第3章 邮 槽

Microsoft Windows NT、Windows 2000、Windows 95和Windows 98（含第二版）——但不包括Windows CE——提供了一种简单的单向“进程间通信”（interprocess communication, IPC）机制。这个机制的名字非常古怪，叫作“邮槽”（Mailslot）。用最简单的话来说，通过邮槽，客户机进程可将消息传送或广播给一个或多个服务器进程。在同一台计算机的不同进程之间，或在跨越整个网络的不同计算机的进程之间，协助进行消息的传输。用邮槽来开发应用程序是一件非常简单的事情，不要求对TCP/IP或IPX这样的基层网络传送协议有着非常深入的了解。由于邮槽是围绕一个广播通信体系设计出来的，所以当然不能指望能通过它实现数据的“可靠”传输。然而，在某些特殊类型的网络编程环境中，假如对数据传输的可靠性要求不高，那么邮槽仍然是一种非常有价值的技术。

例如，要想在自己办公室的所有人员之间建立一个简单的传信系统，便可考虑用邮槽来设计这个程序。想象自己的办公室环境拥有大量工作站。恰巧，办公室目前苏打短缺。每隔几分钟，每名工作站用户都有兴趣知道苏打机里还剩下多少可乐。利用邮槽，便可很轻易地设计出这种程序。我们可简单地编制一个邮槽客户端应用，用它监视苏打数量，并以五分钟为周期，将剩余的可乐数量广播给每一名感兴趣的工作站用户。由于邮槽并不能担保广播消息的可靠传输，所以有些工作站用户也许收不到所有的更新通知。但在这种情况下，少数的传输失败并不是个问题，因为消息每隔五分钟便会发送一遍，所以即使偶尔出错，收到信息的频率也相当高，足以让工作站用户跟上目前的最新情况。

邮槽最大的一个缺点便是只允许从客户机到服务器，建立一种不可靠的单向数据通信。而另一方面，邮槽最大的一个优点在于，它们使客户机应用能够非常容易地将广播消息发送给一个或多个服务器应用。

本章将向大家解释如何开发一个实际的邮槽客户机／服务器应用。在深入解释消息的大小问题之前，首先要介绍邮槽的一系列命名规范，它们对邮槽的总体行为进行着控制。接下来，我们将详细讲述如何开发一个基本的客户机／服务器应用。在本章结束时，则会告诉大家邮槽一些已知的问题，以及存在的各种局限，并提出相应的解决方案。

3.1 邮槽实施细节

邮槽是围绕Windows文件系统接口设计出来的。客户机和服务器应用需要使用标准的Win32文件系统I/O（输入／输出）函数，比如ReadFile和WriteFile等等，以便在邮槽上收发数据，同时利用Win32文件系统的命名规则。邮槽必须依赖Windows重定向器，通过一个“邮槽文件系统”（Mailslot File System, MSFS），来创建及标识邮槽。第2章已对Windows重定向器进行了比较详细的说明。

3.1.1 邮槽的名字

对邮槽进行标识时，需遵守下述命名规则：

\\server\Mailslot\[path]name

请将上述字串分为三段来看：\\server、\Mailslot和\[path]name。第一部分\\server对应于服务器的名字，我们要在上面创建邮槽，并在在上面运行服务器程序。第二部分 \Mailslot是一个“硬编码”的固定字串，用于告诉系统这个文件名从属于 MSFS。而第三部分\[path]name则允许应用程序独一无二地定义及标识一个邮槽名。其中，“path”代表路径，可指定多级目录。举个例子来说，对一个邮槽进行标识时，下面这些形式的名字都是合法的（注意 Mailslot不得变化，必须原文照输，亦即所谓的“硬编码”）：

