DELVING INTO THE WORLD OF INTERNET COMMUNICATION WITH WINDOWS APIS

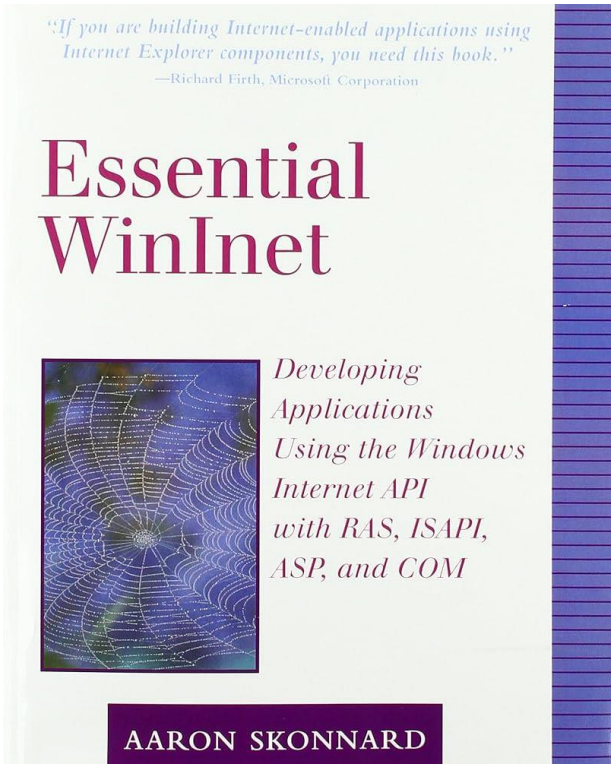
The Internet, a vast network of interconnected computers spanning the globe, has revolutionized personal computing, enabling seamless communication, information access, and resource sharing.

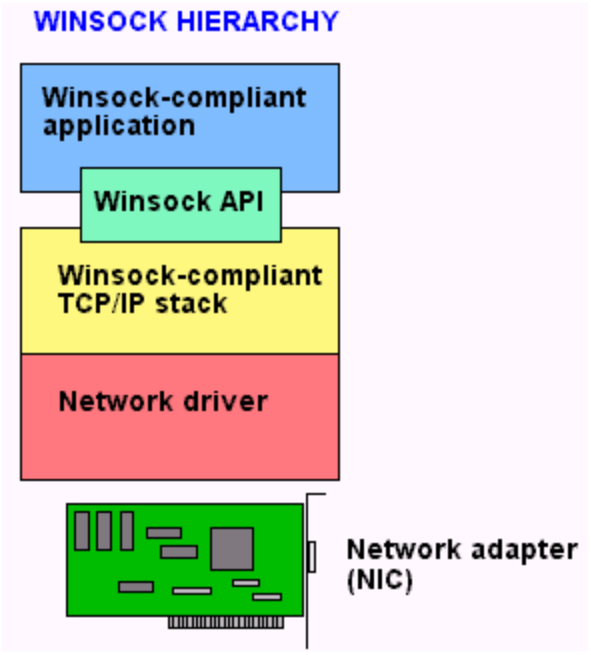
While dial-up information services and electronic mail systems existed previously, their character-based interfaces and isolated nature presented limitations. Each information service required a separate connection and login credentials, and email exchanges were restricted to users within the same system.

The era of isolated information services has given way to a unified Internet experience, brought about by the ubiquity of high-speed connectivity and the adoption of open communication protocols.

With a single Internet connection, individuals can now communicate with anyone worldwide, and the World Wide Web, with its hypertext structure, multimedia elements, and interactive features, has expanded the scope and accessibility of online information.

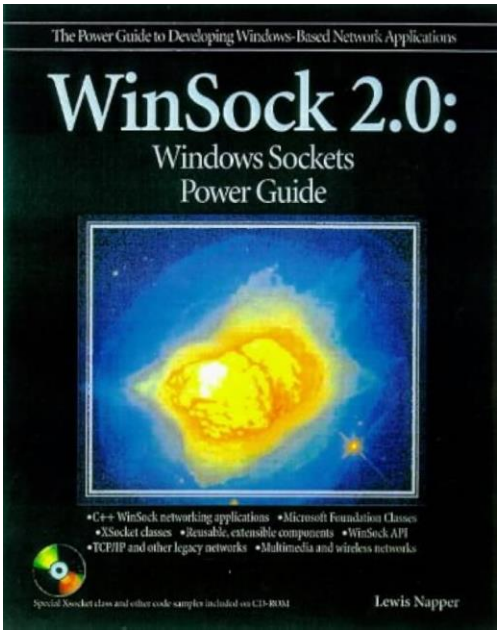
To harness the power of the Internet in Windows applications, developers can leverage various APIs (Application Programming Interfaces) that provide a structured approach to communication and data exchange. Two prominent APIs that stand out for their simplicity and effectiveness are Windows Sockets (WinSock) and Windows Internet (WinInet).



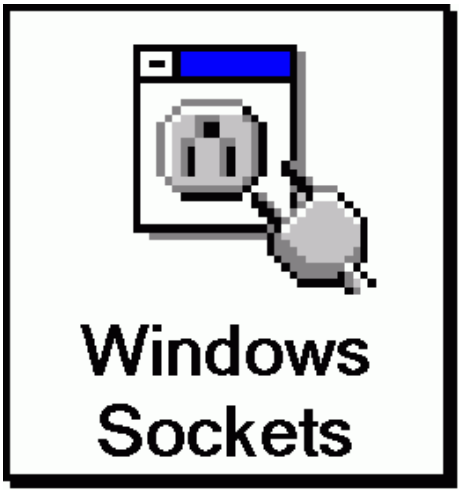


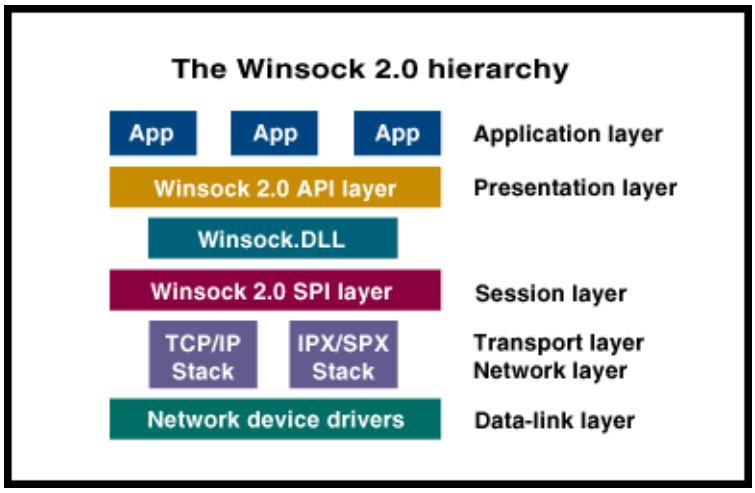
Winsock: The Foundation Of Internet Communication

WinSock, the Windows Sockets API, forms the cornerstone of Internet programming in Windows. It provides a standardized set of functions for creating network sockets, establishing connections, sending and receiving data, and managing network errors.



