Use of FreeRTOS in Teaching Real-time Embedded Systems Design Course

Dr. Nannan He, Minnesota State University, Mankato

Nannan He received the Ph.D. in computer engineering from Virginia Tech. From 2012 to present she is an Assistant Professor at the ECET department in Minnesota State University at Mankato. Her teaching and research interests are in safety-critical embedded software, real-time embedded systems, and software verification. She is an IEEE member and reviewers for many conferences and journals in EDA field.

Dr. Han-Way Huang, Minnesota State University, Mankato

Ph.D. in computer engineering from Iowa State University Professor of ECET Department of Minnesota State University, Mankato

Use of FreeRTOS in Teaching Real-time Embedded Systems Design Course

Abstract

This paper presents our experiences of teaching the course "Real-time Embedded Systems Design" by applying the free and open source Real-Time Operating System (RTOS) called FreeRTOS. The emphasis is placed on how we adopted FreeRTOS as a real-world RTOS example in both lectures and lab sessions from exercises preparation, lab equipment setup to lab organization. Compared with existing real-time computing courses, the main difference of this course is that we focus on teaching students the design and application development of real-time embedded systems from the practitioner's point of view, instead of introducing research or theoretical topics. FreeRTOS is a real-time kernel/scheduler designed to run on a microcontroller for embedded applications. It supports a large number of underlying microcontroller architectures and has become the leading real-time computing platform for microcontrollers. In this course, it has been applied to conducting experiments with multitask scheduling algorithms and the real-time interfacing with microcontrollers for all our lab sessions and course projects. Our primary experiences indicate that FreeRTOS is a richly featured, cost-efficient and well supported RTOS for teaching real-time systems design and developing microcontroller-based real-time applications.

Introduction

Nowadays, with the emergence of new processors and methods of processing, communications and infrastructure, modern industrial automation systems require high real-time control capabilities. There is an on-going work to achieve the education goal of increasing the technical depth and broaden training by investigating deterministic timing techniques in complex real-time automation systems at Minnesota State University, Mankato. As an important exploration step towards this goal, a new real-time embedded system design course has been offered to electrical engineering and computer engineering senior or graduate level students. At the same time, the goal is also an important guideline in the course preparation and teaching practices, as a result some special features of this course are formed compared with most existing real-time systems design courses.

Real-time computing courses are mostly offered to computer science major students at the senior or graduate level, with the aim of equipping them with the knowledge of conducting the scientific research in this area. These courses typically focus on the theoretic topics in real-time computing, such as various reference models of real-time systems, schedulability theory, design and timing analysis of multi-task scheduling algorithms and operating systems. However, the

real-time embedded systems design course presented in this paper emphasizes engineering issues of designing and developing real-time systems in practical embedded applications like automation. The course is taken by senior electrical engineering and computer engineering students and some graduate students pursuing their Master degree. Other engineering students with the appropriate software background could also take the course. It emphasizes teaching students real-time systems design and applications from the practitioner's point of view, instead of targeting at research and theoretical topics as conventional real-time computing courses. After taking this course, students are expected to demonstrate the ability of correctly defining and designing real-time systems, and the ability of basic real-time application development. Active learning and hands-on learning are the fundamental teaching approaches applied to this real-time systems design course.

This new course covers the topics on RTOS relevant topics, such as multi-task scheduling, system services, and resource policies and some application issues for developing real-time systems, such as microcontroller, requirement analysis, performance analysis and verification of real-time system design. Among these topics, RTOS is one of the core components of our new course. In this course, we employed an existing free, open source Real-time Operating System called FreeRTOS as a case study of RTOS in both lectures and lab sessions.

FreeRTOS is a real-time kernel/scheduler designed to be small enough to run on a microcontroller. It provides the real time scheduling functionality, inter-task communication, timing analysis and synchronization primitives for teaching RTOS. It also offers the rich example projects as the bases for developing embedded real-time systems. Moreover, FreeRTOS supports a large number of underlying microcontroller architectures including advanced ARM Cortex^{TM}-Mx series, and has become the standard RTOS for microcontrollers. To simplify the structure of the application code, The FreeRTOS software provides time-related Application Programming Interfaces. As a result, complex embedded real-time applications can be efficiently built to meet their real-time processing deadlines on top of FreeRTOS. In this course, FreeRTOS was applied to conducting experiments with multitask scheduling algorithms and real-time interfacing with microcontrollers for all our lab sessions and course projects.

