Hands-On System Programming with Go

Build modern and concurrent applications for Unix and Linux systems using Golang

Packt

www.packt.com

Alex Guerrieri

Hands-On System Programming with Go

Build modern and concurrent applications for Unix and Linux systems using Golang

Alex Guerrieri

Packt

BIRMINGHAM - MUMBAI

Hands-On System Programming with Go

Commissioning Editor: Richa Tripathi
Acquisition Editor: Shriram Shekhar
Content Development Editor: Tiksha Sarang
Senior Editor: Afshaan Khan
Technical Editor: Sabaah Navlekar
Copy Editor: Safis Editing
Language Support Editor: Storm Mann
Project Coordinator: Prajakta Naik
Proofreader: Safis Editing
Indexer: Rekha Nair
Production Designer: Shraddha Falebhai

First published: July 2019

Production reference: 1040719

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-78980-407-2

`www.packtpub.com`

To my family, who laid the foundation of the path I am on today

– Alex Guerrieri

Contributors

About the author

Alex Guerrieri is a software developer who specializes in backend and distributed systems. Go has been his favorite tool for the job since first using it in 2013. He holds a degree in computer science engineering and has been working in the field for more than 6 years. Originally from Italy, where he completed his studies and started his career as a full-stack developer, he now lives in Spain, where he previously worked in two start-ups—source{d} and Cabify. He is now working for three companies—as a software crafter for BBVA, one of the biggest Spanish banks; as a software architect for Security First, London, a company focusing on digital and physical security; and as a cofounder of DauMau, the company behind Vidsey, software that generates videos in a procedural manner.

To all the people in my life who supported me and put up with me throughout this endeavor of writing this book, thank you.

About the reviewers

Corey Scott is a principal software engineer currently living in Melbourne, Australia. He has been programming professionally since 2000, with the last 5 years spent building large-scale distributed services in Go.

A blogger on a variety of software-related topics, he is passionate about designing and building quality software. He believes that software engineering is a craft that should be honed, debated, and continuously improved. He takes a pragmatic, non-zealous approach to coding, and is always up for a good debate about software engineering, continuous delivery, testing, or clean coding.

Janani Selvaraj is currently working as a data analytics consultant for Gaddiel Technologies, Trichy, where she focuses on providing data analytics solutions for start-up companies. Her previous experience includes training and research development in relation to data analytics and machine learning.

She has a PhD in environmental management and has more than 5 years' research experience with regard to statistical modeling. She is also proficient in a number of programming languages, including R, Python, and Go.

She reviewed a book entitled *Go Machine Learning Projects*, and also coauthored a book entitled *Machine Learning Using Go*, published by *Packt Publishing*.

Arun Muralidharan is a software developer with over 9 years' experience as a systems developer. Distributed system design, architecture, event systems, scalability, performance, and programming languages are some of the aspects of a product that interest him the most. Professionally, he spends most of his time coding in C++, Python, and C (and perhaps Go in the near future). Away from his job, he also develops software in Go and Rust.

> *I would like to take this opportunity to thank my family, who have provided me with unconditional support over the years.*

Table of Contents

Section 4: Deep Dive into Concurrency

Preface

This book will provide good, in-depth explanations of various interesting Go concepts. It begins with Unix and system programming, which will help you understand what components the Unix operating system has to offer, from the kernel API to the filesystem, and allow you to familiarize yourself with the basic concepts of system programming.

Next, it moves on to cover the application of I/O operations, focusing on the filesystem, files, and streams in the Unix operating system. It covers many topics, including reading from and writing to files, among other I/O operations.

This book also shows how various processes communicate with one another. It explains how to use Unix pipe-based communication in Go, how to handle signals inside an application, and how to use a network to communicate effectively. Also, it shows how to encode data to improve communication speed.

The book will, toward the end, help you to understand the most modern feature of Go—concurrency. It will introduce you to the tools the language has, along with sync and channels, and how and when to use each one.

Who this book is for

This book is for developers who want to learn system programming with Go. Although no prior knowledge of Unix and Linux system programming is necessary, some intermediate knowledge of Go will help you to understand the concepts covered in the book.

What this book covers

`Chapter 1`, *An Introduction to System Programming*, introduces you to Go and system programming and provides some basic concepts and an overview of Unix and its resources, including the kernel API. It also defines many concepts that are used throughout the rest of the book.

`Chapter 2`, *Unix OS Components*, focuses on the Unix operating system and the components that you will interact with—files and the filesystem, processes, users and permissions, threads, and others. It also explains the various memory management techniques of the operating system, and how Unix handles resident and virtual memory.

