

Forum:[Hack Forums](#)

Thread: [VB Cache - Instructions](#)

InterGalactic - December 29, 2019, 1:23 am

So i basicly create a simple app and i think that i have a CPU Cache problem based on how the CPU manage the cache-instructions-commands to make processes of information faster.

So what im trying to do in the given code below is to stop-block a button doubleclicking.

An explain is i don't want to allow the user double click the button so the process-code of it is not apply twice.

I know that i can just disable the button and i don't want another code to do it since i know how to create one, i only want your ideas to understand/see if im correct in what i think and also if anyone is able to give a trick to bypass-disable the caching of apps. Since my program is small, i don't need to have anything saved!

So in the code below the label will changed to "Worked" when i first click the button but if i click it second time it will change to cache failed you. I also want to point out that i tried to use this boolean thing in bigger code app and it worked even if the test = true code is above the doit call but will only work only one time as in this example.

Code Private test As Boolean = False

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
```

```
    doit()  
    test = True  
End Sub
```

```
Private Sub doit()  
    If test = False Then  
        Label1.Text = "Worked"  
        test = False  
    Else  
        Label1.Text = "CPU Cache failed you!"  
        test = False  
    End If  
End Sub
```

In the end i want to point out that i have a buffered RAM on unbuffered system but all fuctions of CPU caching like ECC and staff is disabled (some memory things is still enabled). Let me know if you think this is the problem or if the code does not work for you as explained!

Washi - December 29, 2019, 4:03 am

Not sure if I am understanding you correctly, but this doesn't seem like such a low level CPU caching problem? The CLR has certain guarantees about the code it is running; a caching problem you are describing is a lot more likely to occur only when it comes down to the assembly level.

To me, this looks a lot more like a normal logic problem rather than a low level architectural problem.

Initially, `doit()` is called while `test` is set to `False`. This means the condition of the `if` statement inside `doit` will be satisfied, and `Worked` will be the new label's text. Even though `test` is set to `False` again in this sub, after returning from `doit`, `test` is set to `True` back in the button's click handler sub. Therefore, the second time the button is clicked, the `if` condition is not satisfied, which means it goes to the `else` part, setting the "CPU Cache failed you" message as the label's text.

Similarly, moving up the line "`test = True`" to before the call to `doit` will result in the application always setting the "CPU Cache failed you" message as the label's text.

So even though your program puts the "CPU cache failed you" message, this is not really the cause of such a failure. Rather, there's something wrong with the logic that you put in the button's click handler instead.

itcorp - December 29, 2019, 4:56 am

if i understand this correctly this is what you are looking for.

vb

```
Code Public Class SimpleLogic
    Implements INotifyPropertyChanged

    Private _testNumber As Integer = 0
    Public Property testNumber() As Integer
        Get
            Return _testNumber
        End Get
        Set(ByVal value As Integer)
            _testNumber = value
            NotifyPropertyChanged("testNumber")
        End Set
    End Property

    Private _testMessage As String = "Please run the test"
    Public Property testMessage() As String
        Get
            Return _testMessage
        End Get
        Set(ByVal value As String)
            _testMessage = value
            NotifyPropertyChanged("testMessage")
        End Set
    End Property

    Private _isFaild As Boolean = False
    Private Property isFaild() As Boolean
        Get
            Return _isFaild
        End Get
        Set(ByVal value As Boolean)
            _isFaild = value
            NotifyPropertyChanged("isFaild")
        End Set
    End Property
```

```
Private _isRunning As Boolean = False
Private Property isRunning() As Boolean
    Get
        Return _isRunning
    End Get
    Set(ByVal value As Boolean)
        _isRunning = value
        NotifyPropertyChanged("isRunning")
    End Set
End Property
```

```
Public Property Start() As Command
Public Property [Stop]() As Command
```

```
Public Sub New()
    Start = New Command(AddressOf StartCode, Function(o) Not isRunning AndAlso Not isFailed)
    [Stop] = New Command(AddressOf StopCode, Function(o) isRunning)
End Sub
```

```
Private Async Sub StartCode(ByVal ob As Object)
    isRunning = True
    testMessage = "Test is running"
    For i As Integer = 0 To 59

