Threading III

dr Srđan Sladojević

Threading III

- SADRŽAJ:
  - Imenovanje niti
  - Stanja niti
  - Stopiranje rada niti
  - Otkirvanje gresaka – debugging i logging

Imenovanje niti

Imenovanje niti koriscenjem **Name** polja (properti) objekta klase **Thread**.

Prakticno zbog debagovanja i pracenja toka niti.

Moze se samo jednom imenovati nit, pokusaj izmene imena izazvace gresku…

Thread.CurrentThread polje omogucava pristup niti koja se trenutno izvrsava

Imenovanje niti

Imenovanje niti

```
6  public class Test
7  {
8      static void Main()
9      {
10         Thread.CurrentThread.Name = "main";
11         Thread worker = new Thread(Go);
12         worker.Name = "worker";
13         worker.Start();
14         Go();
15     }
16
17     static void Go()
18     {
19         Console.WriteLine("Hello from " + Thread.CurrentThread.Name);
20     }
21 }
```

Prioritet niti

Prioritet niti

Polje niti koje odredjuje koliko je vreme izvrsavanja niti u zavisnosti od drugih aktivnih niti.

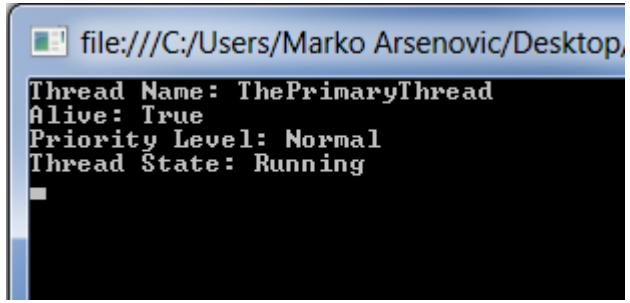
enum ThreadPriority { Lowest, BelowNormal, Normal, AboveNormal, Highest }

Prioritet dolazi do izrazaja tek u situacijama simultanog pristupa resursu.

Prioritet niti

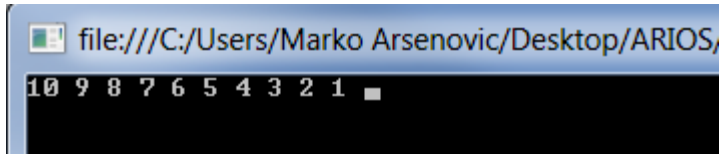
Primer 1

```
4   class MainClass
5   {
6       // [MTAThread]
7       [STAThread]
8       static void Main(string[] args)
9       {
10          Thread primaryThread = Thread.CurrentThread;
11
12          primaryThread.Name = "ThePrimaryThread";
13
14          Console.WriteLine("Thread Name: {0}", primaryThread.Name);
15          Console.WriteLine("Alive: {0}", primaryThread.IsAlive);
16          Console.WriteLine("Priority Level: {0}", primaryThread.Priority);
17          Console.WriteLine("Thread State: {0}", primaryThread.ThreadState);
18      }
19  }
```



Prioritet niti

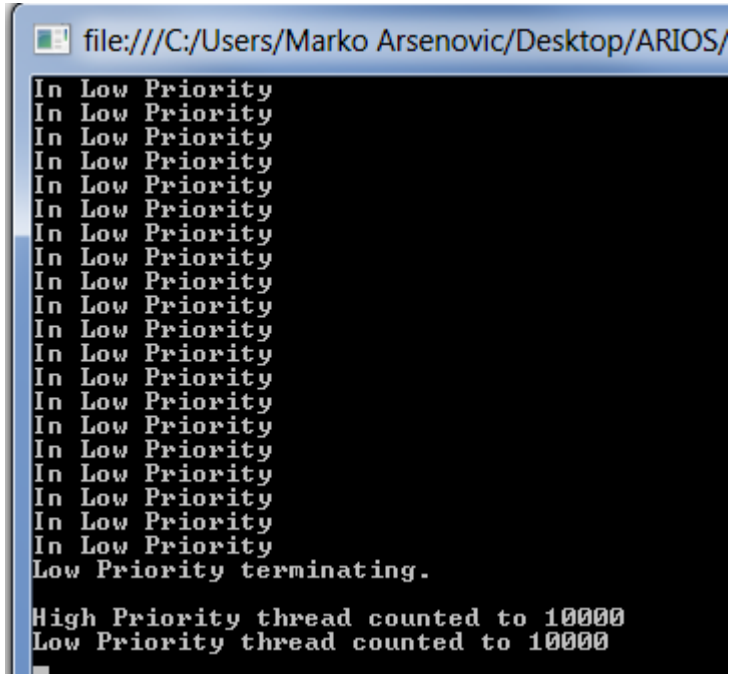
Primer 2




```
1  using System;
2  using System.Threading;
3
4  class MainClass
5  {
6      public static void Countdown()
7      {
8          for (int i = 10; i > 0; i--)
9          {
10             Console.Write(i.ToString() + " ");
11         }
12     }
13
14     public static void Main()
15     {
16         Thread t2 = new Thread(new ThreadStart(Countdown));
17
18         t2.Priority = ThreadPriority.Highest;
19
20         Thread.CurrentThread.Priority = ThreadPriority.Lowest;
21
22         t2.Start();
23
24         Console.ReadLine();
25     }
26 }
```

