A Basic Study on Process control and communication

By

Shajnin akther jarin

Wuhan Institute of Technology

School of Computer Science and Engineering

Abstract

Communication is a fundamental aspect of computing systems, playing a crucial role in facilitating interaction among different components. In parallel and distributed computing, two prominent models of communication are widely utilized: Message Communication and Shared Memory Communication. This report explores these communication models, delving into their concepts, advantages, disadvantages, and real-world applications. Understanding these communication paradigms is essential for designing efficient and scalable computing systems.

Introduction

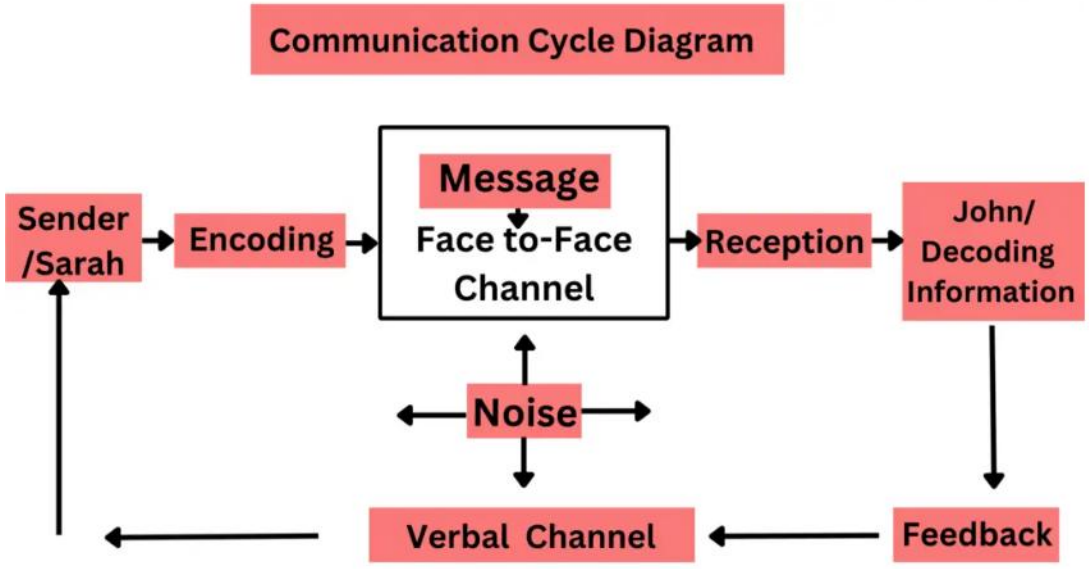
Message communication and shared memory communication are fundamental paradigms in computer science, facilitating interaction between processes in a system. Message communication involves processes exchanging data through messages, offering a structured way to coordinate tasks. On the other hand, shared memory communication allows processes to access a common memory space, enabling direct data sharing. Both methods play crucial roles in distributed computing and parallel processing, influencing the design and performance of systems in diverse applications. Understanding these communication models is essential for building efficient and scalable software solutions.

Masseage communication

Message communication is the exchange of information between individuals or systems through various mediums, such as text, voice, or multimedia. It serves as a crucial means of conveying ideas, thoughts, and data, fostering effective interaction and understanding. Whether through traditional channels like written letters or modern platforms like emails and instant messaging, message communication facilitates seamless connection and collaboration in personal, professional, and technological spheres, playing a pivotal role in the exchange of information in today's interconnected world.

Components of Message Communication

The message is the informational content that the sender aims to communicate to the receiver. It serves as the core of the communication process, carrying the purpose and meaning that the sender wants to communicate. The role of the message is to effectively transfer thoughts, concepts, instructions, or emotions from the sender to the receiver, aimi ng to create mutual understanding and facilitate a specific response or action.



Advantages of Message Communication

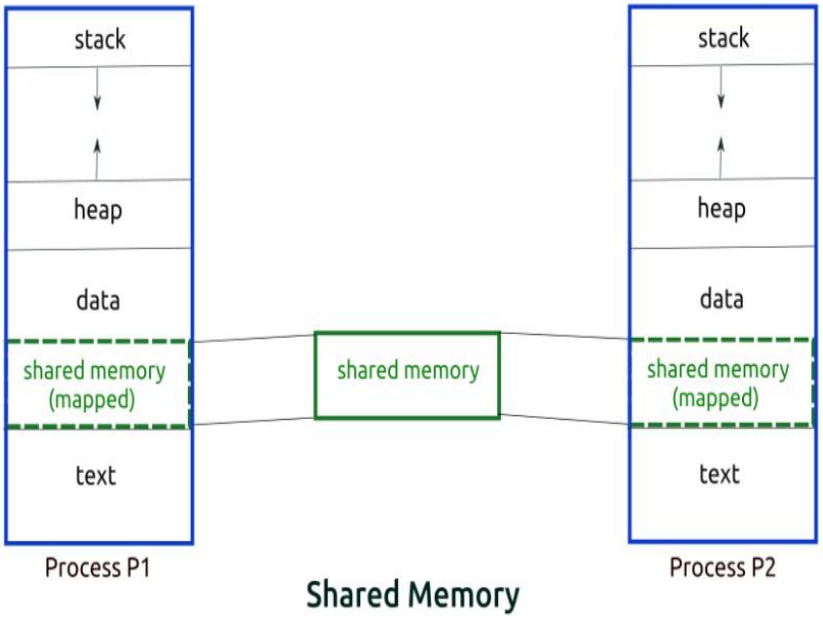
Message communication offers efficiency, allowing instant information exchange across vast distances. It enables asynchronous communication, accommodating different time zones and schedules. Messages can be stored for future reference, aiding documentation and record-keeping. Additionally, written messages provide clarity, reducing the likelihood of misinterpretation. Overall, message communication enhances convenience, accessibility, and accuracy in conveying information.

Disadvantages of Message Communication

Message communication has its drawbacks. It lacks non-verbal cues, leading to potential misunderstandings. Tone and emotions are often lost, making it challenging to convey sentiments accurately. Delays in response time can hinder real-time interaction, affecting swift decision-making. Additionally, the written format may lack the personal touch of face-to-face communication, impacting relationship building. Misinterpretations and the absence of immediate feedback are inherent challenges, compromising the effectiveness of conveying complex or sensitive information.

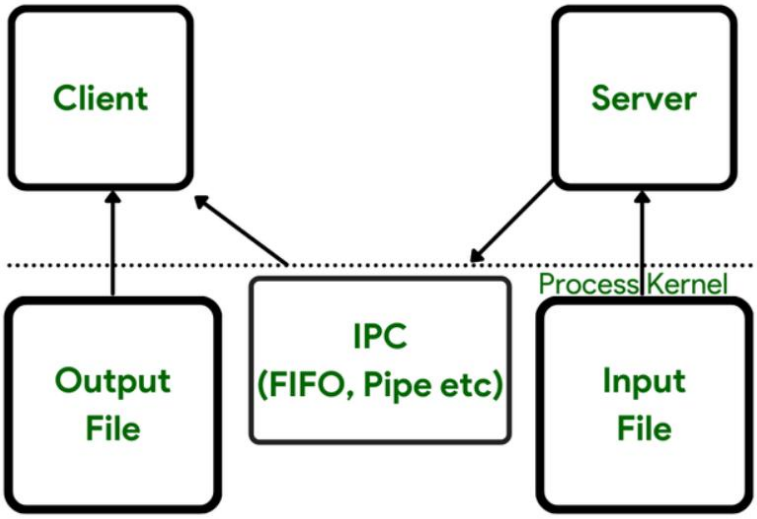
Shared Memory communication

Shared memory communication is a method where multiple processes or threads access a common, shared area of memory to exchange information. This shared memory region allows faster communication compared to other interprocess communication methods. Processes can read from and write to this shared space, enabling efficient data exchange without the need for complex message passing. However, careful synchronization mechanisms are crucial to prevent conflicts and ensure data integrity. Shared memory is widely used in parallel computing and multiprocessing systems, enhancing performance by enabling direct and rapid data sharing among concurrently running components.



Components of Shared Memory Communication

Shared memory communication involves processes or threads accessing a common region of memory for data exchange. Key components include a shared memory segment, accessible by multiple entities, synchronization mechanisms like locks or semaphores to manage concurrent access, and data structures facilitating communication. Processes can read and write to the shared memory, enabling efficient inter-process communication. However, proper synchronization is crucial to prevent data inconsistencies. Shared memory is a fast and effective method for communication between entities running concurrently on a computer system.



Advantages of Shared Memory Communication

1.Fast and productive: Shared memory is one of the quickest ways for between process correspondence as it empowers cycles to share information straightforwardly without including the working framework piece.

2.Low above: Shared memory has a low above contrasted with different types of between process correspondence as it requires no serialization or marshaling of information.

