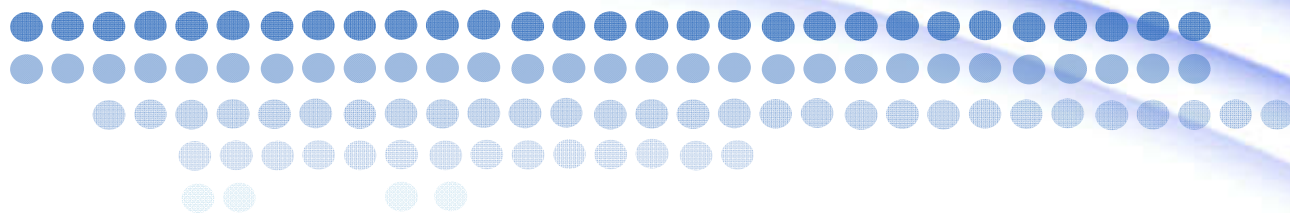


# 2021级期末考试复习版本



《操作系统原理》

## 第3章 用户界面

教师：邹德清，李珍，苏曙光

华中科技大学网安学院

2023年10月-2024年01月

## ● 主要内容

- 用户环境
- 操作界面
- 系统调用

## ● 重点

- 批处理和Shell脚本编程
- 系统调用机制



## 3.1 用户环境和构造

# 用户环境和构造

- 用户环境

- 用户工作的软件和硬件环境。

- ◆ 桌面环境

- ◆ 命令行环境

- 用户环境构造

- 按照用户要求和硬件特性安装和配置操作系统。

- ◆ 提供操作命令和界面

- ◆ 提供系统用户手册





## 3.2 操作系统用户界面

# 用户界面（User Interface）

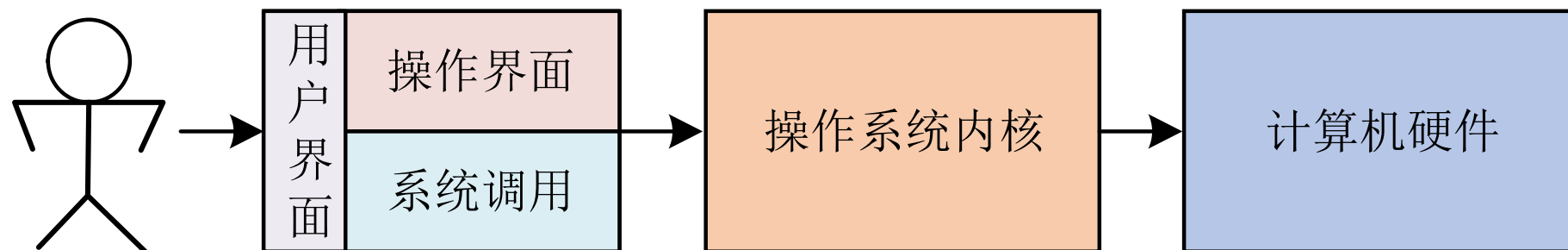
- 用户界面的定义

- 操作系统提供给用户控制计算机的机制（用户接口）

- 用户界面的类型

- 操作界面

- 系统调用（System Call，系统功能调用，程序界面）





## 3.3 操作界面

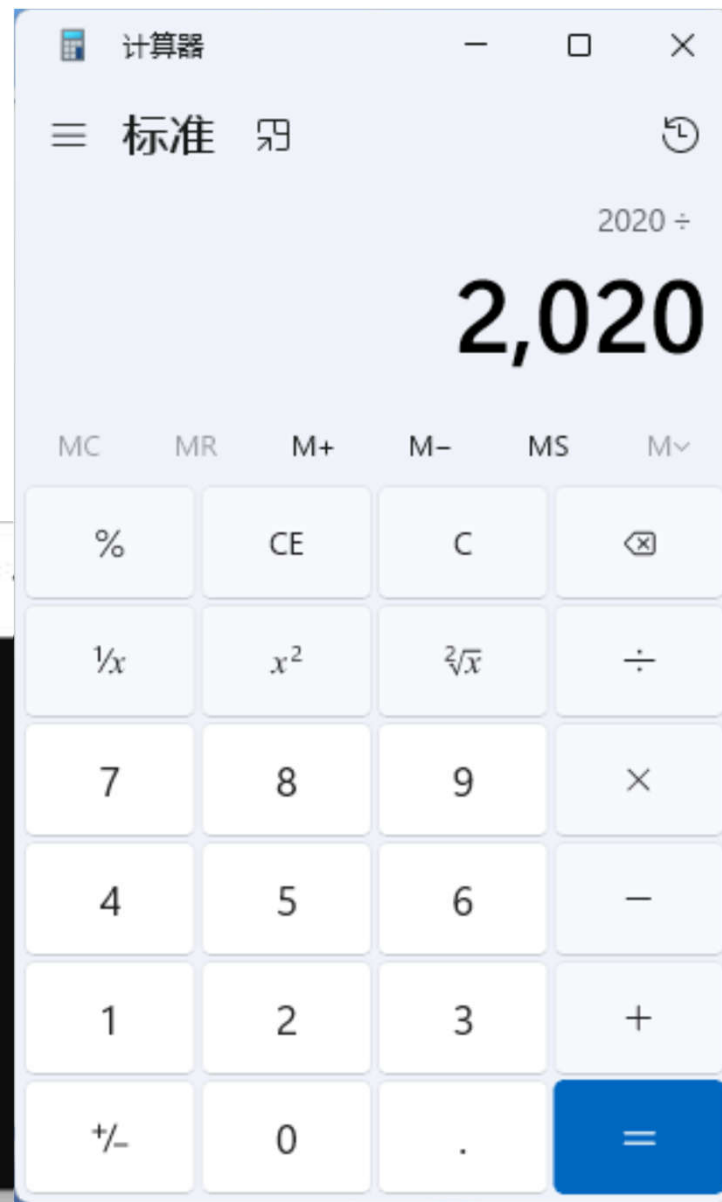
# 操作界面

## ● 类型

- 图形用户接口
- 操作命令（普通命令）
- 批处理与脚本程序



```
C:\WINDOWS\system32\cmd.exe - date  
Microsoft Windows [版本 10.0.22000.493]  
(c) Microsoft Corporation。保留所有权利。  
C:\Users\DELL>date  
当前日期: 2022/03/09 周三  
输入新日期: (年月日)
```





# 操作命令（普通命令）

## ● DOS典型命令

### ■ 文件管理

◆ COPY、COMP、TYPE、DEL、REN

### ■ 磁盘管理

◆ FORMAT、CHKDSK、DISKCOPY、DISKCOMP

### ■ 目录管理

◆ DIR、CD、MD、RD、TREE

### ■ 设备工作模式

◆ CLS、MODE

### ■ 日期、时间、系统设置

◆ DATE、TIME、VER、VOL

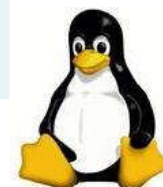
### ■ 运行用户程序

◆ MASM、LINK、DEBUG

# 操作命令（普通命令）

## ● Linux典型命令

命令	作用	命令	作用
ls	列举子目录和文件	find	查找文件
ps	列举进程	whereis	查找文件目录
top	列举进程	man	查看命令帮助信息
echo	输出字符串	cp	拷贝
cat	读文件的内容	inode	查看文件节点
cd	改变目录	tar	压缩和解压
chmod	改变文件属性	rm	删除文件和文件夹
mount	挂载文件系统	umount	卸载文件系统
insmod	安装模块	rmmod	卸载模块



