第1章 OS/2 2.0综述

本章将介绍：

- OS/2 2.0 的主要特点
- OS/2 2.0 与显示管理程序(PM)的优点
- 基本操作系统的概念
- PM 的概念
- 图形程序设计接口的概念

1.1 OS/2 2.0 的主要特点

OS/2 2.0 是一种面向个人计算机的先进的多任务、单用户操作系统。与这个操作系统以前的版本一样，OS/2 2.0 也提供了一个应用程序设计接口(API)，这个 API 可支持多任务、多线程、动态连接、进程通信、图形用户界面以及图形程序设计接口，这个操作系统以前的版本所具有的特性，如高性能的文件系统、扩展的文件属性以及长文件名等特性，在 OS/2 2.0 中也同样具有。

OS/2 2.0 的新特性有：

- 充分利用了 Intel 80386 处理器的特性，如平坦(flat)存储模式、分页虚拟存储以及增强的指令系统
- 在不同处理平台之间的可移植性
- 与版本 1.X 完全兼容
- 可执行多个 DOS 会话
- 支持 DOS 环境虚拟设备的虚拟设备驱动程序

1.1.1 386 特性

与此操作系统原来的版本不同的是 OS/2 2.0 充分利用了 Intel 80386 处理器的特性，如平坦存储模式、分页虚拟存储以及增强的指令系统。这就意味着 OS/2 应用程序将不像在以前操作系统的分段存储模式中那样来操作选择符和偏移量，而是将内存看作是一个大的线性地址空间，该地址空间可由相对于内存起始点的 32 位偏移量来寻址。分页虚拟存储使内存的管理更为高效，而增强的指令系统则使应用程序可在一条指令中处理 32 位数值。

1.1.2 可移植性

OS/2 2.0 采用平坦存储结构的一个好处就是应用程序与操作系统的可移植性好。为其它 32 位线性(平坦)存储模式的操作系统而编写的程序很容易移植到 OS/2 2.0 上,使用线性存储模式的 OS/2 应用程序将来也可方便地移植到在其它微处理器上运行的未来版本的 OS/2 2.0(或其它操作系统)上。由于硬件只需提供一个能对大的分页线性地址空间寻址的基寄存器和一个反映地址空间变址量的偏移寄存器,因此平坦存储结构很容易移植到大多数处理平台上。OS/2 2.0 不是专门为 386 设计的,而是一个在 80386 平台上实现的 32 位 OS/2 操作系统。

1.1.3 与版本 1.X 的兼容性

OS/2 2.0 需要用 80386 的特性,因而它不能运行在采用 Intel 80286 处理器的计算机上。这就是说,面向 OS/2 2.0 开发的应用程序不能在此操作系统的 1.X 版本下运行。但是,在版本 1.X 下所开发的应用程序可以在 OS/2 2.0 下运行,尽管它们不能充分利用 OS/2 2.0 操作系统的全部特性。

OS/2 2.0 虽然提供了一个完整的 32 位函数集,但它也支持对 16 位函数的调用。这就意味着系统将把某些 32 位函数转换成 16 位函数,负责完成转换工作的那部分代码称为*形实替换层*。形实替换层将 32 位参数转换为 16 位参数,并将线性地址映射为分段地址。形实替换层的作用对于应用程序开发人员来说是透明的,在一般情况下,不用管它,除非你要为 16 位动态连接库(DLL)开发自己的形实替换程序,以使 32 位和 16 位应用程序都可调用这些 16 位 DLL。

1.1.4 多 DOS 会话

80386 处理器使 OS/2 2.0 可同时管理一个以上的 DOS 应用程序。在处理器的虚拟 8086 模式下,每个 DOS 应用程序都在其各自的 DOS 环境中运行,OS/2 2.0 的这个特性称为增强的 DOS 会话。

1.1.5 虚拟设备驱动程序

OS/2 2.0 中的 DOS 环境提供了一种可扩展结构,它允许对 DOS 环境进行裁剪。这种可扩展性的核心就是虚拟设备驱动程序结构。所有在 OS/2 2.0 以前的版本中驻留在物理设备驱动程序中的低级 DOS 支持,现都已移至虚拟设备驱动程序中。由于处理器是工作在虚拟 8086 模式下,因而所有的中断处理都以保护方式进行,不再需要双模设备驱动程序。新的驱动程序结构提供了支持基本设备的物理设备驱动程序以及支持 DOS 环境下虚拟设备的虚拟设备驱动程序。从概念上讲,虚拟设备驱动程序提供了一种使得 OS/2 保护模式应用程序与 DOS 应用程序可共享物理设备的机制。

1.2 OS/2 操作系统和显示管理(PM)程序

在一个多任务环境中,每个应用程序都要占据一部分屏幕,通过屏幕来与用户进行交互,这一点是很重要的。OS/2 2.0 的一个主要目标就是能同时为大多数(即使达不到全部)应用程序提供可视访问。这种访问可通过让某个应用程序使用整个屏幕,而其它应用程序在后台等待,或者是让各个应用程序共享屏幕的方式实现。在 OS/2 2.0 中,当应用程序在某个会话中运行时,由这个会话来决定该应用程序是否可完全控制屏幕、还是必须与其它应用程序共享屏幕。

根据应用程序的类型,系统可在全屏会话中启动应用程序。在全屏会话中运行的应用程序对屏幕有完全的控制权。对屏幕拥有完全控制权的应用程序就称为全屏应用程序。

当操作系统一启动,就创建 PM 会话。PM 管理这个会话以及在此会话中运行的应用程序所使用的屏幕、键盘及鼠标资源。在 PM 会话中运行的应用程序称为 PM 应用程序。多个 PM 应用程序是共享屏幕、键盘及鼠标资源的。

