Operating Systems Principles

cosc1112/cosc1114

School of Science

Semester 2, 2017

Lecture 10 – I/O System

Dr. Ke Deng

ke.deng@rmit.edu.au

Outline

- Overview
- I/O Hardware
- Application I/O Interface
- Kernel I/O Subsystem
- Transforming I/O Requests to Hardware Operations
- STREAMS
- Performance

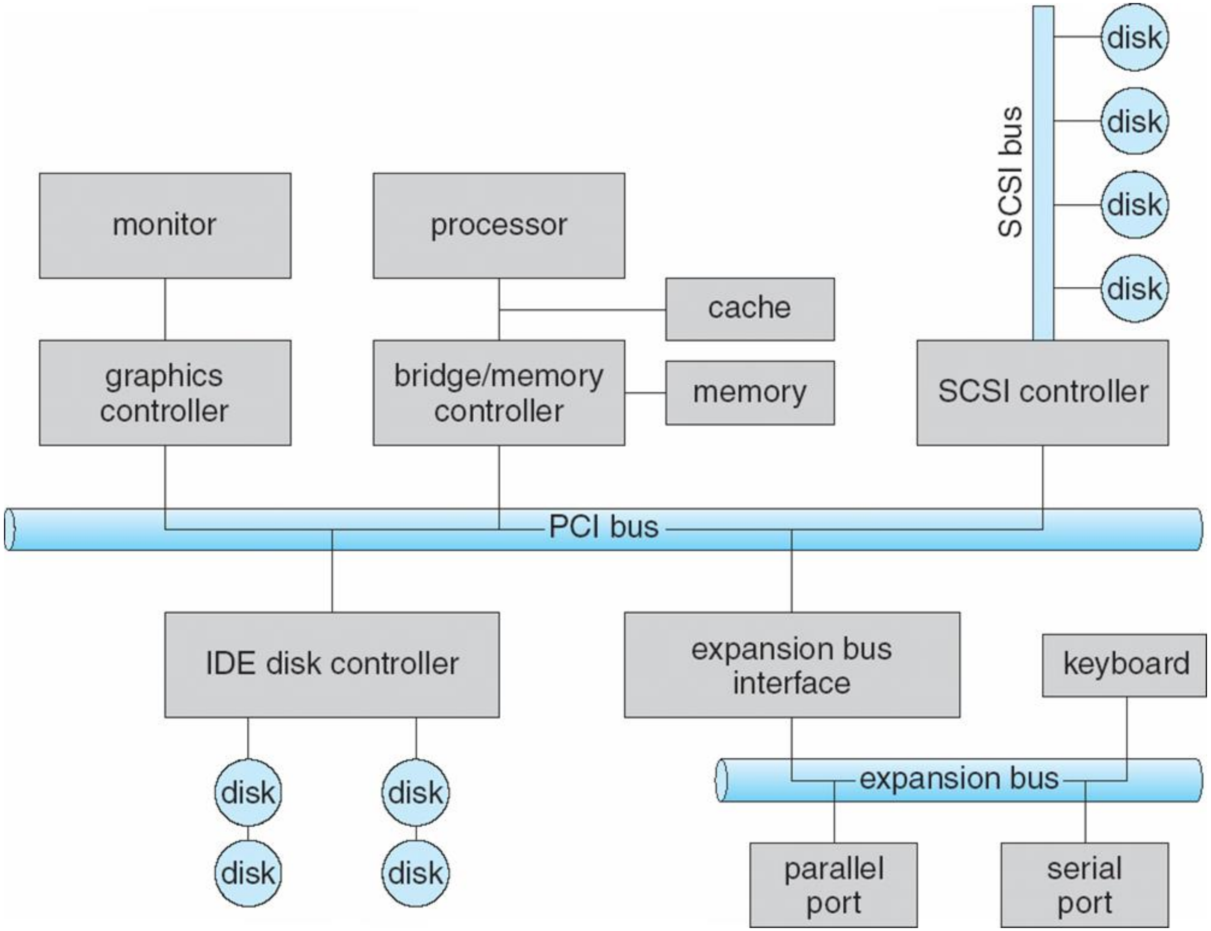
Objectives

- Explore the structure of an operating system's I/O subsystem
- Discuss the principles of I/O hardware and its complexity
- Provide details of the performance aspects of I/O hardware and software

Overview

- I/O management is a major component of operating system design and operation
- Ports, busses, device controllers connect to various devices

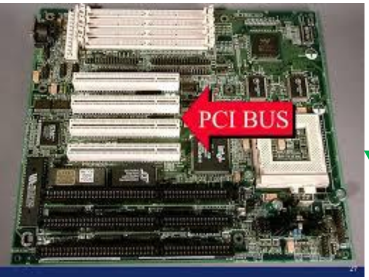
Device drivers encapsulate device details - Present uniform device-access interface to I/O subsystem

