Windows Concurrency Mechanisms

Shuquan Li

March 28, 2014

ECT362 Homework #4

Introduction:

This homework is based on the last homework. The goal is to use at least two different synchronization mechanisms to protect both the file and the local these two shared resources, and avoid the deadlock and starvation. This homework is also desired to have some random delays to simulate a very heavily utilized network.

Design and Analysis:

The flow chart, shown in Figure 1, indicates what the program do when the user run the program.

First, the program reads the context information from the file and shows them in the list box when the program starts. Then the program reads three commands. One command is whether the user presses the “ADD” button; if yes, it will read what user type into the edit box and then show them in the list box and store them into the file; if the user doesn’t click the “ADD” button, the program will do nothing about “ADD”.

One command is whether the user presses the “DELETE” button. If the “DELETE” button is pressed, the program will read the list box to get the index of the contact information which the user select, then it will delete the selected contact information both from the list box and the local array. If the user doesn’t select anything and just clicks “DELETE” button, no contact information will be deleted. After the program deletes the contact information, all the changes will be stored into the file. If the button is not pressed, the program will do nothing about “DELETE”.

Another command is whether the timer’s time is up. If time is up, the program will empty the list box and read the contact information from the file, and then show the contact information in the list box to tell the user what the newest file looks like.

The program will keeping doing these until the user exits the window and ends the program.

