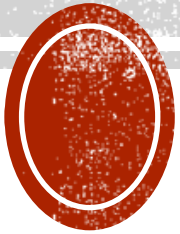
DEVICE MANAGEMENT

Chapter 5.1



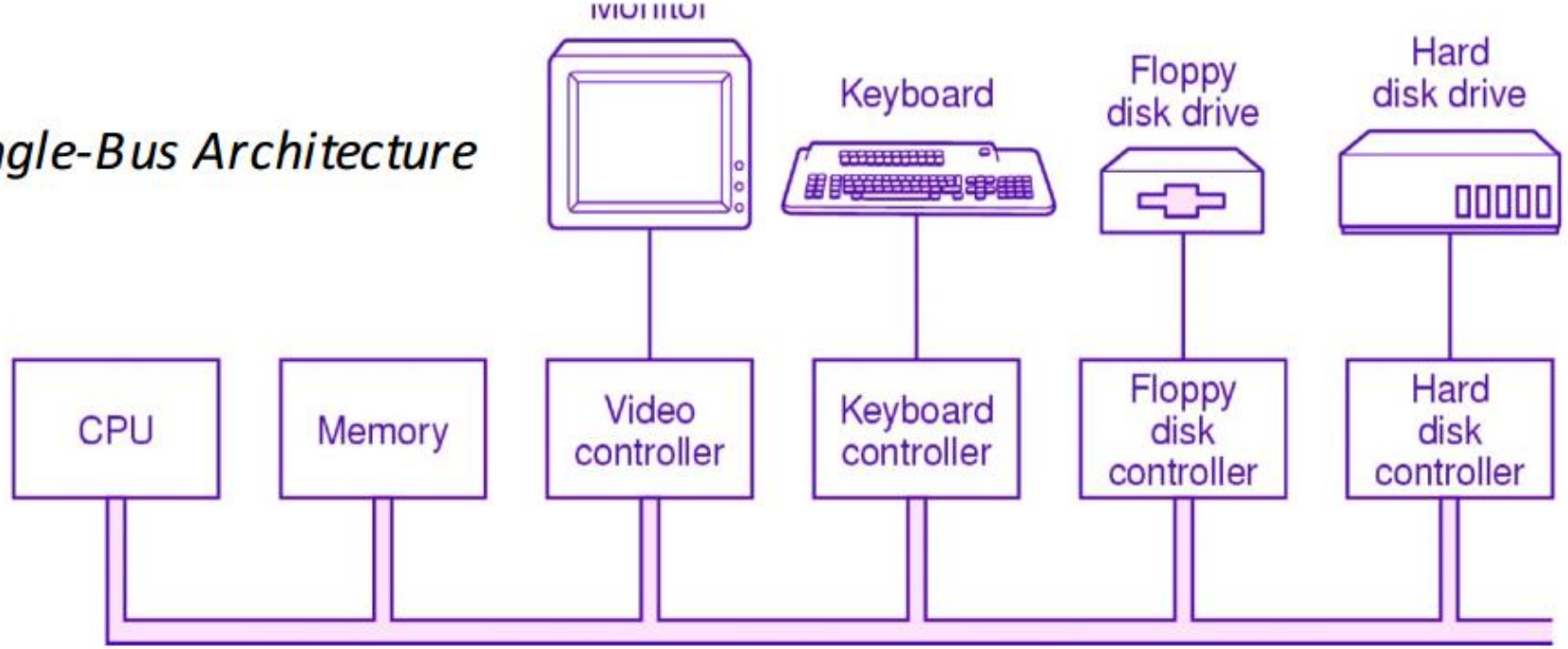
SYLLABUS

- 5.1 Principle of I/O hardware
  - I/O device
  - Device Controller
  - Direct Memory Access
- 5.2 Principle of I/O software
  - Goals of I/O software
  - Interrupt Handler
  - Device Drivers
- 5.4 Terminals:
  - Terminal Hardware
  - Memory-Mapped Terminals
  - Input-Output Software

I/O MANAGEMENT

- All computers have **physical devices for acquiring input and producing output**.
- *OS is responsible to **manage and control** all the I/O Operations and I/O devices.*
- The I/O devices, memory, and the CPU communicate with each other by way of one or more **communication buses.**

Single-Bus Architecture



I/O DEVICES

- *The I/O units which **consist mechanical components** are called I/O devices.*

  e.g. hard-disk drive, printer etc.