WinSock simplifies the process of interfacing with the underlying network protocols, enabling developers to focus on higher-level application logic rather than the intricacies of network-level communication.





Wininet: Simplified File Transfer And Web Access

WinInet, the Windows Internet API, extends the functionality of WinSock by providing a higher-level abstraction for common Internet tasks, such as file transfer through File Transfer Protocol (FTP) and web browsing through HTTP (HyperText Transfer Protocol).

WinInet simplifies the process of downloading and uploading files, navigating web pages, and interacting with web services, making it particularly suitable for web-based applications.

Selecting the Right API

The choice between WinSock and WinInet depends on the specific requirements of the application.

For complex network communication involving custom protocols or low-level data handling, WinSock offers greater control and flexibility.

WinInet is better suited for applications that primarily involve downloading and uploading files, accessing web pages, or performing general Internet-related tasks.

WINDOWS SOCKETS: A COMPREHENSIVE OVERVIEW

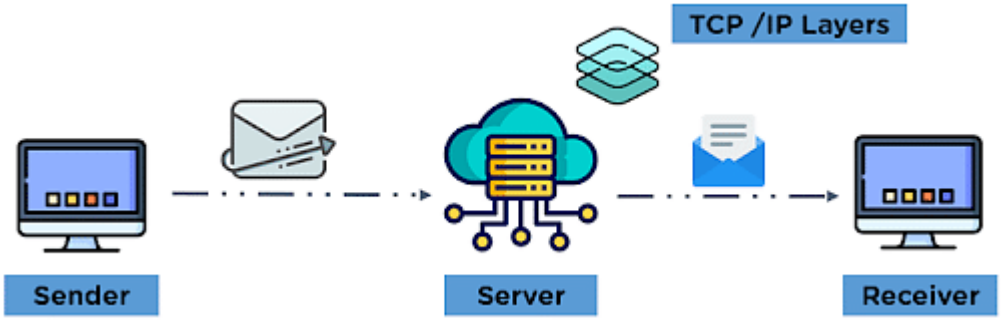
Windows Sockets (WinSock), an Application Programming Interface (API), provides a standardized and efficient method for network programming in Windows operating systems. It serves as a foundational layer for building applications that communicate over the Internet or local networks.

Socket Concept and TCP/IP Connection

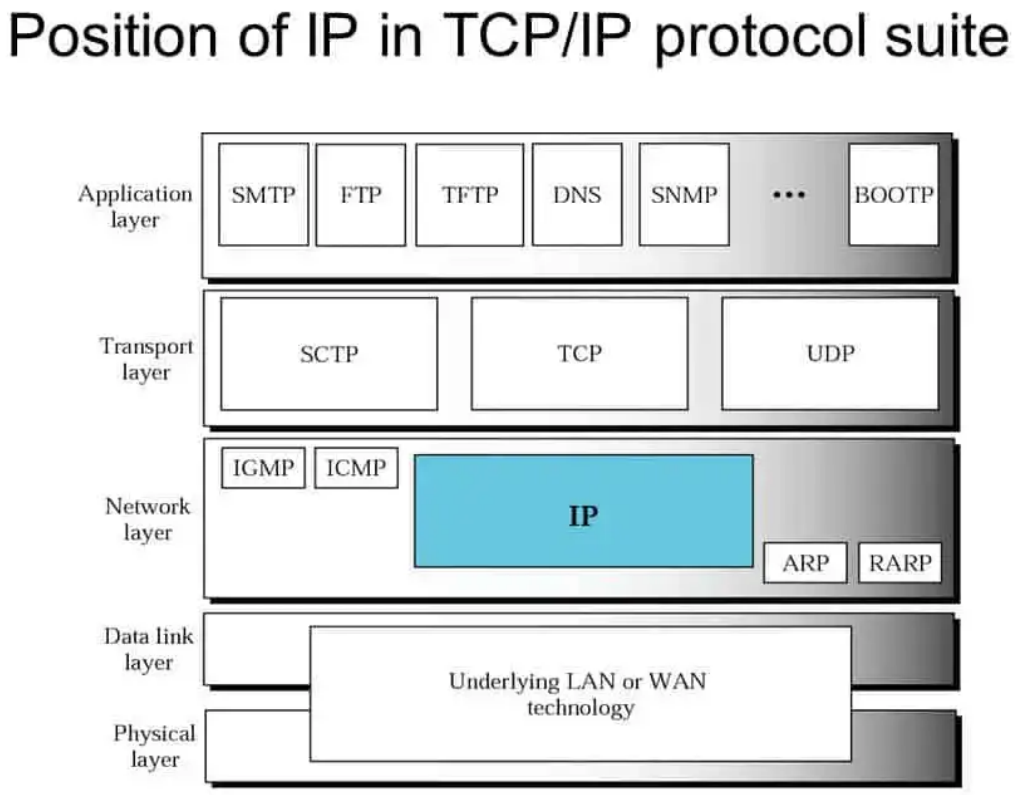
The concept of sockets originated at the University of California, Berkeley, as a way to integrate network communication capabilities into the UNIX operating system. This API, known as the "Berkeley socket interface," has since become the de facto standard for network programming across various operating systems.

Sockets operate in conjunction with the Transmission Control Protocol/Internet Protocol (TCP/IP), the widely adopted set of protocols that govern Internet communications. TCP/IP comprises two primary layers:

Internet Protocol (IP): IP handles the addressing and routing of data packets across the network. It fragments data into smaller packets, assigns each packet with a destination address, and transmits them across the network.

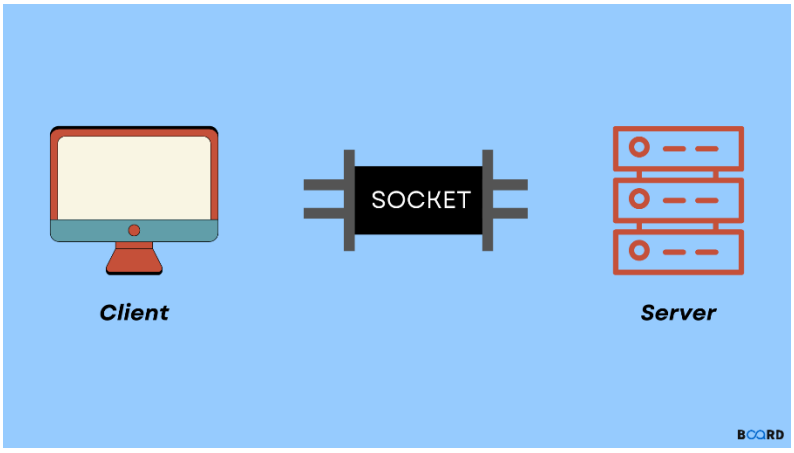


Transmission Control Protocol (TCP): TCP provides reliable data transfer between applications. It establishes a connection between two applications, ensuring that data is transmitted in an error-free and ordered manner.