每个 PM 应用程序都分配到一部分屏幕,称为窗口。PM 应用程序允许在屏幕上同时显示多个窗口,PM 应用程序就是以这种方式来共享屏幕的。

窗口是应用程序的用户接口,它可以包含菜单、滚动条及其它一些控制,用户就是通过这些控制来与应用程序进行交互的。

每个 PM 应用程序都必须在产生输出及接收输入之前创建其自己的窗口。操作系统可向应用程序提供一些诸如用户正在对窗口进行什么操作之类的详细的信息;操作系统还能自动执行许多用户请求的任务,如移动窗口或改变窗口的大小等等。

PM 应用程序可以以多种方式创建并使用任意数目的窗口来显示信息。PM 负责管理屏幕,具体地说,就是控制窗口的位置和显示,以保证不会有两个应用程序同时试图去访问同一部分屏幕。(在这种情况下,PM 会将一个应用程序窗口同另一个应用程序窗口重叠起来。)

1.2.1 队列输入

在传统的程序设计环境中,应用程序是通过显式地调用一个函数(如 GETCHAR)来读取键盘输入的。该函数通常要等到用户按下一个键之后才将字符码返回给应用程序。PM 应用程序则不是通过显式调用函数来读取键盘输入,而是由操作系统将所有来自键盘、鼠标及定时器的输入都接收入系统队列,然后再将每个输入从系统队列复制到应用程序队列,这样就将输入重定向到了应用程序。当应用程序读取输入时,下一个输入再被送到应用程序队列中,以此类推下去。当应用程序读取每个输入时,它会把消息发送到相应的窗口。

在 PM 应用程序中,来自键盘和鼠标的输入会自动送到已创建的每个窗口。操作系统所提供的输入的格式是统一的,这种格式称为输入消息,消息所包含的有关输入的信息远比在其它环境中可得到的信息多得多。它指明了系统时间、鼠标位置、键盘状态、键的扫描码(如果有键按下)、鼠标按键号(如果有鼠标按键按下)以及产生该消息的设备等等。例

如，键盘消息 WM_CHAR 相对应于某一个键的按下或放开。在每个消息中，除了由键盘产生的与设备有关的扫描码外，操作系统还提供了用于标识该键的与设备无关的虚拟键代码。消息中还指明了键盘上其它键的状态，如 SHIFT、CTRL 及 NUMLOCK 等键的状态。所有的键盘、鼠标和定时器消息都具有相同的格式，并以相同的方式被处理。

1.2.2 与设备无关的图形

在 PM 操作中，你可以访问一组与设备无关的图形操作。这就是说，你的应用程序可以方便地画出简单或复杂的形状；而且，在显示器上显示一个高分辨率图形与在点阵打印机上输出一个图形所使用的函数调用和数据都是相同的。

操作系统要求表示驱动程序将图形输出请求转变为打印机、绘图仪、显示器或其它输出设备的输出。表示驱动程序是一个由 PM 加载的可执行库，它是用来在指定设备的环境（即设备驱动程序、输出设备及通信口）中执行图形操作的。

1.2.3 共享资源

作为一个多任务系统，OS/2 2.0 要求 PM 应用程序必须与其它所有的正在相同会话或不同会话中运行的应用程序共享显示器、键盘、鼠标，以至于处理器。因此，操作系统严格控制这些资源，并要求应用程序使用一种特定的程序设计接口，以保证这种控制的正确性。

1.3 控制程序基础

控制程序包含了所有的 OS/2 函数，这些函数是用来创建进程和线程、访问磁盘文件和设备、分配内存以及检索或设置系统信息的。在 PM 应用程序中，这些函数一般用来执行那些设有相应窗口管理程序或图形程序设计接口函数的任务，全屏应用程序甚至几乎只用这些函数来与用户进行交互以及访问计算机设备。

1.3.1 多任务

多任务是 OS/2 2.0 的一个主要特性，它反映了系统对多个应用程序同时执行进行管理的能力。这种能力有助于计算机的优化使用。例如，在一个应用程序等待用户输入时，其它应用程序就可以开始运行，它们可以进行打印文档或重新计算电子表格等工作。

OS/2 2.0 支持两类多任务。一类是应用程序在另一个进程中启动其它程序，使其与应用程序并发运行。这些程序可以是应用程序的一个新的副本，或是另一个与应用程序一起工作的相关程序，或是与应用程序根本无关的程序。另一类多任务是允许应用程序在同一个进程中运行多个线程，不同的操作在应用程序中可分为多个任务来处理。

尽管在用户看起来所有的应用程序是在同时运行的，但实际上处理器在一个时刻内只能执行一个任务。处理器常常要等待像键盘或磁盘这样的较慢的设备的输入或输出，当处理器等待时，就可运行另一个任务。

一个应用程序常常要完成多个可同步的任务。为了提高处理器的使用效率，可将应用

程序设计成当一项工作正在执行时，另一项工作正在等待某件事情的发生。应用程序可编写成能与其它不相关程序共同执行的形式。

计算机资源分配的原则是使应用程序之间互不干扰，操作系统就是由此来管理应用程序的并发运行的。尽管如此，为了要共享 OS/2 环境，应用程序还必须计划并同步其例程的执行。

要想在多任务环境中协调共处，每个应用程序都必须能与其它应用程序进行通信、能同步其各项活动、以及能将其对资源的使用串行化。应用程序应能启动、停止以及控制其各个执行单元。

操作系统在三个层次上实现对多任务环境的控制，这三个层次是：会话、进程和线程。这种三层次结构有助于根据用户的需要来控制并发运行，也有助于根据开发人员的需要来设计可并发运行的应用程序。

