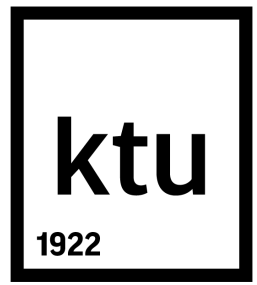
Paskirstyta atmintis. Go kanalai

Karolis Ryselis

Kauno Technologijos Universitetas



Chuck Norris does not need locks when writing concurrent code, he stares at the code until the threads organize themselves.

Paskaitos turinys

Bendros atminties modelio trūkumai

- Bendros atminties modelis reiškia, kad suteikiama galimybė gijoms lygiagrečiai vykdyti veiksmus su ta pačia atmintimi.
- Dažniausiai įgyvendinama per bendrus kintamuosius; dėl to reikalinga:
  - Kritinės sekcijos apsauga — padarius klaidą yra rizika sugadinti duomenis;
  - Sąlyginė sinchronizacija — padarius klaidą gali būti nepažadinamos gijos;

Bendros atminties modelio trūkumai

- Bendra atmintis reiškia, kad su duomenimis gali dirbti **tik vienas** procesas — bendra atmintis galima tik tarp gijų.
- Dėl šios priežasties neįmanoma programos paleisti per keletą kompiuterių — bendra atmintis galima tik tame pačiame kompiuteryje ir tame pačiame procese.
- Bendros atminties modelis reiškia, kad nėra duomenų atskyrimo. Pvz., jei masyvą paskirstome gijoms nurodydami, kurią masyvo dalį reikia apdoroti, gijos vis tiek turi prieigą prie viso masyvo.
- Bendros atminties modelis dažnai verčia naudoti negrynas funkcijas, kurios modifikuoja duomenų struktūrą.

Paskirstyta atmintis

- Paskirstytos atminties modelis — toks atminties modelis, kai kiekviena gija naudojasi tik savo atskira atmintimi.
- Gijos nesinaudoja jokiais bendrais resursais (bendrais kintamaisiais, failais ir kt.).
- Komunikacijai tarp gijų naudojamas koks nors žinučių apsikeitimo mechanizmas (tinklas, kanalai ar kt.).

Procesų sąveika

- Vienas procesas gali naudoti kito proceso skaičiavimo duomenis — procesai gali keistis duomenimis.
- Vienas procesas gali laukti tam tikro įvykio kitame procese — procesai gali būti sinchronizuojami.
- Apsikeičiant pranešimais vienas procesas **siunčia** žinutę, kitas — **laukia** žinutės.
- Jei procesas laukia žinutės, kurios niekas nesiųs arba siunčia žinutę, kurios niekas nelaukia, gali susidaryti aklavietė (*deadlock*).

Procesų sąveika

- Naudojant paskirstytos atminties modelį nebereikalinga duomenų apsauga nuo lygiagrečios prieigos.
- Kadangi monitorius saugo duomenis nuo lygiagrečios prieigos, o jos paskirstytos atminties modelyje nebūna, tai monitoriai paskirstytoje atmintyje nereikalingi.
- Kadangi bendros atminties nėra, o procesai siuntinėja vienas kitam žinutes, tai galima paleisti keletą procesų (ne gijų) skirtinguose kompiuteriuose, sujungti juos tinklu, ir skaičiavimų pajėgumai nebus apriboti vienu kompiuteriu.

Sinchroninis žinučių siuntimas

- Jei procesai nori apsikeisti žinute, vienas turi žinutę siųsti, kitas – gauti. Šie veiksmai visada yra aktyvūs – tiek siuntimą, tiek gavimą reikia suprogramuoti.
- Jeigu siunčiantis procesas laukia, kol gavėjas priims, sakome, kad procesai „susitinka“ (angl. *rendezvous*).
- Sinchroninio žinučių siuntimo variantai:
  - Paprastas susitikimas (*simple rendezvous*) — informacija perduodama viena kryptimi.
  - Išplėstas susitikimas (*extended rendezvous*) — informacija perduodama abiem kryptimis.
  - Selektyvus laukimas — gaunamos žinutės ribojamos.
- Siunčiant žinutes sinchroniškai blokuojami procesai – siunčiantis procesas negali tęsti darbo, kol žinutė nebus priimta.

Asinchroninis žinučių siuntimas

- Asinchroninis siuntimas — siuntėjas nelaukia, kol gavėjas gaus žinutę (*fire and forget*).
- Siunčiant žinutes asinchroniškai programos architektūra tampa sudėtingesnė, sunku grąžinti atsakymą.