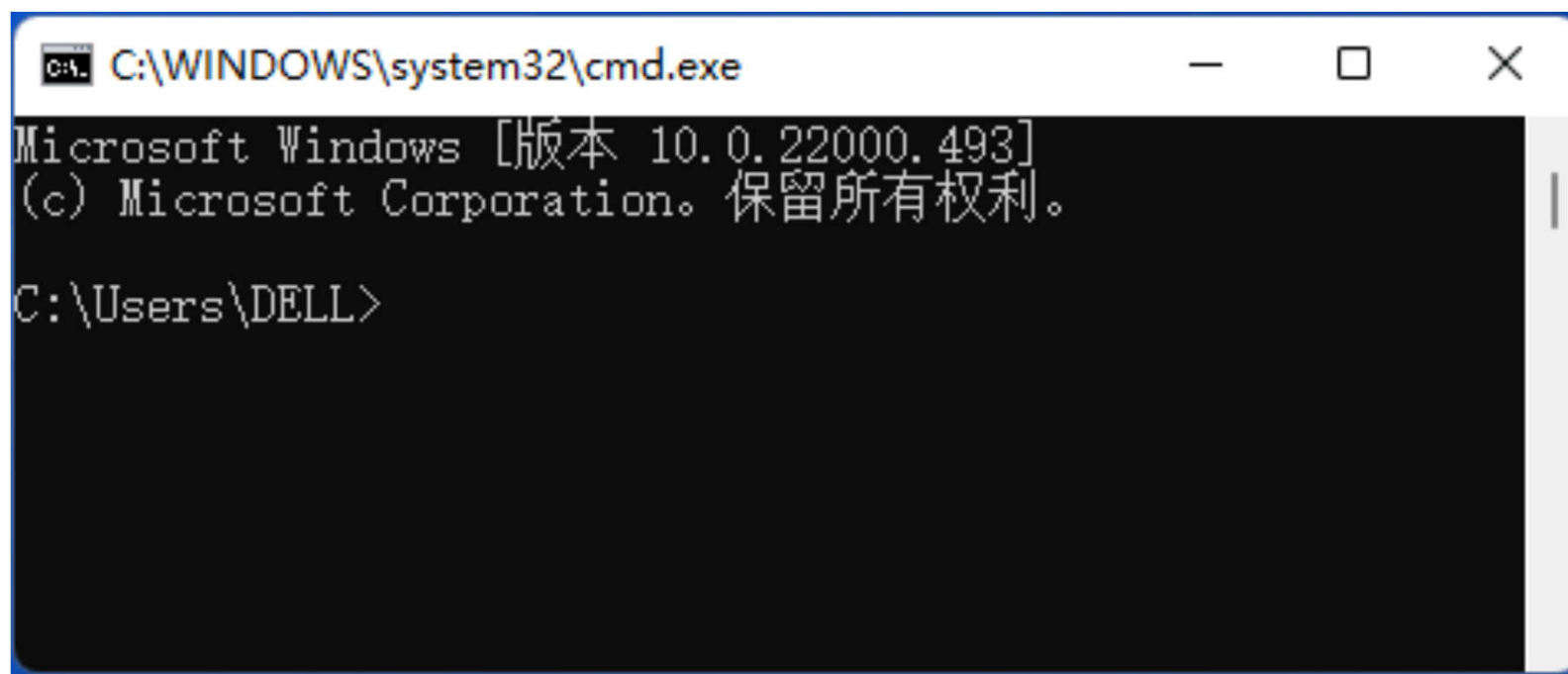
# 操作界面

- 批处理与脚本程序

- 在控制台环境下自动处理一批命令

- ◆ Windows: 批处理程序 (bat/PowerShell)

- ◆ Linux: Shell脚本程序

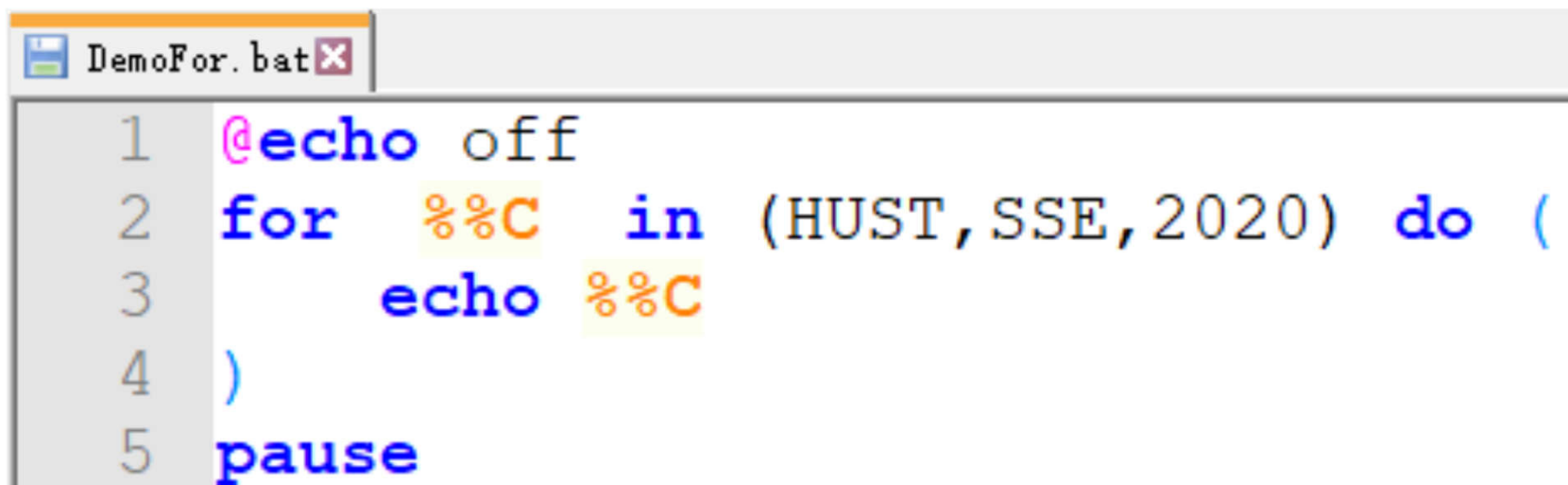


```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [版本 10.0.22000.493]
(c) Microsoft Corporation. 保留所有权利。

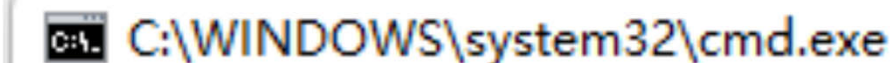
C:\Users\DELL>
```

# 批处理(Windows): BAT/PowerShell

- 例子: 输出字符串数组{ "HUST,SSE,2020" }



```
DemoFor.bat
1 @echo off
2 for %%C in (HUST,SSE,2020) do (
3     echo %%C
4 )
5 pause
```

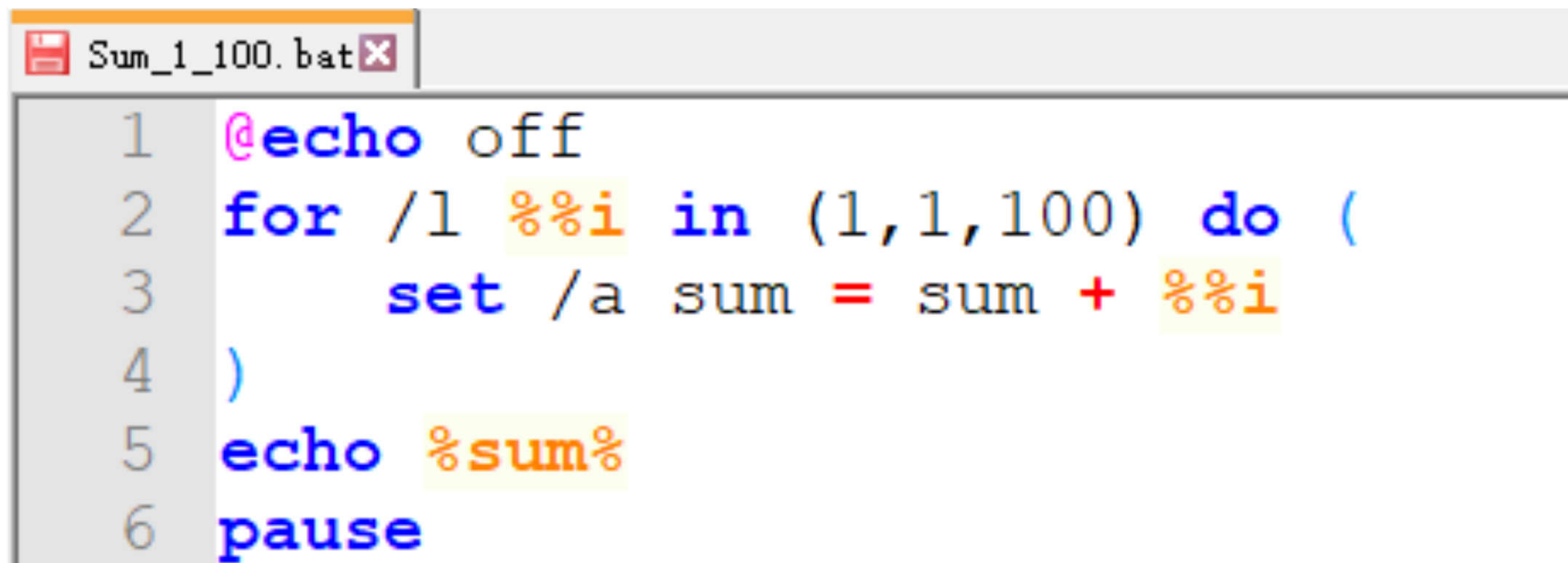


C:\WINDOWS\system32\cmd.exe

```
D:\DemoCode>DemoFor.bat
HUST
SSE
2020
请按任意键继续. . .
D:\DemoCode>
```

