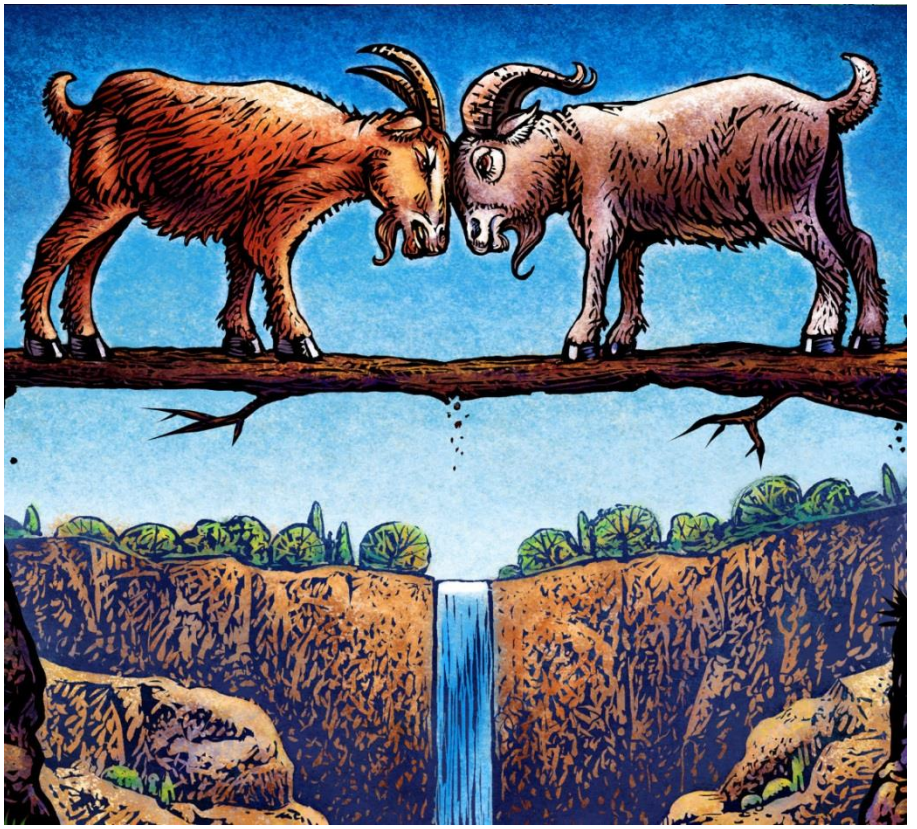
OPERATIVNI SISTEMI

Slajdovi su kreirani na osnovu knjige “Operativni sistemi, principi unutrašnje organizacije i dizajna, 7. izdanje“, William Stallings, CET, Beograd, 2013.

