Top Down and From Scratch – A hybrid Approach of Teaching Real Time Embedded Operating System

Dr. Zhaohong Wang, California State University, Chico

Dr. Zhaohong Wang received his Ph.D. in Electrical Engineering from University of Kentucky in 2016. Prior to joining the faculty of EECE at CSU, Chico, he had worked as an embedded system engineer and software engineer throughout his graduate study. His teaching interests include embedded systems, computer networks, and digital signal processing. His current research is about algorithm design for digital signal processing in the encrypted domain and Internet of Things. He has been an active member of IEEE with the Signal Processing Society and Computational Intelligence Society since 2012 and 2016 respectively.

Dr. Jing Guo, California State University, Chico

Dr. Jing Guo got her PhD in Epidemiology and Biostatistics from University of Kentucky in May, 2015. She has worked as a statistician at Center of Healthcare Services Research at University of Kentucky before she joined California State University, Chico as a lecturer. Her research interests include machine learning, precision medicine, and cancer epidemiology. She has taught courses in statistics, research methodology in nutritional science, and research methods for healthcare education.

Top Down and From Scratch - A hybrid Approach of Teaching Real Time Embedded Operating System

Zhaohong Wang, Jing Guo

California State University, Chico

1. INTRODUCTION

Embedded system design and implementation is a key component in the undergraduate computer engineering curriculum. In an era of mobile and ubiquitous computing, a competent embedded device should have a real-time operating system (RTOS) to make full use of its potentials and to accommodate task needs. Deploying an RTOS also makes the programming job of embedded system engineers easier. Instead of embedded system engineers writing their own code to deal with the complicated scheduling among tasks, an RTOS provides the mechanism of scheduling as part of the RTOS's built-in features. A suitable RTOS helps the designer to focus on the application or computing tasks of the embedded system without worrying about the processor level configuration. Hence, it is not a surprise that real-time embedded system has a strong demand from industry. A good understanding of RTOS opens doors for many high-tech jobs for our students. Students would benefit a lot if they have the skill set of real-time embedded systems.

The course of Embedded Real Time Operating Systems (RTOS) is an important major course for the students in computer engineering. Students from other majors, like electrical engineering and mechatronics engineering, also take the RTOS course.

However, teaching an RTOS course is really challenging. There are mainly two reasons. The first reason lies in the theoretical design of RTOS. An RTOS course is a highly comprehensive course that is built upon the knowledge of computer architecture, peripheral interfacing, and data structures. In addition to a rich set of pre-requisite knowledge, the design of RTOS involves key engineering ideas such as resource management, efficiency, and complexity. The second reason is the fast development of RTOS. Practically, there are quite a few commercial RTOS ready for use. Each of them has its strengths and weaknesses, depending on the specific application scenario. It becomes a difficult choice that which one is the best fit to our students. Learning a specific RTOS has the benefit of being able to utilize it immediately for projects. Nonetheless, the fast development and updates on the area of RTOS may cause the knowledge learned from one specific RTOS outdated in some time and not applicable to another RTOS.

Traditionally, teaching and learning RTOS are implemented in the following ways. First, we teach the students the design principles of RTOS and show them a ready-to use commercial RTOS. We call this a "top-down" approach. The benefit is that students will be able to immediately deploy that specific RTOS [1]. However, without a thorough understanding of why and how tasks are created and managed by the RTOS, students lack the skill set to pick up a new RTOS as it comes out [2]. Second, we can teach RTOS by designing and implementing it from scratch. We call it a “from-scratch” approach. This second approach nicely exposes the design and implementation details of an RTOS [3]. The limitation of the “from-scratch” approach is that it may become too theoretical but not giving students enough practical skills in deploying state-of-art architectures [4]. The inadequate facts observed in the two previous approaches motivate us to design a new approach. Namely, we can use a “hybrid” approach by showing students both a commercial RTOS and a from scratch design.

