0117401: Operating System
计算机原理与设计
Chapter 13: IO Systems (IO管理)

陈香兰
xlanchen@ustc.edu.cn
http://staff.ustc.edu.cn/~xlanchen

Computer Application Laboratory, CS, USTC @ Hefei
Embedded System Laboratory, CS, USTC @ Suzhou

June 1, 2016

温馨提示：

为了您和他人的工作学习，
请在课堂上关机或静音。

不要在课堂上接打电话。

提纲

Chapter Objectives

- Explore the structure of an OS'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 devices
  - vary widely
- The control of devices connected to the computer is a major concern of OS designers.

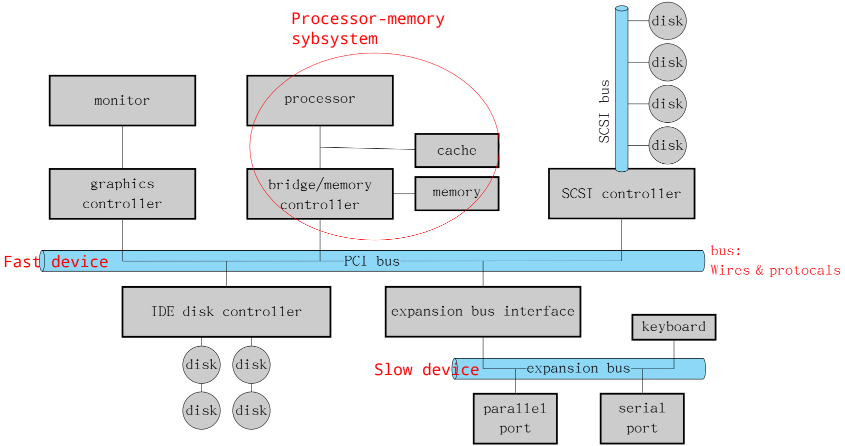
How OS manages and controls various peripherals?

Outline

1. I/O Hardware
   - Polling (轮询方式)
   - Interrupts (中断方式)
   - Direct Memory Access (DMA方式)
   - I/O hardware summary

I/O Hardware

- Incredible variety of I/O devices



I/O Hardware

- Common concepts : CPU→PORT→BUS→Controller
  - Port (端口)
  - Bus (总线) (daisy chain(菊花链) or shared direct access)
    - PCI (Peripheral Component Interconnect(外部器件互连))
    - SCSI (Small computer systems interface)
    - Expansion bus
  - Controller (控制器) (host adapter)
- How can the processor command controller?
  - Controller has one or more registers for data and control signals.
  - The processor communicates with the controller by reading and writing bit patterns in the registers.

I/O Hardware

- Two communication techniques:
  1. Direct I/O instructions
     - Access the port address
     - Each port typically contains of four registers, i.e., status, control, data-in and data-out.
     - Instructions: In, out
  2. Memory-mapped I/O
     - Example: 0xa0000 ~ 0xfffff are reserved to ISA graphics cards and BIOS routines
- Some systems use both techniques.

I/O Hardware

- I/O address range

Device I/O Port Locations on PCs (partial)

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)

I/O Control Methods

1. Polling (轮询方式)
2. Interrupts (中断方式)
3. DMA (DMA方式)
4. (在汤书上：还有通道的概念)

Outline

1. I/O Hardware
   - Polling (轮询方式)
   - Interrupts (中断方式)
   - Direct Memory Access (DMA方式)
   - I/O hardware summary

Polling（轮询方式）

- Need **handshaking（握手）**
- State of device
  1. **command-ready**
     - In command register
     - 1: a command is available for the controller
  2. **busy**
     - In status register
     - 0: ready for the next command; 1: busy
  3. **Error**
     - To indicate whether an I/O is ok.

