《操作系统原理》
实验指导

进程通信

黄伯虎
西安电子科技大学

1. 实验内容

   进程通信：设计并实现多个进程，使得他们之间可以相互发送数据。

2. 要求

   基本要求：必须是进程间的通信；至少实现 1 中进程间通信方式；至少有 2 个进程。

   扩展要求(选作)：实现多个(>2)进程的相互通信；实现多种进程间通信方式。

   平台和工具(原则不限，推荐如下)：Win32 平台；VC++6.0/.net

3. 结果显示：

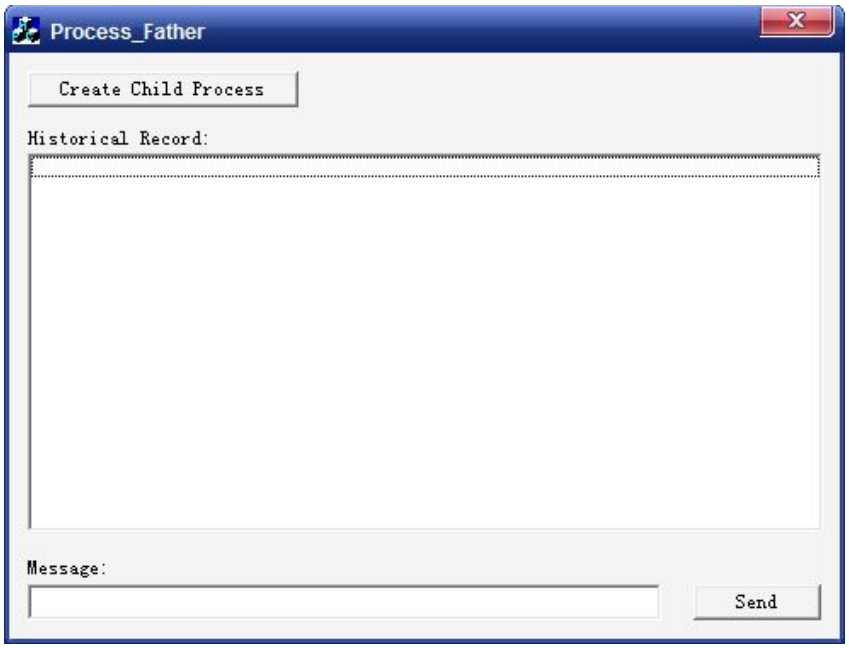
   可以参考 QQ 等聊天工具界面，实现两个进程相互发送字符信息。

   界面不用太花哨，能说明问题即可。

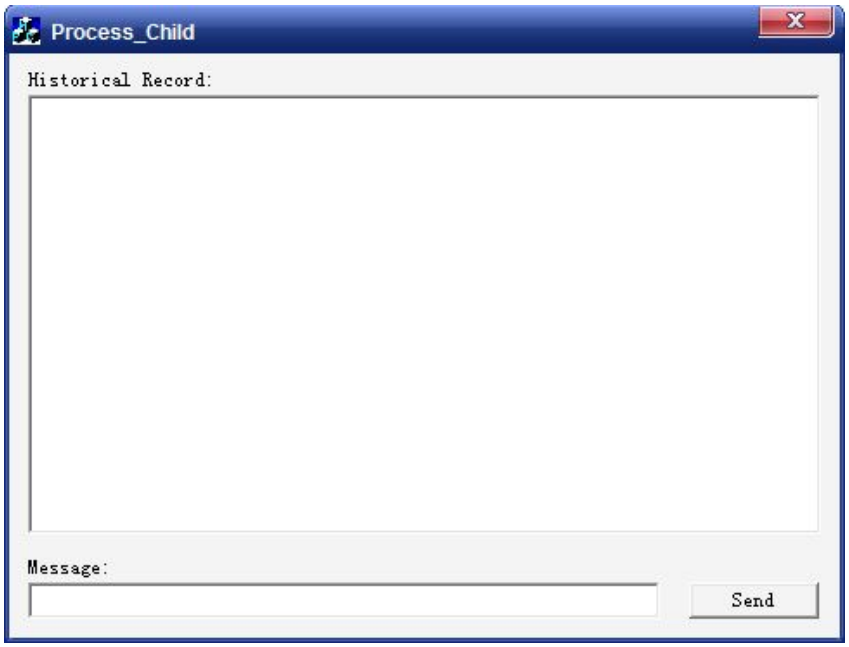
4. 示例

（1）程序界面

父进程：



子进程：



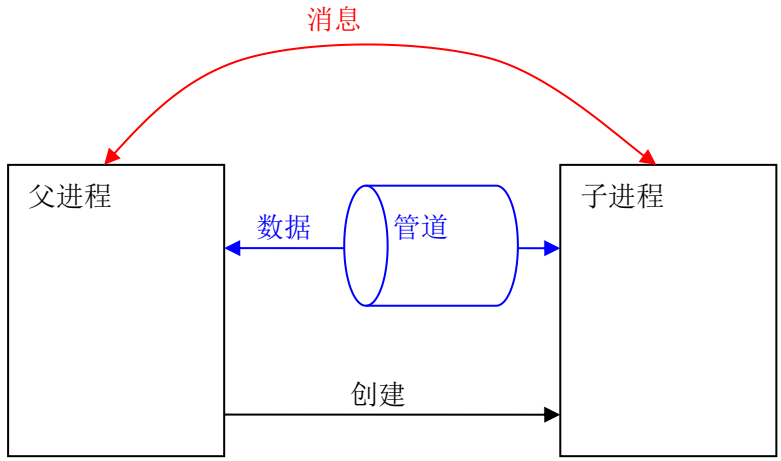
子进程由父进程创建，并进行通信。

进程的通信方式有很多种：如内存映射文件，命名管道，消息，剪贴板，甚至可以是磁盘文件等，单个通信方式的实现较为简单，建议大家多尝试几种。

- 管道方式实现（示例）

原理：

使用管道实现进程之间的通信，为了能够实时接受，相互发送自定义消息。程序的基本框架如图：



1） 父进程如何创建子进程

API 函数：**CreateProcess**

功能：Creates a new process and its primary thread. The new process runs in the security context of the calling process.

语法：

```
BOOL WINAPI CreateProcess(
  __in          LPCTSTR lpApplicationName,
  __in_out      LPTSTR lpCommandLine,
  __in          LPSECURITY_ATTRIBUTES lpProcessAttributes,
  __in          LPSECURITY_ATTRIBUTES lpThreadAttributes,
  __in          BOOL bInheritHandles,
  __in          DWORD dwCreationFlags,
  __in          LPVOID lpEnvironment,
  __in          LPCTSTR lpCurrentDirectory,
  __in          LPSTARTUPINFO lpStartupInfo,
  __out         LPPROCESS_INFORMATION lpProcessInformation
);
```

参数说明：

lpApplicationName

The name of the module to be executed. This module can be a Windows-based application. It can be some other type of module (for example, MS-DOS or OS/2) if the appropriate subsystem is available on the local computer. …

lpCommandLine

The command line to be executed. The maximum length of this string is 32K characters. If

lpApplicationName is NULL, the module name portion of lpCommandLine is limited to MAX_PATH characters….

lpProcessAttributes

A pointer to a SECURITY_ATTRIBUTES structure that determines whether the returned handle can be inherited by child processes. If lpProcessAttributes is NULL, the handle cannot be inherited.

lpThreadAttributes

A pointer to a SECURITY_ATTRIBUTES structure that determines whether the returned handle can be inherited by child processes. If lpThreadAttributes is NULL, the handle cannot be inherited.

bInheritHandles

If this parameter TRUE, each inheritable handle in the calling process is inherited by the new process. If the parameter is FALSE, the handles are not inherited. Note that inherited handles have the same value and access rights as the original handles.

