



第2章 操作系统的运行环境和运行机制

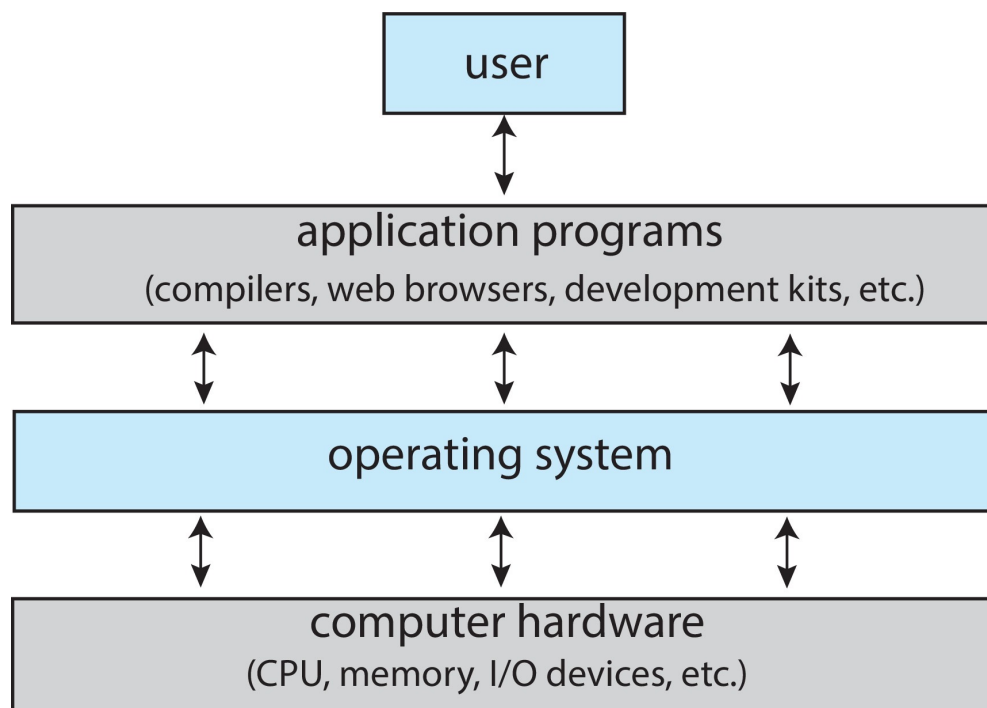
■ 本章内容

- 1. 计算机硬件系统对操作系统的支持，重要寄存器的作用
- 2. 中断的基本知识
- 3. 中断/异常机制的实现
- 4. 系统调用及其实现





计算机系统的抽象结构



■ 用户

- 人, 机器, 其它计算机

■ 应用程序

- Word, 编译器, web 浏览器, 数据库系统, 视频游戏

■ 操作系统

- 控制和协调不同应用程序和用户之间对硬件的使用

■ 硬件

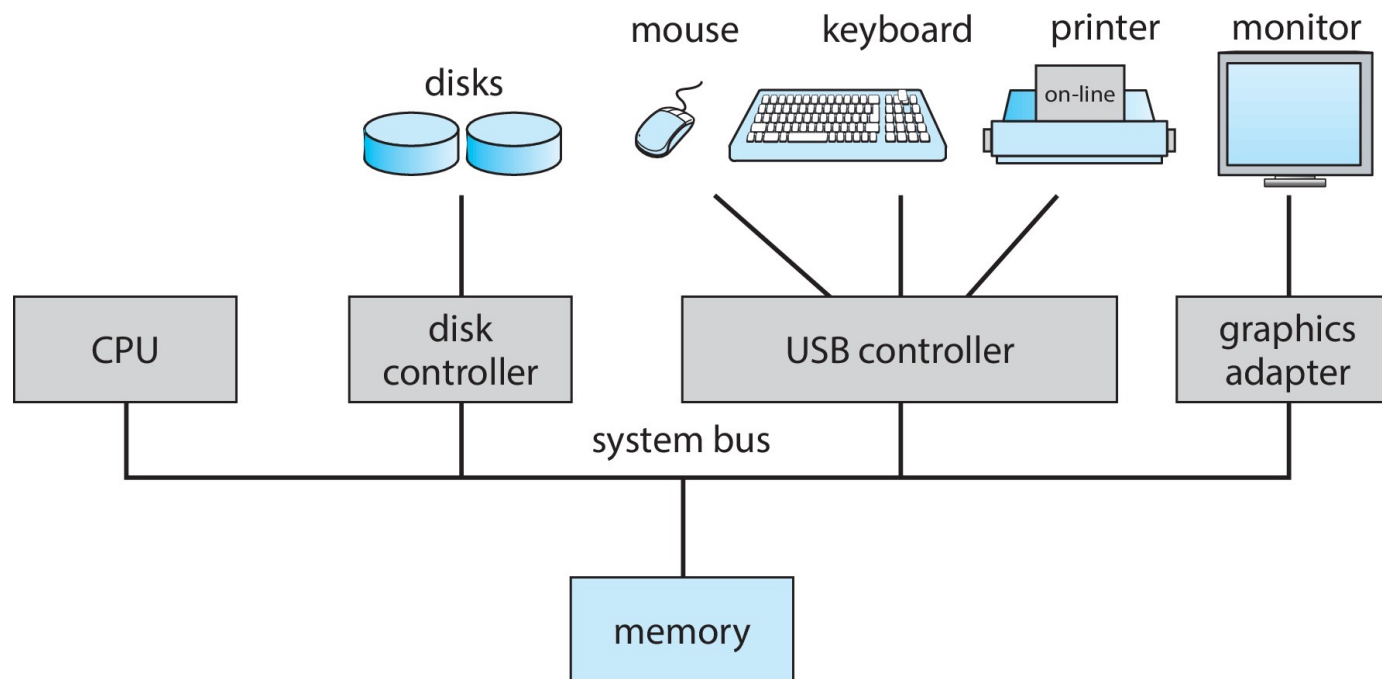
- CPU, 内存, I/O 设备





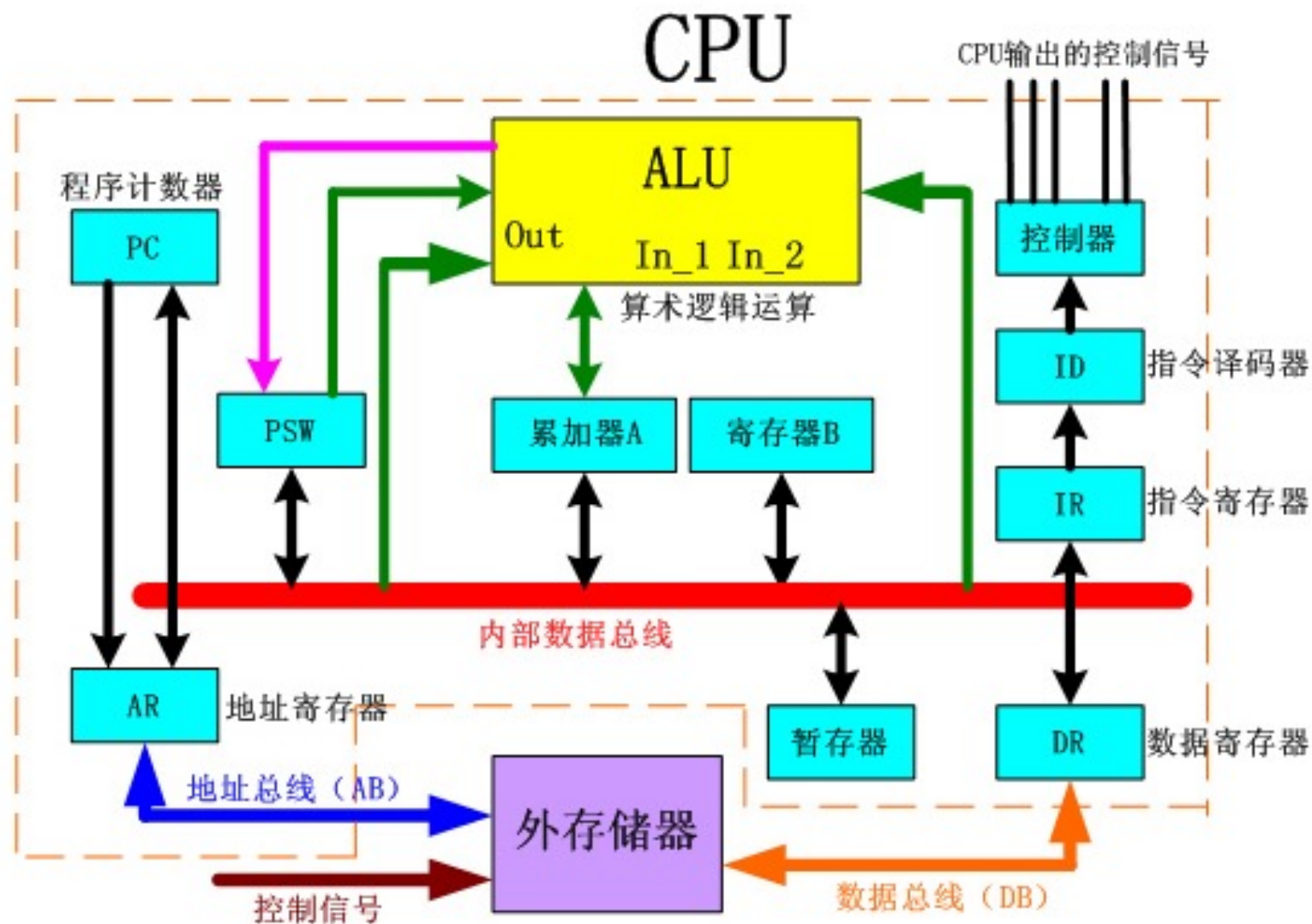
计算机系统的硬件结构

- CPU、设备控制器通过公共总线连接，从而可以访问共享内存
- CPU和设备并发执行，竞争内存周期





CPU的内部结构图





CPU中的重要寄存器

- **程序计数器（PC）**

- 用于记录下一条要执行的指令的地址。当处理器执行完一条指令后，PC寄存器会自动更新为下一条指令的地址。

- **指令寄存器（IR）**

- IR寄存器用于记录最近取出并解码的指令。

- **程序状态字（PSW）寄存器**

- PSW寄存器用于记录处理器的运行状态，如条件码、模式、控制位等。PSW寄存器中的一些位可以影响处理器的行为，如中断允许位、处理器状态位等。

- **通用寄存器（General Registers）**

- 通用寄存器是一组可以被用户程序和操作系统程序使用的寄存器，用于保存数据或地址。

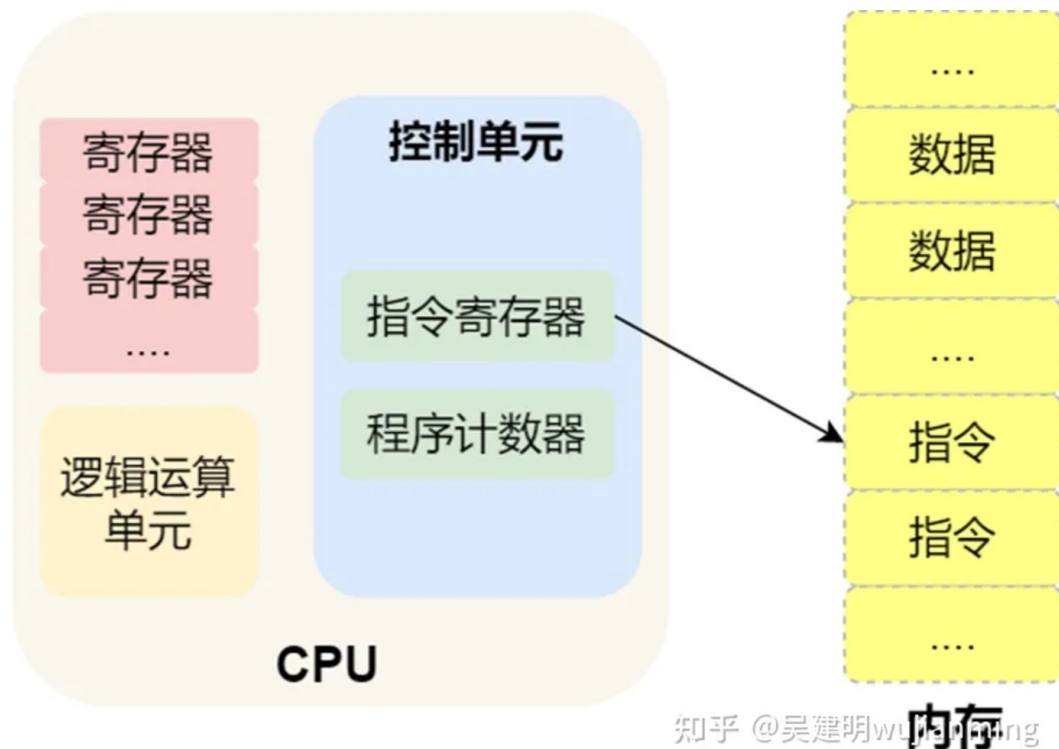
- **控制寄存器（Control Registers）**

- 控制寄存器是一组只能被操作系统程序使用的寄存器，用于控制和配置处理器的一些特性和功能。





CPU的工作流程



- **取指**：CPU读取程序计数器（PC）中的地址作为指令的地址，从内存中读取指令。
- **译码**：CPU将取得的指令进行译码，以确定它要执行的操作。
- **执行**：CPU执行指令的操作，可能涉及到算术、逻辑、移位等操作。
- **写回**：CPU将执行结果写回到寄存器或内存中。





CPU的状态

- 什么是CPU的状态？
 - CPU的态，又称为CPU的特权级，是CPU的工作状态。当前CPU正在执行哪类程序，决定CPU的态。
