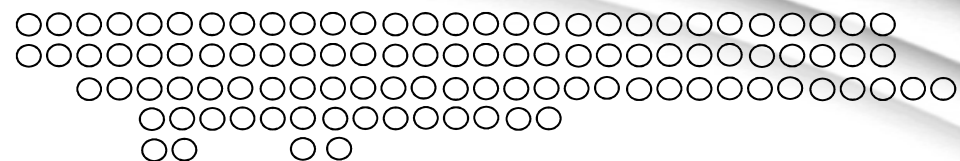


《操作系统原理》



第2章 操作系统逻辑结构

教师：苏曙光

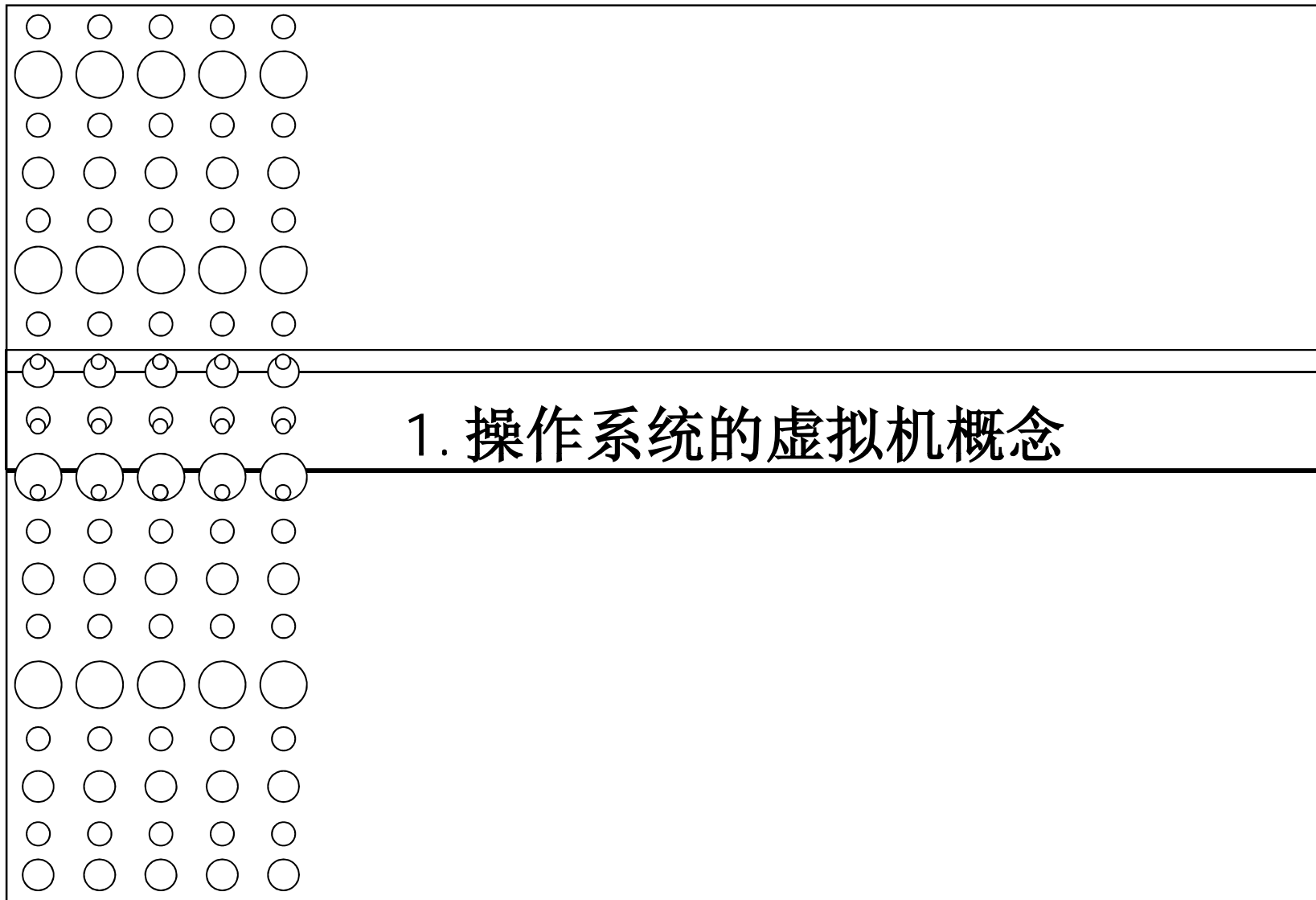
华中科技大学软件学院

I 主要内容

- n 操作系统的虚拟机概念
- n 操作系统的逻辑结构
- n 操作系统依赖的基本硬件环境

I 重点

- n 虚拟机的概念
- n 态的概念
- n 中断机制



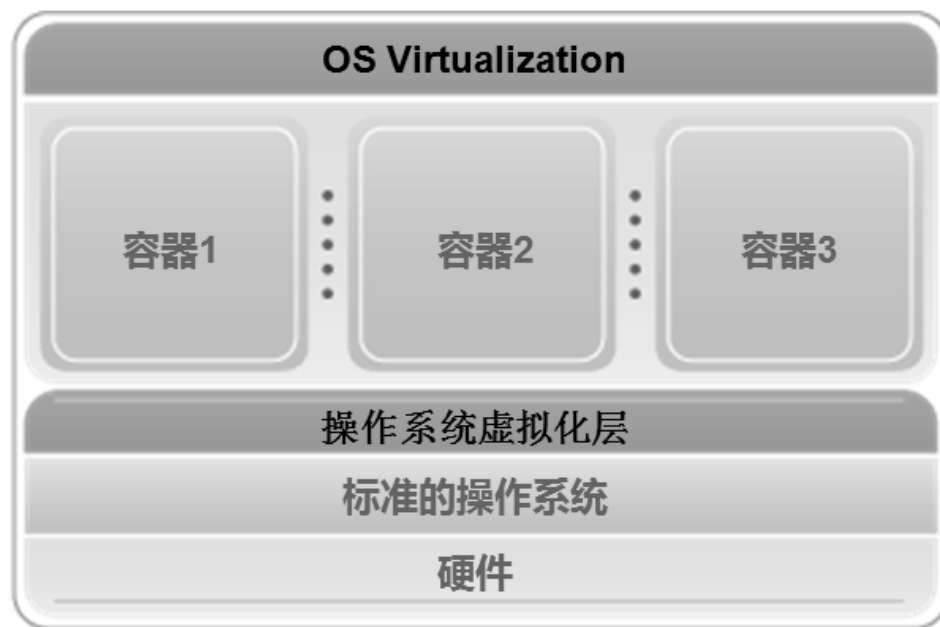
操作系统“虚拟计算机”的概念

I 面对用户，操作系统可以称为虚拟计算机

- n 用户界面
- n 屏蔽硬件细节
- n 扩展硬件功能
- n 系统更安全
- n 系统更可靠
- n 效率更高



操作系统虚拟化技术



操作系统虚拟化(Containers)

