Undergraduate Topics in Computer Science

Undergraduate Topics in Computer Science (UTiCS) delivers high-quality instructional content for undergraduates studying in all areas of computing and information science. From core foundational and theoretical material to final-year topics and applications, UTiCS books take a fresh, concise, and modern approach and are ideal for self-study or for a one- or two-semester course. The texts are all authored by established experts in their fields, reviewed by an international advisory board, and contain numerous examples and problems. Many include fully worked solutions.

Also in this series

Iain D. Craig
Object-Oriented Programming Languages: Interpretation
978-1-84628-773-2

Max Bramer
Principles of Data Mining
978-1-84628-765-7

Hanne Riis Nielson and Flemming Nielson
Semantics with Applications: An Appetizer
978-1-84628-691-9

Michael Kifer and Scott A. Smolka

Introduction to Operating System Design and Implementation

The *OSP 2* Approach

Springer

Michael Kifer, PhD
State University of New York
at Stony Brook, NY, USA

Scott A. Smolka, PhD
State University of New York
at Stony Brook, NY, USA

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Control Number: 2007926598

Undergraduate Topics in Computer Science ISSN 1863-7310
ISBN 978-1-84628-842-5 e-ISBN 978-1-84628-843-2

Printed on acid-free paper

9 8 7 6 5 4 3 2 1

Springer Science+Business Media
springer.com

Contents

Preface

OSP 2 is both an implementation of a modern operating system, and a flexible environment for generating implementation projects appropriate for an introductory course in operating system design. It is intended to complement the use of an introductory textbook on operating systems and contains enough projects for up to three semesters. These projects expose students to many essential features of operating systems, while at the same time isolating them from low-level machine-dependent concerns. Thus, even in one semester, students can learn about page replacement strategies in virtual memory management, cpu scheduling strategies, disk seek time optimization, and other issues in operating system design.

OSP 2 is written in the Java programming language and students program their *OSP 2* projects in Java as well. Therefore as prerequisites for using *OSP 2*, students are expected to have solid Java programming skills; be well-versed in object-oriented programming concepts such as classes, objects, methods, and inheritance; to have taken an undergraduate Computer Science course in data structures; and to have working knowledge of a Java programming environment, i.e., `javac`, `java`, text editing, etc. *OSP 2* is the successor to the original *OSP* software, which was released in 1990 and programmed in C.

OSP 2 consists of a number of modules, each of which performs a basic operating systems service, such as device scheduling, cpu scheduling, interrupt handling, file management, memory management, process management, resource management, and interprocess communication. Projects can be organized in any desired order so as to progress in a manner consistent with the lecture material. The *OSP 2* distribution comes with a reference Java implementation of each module, which is provided to the course instructor.

Each *OSP 2* project has a well-defined API (application programming interface), that the student must implement in order to successfully complete

the project. Thus, among other things, *OSP 2* teaches students to work with "open" environments where programming must be conducted to satisfy concrete sets of project requirements and where APIs must be used to interface to other subsystems.

Each *OSP 2* project consists of a "partial load module" of standard *OSP 2* modules to which the students link their implementation of the assigned modules. The result is a new and complete operating system, partially implemented by the student. Additionally, each project includes one or more "*.java" files, which contain class and method headings for each of the assigned modules. These files serve as *templates* in which the student is to fill in the code for the required methods. This ensures a consistent interface to *OSP 2* and eliminates much of the routine typing, both by the instructor and by the student.

The heart of *OSP 2* is a simulator that gives the illusion of a computer system with a dynamically evolving collection of user processes to be multiprogrammed. All the other modules of *OSP 2* are built to respond appropriately to the simulator-generated events that drive the operating system. The simulator "understands" its interaction with the other modules in that it can often detect an erroneous response by a module to a simulated event. In such cases, the simulator will gracefully terminate execution of the program by delivering a meaningful error message to the user, indicating where the error might be found. This facility serves both as a debugging tool for the student and as teaching tool for the instructor, as it ensures that student programs acceptable to the simulator are virtually bug-free. (Verification by the simulator does not, of course, replace the need to examine student programs to ensure that they are properly designed and acceptable from a software engineering point of view.)

