Operativni sistemi 1

Zastoj
(Deadlock)

Nemanja Maček

Sadržaj

- Sistemski model i osobine zastoja
- Metode upravljanja zastojem
- Izbegavanje zastoja
- Detekcija i oporavak od zastoja

Da li crveni automobil ima šanse da izađe iz gužve?

* Ilustracija preuzeta sa Web lokacije: https://workfolio.files.wordpress.com/2011/02/airport-e1297718137976.jpg

Resursi i instance

- Sistem se sastoji od **konačnog broja resursa** koje koriste konkurentni procesi.
- Svaki resurs se može sastojati od jedne ili više identičnih **instanci**.
  - Primer: server ima četiri procesora.
  - Procesor je resurs sa četiri instance.
- Ako resurs ima više instanci a proces zahteva jednu, alokacija bilo koje slobodne instance će zadovoljiti potrebe procesa.
- U normalnom režimu rada proces može da koristi resurs samo na sledeći način:
  - **Zahtev** (engl. *request*). U ovoj fazi proces zahteva resurs.
    - Ako zahtev za dodelom resursa ne može da se ispuni trenutno, proces mora da čeka dok se resurs ne oslobodi.
  - **Korišćenje** (engl. *use*). U ovoj fazi proces je dobio resurs i može ga slobodno koristiti.
  - **Oslobađanje** (engl. *release*). Nakon korišćenja resursa proces mora da oslobodi resurs.
- Drugim rečima: proces mora zahtevati resurs pre korišćenja i otpustiti resurs nakon korišćenja.

Šta je zastoj?

- U višeprocesnoj okolini više procesa se mogu međusobno takmičiti za konačan broj resursa.
- Proces P1 koji zahteva **neraspoloživ resurs** R1 ulazi u stanje WAIT i postaje blokiran.
- Pitanje: da li blokirani proces P1 može zauvek ostati u tom stanju?
  - Ova pojava je moguća.
  - Jedan proces može zahtevati više različitih resursa.
  - Scenario:
    - Pre ulaska u stanje WAIT procesu P1 je dodeljen drugi resurs R2.
    - Resurs R2 ostaje neraspoloživ za druge procese.
    - Resurs R1 je dodeljen na korišćenje drugom procesu P2.
    - Proces P2 u toku vremena prelazi u stanje čekanja na neraspoloživ resurs R2.
- U ovoj sitaciji niko ne oslobađa svoje resurse a traži nove – procesi ostaju zaglavljeni.
- Takva situacija naziva se **zastoj** (engl. *deadlock*).
  - Zastoj treba izbeći.
  - Sistem doveden u stanje zastoja se mora oporaviti.

Šta je zastoj?

- Primeri zastoja koji nisu vezani za računarski sistem:
  - Dve osobe, od kojih se jedna penje uz merdevine a druga spušta niz merdevine.
  - Dva voza koja se kreću jedan prema drugom istom šinom.
  - Zvezdasta raskrsnica u kojoj lako dolazi do zastoja.
  - ... i varijacije na temu.

Potrebni uslovi za pojavu zastoja

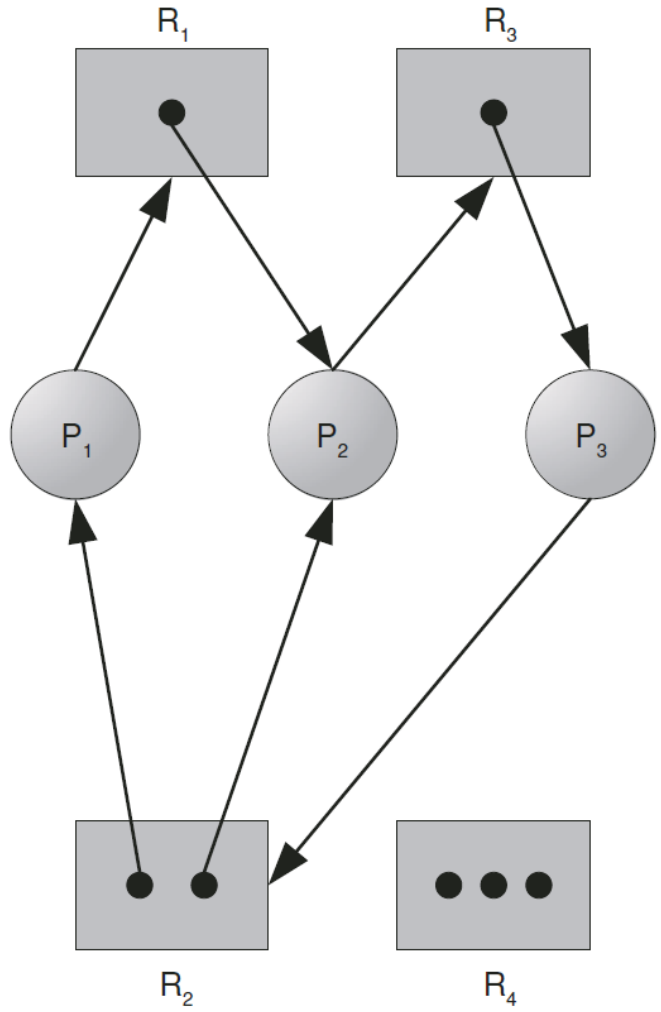
- **Međusobno isključenje**.
  - Samo jedan proces u jednom trenutku može koristiti resurs ili jednu njegovu instancu.
  - Drugi proces koji zahteva taj resurs (instancu) mora da čeka dok se resurs ne oslobodi.
- **Nema pretpražnjenja** („otimačine“ resursa).
  - Resurs se ne može nasilno oduzeti i predati drugom procesu.
  - Proces koji ga koristi mora da završi posao i oslobodi resurs.
- Uslov **zadržavanja resursa i čekanja na drugi** (engl. *hold and wait*).
  - Proces drži jedan resurs.
  - Proces istovremeno čeka na dobijanje resursa koga koristi neki drugi proces.
- **Kružno čekanje** (engl. *circular wait*).
  - Postoji skup procesa $\{P_0, P_1, \ldots, P_0\}$ koji čekaju na resurse u kružnom poretku.
    - Proces P_0 čeka na resurs koga drži proces P_1.
    - Proces P_1 čeka na resurs koga drži proces P_2 …
    - Proces P_{n-1} čeka na resurs koga drži proces P_n.
    - Proces P_n čeka na resurs koga drži proces P_0.