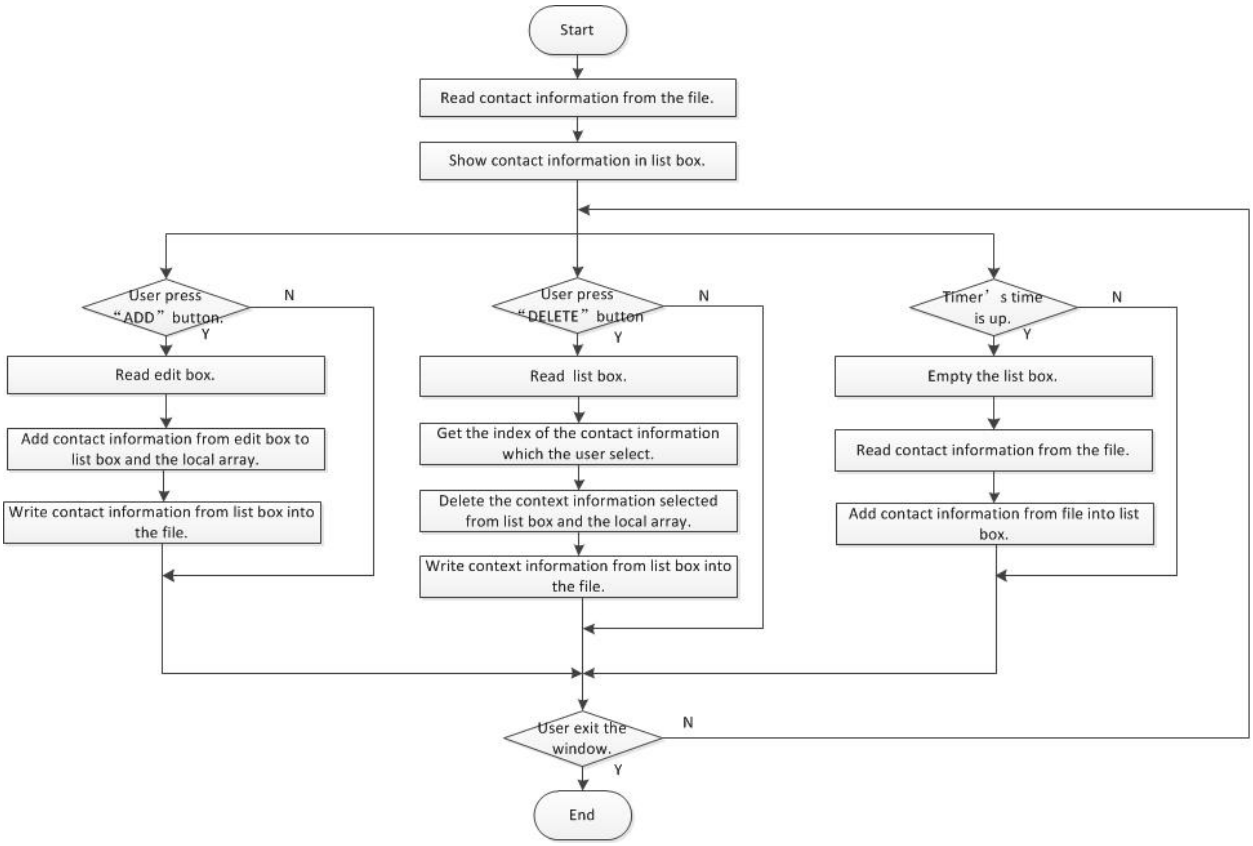

Figure 1: Flow chart of the program design.

In order to make sure three threads don't write into the file and the local array at the same time, two synchronization mechanisms are used to provide mutual exclusion. Mutex object is used to protect the local array and critical section object is used to protect the file. Put "wait()" function at the beginning of the shared resources to get access and put "signal()" function at the end of the shared resources to let other threads have access.

As the three threads all write into the list box, "SendDlgItemMessage()" functions are put into mutex, too. Otherwise, if add thread works when the refresh thread is emptying the list box, the information may show in list box but not store into the file and the local array because the refresh thread is using them.

It is said that a fair system prevents deadlock and starvation. In this program, the system is fair because each thread gets enough access to the shared resources so deadlock or starvation does not occur.

Random delays are used to simulate the network and file I/O delays. C standard general utilities library "<cstdlib>" is needed to generate random numbers.

When the user presses the “ADD” button to add something when the timer’s time is up and the refresh thread should work, the list box won’t read the file and refresh at that time because the add thread is working. After the information is added in the list box, it will refresh. The same when the user presses “DELETE” button. When the program is refreshing and the list box is blank due to the delay, nothing will be added into the file and the local array because the refresh thread is running and using them at that time. If the user selected the information and presses “DELETE” button, during the random delay, nothing can be added into the file and local array, and after the information has been deleted, the added information will be added.

Conclusion

It is desired to modify the program to provide mutual exclusion access and make the three threads run synchronously. The program should have a randomly generated time sleep to simulate delays due to resource sharing in a heavily utilized network.

When using semaphore objects, the program will keep waiting for the access to the file and the local array and there is no problem when using mutex and critical section.

References

header <cstdlib> (stdlib.h). cplusplus.com. nd. np. Web. Mar. 28, 2014
function rand. cplusplus.com. nd. np. Web. Mar. 28, 2014
Fairness, Starvation, and Deadlock. Multithreaded Programs. nd. np. Web. Mar. 28, 2014

