Windows Processes & Threads

Shuquan Li

March 14, 2014

ECT362 Homework #3

Introduction:

This project is an improvement from Homework #2. It is desired to make the program operate in a shared environment. When someone adds or deletes the contact information from the window, it will show other users what has changed. Three threads are needed in this program; one is a periodic thread to refresh the contact information which was added or deleted by others and two threads are used to add or delete the contact information.

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 two 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".

The other 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".

The program will refresh the contact information every couple minutes in order to tell other users what the newest file looks like. Every time it refreshes, 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. The program will keeping doing these unless the user exits the window and ends the program.

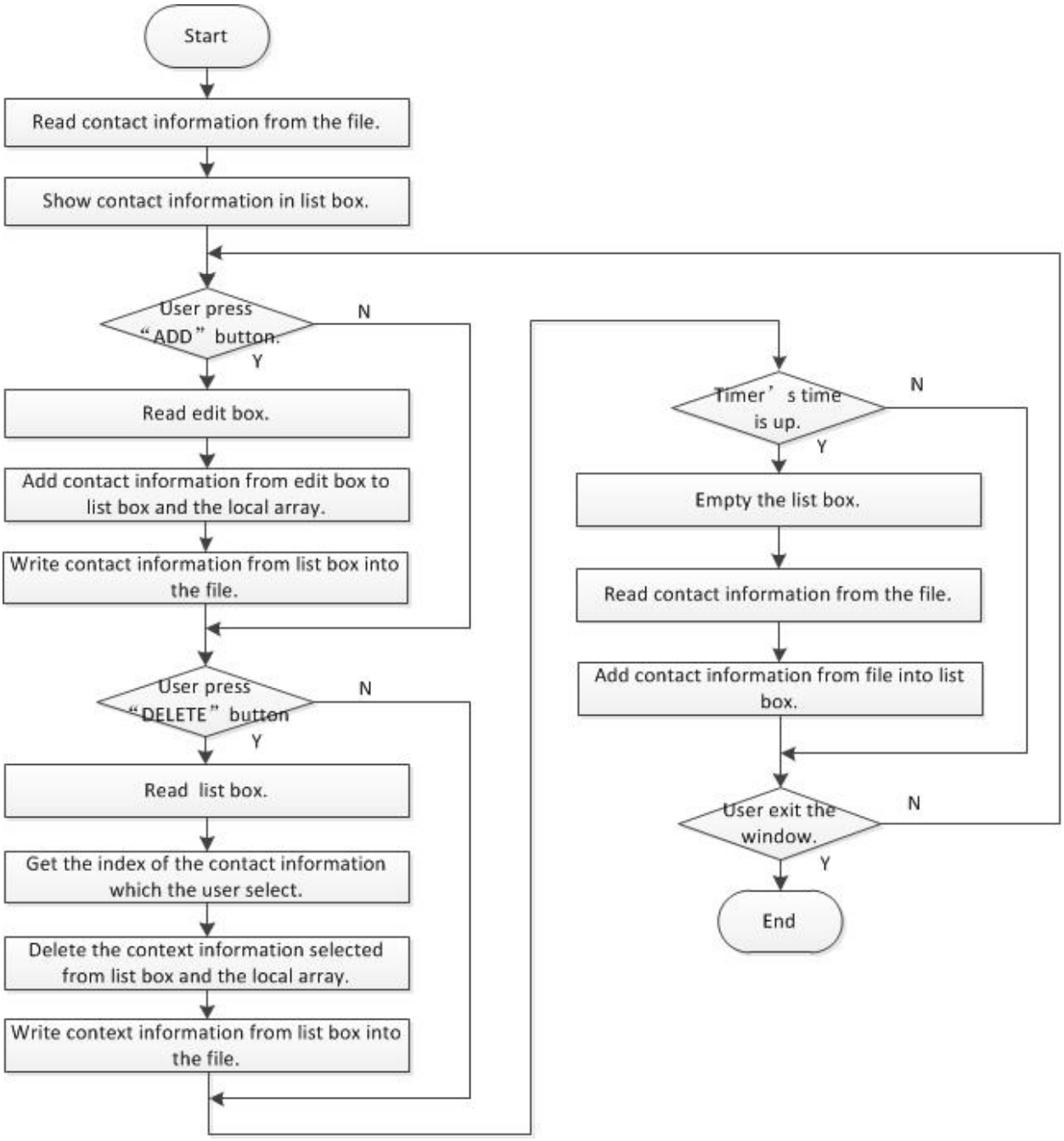

Figure 1: Flow chart of the program design.

