**Konkurentnost: uzajamno blokiranje i gladovanje**

**Uzajamno blokiranje**

* Šta je uzajamno blokiranje?
  + Trajno uzajmno blokiranje skupa procesa koji se nadmeću za resurse sistema ili međusobno komuniciraju
* Skup procesa je uzajamno blokiran kada je svaki proces iz ovog skupa blokiran čekajući na događaj koji može aktivirati samo neki od procesa iz tog skupa
  + Uključuje procese koji se nadmeću za resurse iz istog skupa
* Ne postoji efikasno rešenje uzajamnog blokiranja

**Kategorije resursa**

* Pod resursom ćemo podrazumevati bilo koji objekat koji se može dobiti procesima
* Postoje dve kategorije resursa:
  + Ponovo upotrebljivi resursi
    - Mogu istovremeno biti korišćeni od strane samo jednog procesa i nakon njegovog korišćenja upotrebljivi su za drugi proces
  + Potrošni resursi
    - To su resursi koji se mogu kreirati (proizvesti) i uništiti (potrošiti)

**Ponovo upotrebljivi resursi**

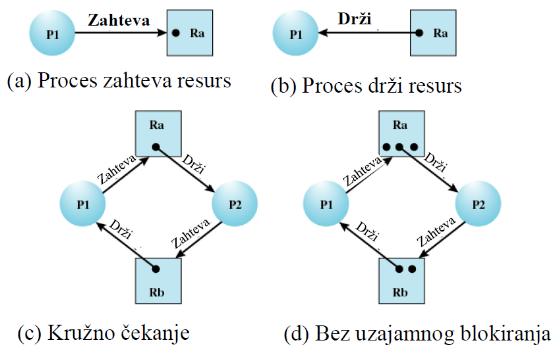
* Procesi zauzimaju instance resursa, koriste ih i kasnije oslobađaju kako bi ih koristili drugi procesi
* Primer ponovo upotrebljivih resursa :
  + Procesori
  + Glavna i sekundarna memorija
  + U/I uređaji
  + U/I kanali
  + Strukture podataka, poput datoteka, baza podataka, semafora
* Uzajamno blokiranje nataje kad svaki proces drži resurs i zahteva drugi

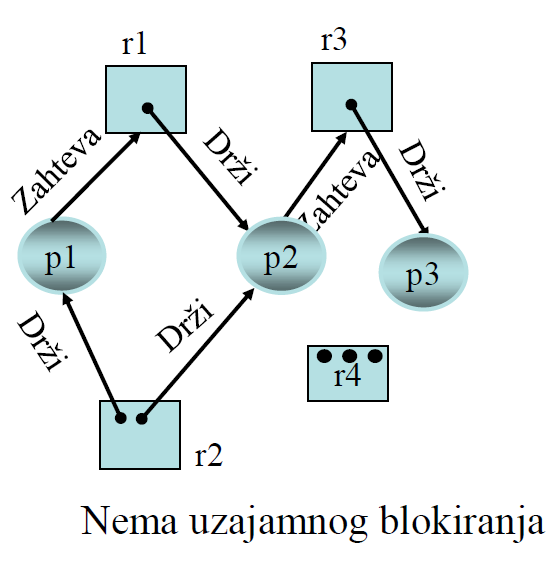
**Potrošni resursi**

* Kada proces potrošač zahteva takav resurs i dobije ga, resurs prestaje da postoji
* Ne postoji ograničenje u broju takvih resursa koje proces može napraviti
* Primer potrošnih resursa :
  + Prekidi
  + Signali
  + Poruke
  + Informacije u U/I baferima
* Uzajamno blokiranje može nastati ako je operacija Receive poruke blokirajuća
* Retke kombinacije događaja mogu prouzrokovati uzajamno blokiranje

**Model sistema**

* Sistem se sastoji od određenog broja resursa koji se dodeljuju procesima
* Resursi su razvrstani u više tipova
  + Npr. štampači, memorija...
* Svaki tip ima više identičnih instanci
  + Ako proces zahteva neki tip resursa, OS mu može dodeliti bilo koju instancu tog tipa resursa
* Proces može zahtevati proizvoljan broj resursa da bi obavio zadatak
  + Broj zahtevanih resursa ne sme prelaziti broj raspoloživih resursa

