第5章　操作系统

操作系统(operating system,OS)是建立在裸机上的第一层软件系统,属于计算机的系统软件。没有操作系统,整个计算机将无法正常工作。本章先说明操作系统的基本功能,然后重点介绍几类物联网前端节点常用的微操作系统。

5.1　操作系统概述

本节对操作系统的分类和功能作简单介绍,使读者可以对操作系统在计算机系统中的地位有一个初步了解。

5.1.1　操作系统的功能及分类

计算机系统包括硬件和软件两大组成部分,硬件是所有软件运行的物质基础,软件能充分发挥硬件潜能和扩充硬件功能,完成各种系统及应用任务,两者相互促进、相辅相成、缺一不可。图5-1给出了一个计算机系统的软硬件层次结构,由图可见,操作系统是最接近硬件的底层软件,是对计算机硬件的功能的扩展,主要具有资源管理、信息的存储和保护、并发任务的调度等功能,并向上层软件运行提供有力的支撑环境。通过引入操作系统,计算机用户不再直接面对底层硬件,不需要处理硬件的物理细节,甚至不需要了解硬件的底层工作原理,把用户从烦琐的硬件控制中解放出来。总之,操作系统向用户提供了一个逻辑的计算机,屏蔽了计算机底层的工作细节,通过应用编程接口(API)向用户提供通用服务,使得用户在更友好的操作界面上工作,提高了计算机用户的工作效率。

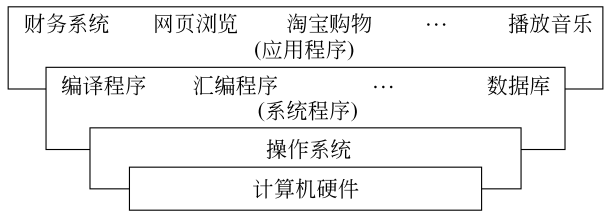


图5-1　计算机系统的层次结构

API

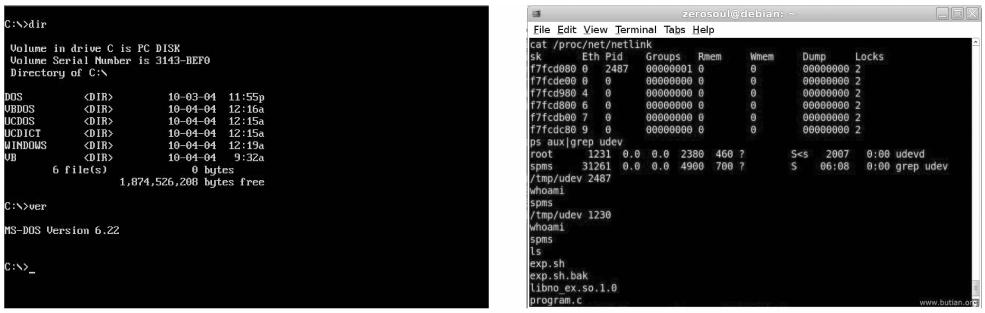
API(application programming interface,应用程序编程接口)是一些预先定义的函数,目的是提供应用程序与开发人员基于某软件或硬件的访问一组例程的能力,而又无须访问源码,或理解内部工作机制的细节。API除了有应用"应用程序接口"的意思外,还特指API的说明文档,也称为帮助文档。

按照向用户提供什么样的操作界面,操作系统可以分为图形界面操作系统和字符界面操作系统。图 5-2 给出了目前常用的图形操作系统。图 5-2(a)是微软公司目前最新的 Windows 10 系统,向用户提供更加人性化的桌面,方便访问常用程序及操作,实现了对无线互联网的优化支持和对触摸屏的支持。图 5-2(b)是 Linux 下的桌面操作系统,同样可以以图形界面操作计算机,方便易用。

(a) Windows 10桌面操作系统　　(b) Linux下的KDE桌面操作系统

图 5-2　两种图形界面操作系统

图 5-3 给出了两种字符界面操作系统:DOS 操作系统和 Linux 的 shell 环境,它们提供了以字符串命令操作计算机的方式。目前,图形和字符界面两种操作系统共存于操作系统市场,在家庭、办公等环境多数用图形界面的操作系统,而在学术界往往更趋向于应用字符界面操作系统。