Prioritet niti

Primer 3




```
1  using System;
2  using System.Threading;
3
4  class MyThread
5  {
6      public int count;
7      public Thread thrd;
8
9      public MyThread(string name)
10     {
11         count = 0;
12         thrd = new Thread(this.run);
13         thrd.Name = name;
14     }
15
16     void run()
17     {
18         Console.WriteLine(thrd.Name + " starting.");
19         do
20         {
21             count++;
22
23             Console.WriteLine("In " + thrd.Name);
24         } while (count < 10000);
25
26         Console.WriteLine(thrd.Name + " terminating.");
27     }
28 }
29
```

Prioritet niti

Primer 3

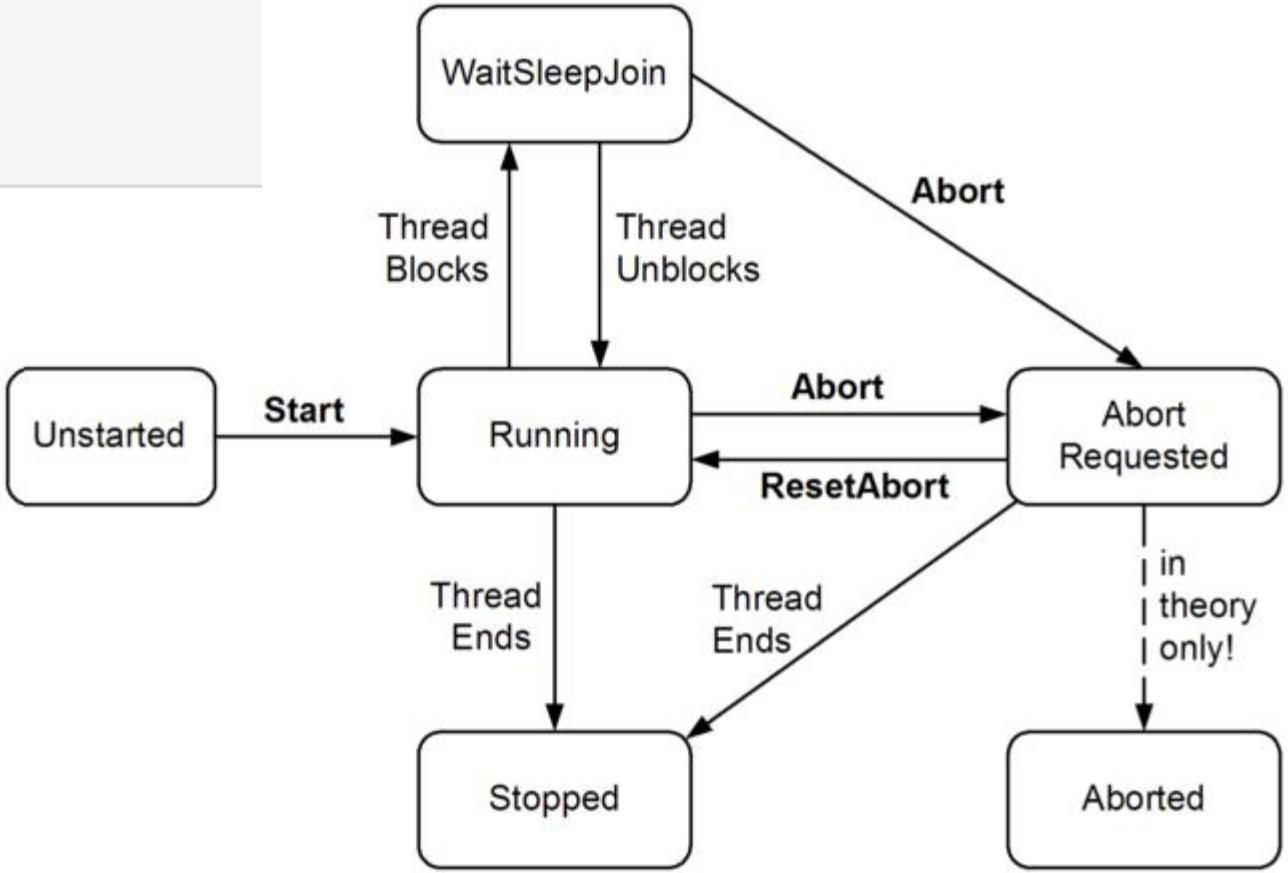
```
file:///C:/Users/Marko Arsenovic/Desktop/ARIOS/
In Low Priority
In Low Priority
In Low Priority
In Low Priority
In Low Priority
In Low Priority
In Low Priority
In Low Priority
In Low Priority
In Low Priority
In Low Priority
In Low Priority
In Low Priority
In Low Priority
In Low Priority
In Low Priority
In Low Priority
In Low Priority
In Low Priority
In Low Priority
Low Priority terminating.

High Priority thread counted to 10000
Low Priority thread counted to 10000
```

```
30 class PriorityDemo
31 {
32     public static void Main()
33     {
34         MyThread mt1 = new MyThread("High Priority");
35         MyThread mt2 = new MyThread("Low Priority");
36
37         mt1.thrd.Priority = ThreadPriority.AboveNormal;
38         mt2.thrd.Priority = ThreadPriority.BelowNormal;
39
40         mt1.thrd.Start();
41         mt2.thrd.Start();
42
43         mt1.thrd.Join();
44         mt2.thrd.Join();
45
46         Console.WriteLine();
47         Console.WriteLine(mt1.thrd.Name + " thread counted to " + mt1.count);
48         Console.WriteLine(mt2.thrd.Name + " thread counted to " + mt2.count);
49     }
50 }
```