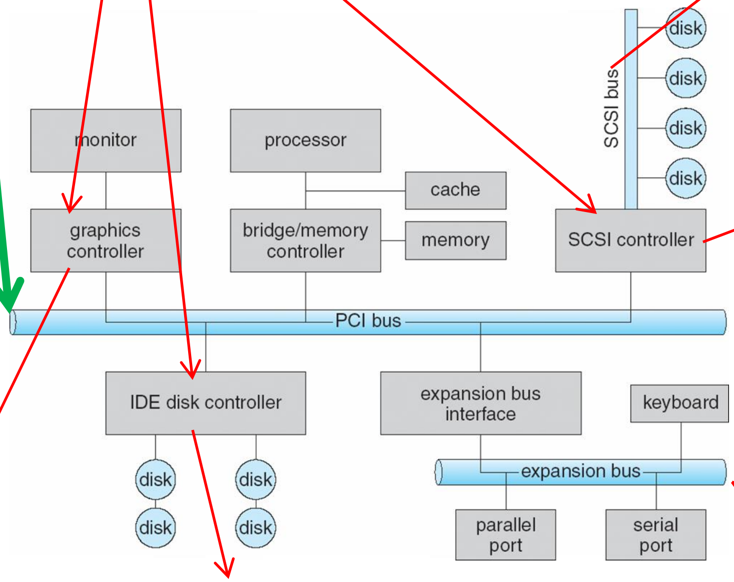


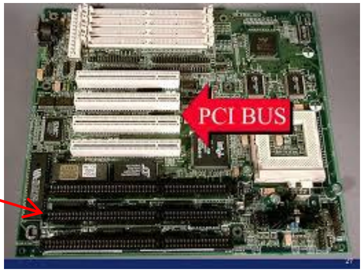
I/O Hardware

https://en.wikipedia.org/wiki/Low_Pin_Count

- Common concepts – signals from I/O devices interface with computer
  - **Port** – connection point for device
  - **Bus** - **daisy chain** or shared direct access
    - **PCI** bus common in PCs and servers, PCI Express (**PCIe**)
    - **expansion bus** connects relatively slow devices
  - **Controller** (**host adapter**) – electronics that operate port, bus, device
    - Sometimes integrated
    - Sometimes separate circuit board (host adapter)
    - Contains processor, microcode, private memory, bus controller, etc
      - Some talk to per-device controller with bus controller, microcode, memory, etc





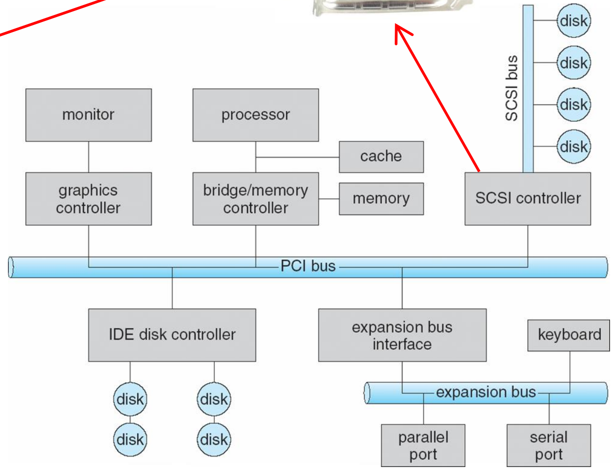


HP Quadro FX 1300 Graphics Controller
PCI Express x16

PCI Express

ISA bus first appeared in PC based on 80286 CPU. The idea was to put a standard in interfacing with peripheral devices. In our days this bus is kept for compatibility with "legacy" peripheral devices such as BIOS ROM,Parallel and Serial Ports,Sound and Modem devices.

I/O Hardware (Cont.)



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

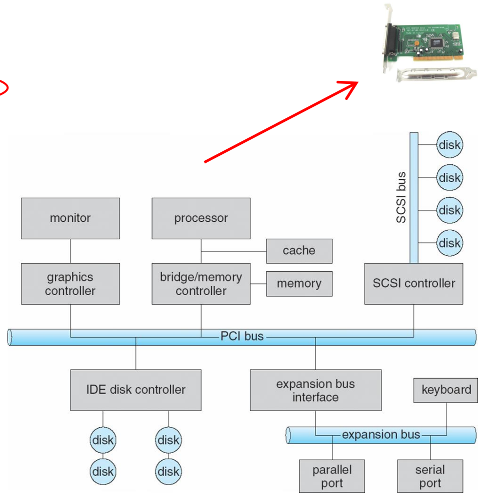
- Devices have registers (typically 1-4 bytes in size, or FIFO buffer)
  - data-in register - read by the host to get input
  - data-out register - written by the host to send output
  - status register - read by the host to indicate such as current command has completed
  - control register – read by the host to start a command and change mode of a device (e.g. parity checking)

- Device driver communicate with registers by two ways
  - Direct I/O instructions
    - Use 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.
  - Memory-mapped I/O
    - 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 the write the device control-registers.
    - For example, the graphics controller has I/O ports for basic control operations; but the controller has a large-memory mapped region to hold screen contents. The process send outputs to the screen by writing data into the memory-mapped region.

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

Polling

- For each byte of I/O
  1. The host repeatedly reads the busy bit in status register until it becomes 0;
  2. The host sets the write bit in the command register and write a byte into the data-out register;
  3. The host sets command-ready bit in command register;
  4. When the controller notices that the command-ready bit is set, it sets the busy bit;
  5. The controller reads the command register and sees the write command, it reads the data-out register to get the bytes and does the I/O to the device;
  6. when transfer done, device controller clears the busy bit, error bit in status register, command-ready bit in command register;



