REAL-TIME OPERATING SYSTEMS

A Practical Perspective

Submitted By:	Matt Verber
Submitted To:	Dr. Sebern
Date:	April 24, 1998
Class:	CS-384, Sec. 2

TABLE OF CONTENTS

TABLE OF FIGURES

INTRODUCTION

Real-time operating systems are an important class of operating systems. Like all operating systems, real-time systems can be examined from a variety of perspectives, including theory and actual implementation. This paper will explore several practical aspects of real-time systems. Specifically, this paper will:

- present a general description of real-time systems
- illustrate typical uses of real-time systems
- discuss an existing real-time system

DEFINITION OF REAL-TIME OPERATING SYSTEMS

Over the years, the fundamental definition of a real-time operating system has remained the same. In the early 1980s, a real-time system was described as:

"[An environment] characterized by processing activity triggered by randomly accepted external events. The processing activity for a particular event is accomplished by sequences of processing tasks, each of which must complete within rigid time constraints. … Characteristically, the computer system is completely dedicated to the control application and has been configured to guarantee on-time responses even at peak loads. The environment is such that utilization of equipment is less important than responsiveness to the environment."[1]

Rigid time constraints and on-time responses are the defining factors of real-time operating systems. The importance of these factors is reiterated in two definitions of real-time systems from the late 1990s. According to the first definition:

[1] Harold Lorin and Harvey M. Deitel, 1981, *Operating Systems* (Reading, Massachusetts: Addison-Wesley Publishing Company, Inc.), p. 65.

"A real-time operating system has well-defined, fixed time constraints. Processing must be done within the defined constraints, or the system will fail. … A real-time system is considered to function correctly only if it returns the correct result within any time constraints."[2]

The second definition explains:

"A real-time system is one in which the correctness of the computations not only depends upon the logical correctness of the computation but also upon the time at which the result is produced. If the timing constraints of the system are not met, system failure is said to have occurred."[3]

The definitions from the 1990s agree with the definition from the 1980s. By the 1990s, however, the definition of real-time systems had grown. Unlike the older reference, the sources from the 1990s divide real-time operating systems into two categories: hard real-time systems and soft-real time systems.

Hard Real-Time Systems

Hard real-time operating systems guarantee critical tasks will complete in a given amount of time. To honor this guarantee, all delays in the system must be bounded. In hard real-time systems, if a process completes correctly but takes longer than its given amount of time, the process fails. An example illustrates this point.

"A … robot … has to pick up something from a conveyor belt. The piece is moving, and the robot has a small window to pick up the object. If the robot is late, the piece won't be

---

[2] Abraham Silberschatz and Peter Baer Galvin, 1998, *Operating System Concepts* (Reading, Massachusetts: Addison Wesley Longman, Inc.), p. 18.

[3] "Comp.realtime FAQ," 18 April 1998. [Internet, WWW]. ADDRESS: http://www.realtime-info.be/encyc/techno/publi/faq/rtfaq.html

there anymore, and thus the job will have been done incorrectly, even though the robot went to the right place."[4]

The scheduler plays an important role in hard real-time systems. One source explains:

"Generally a process is submitted along with a statement of the amount of time in which it needs to complete or perform I/O. The scheduler then either admits the process, guaranteeing that the process will complete on time, or rejects the request as impossible. … Such a guarantee requires that the scheduler know exactly how long each type of operating system function takes to perform, and therefore each operation must be guaranteed to take a maximum amount of time."[5]

Soft Real-Time Systems

Soft real-time systems are less restrictive than hard real-time systems. In these systems, critical tasks are given priority over non-critical tasks. Delays need to be bounded so the critical tasks do not wait forever, but these bounds are not as severe as in hard real-time systems. In soft real-time systems, if a process completes correctly but takes longer than its given amount of time, the result may still be useful.

As in hard real-time systems, the scheduler plays an important role in soft real-time systems. According to one source:

"Implementing soft real-time functionality requires careful design of the scheduler and related aspects of the operating system. First, the system must have priority scheduling, and real-time processes must have the highest priority. The priority of real-time processes must not degrade over time, even though the priority of non-real-time

[4] Ibid.

