# Course Title: Linux System Programming

**Faculty: Rajesh Sola, Srinivas.K & Bharath.G**

L&T Technology Services

LTTS GLOBAL ENGINEERING ACADEMY

# Content

- OS Architecture, Kernel, System calls
- Process Management, Signal Handling
- Threads, File System Concepts
- IPC, Memory Management

# Linux OS Architecture & Kernel

# Operating system

- Operating System is an interface between the user and the system hardware

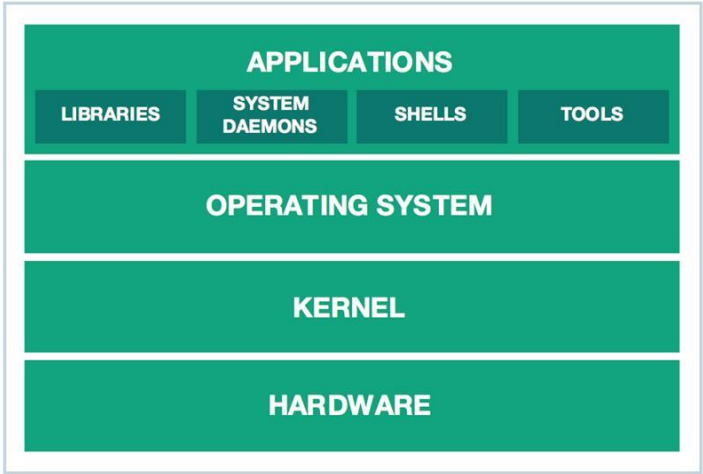
**Types of Operating system**

- Batch OS
- Time-sharing OS (Linux)
- Distributed OS
- Network OS
- Real-time OS

For more info Refer : https://www.tutorialspoint.com/operating_system/os_types.htm

# Linux OS Architecture

- Kernel
  - Core of OS, responsible for all major OS activities, interacts with hardware, provides abstraction to hardware from system / application programs
- Drivers
  - Used for interaction for additional hardware & I/O
- System Libraries
  - Special programs used by system / application programs access kernel's features, implement most of the functionalities of OS
  - Multimedia library, Network library
- System Utilities
  - Used to do specialized, individual level tasks
  - Shell, Terminal



# Kernel

- Mandate component of Operating System
- Resides in memory all the time, rest all depending on kernel
- Provides basic services including memory management, IO management & other management services
- Provides services to application and libraries in the form of SYSTEM CALLS
- Detailed Info: https://en.wikipedia.org/wiki/Linux_kernel

