

Zastoji (*Deadlocks*)

- **Sistemiški model**
- **Karakterizacija zastoja**
- **Metodi upravljanja zastojem**
- **Prevenција od zastoja (*Deadlock Prevention*)**
- **Izbegavanje zastoja (*Deadlock Avoidance*)**
- **Detekcija zastoja (*Deadlock Detection*)**
- **Oporavak od zastoja (*Recovery from Deadlock*)**
- **Kombinovan pristup za uklanjanje zastoja**

Problem zastoja

■ Zastoj: Skup blokiranih procesa:

- ☞ svaki proces **drži** resurs
- ☞ i **čeka** da uzme resurs
- ☞ koji **drži drugi proces**

■ Primer

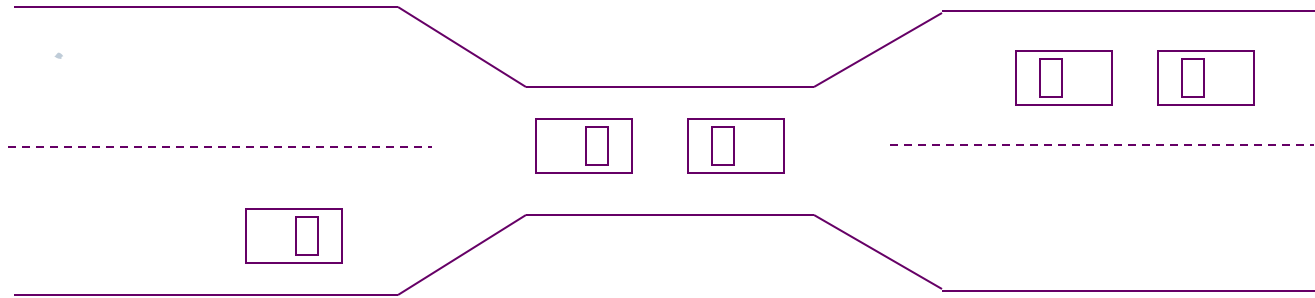
- ☞ Sistem ima **2 trake**.
- ☞ P_1 i P_2 **drže po jednu traku**
- ☞ i oba procesa **žele i drugu**

■ Primer

- ☞ semafori **A** i **B**, inicijalizovani na 1

P_0	P_1
<i>wait (A);</i>	<i>wait(B)</i>
<i>wait (B);</i>	<i>wait(A)</i>

Problem prelaska preko mosta



- Saobraćaj je **samo u jednom** smeru.
- Svaki deo mosta možemo videti kao **resurs**.
- Ako se pojavi zastoј, to možemo rešiti ako se jedan automobil vrati nazad (***preempt resources and rollback***).
- Više automobila treba vratiti nazad ako nastane zastoј.
- Zakucavanje je moguće

Sistemiški model

- **Resursi** tipa R_1, R_2, \dots, R_m
CPU ciklusi, memorijski prostor, I/O uređaji
- **Svaki resurs tipa R_i ima W_i instancu**
- Svaki **proces** koristi **resurs** na sledeći način:
 - ☞ zahtev (*request*)
 - ☞ korišćenje (*use*)
 - ☞ oslobađanje (*release*)

Karakterizacija zastoja

Zastoj nastupa ako su **istovremeno** ispunjena **4 uslova** .

- **1. Mutual exclusion (Međusobno isključenje)**: samo jedan proces u jednom trenutku može koristiti resurs.
- **2. Hold and wait (Drži i čekaj)**: proces drži jedan resurs a istovremeno čeka dobijanje resursa koji koristi neki drugi proces.
- **3. No preemption (Nema otimačine)**: resurs se može osloboditi samo kada proces koji ga koristi **završi svoj posao**.
- **4. Circular wait (Kružno čekanje)**: postoji skup procesa $\{P_0, P_1, \dots, P_n\}$ koji čekaju resurs u sledećem poretku:
 - ☞ P_0 čeka na resurs koji drži P_1 ,
 - ☞ P_1 čeka na resurs koji drži P_2, \dots ,
 - ☞ P_{n-1} čeka na resurs koji drži P_n ,
 - ☞ P_n čeka na resurs koji drži P_0

Graf dodeljenih resursa

Skup objekata V i skup strelica E .

- V sadrži dva skupa:

☞ $P = \{P_1, P_2, \dots, P_n\}$, skup se sastoji od **svih procesa** u sistemu

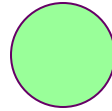
☞ $R = \{R_1, R_2, \dots, R_m\}$, skup se sastoji od **svih resursa** u sistemu

