# Uzajamna blokada i gladovanje

## Operativni sistemi 1

Institut za matematiku i informatiku
Prirodno-matematički fakultet, Kragujevac

doc. dr Miloš Ivanović

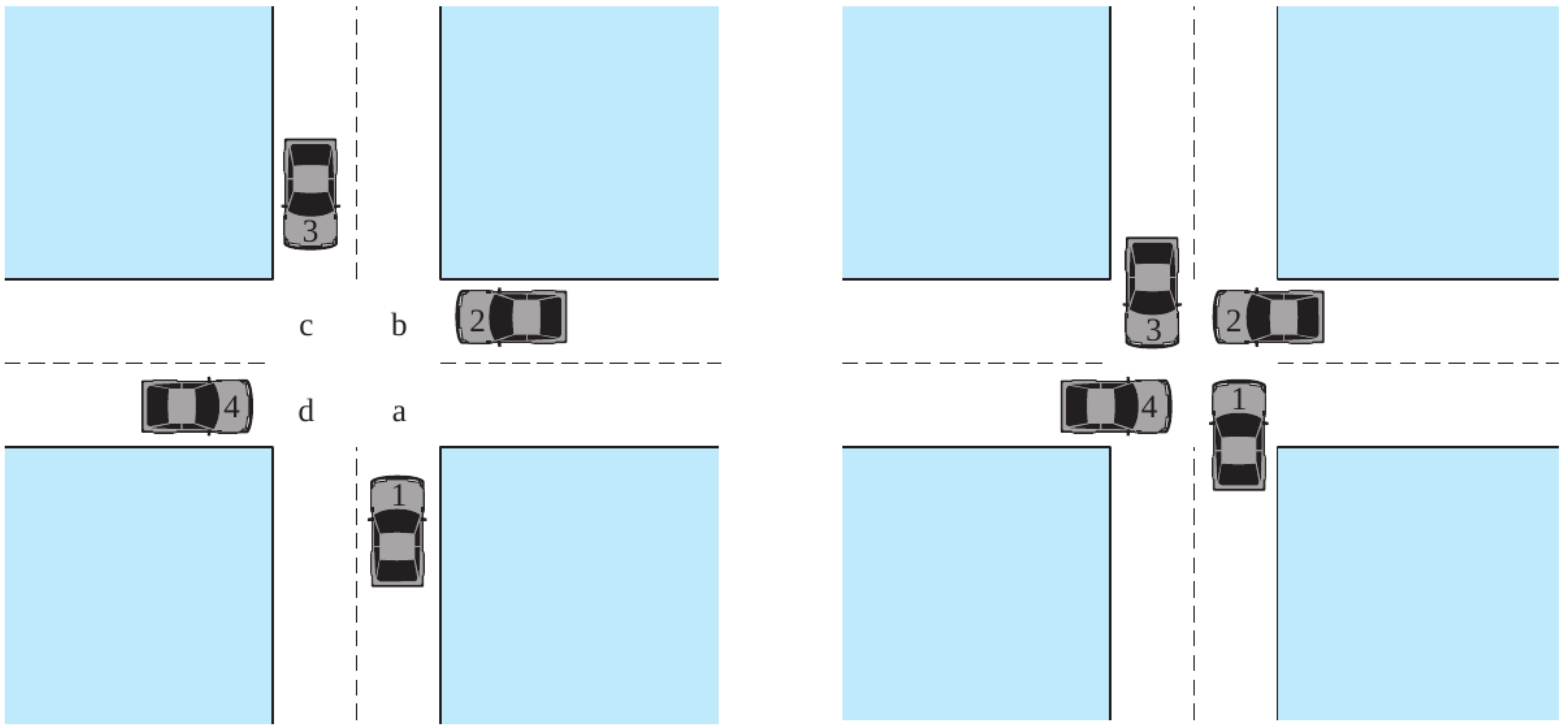
Decembar 2012. god.

