# APPENDIX B

## PROGRAMMING AND OPERATING SYSTEM PROJECTS

Many instructors believe that implementation or research projects are crucial to the clear understanding of operating system concepts. Without projects, it may be difficult for students to grasp some of the basic OS abstractions and interactions among components; a good example of a concept that many students find difficult to master is that of semaphores. Projects reinforce the concepts introduced in this book, give the student a greater appreciation of how the different pieces of an OS fit together, and can motivate students and give them confidence that they are capable of not only understanding but also implementing the details of an OS.

In this text, I have tried to present the concepts of OS internals as clearly as possible and have provided numerous homework problems to reinforce those concepts. Many instructors will wish to supplement this material with projects. This appendix provides some guidance in that regard and describes support material available in the **Instructor's Resource Center (IRC)** for this book accessible from Pearson for instructors. The support material covers eight types of projects and other student exercises:

- OS/161 projects
- Simulation projects
- Programming projects
- Research projects
- Reading/report assignments
- Writing assignments
- Discussion topics
- BACI

## B.1 OS/161

The **Instructor's Resource Center (IRC)** for this book provides support for using OS/161 as an active learning component.

OS/161 is an educational operating system developed at Harvard University [HOLL02]. It aims to strike a balance between giving students experience in working on a real operating system, and potentially overwhelming students with the complexity that exists in a fully fledged operating system, such as Linux. Compared to most deployed operating systems, OS/161 is quite small (approximately 20,000 lines of code and comments), and therefore it is much easier to develop an understanding of the entire code base.

The source code distribution contains a full operating system source tree, including the kernel, libraries, various utilities (ls, cat,…), and some test programs. OS/161 boots on the simulated machine in the same manner as a real system might boot on real hardware.

System/161 simulates a "real" machine to run OS/161 on. The machine features a MIPS R2000/R3000 CPU including an MMU, but no floating-point unit or cache. It also features simplified hardware devices hooked up to the system bus.

These devices are much simpler than real hardware, and thus make it feasible for students to get their hands dirty without having to deal with the typical level of complexity of physical hardware. Using a simulator has several advantages: Unlike other software students write, buggy OS software may result in completely locking up the machine, making it difficult to debug and requiring a reboot. A simulator enables debuggers to access the machine below the software architecture level as if debugging was built into the CPU. In some senses, the simulator is similar to an in-circuit emulator (ICE) that you might find in industry, only it is implemented in software. The other major advantage is the speed of reboots. Rebooting real hardware takes minutes, and hence the development cycle can be frustratingly slow on real hardware. System/161 boots OS/161 in mere seconds.

The OS/161 and System/161 simulators can be hosted on a variety of platforms, including Unix, Linux, Mac OS X, and Cygwin (the free Unix environment for Windows).

The IRC includes the following:

- **Package for instructor's Web server:** A set of html and pdf files that can be easily uploaded to the instructor's site for the OS course, which provides all the online resources for OS/161 and S/161 access, user's guides for students, assignments, and other useful material.
- **Getting started for instructors:** This guide lists all of the files that make up the Web site for the course and instructions on how to set up the Web site.
- **Getting started for students:** This guide explains to students step-by-step how to download and install OS/161 and S/161 on their PC.
- **Background material for students:** This consists of two documents that provide an overview of the architecture of S/161 and the internals of OS/161. These overviews are intended to be sufficient so that the student is not overwhelmed with figuring out what these systems are.
- **Student exercises:** A set of exercises that cover some of the key aspects of OS internals, including support for system calls, threading, synchronization, locks and condition variables, scheduling, virtual memory, files systems, and security.

The IRC OS/161 package was prepared by Andrew Peterson and other colleagues and students at the University of Toronto.

## B.2 SIMULATIONS

The IRC provides support for assigning projects based on a set of simulations developed at the University of Texas, San Antonio. Table B.1 lists the simulations by chapter. The simulators are all written in Java and can be run either locally as a Java application or online through a browser.

The IRC includes the following:

1. A brief overview of the simulations available.
2. How to port them to the local environment.

**Table B.1** OS Simulations by Chapter

| | |
|---|---|
| **Chapter 5 — Concurrency: Mutual Exclusion and Synchronization** | |
| Producer-consumer | Allows the user to experiment with a bounded buffer synchronization problem in the context of a single producer and a single consumer |
| UNIX Fork-pipe | Simulates a program consisting of `pipe`, `dup2`, `close`, `fork`, `read`, `write`, and `print` instructions |
| **Chapter 6 — Concurrency: Deadlock and Starvation** | |
| Starving philosophers | Simulates the dining philosophers problem |
| **Chapter 8 — Virtual Memory** | |
| Address translation | Used for exploring aspects of address translation. It supports 1- and 2-level page tables and a translation lookaside buffer |
| **Chapter 9 — Uniprocessor Scheduling** | |
| Process scheduling | Allows users to experiment with various process scheduling algorithms on a collection of processes and to compare such statistics as throughput and waiting time |
| **Chapter 11 — I/O Management and Disk Scheduling** | |
| Disk head scheduling | Supports the standard scheduling algorithms such as FCFS, SSTF, SCAN, LOOK, C-SCAN, and C-LOOK as well as double buffered versions of these |
| **Chapter 12 — File Management** | |
| Concurrent I/O | Simulates a program consisting of `open`, `close`, `read`, `write`, `fork`, `wait`, `pthread_create`, `pthread_detach`, and `pthread_join` instructions |

3. Specific assignments to give to students, telling them specifically what they are to do and what results are expected. For each simulation, this section provides one or two original assignments that the instructor can assign to students.