**Graf dodele resursa**

* Graf dodele resursa je orijentisani graf G = (V, E), kojim se modelira dodela resursa
* Skup čvorova V se deli na dva podskupa:
  + Čvorovi procesa – Skup procesa u sistemu P = (P1, P2, ..., Pn)
  + Čvorovi resursa – Skup tipova resursa R = (R1, R2, ..., Rm)
* Skup grana E se deli na dva podskupa:
  + Grane zahteva (zahteva, traži) – Skup grana zahteva – par (Pi, Rj)
  + Grane dodele (drži) – Skup grana dodele – to je uređeni par (Rj, Pi)
* Grana zahteva (Pi, Rj) modelira pojavu da proces Pi zahteva instancu resursa Rj
* Grana dodele (Rj, Pi) modelira pojavu da je instanca resursa tipa Rj dodeljena procesu Pi, tj. da proces Pi drži resurs tipa Rj

**Primer grafa dodele resursa**

* Procesi P = (p1, p2, p3)
* Resursi R = (r1, r2, r3, r4)
* Instance resursa
  + R1 ima 1 instancu
  + R2 ima 2 instance
  + R3 ima 1 instancu
  + R4 ima 3 instace
* Stanje:
  + P1 drži 1 instancu resursa tipa r2 i zahteva 1 instancu resursa tipa r1
  + P2 drži r1 i r2 i zahteva r3
  + P3 drži r3

**Modeliranje uzajamnog blokiranja pomoću grafa dodele resursa**

* Ako graf dodele resursa nema cikluse, tada nijedan proces u sistemu nije u uzajamnom blokiranju
* Ako graf dodele resursa ima cikluse, tada može postojati uzajamno blokiranje
* Ako svaki tip resursa ima po 1 instancu:
  + Ciklus u grafu dodele resursa je potreban i dovoljan uslov za postojanje uzajamnog blokiranja
* Ako svaki tip resursa ima više instanci:
  + Tada postojanje ciklusa u grafu dodele resursa je potreban, ali nije dovoljan uslov za pojavu uzajmnog blokiranja.

**Uzajamno blokiranje**

* Uslovi za moguće uzajamno blokiranje (potrebni)

1. Uzajamno isključivanje (Mutual exclusion)
   1. U datom trenutku samo jedan proces može koristiti resurs
2. Drži i čekaj (Hold-and-wait)
   1. Proces drži dodeljene resurse dok čeka dodeljivanje ostalih resursa
3. Bez prekidanja (No preempiton)
   1. Nijedan resurs se ne može nasilno oduzeti procesu koji ga drži

* Uzajamno blokiranje nastaje kad su ispunjena prethodna tri uslova i:

1. Kružno čekanje (Circular wait)
   1. Postoji zatvoreni “lanac” procesa, takav da svaki proces drži bar jedan resurs koji je potreban sledećem procesu u “lancu”
   2. Ukoliko ima više instanci resursa ne mora da dođe do uzajamnog blokiranja

**Potrebni i dovoljni uslovi za pojavu uzajamnog blokiranja**

* Ako su prisutna samo prva 3 uslova postoji mogućnost za pojavu uzajamnog lokiranja
  + Ovi uslovi se odnose na politiku dodele
* Da bi nastalo uzajamno blokiranje potrebno je da istovremeno budu prisutna sva 4 uslova
  + Kružno čekanje je okolnost koja zavisi od sekvence dodele i oslobađanja resursa od strane uključenih procesa

**Metode za upravljanje uzajmnim blokiranjem**

* Sprečavanje uzajamnog blokiranja
  + Onemogućiti nastanak nekog od 4 uslova za pojavu uzajamnog blokiranja
* Izbegavanje uzajamnog blokiranja
  + Donositi odgovarajuće odluke o dodeli resursa u skladu sa trenutnim stanjem dodele resursa
* Otkrivanje uzajamnog blokiranja i oporavak
  + Udovoljiti zahtevu za resursom kada je to moguće, ali periodično proveriti da li je uzajamno blokiranje prisutno i tada oporaviti sistem od uzajamnog blokiranja

**Sprečavanje uzajamnog blokiranja**

* OS se projektuje tako da isključi mogućnost pojave uzajamnog blokiranja
* Dve klase metoda:
  + Indirektne metode sprečavanja uzajamnog blokiranja
    - Sprečavanje pojave jednog od 3 potrebna uslova
  + Direktne metode sprečavanja uzajamnog blokiranja
    - Sprečavanje pojave kružnog čekanja
* Ove metode vode neefikasnom korišćenju resursa i neefikasnom izvršenju procesa

**Sprečavanje uslova “Uzajamno isključivanje”**

* Ako nema resursa koji se dodeljuju ekskluzivno samo jednom procesi, nema uzajmnog blokiranja
* To nije moguće – uzajamno isključivanje se mora zadržati za nedeljive resurse
* Primer nedeljivog resursa: printer
* Primer deljivog resusa: read-only datoteka

