

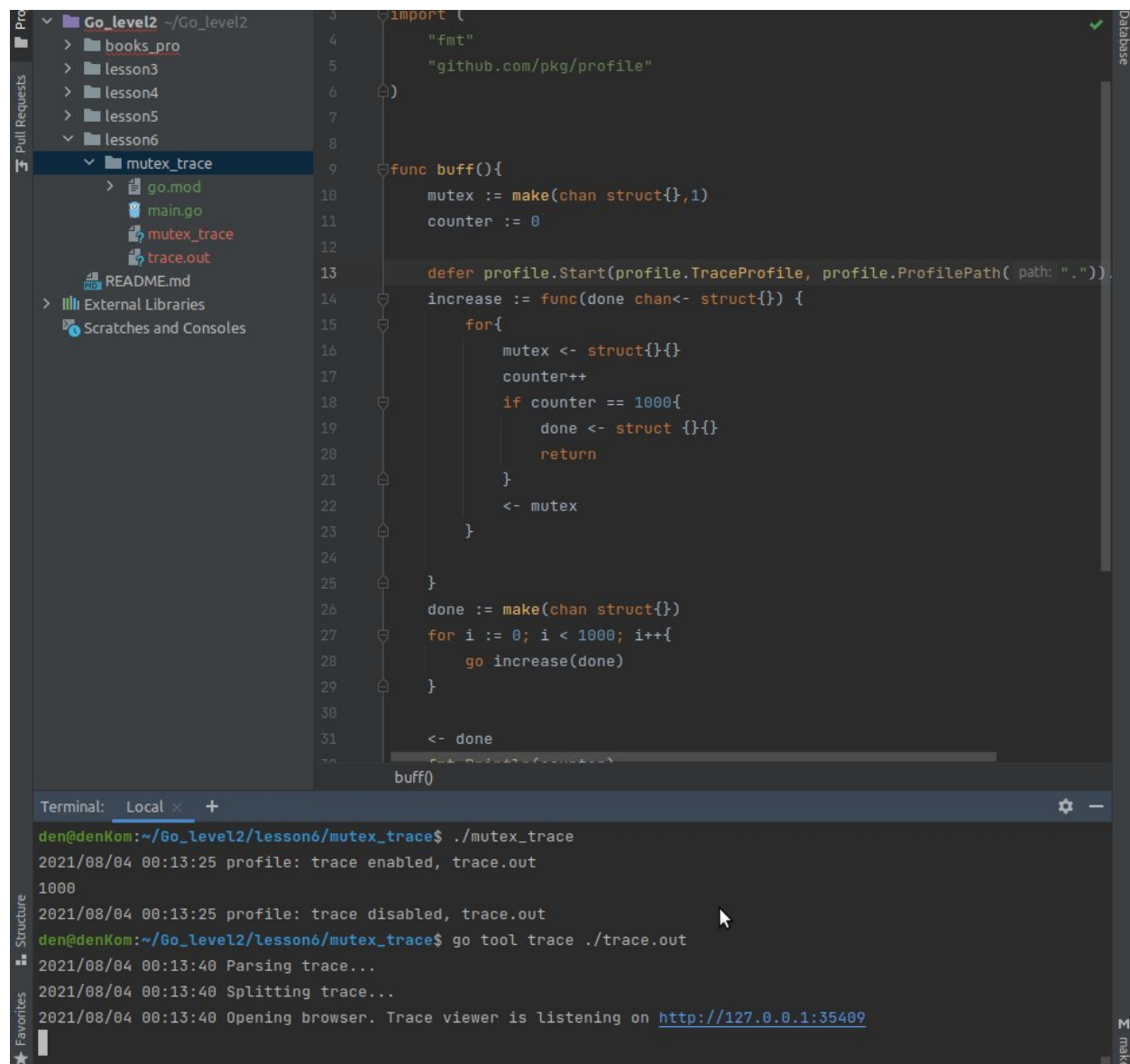
1. Написать программу, которая использует мьютекс для безопасного доступа к данным из нескольких потоков. Выполните трассировку программы
2. Написать многопоточную программу, в которой будет использоваться явный вызов планировщика. Выполните трассировку программы
3. Смоделировать ситуацию “гонки”, и проверить программу на наличии “гонки”

1. Написать программу, которая использует мьютекс для безопасного доступа к данным из нескольких потоков. Выполните трассировку программы

Пишем многопоточный код. Для трассировки добавляем «`defer profile.Start(profile.TraceProfile, profile.ProfilePath(".")).Stop()`»

