IO Management System

OBJECTIVES

- • To explore the structure of an operating system's I/O subsystem.
- • To discuss the principles and complexities of I/O hardware.
- • To explain the performance aspects of I/O hardware and software.

I/O Hardware

- I/O Hardware Computers operate a great many kinds of devices.
- Most fit into the general categories of storage devices (disks, tapes), transmission devices (network connections, Bluetooth), and human-interface devices (screen, keyboard, mouse, audio in and out).

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, or **port**—for example, a serial port.
- If devices share 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.
- In terms of the electronics, the messages are conveyed by patterns of electrical voltages applied to the wires with defined timings.
- When device *A* has a cable that plugs into device *B,* and device *B* has a cable that plugs into device *C,* and device *C* plugs into a port on the computer, this arrangement is called a **daisy chain**.
- A daisy chain usually operates as a bus.

I/O Hardware-Bus Structure

- Buses are used widely in computer architecture and vary in their signaling methods, speed, throughput, and connection methods.
- A typical PC bus structure appears in Figure 1.
- In the figure, a **PCI bus** (the common PC system bus) connects the processor–memory subsystem to fast devices, and an **expansion bus** connects relatively slow devices, such as the keyboard and serial and USB ports.
- In the upper-right portion of the figure, four disks are connected together on a **Small Computer System Interface (SCSI)** bus plugged into a SCSI controller. Other common buses used to interconnect main parts of a computer include **PCI Express (PCIe)**, with throughput of up to 16 GB per second, and **Hyper Transport**, with throughput of up to 25 GB per second.

I/O Hardware-Bus Structure

- 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. 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.
- If you look at a disk drive, you will see a circuit board attached to one side.
- This board is the disk controller. It implements the disk side of the protocol for some kind of connection—SCSI or **Serial Advanced Technology Attachment (SATA)**, for instance.
- It has microcode and a processor to do many tasks, such as bad-sector mapping, prefetching, buffering, and caching.

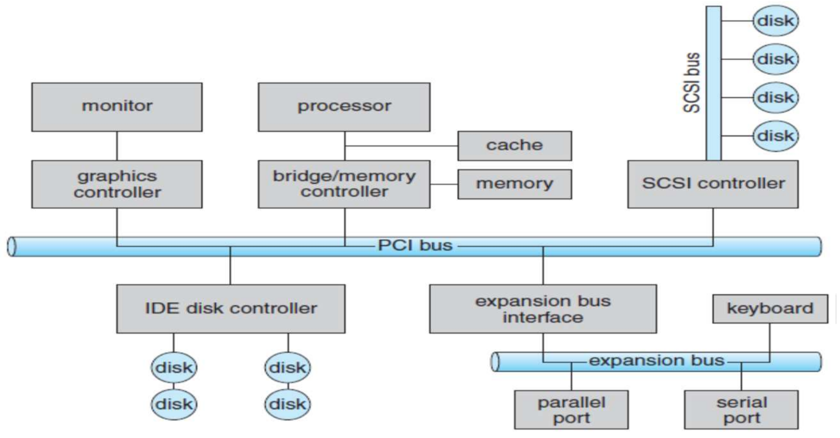

Figure 13.1 A typical PC bus structure.

I/O Hardware-Bus Structure-Registers

- An I/O port typically consists of four registers, called the status, control, data-in, and data-out registers.
- 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.
- • The **control register** can be written by the host to start a command or to change the mode of a device.

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.
- 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.
- We say that the device controller ***raises*** an interrupt by asserting a signal on the interrupt request line, the CPU ***catches*** the interrupt and ***dispatches*** it to the interrupt handler, and the handler ***clears*** the interrupt by servicing the device. Figure 3 summarizes the interrupt-driven I/O cycle.
- The basic interrupt mechanism just described enables the CPU to respond to an asynchronous event, as when a device controller becomes ready for service.