Our specific research question is, among the three possible ways, is the “hybrid” the best in terms of student learning attainment of the RTOS knowledge? Our hypothesis is, teaching one method misses some understanding of the RTOS.

In this paper, we describe our innovative and effective way of teaching RTOS using a “hybrid” approach. To start with, we adopt the ARM Cortex-M and Cortex-A architectures, the most popular architectures in embedded systems. We then deploy a combinate “top-down” and “from scratch” hybrid approach to teach RTOS. The “top-down” approach utilizes an existing RTOS from ARM itself, the Keil RTX. Teaching RTX introduces quickly how a real time application may be designed and implemented. It also introduces high level behaviors and design ideas of an RTOS. After learning RTX, students will be able to immediately apply it to their senior design projects. They have had a working knowledge of RTOS and become curious to learn why things work. Next, the “from scratch” approach addresses the question of why things work in RTOS. The “from scratch” approach teaches how a complete RTOS is built from scratch, i.e., from the very first line of code in ARM’s assembly language. The “from scratch” part also addresses teaching the theoretical part of RTOS. Moreover, students will appreciate the many common design principles in RTOS that are not obvious in ready-to-use commercial RTOSes like RTX. Specifically, we described what components in RTOS are essential and implement them from scratch. Learning the “from scratch” part prepares students with a solid theoretical foundation to learn a new RTOS in the future.

Our main contribution is to show that our “hybrid” approach of teaching RTOS has proved to be effective in students’ learning attainment of the RTOS knowledge. The conclusion is drawn from quantitative data analysis about students’ evaluation on their learning experience and outcomes. Our data analysis specifically reveals that the “from scratch” part does significantly help students understand the processor level configuration than the “top-down” approach.

The remaining of the paper is organized in as follows. Section 2 reviews related work in teaching and learning RTOS. Section 3 describes our proposed “hybrid” approach. Section 4 explains our data analysis and our main discoveries. We conclude our paper and point to future work in Section 5.

2. RELATED WORK

The importance of RTOS has caught the attention by educators in the beginning of the 21st century. Since then, research on how to effectively teach RTOS in many scenarios continues until now. A simulator was developed to show the competing scheduling of 16 tasks on microprocessors [5]. The approach in [5] is basically an effort of showing the operating system “from bottom up”. The simulator saved the instructor and students a lot of time setting up physical hardware. Therefore, it facilitated the teaching and learning of RTOS. Due to quick evolution in technology, the idea of the proposed simulator could be implemented by the popular virtual machine technology today. A Capability-Innovation-Motive (CIM) teaching model was applied to teaching RTOS in the flipped classroom setting [7]. A kernel tracing tool has been used in explaining the concepts in the GNU/Linux Operating System. The target is not really an RTOS, however, the method reveals that understanding the internal mechanism is the key to understand any operating system [8]. A focus on the networking aspect of the RTOS kernel was explained by teaching the CAN bus. The emphasis was to make clear the networking part of the RTOS [9]. Laboratories were developed for teaching RTOS. For example, a virtual machine environment was introduced to reduce the setup time [10]. A commercial grade open source RTOS, FreeRTOS, was taught to show the deployment of RTOS. The emphasis was on using the FreeRTOS but not designing an RTOS [11]. The approach in [11] is a "top-down" approach of teaching RTOS.

As more and more commercial RTOS come to the market, learning a new specific RTOS may become more frequent. For example, as mobile phones transit to "smart phones", operating systems suitable for smart phones have been introduced. The QNX operating system used to be the one on BlackBerry Phones [12]. VxWORKS is among the industry's leading RTOSes for Internet of Things [13].

In order to comprehend new RTOS quickly, the interest on teaching and learning RTOS shifts to showing how an RTOS should be designed. This line of work is of the "from-scratch" approach. In fact, an experienced industry engineer started the work of showing the internal kernel design of an RTOS. The designer of the popular tiny µC/OS described the design principles and revealed its code [3]. A modern updated version based on [3] for a new microcontroller is presented in [13]. In academia, experts also try to educate students through design principles of an RTOS [2]. A noticeable "from-scratch" design is described in detail for the ARM Cortex-M4 processor [14]. A detailed RTOS design that is supposed to run on ARM 9 is presented [15].

