LinksPlatform's Platform.Hardware.Cpu Class Library

./Platform.Hardware.Cpu/CacheLine.cs

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

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Hardware.Cpu
{
    public static class CacheLine
    {
        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.");
        }
    }
}
```

./Platform.Hardware.Cpu/Linux.cs

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

#pragma warning disable IDE1006 // Naming Styles

namespace Platform.Hardware.Cpu
{
    internal static class Linux
    {
        public static int GetSize() => (int)sysconf(_SC_LEVEL1_DCACHE_LINESIZE);

        [DllImport("libc")]
        private static extern Int64 sysconf(Int32 name);

        private const Int32 _SC_LEVEL1_DCACHE_LINESIZE = 190;
    }
}
```

./Platform.Hardware.Cpu/OSX.cs

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

#pragma warning disable IDE1006 // Naming Styles

namespace Platform.Hardware.Cpu
{
    internal static class OSX
    {
        public static int GetSize()
        {
            var sizeOfLineSize = (IntPtr)IntPtr.Size;
            sysctlbyname("hw.cachelinesize", out IntPtr lineSize, ref sizeOfLineSize,
             ↪  IntPtr.Zero, IntPtr.Zero);
            return lineSize.ToInt32();
        }

        [DllImport("libc")]
        private static extern Int32 sysctlbyname(string name, out IntPtr oldp, ref IntPtr
         ↪  oldlenp, IntPtr newp, IntPtr newlen);
    }
}
```

./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
{
    internal static class Windows
    {
        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.RelationCache).ProcessorInformation.Cache.LineSize;
        }

        // http://stackoverflow.com/a/6972620/232574
        [StructLayout(LayoutKind.Sequential)]
        struct PROCESSORCORE
        {
            public byte Flags;
        }

        [StructLayout(LayoutKind.Sequential)]
        struct NUMANODE
        {
            public uint NodeNumber;
        }

        enum PROCESSOR_CACHE_TYPE
        {
            CacheUnified,
            CacheInstruction,
            CacheData,
            CacheTrace
        }

        [StructLayout(LayoutKind.Sequential)]
        struct CACHE_DESCRIPTOR
        {
            public Byte Level;
            public Byte Associativity;
            public UInt16 LineSize;
            public UInt32 Size;
            public PROCESSOR_CACHE_TYPE Type;
        }

        [StructLayout(LayoutKind.Explicit)]
        struct SYSTEM_LOGICAL_PROCESSOR_INFORMATION_UNION
        {
            [FieldOffset(0)]
            public PROCESSORCORE ProcessorCore;
            [FieldOffset(0)]
            public NUMANODE NumaNode;
            [FieldOffset(0)]
            public CACHE_DESCRIPTOR Cache;
            [FieldOffset(0)]
            private UInt64 Reserved1;
            [FieldOffset(8)]
            private UInt64 Reserved2;
        }

        enum LOGICAL_PROCESSOR_RELATIONSHIP
        {
            RelationProcessorCore,
            RelationNumaNode,
            RelationCache,
            RelationProcessorPackage,
            RelationGroup,
            RelationAll = 0xffff
        }

        private struct SYSTEM_LOGICAL_PROCESSOR_INFORMATION
        {
            public UIntPtr ProcessorMask;
            public LOGICAL_PROCESSOR_RELATIONSHIP Relationship;
            public SYSTEM_LOGICAL_PROCESSOR_INFORMATION_UNION ProcessorInformation;
        }
        [DllImport(@"kernel32.dll", SetLastError = true)]
        private static extern bool GetLogicalProcessorInformation(IntPtr Buffer, ref UInt32 ReturnLength);

        private const int ERROR_INSUFFICIENT_BUFFER = 122;

        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;
        }
    }
}
```

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

```
using Xunit;

namespace Platform.Hardware.Cpu.Tests
{
    public static class Tests
    {
        [Fact]
        public static void Test() => Assert.NotEqual(0, CacheLine.Size);
    }
}
```

Index