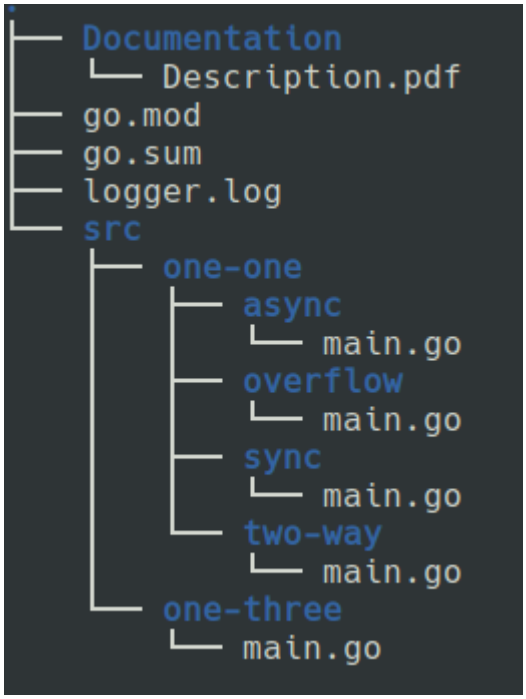
تمرین اول سیستم‌های توزیع‌شده
دانشور امراللهی (۸۱۰۱۹۷۶۸۵)

هرکدام از بخش‌های یک تمرین با یک bullet point مشخص شده‌اند. در هر بخش تکه کد مهم مربوط به آن به همراه نمونه ورودی/خروجی و توضیحات آن آورده شده است.

در تمامی بخش‌ها از یک struct به نام Broker استفاده شده که دارای یک خصیصه ch می‌باشد که کانالی برای تبادل پیام‌های از جنس رشته می‌باشد:

```
type Broker struct {
      ch chan string
}
```

برای اجرای کد هر بخش کافی است به دایرکتوری مربوطه رفته و دستور go run main.go را اجرا کنید.



برای مثال، برای اجرای بخش یک کلاینت - یک سرور - سنکرون بایستی به دایرکتوری src/one-one/sync مراجعه کرده و دستور go run main.go را اجرا کرد.

- **یک کلاینت - یک سرور - سنکرون**

در این حالت برای ارسال پیام از سرور به کلاینت، از یک چنل با ظرفیت صفر استفاده می‌کنیم.

```
func send(broker *Broker, message string) {
	log.Println("SERVER: sending " + message + " on channel")
	broker.ch <- message //blocks here until server reads
}

func runServer(broker *Broker) {

	for i := 1; i <= 5; i++ {
		message := strconv.Itoa(i)
		send(broker, message)
		time.Sleep(time.Millisecond * 500)
	}
	close(broker.ch)
}

func runClient(broker *Broker) {
	for message := range broker.ch {
		log.Println("SERVER: received " + message + " from channel")
	}
}

func main() {
	var broker Broker
	broker.ch = make(chan string, 0)
	go runServer(&broker)
	runClient(&broker)
}
```

در این بخش یک goroutine جدید برای سرور ساخته می‌شود که به طور مرتب برای کلاینت پیام می‌فرستد. ارسال پیام به دلیل صفر بودن ظرفیت کانال در خط `broker.ch <- message` بلاک می‌شود تا کلاینت این پیام را دریافت کند. بنابراین این نوع ارسال و دریافت پیام synchronous است.

```
daneshvar@daneshvar-ZenBook:~/go/src/github.com/daneshvar-amrollahi/DS-Course/A1/src/one-one/sync$ go run main.go
2022/03/29 22:03:12 SERVER: sending 1 on channel
2022/03/29 22:03:12 CLIENT: received 1 from channel
2022/03/29 22:03:13 SERVER: sending 2 on channel
2022/03/29 22:03:13 CLIENT: received 2 from channel
2022/03/29 22:03:13 SERVER: sending 3 on channel
2022/03/29 22:03:13 CLIENT: received 3 from channel
2022/03/29 22:03:14 SERVER: sending 4 on channel
2022/03/29 22:03:14 CLIENT: received 4 from channel
2022/03/29 22:03:14 SERVER: sending 5 on channel
2022/03/29 22:03:14 CLIENT: received 5 from channel
```

- **یک کلاینت - یک سرور - آسنکرون**

ارسال پیام روی کانال با ظرفیت صفر در زبان Go، عملیاتی blocking می‌باشد. یعنی برنامه متوقف می‌شود تا در بخشی دیگر از برنامه (مثلا یک goroutine دیگر) پیام را بردارد. برای asynchronous سازی عملیات ارسال پیام، کافی است ظرفیت کانال را عددی بزرگ قرار دهیم تا block نشود. در این صورت سرور می‌تواند پیام را ارسال کرده و به ادامه کار خود برسد و منتظر نماند تا کلاینت پیام ارسال‌شده را بردارد.

