



C# Professional

Синхронизация доступа к
общему ресурсу



Проверить, идет ли запись

Меня хорошо видно && слышно?



Ставим "+", если все хорошо
"-", если есть проблемы



Тема вебинара

Синхронизация доступа к общему ресурсу

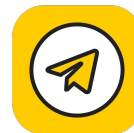


Михаил Дмитриев

Ведущий программист НИПК Электрон

Разрабатываю и поддерживаю приложения для работы с радиологическими комплексами

<https://t.me/sf321>



Правила вебинара



Активно
участвуем



Off-topic обсуждаем
в общем чате учебной группы
в telegram



Задаем вопрос
в чат или голосом



Вопросы вижу в чате,
могу ответить не сразу

Карта курса



Маршрут вебинара



Знакомство

Немного теории

Примитивы синхронизации

Примеры

Практика|ДЗ

Рефлексия

Цели вебинара

К концу занятия вы сможете

1. Обеспечивать потокобезопасный доступ к общим ресурсам в ваших проектах
2. Использовать примитивы синхронизации потоков в вашем коде
3. Рациональнее применять многопоточность в ваших проектах

Смысл

Зачем вам это уметь

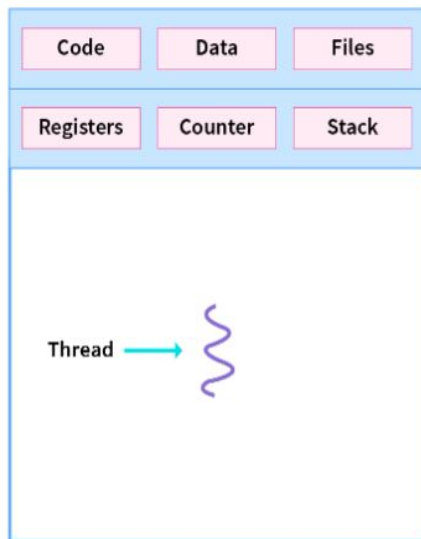
1. Разрабатывать и поддерживать многопоточные приложения
2. Эффективно применять инструменты, обеспечивающие синхронизацию доступа к общим ресурсам



