An Introduction to Real-Time Programming

Dennis Ludwig
Aeronautical Systems Center

Real-time programming requires that you consider things that are hidden from high-level application programmers. Some of those considerations are choice of hardware, operating system, and programming language. Although these same choices have to be made by every programmer, the real-time programmer makes the choice from a different set of options.

Real-time programmers must have a intimate relationship with computer hardware, and if there is one, the operating system. Thus, another name for real-time programming is low-level programming, and at this low level, the silicon world looks a lot different from high-level application programming. This article will familiarize the reader with some of the terms and considerations of real-time programming like the selection of hardware, operating systems, and high-level language selection.

Definitions of *real time* vary [1]. Savitzky [2] provides two definitions, but here, a real-time system is defined as one in which the timing of the result is as important as the logical correctness. These systems can be classified as hard or soft. In a hard real-time system, critical computations have deadlines, and if the deadlines are not met, the system has failed. In a soft real-time system, missing a deadline may not be a failure [3]. Soft deadlines are based on average performance. To be considered hard, the computation times must be deterministic. A system is deterministic if it has the ability to respond to an event in a predictable period of time [4].

Another system classification is embedded or not embedded. The most common definition of an embedded system is one that is part of a larger system. The author prefers to define an embedded system as one that does not interact with a human. Inputs come from a sensor and outputs go to a controller, as opposed to the familiar keyboard input and video display output. Most real-time systems are embedded and consist of machine communicating with machine.

Some real-time systems are *synchronous*, but most are *asynchronous*. A synchronous system has a clock that keeps track of time and provides timing signals. An asynchronous system can accept inputs from the outside world at any time; there is no common timing signal to warn or to synchronize input. The terms synchronous and asynchronous are also applied to message passing, where they have different meanings. In message passing, synchronous means the sender waits for the message to be received; asynchronous means the sender can proceed immediately after sending a message [5].

Many real-time systems must do simultaneous tasks, and making these tasks coordinate available resources is one aspect of real-time programming. Available resources might be physical such as access to hard storage, a printer, an input/output (I/O) port, or they might be logical such as a non-shareable code segment. The following are the types of issues that real-time programmers handle: "How long will each task take to complete?" "How soon can another task be scheduled and start to run?" "Which task is most important?" "What happens if one task takes too long?" "Do the tasks have to communicate, and if so, how?"

> *"Many real-time systems must do simultaneous tasks, and making these tasks coordinate available resources is one aspect of real-time programming."*

In designing real-time systems, choices have to be made about hardware as well as software. Software decisions include operating system considerations as well as language and algorithm choices.

Hardware

Inexpensive hardware choices for controlling a real-time project include microcontrollers like the Motorola 68HC11, Intel 8051, or PIC 16F84. See [6] for an overview of more choices. A personal computer system could be used if the operating system allows access to peripheral ports. The basic requirements are input, some processing capability, and an output.

A microcontroller is optimized for data acquisition and control purposes. It contains a central processing unit (CPU), random access memory, read only memory, serial and parallel I/O ports, an analog to digital converter, and a timer circuit. All of these systems communicate via a data bus. Microcontroller folks do not refer to the CPU as a microprocessor, just as the CPU [7]. However, a microcontroller can be defined as a microprocessor with special hardware support [8].

There are hundreds of microprocessors classified as either eight, 16, or 32 bit. The Lego Mindstorms robot uses a Hitachi eight-bit H8/3292 microprocessor [9]. Another classification is reduced instruction set computing (RISC) or complex instruction set computing (CISC). RISC processors use simple instruction sets, few memory references, lots of registers, and pipelined instruction sets, but that does not make them better for all tasks [10].

Real-time processors have one or more interrupt request lines (IRQ) to connect to peripherals. When a peripheral wants CPU attention, it asserts the IRQ. This is considered an asynchronous event. If the CPU services the request, the current task is preempted, the program counter and other registers are saved on the stack, and a jump is made to the location of an interrupt service routine (ISR). The ISR is the software associated with the device causing the interrupt. When the ISR is finished, the state of the processor is restored, and execution is continued. The program counter and register states for a task are called the context of the program.

