor accesses the system data structures, alleviating the need for data sharing.

Real-Time Scheduling

- Linux has soft real-time scheduling
- Processes with priorities [0, 99] are real-time
- All real-time processes are higher priority than any conventional processes
  - Two real-time scheduling systems, FCFS and round-robin
  - First-come, first-served: process is only preempted for a higher-priority process; no time quanta
- Round-robin: real-time processes at a given level take turns running for their time quantum

Threads

Threads

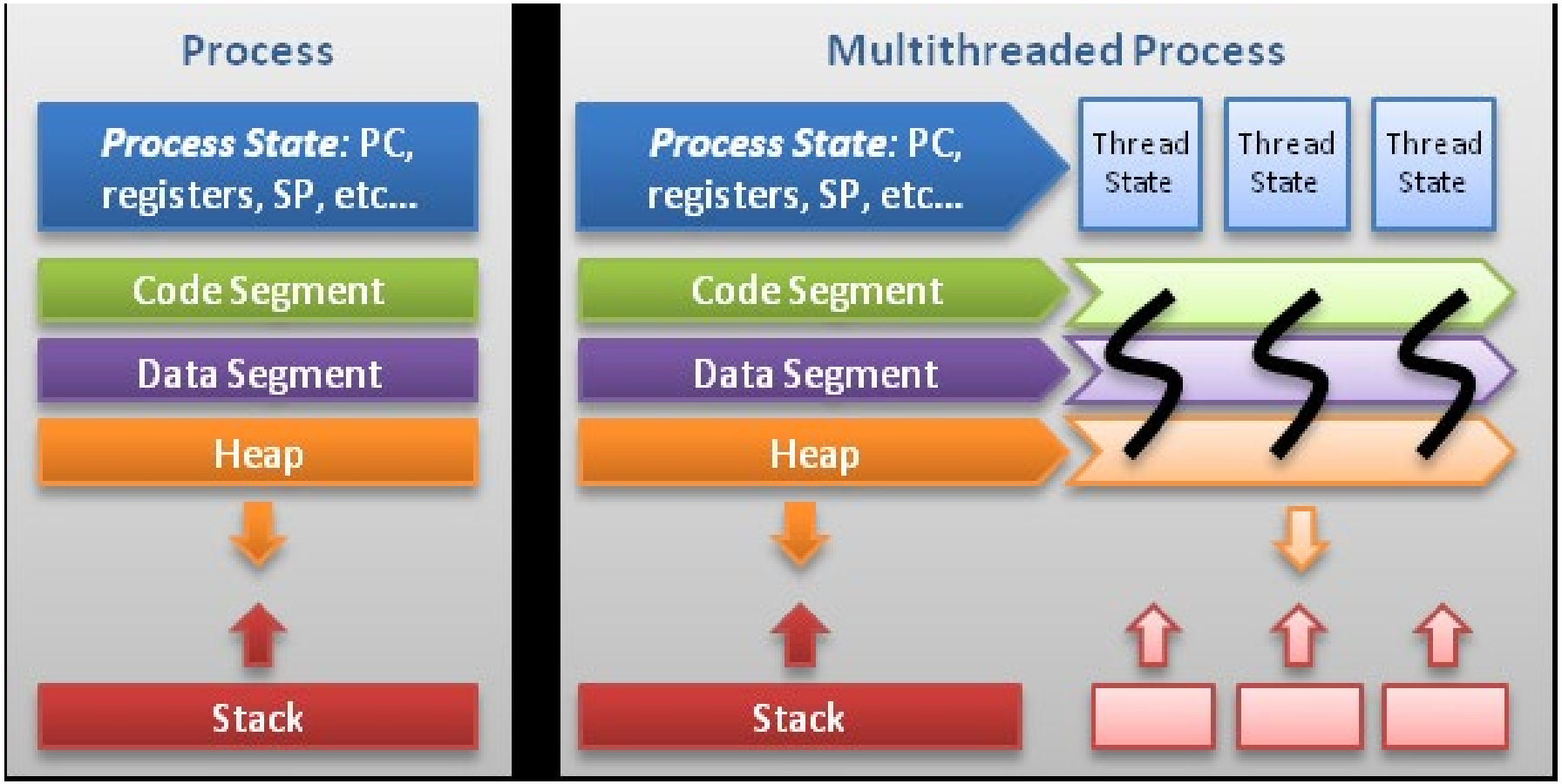
- Overview
- Multithreading Models
- Threading Issues
- P threads

