## 9. Режимы работы нитей. Процессы в Windows

### 9.1 Понятие основной и фоновой нити

По умолчанию нити создаются как основные, что означает, что приложение не будет завершено, пока одна из таких нитей будет исполняться. В языке C# существует понятие фоновых нитей, которые завершаются сразу же, как только все основные нити будут завершены (считается, что они не «продлевают» жизнь приложению).

Статус нити – основная или фоновая определяется состоянием свойства IsBackground, которое имеет логический тип.

В качестве учебной программы рассмотрим работу двух нитей с разным временем задержки:

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
 class Program
 {
  public static void Niti_1()
  {
   Console.WriteLine("Работает нить 1 :");
   for (int i = 0; i < 10; i++)
   {
    Console.Write(i + " ");
    Thread.Sleep(1000);
   }
   Console.WriteLine();
  }
  public static void Niti_2()
  {
   Console.WriteLine("Работает нить 2 :");
   for (int i = 10; i < 20; i++)
   {
    Console.Write(i + " ");
    Thread.Sleep(1000*2);
   }
   Console.WriteLine();
  }
  static void Main(string[] args)
  {
   Thread nt1 = new Thread(Niti_1);
   Thread nt2 = new Thread(Niti_2);
   nt1.IsBackground = false;
   nt2.IsBackground = true;
   //nt1.IsBackground = true;
   //nt2.IsBackground = false;
   nt1.Start();
   nt2.Start();
   Console.ReadLine();
```

```
    }
  }
}
```
Рабрта программ:
Работает нить 1 :
0 Работает нить 2 :
10 1 11 2 3 12 4 5 13 6 7 14 8 9 15
16 17 18 19

Работает нить 1 :
0 Работает нить 2 :
10 1 11 2 3 12 4 5 13 6 7 14 8 9

В первом случае, когда `nt1.IsBackground = true;` - первая нить фоновая, программа заканчивает свою работу по окончании работы второй нити. Во втором случае (фоновой является вторая нить) программа заканчивает свою работу до завершения работ второй нити. «Когда фоновый поток завершается таким способом, все блоки **finally** внутри потока игнорируются. Поскольку невыполнение кода в **finally** обычно нежелательно, будет правильно ожидать завершения всех фоновых потоков перед выходом из программы, назначив нужный таймаут (при помощи **Thread.Join**). Если по каким-то причинам рабочий поток не завершается за выделенное время, можно попытаться аварийно завершить его (**Thread.Abort**).

Превращение рабочего потока в фоновый может быть последним шансом завершить приложение, так как не умирающий основной поток не даст приложению завершиться. Зависший основной поток особенно коварен в приложениях Windows Forms, так как приложение завершается, когда завершается его главный поток (по крайней мере, для пользователя), но его процесс продолжает выполняться. В диспетчере задач оно исчезнет из списка приложений, хотя имя его исполняемого файла останется в списке исполняющихся процессов. Пока пользователь не найдет и не прибьет его, процесс продолжит потреблять ресурсы и, возможно, будет препятствовать запуску или нормальному функционированию вновь запущенного экземпляра приложения.»

## 9.2 Понятие приоритета нити

При определении характеристик нити можно задавать приоритет нити с помощью свойства Priority. В языке C# существует 5 градаций приоритета, которые можно задавать нити: Lowest, BelowNormal, Normal, AboveNormal, Highest. Приоритет нити определяет, сколько времени на выполнение выделяется нити относительно других нитей. Для исследований приоритетов нитей используем предыдущую программу с двумя основными нитями, но с разными приоритетами.
Исходный код программы:

```
using System;
using System.Diagnostics;
using System.Threading;

namespace ConsoleApplication1
{
 class Program
 {
  public static void Niti_1()
  {
   Console.WriteLine("Работает нить 1 :");
   for (int i = 0; i < 10; i++)
   {
    Console.Write(i + " ");
   }
   Console.WriteLine();
  }
  public static void Niti_2()
  {
   Console.WriteLine("Работает нить 2 :");
   for (int i = 10; i < 20; i++)
   {
    Console.Write(i + " ");
   }
   Console.WriteLine();
  }
  static void Main(string[] args)
  {
   Thread nt1 = new Thread(Niti_1);
   Thread nt2 = new Thread(Niti_2);
   nt1.Priority = ThreadPriority.BelowNormal;
   nt2.Priority = ThreadPriority.BelowNormal;
   nt1.Start();
   nt2.Start();
   Console.ReadLine();
  }
 }
}
```

Работа программы при различных значениях приоритетов нитей:

A)