\\Oreo\Mailslot\Mymailslot

\\Testserver\Mailslot\Cooldirectory\Funtest\Anothermailslot

\\.\Mailslot\Easymailslot

*\Mailslot\Myslot

服务器字串部分可表示成一个小数点（.）、一个星号（*）、一个域名或者一个真正的服务器名字。所谓“域”，其实就是一系列工作站和服务器的组合，它们共用一个相同的组名。本章稍后，在讲到一个简单的客户机的细节时，还会就邮槽名字作深入探讨。

由于邮槽要依赖Windows文件系统服务在网上来创建和传输数据，所以接口是“与协议无关”的。编制自己的应用程序时，便不必关心基层网络传送协议的细节，不必知道它在一个网络中如何在进程之间建立通信。邮槽通过一个网络与计算机进行远程通信时， Windows文件系统服务需要依赖Windows重定向器，使用“服务器消息块”（SMB）协议，将数据从客户机传给服务器。消息通常是通过“无连接”传输方式来发送的，但亦可强迫 Windows重定向器在Windows NT和Windows 2000上使用“面向连接”的传输方式。至于具体采用哪种方式，要由消息的长度决定。

3.1.2 消息的长度

邮槽通常用“数据报”（Datagram）在网络上传递消息。数据报实际是一些小数据包，只不过要以“无连接”的形式，通过网络进行传输。“无连接”意味着每个数据包在发给接收者之后，不要求对方提供包的收到确认信息。显然，这是一种“不可靠”的数据传输，因为无法保证消息肯定能正确送达。然而，通过无连接传输，我们确实可将消息从一个客户机广播给多个服务器。当然，上述情况也有例外。在 Windows NT和Windows 2000中，假如消息的长度超过424个字节，便会发生这种例外。

在Windows NT和Windows 2000中，若消息长度超过426个字节，便必须在一个SMB会话之上，通过一种“面向连接”的协议进行传输，而不再采用无连接的“数据报”形式。这样一来，在消息较大的情况下，便可保证它们的稳定、高效传输。然而，此时再也不能将一条消息从客户机广播给多个服务器。对于“面向连接”的传输来说，它必然是一种“一对一”通信：一个客户机对一个服务器！在不同的进程之间，使用“面向连接”的传输方式，往往可保证数据传输的可靠性。但在Windows NT和Windows 2000中，邮槽接口却不能保证一条消息肯定能够写入一个邮槽。举个例子来说，假如从客户机向服务器送出一条较大的消息，但指定的服务器在网络中并不存在，那么邮槽接口不会告诉你的客户机应用，数据到服务器的投递已经失败。由于Windows NT和Windows 2000要根据消息的长度，来更改它们的传输方法，所以假定一台机器运行的是Windows NT或Windows 2000，而另一台机器运行的是Windows 95

或者Windows 98，便会面临两种系统在沟通上的困难。

Windows 95和Windows 98只以“数据报”的形式来传送消息，无论消息长度如何。假如Windows 95或Windows 98客户机试图将一条长度超过 424字节的消息传给 Windows NT或Windows 2000服务器，Windows NT和Windows 2000便会只接受头424个字节，剩下的数据会被无情地“砍掉”。换言之，Windows NT和Windows 2000要求更大的消息通过一个“面向连接”的SMB会话进行传输。

消息从Windows NT或Windows 2000客户机传向Windows 95或Windows 98服务器时，也存在着类似的问题。请记住，Windows 95和Windows 98只通过数据报接收数据。由于对超过426个字节的消息来说，Windows NT和Windows 2000不会通过数据报来传输数据，所以在这种情况下，Windows 95和Windows 98不能从这些客户机收到消息。在表3-1中，我们为大家详细总结了这种消息长度上的限制。

注意 Windows CE已从表3-1中剔除出去了，因为它并未提供邮槽编程接口。另外要注意的是，由于Windows NT和Windows 2000重定向器的限制，长度为425或426字节的消息未在此表列出。