dwCreationFlags

The flags that control the priority class and the creation of the process. For a list of values, see Process Creation Flags. …

lpEnvironment

A pointer to the environment block for the new process. If this parameter is NULL, the new process uses the environment of the calling process.

lpCurrentDirectory

The full path to the current directory for the process. The string can also specify a UNC path. If this parameter is NULL, the new process will have the same current drive and directory as the calling process. (This feature is provided primarily for shells that need to start an application and specify its initial drive and working directory.)

lpStartupInfo

A pointer to a STARTUPINFO or STARTUPINFOEX structure.

To set extended attributes, use a STARTUPINFOEX structure and specify EXTENDED_STARTUPINFO_PRESENT in the dwCreationFlags parameter.

lpProcessInformation

A pointer to a PROCESS_INFORMATION structure that receives identification information about the new process.

Return Value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

参考代码：

```
/***函数功能:下面的代码用于生成一个子进程，为了使得子进程能够继承管道的相关信息，
    首先生成管道，然后再生成子进程。***/

/*----下面代码用于生成一个管道，
    但是首先需要对CreatePipe函数中的相关参数进行初始化，详见MSDN帮助----*/
/*初始化lpPipeAttributes参数，这个参数决定返回的管道句柄是否可以被子进程继承*/
SECURITY_ATTRIBUTES sa; //SECURITY_ATTRIBUTES是VC提供的一个结构体数据类型
sa.nLength=sizeof(SECURITY_ATTRIBUTES); //定义结构体的size
sa.lpSecurityDescriptor=NULL; //定义安全描述符，NULL为默认值
sa.bInheritHandle=TRUE; //决定返回句柄是否可以被子进程继承，TRUE为可以

/*创建匿名管道*/
::CreatePipe(&hPipeRead, &hPipeWrite, &sa, 0);
/*----管道创建完毕----*/

/*----下面的代码用于创建子进程，
    同样首先需要初始化CreateProcess函数中的相关参数----*/
/*初始化lpStartupInfo参数，需要定义为STARTUPINFO类型*/
STARTUPINFO StartupInfo; //STARTUPINFO为VC定义的一个结构体数据类型，
memset(&StartupInfo, 0, sizeof(STARTUPINFO)); //下面6行代码用于StartupInfo的初始化，
                                              //各个field的功能详见MSDN
StartupInfo.cb = sizeof(STARTUPINFO);
StartupInfo.dwFlags=STARTF_USESTDHANDLES;
StartupInfo.hStdInput=hPipeRead;
StartupInfo.hStdOutput=hPipeWrite;
StartupInfo.hStdError=GetStdHandle(STD_ERROR_HANDLE);

/*定义lpProcessInformation参数，需要定义为PROCESS_INFORMATION 类型*/
PROCESS_INFORMATION ProcessInfo; //PROCESS_INFORMATION为VC的结构体类型，用于保存生成进程的信息

/*生成子进程，第1个参数为路径，第5个参数需要设置为TRUE(继承句柄)，第9，10个参数上面已经定义，其他的使用NULL*/
::CreateProcess("Process_Child.exe", NULL, NULL, NULL, TRUE, 0, NULL, NULL, &StartupInfo,
    &ProcessInfo); //注意路径现在给的是相对路径，你自己进行编译时需要修改路径
/*----子进程创建完毕----*/

/*将创建子进程按钮变灰，以便只能创建一个子进程*/
m_BT_CreateChildProcess.EnableWindow(FALSE); //m_BT_CreateChildProcess为按钮对象，
                                             //查看View->ClassWizard获得详情
```

2）进程通信

API 函数：

✍ **创建管道：CreatePipe**

功能：Creates an anonymous pipe, and returns handles to the read and write ends of the pipe.

