Лабораторная работа № 6

Работа с потоками в Delphi

Разработка многопоточных программ в среде Delphi

Среда разработки Delphi представляет программисту полный доступ к функциям API Win32. Таким образом, для создания потока и управления им могут быть использованы рассмотренные ранее функции.

Однако использование API функций не единственный способ разработки многопоточных приложений в данной среде. В Delphi включен специальный класс, инкапсулирующий и упрощающий программный интерфейс работы с потоками. При использовании данного класса, от программиста не требуется знание API функций работы с потоками, меньше вероятность допустить ошибку при передаче параметров, упрощается механизм синхронизации потоков и разделения данных между ними. Для решения конкретной задачи программисту не требуется вдаваться в тонкости механизмов, предлагаемых операционной системой, т.к. базовая функциональность уже реализовано. Для создания потока необходимо лишь добавить в программу новый класс, наследующий от класса TThread, и перекрыть (включить свою реализацию) виртуальный метод Execute. Каждому новому потоку приложения будет соответствовать объект разработанного класса, управление этим потоком будет осуществляться через вызов свойств и методов объекта. Очевидно, данный способ значительно проще работы с API функциями. Также использование класса TThread - это гарантия безопасной работы с библиотекой визуальных компонентов VCL (). Без использования класса TThread во время вызовов VCL могут возникнуть конфликты обращения различнык потоков к одним и тем же элементам управления. Без использования TThread такая ситуация требует специальной синхронизации.

Класс TThread

Рассмотрим основные свойства и методы класса TThread, благодаря механизму наследования функциональность данного класса будет доступна в производном от этого типа классе. Свойства и методы TThread предоставляют основные функции по управлению потоком как объектом операционной системы.

метод Execute

procedure Execute; virtual; abstract;

В данный метод включается программный код, который должен исполняться потоком. Переопределяя метод Execute, программист закладывает в потоковый класс то, что будет выполняться при передаче управления потоку.

Можно выделить два варианта выполнения метода Execute.

Поток рассчитан на однократное выполнение каких-либо действий, в таком случае не требуется специального кода завершения внутри метода Execute. Выполнение потока прекратится после выполнения последнего оператора метода.

В потоке будет выполняться цикл, и поток должен завершиться вместе с процессом, которому он принадлежит. Ниже представлен примерный шаблон метода Execute для такого потока.

```
procedure TMyThread.Execute;
begin
repeat
DoSomething;
```

Until CancelCondition or Terminated;
end;

Логическое выражение CancelCondition - это условие завершения потока, которое устанавливает программист. Например, это могут быть исчерпание данных, окончание вычислений, наступление того или иного события. Свойство Terminated класса TThread сообщает о преждевременном завершении потока. Данное свойство может быть установлено как программистом (например, в процессе выполнения программы произошла ошибка), так и системой (завершается родительский процесс), как изнутри класса потока, так и извне.

Конструктор класса TThread

constructor Create(CreateSuspended: Boolean);

Параметр CreateSuspended влияет на состояние потока после создания его объекта. Если значение данного параметра равно True, вновь созданный поток не начинает выполняться до тех пор, пока не будет сделан вызов метода Resume. В случае, если параметр CreateSuspended имеет значение False, конструктор завершается, затем поток начинает исполнение. Следует обратить внимание на то, что поток не начинает выполняться мгновенно после завершения выполнения конструктора или вызова метода Resume. Данные вызовы следует отличать от простого вызова процедур, которые начинают свое выполнение непосредственно после вызова. После описанных действий поток лишь переходит с состояние готовности к выполнению, а когда поток непосредственно начнет выполнение, решит операционная система, программист прямо не может повлиять на время начала его выполнения.

Метод Resume – возобновление выполнения потока

procedure Resume;

Вызов метода Resume класса TThread возобновляет выполнение после остановки. Также он используется для явного запуска потока, созданного с параметром CreateSuspended, равным True.

Метод Suspend – приостанока выполнения потока

procedure Suspend;

Вызов метода Suspend приостанавливает поток с возможностью повторного запуска впоследствии. Метод suspend приостанавливает поток вне зависимости от кода, исполняемого потоком в данный момент, выполнение продолжается с точки останова.

