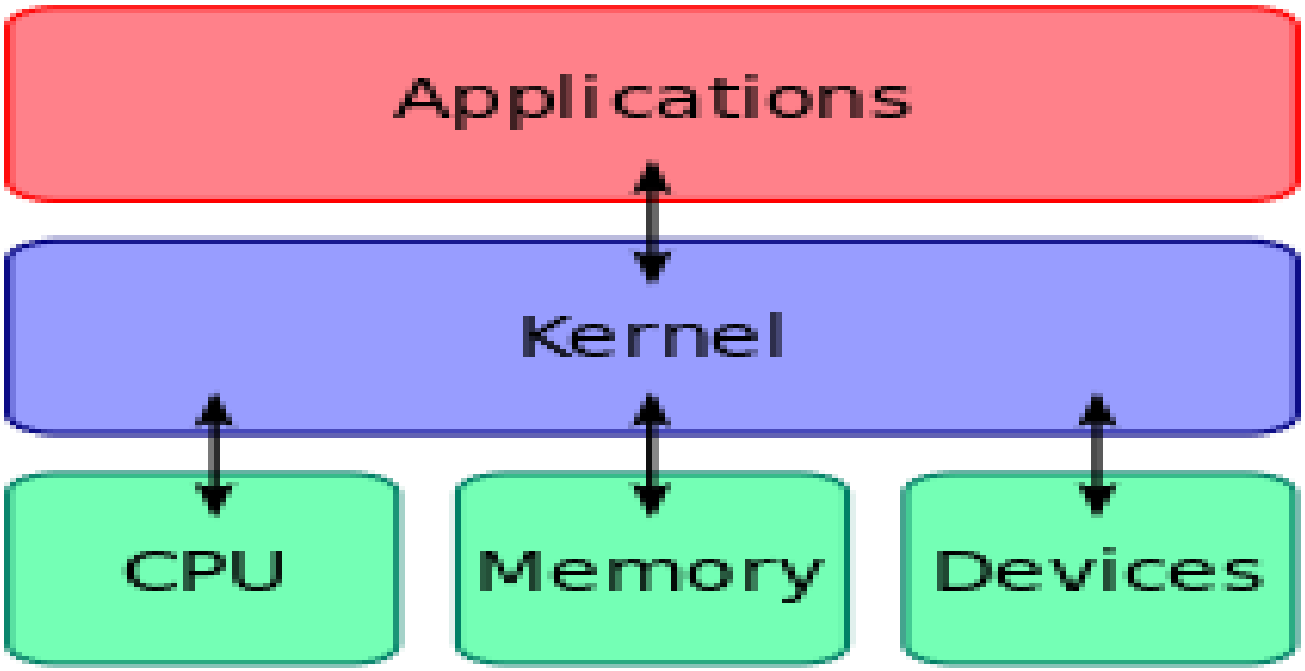
Operating System
Chapter 3: Kernel

Prepared By:
Amit K. Shrivastava
Asst. Professor
Nepal College Of Information Technology

Introduction and Architecture of a Kernel

- A kernel is a central component of an operating system. It acts as an interface between the user applications and the hardware. The sole aim of the kernel is to manage the communication between the software (user level applications) and the hardware (CPU, disk memory etc). The main tasks of the kernel are : Process management, Device management, Memory management, Interrupt handling, I/O communication, File system...etc..
- As a basic component of an operating system, a kernel provides the lowest-level abstraction layer for the resources(especially memory, processors and I/O devices) that application must control to perform their function. It typically make these facilities available to application processes through inter-process communication mechanisms and system calls.