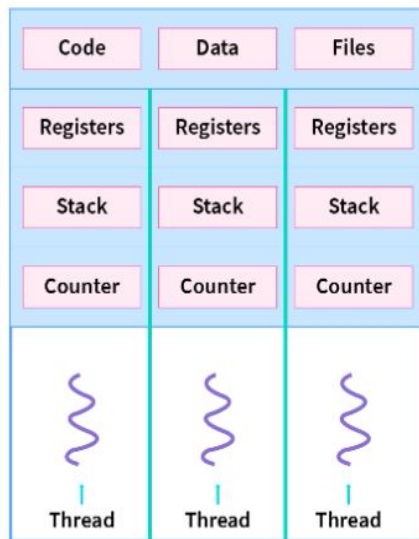
Тестирование

Процессы, потоки, синхронизация

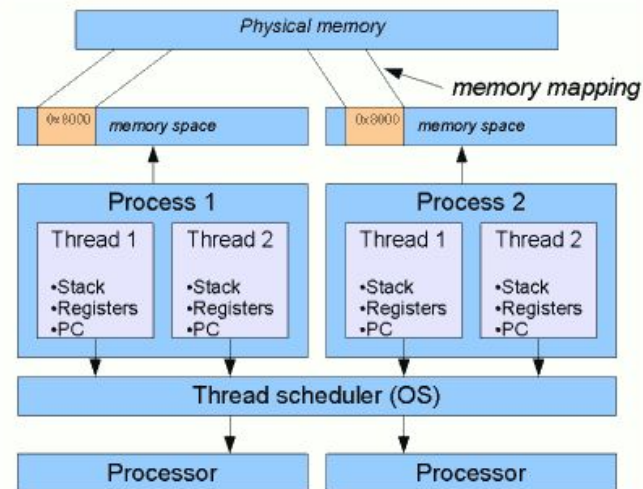
Процессы, потоки , разделяемые ресурсы



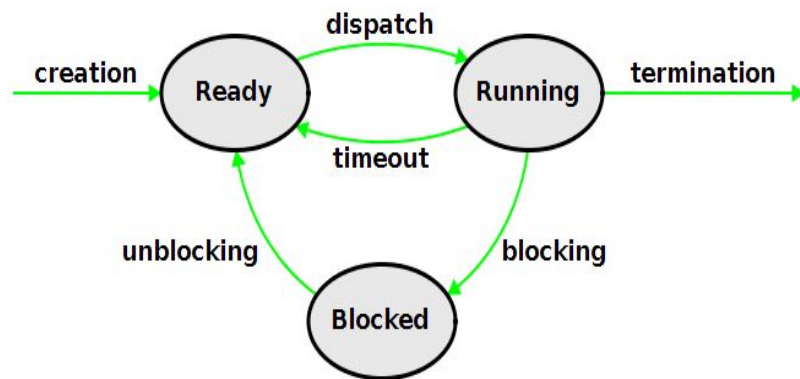
Single-threaded process



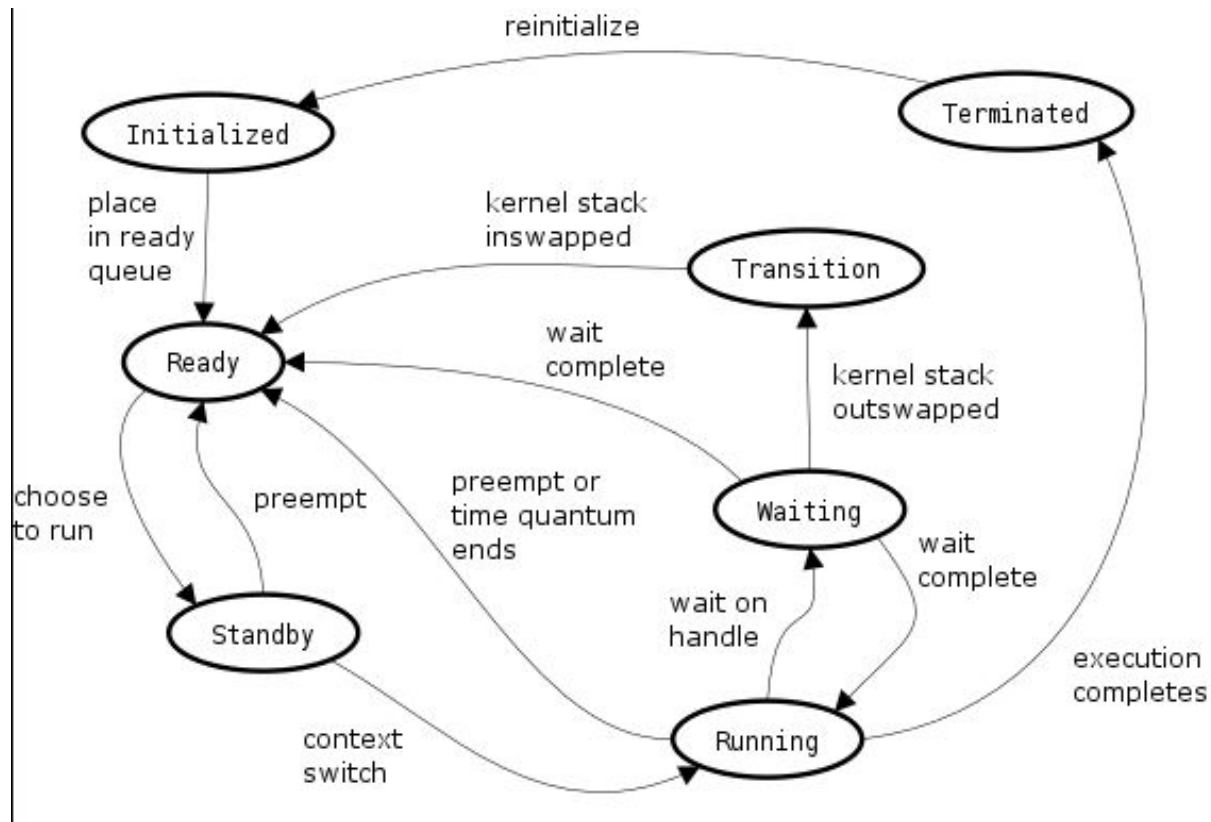
Multithreaded process



Состояния процесса