Thread

- Thread is an execution unit which consists of its own program counter, a stack, and a set of registers.
- Threads are also known as Lightweight processes.
- Threads are popular way to improve application through parallelism.
- The CPU switches rapidly back and forth among the threads giving illusion that the threads are running in parallel.

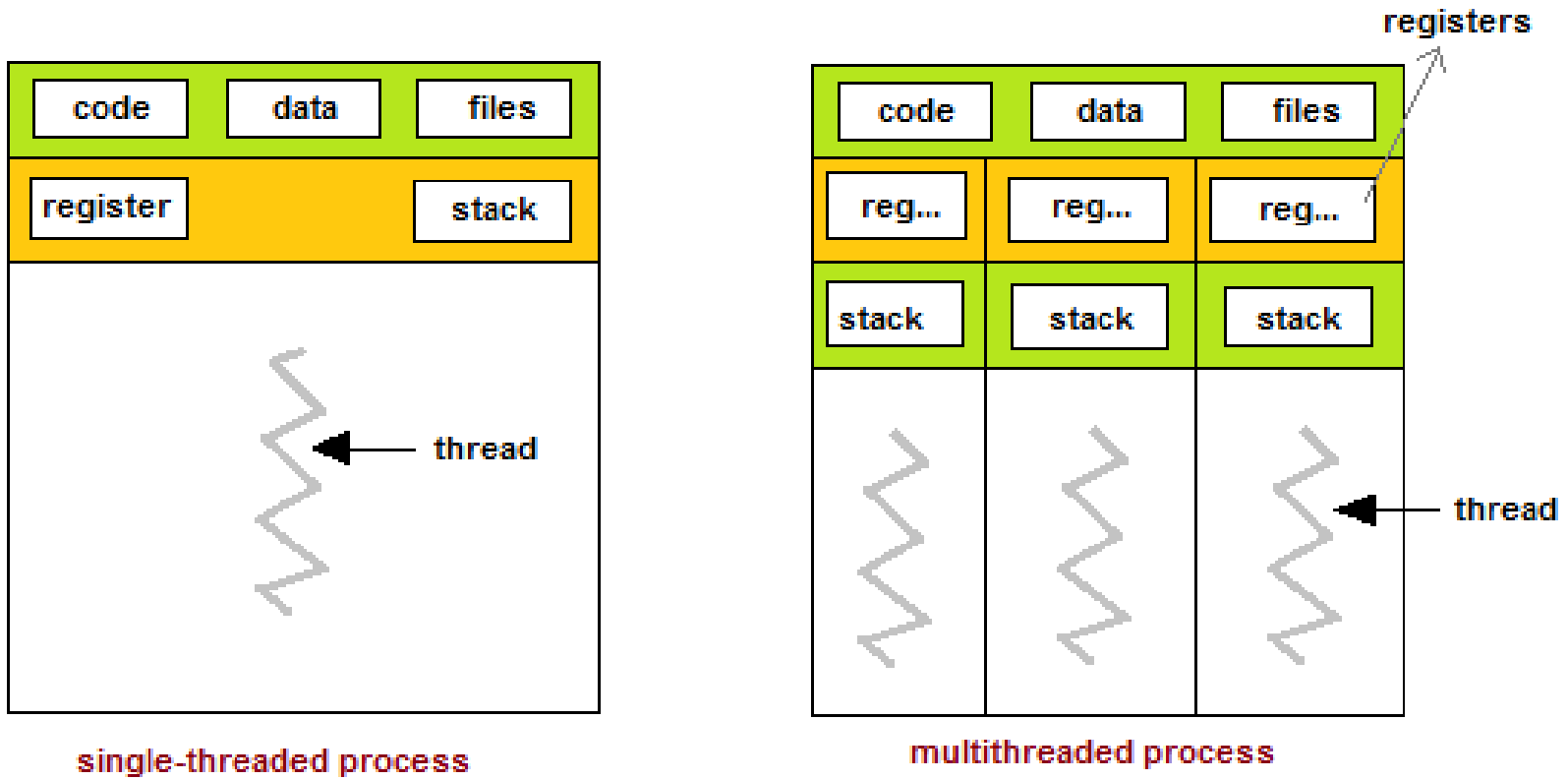
Multi threading

Multiple Flows of Execution / Control



Multiple Threads

- As each thread has its own independent resource for process execution, multiple processes can be executed parallel by increasing number of threads



Benefits

- High Responsiveness
- Resource sharing, hence allowing better utilization of resources.
- Economy. Creating and managing threads becomes easier.
- Scalability. One thread runs on one CPU. In Multithreaded processes, threads can be distributed over a series of processors to scale.
- Context Switching is smooth. Context switching refers to the procedure followed by CPU to change from one task to another.

Thread credentials

- Thread operations include thread creation, termination, synchronization (joins , blocking), scheduling, data management and process interaction.
- A thread does not maintain a list of created threads, nor does it know the thread that created it.
- All threads within a process share the same address space.

- Threads in the same process share:
  - Process instructions
  - open files (descriptors)
  - signals and signal handlers
  - current working directory
  - User and group id

Thread credentials

- Each thread has a unique:
  - Thread ID
  - set of registers, stack pointer
  - stack for local variables, return addresses
  - signal mask
  - Priority, scheduling policy

Difference Between threads and processes

- Processes do not share their address space while threads executing under same process share the address space.

- Since processes execute independent of each other and the synchronization between processes is taken care by kernel, while on the other hand the thread synchronization has to be taken care by the process under which the threads are executing.