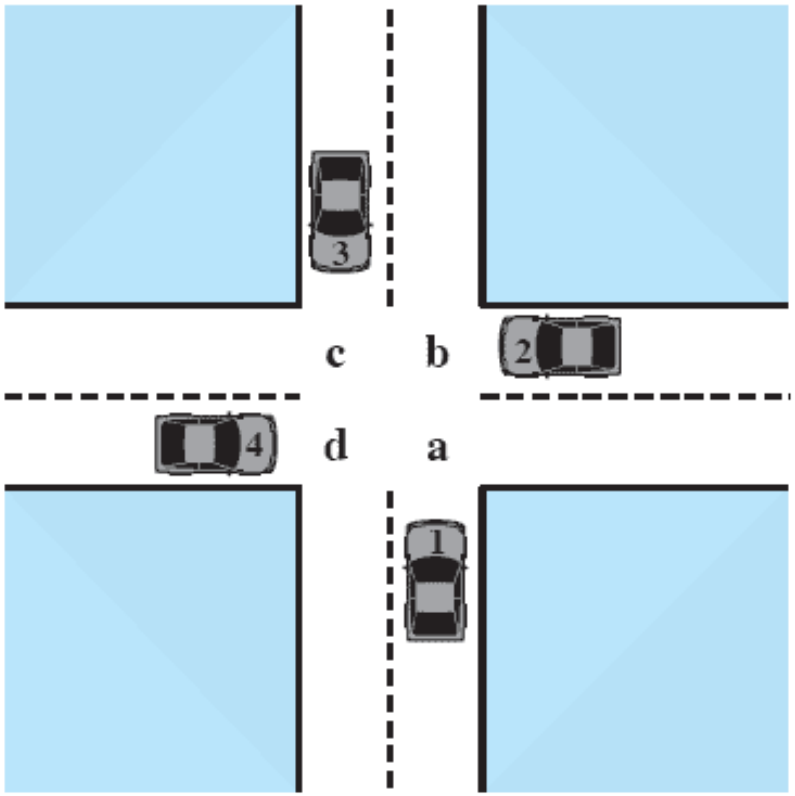
Konkurentnost: Uzajamno blokiranje

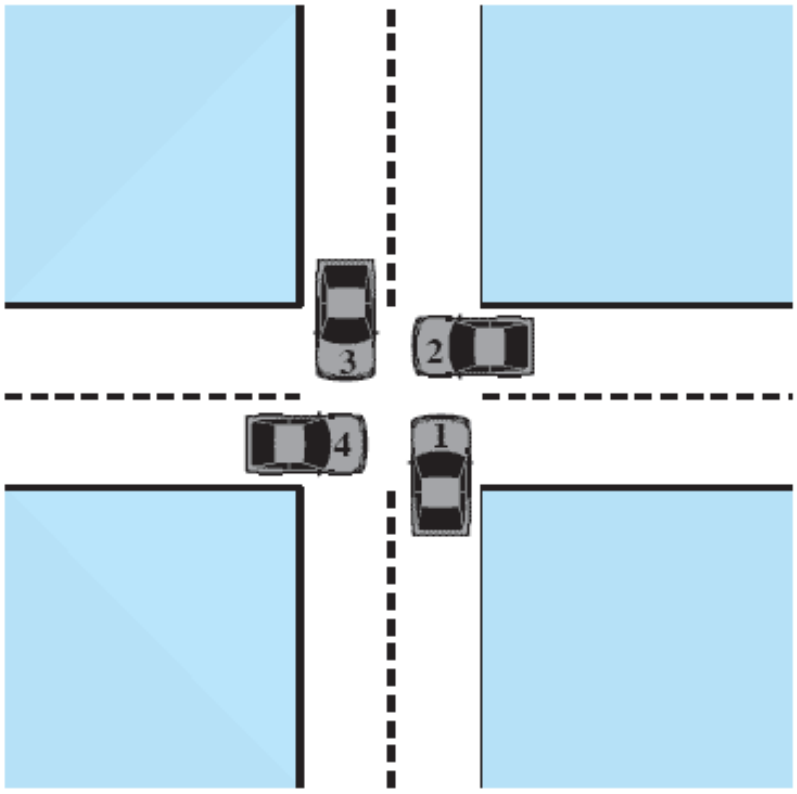
Uzajamno blokiranje

- Trajno blokiranje skupa procesa koji se nadmeću za isti skup resursa
  - Svaki proces u skupu je blokiran čekajući na događaj koji može aktivirati neki blokirani proces iz skupa