# O čemu će biti reči?

1. Principi blokiranja
2. Tretman blokade
   - Uslovi za nastanak
   - Sprečavanje blokade
   - Izbegavanje blokade
   - Otkrivanje blokade
   - Oporavak od blokade
3. Večera filozofa

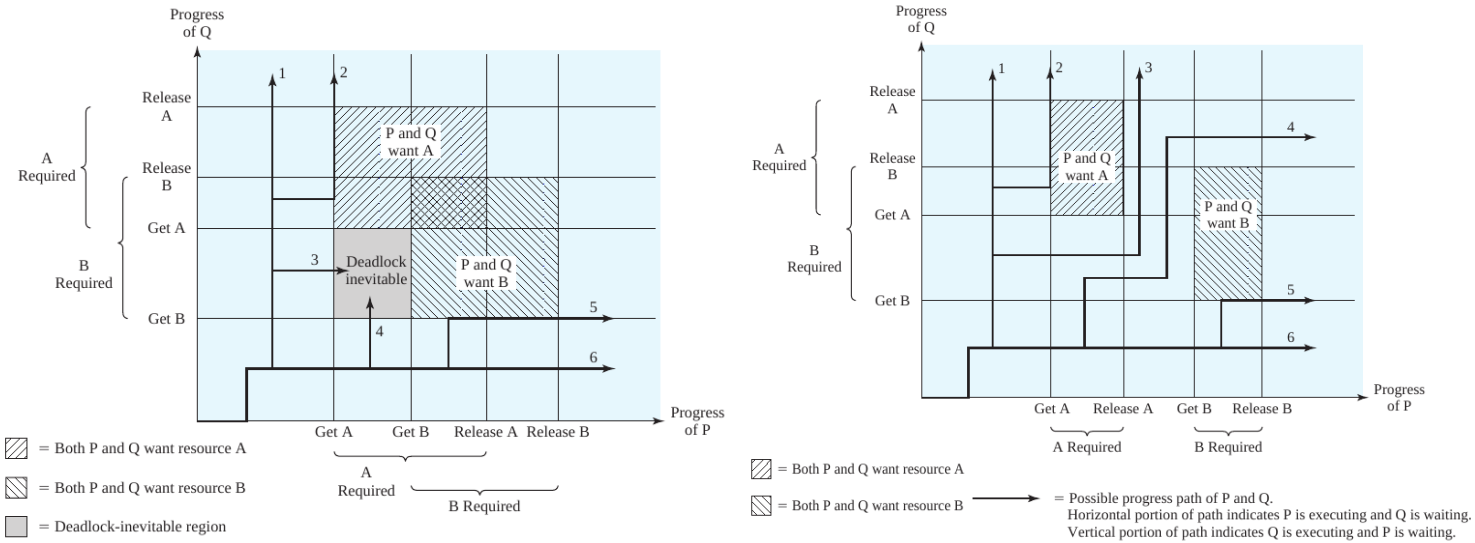
# Primer blokade

Slika : *Levo*: Moguća blokada; *Desno*: Blokada



# *Joint progress* dijagrami za dva procesa

Slika : Primeri dijagrama zajedničkog napredovanja. *Levo*: Moguća blokada; *Desno*: Ne postoje uslovi za nastanak blokade



# Resursi koji se mogu ponovo koristiti

Primer za ovu vrstu resursa su CPU, RAM i spoljna memorija, uređaji, zatim fajlovi, semafori itd.

Slika : *Gore*: Dva procesa se takmiče za resurse. Sekvenca $p_0\ p_1\ q_0\ q_1\ p_2\ q_2$ dovodi do blokade. *Dole*: Memorija na sistemu od 200KB

| Process P | |
|---|---|
| **Step** | **Action** |
| $p_0$ | Request (D) |
| $p_1$ | Lock (D) |
| $p_2$ | Request (T) |
| $p_3$ | Lock (T) |
| $p_4$ | Perform function |
| $p_5$ | Unlock (D) |
| $p_6$ | Unlock (T) |

| Process Q | |
|---|---|
| **Step** | **Action** |
| $q_0$ | Request (T) |
| $q_1$ | Lock (T) |
| $q_2$ | Request (D) |
| $q_3$ | Lock (D) |
| $q_4$ | Perform function |
| $q_5$ | Unlock (T) |
| $q_6$ | Unlock (D) |

| **P1** | **P2** |
|---|---|
| ... | ... |
| Request 80 Kbytes; | Request 70 Kbytes; |
| ... | ... |
| Request 60 Kbytes; | Request 80 Kbytes; |

# Potrošni resursi

Primer za ovu vrstu resursa su poruke, interapti, signali, podaci u baferima isl.