**Sprečavanje uslova “Drži i čekaj”**

* Treba garanovati da kad proces zahteva neki resurs, ne sme držati nijedan drugi resurs
* Može se koristiti pristup kojim će se zahtevati od svakog procesa da pre izvršenja zatraži sve potrebne resurse
  + Proces se blokira dok mu se svi traženi resursi dodele
  + Procesi mogu dugo čekati udovoljenje svih zahteva za resurse
  + Resursi dodeljeni procesu mogu ostati dugo neiskorišćeni iako bi mogao da ih koristi neki drugi proces
  + Proces bi morao da zna sve svoje zahteve za resursima
* Alternativa je da se korsiti pristup po kome proces mora osloboditi sve resurse pre nego što zahteva dodatni resurs
* Oba pristupa imaju dva nedostatka:
  + Iskorišćenje resursa je malo kako je prikazano kod prvog pristupa
  + Gladovanje procesa – proces koji zahteva popularne resurse može beskonačno dugo čekati

**Sprečavanje uslova “Bez prekidanja”**

* Pristup 1: Ako proces drži neke resurse i zahteva drugi resurs koji mu se ne može odmah dodeliti (tj. proces mora da čeka), tada svi resursi koje proces drži se “otpuštaju” (oslobađaju) i proces će ponovo da ih zahteva sa dodatnim resursima
* Pristup 2: Kada proces P zahteva resurse koji su dodeljeni nekom drugom procesu Q koji takođe čeka na dodelu koji takođe čeka na dodelu, oduzimaju se resursi od procesa Q, koji čeka i dodeljuju se procesu P. Problem ako su procesi istog prioriteta
* Kad god proces mora da oslobodi resurs koji je već koristio, stanje to resursa se mora pamtiti za kasniji nastavak procesa
* Stoga, ovaj protokol je praktičan samo za resurse čije se stanje može lako pamtiti i restaurirati, kao što je procesor

**Sprečavanje uslova “Kružno čekanje”**

* Strategija za sprečavanje kružnog čekanja
  + Definisati striktno linearno uređenje O() za sve tipove resursa
  + Svakom tipu resursa se dodeljuje jedinstven ceo broj tako da se resursi mogu urediti. Npr:
    - R1: jedinica trake O(R1) = 2
    - R2: jedinica diska O(R2) = 4
    - R3: štampači O(R3) = 7
  + Proces može zahevati resurse samo u rastućem redosledu njihovog uređenja
    - Proces inicijalno traži određeni broj instanci nekog tipa resursa, Ri
    - Ako proces zahteva više insanci istog tipa resursa mora ih tražiti jednim zahtevom
    - Ako je procesu dodeljen resurs Ri, može zahtevati instance resursa Rj ako i samo ako je O(Rj) > O(Ri)
  + Sprečavanje kružnog čekanja može biti neefikasno, usporava procese i npotrebno odbija pristup resursima

**Izbegavanje uzajamnog blokiranja**

* Pažljivom dodelom resursa može se izbeći uzajamno blokiranje
  + Kada proces traži resurse, sistem proverava da li je bezbedno ili nebezbedno dodeliti tražene resurse
  + Ako je bezbedno, sistem može doneti odluku o dodeli resursa
  + Ako nije bezbedno, sistem odbija dodelu resursa
* Pitanje je da li postoji algoritam koji omogućava sistemu da uvek napravi pravi izbor i da se na taj način uvek može izbeći uzajamno blokiranje
  + Potrebne su unapred informacije o tome kako će procesi zahtevati resurse
* Ova metoda se bazira na konceptu bezbednih stanja (safe states)