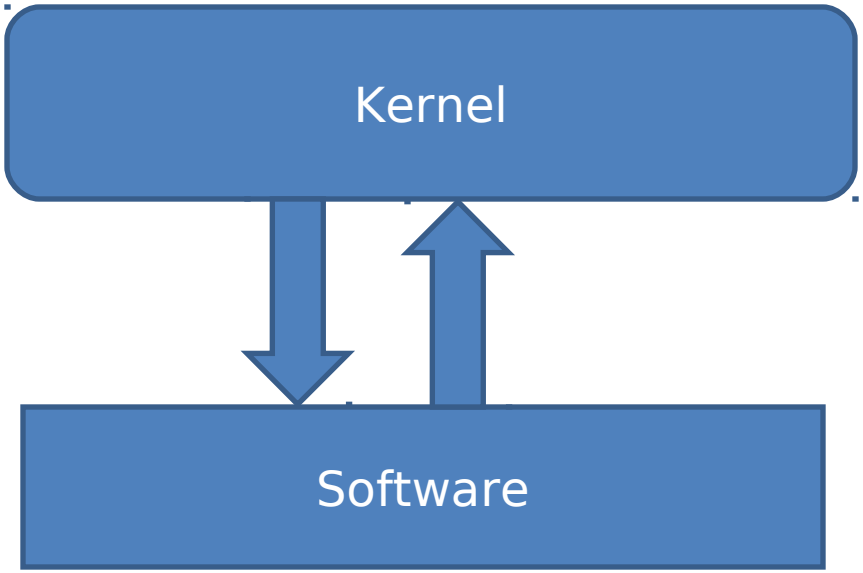
Introduction and Architecture of a Kernel(contd..)



A kernel connects the application software to the hardware of a computer

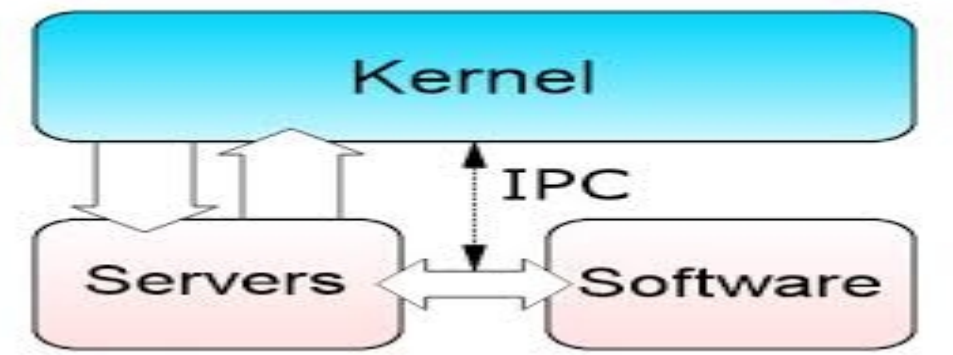
Types of Kernels

- **Monolithic kernel:** A Monolithic kernel is a single large processes running entirely in a single address space. It is a single static binary file. All kernel services exist and execute in kernel address space. The kernel can invoke functions directly. The examples of monolithic kernel based OSs are Linux, Unix.



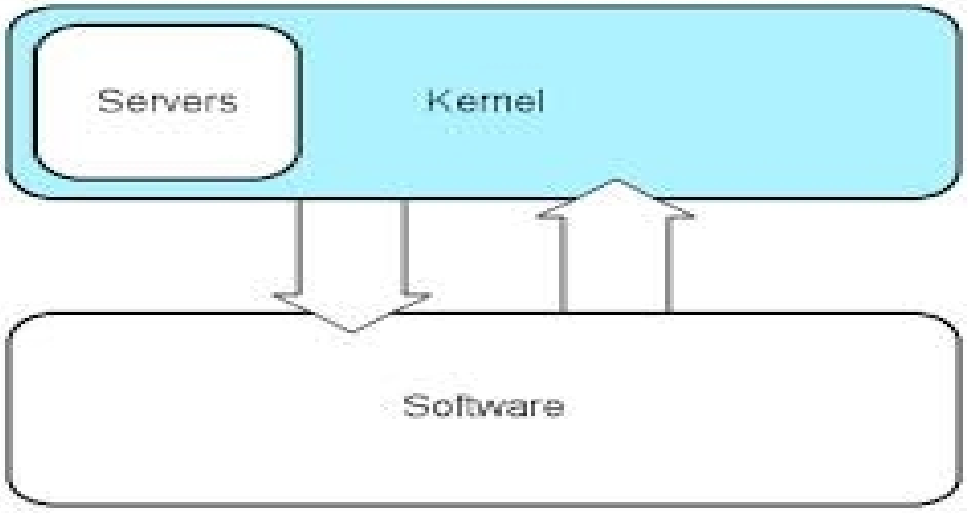
Types of Kernels(contd..)

