国外计算机科学教材系列

操作系统

——并发与分布式软件设计

Operating Systems

Concurrent and Distributed Software Design

英文版

[英] Jean Bacon
Tim Harris 等著

電子工業出版社
Publishing House of Electronics Industry
http://www.phei.com.cn

国外计算机科学教材系列

操作系统

—— 并发与分布式软件设计

（英文版）

Operating Systems

Concurrent and Distributed Software Design

[英] Jean Bacon
Tim Harris 等著

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内容简介

本书是一本用新的观念讲述操作系统原理和设计的高级教程。全书围绕着并发和分布式概念组织章节，避免仅仅将注意力置于传统的集中式系统，反映了分布式系统在当代已经被广泛应用的现实，从而更能为读者所接受。

全书共有30章，分为四个部分，第一部分和第二部分主要阐述了操作系统核心概念和基本功能，并介绍了操作系统理论和实践的最新进展。第三部分和第四部分将操作系统和数据库，以及分布式系统的设计结合起来，对UNIX，Linux，Windows 2000/NT和JVM等典型现代操作系统进行了具体分析，并讨论了Web服务、中间件和事务处理等应用。对于操作系统安全这样一个重要课题，书中还专门安排了一章进行论述。本书的另一个特点是，所有的代码示例均结合Java语言进行，并采用Java描述面向对象的概念、并发算法和分布式程序设计等。

本书适合作为研究生的操作系统课程教材，也适合程序员、软件工程师以及系统设计者作为学习和了解系统软件和操作系统的参考书。

版权贸易合同登记号：图字：01-2003-2040

图书在版编目（CIP）数据

操作系统：并发与分布式软件设计 = Operating Systems: Concurrent and Distributed Software Design/（英）培根（Bacon, J.）著. -北京：电子工业出版社，2003.6

（国外计算机科学教材系列）

ISBN 7-5053-8788-X

Ⅰ.操… Ⅱ.培… Ⅲ.分布式操作系统-教材-英文 Ⅳ.TP316.4

中国版本图书馆CIP数据核字（2003）第044231号

责任编辑：李秦华

印 刷 者：北京天竺颖华印刷厂

出版发行：电子工业出版社 http://www.phei.com.cn

北京市海淀区万寿路173信箱 邮编：100036

经 销：各地新华书店

开 本：787 × 980 1/16 印张：57 字数：1277千字

版 次：2003年6月第1版 2003年6月第1次印刷

定 价：79.00元

出 版 说 明

21世纪初的5至10年是我国国民经济和社会发展的重要时期，也是信息产业快速发展的关键时期。在我国加入WTO后的今天，培养一支适应国际化竞争的一流IT人才队伍是我国高等教育的重要任务之一。信息科学和技术方面人才的优劣与多寡，是我国面对国际竞争时成败的关键因素。

当前，正值我国高等教育特别是信息科学领域的教育调整、变革的重大时期，为使我国教育体制与国际化接轨，有条件的高等院校正在为某些信息学科和技术课程使用国外优秀教材和优秀原版教材，以使我国在计算机教学上尽快赶上国际先进水平。

电子工业出版社秉承多年来引进国外优秀图书的经验，翻译出版了“国外计算机科学教材系列”丛书，这套教材覆盖学科范围广、领域宽、层次多，既有本科专业课程教材，也有研究生课程教材，以适应不同院系、不同专业、不同层次的师生对教材的需求，广大师生可自由选择和自由组合使用。这些教材涉及的学科方向包括网络与通信、操作系统、计算机组织与结构、算法与数据结构、数据库与信息处理、编程语言、图形图像与多媒体、软件工程等。同时，我们也适当引进了一些优秀英文原版教材，本着翻译版本和英文原版并重的原则，对重点图书既提供英文原版又提供相应的翻译版本。

在图书选题上，我们大都选择国外著名出版公司出版的高校教材，如Pearson Education培生教育出版集团、麦格劳－希尔教育出版集团、麻省理工学院出版社、剑桥大学出版社等。撰写教材的许多作者都是蜚声世界的教授、学者，如道格拉斯·科默（Douglas E. Comer）、威廉·斯托林斯（William Stallings）、哈维·戴特尔（Harvey M. Deitel）、尤利斯·布莱克（Uyless Black）等。

