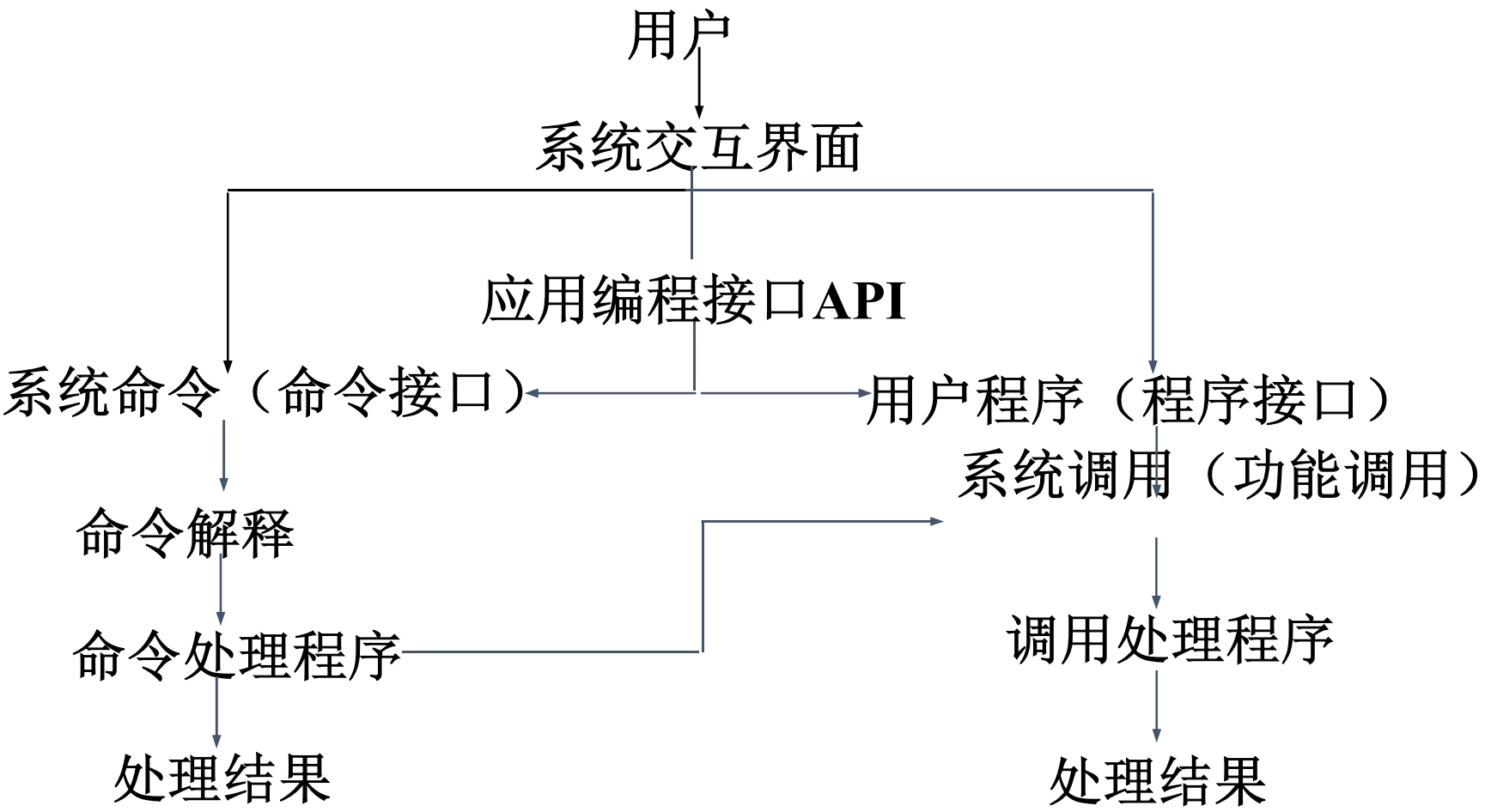
第二章　操作系统用户接口

用户界面是操作系统这部宏篇巨作的封面，用户对操作系统的第一印象从这里开始。

通常操作系统为用户提供两个接口。一个是系统为用户提供的各种命令控制界面接口，用户利用这些操作命令交互地来组织和控制程序的执行或管理计算机系统。另一个接口是程序接口，编程人员在程序中通过程序接口来请求操作系统提供服务。