- **Microkernels:** In Microkernels, the kernel is broken down into separate processes, known as servers. Some of the servers run in kernel space and some run in user-space. All servers are kept separate and run in different address spaces.The communication in microkernels is done via message passing. The servers communicate through IPC (Interprocess Communication). Servers invoke "services" from each other by sending messages. The separation has advantage that if one server fails other server can still work efficiently. The example of microkernel based OS are Mac OS X and Windows NT



Types of Kernels(contd..)

- **Hybrid Kernels:** Hybrid kernel is a kernel architecture based on a combination of microkernel and monolithic kernel architecture used in computer operating systems. This kernel approach combines the speed and simpler design of monolithic kernel with the modularity and execution safety of microkernel. 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.



Types of Kernels(contd..)

- **Nanokernels:** A nanokernel is a very minimalist operating system kernel which represents the closest hardware abstraction layer of the operating system by interfacing the CPU, managing interrupts and interacting with the MMU. The interrupt management and MMU interface are not necessarily part of a nanokernel; however, on most architecture these components are directly connected to the CPU, therefore, it often makes sense to integrate these interfaces into the kernel. It lack system services.
- **Exokernels:** An exokernel is a type of kernel that does not abstract hardware into theoretical models. Instead it allocates physical hardware resources, such as processor time, memory pages, and disk blocks, to different programs.

Context Switching(Kernel mode and User mode)

Kernel Mode:

- In Kernel mode, the executing code has complete and unrestricted access to the underlying hardware. It can execute any CPU instruction and reference any memory address. Kernel mode is generally reserved for the lowest-level, most trusted functions of the operating system. Crashes in kernel mode are catastrophic; they will halt the entire PC.

User Mode:

- In User mode, the executing code has no ability to directly access hardware or reference memory. Code running in user mode must delegate to system APIs to access hardware or memory. Due to the protection afforded by this sort of isolation, crashes in user mode are always recoverable. Most of the code running on your computer will execute in user mode.

Context Switching(Kernel mode and User mode)

- The following three situations result in switching to kernel mode from user mode of operation:

1. The scheduler allocates a user process a slice of time (about 0.1 second) and then system clock interrupts. This entails storage of the currently running process status and selecting another runnable process to execute. This switching is done in kernel mode.
2. Services are provided by kernel by switching to the kernel mode. So if a user program needs a service (such as print service, or access to another file for some data) the operation switches to the kernel mode.
3. Suppose a user process had sought a data and the peripheral is now ready to provide the data, then the process interrupts. The hardware interrupts are handled by switching to the kernel mode.

Context Switching(Kernel mode and User mode)