```
nt1.Priority = ThreadPriority.AboveNormal;
nt2.Priority = ThreadPriority.Normal;
```

Работает нить 1 :
Работает нить 2 :
10 11 12 13 14 15 16 17 18 19
0 1 2 3 4 5 6 7 8 9

B)

```
nt1.Priority = ThreadPriority.Highest;
nt2.Priority = ThreadPriority.Lowest;
```

Работает нить 1 :

0 1 2 3 4 5 6 7 8 9
Работает нить 2 :
10 11 12 13 14 15 16 17 18 19
C)

```
nt1.Priority = ThreadPriority.BelowNormal;
nt2.Priority = ThreadPriority.BelowNormal;
```

Работает нить 2 :
10 11 12 13 Работает нить 1 :
0 1 2 3 4 5 14 15 16 17 18 19
6 7 8 9

Соотношение приоритета и времени выполнения нити?

## 9.3 Определение процесса

Процессом в Windows называется объект ядра, которому принадлежат системные ресурсы, используемые в исполняемом приложении. Обычно говорят, что процесс это исполняемое приложение.

Выполнение каждого процесса начинается с выполнения главной нити – для консольных приложений это метод Main.

Во время выполнения процесса могут создаваться и другие нити процесса или даже другие процессы.

Процесс заканчивается при завершении работы всех его нитей.

Каждый процесс в Windows владеет следующими системными ресурсами:

- виртуальным адресным пространством;
- маркером доступа с информацией системы безопасности;
- таблицей для хранения дескрипторов объектов ядра.

По аналогии с нитью каждый процесс имеет свой идентификатор, который используется служебными программами.

## 9.4 Создание процесса из работающего приложения

Для запуска приложения (создание процесса) из уже рабочего приложения необходимо использовать класса Process.

Класс Process имеет набор свойств и методов, которые представлены в приложении этой лекции. Перечень получен с помощью справки среды при выделении слова `Process` и нажатии кнопки F1.

Для успешного запуска процесса к приложению необходимо подключить пространство имен System.Diagnostics.

Класс Process можно использоваться для запуска процессов путем вызова метода Process.Start(FileName) или Process.Start().

Перед запуском метода Process.Start(FileName) необходимо в качестве параметра метода задать имя файла процесса (свойства FileName) для запуска или полный путь к нему.

Можно создать объект нового процесса, задать значение свойству FileName и запустить процесс с помощью метода Process.Start(), без параметров.

Предложенные варианты запуска реализованы в следующей учебной программе:

```
using System;
using System.Diagnostics;
namespace ConsoleApplication1
{
 class soz_pro
 {
  static void Main()
  {
   Process.Start("sol.exe");
   Console.ReadLine();
   Process myProcess = new Process();
   myProcess.StartInfo.FileName = "Notepad";
   myProcess.Start();
   Console.ReadLine();
  }
 }
}
```

Работа программы заключается в запуске игры «Пасьянс Косынка». По окончании игры необходимо нажать клавишу «Enter» и запустить редактор «Блокнот». После окончания работы с блокнотом необходимо закрыть консольное окно нажатием клавиши «Enter».

Можно использовать методы класса Process, например, GetProcesses для исследования характеристик работающего процесса или процессов.

В следующей учебной программе просматриваются некоторые характеристики первого процесса на компьютере работающего на текущий момент:

```
using System;
using System.Diagnostics;
using System.ComponentModel;

namespace ConsoleApplication1
{
 class soz_pro
 {
  static void Main()
  {
   foreach (Process p in Process.GetProcesses())
   {
    Console.WriteLine("Name:         " + p.ProcessName);
    Console.WriteLine("PID:          " + p.Id);
    Console.WriteLine("Started:      " + p.StartTime);
    Console.WriteLine("Memory:       " + p.WorkingSet64);
```