Graf dodeljenih resursa

- Graf dodeljenih resursa (engl. *resource allocation graph* se sastoji od skupa objekata i skupa strelica E.
- Skup objekata se sastoji od:
  - Skupa svih **aktivnih procesa** u sistemu: P = {P1, P2, ..., Pn}
  - Skupa svih **raspoloživih resursa**: R = {R1, R2, ..., Rm}.
- Skup strelica se sastoji od:
  - **Strelica zahteva** (Pi $\rightarrow$ Rj). Proces Pi zahteva jednu instancu resursa Rj i čeka na nju.
    - Strelica se dodaje u graf uvek kada proces traži resurs.
  - **Strelica alokacije** (Rj $\rightarrow$ Pi). Resurs Rj dodeljen procesu Pi.
    - Strelica se dodaje u graf uvek kada se resurs dodeli procesu.
- Ako graf **ne sadrži kružni tok**, zastoja sigurno nema.
- Ako graf **sadrži bar jedan kružni tok** moguće su dve situacije:
  - ako svi resursi u kružnom toku sadrže **tačno jednu instancu**, zastoj se dogodio;
  - ako resursi u kružnom toku sadrže više instanci može se dogoditi da zastoja nema.

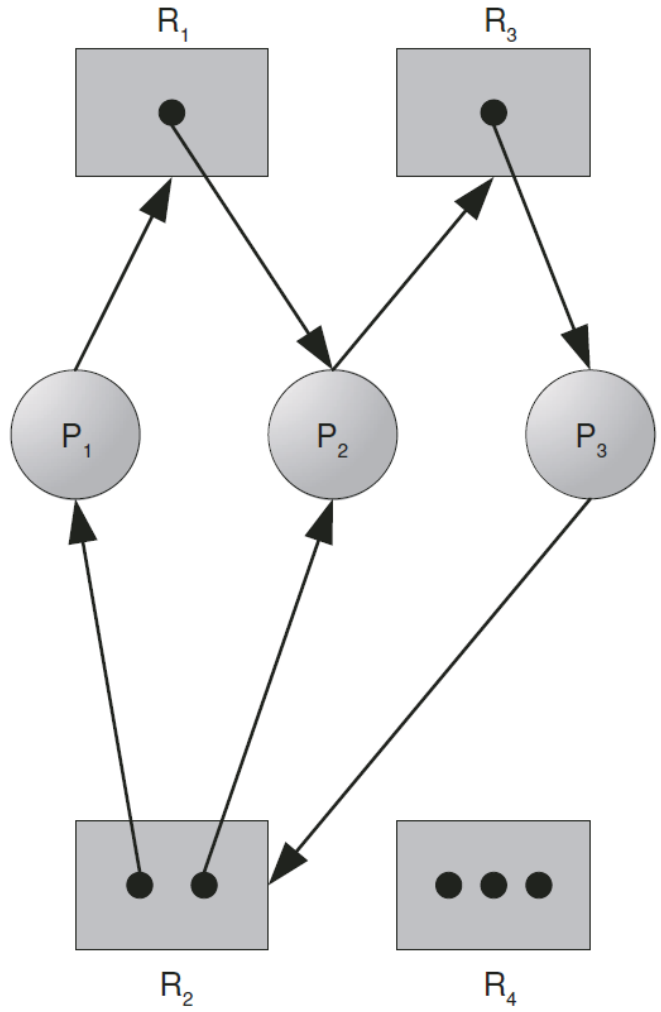
Graf dodeljenih resursa (primer)

- P = {P1, P2, P3}
- R = {R1, R2, R3, R4}.
- Resursi R1, R2, R3 i R4 redom imaju 1, 2, 1 i 3 instance.
- E = {P1 → R1, P2 → R3, P3 → R2,
  R1 → P2, R2 → P1, R2 → P2, R3 → P3}
- Da li je zastoj moguć?



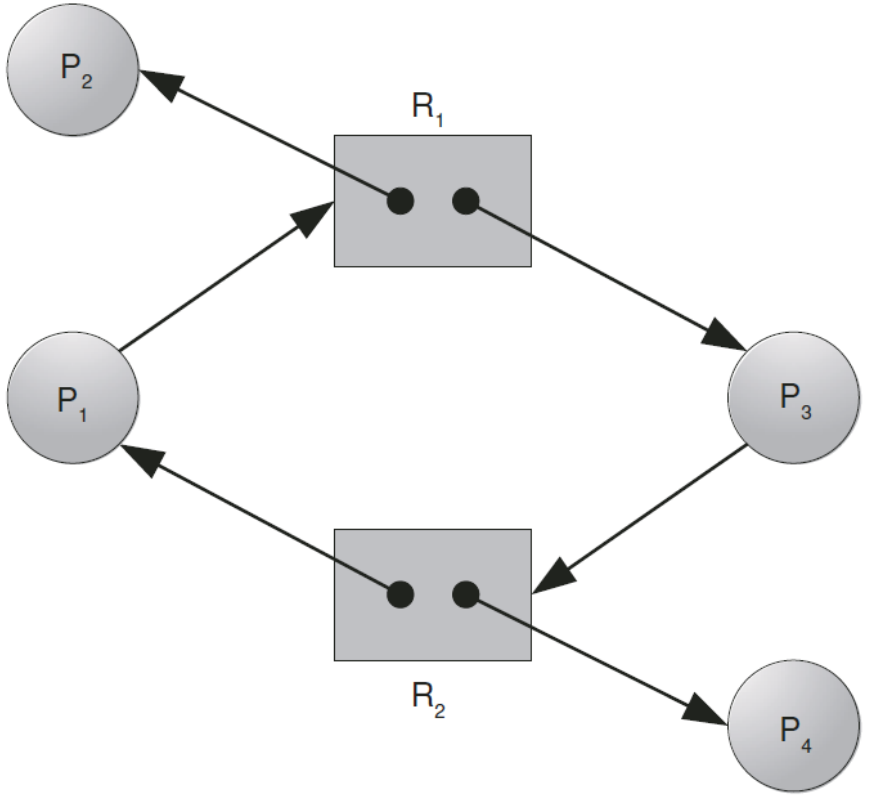
Graf dodeljenih resursa (primer)