# 批处理(Windows): BAT/PowerShell

## ● 例子：求1..100的和并输出



```
1 @echo off
2 for /l %%i in (1,1,100) do (
3     set /a sum = sum + %%i
4 )
5 echo %sum%
6 pause
```

## ● 批处理的特点

- 普通**命令**的集合，按**批**执行，由command解释执行
- 支持变量替换、条件、转移、循环、注释等语法
- 文件后缀\*.BAT，解释执行

# 操作界面

- 批处理与脚本程序

- 在控制台环境下自动处理一批命令

- ◆ Windows批处理程序

- ◆ Linux Shell脚本程序



# Shell脚本例子：安装或更新软件包HUSTLibV30.zip

```
install.shell x
1  #!/bin/bash
2  #创建临时文件
3  sudo mkdir /usr/temp
4  #解压安装包到临时文件
5  sudo echo "正在解压文件"
6  sudo unzip -qd /usr/temp /HUSTLibV30.zip
7  sudo echo "解压完成"
8  #拷贝安装文件
9  sudo cp -rf /usr/temp/HUSTLibV30/HUSTLib /usr/lib
10 #使配置文件生效
11 sudo ldconfig
12 #删除临时文件
13 sudo echo "正在删除临时文件"
14 sudo rm -rf /usr/temp
15 sudo echo "删除临时文件成功"
16 sudo echo "安装完成请重启"
```

## Shell脚本例子2：简单的人机交互

- 若输入Y/y则输出Yes，若输入N/n则输出No (TestYesNo.sh)

```
1  #!/bin/bash
2  read -n 1 -p "Enter your choice (Y/N) :" answer
3  echo
4  case "$answer" in
5  Y|y)
6      echo "Yes" ;;
7  N|n)
8      echo "No" ;;
9  *)
10     echo "Please enter Y or N" ;;
11  esac
```

```
ubuntu@VM-4-6-ubuntu:~/bash$ ./TestYesNO.sh
Enter your choice (Y/N): N
No
ubuntu@VM-4-6-ubuntu:~/bash$ ./TestYesNO.sh
Enter your choice (Y/N): Y
Yes
```



# Shell脚本程序： Shell Script

## ● 特点

- 脚本程序是有一定逻辑顺序和语法结构的命令序列，能完成较复杂的功能和人机交互。
- 所有命令按逻辑逐行执行
- 脚本程序是文本文件（具有可执行属性X）

```
1  #!/bin/bash
2  echo "Enter a File Name:"
3  read FileName
4  if test -e /root/$FileName
5  then
6      echo "The file is exist!"
7  else
8      echo "The file is not exist!"
9  fi
```

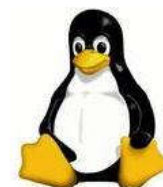
test.sh

# 运行脚本程序的三个方法

- 方法1：直接运行（用缺省版本**Shell**运行脚本程序）
  - `./test.sh` ✓
- 方法2：在命令行上指定某个特定版本**Shell**执行脚本
  - `$bash ./test.sh` ✓
- 方法3：在脚本首行指定特定**Shell**执行当前脚本

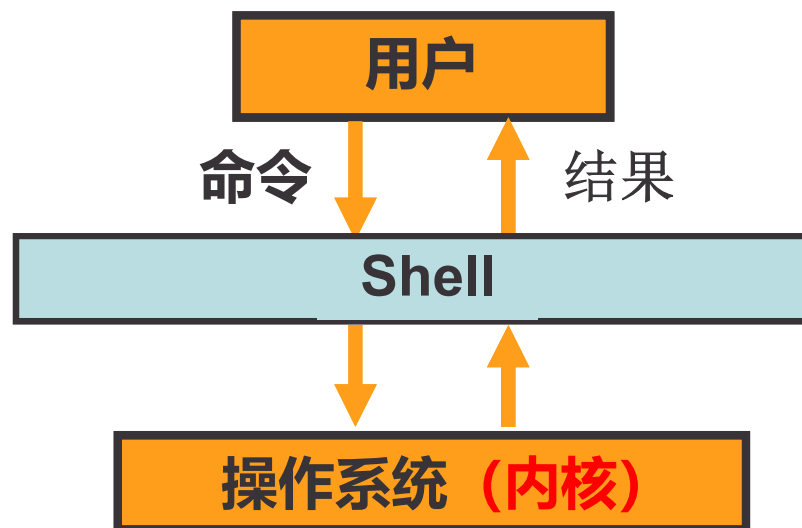
```
1  #!/bin/bash
2  cd ~
3  mkdir shell_test
4  cd shell_test
5  for ((i=0; i<10; i++)); do
6      touch test_${i}.txt
7  done
```

test.sh



# Linux Shell (Windows CMD)

- Shell是操作系统与用户的交互机制（操作界面）

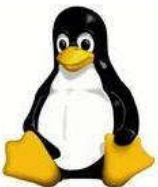
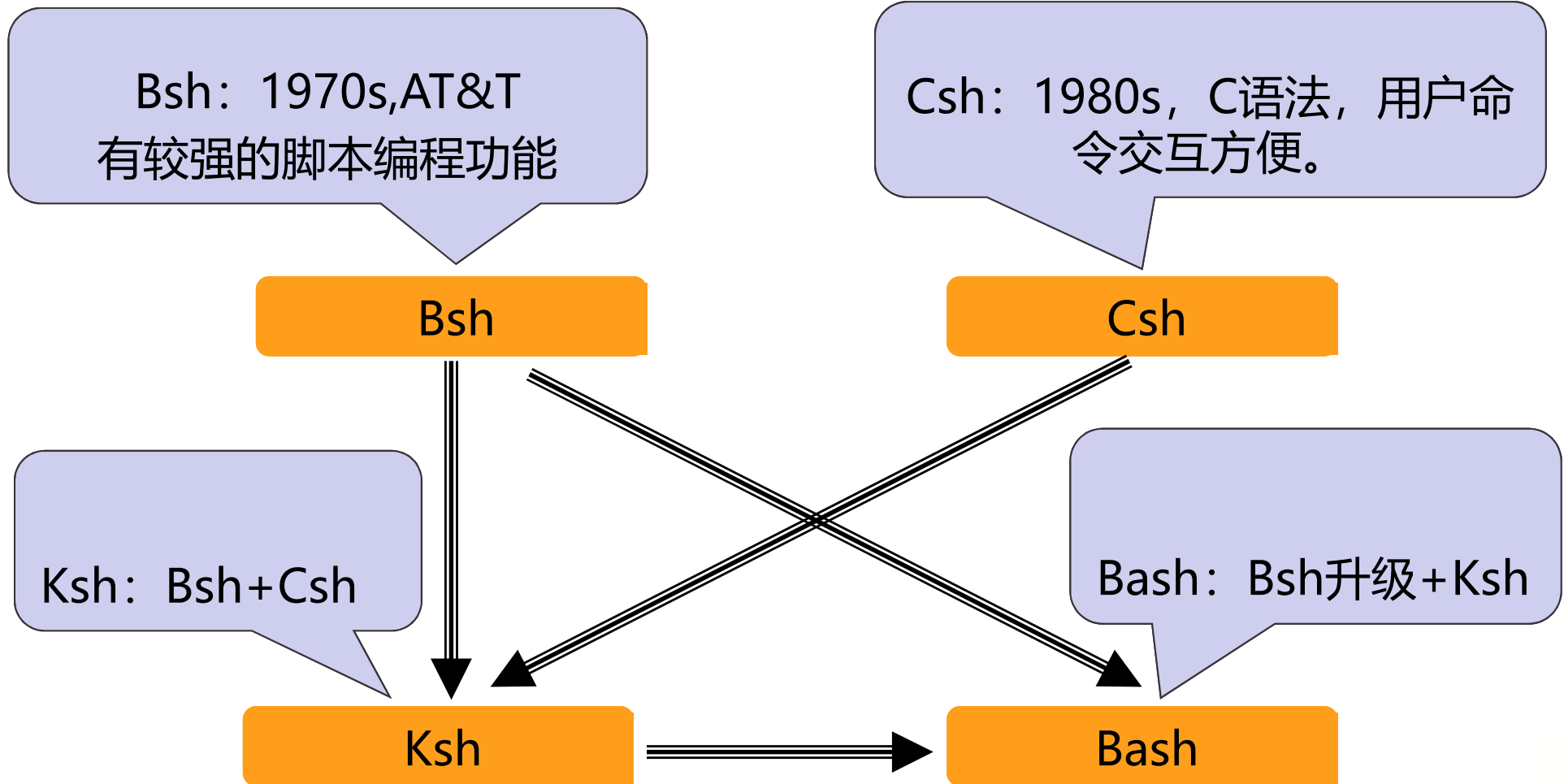