用户与计算机的交互图



用户与计算机的交互

本章主要内容

2.1 命令控制界面接口

2.2 Linux命令控制界面

2.3 程序接口

2.1 命令控制界面接口

2.1.1 联机命令的类型

为了能向用户提供多方面的服务，通常操作系统都向用户提供了几十条甚至上百条的联机命令。根据这些命令所完成的功能不同，可把它们分成以下几类：

系统访问，目录和文件管理，维护管理，通信等。

批处理方式 ：多条命令放在一个文件中，按照用户的设定自动执行。

2.1.2 联机命令的操作方式

键入式：由键盘终端处理程序 接受终端字符 （缓冲，回显，屏幕编辑，特殊字符），由命令解释程序对命令进行分析，然后执行相应的处理程序。操作者必须记住其命令名、字符串形式及其命令行参数，并须将其在键盘上一一敲入。

选择式：不需要用户输入命令名，系统根据选择点击信号进入命令的解释执行，任务完成后再返回原操作环境。菜单系统会限制用户使用命令的数量，因显示菜单而占用额外的存储空间和时间。

视窗型命令界面：当今操作系统所具有的良好的用户交互界面，是系统可视化的一个基础，所以在操作系统领域被很快推广。命令已被开发成一条条能用鼠标点击而执行的简单的菜单或小巧的图标。而且，用户也可以在提示符的提示下用普通字符方式输入各种命令。

可以预见，计算机系统的命令控制界面将会越来越方便和越来越人性化。

2.2 Linux命令控制界面

2.2.1 登录Shell

login:
password:
$

2.2.2 命令句法

command [option] [arguments] <CR>

2.2.3 常用的基本命令

pwd 、ls –Fal、cp、mv、rm、mkdir、cd、man、cat和more及帮助命令man 。

在 Linux系统中 一个文件上有可读(r)、可写(w)、可执行(x)三种 执行模式，分别针对该文件的拥有者(owner)、同组者(group member)以及其他人(other)。

例：$chmod go—w temp

2.2.4 重定向与管道命令

```
$cat filel>file2 ;
$cat filel file4>>file2;
$ out<filel>file0;
$ cat file|WC
```

2.2.5 通信命令

信箱通信命令mail

对话通信命令write

2.2.6 后台命令

命令后面再加上“&”号，以告诉Shell将该命令放在后台执行，以便用户在前台继续键入其它命令。

2.3.2系统调用的类型

设备、文件管理，进程控制，进程通信 ,存储管理

目的：是使得用户可以使用操作系统提供的有关方面的功能，而不必了解程序内部结构和有关硬件细节，从而起到减轻用户负担和保护系统以及提高资源利用率的作用。

2.3.3系统调用的实现

陷入向量，系统调用号，参数设置方式(直接和间接)。

2．3.1系统调用

应用程序又必须取得操作系统所提供的服务，为此，在操作系统中提供了系统调用，使应用程序可以间接调用操作系统的相关过程，取得相应的服务。

当应用程序中需要操作系统提供服务时，如请求I/O资源或执行I/O操作，应用程序必须使用系统调用命令。

由操作系统捕获到该命令后，便将CPU的状态从用户态转换到系统态，然后执行操作系统中相应的子程序(例程)，完成所需的功能。执行完成后，系统又将CPU状态从系统态转换到用户态，再继续执行应用程序。