表3-1 邮槽消息的长度限制

传输方向	通过数据报进行“无连接”传输	进行“面向连接”的传输
Windows 95或Windows 98 ->Windows 95或Windows 98	消息长度高达64KB	不支持
Windows NT或Windows 2000 ->Windows NT或Windows 2000	消息长度必须为424字节或以下	消息必须大于426个字节
Windows NT或Windows 2000 ->Windows 95或Windows 98	消息长度必须为424字节或以下	不支持
Windows 95或Windows 98 ->Windows NT或Windows 2000	消息长度必须为424字节或以下；多余的字节会被截去	不支持

对Windows NT和Windows 2000这两种网络操作系统来说，还有另一项限制值得我们注意，因为它会对数据报数据的传输产生影响。Windows NT和Windows 2000重定向器不能收发长度正好为 425或426字节的一条数据报消息。例如，假定我们从 Windows NT或Windows 2000客户机向Windows 95、Windows 98、Windows NT或Windows 2000服务器发送这样的一条消息，Windows NT重定向器会在消息发给目标服务器之前，将其长度截短为424字节。

要想保证各种Windows平台之间能够完全正常地通信，强烈建议将消息长度限制在 424字节，或者更短。如果进行面向连接的传输，可考虑使用命名管道，而不是简单的邮槽。命名管道将在第4章讨论。

3.1.3 应用程序的编译

用Microsoft Visual C++编制一个邮槽客户机或服务器应用程序时，必须在程序文件中将Winbase.h这个包容文件包括在内。假如已经包含了 Windows.h（大多数应用程序都会这样），那么亦可将Winbase.h省去。此外，应用程序也要负责建立与 Kernel32.lib的链接，这通常需要用Visual C++链接器标志进行配置。

3.1.4 错误代码

开发邮槽客户机和服务器应用时，所有 Win32 API函数（CreateFile和CreateMailslot除外）在调用失败的情况下，都会返回 0值。CreateFile和CreateMailslot这两个 API却会返回INVALID_HANDLE_VALUE（无效句柄值）。若这些API函数调用失败，应用程序随即应调用GetLastError函数，来接收与此次失败有关的特殊信息。至于所有错误代码的一个完整列表，大家可参考本书附录 C提供的标准Windows错误代码列表，或者直接查询 Winerror.h这个头文件。

3.2 基本客户机／服务器

如前所述，邮槽建立了一个简单的客户机／服务器设计体系。在这个体系中，数据只能从客户机传到服务器，数据通信是单向进行的。服务器进程的职责是创建一个邮槽，而且是能从邮槽读取数据的唯一一个进程。邮槽客户机进程则负责打开邮槽的“ 实例 ”，该进程是能够向其中写入数据的唯一一种进程。

3.2.1 邮槽服务器的详情

若想实现一个邮槽，要求开发一个服务器应用，来负责邮槽的创建。下述步骤解释了如何编写一个基本的服务器应用：

1) 用CreateMailslot API函数创建一个邮槽句柄。

2) 调用ReadFile API函数，并使用现成的邮槽句柄，从任何客户机接收数据。

3) 用CloseHandle这个API函数，关闭邮槽句柄。

可以看出，要开发一个邮槽服务器程序，只需使用极少的 API调用。服务器进程是用CreateMailslot这个API调用来创建邮槽的。定义如下：

```
HANDLE CreateMailslot(
    LPCTSTR lpName,
    DWORD nMaxMessageSize,
    DWORD lReadTimeout,
    LPSECURITY_ATTRIBUTES lpSecurityAttributes
);
```

其中，第一个参数lpName指定邮槽的名字。名字的格式如下：

\\.\Mailslot\[path]name

要注意的是，服务器的名字用一个小数点来表示，亦即服务器就是本地机器。这样做是很有必要的，因为我们不能在远程计算机上创建邮槽。在 lpName参数中，名字必须以一种独一无二的形式表达。可将它设为一个独立的名字，也可以在它前面加上一个完整的目录路径。

nMaxMessageSize参数定义的是可写入邮槽的一条消息的最大长度（以字节为单位）。假如客户机写入的字节数多于nMaxMessageSize的设置，服务器便不会接收这条消息。若将它的值设为0，服务器便会接收任意长度的消息。

