UNIVERSIDADE DO ESTADO DA BAHIA – UNEB
LABORATÓRIO DE ORIENTAÇÃO A OBJETOS - OOLAB

Evitando conflitos em aplicações multi-thread no Delphi/Kylix

Edmilson dos Santos de Jesus
Salvador, março de 2002.

ÍNDICE

Agradecimentos

Quero agradecer a todos que contribuíram direta e indiretamente na elaboração desse artigo. Aos colegas do OOLAB (Laboratório de Orientação a Objetos – UNEB), pela força que me deram, ao Mestre Jorge Farias pela orientação, a Borland por ter feito um Help tão preciso para o Kylix e o Delphi, a Andrew S. Tanenbaum, por ser tão claro e direto no livro Sistemas Operacionais Modernos.

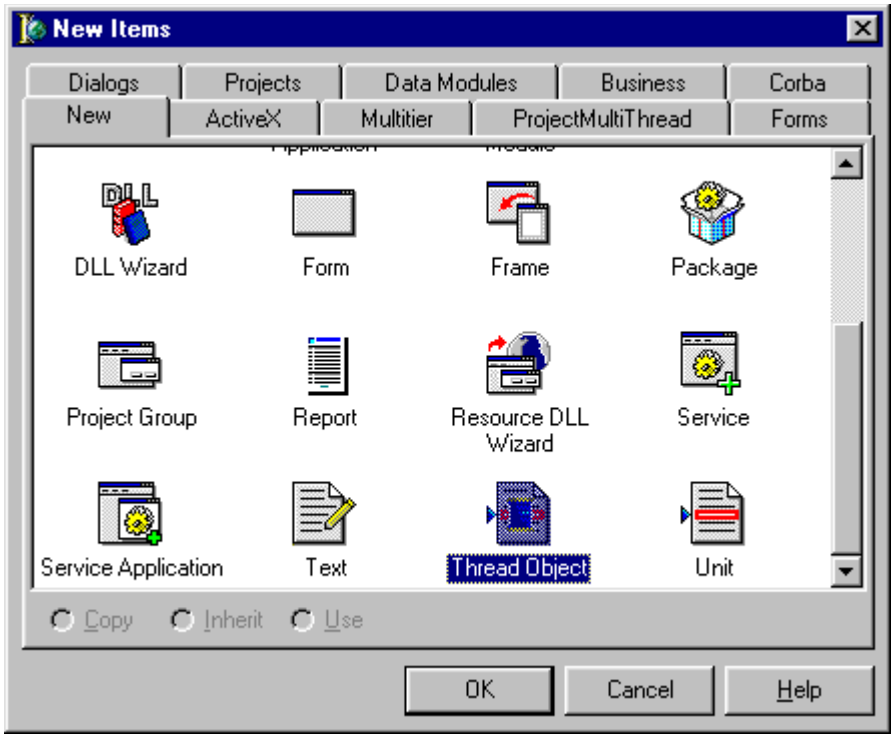
Os obstáculos existem para que sejam vencidos. E afinal o que seria da vida sem os obstáculos?

Aplicações multi-thread

Aplicações Multi-thread são aplicações que incluem várias linhas simultâneas de execução. Com somente uma thread a aplicação deve parar sua execução quando estiver esperando por um processo lento tal como leitura de disco, comunicação com outra máquina, ou exibição de conteúdo multimídia. Com múltiplas threads a mesma aplicação pode continuar executando as outras threads enquanto a thread lenta espera pelo resultado de um processo.

