Uvod u distribuirane informacijske sustave

Sinkronizacija

Uvod

- Procesi
- Komunikacija
- Imenovanje
- Sikronizacija
  - Kako procesi surađuju i međusobno se sinkroniziraju
  - Suradnja –omogućena sustavom imenovanja, dva procesa mogu pristupati istom resursu, ali moramo osigurati sinkronizaciju

Sinkronizacija

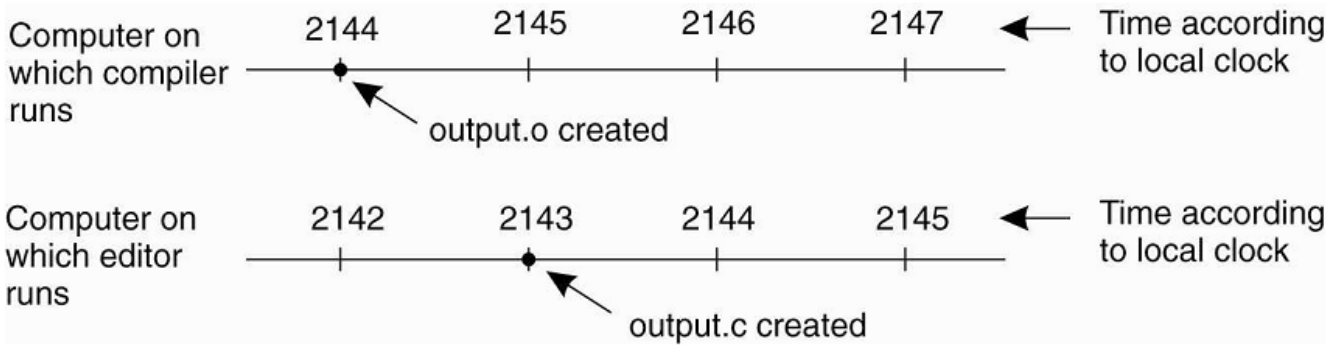
- Sinkronizacija se odnosi na:
  - Osiguravanje da procesi istovremeno ne pristupaju istom resursu:
    - Osigurati “mutual exclusion”
  - Dogovoriti se o poretku događaja:
    - Ako se događaj e1 mora dogoditi prije događaja e2
    - Poruka x koju šalje proces P da li je poslana prije ili nakon poruke y koju šalje proces Q

Sinkronizacija

- Sinkronizacija satova
  - Fizički satovi
  - Logički satovi
  - Vektorski satovi
- Međusobno isključivanje (mutal exclusion)
  - Centralizirani algoritmi
  - Decentralizirani
  - Distribuirani
  - Token ring algoritam
- Algoritmi za izbor koordinatora

Sinkronizacija satova

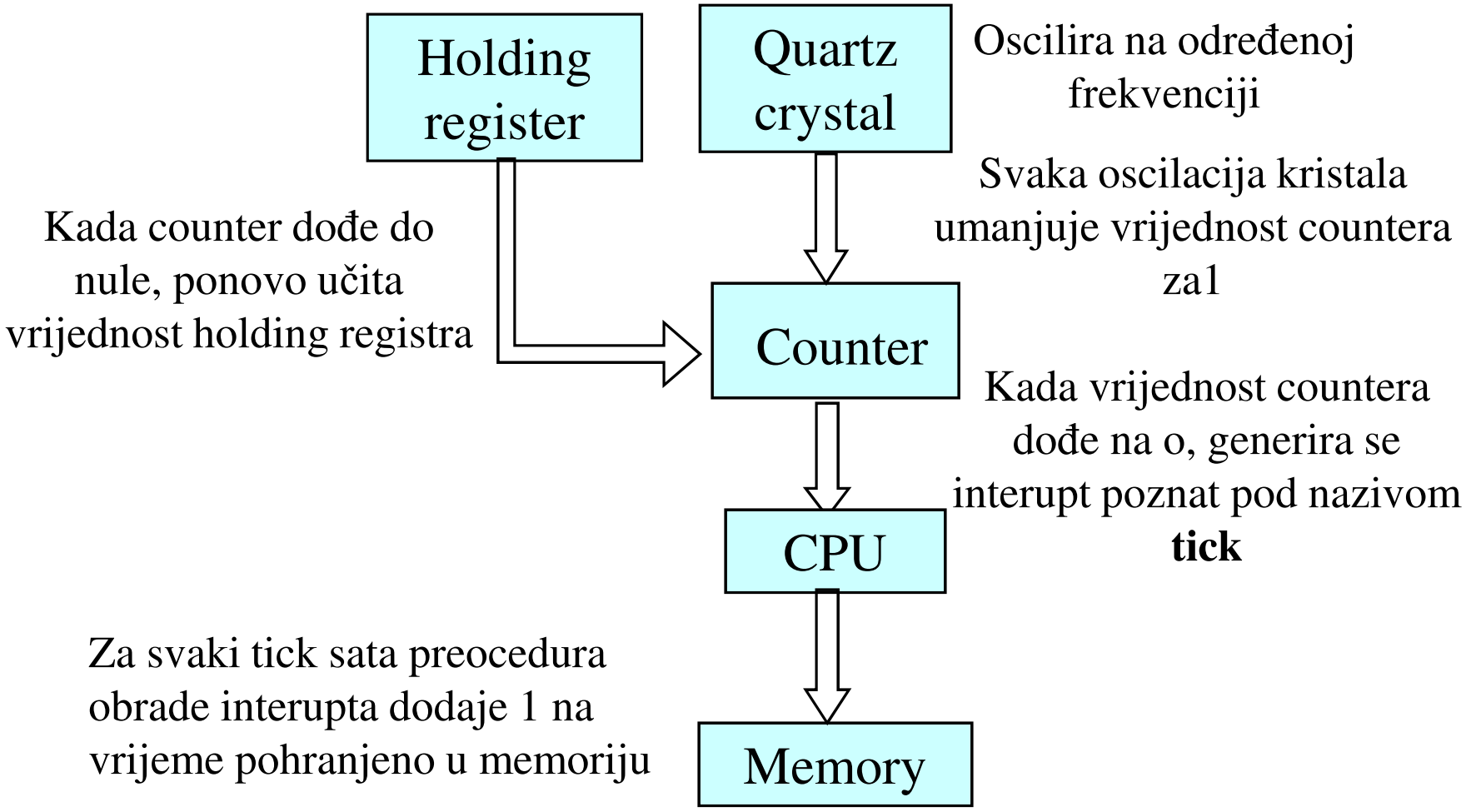
- Program *make* (kompajlira više .c datoteka u jednu .o datoteku)
- Datoteke input.c i output.o imaju naznačeno vrijeme zadnje izmjene
- Kada pozovemo make naredbu, za svaku .c datoteku provjerava se da li je vrojeme zadnje izmjene bilo nakon izmjene output.o,
  - ako je datoteka izmjenjena, treba je ponovo kompajlirati i uključiti u novi output.o
  - Ako nije, možemo koristiti strojni kod dobiven prošlim pozivom make naredbe
- A što ako su kompajler i editor na različitim računalima, i njihovi satovi nisu usklađeni