- **strelica zahteva**: $P_1 \rightarrow R_j$

- **strelica alokacije**: $R_j \rightarrow P_i$

Graf dodeljenih resursa

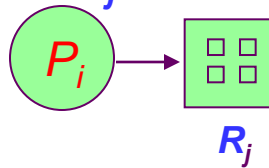
- Proces



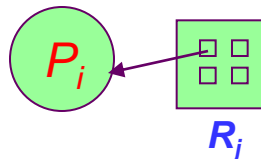
- Resurs sa 4 instance



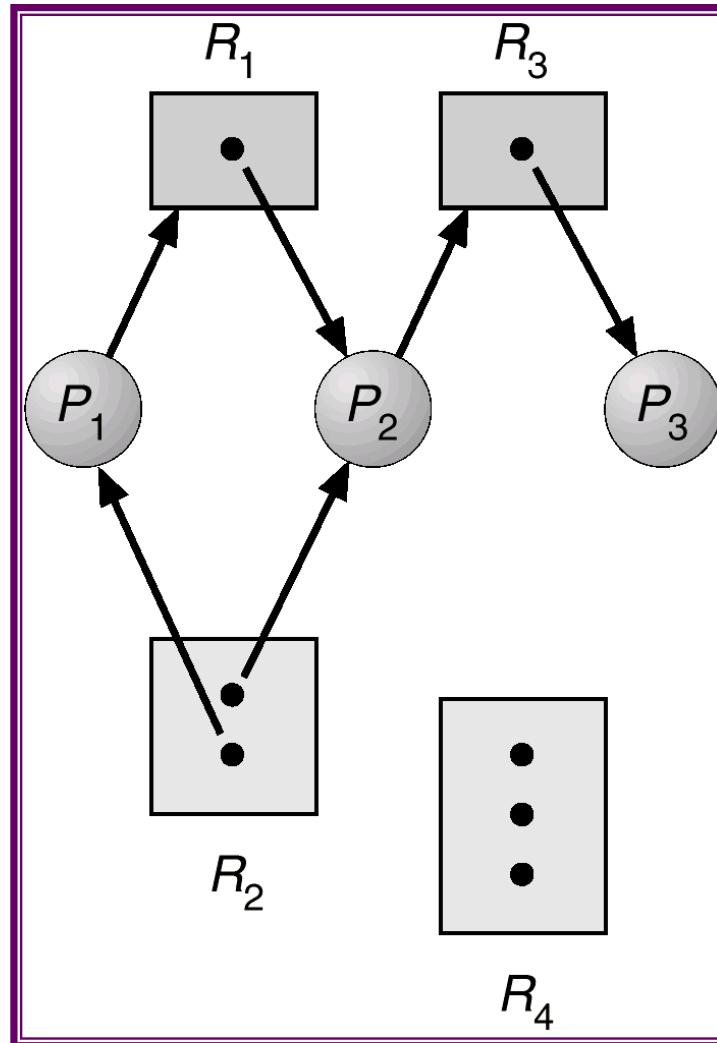
- P_i zahteva instancu R_j



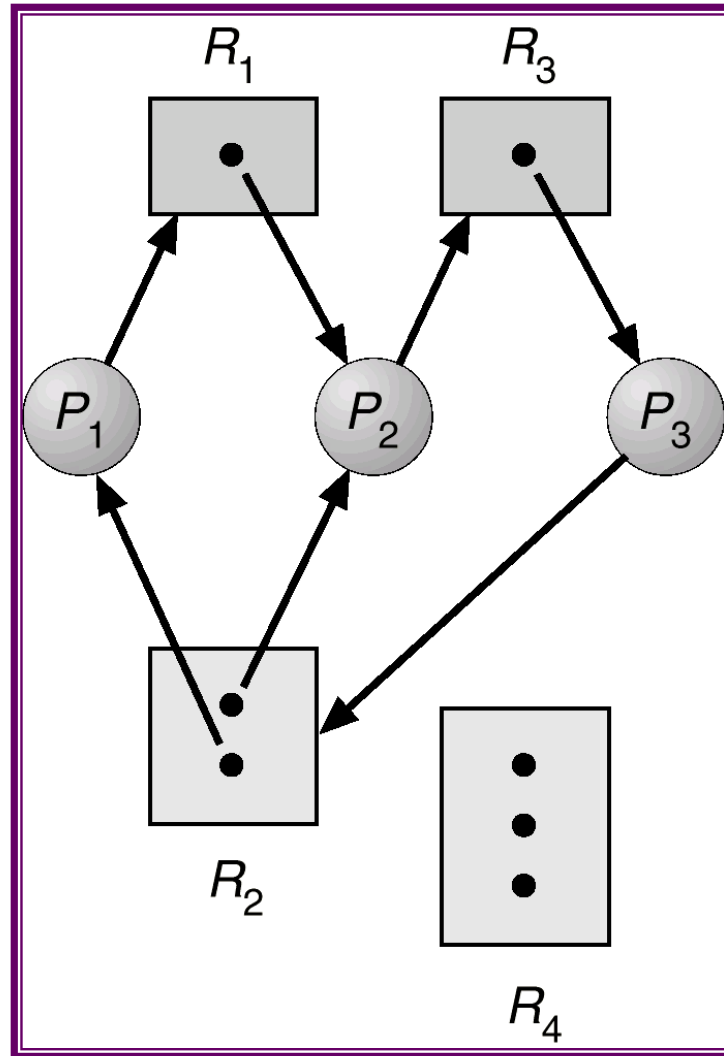
- P_i drži instancu R_j



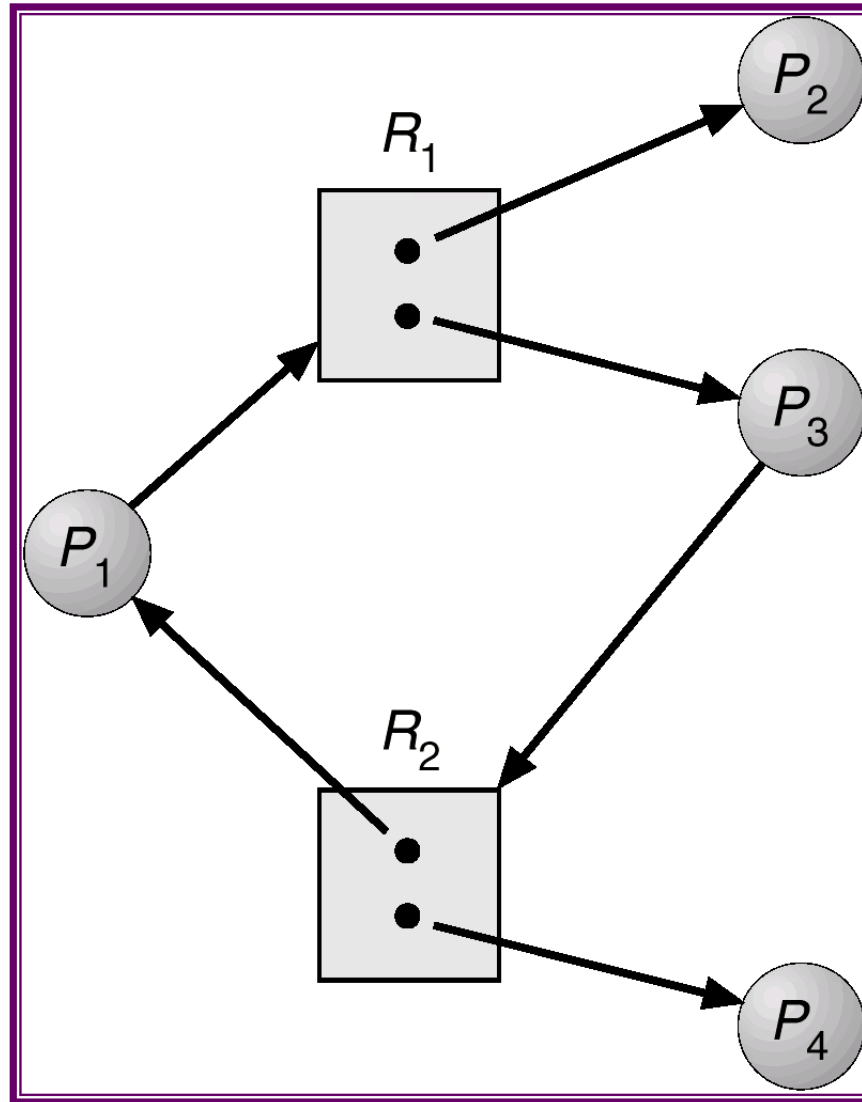
Graf dodeljenih resursa



Graf dodeljenih resursov sa zastojem



Graf dodeljenih resursa – kružni tok **bez zastoja**



Osnovne činjenice

- Ako graf ne sadrži kružni tok \Rightarrow nema zastoja
- Ako graf sadrži kružni tok \Rightarrow
 - ☞ ako resursi u kružnom toku sadrže samo jednu instancu,
 - 📄 Nastupio je zastoj
 - ☞ ako resursi u kružnom toku sadrže više instanci,
 - 📄 moguć je zastoj

Metodi upravljanja zastojem

- 1. Obezbeđuju da sistem **nikada ne uđe** u stanje zastoja.
- 2. **Dozvoljava** da ako sistem **uđe u zastoju**, da iz njega **izađe**
- 3. **Ignoriše** probleme pretvara se da se zastoji nikad ne javljaju u sistemu; **ovo koriste mnogi operativni sistemi**, uključujući i UNIX.

Ostrich algorithm



Prevenција zastoja

Potrebno je sprečiti ispunjavanje **jednog od sledećih uslova**:

- **Međusobno isključenje** – nepotrebno za deljive resurse; obavezno za nedeljive resurse
- **Drži i čekaj** (hold and wait)– ukoliko proces zahteva neki resurs, **nema prava da drži neki drugi resurs (resources)**
