操作系统原理与设计

第一章绪论——CS structures

陈香兰

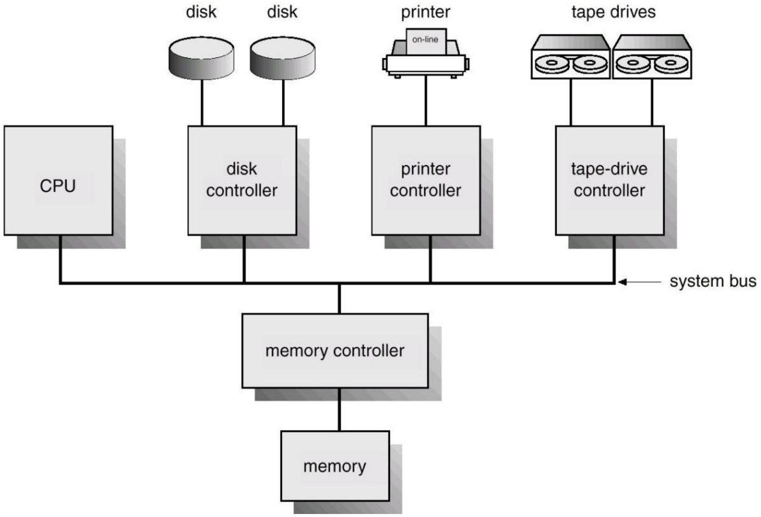
中国科学技术大学计算机学院

December 20, 2009

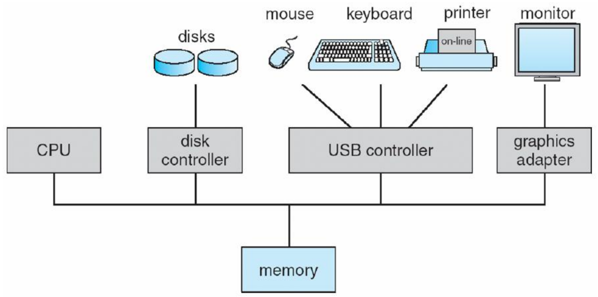
提纲

Outline

A modern computer system I



A modern computer system II



Outline

Start a computer system

- Bootstrap program, a initial program
  - Loaded at power-up or reboot
  - Typically stored in ROM or EPROM, generally known as firmware
  - initializes hardware
    - CPU registers, device controllers, memory content
  - Load at least a part of the OS into main memory & start executing it
- Platform dependent

Example: Linux system startup

Linux (Intel i386)

Refer to appendix A of 《Understanding Linux Kernel》

- →RESET pin of the CPU
- **cs:ip**= 0xFFFF FFF0
- **ROM BIOS**

Example: Linux system startup (cont.)

BIOS

Basic I/O System: A set of programs stored in ROM, including

- Several interrupt-driven low-level procedures
- A bootstrap procedure, who
  - POST (Power On Self-Test)
  - Initializes hardware device
  - Searches for an OS to boot
  - Copies the first sector of the OS into RAM 0x0000 7C00, and jumps & executes

??? After starts up

- Executes prearranged process, or
- Waits for interrupt

Outline

Interrupt I

Interrupt represents an event to be handled

For hardware: Device interrupt

- The completion of an I/O operation, a key stroke or a mouse move, ⋯

For error (also hardware): exception

- Trap for debug
- Fault, example: page fault, division by zero, invalid memory access
- Abort, a serious error

For software: System call

Interrupt II

- To request for some operating-system service
  - Int 0x80, 0x13, 0x21

Modern OSs are interrupt-driven

Interrupt handling I

When the CPU is interrupted

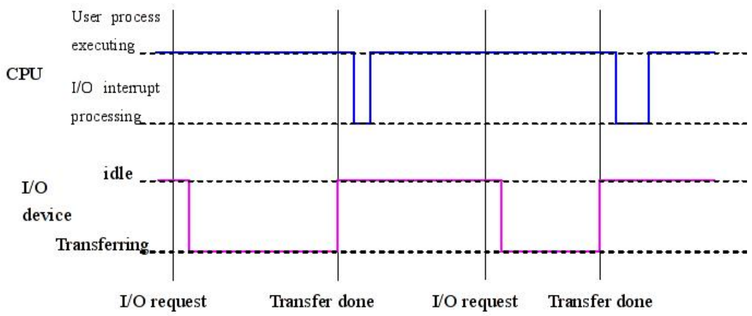
- Stops what it is doing
- Incoming interrupts are disabled to prevent a lost interrupt
- Transfers control to the ISR (**Interrupt Service Routine**)
  - A generic routine in fixed location and then call the interrupt-specific handler
    - **Polling**
    - **interrupt vector table**

