```
LinksPlatform's Platform.Hardware.Cpu Class Library
     ./Platform.Hardware.Cpu/CacheLine.cs
   using System;
   using System.Runtime.InteropServices;
   namespace Platform. Hardware. Cpu
4
5
        /// <summary>
6
       /// <para>Contains constants related to CPUs cache line.</para>
       /// <para>Содержит константы, относящиеся к строке кэша ЦП.</para>
       public static class CacheLine
{
       /// </summary>
10
11
            /// <summary>
12
            /// <para>Gets the size of CPUs cache line in bytes.</para>
13
            /// <para>Получает размер строки кэша ЦП в байтах.</para>
14
            /// </summary>
           public static readonly int Size = GetSize();
16
17
           private static int GetSize()
18
19
                if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
20
                {
                    return Windows.GetSize();
22
                }
23
                  (RuntimeInformation.IsOSPlatform(OSPlatform.Linux))
                if
                {
25
                    return Linux.GetSize();
26
                  (RuntimeInformation.IsOSPlatform(OSPlatform.OSX))
                {
29
                    return OSX.GetSize();
30
                }
31
                throw new NotSupportedException("Unrecognized OS platform.");
32
           }
33
       }
34
1.2
    ./Platform.Hardware.Cpu/Linux.cs
   using System;
   using System.Runtime.InteropServices;
   #pragma warning disable IDE1006 // Naming Styles
   namespace Platform. Hardware. Cpu
6
       internal static class Linux
           public static int GetSize() => (int)sysconf(_SC_LEVEL1_DCACHE_LINESIZE);
11
            [DllImport("libc")]
12
           private static extern Int64 sysconf(Int32 name);
13
           private const Int32 _SC_LEVEL1_DCACHE_LINESIZE = 190;
15
       }
16
   }
17
    ./Platform.Hardware.Cpu/OSX.cs
   using System;
   using System.Runtime.InteropServices;
2
   #pragma warning disable IDE1006 // Naming Styles
   namespace Platform. Hardware. Cpu
6
        internal static class OSX
9
            public static int GetSize()
10
11
                var sizeOfLineSize = (IntPtr)IntPtr.Size;
12
                sysctlbyname("hw.cachelinesize", out IntPtr lineSize, ref sizeOfLineSize,
                return lineSize.ToInt32();
15
16
            [DllImport("libc")]
17
           private static extern Int32 sysctlbyname(string name, out IntPtr oldp, ref IntPtr
18
            → oldlenp, IntPtr newp, IntPtr newlen);
```

```
19
   }
20
     ./Platform.Hardware.Cpu/Windows.cs
1.4
   using System;
using System.Linq;
2
   using System.Runtime.InteropServices;
   #pragma warning disable 0649
5
   #pragma warning disable IDE0044 // Add readonly modifier
   namespace Platform.Hardware.Cpu
8
9
        internal static class Windows
10
11
            public static int GetSize()
12
13
                 var info = ManagedGetLogicalProcessorInformation();
14
                 if (info == null)
15
                 {
16
                     throw new InvalidOperationException("Could not retrieve the cache line size.");
17
                 }
                 return info.First(x => x.Relationship == LOGICAL_PROCESSOR_RELATIONSHIP.RelationCach | 
19
                 → e).ProcessorInformation.Cache.LineSize;
            }
20
21
            // http://stackoverflow.com/a/6972620/232574
22
            [StructLayout(LayoutKind.Sequential)]
23
            struct PROCESSORCORE
^{24}
25
                 public byte Flags;
26
            }
27
            [StructLayout(LayoutKind.Sequential)]
29
30
            struct NUMANODE
            {
31
                 public uint NodeNumber;
32
            }
34
            enum PROCESSOR_CACHE_TYPE
35
            {
36