为确保教材的选题质量和翻译质量，我们约请了清华大学、北京大学、北京航空航天大学、复旦大学、上海交通大学、南京大学、浙江大学、哈尔滨工业大学、华中科技大学、西安交通大学、国防科学技术大学、解放军理工大学等著名高校的教授和骨干教师参与了本系列教材的选题、翻译和审校工作。他们中既有讲授同类教材的骨干教师、博士，也有积累了几十年教学经验的老教授和博士生导师。

在该系列教材的选题、翻译和编辑加工过程中，为提高教材质量，我们做了大量细致的工作，包括对所选教材进行全面论证；选择编辑时力求达到专业对口；对排版、印制质量进行严格把关。对于英文教材中出现的错误，我们通过与作者联络和网上下载勘误表等方式，逐一进行了修订。

此外，我们还将与国外著名出版公司合作，提供一些教材的教学支持资料，希望能为授课老师提供帮助。今后，我们将继续加强与各高校教师的密切联系，为广大师生引进更多的国外优秀教材和参考书，为我国计算机科学教学体系与国际教学体系的接轨做出努力。

电子工业出版社

教材出版委员会

Preface

This book has its origins in Jean Bacon's *Concurrent Systems* editions 1 and 2. Edition 3 was created recently, specifically for the Open University of the UK who have used the book for their course 'Software Systems and their Development' (M301) since 1999. That course does not require a detailed treatment of operating systems' design with case studies and those parts of *Concurrent Systems* were removed in its third edition.

This book extends the treatment of operating systems, making it even more appropriate for the standard Operating Systems curriculum. Editions 1 and 2 of *Concurrent Systems* established the fundamental principles which are necessary for understanding software systems. Features of *Operating Systems: Concurrent and Distributed Software Design* are:

- Java is used throughout to illustrate object-orientation concepts, concurrent algorithms and distributed programming.
- The coverage of operating systems' design and case studies is updated and extended.
- There is a chapter on security which complements the coverage of distributed systems.
- The case studies in Part IV have separated the treatment of classical UNIX from that of current versions and include a new chapter on extensible operating systems. 'The World Wide Web' and a rewrite of 'Middleware' represent recent developments in distributed systems.

The philosophy and approach, focusing on system design issues, remain unchanged and have been applied to the systems that have evolved since 1993.

The aim of this book is to equip students with an integrated view of modern software systems. Modularity, concurrency and distribution are the unifying themes, both within the design of operating systems and in the systems supported by operating systems. The book takes a systems approach rather than a programming language approach, since concurrent

and distributed programming are firmly rooted in system design. The language is an implementation tool for the system designer and programming languages are covered throughout the book from this perspective.

The structure of the book is:

- **Introduction**, in which types of real-world system are described and requirements for building computerized systems established.
- **Part I**, in which the relationship between technology and system design is explored. The basic operating system functions are described in detail and the abstraction, and implementation, of a system as a community of concurrent processes is established. The fundamental properties of distributed systems are set down and the provision of security in such systems is discussed.
- **Part II**, in which the focus is concurrency control in operating systems and application-level systems, and inter-process communication (IPC) is explored in the context of system design. To partition the material, the theme of this part concentrates on how to achieve atomicity of a method invocation on an object in main memory. We conclude by considering the cases when the object invoked is persistent or remote; distributed IPC is covered in detail.
- **Part III**, in which the theme is concurrent composite operations or transactions. The traditional application area for these ideas is database systems but we illustrate the general applicability of these systems concepts.
- **Part IV**, in which some case studies are considered from the perspective developed throughout the book.

Computer systems curriculum

Because distributed systems have come into widespread use comparatively recently, most curricula include them at final-year undergraduate or post-graduate level. Distributed systems are now commonplace and a student is more likely to be using one than a centralized time-sharing system. It is somewhat artificial to cover the functions of a shared, centralized operating system in great detail in a first course, particularly when the rate of development of technology makes it essential constantly to re-evaluate traditional approaches and algorithms.

In general, there is a tendency for closely related specialisms to diverge, even at undergraduate level. An overview of system components and their relationships is desirable from an early stage:

- Operating systems include communications handling.
- Language runtime systems work closely with (and are constrained by) operating systems.
- Real-time systems need specially tailored operating systems.