- 通过Shell (/控制台) 执行用户命令
- 组织和管理用户命令的执行和结果展示



# Linux Shell（类似Windows CMD）

## ● Shell的发展与分类



# Linux Shell

## ● Bash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
- 作业控制功能
- 将命令序列定义为功能键
- 重定向与管道
- 脚本程序



# 重定向操作的例子 (windows)

## ● E:\ShowSPLEMF>tree /F > ShowSPLEMF.txt

管理员: C:\Windows\system32\cmd.exe

```
E:\ShowSPLEMF>tree /F
```

卷 DATA 的文件夹 PATH 列表

卷序列号为 480A-CCE0

E:.

- ShowEMF.rc
- ShowEMFDlg.cpp
- ShowEMFDlg.h
- StdAfx.cpp
- StdAfx.h

Release

- ShowEMF.exe
- ShowEMF.obj
- ShowEMF.pch
- ShowEMF.res

res

- ShowEMF.ico
- ShowEMF.rc2

输出重定向

ShowSPLEMF.txt

1 卷 DATA 的文件夹 PATH 列表

2 卷序列号为 480A-CCE0

3 E:.

- 4 ShowEMF.rc
- 5 ShowEMFDlg.cpp
- 6 ShowEMFDlg.h
- 7 ShowSPLEMF.txt
- 8 StdAfx.cpp
- 9 StdAfx.h

10 Release

- 11 ShowEMF.exe
- 12 ShowEMF.obj
- 13 ShowEMF.pch
- 14 ShowEMF.res

15 res

- 16 ShowEMF.ico
- 17 ShowEMF.rc2



# 标准输入/输出设备（文件）

- 程序缺省输入来源：键盘（文件0）

- 标准输入设备

- 程序缺省输出(含错误)方向：显示器（文件1和文件2）

- 标准输出设备

输入/输出文件	设备	文件描述符
标准输入文件	键盘	0
标准输出文件	显示器	1
标准错误输出文件	显示器	2

```
ubuntu@VM-4-6-ubuntu:~/bash$ ./TestYesNo.sh
Enter your choice (Y/N): N
No
```

# 重定向操作的例子 (Linux)

- 输出重定向：把命令的输出重定向到其它文件

- 将标准输出重定向到特定文件

```
$ ls /etc/ > etc.log
```

- 将标准输出重定向/追加到特定文件

```
$ ls /etc/sysconfig/ >> etc.txt
```

- 将错误输出重定向到特定文件

```
$ ErrCmd 2> err.log
```

- 将标准输出和错误输出重定向到特定文件

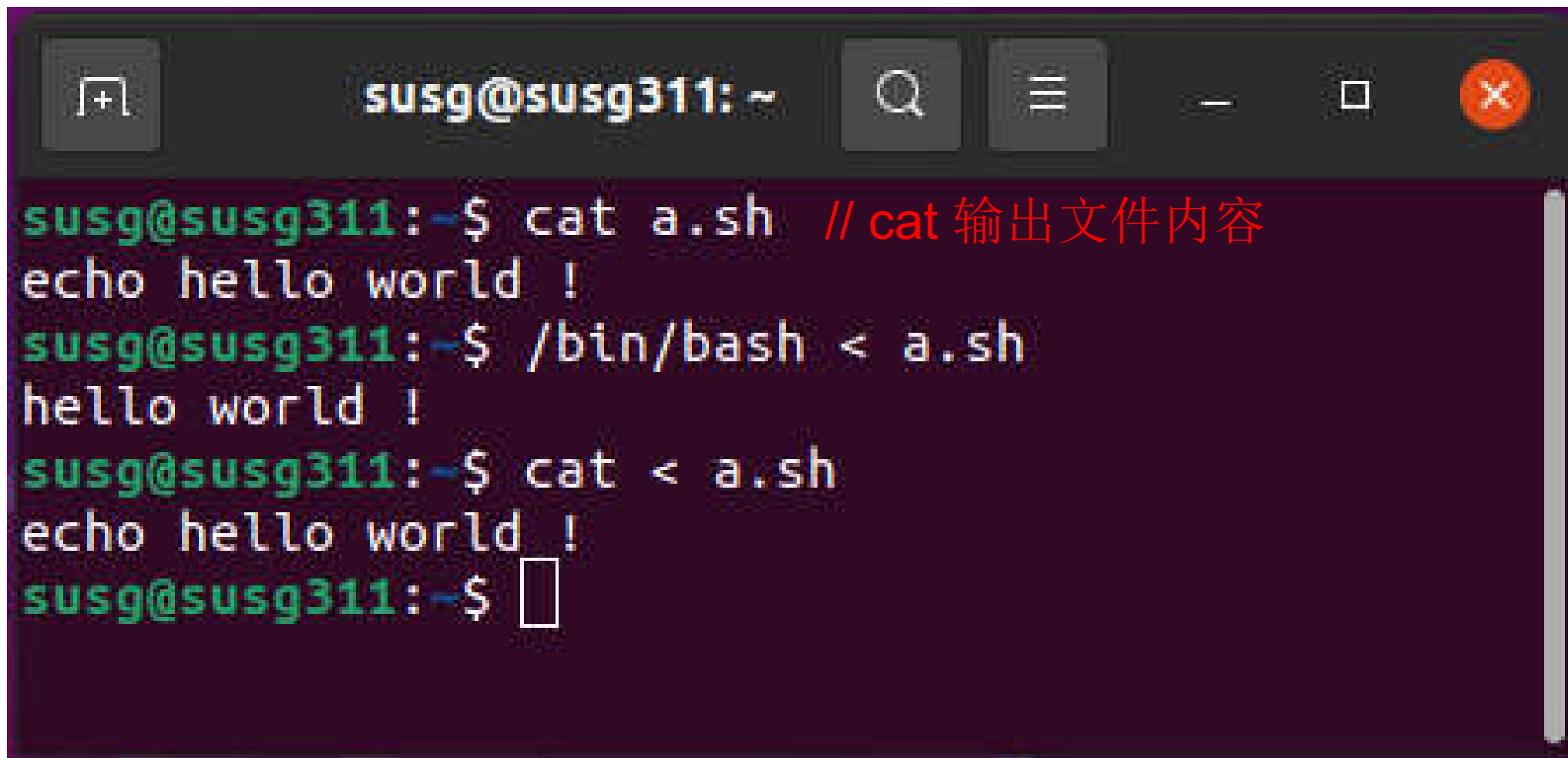
```
$ AComand &> ErrFile 或者
```

```
$ AComand > AComand.out 2>&1
```



# 重定向操作的例子 (Linux)

- 输入重定向：把命令的输入来源改为其它文件

A terminal window with a dark background and light-colored text. The window title bar shows 'susg@susg311: ~' and standard window controls. The terminal content shows a sequence of commands and their outputs, demonstrating input redirection. The first command 'cat a.sh' outputs 'echo hello world !'. The second command '/bin/bash < a.sh' also outputs 'hello world !'. The third command 'cat < a.sh' outputs 'echo hello world !'. The terminal ends with a prompt and a cursor.

```
susg@susg311: ~  
susg@susg311:~$ cat a.sh // cat 输出文件内容  
echo hello world !  
susg@susg311:~$ /bin/bash < a.sh  
hello world !  
susg@susg311:~$ cat < a.sh  
echo hello world !  
susg@susg311:~$
```