Step 1 is busy-wait or polling cycle to wait for I/O from device

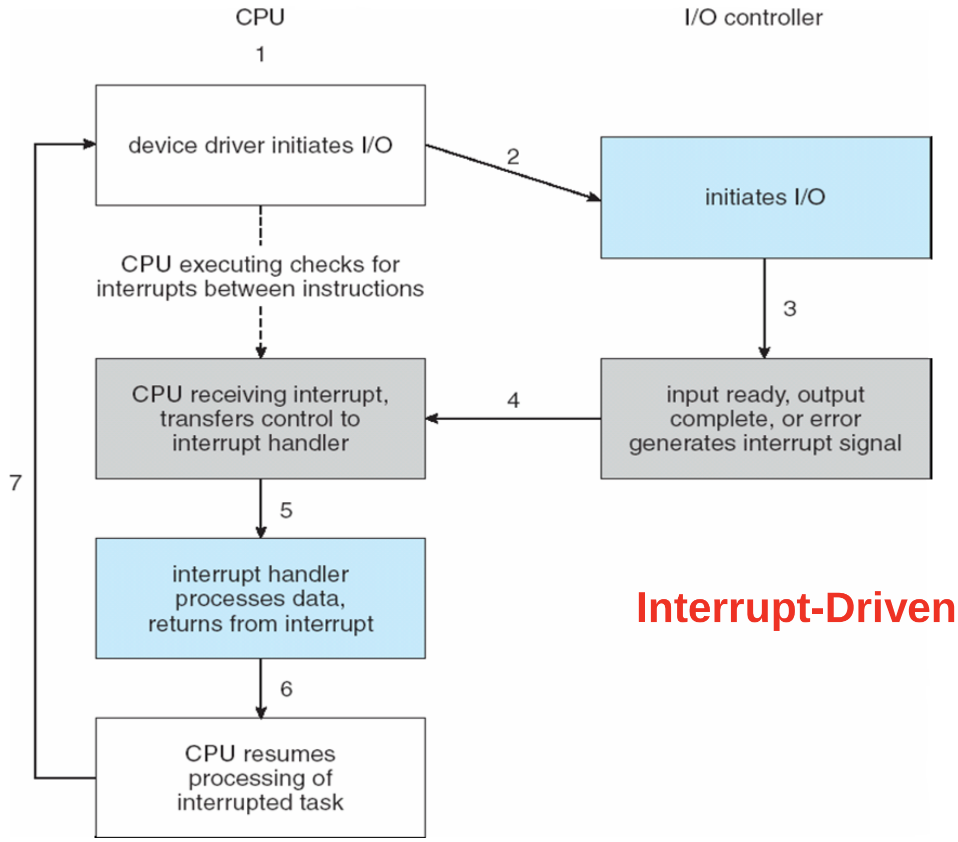
Polling can happen in 3 instruction cycles

- Read status fro status register
- logical-and to extract status bit
- branch if not zero

How to be more efficient if non-zero infrequently?

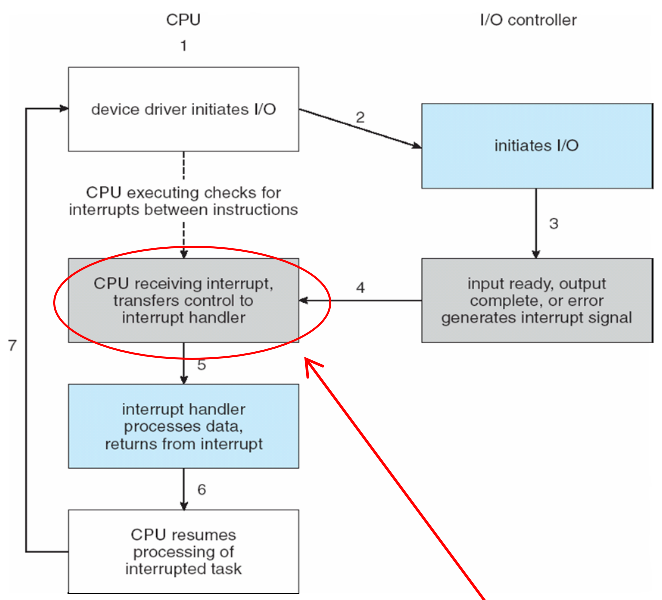
Interrupts

- The CPU has a wire called **Interrupt-request line**
  - It is checked by processor after each instruction
  - When the CPU detects that a device controller has asserted a signal on the interrupted line, the CPU saves current state and jumps to the interrupt-handler routine at a fixed address in memory
  - The interrupt handler determined the cause of the interrupt, performing the necessary processing.
  - The interrupt handler returns the CPU to the execution state prior to the interrupt.



Interrupt-Driven I/O Cycle

Interrupts (cont.)



Interrupt vector contains the memory addresses of specialized interrupt handler.

Intel Pentium Processor Event-Vector Table

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

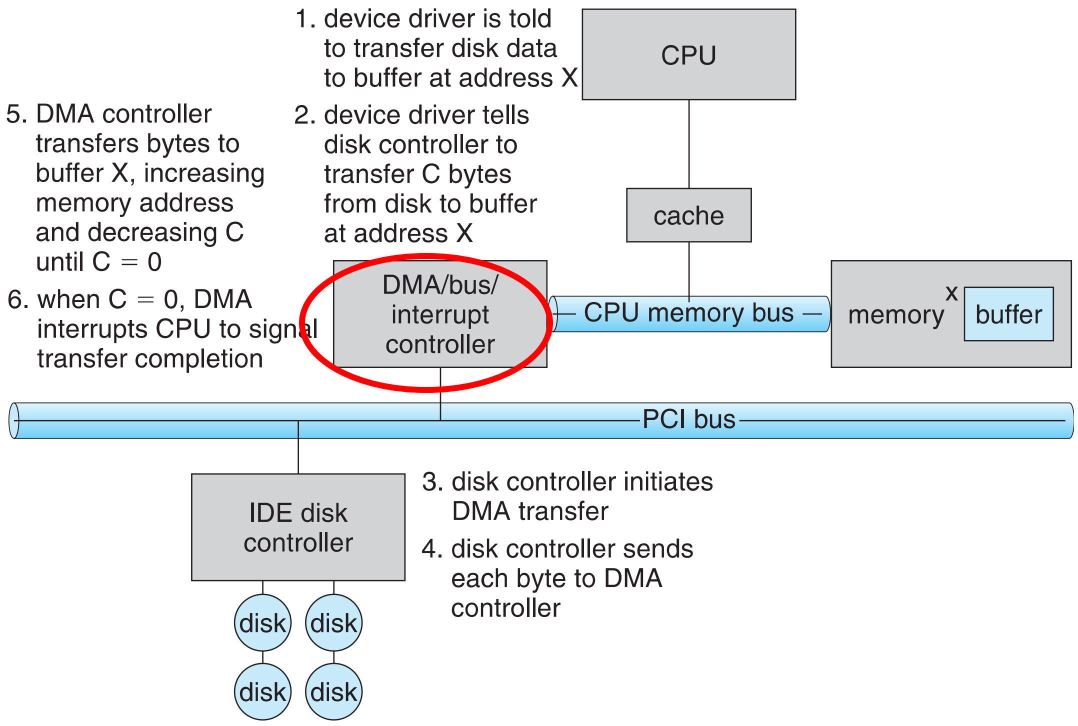
Interrupts (Cont.)