(a) DOS操作系统　　(b) Linux shell界面

图 5-3　两种字符界面操作系统

在操作系统发展过程中,不同历史阶段出现了不同特点的操作系统,主要包括简单批处理、多道程序批处理、分时操作系统、个人计算机操作系统、网络操作系统、分布式操作系统和实时操作系统。它们的出现有其特定的历史背景,比如以大型主机为中心的计算机时代,出现了分时操作系统,在该操作系统的控制下,多个用户分时共享计算机,提高了主机的利用效率,但是用户感觉上好像是在独占这台计算机。再比如,在嵌入式系统发展迅猛的当今,相应地出现了实时操作系统,该操作系统适应嵌入式系统的实时性需求,能够提供系统任务的实时管理,保障系统正确可靠地运行。

5.1.2　操作系统的任务调度

1. 任务的概念

在操作系统中，任务(task)是一个具有独立功能的、无限循环程序段的一次运行活动，也称为进程(process)，是操作系统内核调度的单位，具有动态性、并行性和异步独立性等特点。动态性指任务的状态不断变化，比如，任务可能处于就绪态、运行态和等待态。在多任务系统中，任务的状态将随系统的执行而不断变化。并行性指系统中同时存在多个任务，这些任务在宏观上是同时运行的。异步独立性指每个任务各自按相互独立的、不可预知的速度运行，如果需要的话，任务之间将要交换信息，即任务间进行通信。

图 5-4 为 Windows XP 操作系统的任务管理器，显示了系统中当前有哪些任务在运行。

图 5-4　Windows XP 操作系统的任务管理器

一个任务主要包含四部分内容：代码，即一段可执行的程序；数据，即程序需要处理的相关变量和缓冲区等；堆栈，即用来临时存放程序执行过程中中间信息的存储区域；程序执行的上下文环境(context)，包括了任务优先级、任务的状态等任务属性以及 CPU 的各种寄存器内容。

如上所述，操作系统中的任务具有多种属性，包括任务的优先级(priority)、周期(period)、计算时间(computation)和截止时间(deadline)等。任务的优先级表示任务对应工作内容在处理上的优先程度，优先级越高，表明任务越需要得到优先处理。优先级可分为静态优先级和动态优先级。静态优先级表示任务的优先级被确定后，在系统运行过程中将不再发生变化；动态优先级则意味着在系统运行过程中，任务的优先级是可以被动态改变的。周期表示任务周期性执行的时间间隔。任务的计算时间指任务在特定硬件环境下被完整执行所需要的时间，也称为任务的执行时间(execution time)。由于任务每次执行的软件环境

的差异性,导致任务在每次具体执行过程中计算时间各有不同,因此通常用最坏情况下的执行时间(worst-case time)来表示,也可以用统计时间(statistic time)来表示。任务的截止时间表示任务需要在该时间到来之前被执行完成。

2. 任务管理

在计算机系统中为便于进行任务管理,按照任务运行过程中所处的不同阶段,将任务划分为多个不同的状态。任务拥有的资源情况不断变化,导致任务状态不断变化。一个简单的系统,任务会包含以下 3 种状态。

(1) 等待(waiting)状态:任务在等待某个事件的发生。

(2) 就绪(ready)状态:任务在等待获得处理器使用权。

(3) 运行(running)状态:任务已占用处理器,所包含的代码正在被执行。

在单处理器系统中,任何时候都只有一个任务在 CPU 中执行,如果没有任何事情可做,CPU 就运行空闲任务执行空操作。任何一个可以执行的任务都必须处于就绪状态,内核的调度器从就绪任务队列中选择一个满足条件的任务执行,从而任务进入运行态,处于运行状态的任务如果被高优先级任务所抢占,任务又会回到就绪状态。除了运行和就绪状态外,任务还可以处于等待状态,例如,任务在需要等待 I/O 设备或者其他任务的数据时,就处于等待状态。处于等待状态的任务如果需要的资源得到满足,就会转换为就绪状态,等待调度执行。图 5-5 给出了任务的状态迁移过程。