        If isFailed OrElse Not isRunning Then
            Exit For
        End If

        If Not isFailed Then
            testNumber = i
        End If

        Await Task.Delay(TimeSpan.FromSeconds(1))

        If i = 20 Then
            isFailed = True

            testMessage = $"Test is failed on {i} test"

            If [Stop].CanExecute(Nothing) Then
                [Stop].Execute(Nothing)
            End If

        End If
    Next i
End Sub
```

```
Private Sub StopCode(ByVal ob As Object)
    isRunning = False
    If Not isFailed Then
        testMessage = $"Test is stoped on {testNumber} test"
    End If
End Sub
```

End Sub

```
Public Event PropertyChanged As PropertyChangedEventHandler Implements
INotifyPropertyChanged.PropertyChanged
    Protected Sub NotifyPropertyChanged(ByVal propertyName As String)
        RaiseEvent PropertyChanged(Me, New PropertyChangedEventArgs(propertyName))
    End Sub
End Class
Code Public Class Command
    Implements ICommand
```

```
Private [Do] As Action(Of Object)
Private CanDo As Func(Of Object, Boolean)
```

```
Public Sub New(ByVal [Do] As Action(Of Object))
    Me.New([Do], Nothing)
```

```
End Sub
Public Sub New(ByVal [Do] As Action(Of Object), ByVal CanDo As Func(Of Object, Boolean))
    Me.CanDo = CanDo
    Me.Do = [Do]
    AddHandler CommandManager.RequerySuggested, Sub(o, e) Changed()
End Sub
```

```
Public Event CanExecuteChanged As EventHandler Implements
ICommand.CanExecuteChanged
    Public Function CanExecute(ByVal parameter As Object) As Boolean Implements
ICommand.CanExecute
        Return CanDo(parameter)
    End Function
    Public Sub Execute(ByVal parameter As Object) Implements ICommand.Execute
        [Do](parameter)
    End Sub
```

```
Public Sub Changed()
    CanExecuteChangedEvent?.Invoke(Nothing, EventArgs.Empty)
End Sub
```

End Class

c#

```
Code public class SimpleLogic: INotifyPropertyChanged
{
    private int _testNumber = 0;
    public int testNumber
    {
        get { return _testNumber; }
        set { _testNumber = value; NotifyPropertyChanged("testNumber"); }
    }

    private string _testMessage="Please run the test";
    public string testMessage
    {
        get { return _testMessage; }
        set { _testMessage = value; NotifyPropertyChanged("testMessage"); }
    }
}
```

```

}

private bool _isFailed = false;
private bool isFailed
{
    get { return _isFailed; }
    set { _isFailed = value; NotifyPropertyChanged("isFailed"); }
}

private bool _isRunning = false;
private bool isRunning
{
    get { return _isRunning; }
    set { _isRunning = value; NotifyPropertyChanged("isRunning"); }
}

public Command Start { get; set; }
public Command Stop { get; set; }

public SimpleLogic()
{
    Start = new Command(StartCode, (o) => !isRunning &&!isFailed);
    Stop = new Command(StopCode, (o) => isRunning);
}

private async void StartCode(object ob)
{
    isRunning = true;
    testMessage = "Test is running";
    for(int i=0;i<60;i++)
    {
        if (isFailed || !isRunning)
            break;

        if (!isFailed)
            testNumber = i;

        await Task.Delay(TimeSpan.FromSeconds(1));

        if (i==20)
        {
            isFailed = true;

            testMessage = $"Test is failed on {i} test";

            if (Stop.CanExecute(null))
                Stop.Execute(null);
        }
    }
}

```

