



Тема 6.

Потоки. Многопоточность.

© 2013-2014, Serge Kashkevich

Недостатки проекта alarm I

Если головная программа проекта alarm I реализована так, как показано в предыдущей теме, кажется, что всё нормально:

```
public static void Main() {  
    DateTime wakeUpTime = DateTime.Now;  
    wakeUpTime = wakeUpTime.AddMinutes(300);  
    alarm = new Alarm(wakeUpTime, 1);  
    alarm.AlarmEvent += new AlarmEventHandler(wakeUp);  
    alarm.Set();  
}
```

Недостатки проекта alarm I (продолжение)

Однако рассмотрим ситуацию, когда человек просыпается раньше, чем срабатывает будильник, и желает этот будильник отключить:

```
public static void Main() {  
    DateTime wakeUpTime = DateTime.Now;  
    wakeUpTime = wakeUpTime.AddMinutes(60);  
    // надо проснуться через час  
    alarm = new Alarm(wakeUpTime, 1);  
    alarm.AlarmEvent += new AlarmEventHandler(wakeUp);  
    alarm.Set();  
    System.Threading.Thread.Sleep(3000000);  
    // а проснулся сам через 50 минут  
    alarm.Active = false;  
}
```

Отключение будильника не срабатывает, хотя в методе Set вроде бы этот механизм предусмотрен:

```
public void Set() {  
    while (Active) {  
        ...  
    }  
}
```

Недостатки проекта alarm (причины)

Выделенный красным фрагмент кода срабатывает только после завершения метода alarm.Set()

```
public static void Main() {  
    DateTime wakeUpTime = DateTime.Now;  
    wakeUpTime = wakeUpTime.AddMinutes(60);  
    alarm = new Alarm(wakeUpTime, 1);  
    alarm.AlarmEvent += new AlarmEventHandler(wakeUp);  
    alarm.Set();  
    System.Threading.Thread.Sleep(3000000);  
    alarm.Active = false;  
}
```

Недостатки проекта alarm (подход к решению проблемы)

Разделить выполнение проекта на две параллельно выполняющиеся части:

- работа будильника;
- работа человека, который завёл будильник.

Такой механизм возможен благодаря использованию *тредов* (потоков выполнения, thread)

По умолчанию для каждого приложения создаётся единственный тред.

Треды одного процесса разделяют время процессора и, возможно, конкурируют за другие ресурсы (в первую очередь, за память).

Приложение завершается после того, как завершены все его треды.

Определение треда

Тред представляет собой координируемую единицу исполняемого кода. При организации многозадачности на основе тредов у каждого процесса должен быть по крайней мере один тред, хотя их может быть и больше.

С каждым тредом связывается *тредовая функция*, которую выполняет тред.

Тред может находиться в одном из нескольких состояний:

- выполняющимся;
- готовым к выполнению, как только он получит время и ресурсы ЦП;
- приостановленным, т.е. временно не выполняющимся;
- возобновленным в дальнейшем;
- заблокированным в ожидании ресурсов для своего выполнения;
- завершенным, когда его выполнение окончено и не может быть возобновлено.

Преимущества многотредовых приложений

- Главное преимущество многотредовой обработки заключается в том, что она позволяет писать программы, которые работают очень эффективно благодаря возможности выгодно использовать время простоя, неизбежно возникающее в ходе выполнения большинства программ.
- При моделировании параллельной работы нескольких объектов логично каждый объект обрабатывать в отдельном треде для того, чтобы треды взаимодействовали друг с другом.

Создание и запуск треда

Для создания треда необходимо определить объект класса `Thread`, передав в качестве параметра конструктора тредовую функцию. Возможны два формата этой функции:

- `void func();`
- `void func(object);`

После создания треда он может быть запущен на выполнение вызовом метода `Start()` или `Start(object)`

Завершение тредовой функции влечёт переход треда в *завершённое* состояние.

Повторный запуск метода `Start` вызывает исключение!

