Module - V

Storage Management

OS Module V Part-III

I/O Subsystems

Chapter Outcomes

At the end of this chapter, you will be able to;

- Explain the structure of an operating system's I/O subsystem.
- Discuss the principles and complexities of I/O hardware.

I/O Subsystem

- The two main jobs of a computer are
  i. I/O
  ii. Processing
- In many cases, the main job is I/O, and the processing is merely incidental.
- For instance, when we browse a Web page or edit a file, our immediate interest is to read or enter some information, not to compute an answer.
- The role of the operating system with respect to I/O is to manage and control I/O operations and I/O devices.

Overview

- The control of devices connected to the computer is a major concern of operating-system designers.
- Because I/O devices vary so widely in their function and speed (consider a mouse, a hard disk, and a CD-ROM jukebox).
- Different methods are needed to control them.
- These methods form the ***I/O subsystem*** of the kernel, which separates the rest of the kernel from the complexities of managing I/O devices.

Overview

- The basic I/O hardware elements, such as ports, buses, and device controllers, accommodate a wide variety of I/O devices.

- To encapsulate the details and oddities of different devices, the kernel of an OS is structured to use device-driver modules.

Overview

- The **device drivers** present a uniform device access interface to the I/O subsystem, much as system calls provide a standard interface between the application and the operating system.

I/O Hardware

- Computers operate a many kinds of devices.
- The general categories are of
  - Storage devices - disks, tapes
  - Transmission devices - network cards, modems
  - Human-interface devices - screen, keyboard, mouse.
- Other devices are more specialized, such as those involved in the steering of a military fighter jet or a space shuttle.
- We need to understand how the devices are attached and how the software can control the hardware.

I/O Hardware

- A device communicates with a computer system by sending signals over a cable or even through the air.
- The device communicates with the machine via a connection point, called **port.** eg a serial port.
- If devices use a common set of wires, the connection is called a *bus*.
- A **bus** is a set of wires and a rigidly defined protocol that specifies a set of messages that can be sent on the wires.

I/O Hardware

- Buses are used widely in computer architecture and vary in their signaling methods, speed, throughput, and connection methods.

