

操作系统原理

PRINCIPLES OF OPERATING SYSTEM

北京大学计算机科学技术系 陈向群

Department of computer science and Technology

Peking University

2015 春季

第2讲

操作系统运行环境与运行机制

这一部分很重要！

回顾 — 操作系统的主要工作

- ◎ 程序的执行

启动程序、执行程序以及程序结束的工作

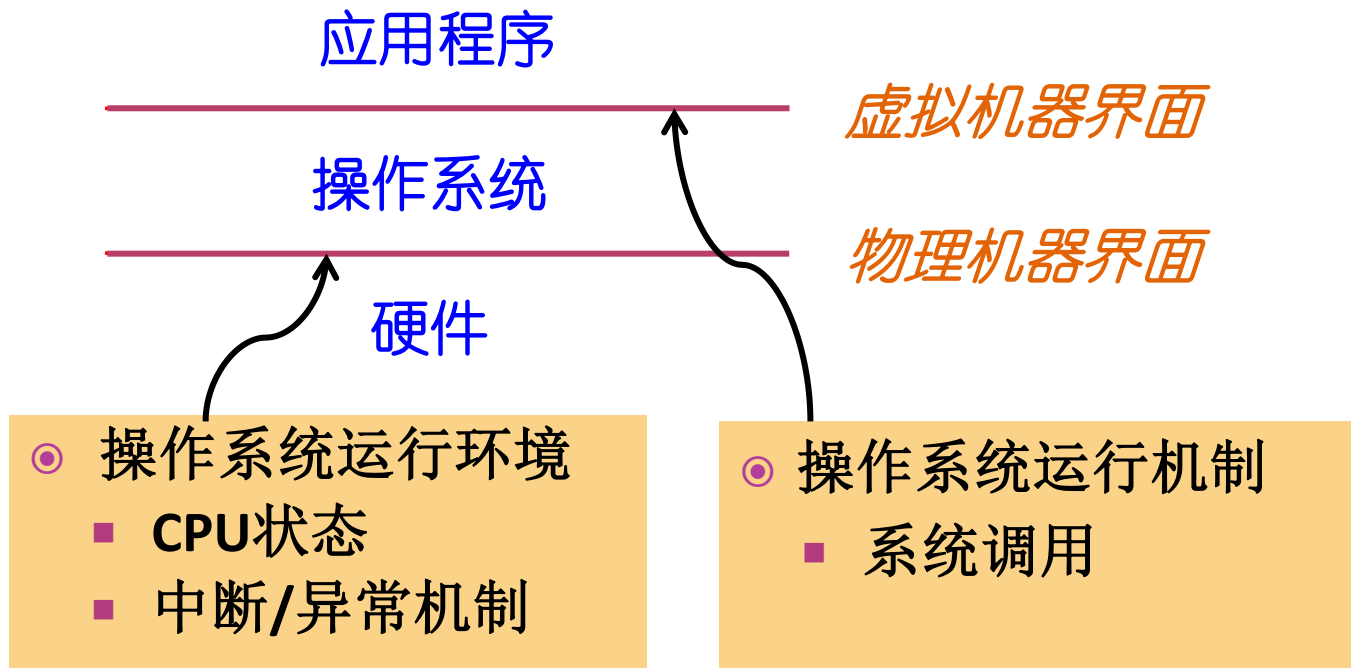
- ◎ 完成与体系结构相关的工作

- ◎ 完成应用程序所需的共性任务

提供各种基本服务

- ◎ 性能、安全、健壮等问题

本讲主要内容



处理器状态(模式)

中央处理器（CPU）

- 处理器由运算器、控制器、一系列的寄存器以及高速缓存构成
- 两类寄存器：
 - 用户可见寄存器：高级语言编译器通过优化算法分配并使用之，以减少程序访问内存次数
 - 控制和状态寄存器：用于控制处理器的操作
通常由操作系统代码使用

控制和状态寄存器

- ◉ 用于控制处理器的操作
- ◉ 在某种特权级别下可以访问、修改
- ◉ 常见的控制和状态寄存器
 - 程序计数器（**PC: Program Counter**），记录将要取出的指令的地址
 - 指令寄存器（**IR: Instruction Register**），记录最近取出的指令
 - 程序状态字（**PSW: Program Status Word**），记录处理器的运行状态如条件码、模式、控制位等信息

操作系统的需求——保护

- ◎ 从操作系统的特征考虑
并发、共享
- ◎ 提出要求 → 实现保护与控制

需要硬件提供基本运行机制：

- 处理器具有特权级别，能在不同的特权级运行的不同指令集合
- 硬件机制可将OS与用户程序隔离

处理器的状态(模式MODE)

- 现代处理器通常将CPU状态设计划分为两种、三种或四种
- 在程序状态字寄存器PSW中专门设置一位，根据运行程序对资源 and 指令的使用权限而设置不同的CPU状态



例:X86架构中的EFLAGS寄存器
描述符中也设置了权限级别

特权指令和非特权指令

2个状态

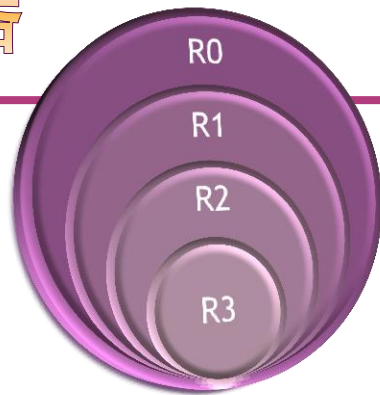
- ◎ 操作系统需要两种CPU状态
 - 内核态(Kernel Mode): 运行操作系统程序
 - 用户态(User Mode): 运行用户程序
- ◎ 特权(privilege)指令: 只能由操作系统使用、用户程序不能使用的指令
- ◎ 非特权指令: 用户程序可以使用的指令



● 下列哪些是特权指令？哪些是非特权指令？

— 启动I/O 控制转移 内存清零 修改程序状态字
设置时钟 算术运算 允许/禁止中断 访管指令
取数指令 停机

实例：X86系列处理器



- ◎ X86支持4个处理器特权级别

特权环：R0、R1、R2和R3

- 从R0到R3，特权能力由高到低
- R0相当于内核态；R3相当于用户态；R1和R2则介于两者之间
- 不同级别能够运行的指令集合不同

- ◎ 目前大多数基于x86处理器的操作系统只用了R0和R3两个特权级别

CPU状态之间的转换

- ◉ 用户态 → 内核态

唯一途径 → 中断/异常/陷入机制

- ◉ 内核态 → 用户态

设置程序状态字PSW

因为内核态也被称为supervisor mode

一条特殊的指令：陷入指令（又称访管指令）

提供给用户程序的接口，用于调用操作系统的功能（服务）

例如：int, trap, syscall, sysenter/sysexit

操作系统的驱动力……

中断与异常机制

INTERRUPT EXCEPTION

中断/异常机制

- ◎ 中断/异常 对于操作系统的重要性
就好比：汽车的发动机、飞机的引擎

→→ 可以说 操作系统
是由“**中断驱动**”或者“**事件驱动**”的

主要作用

- ◎ 及时处理设备发来的中断请求
- ◎ 可使OS捕获用户程序提出的服务请求
- ◎ 防止用户程序执行过程中的破坏性活动
- ◎ 等等

中断/异常的概念

何时?
何处?

