Синхронизация потоков

Задание

- объявить переменную x = 0

- создать метод Print(): объявить цикл for(i=1, i<6,i++), в цикле увеличить x на 1, вывести имя потока, приостановить поток на 100

- в main запустить пять потоков с помощью цикла for(i=1, i<6,i++), каждый запускает Print(), каждому потоку дать имя thread.Name = $“Поток {i}“

Lock

Класс Lock служит для синхронизации доступа потоков к некоторой критической секции для предотвращения гонки за ресурсы между потоками.

```
int x = 0;  // некоторый общий ресурс
Lock lockObj = new(); // объект-заглушка для синхронизации доступа
// запускаем пять потоков
for (int i = 1; i < 6; i++){
   Thread myThread = new(Print);
   myThread.Name = $"Поток {i}";
   myThread.Start();
}
void Print(){
   lock (_lockObj){  // начало критической секции
      x = 1;
      for (int i = 1; i < 5; i++){
         Console.WriteLine($"{Thread.CurrentThread.Name}: {x}");
         x++;
         Thread.Sleep(100);
      }
   }  // завершение критической секции
}
```

Семафоры

Значение семафора может быть равно 0, что будет свидетельствовать об отсутствии сохраненных активизаций или иметь какое-нибудь положительное значение, если ожидается не менее одной активизации.

Как работает семафор

Используется две операции, *down* и *up*. Операция *down* выясняет, отличается ли значение семафора от 0. Если отличается, она уменьшает это значение на 1 (то есть использует одну сохраненную активизацию) и продолжает свою работу. Если значение равно 0, процесс приостанавливается, не завершая в этот раз операцию *down*. И проверка значения, и его изменение, и, возможно, приостановка процесса осуществляются как единое и неделимое атомарное действие.

Как работает семафор

Тем самым гарантируется, что с началом семафорной операции никакой другой процесс не может получить доступ к семафору до тех пор, пока операция не будет завершена или заблокирована.

Атомарная операция — операция, которая либо выполняется целиком, либо не выполняется вовсе.

C#

Семафоры являются еще одним инструментом, который предлагает нам платформа .NET для управления синхронизацией. Семафоры позволяют ограничить количество потоков, которые имеют доступ к определенным ресурсам. В .NET семафоры представлены классом Semaphore.

Semaphore

Конструкторы класса Semaphore

- Semaphore (int initialCount, int maximumCount)
- Semaphore (int initialCount, int maximumCount, string? name)
- Semaphore (int initialCount, int maximumCount, string? name, out bool createdNew)

Semaphore

Для работы с потоками класс Semaphore имеет два основных метода:

WaitOne(): ожидает получения свободного места в семафоре

Release(): освобождает место в семафоре

```
// запускаем пять потоков
for (int i = 1; i < 6; i++)
{
    Reader reader = new Reader(i);
}
class Reader
{
    // создаем семафор
    static Semaphore sem = new Semaphore(3, 3);
    Thread myThread;
    int count = 3;// счетчик чтения

    public Reader(int i)
    {
        myThread = new Thread(Read);
        myThread.Name = $"Читатель {i}";
        myThread.Start();
    }

    public void Read()
    {
        while (count > 0)
        {
            sem.WaitOne();  // ожидаем, когда освободиться место

            Console.WriteLine($"{Thread.CurrentThread.Name} входит в библиотеку");

            Console.WriteLine($"{Thread.CurrentThread.Name} читает");
            Thread.Sleep(1000);

            Console.WriteLine($"{Thread.CurrentThread.Name} покидает библиотеку");

            sem.Release();  // освобождаем место

            count--;
            Thread.Sleep(1000);
        }
    }
}
```

Мьютексы

Мьютекс — это переменная, которая может находиться в одном из двух состояний: в заблокированном или незаблокированном. Следовательно, для их представления нужен только один бит при этом нуль означает незаблокированное, а все остальные значения — заблокированное состояние.

Мьютексы C#

Создане мьютекса:

Mutex mutexObj = new Mutex()

Основную работу по синхронизации выполняют методы WaitOne() и ReleaseMutex().
Метод mutexObj.WaitOne() приостанавливает выполнение потока до тех пор, пока не будет получен мьютекс mutexObj.

```
int x = 0;
Mutex mutexObj = new();
// запускаем пять потоков
for (int i = 1; i < 6; i++){
   Thread myThread = new(Print);
   myThread.Name = $"Поток {i}";
   myThread.Start();
}
void Print(){
   mutexObj.WaitOne();    // приостанавливаем поток до получения мьютекса
   x = 1;
   for (int i = 1; i < 6; i++)   {
      Console.WriteLine($"{Thread.CurrentThread.Name}: {x}");
      x++;
      Thread.Sleep(100);
   }
   mutexObj.ReleaseMutex();   // освобождаем мьютекс
}
```