This paper presents the primary experiences of teaching real-time embedded systems design to engineering students, with the emphasis of how we adopted FreeRTOS as a real-world RTOS example in teaching to improve the teaching effectiveness. The description of this course is first given, including course contents, learning outcomes and instruction approach. Next, a survey of existing real-time operating/runtime systems is reported. Then, this paper describes the software FreeRTOS and how we make use of FreeRTOS in lab assignments and course projects from exercises preparation, software setup and implementation. Finally, the paper gives a conclusion and discusses the future work.

2. Real-time embedded systems design course description

Our real-time embedded systems design course targets the learning of real-time systems design and applications from the practitioner's point of view. It has been offered for two years. This course is organized as two hours of lecture and three hours of laboratory per week. It has three main objectives.

- To improve students' awareness of real-time specifications in critical automation controllers and other embedded systems.
- For engineering students to apply modern development tools and advanced techniques to designing and analyzing performance of small-scale real-time systems.
- To enable students to develop real-time applications to solve problems with specific timing requirements.

Moreover, in order to accomplish the basic instruction approach of active learning and hands-on learning, this course has an experiential component. It allows students to apply advanced techniques learnt from this course to develop an understanding of their advantages and disadvantages in different applications.

Course learning outcomes

Our overall aim is to equip students with the knowledge of designing real-time systems and developing real-time applications to solve engineering problems in practice. In the development of this course, we identify course learning outcomes that stem from this aim and extend to learning activities. We developed twelve learning outcomes, classified into three core components as shown in Figure 1.

1. To demonstrate the ability of correctly defining real-time systems
   1.1 To identify problems as hard, firm or soft real-time system;
   1.2 To articulate and contrast different definitions in real-time systems.
2. To demonstrate the ability of real-time systems design
   2.1 To comprehend formal methods based specification approaches and utilize modeling tools;
   2.2 To understand the impact of hardware for real-time performance;
   2.3 To analyze the scheduling feasibility of a set of independent tasks and derive schedules;
   2.4 To understand resource policies and system services for inter tasks communication and synchronization;
   2.5 To understand the challenges and applications of performance analysis techniques;

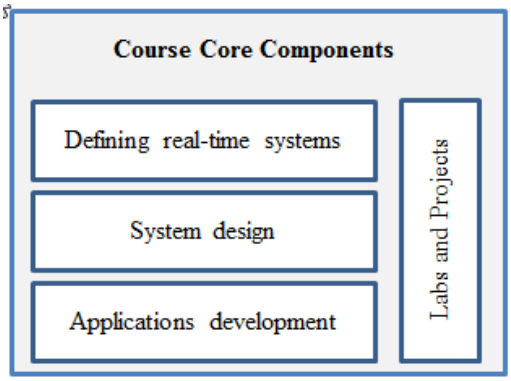

Figure 1. The course is designed based on three core components.

2.6 To understand real-time issues on advanced distributed control networks such as SCADA;

3. To demonstrate the ability of basic real-time application development
   3.1 To understand real-time software testing, verification and system integration.
   3.2 To be aware of performance optimization techniques.
   3.3 To utilize modern tools to simulate executions and critique different implementation choices.
   3.4 To comprehend the architecture, functions and applications of one or two existing RTOS.

Course contents

The topics covered in this course include real-time scheduling approaches such as clock-driven scheduling and static and dynamic priority driven scheduling, resource handling, timing analysis, RTOS, hard and soft real-time systems, distributed real-time systems, concepts involved in the modeling, design, analysis and verification of real-time systems. Course materials were drawn from two text books [1, 2], FreeRTOS tutorial book and reference manual, ARM Cortex-M microcontrollers' datasheets, websites, and other publications. Table 1 shows the classification of these topics. In this table, the time schedule of each top-level topic is given for this one semester course (~ 15 weeks). Please note that the topic 7 - *Case studies (FreeRTOS)* is not labeled with a specific schedule because its sub topics are provided in the combination with other topics throughout the semester.

