11. Algoritmi međusobnog isključivanja

U jednoj od posljednjih vježbi u ovoj akademskoj godini ćemo se baviti algoritmima međusobnog isključivanja.

11.1 Algoritmi međusobnog isključivanja u raspodijeljenim sustavima

Međusobno isključivanje predstavlja jedan od osnovnih problema s kojima se susrećemo prilikom izrade raspodijeljenih sustav. Zadatak međusobnog isključivanja jest osigurati da kritičnoj sekciji u bilo kojem momentu može pristupiti isključivo jedan proces za cijelo vrijeme dok je raspodijeljeni sustav pokrenut.

S problemom međusobnog isključivanja ste se možda i prije susreli na nekim drugim kolegijima kao što su Operacijski sustavi i Paralelno programiranje. Kod višenitnih sustava kontrolu pristupa kritičnoj sekciji možemo ostvariti. npr. korištenjem semafora. Raspodijeljeni sustavi su (ponovno) specifični. Ta specifičnost je posljedica toga što se sve informacije između procesa, koji mogu biti geografski podijeljeni, razmijenjuju putem poruka. Dakle, procesi o međusobnim stanjima saznaju preko poruke koju prome. Stoga se i postupak međusobnog isključivanja mora zasnivati na komunikaciji porukama. I sama odluka u konačnici se donosi na osnovu razmijenjenih poruka.

Dosta je zahtjevno dizajnirati ovakav algoritam zbog toga što poruke mogu kasniti, mogu se gubiti, a pri tome ni jedan proces nema kompletnu sliku stanja cijelog sustava u svakome trenutku. Na vježbama ćemo pretpostaviti da se poruke ne mogu izgubiti.

U sustavima koji se zasnivaju na slanju poruka između procesa, a u te spadaju i raspodijeljeni sustavi, algoritmi se najčešće zasnivaju na jednoj od sljedeće tri paradigme:

1. logički sat - zahtjevi za ulazak u kritičnu sekciju se sortiraju po prioritetuna osnovu vrijednosti logičkog sata.
2. razmjena tokena - proces koji "drži" token je privilegiran.
3. kvorumi - da bi postao privilegiran, proces mora dobiti dopuštenje kvoruma procesa. Svaki par kvoruma bi trebao imati ne prazan presjek.

11.1.1 Zahtjevi međusobnog isključivanja

Algoritmi međusobnog isključivanja moraju udovoljiti sljedećim zahtjevima:

1. Sigurnost - u bilo kojem momentu isključivo jedan proces se može nalaziti u kritičnoj sekciji.
2. Živost - svojstvo zahtjeva da nema deadlock-a ni izgladnjivanja procesa. Deadlock - dva ili više procesa ne smiju vječno čekati na poruke koje nikada neće stići. Izgladnjivanje - svaki proces koji zatraži korištenje kritične sekcije mora u konačnom vremenu dobiti mogućnost korištenja te sekcije - mora doći na red.
3. Pravednost - u kontekstu međusobnog isključivanja, ovo znači da svaki proces ima jednaku mogućnost korištenja kritične sekcije. Ovo najčepće znači da se zahtjevi za korištenjem kritične sekcije odobravaju redosljedom kojim su pristigli, a što se kontrolira logičkim satom.

11.2 Lamportov algoritam međusobnog isključivanja

Algoritam se bazira na korištenju lamportovog logičkog sata.

Lamport je razvio ovaj algoritam kao ilustraciju sheme svog logičkog sata. Algoritam je pravedan u smislu da se zahtjevi izvršavaju redosljedom koji je određen vrijednostima logičkog sata procesa. Lamportovim logičkim satom smo se bavili na ranijim vježbama.

Ako neki proces želi poslati zahtjev za kritičnom sekcijom, isti ažurira vrijednost svog logičkog sata te ju pridjeli zahtjevu koji će poslati. Algoritam obrađuje zahtjeve za kritičnom sekcijom uzlaznim redosljedom. Prisjetite se iz lekcije o lamportovom satu kako možemo postići da su procesi potpuno uređeni.

