EDWARD G. COFFMAN, JR.
PETER J. DENNING

Operating Systems Theory

PRENTICE-HALL SERIES IN AUTOMATIC COMPUTATION

OPERATING SYSTEMS THEORY

EDWARD G. COFFMAN, JR.
Pennsylvania State University

PETER J. DENNING
Purdue University

PRENTICE-HALL, INC.
ENGLEWOOD CLIFFS, NEW JERSEY

Library of Congress Cataloging in Publication Data

COFFMAN, EDWARD GRADY.
Operating systems theory.

(Prentice-Hall series in automatic computation)
Includes bibliographical references.
1. Electronic digital computers—Programming.
2. Algorithms. I. Denning, Peter J., joint author.
II. Title.
QA76.6.C62 001.6'42 73-18
ISBN 0-13-637868-4

10 9 8 7 6 5 4 3 2 1

Printed in the United States of America

PRENTICE-HALL INTERNATIONAL, INC., *London*
PRENTICE-HALL OF AUSTRALIA, PTY. LTD., *Sydney*
PRENTICE-HALL OF CANADA, LTD., *Toronto*
PRENTICE-HALL OF INDIA PRIVATE LIMITED, *New Delhi*
PRENTICE-HALL OF JAPAN, INC., *Tokyo*

OPERATING SYSTEMS THEORY

Prentice-Hall
Series in Automatic Computation

George Forsythe, editor

AHO, editor, *Currents in the Theory of Computing*
AHO AND ULLMAN, *Theory of Parsing, Translation, and Compiling, Volume I: Parsing; Volume II: Compiling*
(ANDREE)^3, *Computer Programming: Techniques, Analysis, and Mathematics*
ANSELONE, *Collectively Compact Operator Approximation Theory and Applications to Integral Equations*
ARBIB, *Theories of Abstract Automata*
BATES AND DOUGLAS, *Programming Language/One*, 2nd ed.
BLUMENTHAL, *Management Information Systems*
BRENT, *Algorithms for Minimization without Derivatives*
COFFMAN AND DENNING, *Operating Systems Theory*
CRESS, et al., *FORTRAN IV with WATFOR and WATFIV*
DAHLQUIST, et al., *Numerical Methods*
DANIEL, *The Approximate Minimization of Functionals*
DEO, *Graph Theory with Applications to Engineering and Computer Science*
DESMONDE, *Computers and Their Uses*, 2nd ed.
DESMONDE, *Real-Time Data Processing Systems*
DRUMMOND, *Evaluation and Measurement Techniques for Digital Computer Systems*
EVANS, et al., *Simulation Using Digital Computers*
FIKE, *Computer Evaluation of Mathematical Functions*
FIKE, *PL/1 for Scientific Programmers*
FORSYTHE AND MOLER, *Computer Solution of Linear Algebraic Systems*
GAUTHIER AND PONTO, *Designing Systems Programs*
GEAR, *Numerical Initial Value Problems in Ordinary Differential Equations*
GOLDEN, *FORTRAN IV Programming and Computing*
GOLDEN AND LEICHUS, *IBM/360 Programming and Computing*
GORDON, *System Simulation*
HARTMANIS AND STEARNS, *Algebraic Structure Theory of Sequential Machines*
HULL, *Introduction to Computing*
JACOBY, et al., *Iterative Methods for Nonlinear Optimization Problems*
JOHNSON, *System Structure in Data, Programs, and Computers*
KANTER, *The Computer and the Executive*
KIVIAT, et al., *The SIMSCRIPT II Programming Language*
LORIN, *Parallelism in Hardware and Software: Real and Apparent Concurrency*
LOUDEN AND LEDIN, *Programming the IBM 1130*, 2nd ed.
MARTIN, *Design of Man–Computer Dialogues*
MARTIN, *Design of Real-Time Computer Systems*
MARTIN, *Future Developments in Telecommunications*
MARTIN, *Programming Real-Time Computing Systems*
MARTIN, *Security, Accuracy and Privacy in Computer Systems*
MARTIN, *Systems Analysis for Data Transmission*
MARTIN, *Telecommunications and the Computer*