3.Memory-proficient: With shared memory, cycles can share a lot of information without consuming an excess of memory since each interaction just has to designate memory for its own information structures.

4.Easy synchronization: Divided memory considers simple synchronization among processes as it gives worked in systems like semaphores and mutexes to control admittance to shared assets.

Disadvantages of Shared Memory Communication

1.Complexity: Shared memory is more mind boggling than different types of between process correspondence as it requires cautious coordination between cycles to stay away from race conditions and halts.

2.Security dangers: Since shared memory is open to different cycles, there is a gamble of one interaction getting to or changing information having a place with another interaction. This can be moderated by utilizing access control instruments like read-just consents, however it adds an additional layer of intricacy.

3.Limited versatility: Shared memory isn’t effectively convenient between various working

frameworks and models since it relies upon the fundamental memory the board components.

4.Debugging troubles: Investigating shared memory can be trying as it requires specific apparatuses and strategies to recognize and determine issues connected with simultaneousness and synchronization.

Implementation and Result analysis

Create process

This is the main application source file that contains the application class CCntrlOtherPrcssApp using Microsoft Foundation Classes (MFC) for Windows GUI development. The InitInstance function initializes the application, creates a dialog (CCntrlOtherPrcssDlg), and runs the main message loop. The dialog is displayed, and based on the user's response (OK or Cancel), corresponding actions are taken. The program structure adheres to MFC conventions, including message maps and initialization routines. It appears to be a simple MFC application with a dialog-based user interface, handling user interactions through the Windows message loop.

CntrlOtherPrcss.cpp

```
#include "stdafx.h"
#include "CntrlOtherPrcss.h"
#include "CntrlOtherPrcssDlg.h"
```

(Header Files Inclusion:This is the header file for the application. "stdafx.h" is typically a precompiled header file that includes frequently used headers to speed up compilation.)

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

(Debug Configuration:The code includes some directives related to debugging. The #ifdef _DEBUG block defines some macros for debugging and sets a file name for debugging purposes.)

```
// CCntrlOtherPrcssApp

BEGIN_MESSAGE_MAP(CCntrlOtherPrcssApp, CWinApp)
    //{{AFX_MSG_MAP(CCntrlOtherPrcssApp)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //       DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    ON_COMMAND(ID_HELP, CWinApp::OnHelp)
```

```
END_MESSAGE_MAP()
```
(Message Map:This defines the message map for the application. Message maps are used in MFC to handle messages and events)
```
// CCntrlOtherPrcssApp construction

CCntrlOtherPrcssApp::CCntrlOtherPrcssApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}
```
(Constructor:The constructor for the CCntrlOtherPrcssApp class. It doesn't contain much code but suggests that significant initialization should be done in the InitInstance function.)
```
// The one and only CCntrlOtherPrcssApp object

CCntrlOtherPrcssApp theApp;

// CCntrlOtherPrcssApp initialization
```
(Main Application Object Creation:An instance of the CCntrlOtherPrcssApp class named theApp is created. This is the main application object.)
```
BOOL CCntrlOtherPrcssApp::InitInstance()
{
    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the size
    //  of your final executable, you should remove from the following
    //  the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();            // Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic();// Call this when linking to MFC statically
#endif
```
(Initialization:The InitInstance method is called during application initialization. It enables the control container and performs standard initialization, such as enabling 3D controls.)
```

    CCntrlOtherPrcssDlg dlg;
    m_pMainWnd = &dlg;
    int nResponse = dlg.DoModal();
```
(Dialog Creation and Display:An instance of the main dialog (CCntrlOtherPrcssDlg) is created. The main window pointer is set, and the modal dialog is displayed. The DoModal method will return when the dialog is closed.)
```
    if (nResponse == IDOK)
    {
```

```
        // TODO: Place code here to handle when the dialog is
        //  dismissed with OK
    }
    else if (nResponse == IDCANCEL)
    {
        // TODO: Place code here to handle when the dialog is
        //  dismissed with Cancel
    }
```
(Handling Dialog Closure:Code blocks are provided to handle responses when the dialog is closed with OK or Cancel. Currently, there are comments indicating that code should be added to handle these cases.)
```
    // Since the dialog has been closed, return FALSE so that we exit the
    //  application, rather than start the application's message pump.
    return FALSE;
}
```

CntrlOtherPrcss.rc: This code appears to be a resource script written in C++ using Microsoft Visual Studio's resource script format (.rc file). Resource scripts are used to define and manage resources such as icons, dialogs, menus,bitmaps, and cursors that are stored in the RES subdirectory version information for Windows applications. The code also contains directives for excluding certain resources based on build configurations and platforms. Overall, the code is essential for defining the visual and version-related aspects of the application.

```
#include "resource.h"
#define APSTUDIO_READONLY_SYMBOLS
#include "afxres.h"
#undef APSTUDIO_READONLY_SYMBOLS
```
(Resource Header Inclusions:These lines include necessary headers for resource handling in a Windows application.)

```
#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_CHS)
#ifdef _WIN32
LANGUAGE LANG_CHINESE, SUBLANG_CHINESE_SIMPLIFIED
#pragma code_page(936)
#endif //_WIN32
```
(Language and Code Page Definitions:This specifies the language and code page for the resources. In this case, it's set for simplified Chinese)

```
#ifdef APSTUDIO_INVOKED
1 TEXTINCLUDE DISCARDABLE
BEGIN
    "resource.h\0"
END
```

```
2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include ""afxres.h""\r\n"
    "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
    "#define _AFX_NO_SPLITTER_RESOURCES\r\n"
    "#define _AFX_NO_OLE_RESOURCES\r\n"
    "#define _AFX_NO_TRACKER_RESOURCES\r\n"
    "#define _AFX_NO_PROPERTY_RESOURCES\r\n"
    "\r\n"
    "#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_CHS)\r\n"
    "#ifdef _WIN32\r\n"
    "LANGUAGE 4, 2\r\n"
    "#pragma code_page(936)\r\n"
    "#endif //_WIN32\r\n"
    "#include ""res\\CntrlOtherPrcss.rc2""    // non-Microsoft Visual C++ edited resources\r\n"
    "#include ""l.chs\\afxres.rc""          // Standard components\r\n"
    "#endif\r\n"
    "\0"
END

#endif    // APSTUDIO_INVOKED
```

(Resource Definition Block:This block contains resource definitions, such as an application icon (IDR_MAINFRAME) and two dialog boxes (IDD_ABOUTBOX and IDD_CNTRLOTHERPRCSS_DIALOG). It also includes version information for the application.)

```
// Icon with lowest ID value placed first to ensure application icon
// remains consistent on all systems.
IDR_MAINFRAME           ICON    DISCARDABLE     "res\\CntrlOtherPrcss.ico"


// Dialog

IDD_ABOUTBOX DIALOG DISCARDABLE  0, 0, 235, 55
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "¹ØÓÚ CntrlOtherPrcss"
FONT 9, "ËÎÌå"
BEGIN
    ICON            IDR_MAINFRAME,IDC_STATIC,11,17,20,20
    LTEXT           "CntrlOtherPrcss 1.0 °æ",IDC_STATIC,40,10,119,8,
                    SS_NOPREFIX
```

```
    LTEXT           "°æÈ¨ËùÓÐ (C) 2002",IDC_STATIC,40,25,119,8
    DEFPUSHBUTTON   "È·¶¨",IDOK,178,7,50,14,WS_GROUP
END

IDD_CNTRLOTHERPRCSS_DIALOG DIALOGEX 0, 0, 207, 99
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION |
WS_SYSMENU
EXSTYLE WS_EX_APPWINDOW
CAPTION "CntrlOtherPrcss"
FONT 9, "ËÎÌå"
BEGIN
    DEFPUSHBUTTON   "È·¶¨",IDOK,15,70,50,14
    PUSHBUTTON      "È¡Ïû",IDCANCEL,135,70,50,14
    PUSHBUTTON      "Æô¶¯",IDC_START,30,20,65,16
    PUSHBUTTON      "ÖÕÖ¹",IDC_STOP,100,20,65,16
END


#ifndef _MAC

#(Version Information: This section provides version information for the application, including file version, product version, and various descriptive strings.)
VS_VERSION_INFO VERSIONINFO
 FILEVERSION 1,0,0,1
 PRODUCTVERSION 1,0,0,1
 FILEFLAGSMASK 0x3fL
#ifdef _DEBUG
 FILEFLAGS 0x1L
#else
 FILEFLAGS 0x0L
#endif
 FILEOS 0x4L
 FILETYPE 0x1L
 FILESUBTYPE 0x0L
BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK "080404B0"
        BEGIN
            VALUE "CompanyName", "\0"
            VALUE "FileDescription", "CntrlOtherPrcss Microsoft »ù´¡ÀàÓ¦ÓÃ³ÌÐò\0"
            VALUE "FileVersion", "1, 0, 0, 1\0"
            VALUE "InternalName", "CntrlOtherPrcss\0"
            VALUE "LegalCopyright", "°æÈ¨ËùÓÐ (C) 2002\0"
            VALUE "LegalTrademarks", "\0"
```

```
            VALUE "OriginalFilename", "CntrlOtherPrcss.EXE\0"
            VALUE "ProductName", "CntrlOtherPrcss Ó¦ÓÃ³ÌÐò\0"
            VALUE "ProductVersion", "1, 0, 0, 1\0"
        END
    END
    BLOCK "VarFileInfo"
    BEGIN
        VALUE "Translation", 0x804, 1200
    END
END

#endif    // !_MAC
```