```
            Console.WriteLine("CPU time:     " + p.TotalProcessorTime);
            Console.WriteLine("Threads:      " + p.Threads.Count);
            Console.WriteLine(); break;
        }
        Console.ReadLine();
    }
  }
}
```

Работа программы:

Name: dwm
PID: 1292
Started: 31.07.2011 7:07:47
Memory: 31358976
CPU time: 00:00:03.2604209
Threads: 6

Домашнее задание: Исследовать все возможные свойства процесса.

Приложение лекции
Свойства и методы класса Process
Свойства

| | Имя | Описание |
|---|---|---|
| | BasePriority | Получает базовый приоритет связанного процесса. |
| | CanRaiseEvents | Возвращает значение, показывающее, может ли компонент вызывать событие. (Унаследовано от Component.) |
| | Container | Возвращает контейнер IContainer, содержащий компонент Component. (Унаследовано от Component.) |
| | DesignMode | Возвращает значение, указывающее, находится ли данный компонент Component в режиме конструктора в настоящее время. (Унаследовано от Component.) |
| | EnableRaisingEvents | Получает или задает наличие вызова события Exited при прекращении процесса. |
| | Events | Возвращает список обработчиков событий, которые прикреплены к этому объекту Component. (Унаследовано от Component.) |
| | ExitCode | Получает значение, заданное связанным процессом при завершении. |
| | ExitTime | Получает время завершения связанного процесса. |
| | Handle | Получает присущий данному объекту дескриптор связанного процесса. |

| | | |
|---|---|---|
| | HandleCount | Получает число дескрипторов, открытых процессом. |
| | HasExited | Получает значение, определяющее завершение связанного процесса. |
| | Id | Получает уникальный идентификатор связанного процесса. |
| | MachineName | Получает имя компьютера, на котором выполняется связанный процесс. |
| | MainModule | Получает главный модуль связанного процесса. |
| | MainWindowHandle | Получает дескриптор главного окна связанного процесса. |
| | MainWindowTitle | Получает заголовок главного окна процесса. |
| | MaxWorkingSet | Получает или задает максимальный размер доступного рабочего множества для связанного процесса. |
| | MinWorkingSet | Получает или задает минимальный размер доступного рабочего множества для связанного процесса. |
| | Modules | Получает модули, которые были загружены связанным процессом. |
| | NonpagedSystemMemorySize | **Устаревшее.** Получает размер невыгружаемой системной памяти, выделенной этому процессу. |
| | NonpagedSystemMemorySize64 | Получает количество невыгружаемой системной памяти, выделенной для |

| | | |
|---|---|---|
| | | связанного процесса. |
|  | PagedMemorySize | **Устаревшее.** Получает размер выгружаемой памяти. |
|  | PagedMemorySize64 | Получает количество выгружаемой системной памяти, выделенной для связанного процесса. |
|  | PagedSystemMemorySize | **Устаревшее.** Получает размер системной выгружаемой памяти. |
|  | PagedSystemMemorySize64 | Получает количество физической памяти, выделенной для связанного процесса. |
|  | PeakPagedMemorySize | **Устаревшее.** Получает максимальный размер выгружаемой памяти. |
|  | PeakPagedMemorySize64 | Получает максимальное количество памяти в файле подкачки виртуальной памяти, используемой связанным процессом. |
|  | PeakVirtualMemorySize | **Устаревшее.** Получает максимальный размер виртуальной памяти. |
|  | PeakVirtualMemorySize64 | Получает максимальное количество виртуальной памяти, используемой связанным процессом. |
|  | PeakWorkingSet | **Устаревшее.** Получает максимальный размер рабочей памяти для связанного процесса. |
|  | PeakWorkingSet64 | Получает максимальное количество физической памяти, используемой |

| | | |
|---|---|---|
| | | связанным процессом. |
|  | PriorityBoostEnabled | Получает или задает значение, указывающее, должна ли операционная система временно увеличить приоритет связанного процесса, когда основное окно процесса получит фокус. |
|  | PriorityClass | Получает или задает общую категорию приоритета для процесса. |
|  | PrivateMemorySize | **Устаревшее.** Получает размер закрытой памяти. |
|  | PrivateMemorySize64 | Получает количество закрытой памяти, выделенной для связанного процесса. |
|  | PrivilegedProcessorTime | Получает права доступа на время процессора для этого процесса. |
|  | ProcessName | Получает имя процесса. |
|  | ProcessorAffinity | Получает или задает процессоры, на которых может быть запланировано выполнение потоков данного процесса. |
|  | Responding | Получает значение, указывающее, отвечает или нет пользовательский интерфейс. |
|  | SessionId | Получает идентификатор сеанса служб терминалов для связанного процесса. |
|  | Site | Получает или задает экземпляр ISite для компонента Component. (Унаследовано от Component.) |

| | | |
|---|---|---|
| | StandardError | Получает поток, используемый для чтения вывода ошибок приложения. |
| | StandardInput | Получает поток, используемый для записи ввода приложения. |
| | StandardOutput | Получает поток, используемый для чтения вывода приложения. |
| | StartInfo | Получает или задает свойства для передачи их методу Start объекта Process. |
| | StartTime | Получает время запуска связанного процесса. |
| | SynchronizingObject | Получает или задает объект, используемый для маршалинга вызовов обработчика событий, происходящих в результате события завершения процесса. |