Stanja niti

Stanja niti

```
public static ThreadState SimpleThreadState (ThreadState ts)
{
  return ts & (ThreadState.Unstarted |
               ThreadState.WaitSleepJoin |
               ThreadState.Stopped);
}
```



Stanja niti

Primer 1

```
class MainClass
{
    public static void Countdown()
    {
        for (int i = 10; i > 0; i--)
        {
            Console.Write(i.ToString() + " ");
        }
    }

    public static void DumpThreadState(Thread t)
    {
        Console.Write("Current state: ");
        if ((t.ThreadState & ThreadState.Aborted) == ThreadState.Aborted)
            Console.Write("Aborted ");
        if ((t.ThreadState & ThreadState.AbortRequested) == ThreadState.AbortRequested)
            Console.Write("AbortRequested ");
        if ((t.ThreadState & ThreadState.Background) == ThreadState.Background)
            Console.Write("Background ");
        if ((t.ThreadState & (ThreadState.Stopped | ThreadState.Unstarted | ThreadState.Aborted)) == 0)
            Console.Write("Running ");
        if ((t.ThreadState & ThreadState.Stopped) == ThreadState.Stopped)
            Console.Write("Stopped ");
        if ((t.ThreadState & ThreadState.StopRequested) == ThreadState.StopRequested)
            Console.Write("StopRequested ");
        if ((t.ThreadState & ThreadState.Suspended) == ThreadState.Suspended)
            Console.Write("Suspended ");
        if ((t.ThreadState & ThreadState.SuspendRequested) == ThreadState.SuspendRequested)
            Console.Write("SuspendRequested ");
        if ((t.ThreadState & ThreadState.Unstarted) == ThreadState.Unstarted)
            Console.Write("Unstarted ");
        if ((t.ThreadState & ThreadState.WaitSleepJoin) == ThreadState.WaitSleepJoin)
            Console.Write("WaitSleepJoin ");
    }
```

Stanja niti

Primer 1

```
39      public static void Main()
40      {
41          Thread t2 = new Thread(new ThreadStart(Countdown));
42          DumpThreadState(t2);
43
44          t2.Start();
45          DumpThreadState(t2);
46
47          Countdown();
48
49          t2.Abort();
50          DumpThreadState(t2);
51
52          Console.ReadLine();
53      }
54  }
```

file:///C:/Users/Marko Arsenovic/Desktop/ARIOS/2/Threading/Threading/bin/Debug/Threading.EXE

```
Current state: Unstarted Current state: Running 10 9 8 7 6 10 9 8 7 6 5 4 3 2 1 5 4 3 2 1 Current state: Aborted
```

Stanja niti

Primer 2

isAlive

Vraća *bool* koji nam govori da li je nit pokrenuta.

```
4   class MyThread
5   {
6       public int count;
7       public Thread thrd;
8
9       public MyThread(string name)
10      {
11          count = 0;
12          thrd = new Thread(this.run);
13          thrd.Name = name;
14          thrd.Start();
15      }
16
17      void run()
18      {
19          Console.WriteLine(thrd.Name + " starting.");
20
21          do
22          {
23              Thread.Sleep(500);
24              Console.WriteLine("In " + thrd.Name +
25                                ", count is " + count);
26              count++;
27          } while (count < 10);
28
29          Console.WriteLine(thrd.Name + " terminating.");
30      }
31  }
32
```

Zaustavljanje rada niti

Zaustavljanje rada niti – metoda Abort()

Metoda **Abort**() pokusava da imperativno prekine izvrsavanje druge niti, pri cemu prouzrokuje izuzetak **ThreadAbortException** unutar niti tacno na mestu gde se nit izvrsava.

Izuzetak ThreadAbortException se moze presresti i obraditi,neobicno je to sto se on ponovo generise na kraju catch bloka, osim ako unutar catch bloka ne pozovete metodu **ResetAbort**()

Zaustavljanje rada niti

Primer 1

```
1  using System;
2  using System.Threading;
3
4  class MyThread
5  {
6      public Thread thrd;
7
8      public MyThread(string name)
9      {
10         thrd = new Thread(this.run);
11         thrd.Name = name;
12         thrd.Start();
13     }
14
15     void run()
16     {
17         Console.WriteLine(thrd.Name + " starting.");
18
19         for (int i = 1; i <= 100; i++)
20         {
21             Console.Write(i + " ");
22             Thread.Sleep(50);
23         }
24         Console.WriteLine(thrd.Name + " exiting.");
25     }
26 }
27
```