Polling（轮询方式）

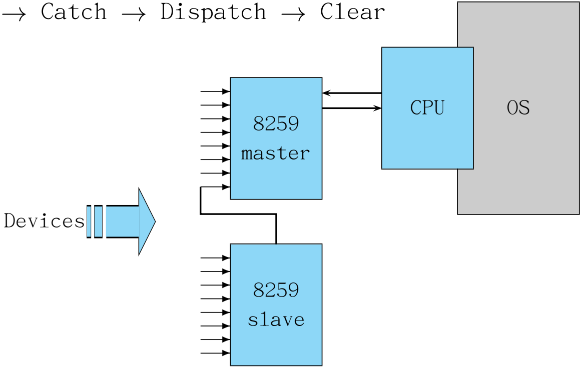
- Basic handshaking notion for writing output
  1. Host repeatedly reads the busy bit until it is 0
  2. Host sets write bit in command register and writes a byte into data-out register
  3. Host sets command-ready bit
  4. When controller notices command-ready, sets busy bit
  5. Controller gets write command and data, and works
  6. Controller clears command-ready bit, error bit and busy bit
- Step1: Busy-wait cycle to wait for I/O from device ≡polling

Outline

1. I/O Hardware
   - Polling (轮询方式)
   - Interrupts (中断方式)
   - Direct Memory Access (DMA方式)
   - I/O hardware summary

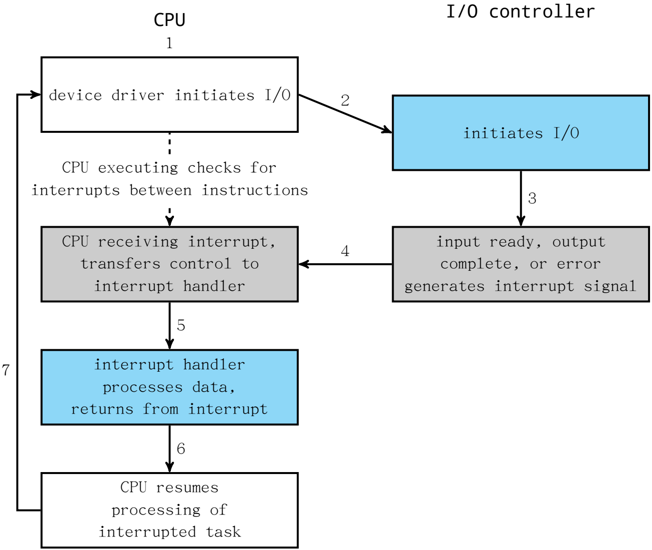
Interrupts (中断方式)

- CPU Interrupt-request line triggered by I/O device
- Interrupt handler receives interrupts
- Basic interrupt scheme
  - Raise → Catch → Dispatch → Clear



Interrupts（中断方式）

- Interrupt-Driven I/O Cycle



Interrupts (中断方式)

- More sophisticated interrupt-handling features:
  Most CPU have **two interrupt request line**.
  1. Nonmaskable
  2. Maskable to ignore or delay some interrupts
- Efficient dispatching without polling the devices
  - Interrupt vector: to dispatch interrupt to correct handler
  - Interrupt chaining: to allow more device & more interrupt handlers
- Distinguish between high- and low-priority interrupts:
  - Interrupt priority: the handling of low-priority interrupts is deferred without masking, even preempted.
- Interrupt mechanism also used for exceptions

Interrupts（中断方式）

- Example: Intel Pentium Processor Event-Vector Table

vector number	description	vector number	description
0	divide error	11	segment no present
1	debug exception	12	stack fault
2	null interrupt	13	general protection
3	breakpoint	14	page fault
4	INTO-detected overflow	15	(Intel reserved, do not use)
5	bound range exception	16	floating-point error
6	invalid opcode	17	alignment check
7	device not available	18	machine check
8	double fault	19-31	(Intel reserved, do not use)
9	coprocessor segment overrun (reserved)	32-255	maskable interrupts
10	invalid task state segment		

