Undergraduate Topics in Computer Science

Undergraduate Topics in Computer Science (UTiCS) delivers high-quality instructional content for undergraduates studying in all areas of computing and information science. From core foundational and theoretical material to final-year topics and applications, UTiCS books take a fresh, concise, and modern approach and are ideal for self-study or for a one- or two-semester course. The texts are all authored by established experts in their fields, reviewed by an international advisory board, and contain numerous examples and problems. Many include fully worked solutions.

Also in this series

Iain Craig
Object-Oriented Programming Languages: Interpretation
978-1-84628-773-2

Max Bramer
Principles of Data Mining
978-1-84628-765-7

Hanne Riis Nielson and Flemming Nielson
Semantics with Applications: An Appetizer
978-1-84628-691-9

Michael Kifer and Scott A. Smolka
Introduction to Operating System Design and Implementation: The OSP 2 Approach
978-1-84628-842-5

Phillip J. Brooke and Richard F. Paige

Practical Distributed Processing

Springer

Phillip J. Brooke, MA, DPhil, MBCS CITP, CMath MIMA
School of Computing,
University of Teesside, UK

Richard F. Paige, BSc, MSc, PhD
Department of Computer Science
University of York, UK

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Control Number: 2007934035

Undergraduate Topics in Computer Science ISSN 1863-7310
ISBN 978-1-84628-840-1 e-ISBN 978-1-84628-841-8

Printed on acid-free paper

9 8 7 6 5 4 3 2 1

Springer Science+Business Media
springer.com

Preface

Overview and goals

This book has grown out of the authors' lecture notes and studies in the field of distributed processing. In our teaching of distributed processing, we have found it necessary to deal with the traditional views of concurrency (e.g., matters such as race conditions, semaphores and mutual exclusion), and also the practical realisation of networked systems and to consider the underlying operating systems.

There are many books that focus on the theory of concurrency, operating systems or programming, but few that take a coherent view of distributed processing as a practical subject. This book thus aims at presenting the reader with a clear overview of the *process* that is followed when building distributed systems, and describes the tools, techniques and theory that can be applied within this process. The book aims to consider the important points of the engineering process, including the *models* that can be used to assess and analyse parts of distributed systems, the implementation techniques —such as sockets— for these models, and the protocols and security concerns that must be considered as part of any realistic distributed system.

Organisation and features

This book can reasonably be divided into three parts: foundations (of both theory and practical implementation concerns), engineering issues (including security and language concerns), and examples and case studies, serving to

integrate material from the previous two parts.

The first part of the book starts with an overview of distributed processing in Chapter 1. The critical concepts of distributed processing and concurrency —like semaphores, deadlock and naming— are discussed in Chapter 2. In Chapter 3 we present several rigorous models of concurrency that help us understand the underlying theory and practice behind the critical concepts of Chapter 2. We also discuss how these models are used in building distributed systems. Chapter 4 discusses operating systems, and their role in building distributed systems, followed by Chapter 5 which deals with interprocess communication. In Chapter 6 we study protocols, which are an important part of building realistic distributed systems. Examples of protocols are presented, as well as the challenges associated with designing them.

The second part of the book moves to engineering issues. Chapter 7 considers general concerns of security for distributed systems. We take an *engineering* perspective of security; it is important to be aware that security is not simply a matter of encrypting communication: a whole-system view must be taken. Chapter 8 presents a selection of languages and focuses on how these languages can be used to build distributed systems. This serves to illustrate many of the ideas that appear in the previous chapters, both theoretical and practical.

The third part of the book focuses on examples and case studies, aiming to integrate all the previous chapters. In Chapter 9 we go through the engineering life-cycle to carry out a selection of case studies in building distributed systems. While we do not implement these systems, we aim to illustrate the *process* of their construction, focusing on identifying requirements, thinking about protocols, and design. Chapter 10 presents a worked example: building a networked game. We identify technical and business requirements, present a design, discuss protocols and security concerns, and construct an implementation. This chapter serves to bring all of the previous chapters together in a coherent structure. Finally, Chapter 11 concludes the book, and identifies some areas for future work, reading, and experiment.

Each chapter concludes with a coherent summary, and highlights a number of exercises. Some exercises focus on thinking about the issues and problems identified in the chapter; others concentrate on building small parts of distributed systems.

A comprehensive glossary is included. Example code is indicated by '$\boxed{\mathcal{EG}}$' in the margin; similarly, related content is marked '$\boxed{\rightsquigarrow}$'.

Target audience

This book is aimed at students who have a reasonable amount of (sequential) programming experience, but who may not have taken a course on operating systems or concurrency. Some experience with C programming may be helpful, but is certainly not a prerequisite to reading and understanding the material in this book. Readers who do not yet have broad programming experience will need to consult additional texts (see the bibliography) on specific programming languages.

We have taught the material in this text to undergraduate students at the second and third year level. With some additions —particularly, a large-scale project— the book could prove suitable to final-year students taking a software engineering specialisation as well.