- 为什么要区分CPU的状态？

管理程序	用户程序
管理系统资源	使用资源，提出申请
控制程序运行	被控制

区分处理机状态的目的：保护操作系统





CPU状态的分类

- 内核态

- 操作系统的管理程序执行时机器所处的状态，在此状态下处理机可使用全部指令(包括一组特权指令)；使用全部系统资源(包括整个存储区域)。

- 用户态

- 用户程序执行时CPU所处的状态，在此状态下禁止使用特权指令，不能直接取用资源与改变机器状态，并且只允许用户程序访问自己的存储区域。





CPU的状态

■ 内核态与用户态的区别

内核态	用户态
操作系统的程序执行	用户程序执行
使用全部指令	禁止使用特权指令
使用全部系统资源 (包括整个存储区域)	只允许用户程序 访问自己的存储区域

■ CPU的特权指令集

- 涉及外部设备的输入/输出指令，修改特殊寄存器的指令，改变机器状态的指令





不同架构的CPU的特权级

■ ARM架构

- 定义了四个特权级别，分别是User（用户模式），FIQ（快速中断模式），IRQ（中断请求模式）和SVC（超级用户模式）。用户模式是普通应用程序运行的模式，而其他三种模式主要用于处理各种中断和异常。

■ RISC-V架构

- 定义了四种特权级别，分别是User（U），Supervisor（S），Hypervisor（H），Machine（M）。User级别用于运行用户程序，Supervisor级别用于运行操作系统内核，Hypervisor级别用于运行虚拟机管理器，Machine级别则是硬件级别，具有最高的特权。

■ x86架构

- 定义了四个特权级别，从Ring 0到Ring 3。其中，Ring 0具有最高的特权，通常用于运行操作系统内核。Ring 1和Ring 2很少使用，而Ring 3具有最低的特权，用于运行用户程序。





CPU特权级切换的三个场景

- 应用程序调用操作系统提供的系统调用，此时应用程序通过执行系统调用指令将CPU的特权级从用户态切换到内核态
- 应用程序执行一条指令触发了异常，导致CPU的特权级从用户态切换到内核态，比如访内存指令触发了缺页异常
- 应用程序执行过程中，CPU收到一条来自外设的中断，对中断的处理导致CPU的特权级从用户态切换到内核态

```
1 int main() {  
2     FILE *fp = fopen("test.txt", "r"); // 这里会发生什么?  
3     return 0;  
4 }
```