Outline

1. I/O Hardware
   - Polling (轮询方式)
   - Interrupts (中断方式)
   - Direct Memory Access (DMA方式)
   - I/O hardware summary

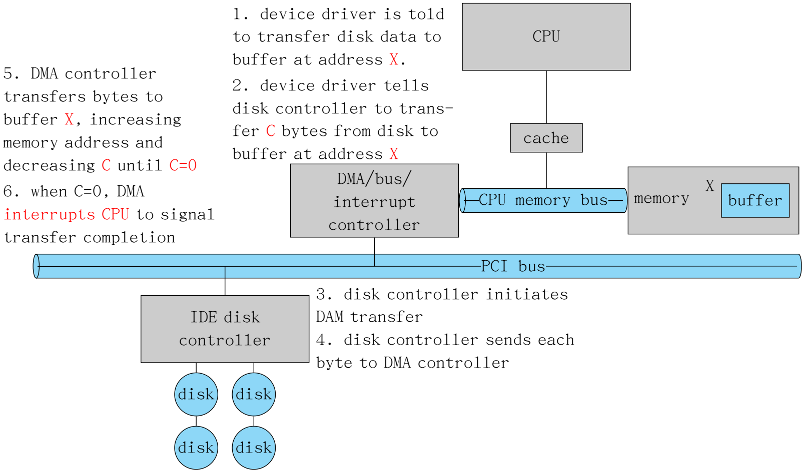
- **Direct Memory Access (DMA方式):**
  Used to avoid programmed I/O **for large data movement**, and bypasses CPU to **transfer data directly between I/O device and memory**
- Requires **DMA controller**
  - the host prepares a **DMA command block** in memory
    - a pointer to the source of a transfer
    - a pointer to the destination of the transfer
    - a count of the number of bytes to be transfered
  - CPU writes the address of the DMA command block to DMA controller, and then goes on with other work.

Direct Memory Access (DMA方式)

- Handshaking between DMA controller & device controller
  1. Device controller **raises DMA-request** when one word is available
  2. DMA controller **seizes memory bus**, places the desired address on memory-address wires, and raises DMA-acknowledge
  3. Device controller **transfers** the word to memory, and removes the DMA-request signal. Goto 1
  4. DMA controller **interrupts** the CPU.

Direct Memory Access (DMA方式)

- Six Step Process to Perform DMA Transfer



- Cycle stealing: when DMA seizes the memory bus, CPU is momentarily prevented from accessing main memory

Outline

1. I/O Hardware
   - Polling (轮询方式)
   - Interrupts (中断方式)
   - Direct Memory Access (DMA方式)
   - I/O hardware summary

I/O hardware summary

- A bus
- A controller
- An I/O port and its registers
- The handshaking relationship between the host and a device controller
- The execution of this handshaing in a pooling loop via interrupts
- the offloading of this work to a DMA controller for large transfer

Outline

2 Application I/O Interface
- Block and Character Devices
- Network Devices
- Clocks and Timers
- Blocking (阻塞) and Nonblocking (非阻塞) I/O

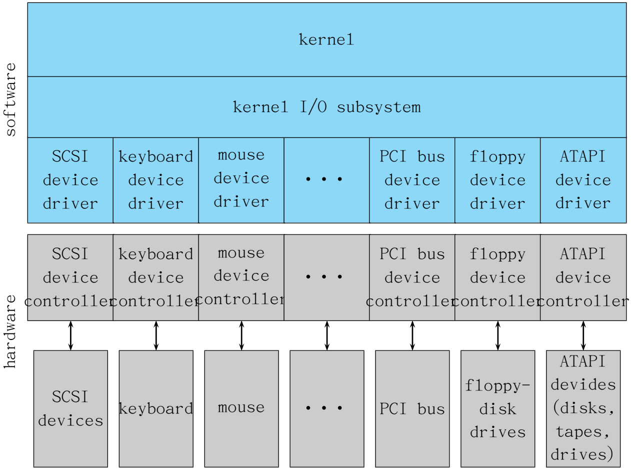
I/O control challenges

- Wide variety of devices
- Two challenges

  Applications $\rightarrow$ OS $\leftarrow$ Devices

  - How can the OS give a convenient, uniform I/O interface to applications?
  - How can the OS be designed such that new devices can be attached to the computer without the OS being rewritten?
- For device manufacturers, **device-driver layer** hides differences among I/O controllers from kernel

I/O control challenges



A Kernel I/O Structure

Application I/O Interface

- For applications, **I/O system calls** encapsulate device behaviors in generic classes
- 设备独立性：应用程序与具体的物理设备无关。
- Device-driver layer hides differences among I/O controllers from kernel
- Devices vary in many dimensions
  - Character-stream or block
  - Sequential or random-access
  - Sharable or dedicated
  - Speed of operation
  - read-write, read only, or write only

Characteristics of I/O Devices

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read-write	CD-ROM graphics controller disk

Major Device Access Conventions

- Block I/O
- Character-stream I/O
- Memory-mapped file access
- Network sockets
- Clock and Time

Outline

Block and Character Devices

1. Block devices include disk drives
   - Commands include read, write, seek
   - Raw I/O or file-system access
   - Memory-mapped file access possible
2. Character devices include keyboards, mice, serial ports
   - Commands include get, put
   - **Libraries** layered on top allow line editing

