

《操作系统原理》

第3章 用户界面

教师：苏曙光

华中科技大学软件学院

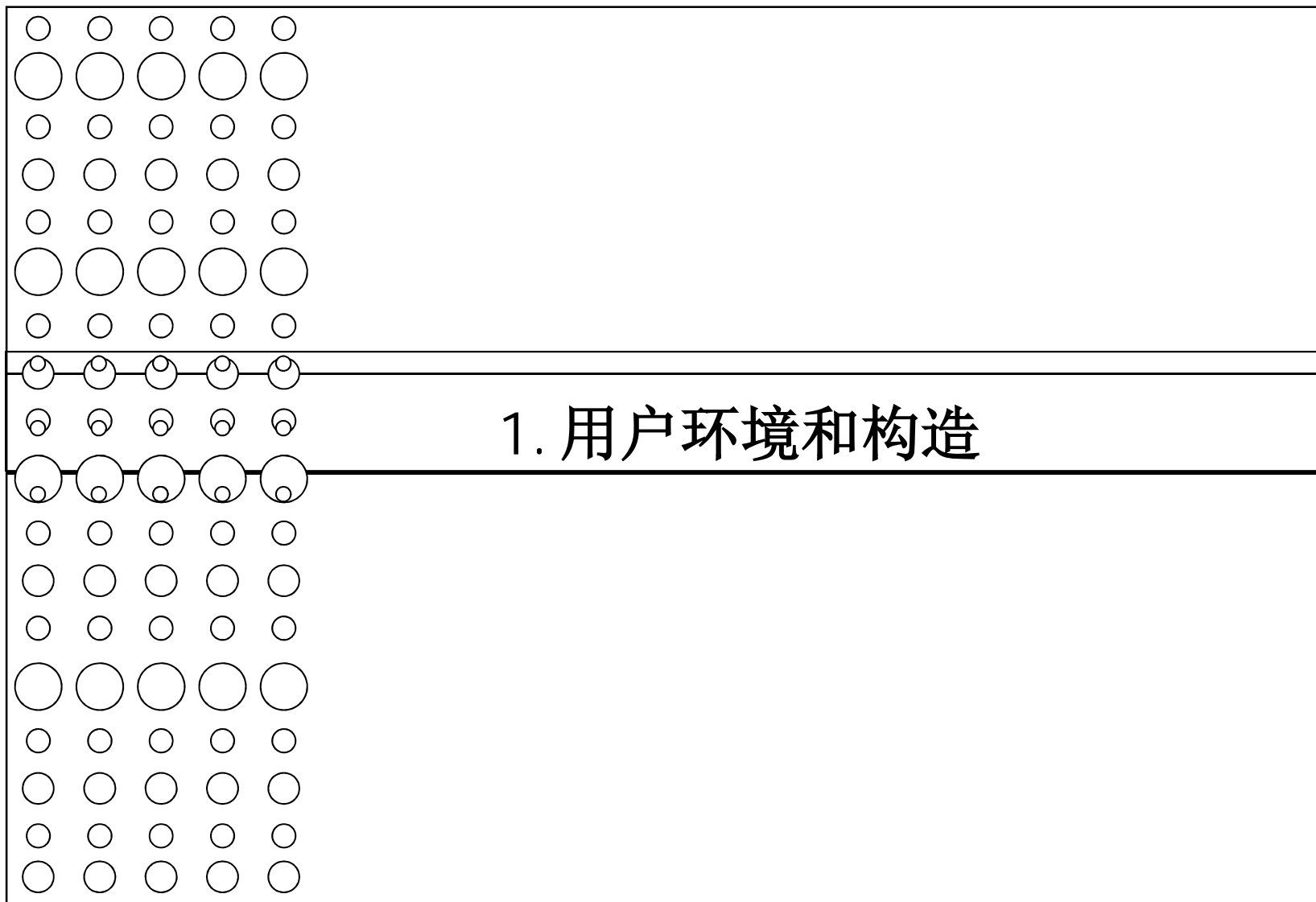
2015年3月-5月

I 主要内容

- n 用户环境
- n 操作系统的启动
- n 操作系统的生成
- n 用户界面
- n 操作界面
- n 系统调用

I 重点

- n 操作系统启动过程
- n 操作系统生成过程
- n 系统调用机制



I 用户环境

- n 用户工作的软件和硬件环境。

I 用户环境构造

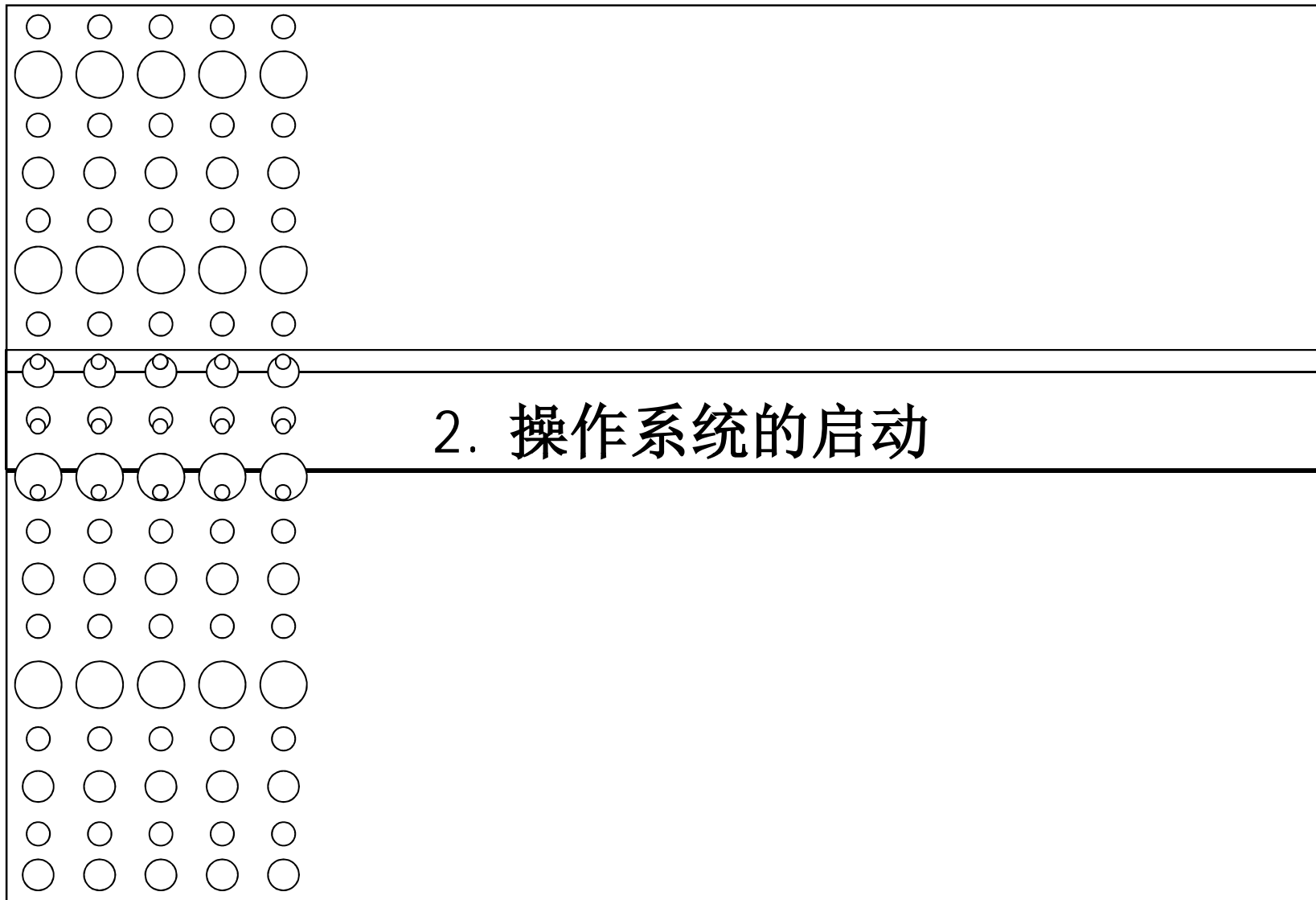
- n 按照用户要求和硬件特性安装和配置操作系统。

- u 提供丰富灵活的操作命令和界面

- u 提供系统用户手册

按软硬件特性和用户需要配置操作系统





I 相关背景知识

n 实模式和保护模式

n BIOS (Basic I/O System)

n POST (Power On Self Test)

n CMOS

n MBR (主引导记录)

n 操作系统的安装

n 多操作系统的启动选择菜单

实模式和保护模式

I 实模式（实地址模式）

- n 程序按照8086寻址方法访问00000h—FFFFFh（1MB大小）
- n 寻址方式：物理地址（20位）=段地址左移4位+偏移地址。
- n CPU单任务运行

I 保护模式（内存保护模式）

- n 寻址方式：段（32位）和偏移量（32位），寻址4GB
 - u段的属性：起始地址，存取属性，权限级别， ...
 - p段描述符表，段描述符，段选择子
 - p段页式寻址机制(段，页)
 - p段寄存器：段选择子
 - p新增32位寄存器：GDR,LDR,CR0,CR1,CR2,...
- n 虚拟地址，进程，封闭空间
- n 应用程序和操作系统的运行环境都被保护
- n CPU支持多任务

I 实模式存取1M空间

n 前面640K 【00000 -- 9FFFF】：基本内存

n 中间128K 【A0000 -- BFFFF】：显卡显存

n 末尾256K 【C0000 -- FFFFF】：**BIOS**

u C0000 -- C7FFF：显示卡BIOS

u C8000 -- CBFFF：IDE控制器BIOS

u F0000 – FFFFF：最后64KB，系统BIOS

系统BIOS

I (Basic I/O System) (Fireware, 固件)

n 基本输入/输出系统

n 位置: F0000-FFFFF

n 类型:

- uAward BIOS

- uAMI BIOS

- uPhoenix BIOS

n 功能:

- uCMOS设置

- u基本I/O设备中断服务

- uPOST(上电自检)

- u系统自举



PLCC BIOS 芯片

POST概念

I POST

nPower On Self-Test（加电自检）

n初始化基本硬件

uCPU，内存，显卡...

n自检正常不提示，错误则通过喇叭提示。

I 按下PowerOn或者Reset键

n开始执行FFFF0处的指令

JUMP POST ; POST位于系统BIOS内部

POST之后.....

- | 查找显卡**BIOS**，调用显卡 **BIOS**;
- | 依次查找其它设备执行相应设备的**BIOS**;
- | 显示启动画面
 - n**BIOS**信息
 - n主板信息
 - n芯片组型号
 - n.....
- | 根据用户指定顺序从硬盘或光驱等媒介启动**OS**。
- | **OS**启动后，由**OS**接管计算机

操作系统的启动

I 启动过程

n 从加电到用户工作环境准备好的过程

u(1)初始引导

u(2)核心初始化

u(3)系统初始化



1)初始引导

I 目的

- n 把OS核心装入内存并使之开始工作接管计算机系统

I 过程

- n 加电, JUMP POST

- n ... BIOS中的启动程序运行

- n 启动程序:

- u 读取0面0道第1扇区内容 (MBR)

- u 加载MBR中的引导程序。

- n 引导程序:

- u 根据相关参数, 读取硬盘指定位置的文件到内存

- u 加载硬盘上OS内核, 并初始化基本参数

- n OS内核: 逐步加载OS剩余部分, 最后完全控制计算机

2)核心初始化

I 核心初始化

n 目的：OS内核初始化系统的核心数据

n 典型工作

- u 各种寄存器的初始化

- u 存储系统和页表初始化

- u 核心进程构建

- u

3)系统初始化

I 系统初始化

n 为用户使用系统作准备，使系统处于待命状态。

n 主要工作

- u 初始化文件系统

- u 初始化网络系统

- u 初始化控制台

- u 初始化图形界面

- u

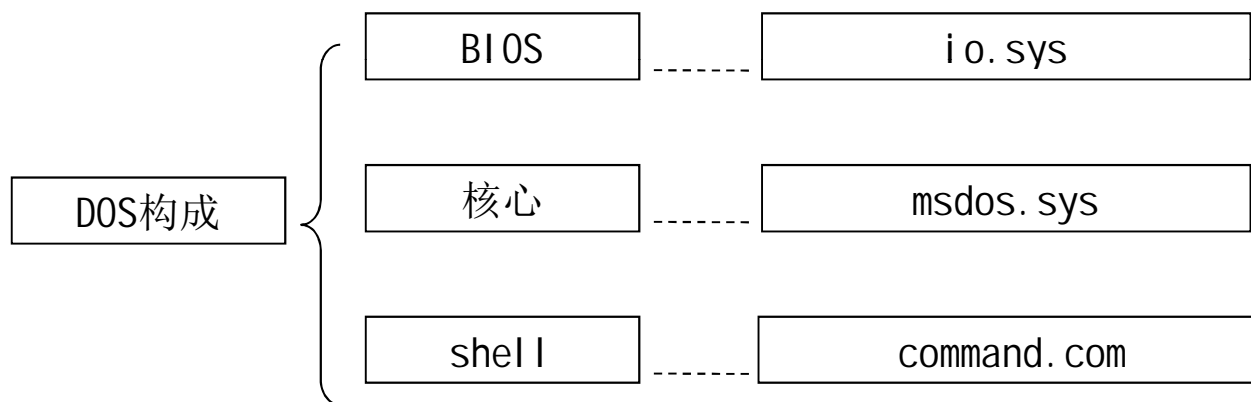
DOS启动实例

I DOS操作系统的构成

n io.sys: 提供DOS与BIOS的调用接口

n msdos.sys: 进程管理、存储管理、文件管理、解释系统调用

n command.com: Shell命令和键盘命令解释及执行



I DOS的启动过程

n POST

- u 加电后BIOS启动主机自检程序

n 初始引导

- u BIOS从MBR读入引导程序，装入内存的特定位置

- u 引导程序运行将io.sys及msdos.sys读入内存

- u DOS运行起来取代BIOS接管整个系统。

n 核心初始化

- u 操作系统读入config.sys配置系统核心

n 系统初始化

- u 读入Command.com，执行autoexec.bat，系统待命

I Windows的启动过程

n POST

- u 加电后BIOS启动主机自检程序

n 初始引导

- u BIOS从MBR读入引导程序，装入内存的特定位置

- u 引导程序启动DOS7.0，调入操作系统核心

- u WINDOWS开始接管系统

n 核心初始化

- u 资源状态、核心数据等初始化；

n 系统初始化

- u GUI 界面生成，系统处于待命/消息接受状态

I LINUX的启动过程

n POST → MBR → KERNEL映像 → KERNEL映像自解压并执行 → 内核初始化 → 内核启动 →

I 注释

n 当KERNEL映像被加载到内存之后，内核阶段就开始了。

n KERNEL映像是一个zlib压缩过的内核映像，通常是一个zImage（小于 512KB）或 bzImage（大于 512KB）。

n 在KERNEL映像最前面是一个可执行例程（实现少量硬件设置，并对映像其余部分解压放入高端内存中）。然后该例程会调用内核，并开始启动内核引导的过程。



MBR (Master Boot Record)

I MBR

- n 存放和OS启动的相关信息

- n 512 BYTES

- n 结束: 0xAA55h

I 安装多操作系统对MBR的影响

- n MBR重写

- n MBR追加

- n 注意: 安装顺序

I 查看和修改MBR内容

- n DiskEdit (DOS) , WinHex, HxD, GDisk (Windows)

- n DD : `dd if=/dev/hda of=./mbr bs=512 count=1`

MBR的工作原理

- I (1) **POST** → **CMOS**设置（硬盘启动） → 读取**MBR** → 控制权交给**MBR**。
- I (2) **MBR**读取分区表(**Partition Table**), 找到其中的活动分区 (**Active Partition**), 并确认其他的分区都不是活动分区。**MBR**读取活动分区的第一个分区（次引导记录**PBR**），并把它加载到内存中去。
- I (3) **PBR**控制后面的引导过程。

I MBR的结构:

nBoot Loader

u000~1BD,446B

nPartition Table

u该硬盘的分区表

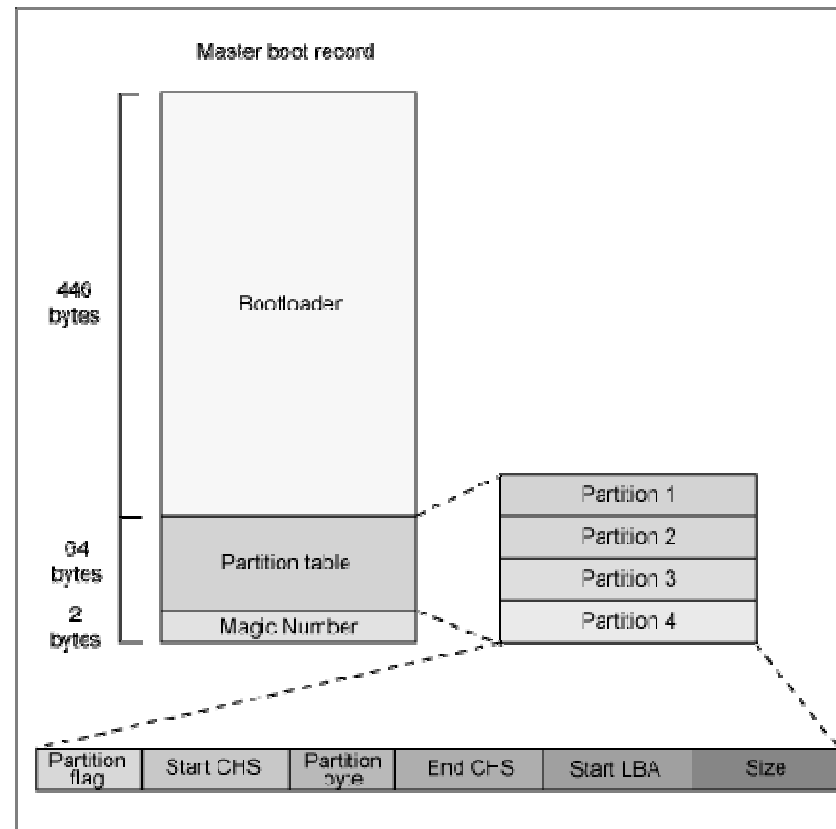
u1BE~1FD, 64B

nMagic number 2B

u1FE~1FF

u55AA

u表示MBR结束。



MBR程序的例子

；功能：加电开机后显示 “Hello, OS world!”

