SPAWAR
Systems Center
San Diego

TECHNICAL REPORT 1817
February 2000

Windows NT® Operating System Primitives

J. Drummond

SSC San Diego

TECHNICAL REPORT 1817
February 2000

Windows NT® Operating System Primitives

J. Drummond

SSC San Diego
San Diego, CA 92152–5001

SSC SAN DIEGO
San Diego, California 92152–5001

E. L. Valdes, CAPT, USN
Commanding Officer

R. C. Kolb
Executive Director

ADMINISTRATIVE INFORMATION

The work described in this Technical Report was performed for the Defense Advanced Research Projects Agency (DARPA) Information Technology Office by the Advanced Concepts and Engineering Division (D41) Technology Team (D4123), SPAWAR Systems Center, San Diego (SSC San Diego).

Released by
G. Leonard, Head
Technology Team

Under authority of
R. B. Volker, Head
Advanced Concepts and
Engineering Division

SB

EXECUTIVE SUMMARY

OBJECTIVE

There are many papers, articles, and technical reports that analyze the Microsoft® Windows NT® operating system (see References section). This report focuses on Microsoft® Windows NT® operating system primitives, specifically low-level kernel primitives. This analysis was conducted under a specifically conditioned environment to minimize variation of results, and I acknowledge the limitations of the results as such.

METHOD

This investigation of the Microsoft® Windows NT® operating system is not a comprehensive study. The low level kernel primitives were examined to provide some insight into Windows NT® system characteristics and capabilities. Initially, it may seem discriminatory to examine the non-real-time Microsoft® Windows NT® system for real-time characteristics; however, this examination will inspect any inherent potential real-time characteristics within the Microsoft® Windows NT® operating system and will not compare this admittedly non-real-time system with the features of a true real-time system.

The software developed for this analysis was based upon an interpretation of the elements from the Rhealstone benchmark (Kar, 1990). System examinations were conducted on the context switch, pre-emption, and interrupt latency areas. The Rhealstone benchmark has been categorized as a "fine-grained benchmark*" in that it focuses upon the examination of low level kernel primitives and, hence, is ideal for this investigation. However, this report does not use the entire benchmark suite of components. Elements that have not been used include semaphore shuffling measurement, the deadlock breaking measurement, and the intertask message latency measurement. One reason for omitting these components was that they did not apply to this investigation. Critical characteristics of a given real-time system that can be considered as foundational elements are those that accentuate timing and guarantee latencies and predictability as noted in Drummond (1996).

CONCLUSION

The following three sections discuss data produced by low-level analysis of the Microsoft® Windows NT® system:

- Context Switch
- Pre-emption
- Interrupt Latency

---

* K. Ghosh, B. Mukherjee, and K. Schwan. 1994. " A Surevey of Real-Time Operation Systems–Draft." Technical Report (Feb).

These three areas of investigation exhibit a high degree of bearing on a given systems capability and potential for real-time response. The overall discoveries and subsequent details confirm earlier findings that the Microsoft® Windows NT® operating system is indeed capable of supporting limited soft real-time deadlines.

CONTENTS

Figures

MICROSOFT® WINDOWS NT® OPERATING SYSTEM

The Microsoft® Windows NT® architecture is designed to include a Hardware Abstraction Layer (HAL) module. The design rationale for the creation of the HAL is that of increased portability across numerous platforms. The HAL module is an essential part of making this portability possible (Solomon, 1998). The HAL module interacts with the hardware on behalf of the kernel. This module also allows for a partitioning between the drivers/kernel and the lower level communication engines and between the input and output interfaces, including various interrupt controllers. User applications sit on top of the kernel layer (figure 1).

