ЛЕКЦИЯ №12 TASK

Москва, 2019

Синхронизация

```
static int x=0;
   static void Main(string[] args)
   {
      for (int i = 0; i < 5; i++)
      {
         Thread myThread = new Thread(Count);
         myThread.Name = "Поток " + i.ToString();
         myThread.Start();
      }

      Console.ReadLine();
   }
   public static void Count()
   {
      x = 1;
      for (int i = 1; i < 9; i++)
      {
         Console.WriteLine("{0}: {1}", Thread.CurrentThread.Name, x);
         x++;
         Thread.Sleep(100);
      }
   }
```

Мы предполагаем, что метод выведет все значения x от 1 до 8 и так для каждого потока

Синхронизация

Lock - это ключевое слово C #; он предотвращает выполнение потоком того же блока кода, что и другой поток выполнения. Такой блок кода называется заблокированным кодом. Поэтому, если поток пытается ввести заблокированный код, он будет ждать, пока объект не будет выпущен.

Для блокировки с ключевым словом lock используется объект-заглушка, в данном случае это переменная locker.

```
public static void Count()
  {
    lock (locker)
    {
      x = 1;
      for (int i = 1; i < 9; i++)
      {
        Console.WriteLine("{0}: {1}", Thread.CurrentThread.Name, x);
        x++;
        Thread.Sleep(100);
      }
    }
  }
```

Dead lock

Dead Lock
В многопоточной среде может возникнуть мертвая блокировка; приложение зависает, потому что два или более
Потока ждут друг друга для завершения. Обычно это происходит, когда общий ресурс заблокирован одним
поток и другой поток ожидает доступа к нему.

Вот как приложение зависло.

1. Tsk1 получает блокировку «thislockA».
2. Tsk2 получает блокировку «thislockB».
3. Tsk1 пытается получить блокировку «thislockB», но она уже удерживается Tsk2 и, таким образом,
Tsk1 блокируется, пока «thislockB» не будет выпущен.
4. Tsk2 пытается получить блокировку «thislockA», но она удерживается Tsk1 и, таким образом, Tsk2.
блокирует, пока "thislockA" не будет выпущен.

На этом этапе оба потока заблокированы и никогда не завершатся.
Следовательно, приложение зависло.
Чтобы предотвратить зависание приложения, важно осторожно использовать оператор блокировки

Пользовательский интерфейс и потоки

Поток пользовательского интерфейса (пользовательского интерфейса) управляет жизненным циклом элементов управления пользовательского интерфейса (кнопок, текстового поля и т.д.).
обычно используется для обработки пользовательского ввода и реагирования на пользовательские события.

Пользовательский интерфейс и потоки

Поток пользовательского интерфейса (пользовательского интерфейса) управляет жизненным циклом элементов управления пользовательского интерфейса (кнопок, текстового поля и т.д.).
обычно используется для обработки пользовательского ввода и реагирования на пользовательские события.

Common language Runtime

Атаки, основанные на переполнении буфера, становятся практически невозможными благодаря жесткому контролю, осуществляемому средой выполнения .NET. Ошибки в коде, связанные с переполнениями или определениями типов, также не будут выполнены благодаря проверкам, которые осуществляет «виртуальная машина» CLR.

иным ресурсам, таким, как файлы, папки, реестр и так далее. Концепция безопасности, основанная на механизме ролей, использует два ключевых понятия: *аутентификация* и *авторизация*. Вопрос аутентификации – «кто вы такой», а вопрос авторизации – «позволено ли вам делать это».

В механизме ролей слово «вы» следует заменить объектом *principal*. Этот объект содержит в себе свойство *identity*, которое представляет идентификатор пользователя, от имени которого выполняется текущий поток. В главе 7 мы рассмотрим, как при помощи механизма ролей достигается безопасность в программах .NET.

CAS

Контроль выполнения кода (CAS) позволяет решать, какие действия коду разрешено выполнять, а какие – нет. CAS основывается на том, что вы можете назначить уровень доверия сборкам с кодом и ограничить разрешенные для этого кода операции, основываясь на заданных разрешениях. CAS тесно связан концепцией безопасности, основанной на *свидетельствах* (evidence). *Свидетельство* – это набор вспомогательной информации, которую CLR использует для принятия решений о том, к каким группам принадлежит код в сборке и, соответственно, какие действия этому коду разрешены. Свидетельства могут состоять, например, из информации о происхождении кода, из цифровой подписи на коде и так далее.

Политика безопасности – это набор правил, который конфигурируется администратором и используется средой CLR для принятия CAS-решений. Политика безопасности может быть задана на уровне предприятия,

CAS

Контроль выполнения кода (CAS) позволяет решать, какие действия коду разрешено выполнять, а какие – нет. CAS основывается на том, что вы можете назначить уровень доверия сборкам с кодом и ограничить разрешенные для этого кода операции, основываясь на заданных разрешениях. CAS тесно связан концепцией безопасности, основанной на *свидетельствах* (evidence). *Свидетельство* – это набор вспомогательной информации, которую CLR использует для принятия решений о том, к каким группам принадлежит код в сборке и, соответственно, какие действия этому коду разрешены. Свидетельства могут состоять, например, из информации о происхождении кода, из цифровой подписи на коде и так далее.

Политика безопасности – это набор правил, который конфигурируется администратором и используется средой CLR для принятия CAS-решений. Политика безопасности может быть задана на уровне предприятия,

MPI (C++, C)

OpenMP (многоядерное программирование), поддерживается на C++

Cuda (Nvidea графический ускоритель)

SIMD (Single Instruction Multiply Data)
SISD (Single Instruction Single Data)
MIMD (Multiply Instruction Multiply Data)

MQTT

Протокол передачи и получение данных

(IBM)

Используется в интернете вещей.