- Context switching between threads is fast as compared to context switching between processes

- The interaction between two processes is achieved only through the standard inter process communication while threads executing under the same process can communicate easily as they share most of the resources like memory, text segment so on…

Types of Thread

- There are two types of threads :
  - User Threads
  - Kernel Threads

User Threads

- Thread management done by user-level threads library
- Thread library contains code for creating and destroying threads, passing messages and data between threads, for scheduling thread execution and for saving and restoring thread contexts
- Examples
- - POSIX P threads
- - Mach C-threads
- - Solaris threads

Kernel Threads

- Supported only by the Kernel

- Thread management is done by the kernel

- Advantage: If one thread in a process is blocked, kernel can schedule another thread of the same process.

  Disadvantage: Transfer of control from one thread to another within the same process requires a mode switch to the kernel

- Examples
- - Windows 95/98/NT/2000
- - Tru64 UNIX
- - Linux
- - Solaris
- - BeOS

File Sub System

File sub system

- File sub system
- Importance of File system
- File representation in the system
- File creation and maintenance
- Block devices and file system

What are filesystems?

- A *filesystem is the methods and data structures that an operating system uses to keep track of files on a disk or* partition; that is, the way the files are organized on the disk.

- Types of file system
- Vfat
- Fat32
- Ntfs
- Ext,ext2,ext3 and ext4
- **ZFS** for Solaris, **JFS** and derived file systems for Unix
- Clustered file systems (ZFS,VMFS,GFS)

Fat file system

- FAT stands for "File Allocation Table".
- The full path of file including filename can be up to 255 characters long.
- FAT does not support local and folder security.
- FAT provides quick access to files.
- The speed of file access depends on file type, file size, partition size, fragmentation and number of files in a folder.