1 org 07C00h ; 程序加载到7C00处

2 mov ax, cs ;

3 call DispStr ; 调用显示字符串例程

4 jmp \$; 无限循环

5 DispStr:

6 mov ax, BootMessage

7 mov bp, ax ; es:bp = 串地址

8 mov cx, 16 ; cx = 串长度

9 mov ax, 1301h ; ah = 13h, al = 01h

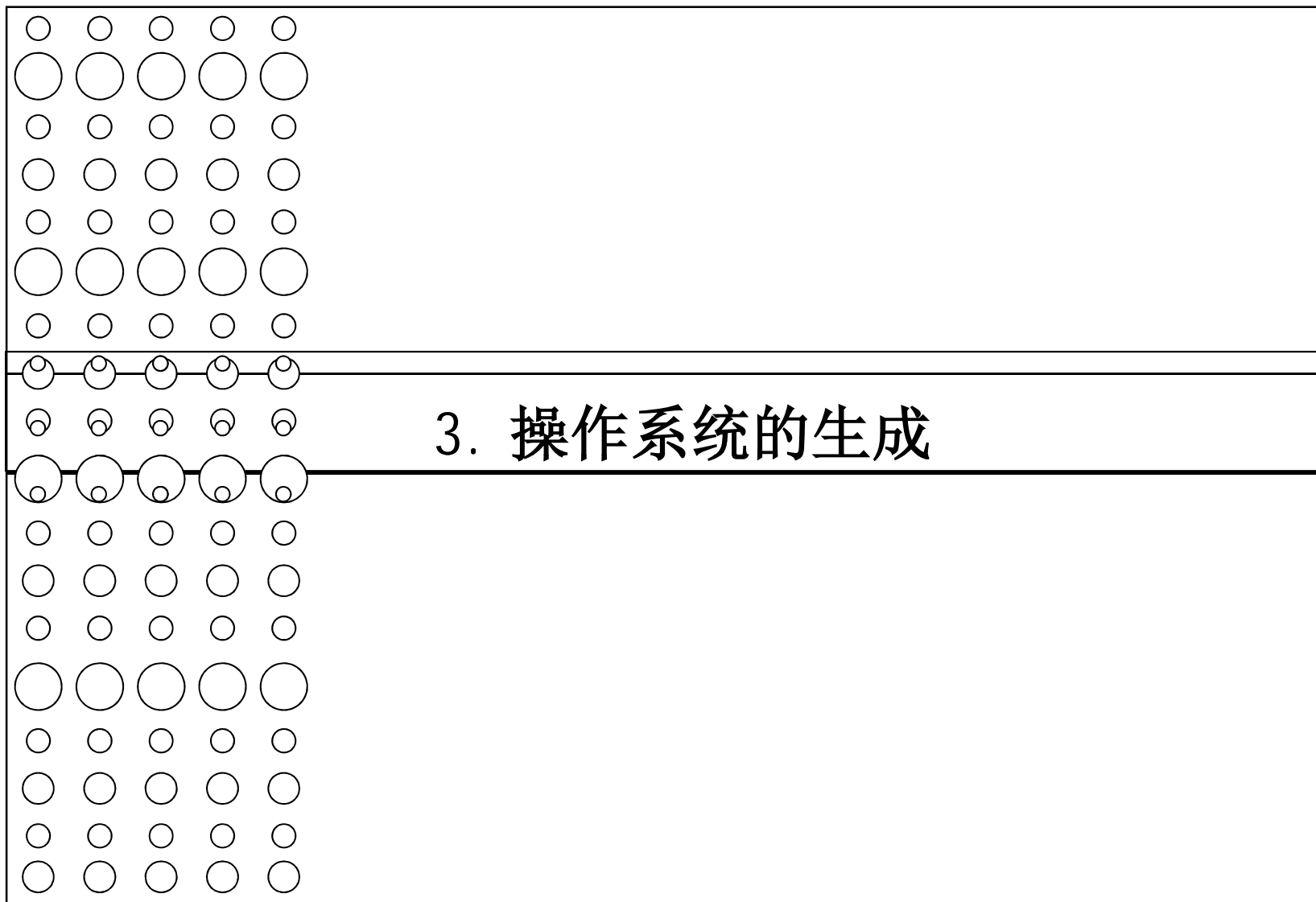
10 int 10h ; BIOS 10h 号中断:显示字符串

11 ret

12 BootMessage: db "Hello, OS world!"

13 times 510-(\$-\$\$) db 0 ; 填充若干个0, 使代码字节为510字节

14 dw 0xAA55 ; 结束标志



I 操作系统的生成

- n 满足特定硬件环境和用户需要，组装和构建操作系统过程。

I 操作系统生成的主要步骤

- n 根据硬件环境/用户要求配置功能模块和构造参数

- n 构建（**build**）OS的映像

I 操作系统的生成的前提

- n 操作系统由可拆装模块构成

- n 有交互式配置工具

- n 有映像构建（**build**）工具

I Linux操作系统的生成

- n1、获取Linux内核的源代码（最好是当前版本）
- n2、选择和启动内核配置程序
- n3、根据需要配置内核模块和参数
- n4、重新编译新的内核
- n5、编译和安装模块
- n6、启动新内核



I 1、获取Linux内核的源代码

n<http://www.kernel.org/>

cd /usr/src

tar zxvf linux-2.6.38-12.tar.gz

I 2、选择和启动内核配置程序

cd /usr/src/linux-2.6

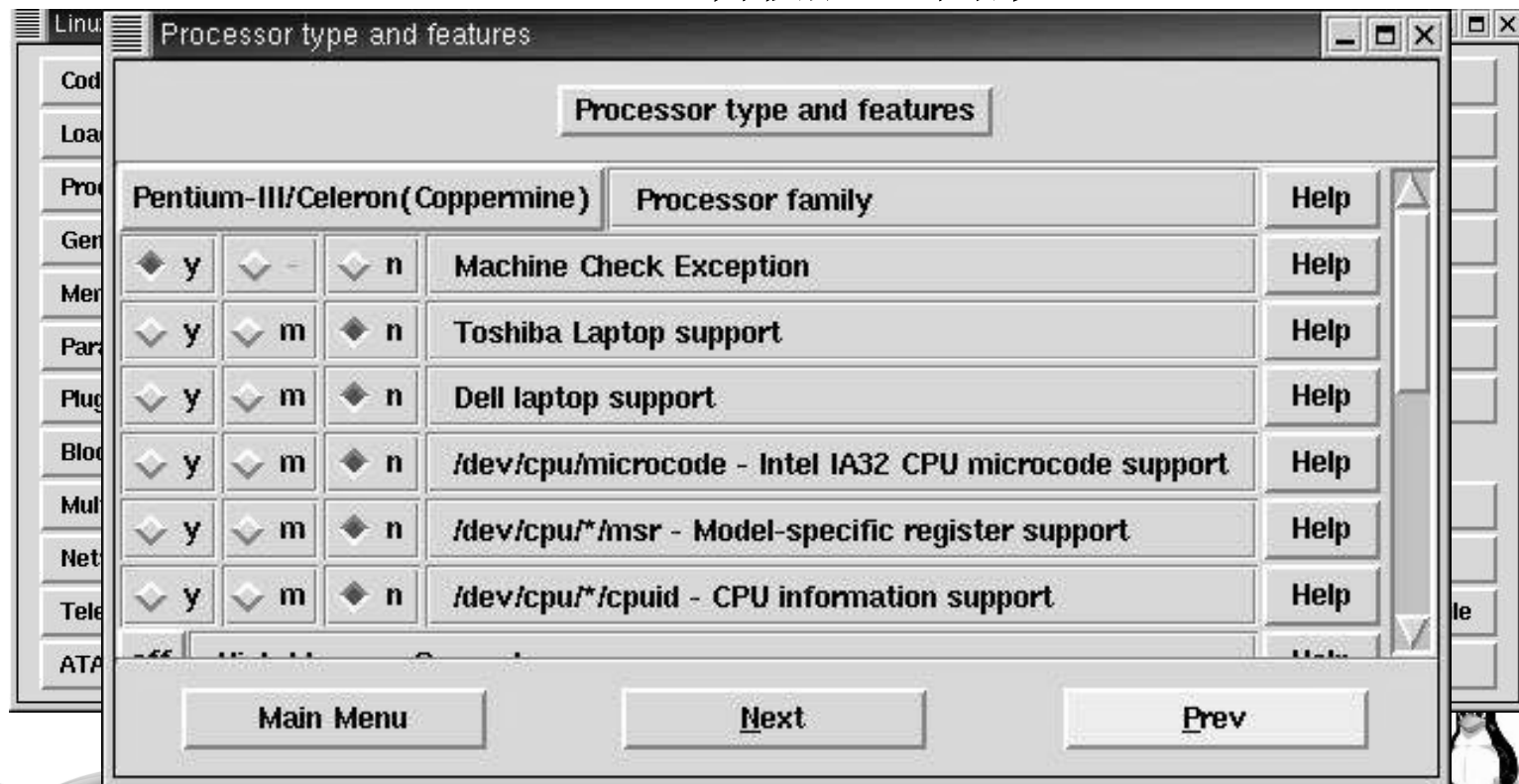
#make config（文本界面，不推荐使用）

#make xconfig（图形窗口模式，xWindow使用）

#make menuconfig（文本选择界面，字符终端）

make xconfig

•Linux内核配置对话框



make menuconfig

root@susg-asus: /home/susg/LinuxKernel/linux-2.6.38.2

文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

.config - Linux/i386 2.6.38.2 Kernel Configuration

USB support

Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [] excluded <M> module < >

^(-)
< > USB Attached SCSI (NEW)
[] The shared table of common (or usual) storage devices
*** USB Imaging devices ***
<M> USB Mystek MDC800 Digital Camera support
<M> Microtek X6USB scanner support
*** USB port drivers ***
<M> USS720 parport driver
<M> **USB Serial Converter support --->**
*** USB Miscellaneous drivers ***
<M> EMI 6|2m USB Audio interface support
v(+)

<Select> < Exit > < Help >



3、根据需配置内核模块和参数

I ① Loadable module support 设置对可加载模块支持。

nEnable loadable module support (y)

nSet version info on all module symbols (n)

nKernel module loader (y)

I ② **Processor type and features** 设置CPU的类型

nProcessor family 选择CPU类型

nHigh Memory Support (n)

nMath emulation (n)

nMTTR support: (n)

nSymmetric multi-processing support (n)



I ③ **General setup** 对普通的一些属性进行设置。

n Networking support: (y)

n PCI support (y)

n PCI access mode PCI卡存取模式: BIOS、Direct和Any

n Support for hot-pluggable devices (n)

n PCMCIA/CardBus support (n)

I ④ **Parallel port support** 并口支持 (y)

I ⑤ **Plug and Play configuration:** 即插即用配置 (y)

I ⑥ **Block devices** 块设备支持的选项

n Normal PC floppy disk support 软盘支持 (y)

n Network block device support 网络块设备支持 (y)



I ⑦ **Networking options** 选取TCP/IP networking选项

I ⑧ **Network device support** 网络设备支持的选项

例如：使用Realtek 8139网卡

n Ethernet (10 or 100Mbit) (y)

n RealTeck RTL-8139 PCI Fast Ethernet Adapter support (y)

I ⑨ **Mice** 鼠标设置选项：串口、**PS/2**等类型鼠标

I ⑩ **File systems** 文件系统类型。

n DOS FAT fs 选项：FAT16, FAT32

n NTFS file system support

n /proc file system support: (y)

I ⑪ **Sound** 声卡驱动，选项：声卡型号

I ⑫ **USB support** USB接口的支持，根据需要选择。



I 4、重新编译新的内核

make dep 生成依赖dependency信息

make clean 清除旧的编译结果

make bzImage ./arch/i386/boot/bzImage

I 5、编译和安装模块

make modules

make modules_install

模块被编译且安装到 /usr/lib/<内核版本号> 目录下。



I 6、启动新内核

ncp bzImage /boot/bzImage

nLILO

u 配置/etc/lilo.conf

```
image=/boot/bzImage
```

```
label=newLinux build by Zhang San Feb.28, 2012
```

u 命令 #lilo 使配置生效

nGRUB (与发行版本有关)

u 配置/boot/grub/grub.conf

```
title newLinux build by Zhang San Feb.28, 2012
```

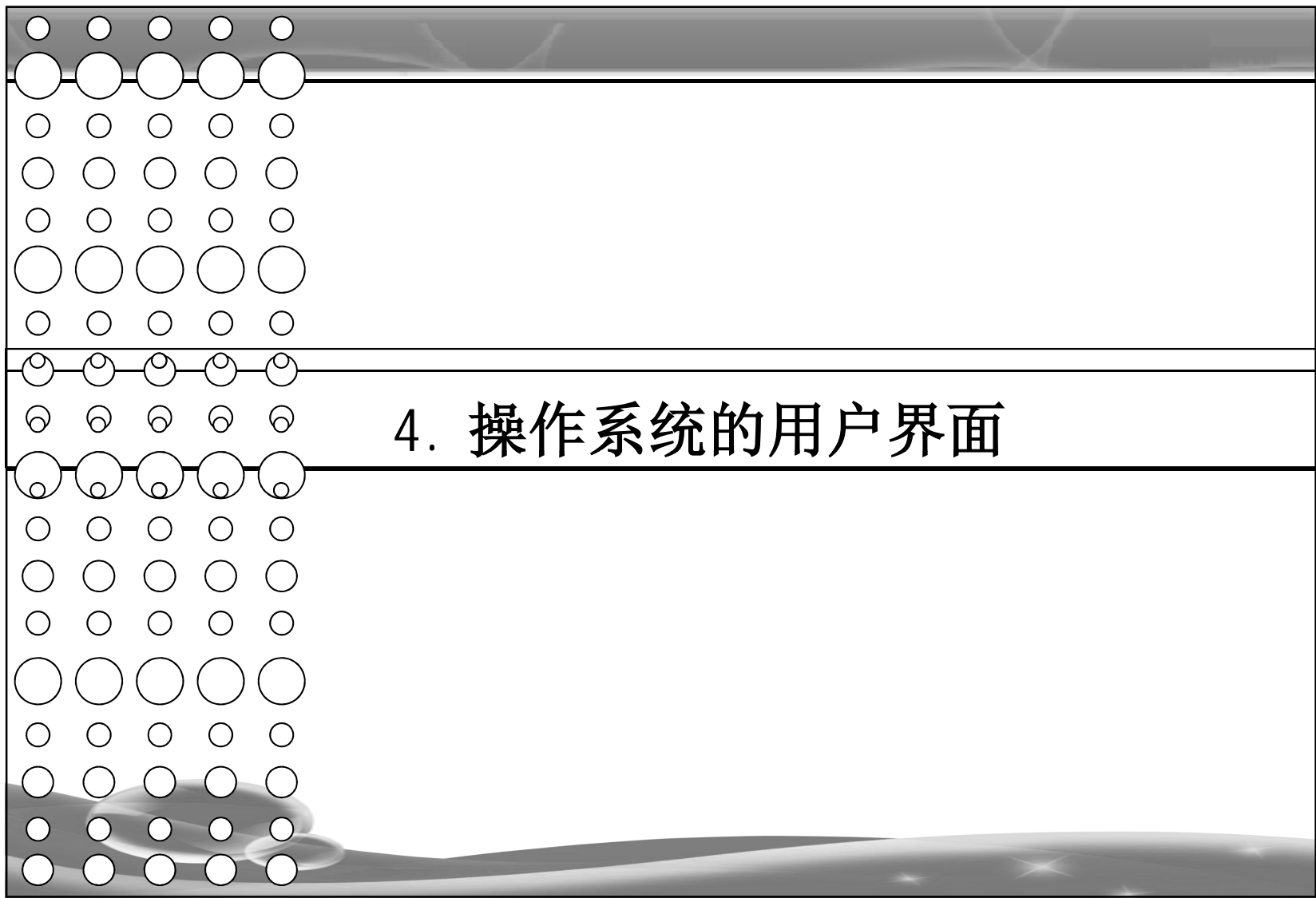
```
root (hd0,1)
```

```
kernel /boot/bzImage ro root=/dev/hda2
```



课后作业1：

- I 在Fedora/Ubuntu中如何编译一个新的内核？请按“STEP BY STEP”的标准写出全过程。先介绍Fedora的版本，从网上下载新的内核源代码开始，写出其具体过程，包括每一个命令的使用。
 - n 可以是DOC文档，也可以是PPT文档
 - n 要求有电脑屏幕截图
 - n 下周同一时间抽查若2名学生上台汇报（8分钟）。
 - u 请提前一天将DOC或PPT压缩发到OSCourse@163.com
 - u 缺交按缺勤1次算。
 - n 课后作业1和课后作业2任选一个



4. 操作系统的用户界面

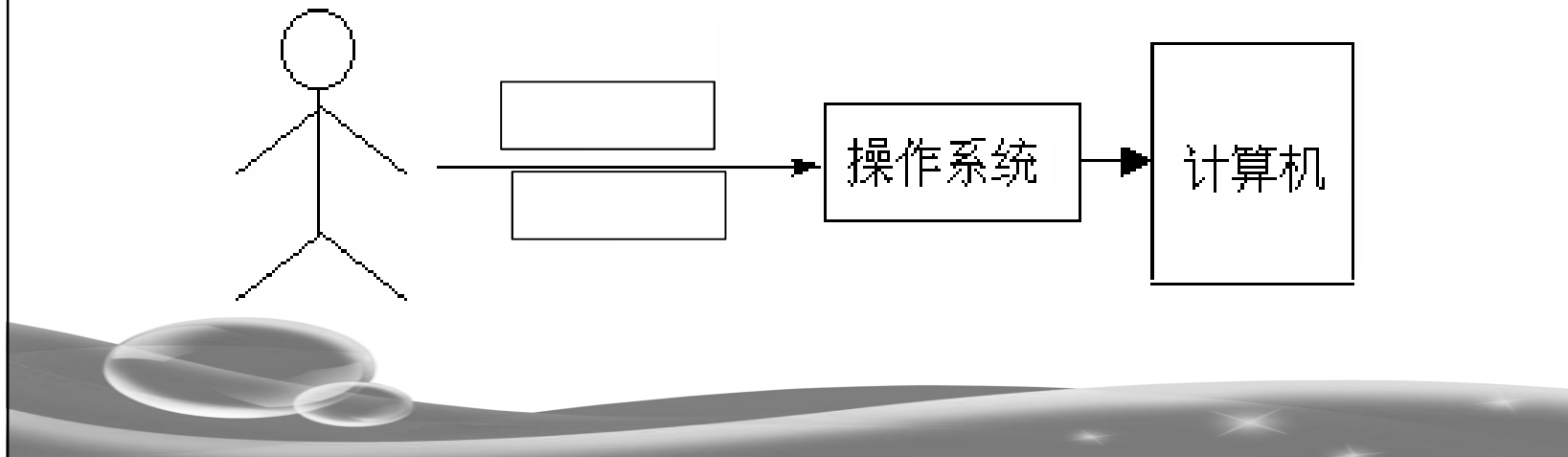
I 用户界面的定义

nOS提供给用户控制计算机的机制， 又称用户接口。

I 用户界面的类型

n操作界面

n系统调用 (System Call, 系统功能调用, 程序界面)



I 操作界面

- u 图形用户接口（GUI, Graphic User Interface）

- u 窗口，图标，菜单，按钮，鼠标(消息，事件)

- u 键盘命令（COMMAND）

- u 在控制台环境下接收键盘输入的命令

- u 类型

- p 普通命令

- p 批处理程序

- p shell

- u 作业控制语言（JCL, Job Control Language）

普通命令

I DOS典型命令

n 文件管理

uCOPY、COMP、TYPE、DEL、REN

n 磁盘管理

uFORMAT、CHKDSK、DISKCOPY、DISKCOMP

n 目录管理

uDIR、CD、MD、RD、TREE

n 设备工作模式

uCLS、MODE

n 日期、时间、系统设置

uDATE、TIME、VER、VOL

n 运行用户程序

uMASM、LINK、DEBUG

普通命令

I LINUX的典型命令

命令	基本功能	命令	基本功能
man	查看命令的帮助	tar	管理 TAR 类型的文件
cd	改变工作路径	chown, chgrp	设置文件/目录的拥有者
ls	列目录信息	chmod	改变文件属性
ps	显示系统中的进程及 ID	find	用于查找某个文件
mount	挂接文件系统	locate	查找文件
umount	卸掉文件系统	whereis	查看文件放在哪个目录