MARTIN, *Teleprocessing Network Organization*
MARTIN AND NORMAN, *The Computerized Society*
MATHISON AND WALKER, *Computers and Telecommunications: Issues in Public Policy*
MCKEEMAN, et al., *A Compiler Generator*
MEYERS, *Time-Sharing Computation in the Social Sciences*
MINSKY, *Computation: Finite and Infinite Machines*
NIEVERGELT, et al., *Computer Approaches to Mathematical Problems*
PLANE AND MCMILLAN, *Discrete Optimization: Integer Programming and Network Analysis for Management Decisions*
PRITSKER AND KIVIAT, *Simulation with GASP II: a FORTRAN-Based Simulation Language*
PYLYSHYN, editor, *Perspectives on the Computer Revolution*
RICH, *Internal Sorting Methods Illustrated with PL/1 Programs*
RUSTIN, editor, *Algorithm Specification*
RUSTIN, editor, *Computer Networks*
RUSTIN, editor, *Data Base Systems*
RUSTIN, editor, *Debugging Techniques in Large Systems*
RUSTIN, editor, *Design and Optimization of Compilers*
RUSTIN, editor, *Formal Semantics of Programming Languages*
SACKMAN AND CITRENBAUM, editors, *On-Line Planning: Towards Creative Problem-Solving*
SALTON, editor, *The SMART Retrieval System: Experiments in Automatic Document Processing*
SAMMET, *Programming Languages: History and Fundamentals*
SCHAEFER, *A Mathematical Theory of Global Program Optimization*
SCHULTZ, *Spline Analysis*
SCHWARZ, et al., *Numerical Analysis of Symmetric Matrices*
SHERMAN, *Techniques in Computer Programming*
SIMON AND SIKLOSSY, editors, *Representation and Meaning: Experiments with Information Processing Systems*
STERBENZ, *Floating-Point Computation*
STERLING AND POLLACK, *Introduction to Statistical Data Processing*
STOUTEMYER, *PL/1 Programming for Engineering and Science*
STRANG AND FIX, *An Analysis of the Finite Element Method*
STROUD, *Approximate Calculation of Multiple Integrals*
TAVISS, editor, *The Computer Impact*
TRAUB, *Iterative Methods for the Solution of Polynomial Equations*
UHR, *Pattern Recognition, Learning, and Thought*
VAN TASSEL, *Computer Security Management*
VARGA, *Matrix Iterative Analysis*
WAITE, *Implementing Software for Non-Numeric Application*
WILKINSON, *Rounding Errors in Algebraic Processes*
WIRTH, *Systematic Programming: An Introduction*

PREFACE

Motivations for Studying Operating Systems Theory

In the years since 1969, the study of computer systems has assumed a role nearly equal in importance to "theory of computation" and "programming" in computer science curricula. In contrast, the subject of computer operating systems was regarded as recently as 1965 as being inferior in importance to these two traditional areas of study. This is a significant change in attitude. The first signs of the change are evidenced in ACM's *Curriculum 68*,† and the speed of its development is evidenced in the report of Task Force VIII of the COSINE (computer science in engineering) Committee of the Commission on Education of the National Academy of Engineering, *An Undergraduate Course on Operating Systems Principles* (June 1971).‡ There are several important reasons for this change.

First, three practical objectives—improving existing and future designs, building systems whose correctness and behavior can be determined a priori, and solving the resource allocation and performance evaluation problems—have stimulated increasing amounts of research in computer system modeling. A principal result of this effort has been the emergence of a "theory" of computer operating system design and analysis. This in turn is having an almost immediate impact on curricula: The traditional "case-study" approach to teaching operating systems concepts, which never proved to be an outstanding success, is giving way rapidly to the "modeling and analysis" approach.

Second, the problems of designing complex software systems have traditionally been considered of less intellectual interest than "theory of com-

†*Comm. ACM*. March 1968.

‡Commission on Education, National Academy of Engineering, 2102 Constitution Avenue, Washington, D.C. 20418.

putation" and "programming." The so-called software problem, i.e., reducing the high cost of software development and improving quality control over software packages, has been found to be increasingly severe. As a result, tradition is being reversed, there is rising interest in software system design as a deep intellectual problem. Much of "computer system theory" is related in one way or another to understanding and managing complexity in software systems.

Third, there is an ever widening appreciation of the view that the world we live in has become a real-time system whose complexity is beyond the reach of the unaided human mind to understand. Again and again we see decisions taken in business systems, economic systems, social systems, and urban systems—all real-time information systems—which, despite the best of intentions, often turn out to have effects quite unlike those intended. This phenomenon is not new in the experience of operating systems designers. Since computer-system "theorists" are deeply involved in the problems of managing complex real-time information systems in order to get them behaving as intended, this subject material appears destined ultimately to have an impact not restricted to the computer industry.

Purpose of this Book

The principal object of this book is studying algorithms arising in the design of computer operating systems. The study includes specifically sequencing and control algorithms designed to avoid various types of failures in systems supporting concurrent processes, scheduling algorithms designed to minimize total execution times and mean flow times, algorithms for allocating processors among multiprogrammed tasks, algorithms for using input/output devices, and algorithms for managing storage. These algorithms are studied from a formal view. In studying a given problem, for example, we shall discuss methods of devising mathematical models for the system and algorithms of interest; we shall then work out analyses whose goals are proofs of optimality, derivations of performance measures, or demonstrations that the systems or algorithms have certain desirable properties.