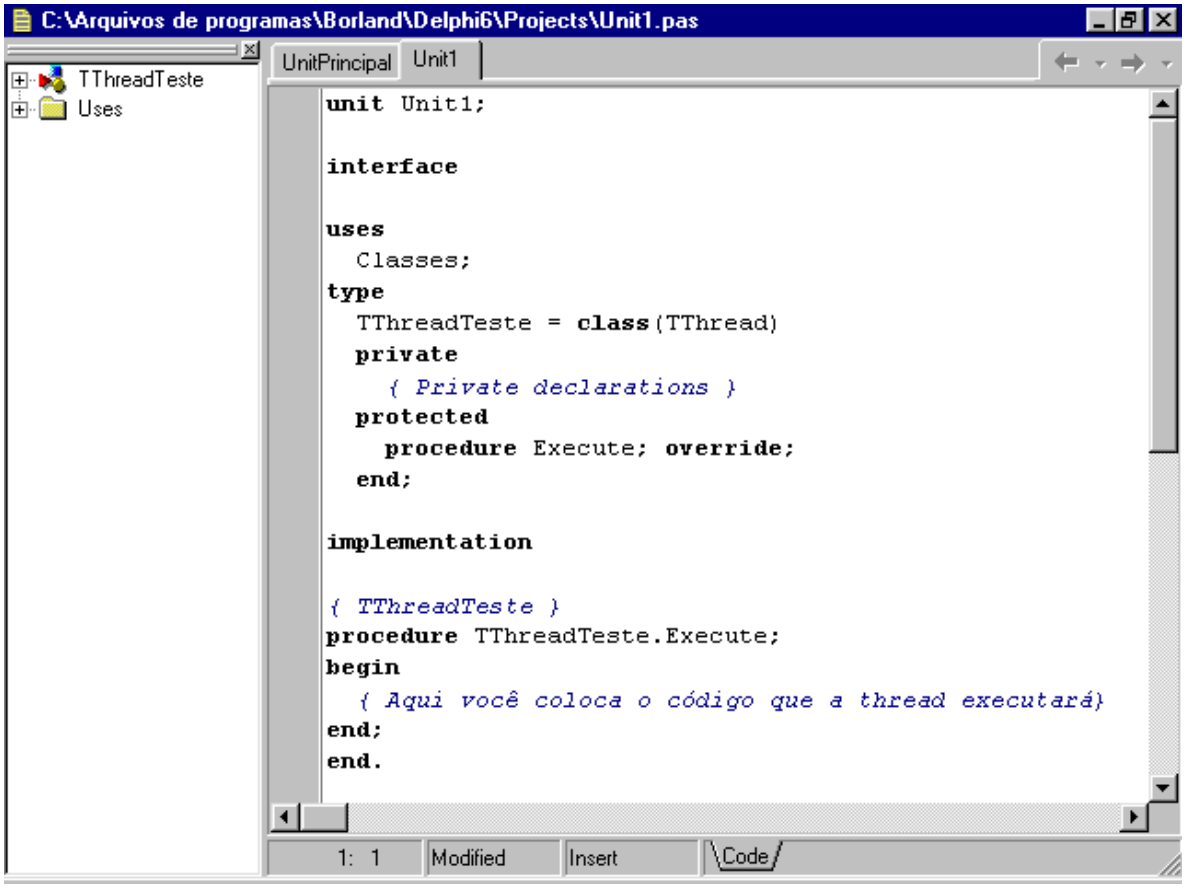
Para construir aplicações multi-thread no Delphi/Kylix você deve criar uma nova classe descendente de TThread, e cada instância dessa nova classe será uma linha de execução que alocará tempo de CPU para o seu processamento.

Para criar uma nova classe descendente de TThread selecione no menu principal do Delphi File | New, ou no caso do Kylix File | New | Other, na caixa de diálogo que aparece selecione Thread Object, deverá aparecer uma janela para você colocar o nome da classe que será criada.



Depois que você colocar o nome, o Delphi/Kylix automaticamente criará uma nova unit para a implementação da thread.

Veja o exemplo abaixo, o corpo da unit gerada pelo Delphi para a classe TThreadTeste.



