
Nonsequential and Distributed Programming with Go

Christian Maurer

Nonsequential and Distributed Programming with Go

Synchronization of Concurrent Processes:
Communication—Cooperation—
Competition

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>

© Springer Fachmedien Wiesbaden GmbH, part of Springer Nature 2021

The translation was done with the help of artificial intelligence (machine translation by the service DeepL.com). A subsequent human revision was done primarily in terms of content.

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

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 “non-sequential 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 “non-sequential” instead of “non-sequential”. Furthermore,

- we will generally use the abbreviation “NSP” for “nonsequential programming”.

¹Latin: *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.

²Latin: *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. There 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 available on the *pages of the book in the worldwide web*:
<https://maurer-berlin.eu/nspbook/4>.

Berlin

Christian Maurer

January 25th and November 20th 2020

Contents

1	Introduction	1
1.1	Definition of Terms	1
1.2	Motivation and Applications	4
1.3	The (Non-)Sense of Testing	7
1.4	Examples of Concurrent Algorithms	8
1.5	Concurrency and the Informal Notion of a Process	11
1.6	Conflicts When Addressing Shared Data	14
1.7	Atomic Statements	19
1.8	Critical Sections and Lock Synchronization	20
1.9	Process States	22
1.9.1	Process Transitions in C, Java, and Go	23
1.9.2	Example in C, Java, and Go	30
	References.	31
2	Packages, Interfaces, and Abstract Data Types	33
2.1	The Role of Packages	33
2.1.1	Packages Only as Interfaces	35
2.2	All Source Codes from This Book in the nUniverse Package	36
2.3	The Package Object	36
2.3.1	Any	36
2.3.2	Interfaces for Describing Objects	37
2.3.3	The Interface of the Package	40
2.3.4	Queues as Abstract Data Type	41
2.3.5	A Queue as an Abstract Data Object	44
2.3.6	Bounded Buffers	45
2.4	On the Problem of References	46
	References.	47
3	Locks	49
3.1	Specification of Locks	49
3.2	Locks in C, Java, and Go	51

3.2.1	Locks in C	51
3.2.2	Locks in Java	52
3.2.3	Locks in Go	52
3.3	Locks Based on Indivisible Machine Instructions	53
3.3.1	Test and Set	54
3.3.2	Compare and Swap	57
3.3.3	Exchange	58
3.3.4	Decrement	59
3.3.5	Fetch and Increment	60
3.3.6	The Counter Problem	61
3.3.7	Evaluation of the Use of Machine Instructions	61
3.4	Lock Algorithms for 2 Processes at High-Level Language Level	63
3.4.1	On the Indivisibility of Value Assignments	63
3.4.2	Approaches to the Development of a Correct Algorithm ...	64
3.4.3	Algorithm of Peterson	67
3.4.4	Algorithm of Kessels	69
3.4.5	Algorithm of Dekker	70
3.4.6	Algorithm of Doran and Thomas	72
3.4.7	Algorithm of Hyman	73
3.5	Lock Algorithms for Several Processes	73
3.5.1	Tiebreaker Algorithm of Peterson	75
3.5.2	Algorithm of Dijkstra	76
3.5.3	Algorithm of Knuth	78
3.5.4	Algorithm of Eisenberg and McGuire	80
3.5.5	Algorithm of Habermann	81
3.5.6	Ticket Algorithm	82
3.5.7	Bakery Algorithm of Lamport	83
3.5.8	Algorithm of Kessels for N Processes	85
3.5.9	Algorithm of Morris	86
3.5.10	Algorithm of Szymanski	89
3.6	Locks as Abstract Data Types	93
	References	95
4	Semaphores	97
4.1	Disadvantages of the Implementation of Locks	97
4.2	Dijkstra's Approach	98
4.3	Binary Semaphores	99
4.3.1	Equivalence of Locks and Binary Semaphores	100
4.3.2	Algorithm of Udding	100
4.4	Buffers in the Nonsequential Case	102