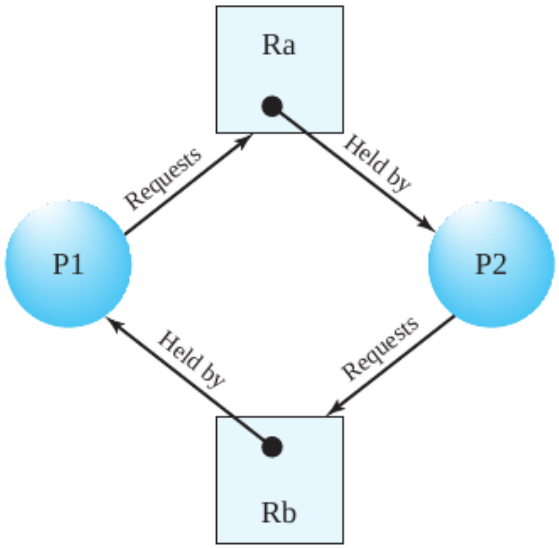
Slika : Primer dva procesa koji se uzajamno blokiraju čekanjem na poruku od onog drugog

| **P1** | **P2** |
|---|---|
| ... | ... |
| Receive (P2); | Receive (P1); |
| ... | ... |
| Send (P2, M1); | Send (P1, M2); |