图 1-1 所示为多任务管理的各部分之间的层次关系。

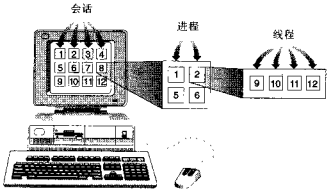

图 1-1 多任务管理的层次结构

会话

会话是 OS/2 多任务层次结构中的最高层，它由虚拟/逻辑设备及有关进程组成，这些虚拟/逻辑设备可以是屏幕(或窗口)、键盘或鼠标。当用户选择一个会话，使其在前台执行 I/O 时，逻辑设备就被映射到这些物理设备上。一般情况下，每个应用程序都在它自己的会话中运行；但在 PM 会话中，可以有多个应用程序共享物理屏幕、键盘及鼠标。

可将会话安排为父子层次结构。在缺省方式下，父会话可使用户能从 Workplace 外壳中选择子会话。一个会话可启动另一个独立的会话；在这种情况下，启动会话无法控制用户是否能选择第二个会话。如果一个会话正在前台运行，那么它可以将其任意一个子会话带到前台。父会话可中止它自己的子会话，但不能中止子会话的后代会话。但是，如果一个子会话中止了，那么操作系统会自动中止与其相关的后代会话。

进程

进程是分配给应用程序的资源的逻辑单元。这些资源包括内存、文件、管道、队列、系统信号灯及设备监控程序。每一个已装入内存并且正在运行的应用程序就称为一个进程。每个会话至少包含一个进程。

一个进程可创建其它进程，形成类似会话那样的父子关系。由父进程所创建的子进程可以继承对父进程所拥有的文件、管道及设备句柄的访问权。父进程对其子进程及后代进程保留有控制权。一个进程可控制其子进程的执行，并且可终止它自身及其子进程。

进程可将控制权交给当进程终止时将要执行的一组例程。这组例程称为出口表，它们负责在正常或异常中止后释放所分配的资源，以及完成一般性的内务处理工作。

线程

在进程中所进行的操作的最小单位称为线程。一个线程由指令、有关的 CPU 寄存器值及堆栈组成。一个进程至少有一个称为主线程或线程 1 的线程，进程也可以创建更多的线程。这些附加线程常用于执行与主线程处理无关的任务。例如，一个进程可能创建一个将数据写入磁盘文件的线程，这就使得主线程能腾出空来继续处理用户的输入。线程不占有系统资源，但可共享由创建线程的进程所拥有的资源。当操作系统将控制交给进程中的线程时，应用程序即可运行。线程不能构成层次结构，进程中的每个线程地位是等同的。

一个线程可利用 API 函数启动进程中的其它线程。一个线程还能暂时挂起和恢复其它线程的执行。这就使得线程可以访问时间关键资源。尽管进程中的各个线程处于同等地位，但如果进程中的第一个线程终止了，那么其余所有由它创建的线程也就终止了。当一个线程执行完毕后，它有必要调用一个 API 函数来终止它自己。

调度优先权：除非由 API 函数显式定义，一般是操作系统给每个线程赋予一个调度优先权。处理器时间一次只能分配给一个线程。操作系统采用时间片的方法来保证优先权相同的线程具有相同的运行机会。当一个线程的时间片已用完或是有更高优先权的线程准备执行时，操作系统可以挂起这个线程。缺省的最小时间片是 32ms。通过 API 来访问的 OS/2 定时器服务支持所有与定时器有关的操作，如线程活动的同步等等。

当一个线程被创建时，它的优先权类别、以及在类别中的优先权级别与创建这个线程的线程是一样的，应用程序可调用 API 来查询或改变线程的优先权类别和级别。优先权类别包括时间关键、固定、常规及空闲状态四类。每一类中又有 0～31 共 32 个优先权级别。优先级为 31 的线程总是在优先级为 30 的线程之前获得时间片，以此类推。线程按如下顺序调度：

优先权类别	何时调度线程
时间关键	按优先级立即并连续调度。(这些线程管理通信工作或实时应用程序。)
固定	当设有时间关键的线程等待时。
常规	根据显示状态(前台或后台)、最近的 I/O 活动及处理器的使用情况来调度。大多数线程都属于常规优先权一类。
空闲状态	在没有常规、固定或时间关键优先权的线程等待时。

尽管应用程序可在任何时候设置线程的优先级，但只有那些具有多个线程或进程的应用程序才需要这么做。优先权的最佳用途是提高那些对其它线程起决定作用的线程的速度。例如，如果有一个线程正在等待文件装入，那么应用程序就可暂时提高负责装入文件的那个线程的优先权。由于一个线程的优先权是相对于系统中其它所有线程而言的，因此，仅仅为了获得额外的 CPU 时间而去提高应用程序中线程的优先权会对整个系统的运

行产生不利影响。

1.3.2 动态连接

动态连接使得应用程序能够在运行时访问一些不属于其可执行代码的函数。这些函数包含在动态连接库(DLL)中,DLL 是包含可执行代码但却不能作为应用程序运行的模块。应该由应用程序装入相应的 DLL,并与其动态连接起来,这样就可执行库中的代码。

DLL 在 OS/2 2.0 中是很普遍的。实际上,操作系统的大部分内容都包含在 DLL 中。DLL 的主要优点是可以减少应用程序所需的内存。应用程序只有在需要执行 DLL 中的函数时才装入 DLL。DLL 一旦被装入,它就可由系统中其它需要它的应用程序所共享。这就是说,在任一时刻,只装入 DLL 的一个副本。

1.3.3 内存管理

所有的 32 位应用程序在任何时候都可将附加内存分配给它们自己使用。在操作系统的 1.X 版本中,应用程序以段的方式来申请内存。段是基于 Intel 80286 处理器体系结构的虚拟和物理内存分配单位。在 OS/2 2.0 中,应用程序以对象的方式来申请内存。对象是 32 位平坦结构中的内存分配单位。