Fizički satovi

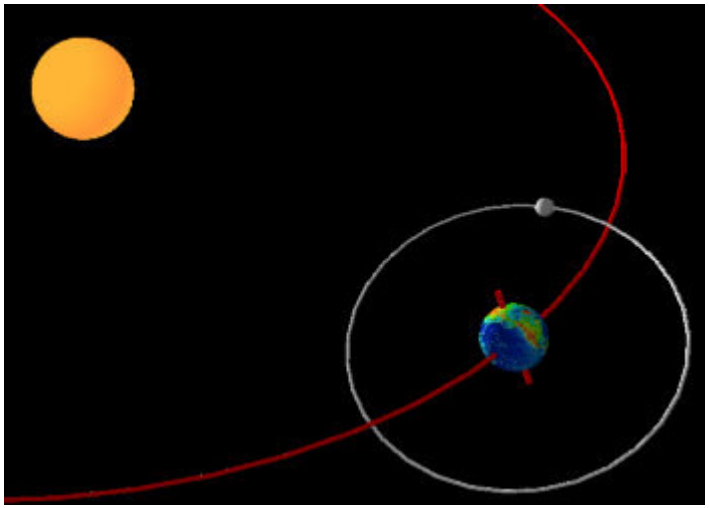
- Dilema sv. Augustina: "*What then, is time? If no one asks me, I know. If I wish to explain it to someone who asks, I know it not.*"
- Fizčki sat – generira interupte
- Softverski sat: broji interupte
  - Vrijednost sata je broj sekundi od nekog predefiniranog vremena
  - Timestamp: Za UNIX sustave od 1.1.1970 (početak gregorijanskog kalendara za Microsoft)
  - Lako se pretvori u stvarno vrijeme

Sat na računalu



Solarni dan

- Tranzit sunca: trenutak kada sunce dosegne najvišu točku na horizontu
- Vrijeme između 2 tranzita sunca = solarni dan
- Solarna sekunda=solarni dan/86400
- Ustanovljeno je da period vrtnje zemlje nije konstantan
- Prije 300 mil godina zemlja se oktretala oko 400 puta oko svoje osi za vrijeme jedne godine
- Solarni dan se produžava!
- Astronomi su izračunali srednju solarnu sekundu mjereći velik broj solarnih dana

