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();
        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);
        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">
        /// <para>The oldlenp.</para>
        /// <para></para>
        /// </param>
        /// <param name="newp">
        /// <para>The newp.</para>
        /// <para></para>
        /// </param>
        /// <param name="newlen">
        /// <para>The newlen.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The int 32</para>
        /// <para></para>
        /// </returns>
        [DllImport("libc")]
        private static extern Int32 sysctlbyname(string name, out IntPtr oldp, ref IntPtr
        ↪  oldlenp, IntPtr newp, IntPtr newlen);
    }
}
```

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

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

#pragma warning disable 0649
#pragma warning disable IDE0044 // Add readonly modifier

namespace Platform.Hardware.Cpu
{
    /// <summary>
    /// <para>
    /// Represents the windows.
    /// </para>
    /// <para></para>
    /// </summary>
    internal static class Windows
    {
        /// <summary>
        /// <para>
        /// Gets the size.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <exception cref="InvalidOperationException">
        /// <para>Could not retrieve the cache line size.</para>
        /// <para></para>
        /// </exception>
        /// <returns>
        /// <para>The int</para>
        /// <para></para>
        /// </returns>
        public static int GetSize()
        {
            var info = ManagedGetLogicalProcessorInformation();
            if (info == null)
            {
                throw new InvalidOperationException("Could not retrieve the cache line size.");
            }
            return info.First(x => x.Relationship == LOGICAL_PROCESSOR_RELATIONSHIP.RelationCach
            ↪  e).ProcessorInformation.Cache.LineSize;
        }

        // http://stackoverflow.com/a/6972620/232574
        /// <summary>
        /// <para>
        /// The processorcore.
        /// </para>
        /// <para></para>
        /// </summary>
        [StructLayout(LayoutKind.Sequential)]
        struct PROCESSORCORE
        {
            /// <summary>
            /// <para>
            /// The flags.
            /// </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;
        }
```

```
78
79          /// <summary>
80          /// <para>
81          /// The processor cache type enum.
82          /// </para>
83          /// <para></para>
84          /// </summary>
85          enum PROCESSOR_CACHE_TYPE
86          {
87              /// <summary>
88              /// <para>
89              /// The cache unified processor cache type.
90              /// </para>
91              /// <para></para>
92              /// </summary>
93              CacheUnified,
94              /// <summary>
95              /// <para>
96              /// The cache instruction processor cache type.
97              /// </para>
98              /// <para></para>
99              /// </summary>
100             CacheInstruction,
101             /// <summary>
102             /// <para>
103             /// The cache data processor cache type.
104             /// </para>
105             /// <para></para>
106             /// </summary>
107             CacheData,
108             /// <summary>
109             /// <para>
110             /// The cache trace processor cache type.
111             /// </para>
112             /// <para></para>
113             /// </summary>
114             CacheTrace
115         }
116
117         /// <summary>
118         /// <para>
119         /// The cache descriptor.
120         /// </para>
121         /// <para></para>
122         /// </summary>
123         [StructLayout(LayoutKind.Sequential)]
124         struct CACHE_DESCRIPTOR
125         {
126             /// <summary>
127             /// <para>
128             /// The level.
129             /// </para>
130             /// <para></para>
131             /// </summary>
132             public Byte Level;
133             /// <summary>
134             /// <para>
135             /// The associativity.
136             /// </para>
137             /// <para></para>
138             /// </summary>
139             public Byte Associativity;
140             /// <summary>
141             /// <para>
142             /// The line size.
143             /// </para>
144             /// <para></para>
145             /// </summary>
146             public UInt16 LineSize;
147             /// <summary>
148             /// <para>
149             /// The size.
150             /// </para>
151             /// <para></para>
152             /// </summary>
153             public UInt32 Size;
154             /// <summary>
155             /// <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
        }
        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);
        private const int ERROR_INSUFFICIENT_BUFFER = 122;
```

```
311         private static SYSTEM_LOGICAL_PROCESSOR_INFORMATION[]
            ↪ ManagedGetLogicalProcessorInformation()
312         {
313             var ReturnLength = 0u;
314             GetLogicalProcessorInformation(IntPtr.Zero, ref ReturnLength);
315             if (Marshal.GetLastWin32Error() != ERROR_INSUFFICIENT_BUFFER)
316             {
317                 return null;
318             }
319             var pointer = Marshal.AllocHGlobal((int)ReturnLength);
320             try
321             {
322                 if (GetLogicalProcessorInformation(pointer, ref ReturnLength))
323                 {
324                     var size = Marshal.SizeOf<SYSTEM_LOGICAL_PROCESSOR_INFORMATION>();
325                     var length = (int)ReturnLength / size;
326                     var buffer = new SYSTEM_LOGICAL_PROCESSOR_INFORMATION[length];
327                     var itemPointer = pointer;
328                     for (int i = 0; i < length; i++)
329                     {
330                         buffer[i] = Marshal.PtrToStructure<SYSTEM_LOGICAL_PROCESSOR_INFORMATION>
                            ↪ (itemPointer);
331                         itemPointer += size;
332                     }
333                     return buffer;
334                 }
335             }
336             finally
337             {
338                 Marshal.FreeHGlobal(pointer);
339             }
340             return null;
341         }
342     }
343 }
```

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

```
1  using Xunit;
2
3  namespace Platform.Hardware.Cpu.Tests
4  {
5      public static class Tests
6      {
7          [Fact]
8          public static void Test() => Assert.NotEqual(0, CacheLine.Size);
9      }
10 }
```

Index