FAT32 File System

- FAT32 is an advanced version of FAT file system.
- It can be used on drives from 512 MB to 2TB in size.
- FAT32 increases the number of bits used to address cluster.
- A cluster is a set of sectors.
- It supports larger disk (up to 2TB) and better storage efficiency.

NTFS File System

- Features of NTFS File System
  - File names can be up to 255 characters
  - File names can contain most characters except “ / \ * | :
  - File names are not case sensitive

- Security
  - NTFS provides file and folder security.
  - Files and folders are safer than FAT.
  - Security is maintained by assigning NTFS permissions to files and folders.
  - Security is maintained at the local level and the network level.

ReFS (Resilient File System):

- ReFS is the latest development of Microsoft presently available for Windows 8 Servers.

- file systems and is mainly organized in form of B+-tree.

Ext2, Ext3, Ext4 - 'native' Linux file system.

- Ext2 stands for second extended file system.
  - It was introduced in 1993. Developed by Rémy Card.
  - This was developed to overcome the limitation of the original ext file system.
  - On flash drives, usb drives, ext2 is recommended, as it doesn’t need to do the over head of journaling.
  - Maximum individual file size can be from 16 GB to 2 TB
  - Overall ext2 file system size can be from 2 TB to 32 TB

Exat3 file system

- Ext3 stands for third extended file system.
- You can convert a ext2 file system to ext3 file system directly
- Maximum individual file size can be from 16 GB to 2 TB
- Overall ext3 file system size can be from 2 TB to 32 TB

Ext4 file system

- Ext4 stands for fourth extended file system.
- Supports huge individual file size and overall file system size.

- Maximum individual file size can be from 16 GB to 16 TB

- Overall maximum ext4 file system size is 1 EB (exabyte). 1 EB = 1024 PB (petabyte). 1 PB = 1024 TB (terabyte).

- Directory can contain a maximum of 64,000 subdirectories (as opposed to 32,000 in ext3)

- You can also mount an existing ext3 fs as ext4 fs (without having to upgrade it).

What are filesystems?

- A *filesystem is the methods and data structures that an operating system uses to keep track of files on a disk or* partition; that is, the way the files are organized on the disk.

- Most UNIX filesystem types have a similar general structure.

- *Each* file system starts with a *boot block. This block is reserved for the code required to boot the* operating system

boot block

- A ***boot block*** located in the first few sectors of a file system.
- The boot block contains the initial bootstrap program used to load the operating system.
- Typically, the first sector contains a bootstrap program that reads in a larger bootstrap program from the next few sectors, and so forth.

Super block

- *The superblock* contains information about the filesystem as a whole, such as its size.

- An inode contains all information about a file, except its name. The name is stored in the directory, together with the number of the inode.

- A directory entry consists of a filename and the number of the inode which represents the file.

Inode block

- The inode contains the numbers of several data blocks, which are used to store the data in the file.
- There is space only for a few data block numbers in the inode, however, and if more are needed, more space for pointers to the data blocks is allocated dynamically.
- These dynamically allocated blocks are indirect blocks; the name indicates that in order to find the data block, one has to find its number in the indirect block first.

Inode block

- An inode is the “handle” to a file and contains the following information:
    - file ownership indication
    - file type (e.g., regular, directory, special device, pipes, etc.)
    - file access permissions. May have setuid (sticky) bit set.
    - time of last access, and modification
    - number of links (aliases) to the file
    - pointers to the data blocks for the file
    - size of the file in bytes (for regular files), major and minor device numbers for special devices.

Data Block

- *Data blocks* blocks containing the actual contents of files.

- The Unix file system allocates *data blocks* (blocks that contain a file's contents) one at a time from a pool of free blocks.