- Na grafu postoje dve kružne putanje koje mogu izazvati zastoj:
- (1) P1 → R1 → P2 → R3 → P3 → R2 → P1
- (2) P2 → R3 → P3 → R2 → P2
- Procesi P1, P2 i P2 su u zastoju!
  - Proces P2 čeka na R3 koga drži P3.
  - Proces P3 traži R2 koga drže P1 i P2.
  - Proces P2 je blokiran sa procesom P3 oko resursa R3.
  - Proces P1 traži R1 koga drži P2.
- Da li bi do zastoja došlo u slučaju da:
  - Resurs R2 ima tri instance?
  - Resurs R3 ima dve instance?



Graf dodeljenih resursa (primer)

- Kružni tok na grafu ne znači obavezno zastoj!
- Uočavamo kružni tok: P1 → R1 → P3 → R2 → P1.
- Oba resursa imaju više instanci tako da nema zastoja.
- Mogući scenario:
  - Proces P4 može da završi svoj posao i da oslobodi jednu instancu resursa R2.
  - Oslobođena instance resursa R2 se može dodeliti procesu P3.
  - Time se prekida krug i eliminiše zastoj.



Metode upravljanja zastojem

- Problem upravljanja zastojem može se rešavati na tri načina:
- **Sprečavanjem i izbegavanjem** zastoja (engl. *deadlock prevention and avoidance*).
  - Metode koje obezbeđuju da sistem nikada ne uđe u stanje zastoja,
- **Detekcijom i oporavkom** od zastoja (engl. *deadlock detection and recovery*).
  - Metode koje dozvoljavaju sistemu da uđe u stanje zastoja.
  - To stanje naknadno detektuju i oporavljaju sistem.
- **Ignorisanjem problema** zastoja.
  - Neki OS se pretvaraju da se te pojave ne dešavaju.
  - U tom slučaju se ne koriste prethodne dve metode.

Sprečavanje pojave zastoja

- Zastoj se dešava ako su četiri uslova istovremeno ispunjena.
- Zastoj se sprečava metodeana koje obezbeđuju da se **najmanje jedan uslov ne ispuni**!
- **Međusobno isključenje**.
  - Obavezan uslov za sve nedeljive resurse.
  - Za nedeljive resurse nije obavezan.
    - Jedan metod sprečavanja je izbegavanje uslova međusobnog isključenja za sve deljive resurse.
- **Uslov zadržavanja resursa i čekanja na drugi**.
  - Može se izbeći na dva načina:
    - Svaki proces mora da traži i alocira sve svoje resurse pre početka izvršavanja.
    - Proces može da traži resurs samo pod uslovom da ne drži ni jedan drugi.
      - To znači da proces može koristiti samo jedan resurs u jednom trenutku
      - Time se značajno smanjuje korišćenje U/I resursa.

Sprečavanje pojave zastoja

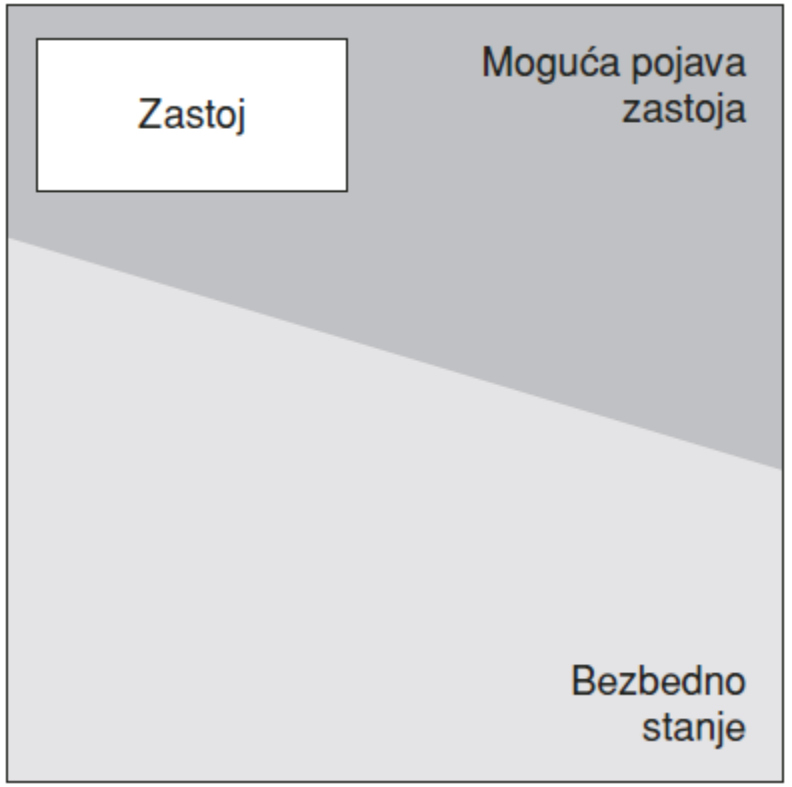
- **Nema pretpražnjenja**.
  - Ovaj uslov se može poništiti korišćenjem sledećeg protokola:
    - Proces koji traži novi neraspoloživ resurs otpušta sve zauzete resurse i prelazi u WAIT.
    - Proces može preći u stanje READY tek kada povrati sve resurse koje je posedovao i resurse koje je tražio na korišćenje.
    - U ovom slučaju proces ne može preći u stanje WAIT sa zauzetim resursima.
  - Drugi način:
    - Nasilna „otimačina“ resursa od drugih procesa koji su blokirani a drže te resurse.
- **Kružno čekanje**.
  - Može se izbeći na sledeći način:
    - Svakom resursu se dodeli broj *N* iz skupa prirodnih brojeva *N*=*F*(Ri)
    - Procesi mogu zahtevati resurse u strogo rastućem redu.
    - Proces koji je zahtevao resurs Ri može zahtevati resurs Rj samo ako je *F*(Rj)>*F*(Ri).
    - Poslednji proces u nizu ne može da se vrati unazad i traži resurs koga drži prvi proces.