Svaki proces u sebi sadrži sortiranu listu primljenih zahtjeva - $requestqueue_i$, gdje su zahtjevi sortirani od prvoga kojem će se odobriti ulazak u kritičnu sekciju prema zadnjem. Algoritam zahtjeva da komunikacijski kanali dostavljaju poruke po FIFO principu.

Algoritam:

Zahtjev za kritičnom sekcijom:

- Ako proces p_i želi pristupiti kritičnoj sekciji, šalje REQUEST<tp_i, i> poruku svim ostalim procesima te stavlja svoj zathjev u $requestqueue_i$. Uređeni par (tp_i, i) predstavlja vremensku oznaku procesa koja se sastoji od vrijednosti lamportovog logičkog sata tp_i procesa p_i i ID-a i procesa p_i.
- Kada proces p_j primi zahtjev REQUEST<tp_i, i> od procesa p_i, stavlja zahtjev u svoj $requestqueue_j$ i odgovara procesu p_i s porukom REPLY<tp_j>.

Izvršavanje u kritičnoj sekciji:

Proces p_i pristupa kritičnoj sekciji ako su zadovoljena sljedeća da uvjeta:

- Proces p_i je primio poruku koja sadrži veću vremensku oznaku od one u svome zahtjevu REQUEST<tp_i, i> od svih ostalih procesa.
- Zahtjeva procesa p_i se nalazi na prvome mjestu u $requestqueue_i$.

Oslobađanje kritične sekcije:

- Proces p_i po izlasku iz kritične sekcije uklanja svoj zahtjev s vrha $requestqueue_i$ te šalje RELEASE poruku s vremenskom oznakom svim ostalima procesima.
- Proces p_j po primitku RELEASE poruke procesa p_i uklanja zahtjev procesa p_i s $requestqueue_j$.

Uklanjanjem zahtjeva iz $requestqueue_i$, postoji mogućnost da zahtjeva procesa p_i dođe na prvo mjesto liste za ulazak u kritičnu sekciju.

11.3 Algoritmi bazirani na kvorumu

Algoritmi bazirani na kvorumu su specifični iz sljedećih razloga:

1. Proces ne traži dopuštenje za ulazak u kritičnu sekciju od svih ostalih procesa, već samo od dijela (podskupa) svih procesa. Ovo je potpuno drugačiji pristup u odnosu na algoritme bazirane na logičkom satu kao što je to Lamportov ili Ricart-Agrawala u kojima svi procesi sudjeluju u rješavanju međusobnog isključivanja. Kod algoritama baziranih na kvorumu, biraju se skupovi procesa na način $\forall i \forall j : 1 \leq i, j \leq N :: R_i \cap R_j \neq \emptyset$. Posljedično - svaki skup procesa ima nekog procesa koji služi kao medijator među konfliktima.
2. Kod algoritama baziranih na kvorumu, proces može poslati samo jednu REPLY poruku u nekom trenutku. Proces može poslati novu REPLY poruku samo nakon što je primio RELEASE poruku za prethodnu REPLY poruku. Stoga, proces p_i *zaključa* sve procese skupa R_i u nekakvom isključnom modu prije nego što izvrši svoju kritičnu sekciju.

Oivm algoritmi značajno smanjuju kompleksnost razmjene poruka u raspodijeljenim sustavima. Iznimno je važno s aspekta ispravnosti da svaka dva skupa procesa imaju zajednički element, a s aspekta optimalnost da nijedan skup nije podskup nekog drugog skupa.

11.3.1 Maekawa-in algoritam

Ovo je prvi algoritam baziran na kvorumu za međusobno isključivanje u raspodijeljenom sustavu. Skupovi procesa, koje ćemo u ostatku teksta zvati *requestsets*, u ovom algoritmu su konstruirani na način da zadovoljavaju sljedeća svojstva:

- $(\forall i \forall j : i \neq j, 1 \leq i, j \leq N :: R_i \cap R_j \neq \emptyset)$ - svojstvo koje kaže da presjek bilo koja dva skupa ne smije biti prazan skup.
- $(\forall : 1 \leq i \leq N :: P_i \in R_i)$.
- $(\forall : 1 \leq i \leq N :: |R_i| = K)$ - svaki proces ima svoj skup procesa kojima šalje poruke, ovo svojstvo kaže da bi ovaj skup svakog procesa trebao biti jednak tako da svaki proces ima jednako posla za odraditi.
- jednak broj procesa bi trebao zatražiti dozvolu od bilo kojeg drugog procesa što implcira da svi procesi imaju "jednaku odgovornost" u dodjeli dopuštenja drugim procesima.