- Dedicated communications-handling computers need specially tailored operating systems.
- Database management systems run on operating systems and need concurrency and file handling with special guarantees.
- Concurrent programs run on operating systems.
- Operating systems (and their components) are concurrent programs.
- Window systems require concurrent processes.
- Many system components employ databases.
- Distributed systems employ distributed databases.
- Distributed databases need communications.
- Distributed operating systems need transactions.

Operating Systems: Concurrent and Distributed Software Design achieves this integration by setting up a common framework of modular structure (a simple object model is used throughout), concurrency control and distribution.

We have used this approach in the Computer Science curriculum at Cambridge since 1988, when a new three-year undergraduate degree programme started. A concurrent systems course, in the second year of a three-year degree course, is a prerequisite for further study in distributed operating systems, communications and networks, theory of concurrency, and various case studies and projects. Figure P.1 suggests an order of presentation of systems material. Courses in the general area of real-time, embedded control systems would also follow naturally from this course.

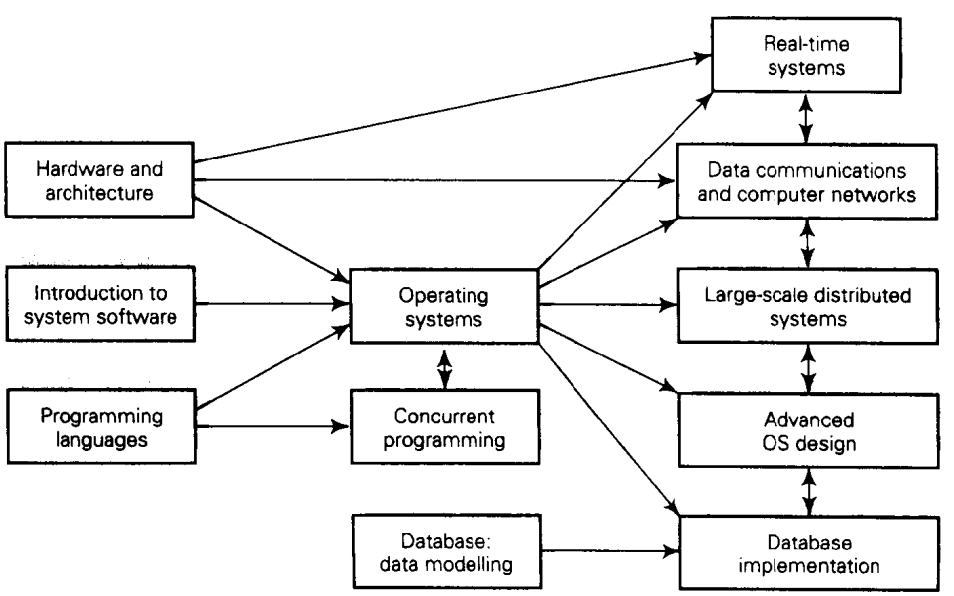

Figure P.1
Operating systems in distributed, concurrent software in the curriculum.

In *Curriculum 91 for Computing*, published by the IEEE Computer Society and the ACM (see Denning *et al.*, 1989; Tucker 1991) the general topic 'Operating Systems' includes distributed operating systems and communications. *Curriculum 91* identifies the three major paradigms of the discipline as: **theory**, which is rooted in mathematics; **abstraction** (modelling), which is rooted in experimental scientific method; and **design**, which is rooted in engineering. Theory deals with the underlying mathematics of each subarea of computing. Abstraction allows us to model large, complex systems in order to comprehend their structure and behaviour and carry out experiments on them. Design deals with the process of implementing a system to meet a specification. The approach taken here embodies abstraction and design and establishes the basis for theory.

In December 2001 IEEE-CS/ACM published, in their *Computing Curricula 2001*, a curriculum for Computer Science. Curricula for Computer Engineering, Software Engineering and Information Systems are to follow. They argue that, since the subject has grown extensively and rapidly in the past decade, a minimal core curriculum should be defined, allowing a variety of extensions. The core topics for the 18 minimum core hours in Operating Systems are: overview of operating systems (2), operating systems principles (2), concurrency (6), scheduling and dispatch (3) and memory management (5). The core topics for the 15 minimum core hours of Net-Centric Computing are introduction (2), communication and networking (7), network security (3), and the web as an example of client–server computing (3). The ten-hour minimal core in Information Management has transaction processing as an early extension. 'Objects First' is one of three proposed models (with 'Imperative First' and 'Functional First') for introducing programming. The systems view presented here includes, integrates and extends this core coverage in systems topics.

