CHAPTER 2 OPERATING SYSTEM OVERVIEW

ANSWERS TO QUESTIONS

2.1 **Convenience:** An operating system makes a computer more convenient to use. **Efficiency:** An operating system allows the computer system resources to be used in an efficient manner. **Ability to evolve:** An operating system should be constructed in such a way as to permit the effective development, testing, and introduction of new system functions without interfering with service.

2.2 The kernel is a portion of the operating system that includes the most heavily used portions of software. Generally, the kernel is maintained permanently in main memory. The kernel runs in a privileged mode and responds to calls from processes and interrupts from devices.

2.3 Multiprogramming is a mode of operation that provides for the interleaved execution of two or more computer programs by a single processor.

2.4 A process is a program in execution. A process is controlled and scheduled by the operating system.

2.5 The **execution context,** or **process state,** is the internal data by which the operating system is able to supervise and control the process. This internal information is separated from the process, because the operating system has information not permitted to the process. The context includes all of the information that the operating system needs to manage the process and that the processor needs to execute the process properly. The context includes the contents of the various processor registers, such as the program counter and data registers. It also includes information of use to the operating system, such as the priority of the process and whether the process is waiting for the completion of a particular I/O event.

2.6 **Process isolation:** The operating system must prevent independent processes from interfering with each other's memory, both data and instructions. **Automatic allocation and management:** Programs

should be dynamically allocated across the memory hierarchy as required. Allocation should be transparent to the programmer. Thus, the programmer is relieved of concerns relating to memory limitations, and the operating system can achieve efficiency by assigning memory to jobs only as needed. **Support of modular programming:** Programmers should be able to define program modules, and to create, destroy, and alter the size of modules dynamically. **Protection and access control:** Sharing of memory, at any level of the memory hierarchy, creates the potential for one program to address the memory space of another. This is desirable when sharing is needed by particular applications. At other times, it threatens the integrity of programs and even of the operating system itself. The operating system must allow portions of memory to be accessible in various ways by various users. **Long-term storage:** Many application programs require means for storing information for extended periods of time, after the computer has been powered down.

2.7 A **virtual address** refers to a memory location in virtual memory. That location is on disk and at some times in main memory. A real address is an address in main memory.

2.8 Round robin is a scheduling algorithm in which processes are activated in a fixed cyclic order; that is, all processes are in a circular queue. A process that cannot proceed because it is waiting for some event (e.g. termination of a child process or an input/output operation) returns control to the scheduler.

2.9 A **monolithic kernel** is a large kernel containing virtually the complete operating system, including scheduling, file system, device drivers, and memory management. All the functional components of the kernel have access to all of its internal data structures and routines. Typically, a monolithic kernel is implemented as a single process, with all elements sharing the same address space. A **microkernel** is a small privileged operating system core that provides process scheduling, memory management, and communication services and relies on other processes to perform some of the functions traditionally associated with the operating system kernel.

2.10 Multithreading is a technique in which a process, executing an application, is divided into threads that can run concurrently.

2.11 Simultaneous concurrent processes or threads; scheduling; synchronization; memory management; reliability and fault tolerance.

ANSWERS TO PROBLEMS

2.1 The answers are the same for **(a)** and **(b)**. Assume that although processor operations cannot overlap, I/O operations can.

Number of jobs	TAT	Throughput	Processor utilization
1	NT	1/N	50%
2	NT	2/N	100%
4	(2N – 1)T	4/(2N – 1)	100%

2.2 I/O-bound programs use relatively little processor time and are therefore favored by the algorithm. However, if a processor-bound process is denied processor time for a sufficiently long period of time, the same algorithm will grant the processor to that process since it has not used the processor at all in the recent past. Therefore, a processor-bound process will not be permanently denied access.

2.3 With time sharing, the concern is turnaround time. Time-slicing is preferred because it gives all processes access to the processor over a short period of time. In a batch system, the concern is with throughput, and the less context switching, the more processing time is available for the processes. Therefore, policies that minimize context switching are favored.

2.4 A system call is used by an application program to invoke a function provided by the operating system. Typically, the system call results in transfer to a system program that runs in kernel mode.

2.5 The system operator can review this quantity to determine the degree of "stress" on the system. By reducing the number of active jobs allowed on the system, this average can be kept high. A typical guideline is that this average should be kept above 2 minutes. This may seem like a lot, but it isn't.

2.6 **a.** If a conservative policy is used, at most 20/4 = 5 processes can be active simultaneously. Because one of the drives allocated to each process can be idle most of the time, at most 5 drives will be idle at a time. In the best case, none of the drives will be idle.

b. To improve drive utilization, each process can be initially allocated with three tape drives. The fourth one will be allocated on demand. In this policy, at most $\lfloor 20/3 \rfloor = 6$ processes can be active simultaneously. The minimum number of idle drives is 0 and the maximum number is 2.

CHAPTER 3 PROCESS DESCRIPTION AND CONTROL

ANSWERS TO QUESTIONS

3.1 An instruction trace for a program is the sequence of instructions that execute for that process.

3.2 New batch job; interactive logon; created by OS to provide a service; spawned by existing process. See Table 3.1 for details.

