第 32 卷 第 13 期 计 算 机 工 程 2006 年 7 月
Vol.32 № 13 Computer Engineering July 2006
·软件技术与数据库· 文章编号：1000—3428(2006)13—0055—03 文献标识码：A 中图分类号：TP311

基于 Windows 管道技术的系统集成方法

郭加树，刘 展，李 旺
（中国石油大学（华东）地球资源与信息学院，东营 257061）

摘 要：在软件系统设计的过程中，常把一个大的系统划分为许多子系统，每个子系统完成各个具体的功能，最后采用一种合适的方式把各个子系统有机的集成为一个完整的整体。该文在对 Windows 的管道机制分析的基础上，提出了一种通过管道技术进行系统集成的方法，通过这种方法，可以大大地降低系统设计的复杂度，提高系统的健壮性和可维护性。

关键词：管道技术；系统集成；Shell 程序

System Integrated Method Based on Windows Pipe Technology

GUO Jiashu, LIU Zhan, LI Wang
(School of Earth Resource & Information, Petroleum University of China (East China), Dongying 257061)

【Abstract】 In the process of software system design, the large system is divided to many subsystems, the subsystems implement its functions, at last subsystems are integrated. This paper provides a kind of method for system integration using windows pipe based on the analysis of Windows pipe system. The method can largely reduce the complexity of the system and improve the robust and maintainability of the system.

【Key words】 Pipe technology; System integration; Shell program

1 概述

在软件系统设计的过程中，常常采用软件工程[1]逐步求精的思想，把一个大的系统划分为许多子系统，每个子系统完成各个具体的功能，最后采用一种合适的方式把各个子系统有机地集成为一个完整的整体。例如常用的软件开发集成开发环境，就是把编辑环境、编译器、链接器等子系统集成为一个整体，如图 1 所示。

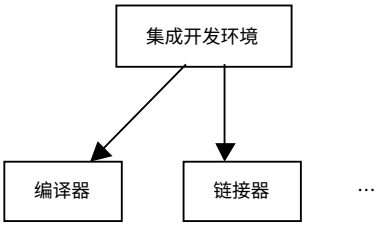

图 1 系统集成示意图

在这样的系统中，主控系统负责用户的交互操作，具有图形用户界面，子系统主要是对一些算法的封装，并不需要复杂的用户交互操作，常常以 Shell 程序的方式建立。Windows 环境下的 Shell 程序即为 DOS 命令行程序，例如 Visual Studio 集成开发环境中的 cl.exe 编译器，link.exe 链接器等都是标准的 Shell 程序。系统的这种开发方式有利于提高系统的健壮性与可维护性，降低系统的开发难度。但是采用何种方式进行系统集成，就成为一个必须解决的问题。针对此问题，本文采用 Windows 的管道技术较好地实现了系统的集成，并在基于 GIS 的 BTEX 数据仓库的开发研究中使用此技术，取得了良好的效果。

2 Windows 管道技术

2.1 模型简介

管道是 Windows 系统进行跨进程通信的一种系统工具，其实质为一 Windows 系统共享的内存区。Windows 系统提供两种类型的管道：(1)匿名管道（Anonymous Pipes）；(2)命名管道（Named Pipes）。匿名管道比命名管道系统资源消耗少，但提供的服务有限。

管道作为进程信息通信的一种手段，提供了两个端点，在每个端点都可进行读写操作；其概念模型如图 2 所示：①为进程 A 的写入端；②为进程 B 的读出端；③为进程 A 的读出端；④为进程 B 的写入端。

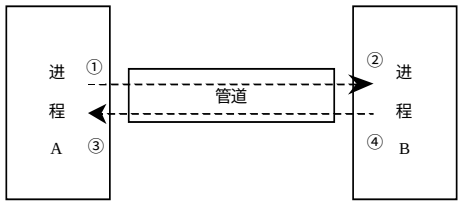

图 2 管道模型

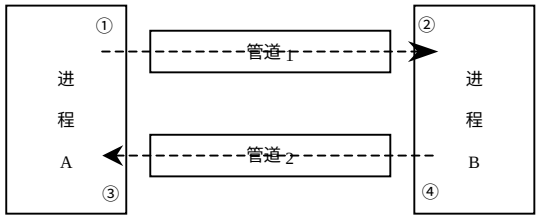

图 3 匿名管道双向通信模型

在利用管道进行跨进程的通信中，一个进程在管道的一端进行写操作，所写入的内容在管道的另一端可以被另一个进程通过读入端口所读取。从图 2 中可以看出，理论上可以利用管道实现进程间单向或双向通信。但是，在 Windows 系

基金项目：国家“863”计划基金资助项目（2002AA615160-2）
作者简介：郭加树(1973—)，男，博士生，主研方向：WebGIS，空间数据挖掘；刘 展，博士后、博导；李 旺，硕士生
收稿日期：2005-09-06 **E-mail：**guojs1973@163.com

统中，匿名管道为单向管道，即在管道的一端只能进行读或写操作，不能同时进行读写操作。如果要利用匿名管道实现双向通信，必须建立两个管道，如图 3 所示：①为进程 A 的写入端；②为进程 B 的读出端；③为进程 A 的读出端；④为进程 B 的写入端。