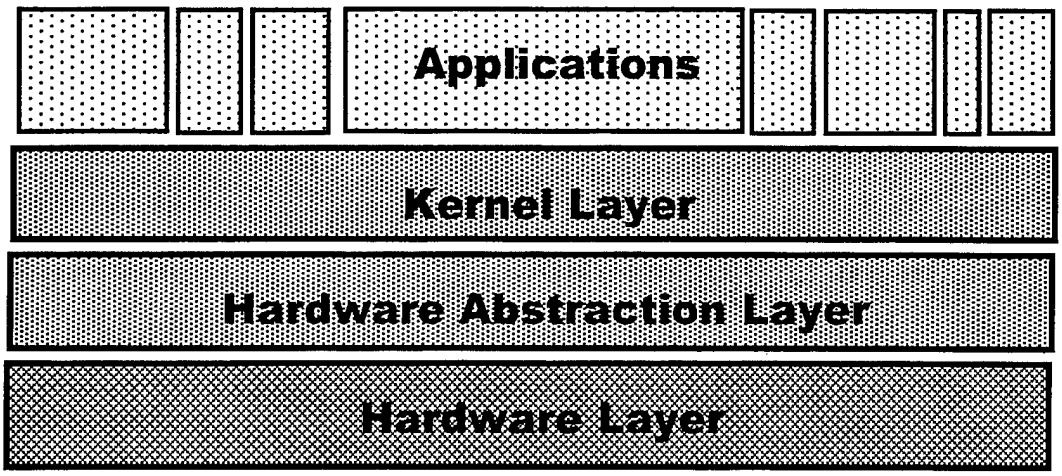

Figure 1. Microsoft® Windows NT® architecture.

The assignment of priority to a typical thread designed to be executed within the Microsoft® Windows NT® environment is based upon two elements. One of these elements consists of the priority of the process or class within which the thread was created. The possible class priorities, which can potentially be assigned, are IDLE_PRIORITY_CLASS, NORMAL_PRIORITY_CLASS, HIGH_PRIORITY_CLASS, and REALTIME_PRIORITY_CLASS. The selection of a class priority has a direct bearing upon the thread's ultimate execution priority level.

The thread priority assignment consists of the following:

THREAD_ PRIORITY_LOWEST is defined as THREAD_BASE_PRIORITY_MIN;

THREAD_ PRIORITY_BELOW_NORMAL is defined as THREAD_PRIORITY_LOWEST + 1;

THREAD_PRIORITY_NORMAL is defined as 0;

THREAD_PRIORITY_HIGHEST is defined as THREAD_BASE_PRIORITY_MAX:

THREAD_PRIORITY_ABOVE_NORMAL is defined as THREAD_PRIORITY_HIGHEST-1;

THREAD_PRIORITY_ERROR_RETURN is defined as MAXLONG;

THREAD_PRIORITY_TIME_CRITICAL is defined as THREAD_BASE_PRIORITY_LOWRT;

THREAD_PRIORITY_IDLE is defined as THREAD_BASE_PRIORITY_IDLE.

The actual execution priority level is obtained from a combination of the threads priority level and the priority setting of the class (Microsoft Corporation, 1998). The thread's priority level is one of the following values:

THREAD_PRIORITY_ABOVE_NORMAL indicates 1 point above normal priority for the priority class;

THREAD_PRIORITY_BELOW_NORMAL indicates 1 point below normal priority for the priority class;

THREAD_PRIORITY_HIGHEST indicates 2 points above normal priority for the priority class;

THREAD_PRIORITY_IDLE indicates a base-priority level of 1 for IDLE_PRIORITY_CLASS, NORMAL_PRIORITY_CLASS, or HIGH_PRIORITY_CLASS processes, and a base-priority level of 16 for REALTIME_PRIORITY_CLASS processes;

THREAD_PRIORITY_LOWEST indicates 2 points below normal priority for the priority class;

THREAD_PRIORITY_NORMAL indicates normal priority for the priority class;

THREAD_PRIORITY_TIME_CRITICAL indicates a base-priority level of 15 for IDLE_PRIORITY_CLASS, NORMAL_PRIORITY_CLASS, or HIGH_PRIORITY_CLASS processes, and a base-priority level of 31 for REALTIME_PRIORITY_CLASS processes.

The above values are defined in the winbase.h header file within the Microsoft® Development Network environment.

The execution priority of each thread, along with any processor affinity details, are used as the basis for the actual scheduling of the threads execution. As noted in Jones and Regehr (1998), thread execution contains assorted complexities "The priorities are divided into three ranges: real-time (16-31), normal (1-15), and idle (0). Priorities of threads in the normal range are boosted following I/O completions and decreased when the thread's time quantum runs out, as is often done in time-sharing systems. The system never adjusts the priorities of threads in the real-time range. The scheduler essentially selects the first thread of the highest runnable priority and runs it for its quantum, then places it at the tail of its priority list."

