TEACHING REAL-TIME CONCEPTS IN AN UNDERGRADUATE COMPUTER SCIENCE PROGRAM

Ron Schroeder¹ and John Rupf²

Abstract - Traditionally, courses in real-time programming and related real-time systems concepts have not played a major role in computer science curricula. However, this somewhat neglected area of computer science has much to offer students, especially when presented late in the curriculum. In this paper we discuss the rationale for including a real-time systems course as a component in a computer science curriculum. We then describe the structure, organization, and content of the course “Real-Time Systems” that we have taught for the past several years as an upper-level elective to computer science students. In this course, model trains are used as the physical processes that are monitored and controlled by software that students develop using a modern suite of commercially available development tools.

Index Terms – Real-Time Systems, Embedded Systems

INTRODUCTION

Computer systems that monitor and/or control physical processes are often referred to as "real-time" in nature. They are characterized as such because of the necessity that they keep up with time constraints imposed by the physical processes or systems with which they are interacting. Real-time computing is often a critical component of embedded systems - those systems whose proper functioning depends on computers contained within them. Traditionally, courses in real-time programming and related real-time systems concepts have not played a major role in computer science curricula.

Teaching the concepts of real-time programming can be abstract and even superficial, especially if there is not a way to clearly demonstrate the need for time constraints and event oriented processing. Unless students can experience the consequences of unmet time constraints and realize that simple, single threaded programs can become complicated and convoluted when multiple events occur at different rates, they will fail to appreciate the need for a different approach to software development. It is the authors’ opinion that students substantially benefit from the experience of interacting with a real, physical system that responds directly to the commands generated by their software; students really need to “see” what happens when incorrect commands are sent or when correct commands are sent but arrive too late to avoid a physical mishap.

This paper describes the approach used to teach real-time concepts in the undergraduate computer science program at Southern Polytechnic State University. It discusses the structure, organization, and content of the course “Real-Time Systems”, taught for the past sixteen years as an upper-level elective to computer science students. In this course, model trains are used as the physical processes that are monitored and controlled by software that students develop using a modern suite of commercially available development tools.

WHY TEACH REAL-TIME CONCEPTS IN COMPUTER SCIENCE?

One observation that we have made of many computer science majors, at least at our university, is that they often perceive the study of computer science as primarily software development and programming. Subjects such as digital systems and computer architecture are viewed as little more than abstractions that are to be understood conceptually and used only in classroom exercises. Computer science students typically are never required to actually build or control a physical system and make it work. Consequently, they often lack a full appreciation of the affect that the underlying hardware environment, especially modern ones that make extensive use of caches and pipelines, has on the performance of their software. It is tempting for students to focus on logically correct behavior, but give little or no attention to performance constraints. How to guarantee that a program will complete in 50ms rather than in 50 seconds is often a complete mystery to them.

Most computer science curricula include a course on operating systems. Many such courses involve exercises in which students write operating system components such as command processors, schedulers, and memory managers (or at least simulations of these). There is seldom time in a first course to require students to write software in which they have to use various API functions provided by an operating system. However, this can be a very enlightening experience for them. Consider the challenge of writing a monitor-based implementation of the readers/writers problem in which the students must use semaphores, via appropriate API function calls, because the operating system API does not directly provide a monitor construct. Just having the chance to debug such concurrent software can be a new and enlightening experience for the students.

¹ Ron Schroeder, Southern Polytechnic State University, rschroed@spsu.edu
² John Rupf, Southern Polytechnic State University, jrupf@spsu.edu

0-7803-7961-6/03/$17.00 © 2003 IEEE

November 5-8, 2003, Boulder, CO

33rd ASEE/IEEE Frontiers in Education Conference

F3C-12
It is our view that a course in real-time systems and software can provide students this experience and more. Since computer science programs typically require operating systems and computer architecture courses, a real-time course provides a logical continuation of these subject areas. Also, because of the anticipated expansion of the market for embedded systems and associated software, a real-time systems course may serve to enhance a student’s employment opportunities.

**THE FOCUS COURSE – REAL-TIME SYSTEMS**

In this section we describe the real-time systems course in its current state of evolution. Future plans are discussed in a later section.

**Current Course Description**

The primary objective of this course is to promote an appreciation for and understanding of real-time computer systems and related software. Considerable emphasis is placed upon the added dimension of performance, especially as it relates to the responsiveness of systems to external stimuli. Such systems are typically characterized by a requirement to meet rigid processing deadlines. The software is typically event-oriented and is often structured as a collection of asynchronous, concurrent processes.

Topics emphasized in the course include:

- real-time system characteristics (hardware and software)
- design and implementation of real-time programs and their execution under the control of a real-time operating system
- real-time programming languages
- real-time system specification and design techniques
- performance characterization of real-time systems
- system integration

The current textbook used is *Real-Time Systems Design and Analysis*, by Phillip A. Laplante.[1] It represents a good survey of the subject and serves as a good outline of many of the topics covered in the course; selected topics are expanded by way of instructor provided handouts. Not all chapters in the book are covered in the course; those that are covered are not presented in the same order as that of the book.