- 虚拟操作系统访问
- 创建多个虚拟OS实例
- 物理服务器拥有1个标准OS内核
- **Parallels Virtuozzo Containers, Sun Solaris Containers, OpenVZ**

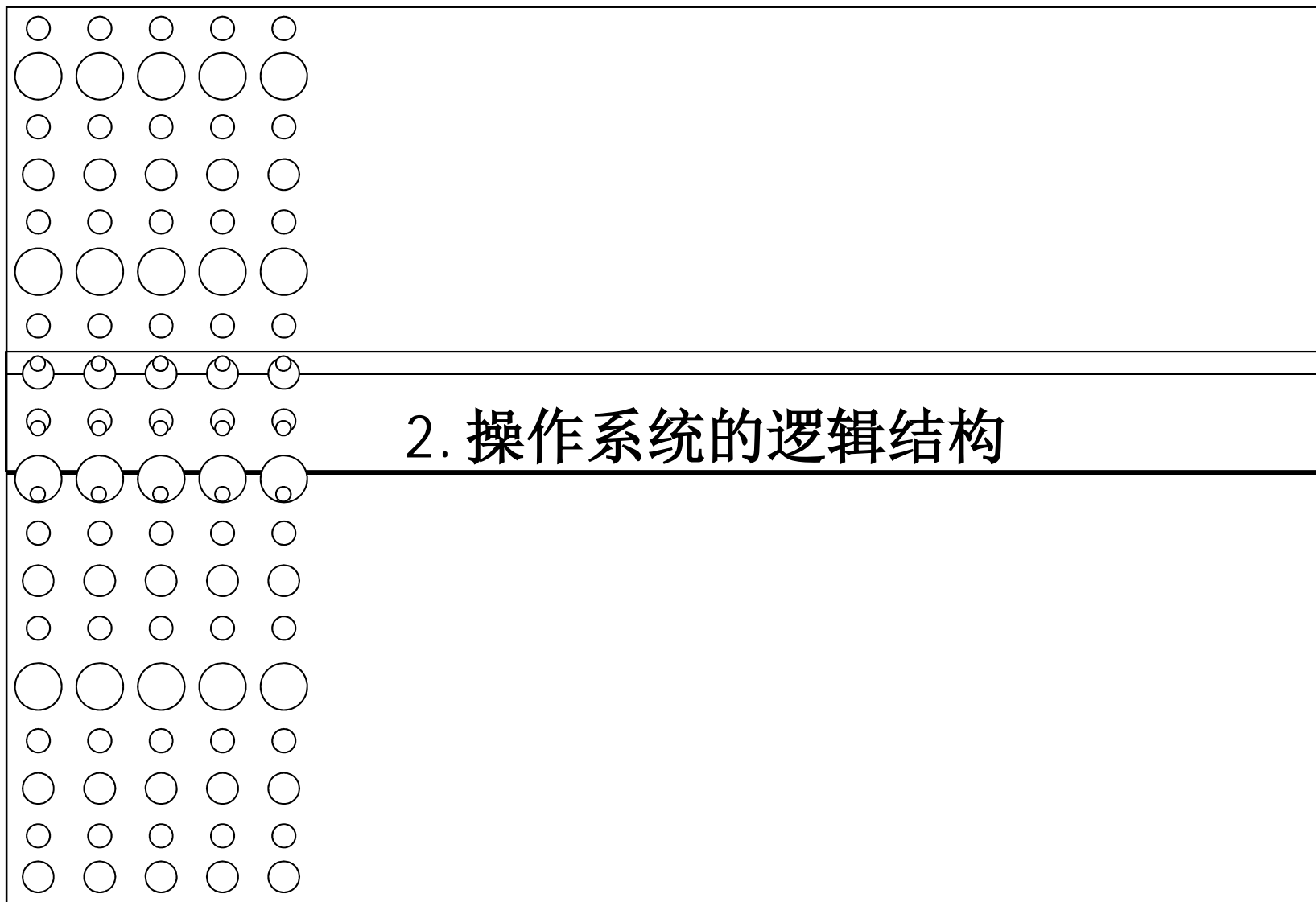
Ⅰ 虚拟化的结果：一台服务器当N台服务器来使用

操作系统虚拟化技术



硬件虚拟化(Hypervisors)