Consistent with this theme, our educational goal is presenting in one place, in as simple a form as possible, the most important formal methods that have been applied to the study of operating systems algorithms. Our interest is explicating the nature of the results, the essence of the analysis, and the power and limitations of the methods. In many cases we have chosen the simplest form extant of a model; we have done this whenever the additional generality would have multiplied the complexity of analysis beyond the value of whatever additional insight would have been gained. The book will succeed in its basic purpose to the extent that a reader moderately experienced in operating system design is enabled to examine a given operating system

and successfully to execute a project of (formal) modeling and analysis with respect to that system.

There are two broad subject areas that we have consciously avoided in our treatment: heuristic and experimental methods. (Indeed, the inclusion of these topics and the corresponding results would surely have doubled the size of the book.) Our exclusion of heuristic and exhaustive search methods for combinatorial problems in scheduling is largely justified by their excellent treatment in Conway, Maxwell, and Miller.† Our exclusion of experimental results relating to storage management is justified by their extensive coverage in the literature.

Experimental work concerned with simulation studies and statistical analysis of system performance (and program behavior) interfaces directly with the content of this book and would constitute a book on its own right. However, our use of models of program and computer-use behavior proceeds only as far as it is necessary to study the properties of specific algorithms. Important and well-known work in modeling program behavior includes graph models of parallelism and locality models of storage referencing. In the course of the book we shall have numerous occasions to reference the various engineering studies that support the assumptions made in the mathematical models.

We have arranged the presentation to be useful both to professionals as a reference and to students as a text. The reader is assumed to be familiar with the concepts, systems, and techniques acquired in the core of a computer science curriculum. For this reason we have omitted lengthy motivations for the models devised and extensive interpretation of the results obtained. The reader is expected to have a mathematical maturity corresponding roughly to a senior undergraduate or beginning graduate student. More specifically, a basic facility with formal systems is assumed along with a knowledge of the elements of applied probability theory and Markov chains. We have included a brief review of the latter material in an appendix.

Unfortunately, one or both of these latter assumptions is very frequently not valid when and where it should be. If it is agreed that a major part of computer science education is the study of algorithms—how to design and write them clearly, how to design optimal or efficient algorithms, and how to assess their performance—it follows from the nature and applicability of probability models that the student must have achieved some competence in applied probability theory and combinatorics. Until he acquires such skills, the student will not generally be capable of designing nontrivial algorithms and communicating to others precisely what he has done—in particularly designing proofs that show his algorithm to have certain properties and

†R. W. Conway, W. L. Maxwell, and L. W. Miller, *Theory of Scheduling* (Reading, Mass.: Addison-Wesley), 1967.

derivations of measures that can be used to judge the effectiveness and performance of his algorithm.

At the end of each chapter we have included a selection of problems relating to extensions or modifications of the material in that chapter. In the majority of cases these are *problems*, rather than *exercises*. It is especially important that the reader examine them, for their purpose is not only to improve the understanding of material in the text, but also to fill out the coverage of the book. There are many problems that extend the methods treated in the various chapters to different but closely related applications not otherwise discussed.

Plan of the Book

According to the COSINE report there are six aspects of computer system modeling in which useful abstractions and theoretical developments have evolved:

Procedures and their implementation
Management of named objects
Protection
Concurrent processes
Management of memory hierarchies
Resource allocation

Of these six sets of concepts, the first three tend to be more descriptive in nature, i.e., the abstractions do not involve any mathematics and are used by designers for immediate guidance in implementations. The last three sets of concepts do, however, rely on mathematical analysis before they produce useful results. Accordingly, we have restricted attention to these three sets of abstractions in this book.

One can identify additional areas in which modeling and analysis is highly desirable; as of 1972, however, there are few results of practical interest available, so we have omitted any treatment of them. They include system reliability and integrity, system performance evaluation, and design methodologies.

The book consists of seven chapters. A brief description of each follows.

Chapter 1. Introduction is an outline of the physical properties of the systems in which the results of our analyses in later chapters can be applied. This includes a discussion of the relevant properties of processor and memory devices; of the implementation features of virtual memory, especially paging; of the general aspects of the memory management and processor scheduling problems; and of the motivations for using concepts of parallelism, scheduling, paging, resource pooling, and program behavior in systems design.