Свойство Suspended

property Suspended: Boolean;

Свойство suspended позволяет определить приостановлен поток в данный момент или нет. С помощью этого свойства можно также запускать и останавливать поток. Установка свойство suspended в значение True аналогична вызову метода Suspend. Установка этого свойства в значение False возобновляет выполнение потока, то есть аналогична вызову метода Resume.

Деструктор

destructor Destroy; override;

Деструктор Destroy вызывается, когда необходимость в созданном потоке отпадает. Деструктор завершает его и освобождает все ресурсы, связанные с объектом.

Свойство Terminated

property Terminated: Boolean;

Данное свойство используется для завершения потока "в мягкой форме", с возможностью корректно освободить ресурсы. Для завершения потока необходимо установить значение данного свойства равным True.

Также следует обратить внимание, что именно значения этого свойства проверяется в цикле метода Execute.

Метод Terminate

function Terminate: Integer;

Метод Terminate используется для окончательного завершения потока. Но этот метод не делает никаких принудительных действий по остановке потока. Происходит только установка свойства Terminated в значение True.

Метод Terminate автоматически вызывается из деструктора объекта. При завершении работы поток - объект класса TThread будет дожидаться, пока завершится поток- объект операционной системы. Таким образом, если поток не умеет завершаться корректно, вызов деструктора потенциально может привести к тупику.

При необходимости немедленно завершить поток необходимо использовать API функцию TerminateThread (см).

Свойство FreeOnTerminate

property FreeOnTerminate: Boolean;

Если значение данного свойства равно True, то деструктор потока будет вызван автоматически по его завершении. Такой подход удобен в тех случаев, когда неизвестно даже примерное время окончания потока. Поток будет запущен, а при завершении сам освободит занятые ресурсы (принцип «выстрелил и забыл» - «fire and forget»).

Метод WaitFor

function WaitFor: Integer;

Метод WaitFor предназначен обеспечения простой и удобной синхронизации потоков, он позволяет одному потоку дождаться момента завершения другого потока. Например, если к коде потока FirstThread написана строка

Code := SecondThread.WaitFor;

то поток FirstThread остановится до момента завершения потока SecondThread. Метод WaitFor возвращает код завершения ожидаемого потока (см. свойство Returnvalue).

Свойства Handle и ThreadID

property Handle: THandle read THandle;

property ThreadID: THandle read TThreadID;

Свойства Handle и ThreadID возвращают дескриптор и идентификатор потока – объекта операционной системы, который создается для каждого объекта класса TThread. Эти свойства дают программисту возможность непосредственной работы с потоком средствами API Win32, разработчик может обратиться к потоку и управлять им, минуя возможности класса TThread. Значения данных свойств непосредственно используются в качестве аргументов функций, управляющих потоком. Значения этих свойств приходятся при досрочном завершении потока (), синхронизации потоков с помощью функций ожидания (ожидание завершения одного - или нескольких потоков) и т.д.

Приоритет потока – свойство Priority

property Priority: TThreadPriority;

Свойство Priority позволяет запросить и установить приоритет потока. Допустимыми значениями свойства Priority объектов класса TThread являются элементы перечислимого типа TThreadPriority: tpidle, tpLowest, tpLower, tpNormal, tpHigher, tpHighest и tpTimeCritical.

Метод Synchronize

procedure Synchronize(Method: TThreadMethod);

Метод используется для безопасного обращения к объектам VCL (формы и элементы управления) из потоков, отличных от потока формы. Для каждой формы в Delphi свой поток, управляющий ее поведением, и только из этого потока можно безопасно обращаться к форме и ее элементам управления, не используя средства синхронизации. Для обращения к форме из других потоков использование метода Synchronize обязательно. Вызов метода synchronize дает гарантию, что к каждому объекту одновременно имеет доступ только один поток, и таким образом исключается ситуация конфликта. Сложность проблемы в том, что Delphi не контролирует выполнение этого правила и не препятствует обращению в форме из другого потока, формального противоречия в данном случае нет, язык Object Pascal позволяет производить такое обращение. И в некоторых случая программа даже будет работать, но до момента наступления конфликта обращения различных потоков к форме. В принципе, такой конфликт может не происходить продолжительное время, в течение многократных запусков программы. Но затем обязательно произойдет и притом в самый ответственный момент. Отладка многопоточных программ одна из основных сложностей разработки таких программ. Ошибки взаимодействия потоков трудно «отловить», т.к. условия, приведшие к ошибке трудно возобновимы.

