SYSTEMS PROGRAMMING

Concepts of Operating and Data Base Systems

DAVID K. HSIAO
The Ohio State University

ADDISON-WESLEY PUBLISHING COMPANY
Reading, Massachusetts · Menlo Park, California
London · Amsterdam · Don Mills, Ontario · Sydney

This book is in the
ADDISON-WESLEY SERIES IN
COMPUTER SCIENCE AND INFORMATION PROCESSING

Consulting Editor
Michael A. Harrison

ISBN 0-201-02950-2
DEFGHIJKL-HA-798

In memory of my grandfather, Hsiao Chio-T'ien 蕭覺天 (1887–1972)

Foreword

The purpose of this book is to discuss general concepts and prevailing notions in systems programming. To this end, we have tried to present the concepts and notions abstractly. Via abstraction, we hope that the basic concepts and notions can be more easily visualized and understood without having detailed expositions of the material involved. Occasionally, we indulge ourselves in closer examination of the systems discussed. In these cases, the aim is to relate our abstract study with real-world problems and to illustrate the feasibility of implementing some of these concepts and notions.

By systems programming we mean the synthesis and analysis of general-purpose operating systems. In this book, the general-purpose operating systems are restricted to assembly systems, input-output control systems, batched processing systems, multiprogramming systems, and data base management systems. There are, of course, many other types of systems worthy of study, e.g., the real-time systems. Due to the lack of time or expertise in developing additional material, we have decided to exclude them from this presentation.

Despite some outstanding theoretical work in systems programming, there is a lack of general theory of systems programming. Thus, we have not strived for a theoretical treatment. Instead, in synthesis and analysis we attempt to separate a whole system into its logical components and to examine the relations among the components. In the process, we identify the components and clarify their roles in the system. Quite frequently we make certain assumptions about the functionality of the system and proceed to substantiate them by demonstrating the logic of the components. We also try to produce a coherent system by logical combination of functionally different components. Whether it is in analysis or synthesis, we always try to select a view or model which tends to provide a better framework for perceiving the complexity of the system. The layered, or structured, approach to input and output control systems and the hierarchical view of off-line batch processing systems are cases in point. These views are, of course, by no means the only views leading toward better system perception for analysis and synthesis.

The advantages of the structural view in synthesis and analysis of a system is that each layer or level of the hierarchy can be dealt with independently, and its relation with other layers or levels of the hierarchy is well-structured. However, there are other views. The finite-state automata and transition-diagrams approach toward the synthesis and analysis of operating systems represents one such view. Here the analysis and synthesis of the system become the identification and clarification of the system in terms of states, transitions of states, physical signals which cause the transitions, and logical signals which hasten the transitions. Logical signals and their generators constitute the heart of systems programming logic. Although the view of an operating system as a finite-state automaton or a transition diagram is not yet in vogue, it may have great potential. We have used this view toward the realization of an input-output control system, and in the discussion of an on-line programming system. We also employ a view from directed graphs to perceive the symbol definition process of assembly systems.

When perceptive views are not available, we rely on generalized models. These models are idealized and are mostly abstract. There are several advantages in developing the generalized models. First, many seemingly diverse and unrelated ideas can be consolidated into a coherent presentation. Second, few working systems have all the needed features for discussion. By incorporating those systems features which underly the principles of these systems, the model can assist in obtaining a better understanding of the systems as a whole. Third, there is less restriction on the nomenclature and conventions used in presenting these systems since the model system need not be the exact image of a working system. This modeling approach has been applied to assembly systems, multiprogramming systems, and data base management systems. Because assembly systems deal with symbols and data base management systems are concerned mainly with logical resources in the form of fields, records, and files, they lend themselves more readily to abstract modeling. On the other hand, multiprogramming systems, which manage physical resources, have somewhat eluded abstraction.

The book is therefore intended as a frame of reference for concepts and notions. We hope that these views and models can provide proper perspective and good insights for those who wish to pursue further study in systems programming. The book is not intended for those who desire to familiarize themselves with the facilities and ramifications of any particular system.