Computer program execution is a sequence of synchronous events con-

trolled by a program counter and a system clock. A software error, like a divide by zero, may cause an exception or system trap. Traps are not the same as interrupts, but they are usually handled the same way. Interrupts and asynchronous events are externally caused while traps are considered synchronous even though they are unexpected. Traps usually result from software errors.

Some items to consider when choosing hardware are the amount of random access memory needed, whether floating point assistance is required and provided, and the granularity of the system time base. The number of processors used is another decision that will also affect the software needed. In a multiprocessor system, several CPUs operate simultaneously and share the processing workload.

One method that is used to distinguish microprocessors is the millions of instructions per second (MIPS) performance (or Meaningless Indicator of Performance for Salesmen, according to [10]). One form of MIPS ratings, called relative MIPS, measures how many instructions a VAX 11/780 could have executed in the same amount of time a given computer can run a benchmark program. Different computer architectures and other factors make this rating less useful, even misleading, but it is still used [11].

Clock speed is also useless as a measure of performance because processors vary in the number of clock cycles required for memory access and other instruction executions [12].

Besides a complicated choice of hardware issues, the real-time programmer has different software issues to consider. Data structures, control structures, and operating systems look different from a low-level perspective.

Data Structures

Real-time programmers have to deal with some data structures that are normally hidden from high-level programmers. The task control block is where the CPU stores the state of the last run task so it can be restored.

The semaphore, invented by Edsger Dijkstra[1], is used to coordinate processes and shared resources. There are two types of semaphores: binary and counting. A binary semaphore is used to provide mutual exclusion. A counting semaphore is used when a resource can be used by more than one task at a time [13]. The basic counting type is an integer variable that is accessed only through two basic operations, wait and signal; however, an initialize operation is also usually provided. Modifications to the integer value of the semaphore must be executed without interruption.

A macro is a label that replaces a block of instructions that is used more than once, but only coded once. It differs from a subroutine in that the assembler inserts the code where the call is made rather than having a jump-to-it command. It works by text substitution and is usually faster than a subroutine but takes up more memory.

A pipe is a stream of data used to connect tasks, or to provide task communication. A buffer, like a first-in-first-out buffer, can implement it. This eliminates the need to use a file to store temporary results. A pipe self-regulates its flow so that it uses less disk space than a temporary file [14].

A script is a file of characters used for input or instructions to a program. The programmer can use it to simulate an interactive user or other I/O device. A script file could be a list of commands for a command interpreter such as a batch file [15].

"Remember that an interrupt request is a request. The processor may have some critical processing to finish before it responds and services the request."

A communications port consists of a queue to hold messages and two semaphores. One semaphore controls producers, or the process that generates messages, and the other controls consumers, which are the processes that use the messages.

Control Structures

Two basic software control structures are the polling loop and event-driven systems. In a polling loop, the program examines each input in turn to see if an event has occurred. The program structure is a loop, and the inputs to be examined are predetermined. If an event occurs, the polling is stopped, some action is taken, and the polling continues. Controlling refrigerator temperature could be done with a simple polling loop. The temperature would be read as input and the compressor turned on or off based on the reading. If the temperature is within controlled limits, no action is taken.

There are three kinds of event-driven systems: foreground/background, multitasking, and multiprocessor [16]. In an event-driven system, the program loops (sometimes called a spin loop) until an interrupt occurs, at which time the loop stops and services the interrupt, and then continues. Interrupt latency is the interval of time measured from the instant an interrupt is asserted until the corresponding ISR begins to execute. Remember that an interrupt request is a request. The processor may have some critical processing to finish before it responds and services the request.

Context switching time is the time the operating system takes to store the state of the processor or the contents of the registers before it begins to process another task. Because the context switching time and interrupt latency may not be constant times, making the system predictable can be a challenge for the real-time programmer.

