1.1 ./csharp/Platform.Hardware.Cpu/CacheLine.cs

```
using System;
using System.Runtime.InteropServices;

namespace Platform.Hardware.Cpu
{
    /// <summary>
    /// <para>Contains constants related to CPUs cache line.</para>
    /// <para>Содержит константы, относящиеся к строке кэша ЦП.</para>
    /// </summary>
    public static class CacheLine
    {
        /// <summary>
        /// <para>Gets the size of CPUs cache line in bytes.</para>
        /// <para>Получает размер строки кэша ЦП в байтах.</para>
        /// </summary>
        public static readonly int Size = GetSize();

        /// <summary>
        /// <para>
        /// Gets the size.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <exception cref="NotSupportedException">
        /// <para>Unrecognized OS platform.</para>
        /// <para></para>
        /// </exception>
        /// <returns>
        /// <para>The int</para>
        /// <para></para>
        /// </returns>
        private static int GetSize()
        {
            if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
            {
                return Windows.GetSize();
            }
            if (RuntimeInformation.IsOSPlatform(OSPlatform.Linux))
            {
                return Linux.GetSize();
            }
            if (RuntimeInformation.IsOSPlatform(OSPlatform.OSX))
            {
                return OSX.GetSize();
            }
            throw new NotSupportedException("Unrecognized OS platform.");
        }
    }
}
```

1.2 ./csharp/Platform.Hardware.Cpu/Linux.cs

```
using System;
using System.Runtime.InteropServices;

#pragma warning disable IDE1006 // Naming Styles

namespace Platform.Hardware.Cpu
{
    /// <summary>
    /// <para>
    /// Represents the linux.
    /// </para>
    /// <para></para>
    /// </summary>
    internal static class Linux
    {
        /// <summary>
        /// <para>
        /// Gets the size.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The int</para>
        /// <para></para>
        /// </returns>
```

```
        public static int GetSize() => (int)sysconf(_SC_LEVEL1_DCACHE_LINESIZE);

        /// <summary>
        /// <para>
        /// Sysconfs the name.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="name">
        /// <para>The name.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The int 64</para>
        /// <para></para>
        /// </returns>
        [DllImport("libc")]
        private static extern Int64 sysconf(Int32 name);

        /// <summary>
        /// <para>
        /// The sc level1 dcache linesize.
        /// </para>
        /// <para></para>
        /// </summary>
        private const Int32 _SC_LEVEL1_DCACHE_LINESIZE = 190;
    }
}
```

1.3 ./csharp/Platform.Hardware.Cpu/OSX.cs

```
using System;
using System.Runtime.InteropServices;

#pragma warning disable IDE1006 // Naming Styles

namespace Platform.Hardware.Cpu
{
    /// <summary>
    /// <para>
    /// Represents the osx.
    /// </para>
    /// <para></para>
    /// </summary>
    internal static class OSX
    {
        /// <summary>
        /// <para>
        /// Gets the size.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The int</para>
        /// <para></para>
        /// </returns>
        public static int GetSize()
        {
            var sizeOfLineSize = (IntPtr)IntPtr.Size;
            sysctlbyname("hw.cachelinesize", out IntPtr lineSize, ref sizeOfLineSize,
            ↪  IntPtr.Zero, IntPtr.Zero);
            return lineSize.ToInt32();
        }

        /// <summary>
        /// <para>
        /// Sysctlbynames the name.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="name">
        /// <para>The name.</para>
        /// <para></para>
        /// </param>
        /// <param name="oldp">
        /// <para>The oldp.</para>
        /// <para></para>
        /// </param>
        /// <param name="oldlenp">
```

```
48          /// <para>The oldlenp.</para>
49          /// <para></para>
50          /// </param>
51          /// <param name="newp">
52          /// <para>The newp.</para>
53          /// <para></para>
54          /// </param>
55          /// <param name="newlen">
56          /// <para>The newlen.</para>
57          /// <para></para>
58          /// </param>
59          /// <returns>
60          /// <para>The int 32</para>
61          /// <para></para>
62          /// </returns>
63          [DllImport("libc")]
64          private static extern Int32 sysctlbyname(string name, out IntPtr oldp, ref IntPtr
            ↪  oldlenp, IntPtr newp, IntPtr newlen);
65      }
66  }
```