- There are **two types** of devices:
- **Block devices:**
  - Stores information in fixed-sized blocks, each one with its own address.
  - Read or write is possible independent to other blocks-**direct access**.
  - Example: *disk*
- **Character devices:**
  - Delivers or accepts a stream of characters without regard to any block structure
  - it is not addressable and does not have any seek operation
  - – e.g., keyboards, mice, terminals, line printers, network interfaces, and most other devices that are not disk-like…

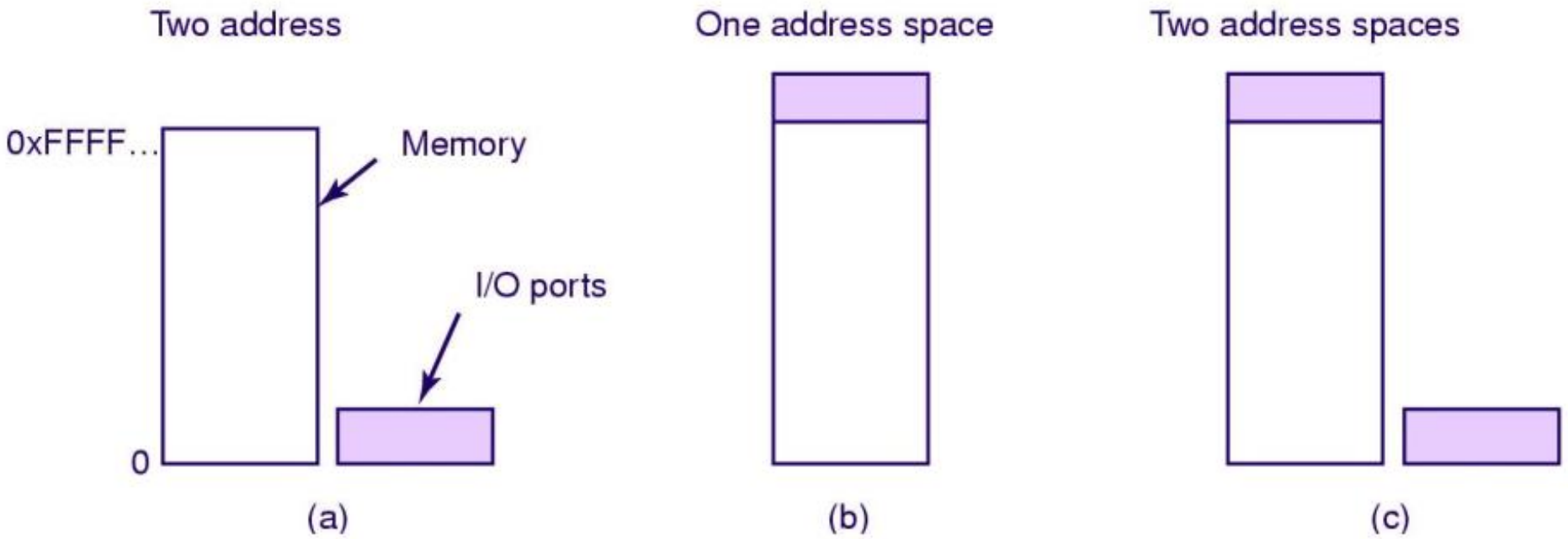
  - Some devices are neither block nor character such as **clock & timers**

DEVICE CONTROLLER

- *A controller is a collection of electronics that can operate a bus or a device.*
- On PC, it often takes the form of printed circuit card that can be inserted into an expansion slot.
- A single controller can handle multiple devices; some devices have their own built-in controller.
- The controller has one or more registers for data and signals.
- The processor communicates with the controller by reading and writing bit patterns in these registers.
- When transferring a disk block of size 512 bytes, the block first assembled bit by bit in a buffer inside the controller. After its checksum has been verified and the block declared to be error free, it can than be copied to main memory.

MEMORY-MAPPED I/O

- Device controller have their own register **(Control Register)** and **Data Buffer** for communicating with the CPU,
  - by writing and reading these register, OS perform the I/O operation.
- **Two approaches** are there to communicate (1)
- each control register is assigned an **I/O port** number, an 8- or 16-bit integer. Using a special I/O instruction such as
- IN R0,4
- MOV R0,4
- the address spaces for **memory and I/O are different.**
- former reads the contents of I/O port 4 and puts it in R0 whereas the latter reads the contents of memory word 4 and puts it in R0
- **Second Approach is called Memory Mapped**



a) separate I/O port and memory b) Memory-mapped I/O c) Hybrid.