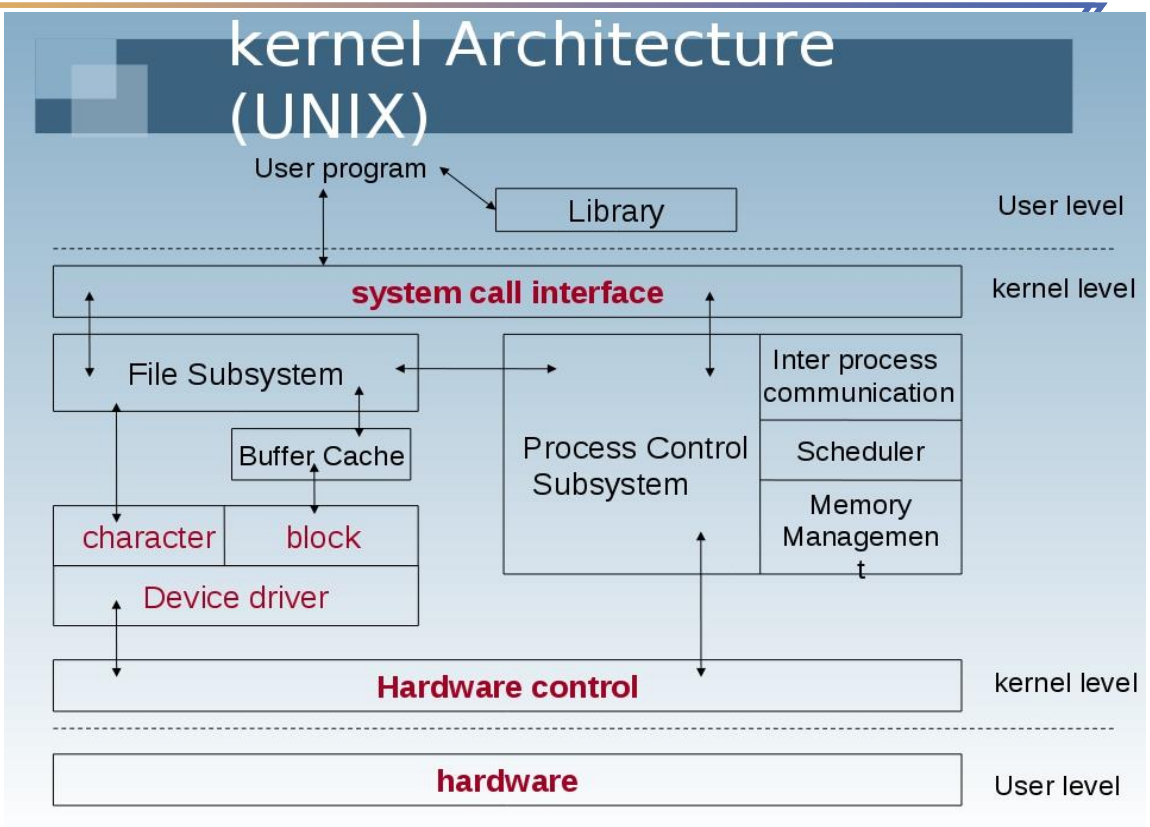



Image Source: The Design of the Unix Operating System, Maurice Bach

# Types of Kernel

- **Micro Kernel**

CPU, memory and IPC in kernel mode, everything else is accessory and runs in user mode.

  - + Portability, Small install footprint, Small memory footprint, Security
  - - Hardware is more abstracted through drivers, may react slower because drivers are in user mode
  - - Processes have to wait in a queue to get information, can't get access to other processes without waiting

- **Monolithic Kernel**

CPU, memory, IPC + device drivers, file system management, and system server calls in kernel mode

  - + More direct access to hardware for programs, Easier for processes to communicate between each other
  - + If your device is supported, it should work with no additional installations, processes react faster because there isn't a queue for processor time
  - - Large install footprint, Large memory footprint, Less secure because everything runs in supervisor mode

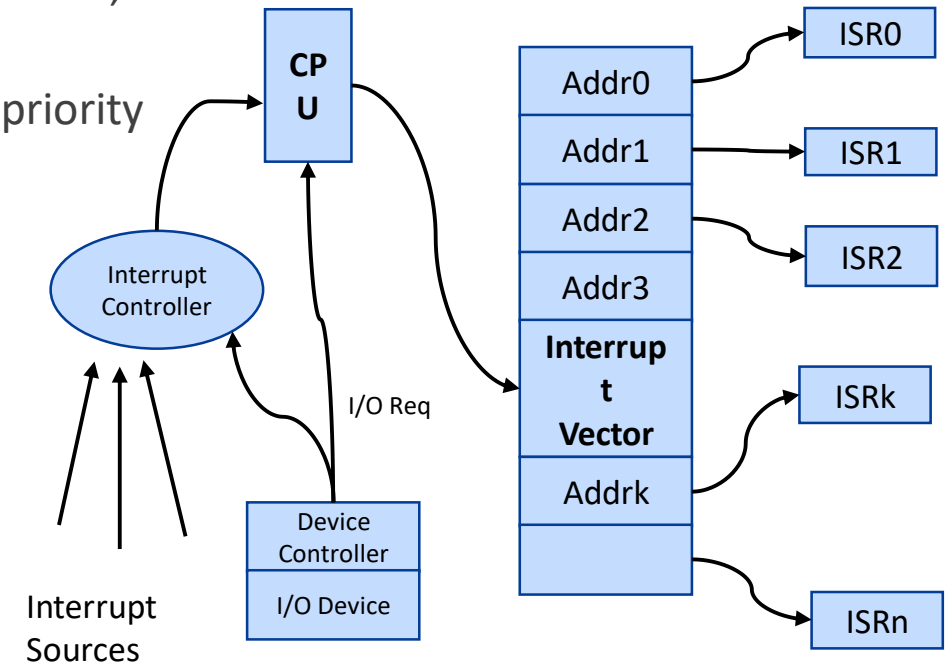
- **Modular kernel :**Linux is modular kernel type, is combination of both monolithic & micro kernel
  - Has collection of both statically loaded & dynamically loadable modules
    - +No need to load everything on boot, less boot time, less size, new need to recompile to add new module
    - -Chances of losing stability, Security Compromise with modules, Coding can be difficult