When the ISR completed,
Back to interrupted program

Interrupt handling II

- HOW ?
  —— OS **preserves the state of the CPU** by storing **registers and the program counter**
  - Old: **Fixed location**, or a location indexed by the device number
  - Recent: system stack

Interrupt time line for a single process doing output

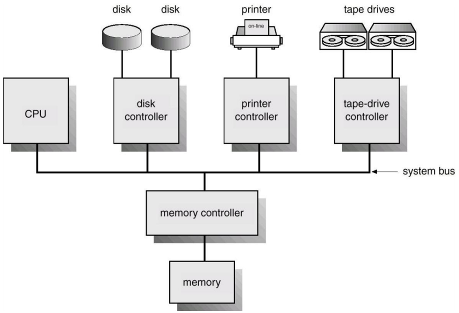


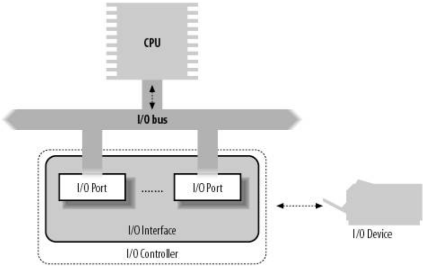
Example: interrupts in I386

- protect mode （保护模式）
  - IDT （**Interrupt Descriptor** Table)
  - OS填写IDT表，包括每个中断处理例程的入口地址等信息
  - 中断发生的时候，CPU根据从中断控制器获得的中断向量号在IDT表中索引到对应的中断处理例程（入口地址），并跳转过去运行
    - 保存上下文
    - 处理中断
    - 恢复上下文

Outline

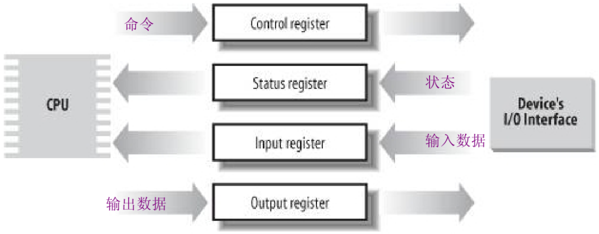
I/O structure





I/O structure

- Each **device controller** is in charge of a particular device type
- Each device controller has
  - a local buffer & a set of special-purpose registers
- Data transfer, two phrase
  - Main memory $\longleftarrow$(CPU)$\longrightarrow$ local buffer of controller
  - device $\longleftarrow$(device controller)$\longrightarrow$ local buffer
- I/O devices & CPU can execute **concurrently**
  - Share/compete memory cycle
  - Memory controller


命令
Control register
CPU
Status register
状态
Device's I/O Interface
Input register
输入数据
输出数据
Output register

Outline

I/O operation

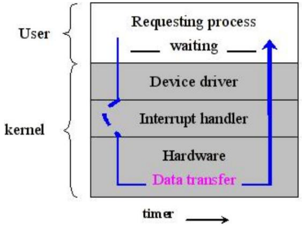
- CPU start an I/O operation by
  - Loading the appropriate registers within the device controller
  - When complete, device controller informs CPU by
    - Triggering an interrupt, or
    - Simply set a flag in one of their registers
- Two I/O methods
  - synchronous VS. asynchronous

I/O method —— analysis

Synchronous

- Waiting
  - Wait instruction
  - Dead loop like

    Loop: jmp Loop

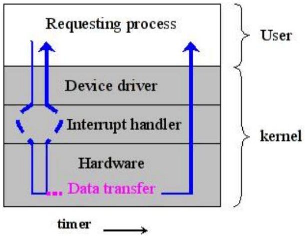


- At most one I/O request is outstanding at a time
  - ???
  - **Advantage**: always knows exactly which device is interrupting
  - **Disadvantage**: excludes concurrent I/O operations & the possibility of overlapping useful computation with I/O

I/O method —— analysis (cont.)

