Nonsequential and Distributed Programming with Go

Christian Maurer

Nonsequential and Distributed Programming with Go

Synchronization of Concurrent Processes: Communication—Cooperation—Competition

Springer

Christian Maurer
Institut für Informatik
Freie Universität Berlin
Berlin, Germany

ISBN 978-3-658-29781-7 ISBN 978-3-658-29782-4 (eBook)
https://doi.org/10.1007/978-3-658-29782-4

Responsible Editor: Sybille Thelen
This Springer imprint is published by the registered company Springer Fachmedien Wiesbaden GmbH part of Springer Nature.
The registered company address is: Abraham-Lincoln-Str. 46, 65189 Wiesbaden, Germany

Dedicated to my golden angel

Preface

First of all, we have to explain, why—by not following the "mainstream"—

"Nonsequential Programming"

was chosen as title of this book instead of

"Concurrent Programming".

In the German language, there exist two words for "concurrent":

- *nebenläufig* and
- *nichtsequentiell.*

The first word is the exact translation of "concurrent"[1] , the second is simply the negation of *sequentiell*, i.e. *"nichtsequentiell"* is just "not *sequentiell*".

So, strictly speaking, "concurrent programming" means, that several developers are working together to write a program, but makes no sense for a book title; whereas "nonsequential programming" means the development of nonsequential programs, which is our topic. *Processes* with access to shared resources may run *concurrently* on one or more computers; *algorithms,* however, cannot be *concurrent*—they are either *sequential* or not. This is probably the reason, why in all German universities lectures on this topic are named "Nichtsequentielle Programmierung".

This is why we prefer to use the German word for "nonsequential" in the context of "programs" and "programming", as in our opinion that adjective is much better suited for this purpose, and stick to the German term in so far, as we use the spelling "nonsequential" instead of "non-sequential". Furthermore,

▶ we will generally use the abbreviation "NSP" for "nonsequential programming".

[1]Latin: *concurrere* = *cum* + *currere*; *cum* = (together) with, *currere* = run, go, walk

Basic techniques of nonsequential programming are either the subject of a separate lecture in the bachelor's program of computer science studies or are part of lectures on areas, in which nonsequential algorithms play an essential role.

This book systematically develops basic concepts for the synchronization of concurrent processes and for their *communication*[2] —even in the distributed case:

- locks,
- semaphores,
- fairness and deadlocks,
- monitors,
- message passing,
- network-wide message passing.
- exploration of networks,
- traversal in networks and
- selection of a leader in them.

The algorithms are formulated in Go (see https://golang.org). Among other things, this language is characterized by the following features:

- a C-like syntax (see https://golang.org/ref/spec)—but with significant influence from Wirth's languages (see https://golang.org/doc/fac#history),
- the trend towards *concise language* and freedom from language stuttering of the kind " foo.Foo* myFoo = new(foo.Foo)",
- garbage collection,
- the "rebellion" against cumbersome type systems such as e.g. in C++ or Java, and consequently
- a very expressive *type system* with *static type check when compiling a source text*, *dynamic type adjustment at runtime* as well as a *rigid dependency analysis,*
- but *without* type hierarchy ("types just are, they don't have to announce their relationships")—so *lighter* than in typical OO languages,
- "orthogonality of the concepts" ("methods can be implemented for any type; structures represent data while "interfaces" represent abstraction.") and
- the installation of various constructs for NSP, including message passing—inspired by Hoare's CSP calculus—as well as support for parallel, multicore and networked computing.

[2]Latin: *communicare* = to communicate, to share, to discuss

More information can be found in the net at the beginning of the *Frequently Asked Questions* by Go (https://golang.org/doc/faq).

To run the programs from the book, the installation of Go version 1.9 (or higher) is assumed.

Basic knowledge of Go is certainly helpful for understanding the basic concepts, but not necessary because of the simple syntax.

However, familiarizing yourself with Go is *significantly* (!) easier than, for example, acquiring the knowledge in Java required to understand algorithms in comparable depth.

If you don't want to do this, you can understand the Go source texts "meta-linguistically"—such as the language-independent representations of algorithms in the textbooks of Andrews, Ben-Ari, Herrtwich/Hommel, Raynal or Taubenfeld .