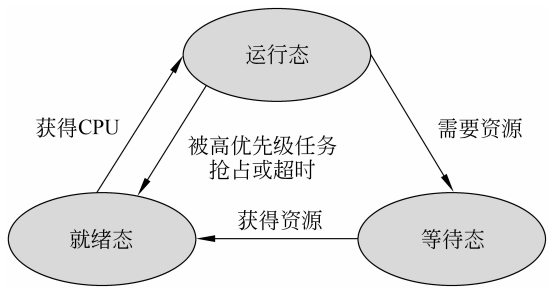


图 5-5 任务的状态迁移

I/O

I/O 是 input/output 的缩写,即输入输出。每个设备都会有一个专用的 I/O 地址,用来处理自己的输入输出信息。CPU 与外部设备、存储器的连接和数据交换都需要通过接口设备来实现,前者被称为 I/O 接口,而后者则被称为存储器接口。存储器通常在 CPU 的同步控制下工作,接口电路比较简单;而 I/O 设备品种繁多,其相应的接口电路也各不相同,因此,习惯上说到接口只是指 I/O 接口。

在操作系统内部,任务管理是通过任务控制块(task control block,TCB)这个数据结构实现的。任务控制块包含了任务执行过程中所需要的所有信息,不同内核的任务控制块包含的信息通常都不太一样,但大都包含任务名字、任务执行的起始地址、任务的优先级、任务的上下文(寄存器、堆栈指针和指令指针等)和任务的队列指针等内容。操作系统使用任务控制块链表来跟踪、管理系统内部所有的任务,涉及许多相关的数据结构。

3. 任务调度

操作系统通过调度程序(scheduler)来实现任务调度功能,调度程序也称为调度器。调度程序以函数的形式实现操作系统具体的调度算法,本身并不是一个任务,而是一个函数调用,可在内核的各个部分进行调用。调用调度程序的具体位置又称为调度点(scheduling point),调度点通常在中断服务程序的结束位置、任务因等待资源而处于等待状态时和任务处于就绪状态时等地方。

可以把一个调度算法描述为在一个特定时刻用来确定将要运行任务的一组规则。1973 年,Liu 和 Layland 首先开始了关于实时调度算法的研究工作,随后,相继出现了很多调度算法,这其中包括任务是否允许打断、任务优先级何时确定等不同调度算法。

1) 任务是否允许打断

根据任务在运行过程中能否被打断,调度算法分为非抢占式调度和抢占式调度两类。在非抢占式调度算法中,一旦任务开始运行,该任务只有在运行完成而主动放弃 CPU 使用权,或是因为等待其他资源被阻塞,才会停止运行。而在抢占式调度算法中,正在运行的任务可能被其他紧急任务打断,紧急任务占先运行。在现代的操作系统中,实时内核大都采用了抢占式调度算法,比如,μC/OS-Ⅱ微实时操作系统,使得关键任务能够打断非关键任务的执行,确保关键任务在截止时间之前运行结束。相对来说,抢占式调度算法更复杂,且需要更多的资源,并可能在使用不当的情况下造成低优先级任务出现长时间得不到执行的情况。非抢占式调度算法常用于那些任务需要按照预先确定的顺序执行,且只有当前任务主动放弃 CPU 资源后,其他任务才能得到执行的场合,例如,传感器微操作系统 TinyOS 就是非抢占式操作系统。

2) 任务优先级何时确定

根据任务优先级的确定时机,调度算法分为静态调度和动态调度两类。在静态调度算法中,所有任务的优先级在设计时就确定下来,且在运行过程中不会发生变化(如单调速率调度算法 RMS)。在动态调度算法中,任务的优先级在运行过程中确定,且在运行过程中不断发生变化(如截止时间优先调度算法 EDF)。静态调度算法比较简单,但缺乏灵活性,不利于系统扩展。动态调度有足够的灵活性来处理系统的变化,但需要更多的额外开销。