Komunikuojantys nuoseklūs procesai

- *Communicating sequential processes* (*CSP*).
- Apsikeitimui žinutėmis naudojami *kanalai*.
- Procesai yra nepririšti prie kanalų — vienas procesas gali dirbti su keletu kanalų, o vienas kanalas naudojamas ne vieno proceso.
- Žinutės siunčiamos siunčiant duomenis į kanalą ir pasiimant duomenis iš kanalo.
- Nereikalingas procesų identifikavimas – tiek siuntėjas, tiek gavėjas nežino, su kuriuo procesu komunikuoja.
- Dažnai paleidžiant procesą jam parametru perduodami reikalingi kanalai.
- Naudojama:
  - occam;
  - Go;
  - C++CSP, PyCSP;
  - Crystal;

Go kanalai

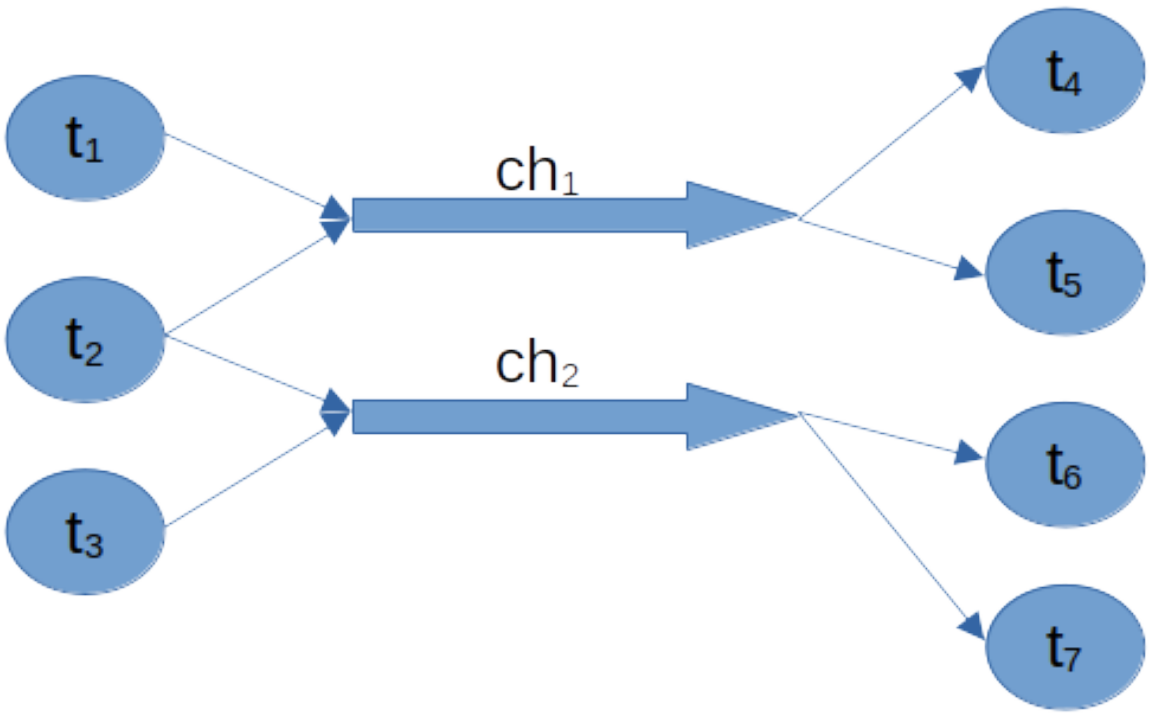
- Go naudojamas CSP modelis (*Communicating sequential processes*).
- Viena gija yra vienas nuoseklus procesas, o tarpusavyje jie veikia lygiagrečiai.
- Gijos viena kitai siunčia žinutes pasinaudojant kanalais (*channels*).
- Kanalas yra žinučių siuntimo mechanizmas. Gijos gali į kanalą rašyti arba iš kanalo skaityti duomenis.

Go kanalai

- Procesai dalinasi vienu bendru resursu — kanalu.
- Jei procesas rašo duomenis į kanalą, kitas procesas turėtų iš jo skaityti. Programoje turėtų būti tiek kartų skaitoma iš kanalo, kiek kartų į jį yra rašoma.
- Go kanalai palaiko tiek sinchroninį, tiek asinchroninį žinučių siuntimą.
- Sinchroniniam žinučių siuntimui naudojami nebuferizuoti kanalai. Tokiu atveju rašanti gija yra blokuojama, iki žinutė bus gauta.
- Asinchroniniam siuntimui naudojami buferizuoti kanalai (nepriimtos žinutės kaupiamos buferyje).