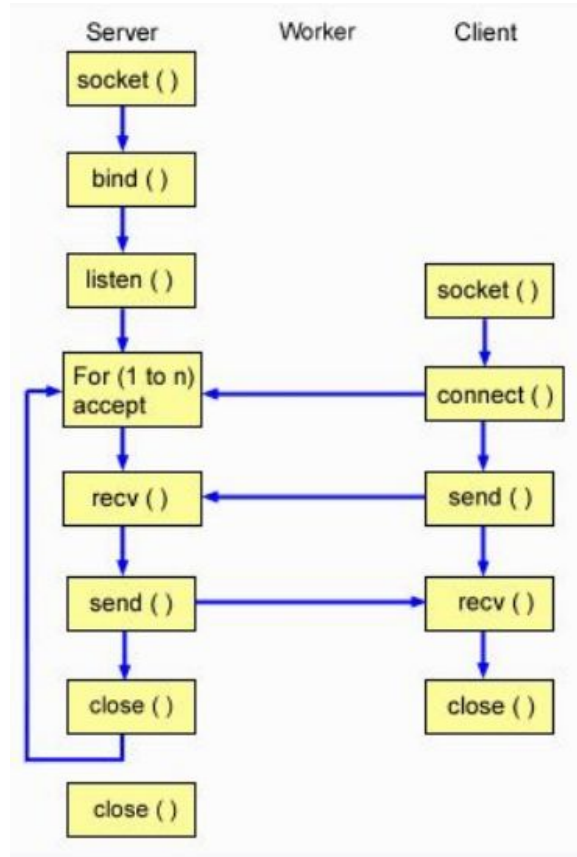
Состояния потока



Синхронизация



Однопоточный блокирующий сервер



Синхронизация - проблема

Начальное состояние:

$y = 0; x = 1;$

Параллельные операции

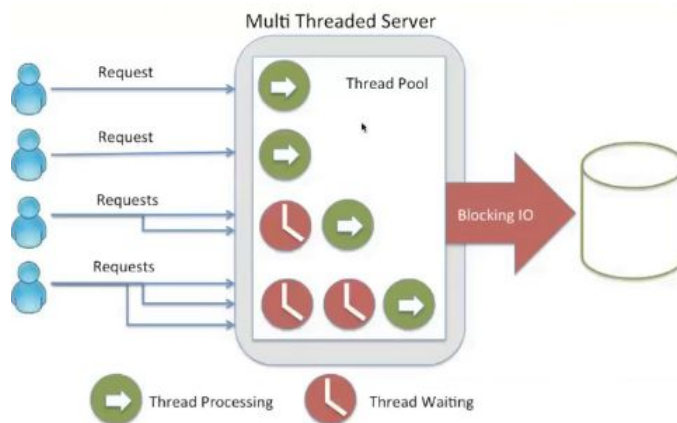
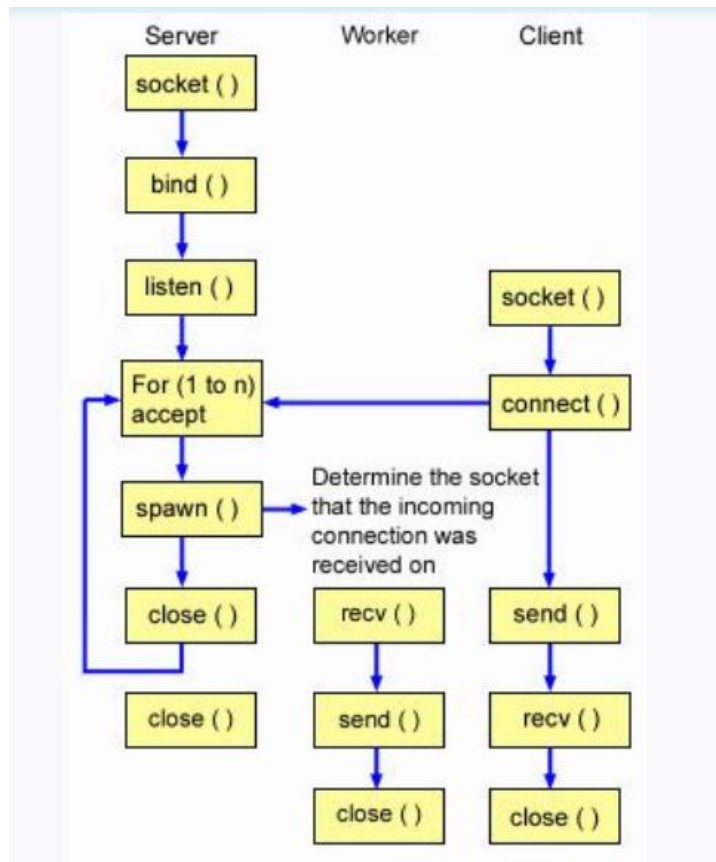
$\{ y = x + 1; \} \parallel \{ x = y + 2; \}$

