IJNRD.ORG ISSN : 2456-4184
INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD) | IJNRD.ORG
An International Open Access, Peer-reviewed, Refereed Journal

REVIEW ON C# AND THE .NET FRAMEWORK

Aniket Tote[1], Chetan Ingle[2], Pranav Mohod[3], Navaj Pasha Sheikh[4], Prof. Ankush Patil[5]

Dept. of Computer Engineering, Government College of Engineering, Yavatmal, Maharashtra, India

[1]anikettote2001@gmail.com
[2]inglechetan1199@gmail.com
[3]pranavmd09@gmail.com
[4]navajpashashaikh@gmail.com
[5]ankushpatil148@gmail.com

Abstract - **The new programming language C#, which is pronounced "C sharp," is part of Microsoft's integrated development environment, Visual Studio .NET. It is intended for the use with .NET Framework. Although the .NET Framework and C# are generally well-documented technologies, little emphasis has been paid to the platform's suitability for real-time systems. Although Microsoft does not state that C# and.NET are designed for real-time systems, many of the general-purpose characteristics of the platform—including type unsafe features, thread synchronisation, and overflow-sensitive arithmetic—apply to real-time systems. The appropriateness of C# and the .NET Framework for real-time systems will be further examined in this article.**

Keywords **— .NET; XML; C#; Web Service; CLR;**

I. INTRODUCTION

The Microsoft approach for creating software for the Web-oriented platform, which enables Internet connectivity through a variety of devices, is represented by .NET. The .NET platform was created to make it easier to develop applications that can be found in various environments. It features programming tools to create and integrate Web services based on XML, programmable services that allow users to access information from any device that is connected to the Internet, customizable applications, services, and devices, and a safe execution environment. The interplay between various distributed application components may indicate if a sufficient security mechanism is required.

The security mechanism used by the .NET platform is distinct from the operating systems' security framework. This security system, which takes its cues from Java, employs access control and component isolation based on user accounts. As a result, "unsecured" code is interpreted in a tightly constrained environment without access to vital resources. The security provided by.NET is based on the implementation of code access and role-based technologies that are based on a shared infrastructure made available by the common language runtime (CLR), which deals with automated memory management.

II. REAL TIME SYSTEMS

Real-time systems are categorised by practitioners as hard, firm, and soft.7 Systems that must operate in hard real-time include those where a single missed deadline might have grave consequences. One or more missed deadlines are tolerated by reliable real-time systems without having a major impact. Missed deadlines only cause performance deterioration in soft real-time systems. These conclusions are false, though. Schedulable and determinism are the two main issues that real-time systems must overcome, especially under stress.1 A system's capacity to meet all deadlines is referred to as schedule ability. Given the system's present state and a set of inputs, determinism enables observers to anticipate the system's subsequent state at any point.

III. REAL TIME C#

Numerous studies have been conducted on real-time Java systems. We will apply a synthesis of these research' methods to the.NET Framework and C# because they have characteristics in common with the Java Virtual Machine and Java. You should take note of the features of the underlying platform—most notably, Microsoft operating systems—when studying C# and.NET for real-time systems. Microsoft and Corel have collaborated on an opensource version of C# and the Common

Language Infrastructure to the FreeBSD operating system, which is similar to UNIX. At least two additional significant initiatives are also in progress to port.NET and C# to UNIX and Linux operating systems. the use of physical memory When running outside of the Common Language Runtime (CLR), unsafe code, which is supported by C#, allows pointers to refer to particular memory regions. For the garbage collector to be unable to change the position of objects that pointers reference, you must pin them (with the keyword "fixed"). Objects that are pinned to the ground are collected by the garbage collector but not moved. This functionality would tend to improve schedulability and also permits direct memory device access to write to certain memory locations—a requirement in embedded real-time systems. The usual C method navigates a data structure to choose an appropriate spot for allocation. Garbage collection. Memory management. The memory is allotted, and after that, the presence of the new item is updated in the data structure. . A stack is used by.NET to handle dynamic, heap-based memory. The location for the subsequent dynamic memory allocation is always pointed to by a system pointer called NextObjPtr. As a result, memory allocation in.NET is an easy, efficient procedure. Measurements from Microsoft back up this assertion. 2.NET provides a generational garbage collection method that aims to reduce thread blocking during mark and sweep. 2 Over time, this technique will be improved by Microsoft's garbage collector and other vendors'.NET implementations. It may even be tailored for specific purposes (such as real-time systems). The technique is predicated on the idea that just a subset of the heap has to be processed, as opposed to the full heap. However, even with. generations, NET's it is difficult to determine the collection's precise timetable or to provide a price guarantee for each collection. The biggest challenge for C#-based hard real-time systems may be this nondeterministic behaviour. Scheduleable threads and determinism Many of the thread management structures that real-time systems, especially difficult ones, sometimes require are not supported by C# and the.NET framework. 7 For instance, the Framework does not permit the establishment of threads with the promise that they would finish by a specific time when started. Many thread synchronisation techniques are supported by C#, but none of them are this precise. Thread management building blocks have been greatly enhanced in Windows CE 3.0. It could offer a pretty robust thread management framework if C# and the.NET Compact Framework are properly able to take advantage of it. But the.NET Framework's current thread priority enumerations are essentially inadequate for real-time applications. Only the highest, lowest, highest, and normal values are present. Compare this to Windows CE 3.0, which was created with 256 thread priorities expressly for real-time systems.