硬件完成
这一过程

在以后某个
时刻继续

- ◎ CPU对系统发生的某个事件作出的一种反应
- ◎ CPU暂停正在执行的程序，保留现场后自动转去
执行相应事件的处理程序，处理完成后返回断点，
继续执行被打断的程序

• 事件的发生改变了
处理器的控制流

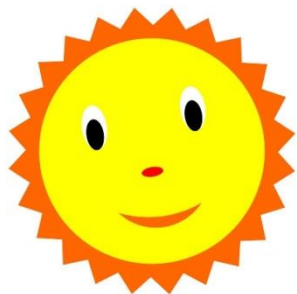
特点:

- 是随机发生的
- 是自动处理的
- 是可恢复的

为什么引入中断与异常？

历史
背景

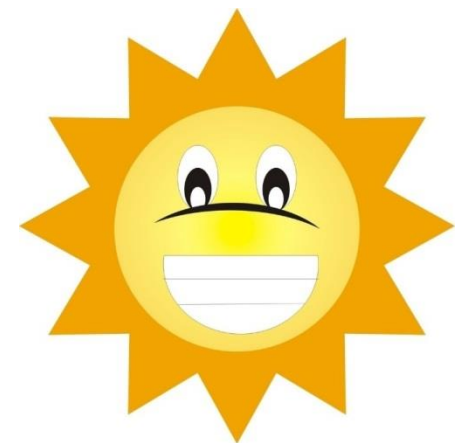
- ◎ 中断的引入：为了支持CPU和设备之间的并行操作
 - 当CPU启动设备进行输入/输出后，设备便可以独立工作，CPU转去处理与此次输入/输出不相关的事情；当设备完成输入/输出后，通过向CPU发中断报告此次输入/输出的结果，让CPU决定如何处理以后的事情
- ◎ 异常的引入：表示CPU执行指令时本身出现的问题
 - 如算术溢出、除零、取数时的奇偶错，访存地址时越界或执行了“陷入指令”等，这时硬件改变了CPU当前的执行流程，转到相应的错误处理程序或异常处理程序或执行系统调用



