**Потоки в Visual Basic**

С появлением оператора AddressOf, часть индустрии ПО стала ориентироваться на авторов, показывающих как с использованием Visual Basic решать ранее невозможные задачи. Другая часть быстро охватила консультантов, помогающих пользователям, имеющим проблемы при решении таких задач.

Проблема не в Visual Basic или в технологии. Проблема в том, что большинство авторов применяют одно и тоже правило к AddressOf методикам, что большинство компаний по разработке ПО считают, что если Вы должны что-то сделать, то Вы сможете. Идея о том, что применение самой новой и последней технологии должно, по определению, быть самым лучшим решением проблемы, широко распространена в индустрии ПО. Эта идея неверна. Развертывание технологии должно управляться прежде всего проблемой, которую необходимо решить решить, а не технологией, которую кто-то пробует Вам впарить;).

Очень плохо, что из-за того, что компании часто пренебрегают упоминанием об ограничениях и недостатках их инструментальных средств, авторы иногда бывают не в состоянии обратить внимание читателей на следствия некоторых методик, которые они описывают. И журналы и книги иногда пренебрегают своей ответственностью, чтобы удостовериться, что практика программирования, которую они описывают, является приемлемой.

Программисту очень важно выбрать необходимый инструмент для своей работы. Это - ваша задача, чтобы разработать код, который работает теперь не только на одной специфической платформе, но также работает на разных платформах и системных конфигурациях. Ваш код должен быть хорошо документирован и поддержан другими программистами, участвующими в проекте. Ваш код должен следовать правилам, продиктованными операционной системой или стандартами, которые Вы используете. Отказ так делать может привести к проблемам в будущем, поскольку системы и программное обеспечение постоянно совершенствуются.

Недавние статьи в Microsoft Systems Journal и Visual Basic Programmer's Journal представили программистам на Visual Basic возможность использования функции API CreateThread, чтобы непосредственно поддерживать многопоточный режим под Visual Basic. После этого, один читатель пожаловался, что моя книга Visual Basic Programmer's Guide to the Win32 API является неполной, потому что я не описал в ней эту функцию и не продемонстрировал эту технологию. Эта статья - частично является ответом этому читателю, и частично - ответом на другие статьи, написанными на эту тему. Эта статья также является дополнением к главе 14 моей книги "Разработка ActiveX компонент на Visual Basic 5.0" относительно новых возможностей, обеспечиваемых Visual Basic 5.0 Service Pack 2.

**Быстрый обзор Многопоточности**

Если Вы уже хорошо разбираетесь в технологии многопоточного режима, то Вы можете пропустить этот раздел и продолжать чтение с раздела, названного "Что нового в Service Pack 2."

Каждый, кто использует Windows, знает, что Windows способно делать больше чем одну вещь одновременно. Может одновременно выполнять несколько программ, при одновременном проигрывании компакт-диска, посылке факса и пересылке файлов. Каждый программист знает (или должен знать) что ЦЕНТРАЛЬНЫЙ ПРОЦЕССОР компьютера может только выполнять одну команду одновременно (проигнорируем существование многопроцессорных машин). Как единственный ЦЕНТРАЛЬНЫЙ ПРОЦЕССОР может выполнять множество задач?

Это делается быстрым переключением между многими задачами. Операционная система содержит в памяти все программы, которые запущены в настоящий момент. Это позволяет ЦЕНТРАЛЬНОМУ ПРОЦЕССОРУ выполнять программы по очереди. Каждый раз происходит переключение между программами, при этом меняется содержимое внутренних регистров, включая указатель команды и указатель вершины стека. Каждая из таких "задач" называется потоком выполнения (thread of execution).

В простой многозадачной системе, каждая программа имеет емеет единственный поток. Это означает, что ЦЕНТРАЛЬНЫЙ ПРОЦЕССОР начинает выполнение команд в начале программы и продолжает следуя инструкциям в последовательности, определенной программой до тех пор, пока программа не завершается.

Скажем, программа имеет пять команд: B C D и E, которые выполняются последовательно (никаких переходов нет в этом примере). Когда приложение имеет один поток, команды будут всегда выполнять в точно том же самом порядке: A, B, C, D и E. Действительно, ЦЕНТРАЛЬНЫЙ ПРОЦЕССОР может потребовать времени для выполнения других команд в других программах, но они не будут влиять на это приложение, если не имеется конфликт над общими ресурсами системы, но это уже отдельная тема для разговора.

Продвинутая многопоточная операционная система типа Windows позволяет приложению выполнять больше чем один поток одновременно. Скажем, команда D в нашем типовом приложении могла создать новый поток, который стартовал командой B и далее выполнял последовательность команд C и E. Первый поток был бы все еще A, B, C, D, E, но когда команда D выполнится, возникнет новый поток, который выполнит команды бы B, C, E (здесь команды D уже не будет, иначе мы получим еще один поток).

В каком порядке будут следовать команды в этом приложении?

Это могло бы быть:

Thread 1 A B C D E

Thread 2 B C E

Или так:

Thread 1 A B C D E

Thread 2 B C E

Или этак:

Thread 1 A B C D E

Thread 2 B C E

Другими словами, когда Вы начинаете новый поток выполнения в приложении, Вы никогда не можете знать точный порядок, в котором команды в двух потоках выполнятся относительно друг друга. Два потока полностью независимы.