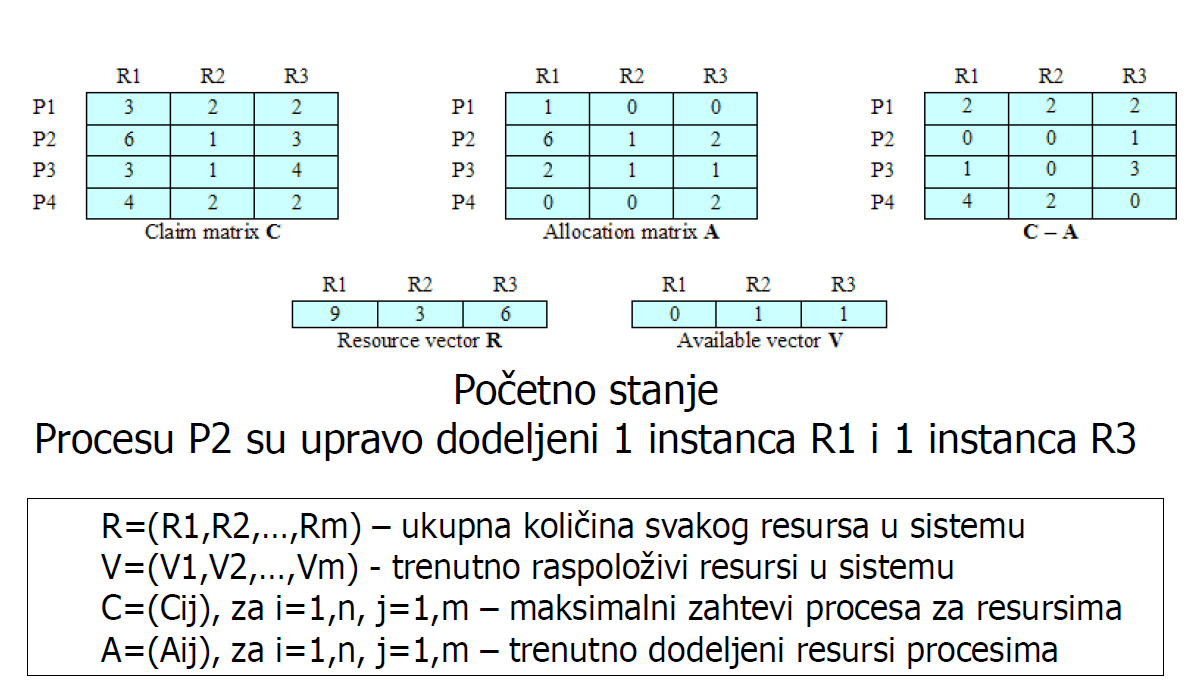
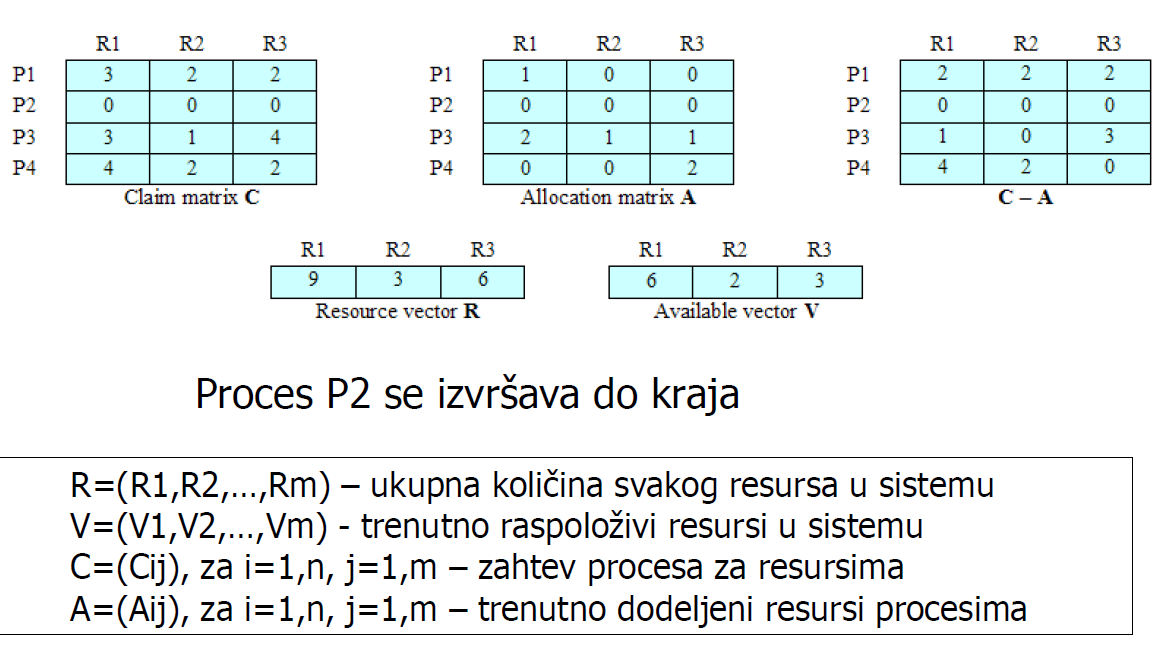
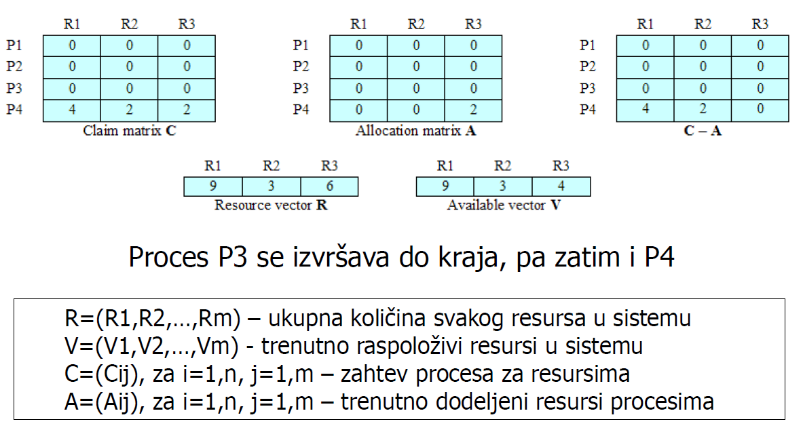
**Stanje sistema**