Primer uzajamnog blokiranja

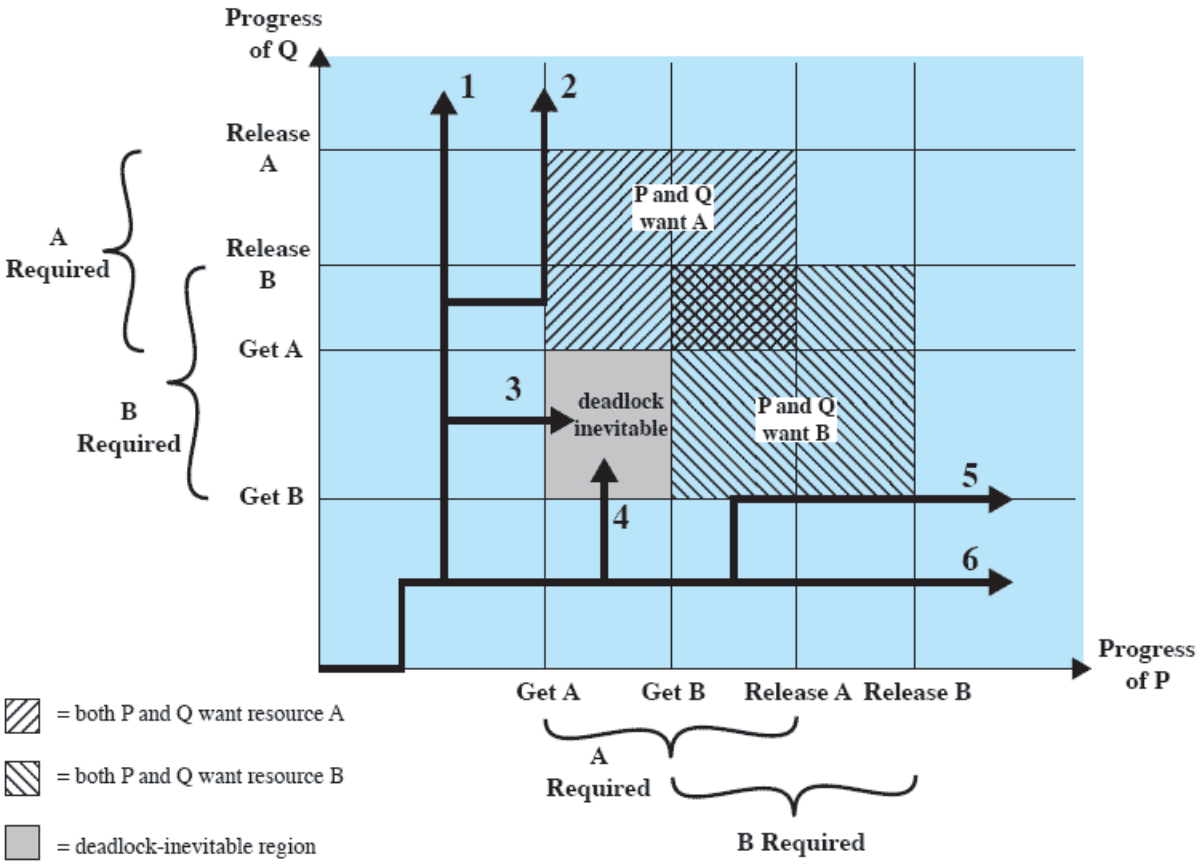




Primer dva procesa i dva resursa

Proces P	**Proces Q**
...	...
Uzmi resurs A	Uzmi resurs B
...	...
Uzmi resurs B	Uzmi resurs A
...	...
Otpusti resurs A	Otpusti resurs B
...	...
Otpusti resurs B	Otpusti resurs A
...	...

Moguće putanje napretka procesa



Putanja napretka procesa

Horizontalna linija - izvršava se P, a Q čeka

Vertikalna linija – izvršava se Q, a P čeka

Model sistema

- Uzajamno blokiranje se može dogoditi u sistemu u kojem procesi koriste resurse na sledeći način
  1. Zahtev
     - Proces pre upotrebe zahteva resurs
     - Ako resurs nije u tom trenutku dostupan, proces odlazi u čekanje
  2. Upotreba
     - U ovoj fazi proces može da pristupa resursu i da ga koristi
  3. Otpuštanje
     - Proces otpušta resurs nakon korišćenja

Uslovi za uzajamno blokiranje

1. Uzajamno isključivanje
   - Samo jedan proces u datom trenutku može koristiti resurs
2. Držanje i čekanje
   - Proces može držati resurs dok čeka dodeljivanje ostalih resursa
3. Bez prekida
   - Resurs se ne može nasilno oduzeti procesu koji ga drži
4. Kružno čekanje
   - Potencijalni rezultat prva tri uslova
   - Postoji zatvoren krug procesa takav da svaki proces drži bar jedan resurs koji je potreban sledećem procesu u krugu

Uslovi za uzajamno blokiranje

Mogućnost uzajamnog blokiranja	**Postojanje uzajamnog blokiranja**
1. Uzajamno isključivanje	1. Uzajamno isključivanje
2. Držanje i čekanje	2. Držanje i čekanje
3. Bez prekida	3. Bez prekida
	4. Kružno čekanje

Rešavanje problema uzajamnog blokiranja

- Sprečavanje
  - Primenjuju se mehanizmi koji eliminišu uslove koji dovode do uzajamnog blokiranja
- Izbegavanje
  - Dinamički se vrši raspodela resursa tako da ne dođe do uzajamnog blokiranja
- Otkrivanje
  - Kada se desi uzajamno blokiranje preduzimaju se akcije za oporavak

Sprečavanje uzajamnog blokiranja

- Sprečavanje
  - Projektovati sistem tako da je isključena mogućnost uzajamnog blokiranja
  - Sprečiti pojavu jednog od četiri uslova za uzajamno blokiranje

izvor: www.youtube.com

Sprečavanje uzajamnog blokiranja

- Nije moguće onemogućiti uslov uzajamnog isključivanja
  - U određenim scenarijima pristupa deljenim resursima procesi moraju pristupati jedan po jedan
- Sprečavanje uslova držanja i čekanja
  - Može se sprečiti ako proces istovremeno zatraži sve potrebne resurse
  - Proces ostaje blokiran dok svi traženi resursi ne budu raspoloživi
  - Neefikasno
    - Proces čeka sve resurse čak i kad može da radi sa onima koje može da zauzme
    - Resursi dodeljeni procesu su zauzeti za druge procese, čak i ako ih proces ne koristi
    - Može doći do gladovanja procesa ako je uvek bar jedan potreban resurs alociran nekom drugom procesu

Sprečavanje uzajamnog blokiranja

- Odbacivanje uslova „bez prekidanja“
  - Naredba procesu da otpusti svoje resurse ukoliko mu je odbijen zahtev za zauzimanje nekog resursa
  - Druga varijanta je da ako proces zatraži resurs koji drži drugi blokirani proces, OS prekida drugi proces, koji mora da otpusti sve svoje resurse
  - Primenjivo samo kod resursa čije se stanje može lako sačuvati i kasnije oporaviti
    - Npr procesorski registri

Sprečavanje uzajamnog blokiranja

- Sprečavanje kružnog čekanja
  - Ako se uradi linearno slaganje tipova resursa
  - Svakom resursu se dodeli indeks
  - Ako je proces zauzeo resurs R_i, onda može zauzeti resurs R_j samo ako je `j > i`
  - U ovoj varijanti ne može doći do kružnog čekanja
  - Može da bude neefikasno
    - Procesu se možda nepotrebno zabranjuje zauzimanje resursa
  - Ako programer ne poštuje predviđeni redosled, opet može doći do uzajamnog blokiranja
    - Moguće je koristiti sistemski softver koji prati da li se resursi zauzimaju u predviđenom redosledu
    - Npr program Witness na FreeBSD OS

