Primljen: 30.03.2018. Stručni rad

Prihvaćen: 15.05.2018. UDK: 004.438

Konkurentnost u programskom jeziku Go

Concurrency in Go programming language

[1]Krunoslav Husak, [2]Tomislav Adamović, [3]Alan Mutka

[1]Tehnička škola Daruvar, Gundulićeva 14, 43500 Daruvar
[2,3]Veleučilište u Bjelovaru, Trg Eugena Kvaternika 4, 43000 Bjelovar
e-mail: [1]kruno@tsd.hr, [2]tadamovic@vub.hr, [3]amutka@vub.hr

Sažetak: *Go, relativno novi programski jezik koji se razvija unutar Googlea, svakodnevno bilježi sve veću popularnost. Jedan od razloga velike popularnosti je ugrađeni mehanizam za konkurentnost. Zbog današnje široke upotrebe višejezgrenih procesora i mrežno povezanih računala, konkurentnost i paralelizam su postale bitne značajke svakoga programskoga jezika. Za razliku od drugih programskih jezika, gdje je konkurentno programiranje zbog raznih suptilnosti otežano, konkurentnost u Gou je podržana pomoću ugrađenih mehanizama, gorutina i kanala koji se vrlo lako koriste. Gorutine, kao pojednostavljeni model dretvi, i kanali koji omogućuju komunikaciju i sinkronizaciju između gorutina, opisani su u ovom radu, uz odgovarajući primjer programskog koda koji ih koristi.*

Ključne riječi: *go, konkurentnost, gorutine*

Abstract: *Go, relatively new programming language which is developing within Google, is getting more popular every day. One of the reasons for this big popularity is the embedded mechanism for concurrency. Due to the today's widespread use of multi-core processors and network-connected computers, concurrency and parallelism have become essential features of every programming language. Unlike other programming languages, where concurrent programming is difficult due to the different subtleties, Go's concurrency is supported by embedded mechanisms, goroutines and channels that are easy to use. Goruoutines, as a simplified model of threads, and channels that allow communication and synchronization*

between goroutines, are described in this paper, with the corresponding example of the programming code that uses them.

Keywords: *go, concurrency, goroutines*

1. Uvod

Go je relativno novi programski jezik koji razvijaju Rob Pike i drugi unutar Googlea [1]. Go je programski jezik opće namjene, dizajniran s namjerom za korištenje u sistemskom programiranju. Veoma je brz, statički pisani (engl. *statically typed*), kompajliran programski jezik koji djeluje kao dinamički pisani (engl. *dynamically typed*), interpretski jezik (engl. *interpreted language*) [2]. Poput Jave ili Pythona, Go sadrži sustav koji se brine da oslobodi memoriju koja više nije potrebna – mehanizam za sakupljanje smeća (engl. *garbage collection*) koji je ugrađen u sustav i automatski se pokreće po potrebi [3].

Go sadrži mehanizme za konkurentnost koji omogućavaju jednostavno pisanje programa koji iskorištavaju maksimum višejezgrenih i mrežno povezanih računala.

2. Konkurentnost i paralelizam

Današnja računala mogu izvršavati nekoliko sljedova naredbi (zadataka) konkurentno. Sljedove naredbi možemo nazivati različitim imenima – dretve, procesi, zadatci itd., ali mnogi principi konkurentnosti se mogu primijeniti na sve navedene. Na računalima s jednojezgrenim procesorima, navedeni sljedovi naredbi se izmjenjuju na način da međusobno dijele kratke dijelove procesorskog vremena. Takvi zadatci se izvršavaju konkurentno.

Pojam konkurentnosti se često poistovjećuje s pojmom paralelizma. Iako su to povezani pojmovi, velika je razlika između konkurentnosti i paralelizma. Dok konkurentnost omogućava izvršavanje više zadataka – svaki zadatak u određenom dijelu vremena, paralelizam omogućava istovremeno izvršavanje više zadataka. Konkurentnost je *briga* oko mnogo zadataka istovremeno, paralelizam je *izvođenje* mnogo zadataka istovremeno. Konkurentnost je moguća na jednojezgrenim računalima, no, za paralelizam je potrebno računalo s višejezgrenim procesorom.