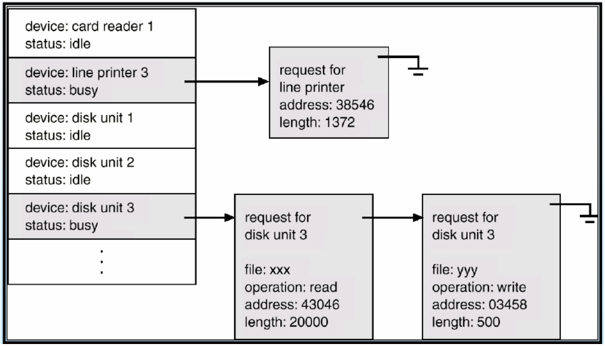
Asynchronous

- Start & cont.
  - with a wait system call



- **Need to keep track of many I/O request**
  - **Device-status table**
    - Each entry: Device type, address, state
  - **A wait queue** for each device
  - When an interrupt occurs, OS indexes into I/O device table to determine device status and to modify table entry to reflect the occurrence of interrupt
  - Main advantage: **system efficiency**↑

device status table



Outline

Direct Memory Access (DMA)

Example1: 9600-baud terminal

- 2us(ISR) per 1000us
- It's ok!

Example2: hard disk

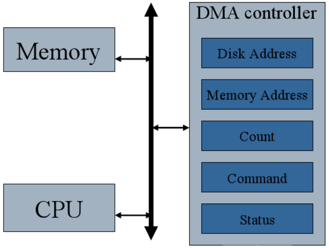
- 2us(ISR) per 4us
- The overhead (per byte) is relatively costly!

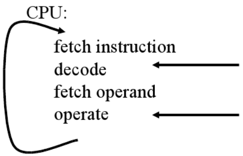
DMA (Direct Memory Access)

- Used for high-speed I/O devices able to transmit information at close to memory speeds.

DMA structure

One interrupt / block of data





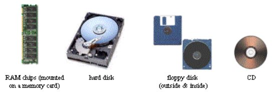
Device controller

- transfers between buffer and main memory directly, without CPU intervention.
- **Memory cycle stealing**

Outline

Storage structure

- **Von Neumann architecture** VS. **Harvard architecture**
  - Separated data & code in different memory???
- Main memory (RAM) is the only large storage media that the CPU can access directly
  - Small, Volatile
- Secondary storage is an extension of main memory that provides large nonvolatile storage capacity
  - Magnetic disk
  - Optical disk
  - Magnetic tape

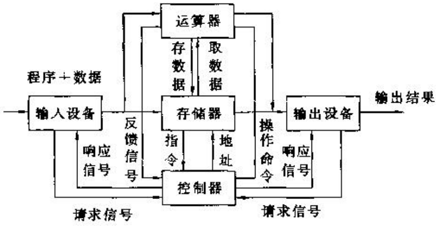

RAM chips (mounted on a memory card)
hard disk
floppy disk (outside & inside)
CD

Von Neumann architecture

- 计算机
  - 不可编程的，强定制，高效
  - 可编程的，灵活
    - 提供指令集，程序就是一个指令序列

冯诺伊曼体系结构

- 五大部件：运算器、控制器、存储器、I/O设备
- 存储器与CPU相分离
- 指令存储与数据存储共享存储器



Storage structure (cont.)

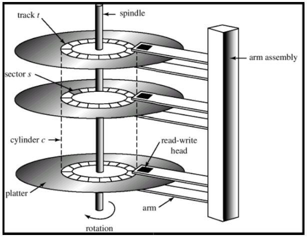
Memory VS. register

- **Same**: **Access directly for CPU**
  - Register name
  - Memory address
- **Different: access speed**
  - Register, one cycle of the CPU clock
  - Memory, Many cycles (2 or more)
- **Disadvantage**:
  - CPU needs to stall frequently & this is intolerable
- **Remedy**
  - **cache**

Magnetic disks

- Magnetic disks – rigid metal or glass platters covered with magnetic recording material
  - Disk surface is logically divided into **tracks**, which are subdivided into **sectors**.
  - The **disk controller** determines the logical interaction between the device and the computer.

- Position time
- Transfer time



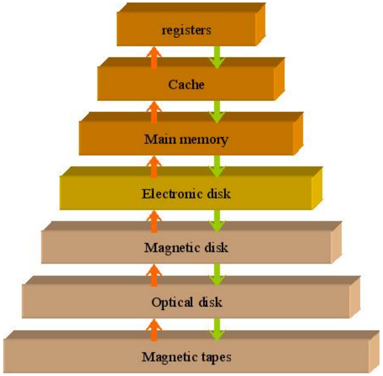
- Transfer time T_T
  - $T_T \approx \mathit{data\,size} \times \mathit{Transfer\,rate}$
  - $\mathit{Transfer\,rate} \approx (n\,M/s)^{-1}$
    $\approx (n\,Byte/us)^{-1}$
    $\approx 1/n\,us/Byte$
- Positioning time T_p
  - Seek time T_s
  - Rotational latency T_R
  - $T_p \approx T_s + T_R \approx m\,ms$
- T_T VS. T_p
  - Please **Store data closely**