Appendix

```
/* Preprocessor Directives */
#include <windows.h>  // Win32 API
#include <process.h>  // process & thread
#include "resource.h" // resources for your application
#include <iostream>   // cin, cout, endl
#include <string.h>   // strcpy()
#include <stdio.h>    // fgets()
#include <stdlib.h>   // random number generation

#define ROWS 100
#define COLS 100

using namespace std;  // using the standard namespace

struct contact
{
	char fnames[100];  // array to hold first name
	char lnames[100];  // array to hold last name
	char emails[100];  // array to hold last name
	char phnums[100];  // array to hold last name
};
```

```
#define ID_ADDBUTTON        1001
#define ID_LIST             1010
#define ID_EDIT             1020
#define ID_DELEBUTTON       1030
#define IDT_TIMER           1040

/* Function Prototypes */
LRESULT CALLBACK WndProc( HWND, UINT, WPARAM, LPARAM );  //the window procedure
/* A callback function is passed (by reference) to another function */
DWORD WINAPI addthread( LPVOID );
DWORD WINAPI delethread( LPVOID );
DWORD WINAPI refreshthread( LPVOID );

/* Global variables */
contact contacts[100];
int in_cnt=0;
char line[COLS];
HANDLE mutexR;
CRITICAL_SECTION CriticalSectionObject;

/* Functions */
/*****************************************************************
 * Function    | WinMain()
 * Description | Entry point for our application, we create and
 *             |  register a window class and then call CreateWindow
 * Inputs      | None
 * Output      | Integer value 0
 *****************************************************************/
int WINAPI WinMain( HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR szCmdLine,
                    int iCmdShow )
{
    static char szAppName[] = "My Window";

    HWND        hwnd;
    WNDCLASSEX  wndclass;  // This is our new windows class
    MSG msg;

    /*  Fill in WNDCLASSEX struct members  */
    wndclass.cbSize         = sizeof(wndclass);
    wndclass.style          = CS_HREDRAW | CS_VREDRAW;
    wndclass.lpfnWndProc    = WndProc;
    wndclass.cbClsExtra     = 0;
    wndclass.cbWndExtra     = 0;
    wndclass.hInstance      = hInstance;
    wndclass.hIcon          = LoadIcon(GetModuleHandle(NULL),
                              MAKEINTRESOURCE(IDI_MYICON));
    wndclass.hIconSm        = (HICON)LoadImage(GetModuleHandle(NULL),
                              MAKEINTRESOURCE(IDI_MYICON), IMAGE_ICON, 16, 16, 0);
    wndclass.hCursor        = LoadCursor(NULL, IDC_ARROW);
    wndclass.hbrBackground  = (HBRUSH) GetStockObject(WHITE_BRUSH);
    wndclass.lpszClassName  = szAppName;
    wndclass.lpszMenuName   = NULL;

    /*  Register a new window class with Windows  */
    RegisterClassEx(&wndclass);
```

```
    /*  Create a window based on our new class  */
    hwnd = CreateWindow(szAppName,  // class name the window is to be based on
                        "My 1st Window",  // the title of the window that will
                                          //  appear in the bar at the top
                        WS_OVERLAPPEDWINDOW,  // window style (a window that
                                              //  has a caption, a system menu,
                                              //  a thick frame and a minimise
                                              //  and maximise box)
                        /* Use default starting location for window */
                        CW_USEDEFAULT, // initial x position (top left corner)
                        CW_USEDEFAULT, // initial y position (top left corner)
                        850, // initial width
                        440, // initial height
                        NULL, // window parent (NULL for not a child window)
                        NULL, // menu (NULL to use class menu)
                        hInstance, // the program instance passed to us
                        NULL);  // pointer to any parameters wished to be
                                //  passed to the window producer when the window
                                //  is created

    if(hwnd == NULL)  // check to see if an error occurred in creating the window
    {
        MessageBox(NULL, "Window Creation Failed!", "Error!",
            MB_ICONEXCLAMATION | MB_OK);
        return 0;
    }

    /* Show and update our window  */
    ShowWindow(hwnd, iCmdShow);
    UpdateWindow(hwnd);


    /*  Retrieve and process any queued messages until we get WM_QUIT  */
    /* Recall that Windows uses a messaging system to notify window of */
    /*  user actions
          */
    while ( GetMessage(&msg, NULL, 0, 0) )
    {
       TranslateMessage(&msg);  // for certain keyboard messages
       DispatchMessage(&msg);  // send message to WndProc
    }

    /*  Exit with status specified in WM_QUIT message  */
    return msg.wParam;
} // end WinMain()


/*************************************************************
 * Function    | WinProc()
 * Description | Whenever anything happens to your window, Windows
 *             |  will call this function telling you what has happened.