4.5	General Semaphores	105
4.5.1	Specification of General Semaphores	106
4.5.2	Development of a Correct Implementation	107
4.6	Unbounded Buffers and the Sleeping Barber	111
4.7	Construction of General Semaphores from Binary Semaphores	115
4.7.1	Representation	116
4.7.2	Naive (Wrong) Approach	116
4.7.3	Correction and Consequence	118
4.7.4	The Algorithm of Barz	119
4.8	Semaphores in C, Java, and Go	120
4.8.1	Semaphores in C	120
4.8.2	Semaphores in Java	121
4.8.3	Semaphores in Go	123
4.9	Additive Semaphores	123
4.9.1	Multiple Semaphores	125
4.10	Barrier Synchronization	125
4.11	Shortest Job Next	127
4.12	The Readers–Writers Problem	128
4.12.1	The First Readers–Writers Problem	128
4.12.2	The Second Readers–Writers Problem	129
4.12.3	Priority Adjustment	130
4.12.4	Implementation with Additive Semaphores	133
4.12.5	Efficient Implementation in Go	133
4.12.6	The Readers–Writers Problem as an Abstract Data Type ...	133
4.13	The left–right Problem	134
4.14	The Dining Philosophers	136
4.15	The Problem of the Cigarette Smokers	142
4.16	Implementation of Semaphores	145
4.16.1	The Convoy Phenomenon	147
	References	147
5	The Baton Algorithm	149
5.1	Development of the Problem	149
5.1.1	The Baton of Andrews	150
5.2	The Readers–Writers Problem	152
5.3	The Second Left–Right Problem	156
	References	157
6	Universal Critical Sections	159
6.1	Basic Idea and Construction	159
6.1.1	Specification	161
6.1.2	Implementation	162

6.2	Semaphores	164
6.3	The Sleeping Barber	165
6.4	The Readers–Writers Problem	166
6.5	The Left–Right Problem	167
6.6	Universal Critical Resources	169
6.7	The Dining Philosophers	172
6.8	The Problem of the Cigarette Smokers	174
	References	174
7	Fairness	175
7.1	Weak Versus Strong Fairness	175
8	Deadlocks	179
8.1	Characterization	179
8.1.1	A Simple Examples	180
8.2	Countermeasures	182
8.2.1	Prevention	182
8.2.2	Detection and Recovery	183
8.2.3	Avoidance	184
8.2.4	The Bankers’ Algorithm	185
8.3	Probability of Deadlocks	188
8.4	Evaluation of the Countermeasures	190
	References	191
9	Monitors	193
9.1	Characterization of Monitors	193
9.1.1	Hoare’s Approach	194
9.1.2	Virtual Monitors in Go	195
9.2	Condition Variables	196
9.3	Monitors in C, Java, and Go	200
9.3.1	Monitors in C	200
9.3.2	Monitors in Java	202
9.3.3	Monitors in Go	204
9.4	The Bounded Buffer	206
9.5	The Readers–Writers Problem	207
9.6	Signal Semantics	209
9.6.1	Signal and Continue	209
9.6.2	Signal and Wait	209
9.6.3	Signal and Urgent Wait	210
9.6.4	Preemptive Versus Nonpreemptive Semantics	210
9.6.5	Comparing Evaluation of the Signal Semantics	211
9.6.6	A Semaphore as Monitor	211
9.6.7	Barrier Synchronization	213

9.7	Broadcast in C, Java, and Go	214
9.7.1	Broadcast in C	215
9.7.2	Broadcast in Java	215
9.7.3	Broadcast in Go	216
9.8	The Sleeping Barber: Haircut as a Rendezvous	216
9.9	Priority Rules	218
9.9.1	Hoare's Alarm Clock	218
9.9.2	Shortest Job next	219
9.10	Equivalence of the Semaphore and the Monitor Concepts	220
9.11	Implementation of the Monitor Concept	223
9.12	The Problem of Nested Monitor Calls	225
	References	225
10	Universal Monitors	227
10.1	The Basic Idea	227
10.1.1	Specification	228
10.1.2	Implementation	229
10.2	Conditioned Universal Monitors	231
10.2.1	Specification	231
10.2.2	Implementation	232
10.3	Semaphores	233
10.4	Account	234
10.5	Bounded Buffers	235
10.6	The Sleeping Barber	236
10.7	Barrier Synchronization	237
10.8	The Readers–Writers Problem	237
10.9	The Left–Right Problem	239
10.10	The Dining Philosophers	240
10.11	The Cigarette Smokers	242
11	Message Passing	245
11.1	Channels and Messages	245
11.1.1	Syntax of Message Passing in Go	247
11.1.2	Synchronous Message Passing with Asynchronous	249
11.2	Asynchronous Communication	250
11.2.1	Semaphores	250
11.2.2	Bounded Buffers	251
11.2.3	The Cigarette Smokers	251
11.3	Networks of Filters	252
11.3.1	Caesar's Secret Messages	252
11.3.2	The Sieve of Eratosthenes	254
11.3.3	Mergesort	255
11.3.4	The Dining Philosophers	257