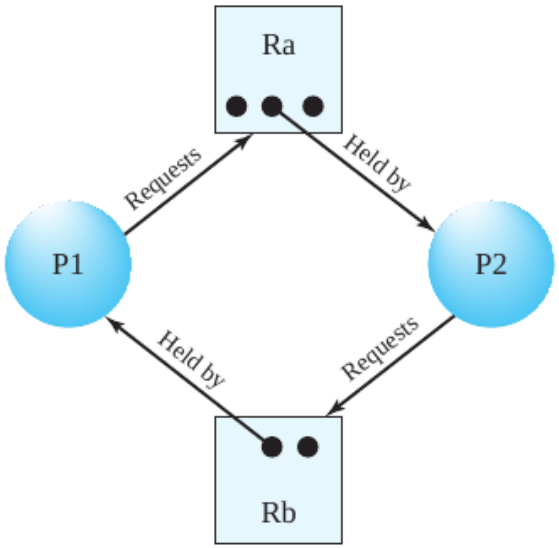
# Grafovi alokacije resursa


(a) Resource is requested


(b) Resource is held

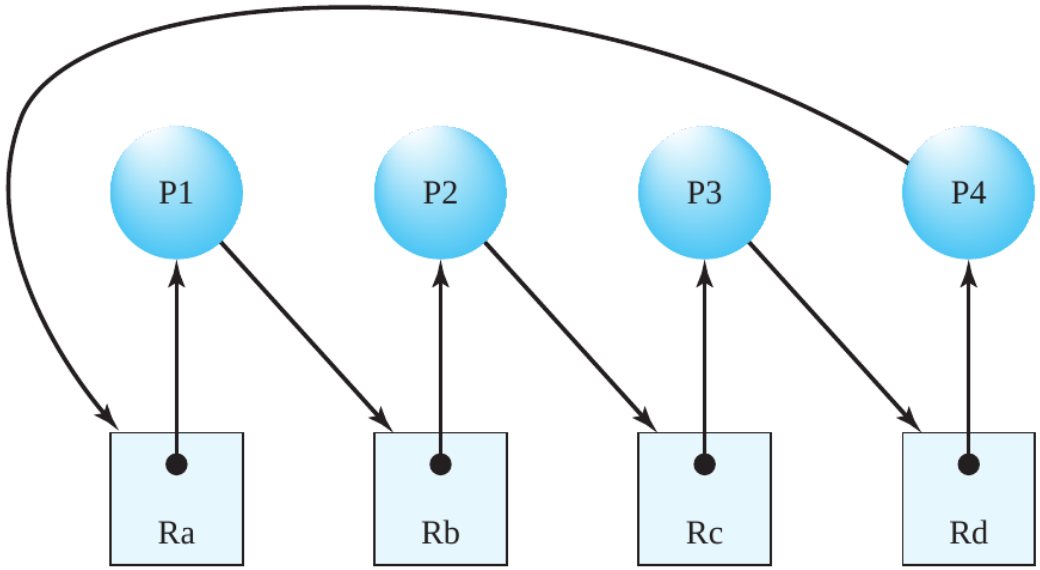



(c) Circular wait




(d) No deadlock

# Grafovi alokacije resursa




**Pitanje**

Koji je od prethodnih primera blokade prikazan na slici?

# Uslovi za nastanak blokade

Sledeća tri uslova moraju biti zadovoljena da bi blokada uopšte bila moguća:

1. **Uzajamno isključenje**. Samo jedan proces može da koristi dati resurs u jednom vremenskom trenutku.
2. **Zauzimanje i čekanje**. Proces može držati resurse zauzetim dok čeka da mu se dodele drugi resursi.
3. **Nema prekidanja**. Resurs se ne može nasilno oduzeti procesu.

## Uslov za stvarni nastanak blokade

Dodatni, četvrti uslov za nastanak blokade je **kružno čekanje**! Prva 3 uslova omogućavaju postojanje fatalne oblasti, dok četvrti uslov znači upadanje sistema u ovu oblast.

## Opšti pristupi za tretiranje blokade

1. **Prevencija**. Izbegava se jedan od gornjih uslova
2. **Izbegavanje**. Pravljenje odgovarajućih dinamičkih izbora baziranih na trenutnom stanju sistema.
3. **Otkrivanje blokade**. Detekcija i pokušaj oporavka od blokade.

# Sprečavanje blokade
## Tretiranje blokade

### Tretiranje blokade

1. Sprečavanje
2. Izbegavanje
3. Otkrivanje

Sprečavanje uzajamne blokade može se postići isključivanjem jednog od četiri uslova, i to:

1. **Onemogućavanje uzajamnog isključivanja**. Realno nije moguće za sve slučajeve. Primer čitalaca i pisaca.
2. **Sprečavanje politike zauzimanja i čekanja**. Moguće ga je postići tako što bi proces pre pokretanja morao da specificira sve resurse koji su mu potrebni. Koje su mane ovog pristupa?
3. **Omogućiti prekidanje**. OS npr. može da prekine proces i da zatraži dodeljene resurse nazad. Ovaj pristup može se upotrebiti samo kod resursa čije se stanje može sačuvati i kasnije oporaviti.
4. **Sprečiti pojavu kružnog čekanja**. Jedan pristup je da proces koji drži resurs $R_i$ može zatražiti resurs $R_j$, samo ako je $j > i$.

# Izbegavanje blokade

## Pristup odbijanja pokretanja procesa

$Resurs = \mathbf{R} = (R_1, R_2, ..., R_m)$ vektor količine svakog resursa u sistemu

$Slobodno = \mathbf{V} = (V_1, V_2, ..., V_m)$ vektor količine svakog resursa koje nijedan proces nije alocirao

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

$C_{ij}$ je zahtev procesa $i$ za resursom $j$

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

$A_{ij}$ je tekuća alokacija resursa $j$ procesom $i$

# Izbegavanje blokade

## Pristup odbijanja pokretanja procesa

### Relacije koje moraju važiti

1. $$R_j = V_j + \sum_{i=1}^{n} A_{ij},\ \forall j \in \{1 \ldots m\}$$

   Svi resursi su ili slobodni ili alocirani

2. $$C_{ij} \leq R_j,\ \forall i, j$$

   Proces ne može da traži više resursa nego što ih ima u sistemu

3. $$A_{ij} \leq C_{ij},\ \forall i, j$$

   Nijedan proces ne može da alocira više resursa nego što je u početku zahtevao.