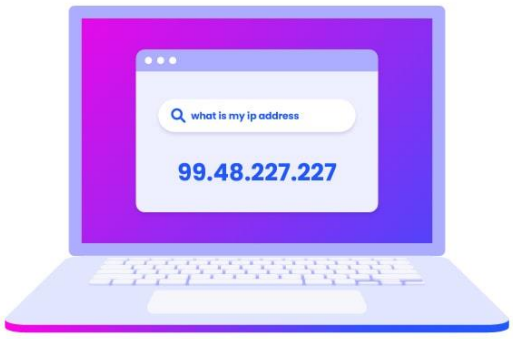


Socket Address and Communication Endpoints

In the context of TCP/IP communications, a socket represents a communication endpoint, identified by a unique combination of an IP address and a port number.



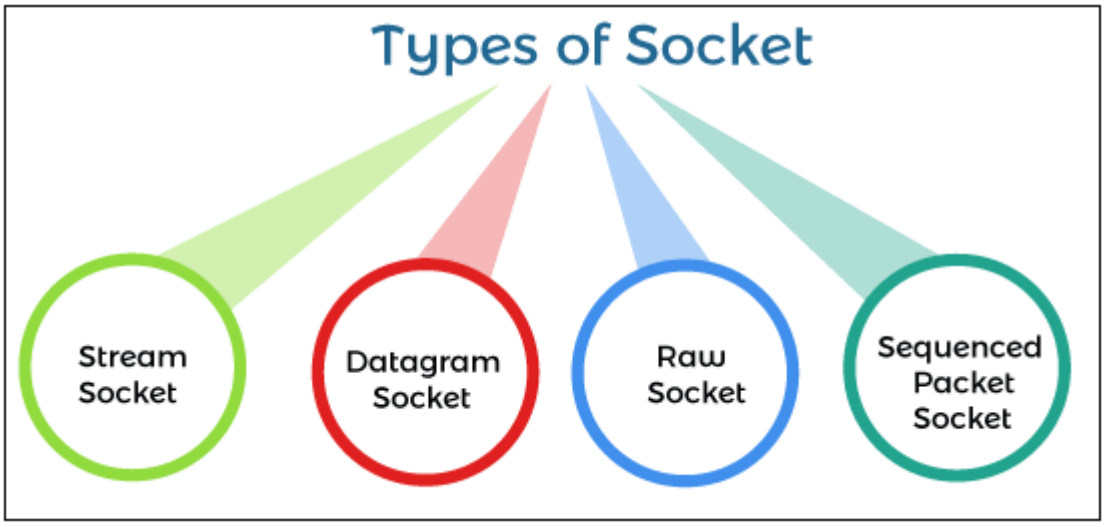
The IP address, typically represented in dotted-quad notation (e.g., 209.86.105.231), identifies the specific network device or server involved in the communication. The port number, a numerical identifier, further specifies the application or service running on that device or server.



IPXO

TYPES OF SOCKETS AND CONNECTION ESTABLISHMENT

WinSock supports two primary types AND two secondary types of sockets:



- Stream sockets: Provide a reliable, connection-oriented communication channel, similar to a telephone call. Data is transferred in a continuous stream, ensuring error-free delivery.
- Datagram sockets: Offer an unreliable, connectionless data transfer mode. Data is sent as discrete packets, each with its own destination address. Datagram sockets are often used for low-latency and high-throughput applications.
- Raw Sockets: Raw sockets provide direct access to lower-level communication protocols, allowing developers to manipulate network packets at the raw IP level. This enables advanced network programming tasks such as packet filtering, packet injection, and development of custom network protocols. Unlike stream and datagram sockets, which handle data transfer in predefined formats, raw sockets allow developers to directly construct and manipulate IP packets, including headers, options, and payload data. This level of control is essential for advanced networking tasks that require granular control over network traffic.
- Sequenced Packet Sockets: Also known as sequenced datagram sockets, offer a middle ground between stream and datagram sockets. They provide a connection-oriented data transfer mechanism, similar to stream sockets, but with the ability to maintain packet boundaries. This allows for more efficient data transfer with low latency and high throughput. They are often used in applications that require reliable data delivery while also maintaining packet integrity. They are particularly useful for streaming applications where the order of packets is crucial, such as video conferencing or real-time audio transmission.

Feature	Raw Socket	Stream Socket	Datagram Socket	Sequenced Packet Socket
Connection	Connectionless	Connection-oriented	Connectionless	Connection-oriented
Data Transfer	Packets	Stream	Packets	Packets
Reliability	Unreliable	Reliable	Unreliable	Reliable
Efficiency	High	Moderate	Low	Moderate
Usage	Advanced network programming tasks, packet filtering, packet injection, custom protocols	Web browsing, file transfers, email	Network gaming, streaming applications	Streaming applications, large-data transfers, applications requiring precise data sequencing and ordering

To establish a connection using WinSock, applications typically follow these steps:

- Create a socket: Allocate a socket object using the appropriate socket function, specifying the desired socket type and protocol.
- Bind the socket: Associate the socket with a specific IP address and port number, which determines the communication endpoint.
- Connect or listen: For stream sockets, applications can connect to a remote server using the connect function. For datagram sockets, servers listen for incoming connections by using the listen function.

WinSock Functions and Data Structures

WinSock provides a comprehensive set of functions for various network operations, including:

- Create and manage sockets: Functions for creating, binding, and closing sockets.
- Data transfer: Functions for sending and receiving data, both in stream and datagram mode.
- Address manipulation: Functions for working with IP addresses and port numbers.
- Error handling: Functions for detecting and handling network errors.

WinSock also defines various data structures, such as sockaddr structures for specifying socket addresses and WSADATA structure for storing WinSock version and API information.

