

Enhancing Student Learning with Hands-On RTOS Development in Real-Time Systems Course

G. Sudha Anil Kumar, R. Mercado, G. Manimaran and D. T. Rover

Iowa State University, anil@iastate.edu, rmercado@iastate.edu, gmani@iastate.edu, drover@iastate.edu

Abstract - Traditionally, real-time systems are built for a very small set of mission-critical applications like space crafts, avionics and other distributed control systems. The various steps in building such systems include, characterizing the workload, designing scheduling algorithms and performing schedulability analysis. Conventional teaching methodologies for real-time systems have primarily focused on these topics and the choice is completely justified for the targeted traditional real-time systems. However with the evolution of small scale real-time embedded systems like cell phones, PDAs, sensor motes and other portable control systems primarily driven by a Real-Time Operating System (RTOS), the conventional teaching methods fall short in several ways. This is because, building such real-time embedded systems poses certain different design and implementation challenges branching out of the severe resource constraints that these devices should operate under.

In order to keep pace with these changing trends, we have enhanced our real-time systems course in two different ways. First, we have included the relevant topics like compiler-level and operating systems-level energy aware real-time scheduling algorithms and further developed corresponding assignments and projects to reinforce student learning in these topics. We present some of these details here. Secondly, we have developed a series of laboratory experiments based on commercial RTOSs which give students a rich hands-on experience in building real-time embedded systems. We have tried two different RTOSs namely, RT-Linux and VxWorks in two consecutive years. In this paper, we present the similarities and differences between two the RTOS platforms and their impact on student learning.

Index Terms – Real-time systems, embedded systems, kernel programming, RTLinux, VxWorks, and Scheduling.

INTRODUCTION

Real-Time Systems (RTS) [1] is a prominent systems course in the computer engineering curriculum. It is regularly offered by different universities across the world in a variety of flavors. Majority of the universities offer RTS as an advanced course [7, 8] amenable only for the graduate students while a good number of universities design the course primarily for undergraduate students [9, 10].

At Iowa State University, the Real-Time Systems course, CprE 458/558, is offered to both undergraduate and graduate students. It is actively pursued by a total of 35 to 45 on-campus students and about 10 to 15 off-campus students every year. In order to cater to the needs and goals of different student classes, the course curriculum is regularly monitored and updated based on the student feedback and the research and industry advances in the area. In this paper, we present the details of two significant improvements that we recently incorporated into CprE 458/558.

Traditionally, real-time systems are built for a very small set of mission-critical applications like space crafts, avionics and other distributed control systems. The various steps in building such systems include, characterizing the workload and capturing the different resource constraints, designing scheduling and resource control algorithms, and performing schedulability analysis. Conventional teaching methodologies for real-time systems have primarily focused on these topics and the choice is completely justified for the targeted traditional real-time systems.

However, with the evolution of small scale real-time embedded systems (e.g., cell phones) which are primarily driven by a Real-Time Operating System (RTOS), the conventional teaching methods fall short. This is because, building such real-time embedded systems poses certain different design and implementation challenges branching out from the severe resource constraints that these small devices operate under. Some of these constraints include limited battery life, small on-board memory and limited computing and communication capabilities.

In order to keep pace with these changing trends, we enhanced our real-time systems course in the following ways. First, we included the relevant topics like compiler-level and operating systems-level energy aware real-time scheduling algorithms and further developed corresponding assignments and projects to reinforce student learning in these topics. We present some of these details here.

Secondly, we developed a series of laboratory experiments based on commercial RTOSs which give students a rich hands-on experience in building real-time embedded systems. We tried two different RTOSs namely, RT-Linux and VxWorks in two consecutive years. In the subsequent sections, we compare and contrast the two platforms and their impact on student learning.

CPRE 458/558: COURSE OVERVIEW

In this section, we present an overview of the general structure of the course and discuss the student learning that is typically observed at different stages of the course.

Cpre 458/558, which is regularly offered in the fall semester, progresses in three phases, *Fundamentals*, *Advanced Topics* and *Industrial Catch Up* (or *Industry Ketchup* as often referred by the students).

The *Fundamentals* phase primarily covers the basic scheduling algorithms, resource management protocols, fault-tolerance scheduling concepts and the corresponding schedulability analyses. This phase effectively contributes about 55 percent of the lecture hours in a semester. Students learn the above fundamental concepts, and start to appreciate the different design challenges (e.g., meeting all the task deadlines and resource constraints) involved in building a real-time system. They steadily develop a deeper understanding of the subject and are ready to correctly perceive systems as real-time and non-real-time. The interaction between the Teaching Assistant (T.A) and students is generally high during this phase.