Izbegavanje zastoja

- Izbegavanje zastoja je moguće ako OS **unapred** ima informacije o resurima koje će procesi zahtevati.
- Sistem koji zna šta procesi žele može da napravi redosled opsluživanja zahteva tako da se izbegne zastoj.
- Najprostija šema izbegavanja zastoja:
  - Od svakog procesa se traži da deklariše **najveći broj potrebnih resursa** svakog tipa.
  - Na osnovu toga se može konstruisati algoritam koji će sprečiti stanje zastoja.
  - Algoritam dinamički ispituje **trenutno stanje dodeljenih resursa**:
    - broj raspoloživih resursa,
    - broj dodeljenih resursa i
    - broj maksimalno traženih resursa u jednom trenutku vremena.
  - Tako se osigurava da sistem nikada ne uđe u stanje kružnog čekanja.

Bezbedno stanje

- Kada proces traži neki resurs, sistem mora da proceni da li će dodela tog resursa ostaviti sistem u bezbednom stanju (engl. *safe state*).
- Stanje je **bezbedno** ako sistem može može alocirati resurse svakom procesu u nekom poretku i još uvek izbeći zastoj.
  - Bezbedno stanje je stanje bez zastoja.
- Stanje koje **nije bezbedno**:
  - ne mora biti zastoj;
  - može težiti ka zastoju;
  - u tom stanju procesi mogu lako dovesti sistem u zastoj.



Bezbedno stanje

- Formalno: sistem je u bezbednom stanju ako postoji bezbedna sekvenca svih procesa.
- Sekvenca <P1, P2, ..., Pn> je **bezbedna** ako za svaki proces Pi u svakom ternutku važi:
  - Resursi koje Pi može još tražiti mogu da se zadovolje iz trenutno raspoloživih resursa.
  - To uključuje i resurse koji pripadaju procesima Pj pokrenutim pre procesa Pi (j<i).
  - Ako resursi nisu trenutno raspoloživi Pi čeka da svi Pj završe poslove i oslobode resurse.
- Primer:
  - Tri procesa (P0, P1 i P2) i 12 resursa istog tipa.
  - U početnom trenutku: P0 dobio 5 resursa, P1 dobio 2 resursa, P2 dobio 2 resursa.
  - Maksimalne potrebe procesa P0, P1 i P2 su redom 10, 4 i 9 resursa.
  - Da li je sekvenca <P1, P0, P2> bezbedna?

Bezbedno stanje

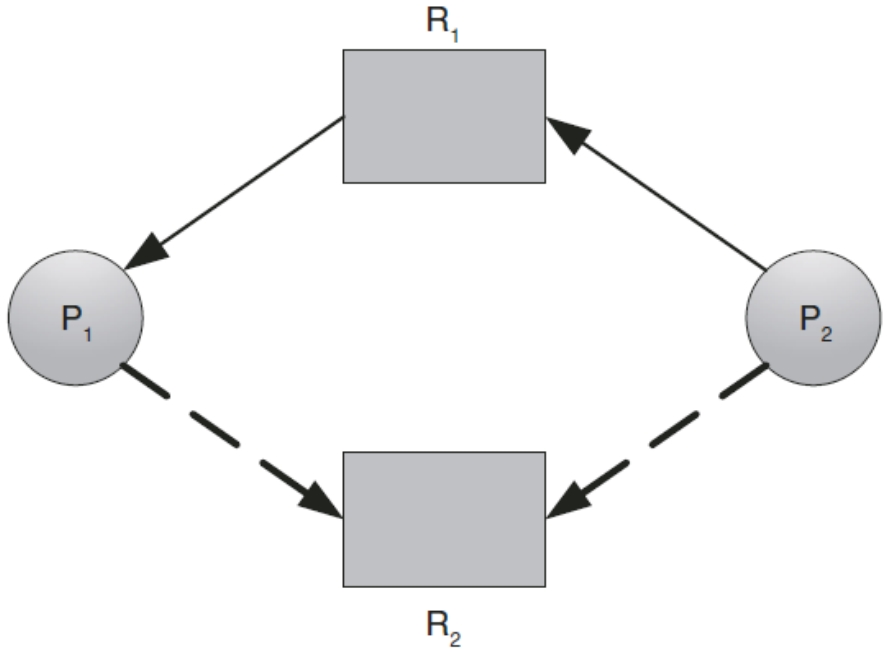
- Sekvenca <P1, P0, P2> je bezbedna!
  - Imamo 3 slobodna resursa.
    - P1 može da zadovolji svoje maksimalne potrebe (uzima još 2 resursa).
    - P1 završava posao i oslobađa 4 resursa.
  - Ostaje 5 slobodnih resursa.
    - P0 može da zadovolji svoje maksimalne potrebe (uzima još 5 resursa).
    - P0 završava posao i oslobađa 10 resursa.
  - Ostaje 10 slobodnih resursa.
    - P2 može da zadovolji svoje maksimalne potrebe (uzima još 7 resursa).
    - P2 završava posao i oslobađa 9 resursa.
- Pitanje: da je u početnom trenutku P0 dobio 5 resursa, P1 dobio 2 resursa, P2 dobio **3 resursa**, da li bi sekvenca <P1, P0, P2> bila bezbedna?
  - Ne.
  - Nakon što P1 završi aktivnosti ostaje 4 slobodna resursa.
  - Ako P0 zatraži svih 10 resursa (max potrebe) doći će do zastoja.