- 虚拟硬件访问
- 创建多个虚拟硬件实例
- 宿主OS及每个GuestOS为完整OS
- Parallels Server, VMware ESX



操作系统的逻辑结构

I 逻辑结构

nOS的设计和实现思路

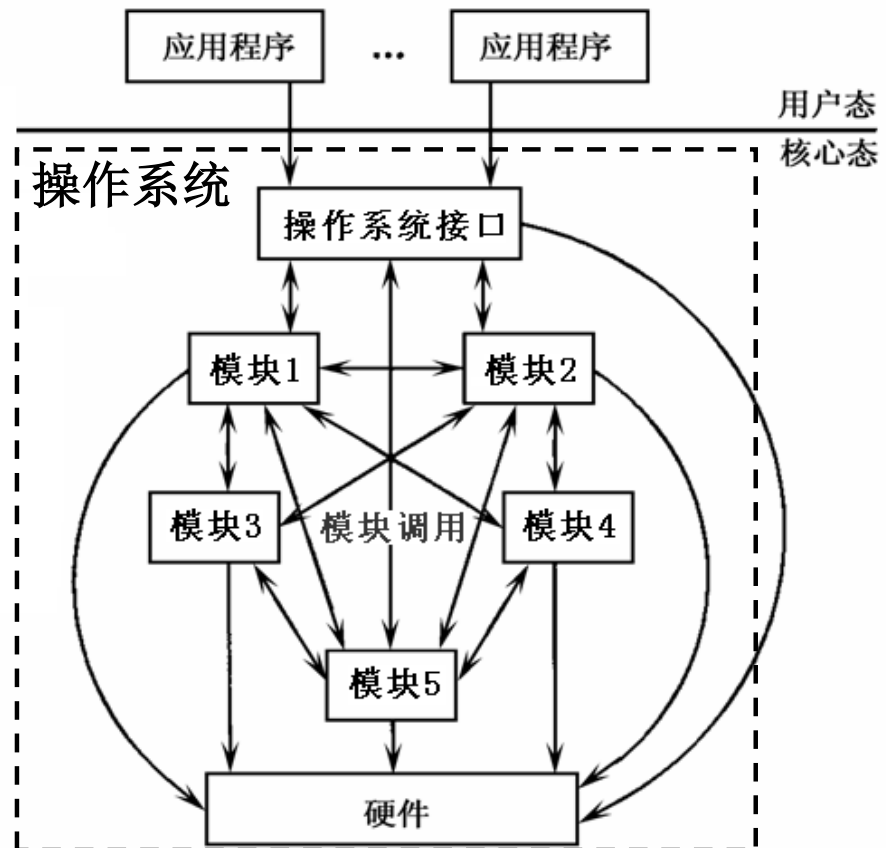
I 逻辑结构的种类

n1.整体式结构

n2.层次式结构

n3.微内核结构（客户/服务器结构，Client / Server）

I 1.整体式结构



I 特点

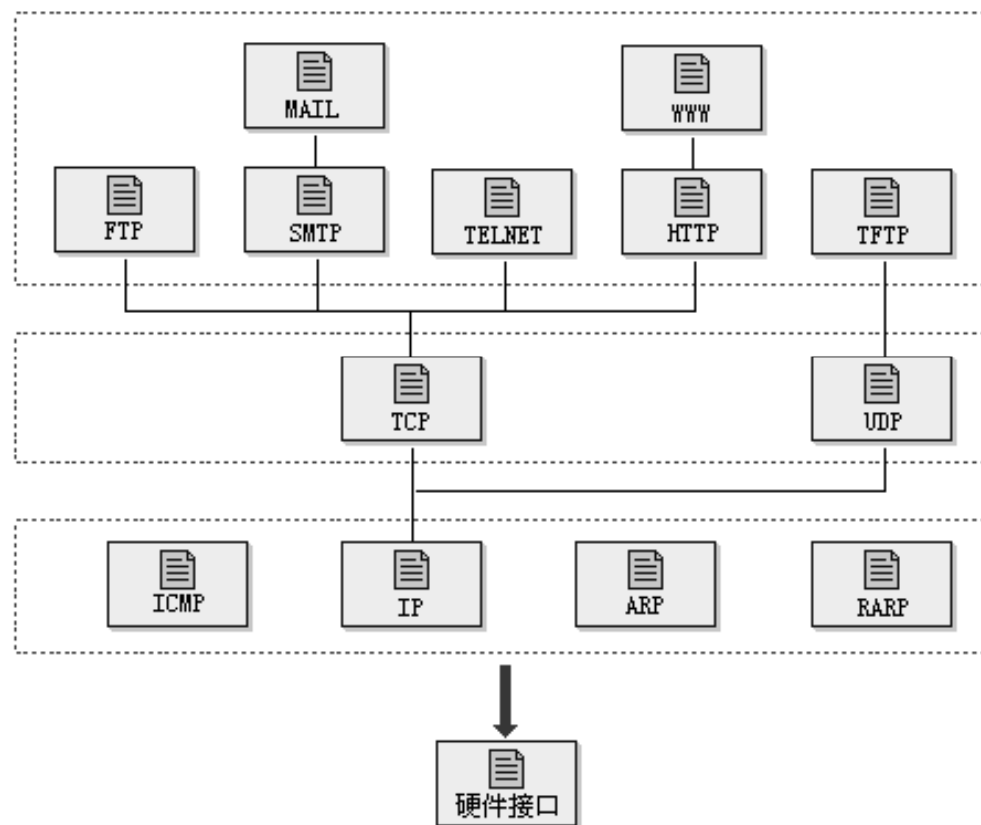
- n 模块设计、编码和调试独立
- n 模块调用自由
- n 模块通信多以全局变量形式完成