Audience

It is assumed that the reader will come to this material with some knowledge and experience of systems and languages. First-year undergraduate courses on programming and systems software are appropriate prerequisites. It is suitable as a text to be read in parallel with specialized courses in communications and databases. It is a suitable starting point for graduate students in the systems area, providing integrating and summarizing study for the current wide variety of undergraduate degrees. Practitioners in systems design including systems programming will find the fundamentals of the subject here. Graduate students who are researching the theory of concurrency will find the practical basis for their subject here.

An outline of the contents

Chapter 1 describes a number of types of system and draws out requirements for supporting them. Software systems can exploit a wide range of hardware topologies and architectures. Although this area is not addressed in great detail their characteristics are noted for reference throughout the book.

Chapters 2 through 8 form **Part I**. System design and implementation require software to be engineered. Software engineering, which involves the specification, design, implementation, maintenance and evolution of software systems, has merited many books in its own right. Here we briefly establish a context of modular software structure, establishing an object model to be used throughout.

Modular system structure is introduced in **Chapter 2** and the modular structure of operating systems is set up. The idea that a minimal kernel or 'microkernel' is an appropriate basis for high-performance specialized services is introduced here. The concepts of process and protocol to achieve the dynamic execution of software are also introduced.

In **Chapter 3** device handling and communications handling are covered. These topics are treated together to highlight the similarities (between communications and other devices) and differences (communications software is larger and more complex than device-handling software). The communications-handling subsystem of an operating system is itself a concurrent (sub)system, in that at a given time it may be handling several streams of input coming in from various sources across the network as well as requests for network communication from local clients.

Chapter 4 gives the detailed concrete basis for the process abstraction that is provided by operating systems. Once the process abstraction is created as one operating system function we can show how processes are used to achieve the dynamic execution of the rest of the system. Operating system processes may be used within operating system modules, while application-level processes may be located within application modules. There are several design options which are discussed throughout the book. Later sections are concerned with language systems and a particular concern is the support for concurrency. The relation between operating system and language system processes is discussed in detail.

Chapter 5 covers memory management. The address space of a process is an important concept, as also are mechanisms for sharing part of it.

Chapter 6 gives the basic concepts of filing systems. File system implementations involve data structures both in main memory and in persistent memory on disk. Both the memory management and file management subsystems of operating systems are concurrent systems

in that they may have in progress both requests from clients and demands for service from the hardware.

Chapter 7 introduces distributed software systems. We focus on their fundamental properties then cover time and naming in some detail. Subsequent chapters can then consider distribution of the various functions being studied.

Chapter 8 is concerned with security in centralized and distributed systems.

Part I is concerned with technology and its impact on system design. Knowledge of the material presented here is necessary for a thorough understanding of software systems. Care must be taken, when working at the language or theoretical modelling levels, that the assumptions made can be justified for the operating system and hardware that will be used to implement a system.

Part II focuses on a major system design requirement: **concurrency control**. It explains the mechanisms for ensuring that a given concurrent process can execute without interference from any other, bearing in mind that processes may be cooperating with other processes (and need to synchronize with them) or competing with other processes to acquire some resource.

Chapters 9 to 16 comprise Part II. In Part II we temporarily ignore the issues of composite operations and the need to access multiple resources to carry out some task and confine the discussion to a single method invocation on an object in main memory. The notion of a single abstract operation is informal and closely related to the modular structuring of systems. A process can, in general, read or write a single word of memory without fear of interference from any other process. Such a read or write is indivisible. In practice, a programming language variable or a useful data abstraction, such as an array, list or record, cannot be read or written atomically. It is the access to such shared abstractions by concurrent processes that is the concern of Part II. Chapters 9 to 14 are mostly concerned with 'load and go' systems that run in a single or distributed main memory. Chapters 15 and 16 start to consider the effect of failures in system components and process interactions which involve persistent memory.

Chapter 9 discusses the major division between processes which share memory, running in a common address space, and those which do not. Examples are given, showing the need for both types of structure.

Chapter 10 is concerned with the lowest level of support for process interactions. The architecture of the computer and the system is relevant here. It is important to know whether any kind of composite read–modify–write instruction is available and whether the system architecture contains shared-memory multiprocessors or only uniprocessors. Concurrency

control without hardware support is discussed and semaphores are introduced.