The *Advanced* phase focuses on building complex real-time systems like networked real-time systems, distributed systems, real-time LAN, etc. In addition, the relevant research results recently published in the premiere real-time conferences and journals are discussed. This phase contributes about 30 percent of the total lecture hours. Students understand the bigger challenges involved in building complex systems and the different ways to tackle them. Further, they also gain a basic understanding of the research stance in this area. During this phase students get an opportunity to choose an interesting topic for their course project.

Finally, the *Industry Ketchup* phase discusses different industrial products that are built based on the concepts discussed in the earlier phases. This phase spans a wide variety of topics from scheduler in Linux operating system to scheduling in commercial CAN bus systems. Students are exposed to the industry standard products and can relate the concepts learned to products discussed. During this phase, typically, the off-campus students who are generally working in a company show an increased level of interest and participate very actively.

In the subsequent sections, we discuss the different extensions that we incorporated into the different phases of the course.

ENERGY-AWARE REAL-TIME SCHEDULING

Real-Time embedded systems like cell phones, etc., have limited battery life and hence every operation (including task scheduling in the RTOS) should be performed in an energy conscious manner. Energy-aware scheduling is an emerging area which addresses the issue of scheduling tasks with the goal of minimizing the total energy consumption while meeting the task deadlines.

The basic idea of energy-aware scheduling [2] is to schedule tasks effectively so that hardware resources can be put into low power states for as much time as possible thereby minimizing the total system's energy.

Significant amount of research has been conducted in the last few years in this area advancing the state-of-the-art real-time scheduling algorithms and resource management. Further, processor technology leaders like Intel and AMD have been regularly producing hardware with relevant support, making energy-aware scheduling a real possibility.

In the light of the above developments, we decided to incorporate energy-aware scheduling into the course curriculum. To this end, we introduced a series of interesting lectures on energy-aware scheduling along with an exciting set of assignments. Enough care is taken to ensure a smooth flow of lectures throughout the semester.

Specifically, we added the following topics which naturally merge into the existing flow of the lectures.

Lecture #1: (Latter part of *Fundamentals*)

- Energy-aware Scheduling; introduction.
- Shortcomings of conventional scheduling mechanisms; examples and illustrations.
- Processor workload variances; illustrations.
- Hardware Level Techniques; Dynamic Voltage Scaling (DVS) and Dynamic Power Management (DPM).
- Energy-Time tradeoffs; equations and analysis.
- Assignment set #1.

Lectures #2 & #3: (Latter part of *Fundamentals*)

- Compiler level Energy-aware scheduling; introduction and algorithms.
- Operating-System level Energy-aware Scheduling; introduction and algorithms.
- Compare and contrast of the above two approaches.
- Assignment set #2.

Lecture #4: (Part of *Industry ketchup*)

- Commercial Processors supporting DVS and DPM.
- DVS Practical issues; overheads and implementation details.
- Energy-aware message scheduling in wireless network.
- Discussion on open research problems in energy-aware scheduling.

I. Observations

In 2005, we tried to present the above material in just two lectures cutting down heavily on lectures #1 and #4 above, and integrating everything into lectures #2 and #3. As a consequence of the two-lecture approach a good number of students were not very comfortable with the background required for the assignments and were depending heavily on the TA. Major areas that needed elaboration were: hardware techniques (DVS and DPM) that are so tightly intertwined with energy-aware scheduling, and some basic compiler concepts for lectures #2 and #3.

As a result, for 2006 we developed the above presented four lecture plan and noticed an improved response from the students. They were able to handle major parts of assignments by themselves with minimal help from the TA.

Based on this observation, we continued a similar lecture plan for the year 2007 as well.

Students expressed elevated levels of interest during these lectures as the above topics helped them connect the concepts discussed in class to different everyday use consumer electronic products like cell phones, and laptops. In addition, a good number of students have pursued their class final projects from this topic. To mention a few, one of the projects implemented a simulator to evaluate various energy-aware scheduling algorithms while another project aimed at improving the performance of an existing algorithm.

HANDS-ON REAL-TIME SYSTEMS LABORATORY

In order to provide the future engineers with a rich hands-on experience in the RTOS environment, we developed an exciting set of laboratories. These labs allow students to implement and evaluate the concepts taught in the class. The rest of this section presents the details of these labs.