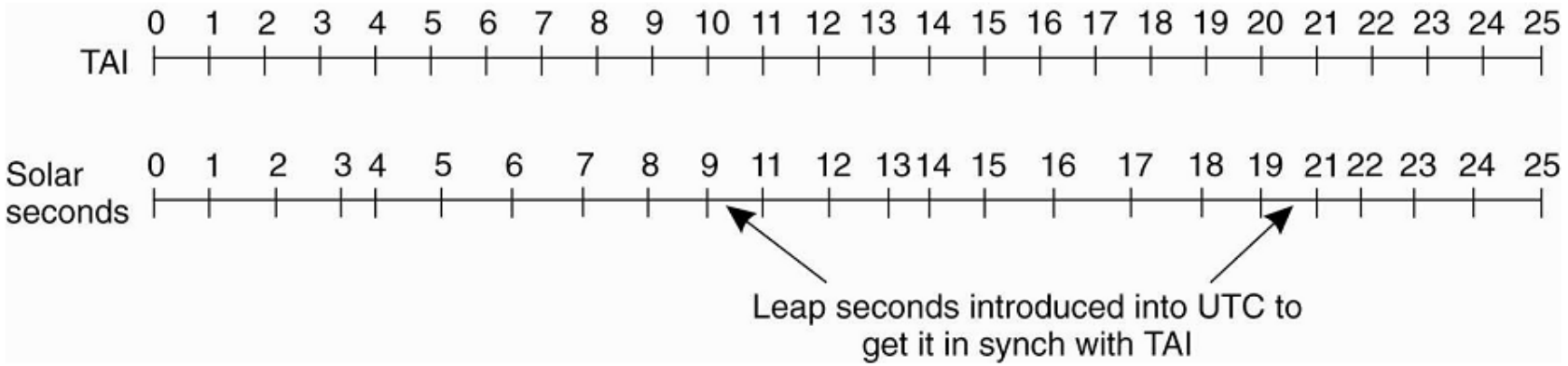


Atomski dan

- 1948 izmišljen je atomski sat koji koristi broj tranzicija atoma cezijum 133 (Cs-133)
- Fizičari su preuzeli zadatak mjerenja vremena koji su do tada obavljali astronomi
- Cs-133 napravi 9 192 631 770 tranzicija u jednoj solarnoj sekundi
- Nekoliko laboratorija na svijetu ima atomski sat
  - Svi periodički dostavljaju broj otkucaja Bureau International de l'Heure (BIH) u Parizu
  - BIH usrednjava podatke i objavljuje Međunarodno atomsko vrijeme TAI
  - TAI=broj interupta od 1.1.1958

Atomski sat

- No, dan se produžava. U ovom trenutku 86400 TAI sekundi je oko 3ms manje od solarnog dana. Ako se nešto ne poduzme podne bi se pomicalo ranije
- Kada razlika između solarnog vremena i Tai pređe 800ms, uvodi se **leap** sekunda
- UTC –Universal Coordinated Time
  - NIST Objavljuje UTC na radio stanici WWV(točnost +/-10 msec)
  - UTC usluga satelita (točnost 0.5 msec)
  - GPS (točnost 20-35 nanosec)



Algoritmi sinkronizacije vremena

- Jedna mašina prati UTC putem WWV (ili sl.) a ostale se sinkroniziraju sa njom
- Ili svaka mašina ima svoje vrijeme, a cilj je da se sve mašine prate koliko je moguće

Sinkronizacija vremena

- Označimo sa C trenutnu vrijednost sata računala, a sa t UTC vrijeme
- $C_p(t)$ je vrijednost sata računala p u trenutku t
- Frekvencija $C_p' = dC_p/dt$ treba biti šta bliža 1. Razlika $C_p' - 1$ naziva se odstupanej sata (skew)
- Offset sata u trenutku t $C_p(t)-t$ što manji
- Ako timer generira 60 interupta u sekundi, trebalo bi generiratii 215 000 interupta na sat.
- U praksi: točnost je oko 10^{-5}, što znči da sat generira 215 998 – 216 002 interupta na sat.
- Ako je iskrivljenje vremenau u različitom smjeru, razlika vremena dva računala će se povećavati s vremenom i potrebno je obaviti resinkronizaciju

Tri filozofije sinkronizacije vremena

- Pokušati zadržati sve satove sikronizirane što bliže stvarnom vremenu (UTC)
- Pokušati zadržati sve satove međusobno sinkronizirane , iako se zajedničko vrijeme razlikuje od UTC
- Poušati sinkronizirati dovoljno da se porcesi koji surađuju mogu dogovoriti o poretku događaja – ovakve satove zovemo logički satovi

Algoritmi sinkronizacije vremena

- Network Time Protocol (NTP):
  - Cilj: zadržati sve satove u sustavu sinkronizirani na UTC vrijeme (1-50 ms točnost) - nije tako dobra u WAN
  - Koristi hijerarhiju pasivnih vremenskih poslužitelja odgovaraju na upite o trenutnom točnom vremenu
- Berkeley Algoritam:
  - Cilj: zadržati sve satove u sustavu sinkronizirani jedni prema drugima (interna sinkronizacija)
  - Koristi aktivne vremenske poslužitelje koji povremeno ispituju mašine i usrednjuju zajedničko vrijeme
- Reference broadcast synchronization (RBS)
  - Cilj: zadržati sve satove u bežičnom sustav međusobno sinkronizirane

Logički satovi

- Nema potrebe za točnim vremenom niti za zajedničkim vremenom
- Kod make naredbe – jedino što se promatra jest da li se nešto dogodilo prije ili kasnije
- Dvije implementacije
  - Lamport logički sat
  - Vektorski sat