In order to refresh contact information every couple minutes, a timer is needed, so the refresh thread is put into "case WM_TIMER", which has a timer running in that case. Putting "SetTimer" function into "case WM_PAINT" because the timer is needed once the window is created.

All the changes will show to the user in the list box when the program refreshes, so when refreshing, the list box should be empty first and then reads and shows the new contact information from the file. To empty the list box, “LB_RESETCONTENT” message is used.

Although the variable “out_cnt” is used in all the three threads, it is declared under each thread, because the three threads don’t share the value of it.

Because the contact information need to be add or delete into the local array and the contact file when “ADD” or “DELETE” button pressed, the contact information from the list box should be written into the file and update the local array every time the add thread or the delete thread runs.

In order to add the contact information into four arrays, four more handles are needed in add thread except the parent window’s handle.

If the user selects nothing, then the list box, the local array, and the file won’t change. So before deleting anything, it should to be sure that the delete index is greater than or equal to zero, otherwise, that thread won’t do anything then.

The front panel of this project is shown in Figure 2. It has an “ADD” button and a “DELETE” button, four edit boxes which let user typed first name, last name, email address and phone number, and a list box show the contact information.

Figure 2: The front panel of the project.

The original file is shown in Figure 3. The contact information in it is the same as the list box of the window.

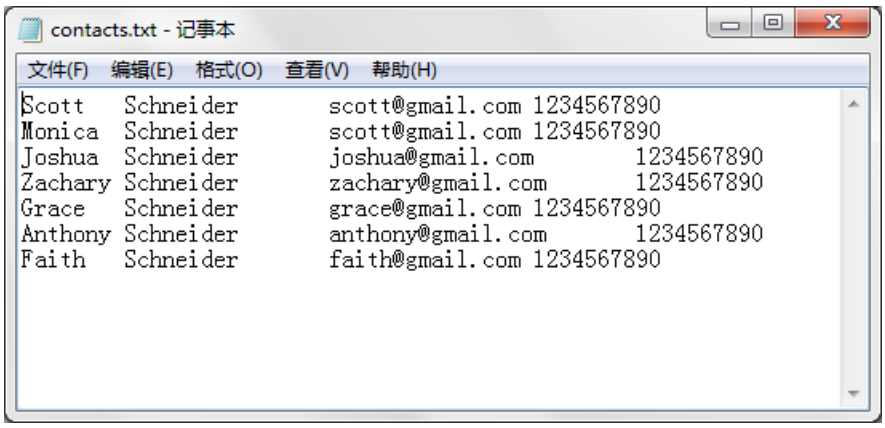

Figure 3: The original file.

Figure 4 is the screen capture of the Spy++ at the original condition. It indicates that this program has a main window named “My 1st window”, two buttons “ADD” and “DELETE”, a list box, four edit boxes, which just like Figure 1 shows.

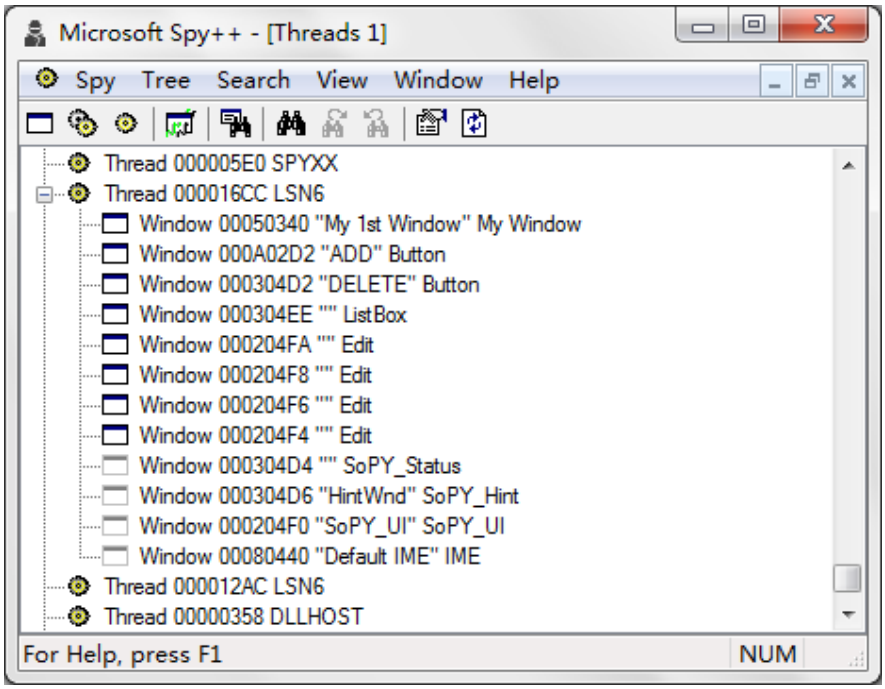

Figure 4: The screen capture of spy++ at original condition.