### Inicijalizuj proces $P_{n+1}$ samo ako:

$$R_j \geq C_{(n+1)j} + \sum_{i=1}^{n} C_{ij},\ \forall j \in \{1 \ldots m\}$$

# Izbegavanje blokade

## Odbijanje dodele resursa–bankarev algoritam

### Definicije

1. **Stanje sistema** prikazuje trenutnu dodelu resursa. Sastoji se iz dva vektora (Resurs-$R$ i Slobodno-$V$) i dve matrice Zahtev-$C$ i Alokacija-$A$)
2. **Bezbedno stanje** je stanje iz koga postoji bar jedan redosled dodele resursa procesima koji neće prouzrokovati uzajamnu blokadu.
3. **Nebezbedno stanje**

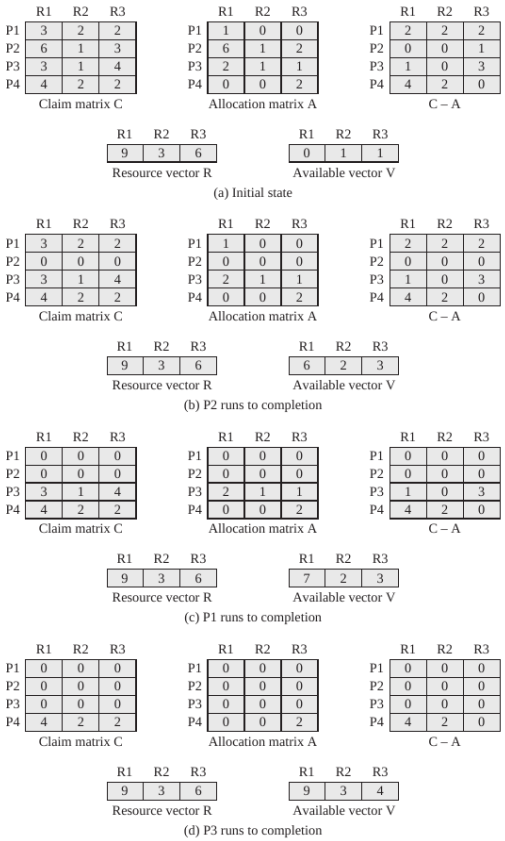
### Neophodan uslov

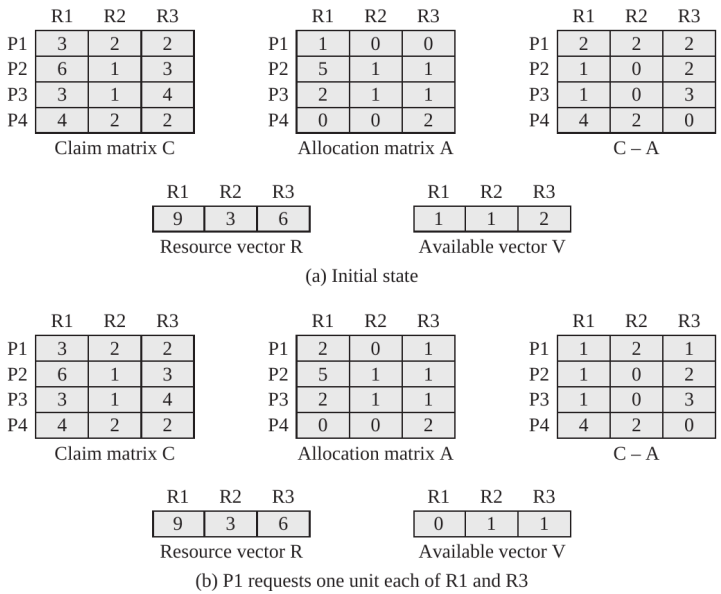
$$C_{ij} - A_{ij} \leq V_j, \text{ za svako } j$$

# Izbegavanje blokade

## Odbijanje dodele resursa–bankarev algoritam

Slika : *Levo*: Utvrđivanje bezbednog stanja, *Desno*: Utvrđivanje nebezbednog stanja