Zaustavljanje rada niti

Primer 1

```
28  class MainClass
29  {
30      public static void Main()
31      {
32          MyThread mt1 = new MyThread("My Thread");
33
34          Thread.Sleep(1000);
35
36          Console.WriteLine("Stopping thread.");
37          mt1.thrd.Abort();
38
39          mt1.thrd.Join();
40
41          Console.WriteLine("Main thread terminating.");
42
43          Console.ReadLine();
44      }
45  }
```

file:///C:/Users/Marko Arsenovic/Desktop/ARIOS/2/Threading/Threading/bi...

```
My Thread starting.
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 Stopping thread.
Main thread terminating.
```

Zaustavljanje rada niti

Primer 2

```
1  using System;
2  using System.Threading;
3
4  class MainClass
5  {
6      static int MyCount = 0;
7      static void Main(string[] args)
8      {
9          MyClassThread me = new MyClassThread();
10         Thread MyNewThread = new Thread(new ThreadStart(me.MyThread));
11
12         MyNewThread.Start();
13         if (MyCount == 0)
14             MyNewThread.Abort();
15
16         Console.ReadLine();
17     }
18 }
19
20 class MyClassThread
21 {
22     public void MyThread()
23     {
24     }
25 }
```

Zaustavljanje rada niti

Primer 3

ResetAbort()

```
4   class MyThread
5   {
6       public Thread thrd;
7
8       public MyThread(string name)
9       {
10          thrd = new Thread(this.run);
11          thrd.Name = name;
12          thrd.Start();
13      }
14
15      void run()
16      {
17          Console.WriteLine(thrd.Name + " starting.");
18
19          for (int i = 1; i <= 100; i++)
20          {
21              try
22              {
23                  Console.Write(i + " ");
24                  Thread.Sleep(50);
25              }
26              catch (ThreadAbortException exc)
27              {
28                  if ((int)exc.ExceptionState == 0)
29                  {
30                      Console.WriteLine("Abort Cancelled! Code is " + exc.ExceptionState);
31                      Thread.ResetAbort();
32                  }
33                  else
34                      Console.WriteLine("Thread aborting, code is " + exc.ExceptionState);
35              }
36          }
37          Console.WriteLine(thrd.Name + " exiting normally.");
38      }
39  }
```

Zaustavljanje rada niti

Primer 3

ResetAbort()

```
41 class MainClass
42 {
43     public static void Main()
44     {
45         MyThread mt1 = new MyThread("My Thread");
46
47         Thread.Sleep(1000);
48
49         Console.WriteLine("Stopping thread.");
50         mt1.thrd.Abort(100);
51
52         mt1.thrd.Join();
53
54         Console.WriteLine("Main thread terminating.");
55     }
56 }
```

file:///C:/Users/Marko Arsenovic/Desktop/ARIOS/2/Threading/Threading/bi...

```
My Thread starting.
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 Stopping thread.
Thread aborting, code is 100
Main thread terminating.
```

Zaustavljanje rada niti

Primer 4

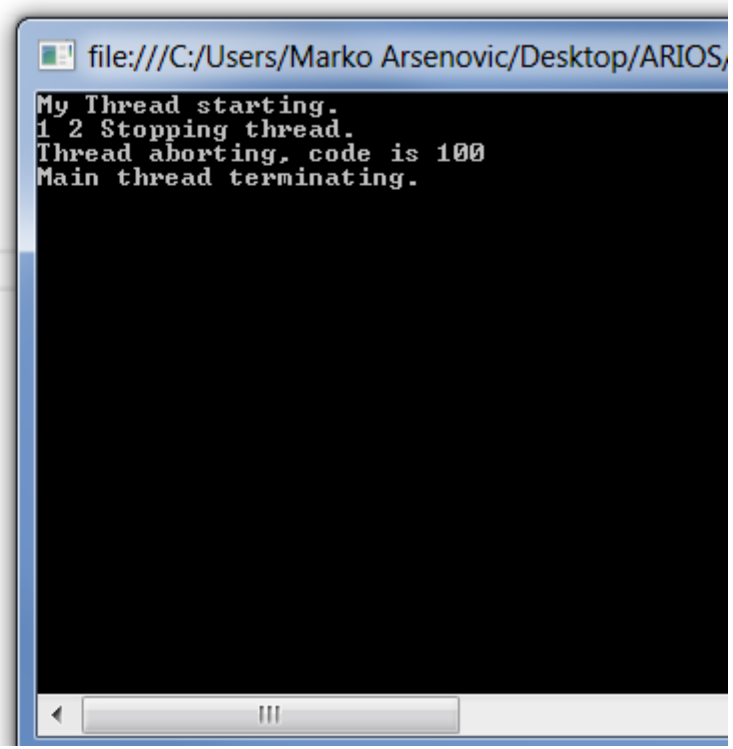
ThreadAbortException

```
1  using System;
2  using System.Threading;
3
4  class MyThread
5  {
6      public Thread thrd;
7
8      public MyThread(string name)
9      {
10         thrd = new Thread(this.run);
11         thrd.Name = name;
12         thrd.Start();
13     }
14
15     void run()
16     {
17         try
18         {
19             Console.WriteLine(thrd.Name + " starting.");
20
21             for (int i = 1; i <= 100; i++)
22             {
23                 Console.Write(i + " ");
24                 Thread.Sleep(50);
25             }
26             Console.WriteLine(thrd.Name + " exiting normally.");
27         }
28         catch (ThreadAbortException exc)
29         {
30             Console.WriteLine("Thread aborting, code is " +
31                                 exc.ExceptionState);
32         }
33     }
34 }
35
```