The primary goal of the laboratory experiments is to enable the students to achieve several important skills. The first skill is to understand the RTOS kernel architecture and be distinguishing an RTOS from a General Purpose Operating System (GPOS) based on the kernel architecture. The second skill is to design and develop applications for RTOS. The third and final skill a student develops is to effectively use different tools like, Makefile, debugger, visualization tools and related libraries.

Every year, one lecture, during the *Fundamentals* phase, is devoted to introduce the RTOS that is used in the labs. This lecture also includes examples and demonstrations to describe the working of the development tools, visualization tool and the related libraries. In the years 2002-2006, we successfully offered interesting lab exercises based on RT-Linux. In 2007, we tried a different RTOS, Wind River's VxWorks. In the following we present the details of RT-Linux labs followed by that of VxWorks labs.

1. RTLinux based Laboratory Experiments

RTLinux is a Linux based RTOS developed by FSMLabs, Inc. It is available in two flavors namely, RTLinuxFree and its professional version, RTLinuxPro. We used RTLinuxFree in our labs. In the rest of the paper, we simply refer to it as RTLinux.

RTLinux is an operating system in which a small real-time kernel coexists with the Linux kernel [3, 4]. The real-time kernel or the RTLinux kernel stands as a layer between Linux kernel and hardware as shown in figure 1, on the next page. On the one hand, all real-time applications or tasks run with the RTLinux kernel as kernel modules and have direct access to the hardware. On the other hand, Linux is run as a background task and is scheduled only when no real-time tasks are active. The following is a list of important characteristics of RTLinux.

- **Kernel Architecture:** Monolithic
- **Inter-Process Communication (IPC):** POSIX

- **Memory Management:** No Virtual Memory (good for RTOS)
- **Code Development Space:** Kernel space
- **Programming Language:** C
- **Integrated Development Environment (IDE):** None
- **Visualization Tools:** None

The most notable of the above characteristics is the fact that the applications need to be developed for kernel space. This means that the students are developing kernel modules as opposed to development user level programs. This provides students exposure to kernel module development and allows them understand several practical and advanced operating system concepts like: how to develop kernel level code, how to integrate code into the kernel and compile it, how kernel timers work, how task switching is performed, etc. Further, students also use POSIX libraries heavily in developing RTLinux applications.

Students typically go through the following steps to develop applications for RTLinux:

- Develop source code in normal Linux using standard editors like VI or Emacs.
- Compile the source code using tools like Makefile.
- If the application is a separate kernel module, then one can test it by simply inserting the module into the running kernel.
- On the other hand, if the scheduler itself is modified then the code needs to be integrated into the kernel and system needs a reboot to test the new scheduler.

Table I lists all the laboratory exercises. All exercises are mandatory except the last one. The last exercise is advanced and requires students to perform some additional efforts to successfully complete it.

TABLE I
LABORATORY EXERCISES

Exercise	Description
Intro	Understanding RTLinux architecture and developing "hello world!" applications
Lab 1	Calibrate the kernel's reaction time and create task sets with predictable execution times.
Lab 2	Design and Implement EDF and RMS schedulers and develop applications to test them.
Lab 3	Design and Implement EDF based (m, k) schedulers.
Lab 4	Design and Implement RMS based (m, k) schedulers.
Lab 5	Design and Implement EDF based Imprecise-Task schedulers.
Lab 6	Design and Implement RMS based Imprecise-Task schedulers.
Lab 7	Feedback based (m, k) scheduler. Advanced and optional.

Most students were excited to write kernel code and call themselves kernel hackers. The fact that the RTLinux applications can be developed in Linux without a need for a

new, complicated tool, has encouraged them immensely. Noticing the student satisfaction we could successfully offer the RTLinux labs for four consecutive years.

Commenting on the challenges to develop and maintain the lab infrastructure, the following issues are noteworthy: (1) A system used by a student for the RTLinux kernel development lab might remain unusable by other users that use the same laboratory (for e.g., in case of a crash due to a kernel bug); (2) Students need root privileges to perform kernel modifications.

