Откриване на максимален елемент (паралелно)

- Даден е масив с числа;
- Масива е с голям р-р;
- Да се реализира алгоритъм за откриване на максимален елемент в масива, чрез множество нишки.

```
static int FindMaxIterative(int[] arr, int l, int r)
{
    var max = arr[l];
    for (var i = l + 1; i < r; i++)
    {
        if (arr[i] > max)
            max = arr[i];
    }

    return max;
}
```

```
static int FindMaxDACThreads(int[] arr, int numThreads)
{
    var threads = new Thread[numThreads];
    var maxVals = new int[numThreads];

    for (int i = 0; i < numThreads; i++)
    {
        threads[i] = new Thread((o) =>
        {
            var n = (int)o;
            var part = arr.Length / numThreads;

            maxVals[n] = FindMaxIterative(
                arr, n * part, (n + 1) * part);
        });
        threads[i].Start(i);
    }

    for (int i = 0; i < numThreads; i++)
        threads[i].Join();

    return FindMaxIterative(maxVals, 0, maxVals.Length);
}
```

Задача „Производител - потребител“

- Дадена е опашка (с **крайна** големина);
- Нишка „производител“ добавя елемент в опашката;
- Нишка „потребител“ чете елемент от опашката;
- Да се реализира конзолно приложение с двете нишки, като те са синхронизирани (потребителя не трябва да чете от празна опашка, а производителя не трябва да записва в пълна опашка).

```
const int CAPACITY = 10;

static Queue<int> queue = new Queue<int>(CAPACITY);
static int count;

static Semaphore semExclude = new Semaphore(1, 1);
static Semaphore semEmpty = new Semaphore(0, CAPACITY);
static Semaphore semFull = new Semaphore(CAPACITY, CAPACITY);
```

```
public static void Producer()
{
    var r = new Random();

    while (true)
    {
        semFull.WaitOne();
        semExclude.WaitOne();

        Console.WriteLine("Producer: " + queue.Count());
        queue.Enqueue(r.Next());

        semExclude.Release();
        semEmpty.Release();
    }
}
```

```
public static void Consumer()
{
    var r = new Random();

    while (true)
    {
        semEmpty.WaitOne();
        semExclude.WaitOne();

        Console.WriteLine("Consumer: " + queue.Count());
        queue.Dequeue();

        semExclude.Release();
        semFull.Release();
    }
}
```

Задача „Читатели-писатели“

- Дадени са два типа процеси: „читатели“ – четящи съдържанието на паметта и „писатели“ – променящи съдържанието на паметта;
- Само един писател може да има достъп до паметта;
- Множество читатели могат да имат достъп до паметта;
- **1) Читател не трябва да чака, освен ако писател не пише в паметта;**
- **2) Ако писател чака за достъп читател не може да стартира;**

Задача „Обядващи философи“

- Пет философа седят на кръгла маса, на която има пет чинии със спагети и пет вилици.
- Когато философ мисли, той не яде;
- Когато иска да яде му трябват две вилици (около чинията му);
- Философ не може да вземе вилицата на друг философ, ако тя се ползва.

```
static Semaphore[] semForks = new Semaphore[]
{
    new Semaphore(1, 1),
    new Semaphore(1, 1),
    new Semaphore(1, 1),
    new Semaphore(1, 1),
    new Semaphore(1, 1)
};
public static void Philosopher(Object o)
{
    var r = new Random();
    var num = (int)o;

    while (true)
    {
        Console.WriteLine("Philosopher " + num + " start thinking.");
        Thread.Sleep(r.Next(1000));
        Console.WriteLine("Philosopher " + num + " stop thinking.");

        semForks[num].WaitOne();
        semForks[(num + 1) % 5].WaitOne();

        Console.WriteLine("Philosopher " + num + " start eating.");
        Thread.Sleep(r.Next(1000));
        Console.WriteLine("Philosopher " + num + " stop eating.");

        semForks[num].Release();
        semForks[(num + 1) % 5].Release();
    }
}
```

Задача „Спящият бръснар“

- В бръснарница има един бръснар;
- Ако няма клиенти бръснарят спи. Когато влезе клиент, той събужда бръснаря;
- Ако влезе клиент, докато бръснарят работи, той чака, ако има място, в противен случай си тръгва.