Go kanalai

$t_1 - t_3$ – siuntėjai, $t_4 - t_7$ – gavėjai. Žinutėmis apsikeičiama per kanalus ch_1 ir ch_2.



Go kanalai

- Nebuferizuotas kanalas, kuriuo galima siuntinėti `int` tipo žinutes, kuriamas `ch := make(chan int)`.
- Buferizuotas kanalas, kuris gali priimti iki 5 žinučių be gavėjo, kuriamas `ch := make(chan int, 5)`.
- Rašymas į tokius kanalus vykdomas `ch <- 10`.
- Skaitymas iš tokių kanalų vykdomas `msg := <- ch`.
- Procesų sinchronizavimui naudojami nebuferizuoti kanalai.

Go kanalų kryptys

- Go kanalus perduodant funkcijoms, funkcijos apraše galima nurodyti, kuris kanalo „galas“ bus naudojamas funkcijoje.
- `func myFunction(channel chan<- int)` parodo, kad kanalas `channel` bus naudojamas rašymui. Jei funkcijos viduje bus bandoma iš kanalo skaityti, funkcija nesikompiliuos.
- `func otherFunction(channel <-chan int)` parodo, kad kanalas `channel` bus naudojamas skaitymui. Jei funkcijos viduje bus bandoma į kanalą rašyti, funkcija nesikompiliuos.

Sinchroninis kanalas Go

```
func main() {
    var channel = make(chan int)
    go func() {
        channel <- 10
    }()
    var x = <- channel
    fmt.Println("Received value", x)
}
```

Programos rezultatas:

```
Received value 10
```

Asinchroninis kanalas Go

```
func main() {
    var channel = make(chan int, 5)
    var names = []string {"First", "Second", "Third", "Fourth", "Fifth"}
    for _, name := range names {
        var message = rand.Intn(65535)
        go sender(channel, name, message)
    }
    for i := 0; i < len(names); i++ {
        var message = <- channel
        fmt.Println("Received value", message)
    }
}

func sender(channel chan<- int, name string, x int) {
    fmt.Println(name, "is going to send value", x)
    channel <- x
}
```

Asinchroninis kanalas Go

Programos rezultatas:

```
First is going to send value 53126
Received value 53126
Fourth is going to send value 15554
Received value 15554
Fifth is going to send value 48091
Received value 48091
Third is going to send value 41687
Second is going to send value 4722
Received value 41687
Received value 4722
```

Neribotas skaitiklis

```
func increase(channel chan<- int) {
    for i := 0; i < 50; i++ {
        channel <- INCREASE
    }
    channel <- FINISHED
}

func decrease(channel chan<- int){
    for i := 0; i < 50; i++ {
        channel <- DECREASE
    }
    channel <- FINISHED
}
```

Neribotas skaitiklis

```
var count = 0
var channel = make(chan int)
for i := 0; i < increaserThreadCount; i++ {
    go increase(channel)
}
for i := 0; i < decreaserThreadCount; i++ {
    go decrease(channel)
}
var receivedFinishSignals = 0
for receivedFinishSignals < increaserThreadCount+decreaserThreadCount {
    var message = <-channel
    switch message {
    case INCREASE: // increase value
        count++
    case DECREASE: // decrease value
        count--
    case FINISHED: //finish
        receivedFinishSignals++
    }
}
```

`select` sakiniai

- `select` sakinys Go kalboje leidžia laukti keleto kanalų vienu metu.
- Paprogramės darbas blokuojamas iki tol, kol bent iš vieno kanalo gaunama žinutė.
- Jei vienu metu gaunamos kelios žinutės, kurią priimti pasirenkama atsitiktinai.
- `select` sakiniui galima nurodyti `default` sekciją. Tokiu atveju paprogramė neblokuojama: jei kuriame nors kanale jau yra žinutė, ji priimama, jei nėra, vykdoma tai, kas numatyta `default` sekcijoje.

