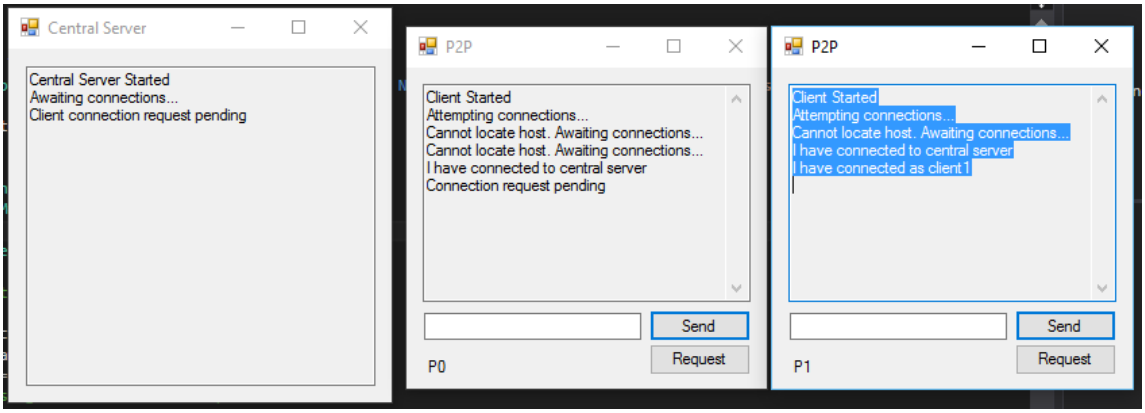
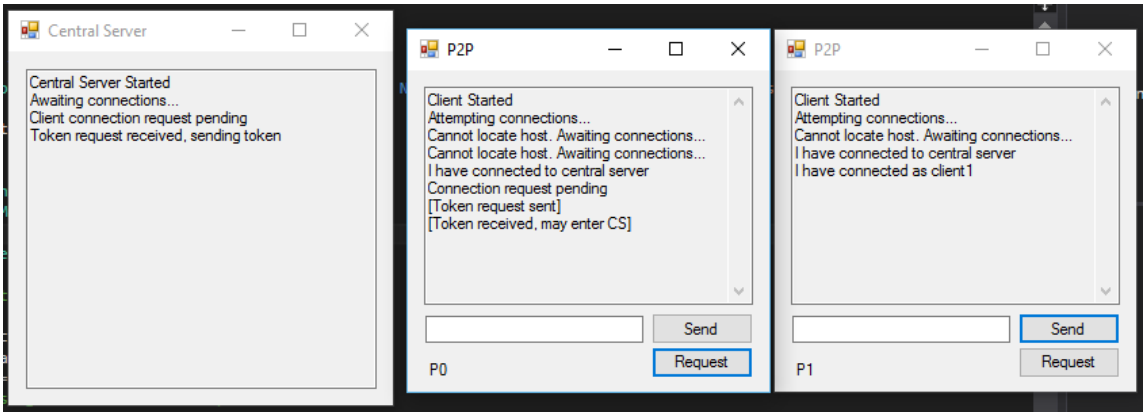
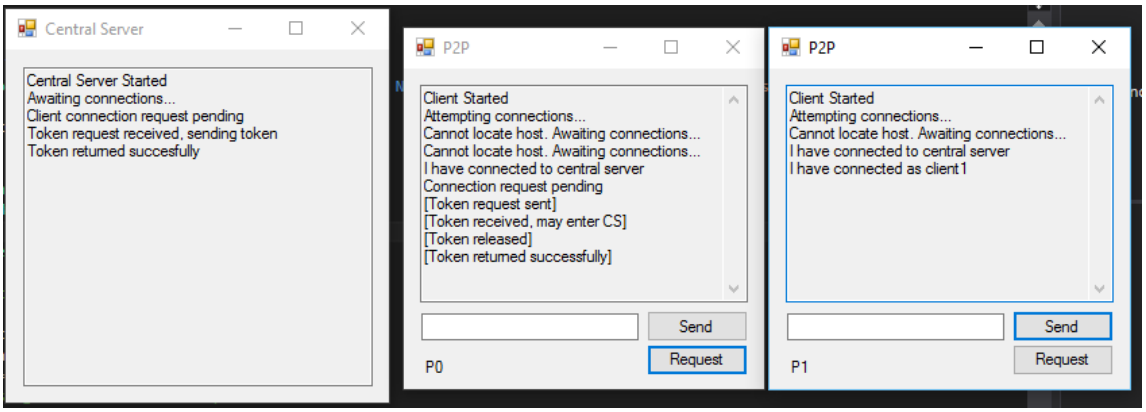
Application Test Case