MEMORY MAPPED I/O

- map all the control registers into the memory space
- Each control register is assigned a unique memory address to which no memory is assigned
- Usually, the assigned addresses are at the top of the address space
- **hybrid scheme**
  - with memory-mapped I/O data buffers and separate I/O ports for the control registers
  - The Pentium uses this architecture,
  - with addresses 640K to 1M being reserved for device data buffers in IBM PC compatibles, in addition to I/O ports 0 through 64K

MEMORY MAPPED I/O

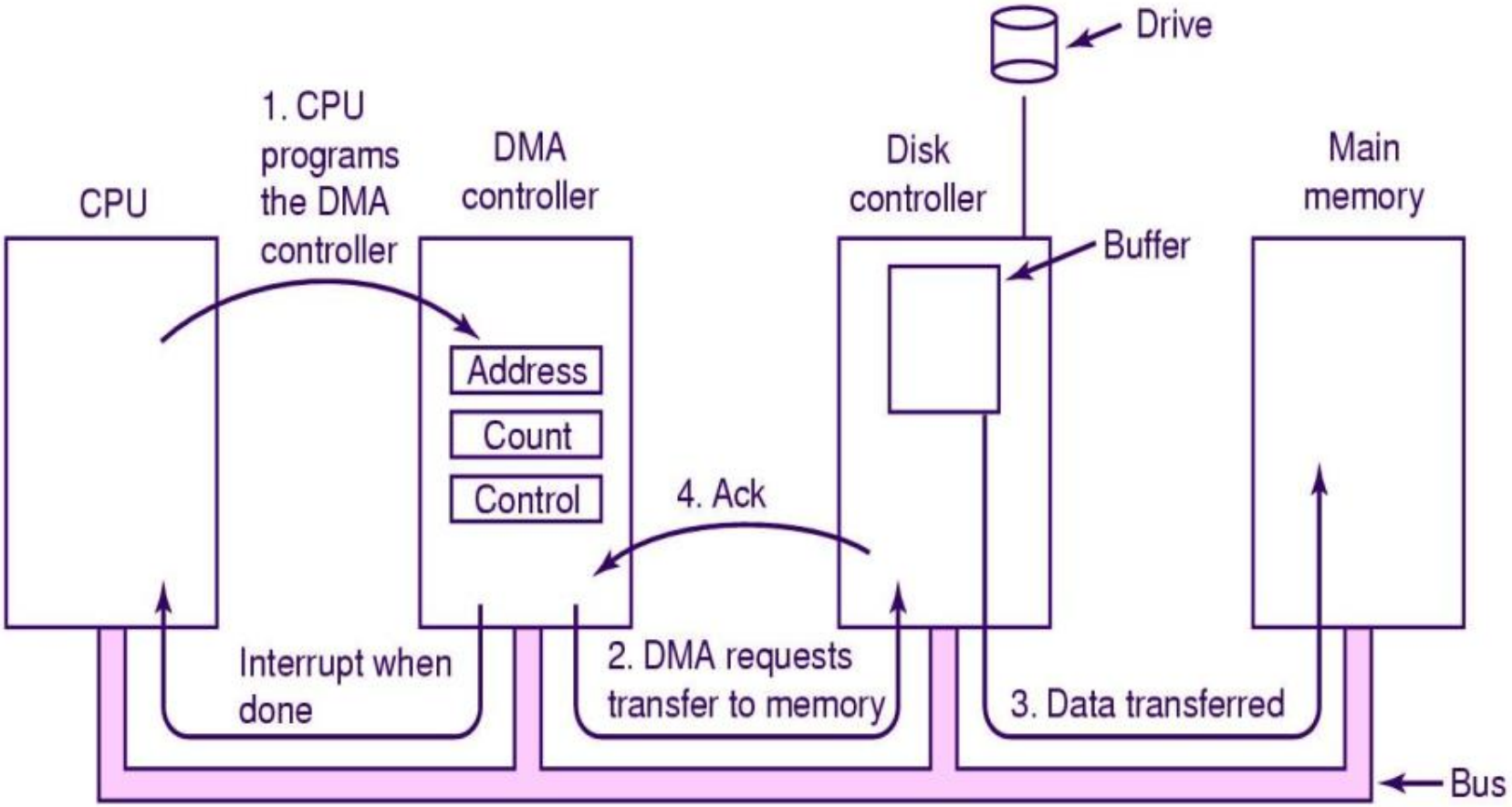
- **Advantages:**
  - Can be **implemented in high-level languages** such as C.
  - **No separate protection** mechanism is needed.
  - Every instruction that can reference the memory can also reference the control registers (in hybrid).