The first chapter is on assembly systems. The study of assembly systems is not aimed at a detailed examination of a present-day assembler. Instead, an abstract assembler is proposed on which the general concepts of the assembly process can be defined and from which the factors, which have complicated the design of assemblers, can be delineated. These factors are directly related to the capability of the assembly language in handling pseudoinstructions of deferred type. Since the flexibility and power of an assembly language are reflected by its capability to handle pseudoinstructions of deferred type, the complexity in assembler design is therefore proportional to the flexibility and power of its assembly language. By studying the abstract assembler, not only can we learn the steps of the assembly process for one of

the most powerful and flexible assembly languages, but we can also determine those steps which are directly involved in the resolution of pseudoinstructions of deferred type. The understanding of these steps enables the system designer to determine the degree of flexibility and power of an assembly language which he wishes to propose and to anticipate the amount of design and implementation complexity which has to be incorporated into the assembler. In addition, the function of the linking loader is discussed. Its role in the assembly system and in the input and output system is emphasized.

The second chapter is on input and output control systems (IOCS). In this chapter, we first discuss the prevailing notions in input and output operations such as cycle stealing, memory interference, operational simultaneity, and buffer switching. An idealized model is introduced for the purpose of demonstrating the phenomenon known as concurrent processing with multiple channels. In the model, the concept of system throughput and a measure of concurrence relative to the channel utilization are developed. Although the model is somewhat oversimplified, we hope the essence of concurrent processing and system throughput has been conveyed. The oversimplification is perhaps unavoidable in view of the complexity and amount of detail involved in the real-world situations. We then present two different approaches in designing input and output control systems—a structural approach which concentrates on the design of input and output control systems in terms of layers of subsystems and programs, and a finite-state automata approach which organizes input and output control systems as sequential machines. Because input and output control systems deal mostly with computer hardware elements and operations which can be easily characterized as states and transitions of states, the finite-state automata approach in designing input and output control systems, although seldom used, is promising. By identifying an input and output control system as an abstract sequential machine with states and transitions of states, the functionality and logic of the input and output control systems can be clearly synthesized, readily understood, and easily programmed.

The third chapter of the book is on batch processing operating systems. Here generalized batch processing operating systems are proposed. The discussion of off-line batch processing concentrates on the characteristics of the system as a hierarchy of processors and subsystems. Efforts to improve the system's ability to accommodate on-line input and output operations and to couple with other systems are also discussed. The characterization of the on-line batch processing operating system in terms of states and transitions of states is straightforward; and the development of the spooling system for on-line job input and output is important.

There are two chapters on multiprogramming systems. In Chapter 4, we identify the factors which have prompted the introduction of multiprogramming systems and classify various multiprogramming systems in terms of the levels of multiprogramming. These levels reflect, to a large extent, the degrees of sophistication of the systems discussed. It is well to note that although the concept of multiprogramming is easy to motivate, approaches to the synthesis and analysis of multiprogramming systems vary widely. These approaches are influenced by the physical resources intended for the systems

and by the design experience derived from earlier systems such as the batch processing systems. We then outline mechanisms such as semaphores by which these multiprogramming systems can be supported and realized, and point out the problems, such as system deadlock, which are inherent in these systems. Chapter 5 is centered on the multiprogramming systems using virtual space. In our study of virtual space, we attempt to answer the questions: (1) Why do we need virtual space? (2) What are the types of virtual spaces and their capabilities? In answering question (1), we point out that virtual space is designed primarily to resolve problems concerning dynamic storage allocation, repetitive program relocation, program overlay, and run-time growth of programs and data. Because different developers of virtual space have aimed to resolve one or more of the above problems, there are various types of virtual spaces. To this end we try to classify the types of virtual spaces and illustrate them with examples. Thus, we can answer the second question. Finally, we try to describe a generalized multiprogramming operating system which summarizes some of the common characteristics prevailing among the virtual memory operating systems. The role of interrupt analyzer and task initiator in the generalized multiprogramming system is outlined. Since physical input and output operations in a multiprogramming system are always performed by the system (instead of by user programs), the expanded function of the input and output control subsystem in the generalized multiprogramming system is discussed. We also contrast the traditional logical input and output operations with the virtual access method which is unique in a virtual space environment. Both dynamic loading and multitasking are also unique to multiprogramming systems. We try to relate these unique system features with material developed in the previous chapters. Solutions to software problems associated with virtual space are also discussed in the context of the generalized multiprogramming operating systems.