Lamportov logički sat

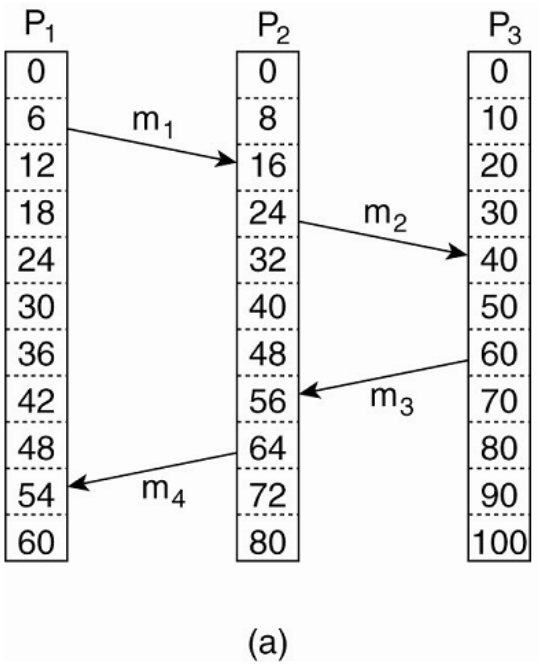
- Uvodi se relacia: događa se prije

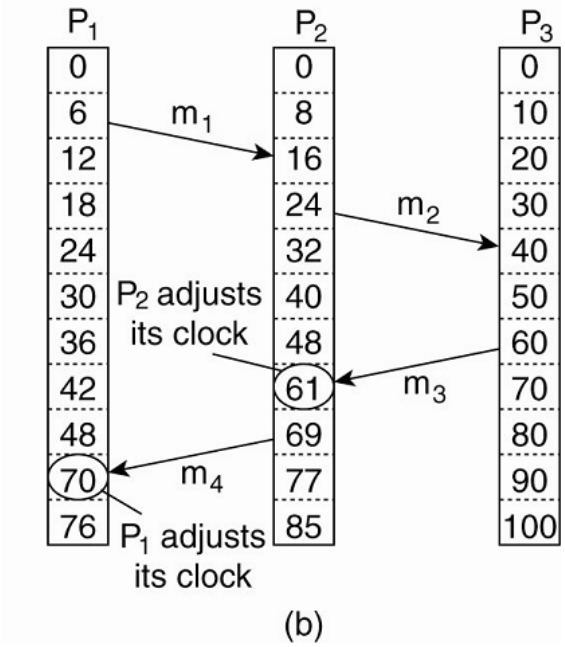
  a->b (a se događa prije b)
- Koja označava :
  - Ako su a i b događaji istog procesa i a se događa prije b tada je a->b istina
  - Ako je a događaj slanja poruke od jednog procesa, a b događaj primanja te iste poruke drugog procesa, tada je a->b istina

- Tranzitivna relacija : a->b i b->c => a->c
- Ako su x i y događaji na različitim procesima koji ne razmjenjuju poruke, direktno ni indirektno, tada x->y nije istina, niti y->x nije istina.
  - Kažemo da su x i y konkurentni događaji

Lamportov logički sat - algoritam

- Tri procesa na različitim računalima koja imaju različite brzine sata.
- Kada P1 pošalje poruku m1 procesu P2, P2 je primi u vremenu 16 koje je veće od vremena pošiljanja - ito je OK
- Kada P2 pošalje m3 u odsječku vremena 60 procesu P2, P2 je primi u odsječku vremena 56 – ova situacija nije moguća
- Proces P2 ubrza svoj sat i postavi vrijeme na za 1 više od vremena pošiljanja