Konkurentno programiranje je u mnogim okruženjima otežano zbog raznih suptilnosti koje otežavaju implementaciju ispravnog pristupa dijeljenim varijablama. Go potiče drugačiji pristup u kojemu se vrijednosti prenose putem kanala (engl. *channels*) i, zapravo, nikada se aktivno ne dijele među zasebnim nitima, odnosno dretvama izvršenja. Samo jedna gorutina (engl. *goroutine*) ima pristup vrijednosti u bilo kojem trenutku. Upravo zato, prema dizajnu,

nije moguća situacija u kojoj se istovremeno pristupa istoj memorijskoj lokaciji. Kako bi potakli ovakav način razmišljanja, Go programeri su to pretočili u svojevrstan slogan: „Ne komunicirajte tako da dijelite memoriju, već dijelite memoriju komunicirajući“ [4].

3. Konkurentnost u Gou

Jezici poput Newsqueaka, Alefa i Limba [5] (u čijoj je izradi također sudjelovao i Rob Pike) imali su veliki utjecaj na način kako će biti izrađena konkurentnost unutar Goa. Navedeni su programski jezici imali zajedničku značajku – bili su izgrađeni pomoću Hoaerovih sekvencijalnih procesa komuniciranja (engl. *Hoare's Communicating Sequential Processes – CSP*). CSP je formalni jezik za pisanje konkurentnih programa [6]. Nakon što je Hoare u svom radu prikazao CSP, u svojoj knjizi o CSP-u je uveo koncept kanala koji omogućavaju komunikaciju između procesa [7]. Kanali u CSP-u su sinkronizirani, što znači da se pošiljatelj i primatelj sinkroniziraju u točki primanja poruke. Na taj način kanali imaju dvojaku svrhu: komunikacije i sinkronizacije.

Sinkronizirani kanali tj. kanali bez međuspremnika (engl. *unbuffered channels*) su zadani kanali u Gou (ako nije zadana veličina međuspremnika). Sinkronizirani kanali dopuštaju slanje samo ako postoji i primatelj koji je spreman primiti poslanu poruku. Kasnije se implementacija dodatno razvila tako da su omogućeni i asinkroni kanali (engl. *buffered channels*).

Go nam omogućava da pišemo konkurentne programe – omogućava korištenje gorutina, i što je još važnije, komunikaciju između njih.

3.1. Gorutine

Gorutine su svoj naziv dobile jer postojeći nazivi – dretva, proces, korutina itd. – nose određene loše konotacije. Gorutine imaju jednostavan model: to su funkcije koje se izvršavaju konkurentno zajedno s ostalim gorutinama u istom adresnom prostoru. One su „lagane“, njihovo stvaranje ne iziskuje puno memorije – dovoljno je 2kB prostora na stogu. Kako raste potreba Gorutina, one zauzimaju ili oslobađaju prostor na stogu [8], [9]. Dretve, za usporedbu, odmah prilikom stvaranja zauzimaju 1MB (500 puta više), zajedno s memorijom koja služi kao zaštićeno mjesto između memorija više dretvi.

Manje zauzeće memorije odmah dolazi do izražaja kod npr. web poslužitelja. Za svaki zahtjev upućen poslužitelju, poslužitelj pokretan Go programskom podrškom može bez poteškoća stvoriti novu gorutinu za stvaranje odgovora. Za usporedbu, Java programski jezik

bi na taj način jako brzo došao do omražene OutOfMemoryError pogreške – Java virtualna mašina više ne bi mogla zauzeti novu slobodnu memoriju ili mehanizam za sakupljanje smeća ne bi mogao osloboditi memoriju koja se više ne koristi. No, ovo nije ograničeno samo na Javu. Svaki programski jezik koji koristi dretve operativnoga sustava bi na ovakav način jako brzo ostao bez slobodne memorije.

Dretve podrazumijevaju skupe troškove stvaranja i uništavanja. Moraju od operativnoga sustava zatražiti resurse te ih osloboditi i vratiti operativnom sustava jednom kada njihovo izvršenje završi. Zaobilazno rješenje ovog problema je održavanje više dretvi (engl. *pool of threads*). Za usporedbu, u Gou, izvršno okruženje (engl. runtime) stvara i uništava gorutine i operacije s njima su prilično „jeftine“. Programski jezik ne podržava ručno upravljanje gorutinama, sve se odvija automatski.