C# supports an array of thread synchronization constructs:

- Lock. A lock and a vital sector are synonymous in meaning (a code segment guaranteeing entry to itself by only one thread at a time).
- Monitor. Lock is a shortened abbreviation for the class type of monitor.
- Mutex. A mutex and a lock are semantically equal, with the latter having the added ability to operate across processes spaces. The performance cost of mutexes is a drawback.
- Interlock. You may increment and decrement numeric in a thread-safe way by using interlock, a collection of overloaded static methods.

Predictability of execution time, or the schedulability we have been talking about, is related to thread management. Runtime code analysis, which proactively estimates the expected runtime of crucial code segments, should be incorporated into real-time systems design. 4 The platform throws an exception if it determines that the code won't run within a certain amount of time. This is preferable to reactively detecting a missed deadline since it enables the programme to respond correctly (for instance, by completing an inaccurate computation and giving less accurate results, which are frequently better than no results). 4 The majority of real-time developer demands aren't met by the .NET Framework. However, developers will be able to get around these real-time deadline restrictions if the.NET Compact Framework ultimately supports 256 priority levels and if the operating system takes care of.NET thread priority inversion as it does for unmanaged threads. Additionally, .NET threads offer synchronisation techniques, are type safe, and are simple to write (instead of raw function pointers, .NET uses delegates). As a potential choice for both soft and firm real-time systems, C# may appeal to architects. Inversion of priorities Priority inversion indicates that lower-priority threads may block higher-priority threads using synchronisation mechanisms (such as mutexes), which is a condition that should be avoided. Priority inversion may result in deadlock or starvation, in which higher-priority threads receive far less CPU time than lower-priority equivalents. These are both typical real-time systems issues. 2 Many operating systems, including the majority of Windows versions (such as CE), temporarily elevate the priority of lower priority threads in certain circumstances to combat priority inversion, releasing synchronisation objects for faster execution of higher-priority threads. The priority ceiling protocol is the name of this resolution. The CLR should gain from Windows priority

inheritance capabilities since the.NET platform, and more especially the CLR, depends on the underlying operating system for its thread management algorithms. Thread inheritance, however, did not seem to function well in the testing done for this post. The test system was an 800-MHz Dell Inspiron 8000 with 523 Mbytes of physical memory, running Windows 2000 SP1 and.NET v1.0.3705 (version one). These poor outcomes persisted across many testing philosophies. The lock statement in C# was the only statement tested. Win32 critical portions are identical to this claim. 7 An intraprocess synchronisation technique known as a crucial section (and lock) serialises threads through defined portions. A thread with normal priority was able to enter a lock before any other threads thanks to the tests' multiple thread generation. A group of threads with lesser priorities blocked the lock of this thread with normal priority in the meanwhile. The main thread of the processes just waited for all worker threads to finish. These tests demonstrated that, as opposed to an inverse connection, the blocked length of the lower priority threads varied directly with their priority—the intended outcome provided the priority inheritance protocol was correctly used. The running thread's priority should have temporarily boosted by Windows to accelerate it through the crucial area when we raised the priority of the stalled threads in later testing. Instead, because blocking threads were assigned a higher priority, the thread that had the lock held it for a longer period of time. By nearly definition, this is the priority inversion problem. Priority inheritance is how Windows 2000, Windows CE 2.1, Windows CE 3.0, Windows XP, and other Microsoft operating systems manage priority inversion. According to Jeffrey Richter, a contributor for MSDN magazine, the CLR uses the thread management mechanism of the underlying operating system, thus priority inheritance ought to operate similarly in theory. 5 However, these trials showed that priority inheritance was not readily apparent, at least for the.NET lock mechanism.