Depois disso você pode colocar o código que a thread executará no método Execute. Para utilizar a classe criada em outras units basta acrescentar o nome da unit na clausula **uses** das outras units, e pode usar classe que você criou. Veja abaixo o exemplo do uso da classe TThreadTeste.

```
unit UnitPrincipal;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Unit1;

type
  TFormPrincipal = class(TForm)
    ButtonCriarThread: TButton;
    procedure ButtonCriarThreadClick(Sender: TObject);
```

```
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormPrincipal: TFormPrincipal;
  Teste:TThreadTeste; //aqui declaramos a variável Teste do tipo TThreadTeste

implementation

{$R *.dfm}

procedure TFormPrincipal.ButtonCriarThreadClick(Sender: TObject);
begin

  Teste:=TThreadTeste.Create(False); //Aqui instanciamos a nova thread

end;

end.
```

Evitando conflitos com múltiplas threads

Quando usamos objetos da VCL e da CLX em uma aplicação multi-thread, não temos garantia de que os métodos e propriedades desses objetos serão acessados apenas por uma thread de cada vez, o código da aplicação é que deve garantir isso. Esses métodos e propriedades podem executar ações sobre a memória a qual não está protegida da ação de outras threads.

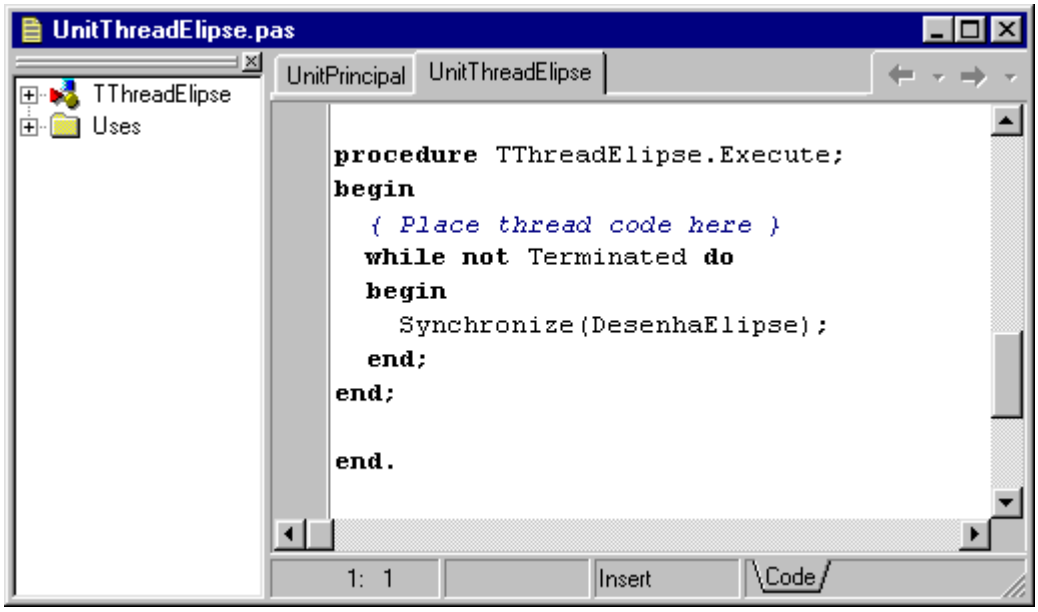
Vamos supor que temos uma aplicação onde várias threads acessam a mesma lista para inserir ou remover itens da mesma, obvio que duas thread não podem inserir ou remover itens da lista ao mesmo tempo, se isso ocorrer podem acontecer erros inesperados, ou pior, pode não ocorrer erro algum e o programa não funcionar corretamente, nesse caso o resultado final do processamento dependerá de quem executou quando, isso é chamado de **condição de corrida**. Para evitar conflitos, as threads devem acessar os objetos de forma sincronizada.

A maioria dos métodos que acessam objetos da VCL ou da CLX e atualizam os formulários devem ser chamados somente pela thread principal ou utilizando um objeto de sincronização tal como os objetos instanciados a partir das classes TMultiReadExclusiveWriteSynchronizer ou TCriticalSection.

Utilizando a Thread Principal através do método Synchronize

Se você acessa os métodos e propriedades dos objetos a partir de uma única thread você não precisa se preocupar com erros causados por acesso simultâneo, como valores lidos incorretamente ou violação de acesso.

Para fazer com que um método de outra thread use a thread principal para executar uma ação, você deve utilizar o método Synchronize. Para fazer isso crie uma rotina separada que executa a ação desejada, e depois chame-a de dentro da thread utilizando o método Synchronize e passando como parâmetro o nome da rotina que você criou, veja o exemplo abaixo.