# Linux OS Kernel

- Compressed Kernel is stored at /boot/vmlinu*
- Dynamic modules of kernel  /lib/modules
- uname –r
  - 5.4.0-33-generic
    (major.minor.release-tagname)
- Versions of kernel
  - 2.x, 2.4, 2.5, 2.6, 3.x, 4.x, 5.x
  - 5.x is current version

# Interrupts

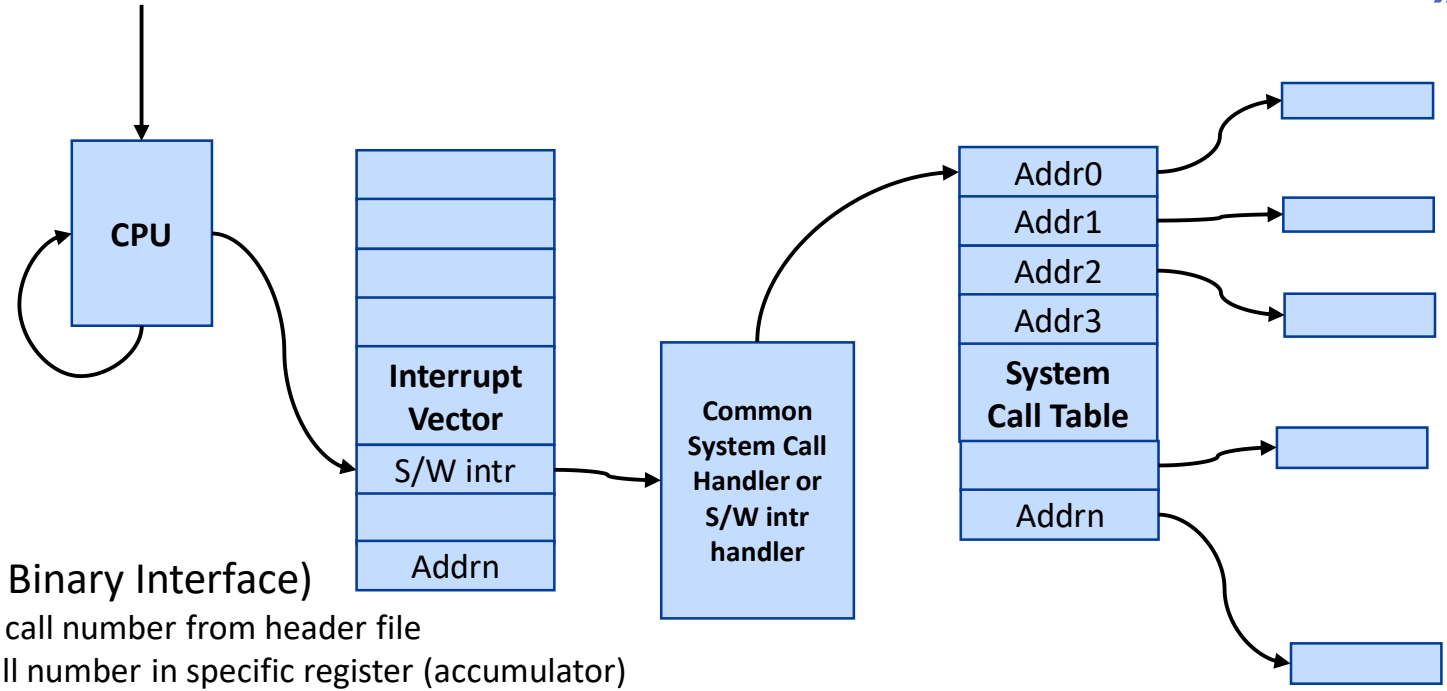
- Asynchronous events
- IRQ (Interrupt Request), Interrupt Vector Table, ISR (Interrupt Service Routine)
- Interrupts must be serviced with utmost priority
- ISR should be as short as possible with no/minimal blocking calls
- Maskable & Non-maskable Interrupts
- Types
  - Hardware Interrupts
  - Software Interrupts



# System calls

- Interface to OS Services, Communication between Kernel mode and User mode.
- System calls initiated by user space, executed by kernel space
- System calls are also referred as software interrupts
- Identified by Unique Number
- Written in C or Assemble within Kernel space
- System call offers the services of the operating system to the user programs via API (Application Programming Interface)
- Follows “standard protocol” for parameter flow and return values
  - No common memory between user space and kernel space, hence system calls use REGISTERS for communications
  - If arguments are more than available registers, then arguments are packed in structures or blocks and address is passed in register
- man syscall → details of registers