`Chapter 3`, *An Overview of Go*, takes a look at Go, starting with some history of the language and then explaining, one by one, all its basic concepts, starting with namespaces and the type system, variables, and flow control, and finishing with built-in functions and the concurrency model, while also offering an explanation of how Go interacts and manages its memory.

`Chapter 4`, *Working with the Filesystem*, helps you to understand how the Unix filesystem works and how to master the Go standard library to handle file path operations, file reading, and file writing.

`Chapter 5`, *Handling Streams*, helps you to learn about the interfaces for the input and output streams that Go uses to abstract data flows. It explains how they work and how to combine them and best use them without leaking information.

`Chapter 6`, *Building Pseudo-Terminals*, helps you understand how a pseudo-terminal application works and how to create one. The result will be an interactive application that uses standard streams just as the command line does.

`Chapter 7`, *Handling Processes and Daemons*, provides an explanation of what processes are and how to handle them in Go, how to start child processes from a Go application, and how to create a command-line application that will stay in the background (a daemon) and interact with it.

`Chapter 8`, *Exit Codes, Signals, and Pipes*, discusses Unix inter-process communication. It explains how to use exit codes effectively. It shows you how signals are handled by default inside an application, and how to manage them with some patterns for effective signal handling. Furthermore, it explains how to connect the output and input of different processes using pipes.

`Chapter 9`, *Network Programming*, explains how to use a network to make processes communicate. It explains how network communication protocols work. It initially focuses on low-level socket communication, such as TCP and UDP, before moving on to web server development using the well-known HTTP protocol. Finally, it shows how to use the Go template engine.

`Chapter 10`, *Data Encoding Using Go*, explains how to leverage the Go standard library to encode complex data structures in order to facilitate process communications. It analyzes both text-based protocols, such as XML and JSON, and binary-based protocols, such as GOB.

`Chapter 11`, *Dealing with Channels and Goroutines*, explains the basics of concurrency and channels and some general rules that prevent the creation of deadlocks and resource-leaking inside an application.

`Chapter 12`, *Synchronization with sync and atomic*, discusses the synchronization packages of the `sync` and `sync/atomic` standard libraries, and how they can be used instead of channels to achieve concurrency easily. It also focuses on avoiding the leaking of resources and on recycling resources.

`Chapter 13`, *Coordination Using Context*, discusses `Context`, a relatively new package introduced in Go that offers a simple way of handling asynchronous operations effectively.

`Chapter 14`, *Implementing Concurrency Patterns*, uses the tools from the previous three chapters and demonstrates how to use and combine them to communicate effectively. It focuses on the most common patterns used in Go for concurrency.

`Chapter 15`, *Using Reflection*, explains what reflection is and whether you should use it. It shows where it's used in the standard library and guides you in creating a practical example. It also shows how to avoid using reflection where there is no need to.

`Chapter 16`, *Using CGO*, explains how CGO works and why and when you should use it. It explains how to use C code inside a Go application, and vice versa.

To get the most out of this book

Some basic knowledge of Go is required to try the examples and to build modern applications.

Each chapter includes a set of questions that will help you to gauge your understanding of the chapter. The answers to these questions are provided in the *Assessments* section of the book. These questions will prove very beneficial for you, as they will help you revisit each chapter at a glance.

Apart from this, each chapter provides you with instructions on how to run the code files, while the GitHub repository of the book provides the requisite details.

Download the example code files

You can download the example code files for this book from your account at `www.packt.com`. If you purchased this book elsewhere, you can visit `www.packt.com/support` and register to have the files emailed directly to you.

You can download the code files by following these steps:

1. Log in or register at `www.packt.com`.
2. Select the **SUPPORT** tab.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box and follow the onscreen instructions.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR/7-Zip for Windows
- Zipeg/iZip/UnRarX for Mac
- 7-Zip/PeaZip for Linux

The code bundle for the book is also hosted on GitHub at `https://github.com/PacktPublishing/Hands-On-System-Programming-with-Go`. In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: `https://static.packt-cdn.com/downloads/9781789804072_ColorImages.pdf`.