IV. TIMERS

The functionality of timers in C# is similar to that of existing Win32 timers. When timers are created, they are given instructions on how long to wait in milliseconds before their initial invocation as well as an interval (also in milliseconds) that specifies how long should pass before the next invocation. Because of their machine-dependent precision and lack of assurance, these clocks are less appropriate for real-time applications.

V. HARDWARE CHARACTERISTICS DISCOVERY

Many of these flaws do not exist in C# or on the .NET platform. For instance, PInvoke, an underlying system application programming interface invocation method, is supported by C# via the.NET Framework's System. Runtime. InteropServices package. The CPU characteristics, memory utilisation, and other information are provided by the Win32 API at both the machine and process levels. Additionally, C# programmers may access the system's performance counters, which include hundreds of system-level measures, via the System. Diagnostics package. CLR Data, CLR Memory, CLR Networking, ASP.NET counters, and a collection of JIT (just in time) compiler counters are just a few of the new.NET-specific performance objects that Microsoft has added. They are all accessible programmatically and through the Performance Manager user interface (Perfmon.exe).

VI. TYPE SAFETY

The.NET platform's type safety feature is essential; it proactively detects software flaws at runtime as well as at compile time, which is helpful for real-time applications. Allocated objects are always accessed in.NET in ways that are compatible. As a result, the CLR will notice and stop attempts to access a method input parameter that is defined to accept a 4-byte value. Similar to the last example, if an object takes up 10 bytes in memory, the programme cannot force it into a format that enables reading of more than 10 bytes. Only well-known destinations will get the execution flow transfer (namely, method entry points). It is impossible to create an arbitrary reference to a memory region and start code there.

VII. EXCEPTION HANDLING

Exception handling is organised in the.NET Framework. The "try, throw, catch, finally" methodology is the foundation for exception handling in several popular object-oriented programming languages. 2 A developer can customise the error messages based on a particular application since exception handling in.NET is flexible and cross-language compatible. Exception objects created by thrown exceptions are captured in catch blocks. Therefore, an exception in C# error handling is actually an object. Additionally, this strategy applies to all languages that target the.NET platform, such as VB.NET, C#, and Managed C++. There are several error-handling techniques in the present Win32/COM (Component Object Architecture) programming model, including HRESULTs (COM), the implementation of specific interfaces to receive a more understandable error message (COM),5, and calls to GetLastError (Win32). One understandable, structured exception is supported by .NET. All languages use the same well-known, structured exception handling method that .NET allows.

VIII. Performance Tests

Although it is not the core of real-time systems, performance is a crucial component. 3 Although speed does not ensure a process will always succeed, it makes it simpler to fulfil deadlines when it runs more quickly. We ran two tests contrasting the performance of C# and C. All tests were created using C++.NET version 55603-652-0000007-18846, while C#.NET was created and run on the.NET platform at version 1.0.3705. These versions correspond to the first non-beta general release of.NET. Every build was an improved release build. The tests were performed on a Windows 2000 Professional, SP1 computer operating at 800 MHz with 523 Mbytes of physical RAM. In the initial test, 10 billion floating-point operations were performed. The whole runtime of the wall clock is displayed in Figure 2. The second test focused on memory use. In both C and C#, we created and released a linked list with 5,000 nodes 24 times (48 total). The size of the nodes on every other list increased. Therefore, each node in the first 5,000 node sets comprised a straightforward numeric value and a string of length 0. In the second set of two lists, each node had a basic integer value once again along with a text that was 2,500 bytes long. This pattern continued for every other list iteration.