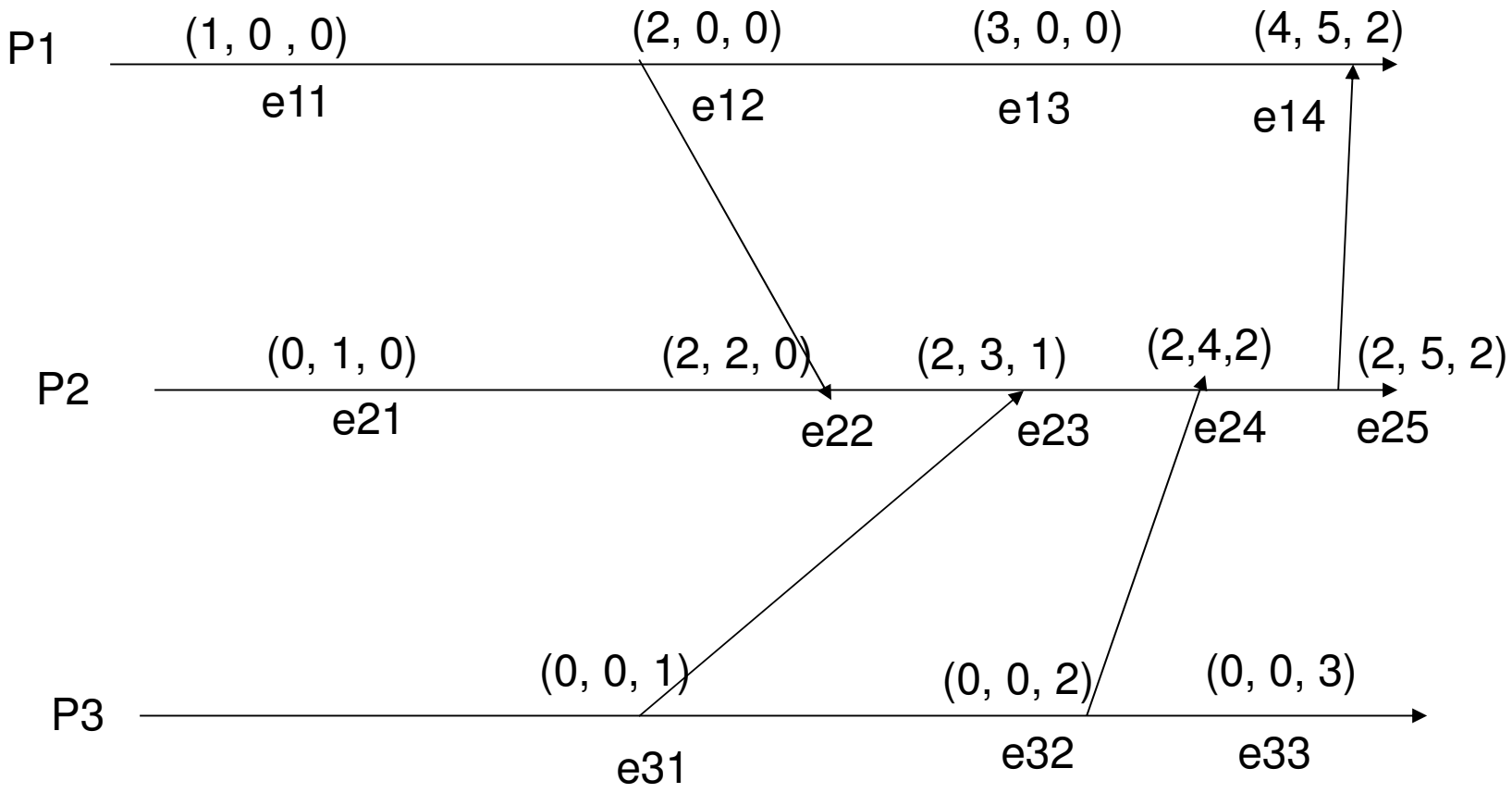




Vektorski sat

- Mana lamportova sata – ne prati uvjetno izvršavanje događaja. Ako je vrijeme jednog događaja manje od vremena drugog ne mora značiti da su događaji morali biti sinkronizirani.
- Vektorski sat: svaki proces prati vektor koji se sastoji od podataka za sve procese sustava.
- Proces ima vektor VC_i takav da
  - $VC_i[i]$ označava 'lokalno vrijeme' P_i tj.broj događaja procesa
  - $VC_i[j]$ predstavlja znanje procesa P_i o broju događaja procesa Pj
  - Sa svakom porukom, proces šalje drugom procesu i svoj vektor vremena, sa svojim vremenom uvećanim za jedan (za događaj slenje poruka
  - Proces koji primi poruku uspoređuje komponente primljenog vektora sa svojim vektorom vremena i ažurira svoj vektor na način da odabire veću vrijednost
- Na ovaj način, proces zna broj događaja na svim procesima koji su mu prethodili

Vrijednosti vektorskog sata za tri procesa



Mutual exclusion

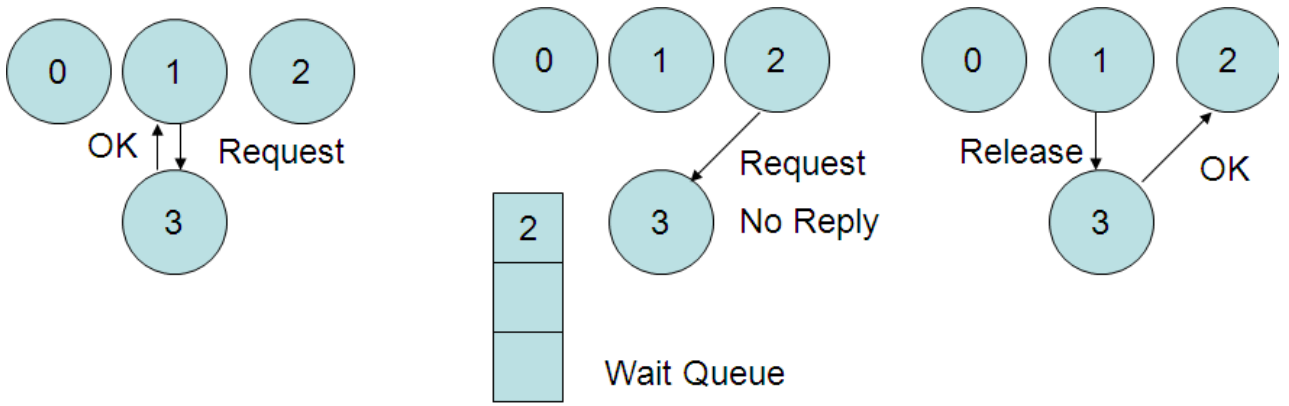
- Sinkronizacija procesa kako ne bi istovremeno pristupali istom resursu
- Ciljevi mutual exclision algoritma :
  1. Izbjeći deadlocks – ne smije se dogoditi da je skup procesa blokiran jer svi čekaju poruku od nekog drugog procesa koji je također blokiran (pile ili jaje problem)
  2. Nema “izgladnjivanja procesa “ (starvation) – ne bi se smjelo dogoditi da jedan proces čeka neodređeno na pristup resursu dok se drugi procesi poslužuju više puta
  3. Pravednost – zahtjevima se udovoljava onim redom kojim dolaze. Ovo znači da se procesi trebaju dogovoriti i oko poretka događaja. Pravednošću se izbjegava izgladnjivanje
  4. Toleriranje grašaka – dobar algoritam trebao bi se moći oporaviti od grešaka na jednom ili više procesa

