Problem međusobnog isključivanja

Tina Marić, Gregor B. Banušić i Mia Filić

Prirodoslovno-matematički fakultet

9. srpnja 2017.

Algoritmi za međusobno isključivanje:

1. Singhal-ov algoritam zasnovan na dinamičkoj strukturi podataka,
2. algoritam Lodha & Kshemkalyany koji predstavlja optimizaciju algoritma R & A s predavanja,
3. algoritam Maekawa zasnovan na kvorumu.

Singhal-ov algoritam

- zasnovan na dinamičkoj strukturi podataka,
- IDEJA: procesi koji često zahtjevaju CS, traže dopuštenje samo od procesa koji često zahtjevaju CS,
- IZVEDBA:
  - svaki proces P_i čuva dodatne skupove znanja R_i i I_i,
  - prioriteti zahtjeva za CS se određuju pomoću jednostavnog Lamportovog sata, žigosanjem poruka,
  - dvije vrste kontrolnih poruka: (1) *request* i (2) *okay*,
  - 3 varijable stanja:$zahtjev$,$mojPrioritet$,u_CS

Singhal-ov algoritam: ulazak u CS

```
1 Korak 1: Zahtjev za CS;
2 zahtjev = True;
3 C_i = C_i + 1;
4 foreach P_j ∈ R_i do
5     pošalji poruku tipa request P_j-u ;
6 while True do
7     if R_i == ∅ then
8         break;
9 zahtjev = False;
```

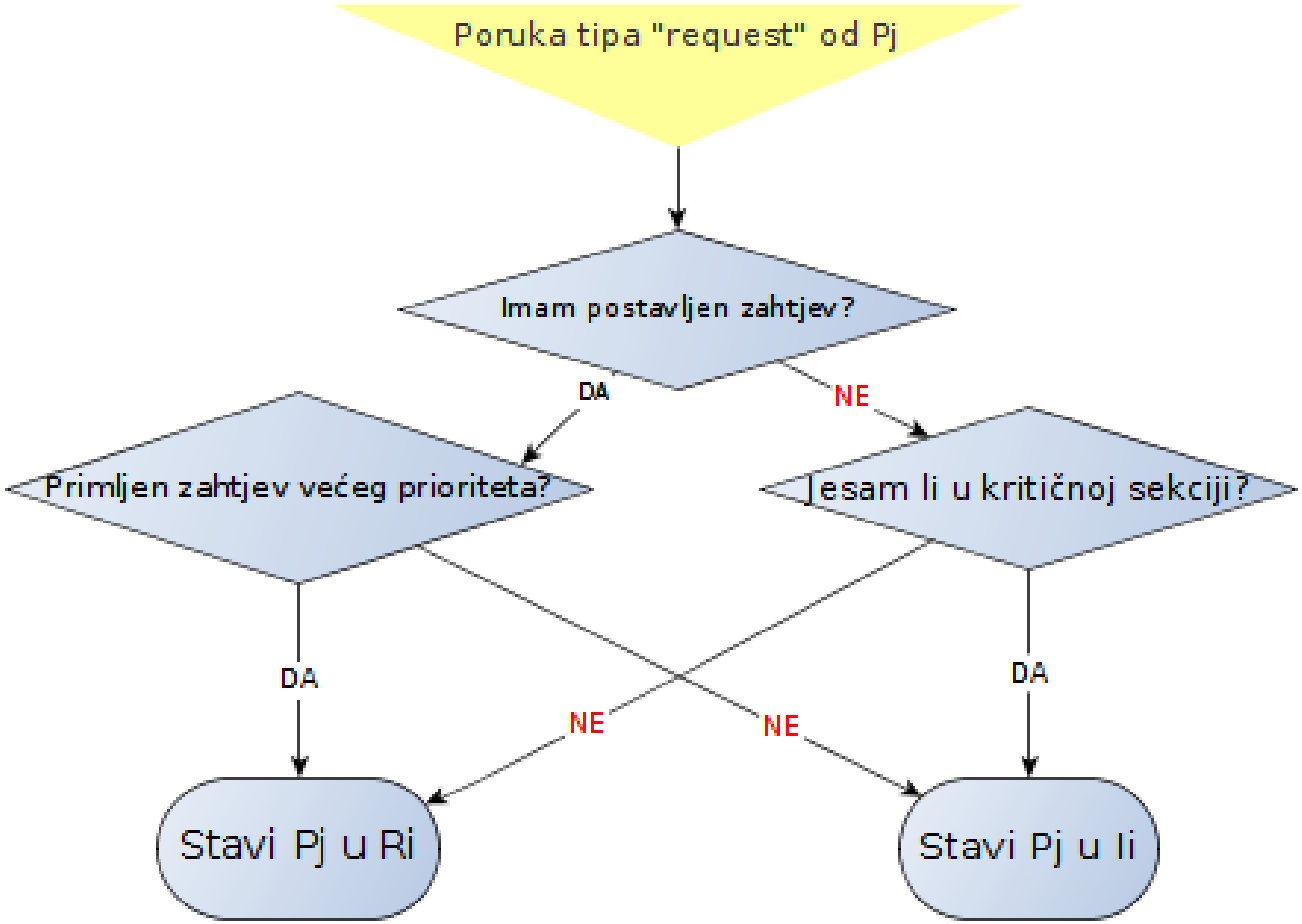
Singhal-ov algoritam: CS i izlazak iz CS