ANALYSIS PROCEDURE

The testbed used for these experiments is composed of several commercial off-the-shelf Intel-based Pentium II single processor systems (400 MHz) and generic components. These systems were connected via a simple 100 BaseT Ethernet. 3Com 100BaseT network cards and typical high-density display systems based upon the Intel740 video accelerator inhabit the network node personal computers. The testbed operating system was the Microsoft® Windows NT® version 4.0 build 1381 with Service Pack 4. Standard generic drivers were used during these tests, and no modifications were performed upon these devices. The primary and secondary storage devices, used on these systems were 512-MB Synchronous Dynamic Random Access Memory (SDRAM) (60ns) and a 16-GB New Technology File System (NTFS) format Small System Computer Interface (SCSI) (with parity check enabled), respectively. The Basic Input Output System (BIOS) used by these systems was the Award Modular BIOS version 4.51, with HAL: MPS 1.4-Advanced Programmable Interrupt Controller (APIC) platform. The internal clock mechanism was the NT multimedia timer, which provided a clock resolution of < 1 ms. The timer device, Stat Inc. (produced by Alpha Logic) had been used on limited specific program execution; this device provided increased clock resolution of 90 to 250 μs.

The Microsoft® Development Network environment version 6.0 compiler was used to develop the analysis programs. The programs were created from an object-oriented design approach based upon coding in the C++ language. The compilation was performed with no optimization functionality. These measurements were completed without considering systems caching. This may or may not have affected the results. The various analysis tests were performed by using the thread as the basic schedulable entity included within a given class. During the tests, the program threads were assigned priorities ranging from lowest, normal, idle, and time critical settings. When testing, the program class priorities were designated as idle, normal, high, and real-time. This approach examined the full range of treatment afforded to typical program thread execution. Report results included were all achieved with a combination of high-priority assignment to the class and a time-critical priority assignment to the executing thread.

As previously mentioned, the basic schedulable entity used within the Microsoft® Windows NT® environment is the thread and, as such, it presents itself as the best candidate for appraisal. All the measurements were based upon this thread entity; thus, there may or may not have been contention for resources such as memory or ports among the system threads and the measured threads. Report results do not account for this possibility of variance. This potential overhead is included in all report measurements.

CONTEXT SWITCH

The context of a processing entity consists of the entity details that can be saved on a stack and retrieved later. The context switch event can include various transition states when an entity is transferred from the Central Processing Unit (CPU) to the wait queue (figure 2). The measured time of this examined event was the time at which one thread (Thread-1) was swapped from the "run" state to the "wait" state and another thread (Thread-2) was swapped from the "wait" state to the "run" state. The context switch was measured periodically. This measured duration indicated the time required to save the "context" thread and place in its location the next thread. The results indicated a time interval average of 20 μs. For these tests, two threads of equal priority were examined 1 million times, and initial overall execution results were subtracted by the overhead of simple loop execution (inside threads), thread startup, and "sleep" calls.

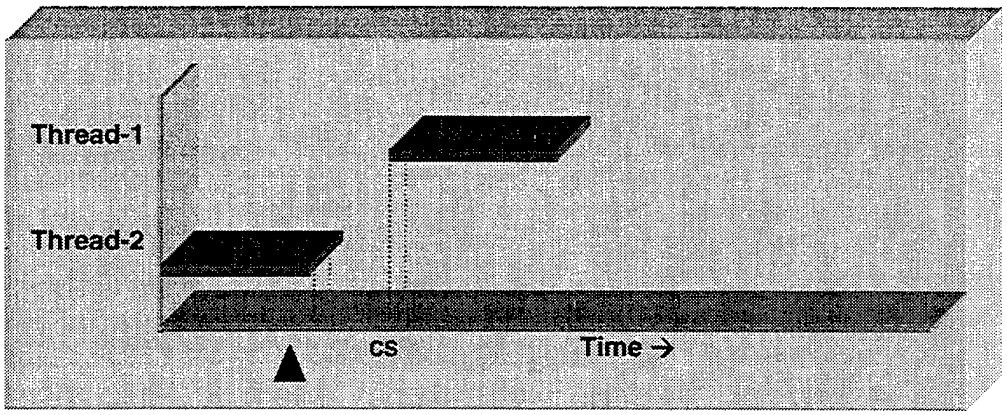

Figure 2. Context switch of threads sharing CPU resource.