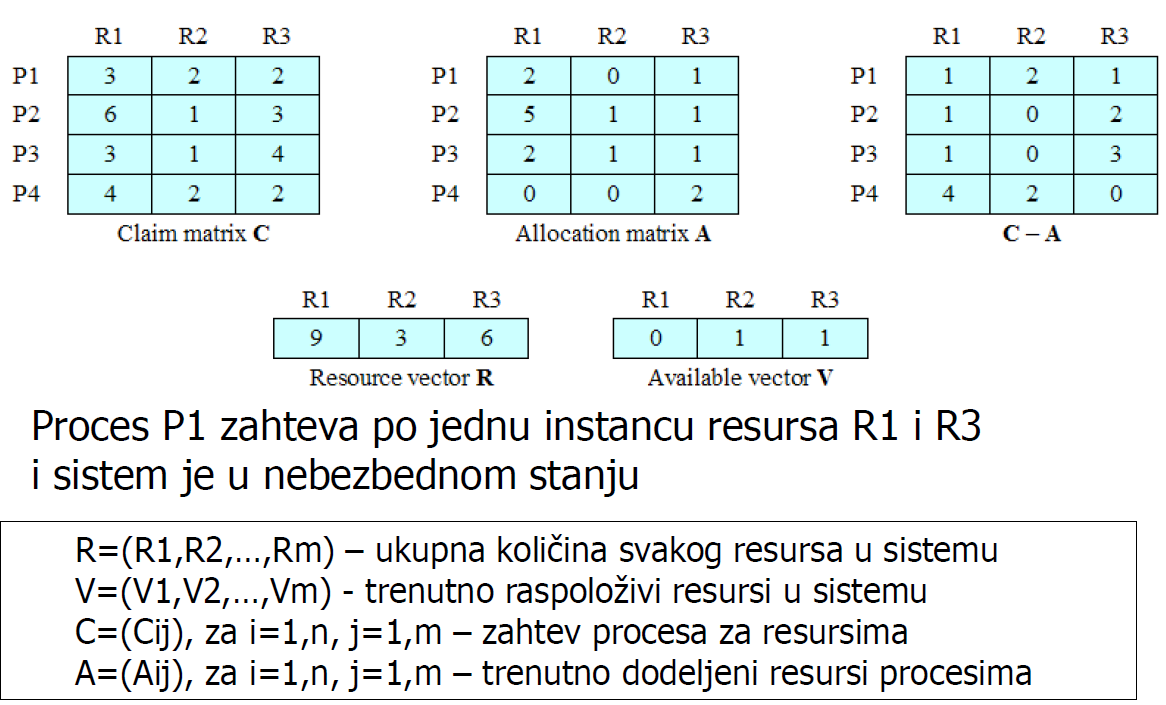
* Stanje sistema oslikava trenutnu dodelu resursa procesima
  + R = (R1, R2, ..., Rm) – Resources – ukupna broj svakog resursa u sistemu
  + V = (V1, V2, ..., Vm) – Available – trenutno raspoloživi resursi u sistemu
  + C = ( Cij ), i = 1, n ; j = 1, m – Claim – max zahtevi procesa za resursima
  + A = ( Aij ), i = 1, n; j = 1, m – Allocation – trenutno dodeljeni resursi procesima (vrste predstavljaju procese, a kolone resurse)
* Važi sledeće:

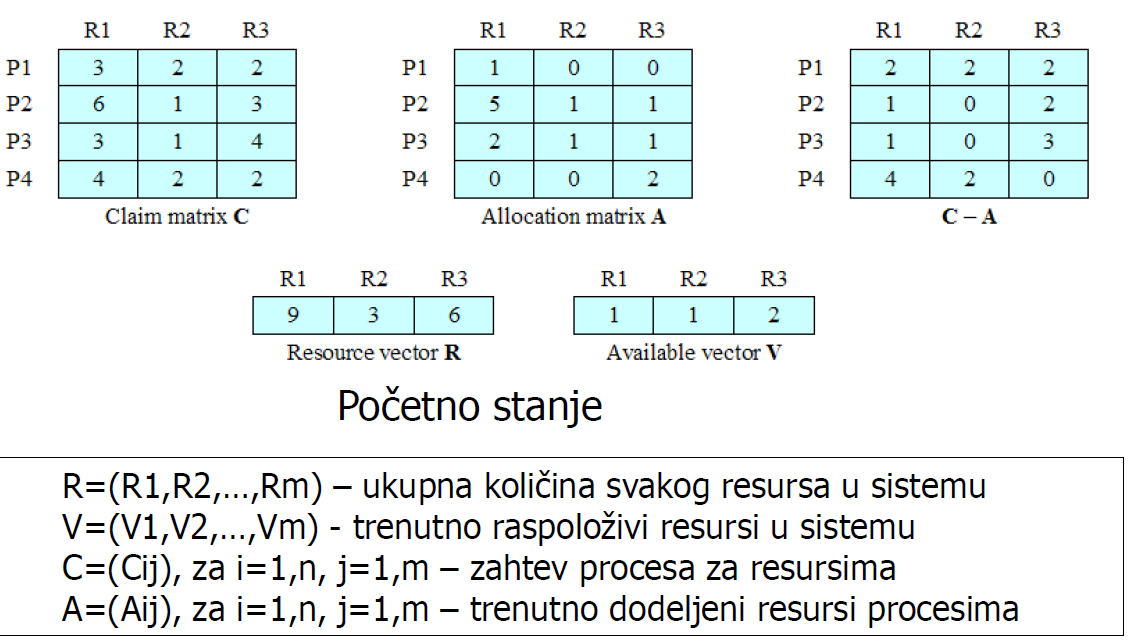
1. Svi resursi su ili raspoloživi ili dodeljeni
   1. za svako j = 1, m
2. Nijedan proces ne može zahtevati više od ukupne količine resursa u sistemu
   1. za svako i, j
3. Nijednom procesu nije dodeljeno više resursa od njegovih maksimalnih zahteva
   1. za svako i, j

**Bezbedno stanje**

* Stanje je bezbedno ako sistem može dodeliti resurse svakom procesu (do max broja resursa) u nekom redosledu i da pri tom izbegne uzajamno blokiranje
* Sistem je u bezbednom stanju samo ako postoji bezbedna sekvenca procesa
* Sekvenca procesa <P0, P1, P2, ..., Pn> je bezbedna sekvenca za tekuće stanje dodele resursa ako za svaki Pi, resursi koje proces Pi može još tražiti mogu biti zadovoljeni trenutno raspoloživim resursima i resursima koje drže svi procesi Pj, j < i, tj. Uslov koji se mora zadovoljiti za proces Pi je
  + za svako j
* Za svaki proces Pi i za svaki proces Rj mora biti ispunjen uslov:
  + Broj zahtevanih – Broj dodeljenih <= Broj raspoloživih resursa

**Utvrđivanje bezbednog stanja primer**

**Utvrđivanje nebezbednog stanja**



**Bankarov algoritam**

* Predložio ga je Dijkstra (1965)
* Ime je dobio po tome što se princip koristi u bankama pri dodeli raspoloživog novca klijentima kako se ne bi došlo u situaciju da više ne mogu biti zadovoljene potrebe klijenata
* Proverava da li dodela resursa vodi u bezbedno stanje
  + Ako DA – vrši se dodela resursa
  + Ako NE – blokirati proces sve dok nije bezbedno dodeliti resurs (uslužiti zahtev)

**Bankarov algoritam (pseudo kod)**

* Potrebne strukture podataka
  + R – vektor ukupnog broja svakog resursa u sistemu
  + V – vektor trenutno raspoloživih resursa
  + A – matrica resursa trenutno dodeljenih procesima
  + C – matrica maksimalnih zahteva procesa za resursima
    - C[i, \*] vetkor maksimalnih zahteva procesa Pi
    - C[i, j] = k znači da proces Pi traži maksimalno k instanci resursa Rj
  + Q – vektor dodatnih zahteva procesa Pi za resursima (it’s a matrix actually)
    - Važi: Qi = Ci – Ai
* Algoritam:

1. **If (A[i, \*] + Q[\*] > C[i, \*] )**

**then greška “proces traži više od maksimuma”**

1. **If (Q[\*] > V[\*])**

**Then traženi resursi nisu raspoloživi i Pi mora da se blokira**

1. **Simulita se dodela zahtevanih resursa Q[i, \*] procesu Pi i menja stanje dodele resursa:**

**V[i, \*] = V[i, \*] – Q[\*] // raspoloživi resursi**

**A[i, \*] = A[i, \*] + Q[\*] // dodeljeni resursi**

1. **Proverava se da li je novo stanje bezbedno (koristi se algoritam ispitivanja bezbednosti)**
2. **IF(bezbedno stanje)**

**Then Proces Pi stvarno dobija resurse**

**Else**

**Then proces Pi se blokira i čeka da mu se zahtevani resursi Q[\*] dodele, a staro stanje se restaurira;**

**Algoritam ispitivanja bezbednosti (pseudo kod)**

* Algoritam ispitivanja bezbednosti proverava da li je sistem u bezbednom stanju

1. **Inicjalizacija**
   1. **TrenutnoRaspoloživi = V**
   2. **Rest = {svi procesi}**
2. **Ponavljati korake 3 i 4**
3. **Naći Pi u Rest za koji važi**
   1. **C[i, \*] – A[i, \*] <= TrenutnoRaspoloživi**
4. **Ako takav Pi postoji, tada**
   1. **TrenutnoRaspoloživi += A[i, \*]**
   2. **Rest -= {Pi}**
5. **Ako je Rest == {}, tada**
   1. **Sistem je u bezbednom stanju**
6. **Sistem nije u bezbednom stanju**

**Otkrivanje i oporavak**

* Sistem ne pokušava da spreči pojavljivanje uzajamnog blokiranja
  + Strategije sprečavanja i izbegavanja uzajamnog blokiranja ograničavaju pristup resursima i nameću ograničenja procesima
* Umesto toga, sistem dozvoljava da se uzajamno blokiranje pojavi, a onda uključuje mehanizme da otkirje uzajamno blokiranje i da oporavi sistem, tj. Da raskine uzajamno blokiranje
* Provera uzajamnog blokiranja može se vršiti:

1. **Pri svakom zahtevu za resursima, ili**
2. **Povremeno – zavisno od toga kolika je verovatnoća pojave uzajamnog blokiranja**

* Prva varijanta je bolja u tome što se ranije otkriva uzajamno blokiranje i algoritam je jednostavan, ali česte provere troše značajno procesorsko vreme

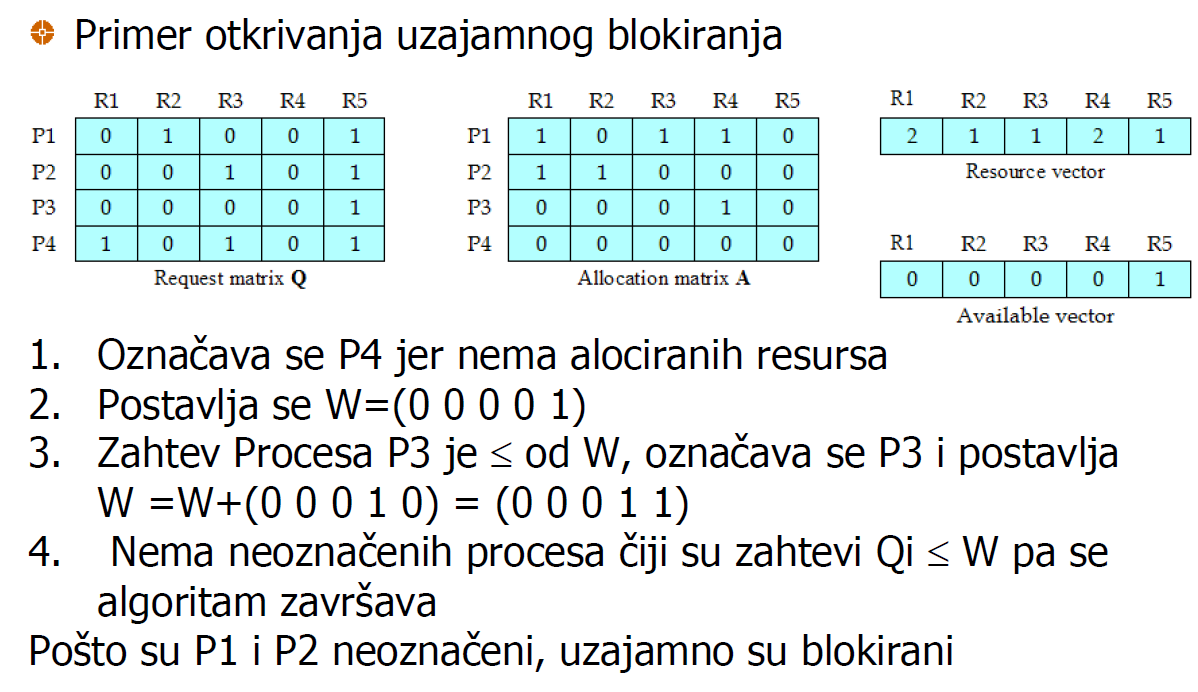
**Algoritam otkrivanja uzajanog blokiranja**