At "switching points"—*locks, semaphores, monitors* and *network-wide message passing*—some basic approaches to programming in C and Java are also presented.

Knowledge of imperative representations of *abstract data objects,* of the principles of *"information-hiding"* and the basic concepts of *object-oriented programming* are however required; some important information—as far as it is needed for this book—can be found in Chap. 2 about packages, interfaces and abstract data types.

The content structure is inductive:

The growing distance from the machine is associated with increasing abstraction, which corresponds very naturally to the historical development of NSP: locks—semaphores—monitors—message passing. This structure can already be described as *traditional,* as it can also be found in most other textbooks.

Reading is therefore essentially only meaningful sequentially, i.e. in the order of the chapters.

Focusing on Go would have offered an alternative order of the contents, starting with the highest level of abstraction, the passing of messages via channels—a core concept of this language—which is particularly recommended by its developers as the means of choice for synchronization (see Sect. 1.9 on process states in the introduction and Chap. 11 about messages). This path is deliberately not followed here, however, in order to be able to use the book in environments in which other languages are favoured.

A fundamental principle of the book consists in the fact that *the same classic examples* are taken up *again and again,* which facilitates the comparison between the presented concepts and language tools.

The readers are *strongly* recommended,

- to work through these examples carefully in order to gain deeper insight into the similarities and differences between the concepts,
- to perceive the suggestions for their own activities, i.e. to take up as many suggestions for exercises as possible,
- to go *ad fontes* (original papers are fundamental sources, because they provide deeper insights into the systematics of the development of NSP),

- and—either—to install Go and *get all examples and programs running*—or—to implement the algorithms in their preferred programming language, in order to acquire practical programming competence.

Compared to the third edition, some bugs have been fixed and minor enhancements added.

The *essential* difference to the 3rd edition, however, is that—due to a change in the Go system—adjustments were required: The previous basic assumption that a value assignment of a constant or variable to a variable of an elementary type is atomic is now only valid in Go if the computer used has only one processor or if `runtime.GOMAXPROCS(1)` was called.

Therefore, the value assignments to shared variables used in the entry and exit protocols to protect critical sections now *principally* need to be replaced by an atomic instruction—realized with an indivisible machine instruction.

For some of the algorithms given in the book test runs can be carried out in which their sequence is visualized dynamically. The corresponding program files, some of which use C-library-routines and header files in /usr/include, are included in the source texts of this book.

I would like to thank Mrs. Dipl.-Inf. Sybille Thelen from Springer-Verlag very much; she also supported the publication of this edition very kindly again.

In June 2018 she informed me, that *Springer Nature* had selected my book among some others to create in cooperation with *DeepL* an English version. I was very happy about that idea and immediately agreed. In the middle of December 2019 I got the result, but I after having read the first pages of it was a bit disappointed on quite a lot of mistakes in the computer aided translation. The were some really funny ones as, e.g., “castle” and “lure” for lock, “trial” and “lawsuit” for process, “net curtain” for “store”, “voltage tree” and “current beam” for spanning tree, “beacon” for signal, “embassy” for message and “episode” for sequence and, furthermore, lots of things like the remarkable statement “Any deterministic algorithm is deterministic, but not *any* deterministic algorithm is necessary deterministic.” But there were also several of really annoying errors such as grammar mistakes, rather strange orders of words in sentences and many wrong translations that completely distorted contents. However, this experience implies, that I will *never* sit in an AI-controlled car.

I have to thank the book production teams of Springer Nature, Heidelberg, and of Scientific Publishing Services Ltd., Chennai: They have improved my corrections of the DeepL-translation. I would also like to thank Mrs. Ivonne Eling for organizing things in the background and Mr. Stefan Schmidt for several suggestions.

Finally, an important hint: Despite constant and thorough comparison of all source texts reproduced in the book in the course of their development, it cannot be ruled out with *absolute* certainty, that there are inconsistencies somewhere. (Hints to discovered discrepancies or errors are of course very gratefully accepted!)

All source texts are is available on the *pages of the book in the worldwide web:* https://maurer-berlin.eu/nspbook/4.

Berlin
January 25th and November 20th 2020

Christian Maurer

Contents

List of Figures

List of Tables