Microcontrollers come with a monitor program that allows programmers to develop and execute software. Do not confuse this monitor with the screen monitor. The word monitor is also used for a shared data structure that contains a semaphore [2]. A monitor for a microcontroller is a program that combines a debugger, some device drivers, and a bootstrap loader program. If provided, it is usually part of the read-only memory. The bootstrap program initializes the system by setting the registers to known states, and then it calls in or loads the rest of the required software routines. A monitor may include an assembler, which is a program that translates source code into object code, and can also produce a listing file.

A linker combines one or more object code files to produce a hex file. Two standard formats for the hex file are Intel hex and Motorola S record files. These are American Standard Code for Information Interchange (ASCII) files so they can be transported through serial ports. A loader converts the hex file into an executable form called a binary file [17].

The foreground/background system is basically a polling loop with interrupts enabled. The loop runs in the background. Only critical processing is done inside the interrupt.

Multitasking is a technique to allocate

CPU processing time among several tasks. While an executing task is using the physical processor resources, other tasks have their resources stored in memory. These resources include the program counter, stack memory area, and stack pointer. These systems are classified as preemptive or nonpreemptive depending on whether they can preempt an existing task or not. In a preemptive system, each task is given a time slice.

Multiprocessor systems have more than one processor. For more information on design considerations for multiprocessor systems, see [18]. Multitasking and multiprocessor systems usually require an operating system to provide task synchronization and inter-task communication.

Operating Systems

Some operating systems are dedicated to a particular controller board. Some are designed exclusively for real time but not a specific board, and others are general-purpose programs that have been enhanced to provide real-time services.

Other names used for software routines that control processing are the executive, monitor, task manager, or kernel. These terms are sometimes used interchangeably. A program that sits quietly in the background until it is called to perform its task is called a daemon.

Some operating systems are available with the source code, but many are not. If a bug appears in the code, and source code is not available, then the programmer has to work closely with the vendor to resolve the problem. A freeware real-time multitasking kernel with source code can be found at Embedded Systems Programming <www.embedded.com> or at <www. ucos-ii.com>.

Information on some real-time operating systems (RTOS) can be found at <www.rtlinux.org>, <www.aero.polimi.it/~rtai>, <www.qnx.com>, <www.windriver.com>, or <http://seg.iit.nrc.ca/projects/harmony>. Other operating systems could be used (like MSDOS) for very simple real-time tasks even though they are not optimized for real time as long they provide access to the system I/O ports. Usually a RTOS must support multithreading, provide timing features, be predictable, and run with low overhead.

Operating systems are complex programs that interface hardware with user programs. Some modules that make up an operating system are the scheduler, dispatcher, context switch, memory manager, inter-process communication module, real-time clock manager, interrupt manager, and file system manager.

The scheduler is sometimes called the dispatcher [19]. The purpose of the scheduler is to select a process from among those ready to run, schedule time for it on the CPU, and maintain a list of ready processes.

The dispatcher dispatches jobs to the CPU, using the list created by the scheduler. Most real-time operating systems use a priority-based preemptive scheduler to keep the system in order. Priority-based means that some type of priority scheme will be used to determine how the schedule is made. Preemptive means that a task can be stopped, or another task can be preempted. In a nonpreemptive system, a task must run to completion or until it suspends itself.

"Predictability is extremely important in real-time programming, and to get it, you need to keep track of time. Response time is the time it takes the computer to recognize and respond to an external event."

A task gives up processor control when it terminates, when it voluntarily suspends, when its time slice is up, or when a higher priority task becomes available and the scheduler preempts the running task to let the higher priority one run. Preemptive ability reduces priority inversion, which is having a higher priority task wait on a lower priority task. Priority inversion cannot be prevented, but it can be reduced. The scheduler also preempts a task when its time slice is up in order to keep one process from completely controlling the CPU and blocking other tasks from running. The scheduler is the part of the operating system that decides who gets to do what and when.