Zaustavljanje rada niti

Primer 5

```
36  class MainClass
37  {
38      public static void Main()
39      {
40          MyThread mt1 = new MyThread("My Thread");
41
42          Thread.Sleep(100);
43
44          Console.WriteLine("Stopping thread.");
45          mt1.thrd.Abort(100);
46
47          mt1.thrd.Join();
48
49          Console.WriteLine("Main thread terminating.");
50          Console.ReadLine();
51      }
52  }
```



Zaustavljanje rada niti

Primer bezbednog zaustavljanja niti

```
main thread: starting worker thread...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: terminating gracefully...
main thread: worker thread has terminated
```

```
using System;
using System.Threading;

public class Worker
{
    // This method will be called when the thread is started.
    public void DoWork()
    {
        while (!_shouldStop)
        {
            Console.WriteLine("worker thread: working...");
        }
        Console.WriteLine("worker thread: terminating gracefully.");
    }
    public void RequestStop()
    {
        _shouldStop = true;
    }
    // Volatile is used as hint to the compiler that this data
    // member will be accessed by multiple threads.
    private volatile bool _shouldStop;
}
```

```
public class WorkerThreadExample
{
    static void Main()
    {
        // Create the thread object. This does not start the thread.
        Worker workerObject = new Worker();
        Thread workerThread = new Thread(workerObject.DoWork);

        // Start the worker thread.
        workerThread.Start();
        Console.WriteLine("main thread: Starting worker thread...");

        // Loop until worker thread activates.
        while (!workerThread.IsAlive);

        // Put the main thread to sleep for 1 millisecond to
        // allow the worker thread to do some work:
        Thread.Sleep(1);

        // Request that the worker thread stop itself:
        workerObject.RequestStop();

        // Use the Join method to block the current thread
        // until the object's thread terminates.
        workerThread.Join();
        Console.WriteLine("main thread: Worker thread has terminated.");
    }
}
```

Otkrivanje gresaka

Otkrivanje gresaka u visenitnom programiranju je znatno teze od otklanjanja gresaka u single-thread aplikacijama.

Bagovi u visenitnim programima su zavisni od vremena dogadjaja u aplikaciji, u debag modu dolazi do menjanja tajminga i kao rezultat tome moze doci do maskiranja problema.

Visual Studio poseduje podrsku za debug visenitnih aplikacija, i to predstavlja jedan od nacina detekcije greske.

Otkrivanje gresaka

Primer konzolne aplikacije na kojoj cemo pokazati nacin otkrivanja greske korisenjem Thread Window-a unutar VS-a.

```
01. namespace DebugThread
02. {
03.     class Program
04.     {
05.         static int i = 0;
06.         static void Main(string[] args)
07.         {
08.             Thread T1 = new Thread(MyMethod);
09.             Thread T2 = new Thread(MyMethod);
10.             T1.Start();
11.             T2.Start();
12.             Console.Read();
13.         }
14.         static void MyMethod()
15.         {
16.             for (int i = 0; i < 10; i++)
17.             {
18.                 i++;
19.                 Thread.Sleep(5000);
20.             }
21.         }
22.     }
23. }
```