什么是中断/异常？

- 广义的中断：指CPU在执行程序过程中，由于发生了某种紧急事件，使得CPU暂停正在执行的程序，转而去执行一段特殊的服务程序，处理完毕后再返回原程序继续执行的过程。
- 中断 (Interrupt) - 来自CPU外部，异步事件：
 - 与当前指令执行无关，随机发生。
 - 示例：I/O设备完成操作、时钟信号、硬件故障、外部设备发来的信号。
- 异常 (Exception / Trap / 内中断) - 来自CPU内部，同步事件：
 - 是执行当前指令的结果，可预知。
 - 示例：除以零、地址越界、非法指令、缺页、系统调用指令(访管指令)、调试断点。





中断/异常的分类

- 外中断 (External Interrupts) / 异步中断:
 - I/O中断: 设备完成操作或出错。
 - 时钟中断: 定时器发出, 用于时间片轮转、计时等。
 - 控制台中断: 如用户按下Ctrl+C。
 - 硬件故障中断: 如电源掉电、内存校验错。





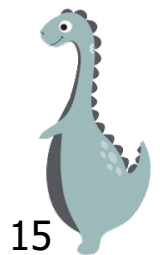
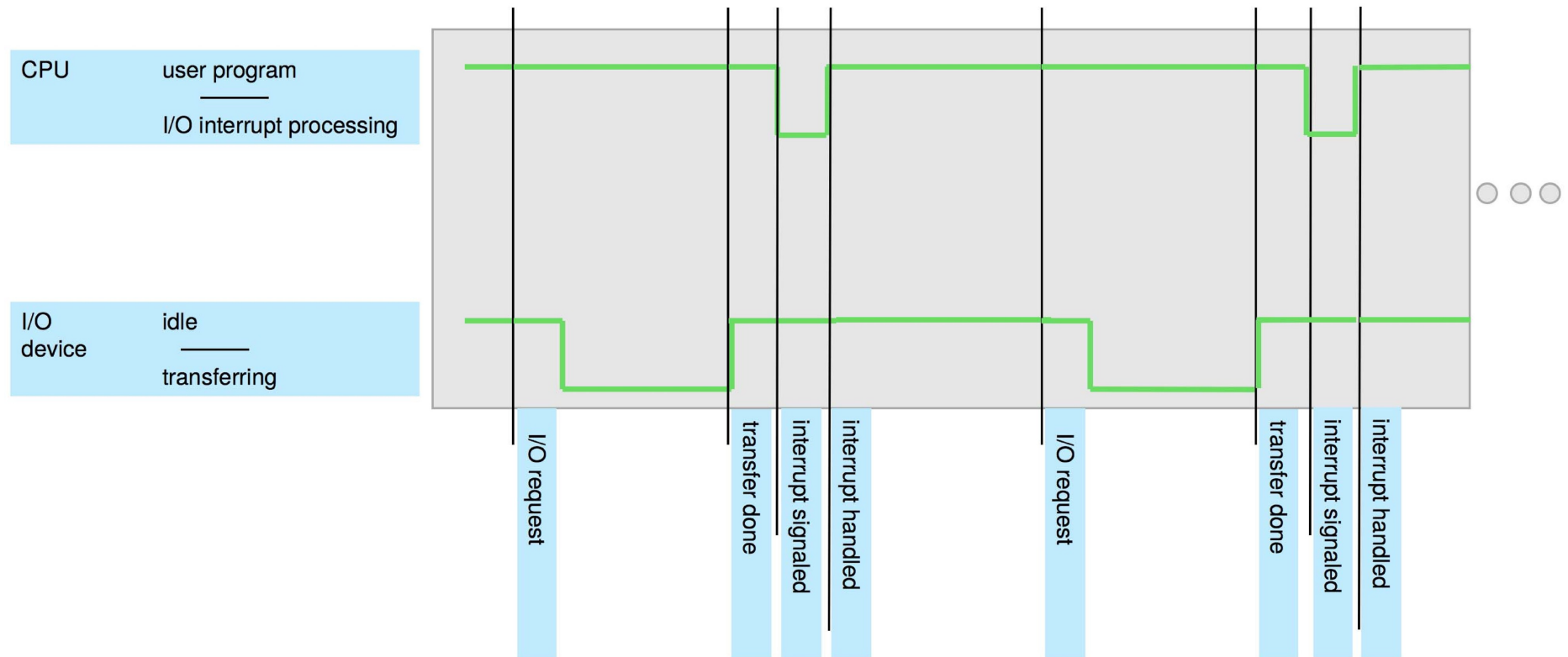
中断/异常的分类

- 内中断 (Internal Interrupts) / 异常 / 同步中断 / 陷入 (Trap):
 - 故障 (Fault): 可被修复的错误。处理后通常返回到引起故障的指令重新执行。
 - 示例: 缺页故障 (Page Fault), 段不存在。
 - 陷阱 (Trap): 有意设置的或程序自愿陷入。处理后通常返回到引起陷阱的指令的下一条指令执行。
 - 示例: 系统调用 (访管指令), 调试断点 (int 3), 单步执行。
- 终止 (Abort): 不可恢复的严重错误。通常导致当前程序或系统终止。
 - 示例: 控制器错误, CPU内部错误。



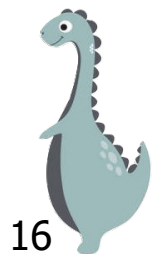
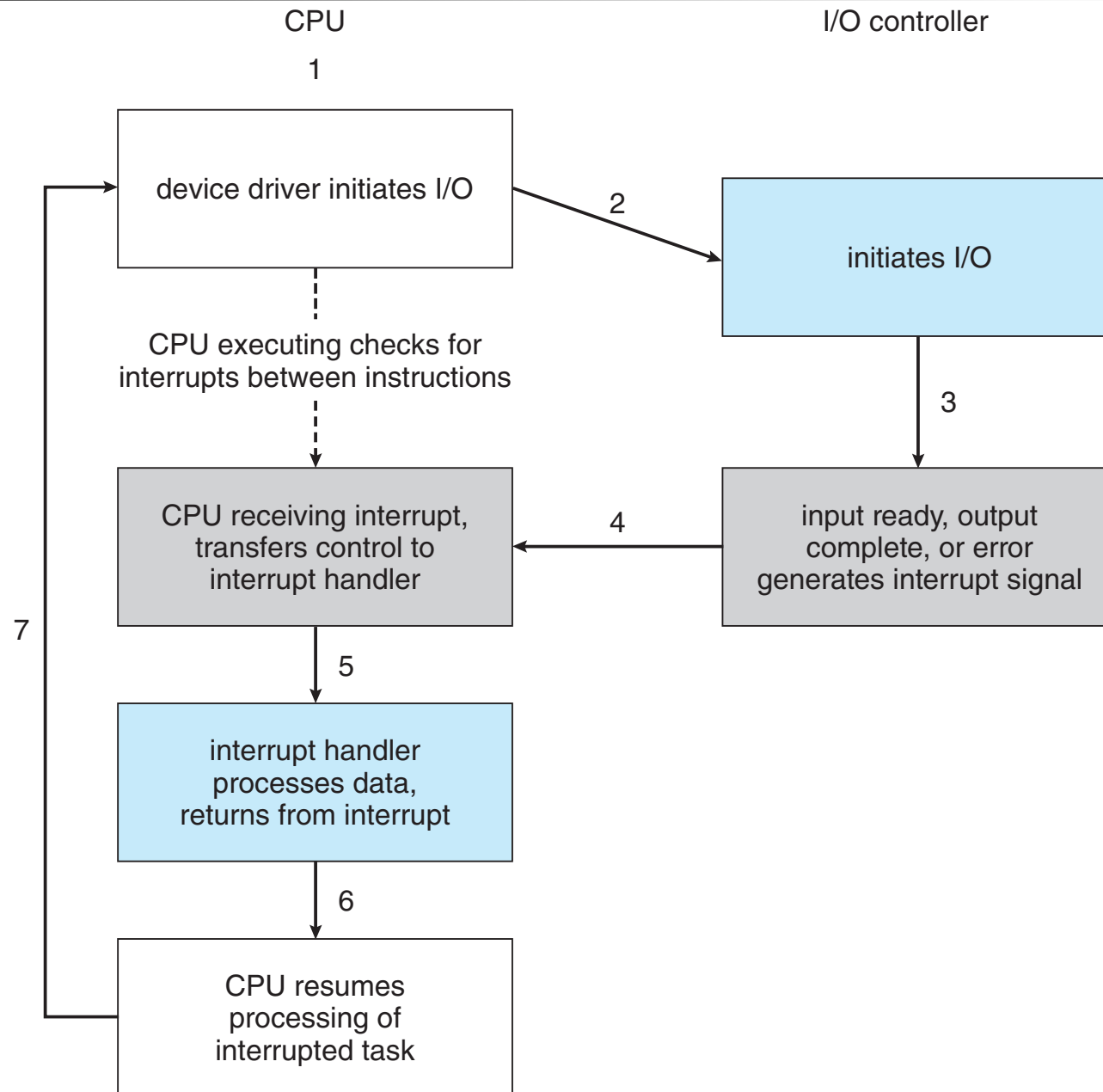


