Arbeitsblatt #4

Einführung in die Programmierung mit *Go*

11. März 2015

1 Nebenläufige Ausführung

Schreiben Sie ein Programm mit zwei Goroutinen, das folgende Schritte ausführt:

a. Das Programm alloziert ein Array mit 10000 `int`-Werten.

b. Eine Goroutine schreitet von vorne nach hinten durch das Array und schreibt die Zahl 1.

c. Eine Goroutine schreitet von hinten nach vorne durch das Array und schreibt die Zahl 2.

d. Sobald die beiden Goroutinen gestartet sind, gibt das Programm das Array aus.

e. Überprüfen Sie die Ausgabe, ohne `GOMAXPROCS` zu setzen, oder mit `GOMAXPROCS` auf 1 gesetzt.

f. Setzen Sie `GOMAXPROCS` auf 4 und testen Sie nun die Ausgabe.

2 Kanäle 1

- Schreiben Sie eine Funktion `multiplex`, die als Multiplexer von zwei `int`-Eingabekanälen c_1, c_2 funktioniert: Die Funktion liefert einen `int`-Eingabekanal zurück, der das nächste Ereignis aus c_1 bzw. c_2 liest, je nachdem, welcher Kanal zuerst ein Ereignis zur Verfügung stellt.
- Testen Sie `multiplex` mit einem Kanal, der nur Einsen produziert, und einem Kanal, der nur Nullen produziert (realisiert durch Goroutinen). Entspricht die Ausgabe Ihren Erwartungen?

3 Kanäle 2

- Schreiben Sie eine Funktion `filter(c <-chan int, p func(int)bool) <-chan int`, die einen Kanal zurückliefert, der alle Werte von `c` weiterreicht, wenn diese das Prädikat `p` erfüllen.
- Testen Sie Ihre Funktion mit einem ungebufferten Kanal `c`, in den Sie mehrere Zahlen schreiben, bevor Sie den Kanal schließen. Wählen Sie ein Prädikat `p`, das einige, aber nicht alle der geschriebenen Zahlen passieren läßt.

4 Grenzfälle von Kanälen

a. Was passiert, wenn Sie einen Kanal mehrfach schließen?

b. Was passiert, wenn Sie **`nil`** als Kanal interpretieren und zu senden versuchen?

c. Was passiert, wenn Sie **`nil`** als Kanal interpretieren und zu empfangen versuchen?

5 Webserver

Go macht es Ihnen einfach, einen Webserver zu implementieren.

- Schreiben Sie ein Programm, das das Paket "net/http" importiert.
- Verwenden Sie die Funktion `http.HandleFunc`[1], um Anfragen an den Webserver zu empfangen.
- `HandleFunc` erwartet eine Funktion als Parameter, die wiederum einen `http.ResponseWriter` als Parameter nimmt. Dieser `ResponseWriter` genügt dem Interface `Writer`; dieses Interface wird auch z.B. von Dateien auf dem Festspeicher verwendet. `fmt` bietet spezielle Operationen zum Schreiben in `Writer`:
  - `fmt.Fprint(w, ...)`
  - `fmt.Fprintln(w, ...)`
  - `fmt.Fprintf(w, fmt, ...)`

  Diese Funktionen sind analog zu `fmt.Print`, `fmt.Println`, und `fmt.Printf`, nehmen aber mit `w` einen `Writer` als zusätzlichen Parameter.

  Verwenden Sie diese Funktionen, um dem Webserver Text oder HTML zu geben, den er an anfragende Webbrowser senden soll.
- Verwenden Sie danach die Funktion `http.ListenAndServe(":8080", nil)` um dem Webserver zu sagen, daß er auf Port `8080` horchen soll (also auf der URL `http://localhost:8080`). Beachten Sie, daß diese Funktion einen Fehlercode zurückliefern kann; wenn die Funktion erfolgreich ist, kehrt sie allerdings nicht mehr zurück.
- Prüfen Sie, daß Sie sich mit einem Webbrowser auf Ihren eigenen Server verbinden können.
- Verwenden Sie eine Goroutine, um die vom Webserver gesendete Nachricht ändern zu können, während der Webserver läuft.

  Beispielsweise können Sie kontinuierlich Texteingaben vom Benutzer lesen und diese in die zu sendende Nachricht integrieren. Dazu können Sie mit der folgenden Funktion eine Texteingabe von der Eingabekonsole lesen:

```
func getLine() string {
    reader := bufio.NewReader(os.Stdin)
    text, _ := reader.ReadString('\n')
    return text
}
```

- Ist Ihre Lösung resistent gegen Speicherfehler?

[1] `http://golang.org/pkg/net/http/`