I 批处理

n 普通命令的集合，批执行,由command解释执行。



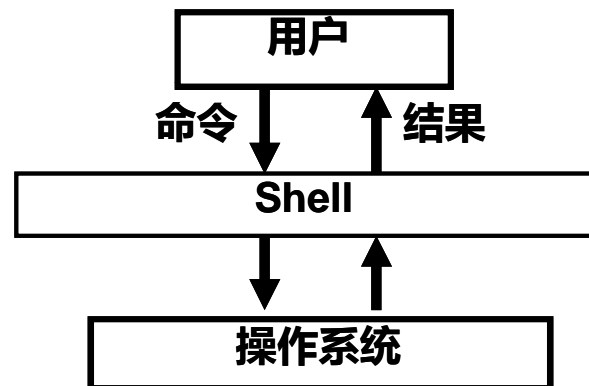
```
MakeDrv.bat - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

@echo off
if "%1"==" " goto usage
if "%3"==" " goto usage
if not exist %1\bin\setenv.bat goto usage
call %1\bin\setenv %1 %4
%2
cd %3
build -b -w %5 %6 %7 %8 %9
goto exit
:usage
echo 使用说明 MakeDrv DDK安装目录 用户盘符 用户目录 free/checked

[build_options]
echo 例如 MakeDrv %%DDKDir%% D: %%WDMUser%% free -cef
:exit
```

Shell

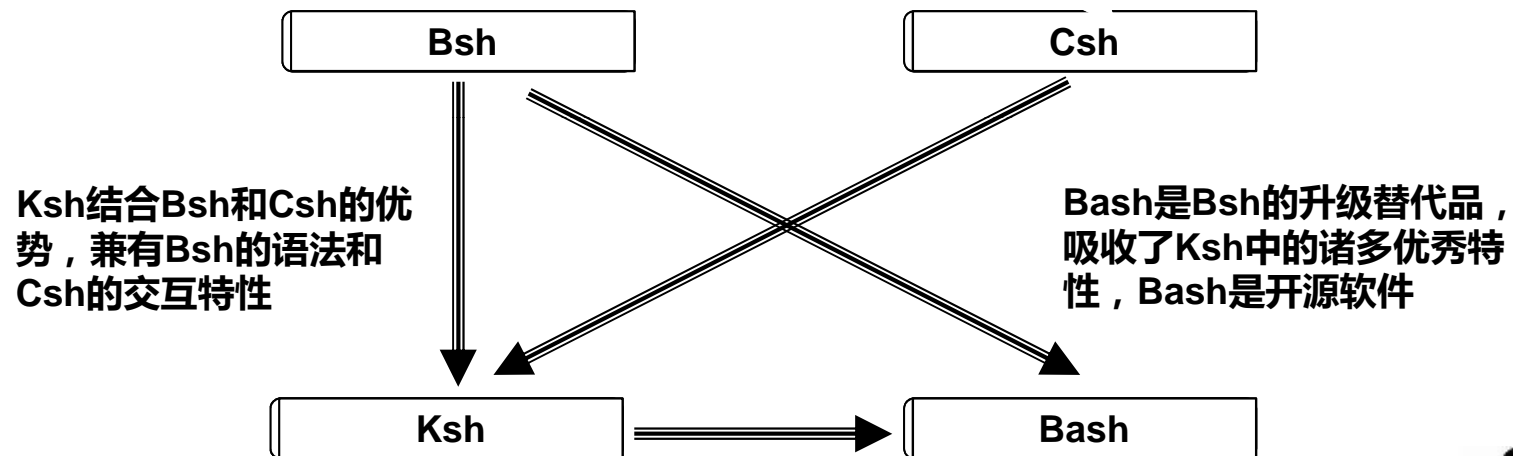
- I **Shell**是一个控制台方式的程序，通过类似程序的方式执行具有一定逻辑顺序的命令序列完成某个较复杂的功能和人机交互，最后返回结果。
- I **Shell**是操作系统与用户进行交互操作的界面



I Shell的发展与分类

Bsh在20世纪70年代中期诞生于新泽西的AT&T贝尔实验室，具有较强的脚本编程功能

Csh在20世纪80年代早期诞生于加利福尼亚大学，使用C语言的语法，用户命令交互更加方便



Bash的主要功能

- | 命令行编辑功能
- | 命令和文件名补全功能
- | 命令历史功能
- | 命令别名功能
- | 提供作业控制功能
- | 管道与重定向
- | 具有将命令序列定义为功能键的功能
- | **Shell脚本编程**



Bash的命令行编辑功能

操作键	功能
左右方向键	使用左右方向键可以使光标在当前命令行中的已有字符间进行任意的移动
退格键	删除命令行中光标左边的字符
Del	删除当前光标处的字符
Home	将光标快速移动到命令行的行首
End	将光标快速移动到命令行的行尾
Ctrl + u	删除当前光标到行首的内容
Ctrl + k	删除当前光标到行尾的内容



Bash的命令行补全功能

I 命令补全功能

n 使用Tab键可在命令查找路径中查找匹配的命令，并进行命令拼写的补全

I 文件补全功能

n 使用Tab键可对文件和目录名进行补全



Bash的命令历史与命令重复

- | 命令历史功能的使用
 - n 使用上下方向键浏览已输入命令（历史命令）
- | 历史命令的查看
 - \$ history
- | 用户命令历史保存文件
 - ~/.bash_history
- | 命令历史的清除
 - \$ history -c
- | 相关的环境变量
 - HISTFILE, HISTSIZE**

管道与重定向

- | 标准输入输出
- | 重定向操作
- | 管道操作

标准输入输出 (Linux)

输入输出文件	文件编号	默认设备
标准输入	0	键盘
标准输出	1	显示器
标准错误输出	2	显示器

重定向操作 (Linux)

类别	操作符	说明
输入重定向	<	输入重定向是将命令中接收输入的途径由默认的键盘更改（重定向）为指定的文件
输出重定向	>	将命令的执行结果重定向输出到指定的文件中，命令进行输出重定向后执行结果将不显示在屏幕上
	>>	将命令执行的结果重定向并追加到指定文件的末尾保存
错误重定向	2>	清空指定文件的内容，并保存标准错误输出的内容到指定文件中
	2>>	向指定文件中追加命令的错误输出，而不覆盖文件中的原有内容
输出与错误组合重定向	&>	将标准输出与错误输出的内容全部重定向到指定文件

重定向操作的例子 (Linux)

I 将命令输出重定向到文件

n将标准输出重定向到文件

```
$ ls /etc/ > etcdir
```

n将标准输出重定向追加到文件

```
$ ls /etc/sysconfig/ >> etcdir
```

n将错误输出重定向到文件

```
$ nocmd 2> errfile
```

n将标准输出和错误输出重定向到文件

```
$ ls afile bfile &> errfile 或者
```

```
$ ls afile bfile > ab.out 2>&1
```

管道 |

| 管道操作符 |

n “|”符用于连接左右两个命令，将“|”左边命令的执行结果（输出）作为“|”右边命令的输入

```
cmd1 | cmd2
```

| 在同一条命令中可以使用多个“|”符连接多条命令

```
cmd1 | cmd2 | ... | cmdn
```

Shell脚本编程

I Shell脚本程序（ Shell Script ）

nShell命令语句的集合，用于实现特定的功能；

- u 所有命令将逐行执行（按逻辑）。
- u 凡是能够在shell下直接执行的命令，都可以在脚本中使用。
- u 脚本中还可以使用一些不能在shell提示符下直接执行的语句，这些语句只有在脚本中使用才有效。

nShell脚本程序保存在文本文件中；

nShell脚本程序由Shell环境解释执行；

n执行Shell脚本文件需要具有可执行属性（x）。



基本Shell脚本编程

- I 使用文本编辑器（如vi）建立Shell脚本文件
- I 脚本可以包括的内容：
 - n1) 脚本语句（命令的有机集合）
 `history -c`
 - n2) 脚本运行环境设置
 `#!/bin/bash` （首行且顶格）
 - n3) 注释行，以#开始
 `# Clean command history,清除用户命令历史`
- I 设置脚本文件为可执行属性



课后作业2

I 在Fedora/Ubuntu中编写一个shell

n 功能从文件中读取每一行显示并统计总行数

u 在shell运行过程中指定文件

n 源代码+DOC文档或PPT文档

n 要求有电脑屏幕截图

n 下周同一时间抽查若2名学生上台汇报（8分钟）。

u 请提前一天将源代码或DOC/PPT压缩发到
OSCourse@163.com

u 缺交按缺勤1次算。

u 课后作业1和课后作业2任选一个

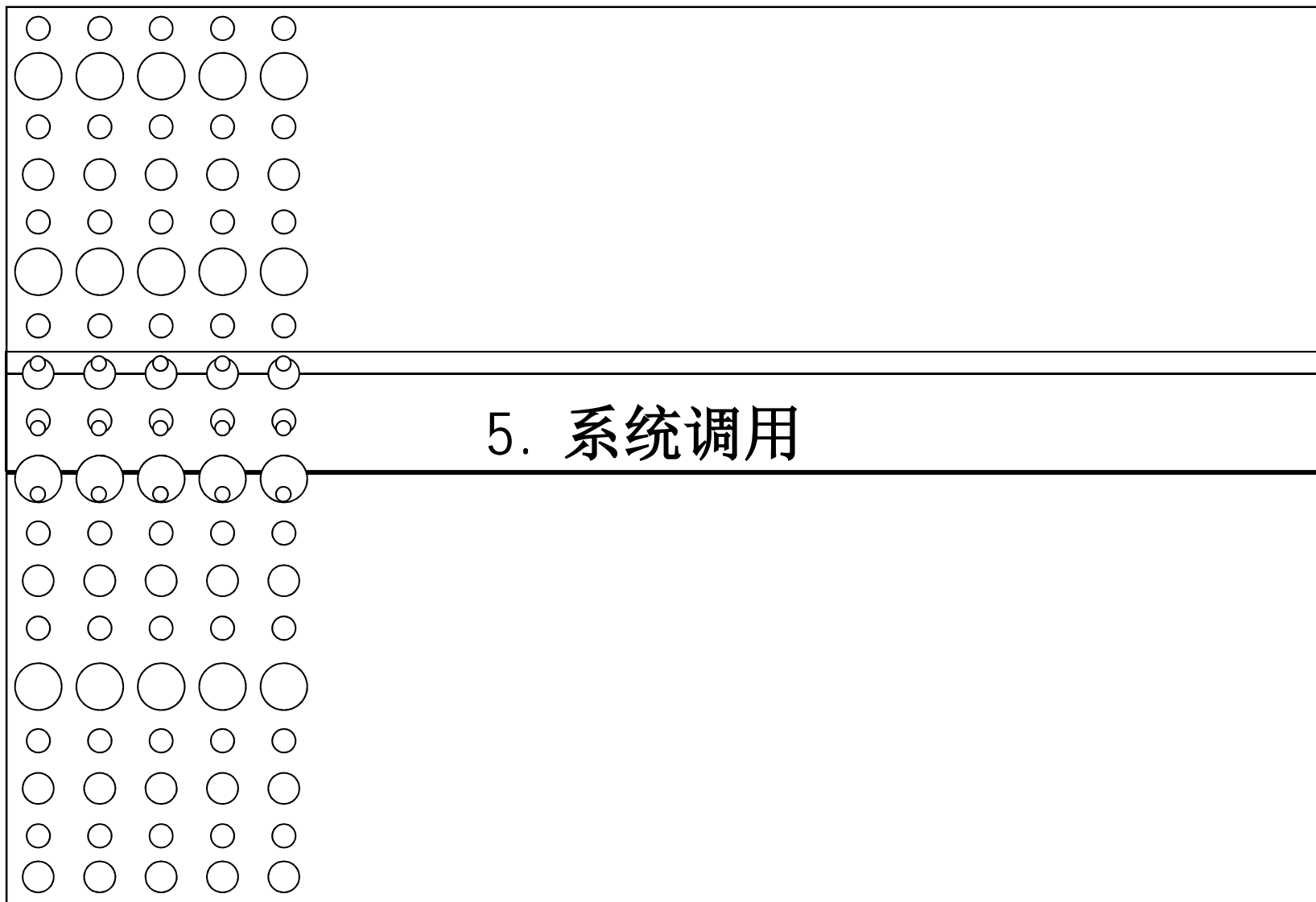
Shell编程例子（课后作业）

I 例子：从文件中读取每一行显示并统计行数

```
#!/bin/bash
#
let COUNTS=0

echo "Please enter a file:"
read FILE

if [ -e $FILE -a -f $FILE ]; then
    while read LINE
    do
        echo $LINE
        COUNTS=$((COUNTS+1))
    done < $FILE
    echo "There are $COUNTS lines."
fi
```



编程时利用ReadFile从文件中读

I 从文件当前位置读取4个字节，存入chBuffer

```
#021      const int BUFSIZE = 4096;
#022      char chBuffer[BUFSIZE];

#046      DWORD dwReadSize = 0;
#047      bRet = ::ReadFile(hFile,chBuffer,4,&dwReadSize,NULL);
#048      if (bRet)
#049      {
#050          //
#051          OutputDebugString(_T("ReadFile 读文件成功\r\n"));
#052      }
```

I ReadFile函数功能

- n 文件系统操作/磁盘操作
- n 从文件当前位置读取若干字节
- n 操作系统内核实现

I ReadFile函数原型

```
BOOL ReadFile (  
    HANDLE hFile,      //文件指针  
    LPVOID lpBuffer,   //数据缓冲  
    DWORD nNumberOfBytesToRead, //要读取的字节数  
    LPDWORD lpNumberOfBytesRead, //已读取的字节数  
    LPOVERLAPPED lpOverlapped    //覆盖缓冲 )
```

DOS : 利用INT 21h 09号子功能输出字符串

I 例：显示字符串

```
string db 'Hello,Everybody !'  
                ; 定义要显示的字符串  
  
...  
mov ah,09h      ; ah←09h 号子功能  
mov dx,offset string ; dx←字符串的偏移地址  
int 21h        ;
```

DOS 21h中断子功能列表（部分）

00.	程序终止(同 INT 20H).
01.	键盘输入并回显.
02.	显示输出.
03.	异步 <u>通讯</u> 输入.
04.	异步 <u>通讯</u> 输出.
05.	打印机输出.
06.	直接控制台 I/O.
07.	键盘输入(无回显).
08.	键盘输入(无回显) 检测 Ctrl-Break.
09.	显示字符串.

Linux例子

I 程序的功能

- n 打开一个文件
- n 显示文件内容
- n 关闭文件

I 系统函数列表

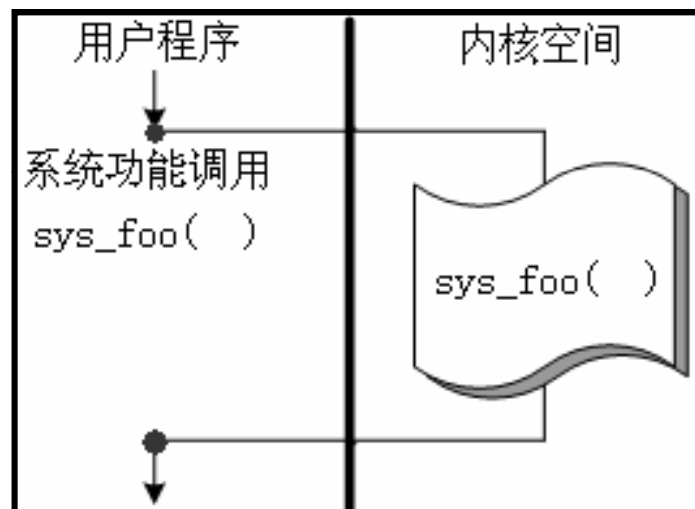
- n fopen
- n printf
- n fgetc
- n putchar
- n exit

```
1  #include <stdio.h>
2  main()
3  {
4      FILE *fp;
5      char ch;
6      if((fp=fopen("test","rt")) == NULL)
7      {
8          printf("\n Can't open file !");
9          exit(1);
10     }
11     ch = fgetc(fp);
12     while(ch != EOF)
13     {
14         putchar(ch);
15         ch = fgetc(fp);
16     }
17     fclose(fp);
18 }
```

系统调用

n 系统调用(System Call, System Service Call)

- n 操作系统内核为方便应用程序编程而提供的一系列服务/函数。这些服务/函数叫做系统调用/系统功能调用/System Call。



Linux的系统调用（部分）

进程控制：

fork	创建一个新进程
clone	按指定条件创建子进程
execve	运行可执行文件
exit	中止进程
_exit	立即中止当前进程
getdtablesize	进程所能打开的最大文件数
getpid	获取进程标识号
getppid	获取父进程标识号
getpriority	获取调度优先级

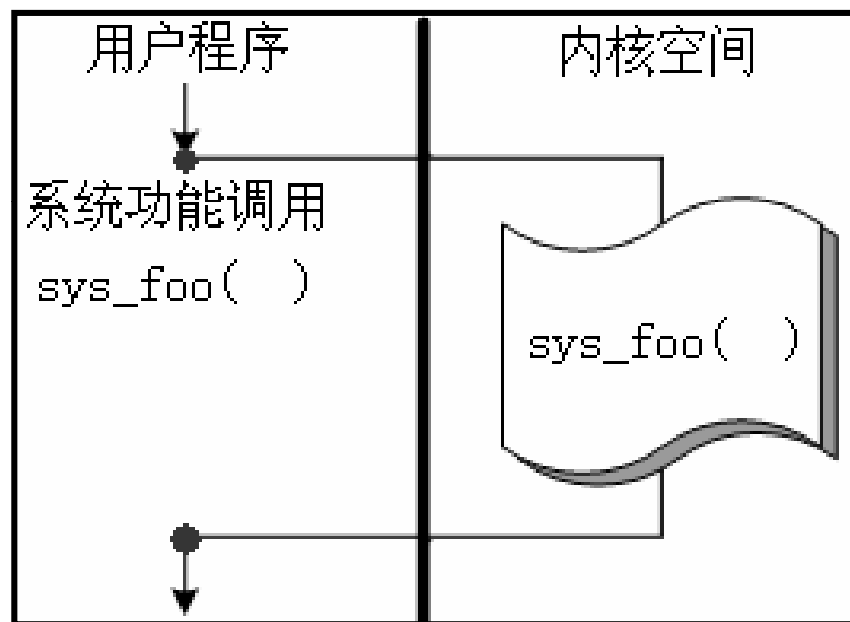
文件读写操作

fcntl	文件控制
open	打开文件
creat	创建新文件
close	关闭文件描述字
read	读文件
write	写文件
readv	从文件读入数据到缓冲数组中



系统调用的特点

- | 一般涉及核心资源的操作或底层功能的操作
- | 由操作系统内核完成服务，运行于核态
- | 调用过程会产生中断



继续：系统调用的原理

I 系统调用的原理

- n 系统调用的函数在**核态**执行。
- n 应用程序工作在**用户态**。
- n 应用程序通过**中断**机制才能进入核态。
 - u 自愿中断
 - u 每个系统调用具有唯一编号ID
 - u OS为系统调用维护入口地址（表）