I 缺点

- n 信息传递随意，维护和更新困难。

2.层次结构

I 层次结构的软件例子：TCP/IP协议栈



I 层次结构

n把所有功能模块按照调用次序分别排成若干层，确保各层之间只能是单向依赖或单向调用。

p 分层原则

n硬件相关——最底层

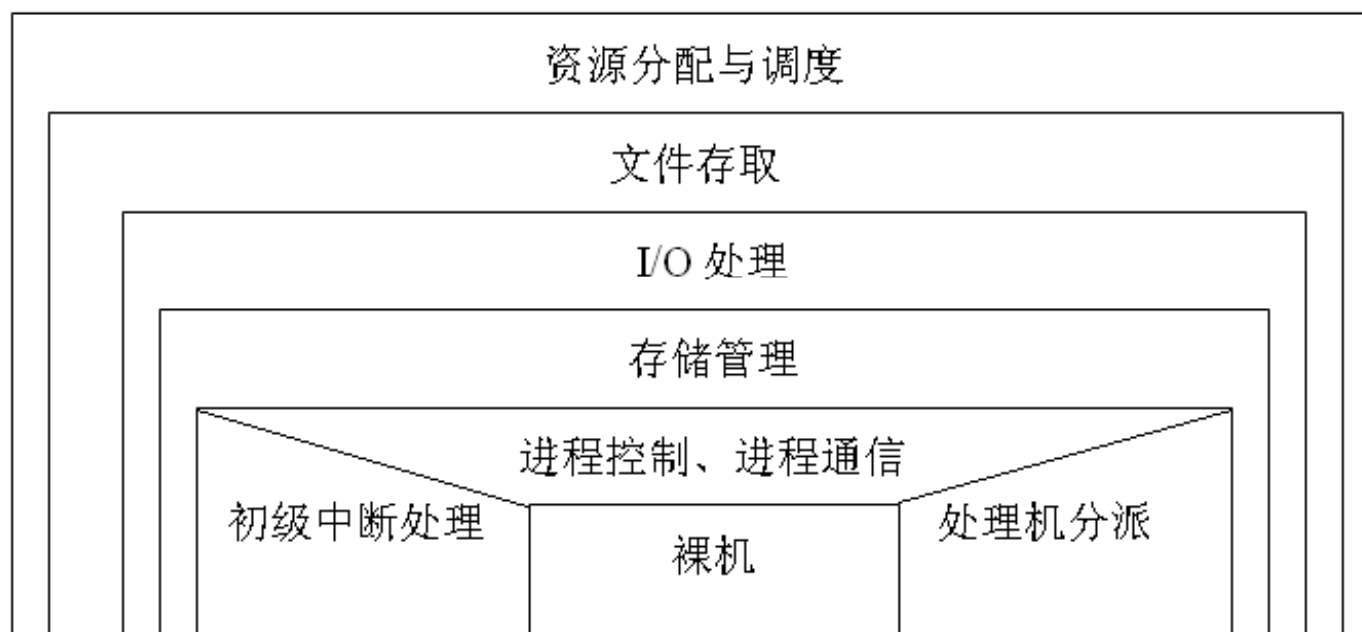
n外部特性——最外层

n中间层——调用次序或消息传递顺序

n共性的服务——较低层

n活跃功能——较低层

I 分层逻辑结构的OS实例



I 层次结构的优点

- n 结构清晰，避免循环调用。
- n 整体问题局部化，系统的正确性容易保证。
- n 有利于操作系统的维护、扩充、移植。

Linux内核结构

I Linux中的调用层次

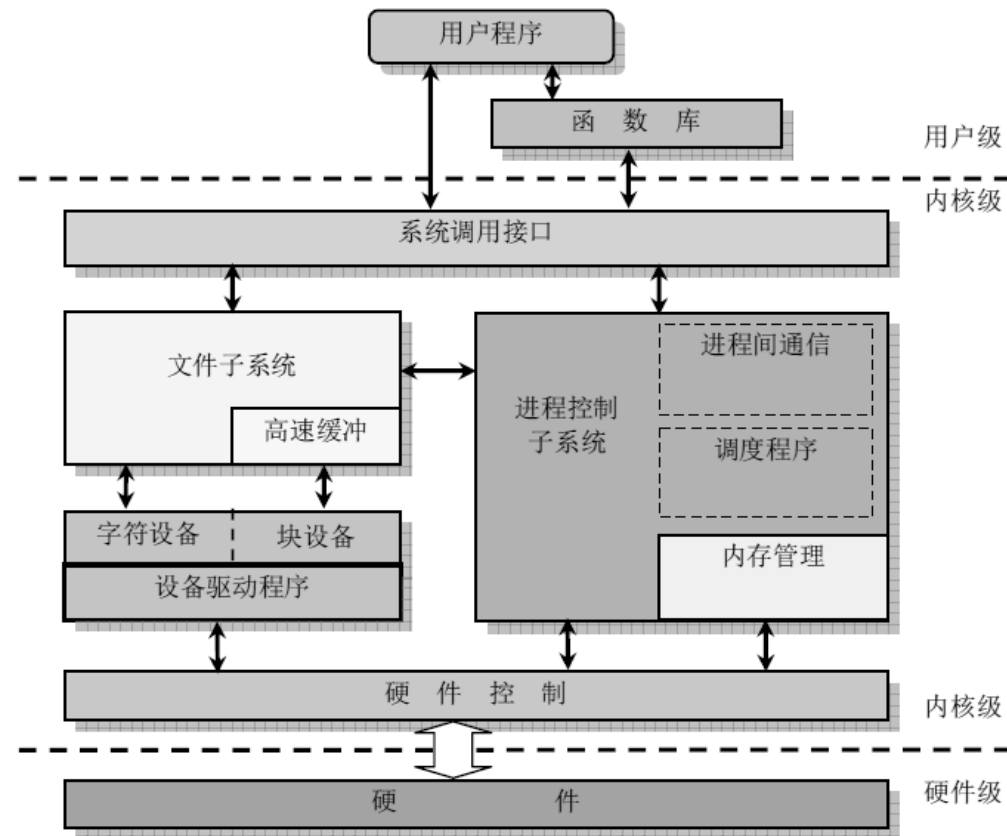
- n应用层：应用程序使用指定的参数值执行系统调用指令(int x80)，使CPU从用户态切换到核心态（调用服务的应用层）
- n服务层：OS根据具体的参数值调用特定的服务程序（执行系统调用的服务层）
- n底层：服务程序根据需要调用底层的支持函数（支持系统调用的底层函数）

I 结论：

- n层次式（具有整体式特点）
- n单体内核（即内核文件一个）

Linux内核结构

I Linux = 内核+Shell+X Window+应用程序



3. 微内核结构 (客户/服务器结构, Client/Server)

l 客户：应用程序

l 操作系统 = 微内核+核外服务器

n 微内核

u 足够小，提供**OS**最基本的核心功能和服务

u ① 实现与硬件紧密相关的处理

u ② 实现一些较基本的功能；

u ③ 负责客户和服务器间的通信。

n 核外服务器

u 完成**OS**的绝大部分功能，等待客户提出请求。

u 由若干服务器或进程共同构成

p 例如：进程/线程服务器，虚存服务器，设备管理服务器等，以**进程形式**运行在用户态。

I 微内核和单体内核的比较

	实质	优点	缺点	代表
单体内核	将图形、设备驱动及文件系统等功能全部在内核中实现,和内核运行在同一地址空间。	减少进程间通信和状态切换的系统开销,获得较高的运行效率。	<ul style="list-style-type: none">●内核庞大,占用资源较多且不易剪裁。●系统的稳定性和安全性不好。	UNIX Linux
微内核	只实现OS基本功能,将图形、文件系统、设备驱动及通信功能放在内核之外。	<ul style="list-style-type: none">●内核精练,便于剪裁和移植。●系统服务程序运行在用户地址空间,系统的稳定性和安全性较高。	用户状态和内核状态需要频繁切换,从而导致系统效率不如单体内核。	Minix WinCE