Почему - это проблема?

**Имитатор Многопоточности**

Рассмотрим проект MTDemo:

Проект содержит один модуль кода, в котором содержится две глобальных переменных:

' MTDemo - Multithreading Demo program

' Copyright © 1997 by Desaware Inc. All Rights Reserved

Option Explicit

Public GenericGlobalCounter As Long

Public TotalIncrements As Long

' Этот проект содержит одну форму - frmMTDemo1, которая содержит

' следующий код:

' MTDemo - Multithreading Demo program ' Copyright © 1997 by Desaware Inc. All Rights Reserved

Option Explicit

Dim State As Integer

' State = 0 - Idle

' State = 1 - Loading existing value

' State = 2 - Adding 1 to existing value

' State = 3 - Storing existing value

' State = 4 - Extra delay

Dim Accumulator As Long

Const OtherCodeDelay = 10

Private Sub Command1\_Click()

Dim f As New frmMTDemo1

f.Show

End Sub

Private Sub Form\_Load()

Timer1.Interval = 750 + Rnd \* 500

End Sub

Private Sub Timer1\_Timer()

Static otherdelay&

Select Case State

Case 0

lblOperation = "Idle"

State = 1

Case 1

lblOperation = "Loading Acc"

Accumulator = GenericGlobalCounter

State = 2

Case 2

lblOperation = "Incrementing"

Accumulator = Accumulator + 1

State = 3

Case 3

lblOperation = "Storing"

GenericGlobalCounter = Accumulator

TotalIncrements = TotalIncrements + 1

State = 4

Case 4

lblOperation = "Generic Code"

If otherdelay >= OtherCodeDelay Then

State = 0

otherdelay = 0

Else

otherdelay = otherdelay + 1

End If

End Select

UpdateDisplay

End Sub

Public Sub UpdateDisplay()

lblGlobalCounter = Str$(GenericGlobalCounter)

lblAccumulator = Str$(Accumulator)

lblVerification = Str$(TotalIncrements)

End Sub

Эта программа для моделирования многопоточного режима использует таймер и простой конечный автомат. Переменная State описывает пять команд, которые эта программа выполняет. State = 0 - неактивное состояние. State = 1 загружает локальную переменную глобальной переменной GenericGlobalCounter. State = 2 увеличивает на единицу локальную переменную. State = 3 запоминает результат в переменной GenericGlobalCounter и увеличивает переменную TotalIncrements (которая считает количество приращений переменной GenericGlobalCounter). State = 3 добавляет дополнительную задержку, представляющую собой время, затраченное на выполнение других команд в программе.

Функция UpdateDisplay обновляет три метки на форме, которые показывают текущее значение переменной GenericGlobalCounter, локального сумматора, и общего количества приращений.

Каждый сигнал таймера моделирует цикл ЦЕНТРАЛЬНОГО ПРОЦЕССОРА в текущем потоке. Если Вы запустите программу, то увидете, что значение переменной GenericGlobalCounter будет всегда точно равно переменной TotalIncrements, потому что переменная TotalIncrements показывает количество увеличений счетчика GenericGlobalCounter потоком.

Но что случится, когда Вы нажимаете кнопку Command1 и запустите второй экземпляр формы? Эта новая форма смоделирует второй поток.

Время от времени, команды выстроятся в линию таким образом, что обе формы загрузят одинаковое значение GenericGlobalCounter, увеличат и сохранят его. В результате, значение счетчика увеличится только на единицу, даже при том, что каждый поток полагал, что он независимо увеличивает значение счетчика. Другими словами, переменная была увеличена дважды, но значение увеличилось только на единицу. Если Вы запускаете несколько форм, то сразу заметите, что число приращений, представляемой переменной TotalIncrements, растет намного быстрее, чем счетчик GenericGlobalCounter.

Что, если переменная представляет объектный счет блокировки - который следит, когда объект должен быть освобожден? Что, если она представляет собой сигнал, который указывает, что ресурс находится в использовании?

Такая проблема может привести к появлению ресурсов, постоянно недоступных в системе, к объекту, блокируемому в памяти, или преждевременно освобожденному. Это может привести к сбоям приложения.

Этот пример был разработан, чтобы достаточно просто увидеть проблему, но попробуйте поэкспериментировать со значением переменной OtherCodeDelay. Когда опасный код относительно небольшой по сравнению со всей программой, проблемы появятся менее часто. Хотя это и звучит обнадеживающе, но истина состоит в следующем. Проблемы Многопоточного режима могут быть чрезвычайно неустойчивы и их трудно обнаружить. Это означает, что многопоточный режим требует осторожного подхода к проектированию приложения.

**Решение проблем Многопоточности**

Имеются два относительно простых способа избежать проблем многопоточного режима.

Избегайте всеобщего использования глобальных переменных.

Добавьте код синхронизации везде, где используются глобальные переменные.

Первый подход используется в основном в Visual Basic. Когда Вы включаете многопоточный режим в Visual Basic приложения, все глобальные переменные станут локальными для специфического потока. Это свойственно способу, с которым Visual Basic выполняет apartment model threading - подробнее об этом позднее.

