# Chapter 3
# Kernel

**-- By R.G.B**

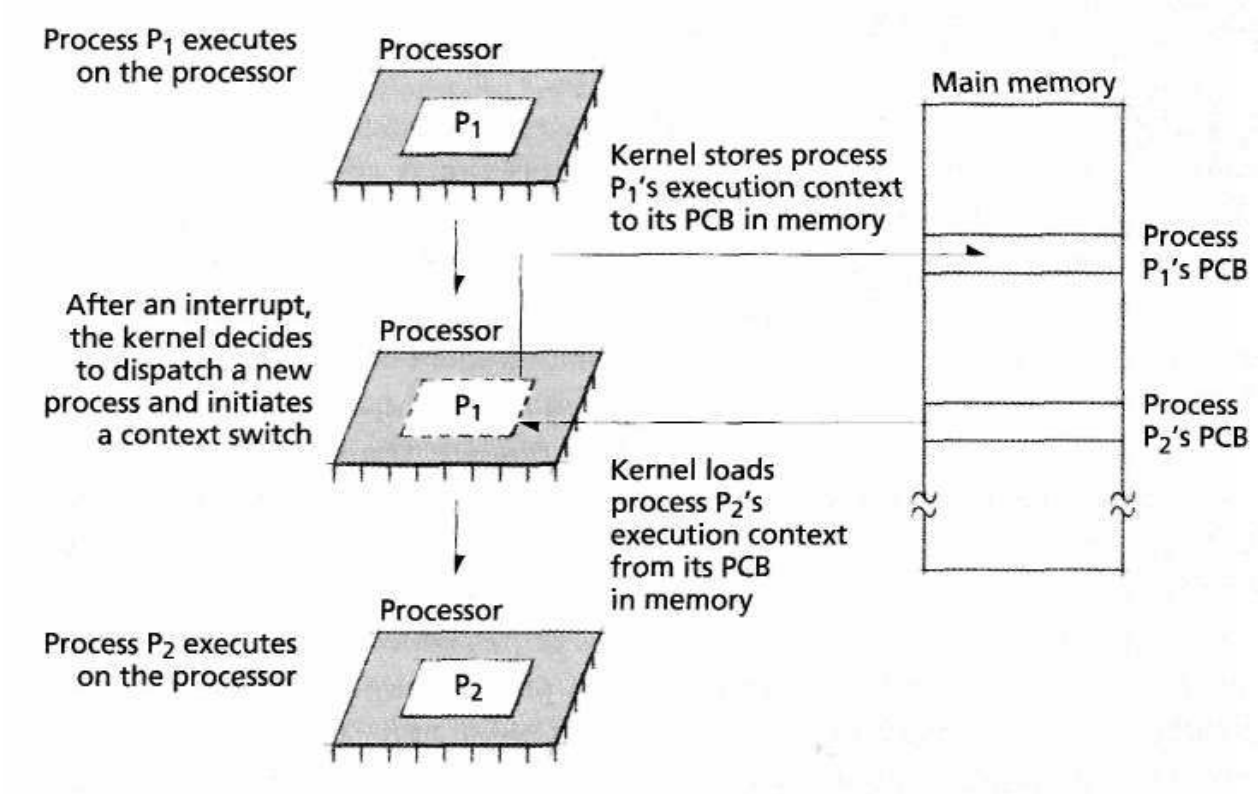
# Kernal

- Central module of an operating system
- Loads first to memory and stays forever(small as possible)
- Provides essential services
- Responsible for memory management, process and task management, and disk management

# Context Switch

- **Context switch** is the computing process of storing and restoring the state (context) of a CPU so that execution can be resumed from the same point at a later time.
- Multiple processes to share a single CPU (Multitasking O.S)
- Computationally intensive and much of the design of operating systems is to optimize the use of context switches
- Switching from one process to another requires
- A certain amount of time for doing the administration - saving and loading registers and memory maps, updating various tables and list etc.