// DESIGNINFO
//This part defines design information for the dialog boxes, specifying margins.

```
#ifdef APSTUDIO_INVOKED
GUIDELINES DESIGNINFO DISCARDABLE
BEGIN
    IDD_ABOUTBOX, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 228
        TOPMARGIN, 7
        BOTTOMMARGIN, 48
    END

    IDD_CNTRLOTHERPRCSS_DIALOG, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 200
        TOPMARGIN, 7
        BOTTOMMARGIN, 92
    END
END
#endif    // APSTUDIO_INVOKED
```

// String Table:This is the string table that contains localized strings. In this case, it includes a string for the "About Box."

```
STRINGTABLE DISCARDABLE
```

```
BEGIN
    IDS_ABOUTBOX                    "¹ØÓÚ CntrlOtherPrcss(&A)..."
END

#endif



//(Resource Inclusion for Non-Studio Invocations:This section includes additional resource files for non-Studio invocations.)
#ifndef APSTUDIO_INVOKED

// Generated from the TEXTINCLUDE 3 resource.
//
#define _AFX_NO_SPLITTER_RESOURCES
#define _AFX_NO_OLE_RESOURCES
#define _AFX_NO_TRACKER_RESOURCES
#define _AFX_NO_PROPERTY_RESOURCES

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_CHS)
#ifdef _WIN32
LANGUAGE 4, 2
#pragma code_page(936)
#endif //_WIN32
#include "res\CntrlOtherPrcss.rc2"    // non-Microsoft Visual C++ edited resources
#include "l.chs\afxres.rc"            // Standard components
#endif


#endif      // not APSTUDIO_INVOKED
```

CntrlOtherPrcssDlg.cpp: The above code is a C++ MFC (Microsoft Foundation Classes) application that implements a simple dialog-based interface to control other processes on Windows. The main functionality revolves around starting and stopping external processes.
The main dialog (CCntrlOtherPrcssDlg) includes buttons to start and stop a predefined external process (in this case, launching Notepad with a specific file). The program utilizes MFC message handlers for initializing the dialog, handling system commands, painting the dialog, and managing the start and stop buttons.
The **OnStart** function creates a new process using **CreateProcess** to launch Notepad with a specific file. If successful, a message is displayed. The **OnStop** function terminates the previously launched process if it exists, and appropriate messages are displayed for success or failure.
The code also includes an About dialog (CAboutDlg) for displaying information about the application. Overall, the application provides a basic GUI for controlling external processes.

Here is the code:

```
// CntrlOtherPrcssDlg.cpp : implementation file
#include "stdafx.h"
#include "CntrlOtherPrcss.h"
#include "CntrlOtherPrcssDlg.h"
#include "winbase.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
```

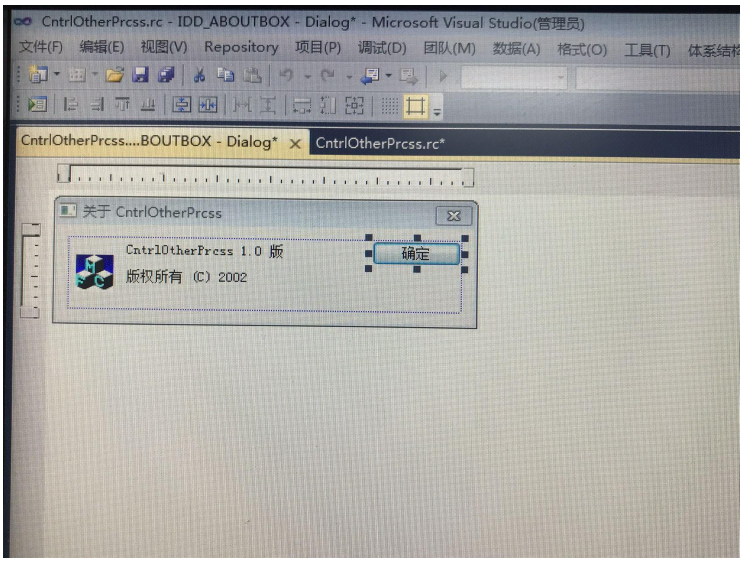
```
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// CCntrlOtherPrcssDlg dialog

CCntrlOtherPrcssDlg::CCntrlOtherPrcssDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CCntrlOtherPrcssDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CCntrlOtherPrcssDlg)
    //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
    m_hPro=NULL;
}

void CCntrlOtherPrcssDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CCntrlOtherPrcssDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CCntrlOtherPrcssDlg, CDialog)
    //{{AFX_MSG_MAP(CCntrlOtherPrcssDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_START, OnStart)
    ON_BN_CLICKED(IDC_STOP, OnStop)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()


// CCntrlOtherPrcssDlg message handlers

BOOL CCntrlOtherPrcssDlg::OnInitDialog()
```

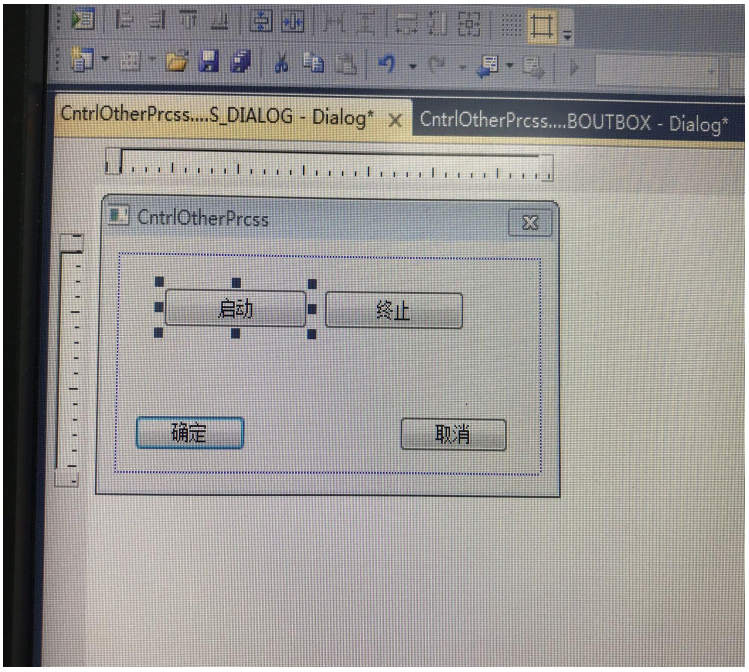
```
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }

    // Set the icon for this dialog.    The framework does this automatically
    //    when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);                 // Set big icon
    SetIcon(m_hIcon, FALSE);          // Set small icon

    // TODO: Add extra initialization here

    return TRUE;    // return TRUE    unless you set the focus to a control
}

void CCntrlOtherPrcssDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialog::OnSysCommand(nID, lParam);
    }
}

// If you add a minimize button to your dialog, you will need the code below
```

```
//    to draw the icon.    For MFC applications using the document/view model,
//    this is automatically done for you by the framework.

void CCntrlOtherPrcssDlg::OnPaint()
{
     if (IsIconic())
     {
          CPaintDC dc(this); // device context for painting

          SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

          // Center icon in client rectangle
          int cxIcon = GetSystemMetrics(SM_CXICON);
          int cyIcon = GetSystemMetrics(SM_CYICON);
          CRect rect;
          GetClientRect(&rect);
          int x = (rect.Width() - cxIcon + 1) / 2;
          int y = (rect.Height() - cyIcon + 1) / 2;

          // Draw the icon
          dc.DrawIcon(x, y, m_hIcon);
     }
     else
     {
          CDialog::OnPaint();
     }
}

// The system calls this to obtain the cursor to display while the user drags
//    the minimized window.
HCURSOR CCntrlOtherPrcssDlg::OnQueryDragIcon()
{
     return (HCURSOR) m_hIcon;
}

void CCntrlOtherPrcssDlg::OnStart()
{
     PROCESS_INFORMATION pi;
     STARTUPINFO si;
     //³õÊ¼»¯±äÁ¿
     memset(&si,0,sizeof(si));
     si.cb=sizeof(si);
     si.wShowWindow=SW_SHOW;
     si.dwFlags=STARTF_USESHOWWINDOW;
```

```
    //´ò¿ª¼ÇÊÂ±¾³ÌÐò
    BOOL fRet=CreateProcess(NULL,
                            "c:\\windows\\notepad.exe c:\\autoexec.bat",
                            NULL,
                            NULL,
                            FALSE,
                            NORMAL_PRIORITY_CLASS|CREATE_NEW_CONSOLE,
                            NULL,
                            NULL,
                            &si,
                            &pi);

    if(!fRet)
    {//´´½¨Ê§°Ü£¬ÏÔÊ¾´íÎóÐÅÏ¢
        LPVOID lpMsgBuf;
        FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
                      FORMAT_MESSAGE_FROM_SYSTEM |
                      FORMAT_MESSAGE_IGNORE_INSERTS,
                      NULL,
                      GetLastError(),
                      MAKELANGID(LANG_NEUTRAL,  SUBLANG_DEFAULT),  //
Default language
                      (LPTSTR) &lpMsgBuf,
                      0,
                      NULL );
        AfxMessageBox( (LPCTSTR)lpMsgBuf);
        LocalFree( lpMsgBuf );

    }
    else
    {
        AfxMessageBox("CreateProcess³É¹¦");
        m_hPro=pi.hProcess;
    }
}

void CCntrlOtherPrcssDlg::OnStop()
{
    //ÅÐ¶Ï½ø³Ì¾ä±úÊÇ·ñºÏ·¨
    if(m_hPro)
    {
        //¸ù¾Ý¾ä±ú£¬ÖÕÖ¹¸Õ´ò¿ªµÄ¼ÇÊÂ±¾³ÌÐò
        if(!TerminateProcess(m_hPro,0))
        {
            //ÖÕÖ¹³öÏ´íÎó£¬ÏÔÊ¾´íÎóÐÅÏ¢
```

```
            LPVOID lpMsgBuf;
            FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
                    FORMAT_MESSAGE_FROM_SYSTEM |
                    FORMAT_MESSAGE_IGNORE_INSERTS,
                    NULL,
                    GetLastError(),
                    MAKELANGID(LANG_NEUTRAL,   SUBLANG_DEFAULT),   //
Default language
                    (LPTSTR) &lpMsgBuf,
                    0,
                    NULL );
            AfxMessageBox( (LPCTSTR)lpMsgBuf);
            LocalFree( lpMsgBuf );
        }
        else
        {
            AfxMessageBox("TerminateProcess³É¹¦");
        }
        m_hPro=NULL;
    }
    else
    {
        AfxMessageBox("m_hProÎª¿Õ");
    }
}
```