[5] Silberschatz and Galvin, p. 142.

processes may. Second, the dispatch latency must be small. The smaller the latency, the faster a real-time process can start executing once it is runnable."[6]

APPLICATIONS OF REAL-TIME OPERATING SYSTEMS

Unlike the fundamental definition of real-time operating systems, typical applications of real-time systems have changed over the years. In the early 1980s, real-time systems were used almost exclusively for control applications. In fact, one definition from the 1980s lists only two types of real-time systems, both of which are controllers. According to this definition:

"There are two types of real-time systems, systems which interact with their environments to perform work: the process control system and the process monitor system. The process control system will take analog or digital sensor input, analyze it, and then cause an action to occur which changes the process which it is controlling. … The process monitor type of system does not affect the process that it is monitoring but merely reports on it; it too accepts both analog and digital input data."[7]

Over time, the uses for real-time systems grew. Control applications remained popular, however. A source from the 1990s explains that in many real-time systems:

"Sensors bring data to the computer. The computer must analyze the data and possibly adjust controls to modify the sensor inputs. Systems that control scientific experiments, medical imaging systems, industrial control systems, and some display systems are real-time systems. Also included are some automobile-engine fuel-injection systems, home-appliance controllers, and weapon systems."[8]

---

[6] Ibid.

[7] Stanley A. Kurzban, Thomas S. Heines, and Anthony P. Sayers, 1984, *Operating Systems Principles* (New York: Van Nostrand Reinhold Company Inc.), p. 60.

[8] Silberschatz and Galvin, p. 18.

With the development of soft real-time systems, the uses for real-time systems increased substantially. Hard real-time systems cannot support certain devices because their waiting time cannot be guaranteed. For example:

“Secondary storage of any sort is usually limited or missing, with data instead being stored in short-term memory, or in read-only memory (ROM). … Most advanced operating-system features are absent too, since they tend to separate the user further from the hardware, and that separation results in uncertainty about the amount of time an operation will take.”[9]

Because soft real-time systems have fewer constraints, these devices can be supported. This allows for additional applications. Areas such as “multimedia, virtual reality, and advanced scientific projects such as undersea exploration and planetary rovers”[10] can utilize soft real-time systems.

EXISTING REAL-TIME OPERATING SYSTEMS

Examining existing real-time operating systems reveals how real-time systems are implemented. There are dozens of real-time systems available for different platforms. Finding details on these systems can be difficult, however. Many manufacturers simply state that their operating systems are real-time without explaining how this is accomplished. One real-time system, QNX, offers more insight, however. This operating system will be examined in detail to explain how real-time processing works.

[9] Ibid.

[10] Silberschatz and Galvin, p. 19.

Characteristics of a Good Real-Time Operating System

Before examining the specific real-time system, it is useful to know how this system should be evaluated. According to the FAQ for the comp.realtime newsgroup:

"A good RTOS is not only a good kernel. A good RTOS should have good documentation and should be delivered with good tools to develop and tune your application. So even if some figures like the interrupt latency and context switch time are important, there are a lot of other parameters that will make a good RTOS. For example, a RTOS supporting many devices will have more advantages than a simple, very good nano-kernel."[11]

In addition, there are certain specifications that are important to recognize in real-time operating systems. According to comp.realtime, RTOS manufacturers should specify the following numbers:

- The interrupt latency (i.e. time from interrupt to task run). This has to be compatible with application requirements and has to be predictable. This value depends on the number of simultaneous pending interrupts.
- For every system call, the maximum time it takes. It should be predictable and independent from the number of objects in the system.
- The maximum time the OS and drivers mask the interrupts.
- System interrupt levels.
- Device driver IRQ levels, maximum time they take, etc.[12]

[11] Comp.realtime FAQ

[12] Ibid.

QNX

QNX Software Systems Ltd. develops real-time operating systems for personal computers. According to QNX:

"The QNX Operating System is ideal for real-time applications. It provides multitasking, priority-driven preemptive scheduling, and fast context switching – all essential ingredients of a real-time system."[13]

Architecture

QNX is built around a kernel known as the QNX Microkernel. The kernel is small and performs only four functions, including:

- Interprocess communication: "The Microkernel supervises the routing of messages; it also manages two other forms of IPC: proxies and signals."

- Low-level network communication: "The Microkernel delivers all messages destined for processes on other nodes."

- Process scheduling: "The Microkernel's scheduler decides which process will be executed next."

- First-level interrupt handling: "All hardware interrupts and faults are first routed through the Microkernel, then passed on to the appropriate driver or system manager."[14]

[13] "The Philosophy of QNX," 18 April 1998. [Internet, WWW]. ADDRESS: http://www.qnx.com/literature/qnx_sysarch/intro.html

[14] "The Microkernel," 18 April 1998. [Internet, WWW]. ADDRESS: http://www.qnx.com/literature/qnx_sysarch/microkernel.html

All other operating system services are handled through QNX processes, including the:

- Process Manager
- File System Manager
- Device Manager
- Network Manager[15]

The following figure illustrates QNX's architecture.