The final chapter is devoted to data base management systems. These systems are important because there are indications that future operating systems will be primarily oriented toward data base management. The increasing emphasis in on-line use of computer systems and the improving cost effectiveness of on-line bulk storage have accelerated the development of centralized data bases and multiaccess terminals. Thus the traditional view of an operating system as a computer physical resource manager is too limited. The sheer size of the on-line data bases in a future computer system requires the operating system to become, in addition, a logical resource manager. Logical resource largely consists of on-line data and programs. The management of logical resource is concerned with the organization, security, access, and storage of data and programs in the data base. Because the area of the data base management is still evolving, our discussion will touch upon some problems which need to be resolved. To this end, we have proposed an abstract model of a generalized data base management system by which the concept of storage cell is defined and from which the frequently used structures such as indexed sequential, multilist, and inverted files, and the more recent cellular multilist structure can be derived.

An access algorithm is provided for data structures derivable from the model. The algorithm, due to its unusual characteristics, tends to minimize the movement

of the access mechanism for movable-head storage devices, to reduce the number of accesses to secondary storage, and to eliminate imprecise data retrieval.

An algorithm for updating the data structures derivable from the model is provided. Dynamic updating of data structures is needed when an addition, deletion, or replacement of keywords of records occurs.

Mechanisms for access control and privacy protection in data base management systems are also discussed in the model.

Knowledge of computer organization and programming principles is a necessary prerequisite for the book. For a minimal reading of the background material, the reader may refer either to the first six chapters in *Computer Organization and Programming* by C. William Gear (McGraw-Hill) or to Chapters 1, 2, 3, and 8 in *Digital Computer System Principles* by Herbert Hellerman (McGraw-Hill). The latter also contains a very concise, but sufficient, description of the IBM System/360 in Chapter 9. Additional reading on introductory material to systems programming may be found in *Systems Programming* by John J. Donovan (McGraw-Hill). At The Ohio State University, Professor Donovan's book is used in a course which is a prerequisite for the course using the present text.

Preface

The material in the book was originally developed at the Moore School of Electrical Engineering, University of Pennsylvania, for a first course in systems programming in the Fall of 1969. Although the purpose was to present systems programming to the first-year graduate students with a computer science major, the book is presently used at The Ohio State University for a systems programming course in which about half of the students are undergraduate juniors and seniors and the other half beginning graduate students. Knowledge of programming languages (e.g., FORTRAN and assembly language) and programming techniques is required. In addition, basic understanding of conventional, general-purpose digital computer organization and system architecture is assumed.

Some parts of Chapters 2, 3, 4, and 5 were used as an introduction to a summer program on *Multiprogramming System Design Principles* which was held in the Summer of 1970, at the University of Pennsylvania and was supported, in part, by the National Science Foundation. In this program, five specific multiprogramming systems were discussed by their principal designers. We had Professor F. J. Corbató lecturing on Honeywell-GE 645 Multics, Mr. William L. Konigsford on IBM 360/67 TSS, Mr. Norman Weiser on Univac-RCA Spectra 70/46 TSOS, Mr. Bernard I. Witt on IBM 360/MVT and Dr. David N. Freeman on IBM 360 TOS/DOS. We also had recitation leaders who led us into the details of the design of these systems. Professor Michael D. Schroeder was responsible for Multics, Mr. Lee Varian for TSS, Mr. George Bean for MVT, Mr. John Gibson for TSOS and Miss Gwendolyn Gartland for TOS/DOS. Their lectures and recitations have greatly helped the author's understanding of these multiprogramming systems in particular and improved the parts of the book on multiprogramming and virtual memory systems in general. To them the author would like to register his indebtedness.

Considerable material in the chapters on on-line programming and data base systems is derived from the work of the author's colleagues and students and was supported by the Information Systems Program in the Office of Naval Research. Without these persons' diligent work and the support of the Program, these chapters

could not have been written. I would like to thank Messrs. Harry A. Freedman and Frank Manola; Drs. Marvin Gelblat, Richard P. Morton, Richard L. Wexelblat and Michael S. Wolfberg; and Professor Noah S. Prywes for their contributions. Mr. Marvin Denicoff, Director of the Information Systems Program, Office of Naval Research, deserves my sincere thanks. His continuing support not only made these contributions possible, but also advanced the state-of-art of systems programming in the areas of on-line programming and data base systems.