- **Disadvantages:**
  - **Adds extra complexity** to both hardware and OS.
  - All memory modules and all I/O devices **must examine all memory references** to see which one to respond to.

DIRECT MEMORY-ACCESS (DMA)

- Reading one byte at time waste CPU time
- so a different scheme, called **DMA (Direct Memory Access)**
- operating system can only use DMA if the hardware has a **DMA controller,**
  - most system have it, these days in main board
- Many computers avoid burdening the CPU by offloading some of its work to special propose processor called **DMA controller**.
- It consists **memory address register**, a **byte count register**, and
  one or more **control registers.**

DMA



Steps in DMA Transfer

DMA: HOW IT WORKS

- 1. The CPU programs the DMA controller by its registers so it knows what to transfer where.
  - It also issue a command to disk controller telling it to read data from disk to its internal buffer and verify the checksum. When valid data are in disk's controller buffer, DMA can begin.
- 2. The DMA controller initiates the transfer by issuing a read request over the bus to disk controller.
- 3. Data transferred from disk controller to memory.
- 4. When transferred completed, the disk controller sends an acknowledgment signal to DMA controller. The DMA controller then increments the memory address to use and decrement the byte count. This continues until the byte count greater than 0.
- 5. When transfer completed the DMA controller interrupt the CPU

EXAMPLE : I/O HANDLING

- ***Scenario: a cook is cooking something in modern microwave oven equipped kitchen.***

- **CASE A:** The cook may regularly peek through the oven's glass door and watch as roast cooks under cook and watch; this kind of regular monitoring is *polling*.

- **CASE B:** The cook may set a timer to expire after as appropriate number of minutes; the buzzer sounding after this interval is an *interrupt*.

I/O HANDLING: POLLING

- *Processing is interrupted at brief intervals to allow the CPU to check back to I/O to see if the I/O operation has completed.*
- Well manner way of getting attention.
- **Advantages**:
  - Simple
- **Problems:**
  - May bear long wait.
  - Can be a high overhead operation-inefficient.
- *Used in embedded system, where CPU has nothing else to do.*