入口 地址表	
1号地址	
2号地址	
X号地址	
N号地址	

I 系统调用的实现形式

I 调用N号系统调用，使用指令：

SVC	N
-----	---

uN: 系统调用的编号

uSVC: SuperVi sor Call，访管指令

uSVC是中断指令

系统功能

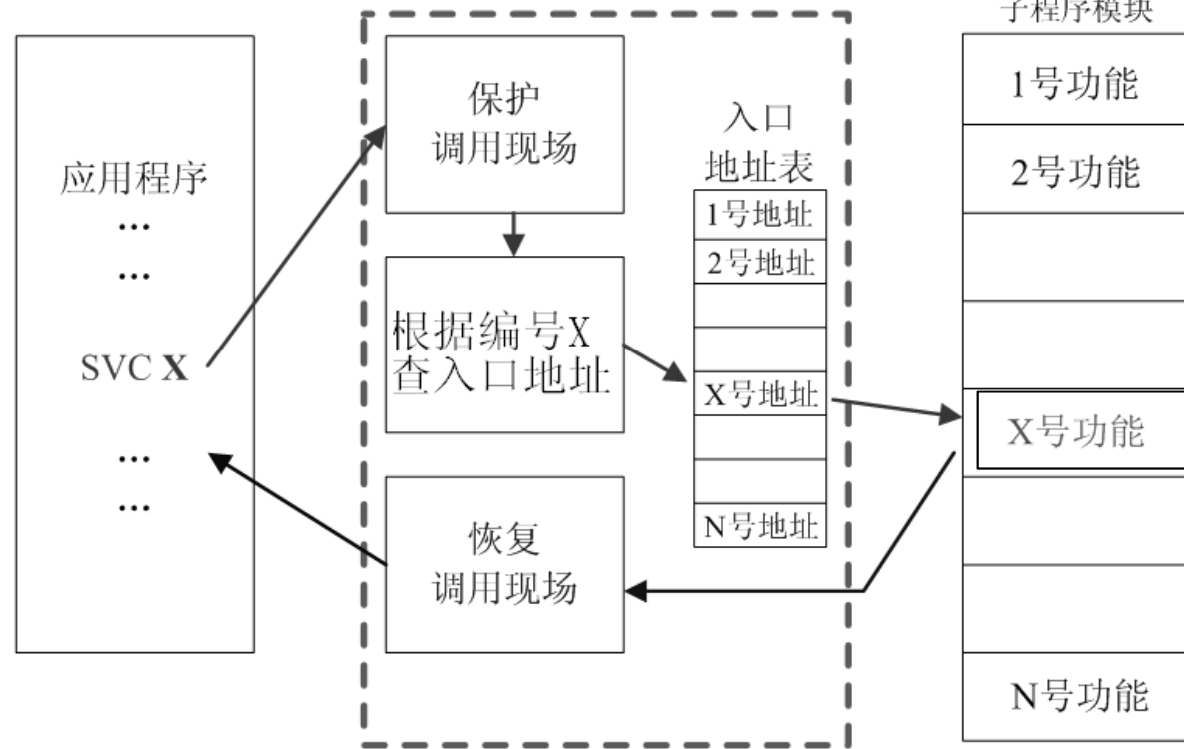
1号功能

2号功能

X号功能

N号功能

I 系统调用的执行过程（中断）



n具体OS中系统调用的实现

nDOS: INT 21H

nLINUX: INT 80H

nWindows 2K: INT 2EH （所有NT内核的系列）

nWindows XP : SYSENTER （WindowsXP及以上）

n注意:

- u**上述指令都是中断指令，等同于“**SVC**”指令

- u**中断之前通过设置参数指定系统功能调用编号**N**。

各种OS系统调用的例子

I DOS的例子

I 例：显示字符串（系统调用编号**09h**）

```
string db 'Hello,Everybody !'
```

； 定义要显示的字符串

...

```
mov ah,09h
```

； 指定编号：ah←09h

```
mov dx,offset string
```

； dx←字符串的偏移地址

```
int 21h
```

； 系统调用

INT 21H = SVC ;

09h = N

I 例：输入字符（系统调用编号**01h**），判断按键Y/N？

```
GetKey:  mov ah, 01h          ; 指定编号: ah←01h
         int 21h              ; 系统调用
         cmp al, 'Y'          ; 处理出口参数al
         je  YesKey           ; 是“Y”
         cmp al, 'N'          ;
         je  NoKey            ; 是“N”
         jne GetKey
```

...

```
YesKey:  ... (处理Y键)  ...
```

```
NoKey:  ... (处理N键)  ...
```


各种OS系统调用的例子

I Linux的例子（在屏幕上输出Hello World）

```
mov eax, 4    ;系统调用编号：4号（文件写）
mov ebx, 1    ;ebx送1表示stdout
mov ecx, msg  ;字符串的首地址送入ecx
mov edx, 14   ;字符串的长度送入edx
int 80h       ;输出字符串
mov eax, 1    ;系统调用编号：1号（进程结束）
int 80h       ;结束
msg: db "Hello World!"
```

INT 80H = SVC ;

4 = N

I 在unistd.h文件中定义系统调用的编号（前缀 __NR__）

```
#define __NR_exit      1
#define __NR_fork      2
#define __NR_read      3
#define __NR_write     4
#define __NR_open      5
#define __NR_close     6
#define __NR_waitpid   7
#define __NR_creat     8
#define __NR_link      9
```



各种OS系统调用的例子

I Win2K

I 例子：获取本机系统信息（进程信息，编号**0x97**）

NtQuerySystemInformationNo = 0x97;

_asm

{

mov eax, NtQuerySystemInformationNo

lea edx, [esp+4]

int 2eh

ret 10h

}

INT 2eH = SVC ;
97h = N

//0x64 openfile; 0xed writefile; 0x29 createprocess

I 系统调用的隐式调用方式

n通过API（Application Program Interface）方式调用

u例 open, write, printf, return等函数都是API

```
file* fp = open("c:/test.txt"); //打开文件
```

```
return 0;
```

u高级语言（C, Vb, Delphi, Java）中使用

u编译时转化为相应的显示调用（INT指令）

隐式调用转化为显示调用

I 编译时隐式调用转化为显示调用（INT指令）

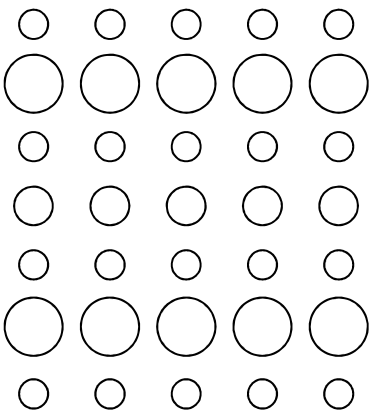



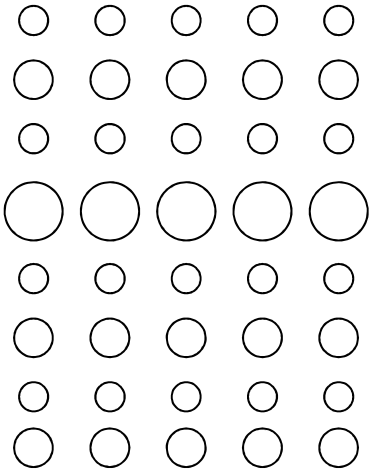

I 例子：printf, exit

```
#include <stdio.h>

int main(void)
{
    printf("Hello World\n");
    exit(0);
}
```

```
• 1 /* shellcode.c */
• 2 void main()
• 3 {
• 4     __asm__ __volatile__("jmp forward;"
• 5                           "backward:"
• 6                           "popl  %esi;"
• 7                           "movl  $4, %eax;"
• 8                           "movl  $1, %ebx;"
• 9                           "movl  %esi, %ecx;"
• 10                          "movl  $12, %edx;"
• 11                          "int   $0x80;"
• 12                          "movl  $1, %eax;"
• 13                          "movl  $0, %ebx;"
• 14                          "int   $0x80;"
• 15                          "forward:"
• 16                          "call  backward;"
• 17                          ".string \"Hello World\\n\";");
• 18 }
```

4→write
1→exit

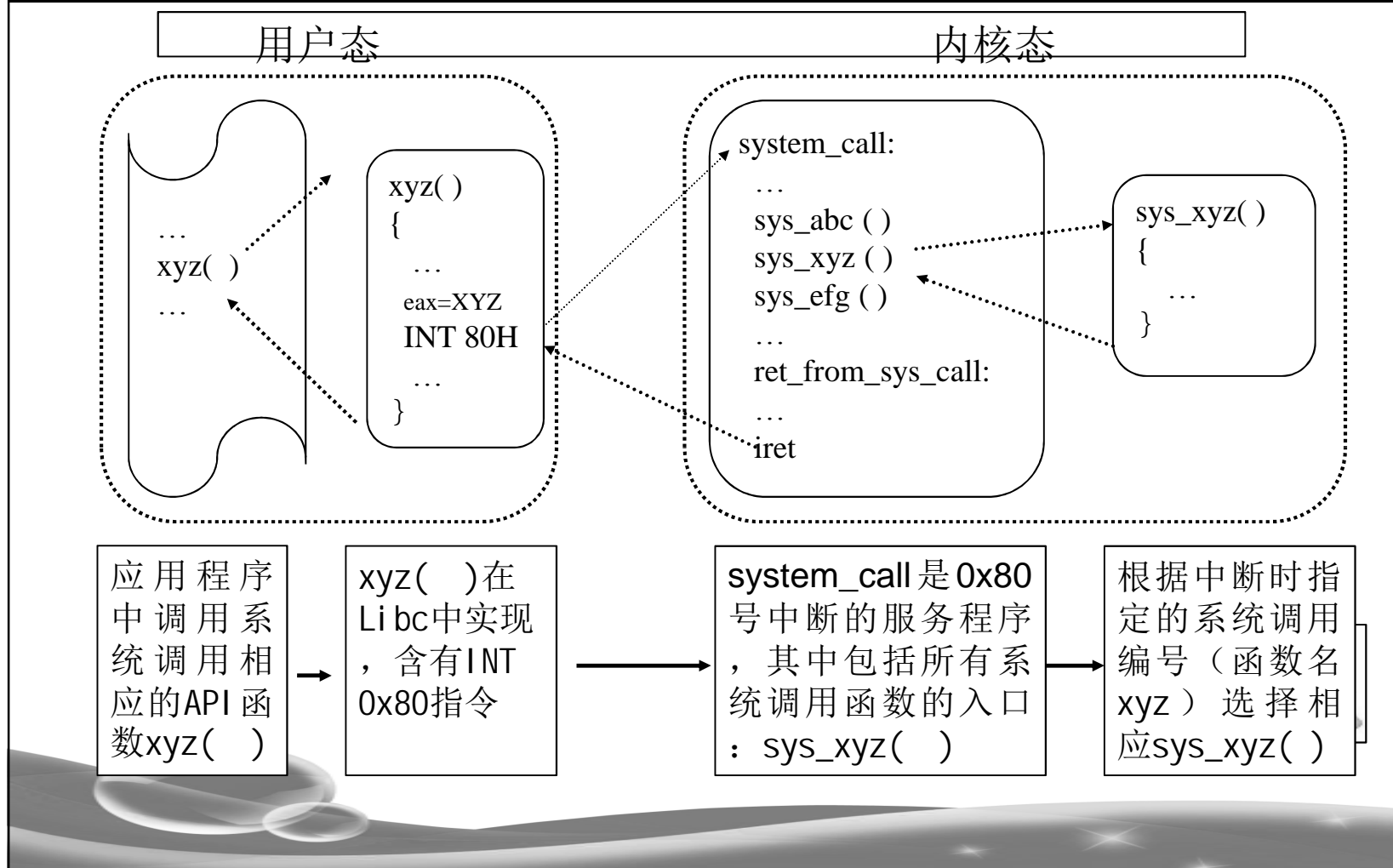
	
  	<h2>5. Linux系统调用的内部机制</h2>
	

I LINUX系统调用的实质

- n 系统调用是Linux内核的出口，给用户程序调用的一组“特殊”接口。
- n 系统调用采用API方式向用户提供，遵循 POSIX标准
- n 系统调用通过软中断(INT 80H)向内核发出服务请求



Linux系统调用的工作原理



80H中断的服务程序system_call()

I system_call()

{

 //.....

 //寄存器%**eax**存放有系统调用编号，

 //以其为索引遍历sys_call_table，查找对应的服务子程序。

 call *SYMBOL_NAME(**sys_call_table**)(%eax, 4)

}



SYS_CALL_TABLE的结构

```
.data
ENTRY(sys_call_table)
.long SYMBOL_NAME(sys_ni_syscall) // 0
.long SYMBOL_NAME(sys_exit)      // 1
.long SYMBOL_NAME(sys_fork)
.long SYMBOL_NAME(sys_read)
.long SYMBOL_NAME(sys_write)
.long SYMBOL_NAME(sys_open)      // 5
.....
```

Ref: /usr/src/linux/arch/i386/kernel/entry.S



系统调用编号的声明

I Linux-source-2.6.38

n./arch/x86/include/asm/unistd_32.h

I 系统调用编号的声明

格式: **#define __NR_CallName ID**

```
#define __NR_perf_event_open 336
#define __NR_recvmmsg 337
#define __NR_fanotify_init 338
#define __NR_fanotify_mark 339
#define __NR_prlimit64 340
#define __NR_mycall 341
#define __NR_addtotal 342
#define __NR_three 343
#ifdef __KERNEL__
```

系统调用函数的声明

I Linux-source-2.6.38

n./arch/x86/kernel/syscall_table_32.S

I 系统调用函数的声明

.long sys_XXXX

注意：1) XXX是函数名，有前缀sys_修饰。

2) 添加位置是末尾，且注意添加顺序。

```
.long sys_rt_tgsigqueueinfo      /* 335 */  
.long sys_perf_event_open  
.long sys_recvmmsg  
.long sys_fanotify_init  
.long sys_fanotify_mark  
.long sys_prlimit64              /* 340 */  
long sys_mycall  
long sys_addtotal  
long sys_three
```



系统调用函数的定义

I 注意格式 **asmlinkage**

n./kernel/sys.c (Linux-source-2.6.38)

```
asmlinkage int sys_mycall(int number)
{
    printk("这是我添加的第一个系统调用");
    return number;
}
asmlinkage int sys_addtotal(int number)
{
    int i=0, enddate=0;
    printk("这是我添加的第二个系统调用");
    while(i<=number)
        enddate+=i++;
    return enddate;
}
asmlinkage int sys_three()
{
    printk("这是我添加的第三个系统调用");
    return 0;
}
```

系统调用函数的调用方法

I 早先的版本

n先声明 (N: 参数个数; FuncName不带sys_前缀)

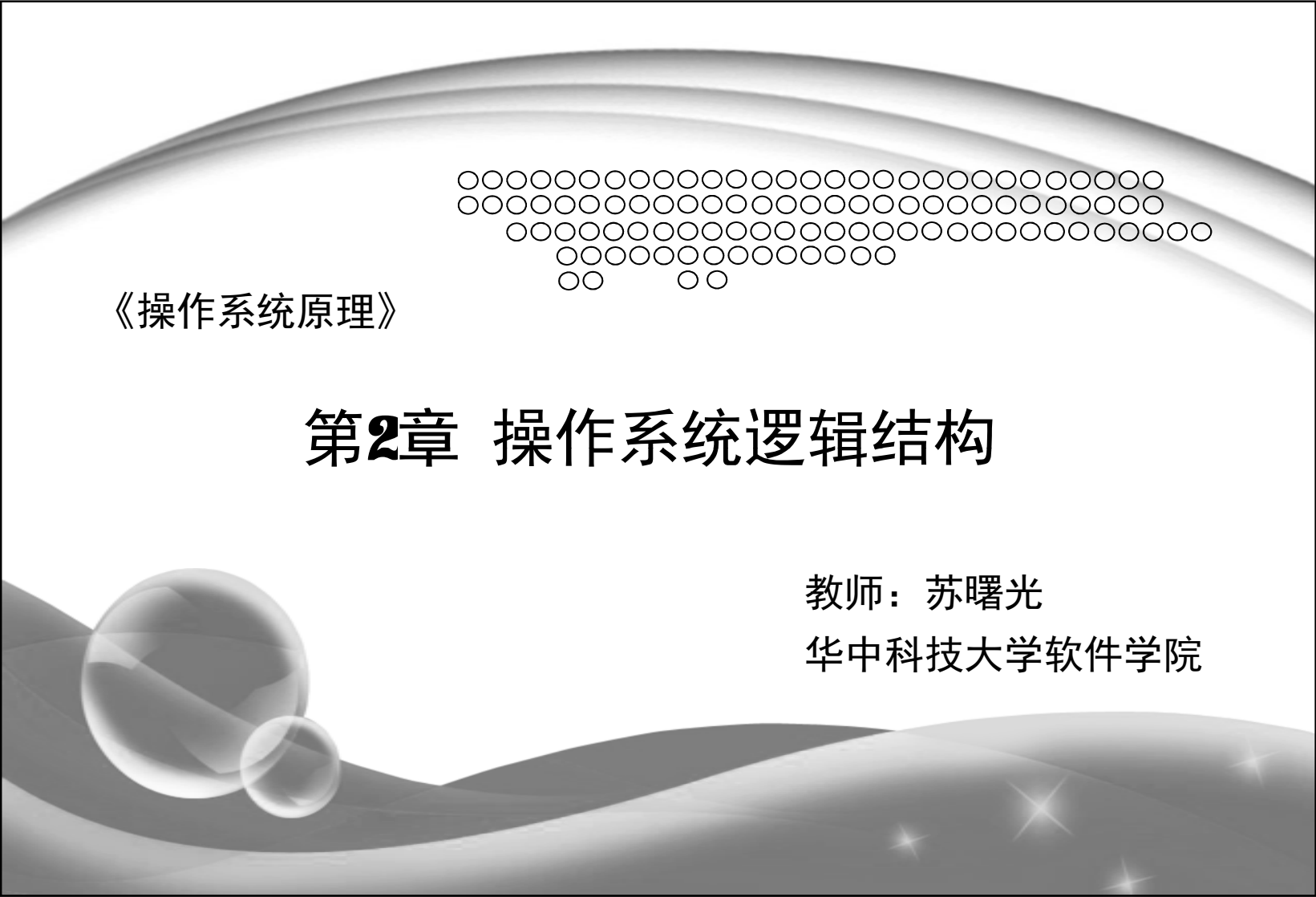
```
#define _syscallN ( type, FuncName ,  
                    type1, arg1,  
                    type2, arg2, .....  
                    typeN, argN )
```

n后调用: type = FuncName (arg1, arg2, ...)