                 CacheUnified
37
38
                 CacheInstruction,
                 CacheData.
39
                 CacheTrace
            }
41
42
            [StructLayout(LayoutKind.Sequential)]
43
            struct CACHE_DESCRIPTOR
44
                 public Byte Level;
46
                 public Byte Associativity;
47
                 public UInt16 LineSize;
                 public UInt32 Size:
49
                 public PROCESSOR_CACHE_TYPE Type;
50
51
            [StructLayout(LayoutKind.Explicit)]
53
            struct SYSTEM_LOGICAL_PROCESSOR_INFORMATION_UNION
54
55
                 [FieldOffset(0)]
56
                 public PROCESSORCORE ProcessorCore;
57
                 [FieldOffset(0)]
58
                 public NUMANODE NumaNode;
59
                 [FieldOffset(0)]
60
                 public CACHE_DESCRIPTOR Cache;
                 [FieldOffset(0)]
62
                 private UInt64 Reserved1;
63
                 [FieldOffset(8)]
64
                 private UInt64 Reserved2;
65
            }
67
            enum LOGICAL_PROCESSOR_RELATIONSHIP
68
69
                 RelationProcessorCore,
70
                 RelationNumaNode,
7.1
                 RelationCache.
72
                 RelationProcessorPackage,
                 RelationGroup,
RelationAll = Oxffff
74
75
            }
76
```

```
private struct SYSTEM_LOGICAL_PROCESSOR_INFORMATION
78
79
                  public UIntPtr ProcessorMask;
public LOGICAL_PROCESSOR_RELATIONSHIP Relationship;
80
81
                  public SYSTEM_LOGICAL_PROCESSOR_INFORMATION_UNION ProcessorInformation;
82
83
84
             [DllImport(@"kernel32.dll", SetLastError = true)]
private static extern bool GetLogicalProcessorInformation(IntPtr Buffer, ref UInt32
85
86

→ ReturnLength);

             private const int ERROR_INSUFFICIENT_BUFFER = 122;
89
             private static SYSTEM_LOGICAL_PROCESSOR_INFORMATION[]
                 ManagedGetLogicalProcessorInformation()
91
                  var ReturnLength = Ou;
92
                  GetLogicalProcessorInformation(IntPtr.Zero, ref ReturnLength);
                  if (Marshal.GetLastWin32Error() != ERROR_INSUFFICIENT_BUFFER)
94
95
                      return null;
96
                  }
97
                  var pointer = Marshal.AllocHGlobal((int)ReturnLength);
99
                  try
100
                      if (GetLogicalProcessorInformation(pointer, ref ReturnLength))
101
102
                           var size = Marshal.SizeOf<SYSTEM_LOGICAL_PROCESSOR_INFORMATION>();
103
                           var length = (int)ReturnLength / size;
                           var buffer = new SYSTEM_LOGICAL_PROCESSOR_INFORMATION[length];
105
                           var itemPointer = pointer;
106
                           for (int i = 0; i < length; i++)</pre>
107
108
                               buffer[i] = Marshal.PtrToStructure<SYSTEM_LOGICAL_PROCESSOR_INFORMATION>,
109
                                    (itemPointer);
110
                               itemPointer += size;
111
                           return buffer;
112
113
114
                  finally
115
116
117
                      Marshal.FreeHGlobal(pointer);
118
                  return null;
119
             }
         }
121
122
1.5
     ./Platform.Hardware.Cpu.Tests/CacheLineTests.cs
   using Xunit;
    namespace Platform. Hardware. Cpu. Tests
 3
         public static class Tests
 5
             [Fact]
             public static void Test() => Assert.NotEqual(0, CacheLine.Size);
```

}

Index

- ./Platform.Hardware.Cpu.Tests/CacheLineTests.cs, 3
 ./Platform.Hardware.Cpu/CacheLine.cs, 1
 ./Platform.Hardware.Cpu/Linux.cs, 1
 ./Platform.Hardware.Cpu/OSX.cs, 1
 ./Platform.Hardware.Cpu/Windows.cs, 2