Далее создаем файл `trace.out` и открываем его в браузере с помощью команды

«`go tool trace ./trace.out`» прописанной в командной строке

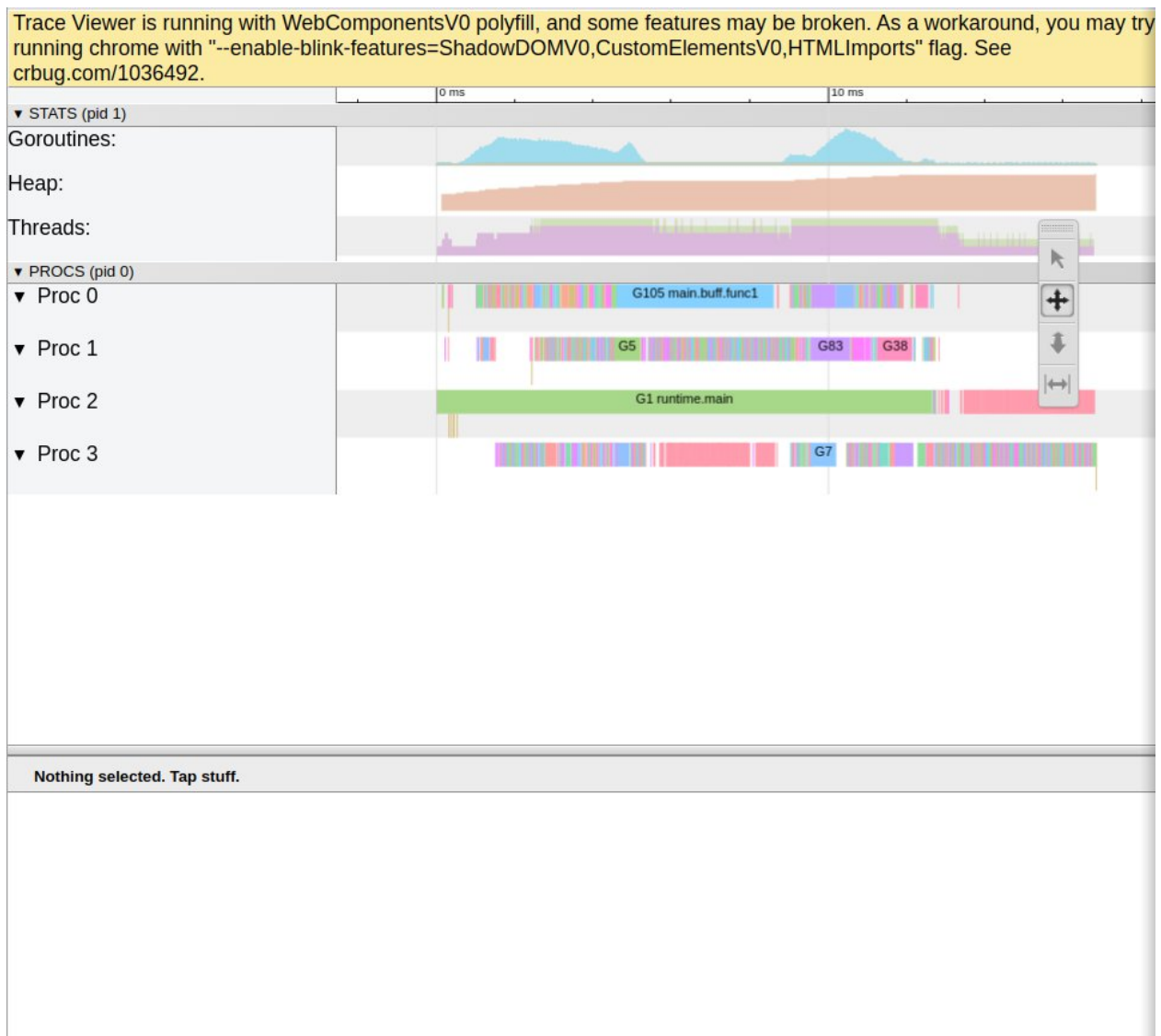


The screenshot shows an IDE with a Go project structure on the left. The project is named `Go_level2` and contains several subdirectories: `books_pro`, `lesson3`, `lesson4`, `lesson5`, `lesson6`, and `mutex_trace`. The `mutex_trace` directory contains `go.mod`, `main.go`, `mutex_trace`, and `trace.out`. The `main.go` file is open in the editor, showing the following code:

```
1 import (
2     "fmt"
3     "github.com/pkg/profile"
4 )
5
6 func buff(){
7     mutex := make(chan struct{},1)
8     counter := 0
9
10    defer profile.Start(profile.TraceProfile, profile.ProfilePath(".")).Stop()
11
12    increase := func(done chan<- struct{}) {
13        for{
14            mutex <- struct{}{}
15            counter++
16            if counter == 1000{
17                done <- struct {}{}
18                return
19            }
20        }
21        <- mutex
22    }
23
24    done := make(chan struct{})
25    for i := 0; i < 1000; i++){
26        go increase(done)
27    }
28
29    <- done
30
31    buff()
32 }
```