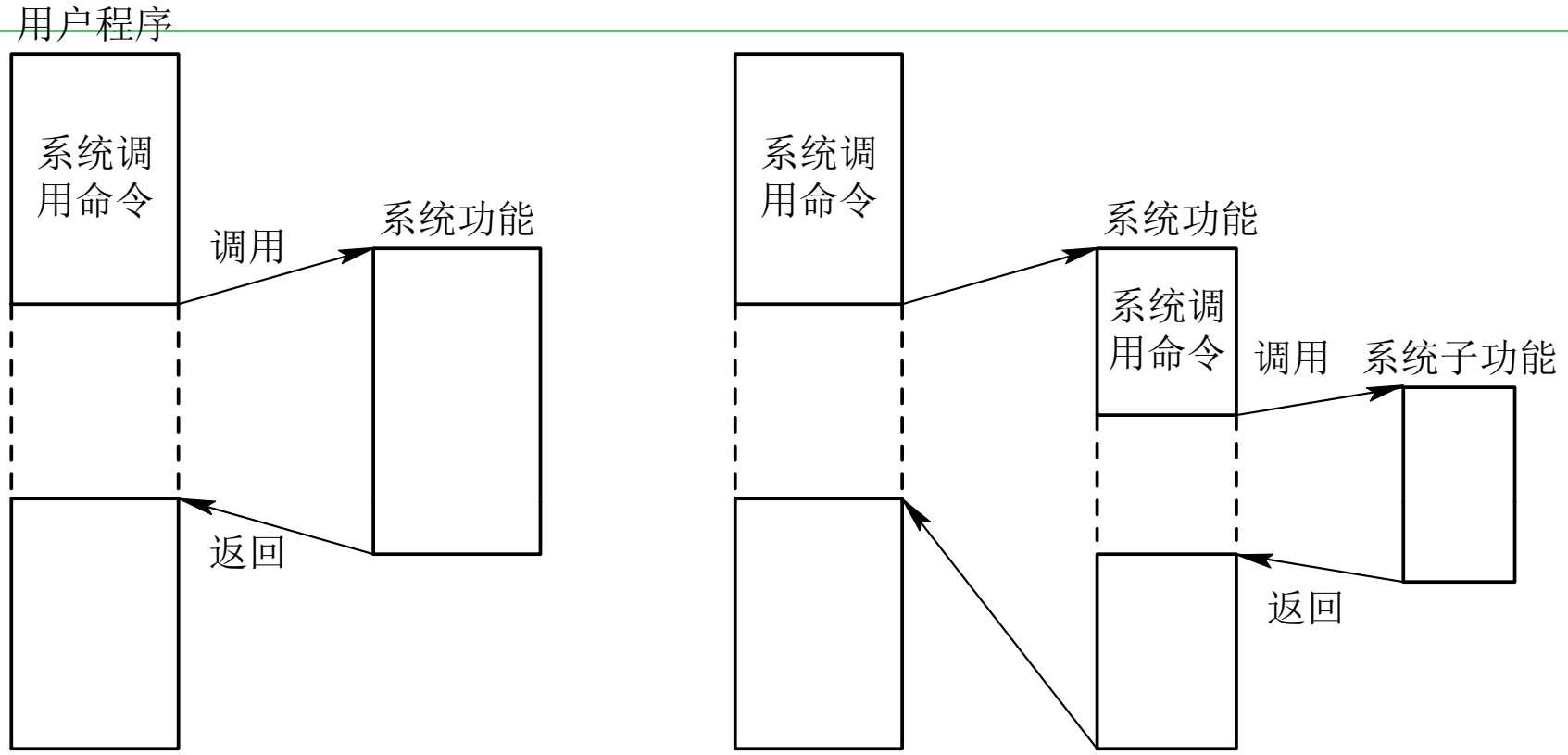
可见，系统调用在本质上是应用程序请求OS内核完成某功能时的一种过程调用，但它是一种特殊的过程调用，它与一般的过程调用有下述几方面的明显差别：

(1) 运行在不同的系统状态。一般的过程调用，其调用程序和被调用程序都运行在相同的状态——系统态或用户态；而系统调用与一般调用的最大区别就在于：调用程序是运行在用户态，而被调用程序是运行在系统态。