# Implementacija Bankarevog algoritma

```
struct state {
        int resource[m];
        int available[m];
        int claim[n][m];
        int alloc[n][m];
}

if (alloc [i,*] + request [*] > claim [i,*])
        < error >;
/* total request > claim*/
else if (request [*] > available [*])
        < suspend process >;
else
{
        /* simulate alloc */
        < define newstate by:
        alloc [i,*] = alloc [i,*] + request [*];
        available [*] = available [*] - request [*] >;
}
if (safe (newstate))
        < carry out allocation >;
else {
        < restore original state >;
        < suspend process >;
}
```

```
boolean safe (state S) {
        int currentavail[m];
        process rest[<number of processes>];
        currentavail = available;
        rest = {all processes};
        possible = true;
        while (possible) {
                <find a process Pk in rest such that
                        claim [k,*] - alloc [k,*] <= currentavail;>
                if (found) {
                        /* simulate execution of Pk */
                        currentavail = currentavail + alloc [k,*];
                        rest = rest - {Pk};
                }
                else possible = false;
        }
        return (rest == null);
}
```

# Izbegavanje blokade

## Problemi i ograničenja

1. Maksimalni zahtevi procesa za resursima moraju se unapred definisati
2. Redosled izvršavanja unutar procesa ne sme biti ograničen bilo kakvim potrebama sinhronizacije
3. Mora postojati fiksni broj resursa za dodelu
4. Proces ne može izaći dok drži resurse

# Otkrivanje uzajamne blokade

## Algoritam za otkrivanje uzajamne blokade

1. Označi u matrici Alokacija(**A**) svaki proces koji ima red sa svim nulama.
2. Inicijalizuj privremeni vektor **W** da bude jedan vektoru Slobodno (**V**)
3. Pronađi indeks $i$, tako da je proces $i$ trenutno neoznačen $i$-ti red matrice **Q** koji je manji ili jedan vektoru **W** - $Q_{ik} \leq W_k$, za $1 \leq k \leq m$. Ukoliko takvog nema, prekini algoritam.
4. Ukoliko pronađeš takav red, označi proces $i$ i dodaj odgovarajući red dodele u vektor **W**, tako da $W_k = W_k + A_{ik}$ za $1 \leq k \leq m$. Vrati se na korak 3.

# Otkrivanje uzajamne blokade

## Primer

1. Označi proces P4, jer P4 nema dodeljene resurse
2. Postavi vektor $\mathbf{W} = (0\,0\,0\,0\,1)$
3. Zahtev procesa P3 manji je ili jednak **W**, i zato označi proces P3 i postavi novu vrednost vektora $\mathbf{W} = \mathbf{W} + (0\,0\,0\,1\,0) = (0\,0\,0\,1\,1)$
4. Nijedan drugi proces nema red u vektoru **Q** koji je manji ili jednak vektoru **W**. Prekini algoritam.

Algoritam završava rad sa dva procesa (P1 i P2) koji nisu označeni. Prema tome, **P1 i P2 su uzajamno blokirani**.

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

Request matrix Q

| | 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 |

Allocation matrix A

| R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|
| 2 | 1 | 1 | 2 | 1 |

Resource vector

| R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |

Available vector

# Oporavak od blokade

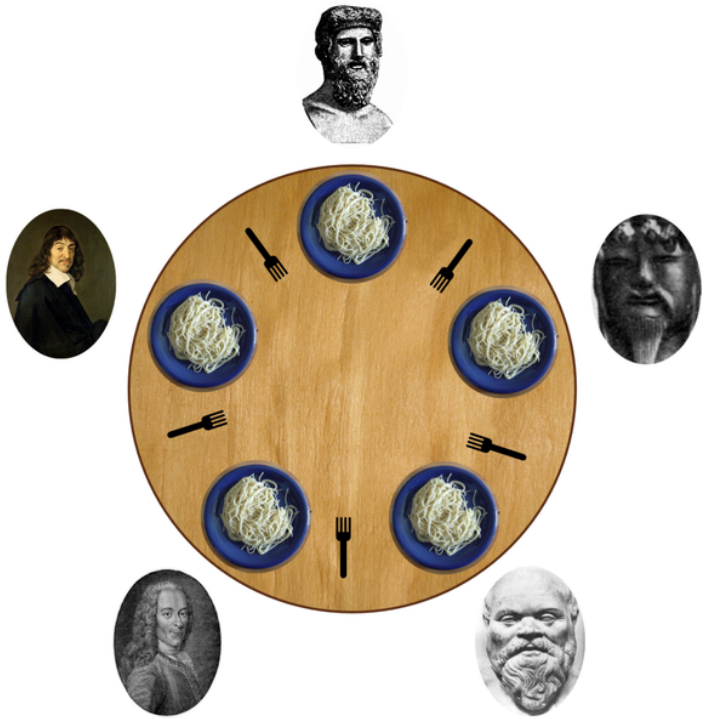
1. **Prekini sve uzajamno blokirane procese**
2. Kopiraj svaki uzajamno blokirani proces na neko kontrolno mesto, a zatim **ponovo pokreni procese**. Da li se blokada može ponoviti?
3. **Redom prekidaj uzajamno blokirane procese** sve dok više ne bude uzajamne blokade. Redosled prekidanja mora da bude na osnovu nekog kriterijuma minimalnog troška. Nakon svakog koraka ponovo mora da se proveri postojanje blokade.
4. **Redom prekidaj resurse** sve dok više ne bude postojalo uzajamno blokiranje. Proces se vraća u stanje pre dobijanja datog resursa.