Chapter 2. Control of Concurrent Processes contains a formalism for studying the important problems of controlling parallel processes, viz.,

determinacy, deadlock, mutual exclusion, and synchronization. We have been able to formulate and study these problems in the context of a single model, a partially-ordered system of tasks. While this model is by no means the most general studied previously, it exhibits the properties one would expect and desire in a practical system. The results of this chapter are: a) the task system model allows a uniform discussion of the four control problems, making evident the differences and similarities among them; b) the task system model permits simple proofs of the determinacy results; c) the deadlock results extend those available in the literature; and d) the synchronization results are all new and demonstrate the generality and power of the synchronizing primitives.

Chapter 3. Deterministic Models of Processor Scheduling is a nearly complete treatment of the results presently available on this subject. Given an n-task system in which the execution time of each task is known, and given k processors, the problem is finding a schedule (assignment of tasks to processors) that completes in minimum time, or minimizes mean flow time, and is consistent with the precedence constraints among tasks. Problems of this type are important not only in the classical job-shop environment, but also in future environments where, for example, task systems of the type studied in Chapter 2 will be implemented.

Chapter 4. Probability Models of Computer Sequencing Problems contains a review of basic queueing processes and their application to scheduling tasks in multiprogramming systems. We have attempted to present a selfcontained treatment in as short a space as possible. A major goal achieved in this chapter is an analysis of the basic computer priority queues.

Chapter 5. Auxiliary and Buffer Storage Models treats problems arising particularly in connection with input or output processes. The methods developed in Chapter 4 are used extensively. The chapter includes a study of the good (and bad) points of "shortest-latency-time-first" policies for serving queues on rotating storage devices (disks, drums). It includes an analysis of the buffering problem, showing the tremendous advantages inherent in pooled buffers as opposed to private buffers. These results have important implications with respect to any pooled resource, especially the partitioning of main memory among tasks under multiprogramming. The chapter includes a treatment of cyclic queue networks.

Chapter 6. Storage Allocation in Paging Systems is a fairly complete treatment of the results known for controlling and analyzing the page traffic resulting from given demand paging algorithms managing a single program in a fixed memory space. All the known results about optimal algorithms are included, as well as a new treatment of the important "stack algorithm" concept. The relation between these results and multiprogramming is studied.

Chapter 7. Multiprogrammed Memory Management specifically deals with the properties of program-behavior models that exhibit "locality of reference" and their implications with respect to multiprogrammed memory management. The properties of fixed and variable partitioning strategies of multiprogramming are treated. The "working set model" is studied in its own right, various important relations among working set size, paging rate, and page reference patterns being obtained.

ACKNOWLEDGMENTS

There are many people whose help and guidance were instrumental in the preparation of this book. Two in particular to whom we must express our special gratitude are the Prentice-Hall Computer Science Series Editor Karl V. Karlstrom, for his continuing patience and encouragement, and Richard R. Muntz, for the many corrections and helpful suggestions resulting from a painstaking reading of the manuscript.

Others whom we should like to acknowledge for their constructive criticisms of portions of the manuscript are Robert Butler, Robert M. Keller, Stephen R. Kimbleton, W. Frank King III, John E. Pomeranz, Barbara Ryan, John Bruno, Vincent Shen, and Jeffrey R. Spirn.

Very special thanks are due Miss Doris Rice whose speed and efficiency made the clerical aspects of preparing this book almost routine. Thanks are due also to Mrs. Hannah Kresse for help in this regard.

Finally, the Pennsylvania State University and Princeton University are to be acknowledged for their implicit support and for stimulating environments in which to write this book. The National Science Foundation under grant GJ-28290 provided partial financial support.

EDWARD G. COFFMAN, JR.

PETER J. DENNING

OPERATING SYSTEMS THEORY

CONTENTS

1 INTRODUCTION

1.1. OPERATING SYSTEMS

The era of electronic computing has been characterized as a series of "generations" [1], the first covering the period 1946–1950, the second covering the period 1950–1964, and the third covering the period since 1964. Although the term generation was intended originally to suggest differences in hardware technology, it has come to be applied to the entire hardware-software system rather than the hardware alone [2]. The development of general-purpose complex software systems did not begin until the third generation and has motivated the development of theoretical approaches to design and resource allocation.

As will be discussed in detail later, the term "process" is used to denote a program in execution. A computer system may be defined in terms of the various supervisory and control functions it provides for the processes created by its users:

1. Creating and removing processes.
2. Controlling the progress of processes, i.e., ensuring that each logically enabled process makes progress and that no process can block indefinitely the progress of others.
3. Acting on exceptional conditions arising during the execution of a process, e.g., arithmetic or machine errors, interrupts, addressing errors, illegal or privileged instructions, or protection violations.
4. Allocating hardware resources among processes.
5. Providing access to software resources, e.g., files, editors, compilers, assemblers, subroutine libraries, and programming systems.