Outline

Storage hierarchy

Storage hierarchy

- Storage systems in a computer system can be organized in a hierarchy
  - Speed, access time
  - Cost per bit
  - Volatility



Caching

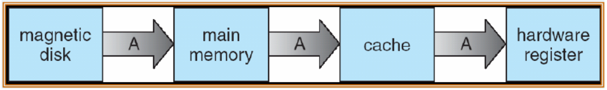
- **Caching** (高速缓存技术）
  - Copying information into faster storage system
  - When accessing, first check in the **cache**,
    - if **In**: use it directly
    - **Not in**: get from upper storage system, and leave a copy in the cache
- Using of caching
  - Registers provide a high-speed cache for main memory
  - **Instruction cache & data cache**
  - Main memory can be viewed as a fast cache for secondary storage
  - ...

Cache management

- **Design problem**
  - Hardware or software?
  - Cache size & Replacement policy is important
  - Hit rate 80%˜99% is OK!

Coherency and consistency

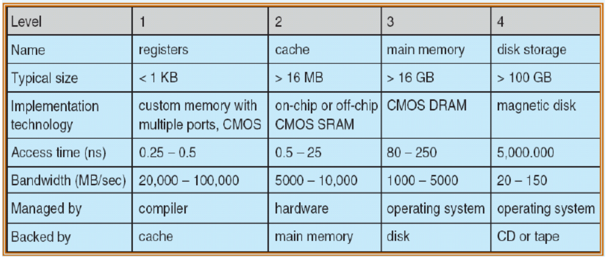
- Multitasking environments must be careful to use most recent value, no matter where it is stored in the storage hierarchy
- Migration of Integer A from Disk to Register



- **The same data may appear in different level of the storage system**
- When
  - Simple batch system, no problem
  - Multitasking, always obtain the most recently updated value
  - Multiprocessor, cache coherency (always implicit to OS)
  - Distributed system?

Performance of Various Levels of Storage

- Movement between levels of storage hierarchy can be explicit or implicit



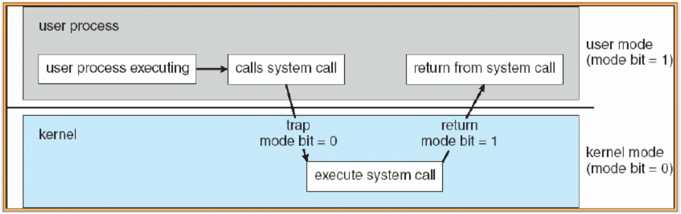
Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

Hardware protection

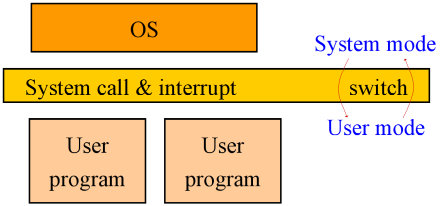
- A properly designed OS must ensure that an incorrect (or malicious) program cannot cause other programs to execute incorrectly.
    - When in dead loop
    - When sharing recourses
    - When one erroneous program might modify the program or data of another program, or even the OS
- Hardware must provide protection
    - **Dual-Mode Operation**
    - **I/O protection**
    - **Memory protection**
    - **CPU protection**

Dual-Mode Operation

- Using **mode bit** to provide different modes of execution
  - **User mode:** execution done on behalf of user
  - **privileged mode**: execution done on behalf of OS
    - Privileged instructions
- User program VS. OS (or Kernel)
  - Switch between user mode (1) and privileged mode(0)
    - Boot
    - Interrupt (include system call)



- Example: i386
  - 4 modes (2 mode bits)
  - Linux uses 2 mode (00b & 11b)

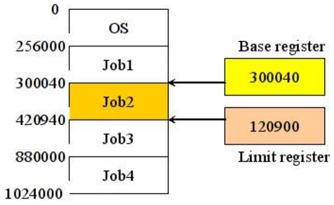


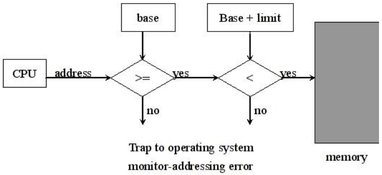
I/O protection

- Preventing the users from issuing illegal I/O instructions
- **All I/O instructions are privileged instructions**
  - instead of performing I/O operation directly, **user program must make a system call**
  - OS, executing in monitor mode, checks validity of request and does the I/O
  - input is returned to the program by the OS
- Smart hacker may···
  - Stores in the interrupt vector a new address, which points to a malicious routine
  - The I/O protection is compromised
  - We need some more protection···