Notes to the instructor

This book can be used for a coherent, one-term course or module on distributed systems. It can also be used as part of a longer course on software engineering where distributed systems elements play a substantial role. For use in a stand-alone course, we recommend following the book roughly sequentially, though within chapters topics can be rearranged to a degree without losing the overall flow. Instructors may choose to selectively cover some of the formal models in Chapter 3, depending on student background and interests, although we strongly recommend the inclusion of statecharts, as these are important and are used in the worked example in Chapter 10. Students who have previously taken a course in operating systems may be able to skip the material on semaphores in Chapter 2, though it may be helpful to use as a refresher.

We have found the case studies in Chapter 9 useful as reading and presentation projects, to be presented by small groups of students in class.

The material in Chapter 10 could form the basis of larger projects in distributed systems. Indeed, we have run both research and development projects for undergraduate and taught Masters students based on the material covered in this chapter. Some suggestions for areas to investigate for larger and more long-term projects are given in Section 11.2.2.

Sketch solutions or hints to many of the exercises are included in Appendix A. Sample code, including the networked game considered in Chapter 10, is available from the book's web site, `http://www.scm.tees.ac.uk/p.j.brooke/dpb/`. Additional extensions to the multiplayer game presented

in Chapter 10 are likely to be added over time. Comments and suggestions are welcome, and can be directed to the authors.

Acknowledgments

We thank our colleagues at the University of Teesside and the University of York (and, previously, the University of Plymouth and York University, Canada) for their suggestions, advice and helpful guidance. We also thank the reviewers of this book for their useful and timely suggestions. Special thanks should go to Catherine Brett and Wayne Wheeler of Springer for their assistance during the editing and publication process, as well as to Ian Mackie and Helen Callaghan who started the whole process.

Several software tools were used in the development of this book; without these tools, the writing process would have been even more fraught than it was. We would particularly like to thank the authors and maintainers of LaTeX and Subversion.

Phil would like to thank Christine, Rebecca and Alexander for their support during evenings and days spent coding and writing.

Rich would like to thank Angelika for putting up with him (and Phil) during the writing, rewriting and gaming processes.

Both Phil and Rich would like to thank their proof-readers for catching errors and improving their writing. All remaining errors are, of course, the responsibility of the authors.

Phil Brooke
Rich Paige

July 2007

Contents

Preface v

Contents ix

1. What is Distributed Processing? 1
1.1 Overview 1
1.2 Evolution of computing and networking 2
1.3 Distributed processing 3
1.4 Application areas 4
1.5 Models 4
1.6 Mobile code 5
1.7 Challenges with distributed systems 6
1.8 Summary 7
Exercises 8

2. Concepts of Concurrency 11
2.1 Overview 11
2.2 Architectures in concurrency 12
2.3 Naming and addressing 12
2.3.1 Examples of names and addresses 13
2.3.2 Address mapping mechanisms 14
2.4 Sharing and synchronisation 16
2.4.1 Allocation of resources 17
2.4.2 Example: File synchronisation 17
2.5 Low-level synchronisation 18
2.5.1 Race conditions 18

2.5.2 Mutual exclusion . . . 19
2.5.3 Semaphores . . . 20
2.5.4 Monitors . . . 23
2.5.5 Rendezvous . . . 25
2.6 Timing and real-time systems . . . 26
2.7 Dependability . . . 26
2.7.1 Types of faults and failures . . . 27
2.7.2 Responding to failure . . . 28
2.8 Server types . . . 28
2.9 Clusters, load-balancing and Grids . . . 29
2.10 Summary . . . 30
Exercises . . . 31

3. Models of Concurrency . . . 33
3.1 Overview . . . 33
3.2 State machines and automata . . . 34
3.3 SPIN and Promela . . . 35
3.4 Process algebras . . . 36
3.4.1 Communicating Sequential Processes . . . 37
3.4.2 π-calculus and mobility . . . 39
3.5 Linda . . . 41
3.5.1 JavaSpaces . . . 42
3.6 Deadlock revisited . . . 43
3.7 Summary . . . 45
Exercises . . . 47

4. Concurrency in Operating Systems . . . 49
4.1 Overview . . . 49
4.2 Why use operating systems? . . . 50
4.3 Processes and threads . . . 51
4.3.1 Concept of a process . . . 51
4.3.2 User and supervisor modes in CPUs . . . 52
4.3.3 Multitasking . . . 52
4.3.4 Threads and lightweight processes . . . 54
4.4 Process and thread examples in Linux . . . 54
4.4.1 Fork . . . 54
4.4.2 Pthreads . . . 56
4.5 Tasking in Ada . . . 58
4.6 Summary . . . 60
Exercises . . . 60