Возможные конечные состояния:

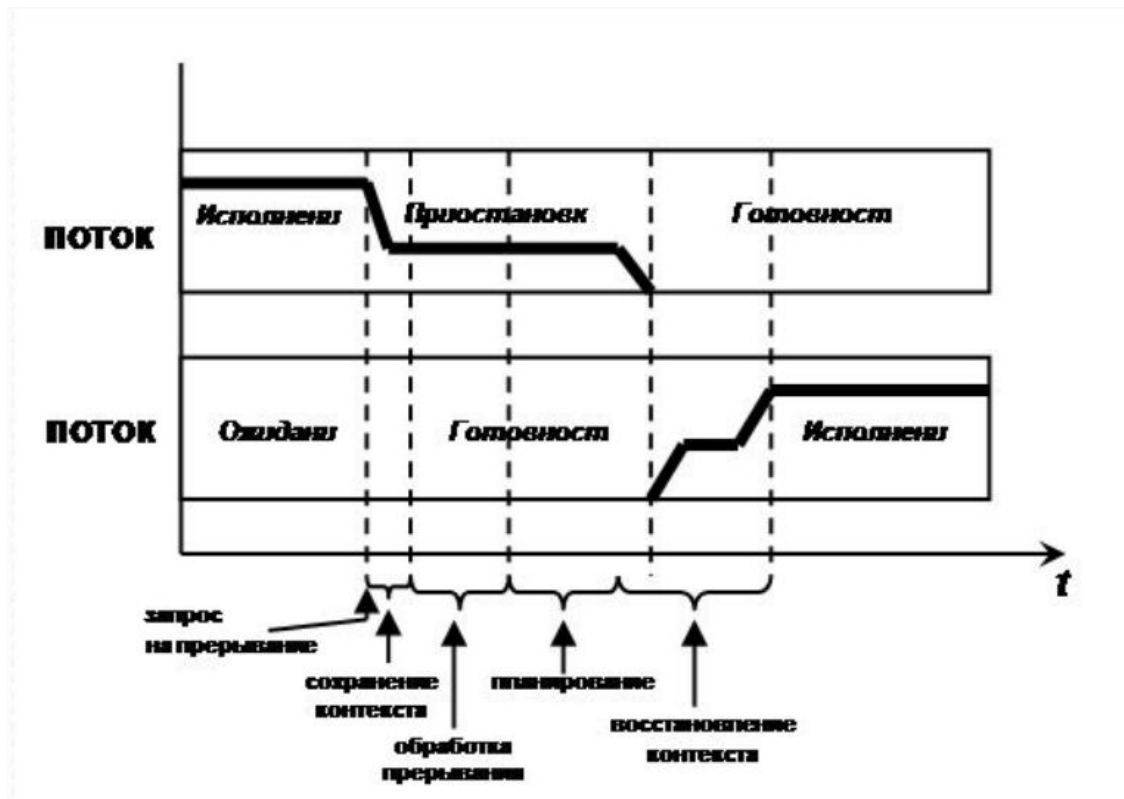
1. $y == 2; x == 2;$
2. $y == 3; x == 2;$
3. $y == 2; x == 4;$