Первоначальный выпуск Visual Basic 5.0 позволял использовать многопоточность только в компонентах, которые не имели никаких элементов пользовательского интерфейса. Так было потому что они не имели безопасного потока управления формами. Например: когда Вы создаете форму в Visual Basic, VB дает ей имя глобальной переменной (таким образом, если Вы имеете форму, именованную Form1, Вы можете непосредственно обращаться к ее методам, используя Form1.метод вместо того, чтобы объявить отдельную переменную формы). Этот тип глобальной переменной может вызывать проблемы многопоточного режима, которые Вы видели ранее. Имелись несомненно другие проблемы внутри управления формами.

С service pack 2, управление формами Visual Basic было сделано безопасным потоком. Это говорит о том, что каждый поток имеет собственную глобальную переменную для каждой формы, определенной в проекте.

**Что нового в Service Pack 2**

Сделав поток управления формами безопасным, Service pack 2 предоставил возможность с помощью Visual Basic создавать клиентские приложения, использующие многопоточный режим.

Приложение должно быть определено как программа ActiveX Exe с установкой запуска из Sub Main:

' MTDemo2 - Multithreading demo program

' Copyright © 1997 by Desaware Inc. All Rights Reserved

Option Explicit

Declare Function FindWindow Lib "user32" Alias "FindWindowA" \_

(ByVal lpClassName As String, ByVal lpWindowName As String) As Long

Sub Main()

Dim f As frmMTDemo2

' We need this because Main is called on each new thread

Dim hwnd As Long

hwnd = FindWindow(vbNullString, "Multithreading Demo2")

If hwnd = 0 Then

Set f = New frmMTDemo2

f.Show

Set f = Nothing

End If

End Sub

Первый раз программа загружает и отображает основную форму приложения. Подпрограмма Main должна выяснить, является ли это первым потоком приложения, поэтому этот код выполняется при старте каждого потока. Вы не можете использовать глобальную переменную, чтобы это выяснить, потому что Visual Basic apartment model хранит глобальные переменные специфическими для одиночного потока. В этом примере используется функция API FindWindow, чтобы проверить, была ли загружена основная форма примера. Имеются другие способы выяснить, является ли это основным потоком, включая использование объектов синхронизации системы - но это отдельная тема для разговора.

Многопоточный режим реализуется созданием объекта в новом потоке. Объект должен быть определен, используя модуль класса. В этом случае, простой модуль класса определяется следующим образом:

' MTDemo2 - Multithreading demo program

' Copyright © 1997 by Desaware Inc. All Rights Reserved

Option Explicit

Private Sub Class\_Initialize()

Dim f As New frmMTDemo2

f.Show

Set f = Nothing

End Sub

Мы можем установить переменную формы как nothing после того, как она создана, потому что после отображения формы она будет сохранена.

' MTDemo2 - Multithreading demo program

' Copyright © 1997 by Desaware Inc. All Rights Reserved

Option Explicit

Private Sub cmdLaunch1\_Click()

Dim c As New clsMTDemo2

c.DisplayObjPtr Nothing

End Sub

Private Sub cmdLaunch2\_Click()

Dim c As clsMTDemo2

Set c = CreateObject("MTDemo2.clsMTDemo2")

End Sub

Private Sub Form\_Load()

lblThread.Caption = Str$(App.ThreadID)

End Sub

Форма отображает идентификатор потока в метке на форме. Форма содержит две командные кнопки, одна из которых использует оператор New, другая -использует оператор CreateObject.

Если Вы запустите программу внутри среды Visual Basic, то увидите, что формы всегда создаются в одном и том же потоке. Это происходит, потому что среда Visual Basic поддерживает только одиночный поток. Если Вы скомпилируете и запустите программу, то увидите, что подход, использующий CreateObject создает и clsMTDemo2 и ее форму в новом потоке.

**Почему многопоточность**

Откуда вся суета относительно многопоточного режима, если он включает так много потенциальной опасности? Потому что, в некоторых ситуациях, многопоточный режим может значительно улучшать эффективность приложения. В некоторых случаях это может улучшать эффективность некоторых операций синхронизации типа ожидания завершения приложения. Это позволяет сделать архитектуру приложения более гибкой. Например, операция Add a long в форме MTDEMO2 со следующим кодом:

Private Sub cmdLongOp\_Click()

Dim l&

Dim s$

For l = 1 To 1000000

s = Chr$(l And &H7F)

Next l

End Sub

Запустите несколько экземпляров формы, используя кнопку cmdLaunch1. Когда Вы нажимаете на кнопку cmdLongOp на любой из форм, то увидите, что это действие замораживает операции на всех других формах. Так происходит, потому что все формы выполняются в одиночном потоке - и этот поток занят выполнением длинного цикла. Если Вы запустите несколько экземпляров формы кнопкой cmdLaunch2 и нажимете кнопку cmdLongOp на форму, то только эта форма будет заморожена - другие формы будут активными. Они выполняются в собственных потоках, и длинный цикл будет выполняться только в собственном потоке. Конечно, в любом случае, Вы вероятно не должны размещать длительные операции такого типа в ваших формах.

Дальше краткое резюме, когда важен многопоточный режим:

Сервер ActiveX EXE – без общих ресурсов.