Algoritmi za distribuirane mutual exclusion

- Algoritmi temeljeni na tokenu (token based)
  - Token je posebna poruka
  - Samo proces koji ima token može pristupiti reursu
  - Kada završi sa resursom, samo predaje token dalje
  - Izbjegava se starvation i deadlock
  - Problem: ako se token izgubi, posebna procedura mora kreirati novi, i osigurati se da je to jedina kopija
- Algoritmi temeljeni na dozvoli
  - Proces kojem treba pristup resursu traži dozvolu
  - Ovo se može implementirati na više načina

Centralizirani algoritam

- Simulacija mutexa u jednoprocesorskim sustavima
- Jedan proces ima ulogu koordinatora
- Kada proces želi pristuo resurse, šalje zahtjev koordinatoru
  - Ako je resurs slobodan dozvoli pristup
  - Ako resurs nije slobodan stavi ga u red čekanja



Centralizirani algoritam

- Garantira međusobno isključivanje – samo jedan proces može u jednom trenutku pristupiti resursu
- Nema “izgladnjivanja” prodesa (starvation) – pravedno se raspoređuju resursi, onim redom kojim zahtjevaju pristup
- Nema deadlock
- Tolerancija grešaka
  - Blokirajući pozivi – zahtjevatelj ne može razlikovati situacije kada se koordinator sruši od samo dugog čekanja
- Velika prednost - jednostavnost

Decentralizirani algoritam

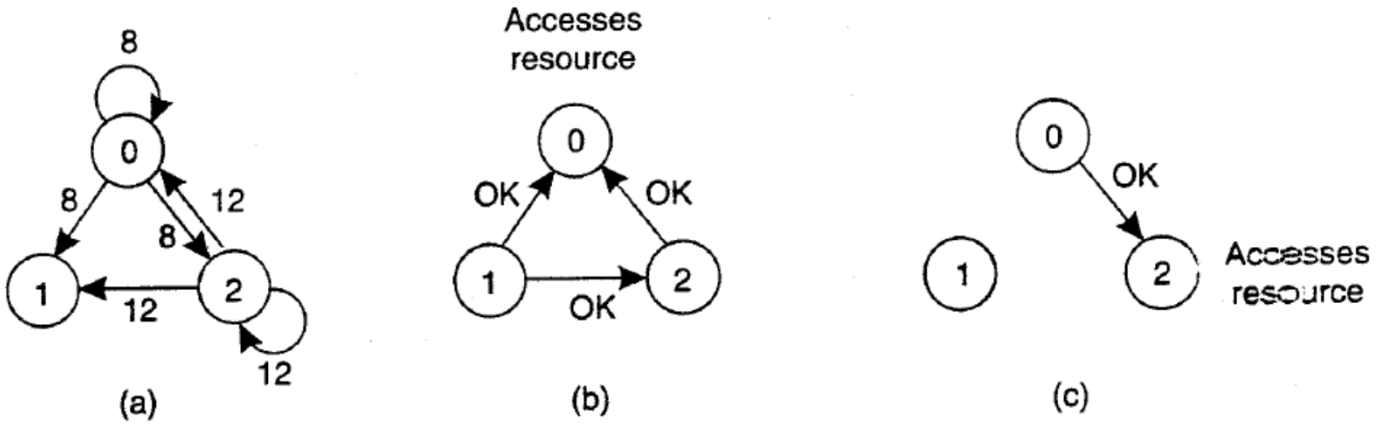
- Može se izvoditi na sustavu distribuiranih hash tablica
- Pretpostavlja se da je resurs repliciran n puta i svaka replika ima svog koordinatora
- Kada proces traži pristup, zahtjet šalje svimkoordinatorima. Dovoljno mu je da dobije m>n/2 dozvola da dobije pristup resursu
- Znatno smanjuje ranjivost centraliziranog pristupa, ali je potpuno ne uklanja
- Kolika je vjerojatnost da će se više od n/2 koordinatora srušiti?

Distribuirani algoritam

- Kada proces želi pristup reusrsu šalej svim procesima zahtjev koji sadrži:
  - ime resursa,
  - broj procesa i
  - lokalno logičko vrijeme
- Kada proces primi poruku mogu se desiti 3 situacija:
  - Proces ne pristupa niti želi pristupati navedenom resursu, tada šalje OK poruku pošiljatelju
  - Proces već ima pristup navedenom resursu – tada jednostavno ne odgovara dok god ne oslobodi resurs
  - Proces također želi pristupatu resursu – tada uspoređuje svoje logičko vrijeme sa logičkim vremenom pošiljatelja. Ako je vrijeme pošiljatelja manje, odgovara sa OK, ako je veće, tada blokira pošiljatelja i stavlja u red čekanja

Distribuirani algoritam

- Proces šalje zahtjev i čeka. Kada mu svi ostali procesi dozvole pristup porukom OK pristupa resursu
- Kada proces završi rad sa resursom, šalje OK poruku svim procesima u svom redu čekana i briše ih iz reda
- Ako više procesa simultano traže pristup, pristup dobiva prvo onaj sa manjim logičkim vremenom