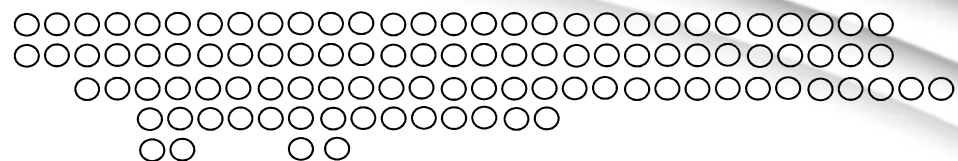
I 较新的版本

n直接调用 (FuncName 不带sys_前缀)

```
type = FuncName(__NR_funcname, arg1, arg2, ...)
```



《操作系统原理》



第2章 操作系统逻辑结构

教师：苏曙光

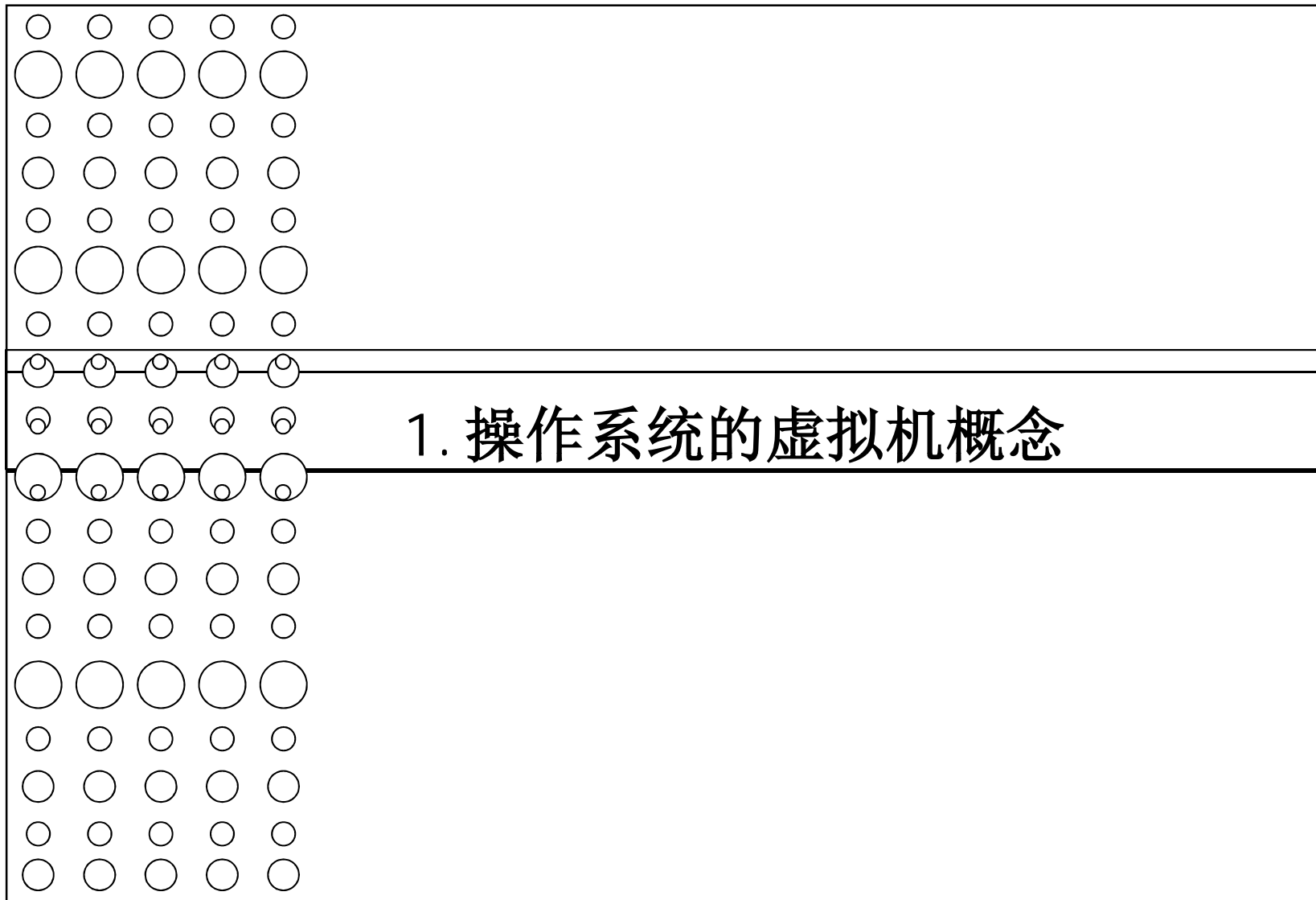
华中科技大学软件学院

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

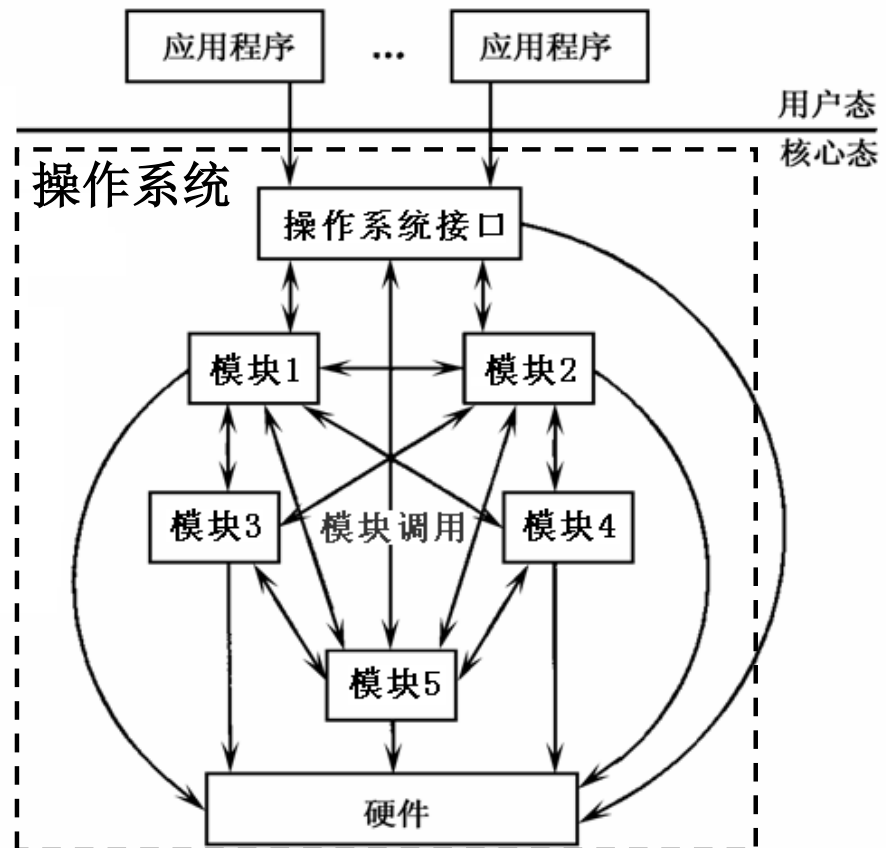
I 逻辑结构的种类

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

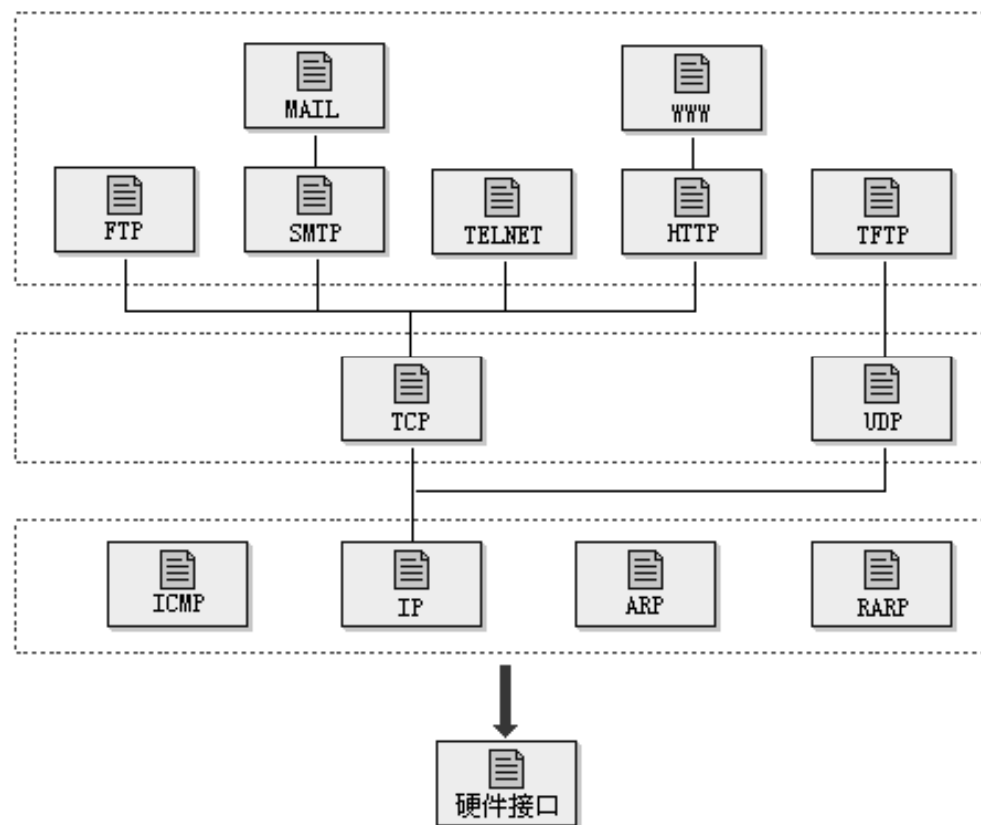
- n 模块设计、编码和调试独立
- n 模块调用自由
- n 模块通信多以全局变量形式完成

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

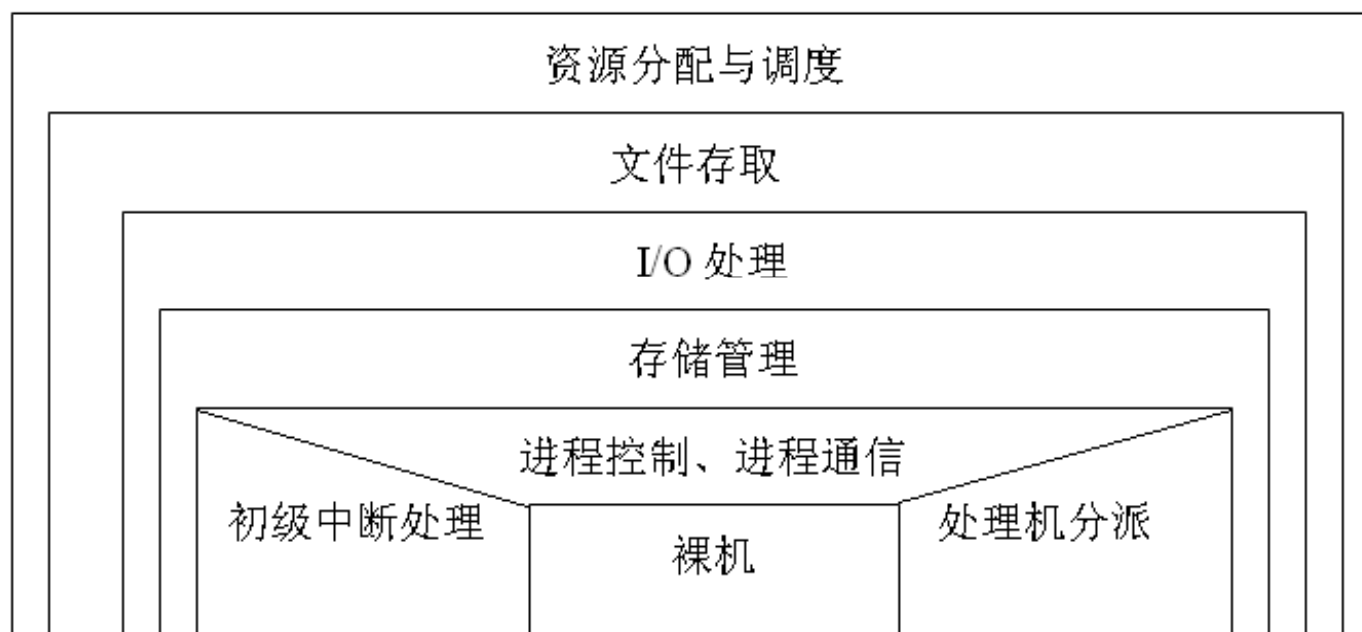
n外部特性——最外层

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

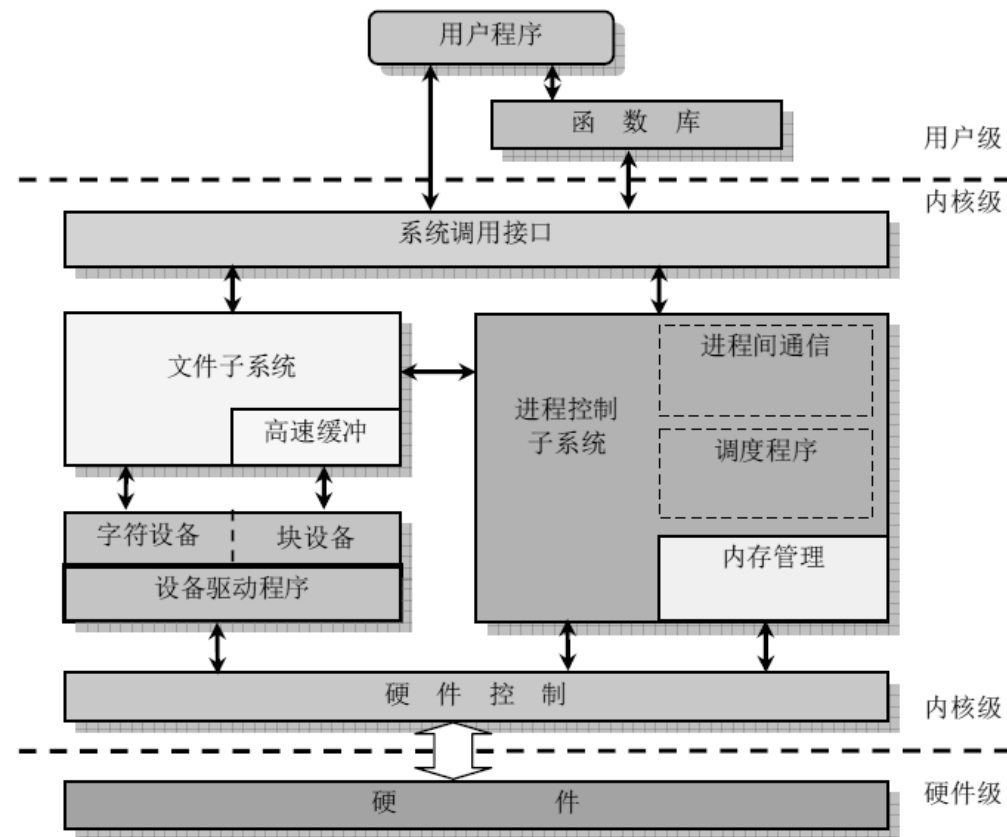
- n应用层：应用程序使用指定的参数值执行系统调用指令(int x80)，使CPU从用户态切换到核心态（调用服务的应用层）
- n服务层：OS根据具体的参数值调用特定的服务程序（执行系统调用的服务层）
- n底层：服务程序根据需要调用底层的支持函数（支持系统调用的底层函数）

