Multithreading (višedretvenost) u Gou

- **Kompleksna tema!**
- Za ozbiljne detalje i specifičnosti bilo bi potrebno odraditi ozbiljan dvosemestralni kolegij
- Ukratko: operativni sustav raspolaže s procesima. Dretva je oblik asinkronog djelovanja unutar tog procesa.
- Primarno, procesi međusobno ne dijele zajednički memorijski prostor
- Da bi dva procesa komunicirala, koriste se razne IPC metode, najpopularniju (i vjerojatno najprimitivnija) koju ste vjerojatno koristili je pipe („|”)
- Važno je razumjeti da višeprocesorski (ili višejezgreni) sustavi nisu nužan i dovoljan uvjet za višedretvenost
- Višedretvenost je svojstvo operacijskog sustava!

Multithreading (višedretvenost) u Gou

- Striktno govoreći, svaki operativni sustav koji podržava hardverske IRQ-ove sposoban je odraditi neki oblik context switchinga
- Primjerice, ako ste morali napisati nekakav kod u DOS-u, koji je striktno single-tasking operativni sustav, bilo je potrebno odraditi ovakav poziv (prototip):

```
void SwapContext(thread_t a, thread_t b)
{
    // Spremi stanje dretve A
    // Vrati stanje dretve B
}
```

- Dretva A pozvala bi ovu funkciju, ali bi se dretva B vratila iz nje.
- Razumno je da je ovakav pristup izrazito kompleksan i zahtijeva veliko znanje i pedantnost te pristup assemblyju

Multithreading (višedretvenost) u Gou

- Srećom, većina modernih (onih ispod 25 godina starosti) nativno podržava višedretvenost
- Unatoč tome, upravljanje dretvama **nije trivijalno**, a na prvi pogled nije niti intuitivno
- Primjerice, trenutačni WinAPI raspolaže s desetak sinkronizacijskih objekata (event, mutex, semaphor, Critical Section, SlimRW lock, uvjetne varijable…) koji nisu međusobno isključivi
- **Go nudi prekrasnu jednostavnos i brzinu izvođenja višedretvenih aplikacija što predstavlja prvu veliku objektivnu vrijednost rada u njemu, nudeći minimalnu kompleksnost koda za maksimalne performanse!**

Multithreading (višedretvenost) u Gou

- Dobrodošli u **goroutine**
- Gorutine (bit će kroatizirane u svakom sljedećem tekstu) su „lagane”
- To znači da možete imati više gorutina nego „pravih” treadova koje podržava vaš operativni sustav
- Gorutine imaju kraće vrijeme pokretanja od standardnih threadova
- Gorutine dolaze s ugrađenim primitivima za međusobnu komunikaciju
- (loš) primjer (i prvi problem):
  https://play.golang.org/p/pQV7cDAJusJ
  VAŽNO: Pokrenite ovaj i sve buduće primjere iz ovog poglavlja u Vašem lokalnom okruženju!

Multithreading (višedretvenost) u Gou

- Go, kao jezik orijentiran na višedretvenost, nudi i toolchain support za detekciju istih
- „Data race” problemi spadaju u najkompleksnije slučajeve debuginga
- Go data race detector se smatra jednim od najkvalitetnijih u branši
- Poslužio je Go developerima da nađu data race probleme u samim Go paketima
- Poziva se dodavanjem *–race* zastavice u go test/run/build/install alatima
- Trenutno radi na */amd64, linux/ppc64le i linux/arm64 platformama i nosi performansne penale
- Primjer: https://play.golang.org/p/7IcTJio7FA6

Multithreading (višedretvenost) u Gou

- Da bi smo učinkovito radili sa asinkronim (višedretvenim) softverom, potreban nam je neki oblik sinkronizacije
- Go nudi odličan koncept kanala (channel) za potrebe dijeljenja podataka između gorutina, kao i mogućnost korištenja mutexa ukoliko podatke nije potrebno dijeliti, kao i ostale primitive za sinkronizaciju
- Za potpuno razumijevanje (bilo kojeg) višedretvenog softvera, potrebno je razmišljati izvan okvira klasičnog proceduralnog programiranja
- Shodno tome, bitno je razumijevanje kompleksnosti višedretvenosti i sigurnost u željeni konačni rezultat
- Ne zaboravite: računala uvijek rade ono što im kažete, ali rijetko kada ono što zapravo želite

Multithreading (višedretvenost) u Gou - WaitGroup

- **WaitGroup** tip objekta je najsličniji semaforu (semaphore)
- Semafori su signalni mehanizmi
- Koristeći **WaitGroup**, možemo odrediti programski tok i osigurati da sve gorutine budu izvršene prije nastavka neke druge operacije
- Pojednostavljeno, **WaitGroup** je brojač
- WaitGroup živi u paketu „**sync**”
- Sintaksa deklaracije **WaitGroup** objekta
  `var myWaitGroup1 sync.WaitGroup`
- Kako bi **WaitGroup** ispravno funkcionirao, prije svega ga moramo inicijalizirati na neki (pozitivni) broj ***n***
  `myWaitGroup1.Add(n)`
- Završetkom svake gorutine, smanjujemo ***n*** pozivom
  `myWaitGroup1.Done()`

Multithreading (višedretvenost) u Gou - WaitGroup

- Na mjestu gdje čekamo završetak svih gorutina koje smo pokrenuli, pozivamo
  `myWaitGroup1.Wait()`
- https://play.golang.org/p/l0gqe-NXv6o