Passing Data Through Message

The passing data through messages represents a pivotal advancement in this field. By leveraging messaging protocols, systems can transmit vital information swiftly and securely, fostering real-time responsiveness and enhancing overall control mechanisms. This approach not only streamlines communication within intricate processes but also lays the foundation for sophisticated automation and optimization, ultimately contributing to heightened productivity and reliability in industrial settings.

Sender.cpp:

```
// Sender.cpp : Defines the class behaviors for the application and  include necessary header files for the application.
#include "stdafx.h"
#include "Sender.h"
#include "SenderDlg.h"

//redefine the new operator to use a debug version and store the current file name for debugging information.
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


// CSenderApp
```

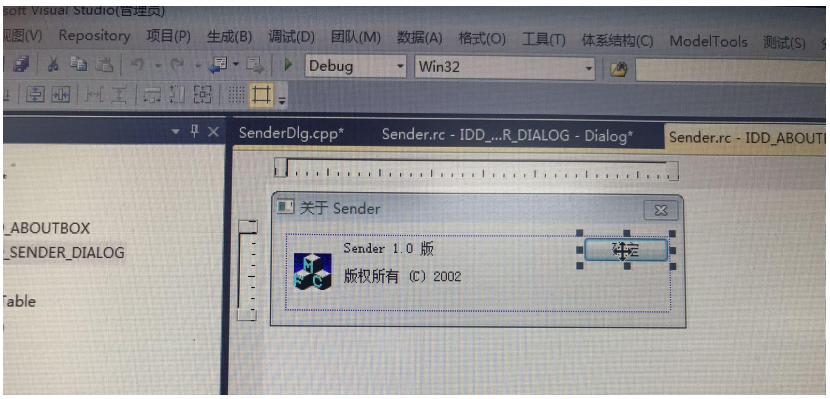
```
(//sets up the message map for the CSenderApp class. Message maps are used in MFC applications to map Windows messages to specific member functions of a class.)
BEGIN_MESSAGE_MAP(CSenderApp, CWinApp)
    //{{AFX_MSG_MAP(CSenderApp)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //       DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()


// CSenderApp construction for the CSenderApp class. Initialization code should be placed in the InitInstance function.

CSenderApp::CSenderApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}

//creates the one and only object of the CSenderApp class, making it a singleton. The instance is named theApp.
CSenderApp theApp;
// CSenderApp initialization
BOOL CSenderApp::InitInstance()
{
    AfxEnableControlContainer();
    // Standard initialization
    // If you are not using these features and wish to reduce the size
    //  of your final executable, you should remove from the following
    //  the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();             // Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic();// Call this when linking to MFC statically
#endif

    CSenderDlg dlg;
    m_pMainWnd = &dlg;
    int nResponse = dlg.DoModal();
    if (nResponse == IDOK)
    {
        // TODO: Place code here to handle when the dialog is
```

```
        //    dismissed with OK
    }
    else if (nResponse == IDCANCEL)
    {
        // TODO: Place code here to handle when the dialog is
        //    dismissed with Cancel
    }
    // Since the dialog has been closed, return FALSE so that we exit the
    //    application, rather than start the application's message pump.
    return FALSE;
}
```

The InitInstance function is a crucial part of MFC applications. It is called to perform application initialization. It enables control containers, initializes controls, creates a dialog (CSenderDlg), and handles the response from the dialog (OK or Cancel). If the dialog is dismissed with OK, you can add code in the corresponding TODO section to handle that situation. The function returns FALSE to signal that the application should exit.

In summary, this code defines the behavior of an MFC application. It creates a singleton instance of the CSenderApp class, initializes the application, creates a dialog (CSenderDlg), and handles user responses.

Sender.rc: The code is a Microsoft Developer Studio-generated resource script for a Windows application called "Sender." The application has a main frame with an icon and an about box. The resource script includes dialog definitions for the main application dialog and an about box dialog, both styled with a modal frame. The main dialog features buttons for sending commands and includes version information in Chinese (P.R.C.) resources. The version information includes details like product name, version, file description, and copyright. The code also defines design guidelines for the dialogs and string table entries.



SenderDlg.cpp:

```
#include "stdafx.h"
```

```
#include "Sender.h"
#include "SenderDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
// CAboutDlg dialog used for App About

#define WM_COMM WM_USER+100
class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
```

```
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()


// CSenderDlg dialog

CSenderDlg::CSenderDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CSenderDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CSenderDlg)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CSenderDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CSenderDlg)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CSenderDlg, CDialog)
	//{{AFX_MSG_MAP(CSenderDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_COMMAND1, OnCommand1)
	ON_BN_CLICKED(IDC_COMMAND2, OnCommand2)
	ON_BN_CLICKED(IDC_COMMAND3, OnCommand3)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

// CSenderDlg message handlers

BOOL CSenderDlg::OnInitDialog()
{
	CDialog::OnInitDialog();
```

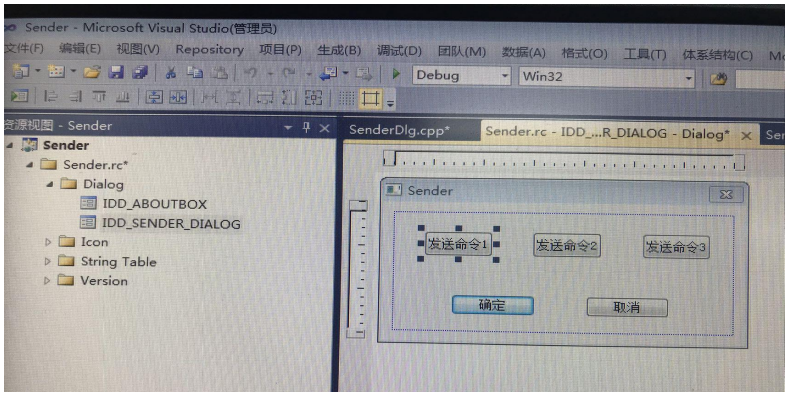
```
	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}
	}

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon
	// TODO: Add extra initialization here
	return TRUE;  // return TRUE  unless you set the focus to a control
}
void CSenderDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}
// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CSenderDlg::OnPaint()
{
	if (IsIconic())
```

