AC 2008-1219: USING PROGRAMMING PROJECTS IN AN OPERATING SYSTEMS COURSE AS A CAPSTONE SOFTWARE ENGINEERING EXPERIENCE

Scott Schneider, University of Dayton

Scott J. Schneider is an assistant professor of Electrical and Computer Engineering Technology at the University of Dayton. He received his M.S. in Electrical Engineering from The Ohio State University. His areas of interest include software development, embedded systems, and automotive technologies.

Using Programming Projects in an Operating Systems Course as a Capstone Software Engineering Experience

Computer Engineering Technology students at the University of Dayton take two fundamental programming courses teaching the basics of algorithmic problem solving along with the VBA and C++ syntaxes. These courses develop a strong programming foundation for the students; however, they lack the ability to introduce software programming within larger software systems. The final course related to software development is a required operating systems course. This course contains three fundamental goals: to develop the students' understanding of key operating system concepts, to increase the students' software engineering capabilities, and to introduce the students to the internal workings of the Linux and Windows operating systems.

The operating systems course relies heavily on software programming exercises as a key instrument in teaching students the framework and applications of modern operating systems. The unique aspect of this approach is that the programming projects are designed not as standalone activities, but instead as individual components of a larger software system. Therefore, each programming project focuses not just on learning new syntax related to operating system concepts, but also in how these concepts are relevant to the software system being developed. Students develop programs for both the Microsoft Windows and Linux operating systems, working heavily with their associated application programming interfaces and investigating processes, threads, synchronization, input and output, and scheduling issues.

A complete course overview and synopsis of the software programming projects will be presented along with student performance and comments from an end of the semester survey.

Introduction

The Computer Engineering Technology (CET) Program at the University of Dayton (UD) was started in 1999 and included two software programming courses. These courses predominately focused on teaching the fundamentals of software programming and the syntax associated with the VBA and C++ languages. Likewise, to satisfy the ABET outcome requirements for computer engineering technology programs, the ETC Program at UD also required a new 3 credit lecture course entitled "Concepts & Applications of Operating Systems" to be taken during the students' Junior year of study.[1] This course was structured to provide students hands-on exposure to the internal workings of modern operating systems.

During the 2004/2005 academic year, the material being taught in the software programming was evaluated. In accordance with the Computing Curricula 2001 (CC2001) report by the Computer Society of the Institute for Electrical and Electronic Engineers (IEEE-CS) and the Association for Computing Machinery (ACM), the two software programming courses being taught fell within the traditional imperative-first approach. This approach starts students off with software programming activities without any real preparations.[2] After careful review, it was found that the student population being served needed to be taught the algorithmic thought process prior to the syntax as many of the students had no prior exposure to software programming and were struggling more with defining the problem solutions than with the specific syntax being taught.[3] A modified algorithms-first approach from CC2001 report was applied to these courses. Instead of an introductory course solely working on developing algorithmic solutions divorced from a specific language, the algorithmic development material was split between the two programming classes, allowing the students to utilize their new skills immediately with respect to a specific programming language.

The operating systems course, likewise, went through a transformation during the 2004/2005 academic year to establish it as a capstone experience related to software system development. Originally, this course was only concerned with teaching the key concepts and technologies of modern operating systems. This course contained a couple hands-on activities including performing system benchmarking tests and experimentation with various system administrative tools. These activities helped enhance the students' understanding of the necessary operating systems concepts; however it did not meet the new expectations of utilizing the course as a software systems capstone experience. In support of this new position, it has been reported that it is important to teach not just the concepts of operating systems but also the applications of the concepts for engineering technology students.[4]

Several key operating system concepts where selected to be used for programming projects in the modified operating system course. These projects explore the use of processes, threads, synchronization, scheduling, and input and output from within the user-space and synchronization and scheduling from the kernel-space. Not only do these projects help students develop advanced programming skills, they also demonstrate how these operating system concepts are related to the formation of advanced software systems. The non-programming projects were also expanded to complement the new software development tasks.

As the sophistications and requirements of the projects increased, the course was modified from a 3 credit lecture to a 2 credit lecture and 1 credit lab format allowing 2 extra hours of course time per week. This change was unanimously recommended by the students in an end of the semester survey and provides more instructor time during the projects to guide students through the programming process.

To support the operating systems course, a dedicated computer lab with dual-boot Windows and Linux computers is used. In this lab, each student is assigned a workstation and is fully responsible for all system maintenance. The students are given full access over modifying the operating systems, and are solely responsible for loading the Linux distribution and performing all necessary kernel modifications including recompilation and patching.