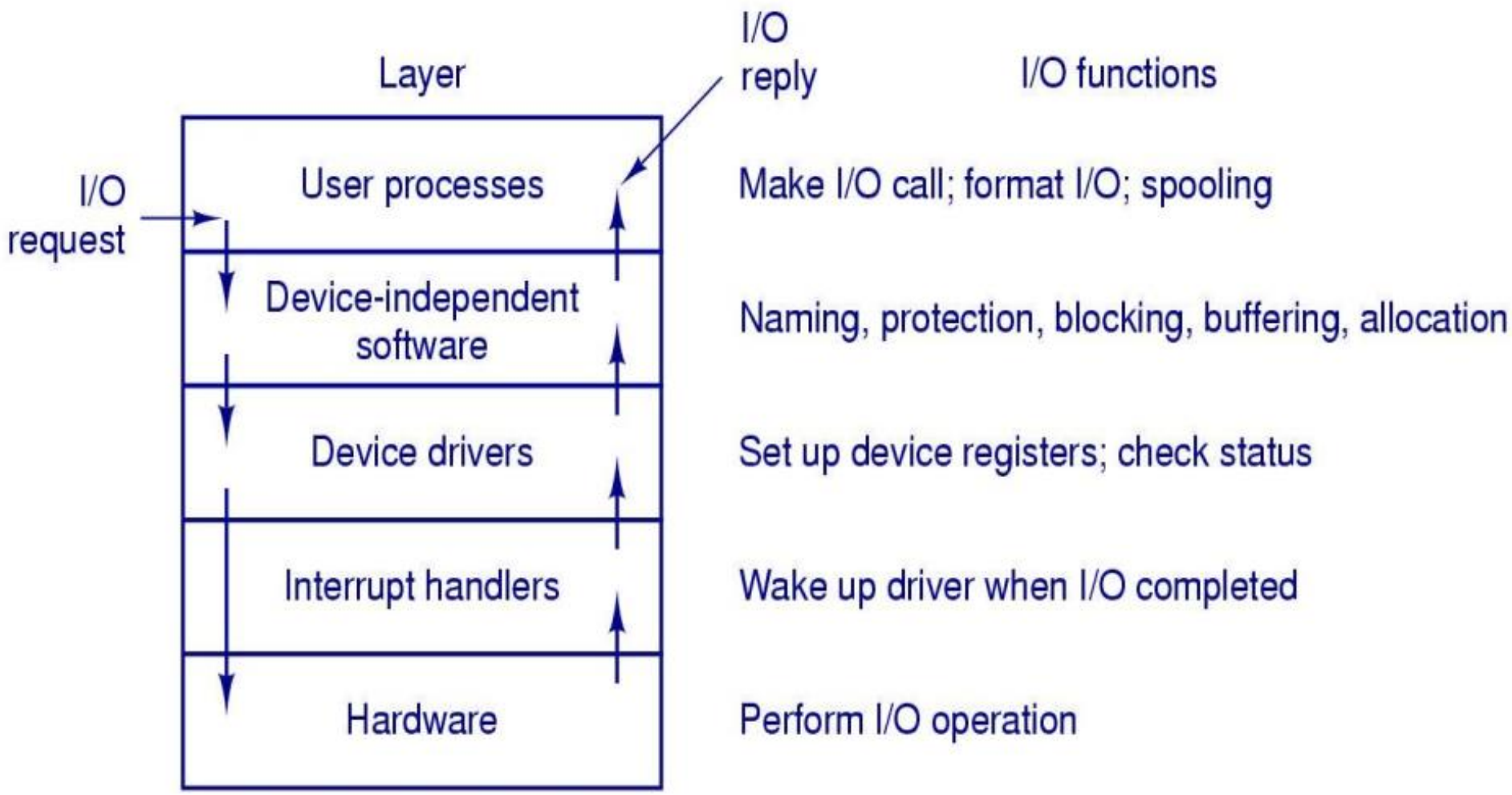
I/O HANDLING: INTERRUPT

- *The hardware mechanism that enables a device to notify the CPU is called an interrupt.*
- Interrupt forced to stop CPU what it is doing and start doing something else.
- **Advantages:**
  - Improves efficiency.
- **Problem:**
  - *Because of the selfish nature, the first interrupt may not be served if the similar second happened before the time needed to serve the first*.

I/O SOFTWARE ISSUES

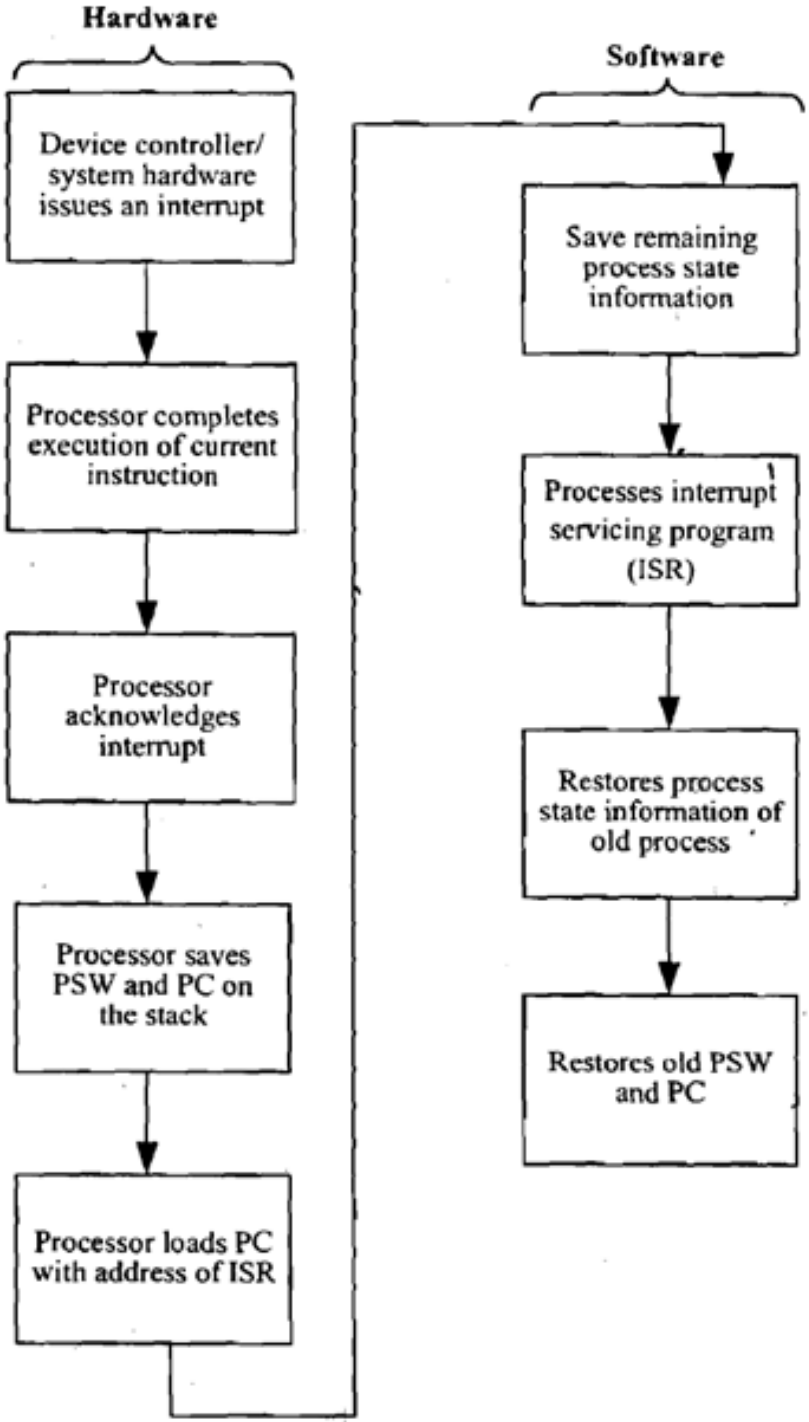
- Device Independence
  - should be possible to write programs that can access any I/O device without having to specify the device in advance
- Uniform Naming
  - name of a file or a device should simply be a string or an integer and not depend on the device in any way
- Error Handling
  - errors should be handled as close to the hardware as possible
- Synchronous vs. Asynchronous transfer
  - **synchronous** (blocking) versus **asynchronous** (interrupt-driven) transfers
- Buffering
  - Data byte from I/O devices should be able to buffered complete before they can be stored in final location.
- sharable versus dedicated devices:
  - Disk vs Tape
  - Should handle both kinds of devices