 *             |  The message parameter contains the message sent
 * Inputs      | None
 * Output      | Integer value 0
 *************************************************************/
LRESULT CALLBACK WndProc( HWND hwnd,
                          UINT iMsg,
                          WPARAM wParam,
```

```
                        LPARAM lParam )
{
    PAINTSTRUCT ps;
    HDC hdc;
    static HWND  addbutton, deletebutton, listbox, editbox_fnames, editbox_lnames,
                 editbox_emails, editbox_phnums, thread_hwnds[5];
    FILE *file_ptr;
    HANDLE aThread; // declarte handle for thread associated with ADD button,
                       DELETE button, and Refresh
    LPSECURITY_ATTRIBUTES lpsa = NULL; //
    DWORD dwStackSize = 0; //
    DWORD dwCreationFlags = 0; //
    DWORD aThreadID; //

    /*  Switch according to what type of message we have received  */
    switch ( iMsg )
    {
       case WM_PAINT:
              /* We receive WM_PAINT every time window is updated  */
              hdc = BeginPaint(hwnd, &ps);
              TextOut(hdc, 10, 350, "My First Window", 15);
              TextOut(hdc, 10, 133, "First Name", 10);
              TextOut(hdc, 10, 173, "Last Name", 9);
              TextOut(hdc, 10, 213, "Email Address", 13);
              TextOut(hdc, 10, 253, "Phone Number0", 12);
              EndPaint(hwnd, &ps);
              break;

       case WM_CREATE:
       /* Operations to be performed whenthis window is created */
              /* Create a child window for a pushbutton */
              addbutton = CreateWindow(  "BUTTON",  // predefined class
                                  "ADD",  // button text
                                  WS_VISIBLE | WS_CHILD | BS_PUSHBUTTON,
                                  // Size and position values are given
                                  //  explicitly, because the CW_USEDEFAULT
                                  //  constant gives zero values for buttons.
                                  10,  // starting x position
                                  40,  // starting y position
                                  100, // button width
                                  30,  // button height
                                  hwnd, // parent window
                                  (HMENU)ID_ADDBUTTON, // no menu
                                  (HINSTANCE) 0,  // ignored for Windows XP
                                  NULL );  // pointer not needed
              deletebutton = CreateWindow(  "BUTTON",  // predefined class
                                  "DELETE", // button text
                                  WS_VISIBLE | WS_CHILD | BS_PUSHBUTTON,
                                  // Size and position values are given
                                  //  explicitly, because the CW_USEDEFAULT
                                  //  constant gives zero values for buttons.
                                  140, // starting x position
                                  40,  // starting x position
                                  100, // button width
                                  30,  // button height
                                  hwnd, // parent window
                                  (HMENU)ID_DELEBUTTON, // no menu
                                  (HINSTANCE) 0, // ignored for Windows XP
```

```
                    NULL );  // pointer not needed
/* Create a list box to display values within the window */
listbox = CreateWindow( "LISTBOX",  // predefined class
                    "",  // initial lst box text (empty)
                    // automatically sort items in list box
                    //  and include a vertical scroll bar
                    //  with a border
                    WS_CHILD | WS_VISIBLE | LBS_NOTIFY |
                    WS_VSCROLL | WS_BORDER,
                    270,  // starting x position
                    10,  // starting y position
                    550,  // list box width
                    400,  // list box height
                    hwnd,  // parent window
                    (HMENU)ID_LIST,  // no menu
                    (HINSTANCE) 0,  // ignored for Windows XP
                    NULL ); // pointer not needed
/* Create edit box for adding and deleting items from list */
editbox_fnames = CreateWindow( "EDIT",  // predefined class
                    "",  // initial lst box text (empty)
                    // create an edit box with a border
                    WS_CHILD | WS_VISIBLE | WS_BORDER,
                    10,  // starting x position
                    150,  // starting y position
                    250,  // list box width
                    20,  // list box height
                    hwnd,  // parent window
                    (HMENU)ID_EDIT,  // no menu
                    (HINSTANCE) 0,  // ignored for Windows XP
                    NULL); // pointer not needed
editbox_lnames = CreateWindow( "EDIT",  // predefined class
                    "",  // initial lst box text (empty)
                    // create an edit box with a border
                    WS_CHILD | WS_VISIBLE | WS_BORDER,