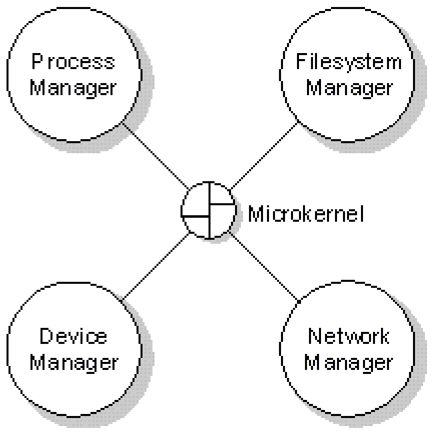

Figure 1: QNX Architecture

SOURCE: "The Philosophy of QNX," 18 April 1998. [Internet, WWW]. ADDRESS: http://www.qnx.com/literature/qnx_sysarch/intro.html

QNX's architecture fits the model of a good real-time system established by comp.realtime. The kernel of this system is limited, but its features are not. Processes are used to expand the functionality of the system. Instead of incorporating these features directly into the kernel, the system is modular. This allows unneeded functions to be left out, conserving memory space.

[15] The Philosophy of QNX.

Scheduler

QNX’s technique of scheduling processes distinguishes it as a real-time operating system. Because of this, the scheduler must be examined in detail.

QNX is a soft real-time system. As mentioned earlier, soft real-time systems prioritize tasks. According to the QNX documentation:

“In QNX, every process is assigned a priority. The scheduler selects the next process to run by looking at the priority assigned to every process that is READY (a READY process is one capable of using the CPU). The process with the highest priority is selected to run.

The priorities assigned to a process range from 0 (the lowest) to 31 (the highest). The default priority for a new process is inherited from its parent; this is normally set to 10 for applications started by the Shell.”[16]

Critical tasks are assigned high priorities. This ensures that they will run before less critical or non-critical tasks. This fits the definition of soft real-time systems.

If multiple processes have the same priority level and are READY at the same time, QNX uses three different scheduling methods to select the next process to run. These scheduling methods include:

- FIFO scheduling
- round-robin scheduling
- adaptive scheduling

[16] The Microkernel.

According to the QNX documentation:

In FIFO scheduling, a process selected to run continues executing until it:

- voluntarily relinquishes control
- is preempted by a higher-priority process

In round-robin scheduling, a process selected to run continues executing until it:

- voluntarily relinquishes control
- is preempted by a higher-priority process
- consumes its timeslice [a timeslice is 50 milliseconds]

In adaptive scheduling, a process behaves as follows:

- If the process consumes its timeslice, its priority is reduced by 1. This is known as priority decay. Note that a “decayed” process won’t continue decaying, even if it consumes yet another timeslice without blocking – it will drop only one level below its original priority.
- If the process blocks, it immediately reverts to its original priority.[17]

Because a higher priority process can preempt a lower priority process, real-time process should be assigned high priorities. This is fundamental to the implementation of a soft real-time system.

[17] Ibid.

<u>**Latency**</u>

QNX offers information concerning latency in its system. QNX explains:

"It's … crucial that you minimize the time it takes from the occurrence of an external event to the actual execution of code within the program responsible for reacting to that event. This time is referred to as latency."[18]

There are two types of latency in a QNX system: interrupt latency and scheduling latency.

Interrupt latency is:

"…the time from the reception of a hardware interrupt until the first instruction of a software interrupt handler is executed. QNX leaves interrupts fully enabled almost all the time, so that interrupt latency is typically insignificant. But certain critical sections of code do require that interrupts be temporarily disabled. The maximum such disable time usually defines the worst-case interrupt latency – in QNX this is very small." [19]

[18] Ibid.

[19] Ibid.

According to QNX, the interrupt latency for a variety of processors is:

Interrupt Latency	Processor
3.3 microsec	166 MHz Pentium
4.4 microsec	100 MHz Pentium
5.6 microsec	100 MHz 486DX4
22.5 microsec	33 MHz 386EX

Table 1: Interrupt Latency

SOURCE: “The Microkernel,” 18 April 1998. [Internet, WWW]. ADDRESS: http://www.qnx.com/literature/qnx_sysarch/microkernel.html

Scheduling latency is:

“… the time between the termination of an interrupt handler and the execution of the first instruction of a driver process. This usually means the time it takes to save the context of the currently executing process and restore the context of the required driver process. Although larger than interrupt latency, this time is also kept small in a QNX system.”[20]

[20] Ibid.