(2) 状态的转换通过软中断进入。运行系统调用时，由于调用和被调用过程是工作在不同的系统状态，因而不允许由调用过程直接转向被调用过程。通常都是通过软中断机制，先由用户态转换为系统态，经核心分析后，才能转向相应的系统调用处理子程序。

(3) 返回问题。在采用了抢占式(剥夺)调度方式的系统中，在被调用过程执行完后，要对系统中所有要求运行的进程做优先权分析。当调用进程仍具有最高优先级时，才返回到调用进程继续执行；否则，将引起重新调度，以便让优先权最高的进程优先执行。此时，将把调用进程放入就绪队列。

(4) 嵌套调用。像一般过程一样，系统调用也可以嵌套进行，即在一个被调用过程的执行期间，还可以利用系统调用命令去调用另一个系统调用。



(a) 系统调用与返回　　(b) 程序之间的嵌套调用

系统功能的调用

3．中断机制

系统调用是通过中断机制实现的，一个操作系统的所有系统调用都通过同一个中断入口来实现。如MS-DOS提供了INT 21H，应用程序通过该中断获取操作系统的服务。

对于拥有保护机制的操作系统来说，中断机制本身也是受保护的，在IBM PC上，Intel提供了多达255 个中断号，但只有授权给应用程序保护等级的中断号，才是可以被应用程序调用的。

如果应用程序进行调用未被授权的中断号，同样会引起保护异常，而导致自己被操作系统停止。

如Linux 仅仅给应用程序授权了4 个中断号：3、4、5以及80h，前三个中断号是提供给应用程序调试所使用的，而80h正是系统调用(system call)的中断号。

2.3.2 系统调用的类型

1．进程控制类系统调用

这类系统调用主要用于对进程的控制，如创建一个新的进程和终止一个进程的运行，获得和设置进程属性等。

1）创建和终止进程的系统调用

在多道程序环境下，为使多道程序能并发执行，必须先利用创建进程的系统调用来为欲参加并发执行的各程序分别创建一个进程。当进程已经执行结束时、 或因发生异常情况而不能继续执行时，可利用终止进程的系统调用来结束该进程的运行。

2）获得和设置进程属性的系统调用

这些属性包括: 进程标识符、进程优先级、最大允许执行时间等。此时，我们可利用获得进程属性的系统调用，来了解某进程的属性，利用设置进程属性的系统调用，来确定和重新设置进程的属性。

3）等待某事件出现的系统调用

进程在运行过程中，有时需要等待某事件(条件)出现后方可继续执行。

例如，一进程在创建了一个(些)新进程后，需要等待它(们)运行结束后，才能继续执行，此时可利用等待子进程结束的系统调用进行等待；

又如，在客户/服务器模式中，若无任何客户向服务器发出消息，则服务器接收进程便无事可做，此时该进程就可利用等待(事件)的系统调用，使自己处于等待状态，一旦有客户发来消息时，接收进程便被唤醒，进行消息接收的处理。

2．文件操纵类系统调用

对文件进行操纵的系统调用数量较多，有创建文件、删除文件、打开文件、关闭文件、读文件、写文件、建立目录、移动文件的读/写指针、改变文件的属性等。

1）创建和删除文件

当用户需要在系统中存放程序或数据时，可利用创建文件的系统调用creat，由系统创建一个新文件；当用户已不再需要某文件时，可利用删除文件的系统调用unlink将指名文件删除。

2）打开和关闭文件

用户在第一次访问某个文件之前，应先利用打开文件的系统调用open，将指名文件打开，当用户不再访问某文件时，又可利用关闭文件的系统调用close，将此文件关闭。

3）读和写文件

用户可利用读系统调用read，从已打开的文件中读出给定数目的字符，并送至指定的缓冲区中；

同样，用户也可利用写系统调用write，从指定的缓冲区中将给定数目的字符写入指定文件中。read和write两个系统调用是文件操纵类系统调用中使用最频繁的。

3．进程通信类系统调用

在OS中经常采用两种进程通信方式，即消息传递方式和共享存储区方式。

当系统中采用消息传递方式时，在通信前，应由源进程发出一条打开连接的系统调用open connection，而目标进程则应利用接受连接的系统调用accept connection表示同意进行通信；然后，在源和目标进程之间便可开始通信。