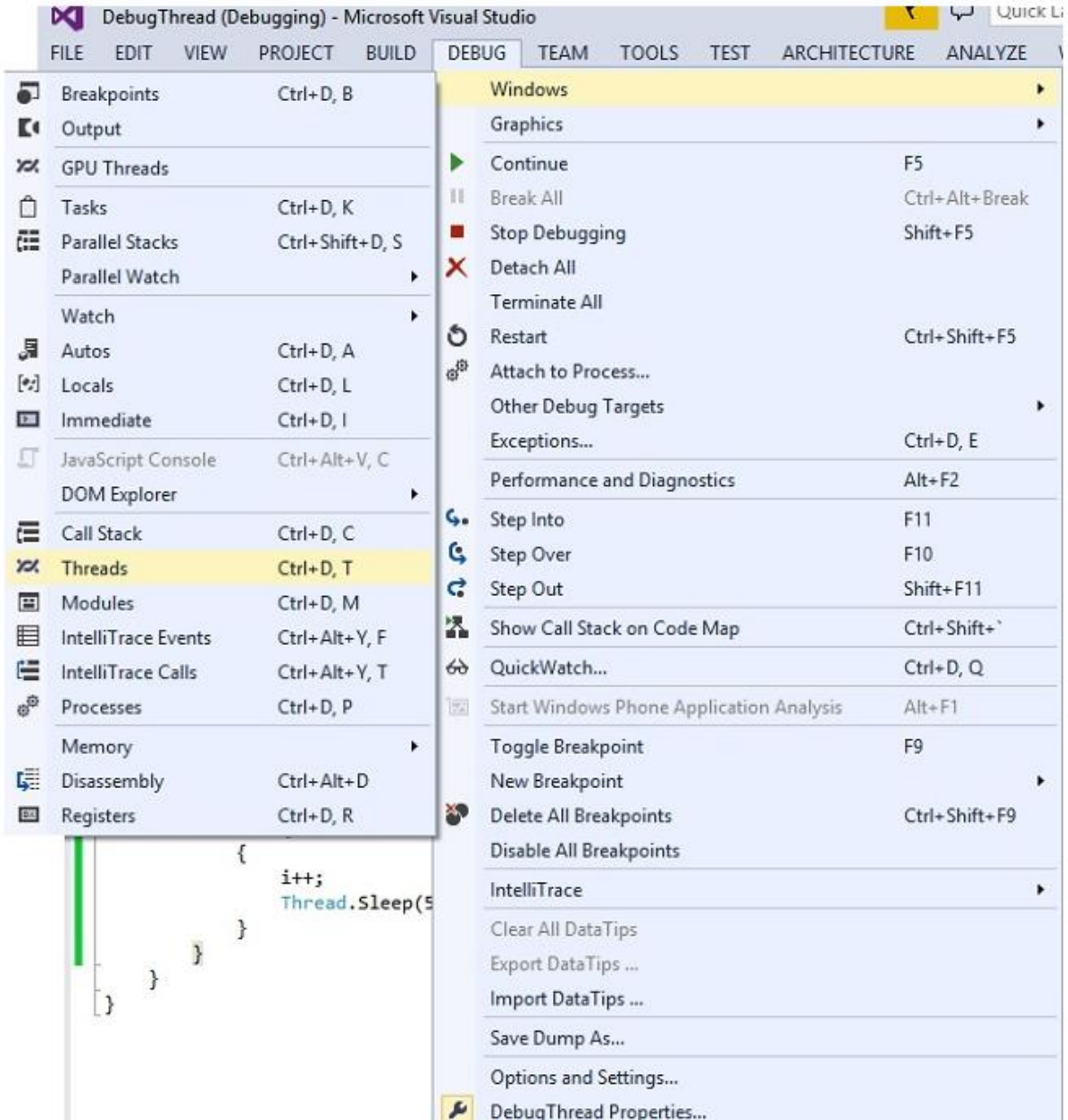
Otkrivanje gresaka

Postavimo breakpoint

```
2 references
static void MyMethod()
{
    for (int i = 0; i < 10; i++)
    {
        i++;
        Thread.Sleep(5000);
    }
}
```

Otkrivanje gresaka

Otvaranje Thread Window-a, tokom debug moda



Otkrivanje gresaka

Kada se aplikacija pokrene, izgled Thread Window-a

Threads

Search: | Search Call Stack | Group by: None | Columns

	ID	Managed ID	Category	Name	Location	Priority
	0	0	Unknown Thread	[Thread Destroyed]	<not available>	
	26976	0	Worker Thread	<No Name>	<not available>	Normal
	19352	3	Worker Thread	<No Name>	<not available>	Normal
	1328	8	Worker Thread	vshost.RunParkingWindow	Microsoft.VisualStudio.HostingProcess.Utilities.dll!Microsoft.VisualStudio.HostingP	Normal
	11556	7	Worker Thread	<No Name>	<not available>	Normal
	21360	0	Worker Thread	<No Name>	<not available>	Normal
	27828	9	Worker Thread	.NET SystemEvents	System.dll!Microsoft.Win32.SystemEvents.WindowThreadProc	Normal
	14352	10	Main Thread	Main Thread	DebugThread.exe!DebugThread.Program.Main	Normal
	39820	11	Worker Thread	<No Name>	DebugThread.exe!DebugThread.Program.MyMethod	Normal
	608	12	Worker Thread	<No Name>	DebugThread.exe!DebugThread.Program.MyMethod	Normal

Dve radne niti nisu imenovane, kada se imenuju…

```
static void Main(string[] args)
{
    Thread T1 = new Thread(MyMethod);
    T1.Name = "T1";
    Thread T2 = new Thread(MyMethod);
    T2.Name = "T2";
    T1.Start();
    T2.Start();
    Console.Read();
}
```