Когда Вы имеете ActiveX EXE сервер, который Вы собираетесь совместно использовать среди нескольких приложений, многопоточный режим предотвращает приложения от нежелательных взаимодействий с друг другом. Если одно приложение выполняет длинную операцию на объекте в однопоточном сервере, другие приложения будут вытеснены, т.е. будут ждать, когда освободится сервер. Многопоточный режим рещает эту проблему. Однако, имеются случаи, где Вы можете хотеть использовать ActiveX EXE сервер, чтобы регулировать доступ к общедоступнному ресурсу (shared resource). Например, сервер stock quote, описанный в моей книге Developing ActiveX Components. В этом случае сервер stock quote выполняется в одиночном потоке и который доступен для всех приложений, использующих сервер по очереди.

Многопоточный клиент – выполняемый как ActiveX EXE сервер

Простая форма этого подхода продемонстрирована в приложении MTDEMO2. Этот подход используется, когда приложение поддерживает множественные окна, которые должны исходить из одного приложения, но работать полностью независимо. Интернет-браузер - хороший пример такого многопоточного клиента, где каждое окно выполняется в собственном потоке. Здесь следует обратить внимание на то, что многопоточный режим не должен использоваться как замена для хорошего событийно управляемого проекта.

Многопоточные серверы DLL или EXE

В архитектуре клиент-сервер, многопоточный режим может увеличить эффективность, если Вы имеете смесь длинных и коротких клиентских запросов. Будьте внимательным, хотя - если все ваши клиентские запросы имеют подобную длину, многопоточный режим может фактически замедлять среднее время ответа сервера! Никогда не принимайте на веру тот факт, что если ваш сервер является многопоточным, то обязательно его эффективность увеличится.

**Соглашение о потоках**

Верите или нет, но все это было введением. Часть этого материала является обзором материала, который описан в моей книге Developing ActiveX Components, другая часть материала описывает новую информацию для service pack 2.

Теперь, позволите задавать вопрос, который имеет отношение к многопоточному режиму, использующему COM (модель многокомпонентных объектов, на которой основаны не только все Visual Basic объекты, но и другие windows приложения, использующие технологии OLE).

Дано:

Многопоточный режим является потенциально опасным вообще, и особенно попытки многопоточного кодирования приложений, которые не разработаны для поддержки многопоточного режима, скорее всего приведут к фатальным ошибкам и сбоям системы.

Вопрос:

Как это возможно, что Visual Basic позволяет Вам создавать объекты и использовать их с одиночными и многопоточными средами безотносительно к тому, разработаны ли они для одиночного или многопоточного использования?

Другими словами - Как многопоточные Visual Basic приложения могут использовать объекты, которые не разработаны для безопасного выполнения в многопоточной среде? Как могут другие многопоточные приложения использовать однопоточные объекты Visual Basic?

Коротко: как COM поддерживает потоки?

Если Вы знаете COM, то Вы знаете, что COM определяет структуру соглашения. Объект COM соглашается следовать некоторым правилам так, чтобы этим можно было успешно пользоваться из любого приложения или объекта, который поддерживает COM.

Большинство людей сначала думает о интерфейсной части соглашения - о методах и свойствах, которые предоставляет объект.

Но Вы не можете не знать того, что COM также определяет поточность как часть соглашения. И подобно любой части соглашения COM - если Вы нарушаете эти условия, то будете иметь проблемы. Visual Basic, естественно, скрывает от Вас большинство механизмов COM, но чтобы понять как использовать многопоточность в Visual Basic, Вы должны разобраться COM модели потоков.

Модель одиночного потока:

Однопоточный сервер - самый простой тип реализации сервера. И самый простой для понимания. В этом случае EXE сервер выполняется в одиночном потоке. Все объекты создаются в этом потоке. Все вызовы методов каждого объекта, поддерживаемого сервером должны прибыть в этот поток.

Но что будет, если клиент выполняется в другом потоке? В том случае, для объекта сервера должен быть создан промежуточный объект (proxy object). Этот промежуточный объект выполняется в потоке клиента и отражает методы и свойства фактического объекта. Когда вызывается метод промежуточного объекта, он выполняет операции, необходимые для подключению к потоку объекта, а затем вызывает метод фактического объекта, используя параметры, переданные к промежуточному объекту. Естественно, что этот подход требует значительного времени на выполнение задачи, однако он позволяет выполнить все соглашения. Этот процесс переключения потоков и пересылки данных от промежуточного объекта к фактическому объекту и обратно называется marshalling. Эта тема обсуждается в главе 6 моей книги Developing ActiveX Components.

В случае DLL серверов, одиночная потоковая модель требует, чтобы все объекты в сервере создавались и вызывались в том же самом потоке что и первый объект, созданный сервером.

Модель Apartment Threading

Обратите внимание, что модель Apartment Threading как определено COM не требует, чтобы каждый поток имел собственный набор глобальных переменных. Visual Basic таким образом реализует модель Apartment Threading. Модель Apartment Threading декларирует, что каждый объект может быть создан в собственном потоке, однако, как только объект создан, его методы и свойства могут вызываться только тем же самым потоком, которая создал объект. Если объект другого потока захочет иметь доступ к методам этого объекта, то он должен действовать через промежуточный объект.