- Software generated interruption
  - Interrupt mechanism also used for **exceptions**

    Terminate process, crash system due to hardware error
  - **Page fault** executes when memory access error
  - System call executes via **trap** to trigger kernel to execute request
- Multi-CPU systems can process interrupts concurrently
  - If operating system designed to handle it

Direct Memory Access (DMA)

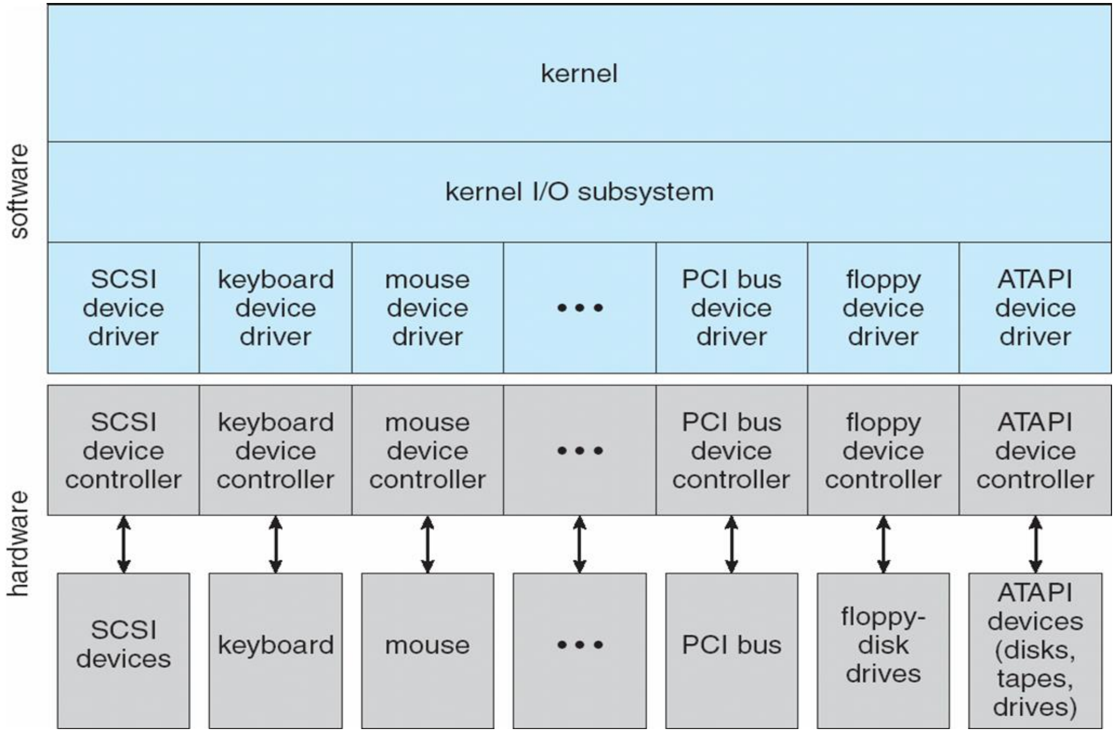
- For large data I/O, **DMA** controller can be used; Bypasses CPU to transfer data directly between I/O device and memory



- A simple DMA controller is a standard component in PCs.
- While DMA controller seizes the memory bus, the CPU is momentarily prevented from accessing main memory – cycling stealing. But CPU can still access its primary and secondary caches.
- Some computer architectures perform direct virtual memory access (DVMA): translate the virtual memory address to physical memory.

Application I/O Interface

- Device-driver layer hides differences among I/O controllers from kernel; Each OS has its own I/O subsystem structures and device driver frameworks
- Devices vary in many dimensions
  - Character-stream or block
  - Sequential or random-access
  - Synchronous or asynchronous (or both)
  - Sharable or dedicated
  - Speed of operation
  - read-write, read only, or write only



Block and Character Devices

- Block devices include disk drives
  - Commands include read, write, seek
  - Raw I/O, direct I/O, or file-system access
  - Memory-mapped file access possible
    - File mapped to virtual memory and clusters brought via demand paging
  - DMA
- Character devices include keyboards, mice, serial ports
  - Commands include **`get()`, `put()`**
  - Libraries layered on top allow line editing

Network Devices

- Varying enough from block and character to have own interface
- Linux, Unix, Windows and many others include **socket** interface
  - Separates network protocol from network operation
  - Includes **`select()`** functionality
- Approaches vary widely (pipes, FIFOs, streams, queues, mailboxes)

Clocks and Timers

- Provide current time, elapsed time, timer
- Normal resolution about 1/60 second
- Some systems provide higher-resolution timers
- **Programmable interval timer** used for timings, periodic interrupts
- **`ioctl()`** (on UNIX) covers odd aspects of I/O such as clocks and timers