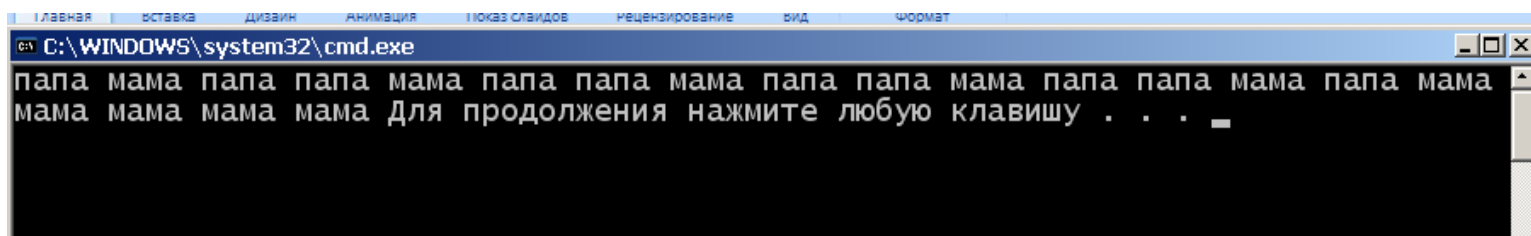
Пример I создания и запуска тредов

```
using System.Threading;
class Program {
    public static void Main() {
        Thread t1 = new Thread(SayMama);
        t1.Start();
        Thread t2 = new Thread(SayPapa);
        t2.Start();
    }

    public static void SayMama() {
        for (int i=0; i<10; i++) {
            Thread.Sleep(1000);
            Console.Write("мама ");
        }
    }

    public static void SayPapa() {
        for (int i = 0; i < 10; i++) {
            Thread.Sleep(500);
            Console.Write("папа ");
        }
    }
}
```

Пример I : результат



A screenshot of a Windows command prompt window. The title bar shows the path 'C:\WINDOWS\system32\cmd.exe'. The command prompt displays two lines of text: 'папа мама папа папа мама папа папа мама папа папа мама папа папа мама папа мама' on the first line, and 'мама мама мама мама Для продолжения нажмите любую клавишу . . . _' on the second line. The window has a standard Windows XP-style interface with a menu bar and window controls.

```
C:\WINDOWS\system32\cmd.exe
папа мама папа папа мама папа папа мама папа папа мама папа папа мама папа мама
мама мама мама мама Для продолжения нажмите любую клавишу . . . _
```

Пример 2 создания и запуска тредов

Рассмотрим пример, когда тредовая функция получает параметр:

```
using System.Threading;

public class SayClass {
    int ms;
    String word;

    public SayClass(int ms, String word) {
        this.ms = ms;
        this.word = word;
    }

    public int MS { get { return ms; } }

    public String Word { get { return word; } }

}
```

Пример 2 (продолжение)

```
class Program {  
    public static void Main() {  
        Thread t1 = new Thread(Say);  
        t1.Start(new SayClass(1000, "мама "));  
        Thread t2 = new Thread(Say);  
        t2.Start(new SayClass(500, "папа "));  
    }  
  
    public static void Say(object ob) {  
        for (int i = 0; i < 10; i++) {  
            Thread.Sleep((ob as SayClass).MS);  
            Console.Write((ob as SayClass).Word + " ");  
        }  
    }  
}
```

Взаимодействие тредов

Треды после их запуска могут взаимодействовать друг с другом. В частности, можно:

- проверить состояние, в котором находится другой тред;
- приостановить другой тред;
- возобновить его работу;
- дождаться завершения работы треда;
- передавать сообщения другим тредам.

Проверка состояния треда

Свойство только для чтения

```
public TreadState Thread.ThreadState {get;}
```

позволяет проверить состояние, в котором находится другой тред. Некоторые возможные значения этого свойства:

Unstarted	тред создан, но метод Start ещё не запущен
Running	тред выполняется
Stopped	тред завершил работу
WaitSleepJoin	тред заблокирован
Suspended	работа треда приостановлена
AbortRequested	поступил запрос на принудительное завершение работы треда

Определение момента окончания работы треда

Свойство только для чтения

```
public bool IsAlive {get; }
```

возвращает логическое значение true, если поток, для которого оно вызывается, по-прежнему выполняется.