11.4	Selective Waiting	259
11.5	The Client–Server Paradigm	261
11.6	Synchronous Communication	262
11.6.1	Semaphores	262
11.6.2	Bounded Buffers	262
11.6.3	Equivalence of Synchronous and Asynchronous Message Passing	263
11.6.4	The Readers–Writers Problem	265
11.6.5	The Left–Right Problem	266
11.7	Guarded Selective Waiting	267
11.7.1	Semaphores	268
11.7.2	Bounded Buffers	269
11.7.3	The Readers–Writers Problem	270
11.7.4	The Left–Right Problem	270
11.8	Equivalence of Message Passing and the Semaphore Concept	271
11.9	Duality Between Monitors and Servers	274
	References	274
12	Comparison of the Previous Language Constructs	275
12.1	Locks	275
12.2	Semaphores	275
12.3	Monitors	276
12.4	Message Passing	276
13	Netwide Message Passing	277
13.1	Channels in the Network	277
13.1.1	Technical Aspects (in C)	278
13.1.2	Remarks on the Realization in Java	279
13.2	Realization in Go	280
13.3	1:1-Network Channels Between Processes on Arbitrary Computers	280
13.3.1	A Simple Example	284
13.4	Distributed Locks Due to Ricart/Agrawala	285
	References	292
14	Universal Far Monitors	293
14.1	Extension of Net Channels to the Case 1:n	293
14.2	Construction of the Far Monitors	296
14.2.1	Specification	296
14.2.2	Implementation	297
14.3	Correctness	299
14.4	Distributed Semaphores	300
14.5	Distributed Queues and Bounded Buffers	301

14.6	Distributed Readers–Writers and Left–Right Problems	302
14.7	Account	304
14.8	Remote Procedure Calls	305
14.8.1	Example of a Remote Procedure Call	309
	References	310
15	Networks as Graphs	311
15.1	Graphs	311
15.1.1	Definition of the Graph Concept	312
15.2	Realization in Go	312
15.2.1	Specification	313
15.2.2	Implementation	317
15.3	Adjacency Matrices	321
15.3.1	Specification	322
15.3.2	Implementation	324
15.4	Distributed Graphs	327
15.4.1	Specification	327
15.4.2	Implementation	327
15.5	Examples	330
15.5.1	Output to the Screen	333
	References	334
16	Heartbeat Algorithms	335
16.1	The Basic Idea	335
16.2	Getting to Know the Network	336
16.3	Preconditions for the Realization in Go	338
16.4	Matrix-Based Solution	339
16.5	Graph-Based Solutions	341
16.5.1	With Knowledge of the Diameter of the Network Graph	341
16.5.2	Without Global Knowledge	342
	References	343
17	Traversing Algorithms	345
17.1	Preconditions for the Realization in Go	345
17.2	Distributed Depth-First Search	348
17.2.1	Transfer of the Spanning Tree to All Processes	355
17.2.2	Realization with Far Monitors and Transfer of the Spanning Tree	356
17.3	Algorithm of Awerbuch	359
17.3.1	Realization with Far Monitors	359
17.3.2	Transfer of the Spanning Tree to All Processes	362
17.3.3	Algorithm of H��lary/Raynal	364

17.4	Construction of a Ring	366
17.4.1	Transfer of the Ring to All Processes	368
17.5	Distributed Breadth-First Search	368
17.5.1	Realization with Far Monitors	375
17.5.2	Realization with Far Monitors and Transfer of the Spanning Tree	377
	References.....	380
18	Leader Election Algorithms	381
18.1	Basics	381
18.2	Preconditions for the Realization in Go	382
18.3	Algorithm of Chang/Roberts	383
18.4	Algorithm of Hirschberg/Sinclair	384
18.5	Algorithm of Peterson	391
18.6	Election with Depth-First Search	394
	References.....	394
	Further Literature	397
	Index.....	401