If several tasks are allowed to have the same priority, they are executed in the order they become ready; this is called round-robin scheduling. In a static priority system, the priorities do not change during run time. Changing the priority of a task during run time is supported by some systems, and the algorithms for assigning dynamic priorities are different from the ones used for static priorities. One dynamic scheduling policy is the earliest-deadline-first algorithm. A static priority policy can be analyzed so the system reaction is more predictable.

If higher priority tasks keep a lower priority task from running, the condition is called starvation. The number of tasks should be kept to a minimum and careful consideration given to priority choices. The selection should be made based on what the task does during run time.

In a deadlock, two tasks are waiting for resources that are held by each other. Neither task has all the resources needed to complete, and will not be able to get them all because the other task is holding resources and waiting to get more. Tasks should be required to get all needed resources before proceeding, and they must get the resources in the same order.

For an application that will be recording, reporting, and storing data simultaneously, each task is a separate, scheduled instruction stream. In some systems, the instruction streams are called processes, in other systems they are called tasks, and sometimes they are called threads. Since some tasks are more important than others are, some sort of prioritization is employed. If a piece of code needs to be executed without interruptions or being preempted, a data structure called a semaphore is used.

Real-Time Languages

A lot of real-time programming is done in assembly language. C is popular, as well as C++ and Forth. Although Forth is an interpreted language, it is efficient because of its stack-oriented design. Java is also being used, or rather a form of Java is being used.

A language with automatic garbage collection is not a good choice for real time because it hinders determinism, but there is a working group making a real-time version of Java, the Real-Time Specification for Java.

Ada was designed for real time and is the most powerful of those mentioned. Annex D of the Ada language specification is devoted to real-time issues, and any compiler that implements annex D will also implement annex C, which is the Systems Programming Annex. The strong type checking can be turned off to increase speed by using a pragma, while

representation clauses allow mapping to the hardware. In Ada, a pragma is a directive to the compiler.

The Time Element

Predictability is extremely important in real-time programming, and to get it, you need to keep track of time. Response time is the time it takes the computer to recognize and respond to an external event. Survival time is the time during which the data will be valid. Throughput is the number of events that the system can handle in a given time period [20].

As an example, consider a red traffic light with a queue of cars waiting to go through. When the light turns green, it takes time for the first driver to comprehend that it is time to go. There is some reaction time for them to move the foot from the brake to the accelerator. The second car undergoes the same time delay as the driver recognizes that the first car is moving, and he or she can now begin to accelerate. Survival time is the time the light remains green. Throughput is the number of cars that get to go through the light.

Time is a factor in reading, storing, or recording data. For the system to store data after it is sensed, a disk may be used. When the read/write heads move to the proper cylinder or track, there is some seek time involved (about 25 milliseconds) and some settling time. The electromechanical movement has to settle before the read begins. The proper head has to be activated. Rotational delay is the time spent waiting for the proper record to rotate under the head. The data transfer rate is the speed at which the data is transferred from the head to the storage medium and is determined by the rotational speed and density of the recording medium. Because these times will be different for each operation, the average times must be calculated and the worst-case times known for proper predictability to be made.

Other time factors considered by a real-time programmer are bus latency and context switching time. Bus latency is the delay incurred when the CPU needs to acquire the bus to transfer a command or data. Switching the CPU from executing one process to executing another requires saving the state, or context, of the old process and loading the context of the new process. The task that does this is called a context switch, and it takes time to execute. First, a process has to be selected from those that are ready. This is performed by the scheduler part of the operating system, and the selection process has more time to be considered and accounted.

To conceptualize how processes have to work together but still compete for resources, most courses on real time use Edsger Dijkstra's dining philosophers problem [21]. There are a group of philosophers who spend their time either thinking or eating. They sit at a round table with a bowl of rice in the middle and one chopstick on either side of them. In order to eat, they have to acquire the chopstick on the left and on the right of them, and return the sticks when finished. This is a classic synchronization problem used to demonstrate allocating resources between competing processes without getting into a deadlock or starvation mode. An Ada implementation for a solution can be found in [22] Chapter 11.

"If several tasks are allowed to have the same priority, they are executed in the order they become ready; this is called round-robin scheduling."