2 Application I/O Interface
- Block and Character Devices
- Network Devices
- Clocks and Timers
- Blocking (阻塞) and Nonblocking (非阻塞) I/O

Network Devices

- Varying enough from block and character to have own interface
- Unix and Windows NT/9x/2000 include socket interface
  - Separates network protocol from network operation
  - Server — socket, bind, listen, accept
  - Client — socket, connect
  - Includes select functionality
- Approaches vary widely (pipes, FIFOs, streams, queues, mailboxes)

Outline

2 Application I/O Interface
- Block and Character Devices
- Network Devices
- Clocks and Timers
- Blocking (阻塞) and Nonblocking (非阻塞) I/O

Clocks and Timers

- Provide current time, elapsed time, timer
- Hardware clocks
  1. Real Time Clock (RTC, 实时时钟)
  2. Time Stamp Counter (TSC, 时间戳计数器)
  3. Programmable Interval Timer (PIT, 可编程间隔定时器)
     - used for timings, periodic interrupts
- ioctl (on UNIX) covers odd aspects of I/O such as clocks and timers

1. **Real Time Clock (RTC, 实时时钟)**
   - Integrated with CMOS RAM, always tick.
   - Seconds from **00:00:00 January 1, 1970 UTC**
   - Can be used as an alarm clock
     - IRQ8
     - Interrupt frequency: 2HZ~8192HZ
   - I/O address (port no): 0x70, 0x71
   - Example:
     - Motorola 146818: CMOS RAM + RTC
   - Second↔ year, month, date, week **HOW?**

2. Time Stamp Counter (TSC, 时间戳计数器)
   - 64bit TSC register in the processor
     - Pentium and after
   - Incremented at each clock signal on CLK input pin
     - example: CPU frequency 400MHZ
       adds 1 per 2.5 ns = adds 400×10^6 per second
   - Instruction: rdtsc
   - How to know CPU frequency?

3. Programmable Interval Timer (PIT, 可编程间隔定时器)
   - 8253, 8254
   - Issues **time interrupt** in a programmable time internal
   - Can also be used to calculate processor frequency during boot up.
   - 8253
     - 14,3178 MHz crystal ⇒4,772,727 Hz system clock ⇒1,193,180 Hz to 8253
     - using 16 bit divisor ⇒ interrupt every 838 ns ~ 54.925493 ms

Outline

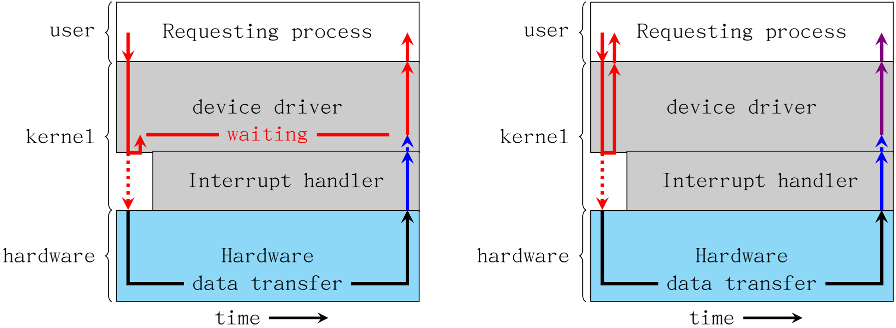
2 Application I/O Interface

- Block and Character Devices
- Network Devices
- Clocks and Timers
- Blocking (阻塞) and Nonblocking (非阻塞) I/O

Blocking（阻塞）and Nonblocking（非阻塞）I/O

- **Blocking**（阻塞）— process suspended until I/O completed
  - Easy to use and understand
  - Insufficient for some needs
- **Nonblocking**（非阻塞）— I/O call returns as much as available
  - User interface, data copy (buffered I/O)
  - Implemented via **multi-threading**
  - Returns quickly with count of bytes read or written
  - **Asynchronous**（异步）— process runs while I/O executes
    - Difficult to use
    - I/O subsystem signals process when I/O completed

Two I/O Methods



(a)
Synchronous

(b)
Asynchronous

Outline

3 Kernel I/O Subsystem

- I/O Scheduling
- Buffering (缓冲机制)
- Caching, Spooling & device reservation
- Error Handling
- I/O Protection
- Kernel Data Structures