语法：BOOL WINAPI CreatePipe(

__out	PHANDLE hReadPipe,
__out	PHANDLE hWritePipe,
__in	LPSECURITY_ATTRIBUTES lpPipeAttributes,
__in	DWORD nSize

);

参数：

hReadPipe

A pointer to a variable that receives the read handle for the pipe.

hWritePipe

A pointer to a variable that receives the write handle for the pipe.

lpPipeAttributes

A pointer to a SECURITY_ATTRIBUTES structure that determines whether the returned handle can be inherited by child processes. If lpPipeAttributes is NULL, the handle cannot be inherited.

nSize

The size of the buffer for the pipe, in bytes. The size is only a suggestion; the system uses the

value to calculate an appropriate buffering mechanism. If this parameter is zero, the system uses the default buffer size.

Return Value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

CreatePipe creates the pipe, assigning the specified pipe size to the storage buffer. CreatePipe also creates handles that the process uses to read from and write to the buffer in subsequent calls to the ReadFile and WriteFile functions….

✍ 写管道(文件)：WriteFile

功能：Writes data to the specified file at the position specified by the file pointer. This function is designed for both synchronous and asynchronous operation.

语法：BOOL WINAPI WriteFile(

```
  __in        HANDLE hFile,
  __in        LPCVOID lpBuffer,
  __in        DWORD nNumberOfBytesToWrite,
  __out       LPDWORD lpNumberOfBytesWritten,
  __in        LPOVERLAPPED lpOverlapped
);
```

参数

hFile

A handle to the file. The file handle must have been created with the GENERIC_WRITE access right. For more information, see File Security and Access Rights.

For asynchronous write operations, hFile can be any handle opened with the FILE_FLAG_OVERLAPPED flag by the CreateFile function, or a socket handle returned by the socket or accept function.

lpBuffer

A pointer to the buffer containing the data to be written to the file.

nNumberOfBytesToWrite

The number of bytes to be written to the file.

A value of zero specifies a null write operation. The behavior of a null write operation depends on the underlying file system.

To truncate or extend a file, use the SetEndOfFile function.

Named pipe write operations across a network are limited to 65,535 bytes.

lpNumberOfBytesWritten

A pointer to the variable that receives the number of bytes written. WriteFile sets this value to zero before doing any work or error checking. …

lpOverlapped

A pointer to an OVERLAPPED structure. This structure is required if hFile was opened with FILE_FLAG_OVERLAPPED.

If hFile was opened with FILE_FLAG_OVERLAPPED, the lpOverlapped parameter must not be NULL. It must point to a valid OVERLAPPED structure. If hFile was opened with FILE_FLAG_OVERLAPPED and lpOverlapped is NULL, the function can incorrectly report that the write operation is complete….

Return Value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

✍ 读管道(文件)：ReadFile

功能：The ReadFile function reads data from a file, starting at the position indicated by the file pointer. After the read operation has been completed, the file pointer is adjusted by the number of bytes actually read, unless the file handle is created with the overlapped attribute. If the file handle is created for overlapped input and output (I/O), the application must adjust the position of the file pointer after the read operation.

语法：BOOL ReadFile(

```
  HANDLE hFile,                    // handle of file to read
  LPVOID lpBuffer,                 // pointer to buffer that receives data
  DWORD nNumberOfBytesToRead,      // number of bytes to read
  LPDWORD lpNumberOfBytesRead,     // pointer to number of bytes read
  LPOVERLAPPED lpOverlapped        // pointer to structure for data
);
```

参数：

hFile

Handle to the file to be read. The file handle must have been created with GENERIC_READ access to the file.

lpBuffer

Pointer to the buffer that receives the data read from the file.

nNumberOfBytesToRead

Number of bytes to be read from the file.

lpNumberOfBytesRead

Pointer to the number of bytes read. ReadFile sets this value to zero before doing any work or error checking. If this parameter is zero when ReadFile returns TRUE on a named pipe, the other end of the message-mode pipe called the WriteFile function with nNumberOfBytesToWrite set to zero.

lpOverlapped

Pointer to an OVERLAPPED structure. This structure is required if hFile was created with FILE_FLAG_OVERLAPPED.