Operating System Course Format

The outline for the operating systems course is shown in Table 1. The lecture material follows closely to the topics discussed in the course text book Operating Systems Internals and Design Principles by William Stallings.[5] However, the laboratory material has been developed utilizing several external resources including some concepts from Kernel Projects for Linux and Operating Systems Projects Using Windows NT by Gary Nutt.[6,7] Note that each exercise demonstrates an application of the current concepts being discussed during the lecture portion of the class.

The real uniqueness of this course is not in the use of software programs to demonstrate operating system concepts, but in the deliberate sequencing of software development projects to demonstrate how and why the operating system concepts are important to developing software systems. The programming exercises in conjunction with the non-programming exercises provide for the development of two distinct software systems that is carried throughout the course as depicted in Figure 1. One system utilizes the Windows operating system while the other system operates within a Linux environment. Two additional projects at the end of the semester are not part of the two systems; however, they rely on many of the concepts developed previously during the course.

Topic
LSN1 - OS Overview
LSN2 - Windows OS
LAB1 - Windows System Admin
LSN3 - Linux OS
LAB2 - Linux Install / Imaging
LAB3 - Linux Shell Scripting
LSN4 - Software Programming
LAB4 - Software Programming
LSN5 - Windows GUI Programming
LAB5 - Windows API / GUI Programming
LSN6 - Processes & Threads
LSN7 - Windows Processes & Threads
LAB6 - Adding Threads to Windows Apps
LSN8 - Process Synchronization / **EXAM1**
LSN9 - Semaphores
LSN10 \ LAB7 - Windows Concurrency Mechanisms
LSN11 - Software Programming in Linux
LAB8 - Linux Processes & Threads
LSN12 - Interprocess Communications
LSN13 - Linux Concurrency Mechanisms
LAB9 - IPC & Linux
LSN14 - Process Scheduling
LSN 15 - Real-Time OS
LAB 10 - Linux Scheduling
LSN16 - Linux Kernel Programming / **EXAM2**
LAB11 - Linux Modules
LSN17 - Operating System IO
LSN18 - Linux IO System Programming
LAB12 - RTAI and Signals
LSN19 - Memory Management
LAB13 - File System Benchmarking
LSN20 - Operating System Security
LAB14 - Linux Firewalls
LSN21 - Future and Network Based Oses
FINAL EXAM

Table 1: Operating systems course topic listing

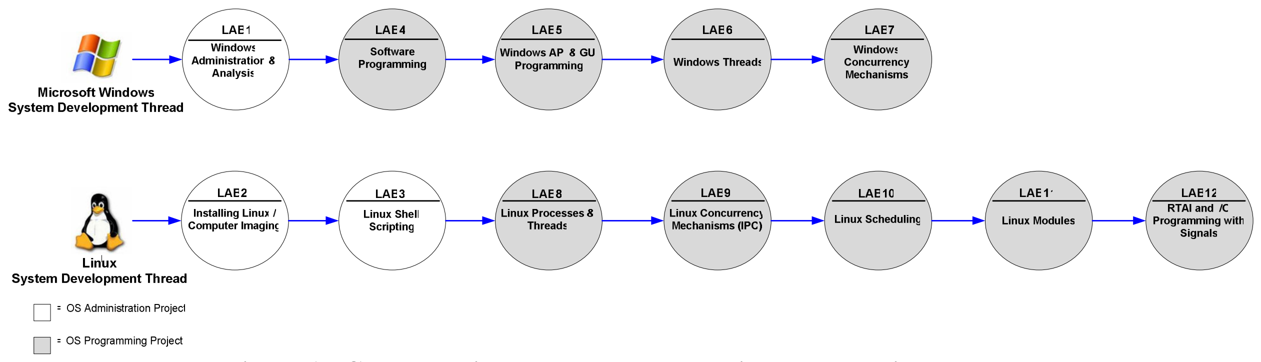

Microsoft Windows
System Development Thread
LAE1
Windows Administration & Analysis
LAE4
Software Programming
LAE5
Windows API & GUI Programming
LAE6
Windows Threads
LAE7
Windows Concurrency Mechanisms
Linux
System Development Thread
LAE2
Installing Linux / Computer Imaging
LAE3
Linux Shell Scripting
LAE8
Linux Processes & Threads
LAE9
Linux Concurrency Mechanisms (IPC)
LAE10
Linux Scheduling
LAE11
Linux Modules
LAE12
RTAI and I/O Programming with Signals
□ = OS Administration Project
▨ = OS Programming Project