- Kernel I/O Subsystem Services
  1. I/O Scheduling
  2. Buffering
  3. Caching
  4. Spooling
  5. Device reservation
  6. Error handling

Outline

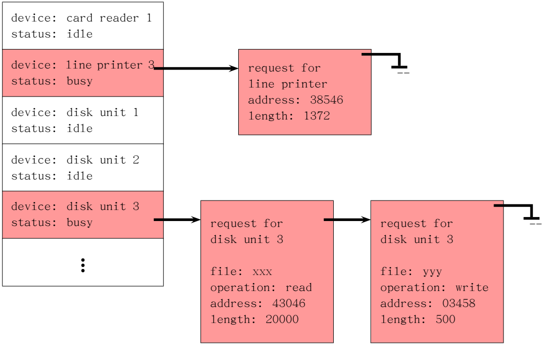
3 Kernel I/O Subsystem
- I/O Scheduling
- Buffering (缓冲机制)
- Caching, Spooling & device reservation
- Error Handling
- I/O Protection
- Kernel Data Structures

I/O Scheduling

- I/O scheduling:
  To schedule a set of I/O requests **means** to determine a good order in which to execute them
  - Origin order: the order in which applictions issue system calls: **May NOT the best order!**
  - Scheduling can
    - Improve overall system performance
    - Share device access fairly among processes
    - Reduce the average waiting time for I/O to complete
  - Example: Disk read request from Apps.
    App1: 0; App2: 100; App3: 50;
    Now at 100;
    The OS may serve the applications in the order App2, App3, App1.

I/O Scheduling

- OS maintaining a wait queue of request for each device
  - **Device-status Table**



  - I/O scheduling,
    Some OSes try fairness, some not

I/O Scheduling

- Another way to improve performance is by using storage space in main memory or on disk
  - Buffering (缓冲机制)
  - Caching
  - Spooling

Outline

3 Kernel I/O Subsystem
- I/O Scheduling
- Buffering (缓冲机制)
- Caching, Spooling & device reservation
- Error Handling
- I/O Protection
- Kernel Data Structures

- Buffering（缓冲机制）
  - Buffer — A memory area that stores data while they are transferred between two devices or between a device and an application
  - Store data in memory while transferring between devices
- Why buffering?
  1. To cope with device speed mismatch.
     Example: Receive a file via modem and store the file to local hard disk.
     - Speed: The modem is about a thousand times slower than the hard disk.
     - Two buffers are used.

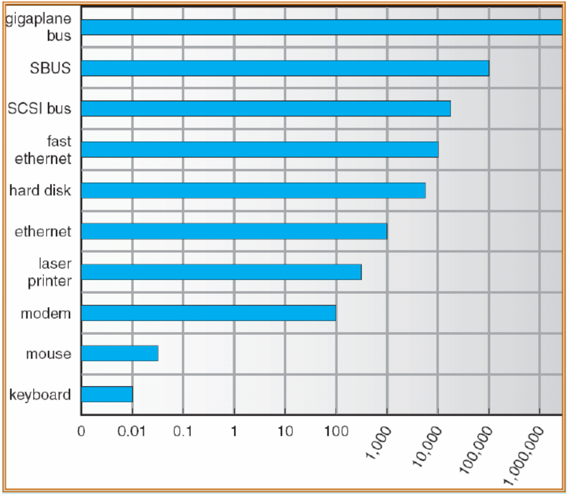
Buffering (缓冲机制)

- Buffering (缓冲机制)
  - Buffer — A memory area that stores data while they are transferred between two devices or between a device and an application
  - Store data in memory while transferring between devices
- Why buffering?
  2. To cope with device transfer **size** mismatch.
     Example: Send/receive a large message via network.
     - At sending side: the large message is fragmented into small network packets.
     - At receiving side: the network packets are placed in a reasembly buffer.

Buffering （缓冲机制）

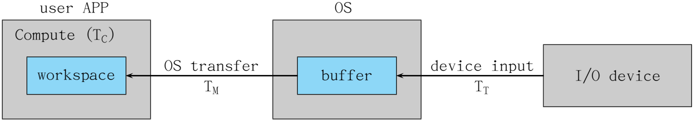
- Buffering （缓冲机制）
  - Buffer — A memory area that stores data while they are transferred between two devices or between a device and an application
  - Store data in memory while transferring between devices
- Why buffering?
  3. To maintain "copy semantics"
     Example: When write() data to disk, it first copy the data from application's buffer to a kernel buffer.