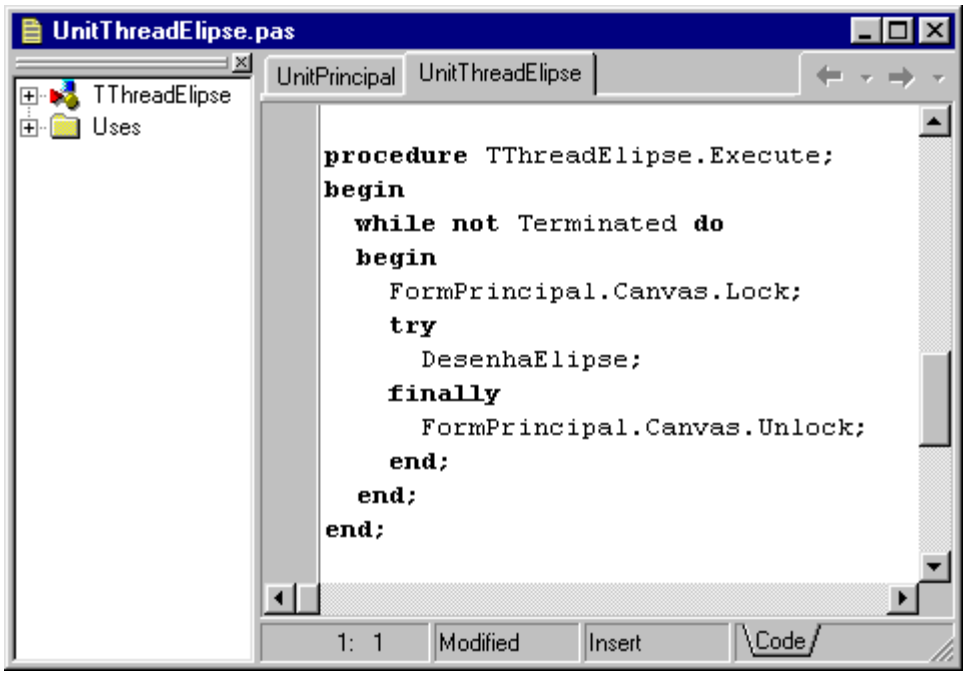
Synchronize causa uma chamada específica ao método passado como parâmetro utilizando a thread principal, evitando conflitos entre as threads. Synchronize espera a thread principal entrar na mensagem de loop e então executa o método passado como parâmetro.

OBS: Porque o método Synchronize utiliza a mensagem de loop ele não pode ser utilizado em uma aplicação console. Você deve utilizar outros mecanismos, tal como seção crítica, para proteger o acesso a objetos em aplicações console.

Evite na medida do possível o uso de Synchronize, isso deverá aumentar a performance da sua aplicação pois muitas vezes você não precisa esperar a thread principal entrar na mensagem de loop para executar o método de outra thread.

Usando o Lock de Objetos

Alguns objetos tem proteção própria para evitar que mais de uma thread utilize a instância desses objetos ao mesmo tempo. Por exemplo o TCanvas tem o método **Lock** que evita que outras threads acessem o canvas até que o método **Unlock** seja chamado.



Você deve proteger o Canvas com um **Lock** em todas as chamadas que o utilizam e depois você deve chamar o método **Unlock** para liberá-lo para outras threads.