操作系统 要好好学习

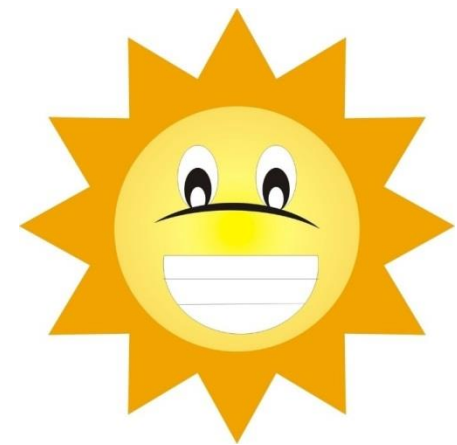
今天要学习的
内容是中断



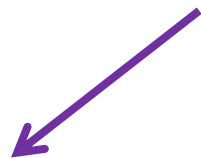


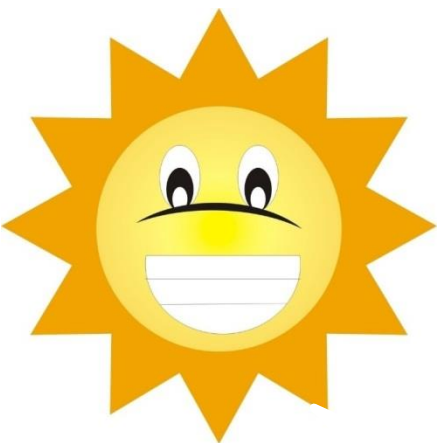
哎呀，来电话了！
接个电话先
再来看书





放个书签
在这里



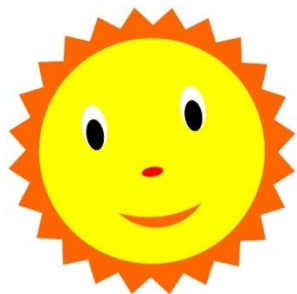


请叫我
书签大人



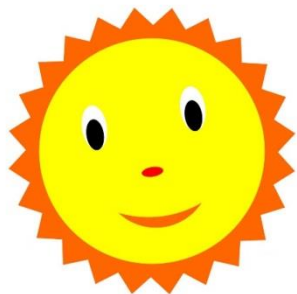
处理
电话





接着学习
中断知识





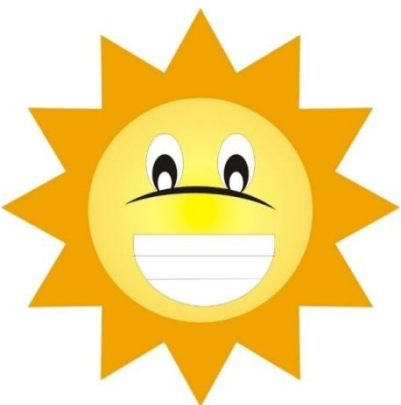
操作系统
要好好学习

今天要学习的内容
是异常

在这里要
放个书签

口渴了
喝口水再来学

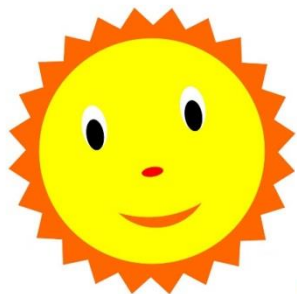




异常
处理

请叫我
书签大人





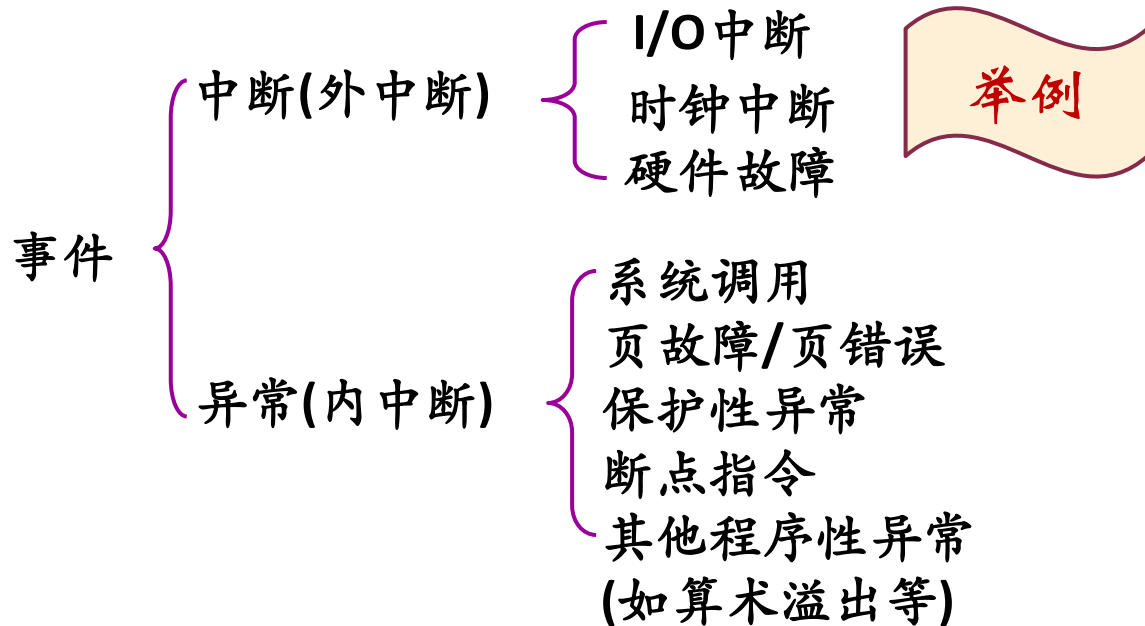
接着学习
异常知识



事件

中断：外部事件，正在运行的程序所不期望的

异常：由正在执行的指令引发



中断与异常的小结

类别	原因	异步/同步	返回行为
中断 Interrupt	来自I/O设备、其他 硬件部件	异步	总是返回到下一条指令
陷入Trap	有意识安排的	同步	返回到下一条指令
故障Fault	可恢复的错误	同步	返回到当前指令
终止Abort	不可恢复的错误	同步	不会返回

硬件做什么？ 软件怎么做？

中断 / 异常机制工作原理

中断/异常机制工作原理

- ◎ 中断/异常机制是现代计算机系统的核心机制之一
硬件和软件相互配合而使计算机系统得以充分发挥能力
- ◎ 硬件该做什么事？ —— 中断/异常响应
捕获中断源发出的中断/异常请求，以一定方式响应，将处理器控制权交给特定的处理程序
- ◎ 软件要做什么事？ —— 中断/异常处理程序
识别中断/异常类型并完成相应的处理

