基于C#跨平台的测试测量初探

王 峰
上海简仪科技有限公司

摘 要：初步探索了测试测量跨平台的方案，论证了使用C＃作为跨平台的开发语言，在Windows中采用Microsoft Visual Studio作为开发平台，在Linux中采用Mono作为开发平台的方案的可行性。首先对Mono开发平台、C＃开发语言及锐视测控平台做了简要介绍，然后介绍了基于C＃和跨平台测试测量方案，通过实际例子说明了如何使用C＃做跨平台的开发，并实际使用简仪的DAQ产品实际演示了数据采集和频谱分析如何实现跨平台开发。最后，初步探索了Linux的实时性。

关键词：测试测量；跨平台；Linux；C＃；实时性

1 开发环境

1.1 当前测试测量的主要开发环境

当前，测试测量使用的主要开发环境是Labview，C/C++，C＃，Python等。其中Labview由于入门容易，适于小团队开发，占市场支配地位。但由于Labview是图形化的编程，在内存管理，代码维护升级，团队协作开发方面有较大的局限性；C/C++对开发人员的要求较高，并且没有系统的一套针对测试测量的行业的工具包可以使用，所以应用相对来说也比较少，只有底层或者实时性要求较高的应用里面才会使用；Python是一种解释型语言，能够快速生成程序的原型，在一些领域（如产线测试）也有较多的应用，但Python由于是解释执行，在执行效率方面并不太高；本文提出的跨平台的测试测量方案采用的C＃，在执行效率，代码维护，内存管理，跨平台，开发入门等方面都有突出的优势，但是C＃相对较新，测试测量业界应用并不广泛。特此本文论述了C＃在测试测量应用中跨平台开发的方法和特点。

1.2 跨平台的测试测量开发环境

1.2.1 跨平台的开发语言C#

C＃是一种安全的、稳定的、简单的、优雅的，由C和C++衍生出来的面向对象的编程语言。它在继承C和C++强大功能的同时去掉了一些它们的复杂特性，并综合了VB简单的可视化操作和C++的高运行效率，以其强大的操作能力、优雅的语法风格、创新的语言特性和便捷的面向组件编程的支持成为很多程序员的首选语言。C＃作为一种编程语言，早在2003年就成为ISO的标准之一，在Linux平台Mono的支持下，C＃可以方便的实现跨平台运行。

1.2.2 跨平台的开发环境Mono

Mono是一个由Xamarin公司（先前是Novell，最早为Ximian）所主持的自由开放源代码项目。它包含了一个C＃语言的编译器，一个CLR的运行时，和一组类库，并实现了 ADO NET和ASP NET。能够使得开发人员在Linux用C＃开发程序。该项目的目标是创建一系列符合标准ECMA（Ecma-334和Ecma-335)的.Net 工具，包括C ＃编译器和共同语言(common language)执行平台(Platform).与微软的.Net不同，Mono项目不仅可以运行于Windows系统内，还可以运行于Linux，FreeBSD，Unix，Mac OS X和Solaris等系统平台。

1.2.3 锐视开源测控软件

锐视测控平台是简仪科技自主开发的强大、易用、开源的测控系统开发工具软件。在锐视测控工具下，

开发者可以获得大量基于Microsoft® Visual Studio C#开发环境的成熟技术和强大功能，例如图形界面、文件I/O、算法类库、硬件驱动、仪器接口、网络访问等，还可以第一时间获得更高效的运行引擎更新，功能更强大的基础开发环境，以及最新的技术和设备支持。同时，借助 Microsoft® .NET 的跨平台支持和Mono开发环境，开发者还可以很容易地将测控系统移植到Microsoft® Windows ™之外的系统平台，例如 Linux、Mac OS X、iOS 和 Andriod 等。

2 跨平台的测试测量

2.1 为何需要跨平台

对于测试测量来说，有些应用需要能够保证长期稳定运行，而Windows长期稳定运行的特性并不明显，很多开发人员选择了开源开放的Linux。对于习惯了Windows开发便捷特性的工程师来说，如果要去学习一种新的开发工具或开发语言，无疑担心会花费大量的时间。但是有些工作和成果已经在Windows平台上积累起来，如果能够把我们已有的工作，很方便的移植到我们需要的另外一个平台，则会节约大量时间，提高我们已有成果的重复利用率。