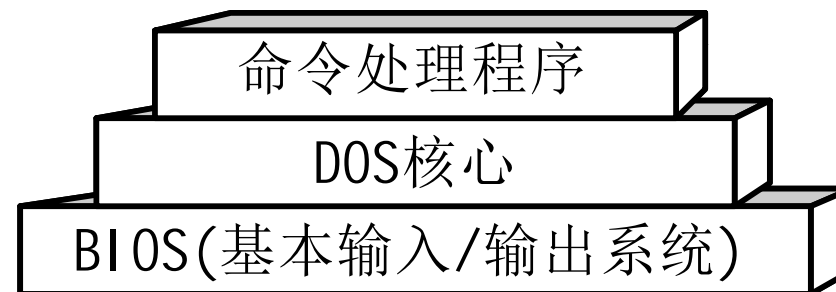
典型操作系统的结构

I MS DOS

nBIOS: Basic Input/Output System

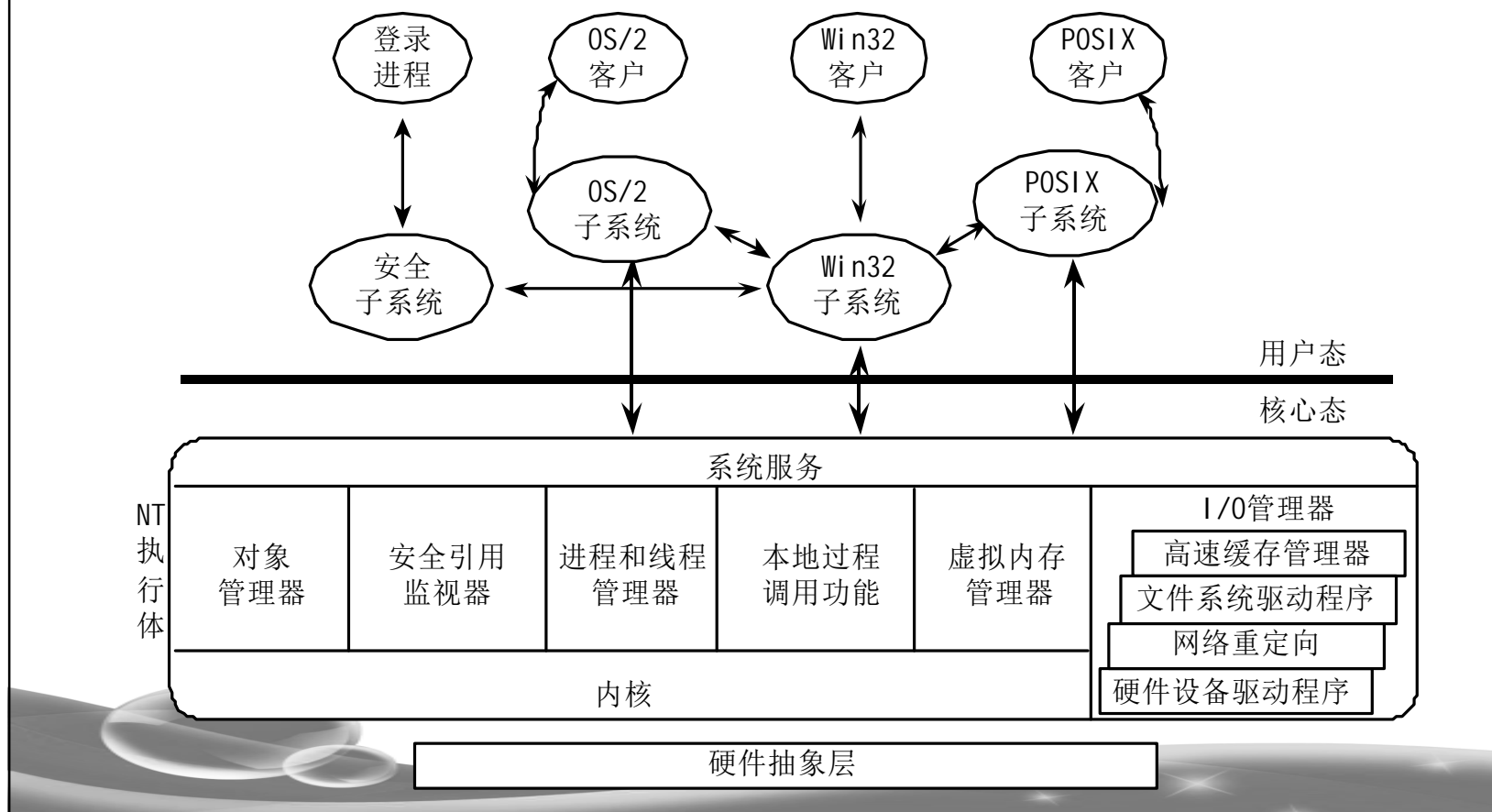
nDOS核心: 内存、文件管理、字符设备和输入/输出

n命令处理程序: 对用户命令进行分析和执行