Пример использования этого свойства:

```
public static void Main() {  
    Thread t1 = new Thread(Say);  
    t1.Start(new SayClass(1000, "мама "));  
    Thread t2 = new Thread(Say);  
    t2.Start(new SayClass(500, "папа "));  
    do {  
        Thread.Sleep(100);  
    }  
    while (t1.IsAlive || t2.IsAlive);  
    Console.WriteLine("Работа тредов завершена!");  
}
```

Приостановка своей работы

Метод `Join()`, вызванный для другого треда, переводит собственный тред в режим ожидания до тех пор, пока поток, для которого он был вызван, не завершится. Его имя отражает принцип ожидания до тех пор, пока вызываемый тред не *присоединится* к треду, который запросил присоединение. Если же вызываемый тред не был начат, то генерируется исключение `ThreadStateException`.

В других формах метода `Join` можно указать максимальный период времени, в течение которого следует ожидать завершения указанного треда.

Пример использования этого метода:

```
public static void Main() {
    Thread t1 = new Thread(Say);
    t1.Start(new SayClass(1000, "мама "));
    Thread t2 = new Thread(Say);
    t2.Start(new SayClass(500, "папа "));
    t1.Join();
    t2.Join();
    Console.WriteLine("Работа тредов завершена!");
}
```


Приостановка и возобновление работы треда

В первоначальных версиях среды .NET Framework тред можно было приостановить вызовом метода `Suspend ()` и возобновить вызовом метода `Resume ()`.

Но теперь оба эти метода считаются устаревшими.

Объясняется это тем, что:

- пользоваться методом `Suspend` на самом деле небезопасно, так как с его помощью можно приостановить тред, который в настоящий момент удерживает блокировку (*механизм блокировки будет объяснён позднее*);
- возобновлённый тред не получает информацию о том, что внешние условия изменились (*прогулка в июле в шубе и валенках, потому что `Suspend` был выполнен в январе, а `Resume` – через полгода*).

Прерывание работы треда

Иногда тред полезно прервать до его нормального завершения. После прерывания тред удаляется из системы и не может быть начат снова.

Для прерывания потока до его нормального завершения служит метод `Abort`:

```
public void Abort ()  
public void Abort (object причина)
```

Этот метод генерирует в треде, для которого он был вызван, исключение `ThreadAbortException`. Это исключение может быть обработано в вызванном треде, для того, чтобы правильно завершить его работу.

Если исключение не обработано, тред «мгновенно умирает».

Повторный вызов метода `Abort` не вызывает никакого эффекта!

Прерывание работы треда (продолжение проекта alarm)


```
public class Alarm {  
...  
    public void Set() {  
        try {  
            while (Active) {  
                Thread.Sleep(2000);  
                DateTime currentTime = DateTime.Now;  
                if (currentTime.Hour == alarmTime.Hour &&  
                    currentTime.Minute == alarmTime.Minute) {  
                    AlarmEventArgs args = new AlarmEventArgs(currentTime);  
                    if (AlarmEvent != null)  
                        AlarmEvent(this, args);  
                    if (!Active)  
                        return;  
                    if (interval == 0)  
                        return;  
                    alarmTime = alarmTime.AddMinutes(interval);  
                }  
            }  
        }  
        catch (ThreadAbortException ex) {  
            Console.WriteLine("Будильник выключен, причина: " +  
                               ex.ExceptionState);  
        }  
    }  
...  
}
```

Прерывание работы треда (alarm, продолжение)

```
static Thread t;
```

```
public static void Main() {  
    DateTime wakeUpTime = DateTime.Now;  
    wakeUpTime = wakeUpTime.AddMinutes(2);  
    alarm = new Alarm(wakeUpTime, 1);  
    alarm.AlarmEvent += new AlarmEventHandler(wakeUp);  
    t = new Thread(alarm.Set);  
    t.Start();  
    Thread.Sleep(2000);  
    t.Abort("Я проснулся сам!");  
    t.Join();  
}
```

```
private static void wakeUp(object sender, AlarmEventArgs ev) {  
    ConsoleKeyInfo s;  
    Console.WriteLine("Звенит будильник! {0}",  
        ev.Time.ToLocalTime());  
    Console.WriteLine(@"Нажмите 1, если Вы проснулись  
        (другая клавиша - продолжаете спать...)");  
    s = Console.ReadKey();  
    if (s.KeyChar == '1') {  
        t.Abort("Меня разбудил будильник!");  
        t.Join();  
    }  
}
```



*Соответствующий пример
находится в файле alarm2.zip*

Отмена прерывания работы треда

Тред, работу которого собираются прервать, может отменить это прерывание, вызвав статический метод **ResetAbort**:

```
public class ThreadWork {  
    public static void DoWork() {  
        try {  
            ...  
        }  
        catch (ThreadAbortException e) {  
            Console.WriteLine("Exception message: {0}",  
                              e.get_Message());  
            Thread.ResetAbort();  
        }  
        Console.WriteLine("Thread - still alive...");  
        System.Threading.Thread.Sleep(1000);  
        Console.WriteLine("Thread - finished working.");  
    } //DoWork  
} //ThreadWork
```

Синхронизация работы тредов

При использовании нескольких тредов приходится координировать их действия. Процесс достижения такой координации называется **синхронизацией**.