4. 几个常见调度算法

1) 基于优先级的抢占调度

在基于优先级可抢占的任务调度中,如果出现具有更高优先级的任务处于就绪状态,则当前任务将停止运行,把 CPU 的控制权交给更高优先级的任务,使更高优先级的任务得到执行。因此,实时内核要确保 CPU 的使用权总是被具有最高优先级的就绪任务所拥有。当一个具有比当前运行任务的优先级更高的任务进入就绪状态时,实时内核应及时进行任务切换,即保存当前正在运行任务的上下文,恢复具有更高优先级任务的上下文。

图 5-6 为基于优先级的可抢占调度方式下的多任务运行情况。任务 1 被具有更高优先级的任务 2 抢占,然后任务 2 又被具有更高优先级的任务 3 抢占。当任务 3 运行完成之后,任务 2 继续执行。当任务 2 运行完成之后,任务 1 才又得以继续执行。

2) 时间片轮转调度

时间片轮转调度(round-robin scheduling)算法是指当有两个或多个就绪任务具有相同

组优先级,且它们是优先级别最高的就绪组时,调度器按照该组中任务就绪的先后次序调度第一个任务,让第一个任务运行一段时间;然后又调度第二个任务,让第二个任务又运行一段时间;依此类推,到该组最后一个任务也得以运行一段时间后,接下来又让第一个任务继续运行。这里,任务运行的这段时间称为时间片(time-slice)。

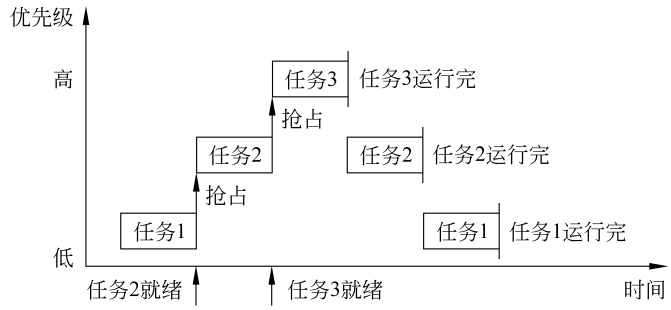


图 5-6 可抢占方式下的多任务调度运行

采用时间片轮转调度算法时,任务的时间片大小要适当选择,时间片大小的选择会影响系统的性能和效率。时间片太大,时间片轮转调度就没有意义;时间片太小,任务切换过于频繁,处理器开销大,真正用于运行应用程序的时间会减少。另外,不同的实时内核实现时间片轮转调度算法可能有一些差异,例如,有的内核允许同优先级组的各个任务具有不一致的时间片,而有的内核要求相同优先级组的任务具有一致的时间片。

图 5-7 为时间片轮转调度方式下的多任务运行情况。任务 1 和任务 2 具有相同的优先级,按照时间片轮转的方式轮流执行。当更高优先级的任务 3 就绪后,正在执行的任务 2 被抢占,高优先级任务 3 得到执行。当任务 3 运行完成后,任务 2 重新在未完成的时间片内继续执行。随后任务 1 和任务 2 又按照时间片轮转的方式执行。

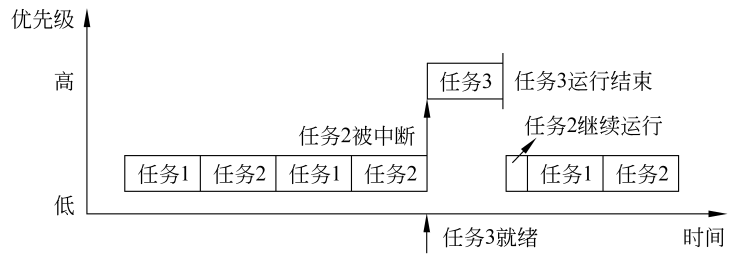


图 5-7 时间片轮转方式下的多任务调度运行

3) 单调速率调度算法

1973 年,Liu 等人在 ACM 上发表了题为 *Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment* 的论文,提出了单调速率调度算法(rate-monotonic scheduling algorithm,RMS),该论文奠定了实时系统中现代调度算法的理论基础。

RMS 的工作基础包括以下一些假设:

(1) 所有任务都是周期任务。

(2) 任务在每个周期内的计算时间都相等,保持为一个常量。

(3) 任务的相对截止时间等于任务的周期,相对截止时间为任务的绝对截止时间减去任务的就绪时间。

(4) 任务之间不进行通信,也不需要同步。

(5) 任务可以在计算的任何位置被抢占,不存在临界区。

RMS 是一个静态的固定优先级调度算法,任务的优先级与任务的周期表现为单调函数关系,任务周期越短,任务的优先级越高;任务周期越长,任务的优先级越低。RMS 是静态调度中的最优调度算法,即如果一组任务能够被任何静态调度算法所调度,则这些任务在 RMS 下也是可调度的。

任务的可调度性可以通过计算任务的 CPU 使用率,然后把得到的 CPU 使用率同一个可调度的 CPU 使用率上限进行比较来获得。这个可调度的 CPU 使用率上限成为可调度上限(schedulable bound),可调度上限表示给定任务在特定调度算法下能够满足截止时间要求的最坏情况下的最大 CPU 使用率。可调度上限的最大值为 100%,与调度算法密切相关。对于一组任务,如果任务的 CPU 使用率小于或等于可调度上限,则这组任务是可被调度的;如果任务的 CPU 使用率大于可调度上限,就不能保证这组任务是可被调度的,任务的调度性需要进一步分析。

在 RMS 中,CPU 使用率的可调度条件为:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leqslant n(2^{\frac{1}{n}} - 1) \tag{5-1}$$

其中,C_i 是任务 i 的最大计算时间,T_i 是任务 i 的周期,C_i/T_i 是任务 i 对 CPU 的利用率。不等式(5-1)是一个充分条件,但不是一个必要条件,即如果任务的 CPU 使用率满足该条件,则任务是可调度的;但如果不满足该条件,也有可能被 RMS 所调度。

RMS 可调度的 CPU 使用率上限如表 5-1 所示。

表 5-1　RMS 可调度的 CPU 使用率上限

任务数量	可调度的 CPU 使用率上限	任务数量	可调度的 CPU 使用率上限
1	1	5	0.743
2	0.828	6	0.735
3	0.780	…	…
4	0.757	∞	ln 2

5.1.3　操作系统的资源管理

在计算机系统中,能分配给用户使用的各种硬件和软件总称为资源,包括两大类:硬件资源和信息资源。其中,硬件资源分为处理器、存储器、I/O 设备等;信息资源则分为程序和数据等。操作系统的重要任务之一是根据各种资源的共性和个性,有序地管理计算机中的硬件、软件资源,跟踪资源使用情况,满足用户对资源的需求,协调各个任务对资源的使用冲突,为用户提供简单、有效的资源使用手段,最大限度地实现各类资源的共享,提高资源利用率。

1. 处理器管理

现代操作系统大多采用多道程序设计技术,系统内通常会有多个任务竞争使用处理器,这就需要系统对处理器进行调度。鉴于处理器调度涉及的任务调度内容在 5.1.2 小节已有

阐述,本小节仅对处理器管理作简单描述。处理器使用权在以下几种情况发生转移:

(1) 当一个任务执行结束时,调度器会立刻选择一个就绪任务投入运行,如果有多个任务需要运行时,究竟选择哪一个任务投入运行,就由调度算法决定。

(2) 当某个任务在执行过程中要进行 I/O 操作或等待其他信息时,系统不会浪费 CPU 时间来等待 I/O 操作的完成,而是由调度器进行处理器的调度,将 CPU 分配给其他就绪任务使用。

(3) 在分时系统中,如果某个任务使用完本次分配的时间片而又未执行完成,也应该由调度器来进行任务调度,重新选择一个可以运行的任务投入运行。

(4) 当系统支持可抢占优先级调度时,如果有一个优先级比当前执行任务优先级更高的任务进入可调度队列,此时,调度器会中断当前任务的执行,调度运行优先级高的任务。

(5) 当有一个 I/O 中断发生时,意味着一个等待执行的任务需要的条件得到满足,又具有被调度执行的可能,此时也应该由调度器选择一个恰当的任务运行。

在所有处理器使用权发生转移的情况中都涉及 CPU 内部寄存器内容的保护和恢复。保护寄存器,即保存任务的上下文,比如,在 8086CPU 中,会保护 AX、BX、CX 和 DX 这 4 个通用数据寄存器,还会保存指针寄存器、状态寄存器(PSW)和相关段寄存器(CS、SS 等),这样可以使得该任务在下次调度执行时恢复这些寄存器的内容,从前面打断的地方继续执行。