Figure 1: Course projects used to develop Windows and Linux systems

Student performance for this operating systems class is assessed using multiple methods. There are fourteen projects ranging from working with built-in administrative operating system tools to developing software programs using operating system concepts to operating system testing and applications. Early projects focus on familiarizing student with key concepts such as processes, threads, scheduling and abstraction. From this foundation, students hone their programming skills through a variety of projects, each building on the previous projects while at the same time addressing an individual operating system concept. Applications of many of the key concepts are developed for both the Windows and Linux operating systems, highlighting their similarities and differences. In addition to the homework assignments, there are two take-home midterm exams and a final in-class exam.

Operating System Course Projects

As stated previously, the course projects fall into one of three categories. The first projects are used to familiarize the student with the Microsoft Windows and Linux operating systems using administrative tools, most notably to highlight key concepts. The next projects relate to the development of software programs. These programs combine to form a Windows and a Linux software system as shown in Figure 1. The final projects look at analysis and applications of operating systems.

Operating System Administrative Projects

The first course project provides the students with an opportunity to work with the Windows operating system using several administrative tools. Students start by investigating and editing the Windows Registry using *regedit.exe* to perform some simple administrative tasks. Next, the students must perform a series of tasks to monitor the system performance in relation to the operating systems concepts of processes, threads, and objects. This series of tasks is based on Exercise 1 contained in Nutt's book.[6] The students use the built-in Windows Task Manager, *taskmgr.exe*, and Performance Monitor, *perfmon.exe*, utilities along with the Microsoft Visual Studios SPY++ tool. After students are familiar with these tools, they run a synthetic computer load application that automatically spawns a user specified number of threads for a user specified amount of time. They are responsible for determining the actual number of threads running for this application and its impact on the system performance using the previously investigated utilities. Familiarity with these tools will also be used extensively throughout the semester to aid the students in analyzing the programs they develop.

Since most of the students have had little to no exposure to Linux, the next two projects allow the students to gain familiarity with installing and working within it. Linux was selected for inclusion to this course because of its growing importance in the computing community and it provides the students with an open source kernel that they can investigate from the inside, unlike the Windows kernel. Furthermore, there are many free Linux distributions available along with a very active support community on the Internet to encourage the students to work with it on their own outside of class.[8, 9]

During the second project, the students must install the Fedora Core (http://fedoraproject.org/) distribution of Linux onto their assigned lab computer. Prior to installing the Linux distribution, the students go through a short exercise related to imaging and resizing a hard drive partition.

The third project utilizes the new Linux install to provide the students with an exposure to the Linux operating system, predominately through shell scripting. A large amount of administrative work in Linux is performed through shell scripting, therefore it is a very useful tool for students to learn.[8] During this exercise, the students start by exercising shell commands and working with the `/proc` virtual file system to answer questions about their computer system similar to Exercise 1 in Nutt's book.[7] The `/proc` virtual file system contains information about the system resources, allowing the students to investigate the kernel state and current processes and their associated properties. Leveraging from this work, the students develop simple BASH shell scripts that can be used to automatically extract specified system information from `/proc`.

Windows Programming Projects

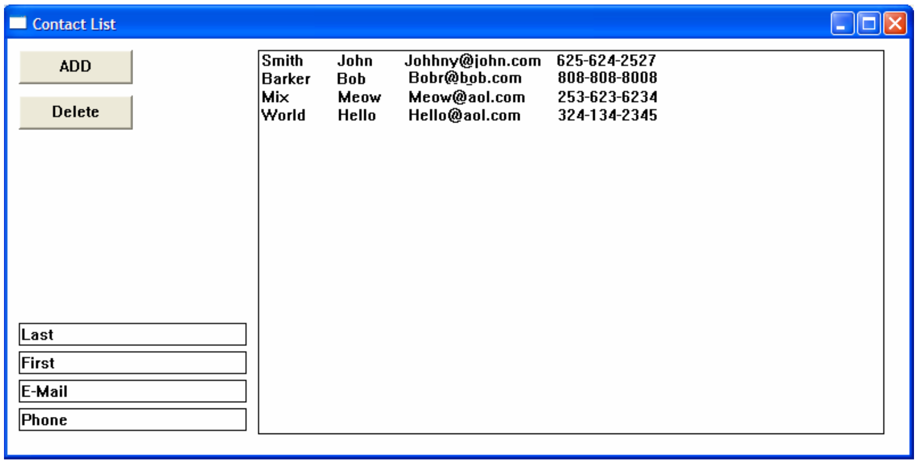
Students start their programming projects during the fourth course exercise with the development of a simple application that helps them refresh their C++ syntax and programming skills while introducing some advanced data structure concepts. The program that the students develop is a simple contacts database manager that interfaces to a file containing the names and information for their contacts. The application must be able to read, add, and delete contacts from the database file. The deliverable for this project is not simply the program code, but also all relevant design and analysis documentation. This exercise provides a starting point for the remaining Windows programming exercises.