| | Threads | Получает множество потоков, выполняющихся в связанном процессе. |
| | TotalProcessorTime | Получает полное время процессора для этого процесса. |
| | UserProcessorTime | Получает пользовательское время процессора для этого процесса. |
| | VirtualMemorySize | **Устаревшее.** Получает размер виртуальной памяти процесса. |
| | VirtualMemorySize64 | Получает количество виртуальной памяти, выделенной для связанного процесса. |
| | WorkingSet | **Устаревшее.** Получает использование физической памяти связанного процесса. |

| | | |
|---|---|---|
| | WorkingSet64 | Получает количество физической памяти, выделенной для связанного процесса. |

В начало страницы

**События класса Process**

| | Имя | Описание |
|---|---|---|
| | Disposed | Происходит при удалении компонента вызовом метода Dispose. (Унаследовано от Component.) |
| | ErrorDataReceived | Происходит, когда приложение записывает в свой перенаправленный поток StandardError. |
| | Exited | Происходит при завершении процесса. |
| | OutputDataReceived | Происходит, когда приложение записывает в свой перенаправленный поток StandardOutput. |

В начало страницы

**См. также**

**Ссылки**

Process Класс

**Методы класса Process**

| | Имя | Описание |
|---|---|---|
| | BeginErrorReadLine | Начинает операции асинхронного считывания с перенаправленного потока StandardError приложения. |
| | BeginOutputReadLine | Начинает операции асинхронного считывания в перенаправленном потоке StandardOutput приложения. |
| | CancelErrorRead | Отменяет операцию асинхронного считывания в перенаправленном потоке StandardError приложения. |

| | | |
|---|---|---|
|  | CancelOutputRead | Отменяет операцию асинхронного считывания в перенаправленном потоке StandardOutput приложения. |
|  | Close | Освобождает все ресурсы, связанные с этим компонентом. |
|  | CloseMainWindow | Закрывает процесс, имеющий пользовательский интерфейс, посылая сообщение о закрытии главному окну процесса. |
|  | CreateObjRef | Создает объект, который содержит всю необходимую информацию для создания прокси-сервера, используемого для взаимодействия с удаленным объектом. (Унаследовано от MarshalByRefObject.) |
|  | Dispose | Перегружен. |
|  | EnterDebugMode | Помещает компонент Process в состояние взаимодействия с работающим системным процессом, выполняющимся в специальном режиме путем включения присущего данному объекту свойства **SeDebugPrivilege** в данном потоке. |
|  | Equals | Определяет, равен ли заданный объект Object текущему объекту Object. (Унаследовано от Object.) |
|  | Finalize | Освобождает неуправляемые ресурсы и выполняет другие операции очистки перед тем, как объект Component будет освобожден при сборке мусора. (Унаследовано от Component.) |
|  | GetCurrentProcess | Получает новый компонент Process и связывает его с активным в |

| | | |
|---|---|---|
| | | данный момент процессом. |
|  | GetHashCode | Играет роль хэш-функции для определенного типа. (Унаследовано от Object.) |
|  | GetLifetimeService | Извлекает объект обслуживания во время существования, который управляет политикой времени существования данного экземпляра. (Унаследовано от MarshalByRefObject.) |
|  | GetProcessById | Перегружен. Создает новый компонент Process и связывает его с существующим заданным ресурсом процесса. |
|  | GetProcesses | Перегружен. Создает массив из новых компонентов Process и связывает их с существующими ресурсами процесса. |
|  | GetProcessesByName | Перегружен. Создает массив из новых компонентов Process и связывает их с существующими ресурсами процесса, для которых имя процесса является общедоступным. |
|  | GetService | Возвращает объект, представляющий службу, обеспечиваемую компонентом Component или его контейнером Container. (Унаследовано от Component.) |
|  | GetType | Возвращает объект Type для текущего экземпляра. (Унаследовано от Object.) |
|  | InitializeLifetimeService | Возвращает объект обслуживания во время существования для управления политикой времени существования данного экземпляра. (Унаследовано от |

| | | |
|---|---|---|
| | | MarshalByRefObject.) |
|  | Kill | Немедленно останавливает связанный процесс. |
|  | LeaveDebugMode | Выбирает компонент Process из состояния, позволяющего ему взаимодействовать с процессами операционной системы, запущенными в специальном режиме. |
|  | MemberwiseClone | Перегружен. |
|  | OnExited | Вызывает событие Exited. |
|  | Refresh | Удаляет любые кэшированные внутри компонента процесса сведения о связанном процессе. |
|  | Start | Перегружен. Запускает ресурс процесса и связывает его с компонентом Process. |
|  | ToString | Преобразует имя процесса в строку, объединенную с родительским типом компонента, если это применимо. (Переопределяет Component.ToString().)<br><br>В .NET Compact Framework 3.5 этот член наследуется от Object.ToString(). |
|  | WaitForExit | Перегружен. Задает период времени для ожидания завершения связанного процесса и блокирует текущий поток выполнения до того, как пройдет это время или процесс завершится. |
|  | WaitForInputIdle | Перегружен. Дает компоненту Process команду ожидать входа связанного процесса в незанятое |

| | | |
|---|---|---|
| | | состояние. |

В начало страницы

**C#** Копировать код