Figure 5 is the screen capture of the front panel when something is added into list box. The user type something in four edit boxes then press “ADD” button, the information in the edit boxes will be added into list box.

Figure 5: The front panel when “ADD” button is pressed.

Figure 6 shows what the file looks like when something is added. The contact information in the file is the same as the list box, so they are already written into file.

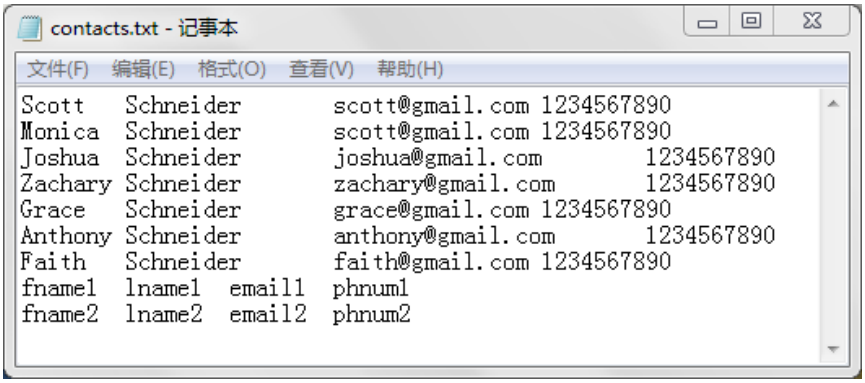

Figure 6: The file when “ADD” button is pressed.

Figure 7 is the screen capture of the Spy++ when “ADD” button is pressed. The first two threads are the original thread when I start the program, and the last thread is the add thread.

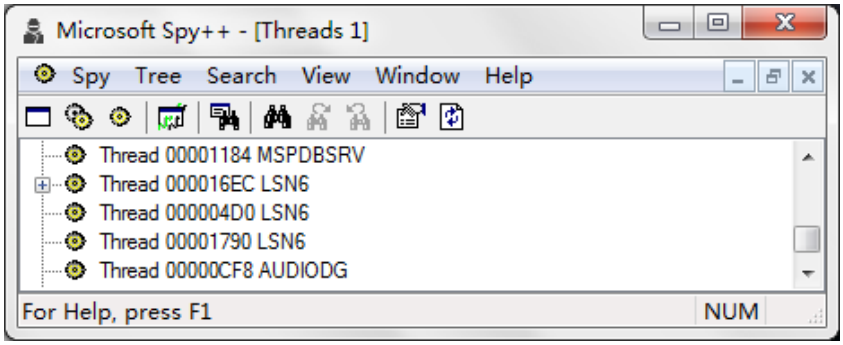

Figure 7: the screen capture of Spy++ when "ADD" button is pressed.

Figure 8 is the screen capture of the front panel when the "DELETE" button is pressed. The selected contact information removes when the user press the "DELETE" button.