中断响应

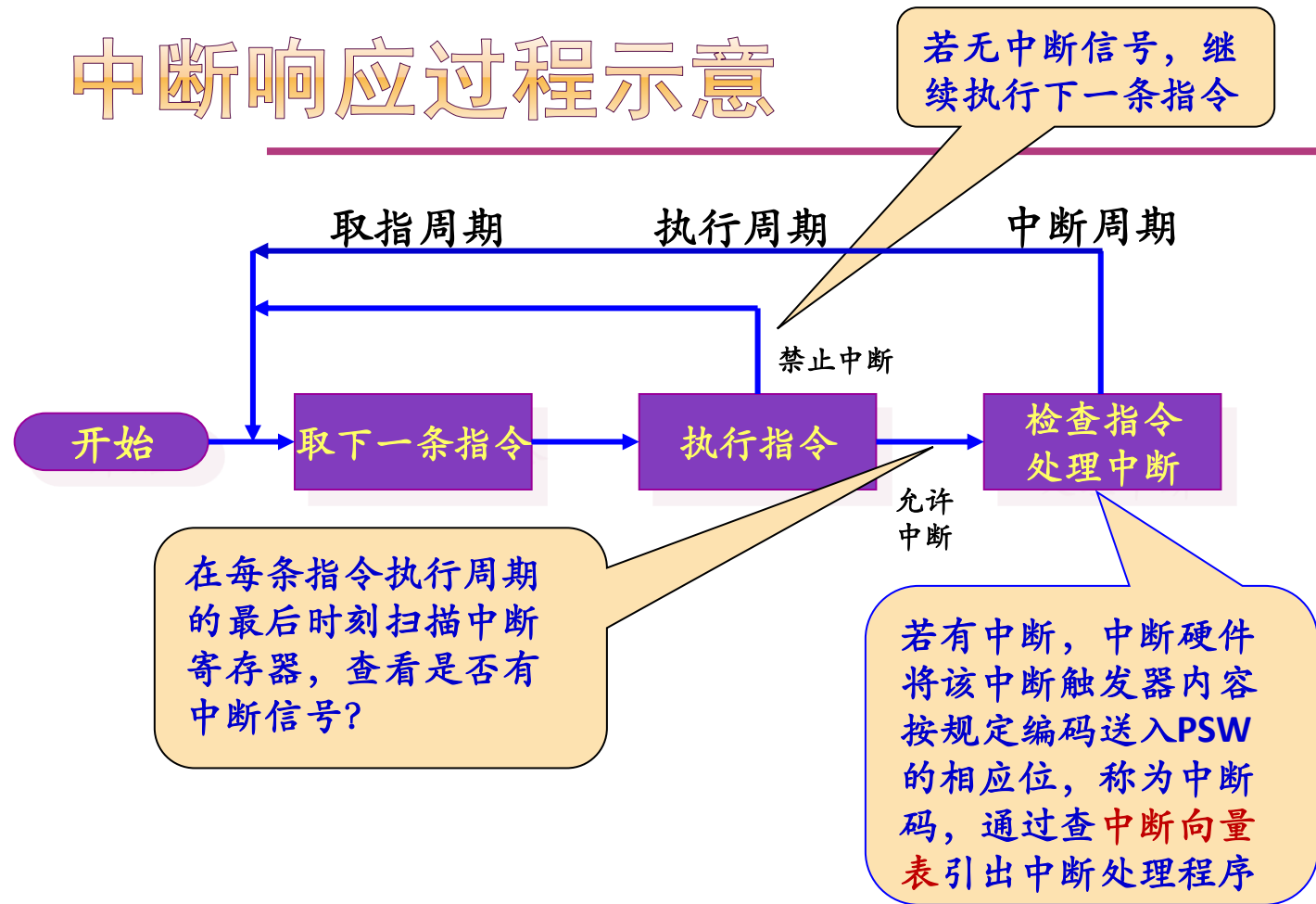
中断响应：

发现中断、接收中断的过程
由中断硬件部件完成

处理器控制部件中 设有 中断寄存器

CPU何时响应中断？

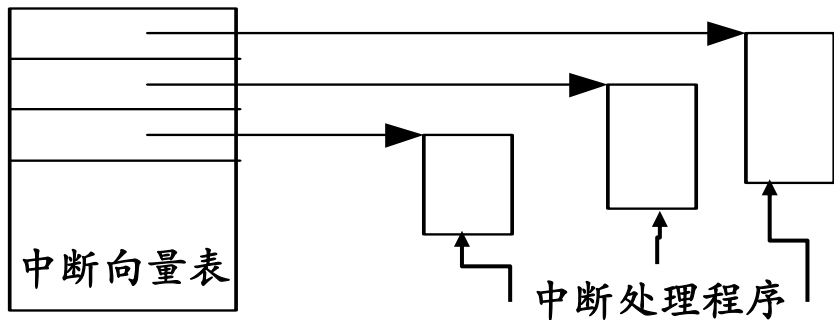
中断响应过程示意



中断向量表

◎ 中断向量：

一个内存单元，存放中断处理程序入口地址和程序运行时所需的处理机状态字



执行流程按中断号/异常类型的不同，通过中断向量表转移控制权给中断处理程序

LINUX中的中断向量表

向量范围	用途
0~19	不可屏蔽中断和异常
20~31	Intel保留
32~127	外部中断 (IRQ)
128 (0x80)	用于系统调用的可编程异常
129~238	外部中断
239	本地APIC时钟中断
240	本地APIC高温中断
241~250	Linux保留
251~253	处理器间中断
254	本地APIC错误中断
255	本地APIC伪中断

不可屏蔽中断/异常

- 0 -- 除零
- 1 -- 单步调试
- 4 -- 算术溢出
- 6 -- 非法操作数
- 12 -- 栈异常
- 13 -- 保护性错误
- 14 -- 缺页异常

中断响应示意图

① 设备发中断信号

③ 根据中断码查表

⑤ 执行中断处理程序

② 硬件保存现场

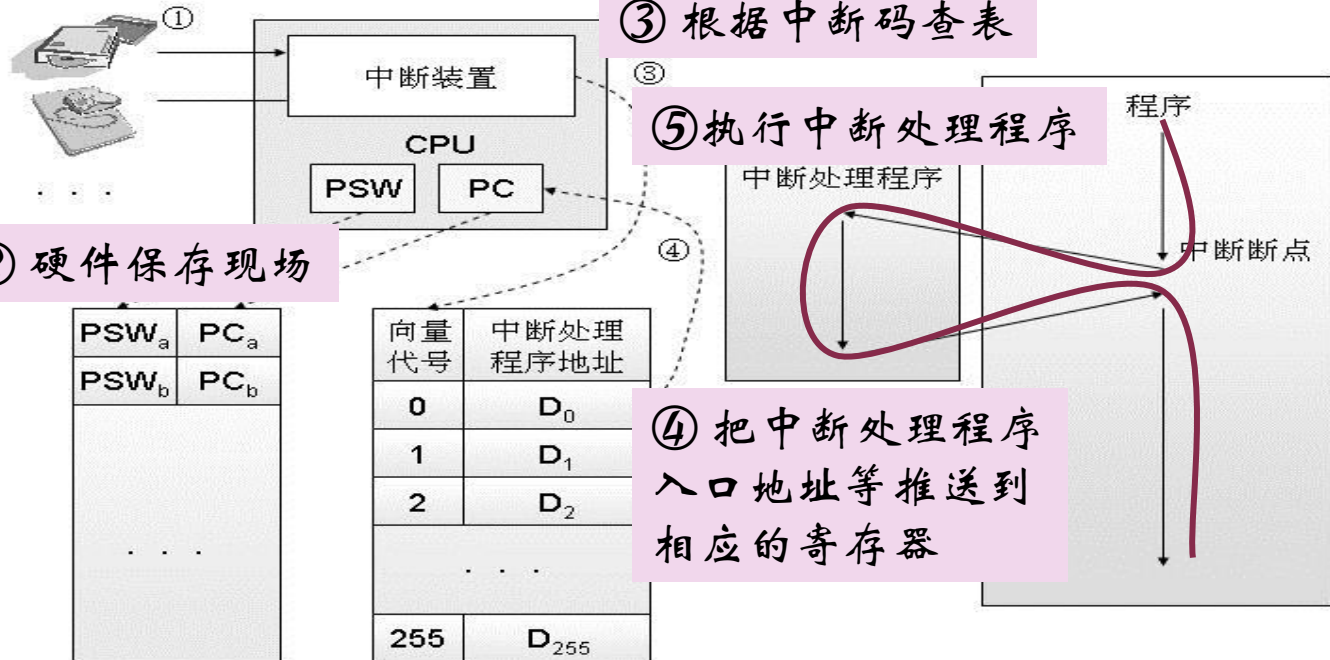
PSW _a	PC _a
PSW _b	PC _b
...	

系统堆栈

向量代号	中断处理程序地址
0	D ₀
1	D ₁
2	D ₂
...	
255	D ₂₅₅

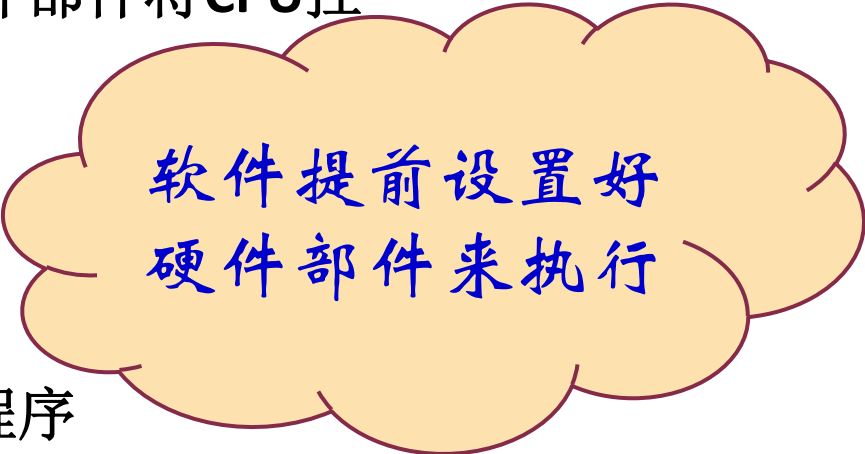
中断向量表

④ 把中断处理程序入口地址等推送到相应的寄存器



中断处理程序

- 设计操作系统时，为每一类中断/异常事件编好相应的处理程序，并设置好中断向量表
- 系统运行时若响应中断，中断硬件部件将CPU控制权转给中断处理程序：
 - 保存相关寄存器信息
 - 分析中断/异常的具体原因
 - 执行对应的处理功能
 - 恢复现场，返回被事件打断的程序