## Kriterijumi odabira procesa za metode (3) i (4)

- najmanje utroška procesorskog vremena do sada
- najduže predviđeno vreme rada
- najduže predviđeno vreme rada
- najmanji ukupno dodeljeni resursi
- najniži prioritet

# Večera filozofa

- Život svakog filozofa sastoji se od razmišljanja i ishrane
- Svakom filozofu potrebne su dve viljuške da bi mogao da jede
- Konstruisati algoritam koji će petorici filozofa za stolom omogućiti da jedu
- Problem uzajamnog blokiranja i gladovanja (sada bez navodnika :))



# Rešenje upotrebom semafora

Slika : *Levo*: Pogrešno, *Desno*: Tačno

```
/* program   diningphilosophers */
semaphore fork [5] = {1};
int i;
void philosopher (int i)
{
     while (true) {
          think();
          wait (fork[i]);
          wait (fork [(i+1) mod 5]);
          eat();
          signal(fork [(i+1) mod 5]);
          signal(fork[i]);
     }
}
void main()
{
     parbegin (philosopher (0), philosopher (1),
          philosopher (2),    philosopher (3),
          philosopher (4));
     }
```

```
/* program   diningphilosophers */
semaphore fork[5] = {1};
semaphore room = {4};
int i;
void philosopher (int i)
{
    while (true) {
     think();
     wait (room);
     wait (fork[i]);
     wait (fork [(i+1) mod 5]);
     eat();
     signal (fork [(i+1) mod 5]);
     signal (fork[i]);
     signal (room);
    }
}
void main()
{
    parbegin (philosopher (0), philosopher (1),
          philosopher (2),  philosopher (3),
          philosopher (4));
}
```

# Rešenje upotrebom monitora

Slika : *Levo*: Implementacija monitora, *Desno*: Korišćenje

```
monitor dining_controller;
cond ForkReady[5];  /* condition variable for synchronization */
boolean fork[5] = {true}; /* availability status of each fork */

void get_forks(int pid)   /* pid is the philosopher id number */
{
   int left = pid;
   int right = (++pid) % 5;
   /*grant the left fork*/
   if (!fork(left)
      cwait(ForkReady[left]);  /* queue on condition variable */
   fork(left) = false;
   /*grant the right fork*/
   if (!fork(right)
      cwait(ForkReady(right);  /* queue on condition variable */
   fork(right) = false:
}
void release_forks(int pid)
{
   int left = pid;
   int right = (++pid) % 5;
   /*release the left fork*/
   if (empty(ForkReady[left])/*no one is waiting for this fork */
      fork(left) = true;
   else              /* awaken a process waiting on this fork */
      csignal(ForkReady[left]);
   /*release the right fork*/
   if (empty(ForkReady[right])/*no one is waiting for this fork */
      fork(right) = true;
   else   /* awaken a process waiting on this fork */
      csignal(ForkReady[right]);
}
```

```
void philosopher[k=0 to 4]   /* the five philosopher clients */
{
   while (true) {
      <think>;
      get_forks(k);  /* client requests two forks via monitor */
      <eat spaghetti>;
      release_forks(k);/* client releases forks via the monitor */
   }
}
```