Самой распространенной причиной применения синхронизации служит необходимость разделять среди двух или более тредов общий ресурс, который может быть одновременно доступен только одному треду.

Синхронизация требуется и в том случае, если один тред ожидает событие, вызываемое другим тредом.

В подобной ситуации требуются какие-то средства, позволяющие приостановить один из тредов до тех пор, пока не произойдет это событие. После этого ожидающий тред может возобновить свое выполнение.

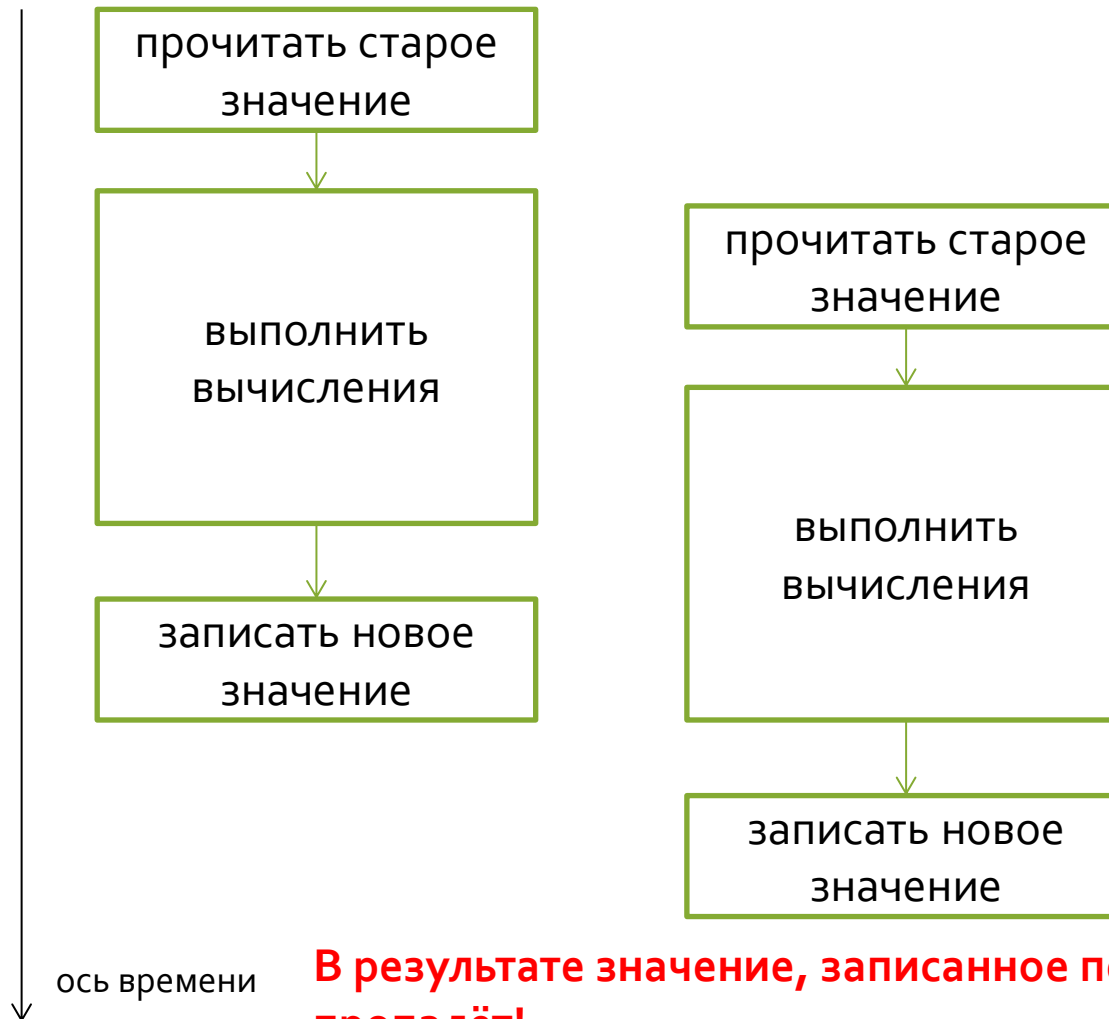
Пример, когда необходима синхронизация работы тредов