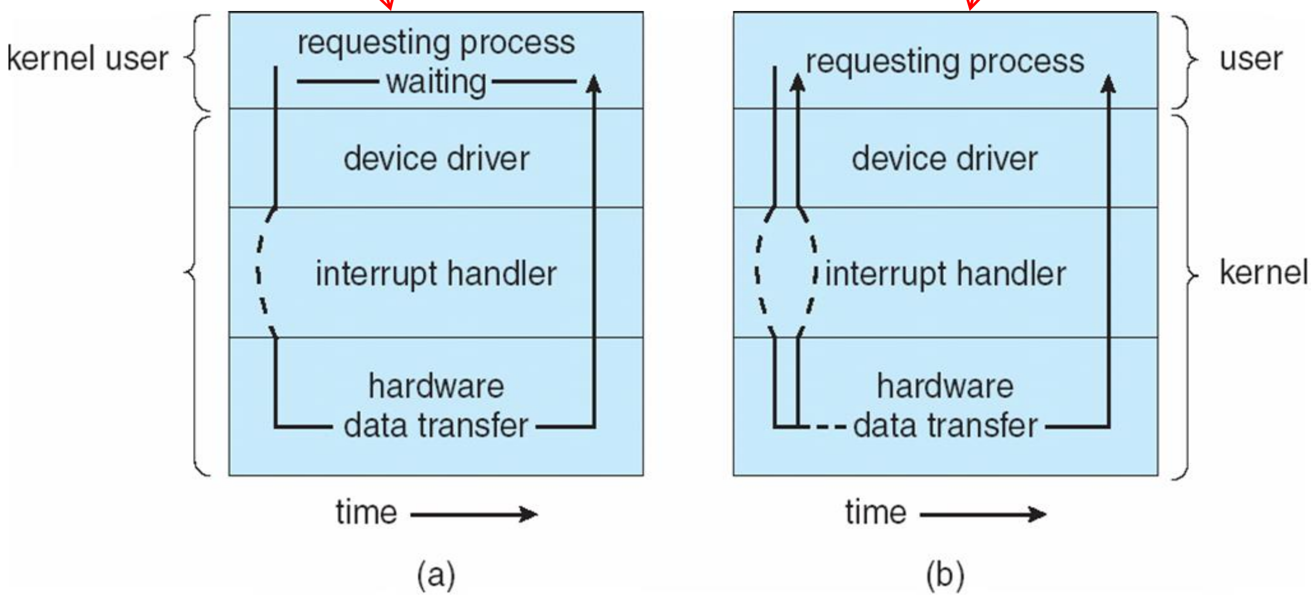
Nonblocking and Asynchronous I/O

Blocking I/O System Call

- When an application issues a blocking system call, the execution of the application is suspended. process suspended until I/O completed;
- The application is moved to a wait queue;
- After the system call completes, the application resume.
- When it resume execution, it will receive the values returned by the system call.
- Synchronous I/O

Non-blocking I/O System Call

- Non-blocking I/O system call returns immediately without waiting for the I/O to complete.
- For example, a video application that reads frames from a file on disk while simultaneously decompressing and displaying on the display.
- Asynchronous I/O



Synchronous

Asynchronous

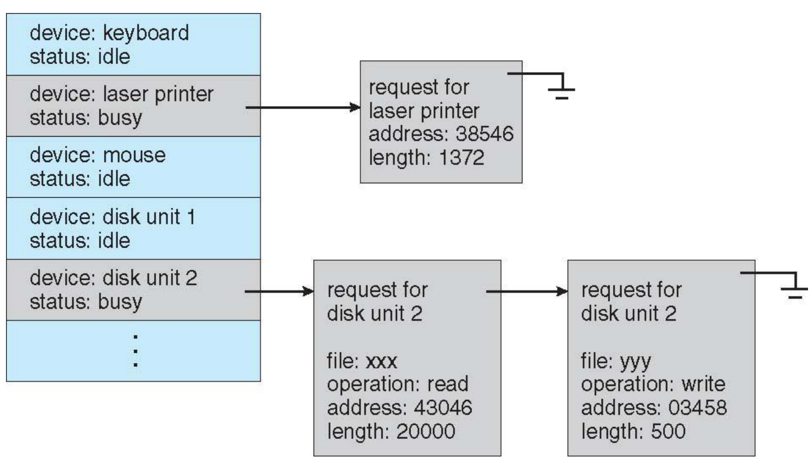
Kernel I/O Subsystem

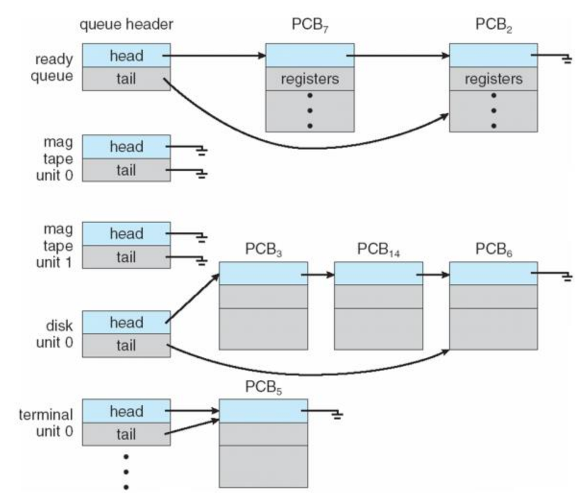
Kernels provide many services related to I/O including

- **I/O Scheduling**

  To determine a good order in which to execute a set of I/O requests

  - Operating system implement scheduling by maintaining a wait queue of requests for each device (e.g., disk, CPU)..
  - When an application issues a <u>blocking I/O</u> system call, the request is placed on the queue for that device; the I/O scheduler rearranges the order of the queue.
  - When a kernel supports <u>asynchronous I/O</u>, it must be able to keep track of many I/O requests at the same time. For this purpose, the operating system might attach the wait queue to a device –status table.
    - Each I/O device – an entry in table
    - Each table entry indicates the device type, address, state; if the device is busy with a request, the type of request and the other parameters will be stored in the table entry.