# System Calls

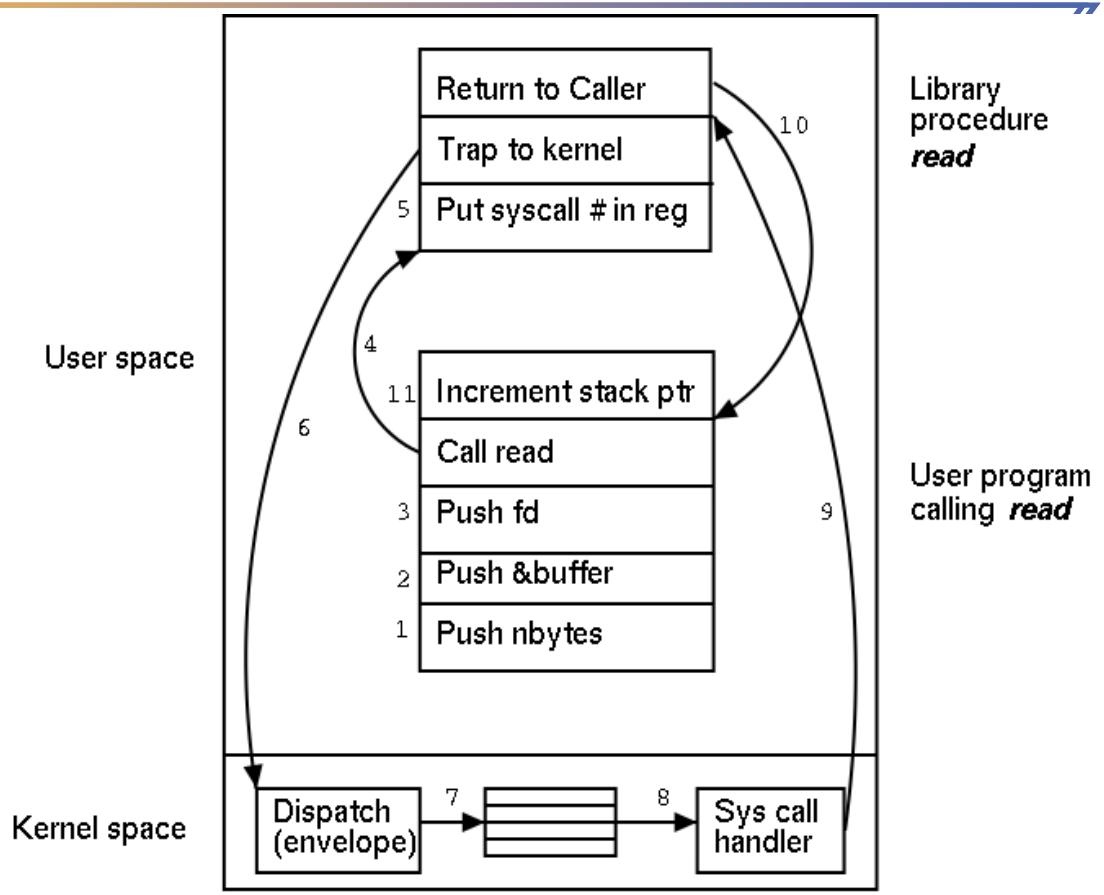



- ABI (Application Binary Interface)
  - Identify system call number from header file
  - Store system call number in specific register (accumulator)
  - Store parameters in other registers
  - Initiate TRAP instruction
- On execution, system call always returns 0 or positive for SUCCESS & negative for FAIL

# System calls

Types of System calls
- Process Control
- File Management
- Device Management
- Information Maintenance
- Communications
- Protection

- Write system call:
  - printf in C, echo in shell, cout in C++
- Read system call:
  - scanf in C, cin in C++
- Trace the system calls:
  - strace man, echo, cp, cat
  - man strace
- list of system call numbers
  - /usr/include/asm*/unistd.h