The difficulty of the job streams generated by the simulator can be dynamically adjusted by manipulating the *simulation parameters*. This yields a simple and effective way of testing the quality of student programs. There are also facilities that allow students to debug their programs, including a detailed system log of events and various hooks into the system that allow student-provided methods to be called when an *OSP 2* warning or error is detected. Also, a graphical user interface (GUI) is available that provides a convenient way for students and instructors to enter simulation parameters and to view various statistics concerning the execution of *OSP 2*.

The underlying model in *OSP 2* is not a clone of any specific operating system. Rather it is an abstraction of the common features of several systems (although a bias towards Unix and the Mach operating systems can be seen, at times). Moreover, the *OSP 2* modules were designed to hide a number of low-level concerns, yet still encompass the most salient aspects of their real-life counterparts in modern systems. Their implementation is well-suited as the project component of an introductory course in operating systems.

How to Use this Book

This book is primarily a manual for students on how to program the *OSP 2* projects. Chapter 1 describes the overall organization of *OSP 2*. Chapter 2 takes the student through an example session with *OSP 2*. Each subsequent chapter constitutes a detailed description of one of the *OSP 2* projects, beginning with a statement of the goals of the project, followed by a short introduction to the basic OS concepts relevant to that chapter's subject matter. The latter is intended to help bridge the gap between the *OSP 2* manual and the course textbook. Before even the first assignment is handed out, students should read this Preface and Chapters 1 and 2. When a specific project is assigned (e.g. the thread-management project, project THREADS) the appropriate chapter (Chapter 4 in the case of THREADS) should be read carefully. Each project chapter provides a complete description of the API for the *OSP 2* module the students have been asked to implement, including a clear account of the functionality of each method in the project. Also provided is a list of methods from other project modules that may be needed to implement the project assignment. The student should refer to the relevant chapters for a more detailed account of these methods.

Goals of this Book

Besides serving as the student project manual for *OSP 2*, the goals of this book, and more broadly the *OSP 2* environment, are the following:

- To teach students fundamental operating system concepts in the following areas:
  - process and thread management
  - memory management
  - file systems
  - interprocess communication
  - I/O device management
  - resource management
- To give students the opportunity to practice these skills in a realistic operating systems programming environment.
- To provide students with challenging individual and group programming assignments which promote "active learning" to reinforce and amplify the

lecture material.

- ◇ To provide programming assignments that involve significant modifications to an actual, working operating system, thereby familiarizing students with the internals of OS implementation.
- ◇ To provide instructors with a flexible OS programming project that can easily accommodate their lecture schedule.

Acknowledgments

We would like to gratefully acknowledge the past members of the $\mathcal{OSP}$ 2 development team, including Sanford Barr, who produced the original design and implementation of the event engine; William Ries, Adam Sah and Tomek Retelewski, who, along with Sanford, designed and implemented an earlier version of $\mathcal{OSP}$ 2 that was written in C++; Fang Yang, who was responsible for porting the event engine and several other modules from the C++ version to Java; Kevin McDonnell and Peter Litskevitch, for designing, implementing and documenting most of the modules in the current version; Jingjing Wei, for implementing the latest configurable version of the GUI; Eric Nuzzi, who devised a systematic testing protocol for the $\mathcal{OSP}$ 2 code; Martin Bruggink, for implementing the PORTS module; Xiaohua Wu, for implementing the RESOURCES module; and David McManamon, for implementing the software that allows students to submit their solutions to $\mathcal{OSP}$ 2 assignments electronically.

Some parts of $\mathcal{OSP}$ 2 rely on third-party software. In particular, we thank Retrologic for developing their excellent Java obfuscator and releasing it under the Lesser Gnu Public License (LGPL).

Finally, we would like to thank Wayne Wheeler and Catherine Brett of Springer London Ltd for bringing their editorial expertise to bear on this project.