Такая модель относительно проста для реализации. Если Вы устраняете глобальные переменные (как делает Visual Basic), модель Apartment Threading автоматически гарантирует безопасность потока - так как каждый объект действительно выполняется в собственном потоке, и благодаря отсутствию глобальных переменных, объекты в разных потоках не взаимодействуют друг с другом.

Модель свободных потоков

Модель свободных потоков (Free Threading Model) заключается в следующем.. Любой объект может быть создан в любом потоке. Все методы и свойства любого объекта могут быть вызываны в любое время из любого потока. Объект принимает на себя всю ответственность за обработку любой необходимой синхронизации.

Это самая трудная в реализации модель, так как требуется, чтобы всю синхронизацию обрабатывал программист. Фактически до недавнего времени, технология OLE непосредственно не поддерживала эту модель! Однако, с тех пор marshalling никогда не требуется и это наиболее эффективная модель потоков.

Какую модель поддерживает ваш сервер?

Как приложение или сама Windows узнает, которую модель потоков использует сервер? Эта информация включена в системный реестр (registry). Когда Visual Basic создает объект, он проверяет системный реестр, чтобы определить, в каких случаях требуется использовать промежуточный объект (proxy object) и в каких - marshalling.

Эта проверка является обязанностью клиента и необходима для строгой поддержки требований многопоточности для каждого объекта, которого он создает.

Функция API CreateThread

Теперь давайте посмотрим, как с Visual Basic может использоваться функция API CreateThread. Скажем, Вы имеете класс, что Вы хотите выполненять в другом потоке, например, чтобы выполнить некоторую фоновую операцию. Характерный класс такого типа мог бы иметь следующий код (из примера MTDemo 3):

' Class clsBackground

' MTDemo 3 - Multithreading example

' Copyright © 1997 by Desaware Inc. All Rights Reserved

Option Explicit

Event DoneCounting()

Dim l As Long

Public Function DoTheCount(ByVal finalval&) As Boolean

Dim s As String

If l = 0 Then

s$ = "In Thread " & App.threadid

Call MessageBox(0, s$, "", 0)

End If

l = l + 1

If l >= finalval Then

l = 0

DoTheCount = True

Call MessageBox(0, "Done with counting", "", 0)

RaiseEvent DoneCounting

End If

End Function

Класс разработан так, чтобы функция DoTheCount могла неоднократно вызываться из непрерывного цикла в фоновом потоке. Мы могли бы поместить цикл непосредственно в сам объект, но вы вскоре увидите, что были веские причины для проектирования объекта как показано в примере. При первом вызове функции DoTheCount появляется MessageBox, в котором показан идентификатор потока, по которому мы можем определить поток, в котором выполняется код. Вместо VB команды MessageBox используется MessageBox API, потому что функция API, как известно, поддерживает безопасное выполнение потоков. Второй MessageBox появляется после того, как закончен подсчет и сгенерировано событие, которое указывает, что операция закончена.

Фоновый поток запускается при помощи следующего кода в форме frmMTDemo3: Private Sub cmdCreateFree\_Click()

Set c = New clsBackground

StartBackgroundThreadFree c

End Sub

Функция StartBackgroundThreadFree определена в модуле modMTBack следующим образом:

Declare Function CreateThread Lib "kernel32" \_

(ByVal lpSecurityAttributes As Long, ByVal \_

dwStackSize As Long, ByVal lpStartAddress As Long, \_

ByVal lpParameter As Long, ByVal dwCreationFlags \_

As Long, lpThreadId As Long) As Long

Declare Function CloseHandle Lib "kernel32" \_

(ByVal hObject As Long) As Long

' Start the background thread for this object

' using the invalid free threading approach.

Public Function StartBackgroundThreadFree (ByVal qobj As clsBackground)

Dim threadid As Long

Dim hnd&

Dim threadparam As Long

' Free threaded approach

threadparam = ObjPtr(qobj)

hnd = CreateThread(0, 2000, AddressOf \_

BackgroundFuncFree, threadparam, 0, threadid)

If hnd = 0 Then

' Return with zero (error)

Exit Function

End If

' We don't need the thread handle

CloseHandle hnd

StartBackgroundThreadFree = threadid

End Function

Функция CreateThread имеет шесть параметров:

lpSecurityAttributes - обычно устанавливается в нуль, чтобы использовать заданные по умолчанию атрибуты защиты.

dwStackSize - размер стека. Каждый поток имеет собственный стек.

lpStartAddress - адрес памяти, где стартует поток. Он должен быть равен адресу функции в стандартном модуле, полученном при использовании оператора AddressOf.

lpParameter - long 32 разрядный параметр, который передается функции, запускающей новый поток.

dwCreationFlags - 32 бит переменная флагов, которая позволяет Вам управлять запуском потока (активный, приостановленный и т.д.). Подробнее об этих флагах можно почитать в Microsoft's online 32 bit reference.

lpThreadId - переменная, в которую загружается уникальный идентификатором нового потока.

Функция возвращает дескриптор потока.