Graf dodele resursa za izbegavanje zastoja

- Uvodi se **strelica mogućih zahteva** (engl. *claim edge*).
- Ova strelica znači da proces može zahtevati resurs u nekom budućem trenutku.
- U grafu se predstavlja isprekidanom linijom.
- Zahtevanje i oslobađanje resursa:
  - U trenutku u kome proces zahteva resurs:
    - Mogući zahtev postaje stvarni zahtev za resurs.
    - Strelica mogućeg zahteva menja se strelicom zahteva.
  - Kada proces oslobodi resurs strelica dodele menja se strelicom mogućih zahteva.
- Svi mogući zahtevi moraju biti poznati pre nego što proces otpočne izvršavanje.
- To će omogučiti da se nacrta **graf sa mogućim zahtevima procesa**.
  - Na bazi tog grafa se odlučuje da li će sistem dozvoliti zahtev ili ne.
  - Ukoliko se ispunjenjem zaheva formiraju kružni tokovi, sistem se može dovesti u nebezbedno stanje pa se procesu ne ispunjava zahtev.

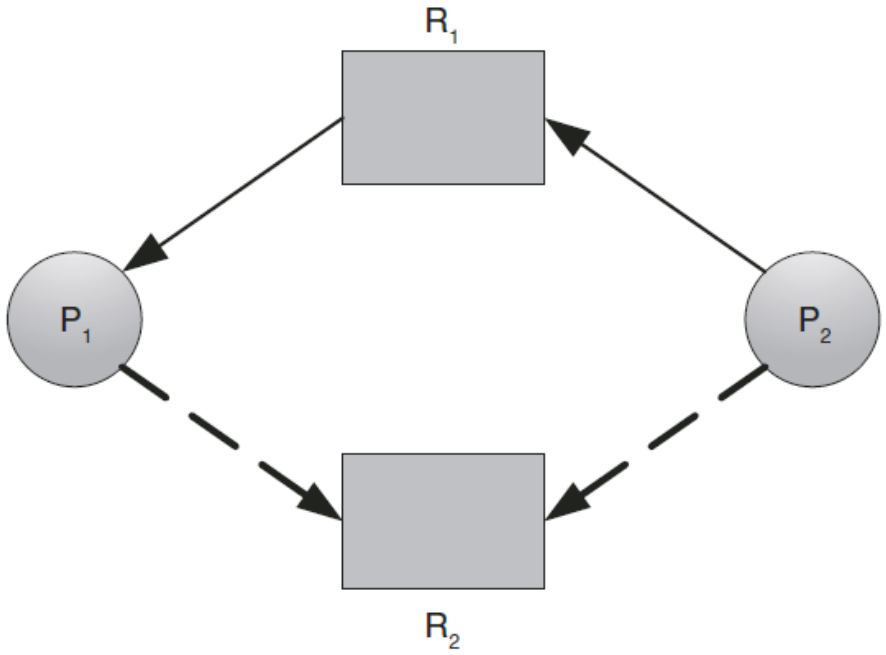
Graf dodele resursa za izbegavanje zastoja (primer)

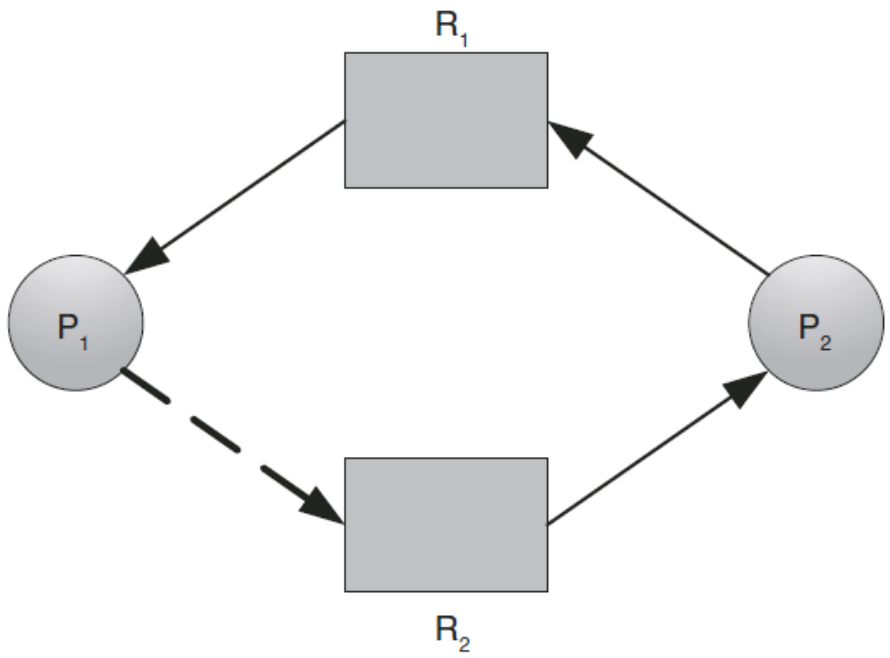
- Resurs R2 je slobodan, ali da postoje mogući zahtevi od procesa P1 i P2.
- Da li je zastoj moguć?



Graf dodele resursa za izbegavanje zastoja (primer)

- Ako bi proces P2 izdao zahtev za korišćenjem R2, sistem to ne bi smeo da dozvoli.
- U slučaju da sistem to dozvoli i dodeli resurs R2 procesu P2 pojaviće se kružni tok.
- Ako i P1 zatraži R2, dolazi do zastoja.





Bankarski algoritam

- U čemu je problem sa grafom koji sadrži strelice mogućih zahteva?
- Nije podesan za resurse koji imaju više instanci!
- U tim situacijama za izbegavanje zastoja koristi se **bankarski algoritam** (engl. *Banker's algorithm*).
- Algoritam se može primeniti ukoliko su ispunjeni sledeći uslovi:
  - Resursi imaju uglavnom više instanci.
  - Svaki proces unapred deklariše najveći broj instanci svakog resursa koji želi da koristi.
  - Kada proces zahteva resurse sistem procenjuje da li će posle toga ostati u stabilnom stanju.
    - Ako ostaje u stabilnom stanju proces će dobiti resurs.
    - U suprotnom proces mora da sačeka da drugi procesi oslobode neke resurse.
  - Proces koji dobije resurse mora da ih vrati u nekom konačnom vremenu.