内存对象是以 4KB 为单位进行分配的。一个 4KB 单位称为一页。对象中的每一页都可能有两种状态:未提交的(即保留了线性地址范围,但还未分配相应的物理存储)和提交的(已经为逻辑地址范围分配了物理存储)。

在 OS/2 2.0 中,需要处理器立即执行的代码和数据保存在物理内存中;而不是处理器马上要执行的代码和数据则保存在外存设备(交换空间)中。

内存分配函数返回指向内存对象的 32 位指针。内存对象的大小可在 1 页(1 页=4KB)到可用交换空间所支持的任意大小之间变化。所有的指针引用都是 32 位近指针。由于此处不涉及段寄存器,因此所有的段寄存器是相等的,即 CS=DS=ES=SS。虚拟内存以请求调页而不是以压缩和段交换的方式工作。这对确定内存对象的大小以获得系统最佳性能具有重要意义。所有的内存分配,无论是私有的还是共享的,都确保用零填充提交页,应用程序开发人员可根据这个来确定内存中的初始内容。

OS/2 2.0 可保护内存,禁止非法使用。内存由分配内存的进程所占有,其它进程不能访问。企图去访问其它进程所拥有的内存将导致保护失败,通常并导致进程终止。

如果两个进程需要共享内存,那么其中一个进程可创建一个共享的内存对象,然后或者是将指针传给另一个要共享此内存的进程,或者是将共享内存的名字传给另一个进程。共享进程必须管理共享内存。

1.3.4 文件系统

OS/2 的文件系统允许应用程序在外部设备上组织和维护数据,并提供了存储介质的逻辑视图,这就使得应用程序即使不熟悉每种设备的特性也能管理数据。

操作系统有两个文件系统:文件分配表(FAT)文件系统和高性能文件系统(HPFS)。FAT 文件系统是操作系统缺省的文件系统,它不需要安装;HPFS 可在系统初始化时安

装。HPFS 可以快速一致的方式管理大的磁盘介质，并支持长文件名格式的文件。

文件系统在物理磁盘上是以分层结构组织的。磁盘可以被进一步分为两个或更多个逻辑盘(即分区)，每个分区都有自己的层次结构。

文件系统中的基本元素就是文件。一个文件代表着一串字符流。由多个文件所组成的层次结构就称为目录。每个目录层次结构中至少包含一个根目录，它位于每个逻辑磁盘的层次结构的最顶层。目录中还可包含其它目录，称为子目录，它们是较低层次上的文件集合。应用程序可用目录和子目录来将磁盘上的内容组织成逻辑上的多个文件组。

操作系统为应用程序提供了用于创建、打开、读、写和关闭文件以及创建和删除子目录的 API 函数。文件一旦被打开，就赋予它一个称为文件句柄的标识符，其它有关的进程可用文件句柄来引用该文件。

OS/2 2.0 还支持扩展属性，这就使应用程序可将各种信息附加在文件对象上。扩展属性包括一个 ASCII 文本名和一个任意的值。有关扩展属性的命名及数据类型的定义等内容，请参看《OS/2 编程指南，第 1 卷》。

由于多任务环境中的任何一个应用程序都可以调用 API 函数，因此两个应用程序有可能在同一时刻请求访问同一文件。为了避免这种冲突，操作系统提供了一些措施，在请求时对访问文件中的信息加以控制。打开文件的进程可以决定其它进程以何种方式共享这个文件。假如文件较大并需经常使用，那么进程可用文件加锁函数来保护文件的一小部分，而使文件的其余部分为可用。

OS/2 2.0 基本上是以同样的方式管理其磁盘文件和设备的。例如，应用程序可与打开并从串行口读入一样用同样的函数来打开并读取磁盘文件。每个打开的文件或设备都用一个唯一的文件句柄来标识。应用程序在系统函数中用该句柄来访问文件或设备。

一个标准设备与文件一样也代表一串字符流。多数情况下，对于应用程序来说，设备与文件是非常相似的。同文件一样，设备也由 ASCII 名来标识。在《OS/2 编程指南，第 1 卷》中介绍了一些特殊的命名约定。

操作系统为应用程序提供了用于创建、打开、读、写和关闭设备的文件子系统 API 函数。在字符设备(一次处理一个字符数据的设备)上进行的 I/O 操作必须以串行方式处理。当设备被打开时，它就被赋予一个称为设备句柄的标识符。进程可用设备句柄来进行输入、输出以及其它从设备读或向设备写等操作。

当进程打开一个文件时，它就指定该文件是否可被共享，即该文件是否可被其它进程访问甚至修改。这种共享机制也适用于一个进程打开设备的情况。进程可直接打开任何设备，包括并行口与串行口。OS/2 2.0 提供了很多 I/O 控制函数，进程可用这些控制函数来访问已打开的设备、或设置已打开设备的模式。

在 OS/2 环境中启动的每个进程都可以使用一组标准的设备句柄，它们是系统的标准输入设备(通常是键盘)、系统的标准输出设备(通常是屏幕)以及用于输出错误信息的标准错误设备(通常是屏幕)。

通常，当应用程序启动时，系统会自动打开三个文件：标准输入、标准输出及标准错误文件。这些文件句柄对应于键盘和全屏显示器。应用程序可利用这些文件句柄来从键盘读入或向全屏显示器输出。

应用程序不能区分文件句柄、设备句柄和管道句柄。使用标准设备句柄的好处在于无需应用程序的干预就可将文件中所存储的数据重定向到设备上。而且，一个应用程序可利用管道将其输出数据流重定向到另一个程序的输入数据流上。