2.2 跨平台的测试测量方案配置

跨平台的测试测量包含了如下几方面的内容：

√ 开发环境：Microsoft Visual Studio（In windows），Mono（In windows and Linux）

√ 操作系统：Windows，Linux

√ 开发语言：C#

√ 工具包：锐视开源测控平台的GUI，DSP等工具包

√ Linux实时补丁（可选）：RT-Preempt Patch

2.3 为何要选用Linux操作系统

◇ 长时间稳定运行

测试测量很多应用都需要进行长时间监测，这就要求整个系统能够长时间稳定运行，而Linux恰好具有较高的稳定性，能够保证长时间稳定可靠。

◇ 较好的实时性

测试测量中，有些应用要求系统具有一定的实时性，Linux加上RT-Preempt Patch正好具有较高的实时特性，满足大多数需要有一定实时特性的测试测量应用的需求。

◇ 安全性

信息安全是所有行业都需要关注的问题，针对Windows的病毒种类繁多，如近期的勒索病毒（Wanna Decryptor）肆虐，俨然是一场全球性互联网灾难，给广大电脑用户造成了巨大损失。Linux是开源的操作系统，依托文件系统，划分了简单明了的权限机制，安全性要明显优于Windows操作系统。

◇ 国产化的需求

当前国家对国产的要求越来越强烈，很多项目都要求国产化，尤其是军工行业，因此，测试测量系统的高国产化率，更加有利于市场的开辟。操作系统方面，Linux具有国产版本，如中标麒麟，红旗等Linux都是国产化的操作系统。

2.4 Linux运行C#程序

2.4.1 C#如何在Linux中运行

C#作为可以跨平台移植的一种开发语言，在Linux操作系统中，在Mono的基础上，可以很容易的运行起来，并且在Windows中用Visual Studio开发的程序，大多数情况下都可以直接到Linux中用Mono编译运行。虽然Mono中有GTK#可以用于设计界面，但对于习惯了Windows编程的工程师来说如果能够把已经有的Winform程序很方便的移植到Linux运行，这无疑会节省大量开发时间。接下来，我们就以一个简单的Winform程序为例来说明C#编写的程序如何在Linux中的使用。

2.4.2 Windows Form in Windows

我们首先在Visual Studio中创建一个Windows Form的Project（具体创建过程这里不详述）。然后，加入SeeSharpTools.JY.GUI.dll库的引用，并添加其中的EasyChart和EasyButton两个控件到界面上，并双击Button生成其单击的事件函数，添加如下代码：

```
var sinWaveform = new double[10000];
for (int i = 0; i < sinWaveform.Length; i++)
{
sinWaveform[i] = Math.Sin(2 *
```

Math.PI * 10 * i / sinWaveform.Length);

}

easyChart1.Plot(sinWaveform);

以上代码的作用是生成10000点，共10个周期的正弦波，然后生成并运行Project，得到的显示解码如图1所示。

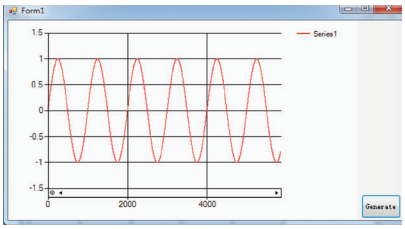

图1 Windows Form简单波形生成和显示范例（in Windows）

2.4.3 Windows Form in Linux

在2.4.2中创建的Windows Form项目如何在Linux中运行呢？答案是直接拷贝，把代码和我们引用的SeeSharpTools.JY.GUI.dll库一起拷贝到Linux，然后用MonoDevelop打开解决方案，直接编译并运行即可。在这个例程里面我们引用的第三方的库在Linux中都不用重新编译就可以直接使用。运行结果如图2所示。

通过以上的例子，我们可以看到C＃具有非常好的移植性，这样一来，我们就可以借助Windows中Visual Studio界面设计的强大功能，先将我们的应用程序界面设计好，然后直接到Linux去调试和布局我们的应用程序。这对于需要在Linux中开发窗体应用是非常方便的。