Метод Synchronize относится к секции protected, т. е. может быть вызван только из потомков TThread. Аргумент, передаваемый в метод Synchronize, - это имя метода, который производит обращение к VCL. Параметр метод (класса TThreadMethod) не должен иметь никаких параметров и не должен возвращать никаких значений. Обычно все операции по вводу-выводу данных потоком группируют в один или несколько таких методов, которые вызывают с помощью метода Synchronize. А проблема отсутствия параметров у метода, передаваемого в качестве аргумента решается за счет переменных уровня класса.

```
procedure TMainForm.SyncShowMessage; begin
ShowMessagedntToStr (ThreadListl. Count) ) ; // другие обращения к VCL
end;
а в потоке для показа сообщения писать не
ShowMessage(IntToStr(ThreadListl.Count));
и даже не
MainForm.SyncShowMessage;
а только так:
Synchronize(MainForm.SyncShowMessage);
```

Итак, можно сформулировать основное правило обращения к формам и их элементам управления из потоков. Производя любое обращение к объекту VCL из потока, убедитесь, что при этом используется метод Synchronize; в противном случае результаты могут оказаться непредсказуемыми.

Свойство ReturnValue

property ReturnValue: Integer;

Свойство ReturnValue позволяет узнать и установить значение, возвращаемое потоком по его завершении. Эта величина полностью определяется пользователем. По умолчанию поток возвращает ноль, но если программист может установить любую другую величину, присвоив свойству ReturnValue внутри потока любое другое значение. Другие потоки затем могут проанализировать это значение. На пример, поток может сообщить о произошедших внутри него ошибках.

Локальные данные потока

Как уже отмечалось ранее одно из основных удобств использования класса TThread, заключается в том, что каждый поток соответствует отдельному экземпляру объекта, и их данные не пересекаться, т.е. каждый поток работает со своими экземплярами переменных класса. Однако, если программист использует для работы с потоками API функции Windows то все потоки будут использовать одни и те же переменные. Для поддержки разделения данных между потоками на уровне функции работы с поттоками в язык Object Pascal введена специальная директива - threadvar, которая отличается от директивы описания переменных var тем, что применяется только к локальным данным потока.

```
Var
datal: Integer; threadvar
data2: Integer;
```

В приведенном примере переменная datal будет использоваться всеми потоками данного приложения, а переменная data2 будет у каждого потока своя.

Подводя итог описания работы с потоками в Delphi, отметим, что в данной среде программирования реализована очень удачная концепция работы с потоками с помощью класса TThread, значительно упрощающая разработку многопоточных приложений. Также у программиста остается возможность работать с потоками с помощью встроенных функций операционной системы, а при необходимости совмещать эти два подхода. Однако не следует совмещать и путать эти два понятия. С точки зрения операционной системы поток - это ее объект, при своем создании он получает дескриптор и отслеживается ОС. Объект класса TThread - это конструкция Delphi, соответствующая потоку ОС. Этот объект VCL создается до реального возникновения потока в системе и уничтожается после его исчезновения.

Упражнение

1. Запустите среду разработки Delphi, откройте новый проект через главное меню **File|New|Application** либо воспользуйтесь проектом, созданным по умолчанию.
2. Расположите на форме две кнопки с надписями «Start» и «Stop», для задания надписей используйте свойство *Caption*, для вызова страницы свойств (*Object Inspector*) используйте клавишу F11 (рис. 1).