Required lab assignments, to be described later, utilize specialized equipment and software available in the school’s labs.

The course is a 4-credit, semester long course consisting of classroom lectures and “open lab” assignments. Limited class time is spent on demonstrations of the equipment and software development tools that students use in the lab; no class time is scheduled for students to work on their lab assignments. Our rationale for this structure is that lab activities in the real-time course are not easily partitioned into small “mini” labs that can be distributed in uniform time blocks throughout an entire semester.

**Prerequisite Topics**

In order to adequately cover the topic areas listed previously, students need to have experience programming in a contemporary language. The software development environment employed in the course uses the GNU C/C++ compiler, so students must be prepared to program in either C or C++. Although our computer science majors begin their studies using Java, they transition to C++ in the data structures course.[2] These prerequisite courses adequately prepare the students for programming in either C++ or C.

Students must also understand computer organization and how it affects the execution of their software. All our majors complete a required two-course sequence in digital design and computer architecture. The digital design course topics include combinational and sequential logic circuit design and number representation. [3] A following required course in computer architecture covers instruction set architectures, assembly language, and the design of single-cycle, multicycle, and pipelined processors. [4]

In many ways, the real-time systems course is an extension of the operating systems course required of all our majors. In the operating systems course, students learn the basic concepts underlying modern operating systems, such as process management (scheduling, synchronization, and inter-process communications), memory management (real and virtual), and file management. [5] Although the students write selected pieces of operating systems (or simulations of them), they do not have the time to study a particular API and use its provided function calls to implement concurrent programs that consist of multiple processes that must work together to implement complex functions. Since the real-time systems course requires students to use a particular operating system API to develop such software, it can be viewed as an extension of the operating systems course.

Although our lab assignments require the use of C or C++, other real-time programming languages are also discussed in the course. In order to benefit from this discussion the students must understand what attributes make a given language suitable or not suitable for real-time software. Therefore, students in the real-time course need to understand language attributes and various approaches to their implementation. Since our computer science majors must complete a course in programming language concepts, that course is recommended prior to the real-time course.

The software life cycle as it applies to real-time systems is also discussed in the real-time course. Modeling techniques such as finite state machines and Petri nets used to define requirements as well as designs are also covered. These discussions build upon concepts previously covered in a software engineering course required of all majors and recommended as a prerequisite for the real-time systems course.
In summary, the following courses are required prerequisites to our real-time course: data structures (follows two introductory programming courses), digital design, computer architecture, and operating systems. The introduction to software engineering and programming language concepts courses are recommended prerequisites.

**Content Areas**

This section describes the content areas covered during the real-time course. Although our course can probably be classified as a survey of real-time systems, a major emphasis is placed on programming real-time software.

During the introduction to the course, students are exposed early on to basic terminology and concepts. Emphasis is placed on characterizing real-time systems and software as “time critical” - meaning that such software has deadlines to meet. Real-time systems are categorized as “hard” versus “soft”; hard real-time systems have no tolerance for missed deadlines. The notion of “event driven” processes is emphasized. Several examples of systems consisting of multiple events occurring at different rates and having specific deadlines associated with them are presented, in some cases by asking the students to read articles from the professional literature. Examples of real-time systems that have caused catastrophic failures seem to be particularly interesting to the students.[6,7] The role of process scheduling in helping processes meet their deadlines is explained. Performance and its characterization is emphasized; determinism is defined and then associated with the software making up real-time systems.

Since determinism is a critical operational criterion, a short review of computer architecture emphasizing those aspects that directly affect the performance of real-time software is presented. For example, the effect of pipelines on interrupt latency is addressed. Students are asked to propose alternatives for structuring systems without heavy reliance on interrupts. Those who are well grounded in computer architecture and who are reasonably creative often propose viable solutions, such as multi-processor systems in which each processor is dedicated to one event and, following each event, communicates with other processors by way of a message passing scheme over a bus or network structure.

As early as possible following the previously described introductory discussions, emphasis is shifted to a review of operating systems. This is immediately followed by a summary of common software structures for real-time scheduling. Simple polling structures are described and their performance limitations identified. Purely interrupt driven systems are contrasted with foreground/background systems; simple performance computations for these are illustrated. Other structures such as state-driven, round robin, and coroutine structures are briefly described and their performance issues identified. Rate monotonic scheduling is presented along with the problem of priority inversion and its solution, priority inheritance. The final and greatest emphasis is placed on real-time operating systems that provided priority preemptive scheduling along with API functions, such as those for handling semaphores and inter-process communication. Once real-time operating systems are introduced in a generic way, the specific operating system that students will use in their lab activities is introduced - VxWorks. VxWorks is contained within an integrated development environment (IDE) called Tornado and is marketed by Wind River Systems, Inc.