В этом случае мы передаем указатель на объект clsBackground, который мы будем использовать в новом потоке. ObjPtr восстанавливает значение указателя интерфейса в переменную qobj. После создания потока закрывается дескриптор при помощи функции CloseHandle. Это действие не завершает поток, - поток продолжает выполняться до выхода из функции BackgroundFuncFree. Однако, если мы не закрыли дескриптор, то объект потока будет существовать даже после выхода из функции BackgroundFuncFree. Все дескрипторы потока должны быть закрыты и при завершении потока система освобождает занятые потоком ресурсы.

Функция BackgroundFuncFree имеет следующий код:

' A free threaded callback.

' A free threaded callback.

' This is an invalid approach, though it works

' in this case.

Public Function BackgroundFuncFree(ByVal param As IUnknown) As Long

Dim qobj As clsBackground

Dim res&

' Free threaded approach

Set qobj = param

Do While Not qobj.DoTheCount(100000)

Loop

' qobj.ShowAForm ' Crashes!

' Thread ends on return

End Function

Параметром этой функции является- указатель на интерфейс (ByVal param As IUnknown). При этом мы можем избежать неприятностей, потому что под COM каждый интерфейс основывается на IUnknown, так что такой тип параметра допустим независимо от типа интерфейса, передаваемого функции. Мы, однако, должны немедленно определить param как тип объекта, чтобы затем его использовать. В этом случае qobj установливается как объект clsBackground, который был передан к объекту StartBackgroundThreadFree.

Функция затем выполняет бесконечный цикл, в течение которого может выполняться любая требуемая операция, в этом случае повторный счет. Подобный подход мог бы использоваться здесь, чтобы выполнить операцию ожидания, которая приостанавливает поток пока не произойдкт системное событие (типа завершения процесса). Поток затем мог бы вызвать метод класса, чтобы сообщить приложению, что событие произошло.

Доступ к объекту qobj чрезвычайно быстр из-за использования подхода свободного потока (free threading) - никакая переадресация (marshalling) при этом не используется.

Обратите внимание на то, что если Вы попробуете использовать объект clsBackground, который показывает форму, то это приведет к сбоям приложения. Обратите также внимание на то, что событие завершения никогда не происходит в клиентской форме. Действительно, даже Microsoft Systems Journal, который описывает этот подход, содержит очень много предупреждений о том, что при использовании этого подхода есть некоторые вещи, которые не работают.

Некоторые разработчики, кто пробовали развертывать приложения, применяющие этот тип многопоточности, обнаружили, что их приложения вызывают сбои после обновления к VB5 service pack 2.

Является ли это дефектом Visual Basic?

Означает ли это, что Microsoft не обеспечила совместимость?

Ответ на оба вопроса: Нет

Проблема не в Microsoft или Visual Basic.

Проблема состоит в том, что вышеупомянутый код является мусором.

Проблема проста - Visual Basic поддерживает объекты и в модели одиночного потока и в apartment model. Позвольте мне перефразировать это: объекты Visual Basic являются COM объектами и они,согласно COM соглашению, будут правильно работать как в модели одиночного потока так и в apartment model. Это означает, что каждый объект ожидает, что любые вызовы методов будут происходить в том же самом потоке, который создал объект.

Пример, показанный выше, нарушает это правило.

Это нарушает соглашение COM.

Что это означает?

Это означает, что поведение объекта подчиненно изменениям, так как Visual Basic постоянно модифицируется.

Это означает, что любая попытка объекта обратиться к другим объектам или формам может потерпеть неудачу и что причины отказов могут изменяться, поскольку эти объекты модифицируются.

Это означает, что даже код, который сейчас работает, может внезапно вызвыть сбой, поскольку другие объекты добавляются, удаляются или изменяются.

Это означает, что невозможно характеризовать поведение приложения или предсказать, будет ли оно работать или может ли работать в любой данной среде.

Это означает, что невозможно предсказать, будет ли код работать на любой данной системе, и что его поведение может зависеть от используемой операционной, числа используемых процессоров и других проблем конфигурации системы.

Вы видите, что как только Вы нарушаете соглашение COM, Вы больше не защищены теми возможностями COM, которые позволяют объектам успешно взаимодействовать друг с другом и с клиентами.

Этот подход является программной алхимией. Это безответственно и ни один программист не должен когда-либо использовать это. Точка.

Обратно к функции API CreateThread

Теперь, когда я показал Вам, почему подход к использованию CreateThread API, показанный в некоторых статьях, является мусором, я покажу Вам, как можно использовать эту API функцию безопасно. Прием прост - Вы должны просто твердо придержаться соглашения COM о потоках. Это займет немного больше времени и усилий, но практика показала, что получаются очень надежные результаты.