PRE-EMPTION

The pre-emption time measurements fundamentally examined the event of a lower priority task replaced by a higher priority task within the CPU run cycle. The measurements were performed periodically and indicate the procedure required for handling an Interrupt Service Routine (ISR) or some other event. The event used here was "sleep" (with the lowest possible argument being 0 ms) alternated with waking. The test uses two threads, each with a different priority. The lower priority was 15 while the higher priority was 31. The program initially started processing the low-priority thread (Thread-L), which was soon pre-empted by the higher priority thread (Thread-H), as shown in figure 3. An ensuing "sleep" call was then performed whereby the lower priority thread (Thread-L) regains the CPU. This sequence was repeated 1 million times. The results indicated an average time of 30 to 40 μs. Again, the overhead time including simple loop execution (inside threads) and the "sleep" call execution were subtracted from the overall execution time. Of course, the experiment result also included the overhead of context switch time.

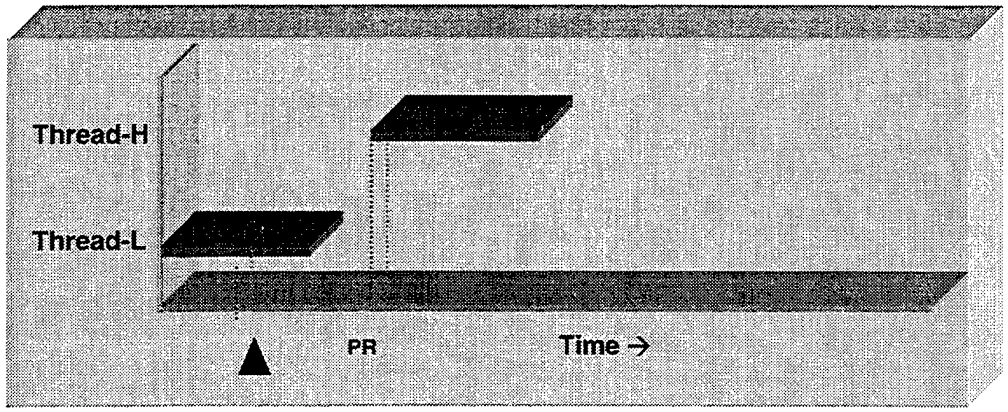

Figure 3. Thread pre-emption time.

INTERRUPT LATENCY

Interrupt latency measurements were used to investigate the execution of a software interrupt. The measurement of this event begins when notification of an interrupt occurred and included execution of the interrupt itself up to the beginning instruction (figure 4). However, because Microsoft® Windows NT® does not provide a simple user interrupt capability, the interrupt latency measurement included the time it takes to go through the interrupt handler itself. In this test, a thread was created with a given period, P. The thread has a loop to execute a small instruction set. The time to execute those instructions was measured. The period, P, was then increased and a new measurement was performed. The whole process was repeated many times (e.g., 1 million times). In this measurement, the average interrupt latency for Microsoft® Windows NT® was 10 to 15 μs. The results showed an increment in time based upon the periodic timer clock interrupt (i.e., interrupt latency).