Tools

Some software tools used by real-time programmers include simulators, debuggers, and analysis algorithms. An instruction level simulator now supports most processors. The debugger is usually provided as part of the monitor package, and the simulator will probably have a debugger associated with it. A calculator that has hex, octal, and binary capability is very useful.

Another tool is a dump routine (the Digital Command Language dump, not the Linux dump) that allows one to dump the binary contents of a file. On Windows systems, this can be done with the debug command. To find out more about it, open a command prompt window and enter: *C:>debug/?*. For Linux, a hex editor like KHexEdit can be used.

When comparing binary files or porting from one computer to another, consideration has to be given to the way bytes are ordered within a word. In Big Endian addressing, the address of a data element is the address of the most significant byte, while in Little Endian addressing, the address of the data element is the least significant byte [23]. Everyone agrees that there are eight bits to a byte and four bits to a nibble, but the definition of a word seems to vary. A word is a grouping of bits moved and processed as a unit in computing structure [24]. With that definition, a 16-bit machine has a 16-bit word, a 32-bit machine has a 32-bit word, and on an eight-bit machine, a word and a byte are the same thing.

If all of the task periods are known in advance, a set of algorithms called Rate Monotonic Analysis can be used to predict timing and throughput requirements. Unfortunately, it is not always possible to know the task periods in advance.

Useful hardware tools are a digitized oscilloscope with memory, a logic analyzer, and a counter-timer. These tools can be used to study timing execution of a routine by altering it to set a bit on a port that can be monitored, and then compensating for the time used by the added code. In-circuit emulators can produce timing information, if one is available for the processor.

Conclusion

Real-time programming involves keeping track of time, coordinating tasks, and within limits, making events predictable. This requires an understanding of hardware timing, operating system concepts, and programming skills. Programming skills involve assembly programming as well as a high-level language. As this article has shown, there is more involved in real-time programming than in application programming for a desktop computer running a popular operating system.◆

References

1. Jensen, Douglas E. "Eliminating the Hard/Soft Real-Time Dichotomy." Embedded Systems Programming Oct. 1994: 28.
2. Savitzky, Steven R. Real-Time Microprocessor Systems. New York: Van Nostrand Reinhold, 1985.
3. Obenland, Kevin M. "POSIX in Real Time." Embedded Systems Programming Apr. 2001: 137 <www.embedded.com/2001/0104>.
4. Wood, Mike, and Tom Barrett. "A Real-Time Primer." Embedded Systems Programming Feb. 1990.
5. Savitzky 75.
6. The EE Compendium <http://eecleversoul.com>.
7. Driscoll, Frederick F., et. al. Data Acquisition and Process Control With the M68HC11 Microcontroller. Mac-

COMING EVENTS

November 12-14
2003 Federal Chief Technology Officer Summit
Washington, DC
www.vanheyst.com/CTOSummit/home.htm

December 7-11
Association for Computing Machinery SIGAda Annual International Conference
San Diego, CA
www.acm.org/sigada/conf/sigada2003

December 8-10
Inside ID Identification Solutions Conference
Washington, DC
www.insideid.com/conference.asp

December 9-10
Institute for Defense and Government Advancement SoldierTech2003
Washington, DC
www.idga.org

December 11
Real-Time and Embedded Computing Conference
Seattle, WA
www.rtecc.com/seattle

January 20-22, 2004
Institute for Defense and Government Advancement Network Centric Warfare
Arlington, VA
www.idga.org

January 26-28
Third Annual Conference on the Acquisition of Software-Intensive Systems
Arlington, VA
www.sei.cmu.edu/products/events/acquisition

March 30-31
3rd Annual Southeastern Software Engineering Conference
Huntsville, AL
www.ndia-tvc.org/SESEC