Пример MTDEMO3 демонстрирует этот подход в форме frmMTDemo3, имеющей код, который запускает класс фона в apartment model следующим образом:

Private Sub cmdCreateApt\_Click()

Set c = New clsBackground

StartBackgroundThreadApt c

End Sub

Пока это выглядит очень похоже на подход свободных потоков. Вы создаете экземпляр класса и передаете его функции, которая запускает фоновый поток. В модуле modMTBack появляется следующий код:

' Structure to hold IDispatch GUID

Type GUID

Data1 As Long

Data2 As Integer

Data3 As Integer

Data4(7) As Byte

End Type

Public IID\_IDispatch As GUID

Declare Function CoMarshalInterThreadInterfaceInStream Lib \_

"ole32.dll" (riid As GUID, ByVal pUnk As IUnknown, \_

ppStm As Long) As Long

Declare Function CoGetInterfaceAndReleaseStream Lib \_

"ole32.dll" (ByVal pStm As Long, riid As GUID, \_

pUnk As IUnknown) As Long

Declare Function CoInitialize Lib "ole32.dll" (ByVal \_

pvReserved As Long) As Long

Declare Sub CoUninitialize Lib "ole32.dll" ()

' Start the background thread for this object

' using the apartment model

' Returns zero on error

Public Function StartBackgroundThreadApt(ByVal qobj As clsBackground)

Dim threadid As Long

Dim hnd&, res&

Dim threadparam As Long

Dim tobj As Object

Set tobj = qobj

' Proper marshaled approach

InitializeIID

res = CoMarshalInterThreadInterfaceInStream (IID\_IDispatch, qobj, threadparam)

If res <> 0 Then

StartBackgroundThreadApt = 0

Exit Function

End If

hnd = CreateThread(0, 2000, AddressOf BackgroundFuncApt, threadparam, 0, threadid)

If hnd = 0 Then

' Return with zero (error)

Exit Function

End If

' We don't need the thread handle

CloseHandle hnd

StartBackgroundThreadApt = threadid

End Function

Функция StartBackgroundThreadApt немного более сложна чем ее эквивалент при применении подхода свободных потоков. Первая новая функция называется InitializeIID. Она имеет следующий код:

' Initialize the GUID structure

Private Sub InitializeIID()

Static Initialized As Boolean

If Initialized Then Exit Sub

With IID\_IDispatch

.Data1 = &H20400

.Data2 = 0

.Data3 = 0

.Data4(0) = &HC0

.Data4(7) = &H46

End With

Initialized = True

End Sub

Вы видите, нам необходим идентификатор интерфейса - 16 байтовая структура, которая уникально определяет интерфейс. В частности нам необходим идентификатор интерфейса для интерфейса IDispatch (подробная информация относительно IDispatch может быть найдена в моей книге Developing ActiveX Components). Функция InitializeIID просто инициализирует структуру IID\_IDISPATCH к корректным значениям для идентификатора интерфейса IDispatch. Значение Это значение получается с помощью использования утилиты просмотра системного реестра.

**Почему нам необходим этот идентификатор?**

Потому что, чтобы твердо придерживаться соглашения COM о потоках, мы должны создать промежуточный объект (proxy object) для объекта clsBackground. Промежуточный объект должен быть передан новому потоку вместо первоначального объекта. Обращения к новому потоку на промежуточном объекте будут переадресованы (marshaled) в текущий поток.

CoMarshalInterThreadInterfaceInStream выполняет интересную задачу. Она собирает всю информацию, необходимую при создании промежуточного объекта, для определенного интерфейса и загружает ее в объект потока (stream object). В этом примере мы используем интерфейс IDispatch, потому что мы знаем, что каждый класс Visual Basic поддерживает IDispatch и мы знаем, что поддержка переадресации (marshalling) IDispatch встроена в Windows - так что этот код будет работать всегда. Затем мы передаем объект потока (stream object) новому потоку. Этот объект разработан Windows, чтобы быть передаваемым между потоками одинаковым способом, так что мы можем безопасно передавать его функции CreateThread. Остальная часть функции StartBackgroundThreadApt идентична функции StartBackgroundThreadFree.

Функция BackgroundFuncApt также сложнее чем ее эквивалент при использовании модели свободных потоков и показана ниже:

' A correctly marshaled apartment model callback.

' This is the correct approach, though slower.

Public Function BackgroundFuncApt(ByVal param As Long) As Long

Dim qobj As Object

Dim qobj2 As clsBackground

Dim res&

' This new thread is a new apartment, we must

' initialize OLE for this apartment

' (VB doesn't seem to do it)

res = CoInitialize(0)

' Proper apartment modeled approach

res = CoGetInterfaceAndReleaseStream(param, IID\_IDispatch, qobj)

Set qobj2 = qobj

Do While Not qobj2.DoTheCount(10000)

Loop

qobj2.ShowAForm

' Alternatively, you can put a wait function here,

' then call the qobj function when the wait is satisfied ' All calls to CoInitialize must be balanced

CoUninitialize

End Function

Первый шаг должен инициализировать подсистему OLE для нового потока. Это необходимо для переадресации (marshalling) кода, чтобы работать корректно. CoGetInterfaceAndReleaseStream создает промежуточный объект для объекта clsBackground и реализует объект потока (stream object), используемый для передачи данных из другого потока. Интерфейс IDispatch для нового объекта загружается в переменную qobj. Теперь возможно получить другие интерфейсы - промежуточный объект будет корректно переадресовывать данные для каждого интерфейса, который может поддерживать.

Теперь Вы можете видеть, почему цикл помещен в эту функцию вместо того, чтобы находиться непосредственно в объекте. Когда Вы впервые вызовите функцию qobj2.DoTheCount, то увидите, что код выполняется в начальном потоке! Каждый раз, когда Вы вызываете метод объекта, Вы фактически вызываете метод промежуточного объекта. Ваш текущий поток приостанавливается, запрос метода переадресовывается первоначальному потоку и вызывается метод первоначального объекта в той же самом потоке, который создал объект. Если бы цикл был в объекте, то Вы бы заморозили первоначальный поток.