1.4 ./csharp/Platform.Hardware.Cpu/Windows.cs

```
1   using System;
2   using System.Linq;
3   using System.Runtime.InteropServices;
4
5   #pragma warning disable 0649
6   #pragma warning disable IDE0044 // Add readonly modifier
7
8   namespace Platform.Hardware.Cpu
9   {
10      /// <summary>
11      /// <para>
12      /// Represents the windows.
13      /// </para>
14      /// <para></para>
15      /// </summary>
16      internal static class Windows
17      {
18          /// <summary>
19          /// <para>
20          /// Gets the size.
21          /// </para>
22          /// <para></para>
23          /// </summary>
24          /// <exception cref="InvalidOperationException">
25          /// <para>Could not retrieve the cache line size.</para>
26          /// <para></para>
27          /// </exception>
28          /// <returns>
29          /// <para>The int</para>
30          /// <para></para>
31          /// </returns>
32          public static int GetSize()
33          {
34              var info = ManagedGetLogicalProcessorInformation();
35              if (info == null)
36              {
37                  throw new InvalidOperationException("Could not retrieve the cache line size.");
38              }
39              return info.First(x => x.Relationship == LOGICAL_PROCESSOR_RELATIONSHIP.RelationCach
                ↪  e).ProcessorInformation.Cache.LineSize;
40          }
41
42          // http://stackoverflow.com/a/6972620/232574
43          /// <summary>
44          /// <para>
45          /// The processorcore.
46          /// </para>
47          /// <para></para>
48          /// </summary>
49          [StructLayout(LayoutKind.Sequential)]
50          struct PROCESSORCORE
51          {
52              /// <summary>
53              /// <para>
54              /// The flags.
55              /// </para>
```

```
            /// <para></para>
            /// </summary>
            public byte Flags;
        }

        /// <summary>
        /// <para>
        /// The numanode.
        /// </para>
        /// <para></para>
        /// </summary>
        [StructLayout(LayoutKind.Sequential)]
        struct NUMANODE
        {
            /// <summary>
            /// <para>
            /// The node number.
            /// </para>
            /// <para></para>
            /// </summary>
            public uint NodeNumber;
        }

        /// <summary>
        /// <para>
        /// The processor cache type enum.
        /// </para>
        /// <para></para>
        /// </summary>
        enum PROCESSOR_CACHE_TYPE
        {
            /// <summary>
            /// <para>
            /// The cache unified processor cache type.
            /// </para>
            /// <para></para>
            /// </summary>
            CacheUnified,
            /// <summary>
            /// <para>
            /// The cache instruction processor cache type.
            /// </para>
            /// <para></para>
            /// </summary>
            CacheInstruction,
            /// <summary>
            /// <para>
            /// The cache data processor cache type.
            /// </para>
            /// <para></para>
            /// </summary>
            CacheData,
            /// <summary>
            /// <para>
            /// The cache trace processor cache type.
            /// </para>
            /// <para></para>
            /// </summary>
            CacheTrace
        }

        /// <summary>
        /// <para>
        /// The cache descriptor.
        /// </para>
        /// <para></para>
        /// </summary>
        [StructLayout(LayoutKind.Sequential)]
        struct CACHE_DESCRIPTOR
        {
            /// <summary>
            /// <para>
            /// The level.
            /// </para>
            /// <para></para>
            /// </summary>
            public Byte Level;
            /// <summary>
            /// <para>
```

```
            /// The associativity.
            /// </para>
            /// <para></para>
            /// </summary>
            public Byte Associativity;
            /// <summary>
            /// <para>
            /// The line size.
            /// </para>
            /// <para></para>
            /// </summary>
            public UInt16 LineSize;
            /// <summary>
            /// <para>
            /// The size.
            /// </para>
            /// <para></para>
            /// </summary>
            public UInt32 Size;
            /// <summary>
            /// <para>
            /// The type.
            /// </para>
            /// <para></para>
            /// </summary>
            public PROCESSOR_CACHE_TYPE Type;
        }

        /// <summary>
        /// <para>
        /// The system logical processor information union.
        /// </para>
        /// <para></para>
        /// </summary>
        [StructLayout(LayoutKind.Explicit)]
        struct SYSTEM_LOGICAL_PROCESSOR_INFORMATION_UNION
        {
            /// <summary>
            /// <para>
            /// The processor core.
            /// </para>
            /// <para></para>
            /// </summary>
            [FieldOffset(0)]
            public PROCESSORCORE ProcessorCore;
            /// <summary>
            /// <para>
            /// The numa node.
            /// </para>
            /// <para></para>
            /// </summary>
            [FieldOffset(0)]
            public NUMANODE NumaNode;
            /// <summary>
            /// <para>
            /// The cache.
            /// </para>
            /// <para></para>
            /// </summary>
            [FieldOffset(0)]
            public CACHE_DESCRIPTOR Cache;
            /// <summary>
            /// <para>
            /// The reserved.
            /// </para>
            /// <para></para>
            /// </summary>
            [FieldOffset(0)]
            private UInt64 Reserved1;
            /// <summary>
            /// <para>
            /// The reserved.
            /// </para>
            /// <para></para>
            /// </summary>
            [FieldOffset(8)]
            private UInt64 Reserved2;
        }