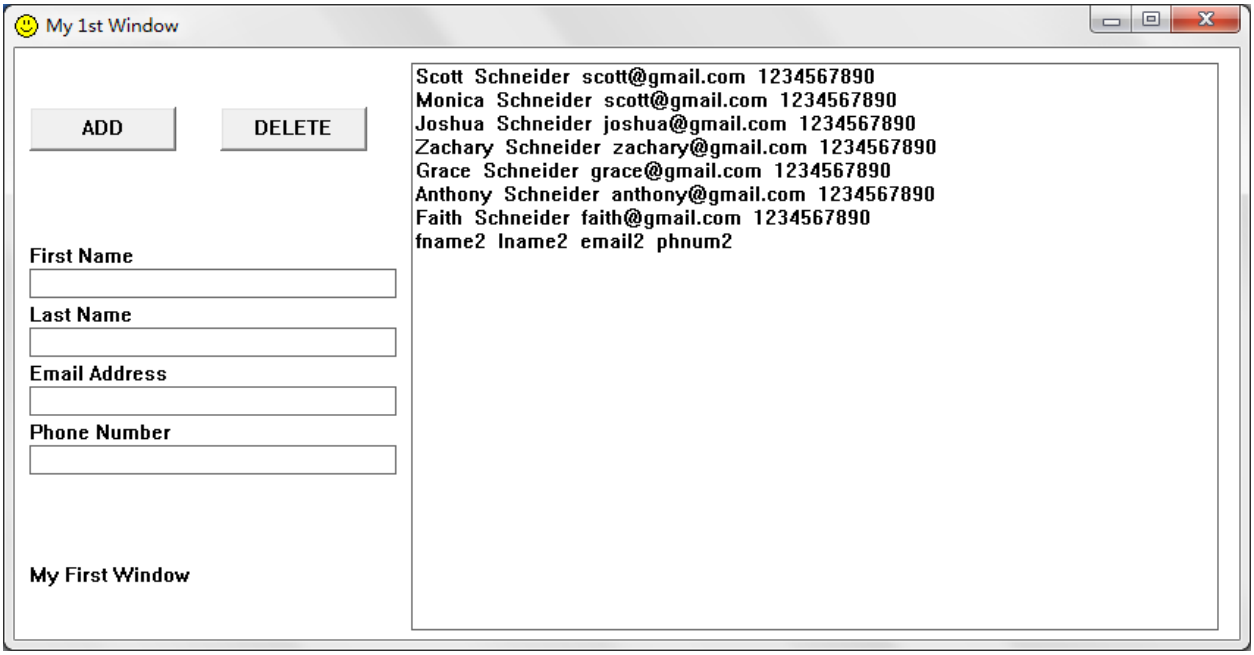

Figure 8: The front panel when "DELETE" button is pressed.

Figure 9 shows what the file looks like when "DELETE" button is pressed. The contact information in the file is the same as the list box.

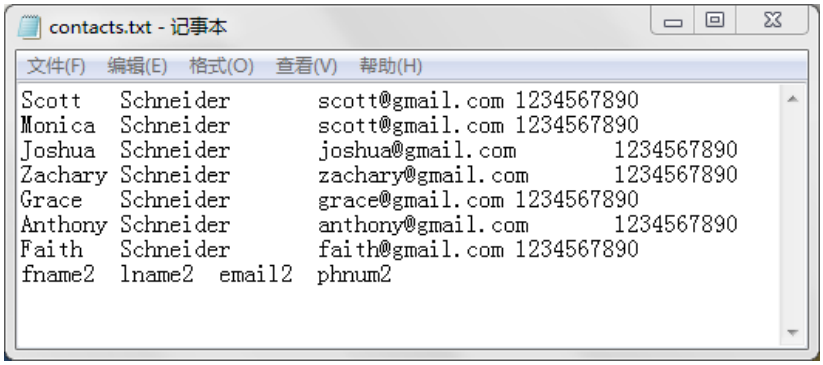

Figure 9: The file when “DELETE” button is pressed.

Figure 10 is the screen capture of the Spy++ when “DELETE” button is pressed. The first two threads are the original thread when I start the program just like Figure 7, and the last thread is the delete thread.

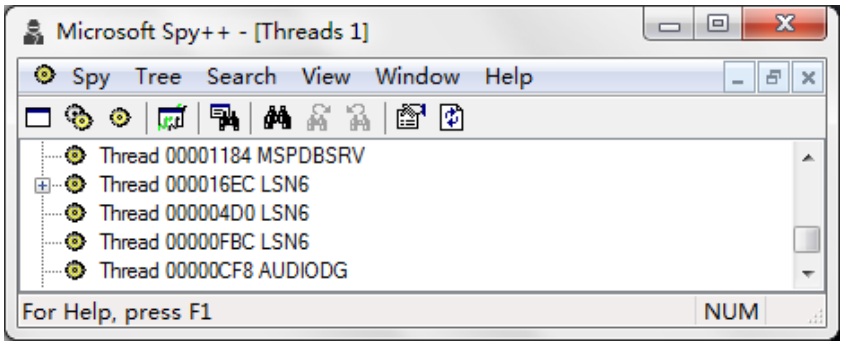

Figure 10: the screen capture of Spy++ when “DELETE” button is pressed.

Figure 11 is the screen capture of the Spy++ when the program refreshes. The last thread is the refresh thread.