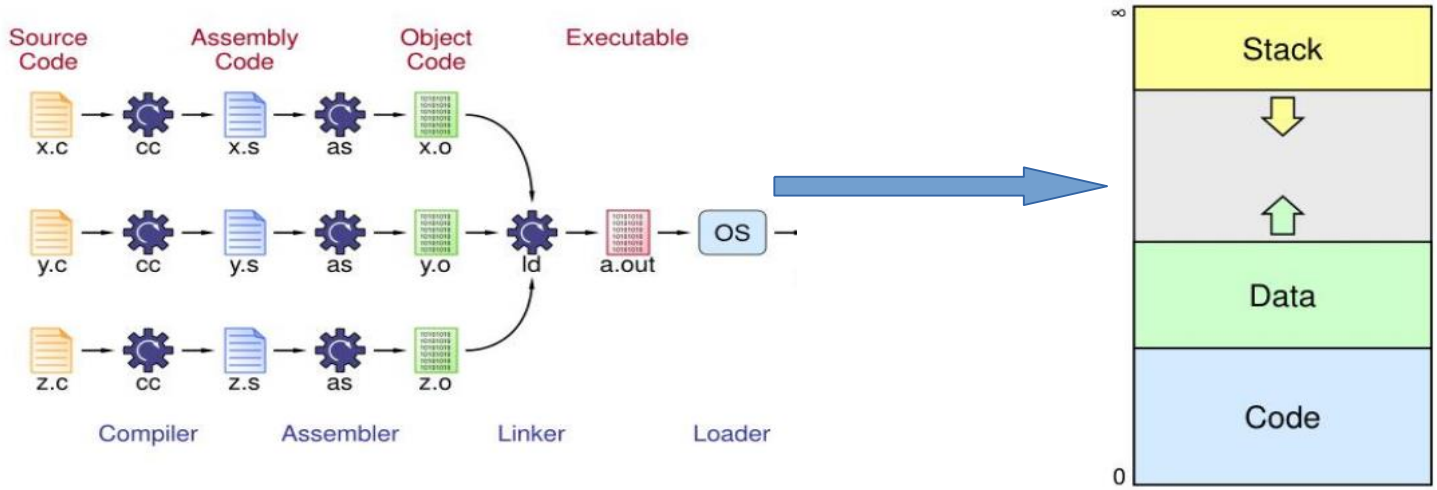
# System calls

- Example folder **“Intro”**
- Standard file descriptors used by any process
  - fd = 0 (stdin) (read)
  - fd = 1 (stdout) (write)
  - fd = 2 (stderr)
- “perror” – always appends error message based on return value of system call