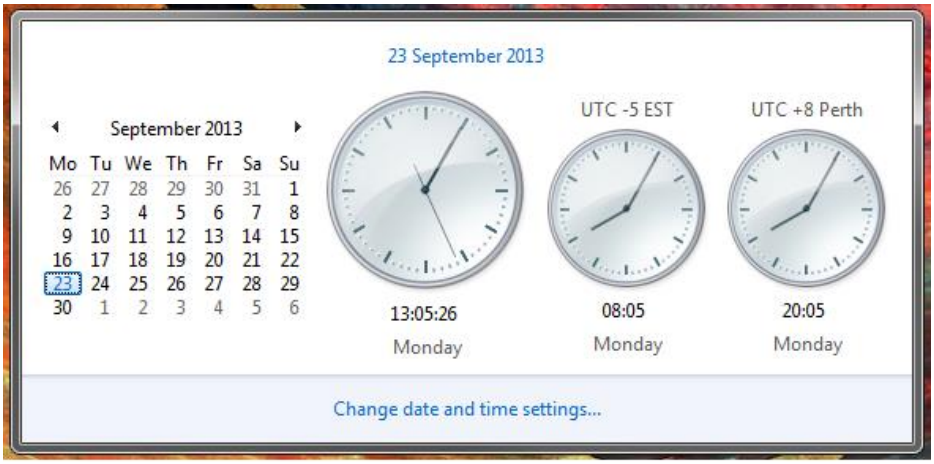
Applications of WinSock

WinSock is widely used in various applications, including:

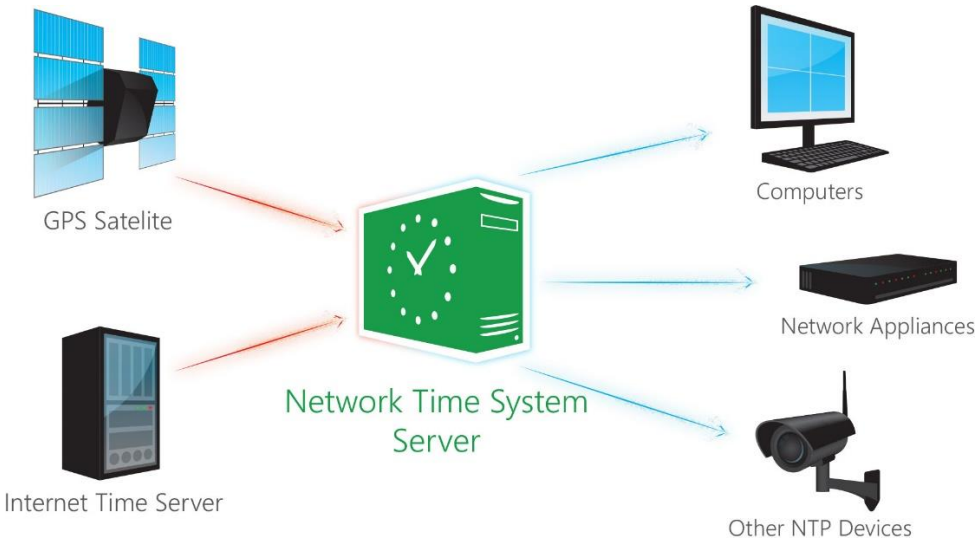
- Web browsers: To connect to web servers and retrieve web pages.
- Email clients: To send and receive emails from remote servers.
- File transfer protocols: To upload and download files from remote servers.
- Network games: To facilitate online multiplayer gaming experiences.
- Remote access tools: To establish remote connections to other computers for administration or support purposes.

ACQUIRING ACCURATE TIME USING NETWORK TIME SERVICES

Precise timekeeping is essential for various applications, including maintaining accurate databases, coordinating synchronized events, and ensuring data integrity. Traditionally, timekeeping was reliant on local clocks, prone to inaccuracies and drift over time. To address this, the concept of network time services emerged, providing synchronized time information from authoritative sources.

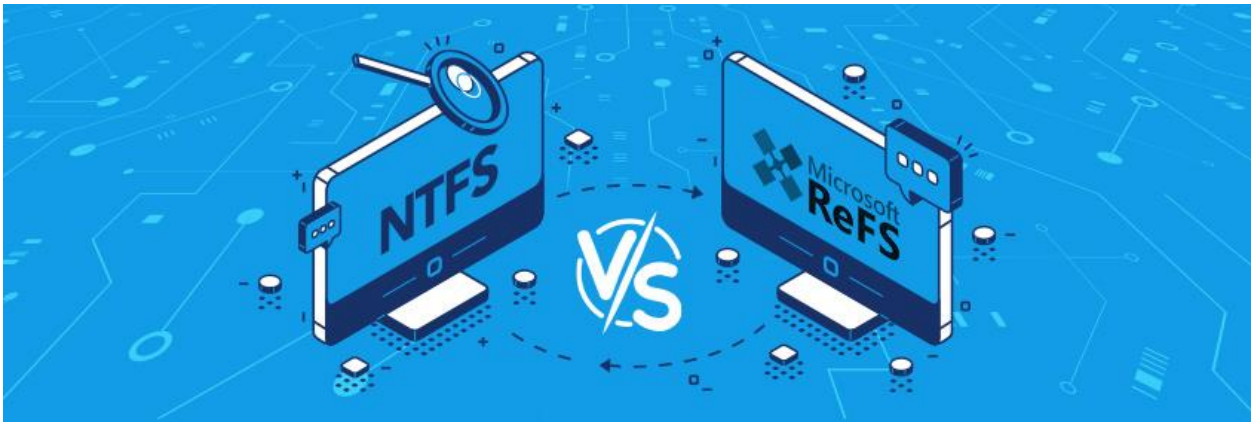


Network time services utilize standardized protocols to exchange time synchronization information across networks. These protocols are typically implemented on dedicated servers maintained by trusted organizations, such as the National Institute of Standards and Technology (NIST) in the United States.



NIST Network Time Service

The NIST Network Time Service (NTFS) is a widely used protocol for obtaining accurate time information over the Internet. It utilizes the Time Protocol (RFC-868), which defines a simple method for exchanging time information between clients and servers.



To access the NTFS, clients connect to a designated server on port 37, the well-known port for the Time Protocol. Upon establishing a connection, the server sends a 32-bit integer representing the number of seconds since midnight on January 1, 1900, in Coordinated Universal Time (UTC).

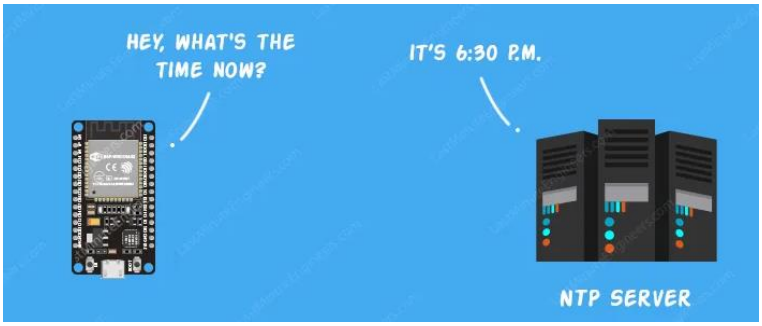
Benefits of Using Network Time Services

Network time services offer several advantages over relying on local clocks:

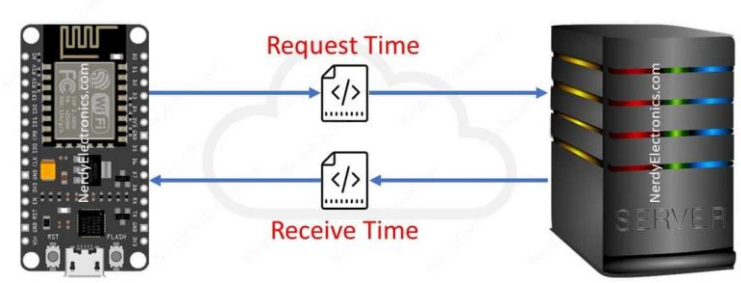
Accurate Timekeeping: NIST servers are maintained with the highest precision, ensuring accurate time synchronization across the network.