 - ☞ Proces traži i alocira **sve** svoje resurse pre početka izvršavanja
 - ☞ ili
 - ☞ proces može tražiti resurs samo pod uslovom da ne drži **ni jedan** drugi
 - ☞ **Mala iskorišćenost resursa**; zakucavanje moguće

Prevenција zastoja

■ Nema pretpražnjenja (no preemption):

- ☞ Ako proces **drži** neki resurs,
- ☞ a traži drugi resurs koga **ne može trenutno da dobije**,
- ☞ onda **proces treba da otpusti sve dosadašnje zauzete resurse**
- ☞ Oslobodjeni resursi **se ubacuju** na listu resursa za koje proces **čeka**
- ☞ **Proces će se restartovati samo** kada bude mogao da povрати **stare resurse**, kao i da dobije **nove** koji su mu trebali

■ Kružno čekanje:

- ☞ uvede se **poredak označavanja** svih resursa
- ☞ natera se **da svaki proces zahteva resurse**
- ☞ u **strogo rastućem redu**

Izbegavanje zastoja (Deadlock avoidance)

Zahteva od procesa **dodatne informacije** o tome koji će mu svi **resursi biti potrebni** i u **kom redosledu**

- **Najprostiji i najviše korišćeni model** zahteva od svakog procesa da objavi **maximalni broj** resursa koji će mu možda trebati.
- **Algoritam izbegavanja zastoja**
- **dinamički ispituje trenutno stanje dodeljenih resursa**
- tako da **osigura** da sistem
- **nikada ne uđe u stanje cirkularnog čekanja**
- Pod **stanjem dodeljenih resursa**
- podrazumeva se stanje definisano **brojem:**
 - ☞ **raspoloživih**
 - ☞ **dodeljenih**
 - ☞ **maksimalno traženih resursa**
 - ☞ u jednom trenutku vremena