# Context Switch



# Modes of Processes

1. **User Mode**

- User process has no access to the memory used by the kernel
- No ability to directly access hardware or reference memory
- Crashes in user mode are always recoverable.
- Most of the code running on your computer will execute in user mode.

# Modes of Processes

**2. <u>Kernel Mode</u>**

- kernel that is running on behalf of the user process
- Directly access the kernel data structures or the kernel programs
- Complete and unrestricted access to the underlying hardware
- Execute any CPU instruction and reference any memory address
- Reserved for the lowest-level, most trusted functions in OS
- Crashes in kernel mode are catastrophic; they will halt the entire PC

# User To Kernel Mode Switching

- In execution of a program, the CPU runs in user mode till the system call is invoked
- User process has access to a limited section of the computer's memory and can execute a restricted set of machine instructions
- When the process invokes a system call, the CPU switches from user mode to a more privileged mode the kernel
- Kernel that runs on behalf of the user process, but it has access to any memory location and can execute any machine instruction
- After the system call has returned, the CPU switches back to user mode

# Types Of Kernels

Kernel is mainly divided into

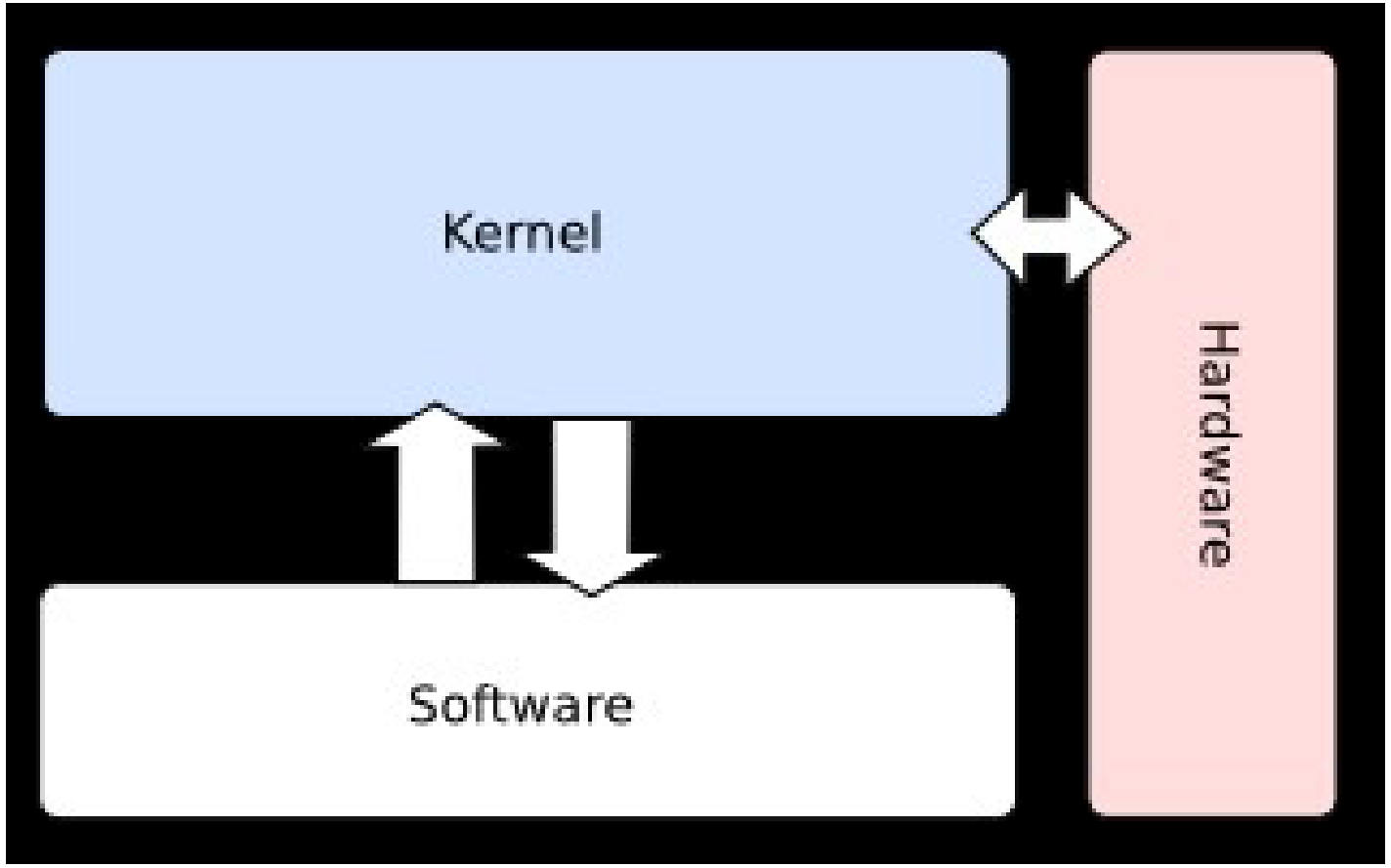
1. **Monolithic kernel**
2. **Micro kernel**
3. **Exo kernel**