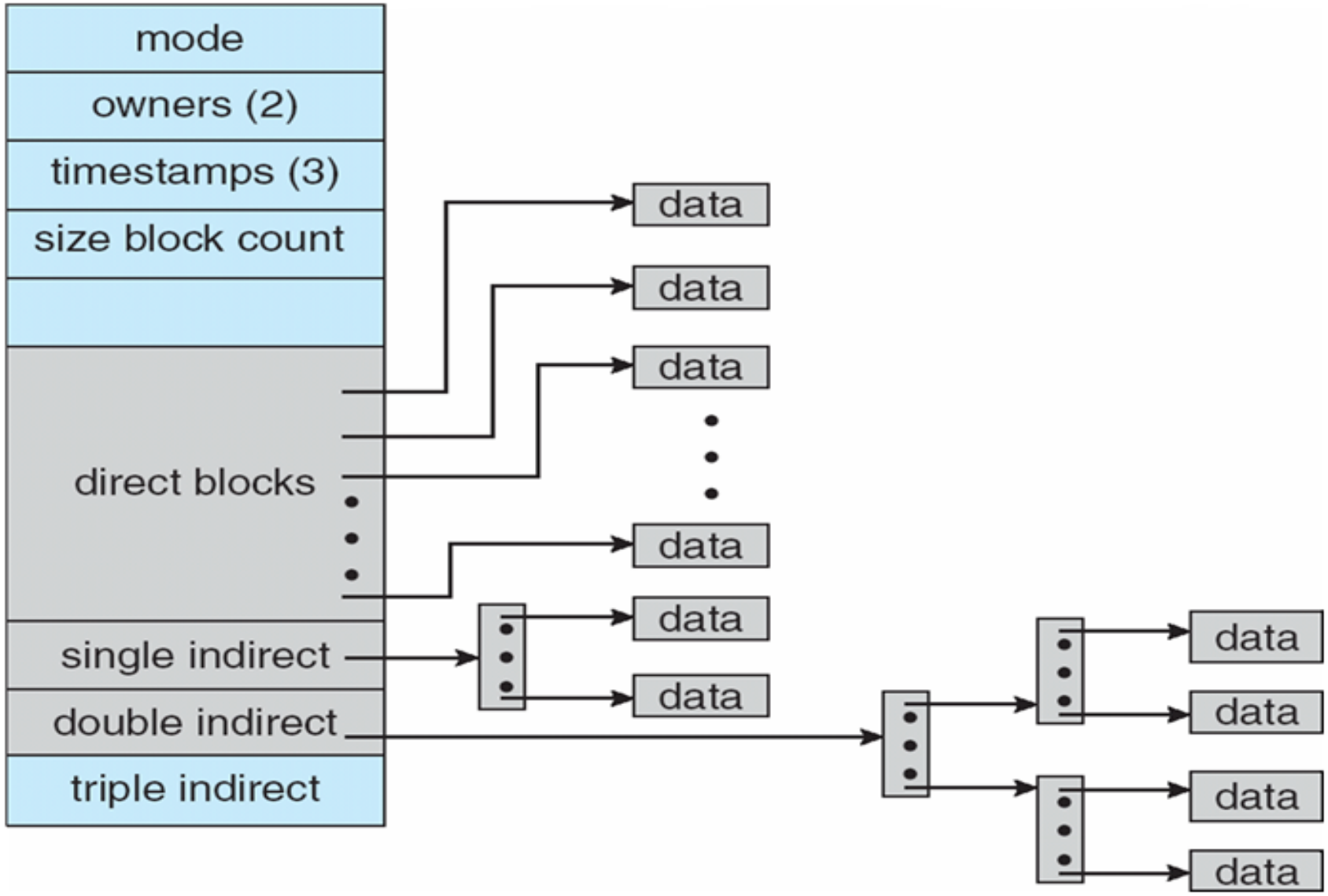
- Unix uses 512, 1K, 2K or 4K blocks (block size can be customised).

- Moreover, a file's blocks are scattered randomly within the physical disk.

Data Block

- Inodes include pointers to the data blocks.
  - Each inode contains 15 pointers:
  - the first 12 pointers point directly to data blocks
  - the 13th pointer points to an indirect block, a block containing pointers to data blocks
  - the 14th pointer points to a doubly-indirect block, a block containing 128 addresses of singly indirect blocks
  - the 15th pointer points to a triply indirect block (which contains pointers to doubly indirect blocks, etc.)

