Joe Student

CSIS 123

LAB 1: Telephone Number Database and Autodialer

January 1, 1980

Due Week 16

Instructor: Tom Wheeler

OPERATIONAL SIGN OFF____

FINAL GRADE____

"THIS IS THE ORIGINAL WORK OF JOE STUDENT"

Joe Student

Introduction:

This project implements a simple telephone directory and "autodialer" capable of making outgoing telephone calls. It also monitors caller ID for incoming calls, and allows the user to semi-automatically add incoming caller information to the directory.

CLEARLY MARK each section of the report.

Theory of Operation:

The program begins in the `main()` function, which simply calls the `run()` function which is the main body of the program. All program activity is initiated from within the `run()` function.

The `run()` function performs the following initial activities:

- It launches a Windows thread to monitor caller ID activity in the background. This thread, implemented in the function `callerID()`, continuously listens on data from COM4, which is a serial port mapped to the physical caller ID decoding device connected to the user's telephone line.
- It displays a menu by calling the `menu()` function.
- It waits for the user's command using `gets()`, then executes the command by running one of the following functions, based on what command the user has entered: `addNumber()`, `browse()`, `search()`, `setOptions()`, `reviewCalls()`, `dialNumber()`, `finalExit()`.

The program's action takes place within eight functions:

`callerID()`: This function implements a thread that continuously waits for data on COM4, interprets that data (which is the calling number delivery or CND text from the telephone company), and when conditions warrant, calls `addCallerIDEntry()` to register a new incoming call.

`addNumber()`: This function gets the user's data, validates the telephone number, and adds it to the internal database, which is implemented as an array of Caller structures.

`browse()`: This function interacts with the user and allows them to see the contents of the internal database (array of `Caller` structures).

`search()`: The search command is implemented in this function. The routine starts by requesting the search information from the; it then walks through the internal database of Caller structures to see if it locates any matches.

`setOptions()`: This command is a stub and is not implemented in this version of the program.

`reviewCalls()`: This command pulls data from the same temporary array that `callerID()` places data into during each incoming call. This data is displayed for the user. The user has the option of making each entry permanent within the database during this function.

`dialNumber()`: This function allows the user to manually type in either a directory entry's name, or an actual phone number; it calls `dialNumberCOM4()` to actually dial the number.

`finalExit()`: This function is called when the program is about to exit. If the database contents have changed, they are updated on the disk so that they'll be available on the next run. Also, the subthread (`callerID()` function) is terminated by setting its global run flag, held in global variable `nCallerIDThreadRun`, to zero.

Sample Run of Telephone Autodialer

```
Welcome to the Telephone Autodialer

There are no entries in the phone book.

1) Add a number to the phone book
2) Browse the telephone book
3) Search for a name or number in the book
4) Set Caller ID Options
5) Review the last 50 telephone calls
6) Dial a Number
7) Exit the program

What do you want to do (1-7)? 1

Manual Number Entry

Name? Joe Cool
Telephone Number? 844-1212
Area Code? 816
Address? 12345 College Blvd
City? Overland Park
State? KS


Welcome to the Telephone Autodialer

There is 1 entry in the phone book.

1) Add a number to the phone book
2) Browse the telephone book
3) Search for a name or number in the book
4) Set Caller ID Options
5) Review the last 50 telephone calls
6) Dial a Number
7) Exit the program

What do you want to do (1-7)? 2


Database Entry 1

James Brown    (816) 844-1212
12345 College
Overland Park  KS

[+] Next     [-] Previous
[D] Dial     [DEL] Delete
             [ESC] Exit to menu
```

