Porazdeljeni sistemi, izpit - rešitve

22. januar 2026

ime in priimek:

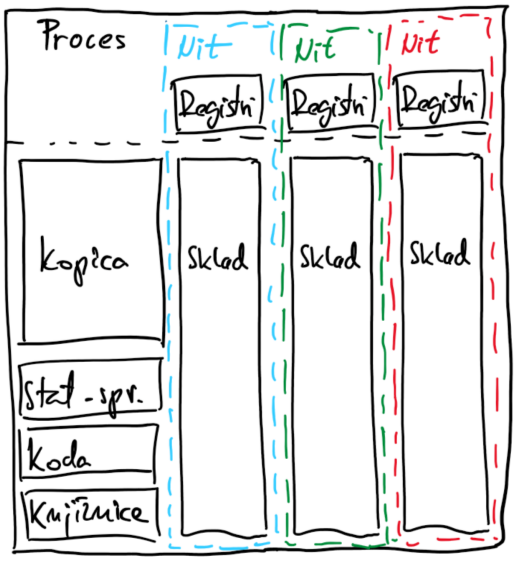
čas reševanja: 60 minut, naloge so enakovredne

vpisna številka:

1. Zakaj je pri velikem številu jeder v procesorju arhitektura NUMA primernejša od arhitekture UMA?

   - *procesorska jedra so razdeljena v domene NUMA*
   - *pomnilniški moduli so enakomerno razporejeni med domene NUMA*
   - *dostopni časi do pomnilniških modulov so različni: krajši do pomnilniških modulov, neposredno povezanih na domeno, daljši do pomnilnikov bolj oddaljenih domen (časi so lahko do 3-krat daljši)*
   - *izvajanje je učinkovito, če procesorska jedra iz izbrane domene čim več delajo s pomnilnikom, ki je neposredno povezan nanjo*

2. Narišite pomnilniške strukture večnitnega procesa v modernem operacijskem sistemu. Jasno označite strukture, ki so skupne vzem nitim, in strukture, ki so lastne posameznim nitim.



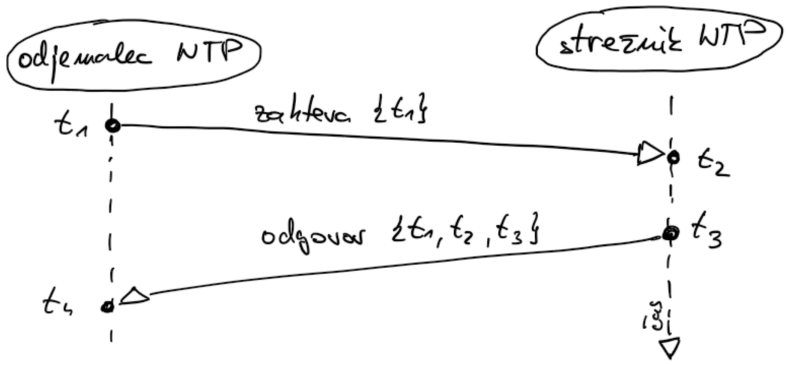
3. Kritični odsek zaklepamo s tremi ključavnicami, pri čemer ne moremo zagotoviti enakega vrstnega redna zaklepanja v vseh gorutinah. Napišite psevdo kodo za zaklepanje kritičnega odseka, ki preprečuje smrtni objem.

```
for {
    lockA.Lock()
    if !lockB.TryLock() {
        forkA.Unlock()
        time.Sleep(time.Millisecond)
        continue
    }
    if !lockC.TryLock() {
        forkA.Unlock()
        forkB.Unlock()
         time.Sleep(time.Millisecond)
         continue
    }
    break
}
```

4. V jeziku go želimo napisati program, v katerem glavna gorutina zažene *N* dodatnih gorutin, da se izvajajo sočasno. Vsaka dodatna gorutina izpiše svojo oznako in zaključi. Program popravite tako, da se bodo vse oznake pojavile na standardnem izhodu. Za sinhronizacijo gorutin uporabite kanale.

```
package main

import "fmt"

const N int = 5

var ch chan struct{} = make(chan struct{})

func worker(id int) {
    fmt.Println(id)
    ch <- struct{}{}
}

func main() {
    for i := range N {
        go worker(i)
    }
    for range N {
        <- ch
    }
}
```

5. Podajte ključno vsebino sporočil, ki si jih odjemalec in strežnik izmenjata med usklajevanjem ure po protokolu NTP. Kako po izmenjavi sporočil popravimo čas na odjemalcu?



zamik ure je tako $\theta = t_S - t_4 = (t_2 - t_1 + t_3 - t_4) / 2$

6. V sistem za verižno replikacijo dodamo novo vozlišče. Opišite postopek dodajanja in usklajevanja shrambe.

   - *najprej uskladimo shrambo novega vozlišča z repom*
   - *potem uredimo prevezavo tako, da rep postane vmesno vozlišče, novo vozlišče pa rep*
   - *o naslovu novega repa obvestimo odjemalce*

7. V gručo za replikacijo z voditeljem (algoritem raft) je vključenih pet vozlišč. Zaradi težav z omrežjem gruča razpade na dva dela: v delu A sta voditelj in sledilec, v delu B pa trije sledilci. Kaj se bo v nadaljevanju ob zahtevah za vpis v dnevnik dogajalo v delu A in kaj v delu B?

   - *A: zahteve za vpis se ne potrjujejo, saj so za potrditev potrebni trije; voditelj obvešča sledilca o svoji vlog*
   - *B: zaznajo da ni voditelja; izvedejo volitve; zahteve za vpis se potrjujejo (imamo tri procese kar je večina v pet-članski gruči raft.*

8. V gruči imamo več vozlišč, vsako vozlišče ima svojo repliko shrambe. Vsako vozlišče prejeto zahtevo za vpis v shrambo razširi na vsa ostala vozlišča. Sistem mora zagotavljati končno skladnost shramb. V katerih primerih si lahko privoščimo sočasno vpisovanje na različna vozlišča?

   Če lahko uporabljamo:
   - *podatkovne tipe CRDT za replikacijo brez sporov (števci, slovarji, množice, ...)*
   - *monotone registre (ohranimo zapis z novejšim časovnim žigom, večvrednostni registri)*

9. Na grafičnih pospeševalnikih so niti organizirane hierarhično: mreža niti je razdeljene na bloke, ti pa na snope. Aplikacija iterativno računa stanje v času *t*+1 iz stanja v času *t*. Kako pravilno zagotovimo, da niti začnejo z novo iteracijo šele potem, ko so vse končale trenutno?

   Globalno sinhronizacijo lahko zagotovimo s ponovnim proženjem ščepca na gostitelju.

10. Imamo vektorja $\boldsymbol{a}$ in $\boldsymbol{b}$ dolžine n, ki jih želimo na grafičnem pospeševalniku sešteti, $\boldsymbol{c} = \boldsymbol{a} + \boldsymbol{b}$. Pripravite ščepec `vectorAdd` tako, da bo vsaka nit izvedla največ eno seštevanje. Določite enodimenzionalno konfiguracijo mreže niti, na kateri zaženemo ščepec. Vzemite, da je število niti v bloku večkratnik števila 32 in je podano v naprej.

```
__global__ void vectorAdd (float *c, const float *a, const float *b, int n) {

    int gid = blockDim.x * blockIdx.x + threadIdx.x;
    if (gid < n)
        c[gid] = a[gid] + b[gid];

}
```

   Število blokov niti: $\lceil 3000 \,/\, 256 \rceil = \lceil 11{,}72 \rceil = 12$.