Sprečavanje uzajamnog blokiranja

- Sprečavanje je suviše restriktivno
  - Slaba iskorišćenost resursa
  - Smanjena propusna moć sistema

- Pristup koji je manje restriktivan i dozvoljava više konkurentnosti je izbegavanje uzajamnog blokiranja

Izbegavanje uzajamnog blokiranja

- Izbegavanje
  - Dinamički se vrši raspodela resursa tako da ne dođe do uzajamnog blokiranja

izvor: www.youtube.com

Izbegavanje uzajamnog blokiranja

- Dozvoljava se postojanje tri neophodna uslova
- Prave se razumni izbori tako da se nikad ne stigne u tačku uzajamnog blokiranja
- Omogućuje više konkurentnosti od sprečavanja
- Dinamički se donosi odluka da li će tekući zahtev dodele resursa dovesti do uzajamnog blokiranja

Izbegavanje uzajamnog blokiranja

- Varijante
  - Ne pokretati proces ukoliko njegovi zahtevi mogu dovesti do blokiranja
  - Ne obrađivati zahtev za resursom ako dodela resursa može dovesti do blokiranja

Stanje sistema

- Ako imamo sistem sa n procesa i m tipova resursa
- Vektor R = [R_1, R_2, …, R_m]
  - Ukupna količina svakog resursa u sistemu

R1	R2	R3
9	3	6

- Vektor V = [R_1, R_2, …, R_m]
  - Ukupna količina svakog resursa u sistemu koja nije dodeljena nijednom procesu (raspoloživo)

R1	R2	R3
4	0	1

Stanje sistema

- Matrica zahteva

$$C = \begin{bmatrix} C_{11} & C_{12} & \dots & C_{1m} \\ C_{21} & C_{22} & \dots & C_{2m} \\ \dots & \dots & \dots & \dots \\ C_{n1} & C_{n2} & \dots & C_{nm} \end{bmatrix}$$

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4

- C_{ij} – zahtev procesa *i* za resursom *j*
- Mora biti unapred određeno za svaki proces

Stanje sistema

- Matrica dodele

$$A = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1m} \\ A_{21} & A_{22} & \dots & A_{2m} \\ \dots & \dots & \dots & \dots \\ A_{n1} & A_{n2} & \dots & A_{nm} \end{bmatrix}$$

	R1	R2	R3
P1	1	0	0
P2	6	1	2
P3	2	1	1

- A_{ij} – trenutna dodeljenost resursa *j* procesu *i*

Stanje sistema

- Za promenljive stanja sistema važi:

1. $R_j = V_j + \sum_{i=1}^{n} A_{ji}$
   - Svi resursi su raspoloživi ili dodeljeni
2. $C_{ij} \leq R_j$, za sva i, j
   - Proces ne može zahtevati više od ukupne količine resursa
3. $A_{ij} \leq C_{ij}$, za sva i, j
   - Nijednom procesu nije dodeljeno više resursa nego što je zahtevao

Odbijanje inicijalizacije procesa

- Proces neće biti pokrenut ako njegovi zahtevi za resursima mogu dovesti do uzajamnog blokiranja
- Pokrenuti novi proces P_{n+1} samo ako važi

$$R_j \geq C_{(n+1)j} + \sum_{i=1}^{n} C_j$$

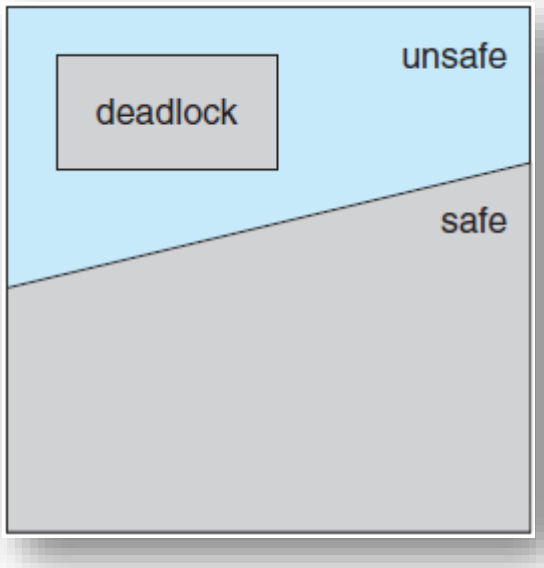
  - Pokreni proces ako se mogu zadovoljiti maksimalni zahtevi tekućih procesa uvećani za zahteve novog procesa
  - Suviše restriktivno
    - Pretpostavlja se da će svi procesi istovremeno koristiti resurse

Odbijanje dodele resursa

- Zasniva se na odluci o dodeli resursa na osnovu stanja sistema
  - Bezbedno stanje sistema
  - Nebezbedno stanje sistema