I/O Hardware

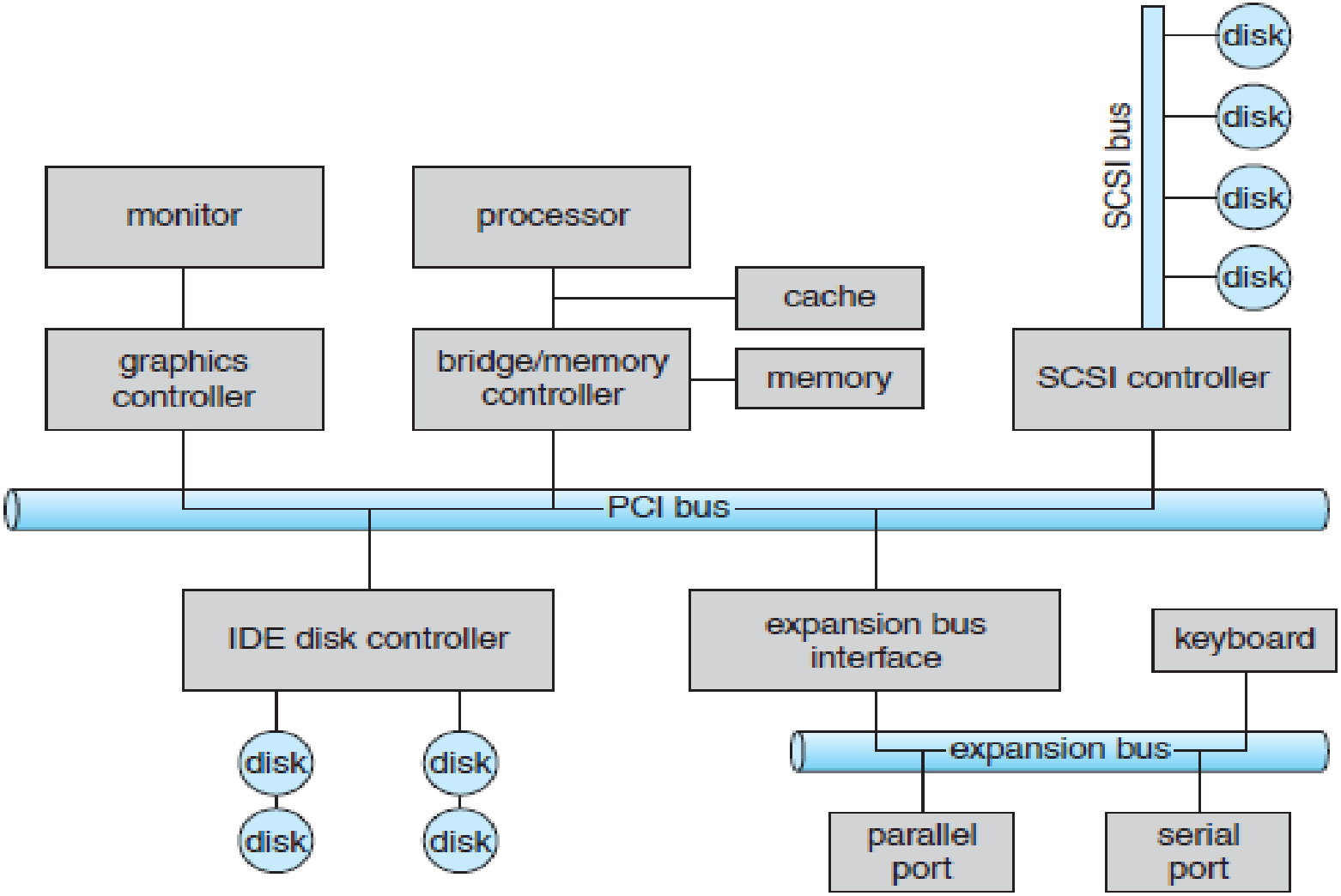

Fig: A typical PC bus structure, ref: OS Concepts by Galvin

I/O Hardware

- A **controller** is a collection of electronics that can operate a port, a bus, or a device.
- A serial-port controller is a simple device controller.
- It is a single chip (or portion of a chip) in the computer that controls the signals on the wires of a serial port.

I/O Hardware

- By contrast, a SCSI bus controller is not simple.
- Because the SCSI protocol is complex, the SCSI bus controller is often implemented as a separate circuit board (or a **host adapter**) that plugs into the computer.
- It typically contains a processor, microcode, and some private memory to enable it to process the SCSI protocol messages.
- Some devices have their own built-in controllers.

I/O Hardware

How can the processor give commands and data to a controller to accomplish an I/O transfer?

- The controller has one or more registers for data and control signals.

- The processor communicates with the controller by reading and writing bit patterns in these registers.

- This communication can be one of two ways;

I/O Hardware

- One way in which this communication can occur is through the use of special I/O instructions that specify the transfer of a byte or word to an I/O port address.

- The I/O instruction triggers bus lines to select the proper device and to move bits into or out of a device register.

I/O Hardware

- Alternatively, the device controller can support **memory-mapped I/O**.
- In this case, the device-control registers are mapped into the address space of the processor.
- The CPU executes I/O requests using the standard data-transfer instructions to read and write the device-control registers.

I/O Hardware

An I/O port typically consists of four registers,

1) Status
2) Control
3) Data-in
4) Data-out registers

I/O Hardware

- The **data-in register** is read by the host to get input.
- The **data-out register** is written by the host to send output.
- The **status register** contains bits that can be read by the host.
  - These bits indicate states, such as
    - whether the current command has completed,
    - whether a byte is available to be read from the data-in register, and
    - whether a device error has occurred.

I/O Hardware

- The **control register** can be written by the host to start a command or to change the mode of a device.