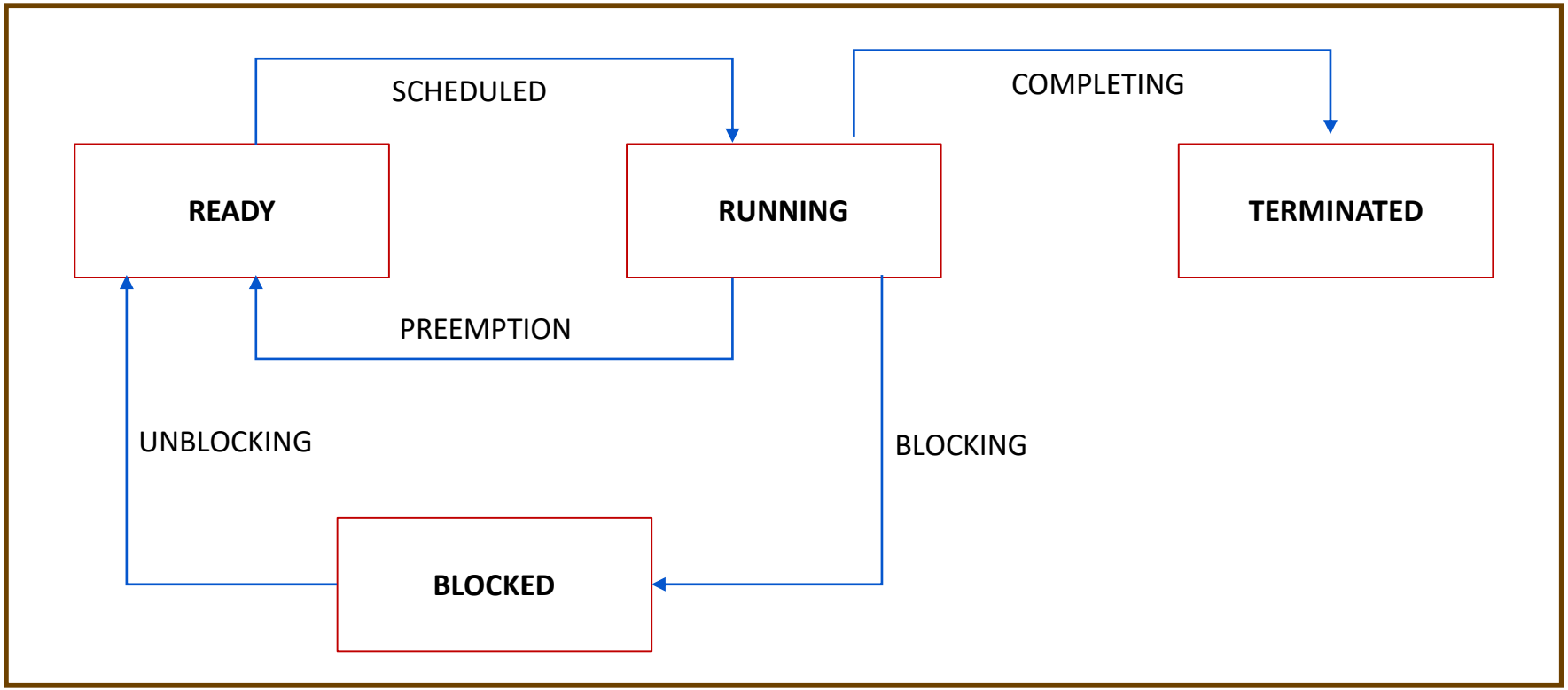
# Process management

# Process

- **Process is a Program under execution**
- Program is passive entity and process is active entity
- Every process has its own independent stack
- Kernel maintains process list table in the form of doubly linked list
- Each process has a unique id (pid)