在一个邮槽上，读操作可以等待或不等待这两种模式进行，具体由 lReadTimeout参数决定。它以毫秒为单位，指定了读操作需要等候进入消息的时间。若将它的值设为MAILSLOT_WAIT_FOREVER，那么在进入的数据可以读取之前，读操作便会无限期地等待下去。若设为 0，读操作就会立即返回。本章稍后，还会就读操作的详情进行探讨。

lpSecurityAttributes参数决定了为邮槽施加的访问控制权限。在 Windows NT和Windows 2000中，这个参数只实现了一部分，所以同时还应指定一个 null（空） 参数。在邮槽上，唯一能够施加的安全措施是针对本地 I/O进行的 —— 客户机试图将服务器的名字设为小数点（ .），以打开一个邮槽。要想绕过这种安全机制，客户机可指定服务器的实际名字，而不是一个小数点（.），亦即相当于发出一个远程 I/O调用。在Windows NT和Windows 2000中，并未针对远程I/O而实现lpSecurityAttributes参数，因为假如每次发出一条消息时，都在客户机与服务器之间建立一个授权会话，那么效率会显得十分低下。因此，邮槽仅一部分符合标准文件系统采用的Windows NT和Windows 2000安全模型。结果便是，网络中的任何邮槽客户机都可将数据发给服务器。

用一个有效的句柄创建了邮槽之后，便可开始数据的实际读取。服务器是唯一能从邮槽读入数据的进程。服务器应使用 ReadFile这个Win32函数，来进行数据的读取。对 ReadFile的定义如下：

```
BOOL ReadFile(
    HANDLE hFile,
    LPVOID lpBuffer,
    DWORD nNumberOfBytesToRead,
    LPDWORD lpNumberOfBytesRead,
    LPOVERLAPPED lpOverlapped
);
```

CreateMailslot会返回一个句柄hFile。lpBuffer和nNumberOfBytesToRead参数决定了可从邮槽读入多少数据。特别值得注意的是，这个缓冲区的大小应该比来自 CreateMailslot API调用的nMaxMessageSize参数的设置大。此外，缓冲区应该大于邮槽上的进入消息；如果不够大，ReadFile调用便会失败，并返回一个 ERROR_INSUFFICIENT_BUFFER错误。lpNumberOfBytesRead参数用于在ReadFile操作完成后，报告读入的实际字节数量。

利用lpOverlapped参数，我们可以通过异步方式，进行数据的读取。该参数采用的是Win32重叠I/O机制，第4章还会对这个问题进行深入的讲解。在默认情况下， ReadFile操作会处于暂停状态，直到有数据可以读入为止。重叠 I/O只能在Windows NT和Windows 2000上完成；如果使用的操作系统是 Windows 95或Windows 98，那么应将该参数设为 NULL。在程序清单3-1中，我们进一步阐释了如何编写一个简单的邮槽服务器应用。

程序清单3-1 服务器邮槽示例

```
// Server1.cpp

#include <windows.h>
#include <stdio.h>

void main(void)
{
    HANDLE Mailslot;
    char buffer[256];
    DWORD NumberOfBytesRead;

    // Create the mailslot
    if ((Mailslot = CreateMailslot("\\\\.\\Mailslot\\Myslot", 0,
        MAILSLOT_WAIT_FOREVER, NULL)) == INVALID_HANDLE_VALUE)
    {
```

```
        printf("Failed to create a mailslot %d\n", GetLastError());
        return;
    }

    // Read data from the mailslot forever!
    while(ReadFile(Mailslot, buffer, 256, &NumberOfBytesRead,
        NULL) != 0)
    {
        printf("%.*s\n", NumberOfBytesRead, buffer);
    }
}
```