# 重定向操作（Linux）

- 把命令缺省输入来源或输出方向修改为其他文件/设备。

类别	操作符	说明
输入重定向	<	将命令输入由默认的键盘更改/重定向为指定的文件
输出重定向	>	将命令输出由默认的显示器更改/重定向为指定的文件
	>>	将命令输出重定向并追加到指定文件的末尾
错误重定向	2>	将命令的错误输出重定向到指定文件（先清空）。
	2>>	将命令的错误输出重定向到指定文件（追加到末尾）。
输出与错误组合重定向	&>	将命令的正常输出和错误输出重定向到指定文件。

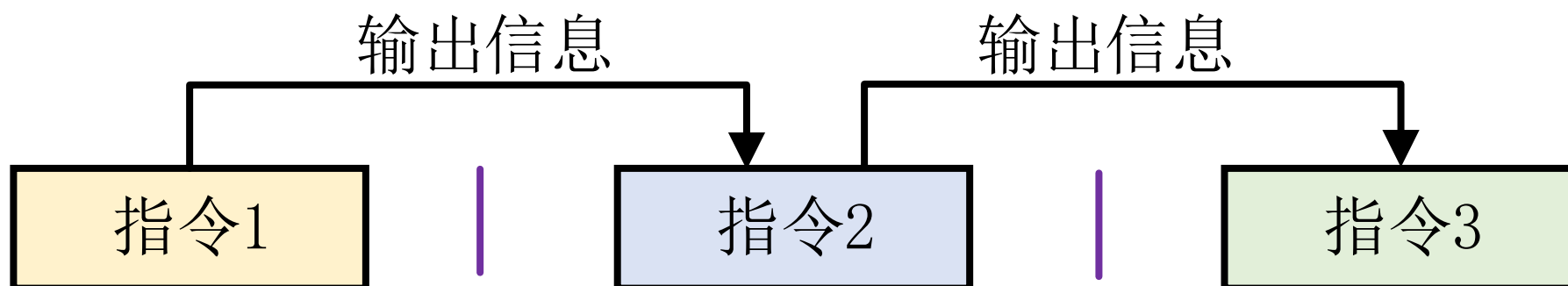
# 管道

## ● Linux管道

■ 程序相连，一个程序的输出作为另一程序的输入

■ 管道操作符 |

◆ “|”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



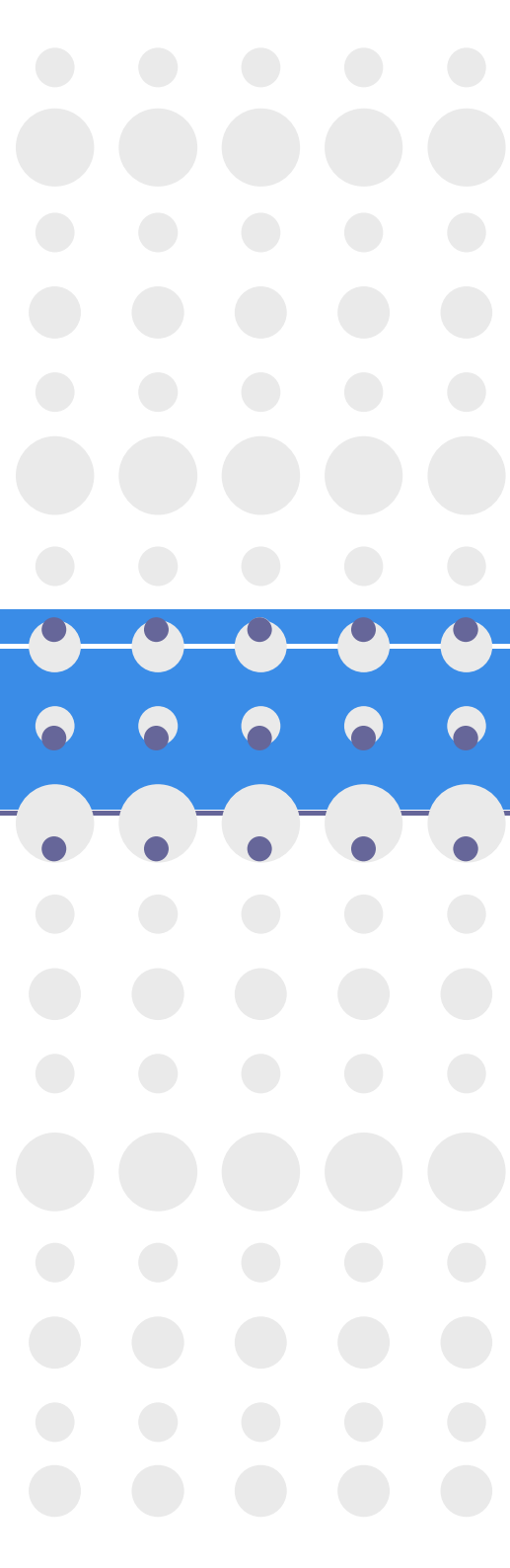
# 管道

## ● Linux管道的例子

```
t@t-virtual-machine:~$ ls /etc
acpi                hostname            ppp
adduser.conf        hosts               profile
alsa                hosts.allow         profile.d
alternatives         hosts.deny           protocols
anacrontab           hp                  pulse
apg.conf             ifplugd             python3
apm                  init                 python3.10
apparmor             init.d              rc0.d
apparmor.d           initramfs-tools     rc1.d
apport               inputrc             rc2.d
```

■ `ls /etc | wc -l`

```
t@t-virtual-machine:~$ ls /etc | wc -l
223
```



## 3.4 系统调用

# 用户界面(User Interface)

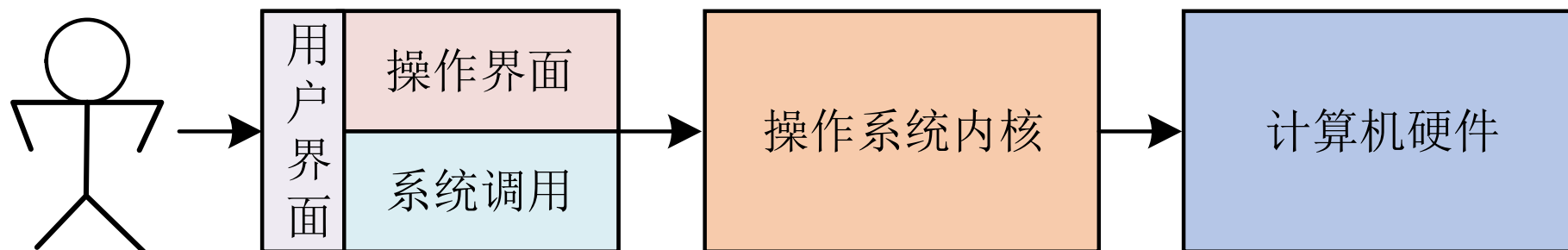
- 用户界面的定义

- OS提供给用户控制计算机的机制，又称用户接口。

- 用户界面的类型

- 操作界面

- 系统调用 (System Call, 系统功能调用, 程序界面)



# printf( ), exit( )与add( )比较

```
#include <stdio.h>

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

■ 特点：涉及显卡和进程操作