Reduced Drift: Network time services minimize time drift compared to local clocks, maintaining consistent timekeeping over extended periods. Most electronic clocks rely on a quartz crystal oscillator to generate their internal "tick" rate. However, these crystals aren't perfect and their frequency can slightly fluctuate over time due to temperature changes, aging, and other environmental factors. This slight fluctuation translates to gradual time drift.

Issues in the software managing the clock can also lead to drift. Bugs or misconfigurations can affect the interpretation of the oscillator's signal, leading to inaccuracies in the displayed time. When relying on network time services, even minimal network delays in synchronizing with the server can contribute to drift. The longer it takes to receive the reference time, the higher the potential for discrepancy.



Seamless Integration: Network time services seamlessly integrate with various applications and systems, ensuring consistent timekeeping across the network.



Reduced Maintenance: Network time services eliminate the need for manual time adjustments on individual devices, streamlining system maintenance.

Network time services help minimize drift in several ways:

- ✓ Regular synchronization: By synchronizing with a highly accurate reference source like a NIST server frequently, network time services can correct any accumulated drift. This periodic update keeps the local clock aligned with the reference time.
- ✓ High-precision servers: NIST servers themselves are meticulously maintained and synchronized with atomic clocks, providing the most accurate possible reference time. This reduces the potential error introduced from the initial time source.
- ✓ Reduced reliance on internal clock: By relying on external reference time, network time services lessen the dependence on the local clock's potentially unstable oscillator. This reduces the impact of individual clock drift on the overall timekeeping.

NETTIME PROGRAM

The NETTIME program is a Windows application that synchronizes the system clock with an Internet time server. It does this by connecting to a time server and then receiving the current time from the server. The program then changes the system clock to reflect the time received from the server.

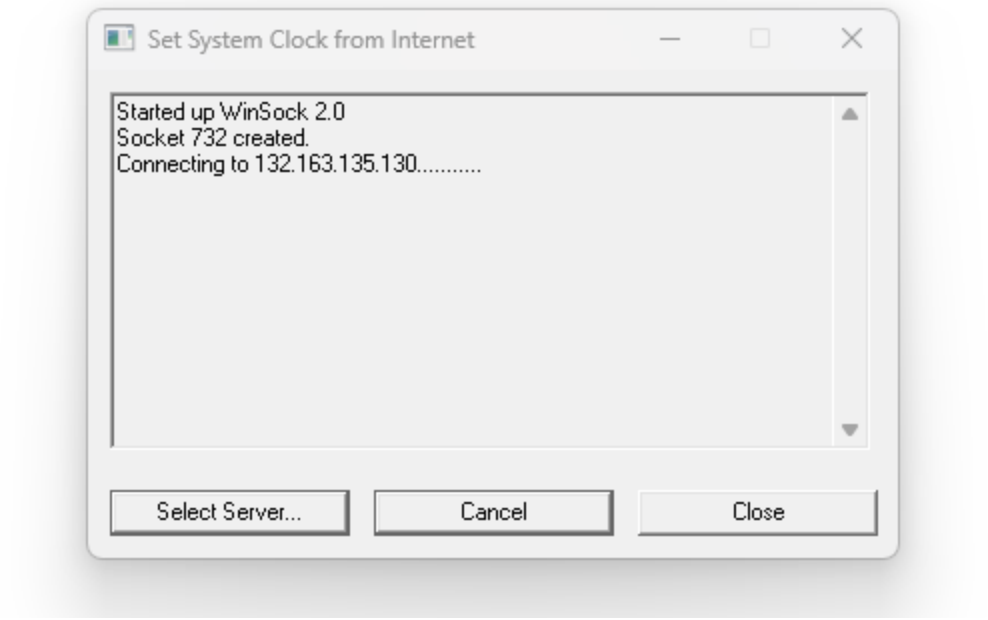
Here is a more detailed explanation of what the NETTIME program does:

- Creates a socket: The program first creates a socket, which is a connection point between the program and the network. This socket is used to send and receive data from the time server.
- Connects to a time server: The program then connects to a time server using the IP address of the server. The NETTIME program uses the time server 132.163.135.130, which is the time server for the National Institute of Standards and Technology (NIST).
- Receives the current time: Once connected to the time server, the program receives the current time from the server. The time is sent in the form of a 32-bit integer, which represents the number of seconds since midnight on January 1, 1900.
- Changes the system clock: The program then changes the system clock to reflect the time received from the server. This is done by calling the SetSystemTime function, which sets the system time to the specified time value.
- Displays a message: The program displays a message to the user indicating that the system clock has been synchronized.

The NETTIME program uses the WinSock API to connect to the time server and receive the current time. The WinSock API is a library of functions that provide a way for programs to communicate with the network.

Here are some of the benefits of using the NETTIME program:

- The time is accurate: The time received from a time server is typically very accurate, since the time servers are synchronized with atomic clocks.
- It is easy to use: The NETTIME program is a simple and easy-to-use program that does not require any technical knowledge to use.
- It is portable: The NETTIME program can be run on any Windows computer that has the WinSock API installed.



Overall, the NETTIME program is a useful tool for synchronizing the system clock with an Internet time server. It is easy to use, accurate, and portable.

NETTIME's Structure

The NETTIME program consists of two main components: a modeless dialog box and the main application module. The modeless dialog box provides a user interface for selecting a time server, synchronizing the system clock, and displaying status information. The main application module handles the network communication with the time server, including socket creation, connection, data transfer, and error handling.

Modeless Dialog Box

The modeless dialog box is created using the DialogBoxParam function. The main application module resizes its window to accommodate the dialog box, ensuring that it occupies the entire client area. The dialog box contains a read-only edit field for displaying status messages, three buttons for selecting a server, synchronizing the clock, and closing the dialog box.