可以利用发送消息的系统调用send message或者用接收消息的系统调用receive message来交换信息。通信结束后，还须再利用关闭连接的系统调用close connection结束通信。

用户在利用共享存储区进行通信之前，须先利用建立共享存储区的系统调用来建立一个共享存储区，再利用建立连接的系统调用将该共享存储区连接到进程自身的虚地址空间上，然后便可利用读和写共享存储区的系统调用实现相互通信。

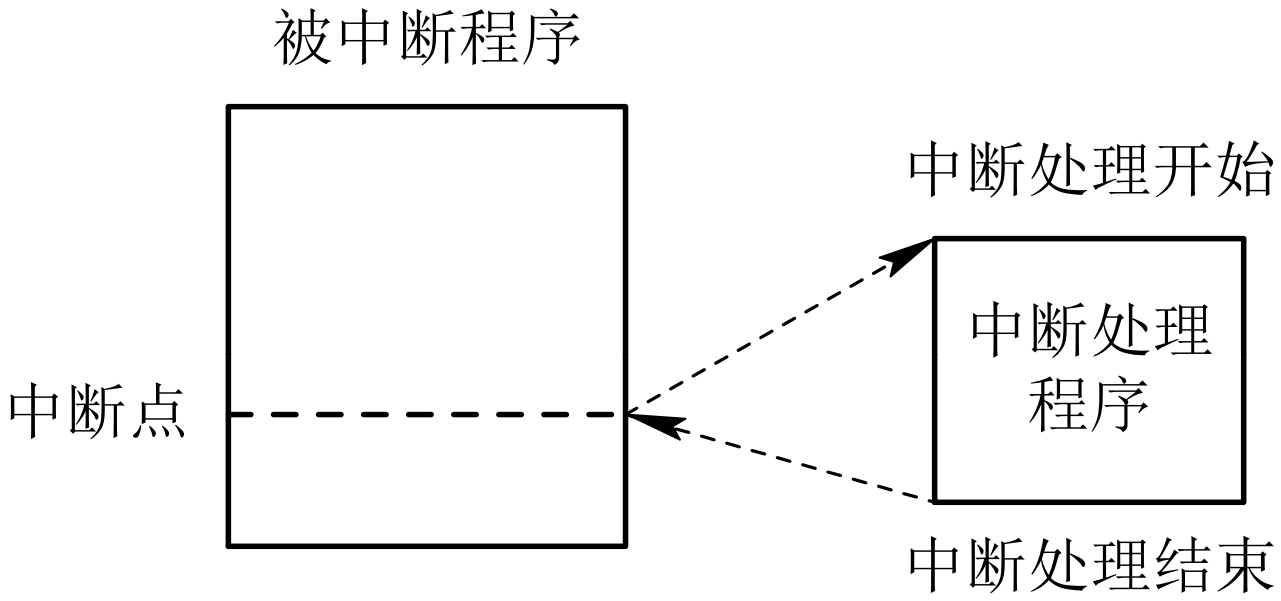
除上述的三类外，常用的系统调用还包括设备管理类系统调用和信息维护类系统调用。

2.3.3　系统调用的实现

1．中断和陷入硬件机构

1）中断和陷入的概念

中断是指CPU对系统发生某事件时的这样一种响应：CPU暂停正在执行的程序，在保留现场后自动地转去执行该事件的中断处理程序；执行完后，再返回到原程序的断点处继续执行。



中断时的CPU轨迹

中断分为外中断和内中断。

外中断，是指由于外部设备事件所引起的中断，如通常的磁盘中断、打印机中断等；而内中断则是指由于CPU内部事件所引起的中断，如程序出错(非法指令、地址越界)、电源故障等。

内中断(trap)也被译为“捕获”或“陷入”。通常，陷入是由于执行了现行指令所引起的；而中断则是由于系统中某事件引起的，该事件与现行指令无关。由于系统调用引起的中断属于内中断，因此把由于系统调用引起中断的指令称为陷入指令。