Bezbedno stanje sistema

- Stanje je bezbedno ako sistem može da ispuni zahteve za resursima svakog procesa u nekom redosledu koji ne dovodi do uzajamnog blokiranja
  - Znači da postoji sekvenca u kojoj procesima treba dodeljivati resurse tako da će svaki proces čekati na resurse konačno vreme
- U bezbednom stanju nema uzajamnog blokiranja
- Kada stanje nije bezbedno, može (ali ne mora obavezno) doći do uzajamnog blokiranja



Utvrđivanje bezbednog stanja

- Stanje je bezbedno ako postoji sekvenca procesa P_1, P_2, ... P_n takva da

- Za svaki P_i, resursi koje P_i zahteva mogu biti zadovoljeni sa trenutno raspoloživim resursima plus resursima koje koriste procesi P_j, pri čemu je $j < i$

Algoritam izbegavanja uzajamnog blokiranja

- Utvrđuje da li postoji sekvenca u kojoj procesi zauzimaju i oslobađaju resurse tako da zahtevi svih procesa budu zadovoljeni
- Cilj je da sistem uvek bude u bezbednom stanju
- Proces napravi zahtev za resursom
- Simulira se da je zahtev odobren
- Proveri se da li je novo stanje u koje bi se prešlo odobravanjem zahteva bezbedno
- Ako je bezbedno, odobrava se zahtev
- Ako nije bezbedno, proces se blokira

Strukture podataka stanja sistema

```
struct state {
    int resource[m]; //ukupno
    int available[m]; //raspolozivo
    int claim[n][m]; //inicijalno zahtevano
    int alloc[n][m]; //trenutno dodeljeno
};
```

Algoritam utvrđivanja bezbednog stanja (Bankarev algoritam)

- Koraci algoritma
  1. Definišemo vektor W koji predstavlja simulaciju stanja raspoloživih resursa u nekom budućem trenutku. Inicijalno jednak vektoru rasploživih resursa V

     $$W = V$$

  2. Pronađemo proces čiji se zahtevi trenutno mogu ispuniti, tj. proces i za koji važi

     $$C_{ij} - A_{ij} \leq W_j \; za \; svako \; j = 1, \ldots, m$$

     Ako nema takvog procesa, algoritam se prekida
  3. Simuliramo da je proces i zauzeo resurse, završio rad i oslobodio ih

     $$W_j = W_j + A_{ij}$$

  4. Vratimo se na korak 2. Ako na ovaj način uspemo da prođemo kroz sve procese i za sve simuliramo dodelu resursa, onda je stanje bezbedno

Algoritam utvrđivanja bezbednog stanja (Bankarev algoritam)

```
bool isSafe(state s) {
   int w[m] = s.available; //available is vector V
   bool finished[n]; //false for all processes initially
   bool possible = true;
   while (possible) {
      int i = findNextProcess(finished, s);
      if (i != -1) { //if next process is found
         for (int j = 0; j < m; j++)
            w[j] += s.alloc[i][j];
         finished[i] = true;
      } else {
         possible = false;
      }
   }
   //see next slide
```

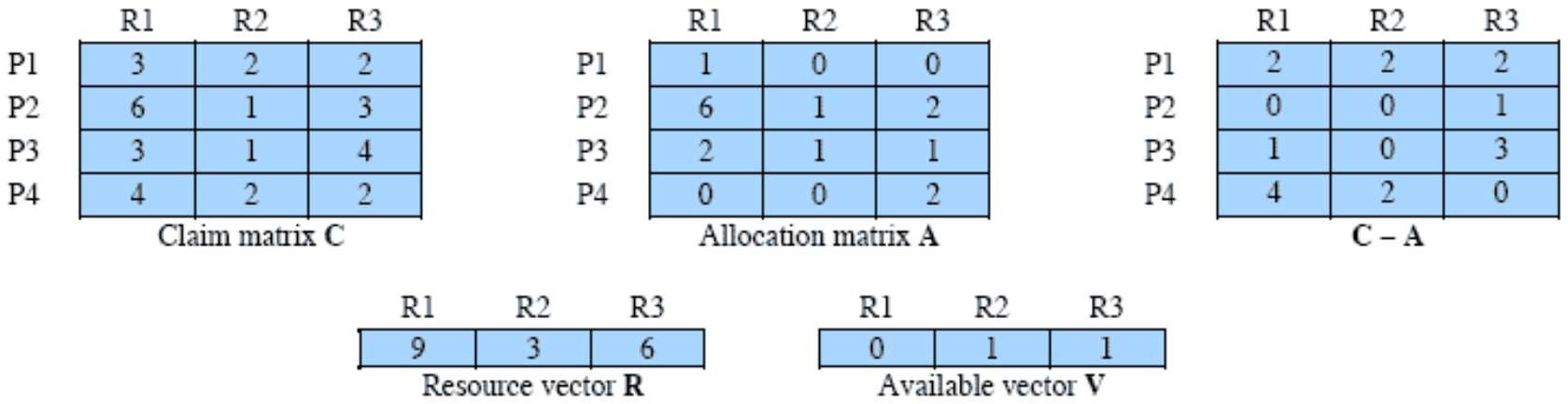
Algoritam utvrđivanja bezbednog stanja (Bankarev algoritam)

```
bool isSafe(state s) {
    ... //see previous slide
    //state is safe if all processes
    //have passed the simulation
    for (int i = 0; i < n; i++)
            if (!finished[i])
                return false;
    return true;
}
```