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

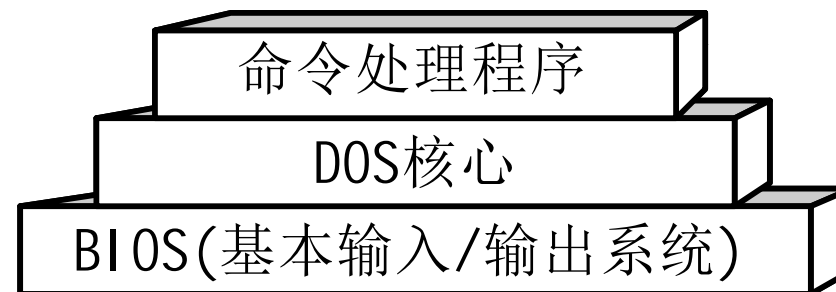
典型操作系统的结构

I MS DOS

nBIOS: Basic Input/Output System

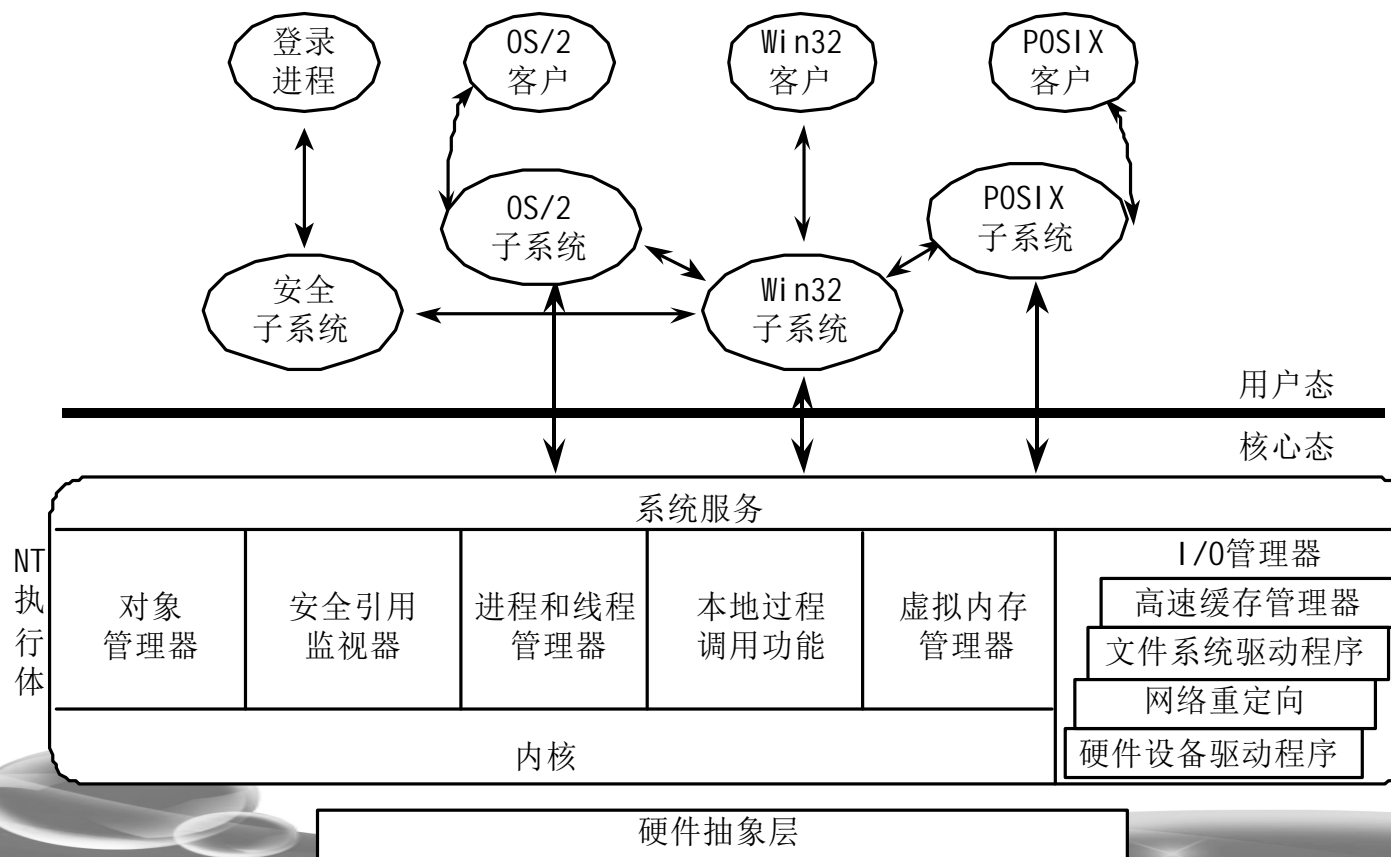
nDOS核心: 内存、文件管理、字符设备和输入/输出

n命令处理程序: 对用户命令进行分析和执行

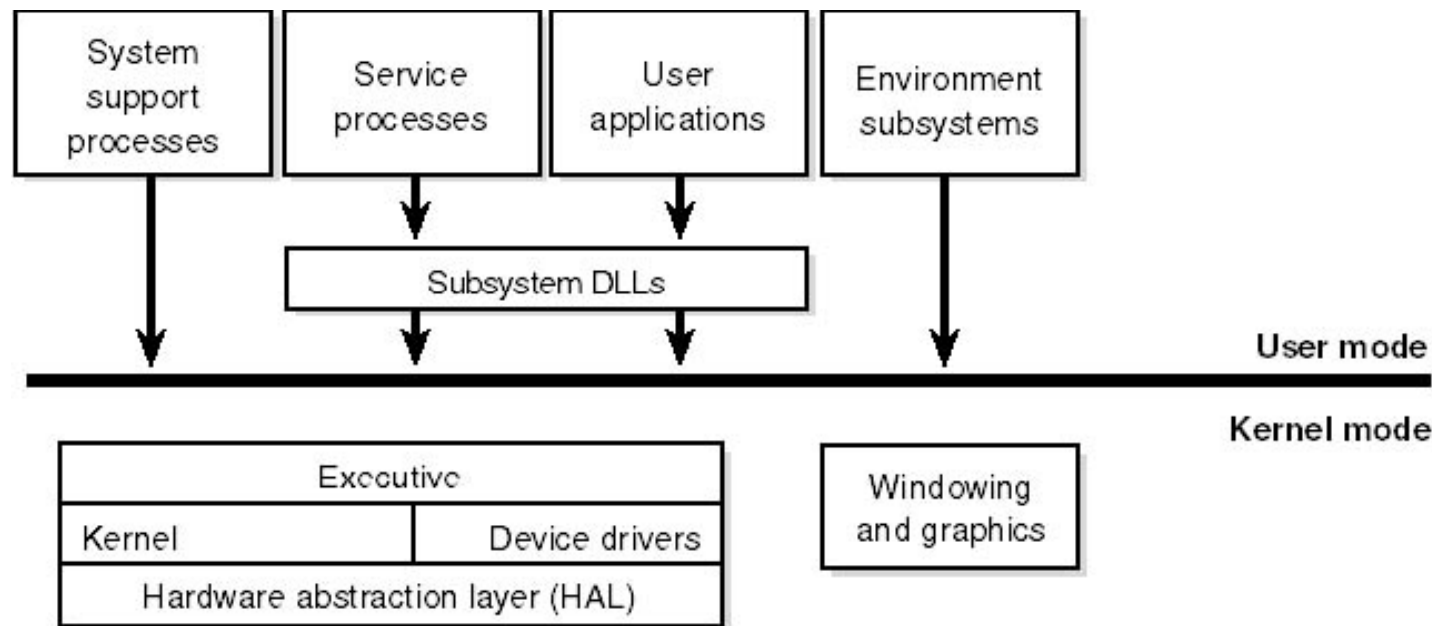


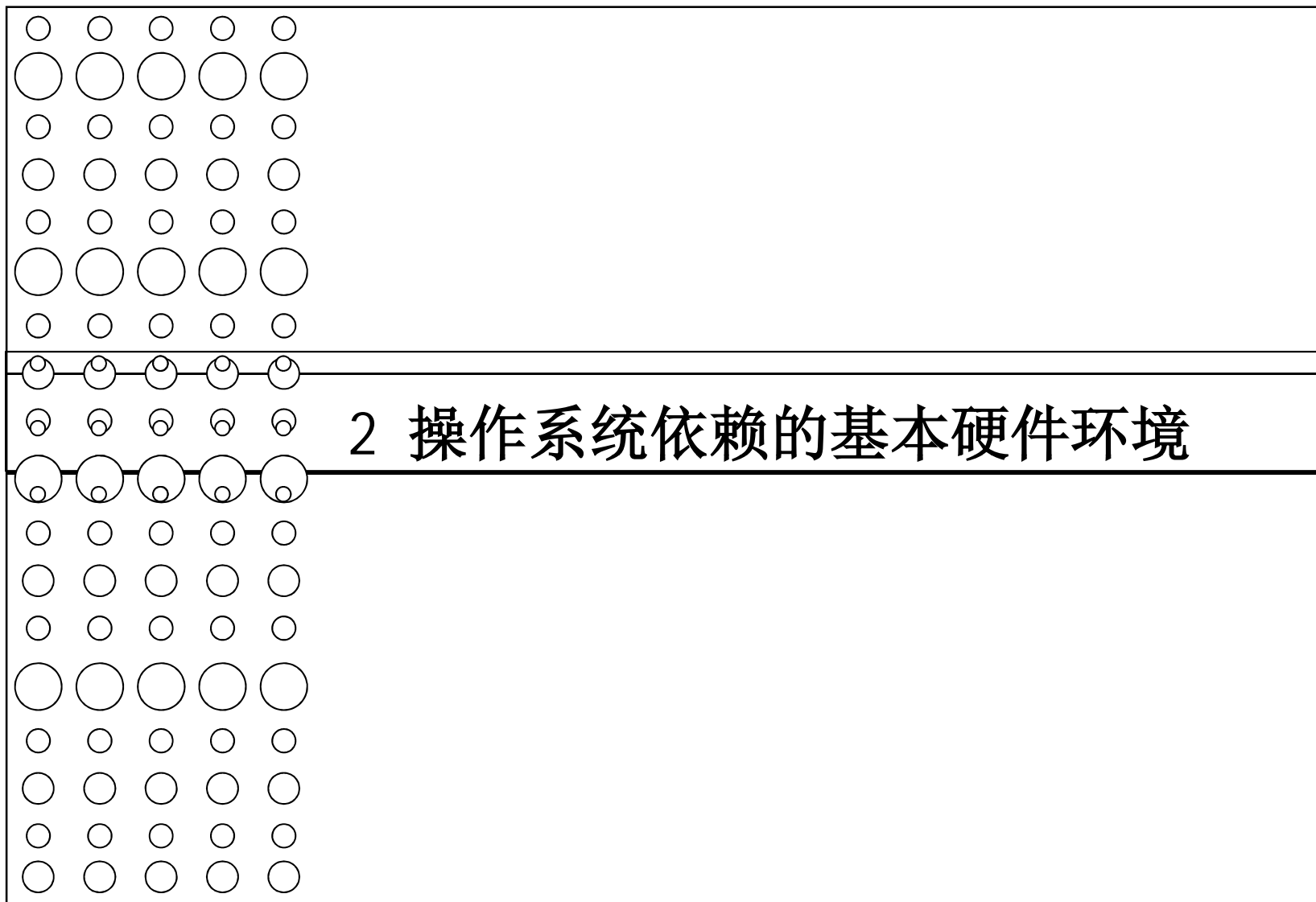
典型操作系统的结构

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 = %I64d\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

l 特权指令仅能在核态下被**OS**使用

l 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

Ⅰ 理想存储器：速度快，容量大，成本低

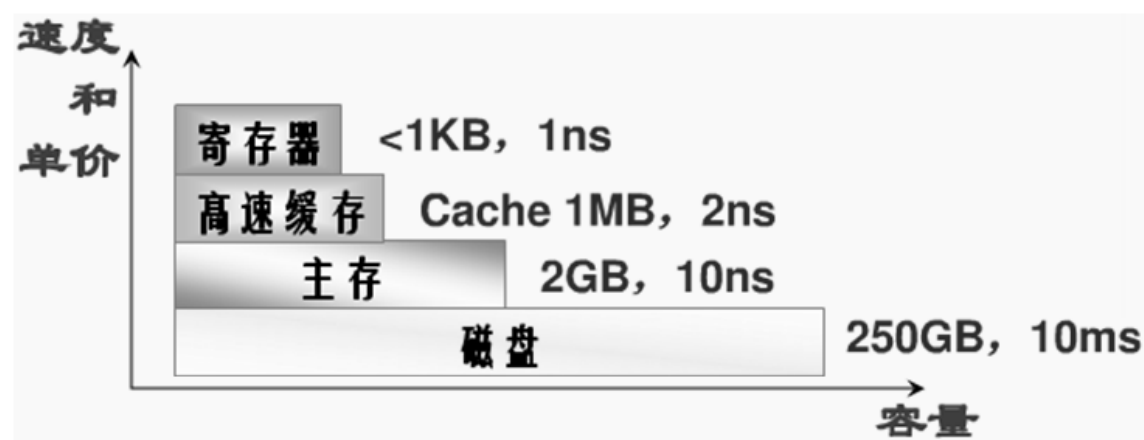
Ⅰ 分级存储系统

n 寄存器

n 高速缓存（CACHE）

n 主存

n 磁盘



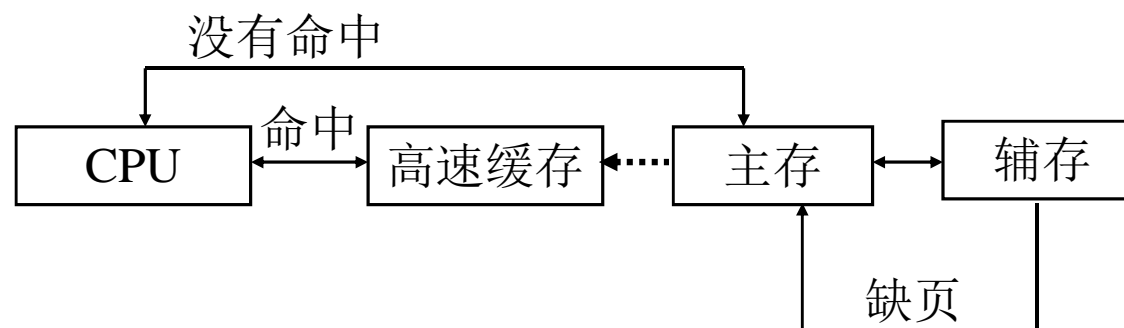
分级存储系统的工作原理

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。

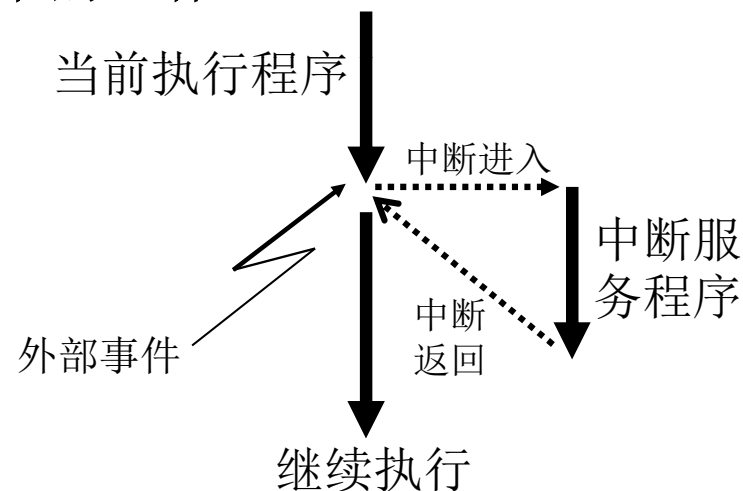
nCPU收到外部信号（中断信号）后，停止当前工作，转去处理该外部事件，处理完毕后回到原来工作的中断处（断点）继续原来的工作。

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）



《操作系统原理》

第1章 操作系统概述

教师：苏曙光

华中科技大学软件学院

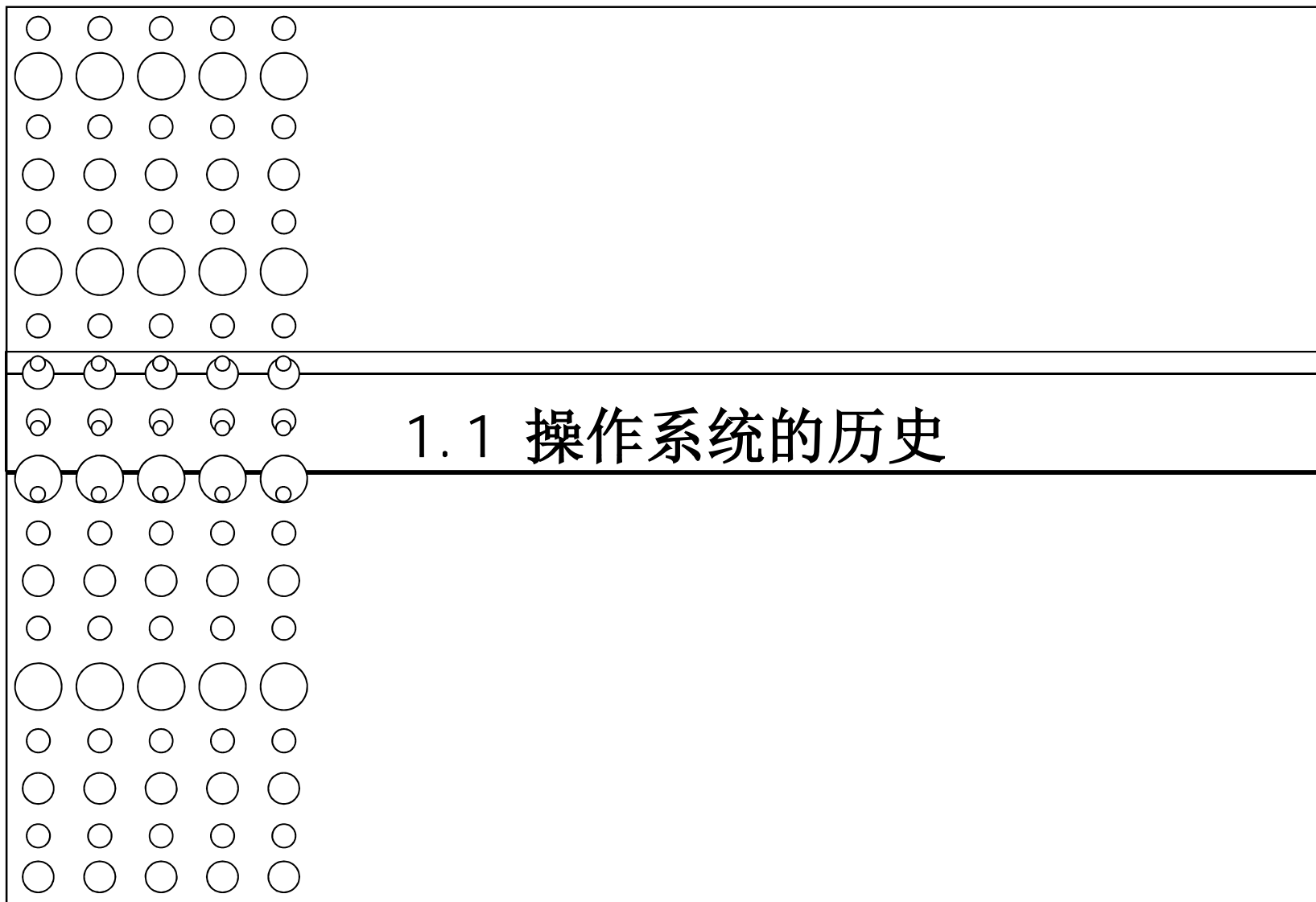
2015年3月-5月

I 主要内容

- n 操作系统的历史
- n 操作系统的定义
- n 操作系统的功能
- n 操作系统的特性

I 重点

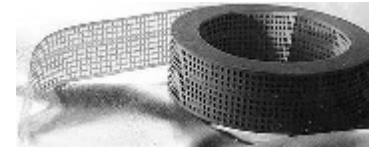
- n 操作系统4个发展阶段的典型特点
- n 多道程序设计技术
- n 分时的概念
- n 理解操作系统的特性