April 19-22
2004 Software Technology Conference

STC

Salt Lake City, UT
www.stc-online.org

Millan Publishers Ltd., 1994: 25.
8. Herzog, James H. Design and Organization of Computer Structures. Franklin Beedle & Assoc., 1996: 576 <www.ee.furg.br/~silviacb/Arq1.html>.
9. Sato, Jin. Jin Sato's Lego Mindstorms: The Master's Technique. Trans. Arnie Rusoff. San Francisco: No Starch Press, 2002: 55.
10. Turley, Jim. "Ten Lies About Microprocessors." Embedded Systems Programming Jul. 2003.
11. Hennessy, John L., David A. Patterson, and David Goldberg. Computer Architecture: A Quantitative Approach. 2nd ed. Burlington, MA: Morgan Kaufmann, 1996: 57.
12. Savitzky 18.
13. Labrosse, Jean J. "Understanding Semaphores." Embedded Systems Programming Oct. 1992.
14. Moritsugu, Steve, et al. Practical UNIX: Contents at a Glance. Que Corporation, 2000: 910.
15. Savitzky 121.
16. Savitzky 9.
17. Spasov, Peter. Microcontroller Technology: The 68HC11. 2nd ed. Englewood Cliffs, NJ: Prentice Hall College Div., 1996: 154.
18. Thompson, Linda M. "Designing With Multiple Processors." Embedded Systems Programming May 1991.
19. Wood, Mike, and Tom Barrett. "A Real-Time Primer." Embedded Systems Programming Feb. 1990: 23.
20. Savitzky 5.
21. Silberschatz, Galvin. Operating System Concepts. Addison Wesley Longman, 1998.
22. Department of Defense Ada Joint Program Office. Ada 95 Quality and Style: Guidelines for Professional Programmers. Herndon, VA: Software Productivity Consortium, Oct. 1995.
23. Hennessy, et. al 74.
24. Herzog 579.

Note

1. Edsger Wybe Dijkstra (1930-2002) is best known for his battle to eliminate the GOTO statement from programming. He also developed an efficient shortest path algorithm and he designed and coded the first Algol 60 compiler. Many of his papers can be found at <www.cs.utestas.edu/users/EWD>.

Additional Reading

1. Clements, Alan. Microprocessor Systems Design. PWS Publishers, 1987.
2. Comer, Douglas. Operating System Design. Englewood Cliffs, NJ: Prentice Hall, 1984.
3. Jones, Steve. "Managing Real-Time Complexity." Embedded Systems Programming Apr. 1992.
4. Sasaki, Stan. "Evaluating Timing Performance." Embedded Systems Programming Oct. 1992.
5. Spasov, Peter. Microcontroller Technology: The 68HC11. 2nd ed. Englewood Cliffs, NJ: Prentice Hall College Div., 1996.
6. VanZandt, Lonnie. "Scheduling Sporadic Events." Embedded Systems Programming Dec. 2002 <www.embedded.com/2002/0212>.
7. White Papers. "Why BlueCat Linux and Real-Time LynxOS?" <www.lynuxworks.com/products/whitepapers.php3>.
8. E. Douglas Jensen's Real-Time for the Real World <www.real-time.org>. (This site has a fun clock to play with as well as much information about real-time computing.)
9. Software Engineering for Real-Time Systems Laboratory <www.enee.umd.edu/serts/bib/index.shtml>.
10. University of North Carolina. "Research in Real-Time Systems at UNC" <www.cs.unc.edu/Research/dirt/real-time.html>.
11. Jim Turley's Silicon Insider <www.jimturley.com>.
12. In-StatMDR. "Microprocessor Report." <www.MDRonline.com>.

About the Author

Dennis Ludwig is a computer engineer at the Simulation and Analysis Facility, Aeronautical Systems Center at Wright-Patterson Air Force Base in Ohio. He has worked with software for more than 20 years. He has a Bachelor of Science in electrical engineering from Louisiana Tech University, a Master of Science Administration from Georgia College, and a Master of Engineering from Mercer University.

ASC/HPEI
2180 8th St.
B145, R 225
WPAFB, OH 45433-7204
Phone: (937) 255-7887
DSN: (785) 255-7887
E-mail: dennis.ludwig@wpafb.af.mil