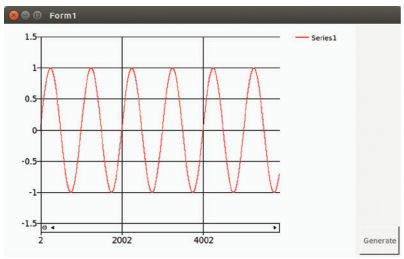

图2 Windows Form简单波形生成和显示范例（in Linux）

2.5 数据采集与频谱分析跨平台范例

下面我们用简仪的DAQ产品PXI62022为采集硬件，以软件定时单点采集的方式采集一个正弦信号，然后用锐视测控平台的频谱分析工具包计算采集到的信号的频谱，结构如图3所示。

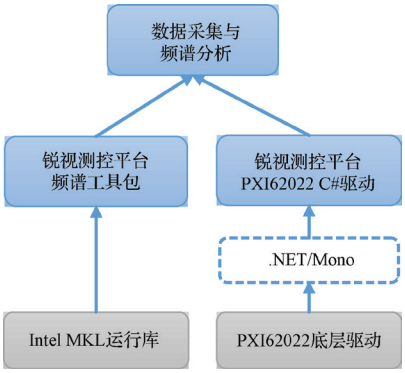

图3 数据采集分析跨平台范例结构

√ 准备工作

1）MKL运行库安装：先安装好MKL的Linux运行库，用于简仪锐视测控平台的频谱工具包分析调用

2）简仪PXI62022底层驱动安装：底层驱动是与硬件接口的通道，C＃控制硬件进行采集需要先安装好底层驱动

√ 界面设计

在Visual Studio中进行采集的界面设计，先新建一个Windows Form的项目，然后拖入两个EasyChart控件，一个LED Bright控件，一个EasyButton控件，及用于配置采集参数的几个控件，在Winform中的设计界面如图4所示。图中的两个EasyChart控件分别用于显示采集到的原始波形和分析得到的频谱结果，EasyButton控件用于启动采集，BoardNum用于选择硬件，Schedule则用于设置Linux中进程的调度策略，AcqInterval用于设置采集的间隔时间（单位为μs），AcqSamples用于设置需要采集的点数。

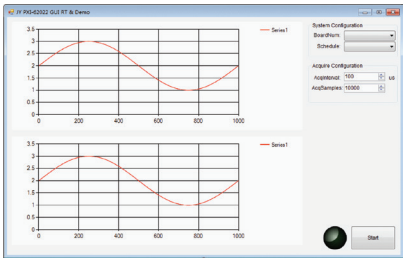

图4 采集界面设计

√ 采集代码

在FormLoad（窗体加载）的事件处理函数中添加频谱工具包初始化的代码，代码片段如下：

```
specTask = new SpectrumTask ();
specTask.InputDataType = InputDataType.Real;
specTask.WindowType = FFTWindowType.Hamming;

specTask.Average.Mode = SpectrumAverageMode.NoAveraging;
specTask.Average.WeightingType = SpectrumWeightingType.LinearMoving;
specTask.Average.Size = 1;

specTask.Unit.Type = SpectrumOutputUnit.dBV;
specTask.Unit.Impedance = 50;
specTask.Unit.IsPSD = false;
```

在Start按钮的单击事件处理函数中添加采集的代码，启动一个线程，线程中进行先关配置之后，开始以for循环的方式每隔100 μs采集一个点，采集部分代码片段如下：

```
for (int i = 0; i < samples; i++) {
    //wait Sleep 100us
    if ((ret = RTLib.JYClockNanosleep ((int)HrtClockType.CLOCK_MONOTONIC,
        (int)HrtTimerMode.TIMER_ABSTIME, ref ts)) < 0) {
        Console.WriteLine ("JYClockNanosleep ret = {0}", ret);
    }

    //do the stuff
    _aiTask.ReadSinglePoint(ref data);
    buf [i] = data [0];
    //calculate next shot
    ts.tv_nsec += interval; //interval为100us

    while (ts.tv_nsec >= NSEC_PER_SEC) {
        ts.tv_nsec -= NSEC_PER_SEC;
        ts.tv_sec++;
    }
}
```

采集部分代码之后，就是计算频谱和显示原始数据和频谱，代码片段如下：

```
this.BeginInvoke(_plotInvoke, easyChart1, buf, 0, interval / 1e9);
specTask.SampleRate = 1e9 / interval;
specTask.Output.NumberOfLines = (int)(numAcqSamples.Value / 2);
specTask.Commit ();
var spectrum = new double[specTask.Output.NumberOfLines];
specTask.GetSpectrum (buf, ref spectrum);
this.BeginInvoke(_plotInvoke, easyChart2, spectrum,
    specTask.SpectralInfomation.FreqStart, specTask.SpectralInfomation.FreqDelta);
```