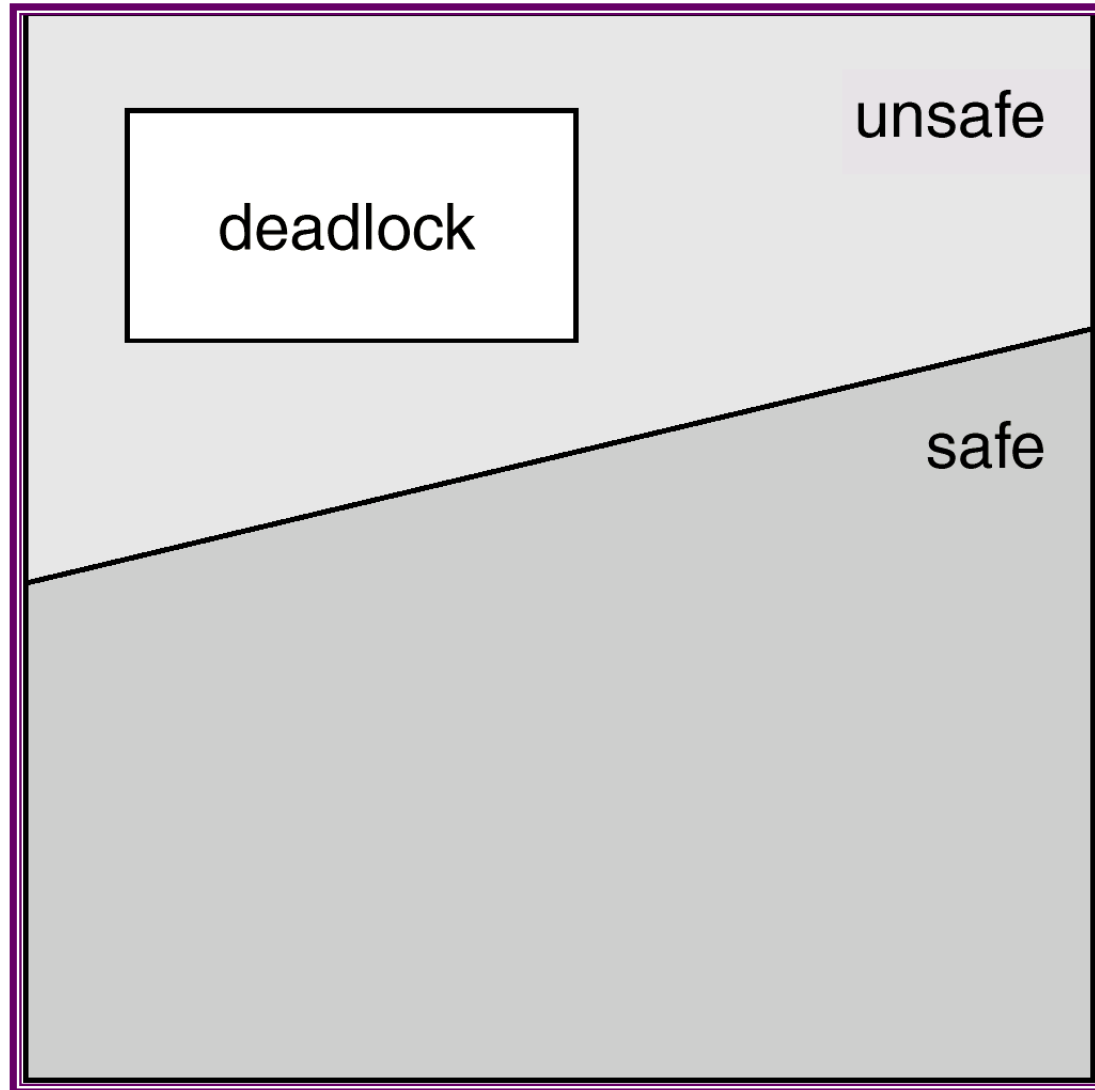
Bezbedno stanje (Safe State)

- Kada proces traži neki resurs, sistem mora da proceni da li će dodela tog resursa ostaviti sistem u **bezbednom stanju** (safe state)
- Sistem je u **bezbednom stanju** ako postoji **bezbedna sekvenca svih procesa**
- Sekvenca $\langle P_1, P_2, \dots, P_n \rangle$ je bezbedna ako za **svaki proces P_i** , resursi koje P_i može još tražiti, **moгу zadovoljiti iz:**
 - ☞ **trenutno raspoloživih resursa** + **resursi koji pripadaju svim P_j , sa $j < i$** .
 - 📄 (svi procesi startovani pre njega)
 - 📄 Ako resursi koje traži proces P_i nisu trenutno raspoloživi, tada će P_i da čeka dok prethodni procesi P_j ne završe svoje poslove
 - 📄 **Kada P_j završi, P_i može dobiti traženi resurs, izvršiti, vratiti deljeni resurs i završiti svoj posao**
 - 📄 **Kada je P_i završi, P_{i+1} može dobiti traženi resurs, itd**