Algoritam utvrđivanja bezbednog stanja (Bankarev algoritam)

```
int findNextProcess(bool* finished, state s) {
   //find a non-finished process which may satisfy its requests
   for (int i = 0; i < n; i++) { //iterate through processes
      if (!finished[i]) {
        bool found = true;
         for (int j = 0; j < m; j++) { //iterate through resources
            if (s.claim[i][j] - s.alloc[i][j] > available[j])
                found = false;
                break;
         }
         if (found) {
            return i;
        }
      }
   }
   return -1;
}
```

Utvrđivanje bezbednog stanja



	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	1	0	0
P2	6	1	2
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	2	2	2
P2	0	0	1
P3	1	0	3
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
0	1	1

Available vector **V**

- Inicijalno stanje
- Proces P2 može da zadovolji svoje zahteve za resursima

Utvrđivanje bezbednog stanja

	R1	R2	R3
P1	3	2	2
P2	0	0	0
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	1	0	0
P2	0	0	0
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	2	2	2
P2	0	0	0
P3	1	0	3
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
6	2	3

Available vector **V**

- Proces P2 se izvršio do kraja
- Resursi koje je koristio su sada raspoloživi
- Svaki od preostalih procesa ima uslove da se izvrši

Utvrđivanje bezbednog stanja

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	1	0	3
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
7	2	3

Available vector **V**

- Proces P1 se izvršio do kraja
- Resursi koje je koristio su sada raspoloživi

Utvrđivanje bezbednog stanja

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
9	3	4

Available vector **V**

- Proces P3 se izvršio do kraja
- Resursi koje je koristio su sada raspoloživi
- Znači da je inicijalno stanje bilo bezbedno

Utvrđivanje nebezbednog stanja

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	1	0	0
P2	5	1	1
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	2	2	2
P2	1	0	2
P3	1	0	3
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
1	1	2

Available vector **V**

- Inicijalno stanje
- Proces P1 napravi zahtev za još jednom jedinicom resursa R1 i R3
- Proverićemo u kakvo bi stanje došli ako odobrimo ovaj zahtev

Utvrđivanje nebezbednog stanja

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	2	0	1
P2	5	1	1
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	1	2	1
P2	1	0	2
P3	1	0	3
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
0	1	1

Available vector **V**

- Stanje u koje bi došli odobravanjem zahteva procesu P1 je nebezbedno
- Nijedan od procesa ne bi mogao da ispuni svoje zahteve
- Zato procesu P1 ovaj zahtev neće biti odobren

Algoritam izbegavanja uzajamnog blokiranja

- Prvo se utvrđuje se da li zahtev premašuje maksimalne zahteve procesa
- *i* - indeks procesa koji postavlja zahtev
- *request[m]* - količine resursa koje proces zahteva u ovom zahtevu

```
for (int j = 0; j < m; j++) {
    if (alloc[i][j] + request[j] >
          claim[i][j])

          <error>
}
```

Algoritam izbegavanja uzajamnog blokiranja

- Da li ima dovoljno raspoloživih resursa?

```
for (int j = 0; j < m; j++) {
     if (request[j] > available[j])
          <suspend process>
}
```

Algoritam izbegavanja uzajamnog blokiranja

- Simulacija da je zahtev odobren
- U *new state* se ubaci novo stanje koje bi nastalo odobravanjem zahteva

```
newstate = state;
for (int j = 0; j < m; j++) {
     newstate.alloc[i][j] += request[j];
     newstate.available[j] -= request[j];
}
```

Algoritam izbegavanja uzajamnog blokiranja

- Provera da li bi ovo potencijalno naredno stanje bilo bezbedno
- Ako bi bilo bezbedno, može se izvršiti zahtev

```
if (isSafe(newState)) {
      state = newState;
      <carry out allocation>
} else
      <suspend process>
```

Izbegavanje uzajamnog blokiranja

- Prednosti
  - Manje restriktivno od sprečavanja
  - Nije neophodno prekinuti i vratiti unazad procese kao kod otkrivanja uzajamnog blokiranja
- Mane
  - Maksimalni zahtevi za resursima svakog procesa moraju se unapred definisati
  - Proces koji se razmatra mora biti nezavisan (redosled izvršavanja nije ograničen sinhronizacijom)
  - Mora postojati fiksan broj resursa za dodelu
  - Proces ne može da otpusti resurs

Otkrivanje uzajamnog blokiranja

- Otkrivanje
  - Kada se desi uzajamno blokiranje preduzimaju se akcije za oporavak

izvor: www.youtube.com

Otkrivanje uzajamnog blokiranja

- Ne ograničava se pristup resursima
- Procesima se dodeljuju zahtevani resursi
- Povremeno se vrši provera da li je došlo do kružnog čekanja
- U slučaju da jeste, vrši se oporavak sistema