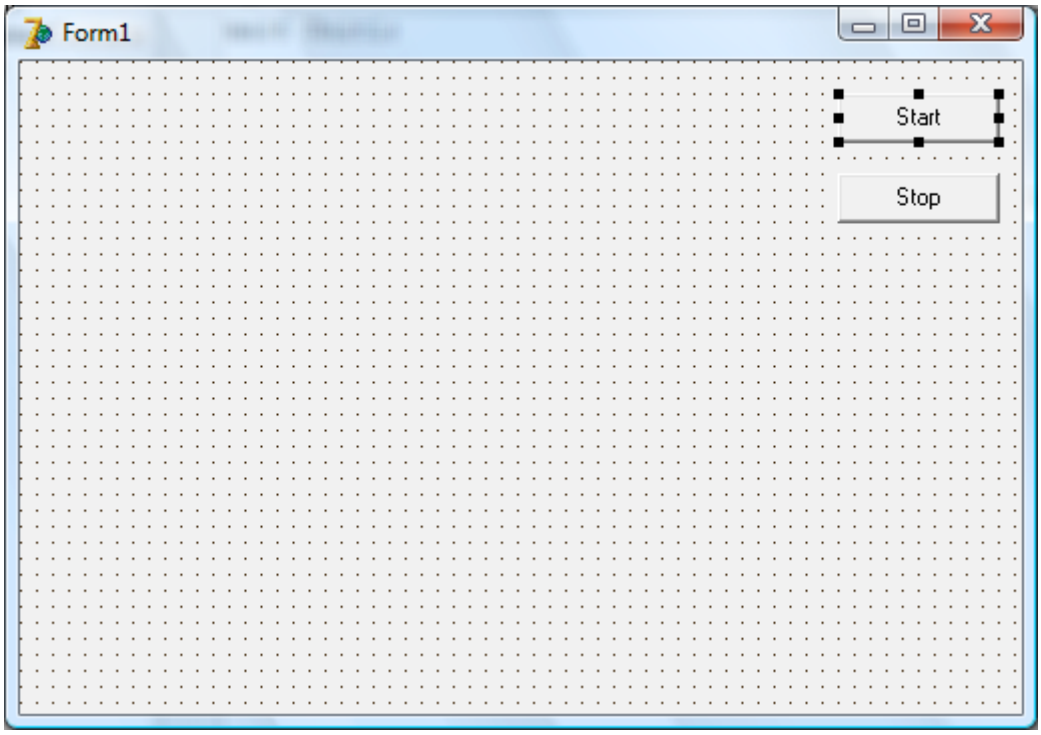


Рис. 1

3. Создайте для формы обработчик события *MouseDown* (нажатие кнопки мыши), рисующий на форме круг в текущем положении курсора мыши. Для этого в окне Object Inspector, предварительно выделив форму, выберете закладку Events, на которой найдите событие *OnMouseDown*. Создайте обработчик события двойным щелчком мыши по полю.

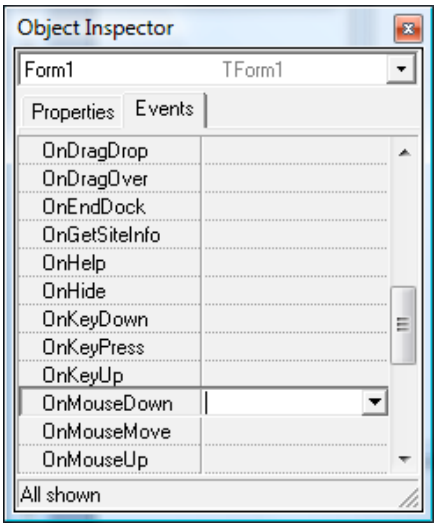


Рис. 2

4. В обработчике события введите следующий программный код.

```
    procedure TForm1.FormMouseDown(Sender: TObject; Button:
TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    begin
      Canvas.Pen.Color:=clYellow;
      Canvas.Brush.Color:=clYellow;
      Canvas.Ellipse (x - 30, y - 30, x + 30, y + 30);
    end;
```

5. Запустите приложение на выполнение, нажав клавишу F9.
6. Добавьте константу с именем Radius и замените ей непосредственное значение радиуса 30.

7. Добавьте программный поток, выполняющий фоновую закраску формы. Для этого выполните команду **File|New|Other…**, на экране появится окно с шаблонами, в котором следует выбрать элемент *Thread Object*.