# Monolithic Kernel

- Single module only available
- Basic system services like process and memory management, interrupt handling etc. were packaged into a single module
- **Drawbacks**

1. Size of kernel, which was huge.
2. Poor maintainability

# Monolithic Kernel



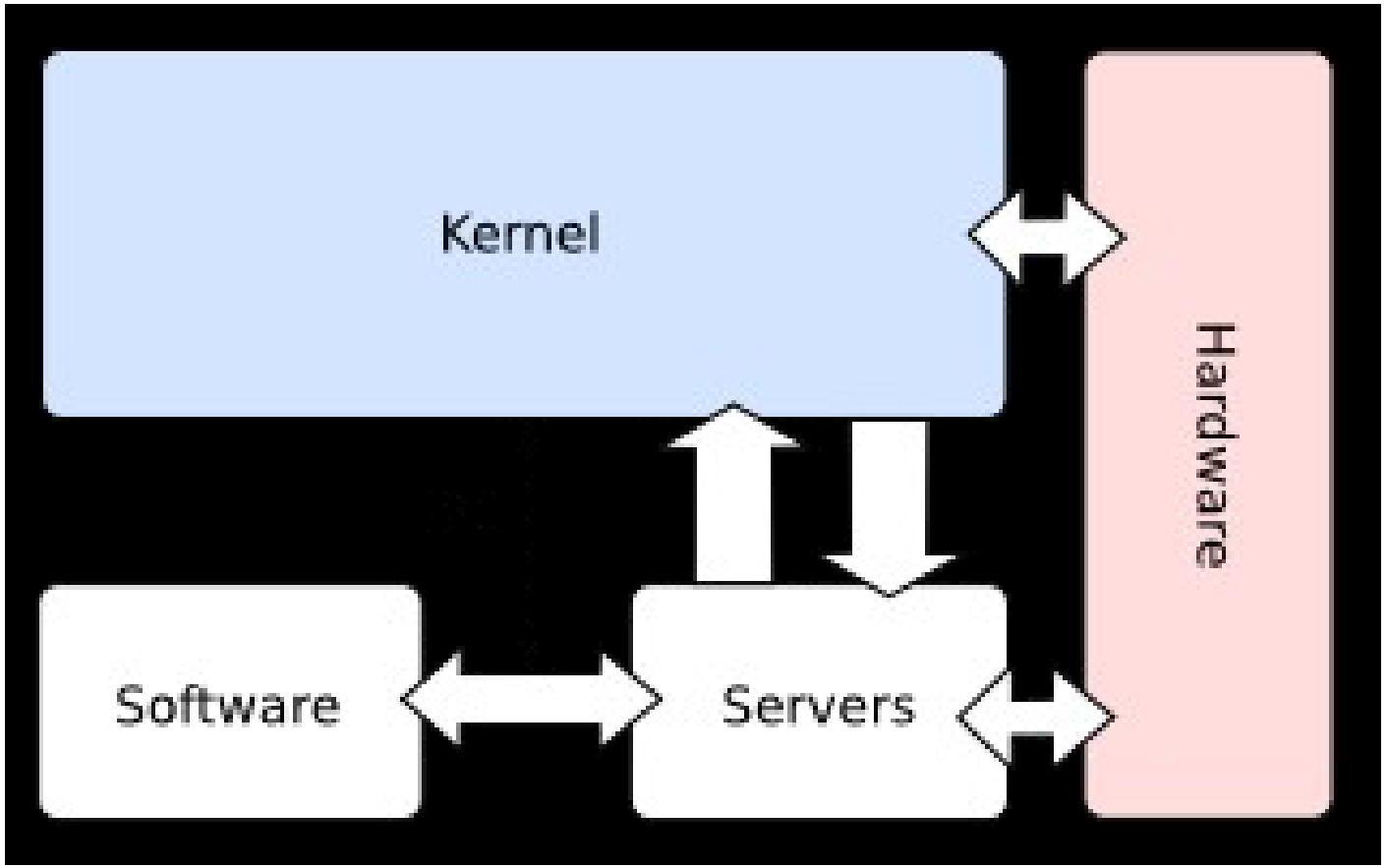
# Monolithic Kernel

- In a modern day, the kernel consists of different modules which can be dynamically loaded and un-loaded.
- Allows easy extension of OS's capabilities
- Maintainability of kernel became very easy as →concerned module
- Also, stripping of kernel for various platforms became very easy
- Examples:
  - ✓Linux
  - ✓Windows 9x (95, 98, Me)
  - ✓Mac OS <= 8.6
  - ✓BSD

# Microkernels

- Most operating system components—such as process management, networking, file system interaction and device management—execute outside the kernel with a lower privilege level
- Kernel → Basic services, like memory allocation, scheduling, and messaging (IPC) →Higher privilege
- Main focus on reducing size of kernel
- Microkernels exhibit a high degree of modularity, making them extensible, portable and scalable