کد برنامه مشابه بخش سنکرون است و صرفا ظرفیت کانال که در main تعیین می‌شود متفاوت است:

```
var broker Broker
broker.ch = make(chan string, 1024)
```

```
2022/03/31 16:39:43 SERVER: sending 43 on channel
2022/03/31 16:39:43 CLIENT: received 29 from channel
2022/03/31 16:39:43 SERVER: sending 44 on channel
2022/03/31 16:39:43 CLIENT: received 30 from channel
2022/03/31 16:39:43 CLIENT: received 31 from channel
2022/03/31 16:39:43 CLIENT: received 32 from channel
2022/03/31 16:39:43 CLIENT: received 33 from channel
2022/03/31 16:39:43 CLIENT: received 34 from channel
2022/03/31 16:39:43 CLIENT: received 35 from channel
2022/03/31 16:39:43 SERVER: sending 45 on channel
2022/03/31 16:39:43 CLIENT: received 36 from channel
2022/03/31 16:39:43 CLIENT: received 37 from channel
2022/03/31 16:39:43 SERVER: sending 46 on channel
2022/03/31 16:39:43 CLIENT: received 38 from channel
2022/03/31 16:39:43 CLIENT: received 39 from channel
2022/03/31 16:39:43 CLIENT: received 40 from channel
```

همان‌طور که در تصویر بالا مشاهده می‌شود، سرور منتظر برداشت پیام از طرف کلاینت نمانده و به گذاشتن پیام هر لحظه توانسته ادامه داده است. کلاینت هم هرموقع توانسته پیام‌ها را برداشته است.
با هر بار اجرای بالا، خروجی‌های متفاوتی مشاهده می‌شود که دلیل آن non-deterministic بودن scheduler است که نمی‌دانیم به چه مدتی و در چه زمانی cpu را در اختیار کدام goroutine (کلاینت یا سرور) قرار می‌دهد.

- **یک کلاینت - یک سرور - مدیریت** overflow

در این بخش ظرفیت چنل را ۳ فرض کردیم. به type Broker یک مشخصه به اسم sz اضافه کرده‌ایم که تعداد پیام‌های داخل بافر را نشان می‌دهد.

```
const BUFFER_SIZE = 3
type Broker struct {
      ch chan string
      sz int
}
...
func main() {
      var broker Broker
      broker.ch = make(chan string, BUFFER_SIZE)
      ...
}
```

تابع send به صورت زیر تغییر می‌کند:

```
func send(broker *Broker, message string) {
      log.Println("SERVER: sending message on channel  " +
strconv.Itoa(broker.sz))
      if broker.sz < BUFFER_SIZE {
            broker.sz += 1
            broker.ch <- message
            log.Println("SERVER: sent message on channel     " +
strconv.Itoa(broker.sz))

      } else {
            log.Println("BUFFER OVERFLOW! CANNOT SEND NEW MESSAGE FOR NOW")
      }
}
```

در صورتی که کانال ظرفیت پیام جدید نداشته باشد، پیام OVERFLOW چاپ می‌شود. در غیر اینصورت مشخصه sz یک واحد افزایش پیدا می‌کند (زیرا یک پیام جدید در بافر قرار می‌گیرد).

همچنین تابع recv به شکل زیر تغییر می‌کند:

```
func recv(broker *Broker) string {
	if broker.sz > 0 {
		broker.sz -= 1
		return <-broker.ch
	} else {
		log.Fatal("BUFFER UNDERFLOW!")
		return ""
	}
}
```

هنگام برداشت یک پیام، از مشخصه sz یک واحد کم می‌شود. زیرا یک پیام از بافر برداشته شده است.

```
2022/03/31 16:53:23 SERVER: sending message on channel  0
2022/03/31 16:53:23 SERVER: sent message on channel     1
2022/03/31 16:53:23 SERVER: sending message on channel  1
2022/03/31 16:53:23 SERVER: sent message on channel     2
2022/03/31 16:53:23 SERVER: sending message on channel  2
2022/03/31 16:53:23 SERVER: sent message on channel     3
2022/03/31 16:53:23 CLIENT: received message from channel
2022/03/31 16:53:23 SERVER: sending message on channel  2
2022/03/31 16:53:23 SERVER: sent message on channel     3
2022/03/31 16:53:23 SERVER: sending message on channel  3
2022/03/31 16:53:23 BUFFER OVERFLOW! CANNOT SEND NEW MESSAGE FOR NOW
```

در مثال بالا، انتهای هر خط تعداد پیام‌های داخل بافر در آن لحظه نشان داده شده است. سرور هنگام ارسال پیغام چهارم با پیام overflow مواجه می‌شود.