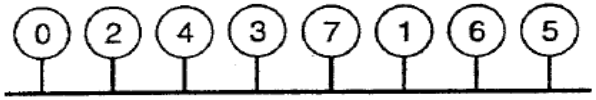


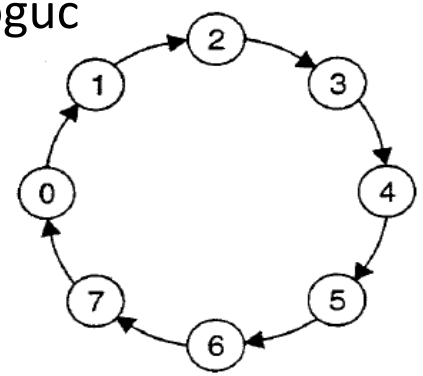
Distribuirani algoritam

- Nema deadlocka ni izgladnjivanja procesa
- Treba slanje 2*(n-1) poruka (n=broj procesa)
- Ali – pogreška jednog procesa prouzrokovat će pad cijelog sustava jer će proces čekati na odgovor i neće pristupati resursu
  - Ovo se može izbjeći uvođenjem novih poruka
- Svaki proces mora poznavati sve procese iz grupe ili se može koristiti multikastiranje
- Sporiji, složeniji, zahtjevniji i manje robustan od centraliziranog pristupa
- Ipak – pokazuje da se i mutual exclusion može izvesti distribuirano

Token-ring algoritam

- Procesi povezani sabirničkom mrežom
- Odredi se poredak procesa (prema ip adresi ili broju procesa)
- Svaki proces zna svog sljedbenika
- Inicijalno se token predaje procesu 0 a zatim svaki proces svom sljedbeniku
- Kada primi token proces provjeri da li mu navedeni resurs treba ili ne
  - Ako treba, drži token i pristupa mu
  - Ako ne treba prosljeđuje token sljedećem procesu
- Mogući problemi: ako se token izgubi – teško je detektirati da je izgubljen
- Ako se jedan proces sruši oporavak od greške je moguć





Usporedba 4 algoritma

Algorithm	Messages per entry/exit	Delay before entry (in message times)	Problems
Centralized	3	2	Coordinator crash
Decentralized	3mk, k = 1,2,...	2 m	Starvation, low efficiency
Distributed	2 (n – 1)	2 (n – 1)	Crash of any process
Token ring	1 to ∞	0 to n – 1	Lost token, process crash

Izbor koordinatora

- Mnogi distribuirani algoritmi zahtjevaju da se jedan proces postavi kao vođa, ili koordinator
- Ako se procesi ne razlikuju po nekoj specifičnoj značajki potrebno je da jedan od njih preuzme ulogu, ali nije bitno koji
- Pretpostavit ćemo da dvaki proces ima jedinstveni identifikator
- Koordinator treba biti proces sa najvećim brojem

Tradicionalni algoritmi

- Bully algoritam
- Ring algoritam

- Bully algoritam: kada neki proces primjeti da nema koordinatora šalje svim procesima sa brojem većim od svog ELECTION poruku
- Ako nijedan ne odgovori tada preuzima koordinaciju
- Ako jedan od procesa odgovori – on preuzima posao odabira koordinatora