5. Interprocess Communication 63
5.1 Overview 63
5.2 Pthreads IPC examples in Linux 64
5.2.1 Mutexes and shared memory 64
5.2.2 Semaphores 66
5.2.3 Condition variables 68
5.3 Mutual exclusion in Ada 71
5.4 BSD sockets 74
5.5 TCP client-server example 75
5.5.1 A simple TCP server 75
5.5.2 String termination and networks 80
5.5.3 A simple TCP client 81
5.5.4 TCP client with name lookup 85
5.6 UDP client-server example 85
5.6.1 UDP server 85
5.6.2 UDP client 87
5.7 Two-way communications 89
5.8 A forking TCP server 91
5.9 Blocking and select 94
5.9.1 Select for two-way communications 95
5.9.2 Select for serving multiple connections 97
5.10 Fault tolerance and IPC timing 97
5.11 Summary 98
Exercises 98

6. Protocols 101
6.1 Overview 101
6.2 Purpose of protocols 102
6.3 Issues in protocols 102
6.3.1 High- and low-level protocols 102
6.3.2 Messages 105
6.3.3 Platform dependence 106
6.3.4 Fault tolerance 107
6.4 Defining protocols 109
6.4.1 Encoding 111
6.4.2 Notation 112
6.5 Example: HTTP 112
6.6 Example: SMTP 113
6.7 Example: Alternating bit protocol 114
6.8 Summary 117
Exercises 117

7. Security 121
7.1 Overview 121
7.2 Definitions, concepts and terminology 122
7.2.1 Risk, threat and vulnerability 122
7.2.2 Objectives of security 122
7.2.3 Design 123
7.3 Security issues in distributed systems 124
7.4 Cryptography 126
7.4.1 Cryptography example: Digital signatures 128
7.4.2 Key management 128
7.4.3 Matching a public key to a user 129
7.5 Case study: Needham-Schroeder 129
7.6 Practical issues 131
7.6.1 C programming 131
7.6.2 Web applications 132
7.6.3 Operating system and network issues 133
7.6.4 SSL 133
7.6.5 Using SSL 134
7.7 Summary 136
Exercises 136

8. Languages and Distributed Processing 139
8.1 Overview 139
8.2 Suitability of languages 140
8.3 Distributed processing in C 141
8.3.1 C generally 141
8.3.2 Debugging C 142
8.4 Distributed processing in Java 143
8.4.1 Overview: the RMI model 144
8.4.2 Example 145
8.4.3 Alternatives 147
8.5 Distributed processing in Ada 148
8.6 Distributed processing in Eiffel and SCOOP 149
8.6.1 SCOOP: A concurrency model for Eiffel 151
8.6.2 Related work and prototypes 152
8.7 Comparison of languages 153
8.7.1 Language paradigm 155
8.7.2 Typing discipline 156
8.7.3 Networking support 156
8.7.4 Concurrency support 157
8.7.5 Interprocess communication support 158

8.8 Summary . . . 158
Exercises . . . 159

9. Building Distributed Systems . . . 161
9.1 Overview . . . 161
9.2 Method . . . 162
9.3 Case study: Email . . . 163
9.3.1 Typical use and requirements . . . 163
9.3.2 Platform and language requirements . . . 164
9.3.3 Architecture . . . 165
9.3.4 Protocols and formats . . . 165
9.3.5 Example: Sending email using PHP . . . 168
9.4 Case study: Secure shell . . . 170
9.4.1 Typical use and requirements . . . 171
9.4.2 Platform requirements . . . 172
9.4.3 Architecture . . . 172
9.4.4 Protocols . . . 173
9.5 Case study: Version control and synchronisation . . . 174
9.5.1 Typical use and requirements . . . 175
9.5.2 Platform requirements . . . 176
9.5.3 Architecture . . . 177
9.5.4 Protocols . . . 178
9.6 Case study: Web applications . . . 178
9.7 Summary . . . 180
Exercises . . . 180

10. Case Study: A Networked Game . . . 183
10.1 Motivation and organisation . . . 183
10.2 Outline structure and basic requirements . . . 184
10.3 Analysis and design . . . 185
10.3.1 Outline use cases . . . 186
10.3.2 Detailed design issues . . . 189
10.3.3 Security . . . 190
10.4 Protocol . . . 191
10.4.1 Protocol messages . . . 191
10.4.2 Client login . . . 192
10.4.3 Map server start-up and shutdown . . . 192
10.4.4 UDP messages . . . 194
10.4.5 Remarks on protocol . . . 196
10.4.6 Data view . . . 197
10.5 Implementation . . . 198
10.5.1 Admin server . . . 199

10.5.2 Map server 200
10.5.3 Player client 200
10.5.4 Running the example 201
10.6 Testing 202
10.7 Summary 202
Exercises 203

11. The End 205
11.1 Summary 205
11.2 Suggestions 207
11.2.1 Future directions 209
11.2.2 Interesting projects 210

A. Exercises: Hints and Comments 213

B. About the Example Code 245

Bibliography 247

Glossary 253

Index 267