软件提前设置好
硬件部件来执行

中断/异常机制小结 (1/2)

以设备输入输出中断为例：

- ◎ 打印机给**CPU**发中断信号
- ◎ **CPU**处理完当前指令后检测到中断，判断出中断来源并向相关设备发确认信号
- ◎ **CPU**开始为软件处理中断做准备：
 - 处理器状态被切换到内核态
 - 在系统栈中保存被中断程序的重要上下文环境，主要是**程序计数器PC**、**程序状态字PSW**



硬件



硬件

小结 (2/2)

硬件

- ◎ CPU根据中断码查中断向量表，获得与该中断相关的处理程序的入口地址，并将PC设置成该地址，新的指令周期开始时，CPU控制转移到中断处理程序
- ◎ 中断处理程序开始工作
 - 在系统栈中保存现场信息
 - 检查I/O设备的状态信息，操纵I/O设备或者在设备和内存之间传送数据等等
- ◎ 中断处理结束时，CPU检测到中断返回指令，从系统栈中恢复被中断程序的上下文环境，CPU状态恢复成原来的状态，PSW和PC恢复成中断前的值，CPU开始一个新的指令周期

软件

硬件

举例：I/O中断处理程序

通常分为两类处理：

◎ I/O操作正常结束

- 若有程序正等待此次I/O的结果，则应将其唤醒
- 若要继续I/O操作，需要准备好数据重新启动I/O

◎ I/O操作出现错误

- 需要重新执行失败的I/O操作
- 重试次数有上限，达到时系统将判定硬件故障

X86 处理器对中断/异常的支持

中断/异常机制实例

基本概念——X86处理器

◎ 中断

- 由硬件信号引发的，分为可屏蔽和不可屏蔽中断

◎ 异常

- 由指令执行引发的，比如除零异常
- **80x86**处理器发布了大约**20**种不同的异常
- 对于某些异常，**CPU**会在执行异常处理程序之前产生硬件出错码，并压入内核态堆栈

◎ 系统调用

- 异常的一种，用户态到内核态的唯一入口

X86处理器对中断的支持(1/6)

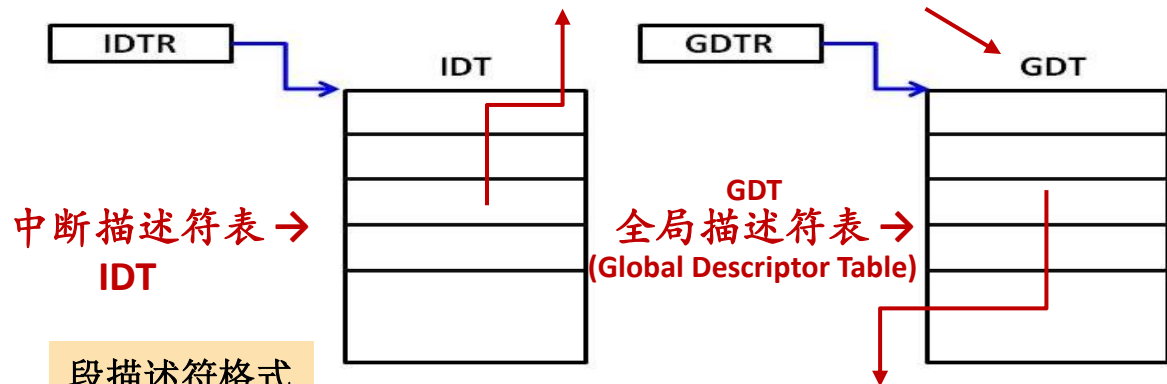
- ◎ 中断控制器（PIC或APIC）
 - 负责将硬件的中断信号转换为中断向量，并引发CPU中断
- ◎ 实模式：中断向量表 (Interrupt Vector)
 - 存放中断服务程序的入口地址
 - $\text{入口地址} = \text{段地址左移4位} + \text{偏移地址}$
 - 不支持CPU运行状态切换
 - 中断处理与一般的过程调用相似
- ◎ 保护模式：中断描述符表 (Interrupt Descriptor Table)
采用门(gate) 描述符数据结构表示中断向量

X86处理器对中断的支持(2/6)

- 中断向量表/中断描述符表
 - 四种类型门描述符
 - 任务门(Task Gate)
 - 中断门(Interrupt Gate)
 - 给出段选择符 (Segment Selector)、中断/异常程序的段内偏移量 (Offset)
 - 通过中断门后系统会自动禁止中断
 - 陷阱门(Trap Gate)
 - 与中断门类似，但通过陷阱门后系统不会自动禁止中断
 - 调用门(Call Gate)

X86处理器对中断的支持(3/6)

中断描述符格式



中断服务程序
入口地址 =
段基址 + 偏移

段描述符格式



段选择符格式



X86处理器对中断的支持(4/6)