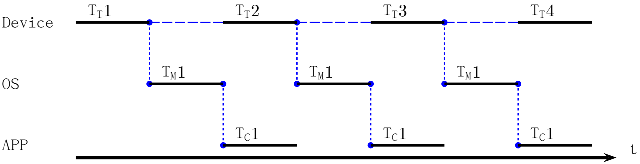
- Sun Enterprise 6000 Device-Transfer Rates



1. Single buffer (单缓冲)

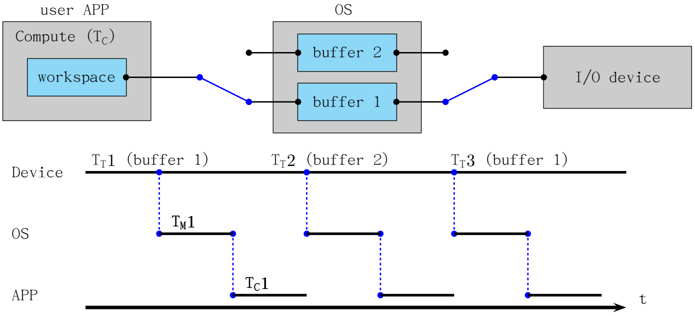
- APP.workspace $\xleftrightarrow{(OS,\ T_M)}$ OS.buffer $\xleftrightarrow{(Device,\ T_T)}$ Device
- Suppose the computing time of APP is T_C,
  if current T_C can parallel with the next T_T,
  we have $T_{average} = \max(T_C, T_T) + T_M$





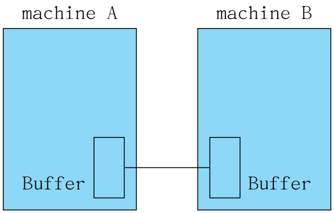
❷ Double buffer（双缓冲）

- $\approx \max(T_C, T_T)$；连续输入（$T_C < T_T$）或者连续计算（$T_C > T_T$）

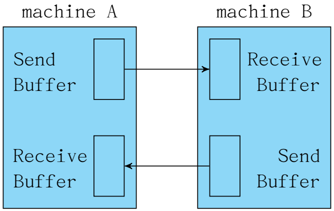


2 Double buffer (双缓冲)

- Another usage of single buffer and double buffers: in communication between two machines



single buffer



3. **Circular buffer（循环缓冲）**
   - Multiple (types of) buffers + multiple buffer pointers
     - Empty buffers and $Next_i$;
       Full buffers and $Next_g$;
       the current buffer in consumption
   - Similar to the PC problem.
4. **Buffer pool（缓冲池）**
   - 前三种，缓冲区是专用的
   - 为提高缓冲区利用率：设置公共的缓冲池

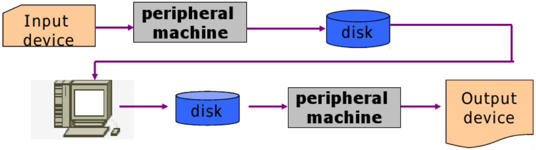
3 Kernel I/O Subsystem

- I/O Scheduling
- Buffering (缓冲机制)
- Caching, Spooling & device reservation
- Error Handling
- I/O Protection
- Kernel Data Structures

Caching, Spooling & device reservation

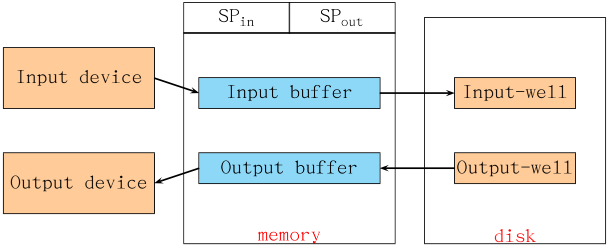
1. **Caching** - fast memory holding copy of data
   - Always just a copy
   - Key to performance
2. **Spooling** - hold output for a device
   - **Dedicated device** can serve only one request at a time
   - Spooling is a way of dealing with I/O devices in a multiprogramming system
   - Example: Printing
3. **Device reservation** - provides exclusive access to a device
   - System calls for allocation and deallocation
   - Watch out for deadlock

- Out-line I/O (脱机I/O), 使用外围机 (peripheral machine)