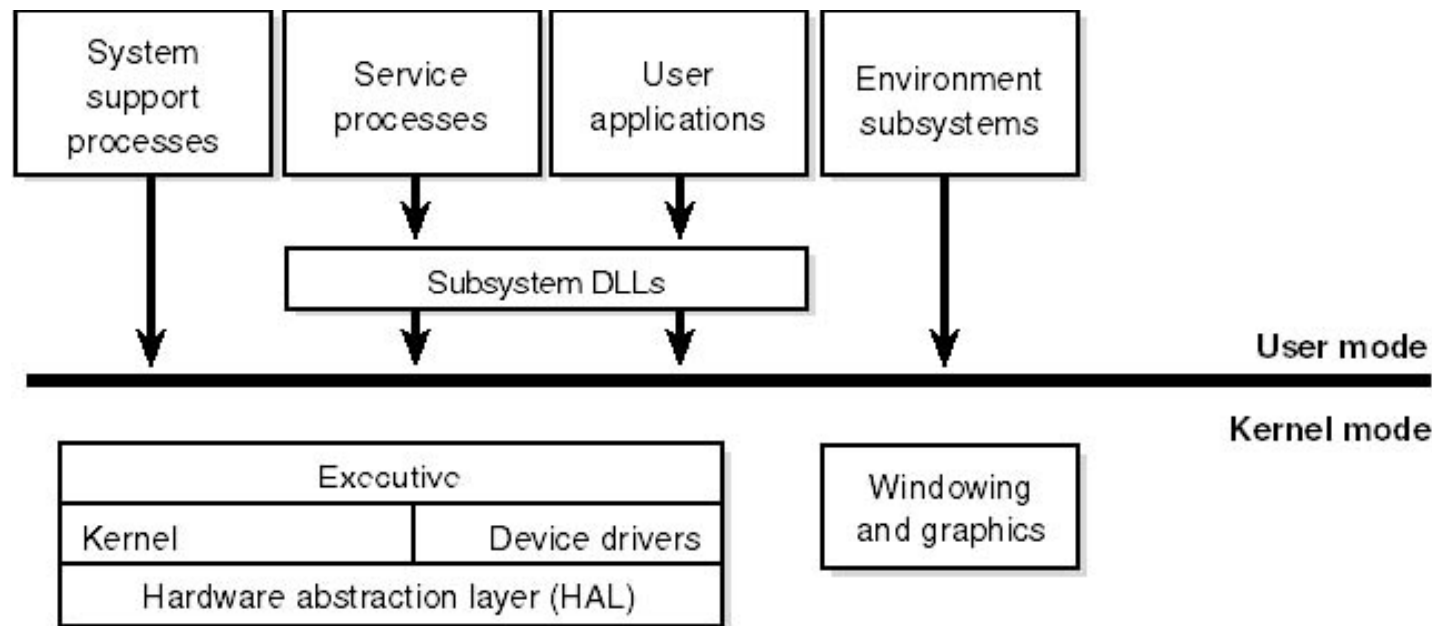


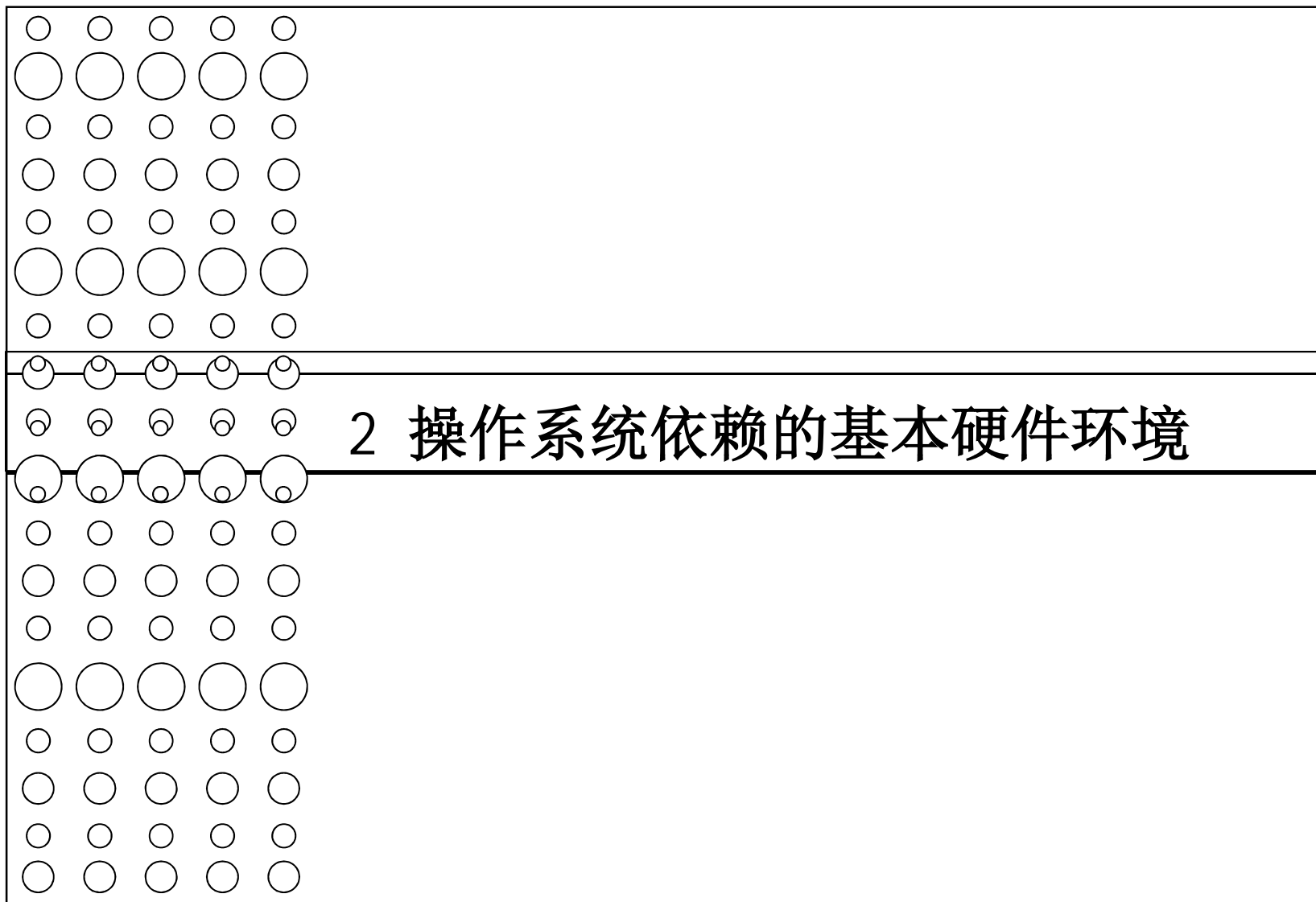
典型操作系统的结构

I Windows NT



I Windows 2000





I 支持操作系统的最基本硬件结构

nCPU

n内存

n中断

n时钟

CPU

I CPU态 (Mode)