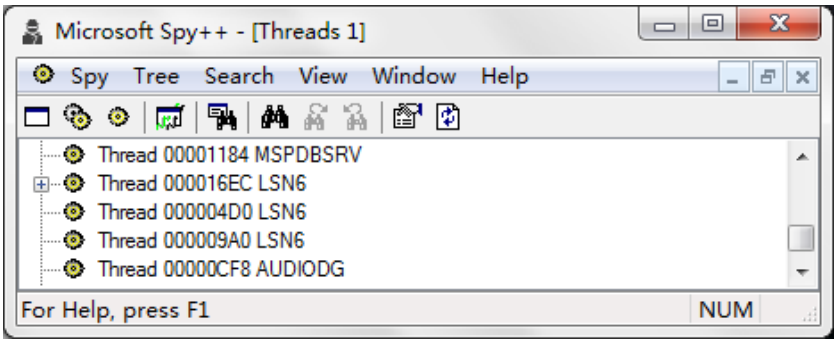

Figure 11: the screen capture of Spy++ when refresh.

When the program refreshes, it will first delete all the information from list box. Figure 12 shows the empty list box when the program refreshes.

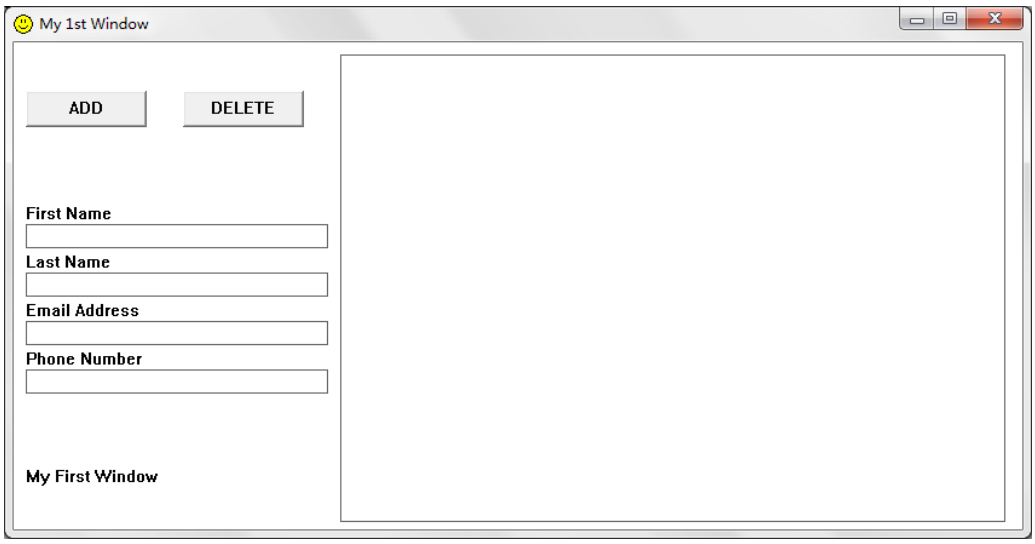

Figure 12: The list box of the front panel is empty.

QUESTIONS:

1. How is the memory structure associated with sibling threads' able to help the program operates?

These threads share the same address space, so they can edit the same file and the same local array. All the changes will be saved into file.

2. Debugging and validating the operation of multithreaded applications is very challenging. Using the Windows tools discovered in Homework #1, document how one can discern if the program is working correctly (right number of threads and thread creation/deletion sequencing). Provide screen captures to support all conclusions.

Figure 7 is the screen capture of add thread. The program has two basic threads in Spy++ which are "Thread 000016EC LSN6" and "Thread 000004D0 LSN6", and when the "ADD" button is pressed, another thread – "Thread 00001790 LSN6" – creates, so this is the add thread for this program.

Figure 10 is the screen capture of delete thread and when the "DELETE" button is pressed, "Thread 00000FBC LSN6" creates and "Thread 00001790 LSN6" is disappear. That is the delete thread creates and the add threads ends.

Figure 11 is the screen capture of refresh thread and "Thread 000009A0 LSN6" is the refresh thread. Now, both "Thread 00001790 LSN6" and "Thread 00000FBC LSN6" disappear, only the refresh thread works.

3. According to Windows the new program is comprised of several objects (windows) for which it must manage. Using the Windows tools used in Homework #1, document how one is able to determine the object hierarchy as seen by Windows, providing all supportive screen captures. Figure 13 is the screen capture of windows in Spy++. It indicates that "'My 1st Window' My Window" is the parent window of this project and it has seven child windows which are two buttons, a list box, and four edit boxes.