Пусть несколько параллельно выполняющихся тредов изменяют значение общей переменной. Изменение идёт по следующему принципу:



Пример, когда необходима синхронизация (продолжение)

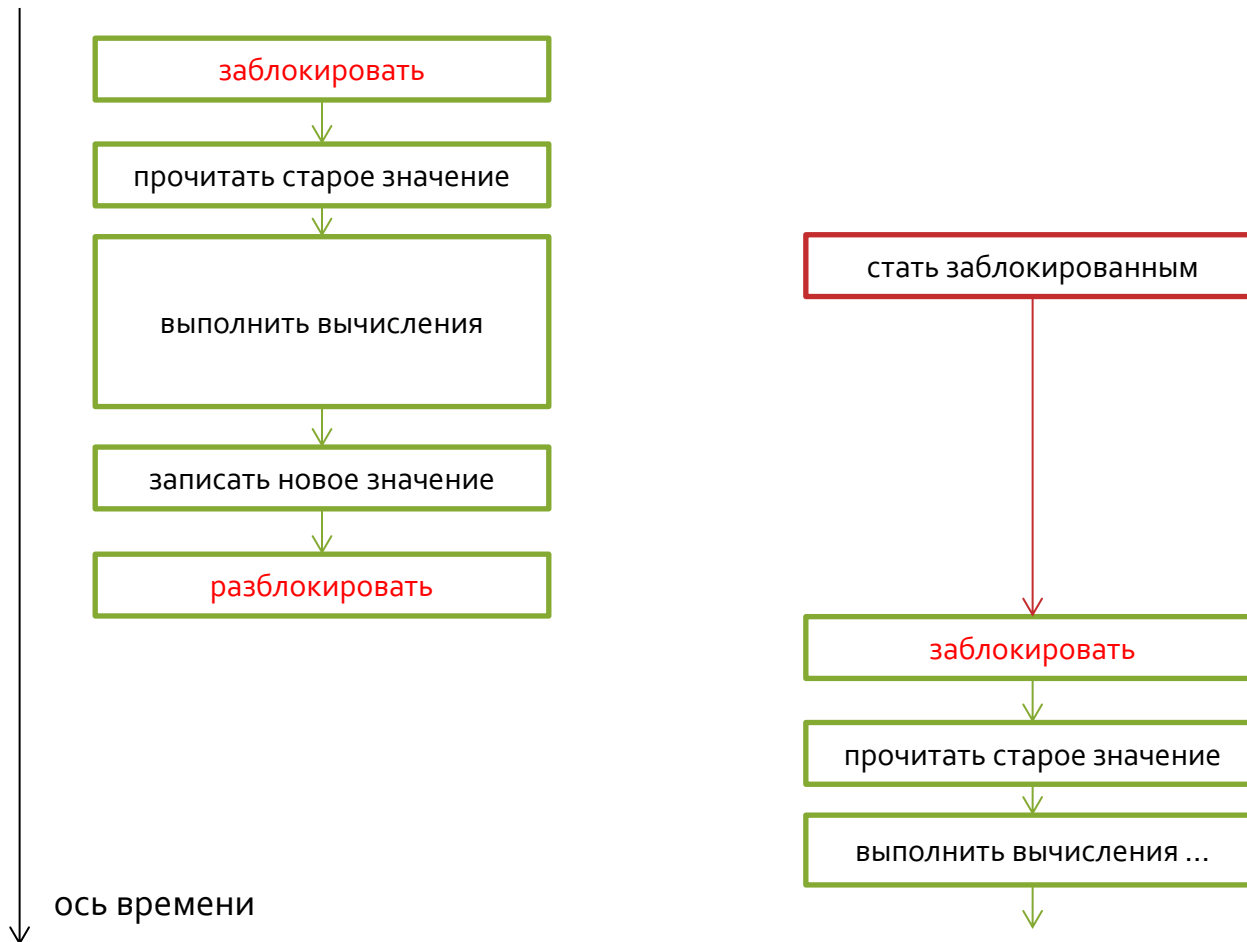
При работе нескольких тредов возможна такая ситуация:



В результате значение, записанное первым тредом, пропадёт!