1.3.5 进程通信

在多任务环境中，各个进程与线程之间要相互通信，以使各事件同步，并对共享资源的访问加以控制。为此，操作系统提供了一套进程通信(IPC)协议。这些协议包括信号灯、管道、队列、共享存储器、异常处理、多 DOS 会话及设备支持。

信号灯

信号灯是进程用来报告某个操作的开始或结束，以防止进程中有多个线程同时去访问某个特定的资源。进程可创建和使用三种信号灯：互斥(mutex)、事件和多重等待(muxwait)。

互斥信号灯是进程中的几个线程或者是几个进程用来对临界区的访问加以保护的。例如，可用互斥信号灯来防止多个线程同时修改一个磁盘文件。

事件信号灯是线程之间或进程之间发送信号的手段。典型的用法是用于管理共享存储器。例如，进程 1 向共享区写入数据后就用事件信号灯通知进程 2 和进程 3，告诉它们可以开始访问共享数据。

多重等待信号灯允许线程同时等待几个事件或互斥信号灯，它是一种最多可包括 64 个事件信号灯或互斥信号灯(这两种信号灯不能混用)的复合信号灯。典型的用法是当一个线程要立即访问几个共享存储区时，这时系统将阻塞这个线程，直到线程获得了所有的保护共享区的互斥信号灯为止，然后线程才能访问这些共享区。

管道

管道是两个进程相互通信的手段，它们是由句柄来标识并且可以像文件那样被访问。为了进行通信，进程先打开一条管道，继而一个进程得到管道的读句柄，另一个进程得到管道的写句柄，然后这两个进程就分别用这两个句柄来向管道写入数据和从管道读出数据，这样就实现了两进程间的通信。一种典型的情况就是用管道将一个进程的输出定向到另一个进程的标准输入上。

OS/2 2.0 中有两种管道：有名管道与无名管道。无名管道仅用于相关进程(父进程与子进程)之间的通信。有名管道可用于运行在相同或不同计算机上的不相关进程间的通信。任何知道管道名字的进程都可打开和使用有名管道。一个进程(服务员进程)创建和连接一条管道，另一个进程(客户进程)则打开该管道。一旦管道被连接与打开，则服务员与客户进程就可以来回传递数据了。

客户进程可以是本地的(与服务员进程在同一台计算机上)，也可以是远程的(客户进程与服务员进程通过局域网(LAN)连接起来)。

队列

队列是一个有序数据表，进程可用它接收来自其它进程的信息。各进程将信息以元素的形式传送给队列，而后拥有该队列的进程就可以从队列中读取各个元素。(PM 应用程序也有队列，称为消息队列，消息队列和此处讲的队列是不一样的。)

队列中的元素可被单独访问，访问它们的顺序可以是先进先出(FIFO)，也可以是后进先出(LIFO)。应用程序还可定义队列元素的优先级。尽管多个进程可向队列写，但只有创建这个队列的进程才能从队列读。队列中的各元素可被检查而不必从队列中删除。

共享存储器

OS/2 2.0 允许系统中的两个进程，甚至全部进程共享一个内存对象。应用程序必须显式地请求访问共享存储器，共享存储器是不允许未获准的应用程序访问的。

两个进程若要共享存储器，则其中一个进程分配一个内存对象并指明该对象可被共享，然后这个进程调用一个函数通知系统此时第二个进程可以共享存储器，操作系统返回线性地址，该地址唯一地标识了上述内存对象。接着第一个进程以某种形式的 IPC(如队列)将该线性地址传给第二个进程。

如果系统中所有的进程要共享一个内存对象，则由分配内存的进程给对象赋予一个名字，然后再利用 IPC(如队列、管道)将名字传给其它进程。其它进程就用此名字来请求访问该对象，并由操作系统将选择符返回给该内存对象。这种技术称为*有名共享存储器*。

异常处理

在 OS/2 2.0 中，应用程序可在内部处理某些意想不到的错误(如内存保护错误或来自其它进程的终止信号)而无需终止。

像 OS/2 2.0 这样的多任务操作系统必须仔细地管理所有应用程序。绝不允许一个应用程序中发生的严重错误(如企图访问被保护的内存)破坏系统中的其它应用程序。为了处理可能破坏其它应用程序的错误，OS/2 2.0 定义了一个称为"异常"的错误情况类，还定义了处理这些错误的缺省操作。

当发生异常、并且引发异常的应用程序没有注册它自己的异常处理例程时，操作系统就要终止该应用程序。应用程序可用异常处理例程来从意外的事件中恢复，而且有可能排除异常，继续运行。

多 DOS 会话

OS/2 2.0 通过 80386 微处理器的虚拟 8086 模式提供了与 DOS 的兼容性，而且提供了多个 DOS 兼容环境。由于 OS/2 2.0 所提供的 DOS 环境是将整个 DOS 环境封装在一个虚拟机里，因此 OS/2 2.0 中的 DOS 环境的 DOS 兼容性比以前版本的要好。OS/2 2.0 还能执行多个并发的 DOS 会话。这种实现方法为 DOS 会话提供了具有优先权的多任务管理，并且允许正常 OS/2 级别的存储保护；也就是将系统内存与其它应用程序的内存隔离开，防止不可预知的应用程序非法访问内存，并具有在应用程序'死机'的情况下终止会话的能力。

DOS 应用程序可在全屏幕(如窗口)中运行，也可变成肖像在后台运行，除了具有较好的保护能力、较好的兼容性、多个 DOS 会话以外，OS/2 2.0 的 DOS 环境还提供了 600KB 以上的空间供应用程序运行。

设备支持

设备驱动程序是应用程序、操作系统以及硬件之间的软件接口。设备驱动程序使应用程序和操作系统在某种程度上可与设备无关。如果没有设备驱动程序这一层，那么每增加一种新的设备，就不得不重写操作系统。