1.手工操作（没有操作系统）

I 电子管时代【1946—1955】

n IBM 701型计算机（1952年，IBM）



I 结构特点

n 硬件：电子管、接线面板（按钮/开关），卡片/纸带

n 程序：二进制目标程序，卡片/纸带打孔

I 使用特点

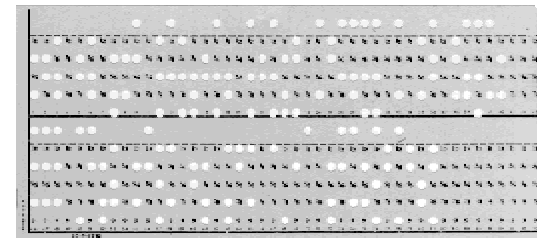
n 上机：编程（打孔），程序员操作机器，预约

n 程序启动与结束：手工处理

I 缺点

n 用户独占，资源利用效率低

n 缺少交互



I 计算机硬件发展的四个典型阶段

n 电子管时代 【1946—1955】

n 晶体管时代 【1955—1965】

n 集成电路时代 【1965—1980】

n 大规模集成电路时代 【1980—至今】

I 操作系统发展的四个典型阶段

n 手工操作（无操作系统） 40年代到50年代早期

n 单道批处理系统 50年代

n 多道批处理系统 60年代初

n 分时系统 60年代中

2.单道批处理系统

I 背景

n 晶体管时代【1955—1965】

n 1955年，IBM 推出第一台晶体管计算机IBM 608型。

I 工作过程

n 用户（程序员）将作业（纸带）交给机房（操作员）

n 操作员将**多个作业**输入到磁盘形成作业队列

n 监控程序依次自动处理磁盘中每个作业：装入—运行—撤出

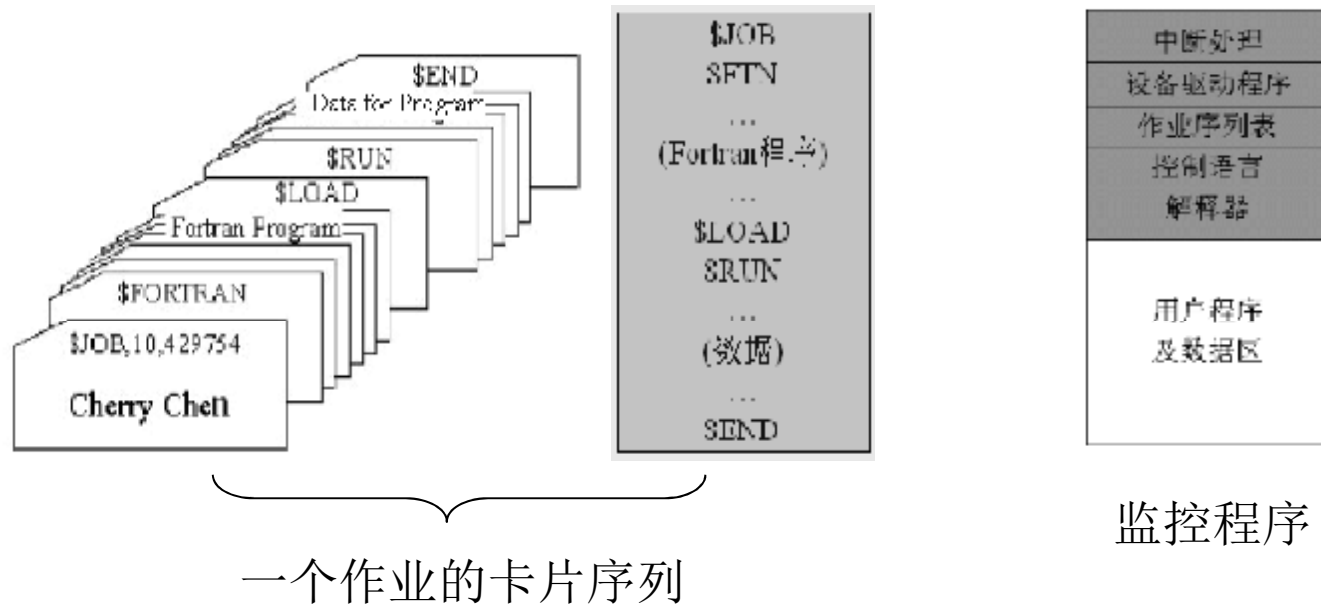
n 运行完毕，通知用户取结果

I 特点

n 系统每次处理一批用户程序，所有程序在监控程序控制下依次被自动装入，运行，完成后被撤出，然后处理下一程序。

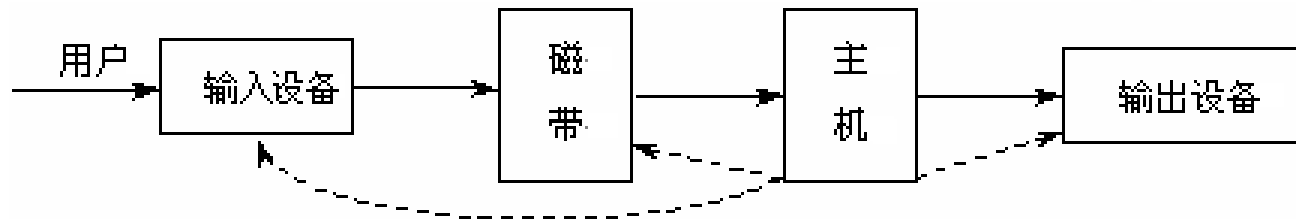
n 概括：一批，自动，串行（单道）

单道批处理系统中的作业



单道批处理的两种实现方式

I 方式1：联机批处理

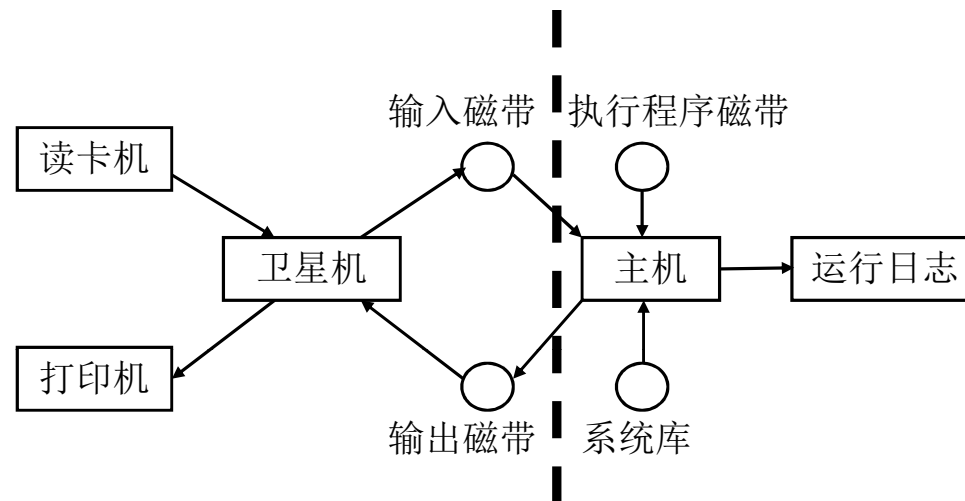


n特点：主机控制输入/输出

n缺点：系统效率低

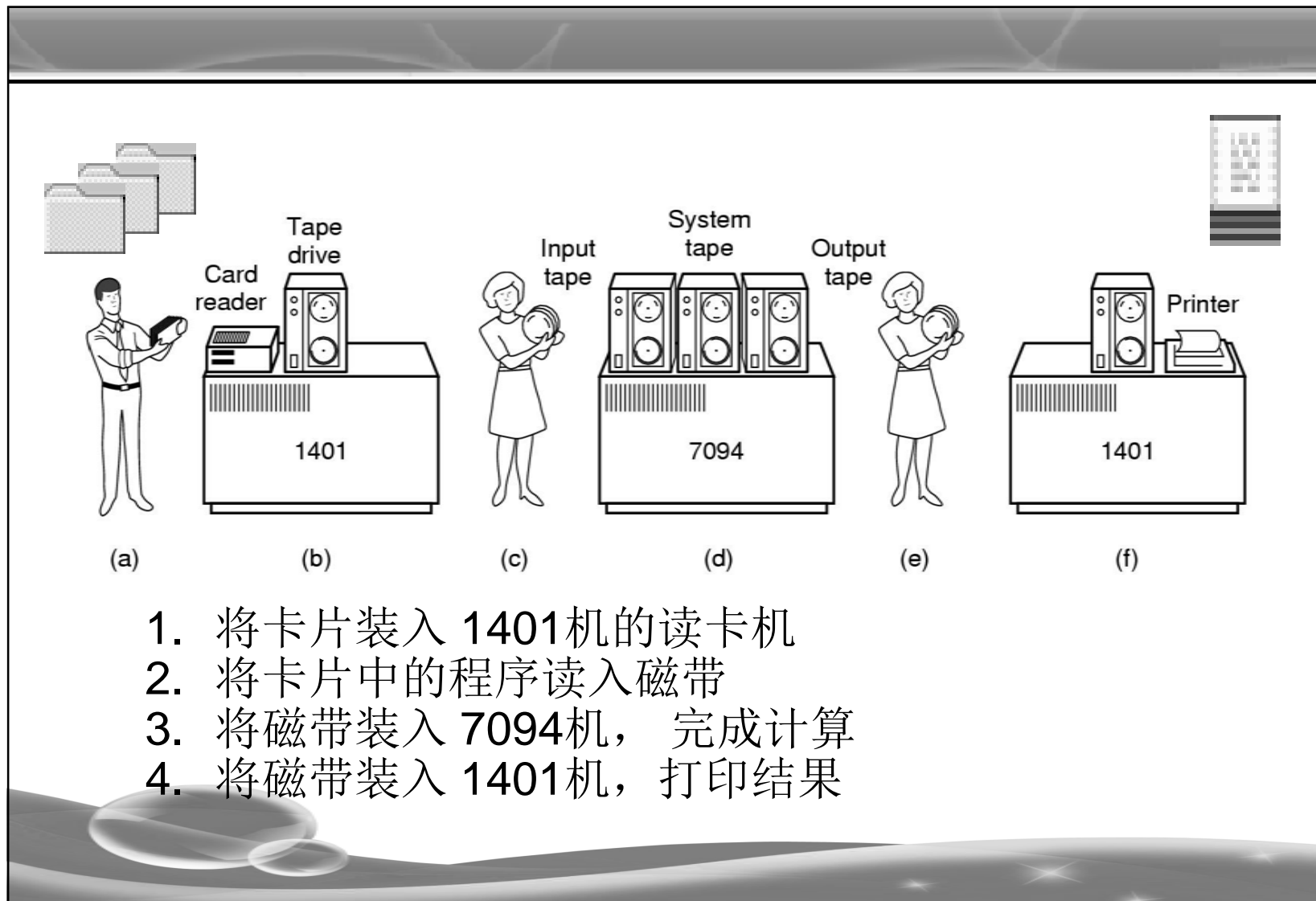
批处理的两种实现方式

I 方式2：脱机批处理

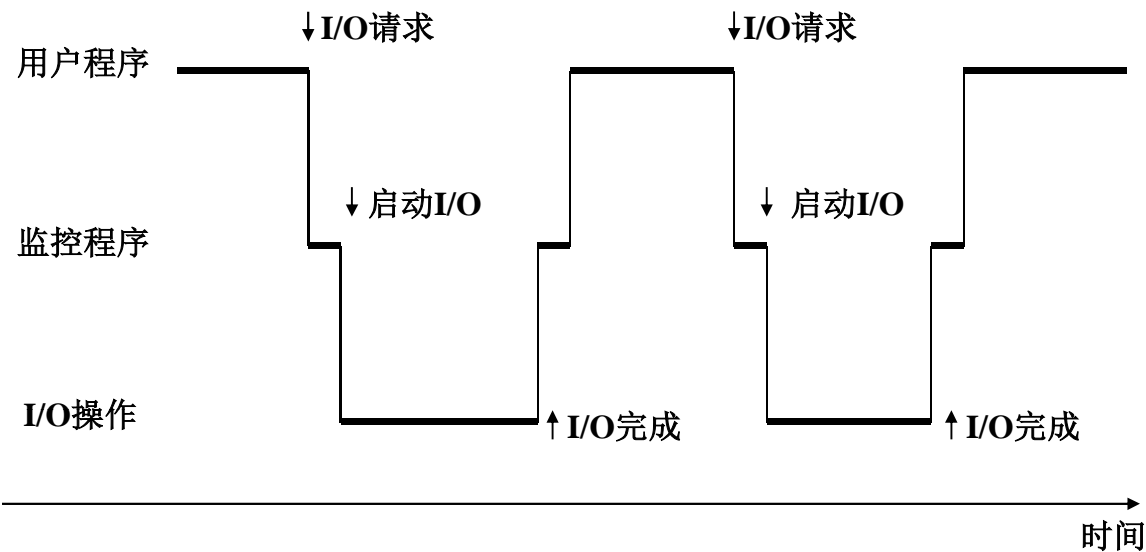


n特点：卫星机控制输入/输出

n优点：系统效率高



单道批处理系统中CPU利用情况



红色：CPU在工作 蓝色：外设在工作

现象： 外设工作时CPU空闲， CPU工作时外设空闲。

结论：CPU和外设效率低。

3.多道批处理系统

┆ 多道程序设计技术

n 在内存中存放多道程序, 当某道程序因为某种原因 (例如请求I/O时) 不能继续运行时, 监控程序便调度另一程序投入运行。这样可以使CPU尽量处于忙碌状态, 提高系统效率。

┆ 多道批处理系统

n 采用多道程序设计技术实现的批处理系统称为多道批处理系统。

I 多道批处理系统的设计目的

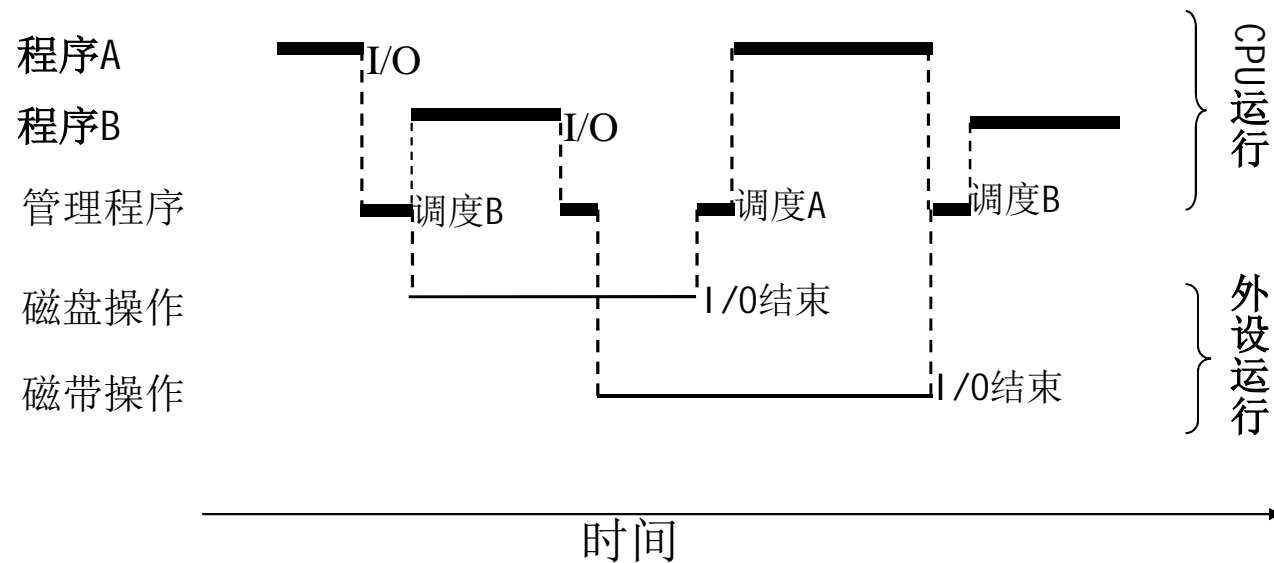
n 提高系统的利用率（或吞吐量）

u CPU与外设并行

u 外设之间也并行

多道程序相互穿插的运行过程

I 两个程序的例子



A, B两道程序相互穿插地运行, 使CPU和外设都尽量忙碌!
思考: A,B都是3分钟运行, 2分钟I/O。求CPU可能的最大利用率?

多道批处理系统的特点

- I 多道

- n 内存同时存放多道程序

- I 并行

- n 宏观上

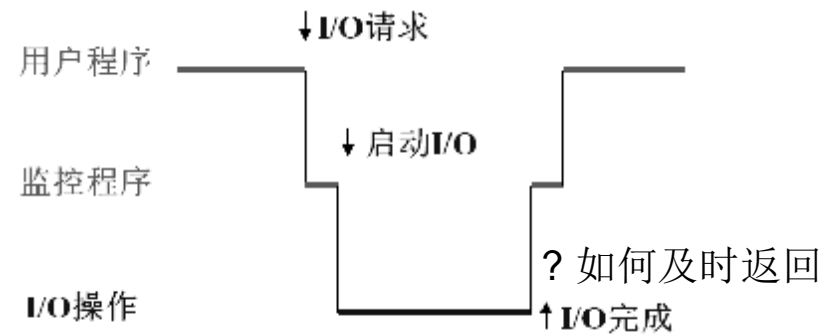
- I 串行

- n 微观上

多道批处理系统的缺点

I 缺点

- n 作业处理时间长
- n 交互能力差
- n 运行过程不确定



I 60年代硬件的两个重大进展

└ 中断技术

└ CPU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。

└ 通道技术

└ 专门处理外设与内存之间的数据传输的处理机。

4.分时操作系统

I 背景

n 事务性任务和程序的涌现

u 交互性高

u 响应快速

n 要求：多任务多用户

I 实用化的分时操作系统

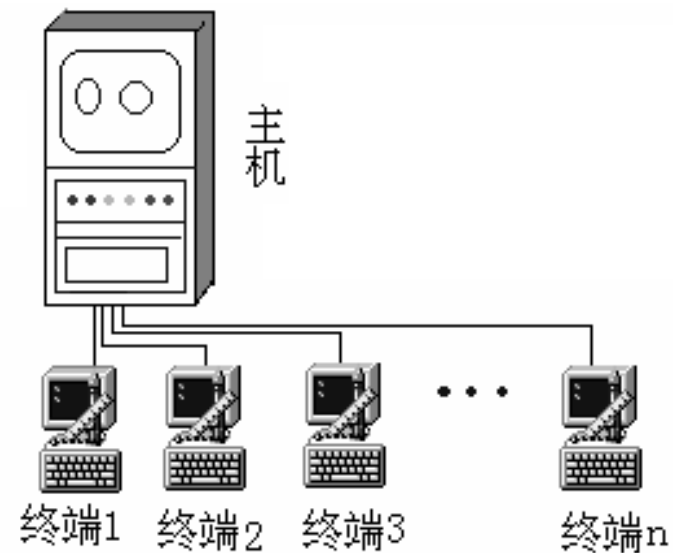
I 硬件水平

n 中断技术

n 大规模集成电路

n 多终端计算机

u 主机采用分时技术轮流为每个终端服务，每个终端都感觉到是“独占”主机！



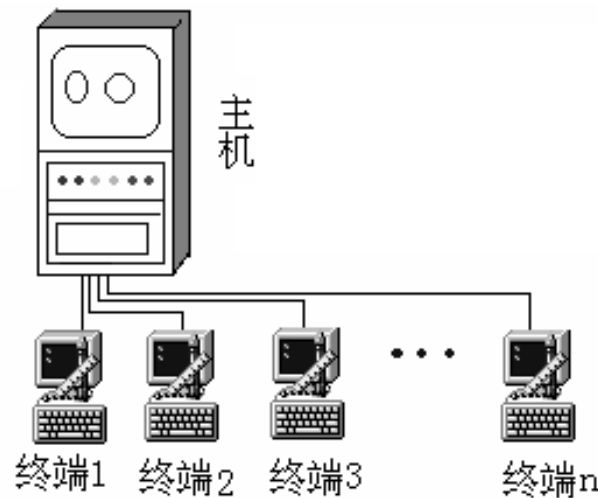
分时技术

I 概念

- n 主机以很短的“时间片”为单位，把CPU循环地分配给每个作业（终端/用户）使用。
- n 即使某个作业在一个时间片内没有完成，主机会暂停该作业，把CPU让给下一个作业使用。
- n CPU在所有作业中被循环使用，直到该作业运行完。

I 特点

- n 在一个“多用户”系统中，每个用户都能得到系统的响应。
- n 分时系统由“主机”和“终端”组成。



I 分时系统的特点

- n 多路调制性

- u 多用户联机使用同一台计算机

- n 独占性

- u 用户感觉独占计算机

- n 交互性

- u 及时响应用户的请求

I 分时操作系统的实例

- n Linux: 100ms或可设置

- n Windows: ?

- n CTSS (Compatible Time Sharing System)

- u MIT开发 200ms

大型分时系统的实践：Multics 项目

- | 1962年，ARPA（国防部高级研究计划局）支持：MIT、BELL和G.E.
- | 开发一种“公用计算服务系统”
- | 希望能够同时支持整个波士顿地区所有的分时用户
- | 称作Multics (MULTiplexed Information and Computing Service)

I Multics设计目标:

- n 使用便利的远程终端通过电话线接入计算机主机
- n 连续工作（无关机）
- n 可变的配置能力，无需用户程序重新配置
- n 高可靠的大型文件系统
- n 大容量的用户信息共享
- n 存储和构造层次化信息结构的能力
- n 支持从数字运算到分时系统各种应用
- n 多种程序设计环境和人机界面
- n 允许随技术的发展，而不断进化系统

- | **Multics**正式研制开始于**1965**年
 - n**1967**年**12**月完成第一阶段的任务
- | **1969**年**4**月**BELL**，**GE**公司先后退出
- | **1969**年**10**月开始在**MIT**投入使用，销售了几十套
- | 多数**Multics**系统在九十年代中陆续被关闭
- | 最后一个**Multics**机系统在加拿大国防部于**2000**年**10**月**30**日**17:08**关闭

- | **Multics**引入了许多现代操作系统的重要概念
- | **Multics**是第一个采用“层次化文件系统”
 - n Hierarchical File System概念的系统
 - n 被Windows, MACOS, DOS, UNIX, Linux等广泛采用
- | 多语言支持能力
 - n 支持EPL、EPLBSA、PL/I、ALM、COBOL、FORTRAN、BCPL等等



Multics的最大贡献

- 丨 Ken Thompson和Dennis Ritchie吸收Multics设计思想和新概念，在PDP-7计算机上实现了UNIX。
 - 在UNIX中，许多命令，控制变量，shell文本等和Multics一样
 - UNIX这个名称也是从Multics的发音中演化而来
- 丨 Dennis在Multics中的工作是实现BCPL语言，后来把BCPL语言改造为C语言，用C重写UNIX。
- 丨 1970年，新系统被命名为UNIX，自此UNIX诞生。

UNIX

I 第一个实用化的分时操作系统

- n 世界上第一个真正体现了操作系统领域各种先进概念和技术的操作系统：UNIX

I 革新和创造

- n UNIX实现了操作系统的可移植性
- n 与计算机硬件无关性
- n 引进了“特殊文件”（**Special File**）的概念,第一次把各种外部设备也看作文件,真正实现了对所有外部设备的统一管理

I 思考?

n多道批处理和分时系统都有多个作业同时在内存中运行，**CPU**会在作业间进行切换。请问这两种切换有什么区别？

I 操作系统的进一步发展（分时系统的衍化）

n 微机操作系统（PC机）

n 多处理机操作系统

n 网络操作系统

n 分布式操作系统

n 实时操作系统

n 嵌入式操作系统

个人计算机时代的微机操作系统

I CP/M 操作系统

- n 随着大规模集成电路发展，进入PC机时代。
- n 1973年Gary Kildall，设计CP/M操作系统
 - u Control Program/Microprocessor
- n CP/M有较好的层次结构。它的BIOS把操作系统的其他模块与硬件配置分隔开，可移植性好
- n 较好的可适应性和易学易用性
- n 1980年初，CP/M成为流行最广的8位操作系统之一。