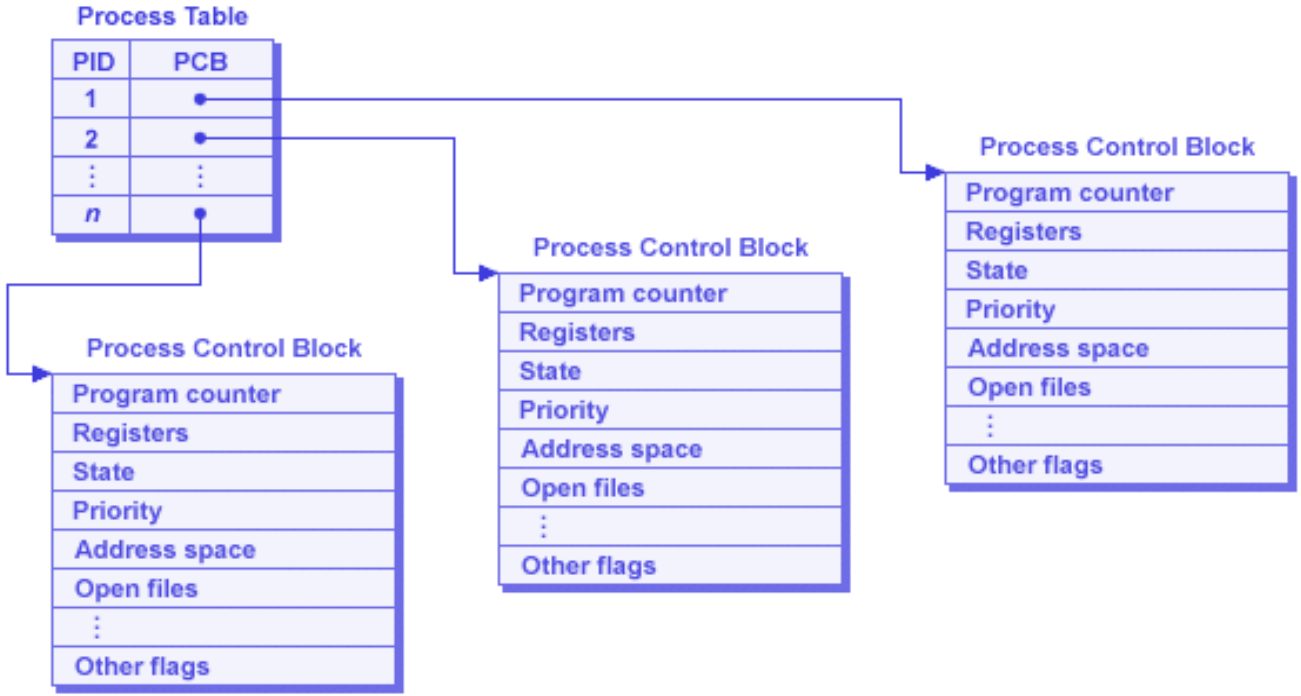


# Process Life Cycle



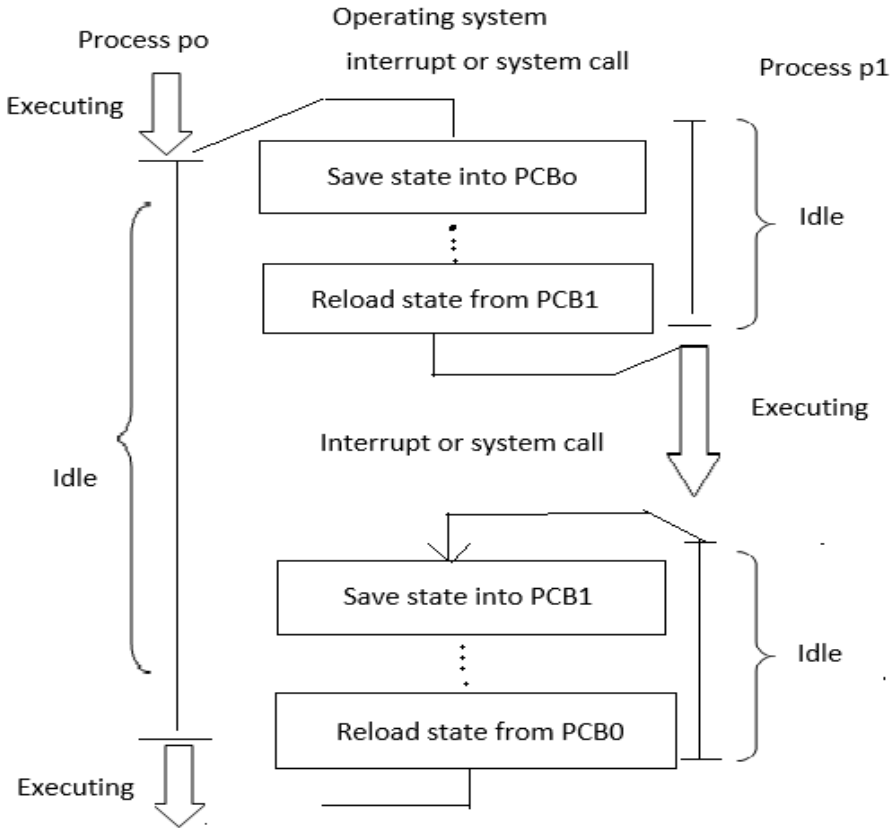
# Process Table and Process Control Block

- Process table is maintained by kernel
- Process Control Block



# Context Switch

- **Context saving**
  - Copy data from CPU to SAVE AREA
- **Context loading**
  - Copy data from SAVE AREA to CPU
- **Context switching**
  - combination of context saving and context loading
  - Switching CPU from one Process to other
  - Occurs on Interrupt or system call or pre-emption
- Every process has individual process stack in user space and kernel space to store register values on context switching



# Process

- To start a process in Background use **&** symbol in command
  - cat file.txt &
  - jobs
- Lis the running processes
  - ps , ps –f
- Stop a process
  - kill -9, pkill
- Parent and Child
  - Each Process(PID) has a parent (PPID)
- Zombie and Orphan Process
  - Orphan process is one whose parent is killed/terminated before itself.
  - Processes which completed the execution but still have entry in process table.
- Daemon Processes
  - Processes that run in background

# Process Hierarchy

- Every process has parent process
- a.out → shell → terminal → .. → init with PID = 1
- init is considered as the ORIGIN of linux process hierarchy
- Commonly used commands
  - pstree, pstree –np
  - top
  - ps, ps –el, ps aux, ps –e –o pid,ppid,stat,cmd
  - pgrep
  - kill, killall, kill -9, pkill

# New Process Creation

**fork**

- Creates a new process known as child process
- New pid, process control block (PCB) / process descriptor (PD) will be allocated to child (new entry in process table)
- Duplicates resources from parent to child
- fork returns zero to child, non zero to parent
- Child resumes from next statement after fork
- Parent & child run concurrently based on architecture