The terminal at the bottom shows the execution of the program:

```
den@denKom:~/Go_level2/lesson6/mutex_trace$ ./mutex_trace
2021/08/04 00:13:25 profile: trace enabled, trace.out
1000
2021/08/04 00:13:25 profile: trace disabled, trace.out
den@denKom:~/Go_level2/lesson6/mutex_trace$ go tool trace ./trace.out
2021/08/04 00:13:40 Parsing trace...
2021/08/04 00:13:40 Splitting trace...
2021/08/04 00:13:40 Opening browser. Trace viewer is listening on http://127.0.0.1:35409
```



2. Написать многопоточную программу, в которой будет использоваться явный вызов планировщика. Выполните трассировку программы

Пишем код добавляем явный вызов планировщика, добавляя в код `runtime.Gosched()`

```
defer profile.Start(profile.TraceProfile, profile.ProfilePath(path: ".")).Stop()
increase := func(done chan<- struct{}) {
    for{
        mutex <- struct{}{}
        counter++
        if counter == 1000{
            done <- struct {}{}
            return
        }
        <- mutex
    }
}

done := make(chan struct{})
for i := 0; i < 1000; i++){
    go increase(done)
    if i % 100 == 0{ // на каждом 100м элементе
        // попросить планировщик прекратить выполнение потока и проверить,
        // нет ли других потоков в состоянии готовности
        runtime.Gosched()
    }
}

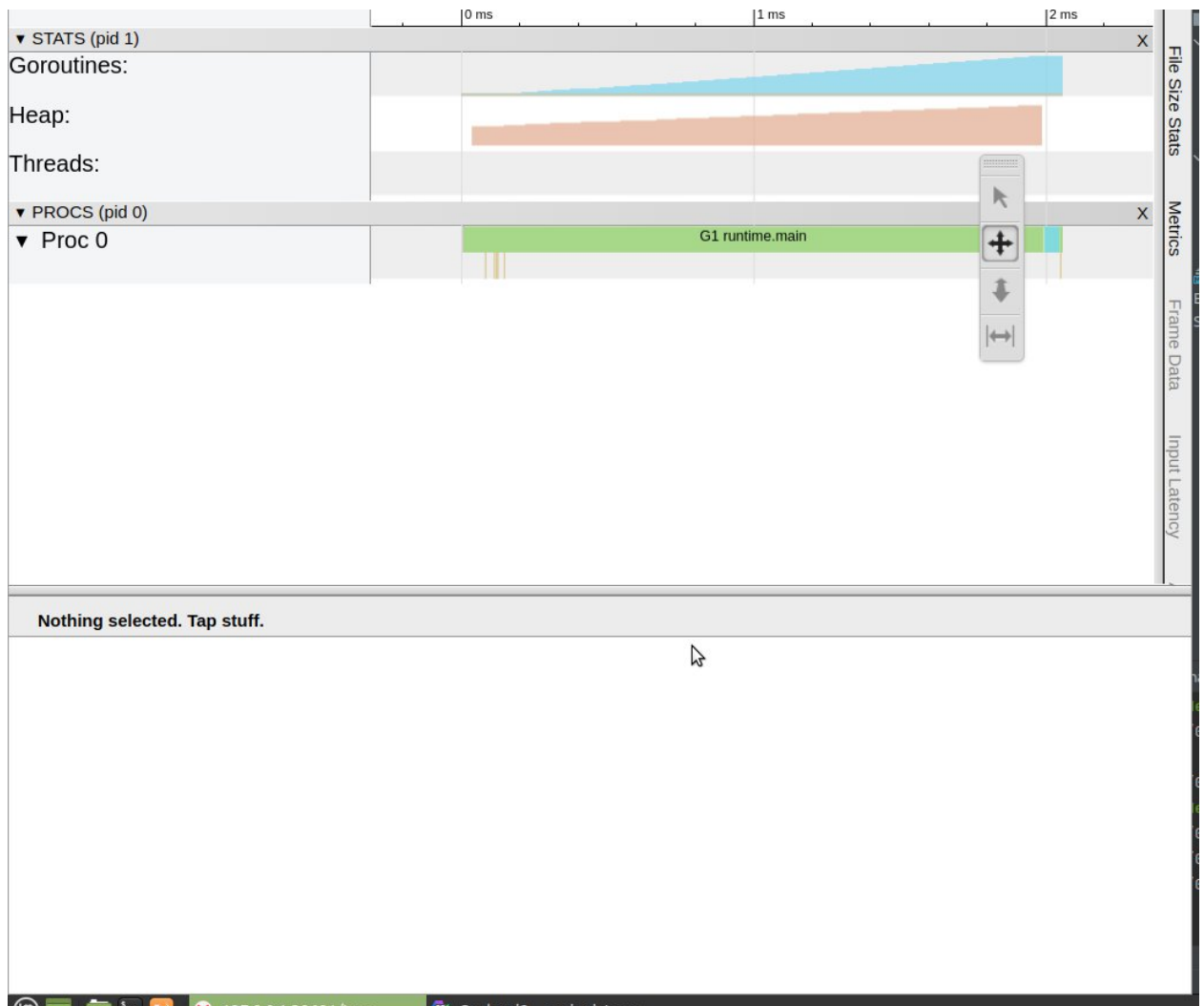
<- done
fmt.Println(counter)

func main() {
```