- SPOOL:
  Simultaneous Peripheral Operation On-Line
  (外部设备联机并行操作, 假脱机)
  - Dedicated device → sharable device
  - Using processes of multiprogramming system

- SPOOL:
  Simultaneous Peripheral Operation On-Line
  (外部设备联机并行操作，**假脱机**)
  - Structure
    - Input-well (输入井), output-well (输出井)
    - Input-buffer, output-buffer
    - Input-process SP_{in}, output-process SP_{out}
    - Requested-queue



Outline

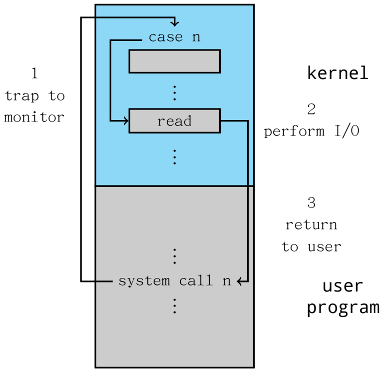
3 Kernel I/O Subsystem
- I/O Scheduling
- Buffering (缓冲机制)
- Caching, Spooling & device reservation
- Error Handling
- I/O Protection
- Kernel Data Structures

Error Handling

- OS can **recover** from disk read, device unavailable, transient write failures
  - Example: read() again, resend(), ..., according to some sepecified rules
- Most return an **error number** or code when I/O request fails
- **System error logs** hold problem reports

Outline

- User process may accidentally or purposefully attempt to disrupt normal operation via illegal I/O instructions
- To prevent users from performing illegal I/O
  - All I/O instructions defined to be privileged
  - I/O must be performed via system calls
    - Memory-mapped and I/O port memory locations must be protected too



Use of a System Call to Perform I/O

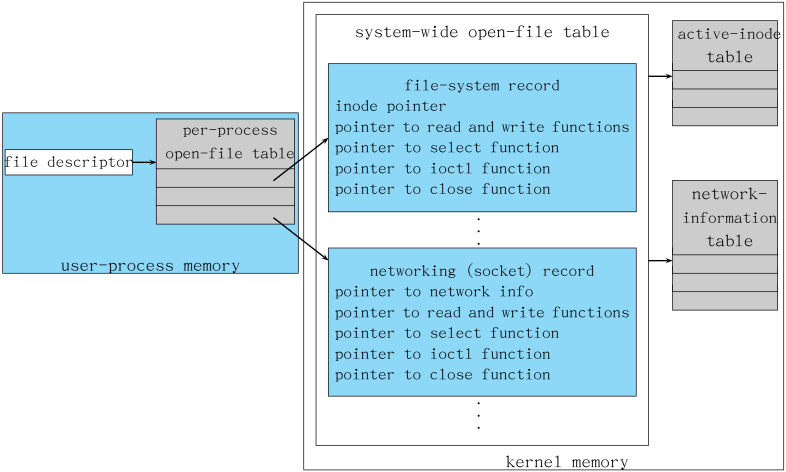
Outline

3 Kernel I/O Subsystem
- I/O Scheduling
- Buffering (缓冲机制)
- Caching, Spooling & device reservation
- Error Handling
- I/O Protection
- Kernel Data Structures

Kernel Data Structures

- Kernel **keeps state** info for I/O components, including
  - open file tables,
  - network connections,
  - character device state
- Many, many complex data structures to **track buffers, memory allocation, "dirty" blocks**
- Some use object-oriented methods and message passing to implement I/O

- Example: UNIX I/O Kernel Structure



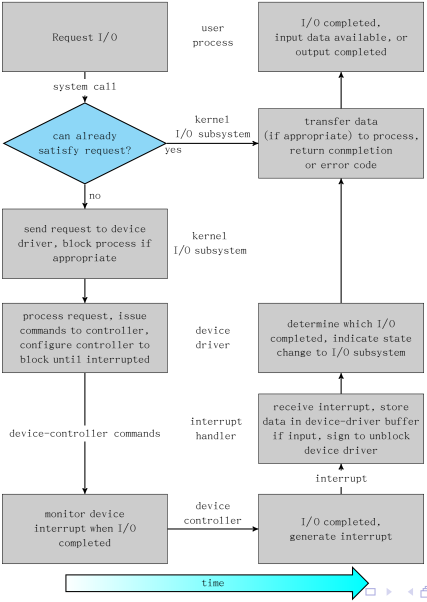
Outline

4 Transforming I/O Requests to Hardware Operations

I/O Requests to Hardware Operations

- Consider reading a file from disk for a process:
  1. Determine device holding file
  2. Translate name to device representation
  3. Physically read data from disk into buffer
  4. Make data available to requesting process
  5. Return control to process