Teaching an RTOS in a "top-down" approach is a typical traditional way. While there are quite a few work on showing the deployment of RTOS from a "top-down" approach, the design details are not obvious from using the RTOS. It became our motivation to show the design details for a much deeper understanding of the RTOS. A "hybrid" approach of showing both the use cases of an RTOS and its design details is missing in the literature.

3. THE PROPOSED APPROACH

In the past, observing the shortcomings of the "top-down" approach missing technical design details, we taught the RTOS in a "from scratch" way by showing the internal design and implementation of RTOS. The "from scratch" design is denoted as EOS. We wished to equip students with the ability of learning any new RTOS in their future career after understanding EOS. However, we still felt that the "from scratch" may be further improved by also showing a commercial RTOS. Our hypothesis was that the "from scratch" approach may miss the deployable convenience of a commercial RTOS, and the “top-down” approach misses technical details that the “from scratch” may complement.

Therefore, in the most recent offering of the RTOS course, we designed a “hybrid” teaching approach by showing both the EOS and RTX, a commercial RTOS from ARM. The target microprocessor running RTOSes is ARM Cortex-M4, a very popular microprocessor for embedded computing.

We designed our RTOS course to cover the following topics, as listed in Table I.

Table I: Topics Covered in the “Hybrid” Approach of Teaching RTOS

Topics	**Learning Outcome**
the ARM architecture, programmer's model, ARM instructions	Set up the development environment; get familiar with the experiment process
Assembly with C programming; device drivers	Be able to write useful device driver programs for the board
Exceptions and interrupts, interrupts processing (IRQ)	Understand key characteristics that enable real-time responses
interrupt-driven device drivers	Be able to write typical interrupt-driven device drivers
vectored interrupts; nested interrupts	Implement nested interrupts
Multitasking; Context switch; dynamic processes;	Write process management programs
Process synchronization: sleep/wakeup	Design even-driven multitasking system using sleep/wakeup
Semaphores; process communication	Design even-driven multitasking system using semaphores
Uniprocessor (UP) Embedded System Kernel	Implement preemptive kernels
Memory Management Unit (MMU) in ARM;	Write programs of memory paging by section
Memory managing schemes including one-level sections and two-level static and dynamic paging	Write programs of memory paging by pages; translating high virtual addresses.
User mode processes with a private user mode virtual address space	Implement an EOS with kernel and user modes

Processes in domains	Write domain-specific applications.

The topics listed in Table I could be sequentially classified into five parts. According to the five parts, we taught EOS and RTX in the following stages. First, we started from introducing the architecture of our target microprocessor. Second, we showed how a microprocessor began to run its first line of code in the assembly language. Third, we explain how and why a microprocessor such as ARM Cortex-M4 can respond to tasks in a real-time sense. The first three parts/stages were explained in EOS. The reason is that RTX did not allow us to change its code for showing the internal mechanism.

Fourth, we introduced the concepts of multitasking. We used RTX to demonstrate concepts like threads, thread synchronization, operating system services such as timing management, and inter-thread communication.

Fifth, we explained the design and implementation of concepts from what we have seen and tried in RTX. The design and implementation were illustrated using EOS.

The rationale behind the above-mentioned stages is to show students a complete computing software system. We have the RTX in part four before EOS, because concrete working examples gave students direct experience of the concepts in RTOS. After stage four, students were much more ready to accept the explanation why those concepts work through a detailed implementation in EOS.

At the end of the course, students went back to compare the RTX and EOS side by side. The last stage gave them a connection between "how" RTOS should be designed and "where" RTOS can be used. Students also participated in a voluntary survey about their learning attainment and experience in the "hybrid" approach. The data analysis and its interpretation are described in the next section.