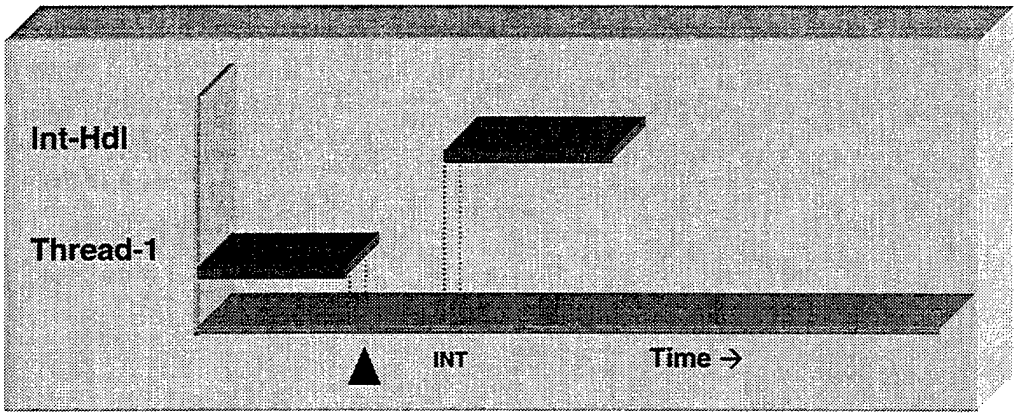

Figure 4. Interrupt latency time.

REFERENCES

Drummond, J. 1996. "Establishing A Real-time Distributed Bench*mark*," IEEE 10th International Parallel Processing Symposium. *Proceedings of the 4th International Workshop on Parallel and Distributed Real-Time Systems*, April 1996, Honolulu, HI. Institute of Electrical and Electronics Engineers.

Jones, M. B. and J. Regehr. 1998. "Issues in Using Commodity Operating Systems for Time-Dependent Tasks: Experiences from a Study of Windows NT." *Proceedings of the Eighth International Workshop on Network and Operating Systems Support for Digital Audio and Video* (pp. 107–110). July 1998. Cambridge, U.K. Institute of Electrical and Electronic Engineers.

Kar, R. 1990. "Implementing the Rhealstone Real-Time Benchmark," *Dr. Dobb's Journal* (Apr).

Microsoft Corporation. 1998. *Microsoft Development Network Library* (Oct). Redmond, WA.

Solomon, D. 1998. *Inside Windows NT*. 2nd ed. Microsoft Press, Redmond, WA.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	February 2000	Final

4. TITLE AND SUBTITLE
WINDOWS NT® OPERATING SYSTEM PRIMITIVES

5. FUNDING NUMBERS
PE: 0602301E
AN: DN307706
WU: CB16

6. AUTHOR(S)
J. Drummond

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
SSC San Diego
San Diego, CA 92152–5001

8. PERFORMING ORGANIZATION REPORT NUMBER
TR 1817

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
(DARPA)
Information Technology Office
3701 North Fairfax Drive
Arlington, VA 22203

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report focuses on Microsoft® Windows NT® operating system primitives, specifically low-level kernel primitives. Low-level kernel primitives were examined to provide some insight into Windows NT® system characteristics and capabilities. Context switch, pre-emption, and interrupt latency exhibited a high degree of bearing on given systems capability and potential for real-time response. The overall discoveries and subsequent details of this report confirm earlier findings that the Microsoft® Windows NT® operating system is capable of supporting limited soft real-time deadlines.

14. SUBJECT TERMS
Mission Area: Software Development
C++ computer programming
hardware abstraction layer module
object-oriented design

15. NUMBER OF PAGES
17

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAME AS REPORT

21a. NAME OF RESPONSIBLE INDIVIDUAL	21b. TELEPHONE *(include Area Code)*	21c. OFFICE SYMBOL
J. Drummond	(619) 553–4131 e-mail: drummond@spawar.navy.mil	D4123

INITIAL DISTRIBUTION

D0012	Patent Counsel	(1)
D0271	Archive/Stock	(6)
D0274	Library	(2)
D027	M. E. Cathcart	(1)
D0271	D. Richter	(1)
D41	R. B. Volker	(1)
D411	M. B. Vineberg	(1)
D411	R. E. Younger	(1)
D4121	A. D. Sandlin	(2)
D4123	J. J. Drummond	(3)
D4123	G. E. Leonard	(1)
D4123	M. A. Neer	(1)

Defense Technical Information Center
Fort Belvoir, VA 22060–6218 (4)

SSC San Diego Liaison Office
Arlington, VA 22202–4804

Center for Naval Analyses
Alexandria, VA 22302–0268

Navy Acquisition, Research and
Development Information Center
Arlington, VA 22202–3734

Government–Industry Data Exchange
Program Operations Center
Corona, CA 91718–8000

Defense Advanced Research Projects Agency/ITO
Arlington, VA 22203–1714 (2)

Dudley Knox Library
Naval Postgraduate School
Monterey, CA 93943–5101 (2)