Otkrivanje gresaka

Mozemo da vidimo dve niti iz nase aplikacije…

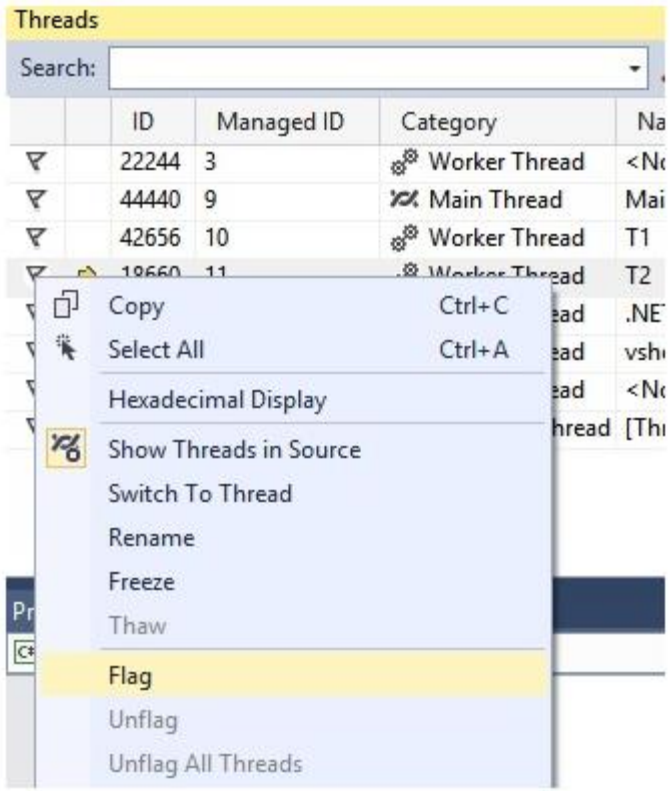
Threads

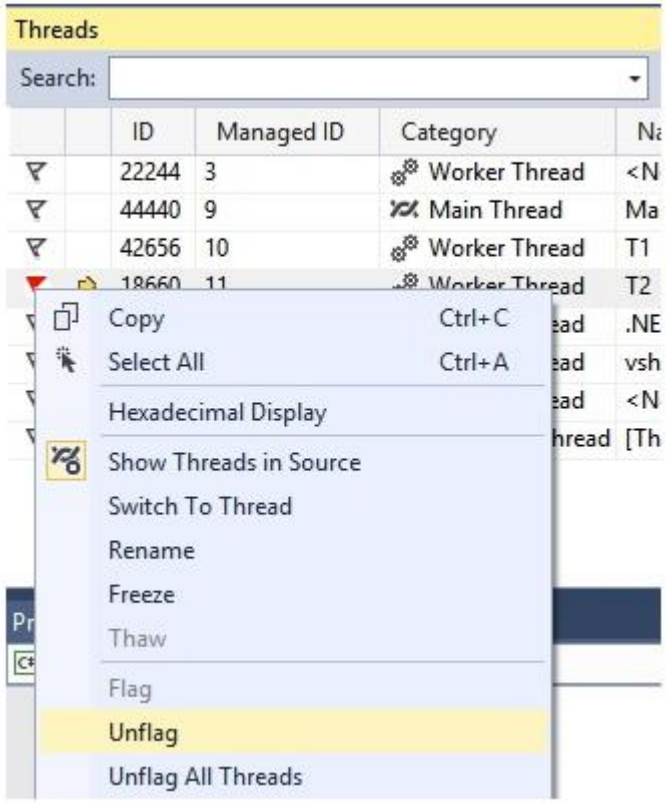
Search: Search Call Stack Group by: None Columns

ID	Managed ID	Category	Name	Location	Priority
0	0	Unknown Thread	[Thread Destroyed]	<not available>	
27540	0	Worker Thread	<No Name>	<not available>	Normal
22244	3	Worker Thread	<No Name>	<not available>	Normal
53680	6	Worker Thread	vshost.RunParkingWindow	Microsoft.VisualStudio.HostingProcess.Utilities.dll!Microsoft.VisualStudio.HostingP	Normal
44440	9	Main Thread	Main Thread	DebugThread.exe!DebugThread.Program.Main	Normal
55168	8	Worker Thread	.NET SystemEvents	System.dll!Microsoft.Win32.SystemEvents.WindowThreadProc	Normal
42656	10	Worker Thread	T1	DebugThread.exe!DebugThread.Program.MyMethod	Normal
18660	11	Worker Thread	T2	DebugThread.exe!DebugThread.Program.MyMethod	Normal