Как это должно работать?

Блокировка запрещает одновременный доступ нескольких тредов к **критической секции**!



Принципы блокировки

1. Блокировка является атомарной операцией. Это означает, что другой тред не может прервать её выполнение.
2. Заблокированный тред может, тем не менее, получать сообщения от других тредов.
3. **Возможна взаимная блокировка (deadlock)!**

Пример неправильной работы тредов

Пусть необходимо просчитать суммарное количество цифр в нескольких текстовых файлах. Обработку каждого файла будем вести в отдельном треде.

```
public class Proceed {  
    string fileName;  
    public static int result;  
    public Thread thr;  
  
    public Proceed(string fileName, string thrName) {  
        this.fileName = fileName;  
        thr = new Thread(Run);  
        thr.Name = thrName;  
        thr.Start();  
    }  
}
```

Пример неправильной работы тредов (продолжение)

```
void Run() {  
    Console.WriteLine(thr.Name + " начат.");  
    StreamReader sr;  
    try {  
        sr = new StreamReader(new FileStream(fileName,  
            FileMode.Open));  
    }  
    catch (IOException ex) {  
        Console.WriteLine("Ошибка открытия файла " + fileName  
            + ": " + ex.Message);  
        return;  
    }  
    string s;  
    string digits = "0123456789";
```

Пример неправильной работы тредов (продолжение)

```
int current = result;
try {
    while ((s = sr.ReadLine()) != null) {
        for (int i = 0; i < s.Length; i++)
            if (digits.IndexOf(s[i]) >= 0) current++;
    }
}
catch (IOException ex) {
    Console.WriteLine("Ошибка при чтении файла "
        + fileName + ": " + ex.Message);
    return;
}
finally {
    sr.Close();
}
result = current;
Console.WriteLine(thr.Name + " завершен.");
} // Run()

} // Proceed
```

Коричневым цветом выделена критическая секция.

Пример неправильной работы тредов (окончание)

Головная программа:

```
class Program {  
    public static void Main() {  
        Proceed.result = 0;  
        Proceed p1 = new Proceed("input1.txt", "Thread1");  
        Proceed p2 = new Proceed("input2.txt", "Thread2");  
        Proceed p3 = new Proceed("input3.txt", "Thread3");  
        p1.thr.Join();  
        p2.thr.Join();  
        p3.thr.Join();  
        Console.WriteLine("Результат подсчёта: {0}",  
                           Proceed.result);  
    }  
}
```

Результат: выдаётся ответ только для того треда, который закончил работу последним!

Реализация блокировки

1. Создать объект (переменную типа `object`). Этот объект будет называться блокирующим объектом (БО).

```
public class Proceed {  
    string fileName;  
    public static int result;  
    public Thread thr;  
    public static object lo = new object()  
  
    public Proceed(string fileName, string thrName) {  
        this.fileName = fileName;  
        thr = new Thread(Run);  
        thr.Name = thrName;  
        thr.Start();  
    }  
}
```


Реализация блокировки

2. Обрамить критическую секцию блоком

```
lock(блокирующий_объект) {
```

```
    ...
```

```
}
```

```
lock (lo) {  
    int current = result;  
    // ...  
    result = current;  
}  
Console.WriteLine(thr.Name + " завершен.");  
} // Run()
```

После этого проблемы разрешаются!

Взаимоблокировка (deadlock)

Взаимоблокировка — это ситуация, в которой один тред ожидает определенных действий от другого треда, а другой тред, в свою очередь, ожидает чего-то от первого треда. В итоге оба треда приостанавливаются, ожидая друг друга, и ни один из них не выполняется.

Часто взаимоблокировка возникает тогда, когда два треда требуют доступа к двум ресурсам, но в разном порядке.

Для исключения взаимоблокировки требуется внимательное программирование и тщательное тестирование. В целом, если многопоточная программа периодически "зависает", то наиболее вероятной причиной этого является взаимоблокировка.