Хорошим результатом применения этого подхода является то, что все работает правильно. Объект clsBackground может безопасно показывать формы и генерировать события. Недостатком этого подхода является, конечно, его более медленное исполнение. Переключение потоков и переадресация (marshalling) - относительно медленные операции. Вы фактически никогда не захотите выполнять фоновую операцию как показано здесь.

Но этот подход может чрезвычайно хорошо работать, если Вы можете помещать фоновую операцию непосредственно в функцию BackgroundFuncApt! Например: Вы могли бы иметь фоновый поток, выполняющий фоновые вычисления или операцию ожидания системы. Когда они будут завершены, вы можете вызывать метод объекта, который сгенерирует событие в клиенте. Храня количество вызовов метода, небольшое относительно количества работы, выполняемой в фоновой функции, Вы можете достигать очень эффективных результатов.

Что, если Вы хотите выполнить фоновую операцию, которая не должна использовать объект? Очевидно, проблемы с соглашением COM о потоках исчезают. Но появляются другие проблемы. Как фоновый поток сообщит о своем завершении приоритетному потоку? Как они обмениваются данными? Как два потока будут синхронизированы? Все это возможно выполнить с помощью соответствующих вызовов API. В моей книге Visual Basic 5.0 Programmer's Guide to the Win32 API имеется информации относительно объектов синхронизации типа Событий, Mutexes, Семафоров и Waitable Таймеров.

Эта книга также включает примеры файлов отображаемых в память, которые могут быть полезны при обмене данных между процессами. Вы сможете использовать глобальные переменные, чтобы обмениваться данные, но надо знать, что такое поведение не гарантируется Visual Basic(другими словами, даже если это сейчас работает, не имеется никаких гарантий, что это будет работать в будущем). В этом случае я мог бы предложить Вам использовать для обмена данными методики, основанные на API. Однако, преимуществом показанного здесь подхода, основанного на объектах, является то, что этот подход делает проблему обмена данными между потоками тривиальной, просто делайте это через объект.

**Заключение**

Я однажды услышал от опытного программиста под Windows, что OLE является самой трудной технологией, которой он когда-либо обучался. Я с этим согласен. Это очень обширная тема, и некоторые части этой технологии очень трудно понять. Visual Basic, как всегда, скрывает от Вас много сложностей.

Имеется сильное искушение, чтобы пользоваться преимуществом продвинутых методов типа многопоточного режима, используя подход "tips and techniques". Это искушение поощрено некоторыми статьями, которые иногда представляют специфическое решение, приглашая Вас вырезать и вставить (cut and past) их методики в ваши собственные приложения.

Когда я писал книгу Visual Basic Programmer's Guide to the Windows API, я выступал против такого подхода к программированию. Я чувствовал, что вообще безответственно включать в приложение код, который Вы не понимаете, и что реальное знание, которое так тяжело получить, стоит затраченных усилий.

Таким образом мои книги по API были разработаны, чтобы обеспечить не быстрые ответы и простые решения, а чтобы обучить использованию API к такой степени, что программисты могли бы интеллектуально правильно применять даже наиболее продвинутые методы. Я применил это тот же самый подход к моей книге Developing ActiveX Components, которая требует много времени для обсуждения принципов ActiveX, COM и объектно-ориентированного программирования перед описанием подробностей реализации этой технологии.

Многое из моей карьеры на ниве Visual Basic и многое из деятельности в фирме Desaware, основано на обучении Visual Basic программистов продвинутым методам. Читатель, кто вдохновил меня на написание этой статьи, критикуя меня за сдерживание технологии многопоточности, пропустил точку.

Да, я обучаю и демонстрирую продвинутые методы программирования - но я пытаюсь никогда не пропустить большую картинку. Продвинутые методы, которым я обучаю, должны быть непротиворечивы с правилами и спецификациями Windows. Они должны быть такими безопасными, насколько это возможно. Они должны быть поддерживаемыми в конечном счете. Они не должны разрушаться, когда изменяются Windows или Visual Basic.

Я могу требовать только частичного успеха, так как Microsoft может изменить правила игры всякий раз, когда им покажется, что это необходимо. Но я всегда помню об этом и пробую предупреждать людей, когда я думаю, что могу протолкнуть ограничение.

Я надеюсь, что приведенное здесь обсуждение многопоточного режима показывает опасности применения "простых методов" без хорошего понимания основной технологии.

Я не могу обещать, что использование apartment model версии CreateThread является абсолютно корректным, но мое понимание проблемы и опыт показывают, что это безопасно.

Могут иметься другие факторы, которые я пропустил. OLE - действительно сложная вещь и модули OLE DLL и сам Visual Basic подвержены изменениям. Я только могу утверждать, что лучшее из моего знания - код, который я здесь показал, удовлетворяет правилам COM и что эмпирическое доказательство показывает, что Visual Basic runtime 5 0's является достаточно безопасным для выполнения фонового кода потока в стандартном модуле.