4. INTERPRETATION AND DISCUSSION OF THE RESULTS

In this section, we present our main discoveries of the data analysis. The data analysis validates our hypothesis. EOS does help students learn IRQ that is less obvious in RTX.

4.1 Research Questions:

The first research question was whether students' confidence in understanding RTOS after learning both EOS and RTX was higher than learning only EOS.

The second research question was whether students' rate on the value of learning both EOS and RTX was higher than learning only EOS for the course of RTOS.

The third research question was how EOS and RTX complement each other in understanding different areas of RTOS.

4.2 Survey Questions:

In the survey, one item asked students to report their confidence in understanding RTOS after learning only EOS, while another item asked students to report their confidence in understanding RTOS after learning both EOS and RTX. Students' responses to each item have five options ranging from 1 (lowest) to 5 (highest). Students' value on learning only EOS was assessed using the following question: "How would you rate the value of learning only EOS?" Similarly, students' value on learning both EOS and RTX was assessed using the following question: "How would you rate the value of learning both EOS and RTX?" The response options of the above two questions range from 1 to 5 (1. Poor 2. Below average 3. Average 4. Above average 5. Excellent).

To address the third research question, two items asked students the following question "After learning EOS, which of the following areas do you feel you have a better understanding?" and "After learning RTX, which of the following areas do you feel you have a better understanding?" There are seven response options to the above two questions (A. System boot up B. IRQ set up C. Process/thread communication and synchronization D. Event driven design pattern E. Task scheduling F. Memory management G. CPU architecture).

4.3 Data Analysis and Discussion on the Results

In our RTOS class, there were 21 students. All of them were given the survey. A total of 20 students responded to the survey and there was no missing data. All their data were used for analysis. The data was analyzed using descriptive statistics, such as median and range, because the data was not normal distributed. We used Wilcoxon signed ranks test to address the first two research questions and McNemar's test to answer the third research question.

We first discuss what we found for the first research question: the confidence of students' understanding of RTOS after learning EOS and RTX has increased compared to learning only EOS. The range of the students' reported confidence in understanding RTOS after learning only EOS was from 2 to 4 with median 3, while the range of the students' reported confidence in understanding RTOS after learning both EOS and RTX was from 2 to 5 with median 4. Results from Wilcoxon signed ranks test [16] indicated that the students' reported confidence in understanding RTOS after learning both EOS and RTX was significantly higher than learning only EOS ($P=0.0313$).

For our second research question, we found that students' rate on the value of learning both EOS and RTX was higher than learning only EOS. The range of the students' reported value of learning EOS was from 2 to 5 with median 3, while the range of the students' reported value of learning both EOS and RTX was from 2 to 5 with median 4. Results from Wilcoxon signed ranks test [11] indicated that the students' reported value of learning both EOS and RTX was significantly higher than learning only EOS ($P=0.0352$).

For our third research question, we found that EOS and RTX complement each other in understanding different areas of RTOS. Figure 1 shows the percentage of students who responded having better understanding in seven different areas after learning EOS or after learning RTX. From Figure 1, 55% of students responded having better understanding of IRQ set up after learning EOS, while only 25% students responded having better understanding of IRQ set up after learning RTX. Result from McNemar test [17] showed that this difference was statistically significant (*P = 0.0313*).