  - For instance, a certain bit in the control register of a serial port chooses between full-duplex and half-duplex communication,
  - another bit enables parity checking,
  - a third bit sets the word length to 7 or 8 bits, and
  - other bits select one of the speeds supported by the serial port.

I/O Handshaking

- The protocol for interaction between the host and a controller can be intricate, but the basic *handshaking*.
    i. Polling
    ii. Interrupts
    iii. Direct Memory Access (DMA)

I/O Handshaking

Polling

1. The host repeatedly reads the *busy* bit until that bit becomes clear.

2. The host sets the *write* bit in the *command* register and writes a byte into the *data-out* register.

3. The host sets the *command-ready* bit.

4. When the controller notices that the *command-ready* bit is set, it sets the *busy* bit.

I/O Handshaking

Polling

5. The controller reads the command register and sees the write command. It reads the *data-out* register to get the byte and does the I/O to the device.

6. The controller clears the *command-ready* bit, clears the *error* bit in the status register to indicate that the device I/O succeeded, and clears the *busy* bit to indicate that it is finished.

I/O Handshaking

Polling

5. The controller reads the command register and sees the write command. It reads the *data-out* register to get the byte and does the I/O to the device.

6. The controller clears the *command-ready* bit, clears the *error* bit in the status register to indicate that the

- Polling becomes inefficient when it is attempted repeatedly yet rarely finds a device to be ready for service, while other useful CPU processing remains undone.

I/O Handshaking

Interrupts

The basic interrupt mechanism works as follows.

- The CPU hardware has a wire called the **interrupt-request line** that the CPU senses after executing every instruction.

- When the CPU detects that a controller has asserted a signal on the interrupt-request line, the CPU performs a state save and jumps to the **interrupt-handler routine** at a fixed address in memory.

I/O Handshaking

Interrupts

- The interrupt handler determines the cause of the interrupt, performs the necessary processing, performs a state restore, and executes a return from interrupt instruction to return the CPU to the execution state prior to the interrupt.

I/O Handshaking

Direct Memory Access

- For a device that does large transfers, such as a disk drive, it seems wasteful to use an expensive general-purpose processor to watch status bits and to feed data into a controller register one byte at a time—a process termed **programmed I/O (PIO)**.

- Many computers avoid burdening the main CPU with PIO by offloading some of this work to a special-purpose processor called a **direct-memory-access (DMA) controller**.

I/O Handshaking

Direct Memory Access

- To initiate a DMA transfer, the host writes a DMA command block into memory.

- This block contains a pointer to the source of a transfer, a pointer to the destination of the transfer, and a count of the number of bytes to be transferred.

- The CPU writes the address of this command block to the DMA controller, then goes on with other work.

I/O Handshaking

Direct Memory Access

- The DMA controller proceeds to operate the memory bus directly, placing addresses on the bus to perform transfers without the help of the main CPU.

- A simple DMA controller is a standard component in PCs, and **bus-mastering I/O boards** for the PC usually contain their own high-speed DMA hardware.

Application I/O Interface

- We can abstract away the detailed differences in I/O devices by identifying a few general kinds.
- Each general kind is accessed through a standardized set of functions—an **interface**.
- The differences are encapsulated in kernel modules called device drivers that internally are custom-tailored to specific devices but that export one of the standard interfaces.
- Fig. on next slide illustrates how the I/O-related portions of the kernel are structured in software layers

Application I/O Interface

- We can abstract away the detailed differences in I/O devices by identifying a few general kinds.
- Each general kind is accessed through a standardized set of functions—an **interface**.
- The differences are encapsulated in kernel modules called device drivers that internally are custom-tailored to specific devices but that export one of the standard interfaces.
- Fig. on next slide illustrates how the I/O-related portions of the kernel are structured in software layers