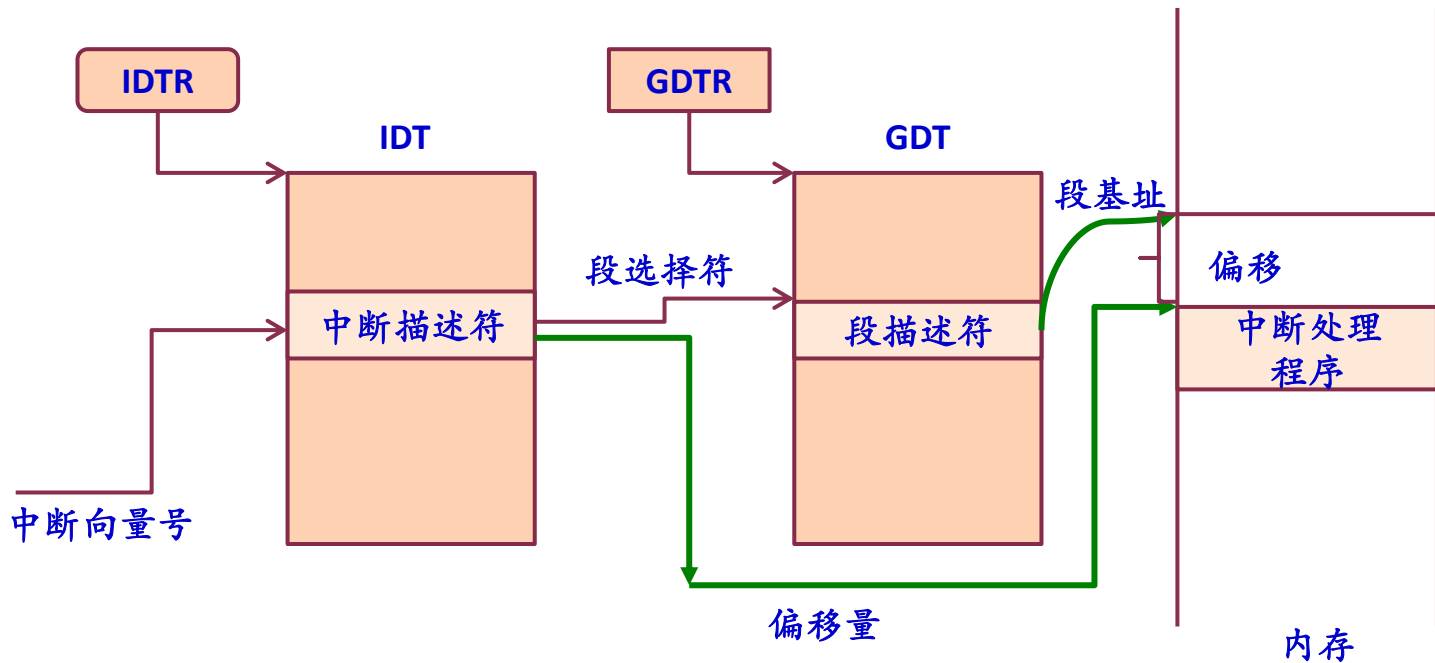
中断/异常的硬件处理过程:

- ◎ 确定与中断或异常关联的向量i
- ◎ 通过IDTR寄存器找到IDT表，获得中断描述符（表中的第i项）
- ◎ 从GDTR寄存器获得GDT的地址；结合中断描述符中的段选择符，在GDT表获取对应的段描述符；从该段描述符中得到中断或异常处理程序所在的段基址
- ◎ 特权级检查

X86处理器对中断的支持(5/6)

- ◎ 检查是否发生了特权级的变化，如果是，则进行堆栈切换(必须使用与新的特权级相关的栈)
- ◎ 硬件压栈，保存上下文环境；如果异常产生了硬件出错码，也将它保存在栈中
- ◎ 如果是中断，清IF位
- ◎ 通过中断描述符中的段内偏移量和段描述符中的基地址，找到中断/异常处理程序的入口地址，执行其第一条指令

X86处理器对中断的支持(6/6)



操作系统向用户程序提供的接口

系统调用机制

系统调用(SYSTEM CALL)

Linux操作系统
提供多少个系
统调用？

系统调用是什么？

系统调用：用户在编程时可以调用的操作系统功能

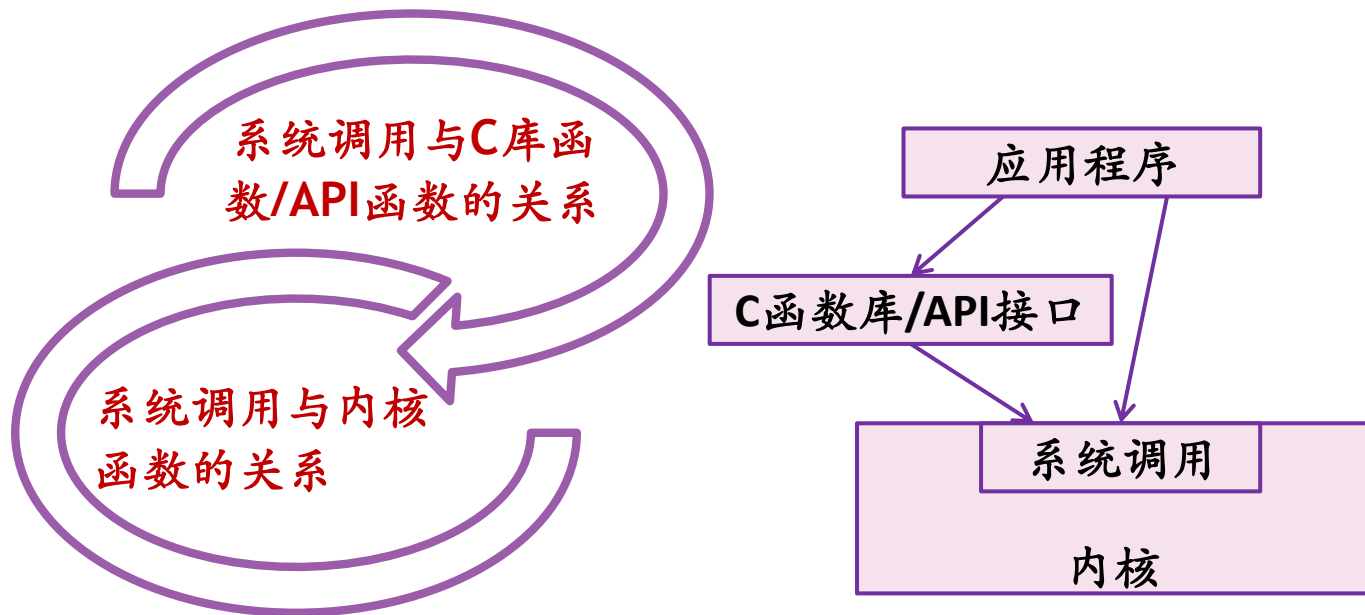
系统调用的作用

- 系统调用是操作系统提供给编程人员的唯一接口
- 使CPU状态从用户态陷入内核态

典型系统调用
举例

每个操作系统都提供几百种系统调用（进程控制、进程通信、文件使用、目录操作、设备管理、信息维护等）

系统调用、库函数、API、内核函数



系统调用机制设计 与 执行过程

系统调用机制的设计

中断/异常机制

支持系统调用服务的实现 ①

选择一条特殊指令：陷入指令(亦称访管指令)