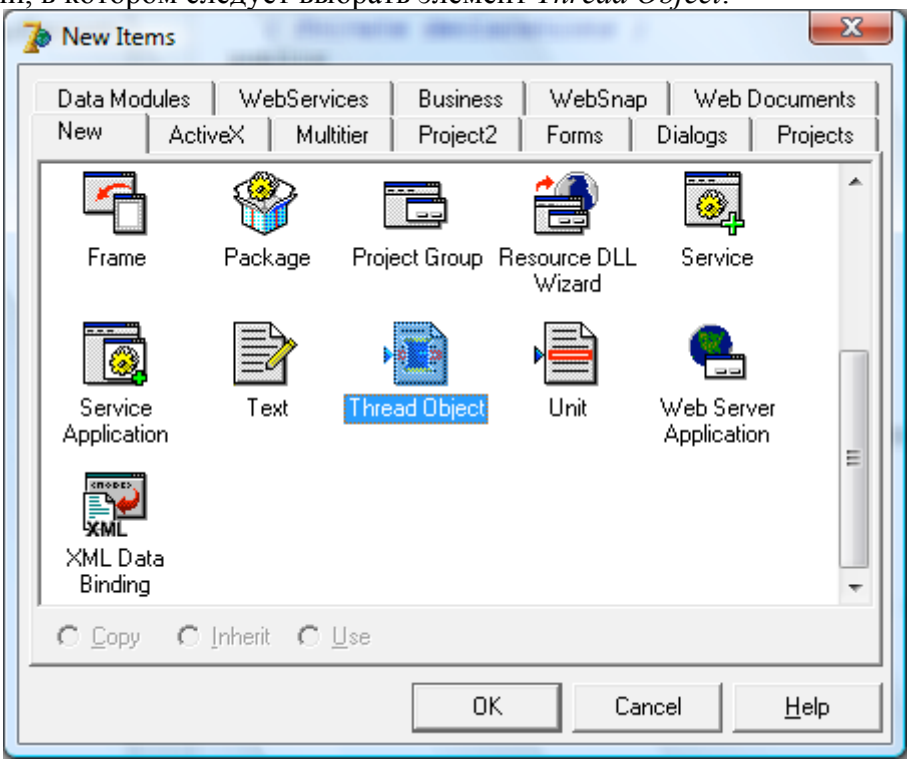


Рис. 3

8. Задайте имя создаваемого класса – *TPaintThread*.

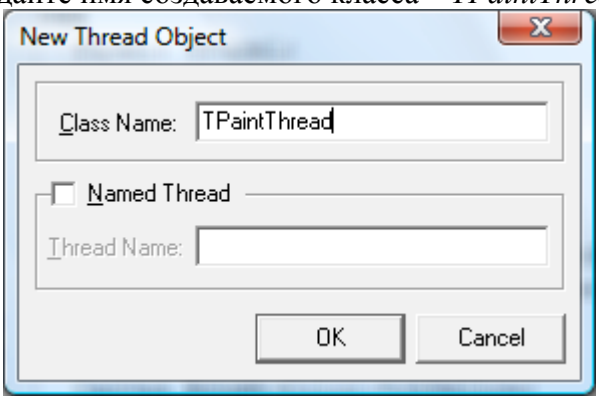


Рис. 3

9. В созданном модуле (Unit2) добавьте ссылку на модуль с главной формой (Unit1) и на модуль Graphics. Для этого добавьте ключевое слово *uses* в разделе implementation.

```
…
implementation
uses Unit1,Graphics;
…
```

10. В модуле формы (Unit1) добавьте ссылку на модуль с потоком (Unit2), дополнив список уже используемых модулей в разделе uses.
11. Добавьте непосредственно исполняемый код потока в метод Execute класса TPaintThread.

```
procedure TPaintThread.Execute;
var
  X, Y: Integer;
begin
```

```
  Randomize;
  repeat
    X := Random (Form1.Button1.Left-10);
    Y := Random (Form1.ClientHeight);
    with Form1.Canvas do
    begin
      Lock;
      Pixels [X, Y] := clBlue;
      UnLock;
    end;
  until Terminated;
end;
```

В данном случае объект *Canvas* (полотно для рисования на форме) будет являтся разделяемым ресурсом, т.к. к нему одновременно будут обращаться несколько потоков. Объект *Canvas* формы имеет встроенные средства синхронизации для предотвращения конфликтов при доступе – *Lock* и *UnLock*. В общем случае при обращении к форме из разных потоков необходимо использовать метод *Synchronize*.

12. Добавьте вызов методов *Lock* и *UnLock* при обращении к объекту *Canvas* в главной форме.
13. Объявите переменную для работы с потоком в модуле формы.