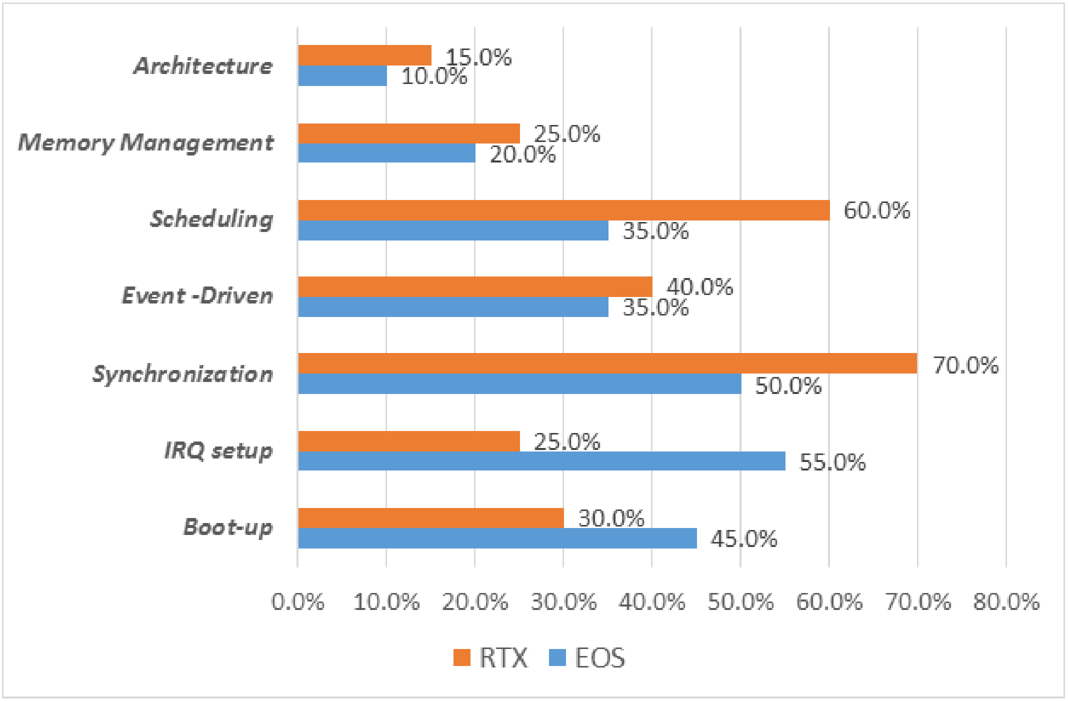

Figure 1. Percentage of students responding having better understanding in different areas after learning RTX or EOS

Therefore, the results validate our hypothesis that the deployable side of RTX does contribute to students learning after they had exposure to EOS. The result is a positive evidence of the effectiveness of the "hybrid" approach.

Finally, we would like to discuss the limitations of our study. The sample size in this study is small. Therefore, it could not give us more power if given a small effect size. We also acknowledged that it is better to use a validated survey specifically designed for our research questions. However, there was not an available survey for our research. Therefore, we specifically designed the survey questions related to our three research questions. We have considered why these designed questions could answer our research questions. We purposely worded our survey to ask for answers to our research questions. For example, we explicitly asked about the level of confidence in understanding RTOS after learning EOS and confidence in understanding RTOS after learning both EOS and RTX. For the above purpose, we used the questions "What is your level of confidence in understanding RTOS after learning EOS?" and "What is your level of confidence in understanding RTOS after learning both EOS and RTX?" Another limitation of this study is that the reliability and validity of the measurements were not reported. Our research questions focused on the comparison rather than the measurements themselves. By asking the similar questions, we expect a better reliability and validity on the results of the comparisons. In the future, it would be desirable to design a set of questions to test the reliability and validity of the survey on the measurements.

5. CONCLUSION AND FUTURE WORK

Some aspects of existing approaches to teaching RTOS open paths to improvement. Traditional ways of teaching an RTOS could be viewed as “top-down”. It shows how embedded system engineers may deploy an RTOS by using its application programming interfaces (API). A second traditional way of teaching RTOS is to show the internal design from scratch. Both methods have merit but may miss some valuable information toward students learning outcomes. Our “hybrid” approach to RTOSes combined both the “top-down” and “from scratch” methods. Our hypothesis is that a combination of the two traditional approach may further improve the learning outcome. Student survey in terms of confidence in understanding RTOS validated out hypothesis quantitatively. In addition, from students’ point of view, the value of the “hybrid” approach is statistical significantly higher than the past “from-scratch” approach. Moreover, the two traditional methods complement each other. The implication of our study is that a thorough knowledge of both the kernel design and the API deployment strengthens the understanding of both. A future direction of the work could be to further determine other factors that EOS may contribute to in addition to the IRQ topic. In the future, it would also be desirable to design a set of questions to test the reliability and validity of the survey on the measurements.