Application I/O Interface

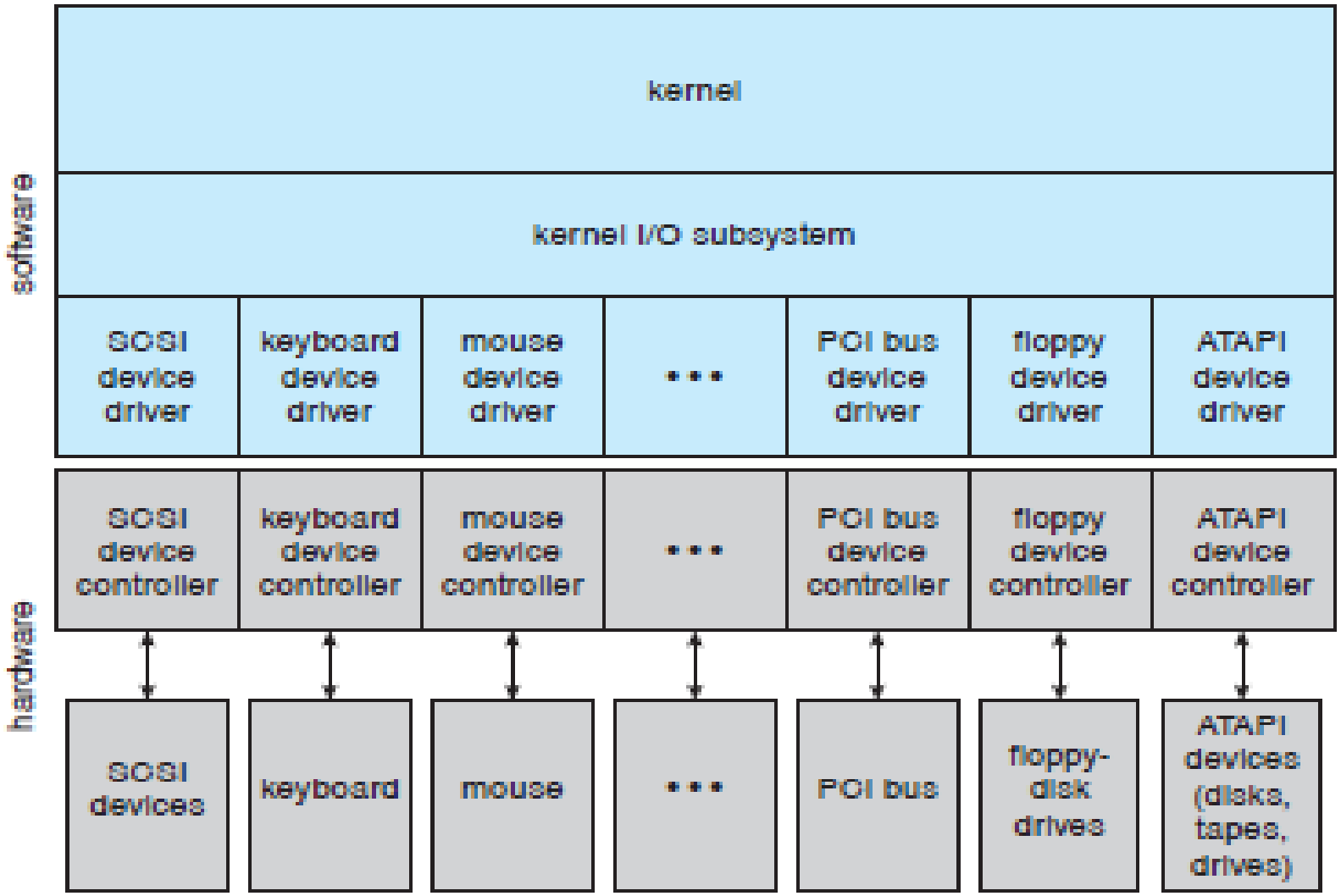

Fig: A kernel I/O structure ref: OS concepts by Galvin

Kernel I/O Subsystem

- Kernels provide many services related to I/O.
- Several services provided by the kernel's I/O subsystem such as —
  - I/O Scheduling
  - Buffering
  - Caching
  - Spooling
  - Device reservation
  - Error handling

Thank You