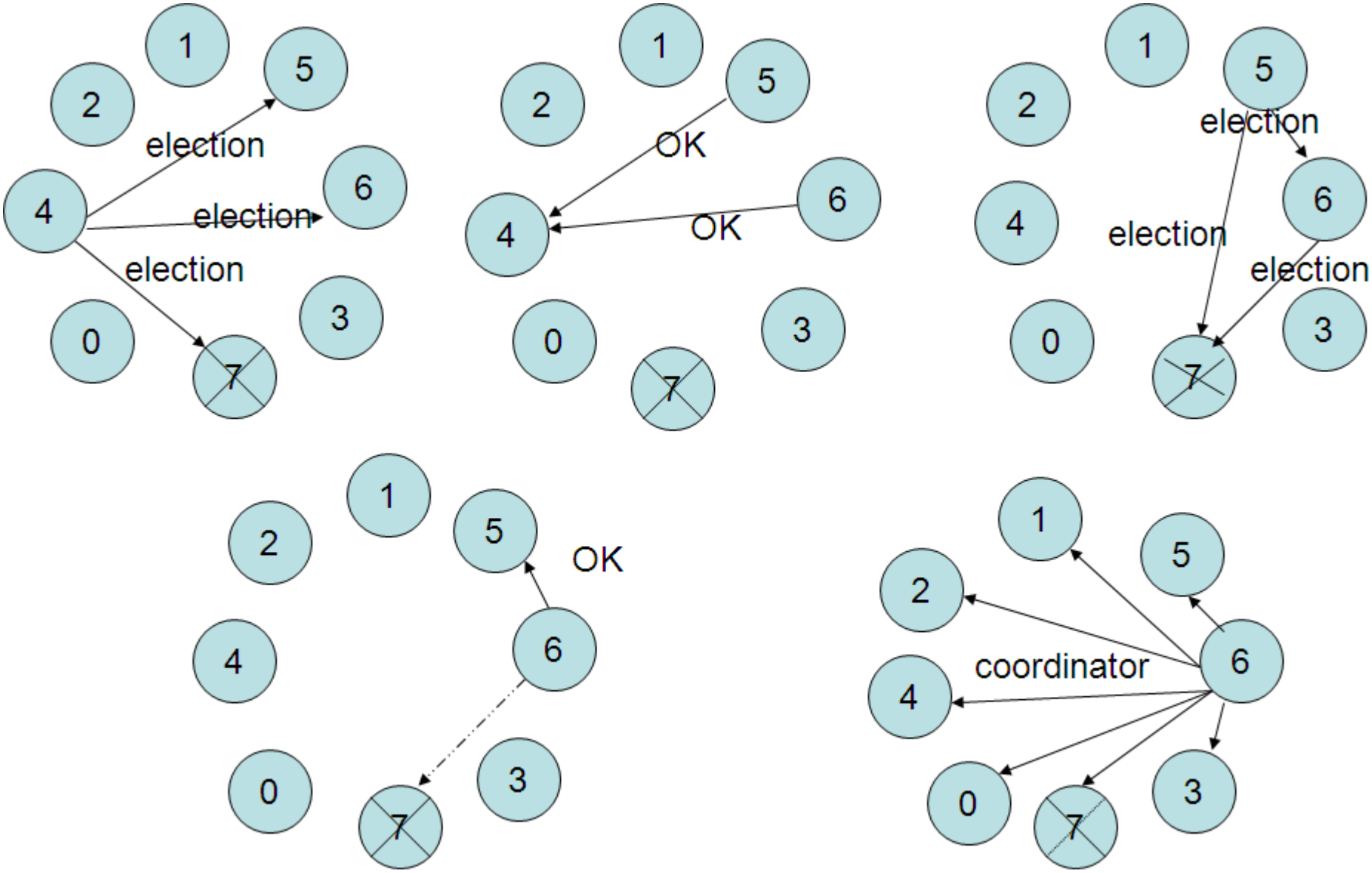
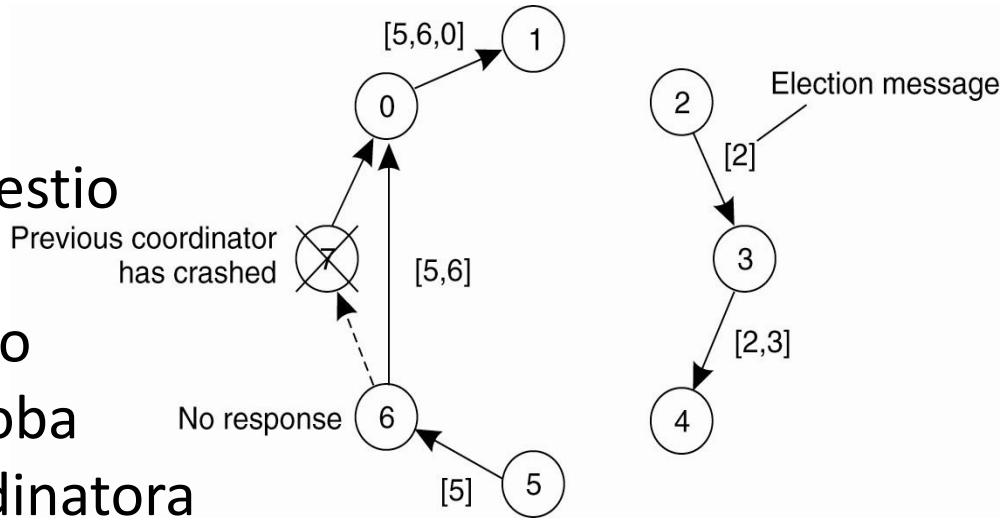

1
2
5
election
4
election
6
election
0
7
3
1
2
5
OK
4
OK
6
0
7
3
1
2
5
election
4
election
6
election
0
7
3
1
2
5
OK
4
6
0
7
3
1
2
5
4
coordinator
6
0
7
3

Figure 6-20

Ring algoritam

- Procesi trebaju biti logički poredani u prsten tako da svaki proces zna tko mu je sljedbenik
- Kada se koordinator sruši, proces koji to primjeti šalje ELECTION poruku sljedbeniku
- Ako slejdbenik ne odgovara šalje ga sljedećem procesu
- Svaki proces dodaje svoj broj na poruku.
- Kada poruka dođe natrag do pošiljatelja on će to primjetiti po tome što poruka sadrži njegov broj

  procesa, pretvara je u COORDINATOR poruku i šalje kroz prsten kako bi sve obavjestio tko je novi vođa
- Ako dva procesa istovremeno primjete pad koordinatora, oba će pronaći istog novog koordinatora



Izbor u bežičnim mrežama

- Topologija i mreža se mijenjaju
- Svaki čvor šalje poruke svojim najbližim susjedima
- Odabire se najbolji kandidat za koordinatora

Zaključak

- "Sinkronizacija je ... raditi pravu stvar u pravo vrijeme.“
- Sinkronizacija u distribuiranim sustavima vezana je za komunikaciju.
- Kompliciran zbog nedostatka globalnog sata i zajedničke memorije.
- Logički satovi podržavaju globalni poredak događaja.
- Distribuirani Mutex: važna klasa algoritama sinkronizacije.
- Ponekad je potrebno izabrati koordinatora – algoritmi za izbor.