* Ilustraija preuzeta sa Web sajta: https://clipartfest.com/

Bankarski algoritam

- Broj procesa: n. Broj resursa: m.
- Strukture podataka za bankarski algoritam:
  - **Vektor rapoloživosti**: available[j], $j \in [0,m]$.
    Ako je available[j]=k, tada je k instanci resursa Rj raspoloživo,
  - **Matrica makimalnih zahteva**: max[n,m].
    Ako je max[i,j] = k, tada proces Pi može tražiti ukupno k instanci resursa Rj.
  - **Matrica alokacije**: allocation[n,m].
    Ako je allocation[i,j] = k, tada je proces Pi trenutno dobio k instanci resursa Rj.
    Vrsta matrice alokacije je vektor allocation[i] (svi resursi koje je proces Pi dobio).
  - **Matrica potreba**: need[n,m].
    Ako je need[i,j] = k, tada proces Pi može tražiti još k instanci resursa Rj.
    Vrsta matrice potreba je vektor need[i] (svi resursi koje proces Pi može tražiti).
- Važi jednakost: need[i,j] = max[i,j] – allocation[i,j].
- Uvodimo definiciju: za dva vektora X i Y od n elemenata važi $X \leq Y$ ako je $X[i] \leq Y[i]$ za svako $i \in [1,n]$.

Bankarski algoritam

- Neka su work i finish vektori dužine m i n, respektivno.
- Sledeći algoritam **određuje da li se sistem nalazi u bezbednom stanju**.

1. Inicijalizacija:
   work = available (vektor raspoloživih resursa)
   finish[i] = 0 za $i \in [1,n]$ (opisuje završetak procesa i)
2. Pronalaženje procesa Pi koji može da zadovolji svoje potrebe, odnosno procesa za koji važi:
   (2.a) finish[i] = 0
   (2.b) need[i] ≤ work
   Ako nema takvih procesa idi na korak 4.
3. Procesu se dodeljuju svi potrebni resursi nakon čega on završava aktivnost i vraća ih u sistem.
   work = work + allocation[i] (oslobađanje resursa)
   finish[i] = 1
   Ići na korak 2.
4. Ako je finish[i] = 1 za sve i, svi procesi mogu da završe posao pa je sistem u stabilnom stanju.

- Algoritam zahteva $m \cdot n^2$ operacija kako bi proverio da li je sistem u stabilnom stanju.

Bankarski algoritam (primer provere bezbednog stanja)

- U sistemu se nalaze tri procesa (P0, P1 i P2) i resurs A sa 12 instanci.
- Stanje sistema je dato sledećom tabelom.

Process	allocation	max	need
P0	5	10	5
P1	2	4	2
P2	2	9	7

- Resurs A ima **3 slobodne instance**.
- Da li je sistem u bezbednom stanju?

Bankarski algoritam (primer provere bezbednog stanja)

- Uvek se polazi od procesa sa manjim zahtevima.
- Kasnije se rešavaju problemi najzahtevnijih procesa!
- Proces P1 najpre uzima još dve instance resursa, a zatim ih vraća.

Process	allocation	max	need	available
P0	5	10	5	1
P1	4	4	0	
P2	2	9	7	

Process	allocation	max	need	available
P0	5	10	5	5
P1	FINISH			
P2	2	9	7	

Bankarski algoritam (primer provere bezbednog stanja)

- Nakon toga proces P0 uzima još pet instanci resursa, a zatim ih vraća.

Process	allocation	max	need	available
P0	10	10	0	0
P1	FINISH			
P2	2	9	7	

Process	allocation	max	need	available
P0	FINISH			10
P1	FINISH			
P2	2	9	7	

Bankarski algoritam (primer provere bezbednog stanja)

- Na kraju, proces P2 uzima još sedam instanci resursa, koje po završetku vraća.
- Sekvenca <P1, P0, P2> doveće do zadovoljenja potreba svih procesa.
- Dakle, sistem je u bezbednom stanju.

Process	allocation	max	need	available
P0	FINISH			3
P1	FINISH			
P2	9	9	7	

Process	allocation	max	need	available
P0	FINISH			12
P1	FINISH			
P2	FINISH			

Bankarski algoritam

- Kada proces zahteva resurs, primenjuje se **algoritam za obradu zahteva i dodelu resursa**.
- Sa $request_i$ označićemo vektor trenutnih zahteva procesa Pi:
  Ako je $request_i[j]=k$ tada proces Pi traži k instanci resursa Rj.
- Pre ulaska u algoritam za izbegavanje zastoja operativni sistem proverava:
  - Da li proces traži više od onoga što je specificirao ($request_i \geq need_i$)?
    - Ako traži, prijavljuje se greška i odbija se zahtev.
  - Da li su resursi raspoloživi ($request_i \leq available$)?
    - Ako nisu, OS uvodi proces u stanje čekanja.
- Ako je $request_i < need_i$ i $request_i \leq available$ obavlja se kvazi-dodela resursa:
  - $available = available - request_i$ (broj raspoloživih resursa)
  - $allocation_i = allocation_i + request_i$ (broj dodeljenih resursa)
  - $need_i = need_i - request_i$ (potrebe procesa)
- Nakon zamišljene dodele resursa ispituje se da li sistem ostaje u bezbednom stanju.
- Ako ostaje, sistem će procesu Pi dodeliti resurse (alokacija postaje stvarna).
- U protivnom proces Pi mora da čeka.

Bankarski algoritam (primer)

- U sistemu se nalazi pet procesa (P0 do P4) i četiri resursa (A, B, C i D).
- U trenutku t_0 je available=(1,5,2,0).
- Stanje sistema u trenutku t_0 opisano je sledećom tabelom:

	allocation				max				need			
Process	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
P0	0	0	1	2	0	0	1	2	0	0	0	0
P1	1	0	0	0	1	7	5	0	0	7	5	0
P2	1	3	5	4	2	3	5	6	1	0	0	2
P3	0	6	3	2	0	6	5	2	0	0	2	0
P4	0	0	1	4	0	6	5	6	0	6	4	2

- Da li je sistem u bezbednom stanju?
- Da li će sistem da odobri zahtev $request_1 = (0, 4, 2, 0)$?