√ Linux运行

直接将设计好界面的项目拷贝到Linux系统中，然后用Mono打开编译就可以直接运行了，运行后，设置好采集间隔和采集点数后，单击Start按钮，开始采集，完成后，在两个EasyChart控件上可以看到采集到的波形和分析的频谱，如图5所示。

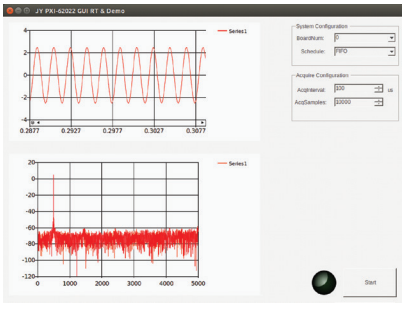

图5 数据采集与频谱分析跨平台范例运行结果

我们外接的信号是500 Hz的正弦，设置的采集间隔是100 μs（相当于10 KS/s的采样率），采集点数为10 K（即1 s的数据），采集后从时域波形可以看出计算用软件定时的方式采集到的波形也十分光滑，从频谱结果来看，我们采集到的信号正好是我们输入的500 Hz，并且从本地噪声来看，我们采集时间的准确度非常高。也就是说我们在Window中开发的程序，可以很方便的在Linux平台上正确执行并得到正确的结果，可见，我们提出的跨平台方案的易用性。

3 实时Linux操作系统初探

3.1 Linux的实时性

在Linux操作系统中，调度算法（其于最大吞吐量准则）、设备驱动、不可中断的系统调用、中断屏蔽以及虚拟内存的使用等因素，都会导致系统在时间上的不可预测性，决定了Linux操作系统不能处理硬实时任务，因此高优先级任务被唤醒到得以执行的时间并不能完全确定。Linux操作系统的种类繁多，目前用户较多的主要有Ubuntu，Open SUSE，Debian等等。接下来就以Ubuntu Linux为例说明硬实时的Linux及其实时性。

3.2 硬实时Linux操作系统

3.2.1 硬实时Linux操作系统的特点

硬实时Linux（RT Linux）在Linux内核与硬件之间增加了一个虚拟层（通常称作虚拟机），构筑了一个小的、时间上可预测的、与Linux内核分开的实时内核，使得在其中运行的实时进程满足硬实时性。并且RT Linux和Linux构成一个完备的整体，能够完成既包括实时部分又包括非实时部分的复杂任务。

3.2.2 RT-Preempt Patch补丁

Linux kernel在Spinlock、IRQ上下文方面无法抢占，因此高优先级任务被唤醒到得以执行的时间并不能完全确定。同时，Linux kernel本身也不处理优先级反转。RT-Preempt Patch是在Linux社区kernel的基础上，加上相关的补丁，以使得Linux满足硬实时的需求。

3.2.3 实时性测试——cyclictest

为了验证RT-Preempt Patch之后的实时性如何，我们采用开源的工具cyclictest来对标准Linux和RT-Preempt Patch的Linux的延迟做一个对比的测试。

cylictest的测试原理是：按照指定的优先级，期望循环时间间隔，测试次数等参数，开启指定数量的线程不断循环测试。测试结果包含了实际循环间隔与期望循环间隔的延迟的统计值，如最小值（Min），最大值（Max），平均值（Avg），还可以指定这个差值的直方图结果。

◇ 标准版Linux中的测试

参数设定：

优先级：80（1～90，最高1，最低90）

√ 期望循环时间：200 μs

√ 测试次数：9000万次

√ 调度策略：FIFO

√ 线程数：4

√ 直方图输出：使能，范围0～200 us

测试结果：

测试时间大约持续近5 h，得到四个并行线程的统计结果如下：

Min Latencies：00002 00002 00002 00002 //最小延迟

Avg Latencies：00006 00005 00006 00006 //平均延迟

Max Latencies：00822 00867 00905 00908 //最大延迟

Histogram Overflows：00012 00251 00411 05045 //超出最大值的次数（即100 μs）

图6所示为统计结果的直方图，从测试结果来看，平均时间延迟在5，6 μs左右，但最大延迟却达到900 μs左右，我们预设的循环时间是200 μs，超过预期值4.5倍，而且从图6的直方图来看，虽然绝大多数延迟都在20 μs以内，但还是有一部分延迟超过50 μs，这说明，标准版的Linux上执行的任务延迟时间有毫秒级的不确定性。