Memory protection

- At least for interrupt vector and the ISR
- **Base register protection scheme**
  - Base register+Limit register
  - Memory outside is protected
  - OS has unrestricted access to both monitor and user’ s memory
  - Load instructions for the base/limit registers are privileged





CPU protection

- OS should be always take control of everything
  - **What if a user program is in dead loop?**
- Timer
  - Interrupts computer after specified period
  - Periodically or one-shot
  - Load-timer is also a privileged instruction
- Usage
  - Time sharing
  - Compute current time
  - Alarm or timer

Timer to prevent infinite loop / process hogging resources

- Set interrupt after specific period
- Operating system decrements counter
- When counter zero generate an interrupt
- Set up before scheduling process to regain control or terminate program that exceeds allotted time

Outline

General system architecture

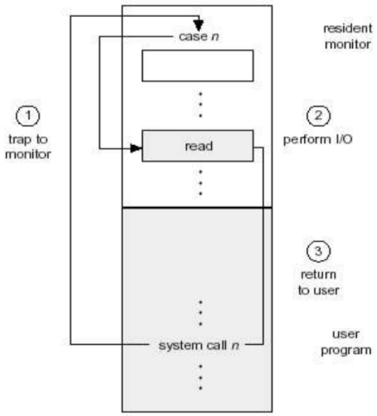
- multiprogramming
- time sharing
- OS: in kernel (privileged) mode
    - control hardware & software resource
    - execute privileged instruction
    - system call

system call

System call—like a common function call, but totally different!

- **Trap** to a specific location in interrupt vector
  - **int** (i386)
  - **trap** (SUN SPARC)
  - **syscall** (MIPS R2000)
- Control passes to **a service routine** in the OS, and the mode bit is set to **monitor mode**
- The kernel
  - Verifies that the parameters are correct and legal
  - Executes the request
  - Returns control to the instruction following the system call

Use of a system all to perform I/O

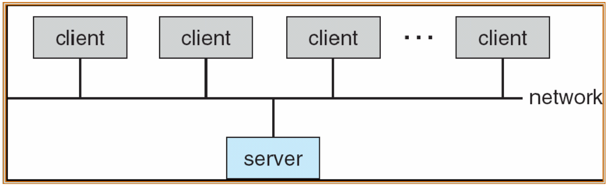


Computing Environments

一、Traditional computer

- 随计算机的发展而变化
- Office environment
  - PCs connected to a network, terminals attached to mainframe or minicomputers providing batch and timesharing
  - Now portals allowing networked and remote systems access to same resources
- Home networks
  - Used to be single system, then modems
  - Now firewalled, networked

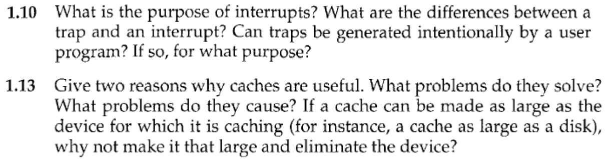
- Client-Server Computing
  - Dumb terminals supplanted by smart PCs
  - Many systems now **servers**, responding to requests generated by **clients**
    - **Compute-server** provides an interface to client to request services (i.e. database)
    - **File-server** provides interface for clients to store and retrieve files



- 其他
    - Peer-to-Peer Computing
    - Web-Based Computing
    - Grid Computing
    - Cloud Computing
    - 普适计算Pervasive/Ubiquitous Computing

小结

作业



1.10 What is the purpose of interrupts? What are the differences between a trap and an interrupt? Can traps be generated intentionally by a user program? If so, for what purpose?

1.13 Give two reasons why caches are useful. What problems do they solve? What problems do they cause? If a cache can be made as large as the device for which it is caching (for instance, a cache as large as a disk), why not make it that large and eliminate the device?

- 1.8
  Which of the following instructions should be privileged?
  - a. Set value of timer.
  - b. Read the clock.
  - c. Clear memory.
  - d. Issue a trap instruction.
  - e. Turn off interrupts.
  - f. Modify entries in device-status table.
  - g. Switch from user to kernel mode.
  - h. Access I/O device.

华夏班额外的作业

1.11 Direct memory access is used for high-speed I/O devices in order to avoid increasing the CPU's execution load.

a. How does the CPU interface with the device to coordinate the transfer?

b. How does the CPU know when the memory operations are complete?

c. The CPU is allowed to execute other programs while the DMA controller is transferring data. Does this process interfere with the execution of the user programs? If so, describe what forms of interference are caused.

谢谢！

下次课交作业