

Preface

As long as the grass grows and the rivers flow....
From American Indians

Homo sum: humani nihil a me alienum puto.
In *Heautontimoroumenos*, Publius Terencius (194–129 BC)

... *Ce jour-là j'ai bien cru tenir quelque chose et que ma vie s'en trouverait changée.*
Mais rien de cette nature n'est définitivement acquis.
Comme une eau, le monde vous traverse et pour un temps vous prête ses couleurs.
Puis se retire et vous replace devant ce vide qu'on porte en soi, devant cette espèce
d'insuffisance centrale de l'âme qu'il faut bien apprendre à côtoyer, à combattre,
et qui, paradoxalement, est peut-être notre moteur le plus sûr.
In *L'usage du monde* (1963), Nicolas Bouvier (1929–1998)

What synchronization is

A concurrent program is a program made up of several entities (processes, peers, sensors, nodes, etc.) that cooperate to a common goal. This cooperation is made possible thanks to objects shared by the entities. These objects are called *concurrent objects*. Let us observe that a concurrent object can be seen as abstracting a service shared by clients (namely, the cooperating entities).

A fundamental issue of computing science and computing engineering consists in the design and the implementation of concurrent objects. In order that concurrent objects remain always consistent, the entities have to synchronize their accesses to these objects. Ensuring correct synchronization among a set of cooperating entities is far from being a trivial task. We are no longer in the world of sequential programming, and the approaches and methods used in sequential computing are of little help when one has to design concurrent programs. Concurrent programming requires not only great care but also knowledge of its scientific foundations. Moreover, concurrent programming becomes particularly difficult when one has to cope with failures of cooperating entities or concurrent objects.

Why this book?

Since the early work of E.W. Dijkstra (1965), who introduced the mutual exclusion problem, the concept of a process, the semaphore object, the notion of a weakest precondition, and guarded commands (among many other contributions), synchronization is no longer a catalog of tricks but a domain of computing science with its own concepts, mechanisms, and techniques whose results can be applied in many domains. This means that process synchronization has to be a major topic of any computer science curriculum.

This book is on synchronization and the implementation of concurrent objects. It presents in a uniform and comprehensive way the major results that have been produced and investigated in the past 30 years and have proved to be useful from both theoretical and practical points of view. The book has been written first for people who are not familiar with the topic and the concepts that are presented. These include mainly:

- Senior-level undergraduate students and graduate students in computer science or computer engineering, and graduate students in mathematics who are interested in the foundations of process synchronization.
- Practitioners and engineers who want to be aware of the state-of-the-art concepts, basic principles, mechanisms, and techniques encountered in concurrent programming and in the design of concurrent objects suited to shared memory systems.

Prerequisites for this book include undergraduate courses on algorithms and base knowledge on operating systems. Selections of chapters for undergraduate and graduate courses are suggested in the section titled “How to Use This Book” in the Afterword.

Content

As stressed by its title, this book is on algorithms, base principles, and foundations of concurrent objects and synchronization in shared memory systems, i.e., systems where the entities communicate by reading and writing a common memory. (Such a corpus of knowledge is becoming more and more important with the advent of new technologies such as multicore architectures.)

The book is composed of six parts. Three parts are more focused on base synchronization mechanisms and the construction of concurrent objects, while the other three parts are more focused on the foundations of synchronization. (A noteworthy feature of the book is that nearly all the algorithms that are presented are proved.)

- Part I is on lock-based synchronization, i.e., on well-known synchronization concepts, techniques, and mechanisms. It defines the most important synchronization problem in reliable asynchronous systems, namely the *mutual exclusion* problem (Chap. 1). It then presents several base approaches which have been proposed to solve it with machine-level instructions (Chap. 2). It also presents traditional approaches which have been proposed at a higher abstraction level to solve synchronization problems and implement concurrent objects, namely the concept of a semaphore and, at an even more abstract level, the concepts of monitor and path expression (Chap. 3).
- After the reader has become familiar with base concepts and mechanisms suited to classical synchronization in reliable systems, Part II, which is made up of a single chapter, addresses a fundamental concept of synchronization; namely, it presents and investigates the concept of *atomicity* and its properties. This allows for the formalization of the notion of a correct execution of a concurrent program in which processes cooperate by accessing shared objects (Chap. 4).
- Part I has implicitly assumed that the cooperating processes do not fail. Hence, the question: What does happen when cooperating entities fail? This is the main issue addressed in Part III (and all the rest of the book); namely, it considers that cooperating entities can halt prematurely (crash failure). To face the *net effect of asynchrony and failures*, it introduces the notions of *mutex-freedom* and associated progress conditions such as obstruction-freedom, non-blocking, and wait-freedom (Chap. 5).