1 Korak 2: CS;
2 $u\_CS = True$;
3 kritična sekcija;
4 $u\_CS = False$;

1 Korak 3: Izlazak iz CS;
2 **foreach** $P_j \in I_i \setminus \{P_i\}$ **do**
3 $\quad I_i = I_i \setminus P_j$;
4 $\quad$ pošalji poruku tipa *okay* P_j-u ;
5 $\quad R_i = R_i \cup P_j$;

Dinamičko očitavanje stanja sustava

- P_i primi poruku tipa *request* od P_j:



- proces P_i primi poruku tipa *okay* od P_j:
  - $R_i = R_i \setminus P_j$,

1. Kako efikasno prepoznati koji procesi trenutno zahtjevaju ulaz u CS?
2. Kako projektirati postavljenje zahtjeva za CS za procese iz grupe 2 (manje aktivne procese)?
3. Kako omogućiti prijelaz iz grupe 2 u grupu 1 i obratno?
4. Ukoliko proces ne zahtjeva odobrenje za ulazak u CS od svih procesa, kako osigurati svojstvo *sigurnosti*?
5. Kako osigurati da prikupljanje informacija o trenutnom stanju sustava ne utječe na efikasnost poništavajući dobiveno poboljšanje?

Algoritam Lodha & Kshemkalyany

- Algoritam Lodha i Kshemkalyani je optimizirana verzija algoritma Ricarta i Agrawale za međusobno isključivanje.
  - sigurnost ✓
  - pravednost ✓
  - odsustvo izgladnjivanja ✓
- Proces ne treba odobrenje svih procesa da uđe u kritični odsječak, već samo od onih čiji zahtjevi prethode njegovom zahtjevu prema prioritetu.

- Vrste poruka:
  - REQUEST
  - OKAY
  - FLUSH

  Poruke imaju vremenski žig.
- konkurentnost
- LQRi sadrži konkurentne zahtjeve posljednjeg zahtjeva Ri

- Poruka **OKAY**:
  - dopuštenje od procesa koji nije zainteresiran za KO
  - kolektivno dopustenje od procesa koji imaju zahtjeve viseg prioriteta
- Poruka **FLUSH**
  - odgovor procesa sa višim zahtjevima
  - poruka procesu sa sljedećim najvecim prioritetom (ostalima OKAY)
- Poruka **REQUEST**
  - nema konkurentnih zahtjeva
  - ima konkurentnih zahtjeva, request nižeg služi kao okay

Da bi ušao u KO, proces pošalje $(N-1)$ poruka request, a primi $(N - |\text{broj konkurentnih zahtjeva}|)$ odgovora u obliku *okay/flush*

Algoritam Maekawa

- prvi algoritam za međusobno isključivanje zasnovan na kvorumu
  - proces ne zahtijeva dopuštenje od svih ostalih procesa nego samo od nekog podskupa procesa
  - podskupovi zadovoljavaju pravilo $\forall$ i, j:1$\leq$i,j$\leq$N $\Rightarrow$ Ri$\cap$ Rj$\neq \emptyset$ i svaki skup se naziva *kvorum*
- Skup kvoruma u algoritumu Maekawa konstruiran je tako da zadovoljava sljedeće uvjete:

  M1 ($\forall$ i,j : i $\neq$ j $\Rightarrow$ Ri $\cap$ Rj $\neq \emptyset$)
  M2 ($\forall$ i : 1$\leq$i$\leq$N $\Rightarrow$ Si $\in$ Ri)
  M3 ($\forall$ i : 1$\leq$i$\leq$N $\Rightarrow$ $\mid$ Rt $\mid$ = K)
  M4 Svaki proces nalazi se u K kvoruma
- Maekawa je koristio teoriju projektivnih ravnina za konstrukciju kvoruma. Pokazao je da vrijedi N = K(K - 1) + 1. Iz te relacije slijedi $\mid$ Ri $\mid$ = $\sqrt{N}$.

Primjeri skupa kvoruma

Tablica: Primjer skupa kvoruma koji zadovoljavaju uvjete algoritma Maekawia gdje je K=2

K=2, N=3	$R_1 = \{1, 2\}$ $R_2 = \{2, 3\}$ $R_3 = \{1, 3\}$

Tablica: Primjer skupa kvoruma koji zadovoljavaju uvjete algoritma Maekawia gdje je K=3

K=3, N=7	$R_1 = \{1,2,3\}$ $R_2 = \{2,4,6\}$ $R_3 = \{3,5,6\}$ $R_4 = \{1,4,5\}$ $R_5 = \{2.5.7\}$ $R_6 = \{1,6,7\}$ $R_7 = \{3,4,7\}$

Pseudokod

1. **Zahtijevanje kritičnog odsječka**
   - Proces Si šalje REQUEST(i) svim procesima koji se nalaze u istom kvorumu Ri
   - Kada proces Sj primi REQUEST(i), šalje REPLY(j) procesu Si, samo ukoliko poruka REPLY nije već poslana nekom procesu inače stavlja u red čekanja REQUEST(i) za kasniju obradu.
2. **Izvršavanje kritičnog odsječka**
   - Proces Si ulazi u kritični odsječak kada primi poruku REPLY od svakog procesa u kvorumu Ri.
3. **Izlazak iz kritičnog odsječka**
   - Proces Si šalje poruku RELEASE(i) svakom procesu u kvorumu Ri
   - Kada proces Sj primi RELEASE(i) od Si, šalje REPLY sljedećem procesu koji je u njegovom redu čekanja te ga miče iz reda. Ukoliko je red prazan proces ažurira svoje stanje tako da tako da može poslati poruku REPLY kao odgovor na REQUEST poruku

Svojstva

- Na prethodno opisan način postiže se međusobno isključivanje
- Ali može doći do zastoja (deadlock) zato što proces može biti blokiran od strane drugih procesa
- Poruke REQUEST nisu prioritizirane s obzirom na vrijeme slanja. Dakle, proces može poslati zahtjev procesu i kasnije prisiliti zahtjev većeg prioriteta da čeka (nije zadovoljeno svojstvo pravednosti).

Dodatne poruke

- FAILED, INQUIRE, YIELD

1. Kada proces Sj blokira zahtjev REQUEST(ts, i) [1] procesa Si zato što je već dao dopuštenje procesu Sk, proces Sj šalje procesu Si poruku FAILED(j) ukoliko Si ima manji prioritet od procesa Sk. Inače proces Sj šalje procesu Sk poruku INQUIRE(j).
2. Kao odgovor na primljenu poruku INQUIRE(j) od procesa Sj, proces Sk šalje poruku YIELD(k) procesu Sj nakon što je primio poruku FAILD od nekog procesa u svom kvorumu i ukoliko je poslao YIELD poruke, a nije primio REPLY kao odgovor.
3. Kada proces Sj primi YIELD(k) poruku od procesa Sk, proces Sj sprema zahtjev porocesa Sk na pravo mjesto u svom redu procesa koji čekaju i šalje REPLY(j) poruku procesu koji je prvi u redu.

[1] `ts je vremenski žig (timestamp)`