The fifth project entails modifying the program from exercise four to include a Windows graphical user interface (GUI) for their developed contact database manager. This project, while highly motivating for the students, also helps introduce them to working with the Windows Win32 application programming interface (API) prior to dealing with processes and threads. Students are already familiar with windows and their associated components, therefore the creation of them is not as foreign as the creation of processes and threads might be. Furthermore, in working with the Win32 API, students see firsthand how the concept of abstraction applies to operating systems.

Students must learn how to utilize the Microsoft Developer Network (MSDN) library to understand the implementation specifics for the necessary Win32 API functions found in the `windows.h` header file. A short listing Win32 API functions used for this project and their associated operation is shown in Table 2. A sample GUI developed by a student is depicted in Figure 2. The students must submit their program, algorithms, and analysis. They are also required to utilize the performance monitoring tools evaluated during the first exercise to validate their window environment.

Win32 API function	Description
`WinMain(…)`	The user-provided entry point for a Windows-based application
`CreateWindow(...)`	Creates an overlapped, pop-up, or child window
`ShowWindow(…)`	Sets the specified window's show state
`UpdateWindow(…)`	Updates the client area of the specified window
`GetMessagae(…)`	Retrieves a message from the calling thread's message queue
`WindowProc(…)`	Function that processes messages sent to a window
`SendDlgItemMessage(...)`	Used to send messages to list boxes
`MessageBox(…)`	Display a dialogue box with a brief application message
`GetWindowText(…)`	Copies the text of the specified text box (window) into a buffer

Table 2: Win32 API subset for GUI class project



Smith	John	Johhny@john.com	625-624-2527
Barker	Bob	Bobr@bob.com	808-808-8008
Mix	Meow	Meow@aol.com	253-623-6234
World	Hello	Hello@aol.com	324-134-2345

Figure 2: Sample student contact manager GUI

The next project requires that the students add threads to their contact database manager application so it can be operated in a shared environment where the database file is stored on a network drive. It is common to develop multithreaded applications for such user interface systems. Students must modify their application to include a separate thread for the “Add” and “Delete” operations with the shared database file. Additionally, students must create a “Refresh” thread to automatically synchronize the local copy of the database file and the file itself since it can be simultaneously accessed by multiple users. Students are asked to validate the program, namely the thread creation and destruction operations, and to document their applications object hierarchy using the Windows tools evaluated during the first exercise. Finally, students must evaluate the performance of their system and note if there are any issues with thread sequencing and the shared database file or its local copy. This project helps students visualize the operations of threads, along with when and why they are used in software systems. The Win32 API functions utilized throughout this exercise are shown in Table 3.

Win32 API function	Description
`CreateThread(…)`	Creates a thread to execute for the calling process starting at the desired instruction
`ExitThread(…)`	Ends the calling thread
`Sleep(…)`	Suspends the execution of the current thread for a specified interval
`CloseHandle(…)`	Closes an open object handle

Table 3: Win32 API subset for threads class project

Most students readily noticed that the threads can get out of sequence in the last project causing unexpected data to be added or removed from the contacts database file and for inconsistencies to exist between the database file and the locally maintained copy of database information. This realization clearly demonstrates the need to add concurrency mechanisms to the application. The seventh project therefore asks the students to protect the shared database file and the local copy of the database file using semaphores, mutexes, or critical sections. The Win32 API functions used for this task are located in Table 4. Furthermore, the design must preclude deadlock or starvation from occurring. The students again need to analyze their application using the system tools introduced during the first experiment. This exercise culminates the Windows programming experience having stepped the students through the development of a complicated software system.