Prvi i drugi uvjeti pružaju garanciju ispravnosti, a treći i četvrti omogućuju algoritmu druga željena svojstva.

Budući da su skupovi procesa složeni tako da postoji barem jedan zajednički proces, postojati će medijator koji garantira isključivost između ta dva skupa. Proces može poslati samo jednu REPLY poruku - može dati svoje dopuštenje samo jednom procesu. Algoritam pretpostavlja da će poruke biti dostavljene redosljedom kojim su poslane.

Algoritam:

Zahtjeva za kritičnom sekcijom:

- Proces P_i zahtjeva ulazak u kritičnu sekciju od svih procesa koji se nalaze u njegovom request setu. To čini slanjem REQUEST(i) poruke.
- Kada proces P_j primi REQUEST(i) poruku, šalje REPLY(j) odgovor procesu p_i ako je to prvi REPLY kojeg šalje ili je u međuvremenu primio RELEASE poruku. U suprotnom, sprema REQUEST(i) za kasnije razmatranje - nazovimo ovu listu deffered.

Izvršavanje kritične sekcije:

- Proces P_i započinje s izvršavanjem kritične sekcije tek po primitku REPLY poruke od svakog člana svog request set-a.

Oslobađanje kritične sekcije:

- Nakon što je proces P_i gotov s kritičnom sekcijom, šalje RELEASE(i) poruku svakom procesu u svom request set-u R_i.
- Kada proces P_j primi RELEASE(i) poruku od procesa P_i, šalje REPLY poruku sljedećem procesu čiji REQUEST se nalazi u deffered listi te briše taj REQUEST iz liste. U slučaju da je lista prazna, proces mijenja svoje stanje na način da naznači kako nije poslao nijedan REPLY od prethodne RELEASE poruke.

Važno je napomenuti da se ovaj algoritam može naći u deadlock-u. Proučite izvore navedene na kraju vježbe te pogledajte kako se taj problem riješava.

11.4 Zadatci

1. U primjeru Lamportovog algoritma za međusobno isključivanje kojeg ste dobili gotovog, svi procesi (actori) u isto vrijeme šalju zahtjev za kritičnom sekcijom. Modificirajte primjer na način da svi actori čekaju X milisekundi prije nego pošalju zahtjev za kritičnu sekciju. X je slučajan broj.
2. Prepoznajte kritičnu sekciju u primjeru te ju izmijenite na način da zapisujete poruku u datoteku.
3. Modificirajte dobiveni primjer na način da koristite Akka.Cluster. Je li vam potrebno onda slati Identify poruke? Treba li vam slučaj za primanje ActorIdentify poruke? Kojom porukom ga možemo zamijeniti i zašto?
4. Izmijenite primjer koji ste dobili na način da svaki actor šalje beskonačan broj zahtjeva.
5. Postoje mnogo optimalniji algoritmi za međusobno isključivanje od Lamportovog. Jedan od tih, koji se također bazira na korištenju logičkog sata jest Ricart-Agrawala algoritam. Proučite ga i implementirajte koristeći Akka.
6. Proučite izvore navedene ispod te pogledajte na koji način se riješava problem deadlock-a u Maekawa-inom algoritmu.

11.5 Izvori

1. `https://en.wikipedia.org/wiki/Maekawa%27s_algorithm`, 31.05.2016. u 23:49.
2. Distributed algorithms: An intuitive approach, Wan Fokkink.
3. `https://en.wikipedia.org/wiki/Lamport%27s_distributed_mutual_exclusion_algorithm`, 31.05.2016. u 23:50.
4. `http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=55221EB5741DE85B82089C1F87B7F doi=10.1.1.135.825&rep=rep1&type=pdf`, 01.06.2016. u 08:06.