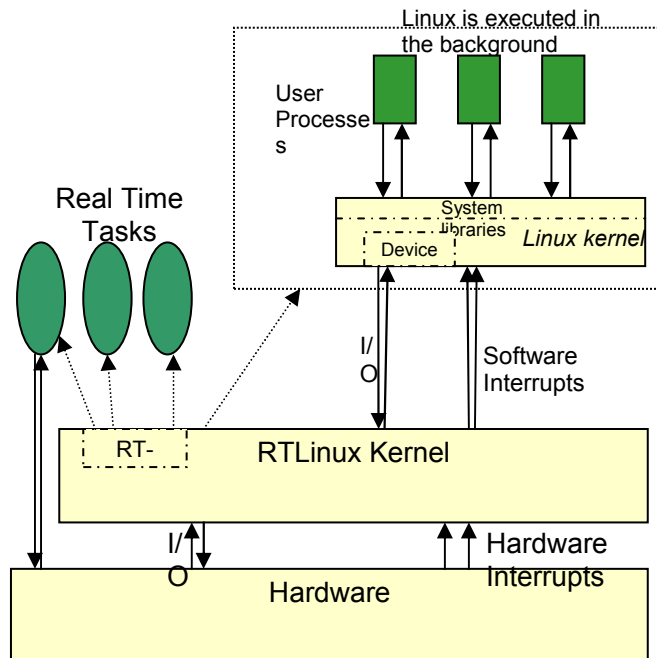


FIGURE 1
RTLinux STRUCTURE

The first challenge is about resource management of the lab infrastructure while the second one pertains to the security of it. We came up with a novel solution to tackle the first hurdle. We issued a removable hard disk for each student group which carries their development kernel and can be plugged into the computer during their lab hours, shown in figure 2. All the other times the computers in the lab carry a normal user hard disk which makes the entire system usable by the other users of the lab.

To combat the second hurdle, we assigned a set of static IP addresses to the machines in the lab and limited their access to the internet during the lab hours. Further, heavy monitoring has been plugged into place to minimize any possible security attacks from outside world.

II. RTLinux based Labs: Observations

Students developed a variety of skills that helped them evolve as good kernel programmers.

They also developed a strong understanding of RTLinux internals scheduler in particular. Further, they acquired an understanding for POSIX interfaces and acquired the skill of creating other POSIX compliant interfaces which applications can use. Students also understand the different

challenges that are unique to kernel development which are not encountered in normal application development.

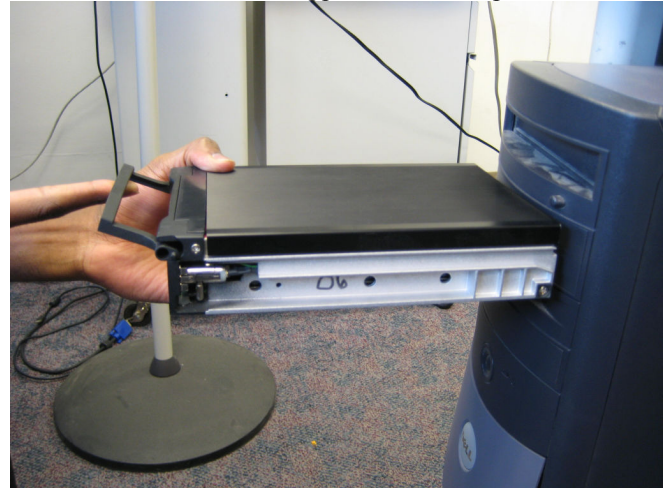


FIGURE 2
REMOVABLE HARD DISK SET UP

On the flip side, RTLinux fell short in several ways. First, there was no Integrated Development Environment (IDE) that could be used for easy code development. Further, it also lacks a good tool to visualize the run-time developments. As a result prime share of the students' efforts were being diverted to understanding the kernel development environment and to figure how to instrument the code so that they can visualize the runtime progress of the developed schedulers. Oftentimes, the instrumented code would interact with the system performance and would deviate from the expected behavior. Although the deviation can be minimized by careful instrumentation, we thought the emphasis on the basic scheduling mechanisms is being downplayed. These observations motivated us to seek an alternative RTOS for the labs.

III. VxWorks based Laboratory Experiments

VxWorks is a popular RTOS developed by Wind River, Inc. Unlike RTLinux, VxWorks is a standalone RTOS which does not co-exist with any other GPOS. It comes with an interesting set of tool chains including a good visualization tool that helps students keep track of the run-time developments.

VxWorks has a significantly different architecture as compared RTLinux. Figure 3, next page, shows the microkernel architecture of VxWorks [5]. The *wind microkernel* supports a range of real-time features, including multitasking, interrupt support, and both preemptive and round-robin scheduling. The microkernel design minimizes system overhead and enables fast, deterministic response to external events.