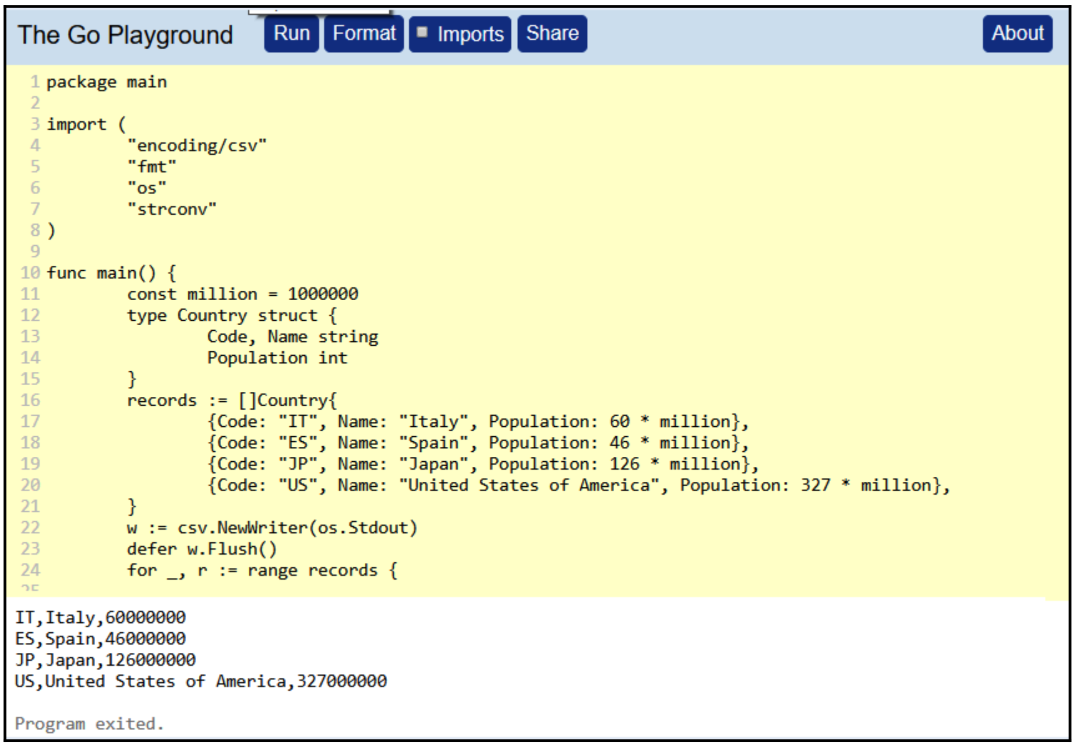
Code in Action

Visit the following link to check out videos of the code being run: `http://bit.ly/2ZWgJb5`.

Playground examples

In the course of the book you will find many snippets of code followed by a link to `https://play.golang.org`, a service that allows you to run Go applications with some limitations. You can read more about it at `https://blog.golang.org/playground`.

In order to see the full source code of such examples, you need to visit the Playground link. Once on the website, you can press the **Run** button to execute the application. The bottom part of the page will show the output. The following is an example of the code running in the Go Playground:



```go
package main

import (
	"encoding/csv"
	"fmt"
	"os"
	"strconv"
)

func main() {
	const million = 1000000
	type Country struct {
		Code, Name string
		Population int
	}
	records := []Country{
		{Code: "IT", Name: "Italy", Population: 60 * million},
		{Code: "ES", Name: "Spain", Population: 46 * million},
		{Code: "JP", Name: "Japan", Population: 126 * million},
		{Code: "US", Name: "United States of America", Population: 327 * million},
	}
	w := csv.NewWriter(os.Stdout)
	defer w.Flush()
	for _, r := range records {
```

```
IT,Italy,60000000
ES,Spain,46000000
JP,Japan,126000000
US,United States of America,327000000

Program exited.
```

If you want, you have the possibility of experimenting by adding and editing more code to the examples, and then running them.

Conventions used

There are a number of text conventions used throughout this book.

`CodeInText`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "This type of service includes `load`, which adds a program to memory and prepares for its execution before passing control to the program itself, or `execute`, which runs an executable file in the context of a pre-existing process."

A block of code is set as follows:

```
<meta name="go-import" content="package-name vcs repository-url">
```

Any command-line input or output is written as follows:

```
Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Windows-
Subsystem-Linux
```

Bold: Indicates a new term, an important word, or words that you see on screen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "In the meantime, systems started to get distributed, and applications started to get shipped in containers, orchestrated by other system software, such as **Kubernetes**."

Warnings or important notes appear like this.

Tips and tricks appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, mention the book title in the subject of your message and email us at `customercare@packtpub.com`.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packt.com/submit-errata`, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packt.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit `packt.com`.