# Microkernels



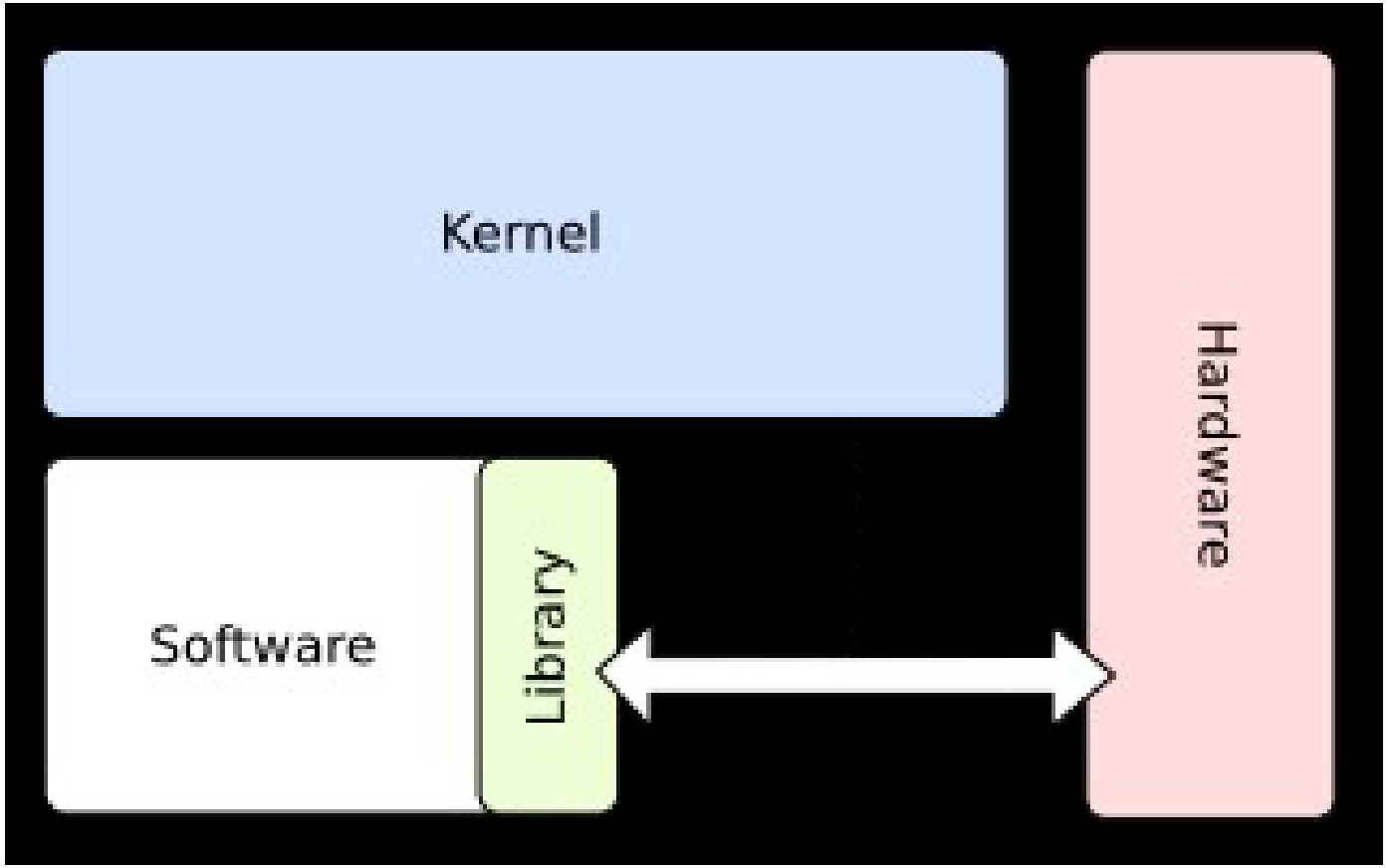
# Microkernels

- Kernel grants some of them privileges to interact with parts of physical memory
- This allows some servers, particularly device drivers, to interact directly with hardware
- Servers are started at the system start-up
- Examples:
  - ✓QNX
  - ✓L4
  - ✓AmigaOS
  - ✓Minix

# Exo-kernel

- Exokernels are an attempt to separate security from abstraction
- The goal is to avoid forcing any particular abstraction upon applications
- Abstractions are best suited to their task
- Moving abstractions into untrusted userspace libraries called "library operating systems" (libOSes), which are linked to applications and call the operating system on their behalf

# Exo-Kernel



# First Level Interrupt Handler (FLIH)

- **Interrupt Handler**(Interrupt Service Routine)
- ISR call back subroutine in operating system or device driver whose execution is triggered by the reception of an interrupt.
- Interrupt handlers have a multitude of functions
- Functions vary based on the reason the interrupt was generated and the speed at which the interrupt handler completes its task.

# First Level Interrupt Handler (FLIH)

- Interrupt handlers are divided into two parts

1. **First-Level Interrupt Handler (FLIH)**

- Hard/Fast interrupt handlers

2. **Second-Level Interrupt Handlers (SLIH)**

- Slow/Soft interrupt handlers

# First Level Interrupt Handler (FLIH)

- Save registers of current process in PCB
- Determine the source of interrupt
- Initiate service of interrupt - calls interrupt handle
- Hardware dependent - implemented in assembler

# First Level Interrupt Handler (FLIH)

- A FLIH implements at minimum platform-specific interrupt handling similarly to interrupt routines.
- In response to an interrupt, there is a context switch, and the code for the interrupt is loaded and executed.
- The job of a FLIH is to quickly service the interrupt, or to record platform-specific critical information which is only available at the time of the interrupt, and schedule the execution of a SLIH for further long-lived interrupt handling

# Implementation of Process

- OS maintains a table called the process table, with one entry per process. (control blocks)
- Entry contains information about the process' state, its program counter,
- stack pointer, memory allocation, the status of its open files, its accounting and scheduling information, alarms and other signals, and everything else about the process that must be saved
- when the process is switched from running to ready state so that it can be restarted later as if it had never been stopped.