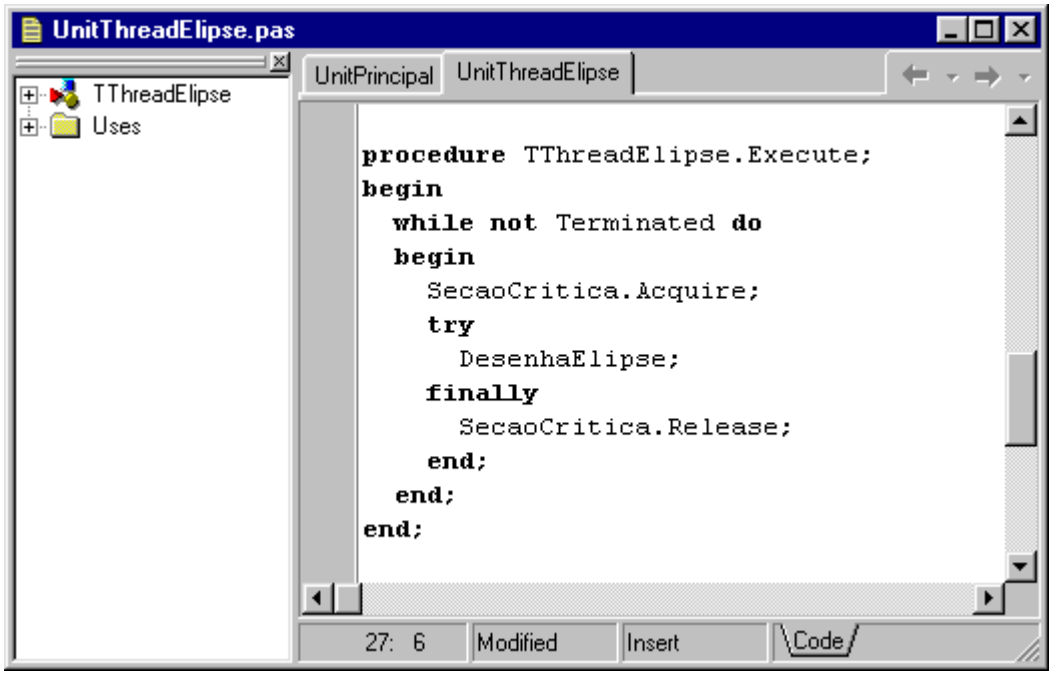
Quando o método Lock é chamado ele verifica se o Canvas já está bloqueado e se estiver Lock espera até que o Canvas seja liberado através da chamada ao método Unlock pela mesma thread que o bloqueou.

Protegendo seções críticas através da classe TCriticalSection

Se um objeto não possui método próprio para proteção você pode utilizar a classe TCriticalSection para proteger uma seção crítica.

Seção crítica é uma parte do programa cujo processamento pode levar à ocorrência de **condições de corrida**. Para utilizar seção crítica você deve criar uma instância global da classe TCriticalSection, acrescentando na cláusula **uses** a unit StdCtrls.

TCriticalSection tem dois métodos, Acquire (que impede que as outras threads entrem na seção) e Release (que remove o bloqueio).