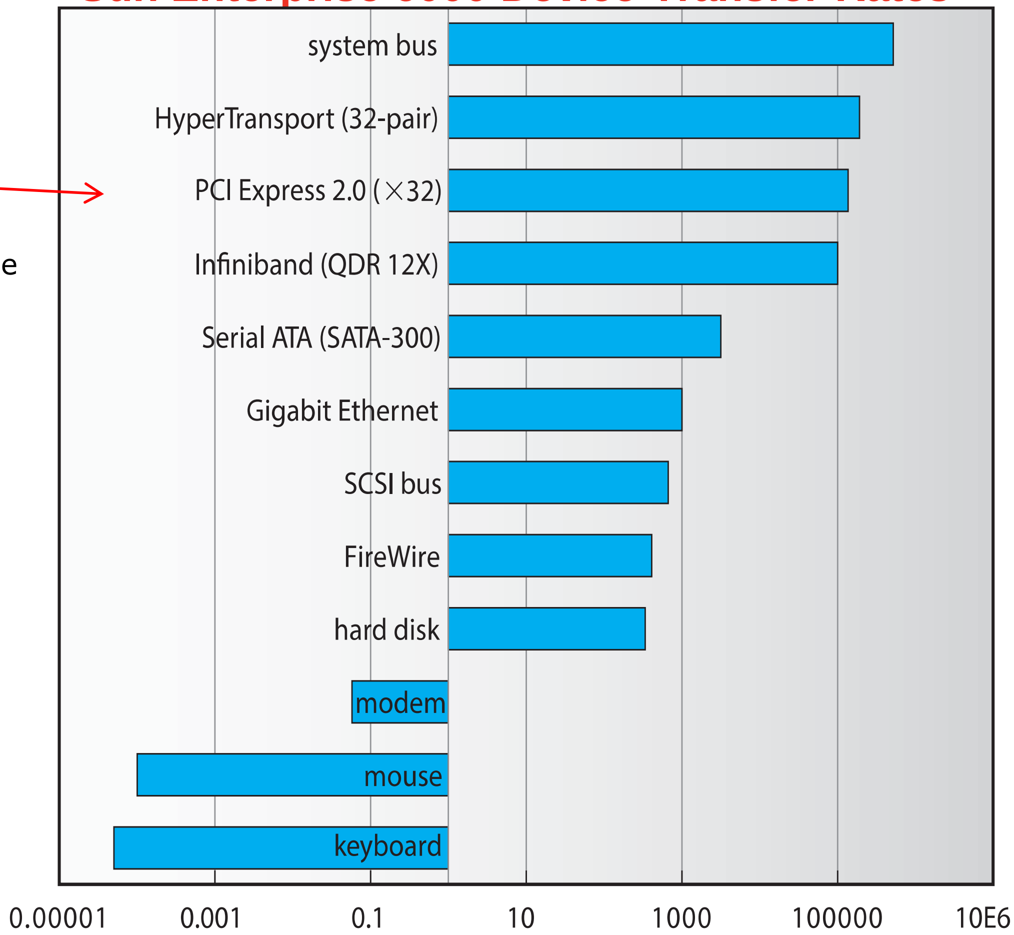
Kernel I/O Subsystem (cont.)

Buffering I/O

- A buffer is a memory area that stores data being transferred between two devices or between a device and an application.

- Buffering is done for three reasons
  - To cope with device speed mismatch
    E.g., a file is received via modem for storage on the hard disk
  - To cope with device transfer size mismatch
    E.g., a large message is fragmented into small network packets; the packets are sent over the network, and the receiving side places them in a buffer to form the image of the source data.
  - To maintain "copy semantics"
    Application buffer changes while kernel buffer has no effect

Sun Enterprise 6000 Device-Transfer Rates



Kernel I/O Subsystem (cont.)

- ## Caching
  - A region of fast memory that holds copies of data
  - Caching and buffering are distinct functions, but sometimes a region of memory can be used for both purposes.
- ## Spooling
  - A spool is a buffer that holds output for a device, such as a printer, that can serve only one request at a time
- ## Error Handling
  - An operating system uses protected memory to guard against many kinds of hardware and application errors.
  - Devices and I/O transfers can fail in many ways
    - Transient reason – network overloaded – can be effectively solved by Operating System
    - Permanent reason – disk controller failure – unlikely to be recovered by Operating System