# Process termination

- exit() function causes normal process termination and the value of status is returned to the parent
- Process normal termination can be
  - success – exit (0)
  - Failure – exit with positive value
- abnormal termination
  - With exceptions

# waitpid

- Blocks parent process till completion of child process
- Collect exit status of child
- Cleans some pending resources of child (else child will become Zombie)
- waitpid paramaters
  - 1st param : pid of child process waiting for, -1 means any one child
  - 2nd param : status of terminated child (pass by address)
  - 3rd param : flags
- man waitpid

# execl

- Overwrites child address space with resources of specified program
- Process remains same, but program/resources will change
- Any code after execl is redundant, if execl succeeds
- Syntax
  - execl(const char *path, const char *arg, ..., NULL);

  For ex
  - excel(“/usr/bin/cal”, “cal”, “2018”, NULL)
- Excel uses absolute path, excelp uses cmd name

# Signals

# Signals overview

- Signals always operate at process level
- Signals communicate between applications at user level
- Used for communication of abnormal termination, illegal memory access & events that go wrong
- Signals are considered as software interrupts, but there is no interrupt vector table
- Signals between processes
  - SENDER send / triggers signals from one process to other process
  - TARGET will set the corresponding bit based on sender's signal bit
  - Target will lookup in the signal handler table for handler addresses for each of signal handler
- Process descriptor (PD) / process control block (PCB) has signal related fields
- Most of the default signal handlers will cause abnormal termination

# Signals in common actions

| Signal Name | Description | Signal Name | Description |
|---|---|---|---|
| SIGINT | User sends INTERRUPT signal (Ctrl + C) | SIGTERM | User sends TERMINATION signal (kill <pid>) |
| SIGQUIT | User sends QUIT signal (Ctrl + \) | SIGCHLD | Child process stopped |
| SIGTSTP | User sends SUSPEND signal (Ctrl + z) | SIGFPE | Floating point exceptio |

- Commands
  - kill –l, will list all the signals
  - kill –SIGxxxx <pid>
  - kill -<signo> <pid>
  - kill <pid>
  - kill -9 <pid> → terminate is SURE KILL
  - kill (pid, signal number) → system call
  - pkill, killall, pgrep → process kill

# Default & Custom handlers

- Signal has default handler
- Custom handler can override the default handler

- APIs – signal, raise, pause, kill, alarm
- Modern APIs – sigaction, sigprocmask, sigsuspend

- Non maskable signals
  - Have NO custom handlers
  - SIGKILL, SIGSTOP

# Threads

# Basics of Threads

- Path of execution within a process
- Various sub-activities within applications are referred as threads
- Referred as Light Weight Process (LWP)
- Significance of threads
  - Concurrent execution (parent – child process / multiple child process)
  - RESOURCE SHARING across threads
- Child process will have own resources, but threads will have shared resources
- Scheduled threads interchangeably use CPU based on time sharing
- Every process is run initially as a single thread, then multiple threads spawn
  - Firefox browser initially will be a single thread, on need basis multiple threads spawn
- Threads are faster than fork
- Common resources during execution run independently

# Advantage of Thread over Process

- Concurrent execution and faster response, less time for context switch
- Effective use of multiprocessor system
- Resource sharing: code, global data, files can be shared among threads
  - PC, Stack and Registers is separate for each thread
  - Private / local data is not shared
- Easier communication between threads
- Enhanced throughput of the system
  - Number of jobs completed per unit time

Note:

If one thread makes a blocking call, whole process gets blocked.

# Thread Models

- Types of threads
  - User threads
    - Threads used by application programmers, are above kernel and without kernel support
  - Kernel threads
    - Supported within kernel, perform multiple simultaneous tasks to serve multiple kernel system calls
- Models
  - Used to map user threads to kernel threads
  - Many to One model
    - Many user-level threads are mapped to single kernel thread, thread management is handled by thread library in user space
  - One to One model
    - Separate kernel thread is created to handle each and every thread, limitation is the count of threads that can be created
  - Many to Many
    - Many user-level threads are mapped to multiple kernel level threads

# Commands

- ps –e –L –o pid,ppid,lwp,nlwp,stat,cmd
- ps –eLf

- To create threads, POSIX thread library is used
  - pthread_create
  - pthread_join
  - pthread_self
  - pthread_equal
  - pthread_yield
  - pthread_cancel

- gcc psample.c -lpthread

# Queries?

# Thank You !

L&T Technology Services