AC 2009-854: TEACHING AN OPERATING SYSTEM COURSE TO CET/EET STUDENTS

Xuefu Zhou, University of Cincinnati

Teaching an Operating System Course to CET/EET Students

Abstract

This paper describes the motivation for teaching an operating system course to computer engineering technology (CET) and electrical engineering technology (EET) students. It presents course topics and teaching approach. The accompanying laboratory exercises are also briefly described.

1. Introduction

An operating system (OS) provides a well-known, convenient, and efficient interface between user programs and the bare computer hardware. As a service provider, it provides the basic function on every computer so that application software can be run. Operating systems are necessary and so important for any use of the computer. For computer and electrical engineering technology (CET/EET) students, it is imperative to understand the principal concepts and mechanisms of operating systems since most of them will practice in the industry and be seriously involved in computer system development and application. They should acquire sound OS knowledge. Specifically, CET/EET students should know how an operating system works, what its major components are, how to utilize an operating system's resources and service to efficiently develop applications, how to tune an operating system for optimal performance, and how to select an operating system for a particular application (i.e., real-time industrial controllers).

Though OS is one of the fundamental and core courses of computer science or computer engineering disciplines, it is not commonly offered in CET programs. This paper describes an operating system course taught to CET/EET students. The remainder of this paper is organized as follows. Section two discusses course development, including course context, course objectives, course content and laboratory exercises. Section three presents our teaching experience and reflection, and Section four gives the conclusion.

2. Course Development

2.1 Curriculum Context and Course Objectives

In our CET curriculum, Computer Architecture, Object Oriented Programming (Data Structure) and Computer Security are three courses pertaining to the operating system course. The first two are prerequisites for the operating system course, and the operating system course itself is one of the prerequisites for computer security course. OS is a required course for CET students while EET students can take it as a technical elective. The primary goal of this course is to have students understand the internal components and underlying activities of an operating system, and learn how to utilize operating system resources to develop their own applications.

Upon completion of this course the student will be able to

- explain basic computer system organization and the objectives and functions of modern operating systems, and describe how operating systems have evolved over time from primitive batch systems to sophisticated multiuser systems.
- explain the difference between a resource, a program, and a process, and understand how computing resources are used by application software and managed by system software, and defend the need for APIs and middleware.
- contrast kernel mode and user mode in an operating system.
- discuss the need for concurrency within the framework of an operating system, and explain the race condition arising from multiprogramming.
- compare and contrast process and thread concepts.
- apply various techniques such as Semaphore, Mutex and Monitor to solve synchronization problems.
- describe the types of processor scheduling and how they affect a computer system's performance.
- analyze and compare various memory management methods and policies (i.e., paging, segmentation, virtual memory) and explain how they are realized in hardware and software.
- explain the concept of thrashing, both in terms of the reasons it occurs and the techniques used to recognize and manage the problem.
- describe how files are stored in secondary storage.
- explain performance characteristics of massive storage devices and describe various RAID levels.
- describe the function of I/O device driver.
- analyze how OS affects the computer system's performance and reliability.

2.2 Teaching Approach and Course Content

We adopted [1] as a textbook for this course. We tailored 10 lecture modules for this course. There are about 30 hours for lecturing (three credit hours for the lecture) and 24 hours (one credit hour for the laboratory) for the laboratory exercises. There are different pedagogical approaches to teach an operating system course. Our emphasis is to understand concepts, and to utilize operating system resources. OS design, implementation and protection are not covered in this course. Security and protection are covered in another course titled "computer security". This course, by nature, is quite abstract to the students. We think a problem-based approach (a theme of problem identification and problem solving) may help students understand the ideas and techniques used by OS designers and apply them to solve general problems in software, or computer system development.

The first module is an introduction to operating systems. Computer system architecture (including computer system organization, interrupt, and I/O) are briefly reviewed. Students are guided to look at the operating system from a problem-solving perspective. OS arose historically as people needed to solve problems associated with using computers. By comparing the first operating system and today's operating systems, students understand how operating systems evolve with new problems arise. By giving an in-depth discussion on the OS evolution, students can involve themselves into the subject matters, and crave to know how these problems have been solved. Topics in this module also include OS components, OS services, and the role of OS in the computer systems and OS history.

The second module is to introduce the concept of process. We illustrate the difference between a process and a program or file through examples. We discussed various ways to implement a multi-tasking application in different operating systems (DOS, UNIX and Window XP) and have students recognize that process also arose as a solution to multi-tasking application. Process control block, process states, process life cycle (creation and termination), context switch and the overhead associated with context switch are discussed in details. Inter-process communications are addressed briefly as well.

By examining the problem with process (i.e, cost, responsiveness, scalability and inflexibility of inter-process communication), students are guided to the concept of thread. Emphasis is to discuss the similarities of process and thread (sharing CPU, creating child, sequential execution, multiprogramming) and the dissimilarities (independence, communication and cooperation). A real-time multi-tasking industrial controller is used as an example to investigate the difference and its multithreading implementation.

The fourth module is to introduce the concept of CPU scheduling. This is a relatively abstract concept. We use numerical examples, to compare the performance of different scheduling algorithms and demonstrate that algorithms have crucial effects on the behavior of the system in terms of overhead due to the number of context swaps, and efficiency and response time (how long it takes to do something).