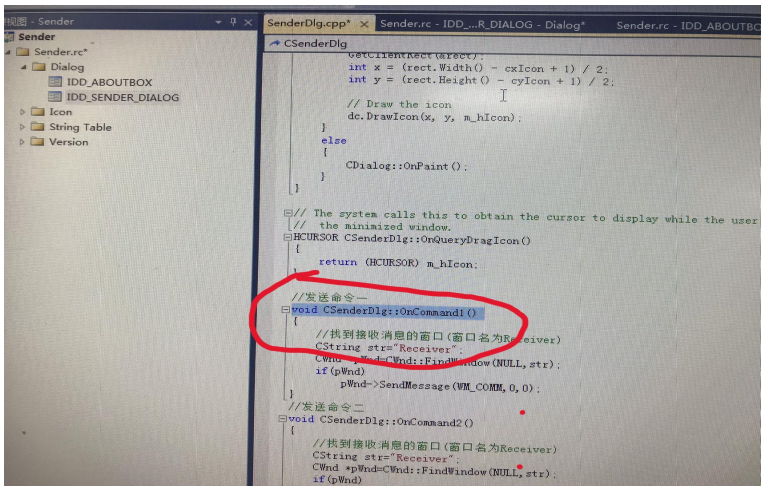
```
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialog::OnPaint();
    }
}
// The system calls this to obtain the cursor to display while the user drags// the minimized window.
HCURSOR CSenderDlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}
void CSenderDlg::OnCommand1()
{
    )
    CString str="Receiver";
    CWnd *pWnd=CWnd::FindWindow(NULL,str);
    if(pWnd)
        pWnd->SendMessage(WM_COMM,0,0);
}
void CSenderDlg::OnCommand2()
{
    CString str="Receiver";
    CWnd *pWnd=CWnd::FindWindow(NULL,str);
    if(pWnd)
        pWnd->SendMessage(WM_COMM,0,1);
}
void CSenderDlg::OnCommand3()
{
```

```
    CString str="Receiver";
    CWnd *pWnd=CWnd::FindWindow(NULL,str);
    if(pWnd)
        pWnd->SendMessage(WM_COMM,1,0);
}
```

This code is a part of a Windows MFC (Microsoft Foundation Classes) application written in C++. The application consists of two dialog boxes: CSenderDlg and CAboutDlg.

CSenderDlg: Represents the main dialog of the application. It contains message handlers for system commands, painting, and handling button clicks (OnCommand1, OnCommand2, and OnCommand3). The dialog has three buttons (IDC_COMMAND1, IDC_COMMAND2, and IDC_COMMAND3) that send custom messages to another window with the class name "Receiver" using the SendMessage function. The messages have different parameters (0, 0), (0, 1), and (1, 0) respectively.

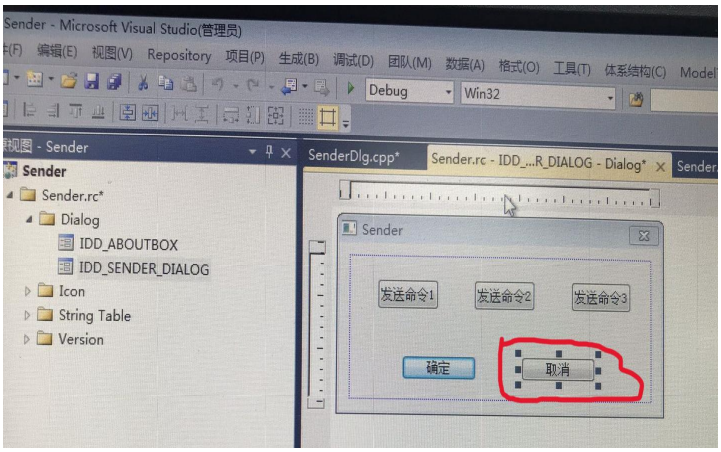


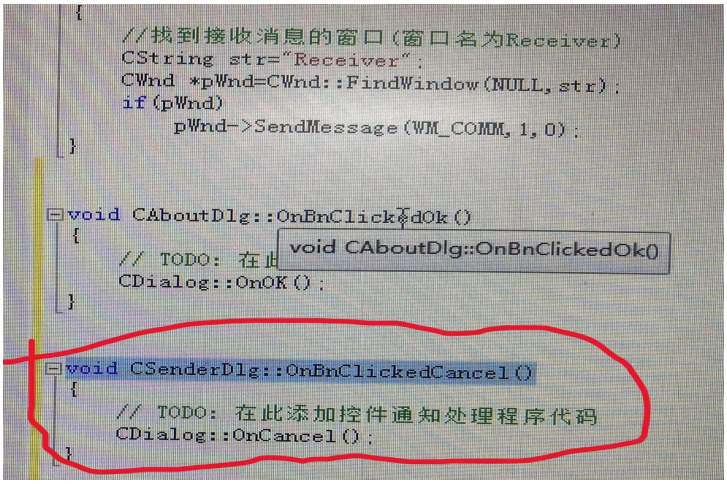


CAboutDlg: Represents the about box dialog, and it doesn't have any specific functionality in this code.

Overall, this code appears to be a basic Windows MFC application that sends custom messages

between dialog boxes using the SendMessage function. The purpose of these messages is not entirely clear without examining the code for the "Receiver" window.





Receiver.cpp: The provided C++ code represents an MFC (Microsoft Foundation Classes) application named "Receiver." The main functionality is implemented in the CReceiverApp class, inheriting from CWinApp. The application initializes a dialog (CReceiverDlg) and displays it, handling user responses such as OK or Cancel. The code includes standard MFC message mapping and initialization routines. The application flow is defined in the InitInstance method, where a dialog is created and shown. This code serves as the entry point for the Receiver application, demonstrating typical MFC structure and event handling for a simple dialog-based Windows application.

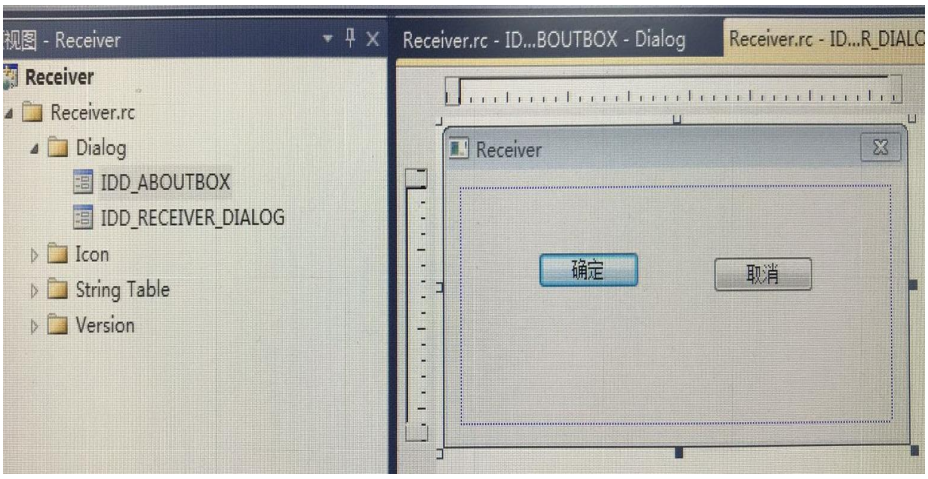
```
// Receiver.cpp : Defines the class behaviors for the application.
#include "stdafx.h"
#include "Receiver.h"
#include "ReceiverDlg.h"

#ifdef _DEBUG
```