Chapter 11 builds on the lowest level to create algorithms to solve classic systems problems. A discussion of the difficulty of writing correct semaphore programs leads on to high-level language support for concurrency in the next chapter.

Chapter 12 looks at language primitives that have been introduced into high-level concurrent programming languages where the underlying assumption is that processes execute in a shared address space. The support for concurrent programming in Java is described here.

Chapter 13 compares inter-process communication (IPC) mechanisms within systems where shared memory is available and where it is not. In both cases processes need to access common information and synchronize their activities.

Chapter 14 covers IPC for processes which inhabit separate address spaces. Pipes and message passing are discussed. The material here is highly relevant to distributed IPC, but the integration of IPC and communications services in left for Chapter 16.

Chapter 15 introduces the possibility that a system might crash at any time and outlines mechanisms that could be used to provide crash resilience. An initial discussion of operations which involve persistent data is also given.

Chapter 16 focuses on IPC in distributed systems, taking account of their fundamental characteristics, introduced in Chapter 7. We see how an operation at one node of a distributed system can be invoked from another node using a remote procedure call protocol. Node crashes and restarts and network failures are considered. Although distributed IPC is the main emphasis of the chapter, it includes a general discussion of naming, location and the binding of names to locations in distributed systems. Socket programming in Java and Java's remote method invocation (RMI) are given as practical examples.

Chapters 17 through 23 comprise **Part III** where the discussion is broadened to composite operations (**transactions**) and the concurrent execution of their component operations. The objects concerned may be in main memory, in persistent memory and/or distributed.

Chapter 17 introduces the problems and defines the context for this study. Composite operations may span distributed systems and involve persistent memory.

Chapter 18 discusses the desirability of dynamic resource allocation and the consequent possibility of system deadlock. An introduction to resource allocation and management is given, including algorithms for deadlock detection and avoidance.

Chapter 19 discusses composite operation execution in the presence of concurrency and crashes and builds up a definition of the fundamental properties of transactions. A model based on abstract data objects is used.

Chapter 20 discusses concurrency control for transactions. Two-phase locking, time-stamp ordering and optimistic concurrency control are described and compared.

Chapter 21 is mainly concerned with crash recovery, although the ability to abort transactions for concurrency control purposes is a related problem. A specific implementation is given.

Chapter 22 extends the object model for transactions in distributed systems and reconsiders the methods of implementing concurrency control. The problem of distributed atomic commitment is discussed and a two-phase commit protocol is given as an example. A validation protocol for optimistic concurrency control is also given.

Chapter 23 covers algorithms which may be used by distributed computations.

Chapters 24 through 30 comprise **Part IV**, in which case studies are presented. Greater depth is possible here than in the examples used earlier. An aim is to show that the approach developed throughout the book helps the reader to comprehend large, complex systems.

Chapter 24 describes the basic UNIX seventh edition design, 'classical UNIX'. The design is evaluated and the process management and interprocess communication facilities, in particular, are criticized.

Chapter 25 shows how these criticisms have been addressed in LINUX, Solaris and contemporary UNIX.

Chapter 26, on extensible operating systems, explores more radical operating system structures, including microkernels.

Chapter 27 is a case study on Microsoft's Windows 2000 which has an object-oriented design. PC hardware is now able to provide protection suitable for supporting multiple applications and users and sufficient power for large-scale applications.

Chapter 28 covers web programming, a subject which was invented after edition 1 of *Concurrent Systems* was published and which has grown extensively since edition 2. This style of programming is set to dominate much of distributed systems design and development.

Chapter 29 covers the broad range of middlewares, contrasting those based on synchronous object invocation and those based on asynchronous message passing. The former include Java and OMG's OMA and CORBA and the latter IBM's MQseries and the TIBCO TIB/Rendezvous. Microsoft's DCOM and .NET are also outlined.

Chapter 30 first discusses how transaction processing monitors are implemented in terms of processes, IPC and communications. Some examples of transaction processing systems in the area of electronic funds transfer are then given, for example, an international automatic teller machine (ATM) network.

The **appendix** presents a historical and functional evolution of software systems in a technological and commercial context. How to achieve concurrency control without any hardware support is also included, together with exercises.