```
using System;
using System.Diagnostics;
using System.ComponentModel;

namespace MyProcessSample
{
      /// <summary>
      /// Shell for the sample.
      /// </summary>
      class MyProcess
      {
            // These are the Win32 error code for file not fou
access denied.
            const int ERROR_FILE_NOT_FOUND =2;
            const int ERROR_ACCESS_DENIED = 5;

            /// <summary>
            /// Prints a file with a .doc extension.
            /// </summary>
            void PrintDoc()
            {
                  Process myProcess = new Process();

                  try
                  {
                        // Get the path that stores user docume
                        string myDocumentsPath =

      Environment.GetFolderPath(Environment.SpecialFolder.Pers

                        myProcess.StartInfo.FileName =
myDocumentsPath + "\\MyFile.doc";
                        myProcess.StartInfo.Verb = "Print";
                        myProcess.StartInfo.CreateNoWindow = tr
                        myProcess.Start();
                  }
                  catch (Win32Exception e)
                  {
                        if(e.NativeErrorCode == ERROR_FILE_NOT_
                        {
                              Console.WriteLine(e.Message + ". C
```

```
the path.");
                        }

                        else if (e.NativeErrorCode ==
ERROR_ACCESS_DENIED)
                        {
                              // Note that if your word processo
might generate exceptions
                              // such as this, which are handlec
first.
                              Console.WriteLine(e.Message +
                                    ". You do not have permissio
print this file.");
                        }
                  }
            }


            public static void Main()
            {
                  MyProcess myProcess = new MyProcess();
                  myProcess.PrintDoc();
            }
      }
}
```

**C#** Копировать код

```
using System;
using System.Diagnostics;
using System.ComponentModel;

namespace MyProcessSample
{
      /// <summary>
      /// Shell for the sample.
      /// </summary>
      class MyProcess
      {

            /// <summary>
            /// Opens the Internet Explorer application.
            /// </summary>
            void OpenApplication(string myFavoritesPath)
            {
```

```
                // Start Internet Explorer. Defaults to the
home page.
                Process.Start("IExplore.exe");

               // Display the contents of the favorites folde
in the browser.
                Process.Start(myFavoritesPath);

            }

            /// <summary>
            /// Opens urls and .html documents using Internet
Explorer.
            /// </summary>
            void OpenWithArguments()
            {
                // url's are not considered documents. They c
only be opened
                // by passing them as arguments.
                Process.Start("IExplore.exe",
"www.northwindtraders.com");

                // Start a Web page using a browser associate
with .html and .asp files.
                Process.Start("IExplore.exe",
"C:\\myPath\\myFile.htm");
                Process.Start("IExplore.exe",
"C:\\myPath\\myFile.asp");
            }

            /// <summary>
            /// Uses the ProcessStartInfo class to start new
processes, both in a minimized
            /// mode.
            /// </summary>
            void OpenWithStartInfo()
            {
                ProcessStartInfo startInfo = new
ProcessStartInfo("IExplore.exe");
                startInfo.WindowStyle =
ProcessWindowStyle.Minimized;
                Process.Start(startInfo);
                startInfo.Arguments =
"www.northwindtraders.com";
                Process.Start(startInfo);
```

```
            }

            static void Main()
            {
                  // Get the path that stores favorite links.
                  string myFavoritesPath =

      Environment.GetFolderPath(Environment.SpecialFolder.Fav
ites);

                        MyProcess myProcess = new MyProcess();

                  myProcess.OpenApplication(myFavoritesPath);
                  myProcess.OpenWithArguments();
                  myProcess.OpenWithStartInfo();

                  }
      }
}
```