引发异常，完成用户态到内核态的切换 ②

系统调用号和参数

每个系统调用都事先给定一个编号(功能号) ③

系统调用表

存放系统调用服务例程的入口地址 ④

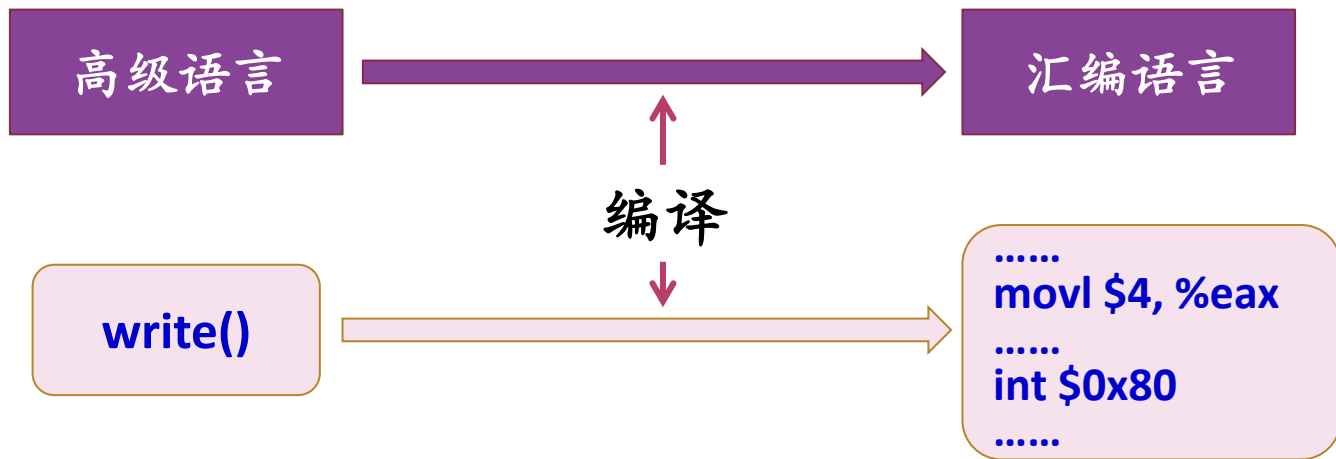
参数传递过程问题

- ◎ 怎样实现用户程序的参数传递给内核？

常用的3种实现方法：

- ◎ **由陷入指令自带参数**：陷入指令的长度有限，且还要携带系统调用功能号，只能自带有限的参数
- ◎ **通过通用寄存器传递参数**：这些寄存器是操作系统和用户程序都能访问的，但寄存器的个数会限制传递参数的数量
- ◎ **在内存中开辟专用堆栈区**来传递参数

系统调用举例(1/3)



系统调用举例(2/3)

```
#include <unistd.h>
```

```
int main(){
```

```
    char string[5] = {'H', 'e', 'l', 'l', 'o', '!', '\n'};
```

```
    write(1, string, 7);
```

```
    return 0;
```

```
}
```

输出结果: Hello!



高级语言视角

系统调用举例(3/3)

汇编语言视角

```
1. .section .data
2. output:
3.     .ascii "Hello!\n"
4. output_end:
5.     .equ len, output_end - output
```

```
6. .section .text
7. .globl _start
8. _start:
```

```
9.     movl $4, %eax
10.    movl $1, %ebx
11.    movl $output, %ecx
12.    movl $len, %edx
13.    int $0x80
```

eax存放系统调用号

```
14. end:
```

引发一次系统调用

```
15.    movl $1, %eax
16.    movl $0, %ebx
17.    int $0x80
```

1这个系统调用的作用?

系统调用的执行过程

当CPU执行到特殊的陷入指令时：

- ◎ **中断/异常机制**：硬件保护现场；通过查中断向量表把控制权转给系统调用总入口程序
- ◎ **系统调用总入口程序**：保存现场；将参数保存在内核堆栈里；通过查系统调用表把控制权转给相应的系统调用处理例程或内核函数
- ◎ **执行系统调用例程**
- ◎ **恢复现场，返回用户程序**

基于x86处理器

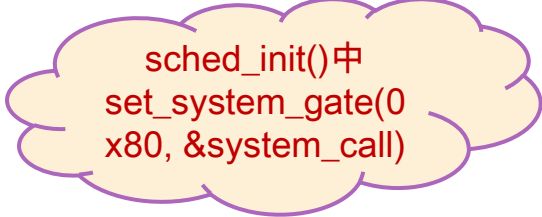
例子:

LINUX系统调用实现

LINUX的系统调用实现

——基于X86处理器

- 陷入指令选择128号
`int $0x80`



```
sched_init()中  
set_system_gate(0  
x80, &system_call)
```

- 门描述符

- 系统初始化时：对IDT表中的128号门初始化
- 门描述符的2、3两个字节：内核代码段选择符
- 0、1、6、7四个字节：偏移量（指向`system_call()`）
- 门类型：`15`，陷阱门，为什么？
- DPL：`3`，与用户级别相同，允许用户进程使用该门描述符

系统调用号示例

(INCLUDE/ASM-I386/UNISTD.H)

#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
#define __NR_unlink	10
#define __NR_execve	11
#define __NR_chdir	12
#define __NR_time	13

...