According to QNX, the scheduling latency for a variety of processors is:

Scheduling Latency	**Processor**
4.7 microsec	166 MHz Pentium
6.7 microsec	100 MHz Pentium
11.1 microsec	100 MHz 486DX4
74.2 microsec	33 MHz 386EX

Table 2: Scheduling Latency

SOURCE: “The Microkernel,” 18 April 1998. [Internet, WWW]. ADDRESS: http://www.qnx.com/literature/qnx_sysarch/microkernel.html

As stated in comp.realtime, it is important to know what these numbers are. These values must be compatible with the requirements of a desired application.

<u>**Applications**</u>

Perhaps the best way to judge the quality of a real-time operating system is to examine its actual applications. QNX boasts a variety of real-world applications, including automation, telecom, hospitals, Internet, point of sale, transportation, inventory, safety, space, and research. For example, QNX helped automate Cadbury Chocolate, establish a point of sale system for Arby’s, and track cargo for Japan Airlines.[21] QNX is widely used, so it should be seen as a successful implementation of a real-time system.

[21] “Real World Apps,” 18 April 1998. [Internet, WWW]. ADDRESS: http://www.qnx.com/realworld/rwindex.html

CONCLUSION

Real-time systems will continue to evolve. As in the past, the general definition of a real-time system will remain the same, but new applications for these systems will develop. Furthermore, new implementations of real-time systems will join the dozens of existing implementations. Without a doubt, real-time systems will continue to occupy an important position in the family of operating systems.

BIBLIOGRAPHY

"CMX." 18 April 1998. [Internet, WWW]. ADDRESS: http://www.cmx.com

"Comp.realtime FAQ." 18 April 1998. [Internet, WWW]. ADDRESS: http://www.realtime-info.be/encyc/techno/publi/faq/rtfaq.html

Farmer, Jerry. 1997. *Application Note AN585: A Real-Time Operating System for PICmicro Microcontrollers.* Microchip Technology Inc.

Janson, Philipe A. 1985. *Operating Systems: Structures and Mechanisms*. New York: Academic Press, Inc.

Kurzban, Stanley A., Thomas S. Heines, and Anthony P. Sayers. 1984. *Operating Systems Principles*. New York: Van Nostrand Reinhold Company Inc.

Lorin, Harold and Harvey M. Deitel. 1981. *Operating Systems*. Reading, Massachusetts: Addison-Wesley Publishing Company, Inc.

"OSE Realtime Kernel." 18 April 1998. [Internet, WWW]. ADDRESS: http://www.enea.se/ose/products/RTK.html

"Real World Apps." 18 April 1998. [Internet, WWW]. ADDRESS: http://www.qnx.com/realworld/rwindex.html

"RTEMS Home Page." 18 April 1998. [Internet, WWW]. ADDRESS: http://lancelot.gcs.redstone.army.mil/rg4/rtems.html

"RTMX OS." 18 April 1998. [Internet, WWW]. ADDRESS: http://www.rtmx.com

"RTOS Buyer's Guide." 18 April 1998. [Internet, WWW]. ADDRESS: http://www.eg3.com/ulc/realxulr.html

Silberschatz, Abraham and Peter Baer Galvin. 1998. *Operating System Concepts*. Reading, Massachusetts: Addison Wesley Longman, Inc.

"Technical Description of Harmony." 18 April 1998. [Internet, WWW]. ADDRESS: http://wwwsel.iit.nrc.ca/projects/harmony/techdes.html

"The IEEE-CS TC-RTS Home Page." 18 April 1998. [Internet, WWW]. ADDRESS: http://cs-www.bu.edu/pub/ieee-trs/Home.html

"The Microkernel." 18 April 1998. [Internet, WWW]. ADDRESS: http://www.qnx.com/literature/qnx_sysarch/microkernel.html

"The Philosophy of QNX." 18 April 1998. [Internet, WWW]. ADDRESS: http://www.qnx.com/literature/qnx_sysarch/intro.html

"VRTX Real-Time Operating System." 18 April 1998. [Internet, WWW]. ADDRESS: http://192.94.39.7/microtec/brochures/vrtx.html

"VxWorks 5.2." 18 April 1998. [Internet, WWW]. ADDRRESS: http://www.wrs.com/products/html/vxwks52.html