Once students are familiar with VxWorks and its API as well as the other tools (debuggers, profilers, target simulators, etc) provided in the IDE, they are prepared to begin their outside lab assignments. After this point in the course, students work on their lab assignments concurrently with the coverage of other topics during class time. Initially, students are assigned a “get acquainted” lab such as one requiring the implementation of a producer/consumer system using semaphores and/or message queues. The second lab is more challenging but still involves concepts they are familiar with from their operating systems course. One such example, mentioned previously, is a lab in which they are required to implement a readers/writers system using a monitor constructed with semaphores. Another example is a pipeline of separate processes, separated by double buffers; all synchronization between processes is handled with semaphores. Since the Tornado IDE contains a target simulator, students are able to develop and test their first two labs entirely within the IDE on a single computer.

After students have become comfortable with the VxWorks environment and its use in solving familiar problems, they are introduced to the physical system that their software will ultimately have to monitor and control - a model train layout. The idea of using a model train to teach real-time programming was first introduced by John McCormick [8,9]. The trains described by McCormick used a traditional 2-rail track and DC voltage to power the trains.

We employ a digital model train system manufactured by Marklin of Germany. It uses a 3-rail track (the center rail is actually a metal stub attached to each cross tie) and AC voltage to power the trains. The train layout is connected by way of a serial interface to a PC. Digital commands are transmitted from the computer to the railway, where they are separately decoded by each locomotive and track switch. Only the specific locomotive or switch to which a command is addressed will respond to such commands. Commands are used to set locomotive throttle levels, change direction of travel, turn headlights on and off, and set switches to their proper positions. Using a 3rail track makes it possible to electrically isolate sections of track and thus makes train detection relatively easy. As a train with metal wheels and axles moves into a section of track, it creates a contact closure that sets a bit in a special register; this register can subsequently be read by the PC software.

The PC that executes the train-control software is connected by way of a network to a “host” PC. The host PC...
contains the previously described Tornado IDE software that students use to edit, compile, and debug their train-control software. Once students have successfully produced an executing version of their software on the host PC, they download it to a “target” PC that is connected to the digital train layout for final testing. Figure 1 illustrates the physical configuration of the lab. There are four host/target pairs, all interconnected by way of an Ethernet network. A serial port from each target computer connects to a four-way switch, thus allowing students to manually connect their target computer to the trains when they are ready to begin testing.

![Figure 1. Laboratory Configuration](image)

Programming languages for real-time software development are surveyed next. Language attributes such as parameter passing techniques and their effects on performance are presented and discussed with examples. Strong versus weak typing is discussed with examples of what can go wrong when automatic type conversion occurs (truncation and round off errors, etc). Performance issues related to object oriented features such as polymorphism and multiple inheritance are likewise discussed. Specific languages such as C, C++, Ada, and Java are compared along with their appropriateness or inappropriateness for various applications.

The discussion of system integration usually proves to be quite timely, as it occurs when the students are fully immersed in the development and testing of their train control software. At this point, they are beginning to deal with the complexities of using the host computer to compile code that is then downloaded to a target computer that in turn connects to the train layout hardware. Their software typically consists of multiple processes, all interacting in some manner with physical trains that move in accordance with the commands issued by the software and exhibit timing constraints that must be met by the software. Students soon realize that software errors often result in physical “catastrophes” in the form of train collisions. The discussion of system integration at this point helps students better organize their often heretofore haphazard approach to getting their software to work correctly.

The final lecture topic is performance and its characterization and analysis. Students are by now convinced of the need to design software that meets performance requirements; the next issue is how to do it in an organized way. A high-level look at performance modeling techniques covers both analytical and simulation models. Simple analytical models for determining interrupt latency and processor loading produced by collections of periodic processes are discussed. Simulation is presented as the method of choice for systems that are very complex, perhaps consisting of large numbers of asynchronous, concurrent processes competing for shared resources and perhaps executing on multiple processors interconnected by complex networks. Determining system bottlenecks is an interesting objective that students see firsthand; they are given simulation models of such systems and are asked to make changes that require the re-distribution of software processes to different processors in order to assure that system-wide response times and loading requirements are met. The simulation tool employed is NETWORK II.5, an easy-to-learn tool with a relatively short learning curve. It is a product of CACI and is written in SIMSCRIPT.

**FUTURE PLANS**

The real-time course has existed in its present form for several years. The lab that supports the course has existed for almost as long, although the equipment has evolved...
through several stages as new products exploiting new technologies have emerged. The train layout is now fully wired and tested (often with the help of students who noted cases in which the equipment did not always function as advertised).

Our lab currently has only one train layout, which seriously limits the number of students that can be accommodated in the real-time course; that number is currently 20 students. Allowing students to work in teams lessens the problem somewhat, at least until the final few weeks of the term when it proves to be a major bottleneck. It has become apparent that a second train layout is needed if the course is to accommodate more students or even the current number without the usual end-of-term “crunch”. When the funds become available, we plan to proceed immediately with the construction of a second layout.

A new area of activity being planned is that of robotics. The recent availability of inexpensive robots and accessories such as Lego Mindstorms has proven to be an attractive and timely incentive. Furthermore, we have concluded that the equipment and software tools needed to develop robot control software should be very similar to that already available in the lab. How to incorporate a robotics component into the real-time systems course is still being considered.

REFERENCES