The rest of Part III focuses on hybrid concurrent objects ([Chap. 6](#)), wait-free implementations of paradigmatic concurrent objects such as counters and store-collect objects ([Chap. 7](#)), snapshot objects ([Chap. 8](#)), and renaming objects ([Chap. 9](#)).

- Part IV, which is made up of a single chapter, is on *software transactional memory* systems. This is a relatively new approach whose aim is to simplify the job of programmers of concurrent applications. The idea is that programmers have to focus their efforts on which parts of their multiprocess programs have to be executed atomically and not on the way atomicity has to be realized ([Chap. 10](#)).
- Part V returns to the foundations side. It shows how reliable atomic read/write registers (shared variables) can be built from non-atomic bits. This part consists of three chapters. [Chapter 11](#) introduces the notions of *safe* register, *regular* register, and *atomic* register. Then, [Chap. 12](#) shows how to build an atomic bit from a safe bit. Finally, [Chap. 13](#) shows how an atomic register of any size can be built from safe and atomic bits.

This part shows that, while atomic read/write registers are easier to use than safe read/write registers, they are not more powerful from a computability point-of-view.

- Part VI, which concerns also the foundations side, is on the *computational power of concurrent objects*. It is made up of four chapters. It first introduces the notion of a *consensus object* and shows that consensus objects are universal objects ([Chap. 14](#)). This means that, as soon as a system provides us with atomic read/write registers and consensus objects, it is possible to implement in a wait-free manner any object defined from a sequential specification.

Part VI then introduces the notion of *self-implementation* and shows how atomic registers and consensus objects can be built from base objects of the same type which are not reliable ([Chap. 15](#)). Then, it presents the notion of a *consensus number* and the associated *consensus hierarchy* which allows the computability power of concurrent objects to be ranked ([Chap. 16](#)). Finally, the last chapter of the book focuses on the wait-free implementation of consensus objects from read/write registers and failure detectors ([Chap. 17](#)).

To have a more complete feeling of the spirit of this book, the reader can also consult the section “What Was the Aim of This Book” in the **Afterword**) which describes what it is hoped has been learned from this book. Each chapter starts with a short presentation of its content and a list of keywords; it terminates with a summary of the main points that have explained and developed. Each of the six parts of the book is also introduced by a brief description of its aim and its technical content.

Acknowledgments

This book originates from lecture notes for undergraduate and graduate courses on process synchronization that I give at the University of Rennes (France) and, as an invited professor, at several universities all over the world. I would like to thank the students for their questions that, in one way or another, have contributed to this book.

I want to thank my colleagues Armando Castañeda (UNAM, MX), Ajoy Datta (UNLV, Nevada), Achour Mostéfaoui (Université de Nantes), and François Taiani (Lancaster University, UK) for their careful reading of chapters of this book. Thanks also to François Bonnet (JAIST, Kanazawa), Eli Gafni (UCLA), Damien Imbs (IRISA, Rennes), Segio Rajsbaum (UNAM, MX), Matthieu Roy (LAAS, Toulouse), and Corentin Travers (LABRI, Bordeaux) for long discussions on wait-freedom. Special thanks are due to Rachid Guerraoui (EPFL), with whom I discussed numerous topics presented in this book (and many other topics) during the past seven years. I would also like to thank Ph. Louarn (IRISA, Rennes) who was my Latex man when writing this book, and Ronan Nugent (Springer) for his support and his help in putting it all together.

Last but not least (and maybe most importantly), I also want to thank all the researchers whose results are presented in this book. Without their work, this book would not exist (Since I typeset the entire text myself (– $\text{\LaTeX}2_{\epsilon}$ for the text and *xfig* for figures–), any typesetting or technical errors that remain are my responsibility.)

Michel Raynal
Professeur des Universités
Institut Universitaire de France
IRISA-ISTIC, Université de Rennes 1
Campus de Beaulieu, 35042 Rennes, France

September–November 2005 and June–October 2011
Rennes, Mont-Louis (ARCHI'11), Gdańsk (SIROCCO'11), Saint-Philibert,
Hong Kong (PolyU), Macau, Roma (DISC'11), Tirana (NBIS'11),
Grenoble (SSS'11), Saint-Grégoire, Douelle, Mexico City (UNAM).



<http://www.springer.com/978-3-642-32026-2>

Concurrent Programming: Algorithms, Principles, and
Foundations

Raynal, M.

2013, XXXII, 516 p., Hardcover

ISBN: 978-3-642-32026-2