Bankarski algoritam (primer)

- Sistem je u bezbednom stanju, jer sledeća sekvenca procesa zadovoljava uslove stabilnosti:
  - P0 uzima (0,0,0,0) a zatim vraća (0,0,1,2) → available=(1,5,3,2).
  - P3 uzima (0,0,2,0) a zatim vraća (0,6,5,2) → available=(1,11,6,4).
  - P2 uzima (1,0,0,2) a zatim vraća (2,3,5,6) → available=(2,14,11,8).
  - P1 uzima (0,7,5,0) a zatim vraća (1,7,5,0) → available=(3,14,11,8).
  - P4 uzima (0,6,4,2) a zatim vraća (0,6,5,6) → available=(3,14,12,12).

Bankarski algoritam (primer)

- Najpre se proverava da li je $request_1 \leq available$.
  - Pošto je (0,4,2,0) ≤ (1,5,2,0) uslov je ispunjen.
- Obavlja se kvazi-dodela resursa na osnovu zahteva.
  - Procesu P1 dodeljujemo (0,4,2,0).
  - Nakon dodele je available=(1,1,0,0).

	allocation				max				need			
Process	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
P0	0	0	1	2	0	0	1	2	0	0	0	0
P1	1	4	2	0	1	7	5	0	0	3	3	0
P2	1	3	5	4	2	3	5	6	1	0	0	2
P3	0	6	3	2	0	6	5	2	0	0	2	0
P4	0	0	1	4	0	6	5	6	0	6	4	2

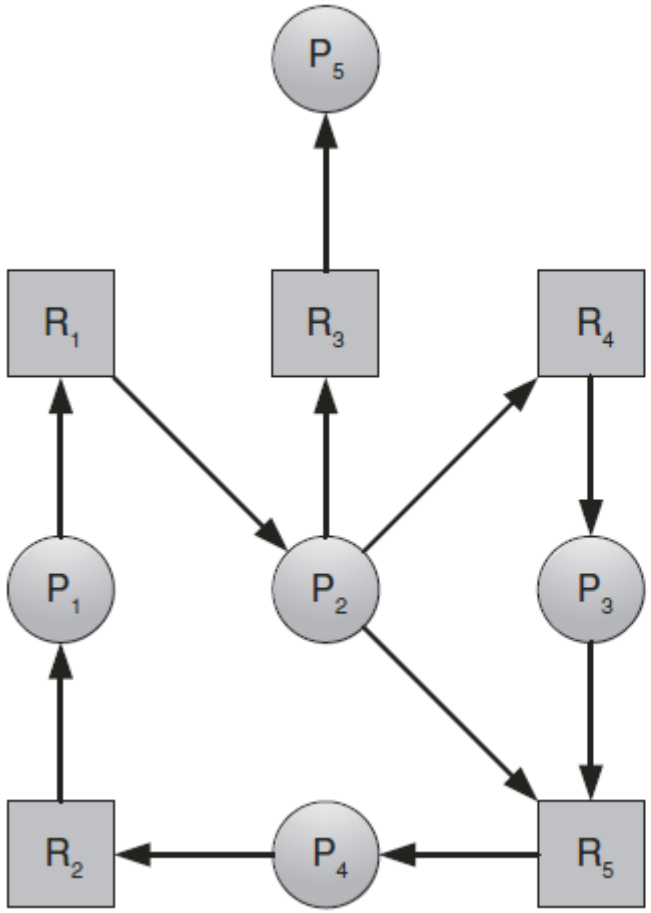
Bankarski algoritam (primer)

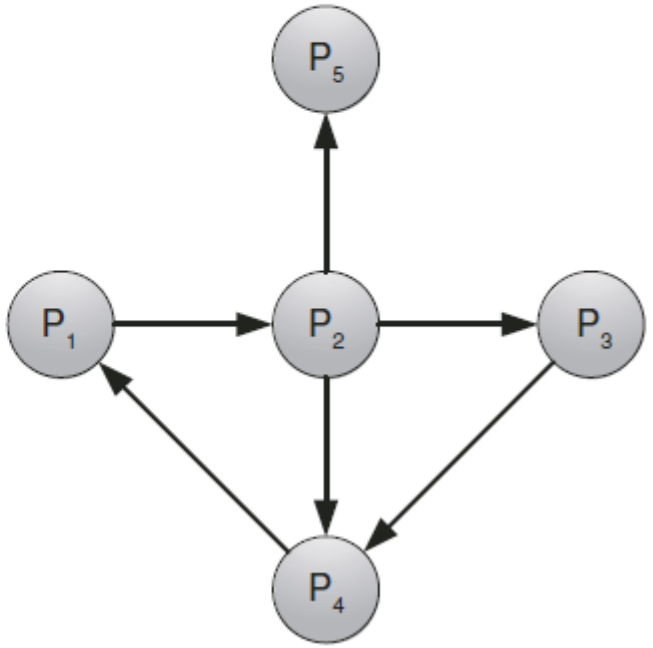
- Sistem nakon kvazi-dodele ostaje u bezbednom stanju, jer sledeća sekvenca procesa zadovoljava uslove stabilnosti:
  - P0 uzima (0,0,0,0) a zatim vraća (0,0,1,2) → available=(1,1,1,2).
  - P2 uzima (1,0,0,2) a zatim vraća (2,3,5,6) → available=(2,4,6,6).
  - P3 uzima (0,0,2,0) a zatim vraća (0,6,5,2) → available=(2,10,9,8).
  - P1 uzima (0,3,3,0) a zatim vraća (1,7,5,0) → available=(3,14,11,8).
  - P4 uzima (0,6,4,2) a zatim vraća (0,6,5,6) → available=(3,14,12,12).
- Sistem **odobrava** zahtev za dodelom resursa.