We first built the code in C#, copied it, and then rewrote it in C to make the tests as comparable as feasible. The generated algorithms varied primarily in two ways. First, the C# algorithm adopted an object-oriented methodology in which each node was composed of a class; the linked list, for instance, was a class. We closely adhered to the structural paradigm when writing the C solution, using structs, pointers, and other tools. Second, after each of the 24 repetitions in C, we explicitly released the 5,000-node linked list. Since the garbage collector was not explicitly invoked in C#, the CLR was responsible for memory management. To reuse it for the new set of 5,000 nodes, we simply set the reference to the entire linked list (5,000 nodes) in C# to null. In C#, there is no need to remove anything because the delete keyword doesn't even exist. 7 Figure 3 displays the outcomes. Figures 2 and 3 demonstrate that Cfloating #'s point performance is nearly similar to that of C, although more optimization of.memory NET's management is necessary to match C efficiency levels.

IX. Memory Management

We sampled the machine's committed bytes during the memory management performance testing. This statistic provides information on how much memory was used during execution. The following is how committed memory is defined by Microsoft Performance Manager: The amount of virtual memory that has been committed is measured in Committed Bytes. (Physical memory that has space allocated on the disc paging file in case it needs to be written back to disc is known as committed memory.) This counter does not reflect an average; it just shows the most recent observed value. 6 The Performance Manager's output metrics provide information on the memory footprints of C and C#. The machine's committed bytes for three representative test runs of the memory management technique mentioned in C are displayed in Figure 4a. 150 Mbytes or such are the average committed bytes. It's interesting to see that memory use seems to become faster with time. Despite having identical input and code, the second and third runs seem to use less RAM than the first. The top watermark is around 280 Mbytes for all three tests. The way C# acts is very different (we only executed two runs, see Figure 4b). We need additional testing to confirm this conclusion, but it does not seem that the.NET memory management architecture can optimise memory utilisation over time. The low watermark (150 Mbytes) memory utilisation looks to be comparable to the C tests, while the high watermark is undoubtedly higher. The high watermark for C# is about 430 Mbytes, which is an increase of 50% over C.

X. Conclusions

These experiments demonstrate C's improved physical memory use. C# seems to have a long way to go before achieving C efficiency standards, especially for embedded devices where efficient resource use is crucial. For a number of reasons, C# and the .NET platform is not yet suitable for demanding real-time systems. In addition to lacking threading structures that would properly allow schedulability and determinism, it has unbounded execution of its garbage-collected environment. Our tests also revealed that the thread inheritance was ineffective. C# and.NET may be suitable for some soft and hard real-time systems, provided that these flaws are not serious. (See the sidebar titled "Suggestions for Further Study"). For instance, the problems of priority inversion and garbage collection execution are lessened by a single-threaded application that may allocate memory fully at programme setup. The fact that C# can communicate with operating system APIs, protect developers from intricate memory management logic, and achieve floating-point performance close to C also contributes. To address the issues that this study has revealed, disciplined programming is necessary for any real-time system.

REFERENCES

[1] Exploring Cross Language independency in .NET Framework Sarker, M.Z.H; Rahmanm, S. 9th International Multitopic Conference,IEEE INMIC 2005 DOI : 10.1109/INMIC.2005.334436 Publication Year:2005.

[2] .NET framework internal Components, Lutz, M.h.;P.A. Software, IEEE Volume:20,issue:1 DOI : 10.1109 /MS.2003.1159034 Publication Year:2003,Page(S) :74-80.

[3] ILJc: Porting Microsoft.NET IL (Intermediate Language) to Java S.Siddique, S.D Sheriff, H. Wijesuriya, C. Wickramaratne, J. Makalanda, First International Conference on Industrial and Information Systems, ICIIS 2006, 8 – 11-2006.

[4] Building Mobile Application With .Net Compact Framework, Hammad-ul-Hasnain, NED University of Engineering and Technology DOI: 10.1109 /SCONEST.2005.4382871 Publication Year: 2005.

[5] A Framework Profile of .NET Lammel,Ralf ; Linke,R.;Pek,E.;Varanovich, A Reverse Engineering (WCRE), 2011 18th Working Conference on DOI: 10- .1109/WCRE.2011.25 Publication Year:2011.

[6] "Conversations on .NET," Microsoft; [Online] www.msdn.microsoft.com/library/enus/dndotnet/html/dotnetconvers.asp?frame=true.

[7] P. Yao, "Windows CE 3.0: Enhanced Real-Time Features Provide Sophisticated Thread Handling," MSDN Magazine, Nov.2000, [Online] http://msdn.microsoft.com/msdnmag/issues/1100/Real CE/default.aspx.

[8] http://msdn.microsoft.com/. [Online]