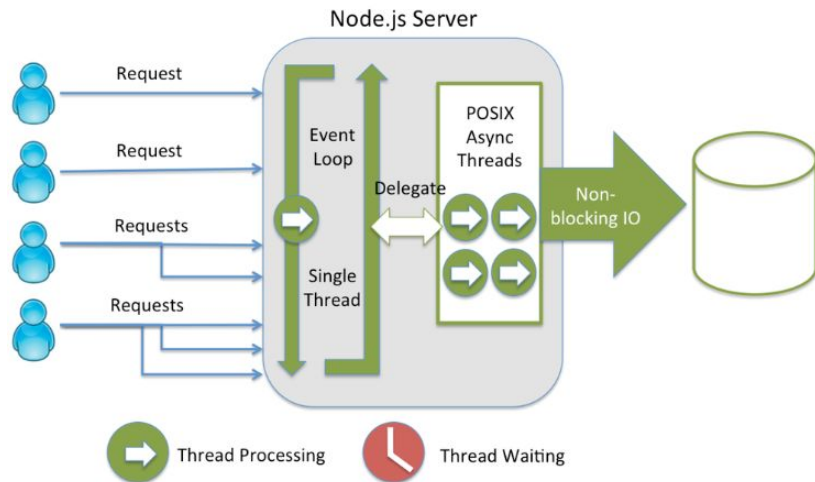
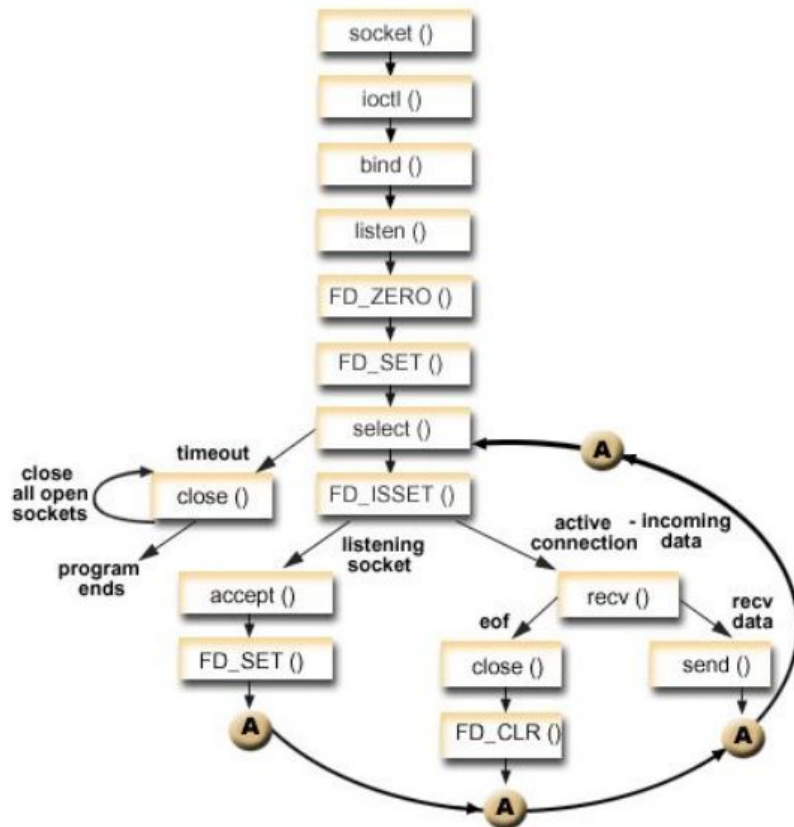
Многопоточный блокирующий сервер



Накладные расходы на потоки



Однопоточный неблокирующий сервер



Блокировка общих ресурсов для (1) потока

- Monitor/lock
- Mutex

Monitor - блокировка для одного потока

- примитив синхронизации, допускающий одновременное выполнение участка кода только одним потоком

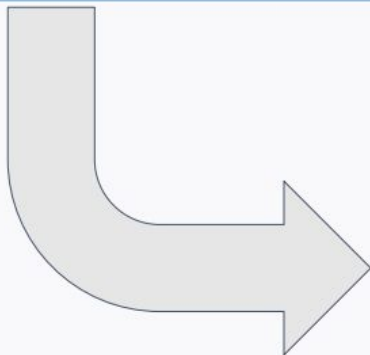
```
// Ставим блокировку (входим в критическую секцию)
Monitor.Enter(_lock);

// Используем общий ресурс
_dic[id] = _dic.ContainsKey(id)
    ? _dic[id] * 10
    : id;

// Выходим из критической секции
Monitor.Exit(_lock);
```

lock(object)

```
lock (x)
{
    // Your code...
}
```




```
object __lockObj = x;
bool __lockWasTaken = false;
try
{
    System.Threading.Monitor.Enter(__lockObj, ref __lockWasTaken);
    // Your code...
}
finally
{
    if (__lockWasTaken) System.Threading.Monitor.Exit(__lockObj);
}
```