Table 1. Course topics

1. Fundamentals (week 1)	Requirement Eng. for real-time systems
Basic concepts and misconceptions	(semi-) Formal methods in system specification
Multidisciplinary design challenges	Real-time kernel Implementation in FreeRTOS
2. Hardware for real-time systems (2^{nd}-3^{rd})	**5. Performance analysis techniques (10^{th} – 12^{th})**
Process architecture	Timing estimation of real-time system
Architectural advancements	Queue theory applications
Peripheral interfacing	Input/output performance
Distributed real-time architecture	**6. Additional application issues (13^{th} -14^{th})**
3. Real-time operating systems (3^{rd} - 6^{th})	Design for fault tolerance
Multi-task scheduling	Software verification & system integration
System services for application programs	Performance optimization
RTOS selection issues	**7. Case studies: RTOS in practices**
4. Requirement engineering (6^{th} – 8^{th})	FreeRTOS (open source)

Instruction approach

Active learning and hands-on learning are fundamental teaching approaches applied to this real-time system design course. All of the classes were held in the laboratory. For this course, this setting eases the flexible adoption of a variety of teaching methods, depending on the characteristics of different course topics in sequence. The main teaching formats and material employed in this course are presented as the following.

At the beginning, we used power point slides presentation and class discussion to introduce students the topics on defining real-time systems. These topics are basis for further learning. Thus, it is important to help students to set up a solid and comprehensive foundation. In the class discussion, some questions are designed to enable students to reflect on key concepts in real-time systems, and to encourage active learning. Here are some examples: 1) Are real-time systems synonymous with '*fast or high performance*' systems?, 2) "In the statement 'All practical systems are ultimately real-time systems', what is your idea of the *degree of 'real-time*'?, 3) "Where could a *response time* requirement of a system come from?". As the discussions proceed, students gradually deepen their comprehension at the same time are inspired to judiciously observe and analyze real-time applications. The initial homework exercises in this course are extracted from practical application problems. One example question is: considering an automation system for the car assembly, describe three different event scenarios, classify the system as hard, firm or soft real-time under each of these scenarios. Such exercises are designed with open-end questions. The goal is for students to think and give the justification for their answers, and to fortify students' understanding so as to be able to apply it in practice.

The main purpose of introducing formal requirement engineering techniques is for students to develop an appreciation of automated formal or semi-formal methods in rigorously specifying real-time system requirements. An example sub-system requirements document for "Fuel Management" from AIRBUS was introduced as a case study. Students are convinced that the formal specification like State chart is not just a scientific research issue, in fact has been widely used in requirement specification of safety-critical embedded systems in industry to avoid the ambiguity caused by conventional text-based specifications. Later on, they showed great enthusiasm in learning formal or semi-formal approaches. In the course evaluation, students made several comments that the requirement engineering offered in this course was one of the most beneficial aspects to them.

There is the increasing number of microcontrollers (MCUs) supporting real-time applications. In this course, the following MCU development boards and Integrated Development Environment (IDEs) are employed to be available for students.

- PIC24, dcPIC (Explorer 16 Development board from Microchip)
- ATmel SAM4S-EK (ARM Cortex-M4 microcontroller from Atmel)
- ATmel SAM4L-EK (ARM Cortex-M4 microcontroller for low power from Atmel)

- µVision IDE for ARM programming from Keil
- Atmel Studio with Atmel Software Framework (ASF) from Atmel
- MPLAB IDE from Microchip

The above MCUs are selected as they are all supported by the FreeRTOS software which is the main RTOS studied in this course.

The last part of this course offers students the topics which are served for the application development core component. As real-time systems are often applied in the 'critical' embedded applications with respect to reliability, safety and security, verification and validation (V&V) is an important issue in real-time application development. An on-going research on the model-based V&V is incorporated in this course so that students could be exposed with latest V&V advances.