If hFile was opened with FILE_FLAG_OVERLAPPED, the lpOverlapped parameter must not be NULL. It must point to a valid OVERLAPPED structure. If hFile was created with FILE_FLAG_OVERLAPPED and lpOverlapped is NULL, the function can incorrectly report that the read operation is complete.

Return Values

The ReadFile function returns when one of the following is true: a write operation completes on the write end of the pipe, the number of bytes requested has been read, or an error occurs.
If the function succeeds, the return value is nonzero.

参考代码：

```
/*----下面代码用于向管道写数据----*/
/*获取EDIT控件中的字符串*/
CString str;
m_EDIT_Message.GetWindowText(str); //m_EDIT_Message为CEdit对象，查看View->ClassWizard获得详情
                                   //GetWindowText函数用于从EDIT控件中取得数据

/*向管道中写入数据*/
DWORD dwWritten;
WriteFile(hPipeWrite, str, 40, &dwWritten, NULL); //WriteFile为写管道函数，
                                                  //各个参数信息详见MSDN
/*----向管道写数据完毕----*/

/*----下面代码用于向子进程发送消息----*/
CString strWinName = "Process_Child"; //注意这里的值应当是子进程窗体的Title
CWnd *pWnd = CWnd::FindWindow(NULL, strWinName); //得到子进程主窗体句柄
if(pWnd) //如果子进程窗体存在
{
    pWnd->SendMessage(WM_FATHER_SEND, 0, 0); //发送自定义消息WM_FATHER_SEND给该窗体
                                             //WM_FATHER_SEND定义见头文件
    /*向LISTBOX控件（记录框）中写数据*/
    str = "Father: " + str;
    m_LISTBOX_Record.InsertString(-1, str); //m_LISTBOX_Record为CLISTBOX对象，
                                            //查看View->ClassWizard获得详情

    /*清空Edit控件内容，以方便下次输入*/
    m_EDIT_Message.SetWindowText("");
}
else //如果子进程窗体不存在
{
    MessageBox("没有发现子进程", "错误");
}
```

3） 如何通知对方有信息传来

实现方法：消息机制

自定义消息建立过程：需要以下 4 步

（1）在头文件中声明一个自定义消息，如下：

```
//自定义消息
#define WM_FATHER_SEND WM_USER+100
#define WM_CHILD_SEND WM_USER+101
```

（2）在头文件中声明消息处理函数，注意添加位置

```
    // Generated message map functions
    //{{AFX_MSG(CProcess_ChildDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg void OnReceiveMsg(WPARAM wParam, LPARAM lParam); //声明消息处理函数
    afx_msg void OnBUTTONSend();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
```

（3）在 CPP 文件中建立消息映射，如下，同样注意位置

```
BEGIN_MESSAGE_MAP(CProcess_ChildDlg, CDialog)
    //{{AFX_MSG_MAP(CProcess_ChildDlg)
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_BUTTON_Send, OnBUTTONSend)
    //}}AFX_MSG_MAP

    ON_MESSAGE(WM_FATHER_SEND, OnReceiveMsg) //消息映射

END_MESSAGE_MAP()
```

（4）在 CPP 文件中定义消息处理函数：

```
void CProcess_ChildDlg::OnReceiveMsg(WPARAM wParam, LPARAM lParam)
{
    /***函数功能：消息处理函数***/
    DWORD dwRead;
    char s[40];
    HANDLE hPipeRead; //声明读句柄

    /*读句柄*/
    hPipeRead=GetStdHandle(STD_INPUT_HANDLE); //获得管道读句柄
    ReadFile(hPipeRead, s, 40, &dwRead, NULL); //读数据

    /*向记录框中写数据*/
    char str[60] = "Father: ";
    strcat(str, s);
    m_LISTBOX_Record.InsertString(-1, str);
}
```