Socket Communication

The main application module handles the network communication with the time server using the Windows Sockets API. The following steps outline the basic sequence of socket operations:

WSAStartup: Initialize the Windows Sockets API by calling the WSAStartup function. This function takes the version number of the Windows Sockets API as an argument.

```
int iError = WSAStartup (wVersion, &WSAData) ;
```

Socket Creation: Create a socket using the socket function. This function takes three arguments: the address family, the socket type, and the protocol. The NETTIME program uses the AF_INET address family, the SOCK_STREAM socket type, and the IPPROTO_TCP protocol.

```
sock = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP) ;
```

Address Resolution: Convert the IP address string into a network address structure. The NETTIME program uses the inet_addr function to convert the IP address string (szIPAddr) into a network byte order representation.

```
struct sockaddr_in serverAddr;
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons (13); // Port number for time synchronization
serverAddr.sin_addr.s_addr = inet_addr (szIPAddr);
```

Connection Establishment: Initiate a connection to the time server using the connect function. This function takes two arguments: the socket handle and the address structure containing the server's address information.

```
iError = connect (sock, (struct sockaddr *)&serverAddr, sizeof (serverAddr)) ;
```

Asynchronous I/O: Set up asynchronous I/O notification using the WSAAsyncSelect function. This function associates the socket with a window (the main dialog box) and a message (WM_SOCKET_NOTIFY) to receive notifications for socket events, such as connection completion and data reception.

```
WSAAsyncSelect (sock, hwndMain, WM_SOCKET_NOTIFY, FD_CONNECT | FD_READ) ;
```

Data Reception: Receive the time data from the server using the recv function. This function takes three arguments: the socket handle, a buffer for storing the received data, and the size of the buffer.

```
iError = recv (sock, buf, 4, 0) ;
```

Time Synchronization: Synchronize the system clock using the SetSystemTime function. This function takes a pointer to a SYSTEMTIME structure containing the synchronized time information.

```
iError = SetSystemTime (&st) ;
```

Socket Closure: Close the socket using the closesocket function. This function takes the socket handle as an argument.

```
closesocket (sock) ;
```

WSACleanup: Terminate the Windows Sockets API by calling the WSACleanup function.

```
WSACleanup () ;
```

Error Handling

The NETTIME program implements error handling for various aspects of the network communication, such as socket creation, connection, data reception, and system clock synchronization. It displays error messages in the modeless dialog box and provides appropriate feedback to the user.

EXPANDING THE DESCRIPTION

Socket Creation:

The first step in the socket communication process is to create a socket using the socket function. This function takes three arguments: the address family, the socket type, and the protocol. The NETTIME program uses the AF_INET address family, which is used for Internet communications, the SOCK_STREAM socket type, which indicates that the socket will be used for TCP connections, and the IPPROTO_TCP protocol, which is the TCP protocol.

```
sock = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP) ;
```

Address Resolution:

The next step is to resolve the IP address string to a network address structure. The NETTIME program uses the inet_addr function to convert the IP address string (szIPAddr) to a network byte order representation.

```
struct sockaddr_in serverAddr;
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons (IPPORT_TIMESERVER); // Port number for time synchronization
serverAddr.sin_addr.s_addr = inet_addr (szIPAddr);
```

Connection Establishment:

Once the socket is created and the address is resolved, the next step is to establish a connection to the time server using the connect function. This function takes two arguments: the socket handle and the address structure containing the server's address information.

```
iError = connect (sock, (struct sockaddr *)&serverAddr, sizeof (serverAddr)) ;
```

Asynchronous I/O:

To avoid hanging the application while waiting for the connection to be established, the NETTIME program uses asynchronous I/O. This is done by calling the WSAAsyncSelect function to associate the socket with a window (the main dialog box) and a message (WM_SOCKET_NOTIFY) to receive notifications for socket events, such as connection completion and data reception.

```
WSAAsyncSelect (sock, hwndMain, WM_SOCKET_NOTIFY, FD_CONNECT | FD_READ) ;
```

Data Reception:

When the connection is established, the recv function is called to receive the time data from the server. This function takes three arguments: the socket handle, a buffer for storing the received data, and the size of the buffer.

```
iError = recv (sock, (char *) &ulTime, 4, 0) ;
```

The recv function is called with the MSG_PEEK option, which means that it only reads the data into the buffer but does not remove it from the socket. This is done because the NETTIME program may need to read the data multiple times.

Time Synchronization:

Once the time data is received, the ChangeSystemTime function is called to synchronize the system clock. This function takes the received time data and converts it to a format that the system clock can understand.

```
ChangeSystemTime (ulTime) ;
```

Finally, the socket is closed using the closesocket function.

The NETTIME program synchronizes the system clock by receiving the current time from a time server and then setting the system clock to that time. The time synchronization process is as follows:

- Receive Time Data from Server: The NETTIME program receives the current time from a time server using TCP/IP sockets. The time data is received in a 32-bit integer format, representing the number of seconds since 0:00 UTC on January 1, 1900.
- Convert Time Data to UTC: The NETTIME program converts the received time data from seconds since UTC to FILETIME, which is a 64-bit integer format representing the number of 100-nanosecond intervals since January 1, 1601.
- Convert UTC to System Time: The NETTIME program converts the UTC time represented by the FILETIME value to a SYSTEMTIME structure, which is a 64-bit structure representing the date and time in a format that the system clock can understand.
- Set System Time: The NETTIME program sets the system clock to the synchronized time by calling the SetSystemTime function. This function takes a pointer to a SYSTEMTIME structure as an argument.
- Update Display: The NETTIME program updates the display by obtaining the updated time with a call to GetLocalTime. The updated time is then passed to the FormatUpdatedTime function, which converts the time to an ASCII string in a format suitable for display.
- Handle Errors: If the SetSystemTime function fails, the NETTIME program displays an error message indicating that the time synchronization failed.

Here are some additional details about the time synchronization process:

- The NETTIME program uses the WSAAsyncSelect function to receive notifications when the connection to the time server is established and when data is received.
- The NETTIME program uses the struct sockaddr_in structure to represent the IP address of the time server.
- The NETTIME program uses the MSG_PEEK option when calling the recv function to read the time data without removing it from the socket. This allows the program to read the data multiple times if necessary.
- The NETTIME program uses the __int64 data type to represent the 64-bit value in the FILETIME structure. This data type is a Microsoft compiler extension to the ANSI C standard.
- The NETTIME program uses the GetTimeFormat and GetDateFormat functions to convert the SYSTEMTIME structure to ASCII strings in a format suitable for display.