◇ RT-Preempt Patch Linux中的测试

参数设定（与标准版Linux一致）：

√ 优先级：80（1～90，最高1，最低90）

√ 期望循环时间：200 μs

√ 测试次数：3000万次

√ 调度策略：FIFO

√ 线程数：4

√ 直方图输出：使能，范围0～100 μs

统计4个线程循环结果如下：

Min Latencies：00002 00002 00003 00002 //最小延迟

Avg Latencies：00007 00006 00007 00007 //平均延迟

Max Latencies：00027 00021 00033 00025 //最大延迟

Histogram Overflows：00000 00000 00000 00000 //超出直方图最大值的次数（即100 μs）

图7所示为统计结果的直方图，从测试结果来看，平均延迟在6，7 μs左右，最大延迟只有33 μs，我们预设的循环时间是200 μs，最大值只超过预期值的15%，而且绝大多数延迟都分布在10%以内。

通过对测试结果的比较，我们可以得出RT-Preempt Patch Linux的延时确定性明显要优于标准版的Linux的延迟确定性，也就是说RT-Preempt Patch Linux实时性比标准版Linux的实时性要好很多。

但是要注意的是，还有一些因素可能降低Linux RT的时间确定性，比如.NET的Garbage Collection操作。

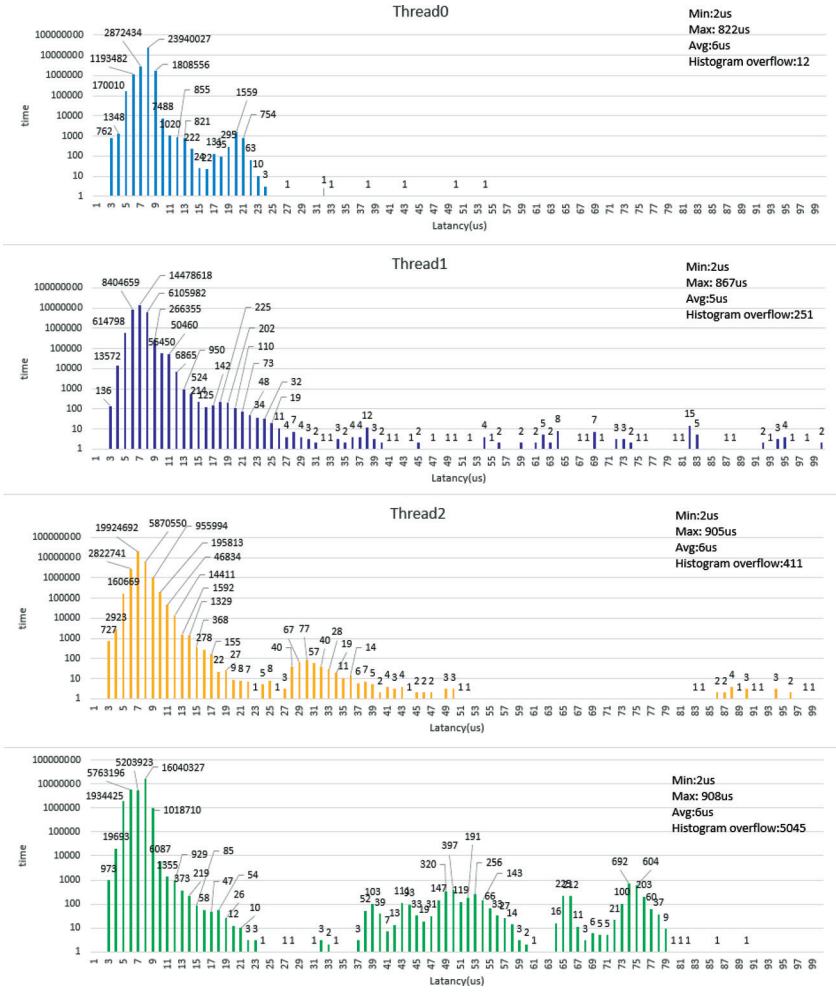

图6 标准版Linux实时性测试结果统计直方图

（下转第8页）