Order of presentation

Figure P.2 indicates dependencies in the material and shows how the chapters might be selected for courses on operating systems and distributed systems and for supplementary material on concurrent and systems programming or databases.

The material in Part II could be taken in a different order. Although there is a flow of argument through the chapters as written, there is no inherent reason why shared-memory IPC has to come before that with no shared memory, although distribution follows naturally from the latter.

A concurrent programming course could supplement Part I and Part II with full details of a language to be used for project work; Java would be the natural choice. Chapters 17 and 18 from Part III should also be included.

A course on concurrency control in database systems would use Part III, but earlier chapters which cover operating system support for databases provide an excellent background.

Further study

The book naturally leads on to advanced systems courses: specialized operating systems, real-time embedded control systems, large-scale distributed systems, distributed databases. The conceptual framework of concurrency control, distribution, naming, location, placement, protection, authentication and encryption are set up, ready for exploitation in systems design.

Objective

The main emphasis of the book is system design (with the operating system as a major component), how to comprehend existing systems and how to design new systems. One can't write certain kinds of system in certain languages above certain operating systems. This book aims to show the reader why. Computers are marketed optimistically. Names

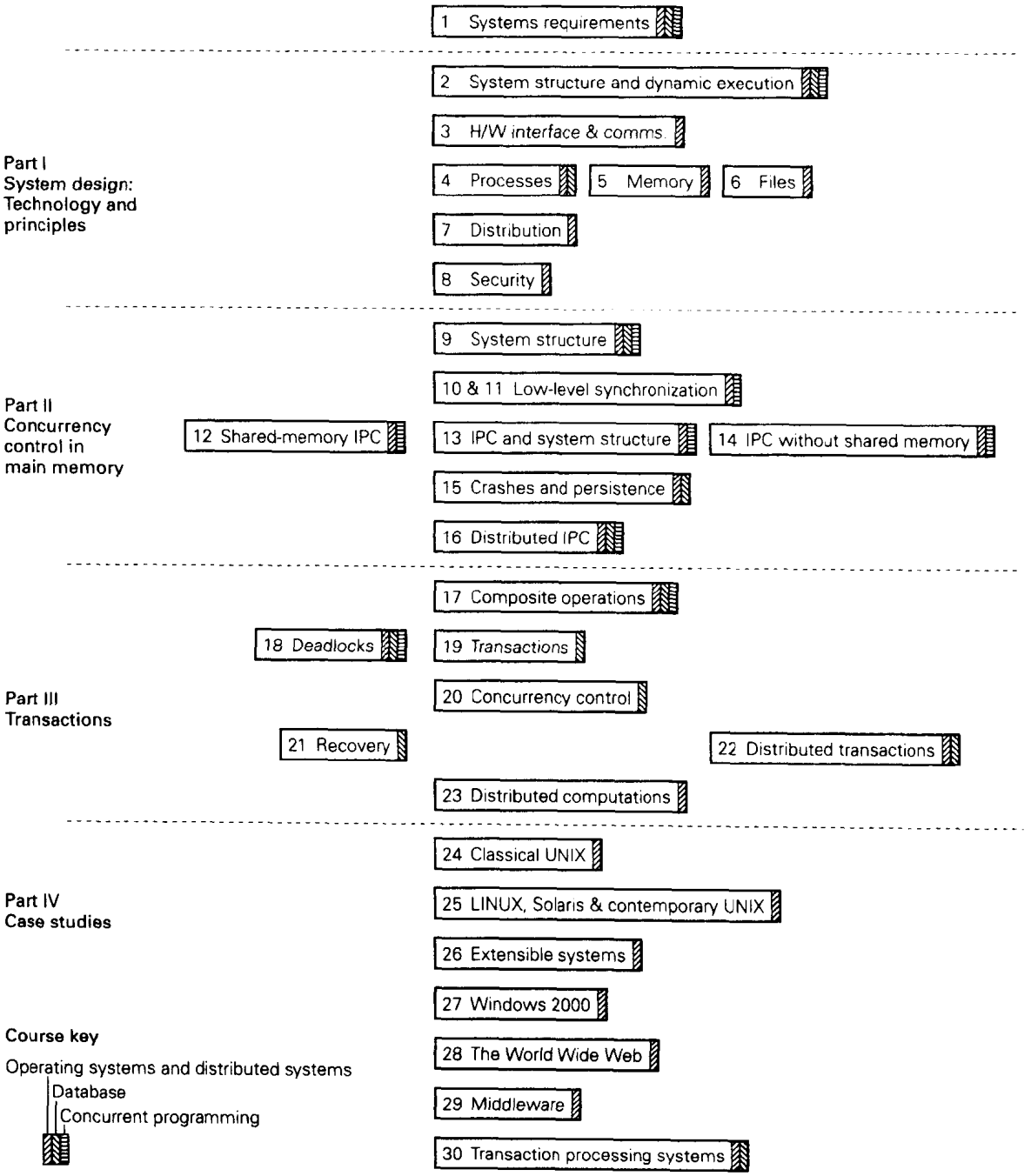

Figure P.2
Presentation of the material.

such as 'real-time operating system' are used with little concern for their meaning. In order to survive an encounter with a salesperson one must know exactly what one wants and must know the pitfalls to look out for in the systems one is offered. I hope the book will help systems

designers to select the right hardware and software to satisfy their requirements.

Instructor's guide

A web-browsable instructor's guide was developed for the second edition of *Concurrent Systems*, containing the following:

- Curriculum design. An outline of parts of the IEEE-CS/ACM *Computing Curricula 1991* is given. Uses of *Concurrent Systems* in the curriculum are discussed.
- Points to emphasize and teaching hints. For each chapter, key points, potential difficulties and suggested approaches to teaching the material are given.