The WindNet networking layer handles the intertask communications mechanisms and permit independent tasks to coordinate their actions within a real-time system. A variety of IPC mechanisms are supported including shared memory (for simple sharing of data), message queues, semaphores, events and pipes (for intertask communications

within a CPU), sockets and remote procedure calls (for network-transparent communication), and signals (for exception handling). For controlling critical system resources, several types of semaphores are provided—binary, counting, and mutual exclusion with priority inheritance.

The different characteristics of VxWorks [6] can be summarized as follows.

- **Kernel Architecture:** Microkernel
- **Inter-Process Communication (IPC):** POSIX
- **Memory Management:** No Virtual Memory (good for RTOS)
- **Code Development Space:** User space
- **Programming Language:** C
- **Integrated Development Environment (IDE):** Tornado
- **Visualization Tools:** Windview

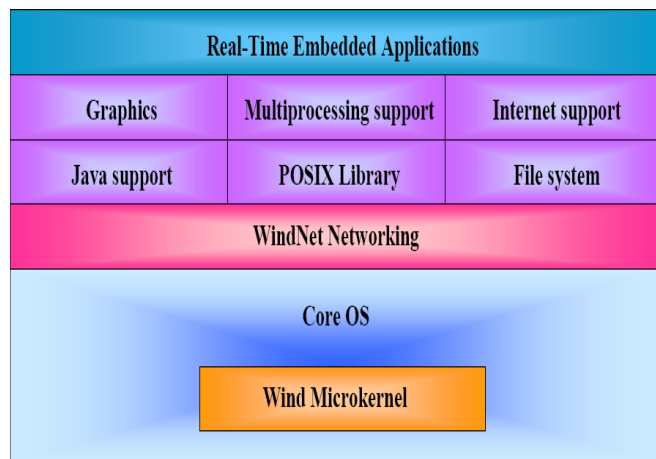


FIGURE 3
VXWORKS MICROKERNEL ARCHITECTURE

The lab infrastructure for the VxWorks laboratory is significantly different from that of RTLinux. It involves a hardware board connected to a desktop machine. Students develop applications on the desktop and download the binary to the board via Ethernet connectivity. The output of the developed code can be observed via a serial console connected back to the desktop. These different tasks are automated via a set of easy-to-use tools. In the following we briefly present the details of the individual tools and describe how they aid in the student learning.

The lab platform hardware consists of a Xilinx Virtex II Pro board from Digilent shown in figure 4. The board have a Xilinx XC2VP7 FPGA chip with 30,816 Logic Cells, 136 18-bit multipliers, 2,448Kb of block RAM, two PowerPC 405 processor cores, and DRAM support of up to 2GB. They also have rich I/O capabilities including Ethernet, USB, Video/audio ports and gigabit serial ports, among others. This platform is very close to the industry standard. It exposes the students to the rich features and full complexity of contemporary embedded systems design toolsets. This platform is part of the Xilinx University Program (XUP).

For real time programming with the VxWorks RTOS the students use the Tornado IDE from Wind River.

The Tornado IDE is the ideal environment for a resource constraint platform, such as the XUP. Tornado executes primary on a host computer, the development system, with access to a symbol table for the remote target system. Tornado includes several development tools that the students use throughout the labs, particularly useful is the WindView logic analyzer, seen in figure 5. WindView provides a graphical view of CPU allocation, and can be useful to determine what is happening in a system in which several tasks are running at once. It generates logs of events at run time for use later.



FIGURE 4
XILINX VIRTEX II PRO XUP BOARD

Utilizing the above resources, we developed the set of laboratory exercises shown in Table II. Similar to the earlier set, the last assignment is advanced and optional.

TABLE II
VXWORKS LABORATORIES

Laboratory	Description
Intro	Understanding VxWorks architecture and developing hello-world applications.
Lab 1	Calibrate the kernel's reaction time and create task sets with predictable execution times.
Lab 2	Design and Implement EDF and RMS schedulers and develop applications to test them.
Lab 3	Design and Implement EDF based (m, k) schedulers.
Lab 4	Design and Implement RMS based (m, k) schedulers.
Lab 5	Create a scenario for priority inversion and fix it using priority ceiling protocol.
Lab 6	Understand and modify an MP3 player for better audio performance. (Based on precedence scheduling, producer-consumer problem and resource access control). Advanced and optional

IV. VxWorks based Labs: Observations

Students learned the tool chain and were able to quickly develop applications. The learning curve was significantly

smaller as compared to kernel programming skills in RTLinux labs. The benefits of the WindView tool are worth mentioning. Students were not bothered with how to keep track of the run-time developments as they can be viewed in WindView. Maintaining the labs also become a relatively easy task as providing root privileges is no longer necessary.