```
#include <stdio.h>

int add( int a, int b)
{
    return (a + b);
}

int main(void)
{
    int sum = add( 100, 300);
}
```

## 例子4：利用DOS 21H中断显示字符串（09号功能）

StrHello DB 'Hello!' ; 定义要显示的字符串

...

MOV DX, StrHello ;  $DX \leftarrow$  字符串地址

MOV AH, 09H ;  $AH \leftarrow 09h$  : 功能编号

INT 21H ; DOS 21H 中断

■ 特点：涉及外设（显卡）操作





# 系统调用

## ■ 系统调用 (**System Call, System Service Call**)

■ 操作系统内核为应用程序提供的服务/函数。

◆ 例: `printf, exit, fopen, fgetc, 21H(09)`

## ■ 特点

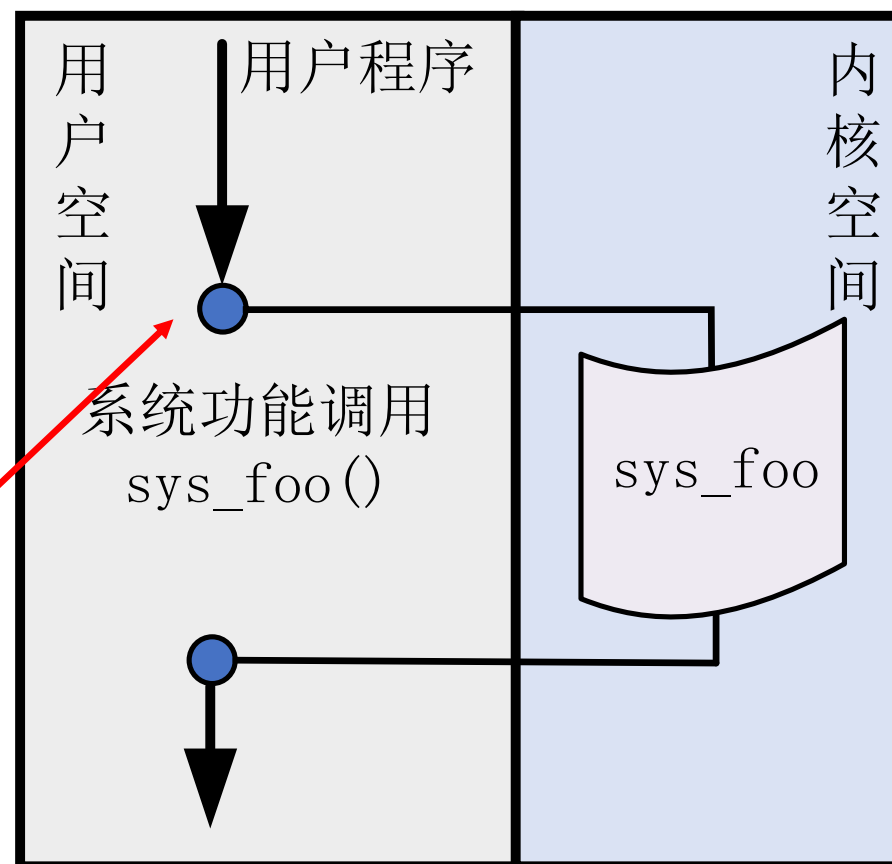
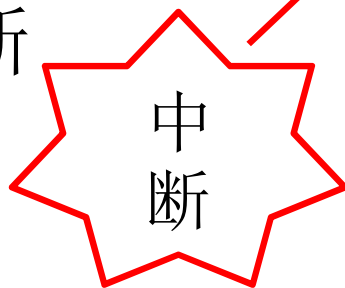
◆ 内核实现

◆ 存取核心资源或硬件

◆ 调用过程产生中断

□ 用户态 ↔ 核态

□ 自愿中断



# 系统调用表和调用形式

- 系统调用表

- 全部系统调用的入口列表

- ◆ 有序排列

- 系统调用号：系统调用的唯一编号

- 系统调用的一般调用形式

- 访管指令： SVC X

- ◆ SVC = SuperVisor Call

- ◆ X = 系统调用的编号

系统调用表

01号系统调用
02号系统调用
03号系统调用
.....
X 号系统调用
.....
.....
N 号系统调用

# 系统调用表和调用形式

## ● DOS的系统调用（部分）

■ 01: 键盘输入无回显	■ 39: 建立子目录
■ 02: 屏幕输出1个字符	■ 3A: 删除子目录
■ 03: 显示输出	■ 3B: 改变当前目录
■ 04: 串口输入1字符	■ 3C: 创建文件
■ ...	■ ...
■ 08: 键盘输入无回显	■ 3F: 读文件
■ 09: 显示字符串	■ 40: 写文件
■ ...	■ 4C: 结束程序

# 系统调用表和调用形式

## ● Linux的系统调用（部分）

```
1 ENTRY(sys_call_table)
2     .long SYMBOL_NAME(sys_ni_syscall) /* 0 */
3     .long SYMBOL_NAME(sys_exit)
4     .long SYMBOL_NAME(sys_fork)
5     .long SYMBOL_NAME(sys_read)
6     .long SYMBOL_NAME(sys_write)
7     .long SYMBOL_NAME(sys_open)      /* 5 */
8     .long SYMBOL_NAME(sys_close)
9     .long SYMBOL_NAME(sys_waitpid)
10    .long SYMBOL_NAME(sys_creat)
11    .long SYMBOL_NAME(sys_link)
12    .long SYMBOL_NAME(sys_unlink)     /* 10 */
13    .long SYMBOL_NAME(sys_execve)
14    .long SYMBOL_NAME(sys_chdir)
15    .long SYMBOL_NAME(sys_time)
16    .long SYMBOL_NAME(sys_mknod)
17    .long SYMBOL_NAME(sys_chmod)      /* 15 */
```

# 系统调用表和调用形式

## ● Linux的系统调用（部分）

```
1 ENTRY(sys_call_table)
2     ...
3     .long SYMBOL_NAME(sys_mkdir)
4     .long SYMBOL_NAME(sys_rmdir)          /* 40 */
5     .long SYMBOL_NAME(sys_dup)
6     .long SYMBOL_NAME(sys_pipe)
7     .long SYMBOL_NAME(sys_times)
8     .long SYMBOL_NAME(sys_ni_syscall)
9     .long SYMBOL_NAME(sys_brk)            /* 45 */
10    .long SYMBOL_NAME(sys_setgid16)
11    .long SYMBOL_NAME(sys_getgid16)
12    .long SYMBOL_NAME(sys_signal)
13    .long SYMBOL_NAME(sys_geteuid16)
14    .long SYMBOL_NAME(sys_getegid16)       /* 50 */
15    .long SYMBOL_NAME(sys_acct)
16    .long SYMBOL_NAME(sys_umount)
17    .long SYMBOL_NAME(sys_ni_syscall)
18    .long SYMBOL_NAME(sys_ioctl)
```

# 系统调用的调用方式

- 例：DOS：9号系统调用（屏幕输出字符串）（输出Hello）

```
1 DATA SEGMENT
2     STR1 DB 'HOW DO YOU DO?',0DH,0AH,'$'
3 DATA ENDS
4
5 CODE SEGMENT
6     ASSUME CS:CODE,DS:DATA
7 START:
8     MOV AX,DATA
9     MOV DS,AX
10    MOV DX,OFFSET STR1 ;字符串首偏移地址放到DX中
11    MOV AH,9
12    INT 21H;输出字符串
13
14    MOV AH,4CH
15    INT 21H
16 CODE ENDS
17 END START
```

# 系统调用的调用方式

- 例：DOS：1号系统调用(键盘输入字符) (键盘输入1/2?)

```
1 InputKey:
2     MOV     AH, 1
3     INT     21H           ;等待输入一个字符
4     CMP     AL, '1'
5     JE      ONE           ;如果输入'1'则跳到ONE处执行
6     CMP     AL, '2'
7     JE      TWO           ;如果输入'2'则跳到TWO处执行
8     CMP     AL, '3'
9     JE      THREE        ;如果输入'3'则跳到THREE处执行
10    JMP     InputKey      ;如果不是1,2,3, 则继续要求输入
11 ONE:      .....
12 TWO:      .....
13 THREE:    .....
```



# 系统调用的调用方式

## ■ DOS中系统调用的调用方式

■ DOS: INT 21H (利用AH存放系统调用的编号)

■ 01: 键盘输入无回显	■ 39: 建立子目录
■ 02: 屏幕输出1个字符	■ 3A: 删除子目录
■ 03: 显示输出	■ 3B: 改变当前目录
■ 04: 串口输入1字符	■ 3C: 创建文件
■ ...	■ ...
■ 08: 键盘输入无回显	■ 3F: 读文件
■ 09: 显示字符串	■ 40: 写文件
■ ...	■ 4C: 结束程序



# 系统调用的调用方式

## ● 例：Linux：4号系统调用(写文件)（屏幕输出字符串）

;输出字符串：Hello World!

MOV EBX, 1 ;EBX=1=stdout

MOV ECX, MSG ;字符串首地址

MOV EDX, 13 ;字符串长度

MOV EAX, 4 ;系统调用编号 (4 = write)

INT 80h ;中断：输出字符串

MSG: DB "Hello World!"

■ 特点：利用EAX寄存器存放系统调用的编号



KYLIN  
银河麒麟

# 系统调用的调用方式

## ● Linux的系统调用表 (Linux2. ++)