                    10,  // starting x position
                    190,  // starting y position
                    250,  // list box width
                    20,  // list box height
                    hwnd,  // parent window
                    (HMENU)ID_EDIT,  // no menu
                    (HINSTANCE) 0,  // ignored for Windows XP
                    NULL); // pointer not needed
editbox_emails = CreateWindow( "EDIT",  // predefined class
                    ",  // initial lst box text (empty)
                    // create an edit box with a border
                    WS_CHILD | WS_VISIBLE | WS_BORDER,
                    10,  // starting x position
                    230,  // starting y position
                    250,  // list box width
                    20,  // list box height
                    hwnd,  // parent window
                    (HMENU)ID_EDIT,  // no menu
                    (HINSTANCE) 0,  // ignored for Windows XP
                    NULL); // pointer not needed
editbox_phnums = CreateWindow( "EDIT",  // predefined class
                    "",  // initial lst box text (empty)
                    // create an edit box with a border
                    WS_CHILD | WS_VISIBLE | WS_BORDER,
```

```
                            10,  // starting x position
                            270,  // starting y position
                            250,  // list box width
                            20,  // list box height
                            hwnd,  // parent window
                            (HMENU)ID_EDIT,  // no menu
                            (HINSTANCE) 0,  // ignored for Windows XP
                            NULL); // pointer not needed
        mutexR = CreateMutex ( NULL, FALSE, NULL ); // create the mutex object with
                                                    //  no ownership (signaled)
                                                    // protect the local array
        InitializeCriticalSection ( &CriticalSectionObject );
                                        // initialize the critical section
                                        // protect the file

        SetTimer(hwnd,       // handle to main window
        IDT_TIMER,           // timer identifier
        10000,               // 10-second interval
        (TIMERPROC) NULL);    // no timer callback

        /* Read command line information */
        file_ptr = fopen("contacts.txt", "r");  // open the contacts.txt file for
                                                //  reading only
        if( file_ptr == NULL ) // failed to open file
        {
            MessageBox(NULL, "Error Opening File", "ERROR",
                       MB_ICONINFORMATION | MB_OK);
        }
        /* Read file line-by-line, storing each line into a seperate row in */
        /*  the names[][] string array                                      */
        in_cnt;
        while( fgets( line, COLS, file_ptr) != NULL )
                                        //no infromation i got from file.
        {
            sscanf(line, "%s\t%s\t%s\t%s", contacts[in_cnt].fnames,
                   contacts[in_cnt].lnames, contacts[in_cnt].emails,
                   contacts[in_cnt].phnums);
            sprintf(line, "%s  %s  %s  %s", contacts[in_cnt].fnames,
                   contacts[in_cnt].lnames, contacts[in_cnt].emails,
                   contacts[in_cnt].phnums);
            SendDlgItemMessage( hwnd, ID_LIST,
                                LB_ADDSTRING, 0,
                                (LPARAM)line);
            in_cnt++;
        }
        fclose(file_ptr);  //close file since all data has been extracted
        break;

    case WM_TIMER:
        switch (wParam)
        {
            case IDT_TIMER:
               /* set the HWND array for passing to thread */
               thread_hwnds[0] = hwnd;  // pass handle of parent
                                        //  window
               // process the 10-second timer

               /* Create thread for refreshing*/
```

```
                aThread = (HANDLE) CreateThread(
                          lpsa,
                          dwStackSize,
                          (LPTHREAD_START_ROUTINE)refreshthread,
                              // long pointer to a thread start routine
                          (LPVOID) &thread_hwnds,
                          dwCreationFlags,
                          (LPDWORD) &aThreadID );
        break;
        }
break;

case WM_DESTROY:
/* Window has been destroyed, so exit cleanly  */
        DeleteCriticalSection(&CriticalSectionObject);
        PostQuitMessage(0);
        break;

case WM_COMMAND:
/* User selected a command from a menu or a control sent a message */
        if (HIWORD(wParam) == BN_CLICKED)
        {
            switch (LOWORD(wParam))
            {
               case ID_ADDBUTTON:
                      /* set the HWND array for passing to thread */
                      thread_hwnds[0] = hwnd;  // pass handle of parent
                                               //  window