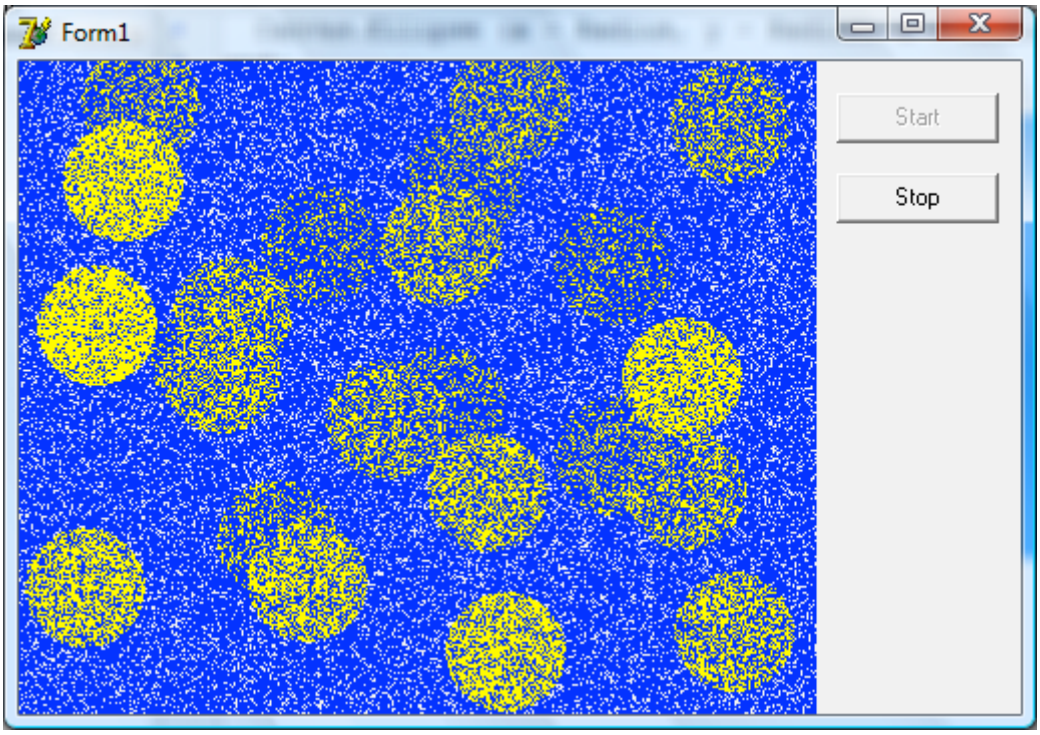
```
…
var
  Form1: TForm1;
  PT: TPaintThread;
…
```