- **یک کلاینت - یک سرور - ارسال پیام دوطرفه**

در این حالت، هر دوی کلاینت و سرور روی یک کانال هم می‌نویسند هم می‌خوانند. کلاینت با دریافت هر پیام سرور (پیام‌های سرور با ارقام ۰ تا ۹ آغاز می‌شوند)، یک سیگنال acknowledge رو کانال برای سرور می‌نویسد که سرور دوباره آن را در خط `message =` `<-broker.ch` می‌خواند.

```
func runServer(broker *Broker) {
	for i := 1; i <= 5; i++ {
		var message string
		message = strconv.Itoa(i)
		log.Println("SERVER: sending " + message + " on channel")
		broker.ch <- message
		time.Sleep(time.Second * 2)
		message = <-broker.ch
		log.Println("SERVER: read \"" + message + "\" from channel")
	}
	close(broker.ch)
}
func runClient(broker *Broker) {
	for message := range broker.ch {
		if message[0] >= '0' && message[0] <= '9' { //message is from
server
			log.Println("CLIENT: sending SERVER ACK " + message + "
on channel")
			broker.ch <- "CLIENT ACK " + message
		}
	}
}
```

```
2022/03/31 17:02:04 SERVER: sending 1 on channel
2022/03/31 17:02:04 CLIENT: sending SERVER ACK 1 on channel
2022/03/31 17:02:06 SERVER: read "CLIENT ACK 1" from channel
2022/03/31 17:02:06 SERVER: sending 2 on channel
2022/03/31 17:02:06 CLIENT: sending SERVER ACK 2 on channel
2022/03/31 17:02:08 SERVER: read "CLIENT ACK 2" from channel
2022/03/31 17:02:08 SERVER: sending 3 on channel
2022/03/31 17:02:08 CLIENT: sending SERVER ACK 3 on channel
2022/03/31 17:02:10 SERVER: read "CLIENT ACK 3" from channel
2022/03/31 17:02:10 SERVER: sending 4 on channel
2022/03/31 17:02:10 CLIENT: sending SERVER ACK 4 on channel
2022/03/31 17:02:12 SERVER: read "CLIENT ACK 4" from channel
2022/03/31 17:02:12 SERVER: sending 5 on channel
2022/03/31 17:02:12 CLIENT: sending SERVER ACK 5 on channel
2022/03/31 17:02:14 SERVER: read "CLIENT ACK 5" from channel
```

- **یک سرور - سه کلاینت**

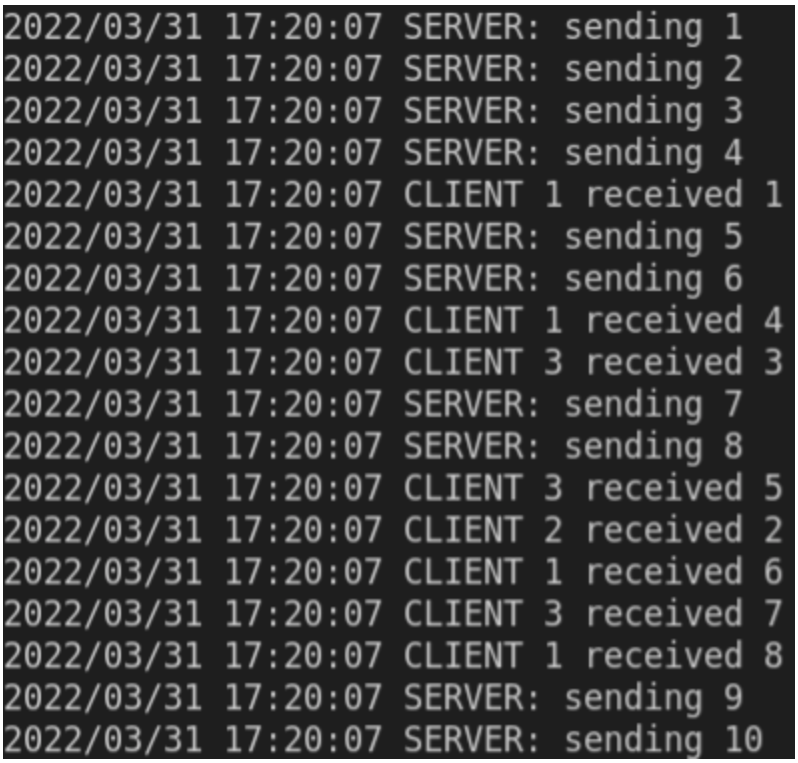
```
func main() {
	var broker Broker
	broker.ch = make(chan string, 1024)

	go runClient(&broker, 1)
	go runClient(&broker, 2)
	go runClient(&broker, 3)

	for i := 1; i <= 100; i++ {
		send(&broker, strconv.Itoa(i))
		if i%4 == 0 {
			time.Sleep(time.Microsecond)
		}
	}
	time.Sleep(time.Second * 2)
	close(broker.ch)
}

func runClient(broker *Broker, id int) {
	for message := range broker.ch {
		if len(message) > 0 {
			log.Println("CLIENT " + strconv.Itoa(id) + " received " +
message)
			time.Sleep(time.Microsecond)
		}
	}
}
```

در این حالت ۳ کلاینت در ۳ تا goroutine متفاوت به صورت موازی پیام‌های ارسال‌شده در کانال را دریافت می‌کنند.



در این حالت نیز با non-determinism مواجه هستیم زیرا از عملکرد دقیق scheduler باخبر نیستیم. به هر کلاینت که نوبت برسد، از بافر پیام برمی‌دارد و لزوما ترتیب خاصی ندارند. با اجراهای مختلف می‌توان خروجی‌های مختلف گرفت.