3.2.2 邮槽客户机的详情

要想实现一个客户机，需要开发一个应用程序，对一个现有的邮槽进行引用和写入。下述步骤解释了如何编写一个基本的客户机应用：

1) 使用CreateFile这个API函数，针对想向其传送数据的邮槽，打开指向它的一个引用句柄。

2) 调用WriteFile这个API函数，向邮槽写入数据。

3) 完成了数据的写入后，用CloseHandle这个API函数，关闭打开的邮槽句柄。

如前所述，采用一种“无连接”的形式，邮槽客户机同邮槽服务器通信。客户机打开指向邮槽的一个引用句柄时，实际并不建立同邮槽服务器的一个连接。要想对一个邮槽进行引用，需要使用CreateFile这个API调用。对它的定义如下：

```
HANDLE CreateFile(
    LPCTSTR lpFileName,
    DWORD dwDesiredAccess,
    DWORD dwShareMode,
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    DWORD dwCreationDisposition,
    DWORD dwFlagsAndAttributes,
    HANDLE hTemplateFile
);
```

lpFileName参数用于描述一个或多个邮槽，我们可用本章早些时候介绍的邮槽命名格式，向其写入数据。在表3-2中，我们更深入地总结了邮槽的命名规范。 dwDesiredAccess参数必须设为GENERIC_WRITE，因为客户机只能向服务器写入数据。 dwShareMode参数必须设为FILE_SHARE_READ，允许服务器在邮槽上打开和进行读操作。 lpSecurityAttributes参数对于邮槽不会有什么效果，应将其设为NULL。dwCreationDisposition标志应设为OPEN_EXISTING。若一台机器既是客户机，也是服务器，这一设置便显得尤其重要——如果服务器没有创建邮槽，对API函数CreateFile的调用便会失败。如果服务器在远程工作，那么。dwCreationDisposition

表3-2 邮槽名字类型

名字格式	说 明
\\.\mailslot\name	标定同一台机器上的一个本地邮槽
\\servername\mailslot\name	标定名为servername的一个远程邮槽服务器
\\domainname\mailslot\name	标定在指定的domain（域）内，使用特定name（名字）的所有邮槽
*\mailslot\name	标定系统主域内，标定特定name（名字）的所有邮槽

参数便没什么意义 dwFlagsAndAttributes参数应定义成 FILE_ATTRIBUTE_NORMAL。hTemplateFile参数应设为NULL。

成功创建一个句柄后，便可开始向邮槽写入数据。请记住，作为客户机，只能将数据写入邮槽。这可以用Win32函数WriteFile来做到，定义如下：

```
BOOL WriteFile(
    HANDLE hFile,
    LPCVOID lpBuffer,
    DWORD nNumberOfBytesToWrite,
    LPDWORD lpNumberOfBytesWritten,
    LPOVERLAPPED lpOverlapped
);
```

其中，hFile参数是由CreateFile返回的一个引用句柄。lpBuffer和nNumberOfBytesToWrite参数决定了有多少字节从客户机发向服务器。一条消息的最大长度为 64KB。如果当初是用一个域或星号（*）格式来创建邮槽句柄，那么在 Windows NT和Windows 2000中，消息的长度限制在424字节之内；而在Windows 95和Windows 98中，限制在64KB之内。如客户机试图发送的消息超出了这一长度限制， WriteFile函数会便失败，而且 GetLastError函数会返回ERROR_BAD_NETPATH错误。之所以会出现这一情况，是由于需要以广播数据报的形式，把消息发送给网络中的所有服务器。lpNumberOfBytesWritten参数返回的是当WriteFile操作完成后，传给服务器的实际字节数量。

通过lpOverlapped参数，我们可采用异步形式，将数据写入一个邮槽。由于邮槽最大的特点便是“无连接”的数据传输，所以 WriteFile函数不会在I/O调用的时候暂停等候。在客户机上，这个参数应设为 NULL。在程序清单3-2中，我们进一步阐述了如何编写一个简单的邮槽客户端应用。