Table 2. A survey of RTOSs

Name	License	Platforms	Description
FreeRTOS	Free	ARM, IA32, Cortex-M3, PIC, MSP430, STM32	A real-time kernel designed to be small enough run on a MCU, be potable to a large number of MCUs. http://www.freertos.org
CooCox CoOS	BSD	ARM, STM32, NXP LPC1000, TI LM3s8963,	An embedded real-time multi-task OS special for ARM Cortex M series. http://www.coocox.org/CoOS.htm
QNX	Mixed	ARM, IA32, MIPS, PowerPC, SH-4, xScale	A commercial Unix-like RTOS, targeting at general purpose usage, being widely used with in a variety of devices including cars and mobile phones. http://www.qnx.com/
Window CE or WinCE	Proprietary	ARM, MIPS, x86, SuperH	An OS and kernel developed by Microsoft for embedded systems; a variety of IDE supporting development for WinCE http://www.microsoft.com/windowsembedded/en-us/windows-embedded-compact-2013.aspx
scmRTOS	Free	ARM, Cortex-M3, MSP 430, AVR	A free tiny preemptive RTOS intended for use with Single-Chip MCUs. The key features are max speed and min code/RAM size requirements. http://scmrtos.sourceforge.net/ScmRTOS
Micrium uc/OS-IIs	Proprietary	ARM, ST32 PIC24	A priority-based pre-preemptive real-time multitasking operating system kernel, featuring unlimited application tasks. http://micrium.com/
CoDeSys SP Runtime System	Proprietary	X86, ARM, Infineon TriCore	Full-feature OS; Component based architecture for customer adaptations to various platforms. www.codesys.com/products/codesys-runtime.html

3. RTOSs survey

Every real-time system contains some operating system (OS) like functionalities: 1) Providing an interface to the input/output hardware; 2) providing coordination of virtual concurrency in a uniprocessor environment; 3) providing true concurrency with multi-core processors and distributed system architectures. In general, a RTOS has three main principal goals: 1) to offer a reliable, predictable, and low-overhead platform for multi-tasking and resource sharing; 2) to make the application software design easier and less hardware bound; 3) to make engineers in various industry fields to concentrate on their core product knowledge. However, full-feature operating system (OS) is not always the good solution in embedded systems for two main reasons. First, it is hard to meet design constraints of low-end embedded systems, such as system cost and complexity, response time and the punctuality. Second, the MCU is heavily occupied by the system software, instead of executing the time-critical application.

Nowadays, there are many RTOSs available from both commercial and research communities. These RTOSs have different license types, target usage and supported platforms. In order to select the suitable RTOS for our teaching purposes, we first did a survey of existing RTOSs whose target usage is in embedded applications, as shown in the table below. We follow three selection principles: 1) the license is low-cost or free; 2) The RTOS is portable to a large number of MCUs; 3) The RTOS is well-documented. Compared with other RTOS systems, FreeRTOS is free and supports most common MCUs from multiple manufactures. Moreover, it is easy to access a comprehensive FreeRTOS reference manual and lab exercises developed using FreeRTOS.

4. Introduction to FreeRTOS

FreeRTOS is chosen as the case study of RTOS to study multitask scheduling policies and real-time interfacing with MCUs. It provides real time scheduling functionality, inter-task communication, timing analysis and synchronization primitives. On top of FreeRTOS, complicated Cortex-M4 MCU applications can be efficiently built to meet their hard real-time requirements. It supports organizing these applications as a collection of independent threads of execution.

Since each of the MCUs used in our lab has one processor core, only one task can be executing at any one time. By computing the scheduling policy designed by the application developer, FreeRTOS is responsible of deciding which task (i.e., one thread of execution) should be executing at a time. Using the priority-based scheduling as an example, the application code first assigns higher priorities to tasks that implement hard real-time requirements, and lower priorities to tasks that implement soft or firm real-time requirements. Then, FreeRTOS could take care of the rest of the work and determine which task should be executing by examining the priority

assigned to each task by the applications. Thus, hard real-time tasks can always guarantee to be executed ahead of soft real-time task.

Besides helping ensure an application meets its processing deadlines, FreeRTOS can bring other benefits. To help students better understand the usage of this RTOS, three other main benefits are introduced through lab exercises. First, the kernel is responsible for timing and provides time-related APIs to the application. This allows the structure of the application code to be simpler and the overall code size to be smaller. Second, the idle tasks which are created automatically by the RTOS can be used to measure spare processing capacity, to perform checks in the background, or to place the processor into a low-power mode. Third, gatekeeper tasks provided by the RTOS can be used to serialize access to peripherals.

5. Application of FreeRTOS

Educators have explored hands-on RTOS development for enhancing student learning in real-time systems course [5, 6]. The lab assignment based on the FreeRTOS is the important experiential component of this course, which aims at gives student rich hands-on experience in building real-time systems.