```
1 ENTRY(sys_call_table)
2     .long SYMBOL_NAME(sys_ni_syscall) /* 0 */
3     .long SYMBOL_NAME(sys_exit)
4     .long SYMBOL_NAME(sys_fork)
5     .long SYMBOL_NAME(sys_read)
6     .long SYMBOL_NAME(sys_write)
7     .long SYMBOL_NAME(sys_open) /* 5 */
8     .long SYMBOL_NAME(sys_close)
9     .long SYMBOL_NAME(sys_waitpid)
10    .long SYMBOL_NAME(sys_creat)
11    .long SYMBOL_NAME(sys_link)
12    .long SYMBOL_NAME(sys_unlink) /* 10 */
13    .long SYMBOL_NAME(sys_execve)
14    .long SYMBOL_NAME(sys_chdir)
15    .long SYMBOL_NAME(sys_time)
16    .long SYMBOL_NAME(sys_mknod)
17    .long SYMBOL_NAME(sys_chmod) /* 15 */
```

# 系统调用的调用方式

## ● Linux的系统调用表 (Linux0.11)

```
1  fn_ptr sys_call_table[] = { sys_setup, sys_exit, sys_fork, sys_read,  
2  sys_write, sys_open, sys_close, sys_waitpid, sys_creat, sys_link,  
3  sys_unlink, sys_execve, sys_chdir, sys_time, sys_mknod, sys_chmod,  
4  sys_chown, sys_break, sys_stat, sys_lseek, sys_getpid, sys_mount,  
5  sys_umount, sys_setuid, sys_getuid, sys_stime, sys_ptrace, sys_alarm,  
6  sys_fstat, sys_pause, sys_utime, sys_stty, sys_gtty, sys_access,  
7  sys_nice, sys_ftime, sys_sync, sys_kill, sys_rename, sys_mkdir,  
8  sys_rmdir, sys_dup, sys_pipe, sys_times, sys_prof, sys_brk, sys_setgid,  
9  sys_getgid, sys_signal, sys_geteuid, sys_getegid, sys_acct, sys_phys,  
10 sys_lock, sys_ioctl, sys_fcntl, sys_mpx, sys_setpgid, sys_ulimit,  
11 sys_uname, sys_umask, sys_chroot, sys_ustat, sys_dup2, sys_getppid,  
12 sys_getpgrp, sys_setsid, sys_sigaction, sys_sgetmask, sys_ssetmask,  
13 sys_setreuid, sys_setregid  
14 };
```

# 系统调用的调用方式

## ● 例：Linux：4号系统调用(写文件)（屏幕输出字符串）

;输出字符串：Hello World!

MOV EBX, 1 ;EBX=1=stdout

MOV ECX, MSG ;字符串首地址

MOV EDX, 13 ;字符串长度

MOV EAX, 4 ;系统调用编号 (4 = write)

INT 80h ;中断：输出字符串

MSG: DB "Hello World!"

MOV EAX, 1 ;系统调用编号 (1 = exit)

INT 80h ;中断：结束进程

■ 特点：利用EAX寄存器存放系统调用的编号



KYLIN  
银河麒麟

# 系统调用的调用方式（小结）

## ■ 系统调用的一般调用形式

■ SVC N

## ■ 具体OS中系统调用的调用形式

■ DOS: INT 21H + AH

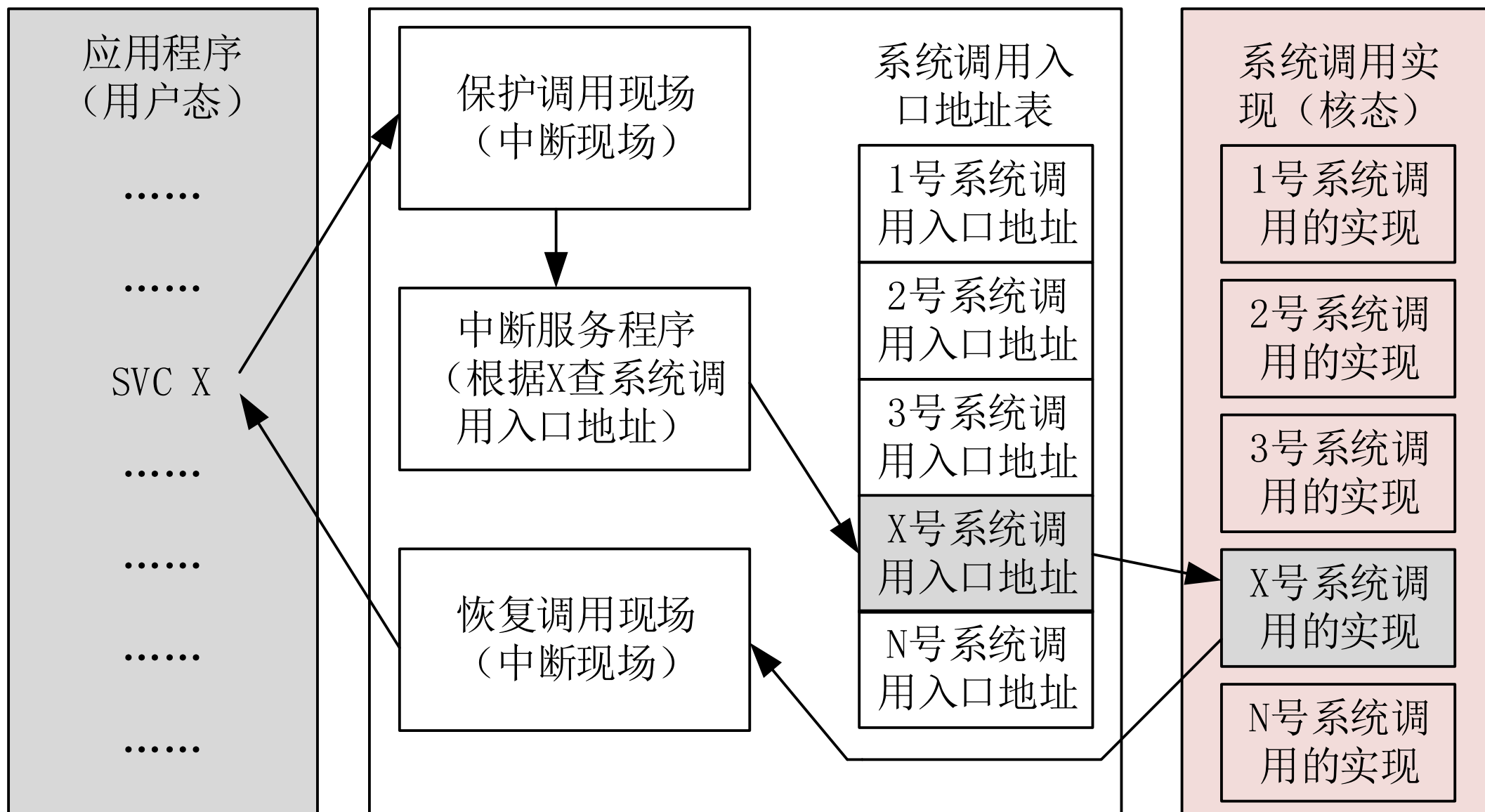
■ Linux: INT 80H + EAX

■ 注意:

◆ INT XXH = SVC指令

◆ AH/EAX = 系统调用的编号: N

# 系统调用的执行过程（中断的过程）





# 隐式系统调用