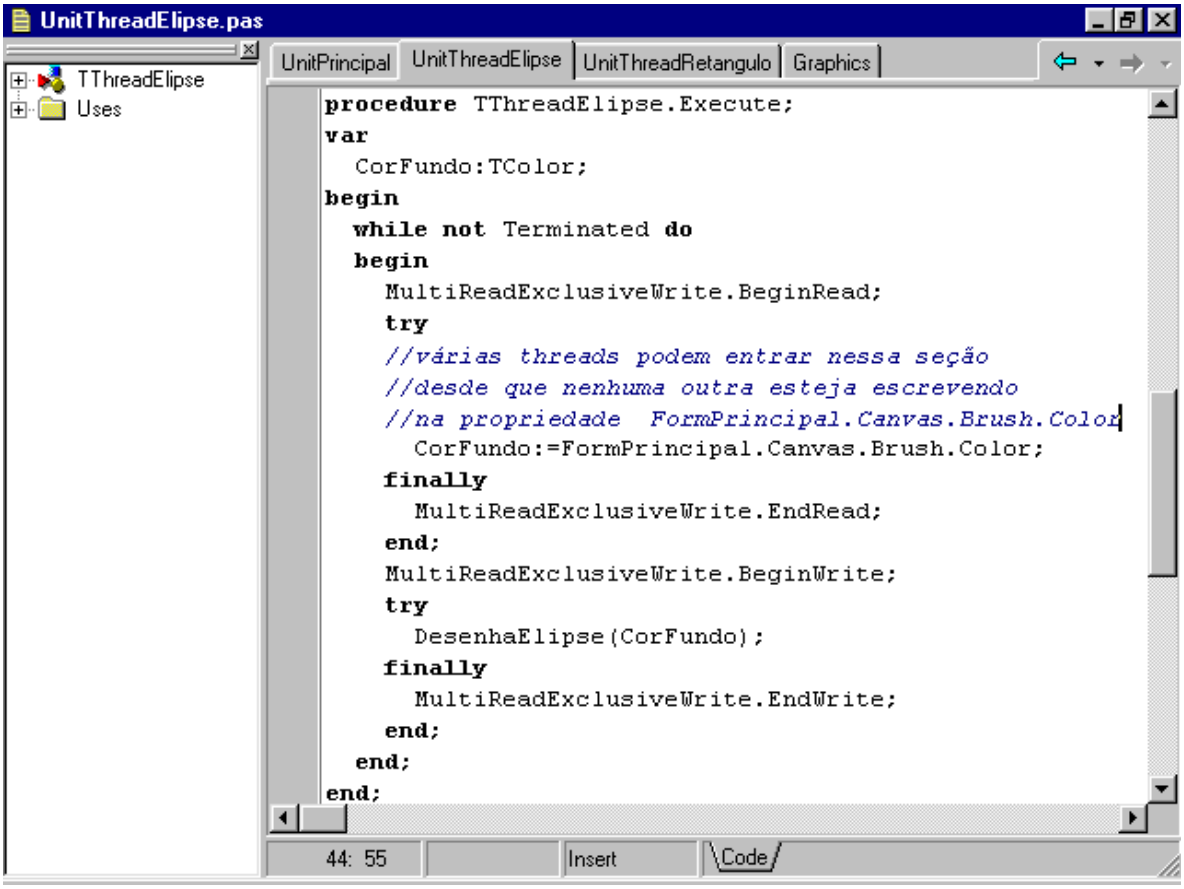


Utilizando múltipla leitura e exclusiva escrita.

Existe uma situação especial onde muitas threads precisam ler o valor de determinada região de memória e que raramente uma thread escreve nessa região, nesse caso podemos utilizar uma instância global da classe **TMultiReadExclusiveWriteSynchronizer.**

A classe TMultiReadExclusiveWriteSynchronizer tem os métodos **BeginRead** e **EndRead** para proteger um bloco que pode ser acessado por várias threads desde que nenhuma outra thread esteja dentro de uma seção **BeginWrite EndWrite**. Os métodos **BeginWrite** e **EndWrite** protegem uma parte do programa que acessa e modifica a memória. Várias threads podem entrar na seção BeginRead-EndRead ao mesmo tempo desde que nenhuma outra thread esteja dentro de uma seção BeginWrite-EndWrite.

Se uma thread A tentar executar BeginWrite e a thread B estiver dentro de uma seção BeginRead-EndRead, a thread A vai dormir, até que a thread B execute EndRead e só então a thread A prosseguirá. Se a thread B executar BeginRead e a thread A estiver dentro de uma seção BeginWrite-EndWrite, a thread B vai dormir até que a thread A execute EndWrite. Veja abaixo.



Conclusões

Desenvolver uma aplicação multi-thread no Delphi/Kylix é relativamente fácil, mas o difícil mesmo é garantir a consistência do programa, trabalhar com threads requer por parte do desenvolvedor muito mais atenção, para se ter uma idéia, em alguns casos atualizar um formulário pode se tornar mais difícil do que criar e manter uma lista generalizada, principalmente se o programador não for experiente.

Em muitas aplicações o uso de threads se faz necessário, para garantir a performance e muitas vezes até o próprio funcionamento da aplicação. Existem aplicações que só são concebíeis através do uso de multi-thread.

Bibliografia

Sistemas Operacionais Modernos, Andrew S. Tanenbaum1999.
O Help do Delphi.
O Help do Kylix.

Autor

Edmilson dos Santos de Jesus
Edsjbr@yahoo.com.br

Estudante do curso de Bacharelado em Análise de Sistemas 8º Sem. UNEB-BA
http://www.uneb.br

Integrante do OOLAB-UNEB (Laboratório de Orientação a Objetos da UNEB)
http://www.uneb.net/oolab

Analista de Sistemas da Actha Multiformática
http://www.actha.com.br