Win32 API function	Description
`WaitForSingleObject(…)`	Waits for the specified object to be in signaled state
`GetLastError(…)`	Retrieves the calling thread's last-error code value set after wait or mutex creation
`CreateMutex(…)`	Creates a mutex object for thread synchronization
`ReleaseMutex(…)`	Releases ownership of a mutex object
`CreateSemaphore(…)`	Creates a semaphore object for thread synchronization
`ReleaseSemaphore(…)`	Releases ownership of a semaphore object
`InitializeCriticalSection(…)`	Initializes a critical section object for thread synchronization
`EnterCriticalSection(…)`	Waits for ownership of the specified critical section object
`LeaveCriticalSection(…)`	Signals the release of the specified critical section object
`DeleteCriticalSection(…)`	Releases all resources used by an unowned critical section object

Table 4: Win32 API subset for Windows concurrency class project

Linux Programming Projects

The remaining nine projects are with the Linux operating system. The eighth and ninth exercises have the students develop programs within Linux containing multiple processes and threads and utilizing process and thread synchronization methods including anonymous pipes for interprocess communications (IPC). These activities are primarily programming exercises, allowing the students to gain familiarity with the Linux software development environment, while at the same time familiarizing them with the Linux system functions and the POSIX `pthread` API. Functions used during these exercises are listed in Table 5 along with their associated operation.

Linux System & pthread Functions	Description
`fork()`	System call to create a new process copied from parent process
`wait()`	Allows parent process to wait for all children processes to terminate
`waitpid(…)`	Allows parent process to wait for specified children processes to terminate
`pthread_create(…)`	Creates a new thread to run in the address space of the calling thread
`pthread_join(…)`	Allows parent thread to wait for a child thread to terminate for synchronization
`pthread_exit(…)`	Called by child thread to terminate
`pthread_mutex_init(…)`	Creates a mutex object for thread synchronization
`pthread_mutex_destroy(…)`	Releases mutex object
`pthread_mutex_lock(…)`	Waits for mutex object to be available then locks it else calling thread is blocked
`pthread_mutex_trylock(…)`	Waits for mutex object to be available then locks it else returns without being blocked
`pthread_mutex_unlock(…)`	Signals the release of ownership of the mutex object
`pipe(…)`	Creates a pair of file descriptors pointing to an inode
`read(…)`	Blocking read from the specified file descriptor (pipe)
`write(…)`	Writes to file descriptor (pipe)
`ioctl(…)`	Manipulates file descriptor to allow for non-blocking reads

Table 5: Linux system functions used for threads, processes and synchronization projects

Students are encouraged to find an editor within Linux for their software development activities after being introduced to Emacs, VI, and gedit. Most students tend to gravitate towards gedit because of its simplicity and resemblance to a Microsoft editor. The programs are compiled from within the BASH shell using the GNU `gcc` and `g++` compilers.

Projects ten through twelve are coordinated to develop a software system that sends a pulse width modulated (PWM) signal out the parallel port of the computer to drive a fan in response to a temperature reading using the serial port and a user input target temperature. Students must take several steps during the formation of this project, many of which are based on a tutorial provided for the Real Time Application Interface (RTAI) patch for the Linux kernel.[10]

First, in exercise ten, students investigate the Linux scheduler using a simple application written during class to generate a waveform on a data output pin on the parallel port. Students must analyze the generated signal using an oscilloscope under different system loads, determining the time quantum and epoch for their Linux system's scheduler. The next step requires students to download a vanilla 2.6.15 kernel from http://kernel.org/ and compile and load it on their system. This step is required to allow the students to work in kernel space through the development of Linux modules.

Linux modules are applications that can be inserted into the kernel during run-time to extend its functionality. The most common Linux modules are device drivers. It has been shown that using Linux modules in an operating systems class provides the students an inside view of how an operating system functions since they execute in kernel-space.[9] During project eleven, after Linux modules are introduced, the students modify their signal generating application to run as a module and retest the system scheduling. Now, as the application runs in kernel-space, they can document its impact on user space applications.

Finally, students patch their Linux systems with the RTAI kernel patch to turn them into real-time (RT) operating systems. In class, students investigate the RTAI patch operations by modifying their signal generating module to run under the RTAI patch and verify the ability of the system to meet scheduling deadlines regardless of system load.

The twelfth project requires that the students integrate the disparate concepts from projects eight through eleven along with course material on Linux input/output programming into a single working software system. This system includes a user-space application that must read the current temperature from the serial port using signals and calculate a PWM signal to drive the fan to reach a target temperature. The calculated PWM signal is transmitted to the kernel-space PWM generator module using IPC. This software system is very complex and highlights several key operating system concepts such as processes, synchronization, input and output, and scheduling. Furthermore, the sophistication of the software solution makes this truly a capstone experience in software development.