程序清单3-2 客户机邮槽实例

```
// Client.cpp

#include <windows.h>
#include <stdio.h>

void main(int argc, char *argv[])
{
    HANDLE Mailslot;
    DWORD BytesWritten;
    CHAR ServerName[256];

    // Accept a command line argument for the server to send
    // a message to
    if (argc < 2)
    {
        printf("Usage: client <server name>\n");
        return;
    }
        sprintf(ServerName, "\\\\%s\\Mailslot\\Myslot", argv[1]);

        if ((Mailslot = CreateFile(ServerName, GENERIC_WRITE,
            FILE_SHARE_READ, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL,
            NULL)) == INVALID_HANDLE_VALUE)
        {
```

```
            printf("CreateFile failed with error %d\n", GetLastError());
            return;
        }

        if (WriteFile(Mailslot, "This is a test", 14, &BytesWritten,
            NULL) == 0)
        {
            printf("WriteFile failed with error %d\n", GetLastError());
            return;
        }

        printf("Wrote %d bytes\n", BytesWritten);

        CloseHandle(Mailslot);
    }
```

3.3 其他邮槽API

对邮槽服务器应用来说，它可使用另外两个 API函数同邮槽打交道： GetMailslotInfo和SetMailslotInfo。其中，一旦邮槽上有消息可以传递， GetMailslotInfo函数便可负责获取消息的长度信息。利用这个函数，程序可针对长度不定的进入消息，动态地调节其缓冲区。GetMailslotInfo函数亦可用来对进入数据进行“ 轮询 ”。对GetMailslotInfo函数的定义如下：

```
BOOL GetMailslotInfo(
    HANDLE hMailslot,
    LPDWORD lpMaxMessageSize,
    LPDWORD lpNextSize,
    LPDWORD lpMessageCount,
    LPDWORD lpReadTimeout
);
```

其中，hMailslot参数指定自CreateMailslot API调用返回的一个邮槽。lpMaxMessageSize参数设置可将多大的一条消息写入邮槽（以字节为单位）。lpNextSize参数则以字节为单位，指出下一条消息的长度。 GetMailslotInfo可能会返回一个MAILSLOT_NO_MESSAGE值，指出在邮槽之上，目前没有等待接收的消息。利用这个参数，服务器便可在邮槽上轮询（不断地查询）是否有进入的消息，防止应用程序在 ReadFile函数调用过程中“ 凝结 ”，傻乎乎地一直等候下去。然而，用这种方式对数据进行轮询并不是一种很好的编程习惯。因为应用程序会连续不停地消耗宝贵的 CPU资源，以检查进入的数据 —— 即使根本没有需要处理的消息。这样一来，便会降低系统的总体性能。如果想防止 ReadFile“ 凝结 ”或暂停执行，建立你使用Win32的重叠I/O。lpMessageCount参数指定一个缓冲区，用于接收等候读入的消息的总量。lpReadTimeout参数则指定了另一个缓冲区，它以毫秒为单位，返回了一次读操作最多能等候多久的时间，让一条消息写入邮槽。

SetMailslotInfo函数用于设置一个邮槽的超时值。超过这个时间，读操作便不再等候进入消息。因此，应用程序完全有能力将读操作从“ 凝结 ”状态转变成“ 非凝结 ”状态；或者相反。对SetMailslotInfo的定义如下：

```
BOOL SetMailslotInfo(
    HANDLE hMailslot,
    DWORD lReadTimeout
);
```

bMailslot参数指定一个自 CreateMailslot API调用返回的邮槽。 lReadTimeout参数以毫秒为单位，指定一次读操作等候一条消息写入邮槽的最长时间。若将其设为 0，在不存在消息的前提下，读操作便会立即返回；若设为 MAILSLOT_WAIT_FOREVER，读操作便会永远等待下去。

