Real Time systems

Bill Crum
New Mexico Institute of Mining and Technology
New Mexico

April 20, 1996

Abstract

This paper will discuss how a Real Time Operating System can be constructed and how it can be used in an research and development environment. The paper will begin with a description of a Real Time System, implementation of this system on a Linux operating system, design and analysis techniques, and its importance in data acquisition and controls. The paper will conclude with a description of a linux real-time for an application in a research environment.

1 Real Time System

A real time system can be an operating system, an application with a operating system, or an application directly on a machine. The real-time systems are those systems in which the correctness of the system depends not only on the logical results of computations, but also on the time at which the results are produced[1]. These timing constraints are attached to processes (or tasks). In order to guarantee that these constraints can be made, predictability of the system is a requirement.

1.1 Timing constraints

Timing constraints are timing qualities that guarantee execution of a task within defined time limit. These tasks are typically controlling another system such as a robot, or controls in a power plant. In these situations, information about its environment must be consistent with the actual state of the environment. Inconsistent information could otherwise result in catastrophic results. To guarantee correctness of the information requires that the task must complete before a specified deadline. Timing constraints can be either aperiodic or periodic. An aperiodic task has a deadline by which it must start of finish, or it may have a constraint on both. A periodic task is one that is required to start at specific intervals.

1.2 Predictability of a Real Time System

Predictability in a real time system has two basic requirements:

- High performance coupled with consistent execution times of real-time operating system functions
- The provision of primitives with which application programs can run both efficiently as well as perform their operations in a timely and consistent fashion, thereby able to guarantee the applications' timing requirements[4].

By use of task scheduling algorithms that translate complex timing constraints into simple resource utilization constraints, predictability of a real-time system can be verified[2]. For the scheduling of tasks, we must define the types of tasks involved. Critical tasks, or tasks that must be executed within defined time limits, and non-critical tasks, or tasks that are run only when critical tasks are not. There are two types of critical tasks, hard real-time tasks, which cannot miss an execution deadline, and soft real-time tasks, which have execution deadlines also, but still have some value if the deadline is missed.

We can say that an real-time operating system is predictable if it can be proven that all hard and soft tasks can be scheduled to meet their execution deadlines. In order to determine that a task can meet its deadline, a worst case execution time must be determined. This can be done either by direct measurements or by language timing tools[5]. Much research has been done on the use of C as a real-time programming language[6] [7] [5]. In an operating system such as Linux where C is the programming language of choice, use of measurement tools and careful programming to prevent indeterminate loops. In situations where the tasks are statistically scheduled these measurements can be determined with the above constraints. When the tasks dynamically scheduled, such as interrupts, or of different priorities where one task of higher priority can disable another task of lower priority, then other techniques such as rate monatomic scheduling algorithms[2] must be used.

2 A Linux real-time system

The linux operating system is being used for research into real-time operating systems at the New Mexico Institute of Mining Technology. Linux was selected because it is an open system with the source code available. Although Linux is large and slow, and has the inability to preempt kernel mode processes, the solution was to make Linux run as a task under a real-time executive. In this way Linux operating system will not need to be rewritten. The real-time executive runs Linux as its lowest priority task, preempting it when needed regardless of whether Linux is in kernel or user mode.

The real-time executive catches all hardware interrupts passes them to the linux kernel by updating a soft interrupt table in memory. Now the Linux kernel cannot disable the clock. When an interrupt is caught by the executive real-time tasks are scheduled and Linux runs using whatever time is not used by the real-time system. Special lock-free data structures[8] [9] [10] may be used to exchange data with Linux processes.

Preliminary experiments on a Pentium 120 have found that real time processes can run on a 50 microsecond period while Linux is heavily loaded with network and disk transactions. An alternate test of a real time task with a compute time of approximately 40 ms and a scheduled period of 55 ms was done while Linux was running a disk copy program and supporting a terminal display over a network. In this test, Linux continued to operate, although with some decreased response times[3].

3 Real Time Operating System in a research Environment

On the New Mexico Institute of Technology campus a research facility, Energetic Materials Research and Testing Facility (EMRTC) is doing research on various aspects of energetic materials. The research requires the use of a program to collect data from remote sites, and requires in some instances real-time control of instrumentation. EMRTC is currently using iX86 cpus, windows operating systems, and Labview software to provide the software requirements of their real-time control and analysis of the tests.

One test is the development of liquid rocket boosters. The use of Labview software on a windows operating system lacks absolute guarantees on timing and data acquisition. This is proven as windows will return to a dos prompt during some tests. When this happens control of the system is lost and data collected that was not stored to secondary memory is also lost. A system to guarantee the collection of data and control of the system must be assured in order to avoid catastrophic events.

Another test is the cook-offs. This is where munitions are heated at various speeds and monitored to evaluate the effects of temperature changes to the munitions. Although controls during these tests are not required, valid data collection must be made. Communications over networks are required for these tests as they are done remotely. Reliable communications is needed in order to change data collection parameters during various stages of the tests, such as higher frequency of readings when temperatures reach various stages.