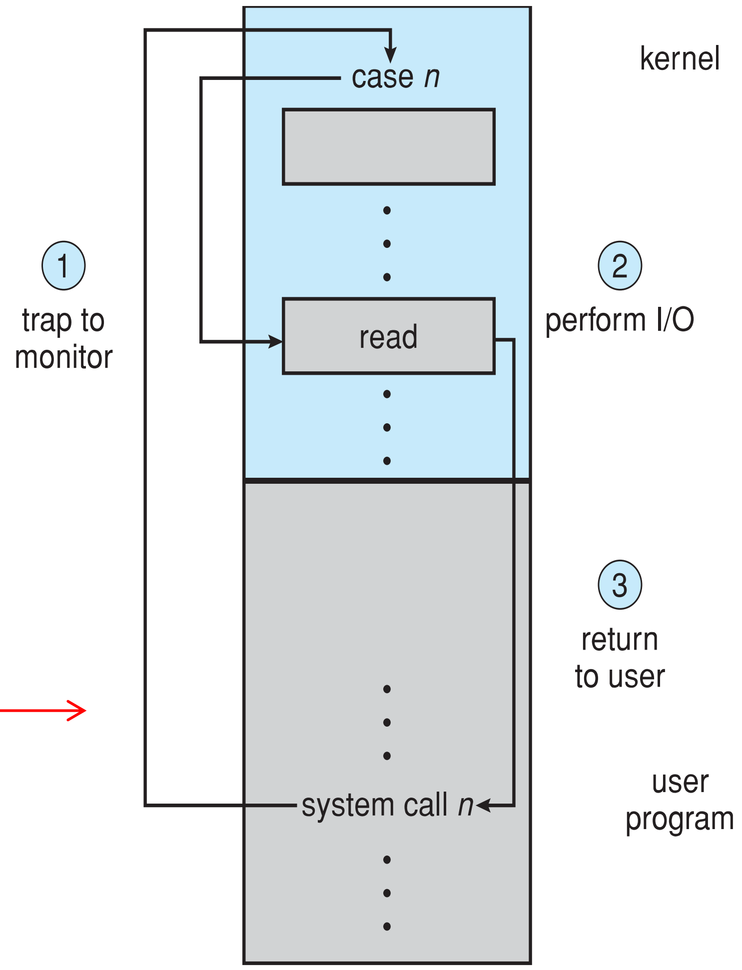
Kernel I/O Subsystem (cont.)

Use of a System Call to Perform I/O

- **I/O Protection**

  A user process may accidentally or purposefully attempt to disrupt normal operation via illegal I/O instructions. To prevent users from performing illegal I/O

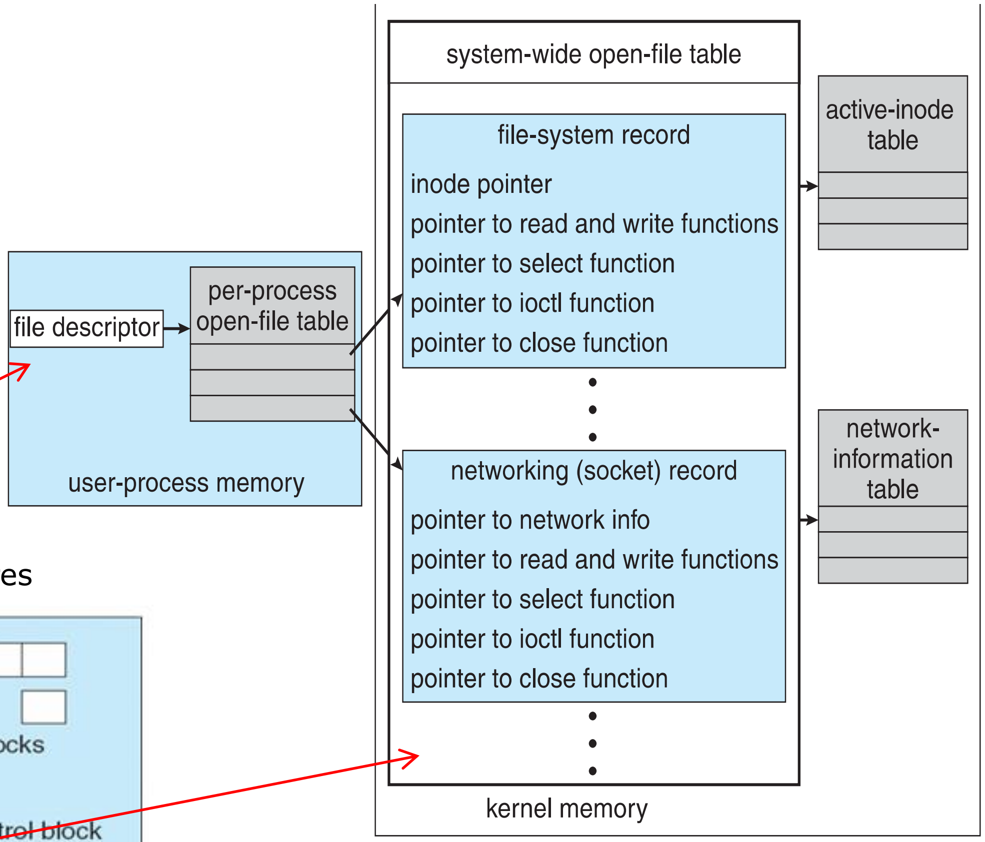
  - We define all I/O instructions to be privileged instructions; thus users cannot issue I/O instructions directly; they must do it through the operating system, i.e., using system call to request the operating system to perform I/O.
  - Memory-mapped and I/O port memory locations must be protected from user access.



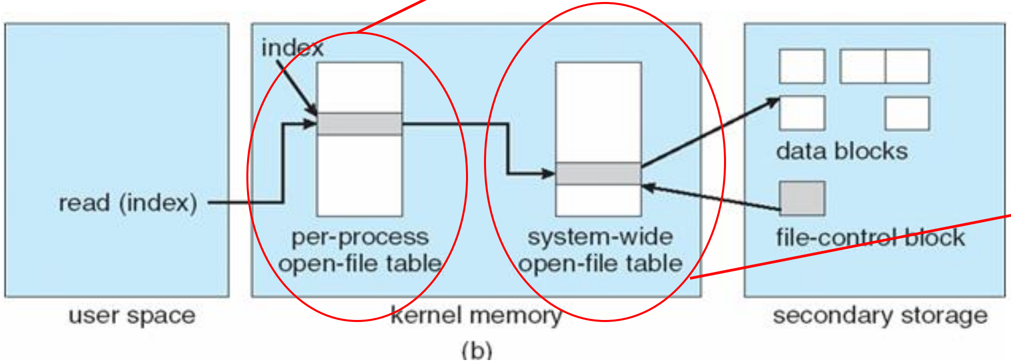
Kernel Data Structures

- Kernel needs to keep state information of I/O components, including open file tables, network connections, character device state.
  - It does so through a variety of in-kernel data structures, such as the open-file table.