8086

8086 是一个由 Intel 公司于 1978 年设计的 16 位微处理器芯片,是 x86 架构的鼻祖。8086 以 8080 和 8085(它与 8080 有汇编语言上的源代码兼容性)的设计为基础,拥有类似的暂存器集合,但是扩充为 16 位。

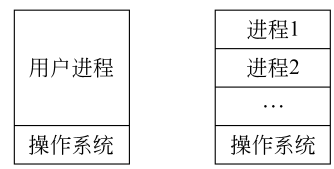
处理器作为计算机系统的关键资源,它的管理策略及调度算法会直接影响整个系统的运行效率。在多用户、多任务系统中,操作系统应该能够按照一定的策略与调度算法组织多个任务在系统中运行。

2. 存储管理

存储管理模块也是操作系统的重要组成部分之一,存储管理机制主要有单一分区、多分区、分页、分段及段页等多种不同管理方式。

1) 单一分区存储管理

该方式下,整个内存除了操作系统外,其余的内存空间只分配给一个进程使用,如图 5-8(a)所示。如果内存大小满足用户进程的要求,该用户进程可以使用;否则就不能运行,MS-DOS 就是采用这种管理方式。单一分区存储管理方式实现简单,但内存利用率不高,不能实现多任务,且系统可靠性不高。



(a) 单一分区　　(b) 多分区

图 5-8　单一分区和多分区管理

2) 多分区存储管理

多分区存储管理方式将内存除操作系统之外的空间划分成多个分区,如图 5-8(b)所示。每个分区分别分配给不同的进程使用,这样就简单地实现了多任务驻留内存,并以此提高了内存利用率。根据分区划分的方式,可以将多分区存储管理方式分为固定的分区管理方式和动态的多分区管理方式

两种。

固定的多分区存储管理方式在系统启动后，分区的大小和数目就是确定的，不再发生变化。动态的多分区管理方式则以更复杂的管理为代价换取更为灵活的存储管理。系统最初就是一个大的分区，根据进程请求分区的大小进行动态划分。这样，可以进一步提高内存利用率，当然，由于分区动态的分配与回收，系统为此需要花费比较大的系统开销。

多分区存储管理方式可以提高多任务管理效率，大大提高内存的利用率，动态多分区存储管理还能够实现一定程度的虚拟存储功能，但这需要更多的系统开销。

3）分页存储管理

分页存储管理首先将用户进程空间划分为一些大小相同的称为“页”的单位，物理内存也划分为与“页”大小相同的一些“块”。分页存储管理就是为这些页分配物理块的过程，这样一来，一个进程分配的物理空间就无须连续，而是以页为单位分散在“块”中。

4）分段存储管理

这种管理方式将进程的虚拟空间按照逻辑性加以划分，可分为程序段、数据段、堆栈等，实际上，8086 的汇编语言程序就是按段组织的，然后利用动态分区存储管理方式进行内存管理，只不过内存分配的对象不再是整个进程，而是进程的一个段。

5）段页存储管理

由于分页可以获得很高的内存利用率，分段方式便于实现共享，为了最大限度地提高内存资源利用率，可以将分页和分段两种方式结合起来，即采用段页式存储管理。

段页式存储管理将分页与分段两种方式相结合，首先将进程划分为若干个段，在段内再分页，这样，最小的逻辑空间单位仍是页；而对于物理空间的管理则采用分页管理方式中的“块”管理即可。段页式存储管理同时具有分页、分段存储管理方式的优点。

3. I/O 设备管理

在计算机系统中配置了多种 I/O 设备，如键盘、鼠标、显示器、网卡、打印机和磁盘等，这些设备可以是共享的，也可以是独占的。在多用户、多任务环境下，为了提高这些设备的利用率，实现资源的有效共享，操作系统应该高效地管理这些设备，主要有查询和中断两种设备管理方式。