2．系统调用号和参数的设置

往往在一个系统中设置了许多条系统调用，并赋予每条系统调用一个唯一的系统调用号。

在系统调用命令(陷入指令)中，怎样把相应的系统调用号传递给中断和陷入机制？

直接把系统调用号放在系统调用命令(陷入指令)中；如IBM 370和早期的UNIX系统，是把系统调用命令的低8位用于存放系统调用号；

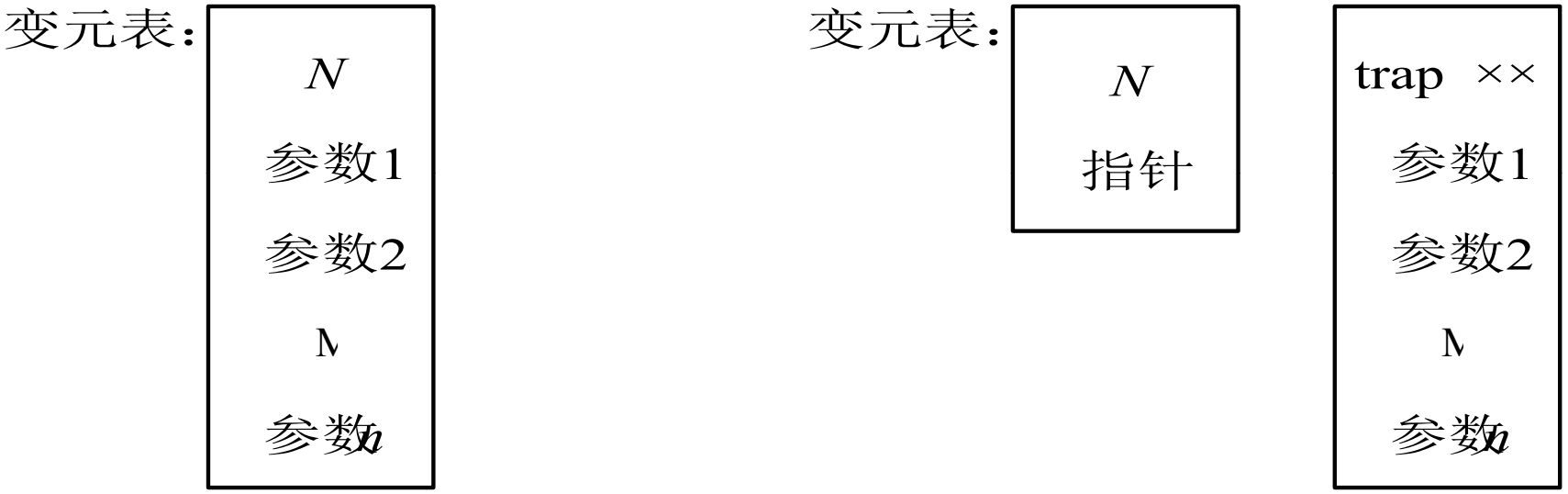
将系统调用号装入某指定寄存器或内存单元中，如MS-DOS是将系统调用号放在AH寄存器中，Linux则是利用EAX寄存器来存放应用程序传递的系统调用号。

每一条系统调用都含有若干个参数，在执行系统调用时，如何设置系统调用所需的参数，即如何将这些参数传递给陷入处理机构和系统内部的子程序(过程)，常用的实现方式有以下几种：

(1) 陷入指令自带方式。陷入指令除了携带一个系统调用号外，还要自带几个参数进入系统内部，由于一条陷入指令的长度是有限的，因此自带的只能是少量的、有限的参数。

(2) 直接将参数送入相应的寄存器中。MS-DOS便是采用的这种方式，即用MOV指令将各个参数送入相应的寄存器中。系统程序和应用程序显然应是都可以访问这种寄存器的。这种方式的主要问题是由于这种寄存器数量有限，因而限制了所设置参数的数目。