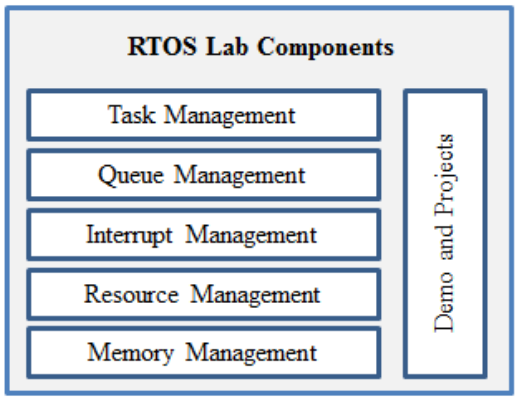

Figure 2. The RTOS lab assignment includes six core components.

Three things are prepared for tutoring each lab session before students start working on the lab assignment. First, a list of questions is designed for students to figure out the answers during each lab session. These questions serve as the guidelines to assist students' hands-on learning. Second, the concepts or algorithms related to each lab work, which have been introduced in lectures, are reviewed. Thirdly, as software programing is the main task in each lab, a set of relevant API functions are introduced to students for the efficient programming. These APIs are provided by either FreeRTOS or certain libraries included in a particular IDE. A standard demo project which incorporates MCU development board, simulator, logic analyzer and miniature RTOS with all basic features, is provided to students. They make use of this demo project as the basis to construct more application specific projects, which could achieve more efficient development compared with creating application from scratch. According to the course evaluation report, students gave the feedback that they benefit most from the hands-on programming experience.

Lab assignments on FreeRTOS

The lab assignments based on FreeRTOS consist of five key components. Figure 2 shows core lab components. More details are presented in the following:

1. Task Management
   Include 8 lab exercises on creating tasks, creating tasks, experimenting with task priorities, using Block state to create a time delay, combining continuous processing tasks and periodic tasks, defining an idle task hook function, changing task priorities, and deleting a task.
2. Queue Management
   Have 4 lab exercises on creating queue for task-to-task communication, sending data to a queue, receiving data from a queue, managing the stored data, blocking on a queue, and the effects of task priorities when sending to and receiving from a queue.
3. Interrupt Management
   3 lab exercises are arranged on deferred interrupt processing, using binary semaphores / counting semaphores to synchronize a task, and using queues within an interrupt service routine.
4. Resource Management
   2 lab assignments are given on mutual exclusion, mutex, priority inheritance and gatekeeper tasks for protecting resource access.
5. Memory Management
   1 lab assignment on different memory allocation schemes to illustrate their pros and cons.

Other tools are also employed for system development in this course, for example, FreeRTOSplusTrace from Atmel is used for simulating and observing execution traces for debugging.

6. Conclusion

This paper presents our primary experience of teaching the new course "Real-time Embedded Systems Design". First, the description of this course is presented. Second, we introduce a survey of RTOSs portable to ARM platform, especially an open source RTOS – freeRTOS with which the lab assignments are designed. In the future, we hope to apply another IDE Keil µVision which is dedicated for ARM programming to this course, refine lab assignments and propose new projects on developing FreeRTOS-based real-time applications.

References

1. "Real-Time Systems" Jane W.S. Liu, Prentice Hall.

2. "Real-Time Systems Design and Analysis: Tools for the Practitioner" 4th edition, Philipp A. Laplante, Seppo J. Ovaska, Wiley Publisher.
3. "The FreeRTOS™ Reference Manual", FreeRTOS online.
4. "Using the FreeRTOS Real Time Kernel – A Practical Guide – Cortex-M3 Edition", FreeRTOS online.
5. Kumar, G.S.A.; Mercado, R.; Manimaran, G.; Rover, D.T., "Enhancing student learning with hands-on RTOS development in real-time systems course," *Frontiers in Education Conference, 2008. FIE 2008. 38th Annual*, vol., no., pp.S2H-11,S2H-16, 22-25 Oct. 2008.
6. Yu JX, Zhao YG, Li Y and Duang HY, "An Example of Course Project of Real-Time Multitask Programming", *I.J. Education and Management Engineering on MECS*, 2012.