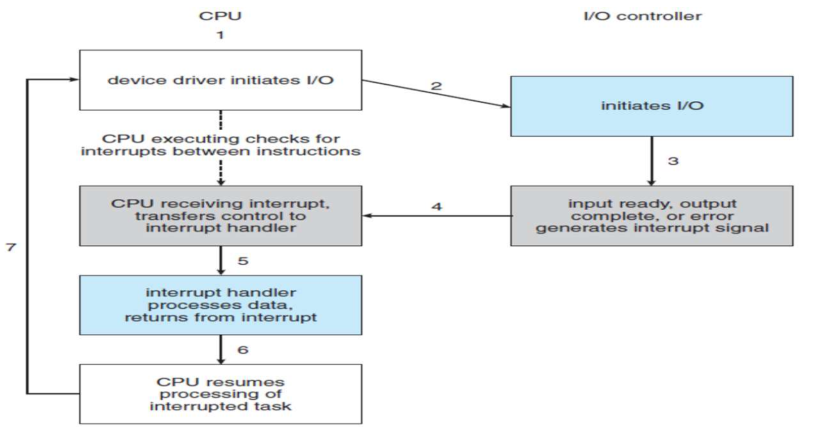
CPU
I/O controller
1
device driver initiates I/O
2
initiates I/O
CPU executing checks for interrupts between instructions
3
CPU receiving interrupt, transfers control to interrupt handler
4
input ready, output complete, or error generates interrupt signal
7
5
interrupt handler processes data, returns from interrupt
6
CPU resumes processing of interrupted task

Figure 13.3 Interrupt-driven I/O cycle.

Application I/O Interface-Kernel I/O structure

- In this section, we discuss structuring techniques and interfaces for the operating system that enable I/O devices to be treated in a standard, uniform way.
- We explain, for instance, how an application can open a file on a disk without knowing what kind of disk it is and how new disks and other devices can be added to a computer without disruption of the operating system.

Application I/O Interface-Kernel I/O structure

- Like other complex software-engineering problems, the approach here involves abstraction, encapsulation, and software layering.
- Specifically, 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.
- Figure 13.6 illustrate how the I/O-related portions of the kernel are structured in software layers.

Application I/O Interface-Kernel I/O structure

- The purpose of the device-driver layer is to hide the differences among device controllers from the I/O subsystem of the kernel, much as the I/O system calls encapsulate the behavior of devices in a few generic classes that hide hardware differences from applications.
- Making the I/O subsystem independent of the hardware simplifies the job of the operating-system developer.
- It also benefits the hardware manufacturers.
- They either design new devices to be compatible with an existing host controller interface (such as SATA), or they write device drivers to interface the new hardware to popular operating systems.
- Thus, we can attach new peripherals to a computer without waiting for the operating-system vendor to develop support code

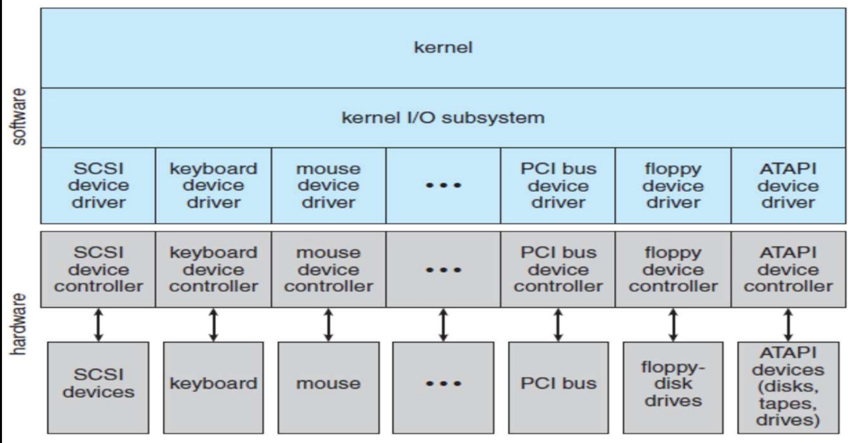

Figure 13.6 A kernel I/O structure.