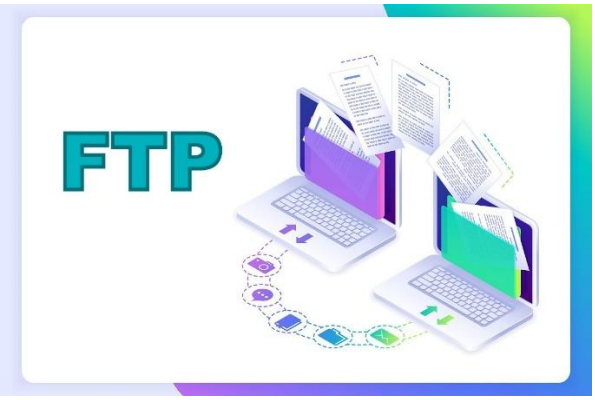
WININET AND FTP:

WinInet is a collection of high-level functions that provide a simplified interface for accessing various Internet protocols, including HTTP, FTP, and Gopher. It simplifies the process of working with these protocols by making their syntax similar to that of standard Windows file functions. This makes it easier for developers to incorporate Internet functionality into their applications without having to deal with the intricacies of low-level network protocols.

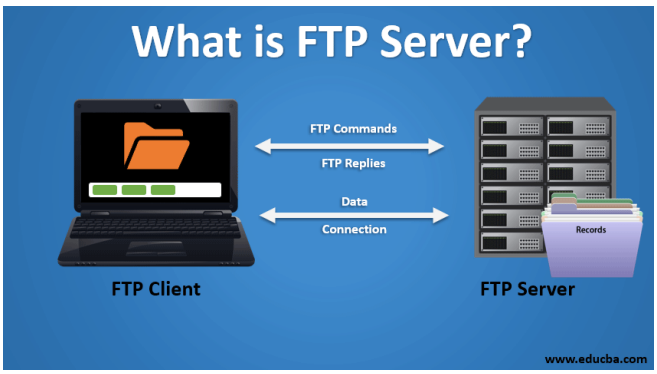
The WinInet API provides functions for various tasks related to Internet access, including:

- Opening and closing connections to remote servers
- Downloading and uploading files
- Manipulating directories
- Sending and receiving data
- Parsing and interpreting responses from remote servers

WinInet supports both anonymous FTP and FTP with authentication. Anonymous FTP allows users to access files without providing a username or password, while FTP with authentication requires a valid username and password.



FTP is a file transfer protocol that allows users to transfer files between computers over a network. It is commonly used for transferring files between a local computer and a remote server. FTP operates in two modes: active and passive. In active mode, the client initiates the connection to the server. In passive mode, the server initiates the connection to the client.



WinInet provides functions for both active and passive FTP. It also supports various features of FTP, such as resuming interrupted downloads, specifying the file transfer mode, and handling errors.

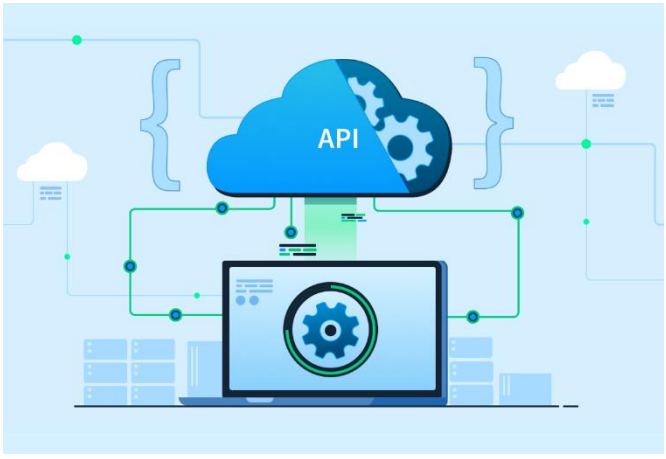
The UPDDEMO program demonstrates how to use WinInet to download files from an FTP site. It connects to an anonymous FTP site, navigates to the desired directory, and downloads the specified file. The program stores the downloaded file in the current directory.

The UPDDEMO program uses the following WinInet functions:

- FtpOpenFile: Opens a connection to a remote file.
- FtpReadFile: Reads data from a remote file.
- FtpCloseFile: Closes a connection to a remote file.
- FtpDeleteFile: Deletes a file from a remote server.
- FtpCreateDirectory: Creates a directory on a remote server.
- FtpSetCurrentDirectory: Navigates to a specific directory on a remote server.
- FtpGetFileAttributes: Retrieves information about a file on a remote server.

OVERVIEW OF FTP

The FTP API (Application Programming Interface) is a collection of functions that provide a simplified interface for accessing FTP servers. It allows developers to connect to FTP servers, download and upload files, and manage directories. The FTP API is part of the WinInet (Windows Internet) API, which also includes functions for HTTP, Gopher, and other Internet protocols.



To use the FTP API, you must first call the InternetOpen function to create an Internet session. This function returns a handle to the Internet session, which you use to call other FTP API functions. You must also link with the WININET.LIB library to use the FTP API functions.

Once you have created an Internet session, you can connect to an FTP server by calling the InternetConnect function. This function requires the Internet session handle, the server name, the port number, and the user name and password (or NULL for anonymous FTP).

After connecting to an FTP server, you can use the FTP API functions to download and upload files, manage directories, and send commands to the FTP server. The FTP API provides functions for the following tasks:

- Downloading files: The FtpOpenFile function opens a connection to a remote file, and the FtpReadFile function reads data from the remote file.
- Uploading files: The FtpCreateFile function creates a new file on the FTP server, and the FtpWriteFile function writes data to the file.
- Managing directories: The FtpCreateDirectory function creates a new directory on the FTP server, and the FtpRemoveDirectory function removes a directory.
- Sending commands: The FtpSendCommand function sends a command to the FTP server.

FTP API Functions

The following is a list of some of the FTP API functions:

- InternetOpen: Opens an Internet session.
- InternetConnect: Connects to an FTP server.
- FtpOpenFile: Opens a connection to a remote file.
- FtpReadFile: Reads data from a remote file.
- FtpWriteFile: Writes data to a remote file.
- FtpGetFileAttributes: Retrieves information about a file on a remote server.
- FtpSetCurrentDirectory: Navigates to a specific directory on a remote server.
- FtpGetCurrentDirectory: Retrieves the current directory on a remote server.
- FtpCreateDirectory: Creates a directory on a remote server.
- FtpRemoveDirectory: Removes a directory on a remote server.
- FtpDeleteFile: Deletes a file on a remote server.
- FtpSendCommand: Sends a command to the FTP server.

Using the FTP API

To use the FTP API, you will need to include the WININET.H header file in your source code. You will also need to link with the WININET.LIB library.