Operating System Analysis and Application Projects

The remaining projects allow the students to work with the Windows and Linux operating systems from an analysis and applications perspective. First, in project thirteen, the students are responsible for comparing the Windows and Linux file systems. They must perform a benchmark analysis of the system. The Sandra system diagnostic tool from SiSoftware is used to benchmark the Windows file system and LmBench is used to benchmark the Linux file system. The students are responsible for completing the tests, analyzing the results and making performance comparisons.

The final project allows the student an opportunity to further develop their Linux administrative skills in the development of a firewall system. The students use the Shorewall firewall tool to setup the built-in Linux packet filter utility, Netfilter. They are required to research, install and run the Shorewall tool using the original Fedora Core distribution on their lab computers. Their setup must also provide an Internet Protocol (IP) filter along with Media Access Control (MAC) access verification. The final setup is tested by the instructor for correctness. The students are responsible for submitting their design documentation along with a full description of how the Shorewall tool is operating using the built-in Linux Netfilter and iptables utilities.

Student Perceptions and Impact on Learning

A student survey is routinely administered at the end of the course to evaluate course content and effectiveness. Unanimously, students have valued the inclusion of both the Windows and Linux operating systems and have found the software development projects both challenging and enjoyable. However, students have historically less favorably commented on the level of programming competency required and the initial Linux "shock" of working in a new operating system environment.

Standalone software projects were added during the 2005 semester, with scheduling exercises using the RTAI Linux patch introduced during the 2006 semester and Windows GUI programming during the 2007 semester. Additionally, the concept of using software programming projects that build upon each other in the formation of software systems was not introduced until the 2007 semester. In reviewing the student feedback from the end of the semester surveys, there was overwhelming support for learning about both the Windows and Linux operating systems, with the lowest total during the 2006 semester of only 80% of the students in support. Likewise, roughly 80% of the students favorably rated the course projects and assignments as aiding in their understanding of the material. During the 2005 and 2006 semesters, the greatest area of dissatisfaction was with roughly 40% of the students responding that they did not have sufficient software programming skills to complete the programming assignments. During the 2007 survey, no students commented on insufficient software programming skills.

Student performance can also be investigated to determine the impact of the course modifications. The class averages for the 2005 through 2007 course offerings was a B (83%), C+ (78%), and a B+ (88%), respectively. Looking at the spread of the final grades show a standard deviation of 6%, 17%, and 4%, respectively for the 2005 through 2007 semesters. This data indicates that student performance decreased as the level of difficulty increased with respect to the programming projects. However, by synchronizing the projects together, and adding the Windows GUI programming as an introduction to using APIs student performance not only improved, but became more consistent.

Conclusions

Coordinating software programming projects in an operating systems course to provide a capstone software development experience proved quite successful. This experience has evolved over four years from a course with no programming exposure to one that engages students to develop complex software systems applying key operating system concepts. Student feedback from course surveys along with their course performance has helped to demonstrate the benefits of synchronizing the course programming projects and the associated impact it has on student learning.

Bibliography

1. Criteria for Accrediting Technology Programs (2007-2008); ABET Technology Accrediting Commission, ABET, Inc.
2. ACM/IEEE-Curriculum 2001 Task Force, Computing Curricula 2001, Computer Science, December 2001. http://www.computer.org/education/cc2001/final/index.htm
3. Schneider, S., "Developing an Introductory Software Programming Course for Engineering Students", *American Society of Engineering Education Annual Conference Proceedings*, June 2005.
4. Loendorf, W., Brzoska, M., Koh, M., Rodriguez, E., "Implementing a Software Engineergin Technology Program within the Context of Experienced-Based Learning", *American Society of Engineering Education Annual Conference Proceedings*, June 2004.
5. William Stallings, "Operating Systems: Internals and Design Principles", Fifth Edition, Pearson Prentice Hall, 2005.
6. Gary Nutt, "Operating System Projects using Windows NT", Addison Wesley, 1999.
7. Gary Nutt, "Kernel Projects for Linux", Addison Wesley, 2001.
8. Eastman, E., "Exploring Linux as an Operating System in The CS Curriculum", *Journal of Computing Sciences in Colleges*, April 2006.
9. Bower, T., "Using Linux Kernel Modules for Operating Systems Class Projects", *American Society of Engineering Education Annual Conference Proceedings*, June 2006.
10. "RTAI: a Beginner's Guide", Dipartimento di Ingegneria Aerospaziale - Politecnico di Milano, https://www.rtai.org/.