Algoritam otkrivanja uzajamnog blokiranja

Matrica zahteva Q

	R1	R2	R3	R4	R5
P1	0	1	0	0	1
P2	0	0	1	0	1
P3	0	0	0	0	1
P4	0	0	0	0	0

Matrica dodele A

	R1	R2	R3	R4	R5
P1	1	0	1	1	0
P2	1	1	0	0	0
P3	0	0	0	1	0
P4	0	0	0	0	0

Vektor W

0	0	0	0	1

Vektor R (resursi)

2	1	1	2	1

Vektor V (raspoloživo)

0	0	0	0	1

- Cilj je da označimo procese koji nisu uzajamno blokirani
- Oni koji ostanu su uzajamno blokirani
- *1. Inicijalizujemo vektor W da bude jednak vektoru V*

Algoritam otkrivanja uzajamnog blokiranja

Matrica zahteva Q

	R1	R2	R3	R4	R5
P1	0	1	0	0	1
P2	0	0	1	0	1
P3	0	0	0	0	1
P4	0	0	0	0	0

Matrica dodele A

	R1	R2	R3	R4	R5
P1	1	0	1	1	0
P2	1	1	0	0	0
P3	0	0	0	1	0
P4	0	0	0	0	0

Vektor W

0	0	0	0	1

Vektor R (resursi)

2	1	1	2	1

Vektor V (raspoloživo)

0	0	0	0	1

- *2. Označimo sve procese koji u matrici dodela imaju sve nule (oni nemaju dodeljene nikakve resurse, pa nisu uzajamno blokirani)*
- Označavamo proces P4

Algoritam otkrivanja uzajamnog blokiranja

Matrica zahteva Q

	R1	R2	R3	R4	R5
P1	0	1	0	0	1
P2	0	0	1	0	1
P3	0	0	0	0	1
P4	0	0	0	0	0

Matrica dodele A

	R1	R2	R3	R4	R5
P1	1	0	1	1	0
P2	1	1	0	0	0
P3	0	0	0	1	0
P4	0	0	0	0	0

Vektor W

0	0	0	0	1

Vektor R (resursi)

2	1	1	2	1

Vektor V (raspoloživo)

0	0	0	0	1

- *3. Pronađemo neoznačen proces čiji su zahtevi manji ili jednaki od W*
- *Ako nema takvog procesa, prekinemo algoritam*
- P3 ispunjava uslov

Algoritam otkrivanja uzajamnog blokiranja

Matrica zahteva Q

	R1	R2	R3	R4	R5
P1	0	1	0	0	1
P2	0	0	1	0	1
P3	0	0	0	0	1
P4	0	0	0	0	0

Matrica dodele A

	R1	R2	R3	R4	R5
P1	1	0	1	1	0
P2	1	1	0	0	0
P3	0	0	0	1	0
P4	0	0	0	0	0

Vektor W

0	0	0	1	1

Vektor R (resursi)

2	1	1	2	1

Vektor V (raspoloživo)

0	0	0	0	1

- *4. Označimo pronađeni proces i dodamo njegov red iz matrice dodele u vektor W (simuliramo da se proces izvršio i oslobodio resurse)*
- Označavamo P3
- W = W + [0,0,0,1,0] = [0,0,0,1,1]

Algoritam otkrivanja uzajamnog blokiranja

Matrica zahteva Q

	R1	R2	R3	R4	R5
P1	0	1	0	0	1
P2	0	0	1	0	1
P3	0	0	0	0	1
P4	0	0	0	0	0

Matrica dodele A

	R1	R2	R3	R4	R5
P1	1	0	1	1	0
P2	1	1	0	0	0
P3	0	0	0	1	0
P4	0	0	0	0	0

Vektor W

0	0	0	1	1

Vektor R (resursi)

2	1	1	2	1

Vektor V (raspoloživo)

0	0	0	0	1

- *5. Vratimo se na korak 3*
- Nema više procesa koji ispunjavaju uslov
- Neoznačeni su ostali P1 i P2, što znači da su uzajamno blokirani

Strategije oporavka od uzajamnog blokiranja

1. Prekini sve uzajamno blokirane procese
   - Najjednostavnije rešenje
   - Velika cena ovakvog pristupa jer se poništava sve što je proces do tada uradio
2. Vrati sve uzajamno blokirane procese na neku raniju kontrolnu tačku i ponovo ih pokreni
   - Može opet doći do uzajamnog blokiranja
   - Zbog nedeterminantnosti konkurentne obrade, u ponovnom izvršavanju možda ne bude blokiranja

Strategije oporavka od uzajamnog blokiranja

3. Redom prekidaj sve uzajamno blokirane procese dok više ne bude blokiranja
   - Redosled kojim će procesi biti prekidani treba da bude na osnovu nekog kriterijuma
     - Prioritet
     - Utrošak procesorskog vremena do sada
     - Predviđeno preostalo vreme rada
     - Tip resursa koje je proces koristio
     - Količina resursa potrebna za završetak rada