Činjenično stanje

- Ako je sistem u **bezbednom stanju** \Rightarrow **nema zastoja**
- Ako sistem je u **ne-bezbednom stanju** \Rightarrow postoji **moгуćnost zastoja**
- **Izbegavanje** \Rightarrow
 - osigurava da sistem
 - **nikad ne uđe u ne-bezbedno stanje**

Bezbedno, Nebezbedno , Zastoj stanje



Resource-Allocation Graph Algorithm

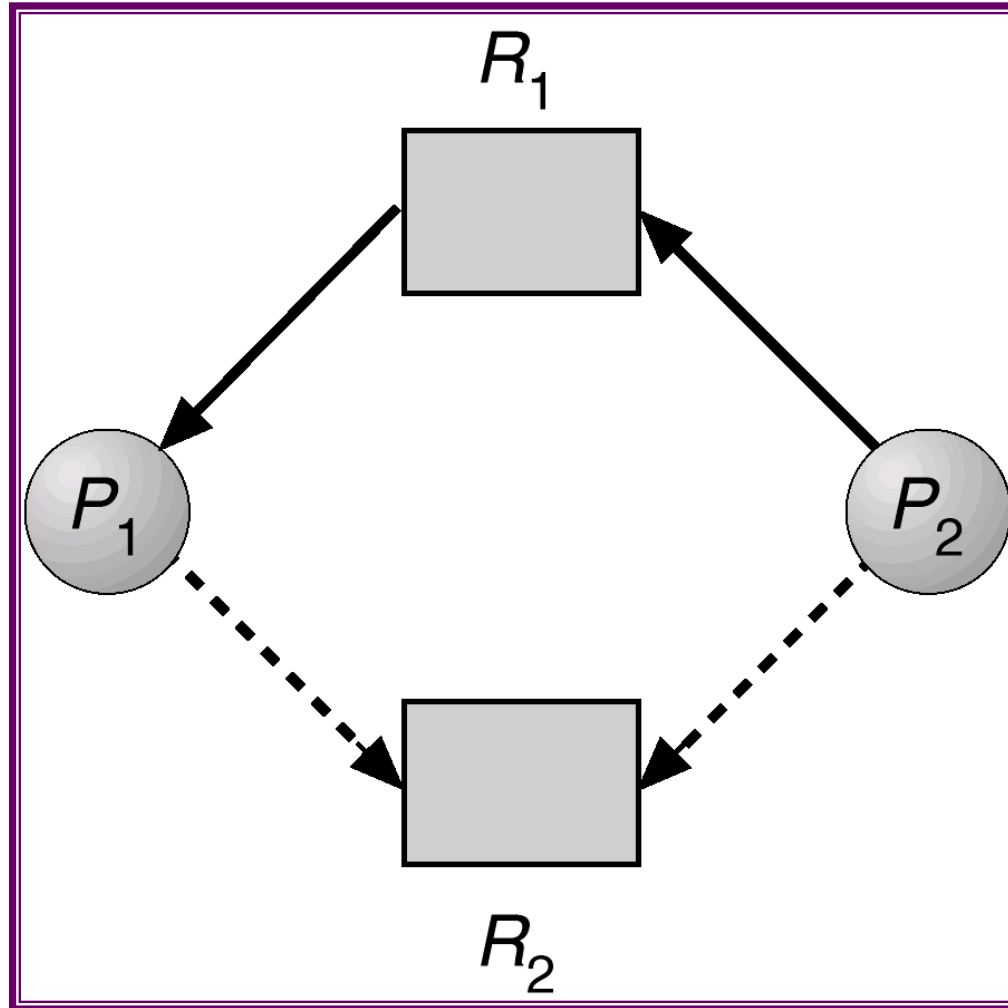
- *Mogući zahtev* $P_i \rightarrow R_j$ (claim edge)
- znači da proces P_j može zahtevati resurs R_j ;
- predstavljeno *isprekidanom linijom*