The following is an example of how to download a file from an FTP server using the FTP API:

```
HINTERNET hInternetSession = InternetOpen ("My Application", INTERNET_OPEN_TYPE_DIRECT,
  INTERNET_SCHEME_HTTP, NULL, 0);

HINTERNET hFtpSession = InternetConnect (hInternetSession, "ftp.example.com", INTERNET_DEFAULT_FTP_PORT,
  "anonymous", "mypassword", INTERNET_SERVICE_FTP, 0, 0);

HINTERNET hFile = FtpOpenFile (hFtpSession, "myfile.txt",
  GENERIC_READ, FTP_TRANSFER_TYPE_BINARY, INTERNET_FLAG_RELOAD, 0);

DWORD dwBytesRead;
char szBuffer[1024];

while (FtpReadFile (hFile, szBuffer, sizeof (szBuffer), &dwBytesRead, NULL, NULL))
{
  // Process the data in the buffer
}

FtpCloseFile (hFile);
FtpCloseHandle (hFtpSession);
FtpCloseHandle (hInternetSession);
```

This code will download the file myfile.txt from the FTP server ftp.example.com. The file will be saved to the current directory on the local machine.

The FTP API is a powerful tool that can be used to download files, upload files, manage directories, and send commands to FTP servers. It is a valuable tool for developers who need to access FTP servers from their applications.

Deleting Files

To delete a file from an FTP server, you can use the FtpDeleteFile function. This function takes two arguments: the FTP session handle and the file name. The file name must be a fully qualified path name, including the server name.

```
hFind = FtpFindFirstFile (hFtpSession, szSearchPattern, &FindFileData) ;
```

This function returns a handle to the first file that matches the search pattern. You can use this handle to call the InternetFindNextFile function to get the names of additional matching files.

```
while (InternetFindNextFile (hFind, &FindFileData))
{
  // Process the file information in FindFileData
}
```

When you are finished searching for files, you should call the InternetCloseHandle function to close the handle.

```
InternetCloseHandle (hFind) ;
```

Opening and Reading Files

To open a file on an FTP server, you can use the FtpFileOpen function. This function takes four arguments: the FTP session handle, the file name, the file access mode, the transfer type, and the flags.

```
hFile = FtpFileOpen (hFtpSession, szFileName, GENERIC_READ, FTP_TRANSFER_TYPE_BINARY, INTERNET_FLAG_RELOAD, 0);
```

Once you have opened a file, you can read data from it using the InternetReadFile function.

```
DWORD dwBytesRead ;
char szBuffer[1024] ;

while (InternetReadFile (hFile, szBuffer,
  sizeof (szBuffer), &dwBytesRead, NULL, NULL))
{
  // Process the data in the buffer
}
```

When you are finished reading from a file, you should call the FtpCloseFile function to close the handle.

```
FtpCloseFile (hFile) ;
```

UPDDEMO.C PROGRAM

This file contains the main code for the Update Demo program. It defines the window procedure, the dialog procedure, and the FtpThread function.

Window Procedure:

The window procedure handles all of the messages for the main window of the program. It includes the following messages:

- WM_CREATE: This message is sent when the window is created. It is used to initialize the window and the scroll bar.
- WM_SIZE: This message is sent when the window is resized. It is used to update the scroll bar and the list of files.
- WM_VSCROLL: This message is sent when the vertical scroll bar is changed. It is used to scroll the list of files.
- WM_USER_CHECKFILES: This message is sent when the user clicks the "Check for Updates" button. It checks if the latest file exists and, if not, it displays the dialog box.
- WM_USER_GETFILES: This message is sent when the user clicks the "Get Updates" button. It reads the list of files from disk and displays them in the list box.
- WM_PAINT: This message is sent when the window needs to be repainted. It draws the list of files in the window.
- WM_DESTROY: This message is sent when the window is destroyed. It is used to clean up resources.

Dialog Procedure:

The dialog procedure handles all of the messages for the dialog box. It includes the following messages:

- WM_INITDIALOG: This message is sent when the dialog box is created. It is used to initialize the dialog box and start the FTP thread.
- WM_COMMAND: This message is sent when the user clicks a button in the dialog box. It stops the FTP thread if the user clicks the "Cancel" button.

FtpThread:

The FtpThread function is a thread that downloads files from the FTP server. It includes the following steps:

- Open an internet session.
- Open an FTP session.
- Set the directory to the directory containing the files to download.
- Get the first file fitting the template.
- Download the file to the local hard disk.
- Close the FTP session.
- Close the internet session.

GetFileList:

The GetFileList function reads files from disk and saves their names and contents. It includes the following steps:

- Open the first file fitting the template.
- Get the size of the file.
- Allocate space for the filename and the contents.
- Read the filename and the contents of the file.
- Close the file.
- Sort the files by filename.

ButtonSwitch:

The ButtonSwitch function is called after the FTP download is complete or the user cancels the download. It displays a final status message and changes the Cancel button to OK.

The function first checks to see if there was an error during the download. If there was an error, it displays an error message. Otherwise, it displays a message indicating that the download was successful.

After displaying the status message, the function changes the Cancel button to OK. This allows the user to close the dialog box.

Here is the code for the ButtonSwitch function:

```
VOID ButtonSwitch(HWND hwndStatus, HWND hwndButton, TCHAR * szText)
{
    if (szText)
    {
        SetWindowText(hwndStatus, szText);
    }
    else
    {
        SetWindowText(hwndStatus, TEXT("Internet Session Cancelled"));
    }
    SetWindowText(hwndButton, TEXT("OK"));
    SetWindowLong(hwndButton, GWL_ID, IDOK);
}
```

The first parameter to the function, hwndStatus, is the handle to the control that displays the status message. The second parameter, hwndButton, is the handle to the Cancel button. The third parameter, szText, is the text of the status message.

The function first checks to see if the szText parameter is NULL. If it is, then the download was cancelled and an error message is displayed. Otherwise, the download was successful and a success message is displayed.

After displaying the status message, the function changes the text of the Cancel button to "OK" and changes its ID to IDOK. This allows the user to close the dialog box.

Compare function

The Compare function helps arrange files in order by their filenames. It determines the relative positions of two files by comparing their filenames.

If the first filename is less than the second, it returns -1, indicating the first file comes before the second. If the filenames are equal, it returns 0. If the first filename is greater, it returns 1, signifying the first file comes after the second.

The Compare function relies on the lstrcmp function to directly compare the filenames. lstrcmp returns -1 for less, 0 for equal, and 1 for greater, matching the Compare function's output.

In the GetFileList function, Compare is employed to sort the list of files by filename.

The qsort function handles the actual sorting process, and Compare serves as the comparison function passed to qsort, ensuring the files are arranged alphabetically.

WE’RE DONE GUYS!!!! DONE!!! NOW IT’S UP TO YOU TO PRACTICE 👍 😅

BEST OF LUCK IN YOUR ENDEAVOURS!

YOU ARE NOW WELL EQUIPPED TO TACKLE OPERATING SYSTEMS BOOKS BY ABRAHAM SILBERSCHATZ WHICH IMPLEMENTS WHAT WE’VE LEARNT THROUGHOUT!

BYE!