Sinhronizacija

Pogojne spremenljivke [SMAP:6.2]

- uporabimo jih za zaustavitev in ponovni zagon gorutin glede na stanje pripadajoče ključavnice
- delovanje
  - gorutina najprej zaklene ključavnico
  - ko pride do funkcije `Wait`, ključavnico sprosti in čaka na signal za nadaljevanje; pri tem se postavi na zadnje mesto v vrsti zaustavljenih gorutin
  - ko pogojna spremenljivka prejme signal, se gorutina zbudi, zaklene ključavnico in nadaljuje
    - s `Signal` zbudimo samo eno spečo gorutino
    - z `Broadcast` zbudimo vse speče gorutine
  - na koncu mora gorutina ključavnico sprostiti
- pogojne spremenljivke ne hranijo vrednosti; bolj gre za vrsto spečih gorutin, ki gredo v izvajanje ob izpolnjenem pogoju
- primer: pregrada (*angl.* barrier) [LBS:3.6]
  - želimo, da se na neki točki vse gorutine počakajo in potem skupaj nadaljujejo
  - rešimo lahko z `wg.Add()`, `wg.Done()` in `wg.Wait()`, vendar vsakič ustvarjamo gorutine
  - pregrada-1.go
    - osnutek, brez nadzora
  - pregrada-2.go
    - štejemo gorutine, ki so že izvedele operacije v zanki
    - do težave pride ob ničenju števca - preden gorutina 0 postavi števec na 0, lahko druga gorutina že izvede naslednjo iteracijo in poveča števec, program se obesi
    - če ne bi imeli več obhodov zanke in s tem večkratne rabe iste pregrade, bi bila rešitev sprejemljiva
  - pregrada-3a.go
    - pregrado si predstavljamo kot dvoja vrata
      - vrata 1 se odprejo šele, ko so vse gorutine šle skozi vrata 0 (faza 0)
      - vrata 0 se odprejo potem, ko so vse gorutine šle skozi vrata 1 (faza 1)
      - spremenljivki števec gorutin med vrati (`g`) in faza (`phase`)
      - jezik go zazna tvegana stanja - ena gorutina piše, druga bere
  - pregrada-3b.go
    - rešitev z dvojimi vrati v kateri ne dovolimo hkratnih pisanj in branj
  - pregrada-4.go
    - podobna rešitev kot pregrada-3b.go
    - rešitev s kanali je bolj elegantna: gorutina je blokirana medtem ko čaka, da se v kanalu pojavi podatek
    - dva kanala, z enim nadziramo prihajanje niti k pregradi (vrata 0), z drugim odhajanje niti od pregrade (vrata 1)
    - kanala sta tipa `struct{}`, s čimer poudarimo, da jih uporabljamo samo za sinhronizacijo
  - pregrada-5.go
    - ustvarimo pogojno spremenljivko in jo povežemo s ključavnico
    - vse gorutine, razen zadnje, damo na čakanje
    - ko preštejemo zadnjo gorutino, postavimo števec na 0 in zbudimo ostale gorutine
- primer: proizvajalci in porabniki (*angl.* producer-consumer) [LBS:4.1]
  - izmenjava podatkov med gorutinami preko medpomnilnika
  - proizvajalci pošiljajo podatke v medpomnilnik
  - porabniki podatke pobirajo iz medpomnilnika
  - proizvajalci in porabniki morajo usklajeno dostopati do medpomnilnika
    - proizvajalec ne sme dodajati podatkov v medpomnilnik, če je ta poln
    - porabnik ne sme jemati podatkov iz medpomnilnika, če je ta prazen
  - proizvajalci-porabniki-1.go
    - ogrodje kode brez sinhronizacijskih elementov
    - medpomnilnik za enega ali več podatkov s števcem števila podatkov in dvema indeksoma za krožno vrsto
  - proizvajalci-porabniki-2.go
    - za vzpostavitev zanesljive izmenjave potrebujemo
      - ključavnico

        - dve pogojni spremenljivki za uspavanje in zbujanje proizvajalcev in porabnikov
    - če je medpomnilnik poln, se proizvajalci ustavijo
    - če je medpomnilnik prazen, porabniki čakajo
    - proizvajalci signalizirajo porabnikom, da lahko poberejo podatke iz medpomnilnika
    - porabniki signalizirajo proizvajalcem, da je v medpomnilniku prostor in lahko proizvajajo
    - s pogojnimi spremenljivkami in ključavnico v bistvu naredimo kanal
- proizvajalci-porabniki-3.go
    - elegantna rešitev s kanalom
    - komunikacija po kanalu brez pomnilnika je enakovredna prejšnji rešitvi z medpomnilnikom velikosti ena
    - z zapiranjem kanala elegantno sporočimo porabnikom, da lahko zaključijo