3.4 平台和性能问题

在Windows 95（含OSR2）和Windows 98（含第二版）这两种平台上，邮槽存在着一些必须引起我们警惕的限制：“ 8.3字符名字限制 ”、不能取消 “ 凝结 ” 的I/O请求；以及由于超时引起的内存废弃。

3.4.1 8.3字符名字限制

Windows 95和Windows 98不声不响地将邮槽名字限制在8.3字符格式的范围之内。这样一来，有时候就会造成Windows 95/98和Windows NT/2000之间的 “ 不和 ”。举个例子来说，假定我们用\\.\Mailslot\Mymailslot这个名字来创建或打开一个邮槽，Windows 95实际会以\\.\Mailslot\Mymailsl的形式，来创建及引用邮槽。尽管会发生名字被截短的现象， CreateMailslot和CreateFile函数执行仍会成功完成。然而，假如一条消息从 Windows 2000发给Windows 95 —— 或者相反，便无法收到这条消息，因为邮槽的名字并不相符。假如客户机和服务器均在Windows 95机器上运行，则不会出现这种问题 —— 名字在客户机和服务器上都会被截短。要想防范这种互不兼容的问题，一定要将邮槽的名字限制在 8个字符或者更短。

3.4.2 不能取消 “ 凝结 ” 的I/O请求

Windows 95和Windows 98在取消暂停或 “ 凝结 ” 的I/O请求时，也会出现问题。邮槽服务器用ReadFile函数来接收数据。假如用 MAILSLOT_WAIT_POREVER标志创建了一个邮槽，读请求便会一直等待下去，直到有数据可用为止。假如在一个ReadFile请求尚未完成的前提下，服务器应用突然中止运行，应用程序便会被永远地 “ 挂起 ”，或者 “ 凝结 ”。要想取消，唯一的办法就是重新启动 Windows。为解决这个问题，可让服务器在单独一个线程中，打开一个句柄，令其指向自己的邮槽。并发送数据，以终止处于暂停状态的读操作。在程序清单 3-3中，我们详细阐述了这一方案。

程序清单3-3 修订过的邮槽服务器

```
// Server2.cpp

#include <windows.h>
#include <stdio.h>
#include <conio.h>

BOOL StopProcessing;

DWORD WINAPI ServeMailslot(LPVOID lpParameter);
void SendMessageToMailslot(void);

void main(void) {
```

```
    DWORD ThreadId;
    HANDLE MailslotThread;

    StopProcessing = FALSE;
    MailslotThread = CreateThread(NULL, 0, ServeMailslot, NULL,
        0, &ThreadId);

    printf("Press a key to stop the server\n");
    _getch();

    // Mark the StopProcessing flag to TRUE so that when ReadFile
    // breaks, our server thread will end
    StopProcessing = TRUE;

    // Send a message to our mailslot to break the ReadFile call
    // in our server
    SendMessageToMailslot();

    // Wait for our server thread to complete
    if (WaitForSingleObject(MailslotThread, INFINITE) == WAIT_FAILED)
    {
        printf("WaitForSingleObject failed with error %d\n",
            GetLastError());
        return;
    }
}

//
// Function: ServeMailslot
//
// Description:
//     This function is the mailslot server worker function to
//     process all incoming mailslot I/O
//
DWORD WINAPI ServeMailslot(LPVOID lpParameter)
{
    char buffer[2048];
    DWORD NumberOfBytesRead;
    DWORD Ret;
    HANDLE Mailslot;

    if ((Mailslot = CreateMailslot("\\\\.\\mailslot\\myslot", 2048,
        MAILSLOT_WAIT_FOREVER, NULL)) == INVALID_HANDLE_VALUE)
    {
        printf("Failed to create a MailSlot %d\n", GetLastError());
        return 0;
    }

    while((Ret = ReadFile(Mailslot, buffer, 2048,
        &NumberOfBytesRead, NULL)) != 0)
    {
        if (StopProcessing)
            break;

        printf("Received %d bytes\n", NumberOfBytesRead);
```

```
        }

        CloseHandle(Mailslot);

        return 0;
    }

    //
    // Function: SendMessageToMailslot
    //
    // Description:
    //     The SendMessageToMailslot function is designed to send a
    //     simple message to our server so we can break the blocking
    //     ReadFile API call
    //
    void SendMessageToMailslot(void)
    {
        HANDLE Mailslot;
        DWORD BytesWritten;

        if ((Mailslot = CreateFile("\\\\.\\mailslot\\myslot",
            GENERIC_WRITE, FILE_SHARE_READ, NULL, OPEN_EXISTING,
            FILE_ATTRIBUTE_NORMAL, NULL)) == INVALID_HANDLE_VALUE)
        {
            printf("CreateFile failed with error %d\n", GetLastError());
            return;
        }

        if (WriteFile(Mailslot, "STOP", 4, &BytesWritten, NULL) == 0)
        {
            printf("WriteFile failed with error %d\n", GetLastError());
            return;
        }

        CloseHandle(Mailslot);
    }
```