Data Block


mode
owners (2)
timestamps (3)
size block count
direct blocks
single indirect
double indirect
triple indirect
data
data
data
data
data
data
data
data
data

Data Block

- Assume one block is of 1 Kbytes and a block number is an integer of 4 bytes (32 bits). Thus a block can have 256 block numbers.

- 10 direct blocks => 10 K bytes
- 1 single indirect block with 256 block number entries => 256 K bytes
- 1 double indirect block with 256 single indirect entries => 64 M bytes
- 1 triple indirect block with 256 double indirect entries => 16 G bytes

- which is far more then what the 4-byte memory address can handle (2^32 => 4 G bytes).

Device Drivers

Device Driver

- The CPU is not the only intelligent device in the system, every physical device has its own hardware controller.
- The keyboard, mouse and serial ports are controlled by a SuperIO chip, the IDE disks by an IDE controller, SCSI disks by a SCSI controller and so on.
- Each hardware controller has its own control and status registers (CSRs) and these differ between devices.
- The CSRs for an Adaptec 2940 SCSI controller are completely different from those of an NCR 810 SCSI controller.

Device Driver

- The CSRs are used to start and stop the device, to initialize it and to diagnose any problems with it.
- Instead of putting code to manage the hardware controllers in the system into every application, the code is kept in the Linux kernel.
- The software that handles or manages a hardware controller is known as a device driver.
- All hardware devices look like regular files; they can be opened, closed, read and written using the same, standard, system calls that are used to manipulate files.

Device Driver

- One class of module is the device driver, which provides functionality for hardware like a TV card or a serial port.
- On unix, each piece of hardware is represented by a file located in /dev named a device file
- which provides the means to communicate with the hardware.
- The device driver provides the communication on behalf of a user program.
- So the es1370.o sound card device driver might connect the /dev/sound device file to the Ensoniq IS1370 sound card.
- A userspace program like mp3blaster can use /dev/sound without ever knowing what kind of sound card is installed.

Major and Minor Numbers

- Let's look at some device files.

  - Here are device files which represent the first three partitions on the primary master IDE hard drive:

```
# ls -l /dev/hda[1-3]
brw-rw----   1 root   disk   3, 1 Jul   5   2000 /dev/hda1
brw-rw----   1 root   disk   3, 2 Jul   5   2000 /dev/hda2
brw-rw----   1 root   disk   3, 3 Jul   5   2000 /dev/hda3
```

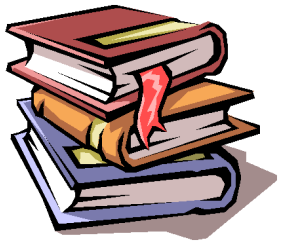
Major and Minor Numbers

- Notice the column of numbers separated by a comma? The first number is called the device's major number.
- The second number is the minor number.
- The major number tells you which driver is used to access the hardware.
- Each driver is assigned a unique major number; all device files with the same major number are controlled by the same driver.
- All the above major numbers are 3, because they're all controlled by the same driver.

Major and Minor Numbers

- Devices are divided into two types: character devices and block devices.
- The difference is that block devices have a buffer for requests, so they can choose the best order in which to respond to the requests.
- whereas character devices are allowed to use as many or as few bytes as they like.
- Most devices in the world are character, because they don't need this type of buffering, and they don't operate with a fixed block size.

References



1. Linux man pages: http://linux.die.net/man/

2. Operating System Concepts
by Abraham Silberschatz

3. Advanced Programming in the UNIX Environment
by W. Richard Stevens

Thank You

Learning & Development, Tata Elxsi, Bangalore.

ITPB Road Whitefield

Bangalore 560 048 India
campus_elxsi@tataelxsi.co.in

www.tataelxsi.com