- Unix provides file-system access to a variety of entities, such as user files, raw devices.



Week 9 file system - In-Memory File System Structures



Transforming I/O Requests to Hardware Operations

- Consider reading a file from disk for a process:
  - A process refers to the data by a file name;
  - Within a disk, the file system maps from the file name through the file-system directories to obtain the space allocation of the file;
  - Physically read data from disk into buffer
  - Make data available to the requesting process
  - Return control to process

Life Cycle of An I/O Request

A process issues a blocking read system call to a file that has been opened previously

The system call code checks;

if the data is already in the buffer cache, the data are returned to the process and the I/O request is completed.

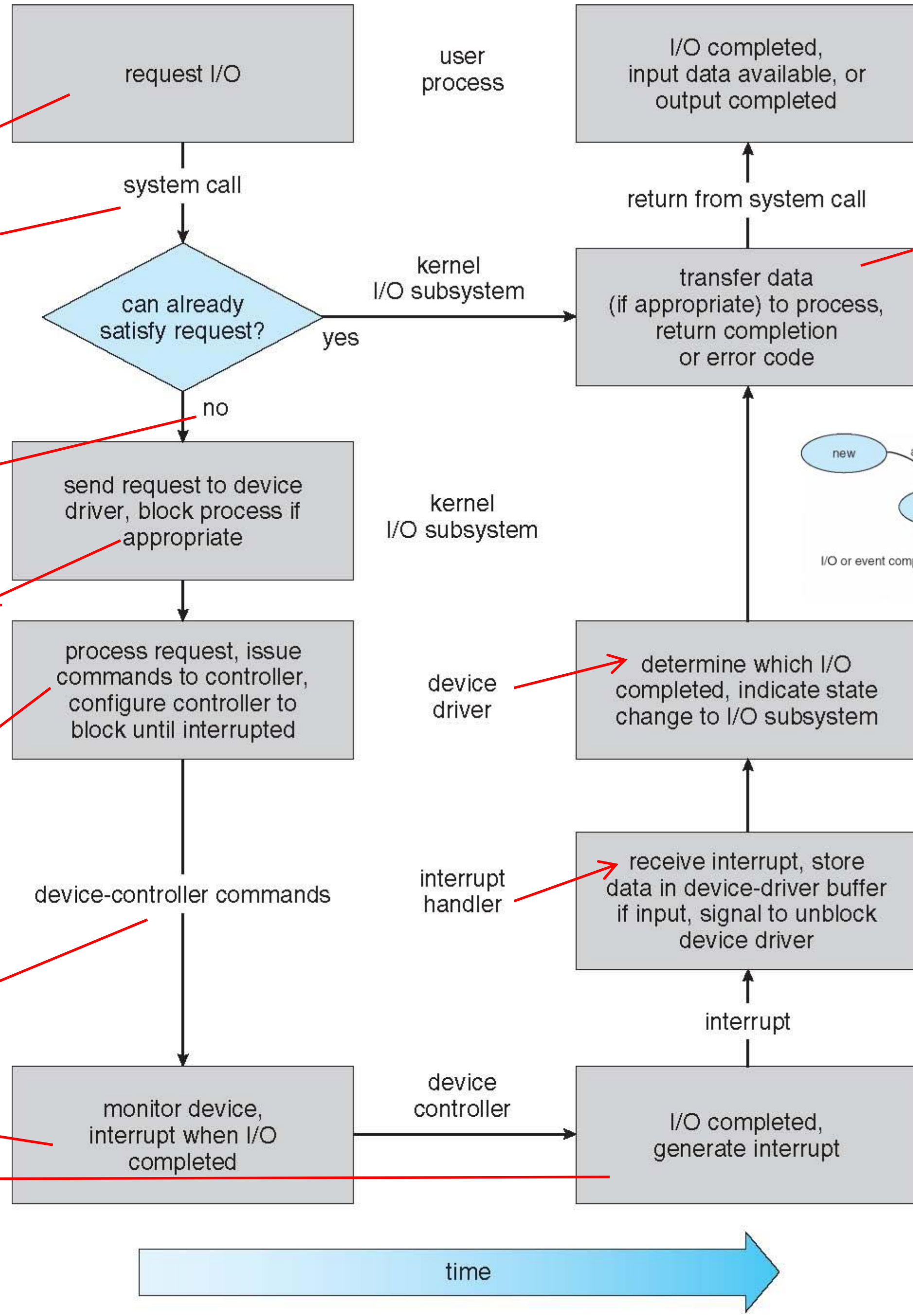
Otherwise, a physical I/O must be performed.

A process is removed to waiting queue

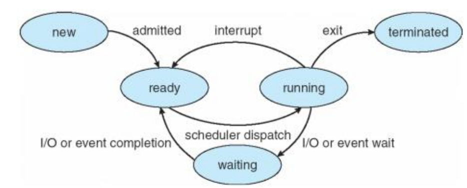
I/O request is scheduled, eventually the I/O subsystem sends the request to the device driver; The device driver allocates kernel buffer space to receive the data.

The device driver sends commands to the device controller by writing into the device-control registers.

Assume DMA manage the data transfer, when the transfer complete, an interrupt is generated.



The kernel transfers data to the address space of the requesting process and moves the process fro the wait queue back to ready queue.



Next Week

Lecture 11 – Protection

Tutorial 10