(3) 参数表方式。将系统调用所需的参数放入一张参数表中，再将指向该参数表的指针放在某个指定的寄存器中。当前大多数的OS中，如UNIX系统和Linux系统，便是采用了这种方式。



(a) 直接方式　　(b) 间接方式

系统调用的参数形式

3．系统调用的处理步骤

在设置了系统调用号和参数后，便可执行一条系统调用命令。不同的系统可采用不同的执行方式。在UNIX系统中，是执行CHMK(change mode to kernel)命令；而在MS-DOS中则是执行INT 21软中断。

系统调用的处理过程可分成以下三步：

一，将处理机状态由用户态转为系统态；

二，由硬件和内核程序进行系统调用的一般性处理，即首先保护被中断进程的CPU环境，将处理机状态字PSW、程序计数器PC、系统调用号、用户栈指针以及通用寄存器内容等，压入堆栈；然后，将用户定义的参数传送到指定的地址保存起来。

三，是分析系统调用类型，转入相应的系统调用处理子程序。

为使不同的系统调用能方便地转向相应的系统调用处理子程序，在系统中配置了一张系统调用入口地址表。

表中的每个表目都对应一条系统调用，其中包含该系统调用自带参数的数目、系统调用处理子程序的入口地址等。因此，核心可利用系统调用号去查找该表，即可找到相应处理子程序的入口地址而转去执行它。