By discussing the classical bounded buffer problem, the fifth module is to teach students why it is critical to synchronize processes or threads in a multi-programming operating system and how to avoid deadlocks. Emphasis is to introduce the POSIX and Windows API synchronization tools (Mutex, Semaphores etc.).

The sixth module is to introduce memory allocation techniques. Segmentation and paging memory management techniques are discussed as solutions to the problem of external fragmentation encountered in contiguous memory allocation.

Observing the need to simplifying programming task, and to increase the degree of multiprogramming, to increase CPU utilization and throughput, and to improve responsiveness performance, we introduce the virtual memory concept to students. Students investigate how virtual memory technique affects a computer system's performance. Concept of demanding paging is introduced with a brief introduction of page replacement algorithms. Thrashing and its solution is addressed in details.

The eighth module is to introduce file systems. Emphasis is placed on the end user interface to a file system and basic file allocation methods. File systems implementation is not covered in this course.

The ninth module is to introduce the massive storage systems. Physical structure of secondary storage, performance characteristics of massive storage devices and operating-system services provided for massive storage are introduced. RAID techniques are discussed in details.

The tenth module is to introduce I/O interface. Emphasis is placed on device driver and system calls for I/O.

2.3 Laboratory Exercises

In companion with the lecture, there is a three-hour weekly laboratory section for this course in which students will gain hands-on experience with various operating system topics discussed in the lecture class. Other than having students do kernel development projects [2] or examine OS performance on a virtual simulator [3], we created lab assignments allowing students to focus on application development projects by efficiently using OS resources. Most of these labs are performed in UNIX and Windows XP environments. The following eight lab exercises we developed for this course are as follows.

A. Lab one is have students be familiar with UNIX operating system, and get hands-on experience with UNIX system programs. Students will learn how to log into the UNIX systems, how to navigate their file directories, how to run a program, how to list files, and how to edit files etc. They also learn how to edit, compile and run a c/c++ program in the UNIX system. Students will have a basic understanding of OS components.

B. Lab two is to teach students how to create a new process and lean how a program can replace its code with that of another executable file, and understand the lifetime of a process. By completing this lab, students should have a good understanding of concurrent programming using c/c++.

C. Lab three is to do a multithreading programming project. In this lab, students will be instructed on how to create a thread and execute a thread in both UNIX (using pthreads API) and Windows OS (Windows API). Students will investigate the difference of creating a thread and process.

D. Lab 4 allows students gain hands-on experience with process synchronization in UNIX. First, students are required to identify the synchronization problems the sample codes provided to them. Then, they are required to revise the sample code using Mutex and Semaphore techniques to solve the synchronization problems.

E. Similar to Lab 4, Lab 5 is designed to teach them thread synchronization in Windows OS. Samples codes were also provided. Students need to identify the problem and revise the sample codes to solve the synchronization problem.

F. Lab 6 is to have students understand the paging memory management technique. They need to write programs to convert a virtual address to a physical address. In addition, they learn how to configure Windows XP options (such as virtual memory size) to improve systems' performance.

G. Lab 7 is to demonstrate some of the aspects of practical file systems and learn how to perform operations on files in a UNIX system. A comparison of UNIX and Windows XP file systems is introduced. Naming, navigating, structure and file attributes are investigated.

H. Lab 8 is to review what system calls are, to demonstrate how they work and discuss the difference between a system call and library routine. Students are required to do projects using system calls to perform I/O operations.

3. Student Feedback and Teaching Reflection

Various methods were used to formally assess the effectiveness of this course, including tests, the evaluation of student work, and the instructor's assessment of laboratory work. The overall response from students regarding whether the course met their expectations was very positive. Summarized student comments are:

- This course presents interesting topics and helps them to learn new technologies.
- They have a better understanding of computer operating systems and how a computer system performance can be affected.
- It helps them to learn techniques to deal with multi-tasking application development. One student in this class was doing a senior design project of developing an ODB II code reader. He commented that the concurrent programming experience is particularly helpful to him.

Given the nature of this course, a few students still felt that it is abstract. This course can be tailored to focus more on multithreading programming. In addition, we are planning to add more practical application topics in this course, such as how to select operating systems for real-time embedded systems. Our future work for the laboratory assignments is to create a few real-time OS application projects for embedded systems. We believe it will be beneficial to students.

4. Conclusion

OS is one of the fundamental and core courses in computer science and computer engineering disciplines. There are a number of good reasons why this course should also be taught to computer engineering and electrical engineering technology students who are going to be seriously involved in computer systems. A good understanding of operating system will especially help CET/EET students working on application programming, computer system development and administration. We have used a problem-based learning approach to teach this course to our CET and EET students, and our experience is positive.

References

1. A. Silberschatz, P. Galvin, and G. Gagne, Operating System Concepts, 8th Edition, John Wiley & Sons, Inc.
2. W. Christopher, S. Procter and T. Anderson,The Nachos Instructional Operating System http://www.eecs.berkeley.edu/Pubs/TechRpts/1993/6022.html
3. S. Robbins, Simulators for Teaching Computer Science, http://vip.cs.utsa.edu/simulators/