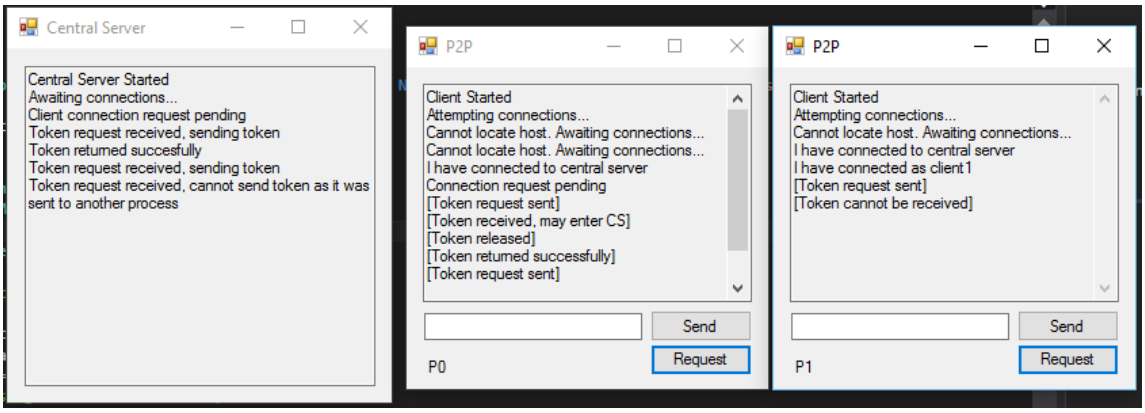
In this example, the user P1 attempts to request a token from the central server, followed by P2 who requests a token sometime after. First, all peers are initialized and connect to the central server. They use TCP protocol to connect on port 8080 under the same IP address (same machine).



P1 sends the request and receives an acknowledgement from the server stating that it has received the request and is prepared to send the token back. The token is sent back, and with that the user P1 is allowed to process to the critical section. Remember that P2 has not yet taken any action. The user P1 will remain in its critical section for roughly 10 seconds.



After P1 is finished, it returns the token to the central server. The server is waiting for a response, eventually receiving the token when it gets the message from P1. P1 has officially exited its critical section and the token may now be picked up by another peer in the network.



Now, let's say that P2 wishes to request the token while P1 is still in its critical section. The token has not been returned to the central server at this point, so when P2 sends the request, it is declined by the server since P1 still has the token. P2 must now wait for P1 to be finished before trying again, at which point it may be able to send its request.

There is a case where the central server will crash before the token is returned to it. Going by the previous example, if the central server distributes the token to P1 and then crashes, the system will still run. P1 will enter its critical section like normal, and P2 still has to wait for it to be finished before it can request a token. When P1 finished, it will attempt to send the token back only to find that the server is down. In this case, P1 releases the token and deletes it, waiting for the central server to start up again. Once the central server starts up, it generates a new token that may be used. At this point, either P1 or P2 may choose to request the token and use it like normal.