中断的时间线



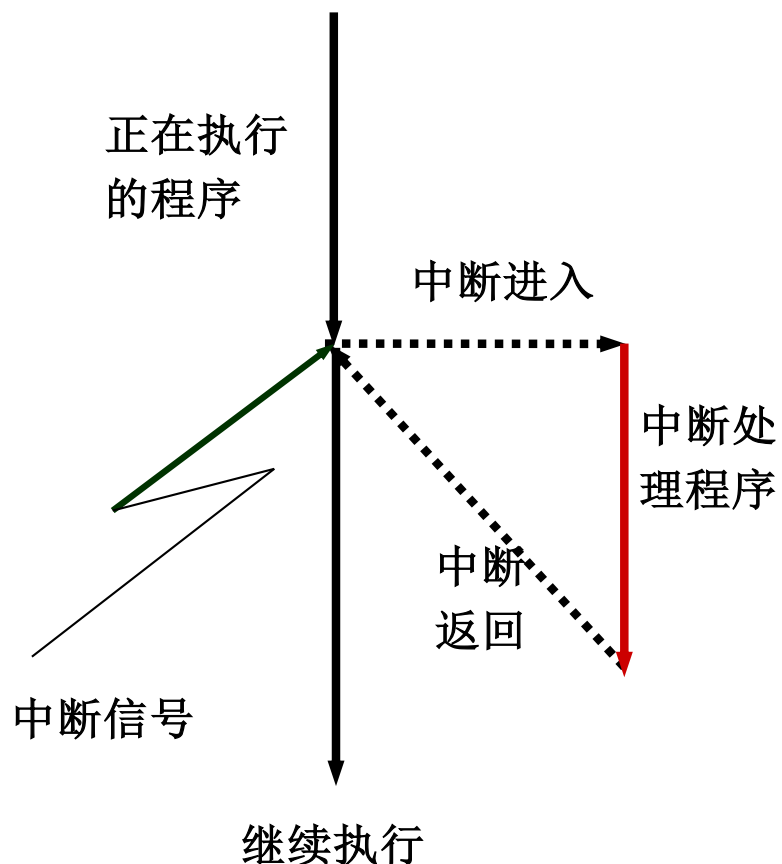


中断驱动的I/O循环





中断响应



中断概念示意图

- **中断响应**：当中央处理机CPU发现已有中断请求时，中止现行程序执行，并自动引出中断处理程序的过程。
- 中断响应需要硬件支持
 - PC—指令计数器
 - PSW—状态寄存器
 - 系统堆栈—内存的固定区域
 - 中断向量表—内存的固定区域





保存现场与恢复现场

■ 什么是现场？

- 在中断的那一时刻能确保程序继续运行的有关信息

- ① 后继指令所在主存的单元号

- ② 程序运行所处的状态

- ③ 指令执行情况

- ④ 程序执行的中间结果等





保存现场与恢复现场

■ 保存现场

- 当中断发生时，必须立即把现场信息保存在主存中，这一工作称之为保存现场

■ 恢复现场

- 程序重新运行之前，把保留的该程序现场信息从主存中送至相应的指令计数器、通用寄存器或一些特殊的寄存器中。完成这些工作称为恢复现场





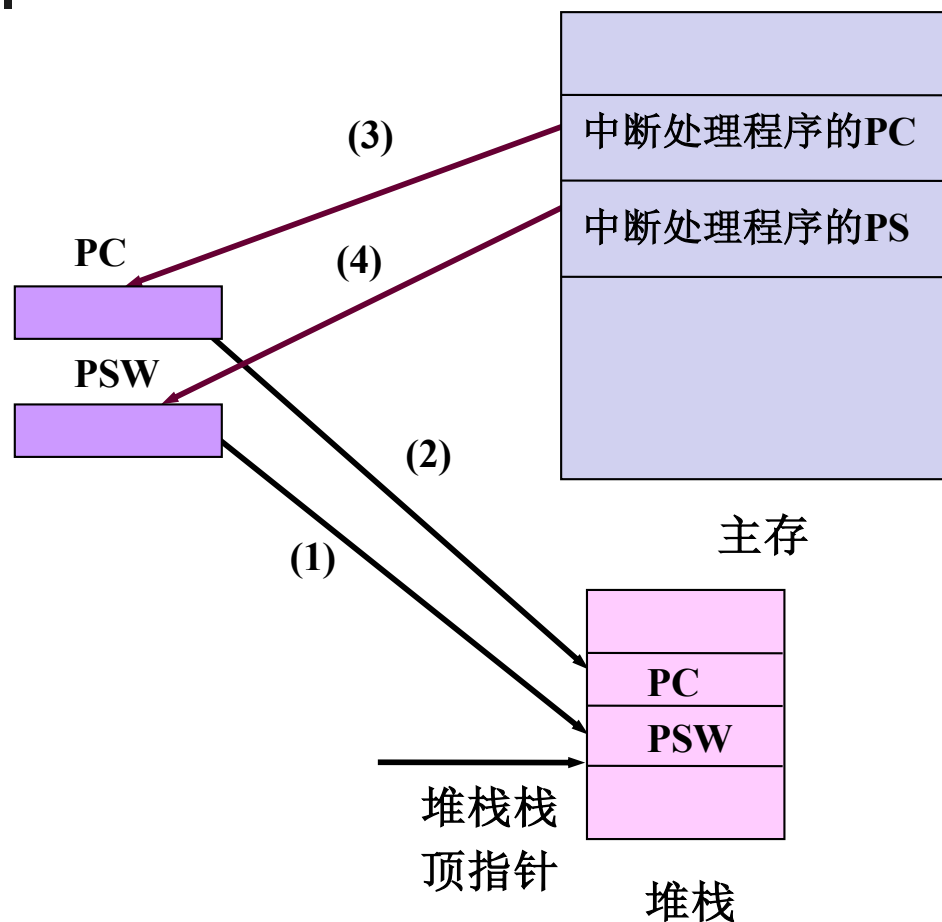
保存现场与恢复现场

- 程序状态字（PSW）
 - 当前程序执行时机器所处的状态信息
 - 通常存储在CPU的一个特殊寄存器中，不同的CPU架构有不同的定义
 - 通常包含以下信息
 - 条件码：记录最近一次算术或逻辑操作的结果，比如：结果是否为零、是否有进位或借位、是否溢出等
 - 中断使能/禁止位：决定是否允许响应中断
 - 用户/内核模式位：标识当前CPU处于用户模式还是内核模式
 - 其他特定于具体处理器架构的标志位