Our proposal is to continue using their current hardware, including the iX86 architecture, but replace the controlling software with an operating system with real-time constraints. We propose the use of real-time Linux described earlier, currently in development. This would provide a real-time programming environment as well as development of tools to insure real-time constraints. RT Linux would provide for the strict timing requirements of the real-time tasks in the research, while also providing all the flexibility of a UNIX type open system environment for things such as soft-real-time tasks, concurrent displays, post-run analysis, and networking.

3.1 The hard real-time tasks

Hard real-time tasks as shown earlier are defined to be any task that must be performed by a specified time or data will be lost. In the case where controls are required to be done in real time, catastrophic results can occur when a hard real-time task misses a deadline. For the rocket project, the hard real-time tasks are startup, shutdown, control, and data acquisition. We shall use the rocket development as an illustration of how it can be implemented on RT Linux.

4 Rocket Test Research

During a rocket test firing three tasks are required to run concurrently throughout the test. Two of these tasks have hard real-time requirements for the test. The first task is responsible for the valid sequence of events in controlling valves to fire the rocket. This task is also responsible for correctly aborting the test in real time. Another task is responsible for the valid collection of data throughout the test in real time. This is also a strict requirement during aborts, as data must be archived for later evaluation of the test. A third task is required to update various displays during the test. These displays must be updated in a timely manner.

Two of these tasks must be done at very specific times. The hard real-time tasks in this system will be highest priority tasks which is guaranteed to run at a predetermined interval and will not be blocked. Both of these tasks are very straight forward as to their function, and measurement techniques described earlier can be used to validate predictability.

4.1 Control

The control subtask is not fully defined yet. Eventually, startup, shutdown, throttling, guidance, and error handling will be done by the rocket. Currently, control will be in the form of startup, shutdown/abort and direct user input. We envision control to make use of a priority command queue emulating a state machine. According to the specifications given for startup the following sequence of events are expected to happen:

- At 31 seconds prior to ignition, the Liquid Oxygen(LOX) is pressurized.
- At 21 seconds, the fuel is pressurized.
- At 14 seconds, the LOX is liquid oxygen lines are cooled. This process last two seconds, and consists of opening and closing the LOX chill down valve.
- At 7 seconds, the ignitor gets activated. The ignitor must reach a specified temperature before the fuel will ignite.
- At zero seconds, the temperature of the ignitor must be tested. After this test is completed, the program must either escape to the shutdown sequence or go to the firing sequence. The firing sequence consists of opening the liquid oxygen valve immediately after the test of the ignitor.
- At .1 second after opening of LOX valve (critical for accurate data) the fuel valve must be opened. The opening of these valves must be very exact in respect to each other.

During rest of test, if a shutdown/abort sequence is required, the following steps must be taken:

- Liquid oxygen valve is first closed.
- A tenth of a second later, the fuel valve is closed.
- Four seconds later the remaining liquid oxygen is vented out of the system.
- Eight seconds later the fuel is vented out of the system.
- Twelve seconds later, the purge valve is opened and the remaining fuel and oxygen is vented out of the system. The closing of the purge valve is not specified and thus we will assume the process is done by human control in the system currently.

This shutdown sequence is expected to be able to run at any time from any point in the test. In the future, part of control will be the handling of errors. Such errors would include the temperature getting too high, a burn through of the rocket, etc. At this point in the project, unusual data brings up an error window giving the operator the option to either shutdown the test or continue. Most control at this point is done by an operator and timings can be done is soft real-time due to human reaction time.

4.2 Data Acquisition

The second subtask that most occur is data acquisition. Currently data is being collected on twenty channels. The hardware can handle at most 512 channels. However to be practical, the number of channels in our model is limited to 64. According to the specifications, each channel will be read at a rate of KHZ. The rate at which the channels are read are programmed into the board. Each channel is twelve bits. However, because the PC stores data in memory in 8 bit units, the twelve bit pieces of data are treated as sixteen bits. This gives a data rate of 128K bytes/sec. The board provides a 4k equivalent FIFO. Thus, the buffer must be emptied at a minimum of 32 times a second before any data will be lost. By reading at a quicker rate, error checking can be done to see if the buffer is getting read quick enough by looking at the amount of space left on the buffer. If the buffer is full then the computer has not kept up. This can be used for verification of predictability during development.

We estimate the following timing necessities for the DAQ portion of the process based on Intel timing specifications:

- read from controller-FIFO assuming 16-bit data paths
- 17 clock cycles * 4k/2 = 34,816 clock cycles
- write to in-memory-FIFO assuming 16-bit data paths
- 1 clock cycle * 4k/2 = 2,048 clock cycles
- Data collection = 36,864 clock cycles
- 486dx 33Mhz = .00112 seconds to collect data
- Maximum 64 channels @ 1khz sampling = 64,000 samples per second
- 2k sample FIFO needs to be read once every 30 mSec (.03125 S).
- Data collection takes .03575 seconds to collect every second.
- (32 * .00112)

Calculations for the in memory fifo based on our calculations show the following:

64 channels at khz = 128,000 bytes for 1 sec (2bytes * khz * 64 points).

