Этот раздел содержит описание шагов, необходимых для создания простого, но показательного примера многопоточного приложения.

Мы будем пытаться вычислить число "пи" с максимальной точностью после запятой. Конечно, встроенная в Delphi константа Pi имеет достаточную точность, правильнее сказать — максимальную, допускаемую самым точным 10-байтным форматом для вещественных чисел **Extended**. Так что превзойти ее нам не удастся. Но этот пример использования потоков может послужить прологом для решения реальных задач.

Первый пример будет содержать два потока: главный (обрабатывающий ввод пользователя) и вычислительный; мы сможем изменять их свойства и наблюдать за реакцией.

Итак, выполните следующую последовательность действий:

1. В среде Delphi откройте меню *File* и выберите пункт *New Application*.

2. Расположите на форме пять меток и один переключатель, как показано на рис. 1. Переименуйте главную форму в *fmMain*.

3. Откройте меню *File* и выберите пункт *Save Project As*. Сохраните модуль как uMain, а проект — как Threads 1.

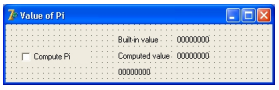

Рис. 1 Внешний вид формы для приложения Threads1

4. Откройте меню *File* и выберите пункт *New*. Затем дважды щелкните на объекте типа

поток (значок **Thread Object**). Откроется диалоговое окно
New Items
, показанное на рис. 2.

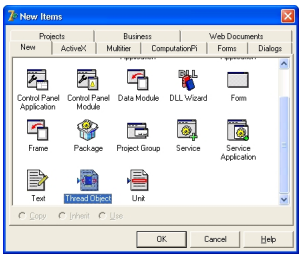

Рис. 2 Диалоговое окно New Items с выбранным объектом типа "поток"

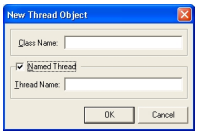

Рис. 3 Диалоговое окно New Thread Object

5. Когда появится диалоговое окно для именования объекта поток, введите **TPiThread** и нажмите клавишу Enter (рис. 3). Помимо этого, при желании, вы можете присвоить создаваемому потоку имя, установив флажок
Named Thread
и задав имя в поле
Thread Name
. Так как имя потока используется только для удобства обозначения, эту возможность мы использовать не будем.

Delphi создаст новый модуль и поместит в него шаблон для нового потока.

6. Код, вносимый в метод **Execute**, вычисляет число, используя сходимость бесконечного ряда Лейбница:

Pi = 4 - 4/3 + 4/5 - 4/7 + 4/9 -...

Разумеется, отображать новое значение после каждой итерации — это то же самое, что

стрелять из пушки по воробьям. На отображение информации система потратит в десятки раз больше времени, чем на собственно вычисления. Поэтому мы ввели константу **updatePeriod**, которая регулирует периодичность отображения текущего значения.

Код метода **Execute** показан ниже:

```
const // Лучше использовать нечетное число для того, чтобы избежать эффекта // мерцания UpdatePeriod = 1000001; procedure TPiThread.Execute; var sign : Integer; PiValue, PrevValue : Extended; i : Int64; begin { Place thread code here } PiValue := 4; sign := -1; i := 0; repeat Inc(i); PrevValue := PiValue; PiValue := PiValue + sign * 4 / (2*i+l); sign := -sign; if i mod UpdatePeriod = 0 then begin GlobalPi := PiValue; GlobalCounter := i; Synchronize(fmMain.UpdatePi); end; until Terminated or (Abs(PiValue - PrevValue) 7. Откройте меню *File* и выберите пункт *Save As*. Сохраните модуль с потоком как uPiThread.pas.
```

8. Отредактируйте главный файл модуля uMain.pas и добавьте модуль **uPiThread** к списку используемых модулей в секции интерфейса. Он должен выглядеть так:

```
uses Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, StdCtrls, uPiThread;
```

9. В секции public формы **TfmMain** добавьте ссылку на создаваемый поток:

```
PiThread : TPiThread;
```

10. Добавьте в модуль **uMain** две глобальные переменные:

```
GlobalPi : Extended; GlobalCounter : Int64;
```

и метод **UpdatePi**:

```
procedure TfmMain.UpdatePi; begin if IsIconic(Application.Handle) then Exit; LaValue.Caption := FloatToStrF(GlobalPi, ffFixed, 18, 18); lalterNum.Caption := IntToStr(GlobalCounter) + ' iterations'; end;
```

Этот метод, если вы обратили внимание, вызывается из потока посредством процедуры Synchronize. Он отображает текущее значение приближения к числу "пи" и количество итераций.

В случае, если главное окно приложения свернуто, отображение не производится; так что после его развертывания вам, возможно, придется подождать некоторое время для обновления.

11. Выполните двойной щелчок на свободном месте рабочей области формы, при этом создастся шаблон метода **FormCreate**. Здесь мы отобразим значение системной константы p±:

```
procedure TfmMain.FormCreate(Sender: TObject);  begin    laBuiltIn.Caption :=
FloatToStrF(Pi, ffFixed, 18, 18);  end;
```

12. Выберите на форме переключатель (его имя *cbcalcuiate*) и назначьте событию **Onclick** код, создающий и уничтожающий вычислительный поток в зависимости от состояния переключателя:

```
procedure TfmMain.cbCalculateClick(Sender: TObject);  begin  if cbCalculate.Checked
then  begin  PiThread := TPiThread.Create(True);  PiThread.FreeOnTerminate := True;
  PiThread.Priority := tpLower;  PiThread.Resume;  end  else  begin  if
Assigned(PiThread) then  PiThread.Terminate;  end;  end;
```

Таким образом, многопоточное приложение готово к запуску. Если все пройдет нормально, вы увидите картинку, подобную той, которая приведена на рис. 4.

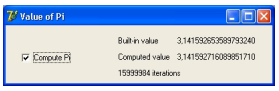

Рис. 4 Выполняющееся приложение Threads1

Пока один из авторов писал текст этого раздела, запущенное одновременно приложение **Threadsl** выполнило пять миллиардов итераций и приблизилось к встроенному значению **Pi** в десятом разряде. Интересно, насколько хватит терпения у вас?

Этот простой пример — первый шаг в усвоении того, как от базового класса rrhread можно порождать собственные классы. Из-за своей простоты он не лишен недостатков; более того — если бы вычислительных нитей было не одна, а более, кое-какие приемы были бы даже ошибочными. Но — об этом ниже.