中断响应的详细过程



中断响应过程示意图

■ 中断响应过程

- 保留程序断点及CPU的状态信息
- 自动转入相应的中断处理程序

■ 中断响应的实质

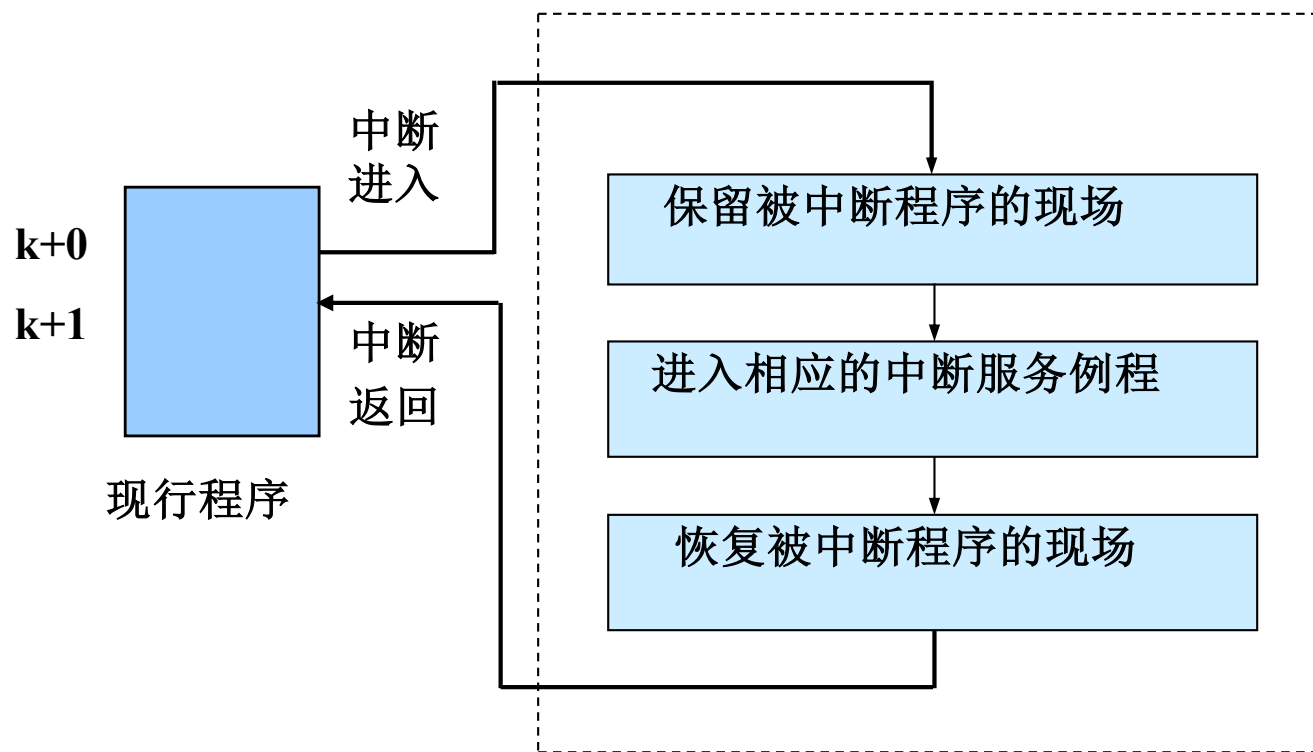
- 交换指令的地址和CPU的状态信息





中断处理程序

- 当硬件完成了中断进入过程后，由相应的中断处理程序得到控制权，进入了软件的中断处理过程



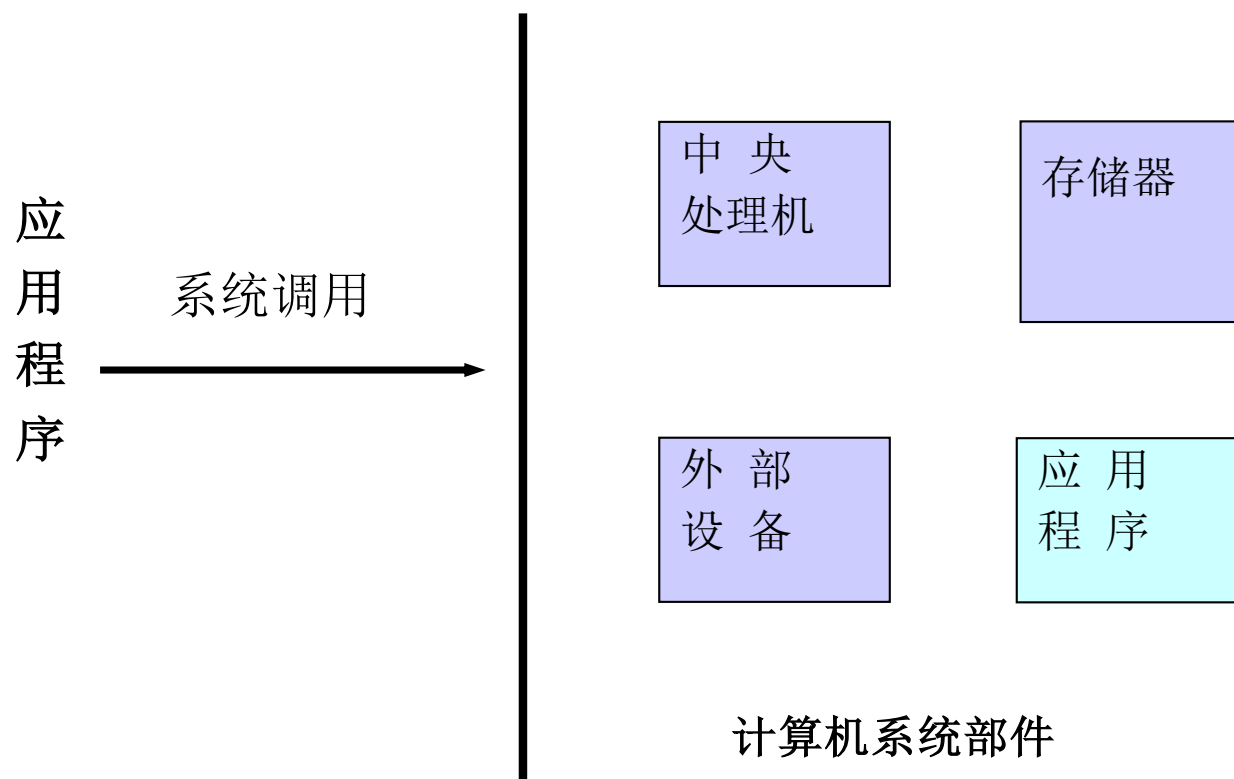
中断处理过程示意图





系统调用

■ 什么是系统调用？



系统调用示意图





系统调用

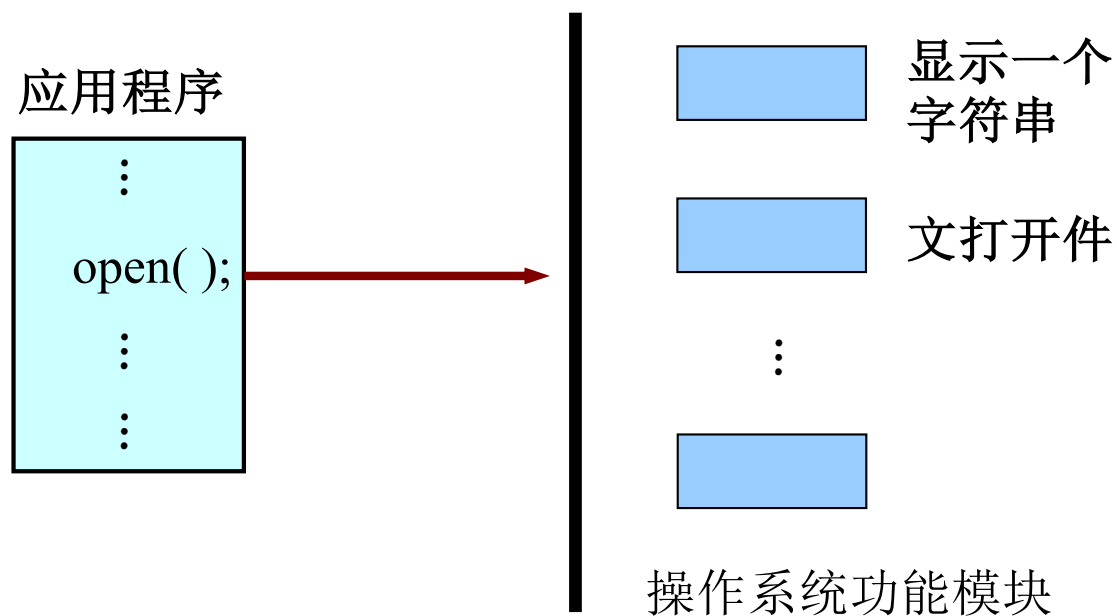
- 是操作系统提供的服务的编程接口
- 通常用高级语言（如 C 或 C++）编写
- 大多数程序通过高级应用程序编程接口（API）而不是直接使用系统调用
- 最常见的三个 API：Windows 的 Win32 API，基于 POSIX 的系统（包括几乎所有版本的 UNIX、Linux 和 Mac OS X）的 POSIX API，以及 Java 虚拟机（JVM）的 Java API





系统调用

■ 如何调用操作系统服务？



系统功能调用示意图

■ 通过API而不是直接访问系统调用





标准API示例

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the man page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

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

```
ssize_t read(int fd, void *buf, size_t count)
```

return
value

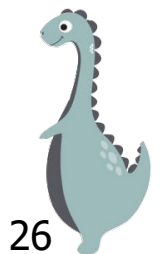
function
name

parameters

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd`—the file descriptor to be read
- `void *buf`—a buffer into which the data will be read
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns `-1`.