I/O S/W LAYERS


Layer
I/O reply
I/O functions
I/O request
User processes
Make I/O call; format I/O; spooling
Device-independent software
Naming, protection, blocking, buffering, allocation
Device drivers
Set up device registers; check status
Interrupt handlers
Wake up driver when I/O completed
Hardware
Perform I/O operation

INTERRUPT HANDLERS

- *Block the driver until the I/O has completed and the interrupt occur.*
- The interrupt-handler determines the cause of the interrupt and performs the necessary processing.
- 1. Save any registers (including the PSW) that have not already been saved by the interrupt hardware.
- 2. Set up a stack for interrupt service procedure.
- 3. Ack interrupt controller.
- 4. Copy registers from where they were saved (stack) to the process table.
- 5. Run the interrupt service procedure.
- 6. Set up the MMU context for the process to run next.
- 7. Load new process' registers, including PSW.
- 8. Start running the new process.


Hardware
Software
Device controller/ system hardware issues an interrupt
Processor completes execution of current instruction
Processor acknowledges interrupt
Processor saves PSW and PC on the stack
Processor loads PC with address of ISR
Save remaining process state information
Processes interrupt servicing program (ISR)
Restores process state information of old process
Restores old PSW and PC

DEVICES DRIVERS

- *Encapsulate detail of devices.*
- Each I/O device attached to a computer needs some device-specific code for controlling it, **called device driver**,
  - is generally **written by the device's manufacturer** and delivered along with the device.
- Each device driver normally **handles one device type,** or at most **one class** of closely related devices.
- In some systems, the OS is a single binary program that **contains all of the drivers** that it will need compiled into it (e.g., **UNIX**).

DEVICES DRIVERS

- **Functions:**
- **Accept read and write requests** from the device independent software above it.
- Initialize the devices if necessary.
- Manage power requirement and log events.
- It checks the status of devices- in use or free.
- Decides which command to issue if there is command queue.

DEVICE-INDEPENDENT OS SOFTWARE

- *perform the I/O functions that are common to all devices and*
- *to provide a uniform interface to the user-level software.*
- **Some common functions are:**
  - Uniform Interfacing for devices drivers-naming and protection.
  - Buffering.
  - Error reporting.
  - Allocating and releasing dedicated devices.
  - Providing a device-independent block size.