The book is organized into six chapters. Each chapter deals with one or more important system concepts. Thus we have assembly languages and assembly systems, input and output operations and input-output control systems, off-line and on-line batch processing systems, multiprogramming, virtual memory systems, and data base systems. Notable omissions are compiler systems and real-time systems. The neglect of the former is deliberate because there is a large number of books on this topic. The omission of the latter is due largely to the author's ignorance of the subject matter. The material in the book is, to a large extent, classical, relying heavily on materials derived from published papers, reports, books, manuals, notes, and private correspondences. To give credit to the original sources of information and to allow further pursuit by the reader, there is a postscript at the end of each chapter which annotates briefly the references made in that chapter.

I would also like to thank Messrs. Joel D. Aron and Charles L. Gold of IBM for their valuable comments and advice which have enhanced the book considerably; my colleague, Professor Douglas S. Kerr, for his careful reading of the manuscript; Professor Andrew Noetzel of the University of Texas for contributing the first draft of Section 3 in Chapter 5; Professor John G. Brainerd who, as the Director of the Moore School, saw the need for a new course in systems programming and was instrumental in getting my course in the curriculum in 1969; and Misses Sally Futrell, Cindy Karr, and Sandy Rich for typing several versions of the manuscript.

Perhaps due to the exposures of the material in the aforementioned summer program and the invited presentation in the 1970 Fall Joint Computer Conference, the earlier drafts of the book have been used either as a text or as a supplement at various institutes by my colleagues, in particular, Professors Stephen W. Ching of Villanova, Richard Eckhous of Massachusetts, Clinton R. Foulk of Ohio State, Lance Hoffman of the University of California at Berkeley, T. Kimura of Delaware, Noah S. Prywes of Pennsylvania, and Richard B. Simmons of Texas A & M. Their enthusiasm about the book is appreciated.

Last but not least, I would like to show my appreciation to my students whose keen interest in systems programming has made the preparation of the book worthwhile, and to the computer industry whose large undertakings in systems programming have provided the field with many fruitful results and challenging problems.

Columbus, Ohio D.K.H.
June 1975

Acknowledgments

The figures and equations in Section 2.4 (pages 65–72) are derived from "Throughput Analysis of Some Idealized Input, Output and Compute Overlap Configurations" by H. Hellerman and H. J. Smith, Jr., in *Computing Surveys* **2**, 2 (June 1970). Copyright 1970 by the Association for Computing Machinery. Reprinted with permission of the authors and publisher.

The example and figures in Section 2.5.1 (pages 72–77) are reprinted with some revision from *Digital Computer Systems Principles* by H. Hellerman. Copyright 1967 by McGraw-Hill Book Company. Used with permission of the publisher.

The figures in Sections 3.2.4.1 and 3.2.4.2 (page 119) are derived from R. F. Rosin, "Supervisory and Monitor Systems," *Computing Surveys* **1**, 1 (March 1969). Copyright 1969 by the Association for Computing Machinery. Reprinted with permission of the author and publisher.

The stories of the sleeping doctor in Section 4.4.2 (page 165) and the smart banker in Sections 4.5.1.1 and 4.5.1.2 (page 167) are adapted from "Co-operating Sequential Processes" written by E. W. Dijkstra and edited by F. Genuys for *Programming Languages*, London: Academic Press, 1968. Reprinted with permission of the author and publisher.

The list of deadlock conditions and the detection algorithm in Sections 4.5.2 and 4.5.3 (pages 169 and 170) are from "System Deadlocks" by E. G. Coffman, Jr., M. J. Elphick, and A. Shoshani in *Computing Surveys* **3**, 2 (June 1971). Copyright 1971 by the Association for Computing Machinery. Reprinted with permission of the authors and publisher.

Contents

Chapter 3 The Batch Processing Operating Systems

Chapter 4 Multiprogramming