Пример взаимоблокировки «на пальцах»

Два человека желают выпить чаю. Один из них берёт кипяток, другой – пакет с заваркой. После чего первый говорит: «Дай мне заварку, я приготовлю себе чай и отдам тебе всё».

Второй отвечает: «Нет, сначала ты дай мне кипяток и подожди, пока я не приготовлю чай для себя».

В результате оба остаются ни с чем: первый хлебает «вар голы», а второй жуёт сухую траву.

Пример взаимоблокировки на C#

Тред 1

```
lock (a) {  
    // работа с ресурсом 1  
  
    lock (b) {  
        // работа с ресурсами 1 и 2  
  
    }  
}  
// освобождение ресурсов
```

Тред 2

```
lock (b) {  
    // работа с ресурсом 2  
  
    lock (a) {  
        // работа с ресурсами 1 и 2  
  
    }  
}  
// освобождение ресурсов
```

ось времени

Красной линией показаны моменты блокировки тредов

Взаимоблокировка (резюме)

Механизмы обхода и разрешения взаимоблокировки выходят за рамки данного курса и будут рассмотрены в курсе «Операционные системы».

Очередной недостаток примера с будильником

Оператор

```
Thread.Sleep(500000);
```

заставляет текущий тред «спать беспробудным сном»: даже после ответа «Меня разбудил будильник» сон продолжается!

Обход этой ситуации: необходимо, чтобы потоки посылали сообщения друг другу, даже если наступает блокировка!

Сообщение между тreads с помощью методов `Wait ()` и `Pulse ()`

Ключевое слово **lock** на самом деле служит в C# быстрым способом доступа к средствам синхронизации, определенным в классе **Monitor**. В этом классе определен также ряд методов для управления синхронизацией, в том числе статические методы `Wait ()` и `Pulse()`.

Эти методы вызываются только в треде, находящемся в критической секции.

Суть метода Wait ()

Когда выполнение треда требуется временно заблокировать (не выходя из критической секции), он вызывает метод

Wait (БО[, время_ожидания])

В итоге этот поток переходит в состояние ожидания, а в критическую секцию может войти другой тред.

Ожидание может быть прервано по двум причинам:

- тред, находящийся в этой же критической секции, вызывает метод Pulse;
- время ожидания истекло.

Суть метода Pulse ()

Метод Pulse «будит» первый из тредов, «заснувший» по методу Wait. Тред, вызвавший метод Pulse, сам переходит в заблокированное состояние.

Формат метода Pulse:

Pulse (БО) ;

Реализация проекта alarm с использованием взаимодействия тредов

- работа будильника и сон человека выполняются в одной критической секции. Для доступа к ней используется БО;
- после установки будильника тред «человек» входит в эту критическую секцию и вызывает метод `Wait`;
- срабатывание будильника происходит в этой же критической секции. Если будильник выключается, он вызывает метод `Pulse`, чтобы разблокировать тред «человек»;
- если время ожидания для треда «человек» истекло, этот тред принудительно блокирует тред «будильник» и продолжает работу в критической секции.

Изменения в исходных текстах проекта alarm

```
public class Alarm
{
    public object lo = new object();

    ...


    if (currentTime.Hour == alarmTime.Hour &&
        currentTime.Minute == alarmTime.Minute) {
        lock (lo) {
            AlarmEventArgs args =
                new AlarmEventArgs(currentTime);
            if (AlarmEvent != null)
                AlarmEvent(this, args);
            if (!Active) {
                Monitor.Pulse(lo);
                return;
            }
        }
    }

    ...

}
```

Изменения в исходных текстах проекта alarm (продолжение)

```
public static void Main() {  
    DateTime wakeUpTime = DateTime.Now;  
    wakeUpTime = wakeUpTime.AddMinutes(1);  
    alarm = new Alarm(wakeUpTime, 1);  
    alarm.AlarmEvent += new AlarmEventHandler(wakeUp);  
    Console.WriteLine("Завёл будильник, засыпаю...");  
    t = new Thread(alarm.Set);  
    lock (alarm.lo) {  
        t.Start();  
        Monitor.Wait(alarm.lo, 150000);  
    }  
    t.Abort("Я проснулся сам!");  
    t.Join();  
}
```



*Соответствующий пример
находится в файле alarm3.zip*