Otkrivanje gresaka

- **Flagging, Unflagging Thread**



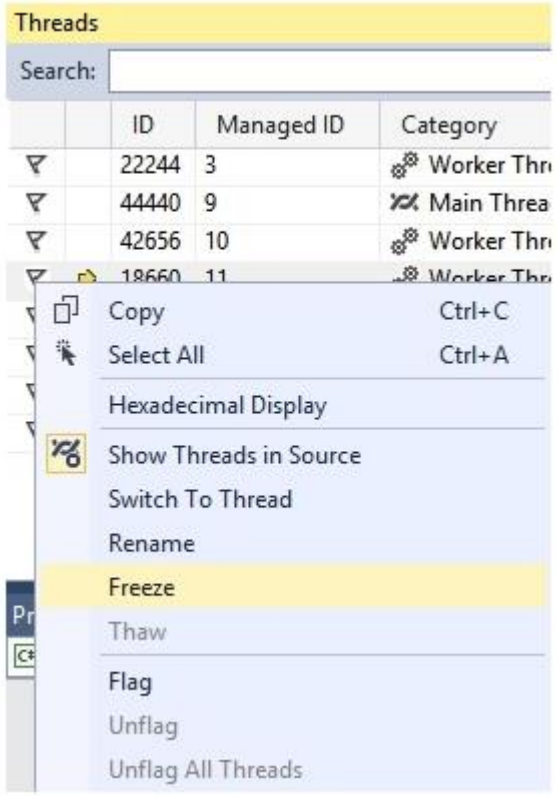


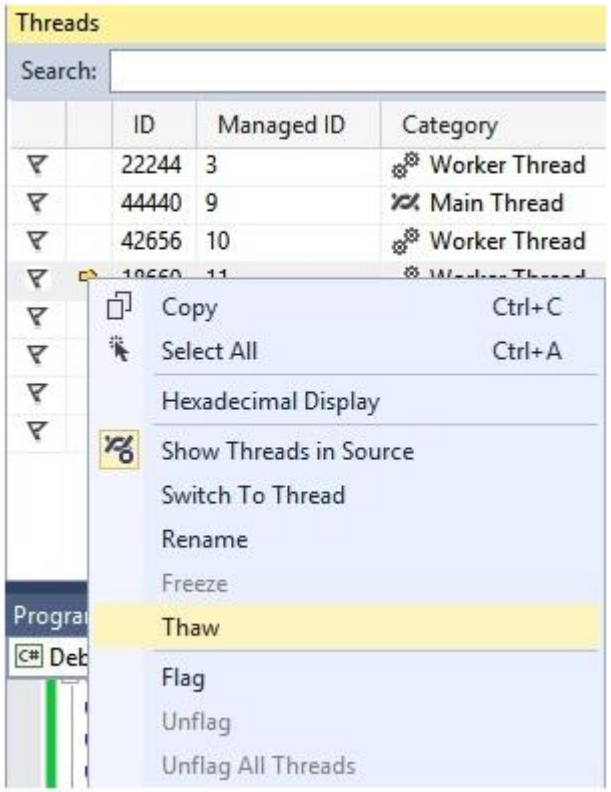
Otkrivanje gresaka

- **Freeze, Thaw niti**
- **Freezing** niti zaustavlja njeno izvrsavanje, u debaging procesu. Mozemo reci da **Freeze** zaustavlja process(***suspand***) a **Thawing** nastavlja process (***resume***).
- Pomocu Freezing i Thawing mozemo imati kontrolu nad izvrsavanjima niti u procesu debagovanja.
- Pomaze nam prilikom resavanja bagova u konkurentnom procesu

Otkrivanje gresaka

- U Threads window-u, desni klik na bilo koju nit i zatim klik na **Freeze**, nad *frozen* niti desni klik zatim **Thaw** za nastavak izvrsavanja.





Otkrivanje gresaka

- Druga, vrlo cesta tehnika je logovanje.
- Real-time belezenje izvrsavanja niti zapisivanjem u fajl.
- Primer thread safe nacina implementacije logovanja.

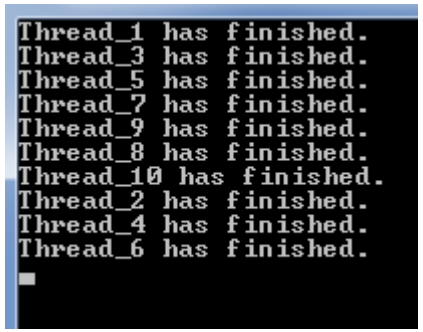
```
public class Logging
{
    public Logging()
    {
    }

    private static readonly object locker = new object();

    public void WriteToLog(string message)
    {
        lock(locker)
        {
            StreamWriter SW;
            SW=File.AppendText("Data\\Log.txt");
            SW.WriteLine(message);
            SW.Close();
        }
    }
}
```

Zadatak

- ZADATAK 1
- U main-u konzolne aplikacije kreirati 10 novih Thread-ova. Svi threadovi se startuju u „isto“ vreme. Svaki Thread treba da nosi name: „Thread_n“, gde je n = (1,2,3 ... 10). Svaki Thread poziva metodu Work, u kojoj spava 2 sekunde (ne vrsi nikakvu kompleksnu operaciju) i nakon toga loguje da je zavrsio svoj process. Implementirati osnovnu sinhronizaciju threado-va tako da parni thread-ovi moraju da cekaju da se zavrse procesi neparnih thread-ova. Ni jedan parni thread ne sme svoj proces da zavrsi pre zavrsetka svih parnih thread-ova. Zavrsetke proces-a thread-ova jednostavno izlogovati.

Zadatak

- ZADATAK 1
- RESENJE




```
public class Program
{
    private static int activeOddThreadsCount = 5;

    static void Main(string[] args)
    {
        for (int i = 0; i < 10; i++)
        {
            Thread t = new Thread(new ThreadStart(() => DoWork()))
            {
                Name = string.Format("Thread_{0}", i+1),
                IsBackground = true
            };

            t.Start();
        }
        Console.ReadKey();
    }

    private static void DoWork()
    {
        Thread.Sleep(2000);
        var currentThreadName = Thread.CurrentThread.Name;
        var threadId = Convert.ToInt32(currentThreadName.Split(new char[1] { '_' })[1]);

        if(threadId%2 == 0)
        {
            while(activeOddThreadsCount>0)
            {
                Thread.Sleep(25);
            }
        }
        else
        {
            activeOddThreadsCount--;
        }

        Console.WriteLine(string.Format("{0} has finished.", Thread.CurrentThread.Name));
    }
```