有两种类型的设备：字符设备和块设备。字符设备以串行方式处理数据流，块设备通常保存着大量数据，这些数据可以串行或随机的方式访问。块设备包括磁盘和虚拟磁盘。成块数据的传送对于应用程序来说是透明的。应用程序利用文件系统的 API 函数来完成 I/O 操作。

在多任务环境中，多个应用程序可共享包括设备在内的资源。设备驱动程序则管理设备以保证应用程序可以访问它。应用程序利用 API 函数与操作系统进行通信，操作系统则利用 I/O 控制(IOCtl)接口与设备驱动程序进行通信。

设备驱动程序常常要访问一些系统服务，如内存管理或进程通信等。这些服务由设备辅助(DevHelp)例程提供。这些服务包括：

- 设备队列管理
- 线程的同步
- 存储管理
- 进程管理
- 中断管理
- 定时器管理
- 数据缓冲区的监控

由于 OS/2 2.0 提供了物理的与虚拟的两种设备驱动程序，因而 OS/2 2.0 的设备驱动程序模型结构可将 DOS 支持与基本的 OS/2 设备支持分开。大多数 16 位的 OS/2 设备驱动程序按照现在这样工作，除非它们用 DevHelp 函数 SetROMVector 来为 DOS 环境提供支持。在后面这种情况下，驱动程序还可以工作，但不是在 DOS 环境下。驱动程序中以双模型代码形式编写的代码仍可在 OS/2 2.0 下运行，但只有保护模式的代码才会被调用。

OS/2 2.0 的基本设备驱动程序模型也可加以增强，以支持分页环境下的设备。大多数采用直接存储访问(DMA)的设备无法处理不相邻的内存块，而一旦允许分页，内存块不相邻的情况是会常常发生的，因此，设备驱动程序应按照设备处理分页环境的能力来编写。

1.4 PM 基础

PM 为 OS/2 环境提供了一个基于消息的、事件驱动的图形用户界面。PM 的主要特性有：

- 窗口环境
- 用户界面
- 输入管理
- 应用程序资源管理
- 数据交换
- 信息表示工具
- 表示驱动程序

PM 使程序员能建立符合系统应用程序结构(SAA)准则的应用程序。有关 SAA 要求的内容,详见《系统应用程序结构(System Application Architecture);通用用户界面设计指南(Common User Access Guide to User Interface Design)》及《系统应用程序结构(System Application Architecture);通用高级界面设计参考手册(Common User Access Advanced Interface Design Reference)》等书。

1.4.1 窗口环境

PM 用户界面是建立在窗口基础上的,窗口是与用户进行交互所用的一块屏幕。大量的 API 函数(均以 Win 开头)可用来对窗口进行控制。这些函数允许应用程序创建或移动窗口、改变窗口大小、控制窗口及窗口中的内容。《OS/2 程序设计指南,第 2 册》中介绍了管理窗口环境的一般编程技术。

定义窗口关系

窗口是一块屏幕,应用程序就在这块屏幕上显示输出、并接收来自用户的输入。一个屏幕上可有多个窗口。作个比喻,屏幕上的多个窗口就像桌面上的许多张纸一样。在这个比喻中桌面就是包含屏幕背景的区域,而窗口就像纸一样可由我们来排列,或者前面的窗口覆盖后面的窗口,或者互相重叠起来。如果窗口是重叠的,则下面的窗口可能被遮盖掉一部分,也可能被全部遮盖掉。可以用 API 来将各个窗口定义在窗口层次中。

图 1-2 为屏幕上所显示的窗口层次结构。

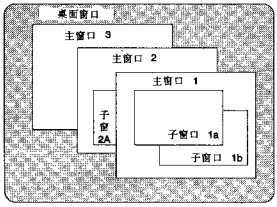

图 1-2 屏幕上的窗口

桌面窗口处于层次结构的最顶层。桌面窗口的下面是顶层窗口,称为主窗口。主窗口可相互重叠,有时,一个主窗口可完全隐藏起来。各个主窗口的操作互不影响。

图 1-3 所示是由应用程序创建的窗口的分层排列。

主窗口可以以传代的父子关系创建从属窗口。子窗口总是根据其父窗口来裁剪,也就是说,只有落在父窗口内的那部分子窗口才是可见的。

具有同一个父窗口的称为兄弟窗口。像主窗口一样,兄弟窗口也可相互重叠。每个兄弟窗口都有一个 Z 坐标,这个 Z 坐标指明它在重叠窗口堆栈中的位置。

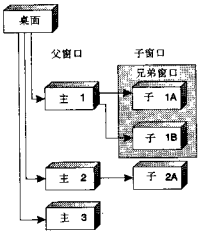

图 1-3 窗口的分层结构

除了分层关系，应用程序还可定义另一种关系。当创建一个窗口时，可定义一个宿主窗口，这两个窗口必须由同一个线程创建。主从关系在层次结构的不同层次上是不同的。子窗口可向其宿主窗口发送消息。如果一个主窗口拥有其它主窗口，并且宿主窗口是隐藏起来、最小化或关闭的，那么所有从属于它的主窗口也是隐藏、最小化或关闭的。

窗口在屏幕上可以是可见、隐藏或部分隐藏的。当一个窗口被隐藏或部分隐藏起来时，其大小、位置及在层次结构中的主从关系均与原来一样。但是，当该窗口重新成为可见窗口时，窗口中原来隐藏的那些区域都要进行重画。一个窗口也可被禁止，即该窗口仍然是可见的，但不能响应鼠标输入。