List of Figures

Fig. 1.1	Rough division of the universe of algorithms	3
Fig. 1.2	State transitions	22
Fig. 1.3	D is blocked until C has terminated	27
Fig. 1.4	D can start immediately after the end of B	28
Fig. 3.1	Rooms and doors in the algorithm of Morris	87
Fig. 4.1	The shop of the barber	112
Fig. 4.2	Oncoming rail traffic	135
Fig. 4.3	The table of Plato, Socrates, Aristoteles, Cicero, and Heraklith	136
Fig. 8.1	Cross road with four quadrants	180
Fig. 8.2	Resource graph	183
Fig. 8.3	Waiting graph	183
Fig. 9.1	State transitions for monitors	224
Fig. 11.1	Channel architecture of the sieve of ERATOSTHENES	254
Fig. 13.1	B and C send their start time to the others	289
Fig. 13.2	A and C send their “ok”	289
Fig. 13.3	The third step	289
Fig. 13.4	The fourth step	290
Fig. 13.5	A has received both “ok” replies	290
Fig. 14.1	Principle of the execution of remote procedure calls	307
Fig. 15.1	Undirected graph with 8 vertices and 10 edges	330
Fig. 16.1	Situation at the beginning of the algorithms	336
Fig. 16.2	Situation after the first heartbeat	337
Fig. 16.3	Situation after the second heartbeat	337
Fig. 16.4	Situation after the third heartbeat for processes 2 and 6	337
Fig. 16.5	Adjacency matrix after four heartbeats	340
Fig. 16.6	Adjacency matrix of process 5 after the first heartbeat	340
Fig. 17.1	Spanning tree of the depth-first search in G8 with process 4 as root	353
Fig. 17.2	Spanning tree of the depth-first search in G8 with process 1 as root	354

Fig. 17.3	Order of the steps at depth-first search in G_8 with process 4 as root.	354
Fig. 17.4	Ring after depth-first search in G_8 with process 4 as root.	367
Fig. 17.5	Spanning tree of the breadth-first search in G_8 with process 4 as root	374
Fig. 18.1	Circular graph with 8 processes.	382

List of Tables

Table 1.1	Fictitious assembler language	16
Table 1.2	Possible sequence of executions	16
Table 1.3	Possible execution sequence	18
Table 3.1	Mutual exclusion not guaranteed	65
Table 3.2	Short time delay	69
Table 3.3	Possible execution sequence in Hyman's algorithm	73
Table 3.4	There is no mutual exclusion.	74
Table 3.5	States in the algorithm of SZYMANSKI	90
Table 3.6	Coding of the status values	90
Table 4.1	Barber error	114
Table 4.2	V-operation lost	117
Table 6.1	Synchronization of the readers–writers problems.	160
Table 6.2	Synchronization of the barber's problems.	165
Table 6.3	Synchronization of the second left–right problem	168
Table 6.4	Synchronization of the philosophers with states	173
Table 6.5	Synchronization of the philosophers with forks	173
Table 8.1	Deadlock between two processes	181
Table 8.2	Safe state	185
Table 8.3	Remaining credit request of 0 can be granted.	186
Table 8.4	Customer 2 can be satisfied.	186
Table 8.5	Unsafe state	186
Table 8.6	Cash balance too small	187
Table 10.1	Synchronization of semaphores.	233
Table 10.2	Account monitor	234
Table 10.3	Barber monitor	236
Table 10.4	Readers–writers monitor	238
Table 11.1	Duality between the procedure-orientated and the message-oriented approach.	273
Table 17.1	The cases for the sendings in the depth-first search	349
Table 17.2	Father and child[ren] of processes 0 to 7 at depth first search	353

Table 17.3	Discovering and finishing times of processes 0 to 7	353
Table 17.4	Output of processes 0 to 7 at the construction of a ring	367
Table 17.5	Father and child[ren] of processes 0 to 7 at depth-first search with root 4	374