Detekcija i oporavak od zastoja

- Ako sistem ne primenjuje algoritme za sprečavanje ili izbegavanje zastoja, tada su zastoji sasvim mogući, čak i vrlo verovatni.
- U tom slučaju sistem mora da obezbedi dve stvari:
  - Algoritam za **detekciju zastoja** (koji treba da ispita da li se zastoj dogodio).
  - Algoritam za **oporavak iz stanja zastoja**.
- U slučaju da svi resursi imaju samo jednu instancu za detekciju zastoja se koristi **graf čekanja na resurse** (engl. *wait-for graph*).
  - Konstruiše se od grafa dodeljenih resursa.
    - Čvorovi grafa su samo procesi (nema resursa).
    - Strelice se crtaju samo između procesa koji čekaju jedan drugog za resurse.
  - Postojanje zastoja utvrđuje se ispitivanjem postojanja **kružnih tokova**.
    - Ako kružni tok postoji to je zastoj.
    - Algoritam se periodično poziva da ispita postojanje kružnih tokova.
    - Za ispitivanje je potrebno n^2 operacija, pri čemu je n broj strelica u grafu čekanja.

Graf čekanja na resurse





Detekcija zastoja u slučaju da resursi imaju više instanci

- Ukoliko resursi imaju više instanci za detekciju zastoja se koristi algoritam sličan bankarskom.
- Strukture podataka:
  - **Vektor rapoloživosti**: available[j], $j \in [0,m]$.
    Ako je available[j]=k, tada je k instanci resursa Rj raspoloživo,
  - **Matrica alokacije**: allocation[n,m].
    Ako je allocation[i,j] = k, tada je proces Pi trenutno dobio k instanci resursa Rj.
  - **Matrica trenutnih zahteva**: request[n,m].
    Ako je request[i,j] = k, tada proces Pi trenutno traži k instanci resursa Rj.

Detekcija zastoja u slučaju da resursi imaju više instanci

- Neka su work i finish vektori dužine m i n, respektivno.
- Sledeći algoritam **određuje da li se sistem nalazi u stanju zastoja**.

1. Inicijalizacija:
   work = available
   finish[i] = {0, ako je $allocation_i \neq 0$; inače je 1} za $i \in [1,n]$
2. Pronalaženje procesa Pi koji može da zadovolji svoje potrebe, odnosno procesa za koji važi:
   (2.a) finish[i] = 0
   (2.b) $request_i \leq work$
   Ako nema takvih procesa idi na korak 4.
3. U ovom koraku se resursi procesa vraćaju u sistem.
   work = work + $allocation_i$ (oslobađanje resursa)
   finish[i] = 1
   Ići na korak 2.
4. Ako je finish[i] = 0 za sve $i \in [1,n]$, sistem je u zastoju (preciznije proces Pi je u zastoju).

- Algoritam zahteva $m \cdot n^2$ operacija kako bi proverio da li je sistem u stanju zastoja.

Detekcija zastoja u slučaju da resursi imaju više instanci (primer)

- Sistem sa pet procesa (P0 do P4) i tri resursa: A (7 instanci), B (2 instance), C (6 instanci).
- U trenutku t_0 nema raspoloživih resursa, odnosno available = (0,0,0).
- Stanje sistema u trenutku t_0 opisano je sledećom tabelom.

vekt.	**allocation**			**request**		
res.	**A**	**B**	**C**	**A**	**B**	**C**
P0	0	1	0	0	0	0
P1	2	0	0	2	0	2
P2	3	0	3	0	0	0
P3	2	1	1	1	0	0
P4	0	0	2	0	0	2

Detekcija zastoja u slučaju da resursi imaju više instanci (primer)

- Kada se primeni algoritam za detekciju:
  - Dokazuje se da postoji sekvenca <P0, P2, P3, P1, P4> koja dovodi do rezultata finish[i]=1 za sve procese.
  - To znači da sistem **neće** biti doveden u stanje zastoja.
- Interesantno:
  - Sistem u trenutku t_0 nema raspoloživih resursa.
  - Počevši od procesa koji najmanje traže (P0 i P2) može se obaviti dodela resursa bez dovođenja do zastoja.

Detekcija zastoja u slučaju da resursi imaju više instanci (primer)

- Ukoliko uvedemo malu izmenu: proces P2 traži jednu instancu resursa C.
  - Proces P0 ima najmanje prohteve, završava, vraća resurse → available=(0,1,0).
  - Nakon toga nemamo proces koji može da zadovolji svoje potrebe.
  - Dakle, sistem je u stanju zastoja - procesi P1, P2, P3, i P4 su zaglavljeni.

vekt.	**allocation**			**request**		
res.	**A**	**B**	**C**	**A**	**B**	**C**
P0	0	1	0	0	0	0
P1	2	0	0	2	0	2
P2	3	0	3	0	0	**1**
P3	2	1	1	1	0	0
P4	0	0	2	0	0	2

Oporavak od zastoja

- Posle detekcije zastoja potrebno je izvršiti oporavak.
- Dve metode za oporavak od zastoja koje se najčešće koriste su:
  - **Prekid izvršenja** procesa u zastoju.
    - Osobađaju se svi resursi koje je zauzimao proces koji se prekida.
    - Postoje dve tehnike:
      - Prekid izvršenja **svih** zaglavljenih procesa.
      - Prekid izvršenja procesa **do razbijanja kružnog toka**.
  - **Nasilno oduzimanje resursa** od procesa koji su u zastoju.
    - Tehnika koja ne prekida izvršenje procesa.
    - Određenom procesu (ili procesima) u zastoju se oduzimaju resursi.
    - Odabir žrtve sa najmanje gubitaka!

Literatura

1. B. Đorđević, D. Pleskonjić, N. Maček (2005): Operativni sistemi: teorija, praksa i rešeni zadaci. Mikro knjiga, Beograd.
2. R. Popović, I. Branović, M. Šarac (2011): Operativni sistemi. Univerzitet Singidunum, Beograd.

Hvala na pažnji

Pitanja su dobrodošla.