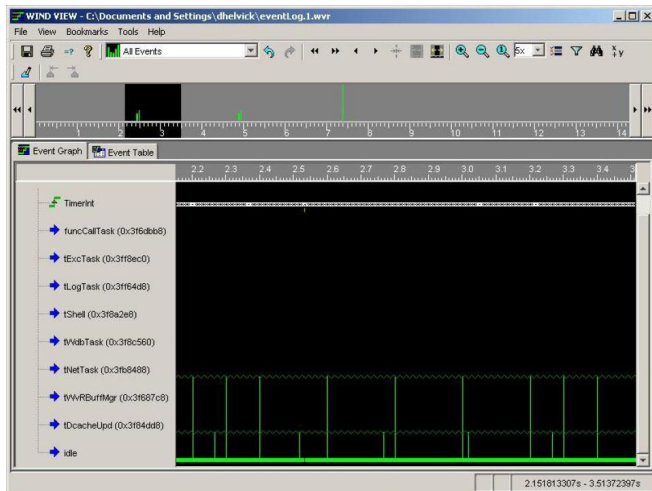


FIGURE 5
WINDVIEW LOGIC ANALYZER

FUTURE WORK

For the future of CprE 458/558, we plan to address two particular issues: (1) Creating lab exercises for energy-aware scheduling and (2) Providing lab exercises for the off-campus students. During the years 2002-2006 we offered RTLinux labs to those who could afford a computer running RTLinux. In future, we plan to provide VxWorks based labs for off-campus students via VxSIM, the VxWorks simulator.

ACKNOWLEDGMENT

This work was partially supported under NSF grant No. 0431924 and a GAANN grant from the U.S. Dept. of Education to the Information Infrastructure Institute at Iowa State.

REFERENCES

- [1] C. Siva Ram Murthy and G. Manimaran, "Resource Management in Real-time Systems and Networks," MIT Press, USA, April 2001.
- [2] G. Sudha Anil Kumar, G. Manimaran, and Z. Wang, "Energy-aware Scheduling of Real-Time Tasks in Wireless Networked Embedded Systems," to appear in *Proc. of IEEE RTSS*, 2007.
- [3] Barabanov, M., "A Linux-based RealTime Operating System", Master Thesis, New Mexico Institute of Mining and Technology, June 1997.
- [4] V. Yodaiken and M. Barabanov, "A Real-Time Linux", Online at <http://rtlinux.cs.nmt.edu/rtlinux/u.pdf>.
- [5] Wind River, "VxWorks 5.X: A Real-Time Operating System", White Paper, www.windriver.com/products/device_technologies/os/vxworks5/vxworks5x_ds.pdf
- [6] Wind River, "VxWorks Programming Guide 5.5", Online at <http://www.slac.stanford.edu/exp/glast/flight/sw/vxdocs/vxworks/guide/index.html>
- [7] Lih-Chyun Shu, "Teaching Real-Time Systems in an Information Systems Program". citeseer.ist.psu.edu/76320.html
- [8] Letia, T.S, Gruita, C, "Teaching real-time systems using Petri nets" Real-Time Systems Education III, 1998. PP. 49 – 56
- [9] Juan A. de la Puente, Alejandro Alonso, Marisol Garcia-Valls, Jose F. Ruiz, "Teaching Real-Time Systems at DIT/UPM", Third IEEE Real-Time Systems Education Workshop p. 117
- [10] Aleardo Manacero Jr., Marcelo B. Miola, Viviane A. Nabuco, "Teaching Real-Time with a scheduler simulator", 31st ASEE/IEEE Frontiers in Education Conference

AUTHOR INFORMATION

G. Sudha Anil Kumar, PhD candidate, Iowa State University, anil@iastate.edu.

Ramon Mercado, PhD candidate, Iowa State University, rmercado@iastate.edu

G. Manimaran, Associate Professor, Iowa State University, gmani@iastate.edu

D. T. Rover, Professor and Associate Dean, Iowa State University, drover@iastate.edu