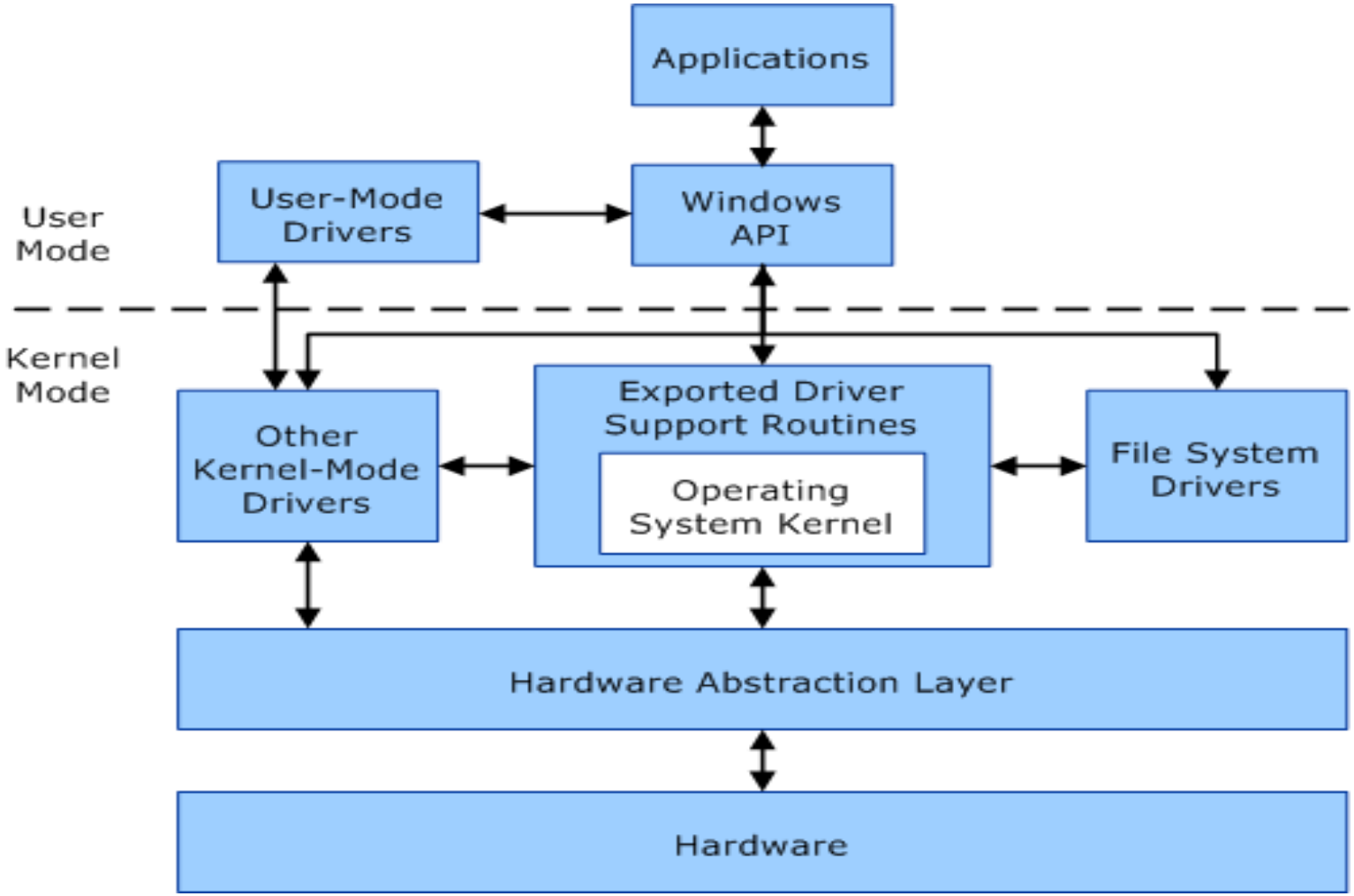


Fig: Diagram illustrating communication between user-mode and kernel-mode components.

First Level Interrupt Handling

- First Level Interrupt Handling(FLIH) occurs on hardware level. It
  - save registers of current process in PCB
  - Determine the source of interrupt
  - Initiate service of interrupt - calls interrupt handler **Hardware dependent - implemented in assembler**
  - FLIH must run with interrupts disabled because Interrupt handler may call kernel to unblock another process (e.g. waiting for I/O) which could result in a different process - rather than interrupted one eventually being run by the dispatcher
  - Handler could be a kernel procedure to service a normal kernel call, e.g. send a message, delay a process etc.

Kernel Implementation of Processes

- A kernel process is a process that is created in the kernel protection domain and always executes in the kernel protection domain. Kernel processes can be used in subsystems, by complex device drivers, and by system calls. They can also be used by interrupt handlers to perform asynchronous processing not available in the interrupt environment. Kernel processes can also be used as device managers where asynchronous input/output (I/O) and device management is required.
- A kernel process controls directly the kernel threads. Because kernel processes are always in the kernel protection domain, threads within a kernel process are kernel-only threads. A kernel process inherits the environment of its parent process (the one calling the **creatp** kernel service to create it), but with some exceptions. . The kernel process does not have a root directory or a current directory when initialized. All uses of the file system functions must specify absolute path names.