2.2 管道的使用方法

(1)管道的创建：匿名管道通过

```
BOOL CreatePipe(
  PHANDLE hReadPipe,   //管道的读入端句柄
  PHANDLE hWritePipe,   //管道的写出端句柄
  LPSECURITY_ATTRIBUTES lpPipeAttributes,
   //管道属性
  DWORD nSize)
```

函数创建，命名管道通过 HANDLE CreateNamedPipe()函数创建。

(2)管道的连接：匿名管道只能通过继承的方式从父进程获得管道的句柄，而命名管道还可以通过调用 CreateFile()或者 CallNamedPipe()函数获得管道连接句柄。

(3)管道的读写：管道数据的读写分为同步和异步两种方式，匿名管道仅仅支持同步读写，同步读写通过调用 ReadFile()和 WriteFile()函数实现。命名管道通过 ReadFileEx()和 WriteFileEx()函数支持异步读写操作。

(4)管道的关闭：通过调用 CloseHandle()函数关闭管道连接。

我们主要介绍匿名管道在系统集成中的应用以及相关的问题。

3 利用管道技术进行系统集成的关键问题

3.1 系统间管道的建立

主控系统负责管道的建立，并通过继承的方式传递给需要进行通信的子系统。在我们的研究中，子系统以 Shell 程序的方式建立。通常，Shell 程序中缺省状态下有 3 个 I/O 通道，标准输入（stdin），标准输出（stdout），标准错误输出（stderr）。标准输入通常情况下为键盘输入，标准输出与标准错误输出通常情况下为计算机屏幕。为了实现主控系统和子系统的通信，须把 Shell 子系统的 3 个 I/O 通道重定向为主控系统所建立的输入、输出管道。

为了实现 3 个标准 I/O 管道的重定向，在主控程序中建立新的匿名输入、输出管道，并在启动子系统时将子系统的标准 I/O 管道重定向为新建立的输入输出管道。这一过程通过调用 Windows 系统的 CreateProcessA()API 函数实现。其定义如下：

```
BOOL CreateProcess(
  LPCTSTR lpApplicationName, //“要执行的应用程序名称”
  LPTSTR lpCommandLine,   //  “命令行参数”
  …
  LPSTARTUPINFO lpStartupInfo,
);
```

其中的 lpStartupInfo 参数的定义如下：

```
typedef struct _STARTUPINFO {
…
DWORD dwFlags;
    HANDLE   hStdInput;
    HANDLE   hStdOutput;
    HANDLE   hStdError;
} STARTUPINFO, *LPSTARTUPINFO;
```

通过其中 3 个句柄参数 hStdInput、hStdOutput、hStdError 可以分别指定标准的 I/O 到主控系统所建立的匿名管道，在设置以上 3 个参数时，需要将 dwFlags 参数设为 STARTF_USESTDHANDLES，否则对 hStdInput、hStdOutput、hStdError 的设置无效。

3.2 主控系统数据的收发

在建立了与子系统的管道连接后，主控程序可以通过 WriteFile()向子系统发送数据，并通过 ReadFile()读取子系统所发送的数据。

3.3 子系统数据的收发

在 Shell 子系统中，因为系统的标准 I/O 已经被重定向为主控进程所建立的匿名通道，所以数据的收发通过标准的 I/O 操作即可实现。如通过 printf()向主控进程发送数据，通过 Scanf()从主控进程读取数据。

3.4 注意事项

在以 NT 为核心的 32 位 Windows 操作系统上，通过上述过程即可实现系统间的管道通信集成，但是在涉及到 Win95 兼容的 16 位、32 位混合系统时，调用 16 位的 Dos 程序可能会导致主控进程的挂起，这主要是因为 NT 核心和 95 核心对 Shell 的实现不同。在 Windows95 中，管道并不随着 16 位 Shell 程序的终止而关闭，因此，当主控进程读取已经关闭的 Shell 子系统的重定向 I/O 管道时，主控系统进程就会挂起。

这个问题可以通过用一个 Win32 程序作为代理来解决，即设计一个 Win32 类型的 Shell 程序，主控系统调用代理程序，代理程序启动相应的 Shell 程序，该 Shell 程序就会继承代理程序的重定向 I/O。随着 Shell 程序的结束，代理 Shell 程序也会结束，因为代理程序为 Win32 程序，随着代理程序的结束，相应的重定向 I/O 管道也会关闭。这样就避免了主控系统的挂起。