Most of the code that was added to supplement the central server was created in a separate project. Some code for the clients has been added to the main program however, such as the thread for entering the critical section, and the token receiver. This is illustrated below:

```
Private Sub btnReqTok_Click(sender As Object, e As EventArgs) Handles
btnReqTok.Click

    If hasToken Then
        txtChat.Text = txtChat.Text & "[Token already received] " & vbNewLine
        Return
    End If

    data = System.Text.Encoding.ASCII.GetBytes("Requesting token")
    Dim messageToSend As UserMessage

    messageToSend = New UserMessage(process.User, data, vectorClock)

    If serverClient.Connected = True Then
        If Not serverStream Is Nothing Then
            serverStream = serverClient.GetStream()
```

```
            ' Send the message to the connected TcpServer.
            serverStream.Write(data, 0, data.Length)
        End If
    End If

    txtChat.Text = txtChat.Text & "[Token request sent] " & vbNewLine
End Sub


Private Sub EnterCS()
    'Enter CS for 10 seconds before releasing resource
    Thread.Sleep(10000)
    hasToken = False

    If txtChat.InvokeRequired Then
        txtChat.Invoke(New AppendTextBoxDelegate(AddressOf AppendTextBox),
New Object() {txtChat.Text & "[Token released]" & vbNewLine})
    Else
        txtChat.AppendText(txtChat.Text & "[Token released]" & vbNewLine)
    End If

    data = System.Text.Encoding.ASCII.GetBytes("Returning token")
    Dim messageToSend As UserMessage

    messageToSend = New UserMessage(process.User, data, vectorClock)

    'Try catch for returning token if server crashes before getting token
    Try
        If serverClient.Connected = True Then
            If Not serverStream Is Nothing Then
                serverStream = serverClient.GetStream()
                ' Send the message to the connected TcpServer.
                serverStream.Write(data, 0, data.Length)
            End If
        End If
    Catch
        If txtChat.InvokeRequired Then
            txtChat.Invoke(New AppendTextBoxDelegate(AddressOf
AppendTextBox), New Object() {txtChat.Text & "[Could not detect server, token
still released]" & vbNewLine})
        Else
            txtChat.AppendText(txtChat.Text & "[Could not detect server,
token still released]" & vbNewLine)
        End If
    End Try

End Sub
```

The Receiver function has also been modified but it is too large to include in the whole document. The main change has been included here for communicating with the central server:

```
If Not serverStream Is Nothing Then
    If serverStream.DataAvailable Then
        'Variable to store bytes received
        data = New [Byte](256) {}
```

```
        'Variable to store string representation
        Dim receivedData As [String] = [String].Empty

        'Read in the received bytes
        Dim bytes As Int32 = serverStream.Read(data, 0, data.Length)
        receivedData = System.Text.Encoding.ASCII.GetString(data, 0, bytes)
        If receivedData = "True" Then
            messageString = "[Token received, may enter CS]"
            hasToken = True
            'Create a listener thread for other clients to connect to
            Dim CSThread As New Thread(New ThreadStart(AddressOf EnterCS))
            CSThread.IsBackground = True
            CSThread.Start()
        ElseIf receivedData = "[ACK]" Then
            messageString = "[Token returned successfully]"
            hasToken = False
        Else
            messageString = "[Token cannot be received]"
        End If
        If txtChat.InvokeRequired Then
            txtChat.Invoke(New AppendTextBoxDelegate(AddressOf
AppendTextBox), New Object() {txtChat.Text & messageString & vbNewLine})
        Else
            txtChat.AppendText(txtChat.Text & messageString & vbNewLine)
        End If

        If serverStream.DataAvailable Then
            Dim Buffer As [Byte]()
            Buffer = New [Byte](256) {}
            While serverStream.DataAvailable
                serverStream.Read(Buffer, 0, Buffer.Length)
            End While
        End If
    End If
End If
```

Other than this, the majority of changes are in the CentralServer project file. This can be found included in the folder.

For the second test case, it works very similar except dealing with three peers this time. The idea is similar to above however, where P1 sends a token request to the central server. P2 and P3 must both wait for P1's request to be sent before sending a request of their own, and so on and so forth with any additional peers.