Inter thread message based communication

Alexandr Štefek

Military Academy in Brno

alexandr@stefek.cz

Abstract

This paper present implementation of parallel execution. If we use parallel proces in programs we have to solve synchronization. We show the effective implementation of parallel execution without using critical sections, semaphores, mutexes or events.

Thread

The thread is system object defined on platform Win32. SDK defines some method for thread creating and handling. In real implementation must be defined the procedure for all parallel procedures. If in system Win32 thread create new window handle then all messages are handled by this thread. For this fact commes the idea of problem solution.

We can simply create window handle on executing thread. When this thread executes loop for message handling it is possible to send special message to thread window handle. The parameters of message can be method to execute and parameter for this method. But method has 8 bytes and the message parameter only 4. So we have allocate memory block, copy method to this block and send adress of allocated block.

Coding the method

We want to create class that has method for sync parallel and async parallel execution of methods. Now we show, how to code the 8 byte method to 4 byte adress of method.

```
function TAxThread.NotifyEventToPointer(Proc: TNotifyEvent): Longint;
var
  Method: TMethod absolute Proc;
  PMethod: ^TMethod;
begin
  New(PMethod);
  PMethod^ := Method;
  Result := Longint(PMethod);
```

```
end;

procedure TAxThread.ExecProcedure(var Message: TMessage);
var
  PMethod : ^TMethod;
  Method : TMethod;
  Event: TNotifyEvent absolute Method;
begin
  PMethod := Pointer(Message.WParam);

  Method := PMethod^;
  Event(TObject(Message.LParam));

  Dispose(PMethod);
  if FThreadID = GetCurrentThreadId then
    InterlockedDecrement(FMethodsToExecute);
end;
```

`NotifyEventToPointer` is method for copying method to memory block. Result of this method is adress of memory block. Method `ExeProcedure` takes `WParam` of message, convert it back to method and call decoded method with parametr defined by `LParam` of message.

Thread loop

We have to define thread message loop. Delphi define basic thread class `TThread`. This class has virtual method execute. Defined class `TAxThread` is inherited from `TThread`. Method `TAxThread.Execute` is overrided;

```
procedure TAxThread.Execute;
var
  Msg: TMsg;
  Done: Boolean;
begin
  CreateHandleParallel;
  FThreadID := GetCurrentThreadId;

  while not Terminated do
  begin
    if Done then
    begin
      FIdleData := nil;
      WaitMessage;
    end;
    while ProcessMessage(Msg) do {loop};
    Idle(FIdleData, Done);
  end;
end;
```

At first the class has to create handle (`CreateHandleParallel`). Loop while waits for messages and handles incomming messages (`ProcessMessage`).

```
function TAxThread.ProcessMessage(var Msg: TMsg): Boolean;
begin
  Result := False;
  if PeekMessage(Msg, 0, 0, 0, PM_REMOVE) then
  begin
    Result := True;
    if Msg.Message <> WM_QUIT then
    begin
      TranslateMessage(Msg);
      DispatchMessage(Msg);
    end
    else
      Terminate;
  end;
end;
```

Calling `DispatchMessage` dispatch current message to objects for execution. If incomming method is `CM_EXECUTE` (defined in Delphi), then method `ExecProcedure` is called.

Sync and async execution

All parallel processing is sended to execution by message. If we use for sending the API function `SendMessage` then the execution is synchronized (actual thread is suspend, the context is switched, message is immediately handled and control is returned to sending thread). The API function `PostMessage` puts the message to message queue and continue in execution. When the message is peek from queue, is handled and method is executed.

```
//Asynchro execute on parallel thread
procedure TAxThread.AsyncExecuteParallel(Proc: TNotifyEvent; ParamSender:
   TObject);
begin
  InterlockedIncrement(FMethodsToExecute);
  PostMessageParallel(CM_EXECPROC, NotifyEventToPointer(Proc),
   Longint(ParamSender));
end;

//Synchro execute on parallel thread
procedure TAxThread.SyncExecuteParallel(Proc: TNotifyEvent; ParamSender:
   TObject);
begin
  InterlockedIncrement(FMethodsToExecute);
  SendMessageParallel(CM_EXECPROC, NotifyEventToPointer(Proc),
   Longint(ParamSender));
end;
```

Introduced method `AsyncExecuteParallel` is used for async parallel execution (execution on selected thread) of method `Proc` with parameter `ParamSender`. Method `SyncExecuteParallel` runs method `Proc` with parameter `ParamSender` synchronously (waits for execution). The thread defines method for execution on main thread (`AsyncExecuteMain`, `SyncExecuteMain`).

Using

For example of using define class

```
type
  TMainForm = class(TForm)
    btnRandomize: TButton;
    imgResult: TImage;
    pbProgress: TProgressBar;
    btnMulti: TButton;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure btnRandomizeClick(Sender: TObject);
    procedure btnMultiClick(Sender: TObject);
  private
    { Private declarations }
    FExecutingThread: TAxThread;
  public
    { Public declarations }
    procedure RandomizeBMP(Data: TObject);
    procedure DoUpdate(Data: TObject);
    procedure Progress(Data: TObject);
  end;
```

Private variable `FExecutingThread` is thread on witch the metods will be executed. There are three public method in form of `TNotifyEvent` (can be executed on selected thread).

```
procedure TMainForm.btnRandomizeClick(Sender: TObject);
var
  PomBMP : TBitmap;
begin
  if FExecutingThread = nil then
    FExecutingThread := TAxThread.Create;

  PomBMP := TBitmap.Create;
  PomBMP.Width := 200;
  PomBMP.Height := 200;
  PomBMP.PixelFormat := pf24bit;

  FExecutingThread.AsyncExecuteParallel(RandomizeBMP, PomBMP);
end;
```

When user clicked on button then method `btnRandomizeClick` is called. Method calls `AsyncExecuteParallel`. Main thread continue in responsing to user interaction. When the thread contrext is switched, `FExecutingThread` begins execute then method `RandomizeBMP`.

```
procedure TMainForm.RandomizeBMP(Data: TObject);
var
  CurrentThread : TAxThread;
  PomBMP : TBitmap;
  I : Longint;
  X : Longint;
  Y : Longint;
  Color : Longint;
begin
  if not(Data is TBitmap) then
    Exit;
  PomBMP := Data as TBitmap;

  CurrentThread := TAxThread.GetCurrentThread;
  I := 0;

  try
    PomBMP.Canvas.Lock;
    while not CurrentThread.Terminated do
    begin
      Inc(I);
      if I > MaxPoints then
        Break;

      X := Random(200);
      Y := Random(200);
      Color := Random(256) * 256 * 256 + Random(256) * 256 + Random(256);

      if (I mod (MaxPoints div 100)) = 0 then
        CurrentThread.SyncExecuteMain(Progress, TObject(I));

      //slow for demonstration
      PomBMP.Canvas.Pixels[X, Y] := Color;

    end;
  finally
    PomBMP.Canvas.Unlock;
    CurrentThread.AsyncExecuteMain(DoUpdate, PomBMP);
  end;
end;
```

Method RandomizeBMP decode parameter as Bitmap and fills it with some random points. When randomize is finished, the thread notify main thread (`CurrentThread.AsyncExecuteMain`).

Conclusion

We present implementation of thread class for async (sync) execution. The designed library has very effective method for developing of parallel computing.