在现代操作系统中，处理器通常与进程的输入/输出是并行工作的，即处理器在进程输入/输出时并不等待，而是执行其他任务。那么设备完成输入/输出后，如何通知系统以便进程的下一步执行呢？这就要借助于中断技术。

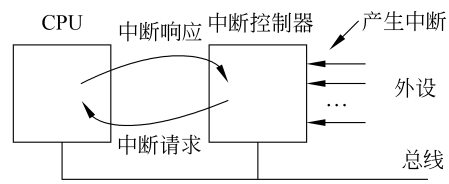


图 5-9　外部设备的中断管理结构

图 5-9 给出了外设以中断方式工作的组织结构。当设备完成了输入/输出后，会通过中断控制器向 CPU 发出硬件中断，这些设备的硬件中断事先都有固定的编号，称为中断号，同时，每个硬件中断都有相应的中断服务程序来完成特定的功能，比如，键盘中断服务程序负责把用户输入的按键值读取到 CPU 内部，以做进一步处理。中断服务程序的入口地址通常事先放在中断向量表中，在 8086 系统中该表在系统启动后就驻留在内存的低端（地址范围为 0～3FFH）。CPU 检测到中断请求后，如果决定响应，CPU 会从总线上获得中断号，以中断号为索引到中断向量表中找到中断服务程序的入口地址，并转入中断程序。中断服务程序开始执行后，会通过中断

控制器向设备使用的 I/O 端口做出应答,进行数据的传输。此后,中断控制器又可以继续向 CPU 发出下一个中断请求。

在操作系统管理外部设备的过程中,会涉及中断处理程序、设备驱动程序、设备无关软件和用户 I/O 软件,它们工作在不同的层次上,既相互独立又相互配合,共同完成外部设备的数据传输。比如,设备驱动程序是运行在系统内核与设备控制器之间交互的代码。由于要对设备进行控制,因此,设备驱动程序往往要了解设备控制器的特性。设备类型不同,驱动程序也不同,一般驱动程序都会由设备厂商提供。设备无关软件是操作系统内核中比较复杂、管理功能较为集中的模块。设备无关软件的主要任务是执行对所有设备公共的 I/O 功能,除设备管理功能外,它还要向下为所有的设备驱动程序提供一个统一的接口,并实现逻辑设备到驱动程序的映射功能及缓冲管理,另外,设备无关软件要向上为用户 I/O 软件提供统一的逻辑设备。

4. 文件管理

文件是计算机系统中保存信息的一种主要形式,文件的命名、组织、操作、逻辑结构、存取方式、物理结构、目录管理等都是文件管理的内容,通常把操作系统中进行文件管理的部分归为文件系统。

文件管理要求能够对文件进行逻辑组织,并且安排文件在外存上的物理存储,这就要求文件管理能够管理外存空间。文件是用户频繁使用的信息,文件管理还应该解决用户操作文件的一系列问题,如文件的创建、打开、关闭、读、写、删除等。为了方便用户使用文件,文件系统一般都支持以文件名进行访问的操作方式,用户无须了解文件的物理组织结构。当系统中文件数目较多时,文件管理应该能够将文件按照一定的方式组织起来,如采用树型目录结构,提供文件的高效存取访问功能。

另外,文件信息的安全问题也是文件管理的一个任务,可以通过设置文件的保护属性实现文件的安全管理。当然,文件系统的可靠性和文件的共享也是文件管理应该考虑的问题。

5.2 几种流行嵌入式操作系统平台

嵌入式系统无处不在,它是指执行专用功能并被内部计算机控制的设备或者系统,从消费电子设备的 PDA、MP3、手机、智能家电和车载电子设备到工业领域的数控机床、智能工具和工业机器人等各个行业,无一不在应用着嵌入式技术。嵌入式系统不使用通用计算机,而且运行的是固化的软件(固件,firmware),终端用户很难或者不可能改变固件,因此,在其上运行的嵌入式操作系统在诸多方面有别于通用计算机操作系统,目前已经出现了许多性能优良的嵌入式操作系统平台。

5.2.1 嵌入式实时操作系统 μC/OS-Ⅱ

1. μC/OS-Ⅱ概述

μC/OS-Ⅱ是一个实时操作系统(real-time operating system,RTOS),在嵌入式应用领