窗口的创建与分类

我们可以把一个窗口和与之相关的窗口函数看作是一个程序对象。窗口函数控制着窗口的各个方面，如窗口的模样、窗口如何响应各种变化以及窗口如何处理输入等，因此从这个意义上讲，窗口函数就代表着窗口。请看"消息处理"一节中有关窗口函数的详细介绍。

一个窗口类是由同一个窗口函数来实现的一组窗口。可有许多窗口属于一个窗口类，这些窗口之间的区别仅仅在于它们所处理的数据不同。如果有多个应用程序需要同一种窗口，那么实现共同的窗口类则可算是高效利用系统资源的一种方法。

OS/2 2.0 提供了许多预注册的窗口类。这些类中所定义的窗口是专门用于适应基于图形的标准用户接口的需要的。其中有的窗口类就实现了如在"标准和控制窗口"一节中所介绍的那些窗口。如果没有提供预注册的窗口类，则应用程序必须在进程层次上注册该窗口类。一些 API 函数可被应用程序用来为其注册的窗口类中的窗口保留一小块内存（窗口字区域）。如果一个窗口要处理大量的数据，则应将这些数据保存在内存中，并从窗口字区域中引用。

窗口类可被定义成公有(public)或私有(private)。公有类或私有类中所创建的窗口均可被系统中的任何进程使用。"公有"及"私有"仅仅是在创建窗口时说明窗口类的。

只有私有类注册的那个进程才可创建该类的窗口。类名对于该进程来说必须是唯一的。但其它进程也可用相同的类名注册私有类。

任何进程都可创建公有类的窗口。公有类窗口的窗口函数必须可为所有的进程使用，因此，这样的类必须定义在 DLL 中。公有类的类名必须对所有的进程是唯一的。

所有的窗口都具有某些属性。每个窗口都由一个窗口句柄来标识。每个窗口都代表一个矩形，该矩形规定了屏幕上窗口的大小和位置。窗口的大小用相对于父窗口原点的图象元素(pel)数来表示。窗口的原点，即窗口左下角是 x,y 轴坐标系中的 0,0。x 和 y 坐标定义了窗口的顶、底和边。坐标在每个方向上的变化范围是－32768 到＋32767pel，因此在任一方向上可表示的最大尺寸为 65535pel。

应用程序可通过定义窗口到父窗口原点(0,0)的相对距离来对窗口进行定位。如果在允许的情况下，应用程序将子窗口定位于其父窗口之外，那么只有在父窗口内的那部分子窗口在屏幕上才是可见的。

应用程序可用一组 API 函数来改变窗口类中的窗口行为，或者从现有窗口类再创建一个新的类。这种处理称为再分类，它使应用程序可修改单个窗口的行为而无需重写整个窗口函数。

1.4.2 提供用户界面

通用用户访问(CUA)是一个设计与编写应用程序用户界面的规则集。这些规则包括标准的菜单条项、交互技术及窗口类型等。请使用 CUA 规则来帮助你设计应用程序的用户界面。

许多缺省的 PM 服务都有助于保持那些按照 CUA 规则设计与编写的应用程序之间用户界面的一致性。选择适当的选项也可保证一致性。保证这种一致性是应用程序开发人员的责任。

标准窗口与控制窗口

系统是通过窗口在屏幕上显示信息的。PM 支持一种标准窗口，这种窗口的各个元素一般都符合 CUA 规则。

应用程序可用 API 函数来控制标准窗口，标准窗口可包含下列元素的全部或其中几个：

- 标题条图标
- 窗口边框
- 窗口大小按钮
- 菜单条
- 滚动条
- 窗口标题
- 信息栏

图 1-4 给出了一个标准窗口及其元素。

标题条图标、窗口边框及窗口大小按钮都能使用户改变窗口的大小和位置。菜单条和滚动条使用户能与窗口中的内容打交道。窗口标题指示窗口中对象的名称，同时它也指出

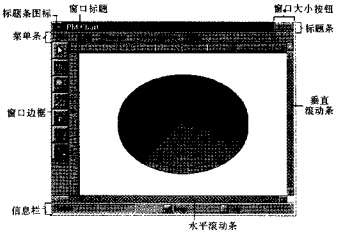

图 1-4　标准窗口

所显示的是哪种视图。视图是一种观察对象信息的方式。不同的视图以不同的形式来显示信息，即模拟现实世界里信息的表示方法。信息栏向用户显示一些有关光标所在的对象或选项的简短信息。信息栏中还可能出现某些处理正常完成的信息。例如，如果用户要将一个容器中的几个对象复制到另一个容器，那么容器窗口的信息栏会显示一条简短的信息，告诉用户复制何时完成。

标准窗口是用标准的框架窗口创建的。标准窗口中的各个元素，如标题条和菜单条，都是标准框架窗口的子窗口。这些子窗口称为控制窗口。系统维护着一组预注册的控制窗口，任何应用程序都可用这些控制窗口来进行 I/O 操作。

从应用程序的角度来看，控制窗口与系统中的其它窗口没什么区别，应用程序可用窗口管理函数来管理它们。每个控制窗口都有自己的窗口标识符和一组特定的消息。应用程序可向系统查询控制窗口的父窗口。

控制窗口可被用作对话窗口的一部分，对话窗口可用对话模板来创建，对话模板定义了对话窗口及其每个子窗口的位置、外观和标识符。模板可像资源那样加载或者在内存中动态地创建。它可被用于创建各种窗口类的对话窗口，窗口类可包含所有窗口类的控制窗口。另外，应用程序也可通过创建和预注册其自己的控制窗口类来创建自己的对话控制。

对话窗口是由一种称为对话函数的窗口函数来控制的。对话函数负责对所有发给对话窗口的消息作出响应，它或是将这些消息发送给控制窗口，或是将其返回给缺省的对话函数。有一组 API 函数可供应用程序用来创建、加载、处理及删除对话窗口。对话函数可获取其子窗口的句柄、向子窗口发送消息以及处理它自己的消息和正文串。