6. REFERENCE

[1] Andrus, Jeremy, and Jason Nieh. "Teaching operating systems using android." In Proceedings of the 43rd ACM technical symposium on Computer Science Education, pp. 613-618. ACM, 2012.

[2] Stallings, William, and Moumita Mitra Manna. Operating systems: internals and design principles. Pearson, 2015.

[3] Labrosse, Jean J. MicroC/OS-II: The Real Time Kernel. CRC Press, 2002.

[4] Catarinucci, Luca, Danilo De Donno, Luca Mainetti, Luca Palano, Luigi Patrono, Maria Laura Stefanizzi, and Luciano Tarricone. "An IoT-aware architecture for smart healthcare systems." IEEE Internet of Things Journal 2, no. 6 (2015): 515-526.

[5] Pack, D., and Barrett, S., "Real Time Operating Systems: A Visual Simulator", in 2004 American Society of Engineering Education Annual Conference, Salt Lake City, Utah. June, 2004

[6] Huang, Y., and Cheng, C., "Work in Progress: Tackling the Problems of Knowledge Integration and Barriers to Active Learning in a CDIO Course of Embedded Operating Systems – the Flipped Classroom Approach", in 2018American Society of Engineering Education Annual Conference, Salt Lake City, Utah. June, 2018

[7] Desnoyers, M., and Dagenais, M., "Teaching Real Operating Systems With The Lttng Kernel Tracer", in 2008 American Society of Engineering Education Annual Conference, Pittsburgh, Pennsylvania. June, 2008

[8] Rawashdeh, Z., and Mahmud, S. M., "Teaching Real Time Embedded Systems Networking And Assessment Of Student Learning", in 2009 American Society of Engineering Education Annual Conference, Austin, Texas. June, 2009

[9] Shirvaikar, M., and Satyala, N., "A Virtual Machine Environment For Real Time Systems Laboratories", in 2007 Annual Conference & Exposition, Honolulu, Hawaii. June, 2007

[10] He, N., and Huang, H., "Use of FreeRTOS in Teaching Real-time Embedded Systems Design Course", in 2014 American Society of Engineering Education Annual Conference & Exposition, Indianapolis, Indiana. June, 2014

[11] BlackBerry, "The QNX Neutrino Real Time Operating System (RTOS)." Available: QNX, http://blackberry.qnx.com/en/products/neutrino-rtos/neutrino-rtos [Accessed: February 4, 2019].

[12] VxWORKS, "The Safe and Secure RTOS for the Internet of Things" Available: WindRiver, https://www.windriver.com/products/product-overviews/2691-VxWorks-Product-Overview/ [Accessed: February 4, 2019].

[13] J Labrosse Jean, µC/OS-III: The Real-Time Kernel for the Infineon XMC4500. Micrium, 2012.

[14] Jonathan Valvano, Embedded Systems: Real-Time Operating Systems for ARM Cortex-M Microcontrollers. Volume 3, Fourth Edition. CreateSpace, 2017

[15] CHU, Hongyu, Leimin LI, Yuqing HUANG, and Jing ZHANG. "Implementation of Porting RTOS uC/OS-II to ARM9 [J]." Computer Engineering 20 (2005).

[16] Woolson RF. Wilcoxon signed-rank test. Wiley encyclopedia of clinical trials. 2007 Mar 9:1-3.

[17] Trajman, A., and R. R. Luiz. "McNemar χ2 test revisited: comparing sensitivity and specificity of diagnostic examinations." Scandinavian journal of clinical and laboratory investigation 68, no. 1 (2008): 77-80.