Telephone Autodialer Program Listing

```
//
// Program Name: Telephone Autodialer
// Author: Joe Student
// Date: January 1, 1980
//
// Revision history: 1/10/80 -- Fixed overrun p
//                   1/12/80 -- Modified communicati
//                              modem compatibility.


/////////////////////////////////////////////////////////////////////
// Function: OnPaint()
// Description: This function responds to the WM_PAINT message that is
//              generated when a previously-covered portion of the
//              window is uncovered. The global background bit-map is
//              also repainted.
//
// Parameters passed: None, the OS gives information through CpaintDC
// Parameters returned: None. The window is repainted.
/////////////////////////////////////////////////////////////////////

void CSignalWindow::OnPaint()
{

//
// The following instruction forces the ENTIRE client area
// to always be repainted. It MUST take place before the c
// of the CPaintDC so that the CPaintDC PAINTSTRUCT will be fi
// with proper information.
//
// This approach is used because calculating the bounding rectangle
// of signal waveforms might be time consuming, so the compromise
// here is to always repaint the entire window.
//
// Failure to repaint the window properly could leave pieces of
// the old waveform in the client area, which invalidates the
// background!
//

InvalidateRect(NULL);

//
// Build a CPaintDC (BeginPaint() called here)
//

CPaintDC dc(this);     // device context for painti

CRect aRect;           // Will hold the rectangle c
int nResult;

GetClientRect(&aRect); // Get dimensions of our "play" area
                       // (Needed for painting the window background)

m_bIsPainting = TRUE;  // Let draw routine know we are painting,
                       // since it may be called upon during the
                       // painting process by a multithreaded
                       // application.

//
// If we had created a bit-mapped background before, delete it, since
// we're creating a new one.
//

if (m_pBackground)
       delete m_pBackground;

//
// Paint the gridlines and produce the scale legend
```

Your name <u>must</u> appear at the top of all program listings. It's also a good idea to keep a revision history, as well as any other related information.

Header comment tells what purpose of function is. Many people use the "//" characters to make a box.

The body of this program isn't an autodialer, however, it is an example of good commenting style. Note that each *concept* or *thought* is <u>thoroughly explained</u>.

Don't worry if the code looks complicated. The style is what is important!

Spacing program statements gives a clean appearance, making reading (and understanding!) much easier.

```
//

PaintNewBackGround( &dc, m_WindowBackgroundColor,
                                             m_ScaleColor,  aRect );

//
// Copy the window frame client area into the m_pBackground
// bit map (which must be created first in the same size
// dimensions as the window client area)
//

m_pBackground = new CBitmap;

//
// When creating the CBitmap object that will hold t
// image, its characteristics must be made compatibl
// DEVICE CONTEXT "memdc", which is forced to be com
// the drawing DC by "CreateCompatibleDC()"
//

int nPlanes = dc.GetDeviceCaps( PLANES );
int nBitPixel = dc.GetDeviceCaps( BITSPIXEL );

nResult = m_pBackground->CreateBitmap( m_cx, // Width
                                              m_cy,  // Height
                                              nPlanes,             // Planes
                                              nBitPixel,    // nBitCount
                                              NULL); // const void* lpBits

// create an in-memory device context

CDC memdc;
memdc.CreateCompatibleDC(&dc);
memdc.SelectObject( m_pBackground );
//
// Copy our newly-generated background to the bitmap selected into
// the memory device context "memdc"
//
nResult = memdc.BitBlt(       0,0,           // (x,y) of destination
                              m_cx,m_cy,     // (width,height) of destination
                              &dc,           // copy FROM this dc
                              0,0,           // copy FROM this (x,y) in src dc
                              SRCCOPY);      // copy mode



//
// Try painting the last waveform we were passed. If none,
// don't paint anything, since owner hasn't sent us any wave
// data yet.
//

if ((m_pOldWaveform) && (m_bIsDrawing == FALSE))
        {
        m_bIsPainting = FALSE; // Must reset this or drawing routine
                                                    // won't do anything

        DrawWaveform(&dc, (*m_pOldWaveform),
                        m_DCOffset , m_color , m_FullScale);
        }

//
// Let others know that we're done painting so that they can resume
// execution
//
m_bIsPainting = FALSE;

}
```

Related statements (or in this case, parts of a statement) should be indented appropriately and consistently.

The indentation of this block of code (marked by the "{ }" characters) makes it clear what belongs -- and what doesn't!

Conclusion

In this experiment an autodialing program with Caller ID was implemented. This was our second program that used Win32 threading features, and we saw that debugging a program that is doing more than one task at a time could be a challenging effort.

There were a couple of hard to locate bugs. First, if a user was browsing the caller ID information while a call came in, sometimes the data would get garbled in the caller ID array. We discovered that two sections of a program shouldn't attempt to access memory data without proper coordination! We solved this by adding a mutex object to the caller ID data array, so that only one thread could access it at a time.

The second bug involved reading data from the serial port. Occasionally the subthread `callerID()` would stop registering any further incoming calls. This was very puzzling. We eventually discovered that every so often, a data error occurred in the CND data stream from the telephone company, and our `callerID()` function wasn't robust enough to recover. Our solution was simple: `callerID()` waits in a loop for 10 seconds for each CND data packet. After the wait is up, we validate the incoming data in the buffer; if it's incomplete, we simply erase it, and return to the top of the loop.