Go izvršno okruženje prilikom pokretanja programa zauzima nekoliko dretvi na kojima se sve gorutine multipleksiraju. U bilo kojem određenom trenutku, svaka će dretva izvršavati jednu gorutinu. Ukoliko ta gorutina počne blokirati, bit će zamijenjena s drugom gorutinom koja će se izvršavati unutra navedene dretve [10]. Budući da su gorutine „lagane“, one neće blokirati dretvu u kojoj su pokrenuti ako su blokirane zbog jednog od navedenih razloga: mrežnog pristupa, spavanja, operacija s kanalima ili blokiranja na primitivima iz *sync* paketa. Bez obzira je li pokrenuto više od tisuću gorutina, one ne predstavljaju gubitak resursa sustava ako je većina njih blokirana. Izvršno će se okruženje pobrinuti za izvršavanje druge gorutine koja nije blokirana. Ukratko, gorutine su jednostavna apstrakcija dretvi. Go programer ne mora brinuti o dretvama, no isto tako operativni sustav nije svjestan postojanja gorutina. Iz perspektive operativnog sustava, Go program će se ponašati poput običnog C programa upravljanoga događajima (engl. *event-driven program*) [11].

Gorutine se pokreću dodavanjem ključne riječi *go* prije poziva funkcije ili anonimne funkcije. Dokumentacija programskog jezika kaže: „Go naredba pokreće izvršavanje funkcije kao neovisnu dretvu, odnosno gorutinu, unutar istog adresnog prostora“ [12]. Drugim riječima, *go* naredba označava asinkroni poziv funkcije koji stvara novu gorutinu i vraća se bez čekanja na završetak izvođenja gorutine. S gledišta programera, gorutine su način određivanja aktivnosti koje se moraju pokretati konkurentno. Hoće li se pokrenuti paralelno, odlučit će operativni sustav [5].

Slika 1. Programski kôd za pokretanje gorutine

```
func main() {
  go spavaj() // pokreni spavanje i nemoj čekati na završetak
  spavaj()    // pokreni spavanje i čekaj dok funkcija ne završi
}

func spavaj() {
  time.Sleep(time.Second * 5)
  fmt.Println("Gotovo spavanje")
}
```

Izvor: autor

Programski će kôd iz slike 1 dvaput pokrenuti funkciju *spavaj()*. Prvi poziv sadrži naredbu *go* što znači da će se funkcija pokrenuti u zasebnoj gorutini, odnosno, konkurentno. Nakon toga, ponovno se pokreće funkcija *spavaj()*, ali ovoga se puta neće pokrenuti u zasebnoj gorutini, već u istoj gorutini u kojoj se pokreće glavni program. Funkcija *spavaj()* blokira pet sekundi spavanjem i nakon toga ispisuje „Gotovo spavanje“. Budući da funkcija blokira spavanjem, a prvi poziv funkcije je pozvan konkurentno, nakon pet sekundi će se dvaput prikazati „Gotovo spavanje“, budući da su funkcije bile pokrenute u zasebnim gorutinama.

Navedeni primjer nije praktično upotrebljiv jer funkcija nema načina da javi kada je završila. Za to su potrebni kanali.

3.2. Kanali

U širem smislu, kanali omogućuju komunikaciju između gorutina. U Gou, kanal pruža mehanizam za sinkronizaciju dvije konkurentno pokrenute funkcije te komunikaciju, razmjenjujući vrijednosti dogovorenoga tipa podatka.

Kanali su objekti prve klase (engl. *first-class objects*). Mogu biti pohranjeni u varijable, prosljeđeni kao argumenti funkcijama, vraćeni iz funkcija, biti poslani preko kanala itd. Budući da su kanali tip podatka, moguće je otkrivanje programerskih pogrešaka poput slanja pokazivača preko kanala koji je namijenjen za cijele brojeve.

Kanali se stvaraju koristeći *make* naredbu. Ukoliko se prilikom stvaranja kanala prosljedi dodatni parametar (cijeli broj), to će podesiti međuspremnik kanala. Zadana vrijednost je nula, što predstavlja stvaranje kanala bez međuspremnika, odnosno sinkronizirajući kanal.