These simulation assignments were developed by Adam Critchley (University of Texas at San Antonio).

## B.3 PROGRAMMING PROJECTS

Three sets of programming projects are provided.

### Textbook-Defined Projects

Two major programming projects, one to build a shell, or command line interpreter, and one to build a process dispatcher are described in the online portion of the textbook. The projects can be assigned after Chapter 3 and after Chapter 9,

respectively. The IRC provides further information and step-by-step exercises for developing the programs.

These projects were developed by Ian G. Graham of Griffith University, Australia.

## Additional Major Programming Projects

A set of programming assignments, called machine problems (MPs), are available that are based on the Posix Programming Interface. The first of these assignments is a crash course in C, to enable the student to develop sufficient proficiency in C to be able to do the remaining assignments. The set consists of nine machine problems with different difficulty degrees. It may be advisable to assign each project to a team of two students.

Each MP includes not only a statement of the problem but a number of C files that are used in each assignment, step-by-step instructions, and a set of questions for each assignment that the student must answer that indicate a full understanding of each project. The scope of the assignments includes:

1. Create a program to run in a shell environment using basic I/O and string manipulation functions.
2. Explore and extend a simple Unix shell interpreter.
3. Modify faulty code that utilizes threads.
4. Implement a multithreaded application using thread synchronization primitives.
5. Write a user-mode thread scheduler.
6. Simulate a time-sharing system by using signals and timers.
7. A six-week project aimed at creating a simple yet functional networked file system. Covers I/O and file system concepts, memory management, and networking primitives.

The IRC provides specific instructions for setting up the appropriate support files on the instructor's Web site of local server.

These project assignments were developed at the University of Illinois at Urbana-Champaign, Department of Computer Science and adapted by Matt Sparks (University of Illinois at Urbana-Champagne) for use with this textbook.

## Small Programming Projects

The instructor can also assign a number of small programming projects described in the IRC. The projects can be programmed by the students on any available computer and in any appropriate language: They are platform and language independent.

These small projects have certain advantages over the larger projects. Larger projects usually give students more of a sense of achievement, but students with less ability or fewer organizational skills can be left behind. Larger projects usually elicit more overall effort from the best students. Smaller projects can have a higher concepts-to-code ratio, and because more of them can be assigned, the opportunity exists to address a variety of different areas. Accordingly, the instructor's IRC

contains a series of small projects, each intended to be completed in a week or so, which can be very satisfying to both student and teacher. These projects were developed at Worcester Polytechnic Institute by Stephen Taylor, who has used and refined the projects in the course of teaching operating systems a dozen times.

## B.4 RESEARCH PROJECTS

An effective way of reinforcing basic concepts from the course and for teaching students research skills is to assign a research project. Such a project could involve a literature search as well as a Web search of vendor products, research lab activities, and standardization efforts. Projects could be assigned to teams or, for smaller projects, to individuals. In any case, it is best to require some sort of project proposal early in the term, giving the instructor time to evaluate the proposal for appropriate topic and appropriate level of effort. Student handouts for research projects should include

- A format for the proposal
- A format for the final report
- A schedule with intermediate and final deadlines
- A list of possible project topics

The students can select one of the listed topics or devise their own comparable project. The IRC includes a suggested format for the proposal and final report as well as a list of possible research topics developed by Professor Tan N. Nguyen of George Mason University.

## B.5 READING/REPORT ASSIGNMENTS

Another excellent way to reinforce concepts from the course and to give students research experience is to assign papers from the literature to be read and analyzed. The IRC includes a suggested list of papers to be assigned, organized by chapter. A PDF copy of each of the papers is available at box.com/OS8e. The IRC also includes a suggested assignment wording.

## B.6 WRITING ASSIGNMENTS

Writing assignments can have a powerful multiplier effect in the learning process in a technical discipline such as OS internals. Adherents of the Writing Across the Curriculum (WAC) movement (http://wac.colostate.edu/) report substantial benefits of writing assignments in facilitating learning. Writing assignments lead to more detailed and complete thinking about a particular topic. In addition, writing assignments help to overcome the tendency of students to pursue a subject with a minimum

of personal engagement, just learning facts and problem-solving techniques without obtaining a deep understanding of the subject matter.

The IRC contains a number of suggested writing assignments, organized by chapter. Instructors may ultimately find that this is an important part of their approach to teaching the material. I would greatly appreciate any feedback on this area and any suggestions for additional writing assignments.

## B.7 DISCUSSION TOPICS

One way to provide a collaborative experience is discussion topics, a number of which are included in the IRC. Each topic relates to material in the book. The instructor can set it up so that students can discuss a topic either in a class setting, an online chat room, or a message board. Again, I would greatly appreciate any feedback on this area and any suggestions for additional discussion topics.

## B.8 BACI

In addition to all of the support provided at the IRC, the Ben-Ari Concurrent Interpreter (BACI) is a publicly available package that instructors may wish to use. BACI simulates concurrent process execution and supports binary and counting semaphores and monitors. BACI is accompanied by a number of project assignments to be used to reinforce concurrency concepts.

Appendix O provides a more detailed introduction to BACI, with information about how to obtain the system and the assignments.