The Typical Life Cycle of An I/O Request



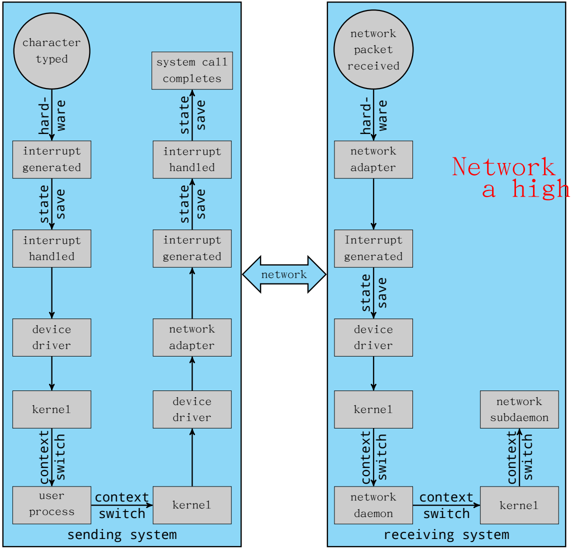
Outline

5 Performance

Performance

- I/O is a major factor in system performance:
  - Demands CPU to execute device driver, kernel I/O code
  - Context switches due to interrupts
  - Data copying
  - Network traffic especially stressful

Intercomputer Communications



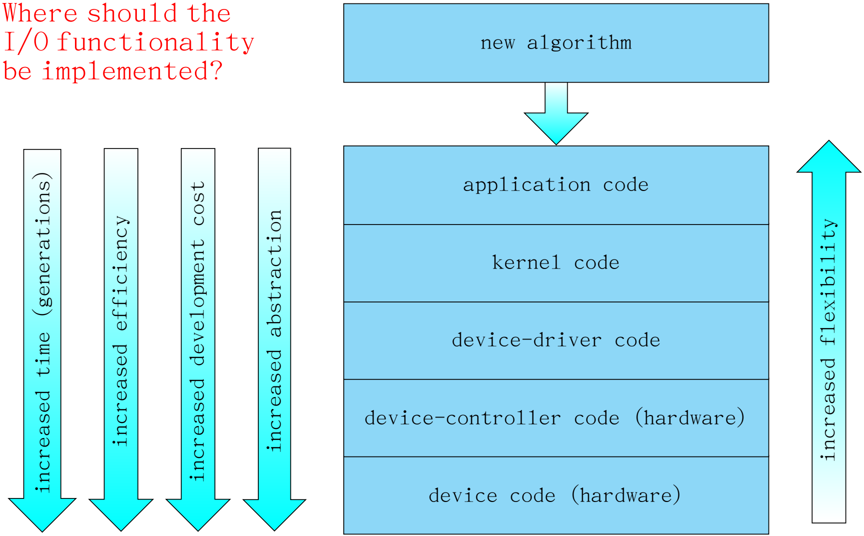
Network traffic can also cause a high context-switch rate

Improving Performance

1. Reduce number of context switches
2. Reduce data copying
3. Reduce interrupts by using large transfers, smart controllers, polling
4. Use DMA
5. Move processing primitives into hardware
6. Balance CPU, memory, bus, and I/O performance for highest throughput

Device-Functionality Progression

Where should the I/O functionality be implemented?



Outline

6 小结和作业

小结

1. I/O Hardware
   - Polling (轮询方式)
   - Interrupts (中断方式)
   - Direct Memory Access (DMA方式)
   - I/O hardware summary
2. Application I/O Interface
   - Block and Character Devices
   - Network Devices
   - Clocks and Timers
   - Blocking (阻塞) and Nonblocking (非阻塞) I/O
3. Kernel I/O Subsystem
   - I/O Scheduling
   - Buffering (缓冲机制)
   - Caching, Spooling & device reservation
   - Error Handling
   - I/O Protection
   - Kernel Data Structures
4. Transforming I/O Requests to Hardware Operations
5. Performance
6. 小结和作业

谢谢！