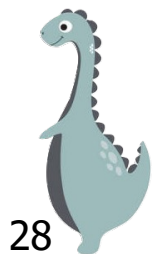
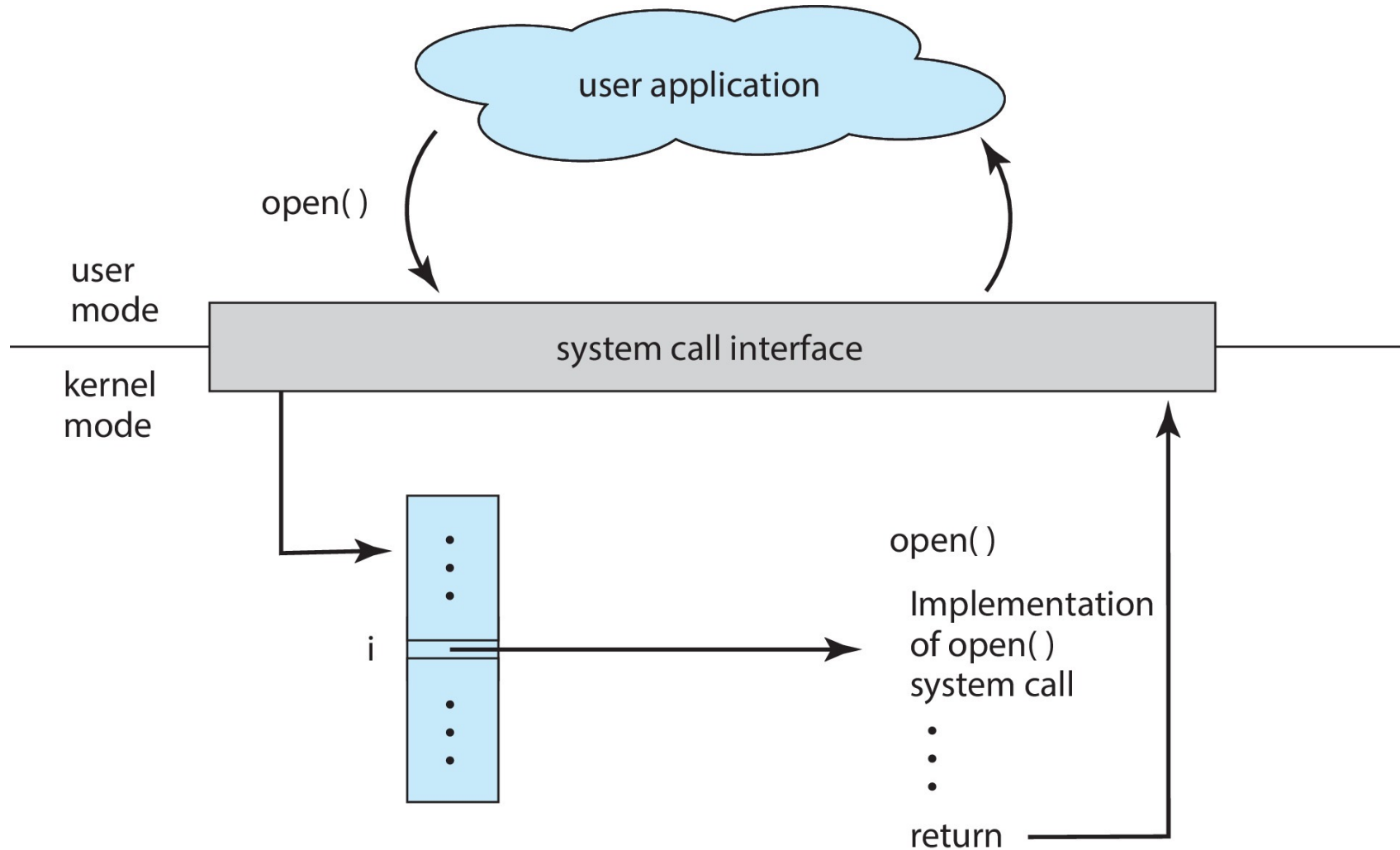
系统调用的实现

- 每个系统调用都与一个编号相关联
 - 系统调用接口根据这些编号维护一个索引表
- 系统调用接口在操作系统内核中调用所需的系统调用，并返回系统调用的状态和任何返回值
- 调用者不需要了解系统调用的具体实现方式
 - 只需要遵循 API 并理解操作系统对调用的结果会做什么
 - API 将操作系统接口的大部分细节隐藏起来，对程序员不可见
 - 由运行时支持库管理（一组内置于编译器所包含的库中的函数）