SyncBlockIndex

```
int[] a = new int[5];  
for (int i = 0; i < 5; i++)  
{  
    a[i] = i;  
}
```

```
0x03022424      0 // SyncBlockIndex  
0x03022428 0x61B9C448 // *MethodTable  
0x0302242C      5 // a.Length  
0x03022430      0 // a[0]  
0x03022434      1 // a[1]  
0x03022438      2 // a[2]  
0x0302243C      3 // a[3]  
0x03022440      4 // a[4]
```

```
.load sos.dll  
!DumpArray 0x03022428  
Name:          System.Int32[]  
MethodTable:   61B9C448  
EECL:          6180C0D0  
Size:          32(0x20) bytes  
Array:         Rank 1, Number of elements 5, Type Int32  
Element Methodtable: 61B9C480  
[0] 03022430  
[1] 03022434  
[2] 03022438  
[3] 0302243C  
[4] 03022440
```



Double-checked locking

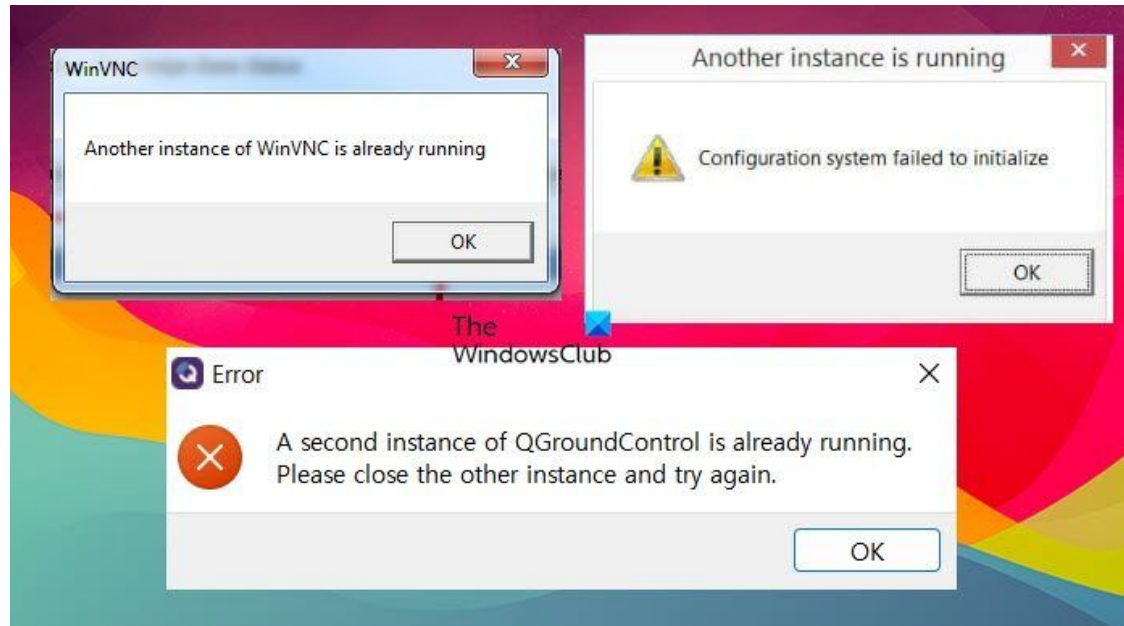
```
public sealed class Singleton
{
    private Singleton()
    {
        // инициализируем новый экземпляр объекта
    }

    private static volatile Singleton _instance;
    private static readonly object _lock = new object();

    public static Singleton GetInstance()
    {
        // если объект не создан
        if (_instance == null)
        {
            // входим в критическую секцию
            lock (_lock)
            {
                // проверяем, создал ли объект другим потоком
                if (_instance == null)
                {
                    // создаём
                    _instance = new Singleton();
                }
            }
        }
        return _instance;
    }
}
```


Mutex

- принцип работы такой же как у Monitor, но **он реализован на уровне ОС**, поэтому может быть использован для межпроцессной синхронизации



LIVE

Блокировка общих ресурсов для нескольких потоков (1+)

- Semaphore
- ReaderWriterLock

Semaphore (1 и более потоков)

примитив синхронизации, допускающий одновременное выполнение участка кода одним и более потоками

ReaderWriterLock

примитив синхронизации, позволяющий организовать множественный доступ на чтение и лишь один доступ на запись в единицу времени.