14. Добавьте обработчики события Click кнопок Start и Stop.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Button1.Enabled:=False;
  Button2.Enabled:=True;
  PT := TPaintThread.Create(False);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  PT.Terminate;
  PT.Free;
  Button1.Enabled:=True;
  Button2.Enabled:=False;
end;
```

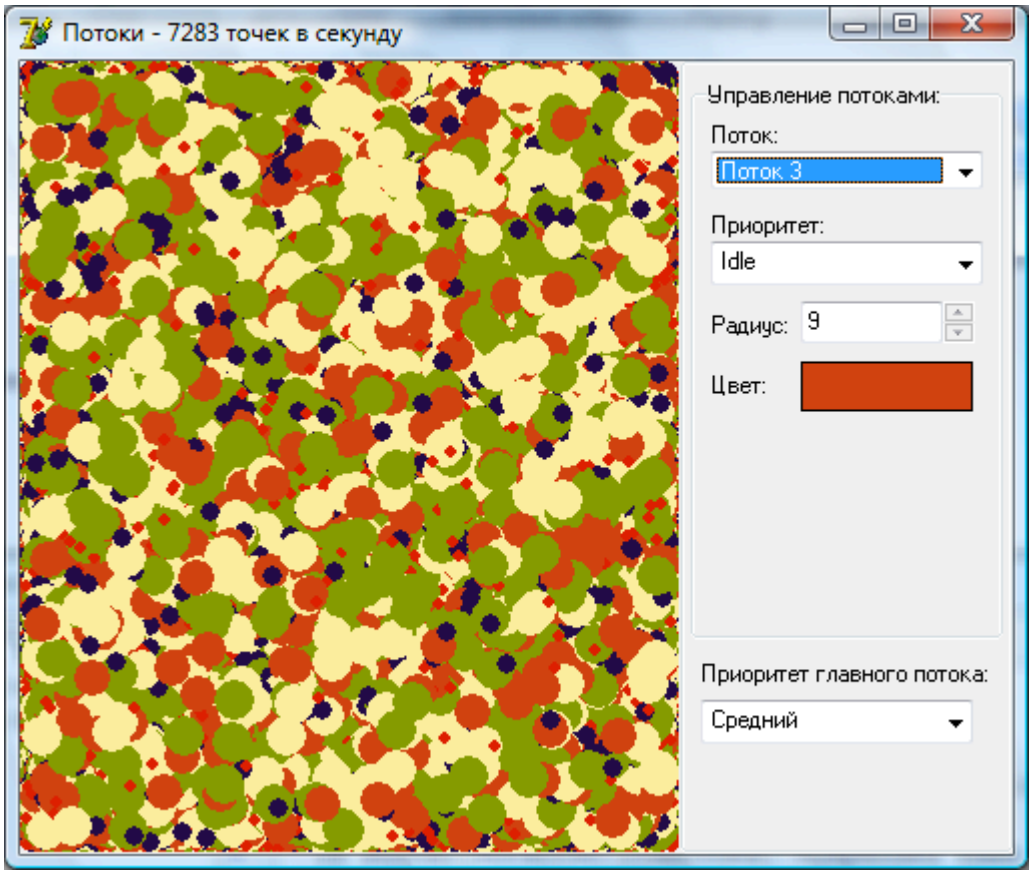
15. Запустите приложение на выполнение, нажав клавишу F9.



16. При завершении основного потока должны завершиться и все дочерние потоки, для этого в событии FormDestroy нужно завершить и уничтожить поток аналогично тому, как это делается при нажатии на кнопку Stop.
17. Перепишите код обращения к объекту *Canvas* с использованием метода *Synchronize* вместо вызовов методов блокировки.

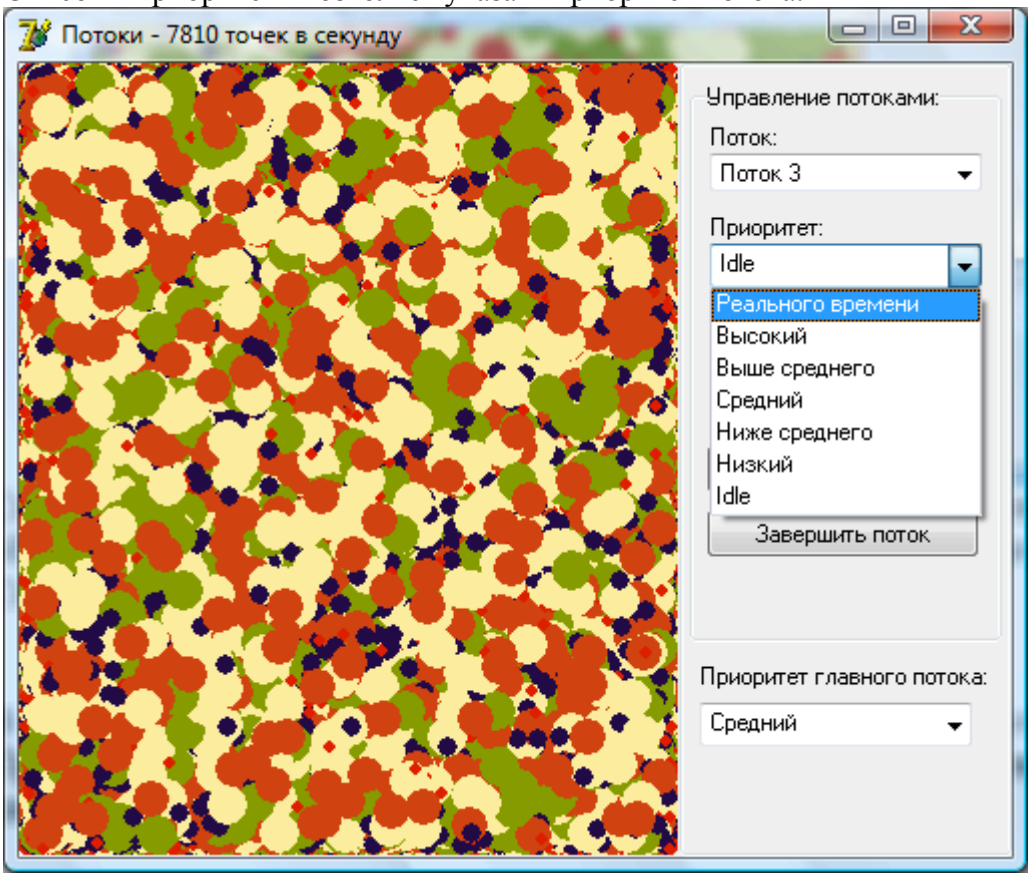
Задание

Разработать многопоточное приложение, позволяющие динамически управлять потоками, выводящими на экран круги произвольного радиуса и цвета. Примерная экранная форма представлена на рис.