3.5 示例程序

3.5.1 主控端程序

```
SECURITY_ATTRIBUTES sa= {0};
STARTUPINFO     si= {0};
PROCESS_INFORMATION    pi= {0};
HANDLE    hPipeOutputRead, hPipeOutputWrite,
hPipeInputRead , hPipeInputWrite;
DWORD    dwNumberOfBytesRead = 0;
CHAR    szBuffer[256];
BOOL    fSuccess;

sa.nLength = sizeof(sa);
sa.bInheritHandle = TRUE;
sa.lpSecurityDescriptor = NULL;

 // 生成标准输出重定向管道.
CreatePipe(&hPipeOutputRead,   //管道的读入句柄
&hPipeOutputWrite, // 管道的写出句柄
&sa, // 安全属性设置
0 );
HANDLE    hPipeOutputReadDup;
//复制并关闭管道的写出端，使其不为子系统所用
fSuccess = DuplicateHandle(GetCurrentProcess(),
hPipeOutputRead,
GetCurrentProcess(),
```

```
    &hPipeOutputReadDup,
    0,
    FALSE,
    DUPLICATE_SAME_ACCESS);
    if (!fSuccess)
      ::AfxMessageBox("Duplicate failed");

    CloseHandle(hPipeOutputRead);

    HANDLE    hPipeInputWriteDup;
    //生成标准输入重定向管道
    CreatePipe(&hPipeInputRead,    //管道的读入句柄
    &hPipeInputWrite, //管道的写出句柄
    &sa, //安全属性设置
     0    );
    //复制并关闭管道的读入端，使其不为子系统所用
    fSuccess = DuplicateHandle(GetCurrentProcess(),
    hPipeInputWrite,
    GetCurrentProcess(),
    &hPipeInputWriteDup,
    0,
    FALSE,
    DUPLICATE_SAME_ACCESS);
    if (!fSuccess)
      ::AfxMessageBox("Duplicate failed");

    CloseHandle(hPipeInputWrite);
    //启动子系统进程
    si.cb = sizeof(si);
    si.dwFlags= STARTF_USESHOWWINDOW |
STARTF_USESTDHANDLES;    //必须设置
//STARTF_USESTDHANDLES 才能使用输入输出重定向功能
    si.wShowWindow = SW_HIDE;    //子系统运行时不显示
    si.hStdInput= hPipeInputRead;    //设置 Shell 子系统的标准输入
//为句柄 hPipeInputRead 所指的管道的输入端
    si.hStdOutput= hPipeOutputWrite; //设置 Shell 子系统的标准输出
//为句柄 hPipeOutputWrite 所指的管道的输出端
    si.hStdError= hPipeOutputWrite; //设置 Shell 子系统的错误输出
//为句柄 hPipeOutputWrite 所指的管道的输出端

    CreateProcess (NULL, "testpoint.exe", //子系统程序
      …
    &si, &pi);

    //关闭重定向管道的主控系统不能操作的一端
    CloseHandle(hPipeOutputWrite);
    CloseHandle(hPipeInputRead);

    //从重定向管道 hPipeInputWrite 发送数据
    DWORD    dwWritten = 0;
    char *szBuffer;

    szBuffer="hello, world\n-10.2\n12\n "; //通过管道传送数据
    WriteFile(hPipeInputWriteDup,
    szBuffer,
    strlen(szBuffer),
    &dwWritten,
    NULL);

    //从重定向管道 hPipeOutputRead 读入数据
      while(TRUE)
    {  if(!ReadFile(hPipeOutputReadDup,szBuffer,256,&dwNumberOf
BytesRead, NULL)||dwNumberOfBytesRead==0 ) break;
    else
    { szBuffer[dwNumberOfBytesRead] = 0;
      CString strOutPut(szBuffer);
      GetEditCtrl().ReplaceSel(strOutPut);不过}
    }
      // 等待子系统进程结束
      WaitForSingleObject (pi.hProcess, INFINITE);
      CloseHandle (pi.hProcess);
      CloseHandle (hPipeOutputReadDup);
      CloseHandle (hPipeInputWriteDup);
```

3.5.2 子系统的设计

子系统的实现比较简单，利用子系统的标准输入输出即可，在此不再举例。

3.5.3 Win32 代理

```
    STARTUPINFO    si= {0};
    PROCESS_INFORMATION pi= {0};

    // 重定向子进程的标准管道
    si.cb = sizeof(si);
    si.dwFlags= STARTF_USESTDHANDLES;
    si.hStdInput= GetStdHandle (STD_INPUT_HANDLE);
    si.hStdOutput = GetStdHandle (STD_OUTPUT_HANDLE);
    si.hStdError= GetStdHandle (STD_ERROR_HANDLE);

    BOOL bRet = CreateProcess(NULL,argv[1], NULL, NULL, TRUE,
0, NULL, NULL,&si,&pi);
    if (bRet)
    { WaitForSingleObject (pi.hProcess, INFINITE);
      CloseHandle (pi.hProcess);
      CloseHandle (pi.hThread);
    }
```

4 结束语

利用 Windows 的管道机制，可以通过设计全新的主控程序对已有的 Shell 程序系统集成，充分地利用已有资源，减少重复开发；同时，在开发新的系统模块时，也可以利用这种方法对系统进行模块化划分，实现系统的高内聚、低耦合；可以大大地降低系统设计的复杂度，提高系统的健壮性和可维护性。

参考文献

1 张海藩. 软件工程导论[M]. 北京: 清华大学出版社, 2001-01.

2 Williams M. Programming Microsoft Windows 2000 Unleashed [M]. USA: Sams Publishing, 1999.

3 Forta. 杜大鹏译. Windows2000 开发人员指南[M]. 北京: 中国水利水电出版社, 2001-04.