- n CPU的工作状态。
- n 对资源和指令使用权限的描述

I 态的分类

- n 核态(Kernel mode):
 - u 能够访问所有资源和执行所有指令
 - u 管理程序/OS内核
- n 用户态 (User mode, 目态):
 - u 仅能访问部分资源，其它资源受限。
 - u 用户程序
- n 管态(Supervisor mode)
 - u 介于核态和用户态之间

I 硬件和OS对CPU的观察

n 硬件按“态”来区分CPU的状态

n OS按“进程”来区分CPU的状态

u A,B,C,D: 四个进程

u K: Kernel mode, 核心态

u U: User mode, 用户态

进程	A	B	C	D
核心态	K			K
用户态		U	U	

Intel CPU和Windows下的态

- I Intel CPU: Ring 0 ~ Ring 3 (Ring 0 最核心, Ring 3最外层)
- I Windows OS: 仅支持Ring 0和Ring 3
 - n Ring 0: 特权指令, OS内核或驱动程序
 - n Ring 3: 应用程序
- I 通信方式: **DeviceIoControl (kernel32.dll)**
 BOOL DeviceIoControl(
 HANDLE hDevice, // 设备句柄 //CreateFile打开创建
 DWORD dwIoControlCode, // 控制码//指明需要内核完成的操作类型
 LPVOID lpInBuffer, // 输入数据缓冲区 //Ring3输入
 DWORD nInBufferSize, // 缓冲区长度 //Ring3输入
 LPVOID lpOutBuffer, // 输出数据缓冲区 //Ring0返回
 DWORD nOutBufferSize, // 缓冲区长度 //Ring0返回
 LPDWORD lpBytesReturned, // 输出数据实际长度
 LPOVERLAPPED lpOverlapped // 重叠操作结构指针
);

DeviceIoControl的例子

//应用程序试图去得到磁盘设备分区状况：柱数，磁道数，扇区数,字节数等。

```
int main(int argc, char *argv[])
{
    DISK_GEOMETRY pdg;           // disk drive geometry structure
    BOOL bResult;                // generic results flag
    ULONGLONG DiskSize;          // size of the drive, in bytes
    bResult = GetDriveGeometry (&pdg);
    if (bResult)
    {
        printf("Cylinders = %l64d\n", pdg.Cylinders);
        printf("Tracks/cylinder = %ld\n", (ULONG) pdg.TracksPerCylinder);
        printf("Sectors/track = %ld\n", (ULONG) pdg.SectorsPerTrack);
        printf("Bytes/sector = %ld\n", (ULONG) pdg.BytesPerSector);
    }
    return ((int)bResult);
}
```

DeviceIoControl的例子

//IOCTL_DISK_GET_DRIVE_GEOMETRY: 操作代码: 获取柱数, 磁道数, 扇区数, 字节数等。

```
BOOL GetDriveGeometry ( DISK_GEOMETRY *pdg)
{
    HANDLE hDevice;           // handle to the drive to be examined
    BOOL bResult;           // results flag
    DWORD junk;             // discard results
    hDevice = CreateFile("\\\\.\\PhysicalDrive0", // 通过物理设备名的方式打开设备。
        0, // no access to the drive
        FILE_SHARE_READ | FILE_SHARE_WRITE,
        NULL, OPEN_EXISTING, 0, // file attributes
        NULL); // do not copy file attributes

    bResult = DeviceIoControl( hDevice, // device to be queried
        IOCTL_DISK_GET_DRIVE_GEOMETRY, // operation to perform
        NULL, 0, // no input buffer
        pdg, sizeof(*pdg), // output buffer
        &junk, // # bytes returned
        (LPOVERLAPPED) NULL); // synchronous I/O
}
```

//驱动程序：得到一个磁盘设备分区状况

```
NTSTATUS DriverEntry(  
    IN PDRIVER_OBJECT DriverObject, // 指向一个刚被初始化的驱动程序对象  
    IN PUNICODE_STRING RegistryPath )  
{  
    DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] =  
        DispatchDeviceControl;  
}  
  
NTSTATUS DispatchDeviceControl(PDEVICE_OBJECT DeviceObject, PIRP Irp )  
{  
    ULONG code; //DeviceIoControl访问码  
    PIO_STACK_LOCATION p_IO_STK;  
    code = p_IO_STK->Parameters.DeviceIoControl.IoControlCode;  
    switch( code )  
    {  
        case IOCTL_DISK_GET_DRIVE_GEOMETRY: { //具体实现硬盘参数的获取 }  
        case IOCTL_DISK_GET_DRIVE_TOTALSPACE: { }  
        case IOCTL_DISK_GET_DRIVE_PRODUCTID: { }  
    }  
}
```

特权指令集

Ⅰ 特权指令仅能在核态下被**OS**使用

Ⅰ 特权指令集

- n 允许和禁止中断；

- n 在进程之间切换**CPU**；

- n 存取用于内存保护的寄存器；

- n 执行**I/O**操作；

- n 停止**CPU**的工作。

- n 从管态转回用户态

- n

用户态和核态之间的转换

I 用户态向核态转换

- n 用户请求OS提供服务
- n 发生中断
- n 用户进程产生错误（内部中断）
- n 用户态企图执行特权指令

I 核态向用户态转换的情形

- n 一般是中断返回：IRET

存储器

- I 存储程序和数据的部位

- I 分类

- n 按与CPU的联系

- u 主存：直接和CPU交换信息.