Slika 2. Programski kôd za stvaranje kanala

```
kanal1 := make(chan int)     // kanal za slanje cijelih brojeva
kanal2 := make(chan string)  // kanal za slanje teksta
kanal3 := make(chan int, 10) // kanal s međuspremnikom
```

Izvor: autor

Kanali bez međuspremnika omogućavaju razmjenu vrijednosti (sa sinkronizacijom), garantirajući da su dvije gorutine u poznatom stanju. Za razmjenu vrijednosti koristeći kanale koristi se operator <- koji omogućuje slanje i primanje podataka:

Slika 3. Programski kôd za primanje i slanje podataka koristeći kanal

```
kanal1 <- 5      // slanje vrijednosti putem kanala
broj := <-kanal1 // primanje vrijednosti putem kanala
                 // i pohranjivanje u varijablu broj
```

Izvor: autor

Primjer iz slike 1 može se izmijeniti tako da funkcija koristi kanal kako bi glavnoj funkciji javila kada je završila:

Slika 4. Programski kôd za prikaz korištenja kanala

```
func main() {
  c := make(chan int)
  go spavaj(c)
  <-c     // cekanje na podatke iz kanala
}

func spavaj(c chan int) {
  time.Sleep(time.Second * 5)
  fmt.Println("Gotovo spavanje")
  c <- 1  // slanje podatka putem kanala
}
```

Izvor: autor

U glavnoj se funkciji stvara kanal za razmjenu cijelih brojeva (tip podataka nije bitan u ovom primjeru). Nakon toga se pokreće funkcija *spavaj(),* u novoj gorutini, kojoj se prosljeđuje kanal kako bi funkcija mogla putem istog kanala odgovoriti kada završi s izvođenjem. Glavna funkcija tada čeka na podatak. Funkcija *spavaj(),* jednom kada završi s izvođenjem, putem kanala će poslati informaciju (sama informacija u ovom primjeru nije bitna). Kada se podatak primi u glavnoj funkciji, završava i program.

4. Zaključak

Programski jezik Go svakim danom bilježi sve veću popularnost. Jednostavan programski jezik namijenjen sistemskog programiranju sve češće postaje izbor za izgradnju web poslužitelja, mikroservisa te znanstvenih programa, odnosno, svugdje gdje je konkurentnost itekako potrebna.

Konkurentnost je jedna od važnijih značajki programskog jezika Go. Ugrađeni mehanizmi poput gorutina i kanala omogućavaju jednostavno pisanje konkurentnih programa. Programeri naredbom *go* mogu vrlo jednostavno stvarati nove gorutine. Izvršno okruženje brine se oko upravljanja svim gorutinama te se programer ne mora brinuti o ručnom upravljanju njima, sve se odvija automatski. Upravo zbog svoje jednostavne sintakse i mehanizmima za konkurentnost, Go ima vrlo svijetlu budućnost u računalnom svijetu.

Literatura

[1] Donovan, A.; Kernighan, B. (2015.). The Go Programming Language, Addison-Wesley, str. xi

[2] The Go Programming Language – Documentation. https://golang.org/doc/ (15.03.2018.)

[3] Seguin, K. The Little Go Book http://openmymind.net/The-Little-Go-Book/, str. 17. (15.03.2018.)

[4] Effective Go – Concurrency. https://golang.org/doc/effective_go.html#concurrency (23.03.2018.)

[5] Prell, A.; Rauber, T. (2012). „Go's Concurrency Constructs on the SCC", The 6th *Many-core Applications Research Community (MARC) Symposium*, hal-00718924

[6] Hoare, C. A. R. (1978). „Communicating Sequential Processes", *Communications of the ACM*, vol. 21(8), str. 666.–677.

[7] Hoare, C. A. R. (1985). Communicating Sequential Processes, Upper Saddle River, NJ, Prentice-Hall

[8] Effective Go – Goroutines. https://golang.org/doc/effective_go.html#goroutines (23.03.2018.)

[9] Go 1.4 Release Notes – runtime. https://golang.org/doc/go1.4#runtime

[10] Deshpander, N.; Sponsler, E; Weiss, N. Analysis of the Go Runtime Scheduler. http://www.cs.columbia.edu/~aho/cs6998/reports/12-12-11_DeshpandeSponslerWeiss_GO.pdf (12.03.2018.)

[11] Vyukov, D.; Konecek, M. (2013) Goroutines vs OS threads - Golang Forums. https://groups.google.com/forum/#!topic/golang-nuts/j51G7ieoKh4/ (25.03.2018.)

[12] The Go Programming Language Specification - Version of February 1, 2018. https://golang.org/ref/spec (17.03.2018.)