Strategije oporavka od uzajamnog blokiranja

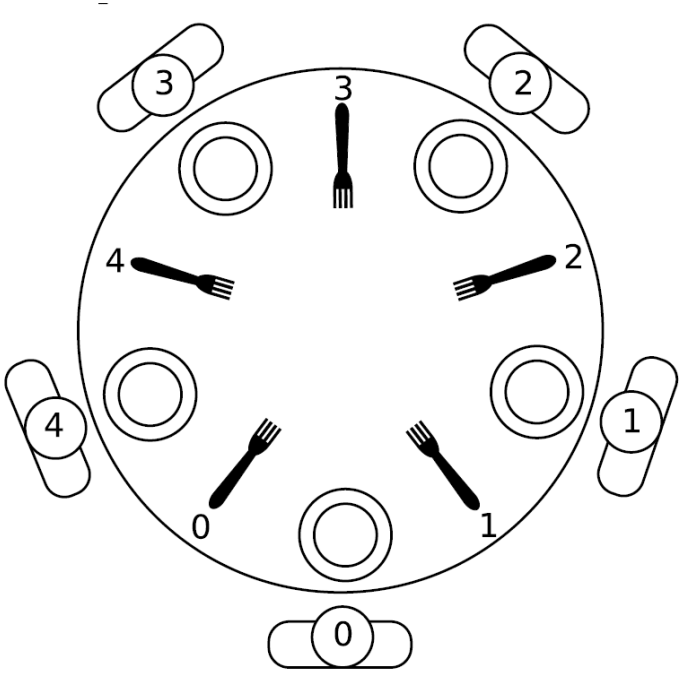
4. Redom oduzimaj resurse od procesa i dodeljuj ih drugim procesima dok više ne bude blokiranja
   - Izabrati resurse i procese tako da se minimizuju troškovi
   - Proces kojem je oduzet resurs mora se vratiti na neko prethodno stanje u kojem je bio pre nabavke resursa
   - Sprečiti gladovanje procesa
     - Ne smeju se resursi uvek oduzimati istom procesu

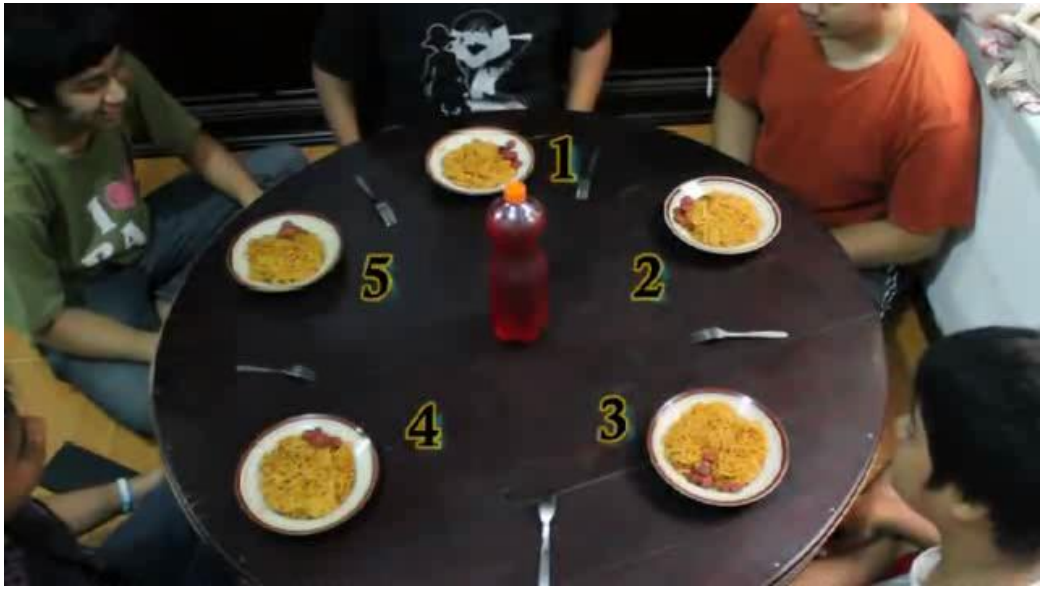
Strategije za uzajamno blokiranje u savremenim OS

- Tipično ne pružaju podršku za zaštitu od uzajamnog blokiranja
- Ignoriše se mogućnost pojave uzajamnog blokiranja
- Ne isplati se angažovati resurse za sprečavanje/izbegavanje/otkrivanje uzajamnog blokiranja
  - OS se implementira sa namerom da ne ulazi u *deadlock*, a korisnički programi su odgovornost programera
- Windows i Linux ovako funkcionišu

Primer uzajamnog blokiranja

- Problem 5 filozofa





izvor: www.youtube.com

Primer uzajamnog blokiranja

- Primeri/Blokiranje/Filozofi