Žinučių priėmimas iš keleto kanalų

```
var intChannel = make(chan int)
var floatChannel = make(chan float64)
go generateIntegers(intChannel)
go generateFloats(floatChannel)
var finishedThreads = 0
var intSum = 0
var floatSum = 0.0
```

Žinučių priėmimas iš keleto kanalų

```
for finishedThreads != 2 {
    select {
    case intValue := <-intChannel:
        if intValue != intEndMessage {
            intSum += intValue
        } else {
            finishedThreads++
        }
    case floatValue := <-floatChannel:
        if floatValue != floatEndMessage {
            floatSum += floatValue
        } else {
            finishedThreads++
        }
    }
}
fmt.Println("Integer sum:", intSum)
fmt.Println("Float sum:", floatSum)
```

Žinučių priėmimas iš keleto kanalų

```
func generateIntegers(outChannel chan<- int) {
    for i := 0; i < 15; i++ {
        outChannel <- i
    }
    outChannel <- intEndMessage
}

func generateFloats(outChannel chan<- float64) {
    for i := 0.5; i < 20; i += 1 {
        outChannel <- i
    }
    outChannel <- floatEndMessage
}
```

Paprastas ribotas skaitiklis, keletas kanalų

```
for remainingThreads != 0 {
    var activeChannels = []chan int{finisherChannel}
    if counter.count > min {
        activeChannels = append(activeChannels, decreaserChannel)
    } else {
        activeChannels = append(activeChannels, nil)
    }
    if counter.count < max {
        activeChannels = append(activeChannels, increaserChannel)
    } else {
        activeChannels = append(activeChannels, nil)
    }
    var cases []reflect.SelectCase
```

Paprastas ribotas skaitiklis, keletas kanalų

```
    for _, c := range activeChannels {
        cases = append(cases, reflect.SelectCase{
            Dir:  reflect.SelectRecv,
            Chan: reflect.ValueOf(c),
        })
    }
    chosenIndex, _, _ := reflect.Select(cases)
    switch chosenIndex {
    case 0:
        remainingThreads--
    case 1:
        counter.Decrease()
    case 2:
        counter.Increase()
    }
}
```

Interneto robotas (*web crawler*)

- Programos idėja:
  1. atsisiųsti nurodytų svetainių turinį;
  2. išspausdinti jų pavadinimus;
  3. surasti visas esančias nuorodas;
  4. atsisiųsti nurodų adresais esančius puslapius;
  5. išspausdinti jų pavadinimus.
- Naudojami tokie procesai:
  1. Pagrindinis: perduoda URL adresus parsiuntėjams;
  2. Parsiuntėjas: gauna URL adresus, parsiunčia turinį, suranda visas nuorodas, taip pat parsiunčia turinį. Visus turinius siunčia analizatoriams;
  3. Analizatorius: gauna puslapių turinius, randa pavadinimus, siunčia spausdintojui;
  4. Spausdintojas: gauna pavadinimus ir spausdina į ekraną;
  5. koordinatorius: siuntinėja ir gauna darbo pabaigos signalus.

Interneto robotas (*web crawler*): pagrindinis procesas

```
urlChannel := make(chan string)
contentChannel := make(chan string)
titleChannel := make(chan string)

coordinatorToMain := make(chan bool)
crawlerToCoordinator := make(chan bool)
coordinatorToPrinter := make(chan bool)
urls := [4]string{"https://www.google.com",
    "https://duckduckgo.com/",
    "https://www.bing.com", "https://yahoo.com"}
//launch all other threads
for _, url := range urls {
    urlChannel <- url
}
for i := 0; i < crawlerCount; i++ {
    urlChannel <- ""
}
<-coordinatorToMain
```

Interneto robotas (*web crawler*): parsiuntėjas

```
for {
    url := <-urlChannel
    if url == "" {
        crawlerToCoordinator <- true
        break
    }
    content := getWebpageContent(url)
    contentChannel <- content
    for _, link := range analyzeWebContent(content) {
        linkContent := getWebpageContent(link)
        contentChannel <- linkContent
    }
}
```

Interneto robotas (*web crawler*): analizatorius

```
for {
    content := <-contentChannel
    title := extractWebpageTitle(content)
    if title == "<die>" {
        break
    }
    if len(title) > 0 {
        titleChannel <- title
    }
}
```

Interneto robotas (*web crawler*): spausdintojas

```
for {
    select {
    case title := <-titleChannel:
        fmt.Println(title)
        break
    case <-coordinatorToPrinter:
        return
    }
}
```

Interneto robotas (*web crawler*): koordinatorius

```
for i := 0; i < crawlerCount; i++ {
    <- crawlerToCoordinator
}
for i := 0; i < analyserCount; i++ {
    contentChannel <- "<die>"
}
coordinatorToPrinter <- true
coordinatorToMain <- true
```

`close` funkcija

- Kanalą galima uždaryti pasinaudojant `close` funkcija ir jai perduodant reikiamą kanalą.
- Iš kanalo skaitanti gija tada gali naudoti kanalą for-each stiliaus cikle kaip masyvą – kiekvienoje ciklo iteracijoje bus priimama žinutė.
- Ciklas baigiasi, kai uždaromas kanalas.

```
for message, index := range channel {
    // do something with message and index
}
```