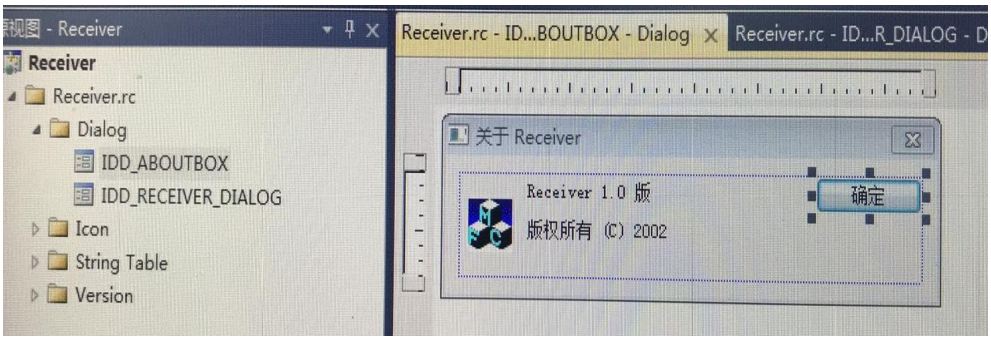
```
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
// CReceiverApp

BEGIN_MESSAGE_MAP(CReceiverApp, CWinApp)
	//{{AFX_MSG_MAP(CReceiverApp)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//	DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG
	ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()
// CReceiverApp construction

CReceiverApp::CReceiverApp()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}
// The one and only CReceiverApp object

CReceiverApp theApp;

// CReceiverApp initialization

BOOL CReceiverApp::InitInstance()
{
	AfxEnableControlContainer();

	// Standard initialization
	// If you are not using these features and wish to reduce the size
	//  of your final executable, you should remove from the following
	//  the specific initialization routines you do not need.

#ifdef _AFXDLL
	Enable3dControls();			// Call this when using MFC in a shared DLL
#else
	Enable3dControlsStatic();// Call this when linking to MFC statically
#endif

	CReceiverDlg dlg;
	m_pMainWnd = &dlg;
	int nResponse = dlg.DoModal();
	if (nResponse == IDOK)
```

```
    {
        // TODO: Place code here to handle when the dialog is
        //  dismissed with OK
    }
    else if (nResponse == IDCANCEL)
    {
        // TODO: Place code here to handle when the dialog is
        //  dismissed with Cancel
    }

    // Since the dialog has been closed, return FALSE so that we exit the
    //  application, rather than start the application's message pump.
    return FALSE;
}
```

ReceiverDlg.cpp : The application in the context of a message receiver. The main components include a main dialog (CReceiverDlg) and an About dialog (CAboutDlg). The application handles system commands, initializes the dialog, and defines message handlers.
The CReceiverDlg class initializes the dialog window, sets icons, and handles system commands. It also includes a custom message handler (OnReceiveMsg) that displays different messages based on the values of the message parameters (wParam and lParam).
The application defines a custom message, WM_COMM, and associates it with the OnReceiveMsg function using ON_MESSAGE. This implies that the dialog can receive and process custom messages during runtime.

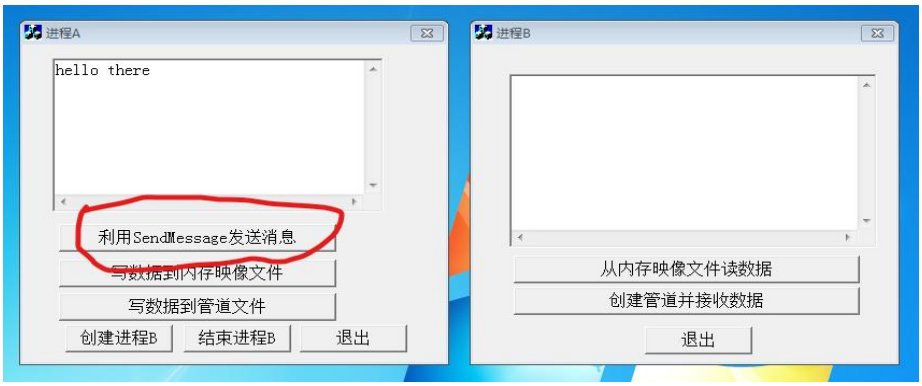
Overall, the code appears to be a skeleton for a Windows application that involves inter-process communication or messaging, with the specific message content triggering different message box alerts.





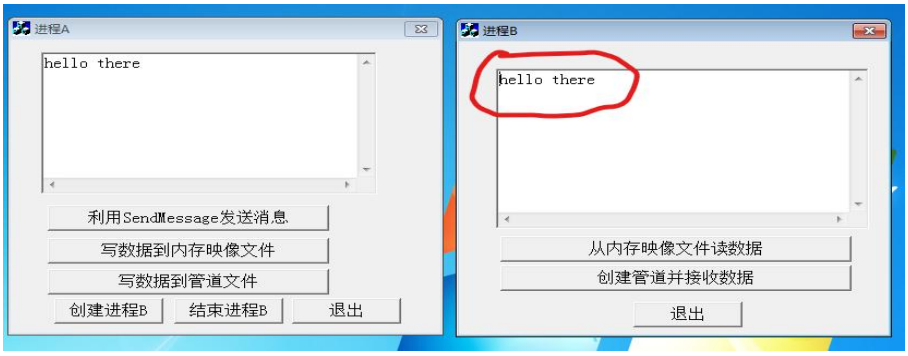
Final Result Analysis : After running the program, it will show the Process A and process B interface, Then

1.Input some characters in process A,Here we write "Hello there"



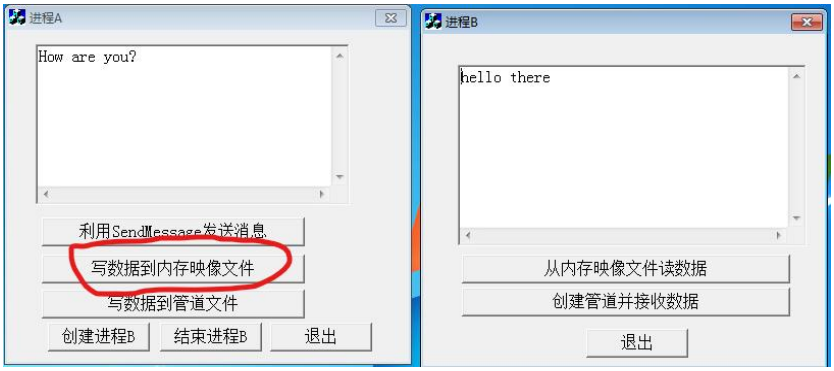
Then tap "Send Message with SendMessage" button to send the message to process B.

After clicking the "Send Message with SendMessage" button, We will see the output at process B.

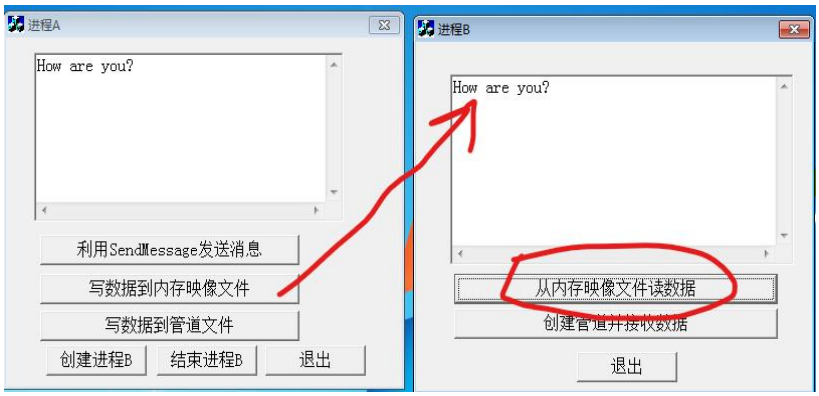


2.Input some characters in process A, tap "Write Data to Memory Image File" button, here we

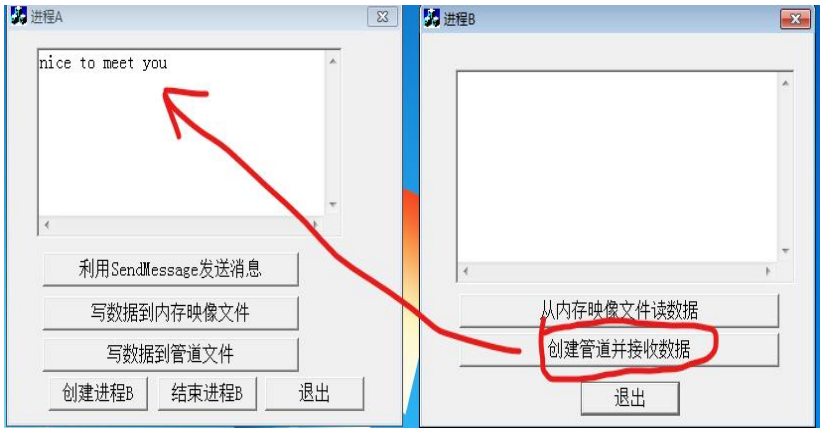
write “How are you?’



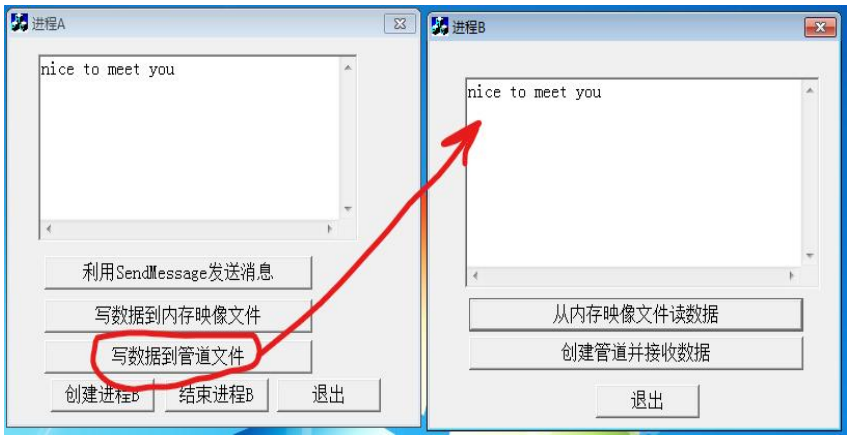
then tap "Read Data from Memory Image File" button in process B to receive the message.