- *Mogući zahtev* preći će u *realni zahtev* kada proces zahteva resurs

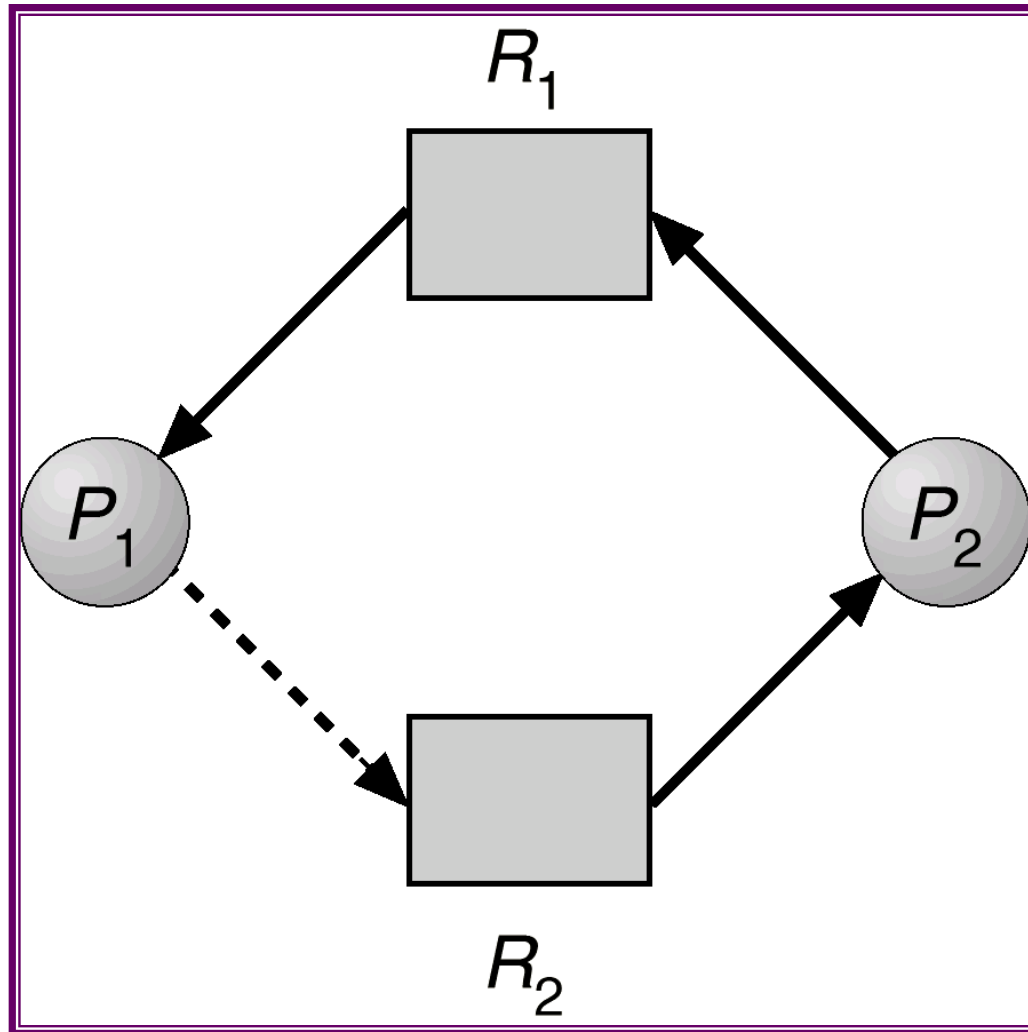
- Kada se takav resurs otpusti od procesa, strelica dodele opet prelazi u strelicu mogućih zahteva

- *Svi mogući zahtevi* za resurse moraju biti poznati unapred sistemu

Resource-Allocation Graph For Deadlock Avoidance



Unsafe State In Resource-Allocation Graph



ako P2 dobije
resurs

Bankarski algoritam

- **Više instanci**
- Svaki proces **unapred** mora deklarirati **maksimalan broj** instanci.
- Kada proces zahteva resurse **možda će trebati da sačeka**
- Kada proces dobije sve njegove resurse mora **mora da ih vrati u nekom konačnom vremenu**

Strukture podataka za bankarski algoritam

Neka je n = broj procesa, i m = broj resursa

- **Vektor raspoloživosti:** Vektor dužine m . Ako je element available $[j] = k$, tada je k instanci resursa R_j raspoloživo
- **Matrica maksimalnih zahteva:** Ako je $Max [i,j] = k$, tada proces P_i može tražiti najviše k instanci resursa R_j
- **Matrica alokacije:** Ako je $Allocation[i,j] = k$ tada je proces P_i trenutno dobio k instanci resursa R_j .
- **Matrica potreba:** Ako je $Need[i,j] = k$, tada proces P_i može tražiti još k instanci resursa R_j da završi njegov posao

$$Need [i,j] = Max[i,j] - Allocation [i,j].$$

Bezbednosni algoritam (Safety Algorithm)

1. Neka **Work** i **Finish** budu **vektori** dužine m i n , respektivno.
Inicijalizacija:

Work = Available

Finish [i] = false for $i = 1, 3, \dots, n$.

2. Pronalaženje procesa P_i koji može da zadovolji svoje potrebe:

(a) **Finish [i] = false**

(b) **Need_i ≤ Work** /*trazi manje od raspolozivih R*/

Ako nema takvih procesa idi na korak 4

3. **Work = Work + Allocation_i** **vraća resurse**
Finish[i] = true (proces završio)

Idi na korak 2

4. Ako je **Finish [i] == true** za svako i , tada je sistem u stabilnom stanju.

Algoritam za zahtevanje resursa

$Request_i$ = vektor trenutnih zahteva za proces P_i .

Ako je $Request_i[j] = k$ tada proces P_i traži k instanci resursa R_j .

1. Ako je $Request_i \leq Need_i$, idi na korak 2. Ukoliko ovo nije ispunjeno postavlja se greška, zato što proces traži više od onoga što je specificirao kao maksimalan zahtev.
2. Ako je $Request_i \leq Available$, idi na korak 3. U suprotnom proces P_i mora da čeka, resursi nisu raspoloživi

3. **Zamislite** da ste dodelili resurse procesu P_i po sledećoj formuli :

$$Available = Available - Request_i;$$

$$Allocation_i = Allocation_i + Request_i;$$

$$Need_i = Need_i - Request_i;$$

- **Ako je stanje stabilno** \Rightarrow resursi će biti **dodeljeni** procesu P_i .
- **Ako stanje nije stabilno** $\Rightarrow P_i$ mora da čeka, i resursi se vraćaju u staro stanje

Primer bankarskog algoritma

- 5 procesa P_0 do P_4 ;
- 3 resursa:
- A (10 instanci), B (5instanci), i C (7 instanci)
- Stanje sistema u trenutku T_0 :