API: 打开文件的系统调用





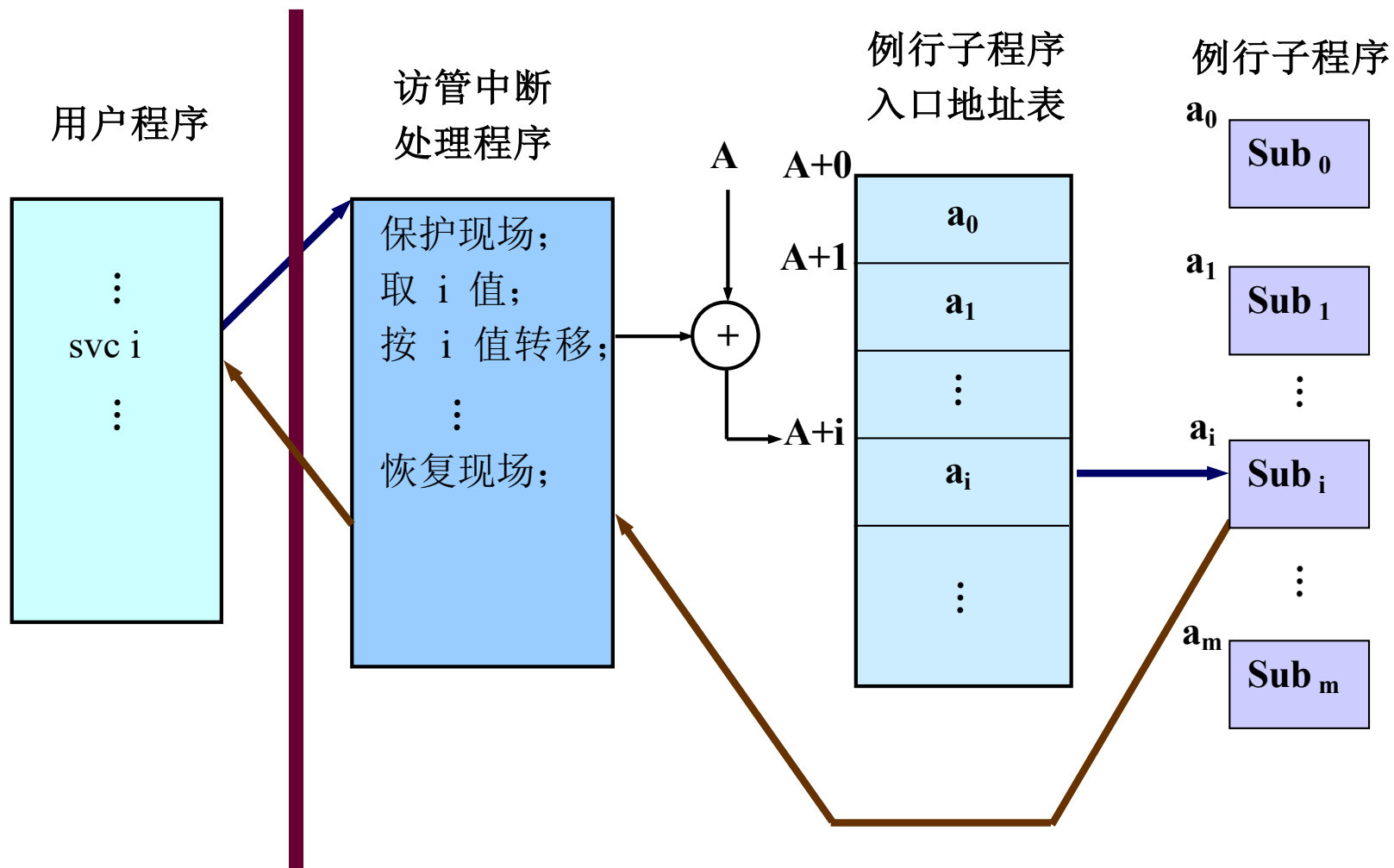
访管中断

- 当处理机执行到访管指令时发生中断，该中断称为访管中断，它表示正在运行的程序对操作系统的某种需求。
- 操作系统提供实现各种功能的例行子程序，其中的每一个功能对应访管指令的一个**功能号**。例如：
 svc 0 显示一个字符
 svc 1 打印一个字符串
- 系统调用是用户在程序一级请求操作系统服务的一种手段，它是带有一定功能号的“访管指令”。其功能是由操作系统中的程序完成的，即由软件方法实现的。





系统调用的实现



系统调用的执行过程





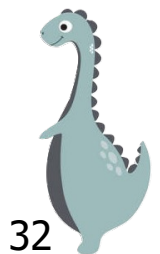
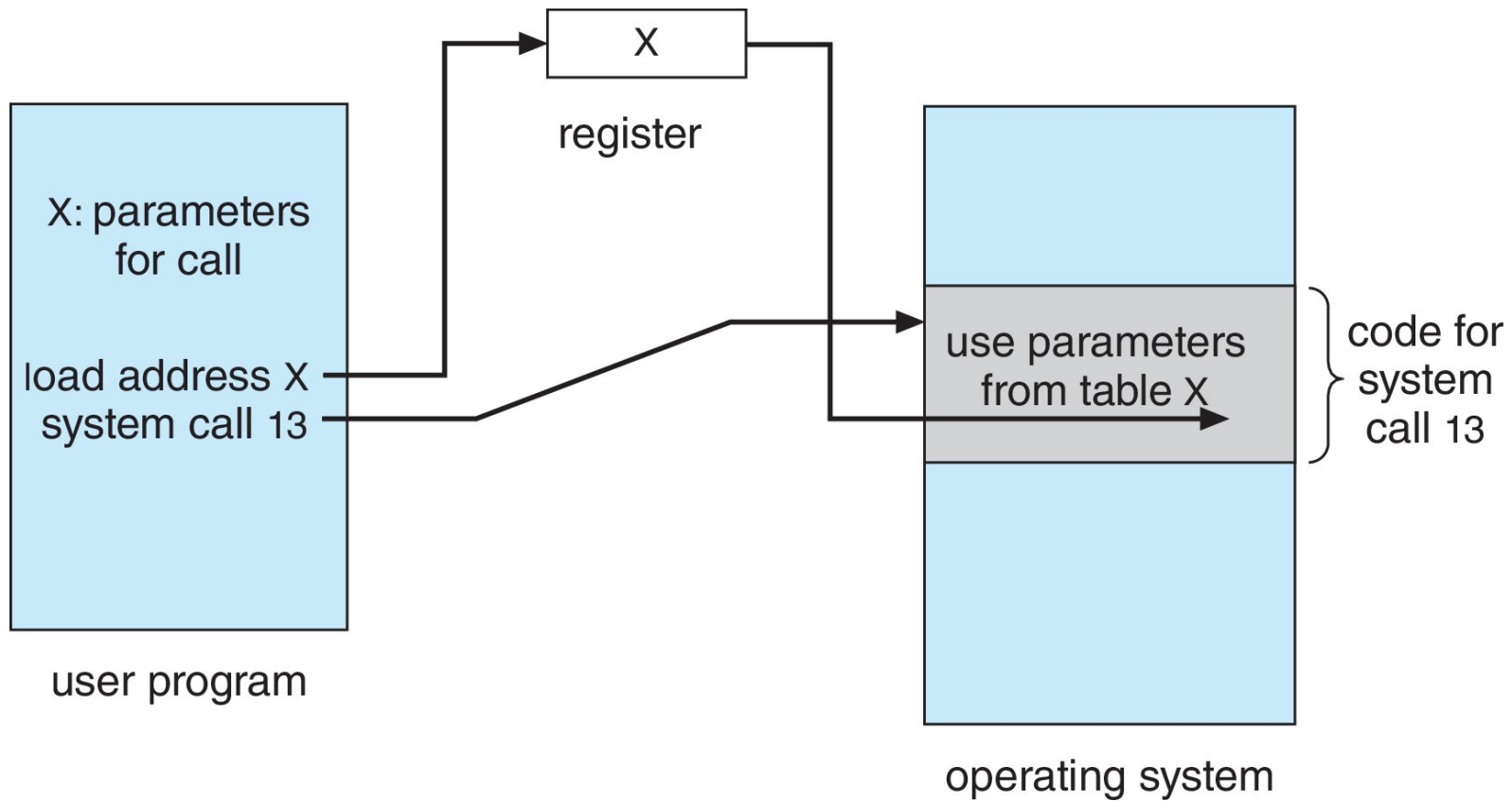
系统调用的参数传递

- 除了所需系统调用的标识符之外，还需要更多的信息
- 具体的信息类型和数量根据操作系统和调用而变化
- 有三种常见的方法用于将参数传递给操作系统
 - 将参数传递到寄存器中
 - 在某些情况下，可能会有多个参数超过寄存器数量
 - 将参数存储在内存中的块或表中，并将块的地址作为参数传递到寄存器中
 - 这种方法被 Linux 和 Solaris 使用
 - 程序将参数放置或推送到堆栈中，并由操作系统从堆栈中弹出
 - 块和堆栈方法不限制传递的参数数量或长度





通过表来进行参数传递





系统调用的类型

- 进程控制
 - create process, terminate process
 - end, abort
 - load, execute
 - get process attributes, set process attributes
 - wait for time
 - wait event, signal event
 - allocate and free memory
 - dump memory if error
 - **Debugger** for determining **bugs, single step** execution
 - **Locks** for managing access to shared data between processes





系统调用的类型

- 文件管理
 - create file, delete file
 - open, close file
 - read, write, reposition
 - get and set file attributes
- 设备管理
 - request device, release device
 - read, write, reposition
 - get device attributes, set device attributes
 - logically attach or detach devices





系统调用的类型

- 信息维护 Information maintenance
 - get time or date, set time or date
 - get system data, set system data
 - get and set process, file, or device attributes
- 通信 Communications
 - create, delete communication connection
 - send, receive messages if **message passing model** to **host name** or **process name**
 - From **client** to **server**
 - **shared-memory model** create and gain access to memory regions
 - transfer status information
 - attach and detach remote devices





系统调用的类型

- 保护

- control access to resources
- get and set permissions
- allow and deny user access





Windows和Unix的系统调用

EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

The following illustrates various equivalent system calls for Windows and UNIX operating systems.

