Web Server Assignment

- **Assignment specification:**
  - Implement a basic web server in C# or PowerBasic.
  - Implement basic server-side scripting.
  - Any enhancements you deem fit.
  - You may use the supplied HTTP.bas PowerBasic program as a basis.
  - The web server may be a tray application or NT service.

General Architecture (1)

- The web server should be able to handle and server multiple connections simultaneously.
- Each connection will be handled by a thread.
- You should maintain a **Thread Pool**.
- The pool may be implemented as a linked list of records. The records may be defined as:

```
Type ConnectionThread
  ThreadHandle         as Long
  LastInterationTime as Time
  SomeFlags            as ...
End Type
```

General Architecture (2)

- Clearly, you should be able to append (insert) and delete from the thread pool list.
- You may also require searching for a thread in the pool. You could keep the list sorted to allow efficient binary searching.
- Apart from a thread per connection, there could also be a **Monitor Thread**.
- The Monitor thread will be used to manage the thread pool (list).

General Architecture (3)

- Depending on the language you choose to implement the web server, you might need to create a window whose **Callback Function** would respond to socket/network events.
- This window could also server as the UI of the web server (i.e. a status window).

Starting the Web Server

- When the server window is created (**WM_CREATE** or **WM_INITDIALOG** in the callback function), the server should be started.
- **TCP Accept** and **Connect** Messages are mapped to a custom windows message.
- In HTTP.bas:

```
fnum = FreeFile
Tcp Open Server Port %PORTNUM As #fnum
...
Tcp Notify #fnum, Accept Connect To hDlg _
                            As %WM_PAGEREQUEST
```

Handling Client Connections (1)

- When a client interacts with the server, the custom message specified for the window fires (%WM_PAGEREQUEST in this example).
- There may be several client interactions that may occur. The one that interests us is the interaction we must respond to by accepting a connection.
- The **loword** of the **lParam** in the callback determines this interaction type - **%FD_ACCEPT**.

Handling Client Connections (2)

- When accepting a connection there are a number of things we have to do:
  - Create a socket, the client will use to communicate over.
  - Create a thread that will handle the socket from now on.

Creating the Connection Handler Thread

- Create the thread (passing the socket number as to the thread function).
- Create a new item in the thread pool.
- In the new item store:
  - The windows handle of the thread.
  - The date/time the thread was created.
  - Any other details.

The Monitor Thread (1)

- For efficiency, a connection thread is not allowed to run for more than **T** seconds.
- Expiry of a thread **S** may be determined by **(TimeNow – TimeThread(s)) > T**.
- The monitor thread will periodically scan through the pool and expire connections.
- When a connection thread **S** completes within it’s time-to-live, flag may be marked for the monitor thread to kill it.

The Monitor Thread (2)

```
Do
  Wait 10 seconds (for example)
  For Each Item X in the Pool
    If X.ReadyFlag Then
      Kill X.ThreadHandle
    End If

    If (TimeNow-X.LastInteractionTime) > TTL Then
      Close the Socket Handled by Thread
      Kill X.ThreadHandle
    End If
  Next
Loop (Forever)
```

Connection Handler Thread

```
Declare a Data Buffer String
Do
  Append TCP Data To Buffer
Loop until we have all the HTTP GET Header

Determine the File Requested from Buffer (Header)
                  Take Care of ‘File Not Found’s

Read the File Requested From Disk
Send It

Close Socket
Set Flag to tell Monitor to Kill Us
```

Server-Side Stuff

- You should implement basic server-side tags such including at least:
  - `<%TIME%>`
  - `<%DATE%>`
  - `<%FILESIZE%>`
  - `<%DATECREATED%>`
  - `<%DATELASTCHANGED%>`
  - `<%DATELASTACCESSED%>`
  - `<%FILENAME%>`