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Primer

- Stanje matrice potreba **Need[]** se definiše kao:
- **Max[]** – **Allocation[]**

			<u>Need</u>			<u>Available</u>			
			A	B	C	A	B	C	
<u>Allocation</u>	<u>Max</u>	<u>Available</u>							
A B C	A B C	A B C							
P_0	0 1 0	7 5 3	3	3	2	P_0	7	4	3
P_1	2 0 0	3 2 2	1	2	2	P_1	1	2	2
P_2	3 0 2	9 0 2	6	0	0	P_2	6	0	0
P_3	2 1 1	2 2 2	0	1	1	P_3	0	1	1
P_4	0 0 2	4 3 3	4	3	1	P_4	4	3	1

- Sistem je u bezbednom stanju u sekvenci $\langle P_1, P_3, P_4, P_2, P_0 \rangle$ a uvek se polazi od procesa sa manjim zahtevima.

☞ (pogledajte najmanje zahteve $P_1.P_3..$)

Primer gde P_1 zahteva (1,0,2)

- Proverava se da li je $\text{Request} \leq \text{Available} \ (1,0,2) \leq (3,3,2) \Rightarrow \text{true}$.

	<u>Allocation</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 1	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

- Izvrši se bankarski algoritam i dokaže da postoji sekvenca $\langle P_1, P_3, P_4, P_0, P_2 \rangle$ koja će zadovoljiti uslove stabilnosti.
- Da li se može odobriti zahtev procesu P_4 (3,3,0)? (ne < od raspoloživog)
- Da li se može odobriti zahtev procesu P_0 (0,2,0)? (ne, ne-bezbedno)

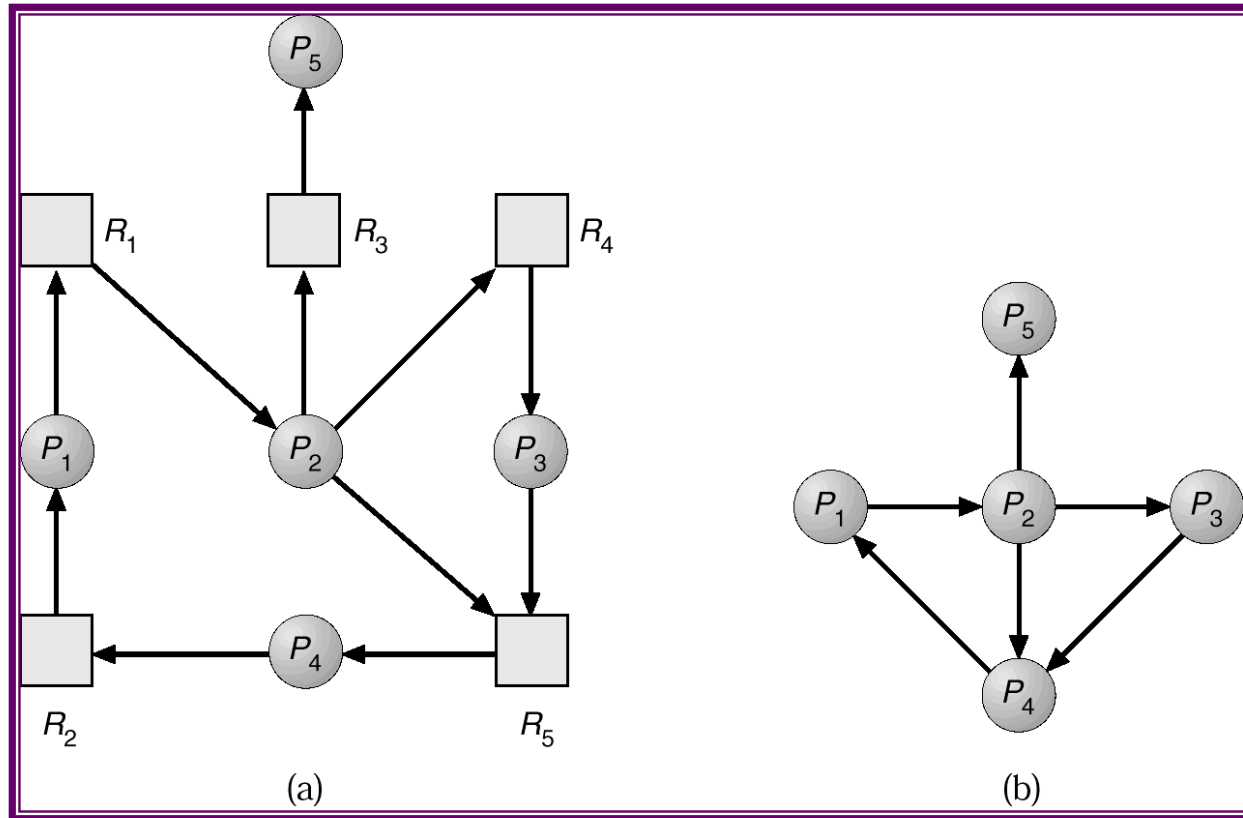
Detekcija zastoja (Deadlock Detection)

- Ako sistem uđe u stanje zastoja
- Algoritam za **detekciju** zastoja
- Algoritam za **oporavak**