- u 辅存：不能直接和CPU交换信息

- n 按存储元的材料

- u 半导体存储器(常作主存)

- u 磁存储器(磁带，磁盘)

- u 光存储器(光盘)

- n 按存储器(半导体存储器)读写工作方式

- u RAM

- u ROM

存储体系

理想存储器：速度快，容量大，成本低

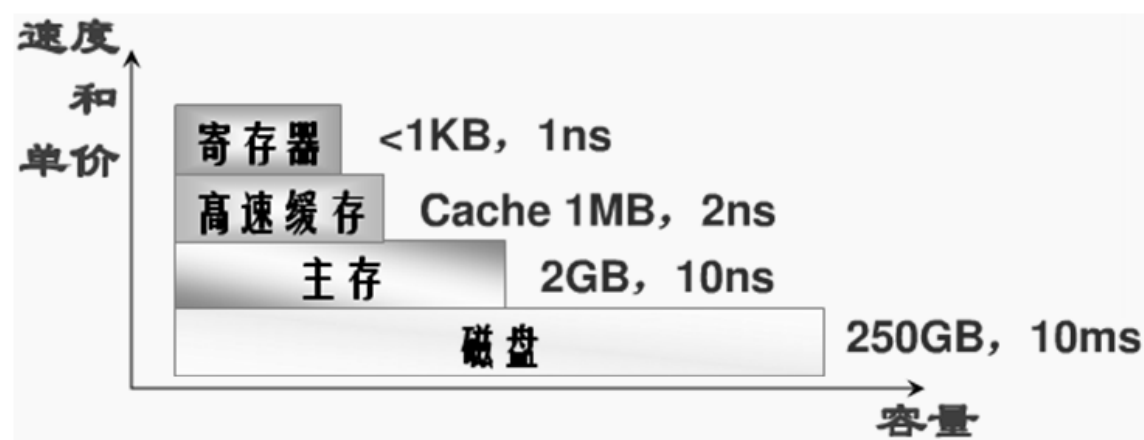
分级存储系统

n 寄存器

n 高速缓存（CACHE）

n 主存

n 磁盘



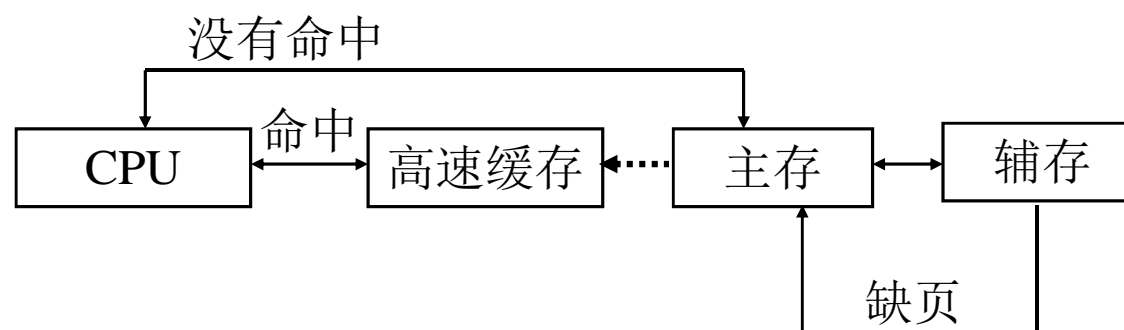
分级存储系统的工作原理

I CPU读取指令或数据时的访问顺序

n1) 访问缓存(命中, HIT)

n2) 访问内存(没有命中, MISS)

n3) 访问辅存(缺页, PAGE_FAULT)



时钟

- | 以固定间隔产生时钟信号，提供计算机所需的节拍
- | 时钟的作用
 - n 时间片；
 - n 提供绝对时间
 - n 提供预定的时间间隔
 - n WatchDog
- | 时钟的类型
 - n 绝对时钟
 - n 相对时钟

中断

I 中断定义

n指CPU对突发的外部事件的反应过程或机制。

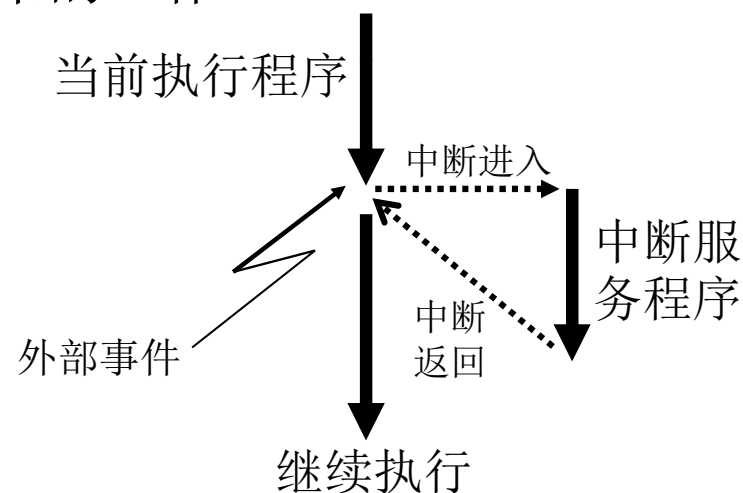
nCPU收到外部信号（中断信号）后，停止当前工作，转去处理该外部事件，处理完毕后回到原来工作的中断处（断点）继续原来的工作。

I 引入中断的目的

n实现并发活动

n实现实时处理

n故障自动处理



中断的一些概念

I 中断源和中断类型

n 引起系统中断的事件称为中断源

n 中断类型

u 强迫性中断和自愿中断

p 强迫性中断：程序没有预期：例：I/O、外部中断

p 自愿中断：程序有预期的。例：执行访管指令

u 外中断（中断）和内中断（俘获）

p 外中断：由CPU外部事件引起。例：I/O，外部事情。

p 内中断：由CPU内部事件引起。例：访管中断、程序中断

u 外中断：可屏蔽中断和不可屏蔽中断

p 不可屏蔽中断：中断的原因很紧要，CPU必须响应

p 可屏蔽中断：中断原因不很紧要，CPU可以不响应

中断响应过程

I 中断响应过程

n(1)识别中断源

n(2)保护断点和现场

n(3)装入中断服务程序的入口地址（ CS:IP ）

n(4)进入中断服务程序

n(5)恢复现场和断点

n(6)中断返回： IRET

中断的一些概念

I 断点

n 程序中中断的地方，将要执行的下一指令的地址

n CS:IP

I 现场

n 程序正确运行所依赖的信息集合。

u PSW（程序状态字）、PC、相关寄存器

u 部分内存数据

I 现场的两个处理过程

n 现场的保护：进入中断服务程序之前，栈

n 现场的恢复：退出中断服务程序之后，栈

I 中断响应的实质

n 交换指令执行地址

n 交换CPU的态

p 工作

p 现场保护和恢复

p 参数传递（通信）