1. Tap "Create Pipeline and Receive Data" button in process B, then enter some characters in process A



and tap "Write Data to Pipeline File" button to send the message to process B.



Difficulties Encountered and Solutions

In message communication, challenges may arise due to asynchronous processing and potential message loss. To address this, employ robust message queuing systems like RabbitMQ or Apache Kafka. Ensure message acknowledgment mechanisms are in place for reliability. Shared memory communication faces issues such as data consistency and synchronization problems. Implementing locking mechanisms, like semaphores or mutexes, can mitigate these challenges. Deadlocks may occur, so adopt deadlock detection and resolution strategies. Additionally, careful memory management and error handling are crucial. Ultimately, understanding the specific context and requirements will guide the selection and implementation of appropriate solutions for both message and shared memory communication.

Summery

Message communication and shared memory communication are two fundamental paradigms in inter-process communication (IPC). Message communication involves processes exchanging data through messages, allowing for asynchronous communication and independence. In contrast, shared memory communication involves processes accessing a common area of memory, facilitating faster data exchange but requiring synchronization mechanisms. Each approach has its strengths and weaknesses, influencing their suitability for different applications and system architectures.

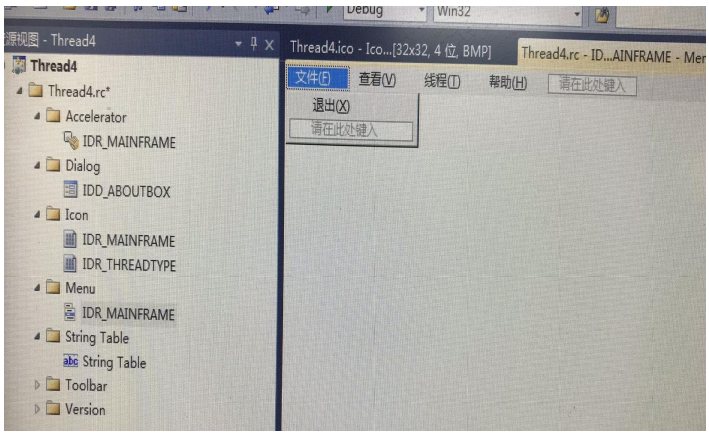
Multi-task system using system function call

A multi-tasking system using system function calls enables concurrent execution of multiple processes on a computer. The system function call allows the operating system to initiate separate tasks or programs, each with its own execution space. This approach enhances efficiency by enabling the system to handle various operations simultaneously, facilitating better resource utilization. System calls are crucial for managing the execution of diverse tasks, ensuring effective multitasking capabilities within an operating system environment

Thread Synchronization

Thread synchronization ensures orderly execution of concurrent threads, preventing data corruption or race conditions. Methods like locks, semaphores, and mutexes coordinate access to shared resources. Synchronization mechanisms facilitate communication and coordination among threads, ensuring consistent and predictable program behavior. Proper synchronization is crucial for maintaining data integrity and preventing conflicts in multithreaded environm

MainFrm.cpp : The provided C++ code defines the implementation of the CMainFrame class in the context of an MFC (Microsoft Foundation Class) application. The class inherits from CFrameWnd and includes a toolbar (m_wndToolBar) and a status bar (m_wndStatusBar). The code sets up the toolbar and status bar with necessary features and functionality, including docking capabilities. The class also overrides functions such as OnCreate and PreCreateWindow to customize the window creation process. The code mainly focuses on creating the main frame window for an MFC application, integrating a toolbar and a status bar, and configuring their properties.



Thread4.cpp : The provided code appears to be the implementation of a simple MFC (Microsoft Foundation Classes) application in C++ called "Thread4." The application consists of a main application class (CThread4App), a document class (CThread4Doc), a view class (CThread4View), and a main frame class (CMainFrame). The code includes standard MFC macros, message maps, and initialization routines.

The CThread4App class handles the initialization of the application, sets up document templates, and defines an About dialog (CAboutDlg). The main window is created with a specified title, and the application responds to standard commands such as file creation, opening, and printing.

```
// Thread4.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "Thread4.h"

#include "MainFrm.h"
#include "Thread4Doc.h"
#include "Thread4View.h" //These lines include necessary header files for the application. stdafx.h is often a precompiled header that includes frequently used headers to speed up compilation.

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif      //These lines are typically used for debugging. They redefine the new operator to use a debug version, and _DEBUG is a preprocessor directive used to include or exclude debugging code.

// CThread4App

BEGIN_MESSAGE_MAP(CThread4App, CWinApp)      //defines the message handlers for the application. It uses the ON_COMMAND macro to map menu commands to member functions.
    //{{AFX_MSG_MAP(CThread4App)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
    // Standard file based document commands
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    // Standard print setup command
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

// CThread4App construction

CThread4App::CThread4App()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}
// The one and only CThread4App object

CThread4App theApp; //Declaration and instantiation of the one and only object of CThread4App. It's a
```

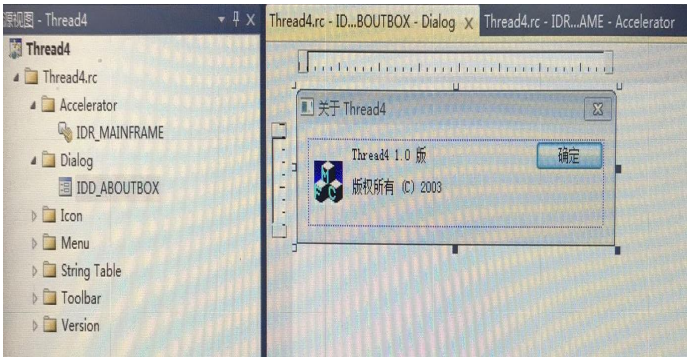
```
global object representing the application.

// CThread4App initialization

BOOL CThread4App::InitInstance()
{
	AfxEnableControlContainer();

	// Standard initialization
	// If you are not using these features and wish to reduce the size
	//  of your final executable, you should remove from the following
	//  the specific initialization routines you do not need.

#ifdef _AFXDLL
	Enable3dControls();			// Call this when using MFC in a shared DLL
#else
	Enable3dControlsStatic();	// Call this when linking to MFC statically
#endif

	// Change the registry key under which our settings are stored.
	// TODO: You should modify this string to be something appropriate
	// such as the name of your company or organization.
	SetRegistryKey(_T("Local AppWizard-Generated Applications"));

	LoadStdProfileSettings();  // Load standard INI file options (including MRU)

	// Register the application's document templates.  Document templates
	//  serve as the connection between documents, frame windows and views.

	CSingleDocTemplate* pDocTemplate;
	pDocTemplate = new CSingleDocTemplate(
		IDR_MAINFRAME,
		RUNTIME_CLASS(CThread4Doc),
		RUNTIME_CLASS(CMainFrame),       // main SDI frame window
		RUNTIME_CLASS(CThread4View));
	AddDocTemplate(pDocTemplate);

	// Parse command line for standard shell commands, DDE, file open
	CCommandLineInfo cmdInfo;
	ParseCommandLine(cmdInfo);

	// Dispatch commands specified on the command line
	if (!ProcessShellCommand(cmdInfo))
		return FALSE;
```

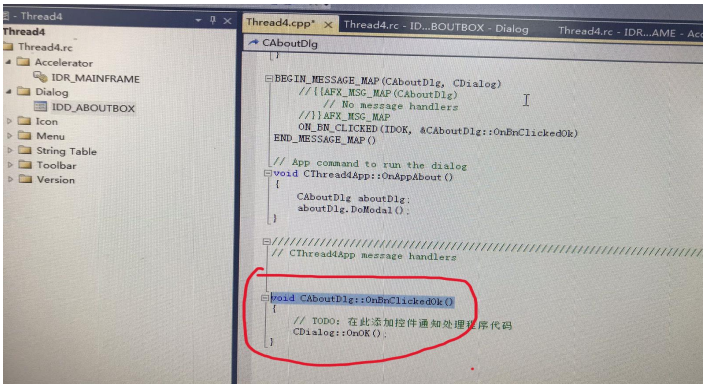
```
	// The one and only window has been initialized, so show and update it.
	m_pMainWnd->SetWindowText("Ïß³ÌÍ¬²½--Ê¹ÓÃÐÅºÅÁ¿");
	m_pMainWnd->ShowWindow(SW_SHOW);
	m_pMainWnd->UpdateWindow();

	return TRUE;
}

// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
		// No message handlers
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
```

```
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CThread4App::OnAppAbout()
{
	CAboutDlg aboutDlg;
	aboutDlg.DoModal();
}

// CThread4App message handlers
```

The overall structure is typical of a basic MFC application with a main frame, document, and view classes. The code initializes the application, sets up document templates, handles messages, and includes an about dialog. The application appears to be in Chinese based on the use of Chinese characters in some string literals.