Detekcija kad resursi imaju samo jednu instancu

- Pruža **wait-for** graf:
 - ☞ Čvorovi su **samo procesi**
 - ☞ $P_i \rightarrow P_j$ ako P_i čeka na P_j .
- Algoritam se **periodično pozove** da pretraži krugove u grafu
- Algoritam detektuje krugove u grafu
 - ☞ preko **n^2 operacija**,
 - ☞ gde je n broj strelica u grafu

Graf alokacije resursa i Wait-for graf



Graf alokacije resursa

Wait-for graf

Detekcija kad resursi imaju više instanci

- **Vektor raspoloživosti:** Vector dužine m . Ako je element $available[j] = k$, tada je k instanci resursa R_j raspoloživo .
- **Matrica alokacije:** Ako je $Allocation[i,j] = k$ tada je proces P_i trenutno dobio k instanci resursa R_j .
- **Matrica trenutnih zahteva:** Matrica ukazuje na trenutne zahteve procesa. Ako je $Request [i,j] = k$, tada proces P_i trenutno traži k instanci resursa R_j .

Algoritam za detekciju

1. Neka **Work** i **Finish** budu vektori dužine m i n , respektivno Inicijalizacija:
 - (a) **Work** = Available
 - (b) For $i = 1, 2, \dots, n$, if **Allocation_i ≠ 0**, then
Finish[i] = false; otherwise, Finish[i] = true
2. **Pronalazi se proces** koji može da zadovolji sve potrebe:
 - (a) **Finish[i] == false**
 - (b) **Request_i ≤ Work**

Ako takvi ne postoje, **idi na korak 4**

Algoritam za detekciju

3. $Work = Work + Allocation_i$
 $Finish[i] = true$
idi na korak 2
4. Ako je $Finish[i] == false$, za bilo koji proces i , $1 \leq i \leq n$, tada je sistem u stanju zastoja
Preciznije, ako je $Finish[i] == false$, tada je P_i u zastoju

Algoritam zahteva $m \times n^2$ operacija da bi ustanovio da li je sistemu stanju zastoja

Primer algoritma za detekciju

- Pet procesa od P_0 do P_4 ;
- tri resursa tipa
A (7 instanci), B (2 instance), i C (6 instanci)
- Stanje sistema u trenutku T_0 :

	<u>Allocation</u>			<u>Request</u>			<u>Available</u>		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0
P_1	2	0	0	2	0	2			
P_2	3	0	3	0	0	0			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

- Sekvenca $\langle P_0, P_2, P_3, P_1, P_4 \rangle$ koja bi dovela da bude rezultat $Finish[i] = true$ za sve procese i

Primer 2

- P_2 traži jednu instancu resursa C

	<u>Allocation</u>	<u>Request</u>	<u>Available</u>		<u>Request</u>
	A B C	A B C	A B C		A B C
P_0	0 1 0	0 0 0	0 0 0	P_0	0 0 0
P_1	2 0 0	2 0 2		P_1	2 0 1
P_2	3 0 3	0 0 0		P_2	0 0 1
P_3	2 1 1	1 0 0		P_3	1 0 0
P_4	0 0 2	0 0 2		P_4	0 0 2

- Stanje sistema?

- ☞ **P0** ima najmanje prohteva i ako bi završio svoje i vratio resurse, stanje raspoloživih resursa bi bilo **(0,1,0)** a iza toga nemamo proces koji može da zadovolji svoje potrebe
- ☞ Zastoj postoji, zaglavljani su procesi **P_1, P_2, P_3 i P_4**

Korišćenje algoritma za detekciju

- Kada, i **koliko često**, pozvati algoritam za detekciju zavisi od toga:
 - ☞ **Koliko često se zastoј dešava ?**
 - ☞ **Koliko procesa je zahvaćeno zastoјom ?**
- Zastoј se mora detektovati
- **jer su svi procesi i svi resursi u njima zaglavljени,**
- ali **zahteva prilično veliko vreme za detekciju**
- pa se ne sme pozivati veoma često

Oporavak od zastoja: Prekidanje procesa

- **Prekid svih** zaglavljenih procesa
- **Prekid samo jednog procesa** dok se ne razbije krug
- Po **kom redu** se prekidaju procesi?
 - ☞ Prioritet procesa
 - ☞ Koliko dugo proces već radi, odnosno šta mu je ostalo do završetka
 - ☞ Koje resurse proces koristi
 - ☞ Koje resurse proces traži da kompletira aktivnost
 - ☞ Koliko procesa je potrebno prekinuti
 - ☞ Da li su to interaktivni ili grupni procesi?

Oporavak od zastoja: Oduzimanje resursa

■ Biranje žrtve:

- ☞ najmanji gubitak

■ Oporavak procesa (Rollback):

- ☞ proces se vraća u bezbedno stanje,
- ☞ ponovo se pokreće iz tog stanja.

■ Zakucavanje (Starvation):

- ☞ neki proces možda uvek bude biran kao žrtva,
- ☞ zbog toga se neće dugo izvršiti

Kombinovan pristup za uklanjanje zastoja

- **Kombinacija tri osnovna** pristupa:

- ☞ **prevencija**

- ☞ **izbegavanje**

- ☞ **detekcija**

dopušta korišćenje najpovoljnijeg pristupa za svaki resurs u sistemu

- **Podeljeni** resursi u hierarhijski poređanim **klasama**

- **Koristi se najodgovarajuća tehnika** za uklanjanje zastoja u svakoj klasi

Zastoj u saobraćaju