запускаем код командой `GOMAXPROCS=1 go run main.go`.

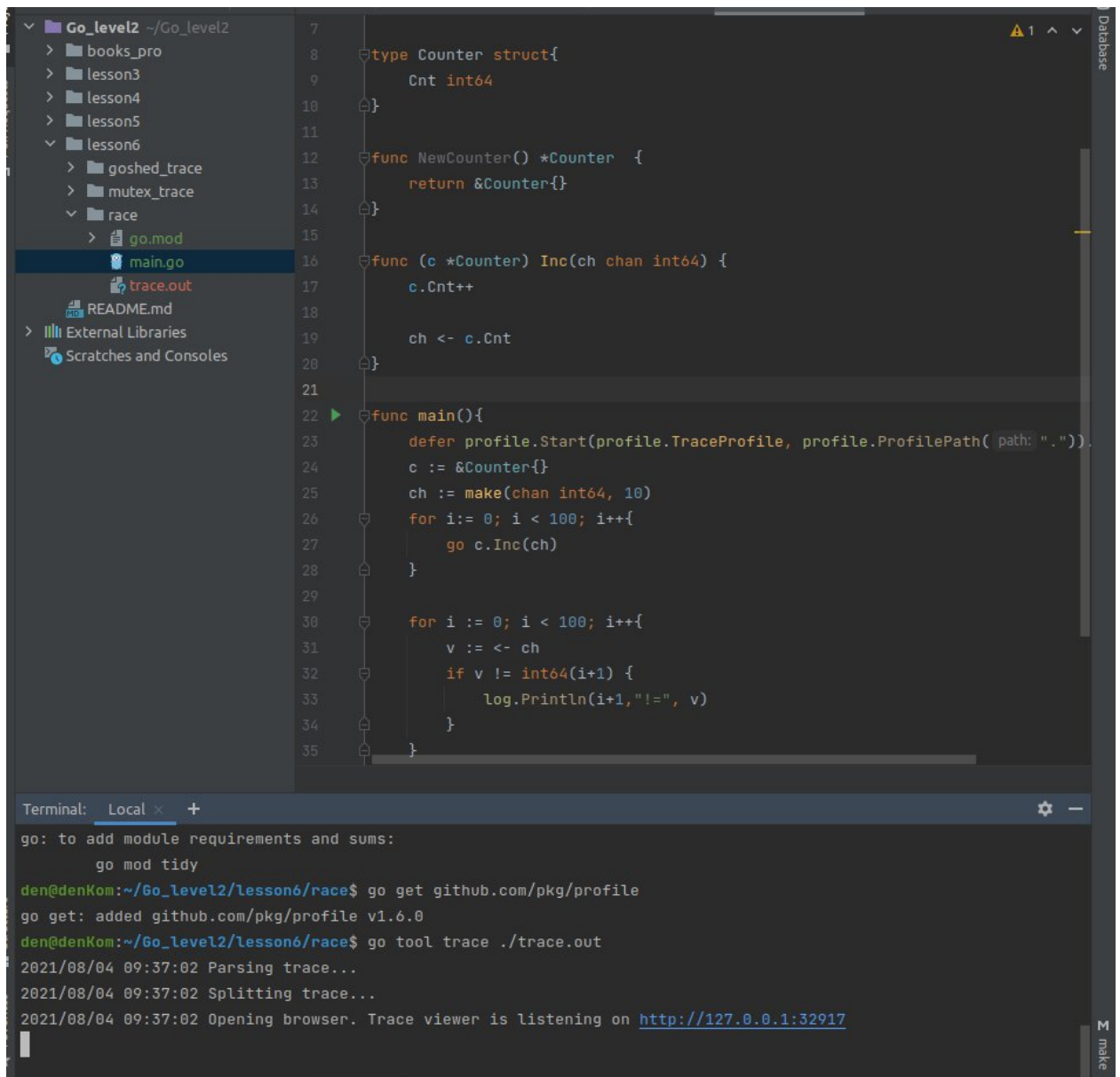
Далее создаем файл `trace.out` и открываем его в браузере с помощью команды

« `go tool trace ./trace.out` » прописанной в командной строке.