                      thread_hwnds[1] = editbox_fnames; // pass handle to textbox
                      thread_hwnds[2] = editbox_lnames; //  window
                      thread_hwnds[3] = editbox_emails;
                      thread_hwnds[4] = editbox_phnums;
                      /* Create thread for performing actions associated */
                      /*  with ADD button                                */
                      aThread = (HANDLE) CreateThread(
                                lpsa,
                                dwStackSize,
                                (LPTHREAD_START_ROUTINE)addthread,
                                    //long pointer to a thread start routine
                                    //start from the basic address of addthread
                                (LPVOID) &thread_hwnds,
                                dwCreationFlags,
                                (LPDWORD) &aThreadID );
                      break;

               case ID_DELEBUTTON:
                      /* set the HWND array for passing to thread */
                      thread_hwnds[0] = hwnd;  // pass handle of parent
                                              //  window
                      /* Create thread for performing actions associated */
                      /*  with DELETE button                             */
                      aThread = (HANDLE) CreateThread(
                                lpsa,
                                dwStackSize,
                                (LPTHREAD_START_ROUTINE)delethread,
                                    //long pointer to a thread start routine
                                (LPVOID) &thread_hwnds,
                                dwCreationFlags,
```

```
                                        (LPDWORD) &aThreadID );
                                break;
                        }
                }
                break;
        default:
        /* We do not want to handle this message so pass back to Windows */
        /*  to handle it in a default way                                 */
                return DefWindowProc(hwnd, iMsg, wParam, lParam);
    }
    return 0;

} // end WndProc

DWORD WINAPI
addthread(LPVOID hwnd_addr)
{
        int len, out_cnt, random;
        HWND *thread_hwnds, hwnd, editbox_fnames, editbox_lnames, editbox_emails,
                editbox_phnums; //thread_hwnds is a pointer to a handle
        DWORD result = 0;
        FILE *file_ptr;

        thread_hwnds = (HWND *)hwnd_addr;
        hwnd = thread_hwnds[0]; // extract parent window handle
        editbox_fnames = thread_hwnds[1];  // extract textbox handle
        editbox_lnames = thread_hwnds[2];
        editbox_emails = thread_hwnds[3];
        editbox_phnums = thread_hwnds[4];
        /* Read length of string entered into the text box */
        len = GetWindowTextLength(editbox_fnames);
        len = GetWindowTextLength(editbox_lnames);
        len = GetWindowTextLength(editbox_emails);
        len = GetWindowTextLength(editbox_phnums);
        /* Read string from edit box and store to name[] */

        WaitForSingleObject( mutexR, INFINITE ); // wait
                                                 // array critical section
        GetWindowText(editbox_fnames, contacts[in_cnt].fnames, len + 20);
        GetWindowText(editbox_lnames, contacts[in_cnt].lnames, len + 20);
        GetWindowText(editbox_emails, contacts[in_cnt].emails, len + 20);
        GetWindowText(editbox_phnums, contacts[in_cnt].phnums, len + 20);

        sprintf(line, "%s  %s  %s  %s", contacts[in_cnt].fnames, contacts[in_cnt].lnames,
                contacts[in_cnt].emails,contacts[in_cnt].phnums);
        in_cnt++;

        /* Simulate file I/O time over the congested network */
        random = rand() % 1000 + 1000; // random is in the range of 1000-1999
        Sleep(random); // random sleep intervals

        SendDlgItemMessage( hwnd, ID_LIST,
                            LB_ADDSTRING, 0,
                            (LPARAM)line);

        ReleaseMutex( mutexR ); // signal
                                // array critical section
```

```
        EnterCriticalSection( &CriticalSectionObject ); // file critical section

        /* Read command line information */
        file_ptr = fopen("contacts.txt", "w");  // open the contacts.txt file for
                                                //  writing only
        if( file_ptr == NULL ) // failed to open file
        {
            MessageBox(NULL, "Error Opening File", "ERROR",
                       MB_ICONINFORMATION | MB_OK);
        }

        /* create new contact line for file */
        for( out_cnt = 0; out_cnt < in_cnt; out_cnt++ )
        {
               strcpy(line, contacts[out_cnt].fnames);
               strcat(line, "\t");
               strcat(line, contacts[out_cnt].lnames);
               strcat(line, "\t");
               strcat(line, contacts[out_cnt].emails);
               strcat(line, "\t");
               strcat(line, contacts[out_cnt].phnums);
               strcat(line, "\n");