```
#include <stdio.h>

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

## ■ 特点

- 系统API函数
- 在高级语言中使用
- 包含INT 80h指令（软中断）

int main() {;省略了部分代码

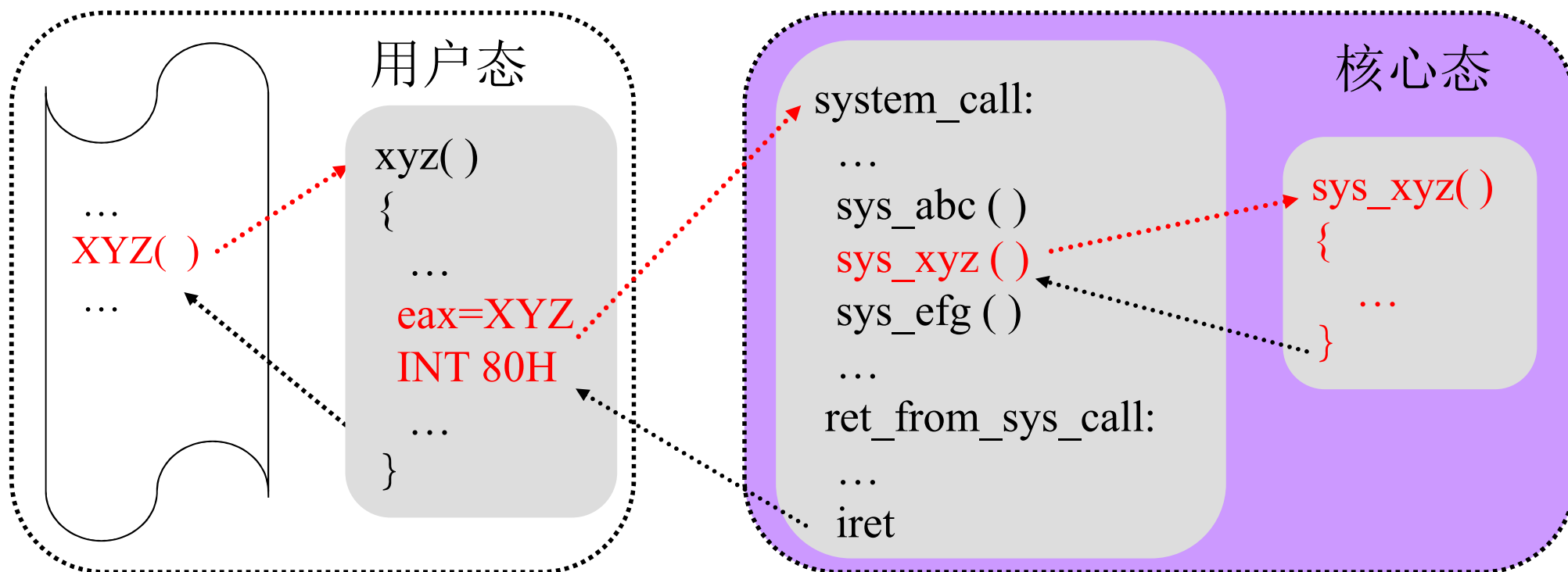
```
__asm__ (
    "POPL  %ESI;"
    "MOVL  $1, %EBX;"
    "MOVL  %esi, %ECX;"
    "MOVL  $12, %EDX;"
    "MOVL  $4, %EAX;"
    "INT   $0x80;"
    "MOVL  $0, %EBX;"
    "MOVL  $1, %EAX;"
    "INT   $0x80;"
    ".string \"Hello World\\n\";");
}
```

## 3.4 Linux系统调用原理





# Linux系统调用的工作原理



应用程序  
使用隐式  
方式调用  
系统调用  
`xyz( )`

`xyz( )`在  
Libc中编译  
为含有INT  
0x80指令的  
代码。

`system_call`是0x80  
号中断服务程序的一  
部分，指定各系  
统调用的入口。例  
`sys_xyz( )`

具体实现各个  
系统调用。例  
：`sys_xyz( )`

# 案例：read系统调用的工作原理

● 例： `int read( int fd, char * buf, int n) // unistd.h`

```
1 int read(int fd, char *buf, int n)
2 {
3     long_res;
4     __asm__ volatile (
5         "int$0x80"
6         : "=a" (__res)
7         : "0" (__NR_read), "b" ((long) (fd)), "C" ((long) (buf)),
8         "d" ((long) (n)));
9     if (__res >= 0)
10         return __res;
11     errno = -res;
12     return -1;
13 }
```



# 案例：read系统调用的工作原理

## ● system\_call设置为0x80号中断的处理程序

```
1 void sched_init(void)
2 {
3     .....
4     set_intr_gate(0x20, &timer_interrupt);
5     .....
6     set_system_gate(0x80, &system_call);
7 }
```



# 案例：read系统调用的工作原理

```
1  system_call:
2      push %ds
3      push %es
4      .....
5      mov %dx,%fs
6      call sys_call_table(,%eax,4)
7      pushl %eax # 把系统调用号入栈。
8      movl _current,%eax
9      cmpl $0,state(%eax) # state
10     jne reschedule
11     cmpl $0,counter(%eax) # counter
12     je reschedule
13 ret_from_sys_call:
14     .....
15     movl signal(%eax),%ebx
16     movl blocked(%eax),%ecx
17     notl %ecx # 每位取反。
18     .....
19     call _do_signal # 调用信号处理程序
20     popl %eax
21     .....
22     pop %es
23     pop %ds
24     iret
```



# 案例：read系统调用的工作原理

## ● 系统调用表

```
1  fn_ptr sys_call_table[] = { sys_setup, sys_exit, sys_fork, sys_read,  
2    sys_write, sys_open, sys_close, sys_waitpid, sys_creat, sys_link,  
3    sys_unlink, sys_execve, sys_chdir, sys_time, sys_mknod, sys_chmod,  
4    sys_chown, sys_break, sys_stat, sys_lseek, sys_getpid, sys_mount,  
5    sys_umount, sys_setuid, sys_getuid, sys_stime, sys_ptrace, sys_alarm,  
6    sys_fstat, sys_pause, sys_utime, sys_stty, sys_gtty, sys_access,  
7    sys_nice, sys_ftime, sys_sync, sys_kill, sys_rename, sys_mkdir,  
8    sys_rmdir, sys_dup, sys_pipe, sys_times, sys_prof, sys_brk, sys_setgid,  
9    sys_getgid, sys_signal, sys_geteuid, sys_getegid, sys_acct, sys_phys,  
10   sys_lock, sys_ioctl, sys_fcntl, sys_mpx, sys_setpgid, sys_ulimit,  
11   sys_uname, sys_umask, sys_chroot, sys_ustat, sys_dup2, sys_getppid,  
12   sys_getpgrp, sys_setsid, sys_sigaction, sys_sgetmask, sys_ssetmask,  
13   sys_setreuid, sys_setregid  
14   };
```





# 80H中断的中断服务程序system\_call( )

## ● system\_call( )

```
{
```

```
//.....
```

```
//寄存器%eax存放有系统调用编号，
```

```
//以%eax遍历表sys_call_table，查找对应的服务子程序。
```

```
CALL *SYMBOL_NAME(sys_call_table)(%eax, 4)
```

```
}
```



KYLIN  
银河麒麟

# 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) // 2

.long SYMBOL\_NAME(sys\_read) // 3

.long SYMBOL\_NAME(**sys\_write**) // 4

.long SYMBOL\_NAME(sys\_open) // 5

.....

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



KYLIN  
银河麒麟



# 系统调用编号的声明

- 系统调用编号的声明

格式: **#define \_\_NR\_CallName ID**

```
#define __NR_exit      1  /* exit 的系统功能号 */
#define __NR_fork      2  /* fork 的系统功能号 */
#define __NR_read      3  /* read 的系统功能号 */
#define __NR_write     4  /* write 的系统功能号 */
#define __NR_open      5  /* open 的系统功能号 */
#define __NR_close     6  /* close 的系统功能号 */
.....
// 目前已定义到340
```

- **Linux-source-2.6.38**

- ./arch/x86/include/asm/unistd\_32.h



# 系统调用编号的声明

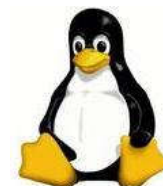
- 系统调用编号的声明 【例子】

格式: **#define \_\_NR\_CallName ID**

```
#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__
```

- **Linux-source-2.6.38**

- ./arch/x86/include/asm/unistd\_32.h



# 系统调用函数的声明

- 系统调用函数的声明和例子

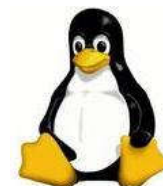
**.long sys\_XXXX**

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_recvmmsg  
.long sys_fanotify_init  
.long sys_fanotify_mark  
.long sys_prlimit64          /* 340 */  
.long sys_mycall  
long sys_addtotal  
long sys_three
```

- **Linux-source-2.6.38**

- ./arch/x86/kernel/syscall\_table\_32.S



# 系统调用函数的定义和例子

```
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_three()
{
    printk("这是我添加的第三个系统调用");
    return 0;
}
```

- 注意修饰词 **asmlinkage**

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

# 系统调用的两种调用方法(例:sys\_FuncName )

- 较新的版本

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

```
#include<unistd.h>
#include<sys/syscall.h>
#include<stdio.h>
#include<linux/kernel.h>

int main()
{
    syscall(442,11);
    return 0;
}
```