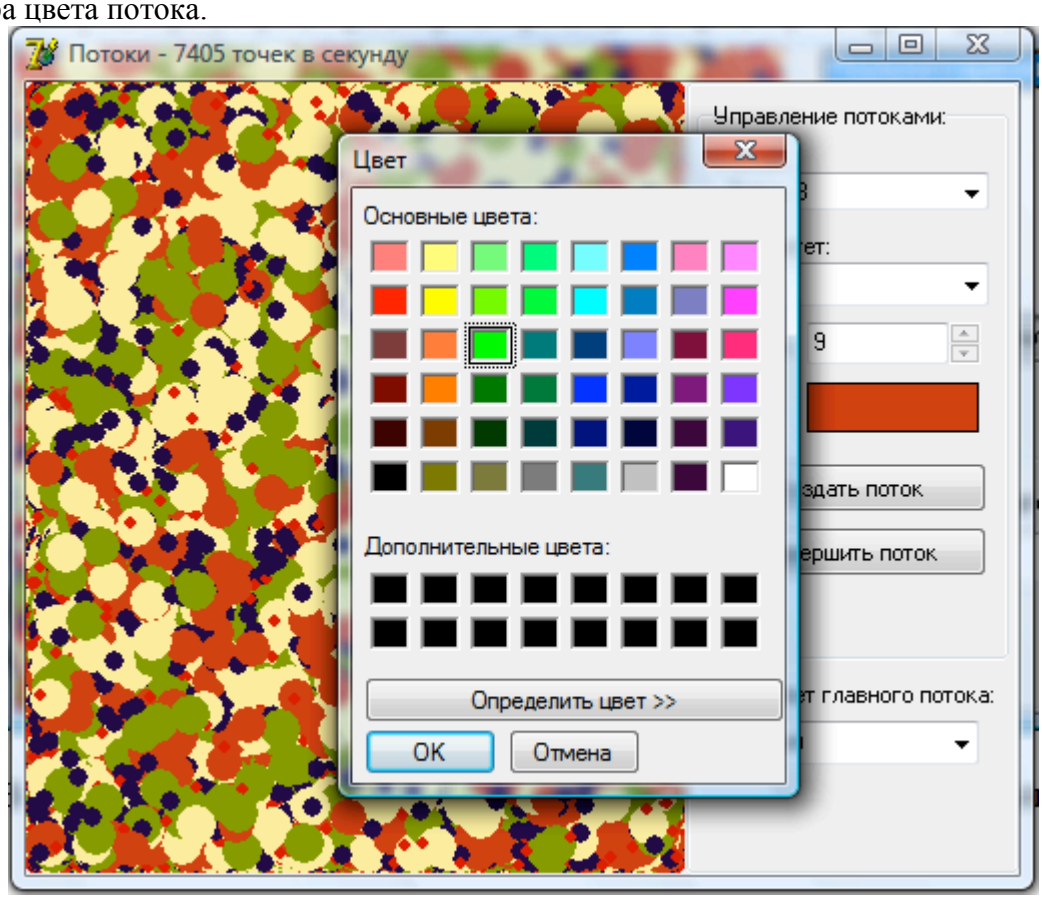
Список «Поток» содержит созданные в программе потоки. Изменение параметров «Приоритет», «Радиус» и «Цвет» применяются к выбранному потоку.

Список «Приоритет» позволяет указать приоритет потока.



В поле «Радиус» вводится радиус кругов, рисуемых текущим потоком.

При щелчке на поле с цветом выводится стандартное диалоговое окно для выбора цвета потока.



При нажатии на кнопку «Создать поток» создается новый поток с параметрами по умолчанию, имя потока добавляется в список потоков.

Нажатие кнопки «Завершить поток» уничтожает выбранный в списке поток и удаляет его из этого списка.

Список «Приоритет главного потока» позволяет задать приоритет потока формы.

Рекомендации

Ссылки на созданные потоки удобно хранить в динамическом массиве.

Для изменения приоритета главного потока можно использовать приведенные ниже API-функции.

```
MainTh := GetCurrentThread;
SetThreadPriority(MainTh, THREAD_PRIORITY_NORMAL);
```