```

private void StopCode(object ob)
{
    isRunning = false;
    if(!isFailed)
        testMessage = $"Test is stoped on {testNumber} test";
}

public event PropertyChangedEventHandler PropertyChanged;
protected void NotifyPropertyChanged(string propertyName)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
    }
}
}
Code public class Command : ICommand
{
    Action<object> Do;
    Func<object, bool> CanDo;

    public Command(Action<object> Do) : this(Do, null)
    {
    }

    public Command(Action<object> Do, Func<object, bool> CanDo)
    {
        this.CanDo = CanDo;
        this.Do = Do;
        CommandManager.RequerySuggested += (o, e) => Changed();
    }

    public event EventHandler CanExecuteChanged;
    public bool CanExecute(object parameter) => CanDo(parameter);
    public void Execute(object parameter) => Do(parameter);

    public void Changed() => CanExecuteChanged?.Invoke(null, EventArgs.Empty);
}

```

InterGalactic - December 29, 2019, 8:09 am

(Dec 29, 2019 05:03 AM)Washi Wrote: [\[CITE\]](#) Not sure if I am understanding you correctly, but this doesn't seem like such a low level CPU caching problem? The CLR has certain guarantees about the code it is running; a caching problem you are describing is a lot more likely to occur only when it comes down to the assembly level.

To me, this looks a lot more like a normal logic problem rather than a low level architectural

problem.

Initially, `doit()` is called while `test` is set to `False`. This means the condition of the `if` statement inside `doit` will be satisfied, and `Worked` will be the new label's text. Even though `test` is set to `False` again in this sub, after returning from `doit`, `test` is set to `True` back in the button's click handler sub. Therefore, the second time the button is clicked, the `if` condition is not satisfied, which means it goes to the `else` part, setting the "CPU Cache failed you" message as the label's text.

Similarly, moving up the line `"test = True"` to before the call to `doit` will result in the application always setting the "CPU Cache failed you" message as the label's text.

So even though your program puts the "CPU cache failed you" message, this is not really the cause of such a failure. Rather, there's something wrong with the logic that you put in the button's click handler instead.

Hello.

You are 100% correct of what you said about the setting of the boolean.

I miss thought how the process of the code is done.

Thanks a lot for the reply. ?

(Dec 29, 2019 05:56 AM)itcorp Wrote: [\[CITE\]](#) if i understand this correctly this is what you are looking for.

My answer to you is BROOOOOOOOO!

That is sick, i would like to ask for a advise on changing values of textboxes etc for `BeginInvoke` that is the only one that work for me to completly unlag the interface of the program when doing internet checks etc. Also what you suggest when doing the `Invoke` code a timer and if you suggest a better way to do it without lag.

```
Code Private Sub aaa()
```

```
Await Task.Delay(1)
```

```
blahblah
```

```
aaa()
```

```
End Sub
```

Explain: Lets say that the `blahblah` on the above code is a connection that open each time the code is executed to get some text information from a server.

itcorp - December 29, 2019, 9:17 am

In order to continue to give you quality answers it would be nice to show us your implementation of course if you are able to do that, because I really can't imagine what you're trying to do and how because it can be done in various ways. I wish you all the best buddy.

InterGalactic - January 1, 2020, 2:35 am

(Dec 29, 2019 10:17 AM)itcorp Wrote: [\[CITE\]](#) In order to continue to give you quality answers it would be nice to show us your implementation of course if you are able to do that, because I really can't imagine what you're trying to do and how because it can be done in various ways. I wish you all the best buddy.

Happy new year. Yeah ofc here it is.

Its kinda something like this.

```
Code Call New Action(AddressOf connect).BeginInvoke(Nothing, Nothing)
```

```
Private Async Sub connect()
```

```
Await Task.Delay(10)
```

```
If My.Computer.Network.IsAvailable = True Then
```

```
Try
```

```
Dim wureq As HttpWebRequest
```

```
Dim wuresp As HttpWebResponse
Dim userstream As StreamReader
wureq = HttpWebRequest.Create($"blahblah/{whatever.ToString}/read.txt")
wureq.Method = "GET"
wureq.KeepAlive = False
wureq.AllowAutoRedirect = False
wureq.AllowWriteStreamBuffering = False
wuresp = usersreq.GetResponse()
userstream = New StreamReader(wuresp.GetResponseStream)
users = userstream.ReadToEnd
wureq.GetResponse.Close()
wuresp.GetResponseStream.Close()
wuresp.Close()
userstream.Close()

If users.Length <> 0 Then
    textbox1.text = users
End If
Catch ex As Exception
    connect()
End Try
Else
    End
End If
connect()
End Sub
```