3.4.3 超时引起的内存废弃

对于Windows 95和Windows 98来说，值得注意的最后一个问题便是内存空间的“废弃”，或者说内存空间产生了“漏洞”。在邮槽上使用超时设定时，便有可能出现这一现象。假如用CreateMailslot函数创建一个邮槽，同时将超时时间设为大于0的一个值，那么一旦超过这一时间，ReadFile函数便会造成内存空间的废弃，生成所谓的“内存漏洞”。同时，函数会返回FALSE。经过多次ReadFile函数调用之后，系统就会变得极不稳定，以后超时的ReadFile调用会开始返回TRUE。因此，系统不能再执行其他MS-DOS程序。为解决这个问题，请将超时值设为0或者MAILSLOT_WAIT_FOREVER。这样便可禁止应用程序使用超时机制，从而避免了实际产生的内存“漏洞”。

在微软知识库中，描述了下述问题，以及它们存在的限制。这个知识库是完全公开的，网上地址在http: //support.microsoft.com/support/search。这里只对相关的主题进行简单介绍：

- Q139715：ReadFile为邮槽返回错误的“错误代码”

假如服务器用CreateMailslot打开一个邮槽，指定一个超时，然后用 ReadFile接收数据，那么假如没有数据可用（可以读取），ReadFile便会失败。此时用GetLastError函数会返回一个错误代码5（拒绝访问）。

■ Q192276：GetMailslotInfo返回不正确 lpNextSize值

如果在Windows 95 OEM Service Release 2（OSR2）或Windows 98中调用API函数GetMailslotInfo，同时没有安装一个网络客户机组件，那么对 lpNextSize参数来说，便会通过它接收到一个不确切的值（通常有百万之大），或者接收到一个负数。如再次调用该函数，则往往会恢复正常，返回正确的值。

■ Q170581：在Win95中创建的邮槽只允许4093个字节的消息

如调用WriteFile这个API函数，将多于4093个字节的数据写入已在Windows 95工作站上创建好的一个邮槽，操作便会失败。

■ Q131493：CreateFile和邮槽的问题

在API函数CreateFile的用户文档说明中，错误描述了在打开一个邮槽的客户机那一端时，CreateFile可能返回的值。

3.5 小结

本章讲解了邮槽（Mailslot）网络编程技术。利用这一技术，应用程序可以在 Windows重定向器的帮助下，实现简单的单向进程间数据通信。对邮槽来说，它最有价值的一项功能便是通过网络，将一条消息广播给一台或多台计算机。然而，邮槽并未提供对数据可靠传输的保障。假如希望用 Windows重定向器实现“ 可靠 ”的数据通信，请考虑使用命名管道，这是下一章的主题！