3.3 **Running:** The process that is currently being executed. **Ready:** A process that is prepared to execute when given the opportunity. **Blocked:** A process that cannot execute until some event occurs, such as the completion of an I/O operation. **New:** A process that has just been created but has not yet been admitted to the pool of executable processes by the operating system. **Exit:** A process that has been released from the pool of executable processes by the operating system, either because it halted or because it aborted for some reason.

3.4 Process preemption occurs when an executing process is interrupted by the processor so that another process can be executed.

3.5 Swapping involves moving part or all of a process from main memory to disk. When none of the processes in main memory is in the Ready state, the operating system swaps one of the blocked processes out onto disk into a suspend queue, so that another process may be brought into main memory to execute.

3.6 There are two independent concepts: whether a process is waiting on an event (blocked or not), and whether a process has been swapped out of main memory (suspended or not). To accommodate this 2×2 combination, we need two Ready states and two Blocked states.

3.7 **1.** The process is not immediately available for execution. **2.** The process may or may not be waiting on an event. If it is, this blocked condition is independent of the suspend condition, and occurrence of the blocking event does not enable the process to be executed. **3.** The process was placed in a suspended state by an agent; either itself, a

Note: a number of the files related to the projects are at an **instructor support folder (ISF)** at box.com/OS9e-Projects. This document, plus some other files are at the **Instructor Resource Center (IRC)** for this book maintained by Pearson.

PART 1 SEMAPHORE PROJECTS

The files needed for these projects are in the folder **`Semaphore-Projects`** at the ISF.

PART 2 OS/161

OS/161 is an educational operating system developed at Harvard University. 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 ISF includes the following in the folder `OS161` at box.com/OS9e-Projects:

- **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. The folder `OS161/web` contains a webpage shell that links to all of the documents the students need.
- **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, at `OS161/instructor_notes.shtml`.

- **Getting started for students:** This guide explains to students step by step how to download, install, and debug OS/161 and S/161 on their computer, at `OS161/web/docs/start_guide.shtml`.
- **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, at `OS161/web/docs/manual/shtml`.
- **Student exercises:** A set of exercises that cover some of the key aspects of OS internals, include support for system calls, threading, synchronization, locks and condition variables, scheduling, virtual memory, files systems and security, at `OS161/exercises`. A problem statement and a solution outline are provided for each exercise.

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

PART 3 SIMULATION PROJECTS

The folder **Semaphore-Projects** at the ISF provides support for assigning projects based on a set of simulations developed at the University of Texas, San Antonio. Table 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 folder includes the following:

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

All of the documentation and support material are contained in the folder `Simulation-Projects` at box.com/OS9e-Projects, which contains three folders:

- `SimulationAssignments` contains seven documents, each containing the instructions for the student for one of the simulations.
- `SimulationAnswers` contains expected answers to be provided by the student.
- `SimulationWebPage` contains the html files needed to set up a simulation Web page on your server. Alternatively, you can direct the students to http://williamstallings.com/OS/Simulations.html

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

Table 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

PART 4 PROGRAMMING PROJECTS

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 and provided in the ISF. The projects can be assigned after Chapter 3 and after Chapter 9, respectively.

The ISF provides a set exercises designed to provide incremental solutions to the projects, with each successive exercise discussing different aspects of the project. Both projects are designed around using the C language on a UNIX platform with descriptions of all the major system functions that are required being supplied as well as a comprehensive C Standard Library reference. Progressive solutions to the projects for each exercise are supplied separately, as are fully functional final project programs and marking scripts to evaluate student solutions.

The project and exercise documentation is provided as a complete Web site with all documents in HTML. The site can be used as is or it can be customized for use by individual instructors. The suite was built using Adobe Dreamweaver and uses the Template and Library features of that application to enable bulk changes to format and repeated content, titles etc.

All of the documentation and support material are contained in the folder `OS9e-Text-Projects` at the ISF.

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.

All of the documentation and support material are contained in the folder `OS-Programming-Projects` in the ISF, which contains three folders:

- `mps`: This folder can be uploaded directly to your Web server. It contains the index.html file that is the Web page seen by the student, plus all of the files for the machine problems. Each problem consists of a README.txt file, a compressed zip file and a compressed tar.gz file. Once the student downloads and decompresses the file for a particular MP, he or she will have all the files needed for that assignment.
- `mps_source`: All uncompressed versions of the MPs and their solutions.

- `mps_solutions`: This folder can also be uploaded directly to your Web server. As with the mps folder, it contains the index.html file that is the Web page seen by the student, plus compressed, downloadable files containing solutions for each MP. **If you choose to provide the solutions to the students by this means, please make use of a password protected Web page, so that the solution files are not publicly available on the Internet.**

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

Smaller Programming Projects

Steve Taylor of Worcester Polytechnic Institute has developed a set of 9 programming projects, each of which could be done in about one week. The project definitions are contained in the file `Programming-Projects.doc`.