系统执行INT \$0X80指令

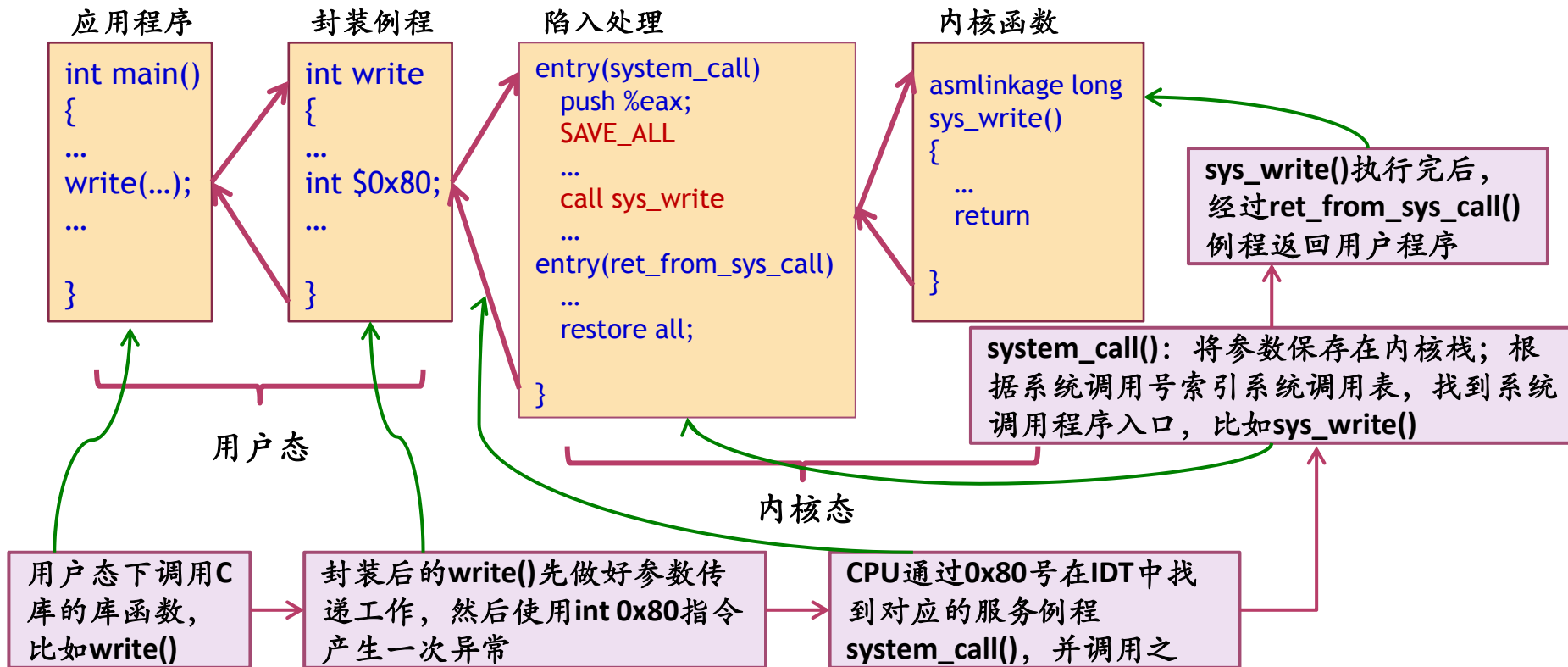
- 由于特权级的改变，要切换栈

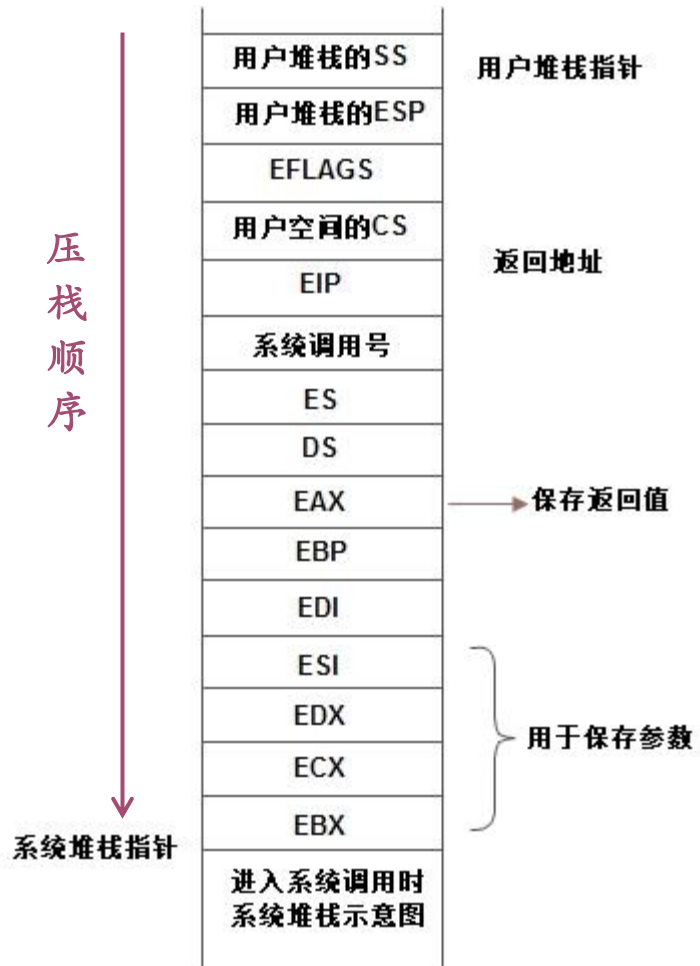
用户栈 → 内核栈

CPU从任务状态段TSS中装入新的栈指针（SS：ESP），指向内核栈

- 用户栈的信息（SS：ESP）、EFLAGS、用户态CS、EIP 寄存器的内容压栈（返回用）
- 将EFLAGS压栈后，复位TF，IF位保持不变
- 用128在IDT中找到该门描述符，从中找出段选择符装入代码段寄存器CS
- 代码段描述符中的基地址 + 陷阱门描述符中的偏移量 → 定位 system_call()的入口地址

LINUX系统调用执行流程





```
#define SAVE_ALL \
    cld; \
    pushl %es; \
    pushl %ds; \
    pushl %eax; \
    pushl %ebp; \
    pushl %edi; \
    pushl %esi; \
    pushl %edx; \
    pushl %ecx; \
    pushl %ebx; \
    movl $(__USER_DS), %edx; \
    movl %edx, %ds; \
    movl %edx, %es;
```

中断发生后OS低层工作步骤

1. 硬件压栈：程序计数器等
2. 硬件从中断向量装入新的程序计数器等
3. 汇编语言过程保存寄存器值
4. 汇编语言过程设置新的堆栈
5. C语言中断服务程序运行（例：读并缓冲输入）
6. 进程调度程序决定下一个将运行的进程
7. C语言过程返回至汇编代码
8. 汇编语言过程开始运行新的当前进程

教材第52页的图2-5

本讲重点

- ◎ 理解计算机系统的保护机制
 - 掌握处理器状态
 - 掌握特权指令与非特权指令
- ◎ 掌握中断/异常机制
 - 掌握中断/异常的基本概念
 - 理解中断/异常机制的工作原理
- ◎ 掌握系统调用机制
 - 掌握系统调用设计原理
 - 掌握系统调用执行过程

本周要求

◎ 重点阅读教材

第1章相关内容：1.3、1.6

第2章 第52页 图2-5及说明该图思路的段落

◎ 重点概念

CPU状态 内核态/用户态 特权指令/非特权指令
中断 异常 中断响应 中断向量 中断处理程序
系统调用 陷入指令 系统调用号 系统调用表

THANKS

The End