               /* Update file with added contacts */
               fputs(line, file_ptr);  // write new contact information to file
        }
        fclose(file_ptr);

        LeaveCriticalSection( &CriticalSectionObject ); // file critical section

        return result;
}

DWORD WINAPI
delethread(LPVOID hwnd_addr)
{
        int out_cnt, del_idx, random;
        HWND *thread_hwnds, hwnd; // thread_hwnds is a pointer to a handle
        DWORD result = 0;
        FILE *file_ptr;

        thread_hwnds = (HWND *)hwnd_addr;
        hwnd = thread_hwnds[0]; // extract parent window handle

        del_idx = SendDlgItemMessage( hwnd, ID_LIST,
                                      LB_GETCURSEL,
                                      (WPARAM) 0, 0);

        if( del_idx >= 0 ) // do nothing if no contact information is selected
        {
            /* Simulate file I/O time over the congested network */
            random = rand() % 1000 + 1000; // random is in the range of 1000-1999
            Sleep(random); // random sleep intervals

            WaitForSingleObject( mutexR, INFINITE ); // wait
                                                     // array critical section
            SendDlgItemMessage( hwnd, ID_LIST,
                                LB_DELETESTRING,
```

```
                                (WPARAM) del_idx, 0);

            /* Update local array of structures */
            for( out_cnt = del_idx; out_cnt < in_cnt; out_cnt++ )
            {
               contacts[out_cnt] = contacts[out_cnt+1];
            }
            in_cnt--;

            EnterCriticalSection( &CriticalSectionObject ); // file critical section

            /* Read command line information */
            file_ptr = fopen("contacts.txt", "w");  // open the contacts.txt file for
                                                    // writing only
            if( file_ptr == NULL ) // failed to open file
            {
               MessageBox(NULL, "Error Opening File", "ERROR",
                               MB_ICONINFORMATION | MB_OK);
            }

            /* create new contact line for file */
            for( out_cnt = 0; out_cnt < in_cnt; out_cnt++ )
            {
               strcpy(line, contacts[out_cnt].fnames);
               strcat(line, "\t");
               strcat(line, contacts[out_cnt].lnames);
               strcat(line, "\t");
               strcat(line, contacts[out_cnt].emails);
               strcat(line, "\t");
               strcat(line, contacts[out_cnt].phnums);
               strcat(line, "\n");
               ReleaseMutex( mutexR ); // signal
                                       // array critical section

               /* Update file with added contacts */
               fputs(line, file_ptr);  // write new contact information to file
            }
            fclose(file_ptr);
        }
        LeaveCriticalSection( &CriticalSectionObject ); // file critical section

        return result;
}

DWORD WINAPI
refreshthread(LPVOID hwnd_addr)
{
        int out_cnt, random;
        HWND *thread_hwnds, hwnd; //thread_hwnds is a pointer to a handle
        DWORD result = 0;
        FILE *file_ptr;

        thread_hwnds = (HWND *)hwnd_addr;
        hwnd = thread_hwnds[0]; // extract parent window handle

        EnterCriticalSection( &CriticalSectionObject ); // file critical section

        /* Read command line information */
```

```
	file_ptr = fopen("contacts.txt", "r");  // open the contacts.txt file for
                                                //  reading only
	if( file_ptr == NULL ) // failed to open file
	{
		MessageBox(NULL, "Error Opening File", "ERROR",
                           MB_ICONINFORMATION | MB_OK);
	}
	/* Read file line-by-line, storing each line into a seperate row in */
	/*  the names[][] string array                                       */

	WaitForSingleObject( mutexR, INFINITE ); // wait
                                                 // array critical section
	/* Empty the list box */
	SendDlgItemMessage( hwnd, ID_LIST,
                            LB_RESETCONTENT,
                            (WPARAM) 0, 0);

	/* Simulate file I/O time over the congested network */
	random = rand() % 1000 + 1000; // random is in the range of 1000-1999
	Sleep(random); // random sleep intervals

	in_cnt = 0;
	while( fgets( line, COLS, file_ptr) != NULL ) //no infromation i got from file.
	{
		sscanf(line, "%s\t%s\t%s\t%s", contacts[in_cnt].fnames,
			contacts[in_cnt].lnames, contacts[in_cnt].emails,
			contacts[in_cnt].phnums);
		sprintf(line, "%s  %s  %s  %s", contacts[in_cnt].fnames,
			contacts[in_cnt].lnames, contacts[in_cnt].emails,
			contacts[in_cnt].phnums);
		SendDlgItemMessage( hwnd, ID_LIST,
                                    LB_ADDSTRING, 0,
                                    (LPARAM)line);
		in_cnt++;
	}

	ReleaseMutex( mutexR ); // signal
                                // array critical section

	fclose(file_ptr);  //close file since all data has been extracted

	LeaveCriticalSection( &CriticalSectionObject ); // file critical section

	return result;
}
```