```

```
        /// <summary>
        /// <para>
        /// The logical processor relationship enum.
        /// </para>
        /// <para></para>
        /// </summary>
        enum LOGICAL_PROCESSOR_RELATIONSHIP
        {
            /// <summary>
            /// <para>
            /// The relation processor core logical processor relationship.
            /// </para>
            /// <para></para>
            /// </summary>
            RelationProcessorCore,
            /// <summary>
            /// <para>
            /// The relation numa node logical processor relationship.
            /// </para>
            /// <para></para>
            /// </summary>
            RelationNumaNode,
            /// <summary>
            /// <para>
            /// The relation cache logical processor relationship.
            /// </para>
            /// <para></para>
            /// </summary>
            RelationCache,
            /// <summary>
            /// <para>
            /// The relation processor package logical processor relationship.
            /// </para>
            /// <para></para>
            /// </summary>
            RelationProcessorPackage,
            /// <summary>
            /// <para>
            /// The relation group logical processor relationship.
            /// </para>
            /// <para></para>
            /// </summary>
            RelationGroup,
            /// <summary>
            /// <para>
            /// The relation all logical processor relationship.
            /// </para>
            /// <para></para>
            /// </summary>
            RelationAll = 0xffff
        }

        /// <summary>
        /// <para>
        /// The system logical processor information.
        /// </para>
        /// <para></para>
        /// </summary>
        private struct SYSTEM_LOGICAL_PROCESSOR_INFORMATION
        {
            /// <summary>
            /// <para>
            /// The processor mask.
            /// </para>
            /// <para></para>
            /// </summary>
            public UIntPtr ProcessorMask;
            /// <summary>
            /// <para>
            /// The relationship.
            /// </para>
            /// <para></para>
            /// </summary>
            public LOGICAL_PROCESSOR_RELATIONSHIP Relationship;
            /// <summary>
            /// <para>
            /// The processor information.
            /// </para>
```

```
            /// <para></para>
            /// </summary>
            public SYSTEM_LOGICAL_PROCESSOR_INFORMATION_UNION ProcessorInformation;
        }

        /// <summary>
        /// <para>
        /// Determines whether get logical processor information.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="Buffer">
        /// <para>The buffer.</para>
        /// <para></para>
        /// </param>
        /// <param name="ReturnLength">
        /// <para>The return length.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [DllImport(@"kernel32.dll", SetLastError = true)]
        private static extern bool GetLogicalProcessorInformation(IntPtr Buffer, ref UInt32
        ↪  ReturnLength);

        /// <summary>
        /// <para>
        /// The error insufficient buffer.
        /// </para>
        /// <para></para>
        /// </summary>
        private const int ERROR_INSUFFICIENT_BUFFER = 122;

        /// <summary>
        /// <para>
        /// Manageds the get logical processor information.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The system logical processor information array</para>
        /// <para></para>
        /// </returns>
        private static SYSTEM_LOGICAL_PROCESSOR_INFORMATION[]
        ↪  ManagedGetLogicalProcessorInformation()
        {
            var ReturnLength = 0u;
            GetLogicalProcessorInformation(IntPtr.Zero, ref ReturnLength);
            if (Marshal.GetLastWin32Error() != ERROR_INSUFFICIENT_BUFFER)
            {
                return null;
            }
            var pointer = Marshal.AllocHGlobal((int)ReturnLength);
            try
            {
                if (GetLogicalProcessorInformation(pointer, ref ReturnLength))
                {
                    var size = Marshal.SizeOf<SYSTEM_LOGICAL_PROCESSOR_INFORMATION>();
                    var length = (int)ReturnLength / size;
                    var buffer = new SYSTEM_LOGICAL_PROCESSOR_INFORMATION[length];
                    var itemPointer = pointer;
                    for (int i = 0; i < length; i++)
                    {
                        buffer[i] = Marshal.PtrToStructure<SYSTEM_LOGICAL_PROCESSOR_INFORMATION>(
                        ↪  (itemPointer);
                        itemPointer += size;
                    }
                    return buffer;
                }
            }
            finally
            {
                Marshal.FreeHGlobal(pointer);
            }
            return null;
        }
```

```
    }
}
```

1.5 ./csharp/Platform.Hardware.Cpu.Tests/CacheLineTests.cs

```
using Xunit;

namespace Platform.Hardware.Cpu.Tests
{
    /// <summary>
    /// <para>
    /// Represents the tests.
    /// </para>
    /// <para></para>
    /// </summary>
    public static class Tests
    {
        /// <summary>
        /// <para>
        /// Tests that test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void Test() => Assert.NotEqual(0, CacheLine.Size);
    }
}
```

Index