- Solutions to exercises and some additional exercises. The solutions include examples of how the various designs that are discussed have been used in practice.

Extensions for this book are underway. The guide is available from www.booksites.net/bacon
Contact your Pearson Education or Addison-Wesley sales representative for a password.

Acknowledgements

We acknowledge the contributions of the Open University team, Robin Laney, Pete Thomas and Janet van der Linden, who provided input to *Concurrent Systems* edition 3, based on using *Concurrent Systems* for the OU's M301 course 'Software Systems and their Development'. Their input on concurrent and distributed programming in Java and study questions have been retained. We are also grateful to Cambridge students who have taken the courses based on *Concurrent Systems* for feedback, and to graduate students and colleagues in the systems area for their comments. Thanks are due to those who have used the book in their teaching and were helpful on how it should evolve. We thank Ken Moody for input on all aspects of the book but especially on the database material. Thanks to Austin Donnelly for checking the UNIX chapters. Pearson Education has provided support, motivation and encouragement throughout.

Jean Bacon and Tim Harris
July 2002

Jean.Bacon@cl.cam.ac.uk
Tim.Harris@cl.cam.ac.uk
http://www.cl.cam.ac.uk/~jmb
http://www.cl.cam.ac.uk/~tlh20

Trademark notice

The following designations are trademarks or registered trademarks of the organizations whose names follow in brackets:

ACMS, Alpha, DECintact, DECNET, DECSystem 10, DECthreads, DEQNA, Firefly, GNS, ULTRIX, VAX and VMS (Digital Equipment Corporation); ACTIVEX.COM (used by CNET under licence from owner – ACTIVEX.COM is an independent online service); Amoeba (Vrije Universiteit); ANSA (ANSA Internet Inc.); ARM and CHORUS (Acorn Computers); CORBA (Object Management Group); Courier and X25 (Xerox Corporation); HP-UX (Hewlett-Packard Company); Java, JavaScript, NFS, SunOS and XDR (Sun Microsystems, Inc.); MC68000 (Motorola); MIPS (Silicon Graphics, Inc.); Netscape Navigator (Netscape Communications Corporation); Netware (Novell); occam (INMOS Group of companies); Appletalk, Macintosh, MacOS X and OpenDoc (Apple Computers); MQSeries, OS/2, SNA and System 370 (International Business Machines Corporation); Pathway (Tandem); Itanium, Pentium, 8086, 80x86 and 860 (Intel); Rendezvous (TIBCO); REXX (Oracle Corporation); Seagate Elite 23 and BarracudaATA IV (Seagate Technology Products); Tina-C (National Semiconductor Corporation); UNIX (X/Open Company Ltd); ActiveX, COM, DCOM, MS-DOS, .NET, Visual Basic, Windows 95, Windows 2000, Windows NT and Windows XP (Microsoft Corporation).