最后，在系统调用处理子程序执行完后，应恢复被中断的或设置新进程的CPU现场，然后返回被中断进程或新进程，继续往下执行。

系统调用处理过程

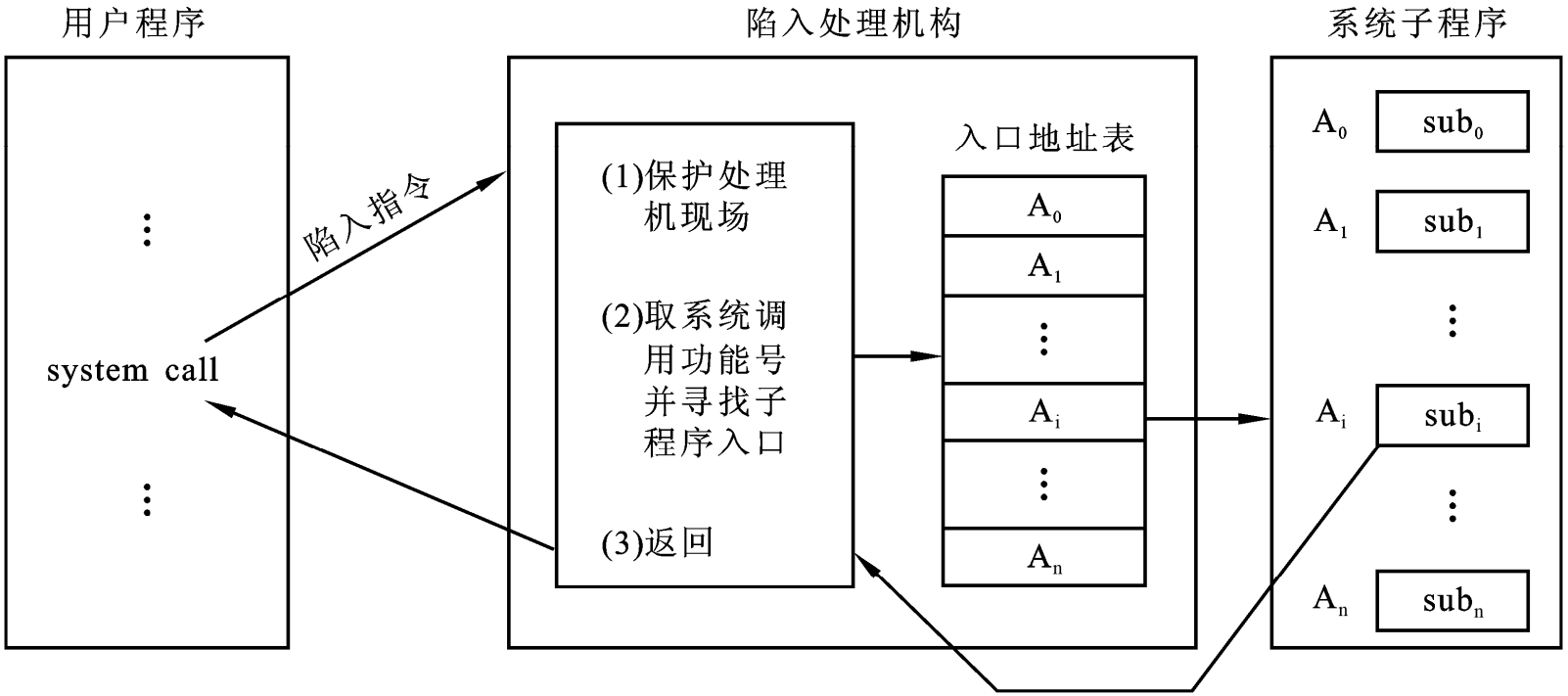


图2-2 系统调用处理过程

Linux通过软中断指令int 0x80 来陷入内核态（在Intel Pentium II 又引入了sysenter指令）通过寄存器传递参数。中断0x80 把控制权传给核心入口地址中的_system_call()，_system_call()将所有的寄存器内容压入用户栈，并检查系统调用是否合法，如果合法，确定系统调用号，从_sys_call_table中找出的相应的系统调用入口地址，执行相应的系统调用功能。

2.3.4 Linux系统调用

系统调用接口默认链接到所有的用户程序的应用编程接口—lib标准C库函数（每个系统调用封装了一个函数）。

1.设备调用：设备读写控制等。
2.文件调用：read、write、open、close。
3. 进程控制：进程创建fork、等待wait、退出exit、杀死kill、进程同步控制lockf 、signal、pause、通信pipe等 。
4. 进程通信的系统调用：进程通信用的系统调用主要包括套接字(socket)的建立、链接、控制和删除，以及进程 间通信用的消息队列、共用存储区以及有关同步机制的建立、链接、控制和删除等有关系统调用。
5 .存储管理的系统调用：系统调用包括获取内存现有空间大小、检查内存中现有进程以及对内存区的保护和改变堆栈大小等功能
6 .管理用系统调用：例如，设置和读取日期和时间，取用户和主机等的标识符等系统调用。

例2-1:

目录系统调用的例子，实现了Linux命令pwd的功能

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define MAX_PATH 255
int main()
{
   char name[MAX_PATH+1];
   if(getcwd(name,MAX_PATH)==NUll)
    printf("error:Failure getting pathname");
    printf("Current Directory:%S\n",name);
}
```

程序的运行结果：Current Directory:/home/student

由上所述可见，一个用户程序将频繁地利用各种系统调用以取得操作系统所提供的多种服务。

2.3.5 Windows应用编程接口

为了进一步促进操作系统和应用软件之间互通，方便用户编写 Windows 应用程序，Windows 提供了大量的子程序和函数，也就是 Windows应用编程接口（Application Programming Interface—API）。它是一种公共的标准的应用接口，使不同的软件系统可以互相利用，共享信息。

标准Windows API函数可以分为以下几类：

系统服务:资源管理

通用控件库：用户界面

图形设备接口：生成图形

网络服务：网络连接，通信

用户接口：用户交互

系统Shell：增强系统Shell

Windows系统信息

例2-2：

应用Windows API函数实现了例2-1的功能。

```
include "stdafx.h"
include "windows.h"
define DIRNAME_LEN MAX_PATH+2
int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR     lpCmdLine,
                   int       nCmdShow)
{
   TCHAR PwdBuffer[DIRNAME_LEN];
            DWORD LenCuDir;
   LenCuDir=GetCurrentDirectory(DIRNAME_LEN,PwdBuffer);
           if(LenCuDir==0)
MessageBox(NULL,"Failure getting pathname,",NULL,MB_OK);
           MessageBox(NULL,PwdBuffer,"Current Directory",MB_OK);
   return 0;
}
```