Multithreading (višedretvenost) u Gou - mutex

- https://play.golang.org/p/Bg_MyCi_RlH
- Mehanizam za zaključavanje resursa
- Služi za sinkronizaciju pristupa nekom resursu blokirajući pristup nakon „zaključavanja" pa sve do otključavanja
- Mutexi, kao i WaitGroup, žive u „**sync**" paketu
- Mutex deklariramo sljedećom sintaksom
  `var mux sync.Mutex`
- Mutexte zaključavamo, odnosno otključavamo na sljedeći način
  `mux.Lock()` `mux.Unlock()`
- https://play.golang.org/p/ac2ZSdLxEmF
- Nemojte se zaigrati s defer naredbom!
  https://play.golang.org/p/DeDMaZxW753

Multithreading (višedretvenost) u Gou - Channel

- Go nudi gotovo trivijalan (naspram drugih programskih jezika) način za komunikaciju između gorutina – channel (kanal)
- Nekoliko važnih informacija i pretpostavki
  - Ne zaboravite da je i main() gorutina
  - Pokušajte koristiti kanale za ono za što su namijenjeni; Go raspolaže s različitim mehanizmima sinkronizacije koji imaju svoju primjenu
  - Kanali su „blocking by default”
  - Ako se nešto u kanal šalje, očekuje se da postoji gorutina koja taj podatak čeka-prima. U suprotnom, problem – **deadlock**
- https://play.golang.org/p/u4nlPEAwBPe

Multithreading (višedretvenost) u Gou - Channel

- Baš kao i mape, kanali se moraju kreirati prije korištenja

```
var ch chan int
ch = make(chan int)
//ch := make(chan int)
```

- Jednostavna sintaksa za pridruživanje

```
data := <- ch      // čita s kanala ch
ch <- data         // piše u kanal ch
```

- Kanali mogu imati **buffer**

```
ch := make(chan int, 100)
```

- Slanje podataka u buffered kanal biti će blokirajuće samo kada je kanal pun
- Dohvat podataka iz buffered kanala blokirat će samo kada je kanal prazan
- Nema on the fly promjene veličine kanala

Multithreading (višedretvenost) u Gou - Channel

- Primjeri blokiranja kanala:
  https://play.golang.org/p/ZY_lVXdHnGV
  https://play.golang.org/p/POM9OKYDZd4
  https://play.golang.org/p/FxUtbMD-7Dp
- Po kanalu možemo iterirati ne bi smo li pronašli njegove vrijednosti
- Kanal se može zatvoriti
- Kanali (buffered), kao i sliceovi, imaju ***len()*** i ***cap()*** za određivanje trenutnog broja „nepročitanih" poruka u kanalu, odnosno ukupnog kapaciteta kanala
- Zatvoreni kanal više ne može primati vrijednosti (panic); zatvoreni kanal ne može se ponovno zatvoriti (panic)

Multithreading (višedretvenost) u Gou - Channel

- S čitanjem zatvorenih kanala stvar je malo drugačija; čitanje sa zatvorenog i praznog kanala neće blokirati, a povratna vrijednost biti će nulvrijednost tipa podatka kanala
- https://play.golang.org/p/8Qx2q4GKWHF
- ***range***, kojeg smo koristili prilikom iteriranja mapa, sliceova i polja, možemo koristiti i kod kanala, ali je nužno da kanal bude zatvoren!
- Paradigma višedretvenosti u Gou putem kanala omogućuje trivijalnu manipulaciju programskog toka putem kanala
- Sintaksa je klasični ***select*** https://play.golang.org/p/ihah2Uo9V0S

Multithreading (višedretvenost) u Gou - Once

- Izvršava se na prvi poziv funkcije
- Jednakovrijedno tokom runtimea bez obzira radi li se o gorutini ili pozivu iz maina
- Očekuje funkciju kao parametar
- Korisno kada je potrebno napraviti neku inicijalizaciju koju više nema potrebe pozivati tokom izvršenja programa
- Deklaracija
  var myOnceCall sync.Once
- Poziv
  myOnceCall.Do(imeFunkcije)
- https://play.golang.org/p/zNnv9K_SHwF

Multithreading (višedretvenost) u Gou - Atomic

- Generalno, atomarne operacije su one operacije koje se izvršavaju neovisno u paralelizacijskim procesima
- Iako operativno dijele ideju s mutexima, ne treba ih miješati ili neispravno koristiti
- Principijelno se koriste kada su potrebne performanse, s obzirom da su mutexi ipak „skupi” (ili barem skuplji od atomarnih operacija)
- Nemojte smetnuti s uma da atomičnost pisanja nije ni na koji način vezana s atomičnošću čitanja varijable
- https://play.golang.org/p/66glQ837ZmE

Multithreading (višedretvenost) u Gou

- Dosadašnji kratki sažetak:
  - Mutexi i semafori (WaitGroup) generalno serijaliziraju pristup
  - Kanali rade obrnuto – orkestriraju više paralelnih zadaća
  - Go idiomatski pristup paralelnom programiranju – ne komunicirajte dijeleći memoriju, već dijelite memoriju komunicirajući
  - Atomic operacije koristimo onda kada imamo opravdani razlog i specifičnu namjenu
  - Ideja Go pristupa paralelizaciji je CSP (Communicating sequential processes) od C. A. R. Hoare izadan 1985. godine http://www.usingcsp.com/cspbook.pdf