Thread4.rc : The provided code appears to be a Microsoft Developer Studio generated resource script for a C++ application, possibly using MFC (Microsoft Foundation Classes). The script includes definitions for various resources such as icons, bitmaps, toolbars, menus, accelerators, dialogs, version information and string tables. It seems to be tailored for a Chinese (P.R.C.) locale, as indicated by language settings. The application, named "Thread4," has a main frame with icons, a toolbar, menu items, and accelerator keys. The resource script also defines dialog boxes, version information, and string tables for localization. Additionally, there are settings for resource DLLs and conditional compilation based on target settings.





Thread2.cpp : The application, named "Thread2," includes a main frame, document, and view classes. The code involves setting up the application, initializing various components, and handling user interface interactions.

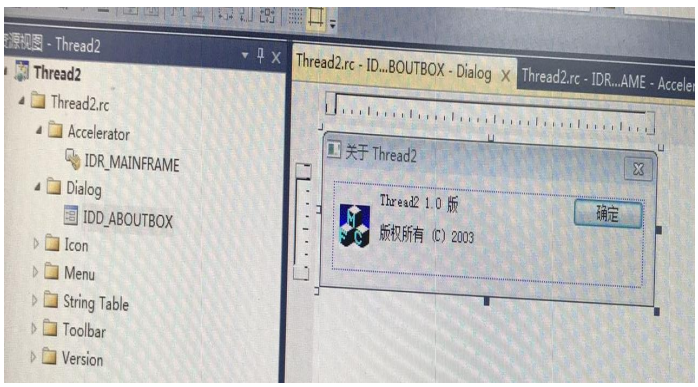
Key components include the definition of the CThread2App class, which inherits from CWinApp, and its initialization functions (InitInstance). The code sets up the main frame, document templates, and handles user commands such as file opening and printing. There is also an about dialog (CAboutDlg) that can be triggered by the user.

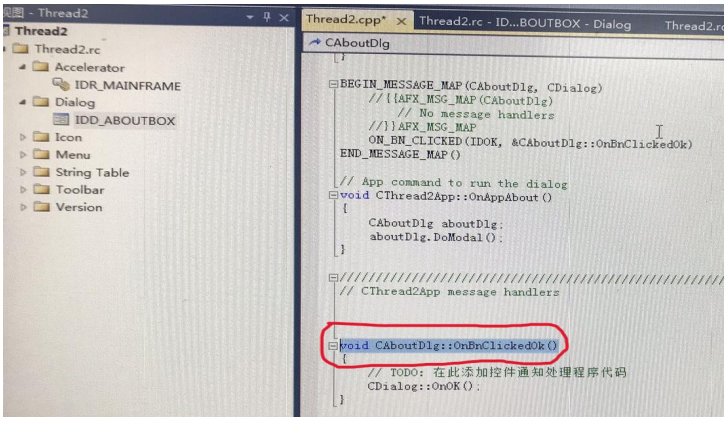
```
// Thread2.cpp : Defines the class behaviors for the application.

#include "stdafx.h"
#include "Thread2.h"

#include "MainFrm.h"
#include "Thread2Doc.h"
#include "Thread2View.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
// CThread2App

BEGIN_MESSAGE_MAP(CThread2App, CWinApp)
	//{{AFX_MSG_MAP(CThread2App)
	ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG_MAP
	// Standard file based document commands
	ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
```

```
	ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
	// Standard print setup command
	ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

// CThread2App construction

CThread2App::CThread2App()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}

// The one and only CThread2App object

CThread2App theApp;

// CThread2App initialization

BOOL CThread2App::InitInstance()
{
	AfxEnableControlContainer();

	// Standard initialization
	// If you are not using these features and wish to reduce the size
	//  of your final executable, you should remove from the following
	//  the specific initialization routines you do not need.

#ifdef _AFXDLL
	Enable3dControls();			// Call this when using MFC in a shared DLL
#else
	Enable3dControlsStatic();	// Call this when linking to MFC statically
#endif

	// Change the registry key under which our settings are stored.
	// TODO: You should modify this string to be something appropriate
	// such as the name of your company or organization.
	SetRegistryKey(_T("Local AppWizard-Generated Applications"));

	LoadStdProfileSettings();  // Load standard INI file options (including MRU)

	// Register the application's document templates.  Document templates
	//  serve as the connection between documents, frame windows and views.

	CSingleDocTemplate* pDocTemplate;
```

```
	pDocTemplate = new CSingleDocTemplate(
		IDR_MAINFRAME,
		RUNTIME_CLASS(CThread2Doc),
		RUNTIME_CLASS(CMainFrame),		// main SDI frame window
		RUNTIME_CLASS(CThread2View));
	AddDocTemplate(pDocTemplate);

	// Parse command line for standard shell commands, DDE, file open
	CCommandLineInfo cmdInfo;
	ParseCommandLine(cmdInfo);

	// Dispatch commands specified on the command line
	if (!ProcessShellCommand(cmdInfo))
		return FALSE;

	// The one and only window has been initialized, so show and update it.
	m_pMainWnd->SetWindowText("Ïß³ÌÍ¬²½--Ê¹ÓÃÁÙ½çÇø");
	m_pMainWnd->ShowWindow(SW_SHOW);
	m_pMainWnd->UpdateWindow();

	return TRUE;
}


// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
```

```
        // No message handlers
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CThread2App::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}
// CThread2App message handlers
```





Final Result analysis

1.Tap the "Create Thread" button to create two threads. One thread keeps adding 1 to a variable and the result is displayed in the first text box.

The other thread keeps subtracting 1 from another variable, and the result is displayed in the second text box. There is no interaction between the two threads, and it is only used to demonstrate the creation of the thread.

2.Demonstrate thread mutual exclusion. Tap the "Thread Mutual Exclusion" button to create two threads, one thread keeps looping, each loop does 100 times plus 1 operation on the shared variable x (these 100 times plus 1 operation as a critical zone CSa),

the other thread keeps looping, each loop does 100 times minus 1 operation on the shared variable x (these 100 times minus 1 operation as a critical zone CSa), the other thread keeps looping, each loop does 100 times minus 1 operation on the shared variable x (these 100 times minus 1 operation as a critical zone CSa). These 100 minus 1 operations act as a critical area CSb), and the result is displayed in the third text box. You can see that the result is from 0 to 100, and then from 100 back to 0. You can see that the two critical areas CSa and CSb are mutually exclusive.

Same as step 2 except no mutex operation, the results are displayed in the fourth text box. It can be seen that the execution of the two critical areas CSa and CSb are intersected, the execution of CSa may be interrupted by CSb, the execution of CSb may also be interrupted by CSa.

Difficulties Encountered and Solutions

Difficulties in a multi-task system using system function calls include:

Concurrency Issues: Simultaneous execution of tasks may lead to race conditions or data corruption.
Resource Contentions: Competing tasks may vie for system resources, causing bottlenecks.
Dependency Management: Inter-task dependencies can result in execution order conflicts.
Error Handling: Managing errors across multiple tasks can be complex and may impact overall system stability.

Solutions involve:

Synchronization Mechanisms: Implement locks or semaphores to manage access to shared resources.
Prioritization: Assign task priorities to resolve resource contention.
Dependency Graphs: Use dependency graphs to visualize and manage task relationships.
Error Logging and Handling: Implement robust error-handling mechanisms to isolate and address issues without affecting the entire system.

Summery

A multi-task system employing system function calls is designed to execute multiple tasks concurrently through distinct function calls. This approach enhances efficiency by allowing parallel execution of diverse functions within the same system. System function calls enable seamless communication between tasks, optimizing resource utilization and promoting modular task management. This methodology enhances overall system performance, enabling tasks to run concurrently, share information, and achieve streamlined execution.

Future Enhancement

Future enhancements in message communication will focus on real-time, immersive experiences through advanced multimedia integration, ensuring seamless transmission of high-quality audio, video, and holographic content. Additionally, improved security measures, such as quantum-resistant encryption, will fortify data integrity. Shared memory communication will evolve with decentralized architectures, enabling efficient collaboration across global networks. AI-driven context-awareness will enhance message comprehension, automating tasks based on user intent. As technology progresses, these advancements will redefine the way people connect, collaborate, and share information, ushering in a new era of intelligent and interactive communication.