* Algoritam označava sve porcese koji nisu uzajamno blokirani
  + Inicijalno svi procesi su neoznačeni

1. Označiti svaki proces čija vrsta u matrici dodele A sadrži sve nule
2. Inicijalizovati privremeni vektor W, koji je jednak vektoru raspoloživih resursa V
3. U matrici zahteva Q naći neoznačeni proces Pi koji zadovoljava uslov Qi < W, k = 1, m. Ako takav proces ne postoji, tada završiti algoritam
4. Ako takav proces postoji, označiti proces Pi i vektoru W dodati i-tu vrstu matrice A (Wk = Wk + Aik , k = 1, m) i vratiti se na korak 3
5. Uzajamno blokiranje postoji ako i samo ako postoje neoznačeni procesi po završetku algoritma

* U koraku 3 se traži proces koji može da se izvršava – njegovim zahtevima za resursima sistem može odgovoriti.
* Takav proces se izvršava i nakon toga oslobađa resurse koje drži
* Resursi se vraćaju u raspoložive, dodaju se u W (korak 4) – Proces se označava kao obrađen
* Uzajamno blokiranje postoji ako i samo ako postoje neoznačeni procesi po završetku algoritma

**Otkrivanje uzajamnog blokiranja**



**Oprovaka iz uzajamnog blokiranja**

1. Prekinuti sve procese koji su uzajamno blokirani
2. Vratiti sve uzajamno blokirane procese do neke prethodno definisane kontrolne tačke i tada restartovati sve procese
   1. Ovo zahteva postojanje rollback i restart mehanizma ugrađenih u OS
3. Sukcesivno prekidati procese koji su uzajamno blokirani sve dok postoji uzajamno blokiranje
   1. Redosled selektovanja procesa koji će biti prekinuti se zasniva na kriterijumu minimalnih troškova. Bira se proces koji ima:
      1. Najmanje korišćeno procesorsko vreme, najmanju količinu generisanog izlaza, najduže procenjeno presotalo vreme, najmanje alociranih resursa, najamnji prioritet itd
   2. Nakon svakog prekidanja treba proveriti da li još uvek postoji uzajamno blokiranje algoritmom otkrivanja uzajamnog blokiranja
4. Selektivno oduzeti resurse procesima sve dok postoji uzajamno blokiranje (korišćenjem algoritma detekcije posle svakog oduzimanja)
   1. Proces kome su oduzeti resursi mora se vratiti do taške izvršenja pre dodele resursa.

**Strategije rešavanja uzajamnog blokiranja – rezime**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Approach** | **Resource Allocation Policy** | **Different Schemes** | **Major Advantages** | **Major Disadvantages** |
| Prevention | Conservative, undercommits resources | Requesting all resources at once | Works well for processes that perform a single burst of activity | Inefficient |
| Delays process initiation |
| No preemption necessary | Future resource requirements must be known by processes |
| Preemption | Convenient when applied to resources whose state can be saved and restored easily | Preempts more often than necessary |
| Resource ordering | Feasible to enforce via compile-time checks | Disallows incremental resource requests |
| Needs no run-time computation since problem is solved in system design |
| Avoidance | Midway betwen that of detection and prevention | Manipulate to find at least one safe path | No preemption necessary | Future resource requirements must be known by OS |
| Processes can be blocked for long periods of time |
| Detection | Very liveral; requested resources are granted where possible | Invoke periodically to test for deadlock | Never delays process initiation | Inherent preemption losses |
| Facilitates on-line handling |

**Koji operativni sistemi imaju kakve alate za borbu protiv smrtnih zagrljaja... ( sa prezentacije, mrzi me više da kucam)**