3. Смоделировать ситуацию “гонки”, и проверить программу на наличии “гонки”

моделируем ситуацию гонки



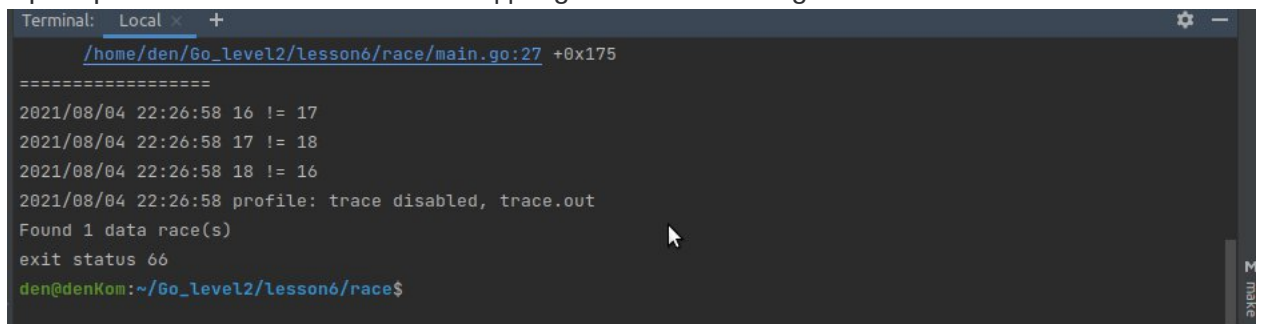
The screenshot shows an IDE with a Go project structure on the left. The main editor displays the code for `main.go` in the `Go_level2` directory. The code defines a `Counter` struct, a `NewCounter` function, an `Inc` function, and a `main` function. The `main` function uses a `profile` package to start a trace and prints the value of `ch` for each iteration of a loop. The terminal window at the bottom shows the execution of `go mod tidy`, `go get github.com/pkg/profile`, and `go tool trace ./trace.out`. The output of the trace tool is displayed, showing the parsing and splitting of the trace data.

```
7
8 type Counter struct{
9     Cnt int64
10 }
11
12 func NewCounter() *Counter {
13     return &Counter{}
14 }
15
16 func (c *Counter) Inc(ch chan int64) {
17     c.Cnt++
18
19     ch <- c.Cnt
20 }
21
22 func main(){
23     defer profile.Start(profile.TraceProfile, profile.ProfilePath(path: ".")).
24     c := &Counter{}
25     ch := make(chan int64, 10)
26     for i:= 0; i < 100; i++){
27         go c.Inc(ch)
28     }
29
30     for i := 0; i < 100; i++){
31         v := <- ch
32         if v != int64(i+1) {
33             log.Println(i+1,"!=", v)
34         }
35     }
36 }
```

Terminal: Local x +

```
go: to add module requirements and sums:
go mod tidy
den@denKom:~/Go_level2/lesson6/race$ go get github.com/pkg/profile
go get: added github.com/pkg/profile v1.6.0
den@denKom:~/Go_level2/lesson6/race$ go tool trace ./trace.out
2021/08/04 09:37:02 Parsing trace...
2021/08/04 09:37:02 Splitting trace...
2021/08/04 09:37:02 Opening browser. Trace viewer is listening on http://127.0.0.1:32917
```

Проверяем на наличие гонки командой `go run -race main.go`



The screenshot shows a terminal window with the output of the `go run -race main.go` command. The output indicates that a data race was found between two goroutines accessing the `ch` channel. The race was detected at line 27 of `main.go`. The terminal also shows the exit status of the program.

```
Terminal: Local x +
/home/den/Go_level2/lesson6/race/main.go:27 +0x175
=====
2021/08/04 22:26:58 16 != 17
2021/08/04 22:26:58 17 != 18
2021/08/04 22:26:58 18 != 16
2021/08/04 22:26:58 profile: trace disabled, trace.out
Found 1 data race(s)
exit status 66
den@denKom:~/Go_level2/lesson6/race$
```