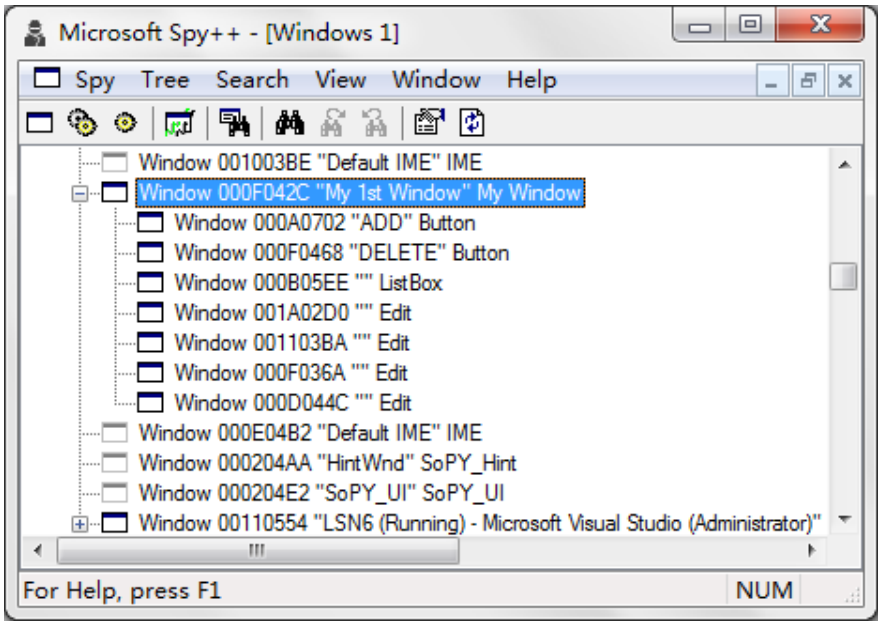

Figure 13: The screen capture of windows.

4. Manipulating the threads in various sequences and rates can cause errors to occur in how data is handled with respect to the shared resources (local arrays and file). Document any errors that result in your program as a result of using threads, looking explicitly for data inconsistencies.
If let the delete thread has a ten seconds time delay, it may delete the wrong line because the refresh thread works at the same time.
If the timer's time is too short, the user can't delete things because the refresh thread runs too fast.

Conclusion

It is desired to put add, delete and refresh these three functions into three threads so that they can run concurrently.

In doing this project, the biggest difficulty is the delete command. If the user selects nothing, nothing is expected to be deleted. So the delete command will be executed only when the user selects something.

In order to see the three threads work in Spy++, “Sleep” function is used in each thread for refreshing the Spy++.

References

WM_TIMER message. Microsoft Windows Dev Center - Desktop. nd. np. Web. Mar. 14, 2014
SetTimer function. Microsoft Windows Dev Center - Desktop. nd. np. Web. Mar. 14, 2014
Using Timers. Microsoft Windows Dev Center - Desktop. nd. np. Web. Mar. 14, 2014
LB_RESETCONTENT message. Microsoft Windows Dev Center - Desktop. nd. np. Web. Mar. 24, 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()
#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];

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

     /* 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 Number", 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

    /* 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  */
    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;
       HWND *thread_hwnds, hwnd, editbox_fnames, editbox_lnames, editbox_emails,
             editbox_phnums; //thread_hwnds is a pointer to a handle
```

```
        DWORD result = 0;
        int out_cnt;
        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[] */
        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++;
        SendDlgItemMessage( hwnd, ID_LIST,
                            LB_ADDSTRING, 0,
                            (LPARAM)line);

        /* 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);
        return result;
}

DWORD WINAPI
delethread(LPVOID hwnd_addr)
{
```

```
        HWND *thread_hwnds, hwnd; //thread_hwnds is a pointer to a handle
        DWORD result = 0;
        int out_cnt;
        int del_idx;
        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
        {
            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--;

            /* 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);
        }
        return result;
}

DWORD WINAPI
refreshthread(LPVOID hwnd_addr)
{
```

```
        HWND *thread_hwnds, hwnd; //thread_hwnds is a pointer to a handle
        DWORD result = 0;
        int out_cnt;
        FILE *file_ptr;

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

        /* Empty the list box */
        SendDlgItemMessage( hwnd, ID_LIST,
                            LB_RESETCONTENT,
                            (WPARAM) 0, 0);

        /* 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 = 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++;
        }
        fclose(file_ptr);  //close file since all data has been extracted
        return result;
}
```