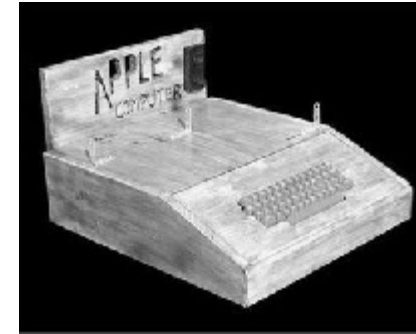
个人计算机时代的微机操作系统

I 苹果(1976年 乔布斯,沃兹)

nLi sa机

nAppl eIII机

nMaci ntosh和MAC OS



u(1984年)Maci ntosh是配有图形界面操作系统MAC OS和鼠标的新型个人计算机

uMac OS是运行Maci ntosh系列电脑上的操作系统。

uMac OS是首个在商用领域成功的图形用户界面。



个人计算机时代的微机操作系统

I 微软的MS DOS

- n 1980年IBM公司决定尽快生产微型计算机，以应付挑战
- n IBM试图洽谈CP/M操作系统失败，机遇落到了微软公司
- n 微软通过经销西雅图计算机产品公司的QDOS操作系统转卖给IBM。
- n QDOS → MS DOS

个人计算机时代的微机操作系统

l 微软Windows操作系统

n 1983年10月，多家PC机厂家图形界面产品上市

n 比尔·盖茨1983年11月宣布将推出Windows操作系统

n 直到1985年11月20日，Windows 1.0才正式上市

n 1992年4月，推出Windows 3.1

n 1993年5月，发表Windows NT

l Windows 95, CE, 98, 2000, XP, Server, Win7...

l Windows占PC OS的90%以上，“微软 = 垄断”

实时操作系统

I 产生背景

n 实时事务：军事，工业控制，智能仪器等

n 要求：某些任务要优先紧急处理

I 特点

n 强调作业完成的时限

I 必须限时完成：硬实时系统

- n 火炮控制系统
- n 航空航天
- n 制导系统
- n 目标识别和跟踪
- n 工业控制
- n 汽车电子系统
- n



近地报警系统是一个实时系统

Ground Proximity Warning System

I 尽可能快完成：软实时系统

- n 网络视频
- n 互动网游
- n 广播
- n 通讯
- n



实时操作系统

I 产生背景

n 实时事务：军事，工业控制，智能仪器等

n 要求：某些任务要优先紧急处理

I 特点

n 强调作业完成的时限

u 必须限时完成：硬实时系统

u 尽可能快完成：软实时系统

n 强调可靠性

嵌入式操作系统

- | 嵌入式操作系统≈实时操作系统
- | 嵌入式操作系统多用于嵌入式系统
- | 嵌入式系统
 - u 软硬件可以裁剪，软硬件一体化的系统。

- | 典型嵌入式操作系统

- n Linux
- n ucOS
- n ucLinux
- n vxWorks
- n WinCE
- n Symbian
- n



网络操作系统

I 网络操作系统

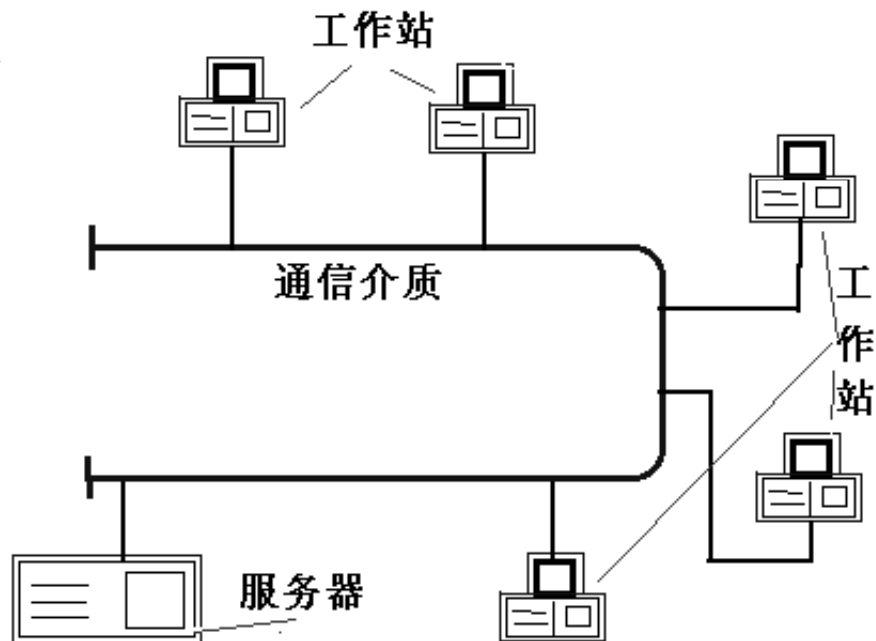
n 普通操作系统 + 网络通信 + 网络服务。

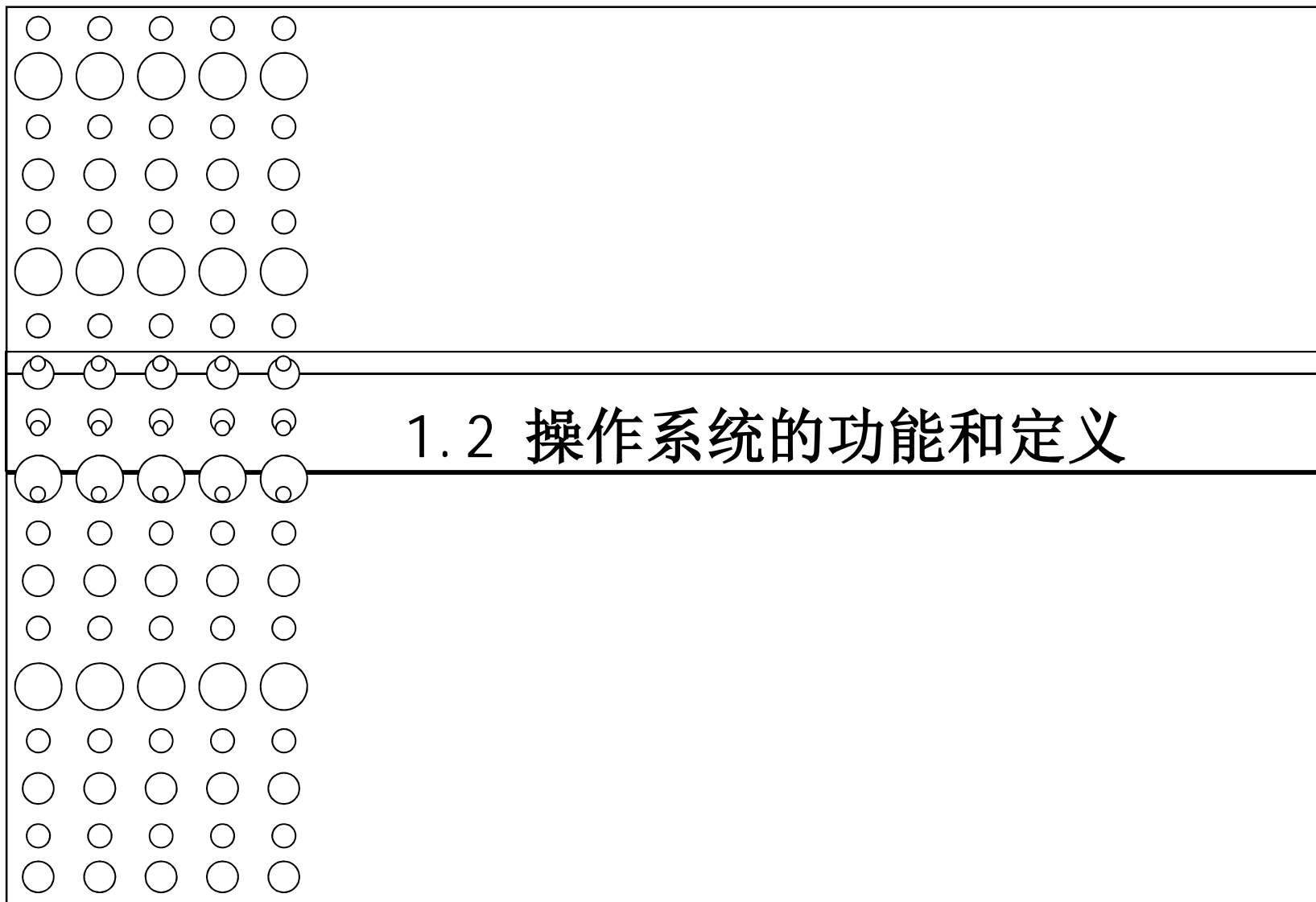
n UNIX/ LINUX/ WINDOWS 。

I 网络操作系统功能

n 透明存取

n 存取控制





I 操作系统的功能一

n 进程管理

- u CPU管理

- u 处理机管理

n 具体子功能

- u 进程控制：创建，暂停，唤醒，撤销

- u 进程调度：调度策略，优先级

- u 进程通信：进程间通信

I 操作系统的功能二

- n 存储管理

- n 作用

 - u 内存分配

 - u 内存共享

 - u 内存保护

 - u 虚拟内存

I 操作系统的功能三

n 设备管理

- u 设备无关性
- u 设备的传输控制
- u 设备的驱动

I 操作系统的功能四

n 文件管理：文件和目录的管理

- u 存储空间管理
- u 文件的操作
- u 目录的操作
- u 文件和目录的存取权限管理

I

操作系统的定义

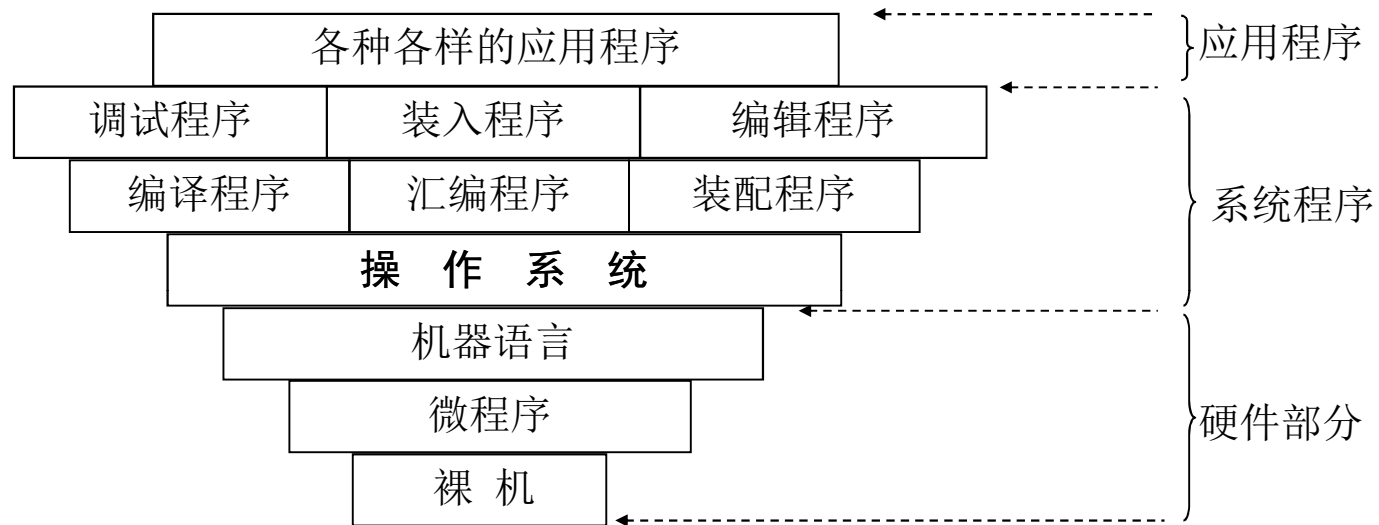
I 操作系统是一个大型系统程序

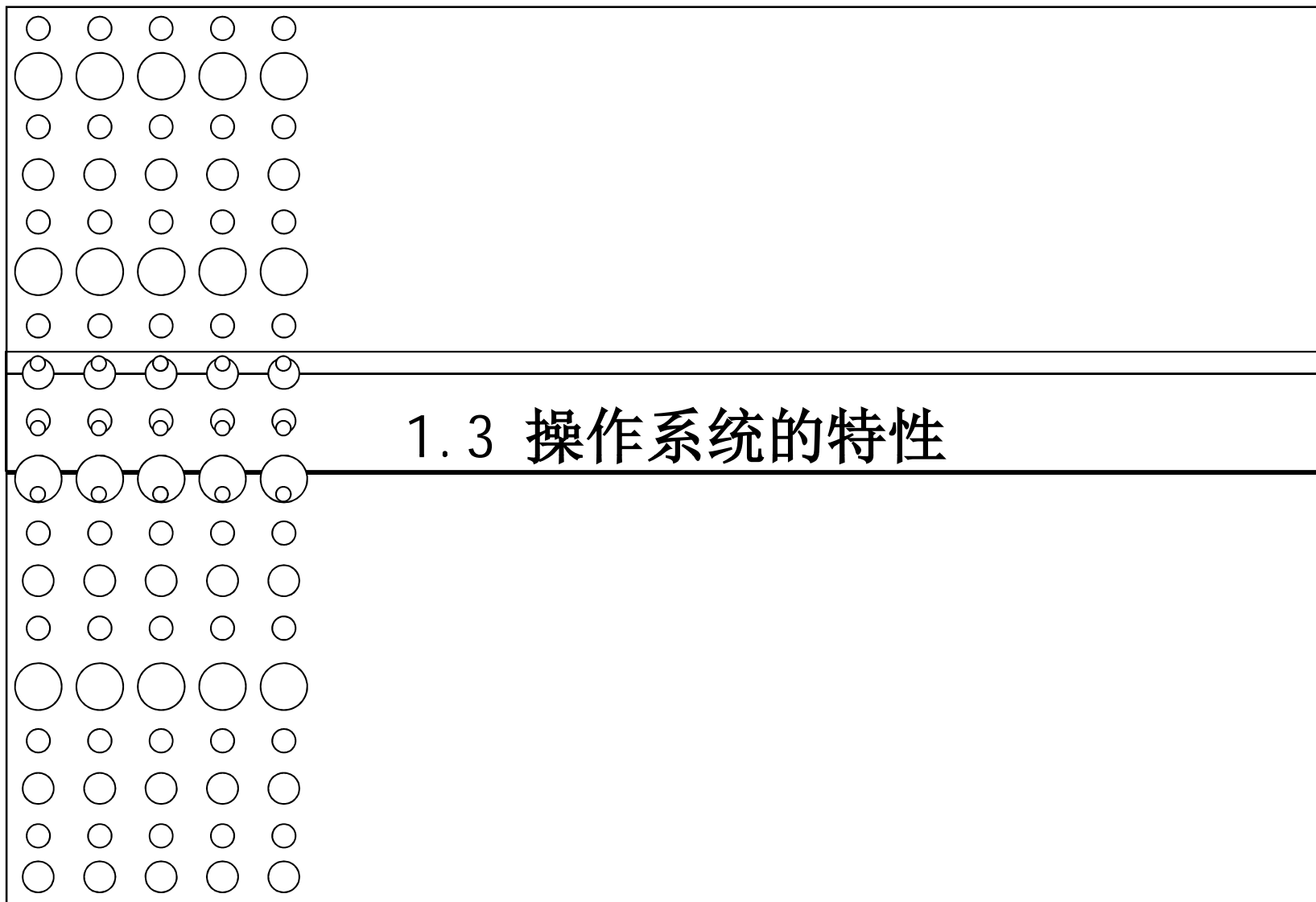
- n 提供用户接口，便利用户使用和控制计算机；
- n 负责计算机的全部软、硬件资源的分配与调度；控制与协调并发活动；实现信息的存取与保护。

I 简而言之

- n 为用户提供友好的接口
- n 管理并调度系统资源；

I 操作系统的地位





I 操作系统的特性

n 并发性

u 同时处理多个任务的能力

n 共享性

u 为多个并发任务提供资源共享

n 不确定性

u 具有处理随机事件的能力

p 中断处理的能力…

丨 操作系统的评价指标

丨 吞吐率

n在单位时间内处理信息的能力。

丨 响应能力

n从接收数据到输出结果的时间间隔。

丨 资源利用率

n设备使用的频度

丨 可移植性

丨 改变硬件环境仍能正常工作的能力。

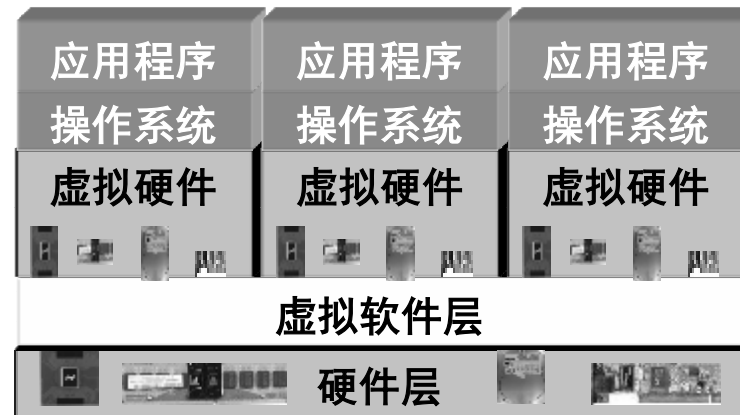
丨 可靠性

n发现、诊断和恢复系统故障的能力。

虚拟化操作系统和虚拟化技术



传统架构



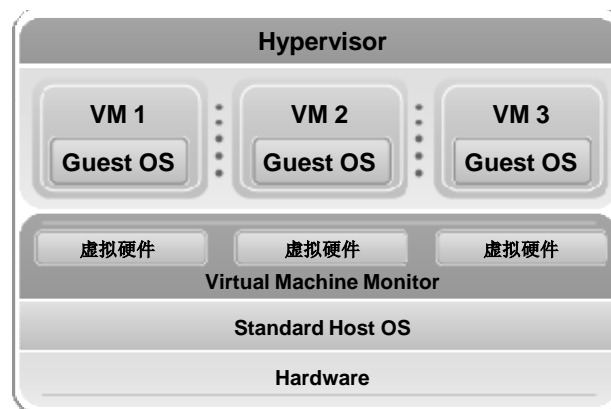
虚拟化架构

虚拟化的结果：一台服务器当N台服务器来使用

虚拟化技术

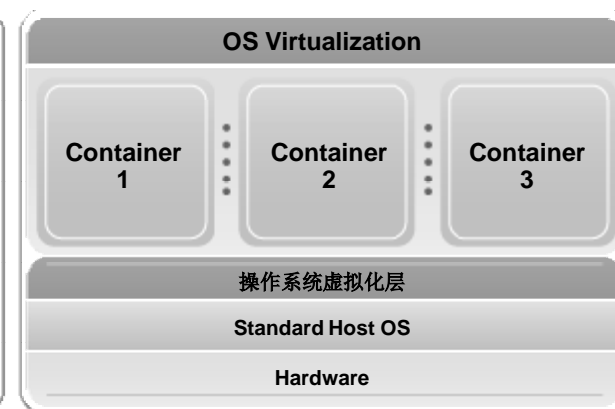
硬件虚拟化 (Hypervisors)

- 虚拟硬件访问
- 创建多个虚拟硬件实例
- 宿主OS及每个Guest OS为完整OS
- **Parallels Server, VMware ESX, MS Hyper-V**



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
- 物理服务器拥有单个、标准的OS内核
- **Parallels Virtuozzo Containers, Sun Solaris Containers, OpenVZ**



I 思考

n 自己总结Windows和Linux的特点和比较?

n 中断技术: 发生中断后, 系统的响应过程?

n 一个简单的应用程序 “hello world” 从开始运行到结束的整个过程中, 操作系统提供了什么样的支持?

n Bai du/Google