标准的框架窗口与控制窗口都是用标准的预注册窗口类实现的。当用户与控制窗口和用户窗口进行交互时，标准框架窗口对这两种窗口进行管理。框架窗口还负责消息的路径选择，使之能发送到正确的控制窗口和用户窗口。

主要窗口与次要窗口

CUA 规则定义了两种窗口：主要窗口与次要窗口。在 PM 程序中，主要窗口就是标准

窗口，而次要窗口就是控制窗口或是主窗口的子窗口。

主要窗口是对象与用户之间的主要接口点，它在用户打开一个对象时出现，并当所显示的有关对象或对象组的信息与其它对象无关时被用来显示对象或对象组的视图。

对象信息常显示在窗口菜单条下面的区域中。用户可以控制主要窗口在屏幕上的大小和位置。

次要窗口看上去很象主要窗口。例如，二者都具有窗口边框和标题条。主要窗口与次要窗口的主要区别在于如何使用它们。一般，次要窗口都与某个主要窗口相关联，并包含一些与主要窗口中的对象有关的信息。次要窗口的用途之一是用来进一步阐明动作请求。次要窗口总是在主要窗口关闭或最小化时被删除，并在主要窗口打开或恢复时被重新显示。

对话框

对话框在用户与主要、次要窗口之间展开对话。它通常是在用户从菜单条中选中了一个选项并产生下拉菜单时出现。从下拉菜单中选择一项就会产生对话框。对话框可包含按钮、入口字段、图标和正文、列表框以及标题条等。对话框及其支持窗口使应用程序能接收用户的输入。有时，为了特殊目的的输入，可创建一个临时对话窗口，然后再删除。

对话框有两种类型：有模式与无模式。有模式对话框保持控制，直到应用程序发出调用删除它。用户在尚未结束与有模式对话框的交互之前不能激活应用程序的其它窗口。无模式对话框在创建后则允许激活其它应用程序的窗口。

1.4.3 处理鼠标和键盘输入

操作系统中的会话管理器管理着在 PM 环境下运行的应用程序，其中包括它们的输入和输出操作。但是，PM 管理着 PM 应用程序，其中也包括输入和输出。PM 将所有的输入都作为消息来处理，消息就是数据包。

PM 支持来自键盘和鼠标箭头的用户输入。鼠标箭头是与鼠标指示装置相关的符号。鼠标输入通常是由按下一个按钮所产生的，并且通常是指向鼠标箭头下面的窗口的。屏幕上被激活的那个精确的位置称为热点，鼠标箭头还可在屏幕上移动，这是由操作系统来提供支持的。应用程序可将所有的鼠标输入都定向到一个窗口，该窗口称为*鼠标捕获窗口*。鼠标捕获窗口使得无论鼠标箭头移到屏幕上什么地方，应用程序都能跟踪所有的鼠标输入。

当按下键盘上的任意键时就会产生键盘输入。所有的键盘输入在一段时间内都定向到一个窗口。能接收键盘输入的窗口称为*活动窗口*。主窗口或它的其中一个子窗口使得接收输入的窗口在屏幕上总是可见的。

光标是在窗口中显示的一个记号，它表示了从键盘输入的字符将要放置的位置。光标可移动到窗口内的任一位置，其大小和位置是用相对于它所在窗口的坐标定义的。应用程序可以创建、显示、移动和删除系统光标。

消息处理

通用程序设计接口(CPI)定义了一个应用程序结构，该应用程序结构用一个队列和应用程序窗口函数的系统来进行消息处理。PM 消息系统符合 CPI，它是 PM 环境正确运

行的基础。SAA 库中提供了一套完整的 CPI 参考手册。

图 1-5 为 PM 消息处理系统。

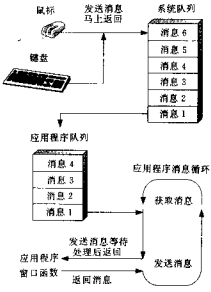

图 1-5 消息处理系统

在 PM 环境中，所有的鼠标和键盘输入都是作为消息传递给应用程序的。消息的处理不能在其前面的输入以前进行，这是因为只有将所有前面的输入都处理完才能知道所要接收输入的具体是哪个应用程序和窗口。

所有的输入都首先被放在一个称为系统队列的队列中。系统队列由系统中所有的应用程序所共享，它接收用户产生的鼠标和键盘消息。系统队列大约可容纳 60 个按键和鼠标按钮的输入。系统队列可接收许多来源的输入，如系统本身、定时器及其它应用程序等，但是只有用户输入是同步处理的。

系统队列可暂时保存用户输入，因而即使用户输入数据的速度超过应用程序处理的速度，也不会丢弃任何信息。一般情况下，是按照输入在队列中的次序来处理输入的，但应用程序也可用过滤输入的办法来改变处理顺序。过滤是通过 API 函数完成的。但进行过滤必须慎重，因为对一个输入的处理常常会影响后面的情况。

每个接收输入的线程都具有一个由 API 函数所分配的应用程序队列。这种队列不直接接收用户输入，但可直接接收其它消息，如来自系统或定时器的消息。当消息所要到达的线程"获得"了该消息时，该消息就从应用程序队列中删去。如果有多个消息在等待，则消息按优先级排队。如果没有消息，则线程被挂起，直到某个消息来到。

如果你在设计应用程序时，保留一个线程响应用户输入、而使其它线程连续进行处理工作，那么这样使用系统将是最高效的。为了能对用户及时作出响应，系统必须在 0.5 秒钟之内完成对输入的处理。也就是说，处理输入的线程必须在这段时间内检测队列中的下一个消息。