	Windows	Unix
Process control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File management	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device management	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communications	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shm_open() mmap()
Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()



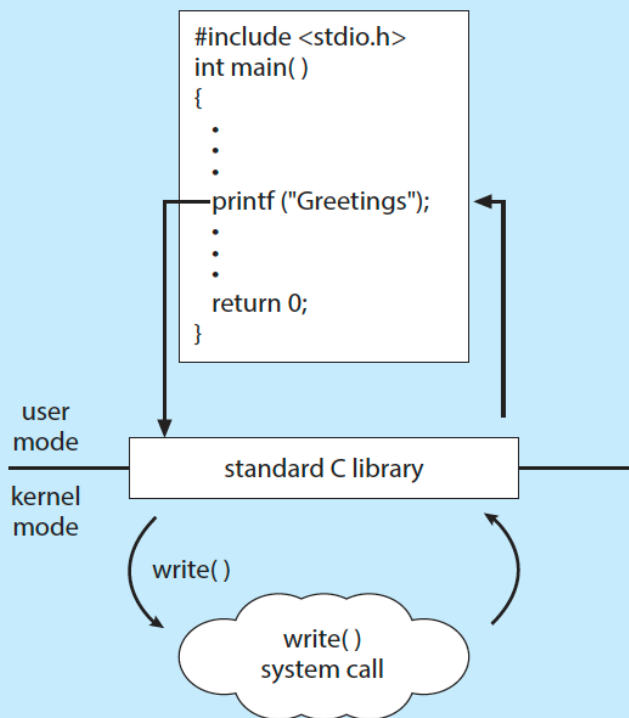


标准C函数库

- C 程序调用 `printf()` 库函数，该函数调用 `write()` 系统调用

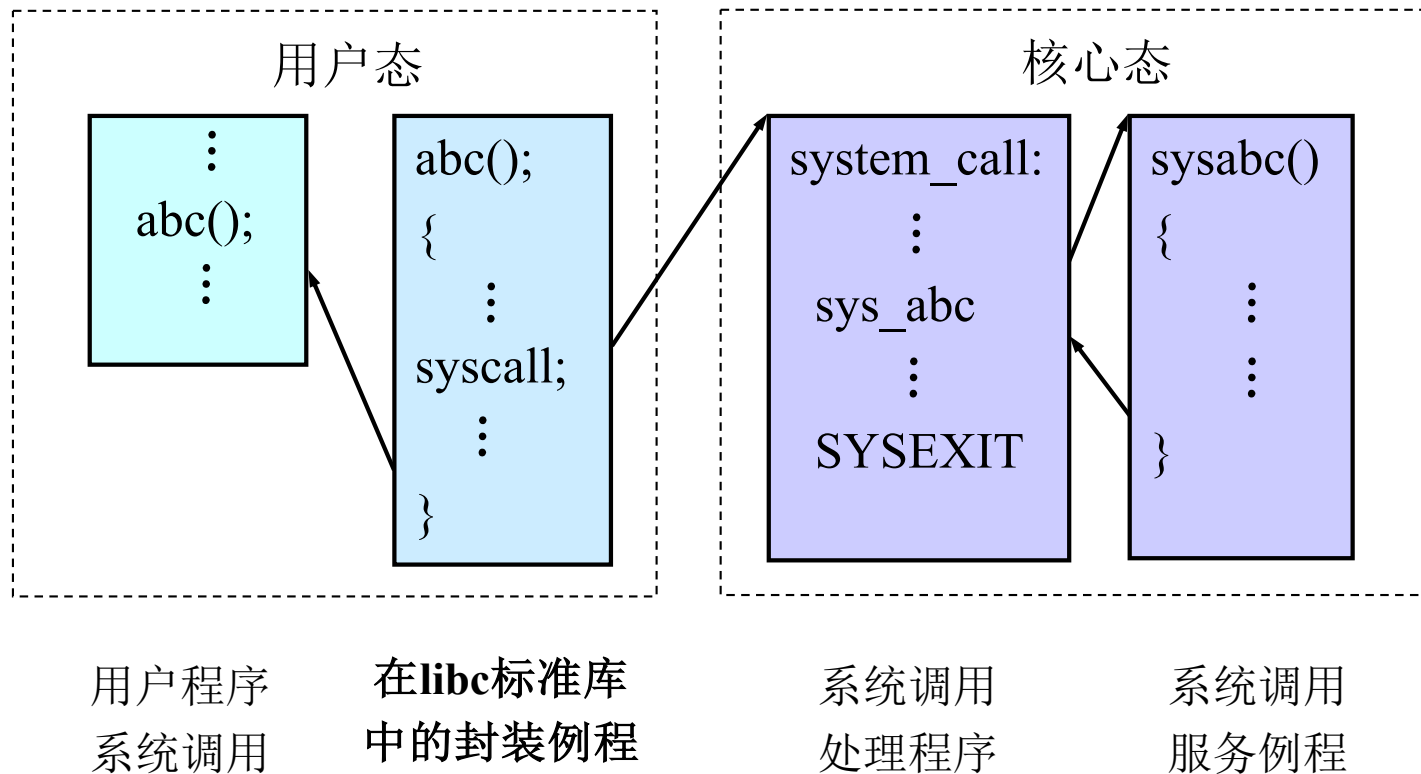
THE STANDARD C LIBRARY

The standard C library provides a portion of the system-call interface for many versions of UNIX and Linux. As an example, let's assume a C program invokes the `printf()` statement. The C library intercepts this call and invokes the necessary system call (or calls) in the operating system—in this instance, the `write()` system call. The C library takes the value returned by `write()` and passes it back to the user program:





Linux的系统调用



Linux系统调用过程





Linux的系统调用实现机制

■ 系统调用的进入

- ① 执行指令触发系统调用
 - X86触发系统调用的指令为int 0x80或syscall
 - Riscv触发系统调用的指令为ecall
- ② OS查看寄存器中的值来决定具体执行哪个系统调用
- ③ 处理机的状态由用户态陷入到内核态
- ④ 执行系统调用处理程序。
- ⑤ 当系统调用处理完毕后，通过特定的指令返回到用户态。





Linux的系统调用实现机制

■ 系统调用号

- ① linux中，每个系统调用被赋予一个唯一的系统调用号
- ② 系统调用号定义在include/asm-i386/unistd.h头文件中
- ③ 系统调用号格式如下

```
#define __NR_restart_syscall      0
#define __NR_exit                  1
#define __NR_fork                  2
#define __NR_read                  3
#define __NR_write                 4
#define __NR_open                  5
.....
.....
#define __NR_mq_getsetattr        282
```





Linux的系统调用实现机制

■ 系统调用表

① 系统调用表记录了内核中所有已注册过的系统调用，

它是系统调用的跳转表。

② 系统调用表是一个函数指针数组，表中依次保存所有

系统调用的函数指针

③ Linux系统调用表保存在arch/i386/kernel/下的entry.S中





Linux的系统调用实现机制

- 系统调用表

- ④ 系统调用表格式如下

ENTRY(sys_call_table)

.long sys_restart_syscall	/* 0 */
.long sys_exit	/* 1 */
.long sys_fork	/* 2 */
.long sys_read	/* 3 */
.long sys_write	/* 4 */
.long sys_open	/* 5 */
.....	
.....	
.long sys_mq_getsetattr	/* 282 */





Linux的系统调用实现机制

- 系统调用处理程序
 - 保存和恢复用户空间的上下文（即寄存器的状态）。
 - 从寄存器中获取系统调用的编号和参数。
 - 根据系统调用的编号在系统调用表中查找对应的处理函数并执行。
 - 将系统调用的结果返回给用户空间。





为什么应用程序是OS特定的

- 在一个系统上编译的应用程序通常无法在其他操作系统上执行
- 每个操作系统都提供自己独特的系统调用
 - 拥有自己的文件格式等
- 应用程序可以是多操作系统的
 - 使用解释型语言编写，如 Python，Ruby，并且解释器可用于多个操作系统
 - 使用包含运行应用程序的虚拟机的语言编写的应用程序（如 Java）
 - 使用标准语言（如 C），在每个操作系统上单独编译以便在每个操作系统上运行
- 应用程序二进制接口（ABI）是 API 的体系结构等价物，定义了不同二进制代码组件如何在给定操作系统、体系结构、CPU 等条件下进行接口。





练习

- 从应用程序的视角来看，异常和中断的区别是什么？
- 在发生CPU的特权级切换时，CPU会自动保存当前的执行状态，包括程序计数器(PC)、栈指针(Stack Pointer, SP)等。请分析：如果不保存PC和SP，会出现什么问题？





练习

- 系统调用和库函数或API之间是什么关系？
- 操作系统提供的系统调用有哪几种参数传递的方法？
- 请解释说明系统调用机制涉及的概念：访管指令、系统调用号、参数传递、系统调用表、系统调用实现函数。





选择题1

- 操作系统提供给编程人员的接口是_____。
 - A.系统调用
 - B.子程序
 - C.库函数
 - D.高级语言