This means it is not unreasonable to save data to memory and let a soft real-time process deal with moving it to disk. A 200 second test will take 25,600,000 bytes of memory for 64 channels at 1khz sampling.

Data Acquisition is done from the very beginning of the startup sequence to the end of the shutdown sequence. This process must be ran on a periodic basis no matter what other processes are being ran.

4.3 Soft Real-Time Tasks

The soft real time processes initially consists of data display and writing the data to disk. The display is used to carry out user control functions and give the operators an idea what is happening with the rocket test.

The display consists of a window showing the data. This window is currently being updated at 10Hz. However, the display is not critical because a person cannot react very fast. Thus, the display is a soft real-time process of low priority.

Currently the data is being stored to ram-disk. But, the data needs to be put on a hard drive eventually. To help speed up this process, instead of letting the computer be idle, use this time to write to disk. By writing to disk, less ram would be necessary, and the tests could last longer. However, as of yet writing all the data to the hard drive takes too long. By treating the ram disk as a FIFO, the data then could be sent to the hard drive when time permits. Once the test is over, the remaining data could be transferred to the hard drive. From the hard drive, the data could be handled in what ever way considered necessary.

5 Cook-offs

In the cook-off experiments, the hard real-time tasks consist of the valid collection of data from the site. This includes the movement of the data to secondary memory. Another would be the proper communication over a network to a remote site.

This can be done with a data collection task like the one described in the rocket test that is scheduled at highest priority to read in the data. This task can then store the data at the appropriate frequency defined by a memory location that is updated by the communication task.

The communication task will be hard real-time to accommodate update changes in data collection frequency and move data into network queues to be transmitted to a remote site by a soft real-time task.

This soft real-time communication task will use whatever CPU time is left to store data to hard disk, and move data over the network.

6 Conclusion

The overall premise is to use a Linux real time system with two hard real-time task. Our base, worst case platform, a 486dx 33 Mhz with 32Mbyte of ram, 1gig hard drive, AT-MIO-16E-2 DAQ card and some Linux supported SVGA terminal. With this hardware we have shown that data collection will be only 4 percent of the CPU processing time.

The Control/DAQ tasks are the only ones that is run in hard real-time. It is not dependent on any other tasks running so they cannot be locked out. All the soft real-time tasks can be run as lower priority, regular Linux tasks. Display routine will only read from DAQ data and cannot lock out above task from running.

We can verify this test by simulation of input data (varying voltages on the inputs), controlling relays to LED lamps, with feedback into inputs. A switch can simulate an abort condition during test. A logic analyzer can verify valve changes are made at correct intervals, and then this project can be verified by installing linux system in parallel with system in place. Controls can be routed to relays (LED's) not on rocket, but input Data can be collected in parallel with other system without interference to rocket.

References

[1] John A. Stankovic, Krithi Ramamritham, *What is predictability for Real-Time Systems?*, J. Real-Time Systems, Vol. 2, December 1990

[2] Lui Sha, Ragunathan Rajkumar, John Lehoczky, Drithi Ramamritham, *Mode Change Protocols for Priority-Driven Preemptive Scheduling*,J. Real-Time Systems, Vol 1, 1989

[3] Victor Yodaiken, *Cheap Operating Systems Research and Teaching with Linux*, 1995

[4] Kaushik Ghosh, Bodhisattwa Mukherjee, Karsten Schwan, *A Survey of Real-Time Operating Systems - Draft*, Georgia Institute of Technology, GIT-CC-93/18, 1994

[5] Chang Y. Park, Alan C. Shaw, *Experiments with a Program Timing Tool Based on Source-Level Timing Schema*, IEEE Computer, Vol. 24, No. 5, May 1991

[6] N. Gehani, K. Ramamritham, *Real-Time Concurrent C: A Language for Programming Dynamic Real-Time Systems*, J. Real-Time Systems, Vol. 3, No. 4, December 1991

[7] Y. Ishikawa, H. Tokuda, and C. W. Mercer, *Object-Oriented Real-Time Language Design: Constructs for Timing Constraints*, ECOOP/OOPSLA '90, Oct 1990

[8] M. P. Herlihy, *Wait-free synchronization*, ACM Transactions on Programming Languages, 13:124-149, 1991

[9] H. Massalin, C. Pu, *Threads and input/output in the synthesis kernel*, Proc. Twelfth ACM Symp. on Operating Sys., Operating Systems Review, Page 191, December 1989

[10] H. Massalin, C. Pu, *A lock-free multiprocessor OS kernel*, Technical Report CUCS-005-91, Columbia University, 1991