Используется, например для чтения из файла, запись в который происходит гораздо реже.

LIVE

Примитивы синхронизации, использующие Spin-wait

- SpinLock
- SemaphoreSlim
- ReaderWriterLockSlim

SpinLock

это тот же Monitor, только с периодом циклических проверок на выход из блокировки, схематично это выглядит так:

```
while (!Monitor.TryEnter(syncObject))  
{  
    :  
}
```


LIVE

Практика / ДЗ

Список материалов для изучения

1. [Рихтер Дж. "CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#"](#)
2. Грегори Р. Эндрюс "Основы многопоточного, параллельного и распределенного программирования"
3. Стивен Клири "Конкурентность в C#. Асинхронное, параллельное программирование"
4. <https://learn.microsoft.com/ru-ru/dotnet/standard/threading/the-managed-thread-pool>
5. <https://stackoverflow.com/questions/301160/what-are-the-differences-between-various-threading-synchronization-options-in-c>
6. <https://learn.microsoft.com/ru-ru/windows/win32/procthread/processes-and-threads>
7. <https://learn.microsoft.com/en-us/windows/win32/procthread/user-mode-scheduling>
8. <https://stackoverflow.com/questions/796217/what-is-the-difference-between-a-thread-and-a-fiber>
9. <https://www.c-sharpcorner.com/UploadFile/1d42da/threading-with-mutex/>
10. <https://slideplayer.com/slide/13105070/>
11. <https://learn.microsoft.com/en-us/windows/win32/api/winternl/ns-winternl-teb>
12. <https://learn.microsoft.com/en-us/windows/win32/api/winternl/ns-winternl-peb>
13. <https://habr.com/ru/companies/otus/articles/343566/>
14. <https://yonifedaali.blogspot.com/2017/03/sync-block-index-sbi-object-header-word.html>
15. <https://www.c-sharpcorner.com/UploadFile/8911c4/singleton-design-pattern-in-C-Sharp/>
16. <https://www.c-sharpcorner.com/UploadFile/1d42da/threading-with-mutex/>
17. <https://www.c-sharpcorner.com/UploadFile/1d42da/readerwriterlock-class-in-C-Sharp-threading/>
18. <https://referencesource.microsoft.com/#mscorlib/system/threading/SpinLock.cs>
19. <https://stackoverflow.com/questions/2416793/why-is-lock-much-slower-than-monitor-tryenter>
20. <https://learn.microsoft.com/en-us/dotnet/standard/threading/how-to-use-spinlock-for-low-level-synchronization>
21. [How to: Enable Thread-Tracking Mode in SpinLock - .NET | Microsoft Learn](#)
22. <https://learn.microsoft.com/en-us/dotnet/api/system.threading.threadstate?view=net-8.0>
23. <https://stackoverflow.com/questions/17593699/tcp-ip-solving-the-c10k-with-the-thread-per-client-approach>
24. <https://stackoverflow.com/questions/5983779/catch-exception-that-is-thrown-in-different-thread>
25. <https://stackoverflow.com/questions/65661244/why-do-locks-require-instances-in-c>
26. <https://learn.microsoft.com/en-us/visualstudio/debugger/get-started-debugging-multithreaded-apps?view=vs-2022&tabs=csharp>
27. <https://habr.com/ru/articles/447898/>
28. <https://gilllab.com/otus-demo/multi-threading-synchronization>



Вопросы?



Ставим "+",
если вопросы есть



Ставим "-",
если вопросов нет

Рефлексия

**Заполните, пожалуйста,
опрос о занятии
по ссылке в чате**

Спасибо за внимание!

Приходите на следующие вебинары



Михаил Дмитриев

Ведущий программист НИПК Электрон

Разрабатываю и поддерживаю приложения для работы с радиологическими комплексами

<https://t.me/sf321>

