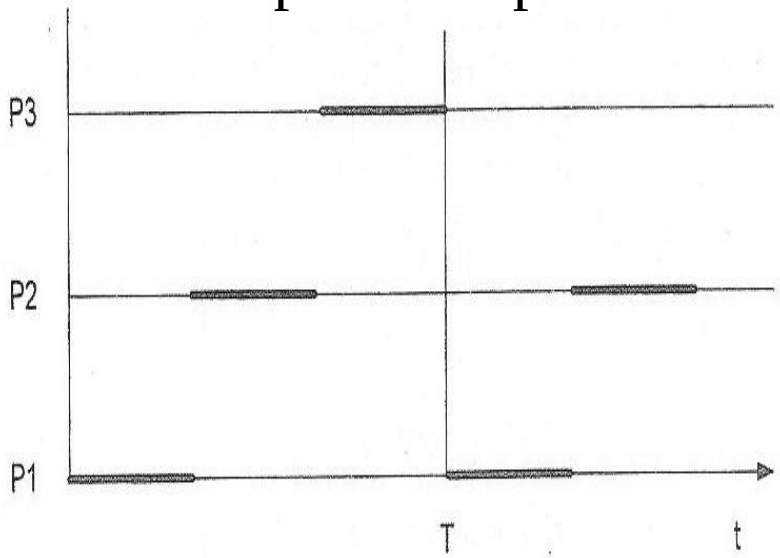
S A D R Ž A J

4.1-Problem sinhronizacije

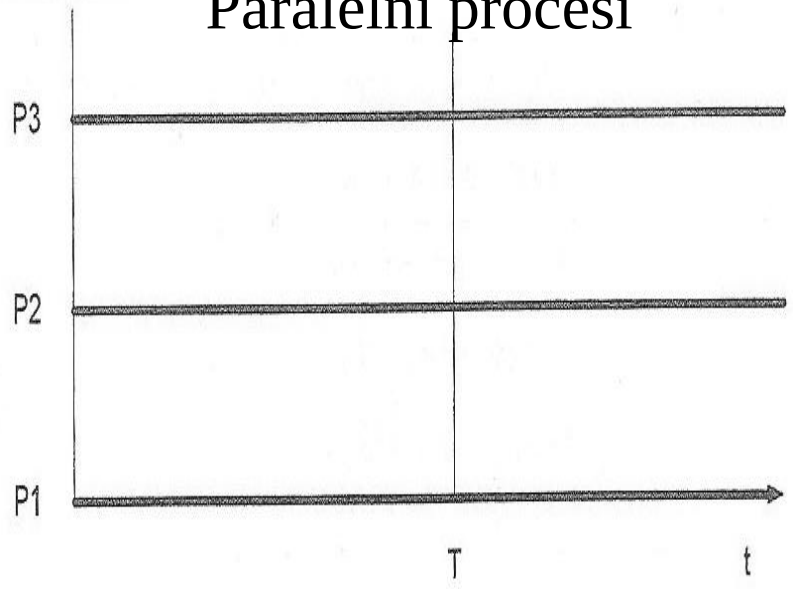
- Procesi mogu biti međusobno nezavisni ili zavisni.
- Ako se procesi izvršavaju uporedo i nezavisno onda nema iterakcije niti razmene informacija između njih
- Sa druge strane nalaze se kooperativni procesi koji razmenjuju informacije pa je potrebno da se međusobno sinhronizuju
- U multiprocesnom OS izolovanih procesa praktično nema – svi oni dele zajedničke resurse sistema (memoriju, fajlove, I/O uređaje)
- Kod nezavisnih procesa rezultat izvršavanja procesa ne zavisi od redosleda izvršavanja i preplitanja sa drugim nezavisnim procesima
- Kod procesa koji dele podatke rezultat izvršavanja zavisi od redosleda izvršavanja i preplitanja - dakle zavisi od raspoređivanja.
- Problem koji prati ovakav način rada je nedeterminisan rezultat, pa je uočavanje i ispravljanje grešaka veoma teško jer su one često nepredvidljive i ne ponavljaju se

4.1-Zavisni-konkurentni procesi

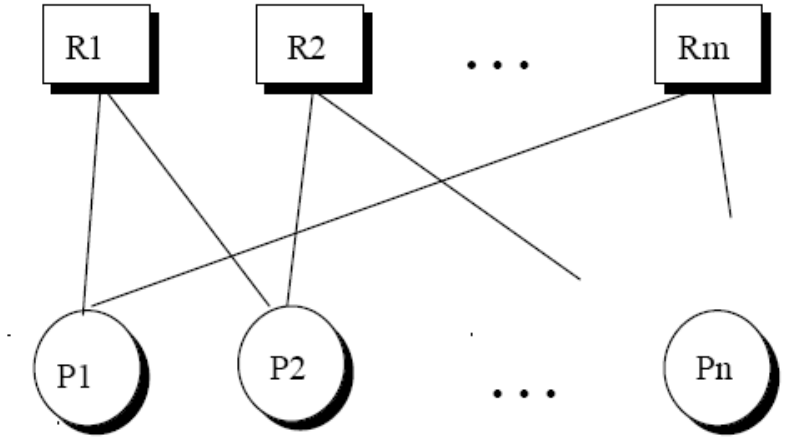
Kvazi paralelni procesi



Paralelni procesi



Shared Resources



Concurrent Processes

4.1-Zavisni-konkurentni procesi

Osnovno pravilo svih zavisnih-konkurentnih procesa glasi:

"***Rezultat (logička ispravnost) programa ne sme da zavisi od redosleda izvršavanja i preplitanja, tj. od načina raspoređivanja procesa***".

- Tri su osnovna problema koje treba rešiti kod razmatranja zavisnih - konkuretnih procesa:

1. **sinhronizacija procesa** (*synchronization*)
2. **međusobno isključenje procesa** (*mutual exclusion*)
3. **uzajamno blokiranje - zastoj** (*deadlock*)

4.1-Problem sinhronizacije

Primer: *Imamo dva procesa, P0 i P1, koji žele da privremeno sačuvaju neku vrednost na istoj memorijskoj lokaciji.*

Procedura odvijanja događaja je sledeća:

1. **P0** proverava da li je memorijska lokacija A slobodna. Slobodna je. **P0** je obavešten da je lokacija A slobodna;
2. **P1** proverava da li je memorijska lokacija A slobodna. Slobodna je. **P1** je obavešten da je lokacija A slobodna;
3. Proces **P0** upisuje podatak na lokaciju A;
4. Proces **P1** upisuje svoj podatak na lokaciju A;
5. Proces **P0** čita podatak sa lokacije A – POGREŠAN.

***OS mora da obezbedi mehanizme konkurentnog (paralelnog) izvršavanja procesa*.**

4.1-Berštajnova pravila

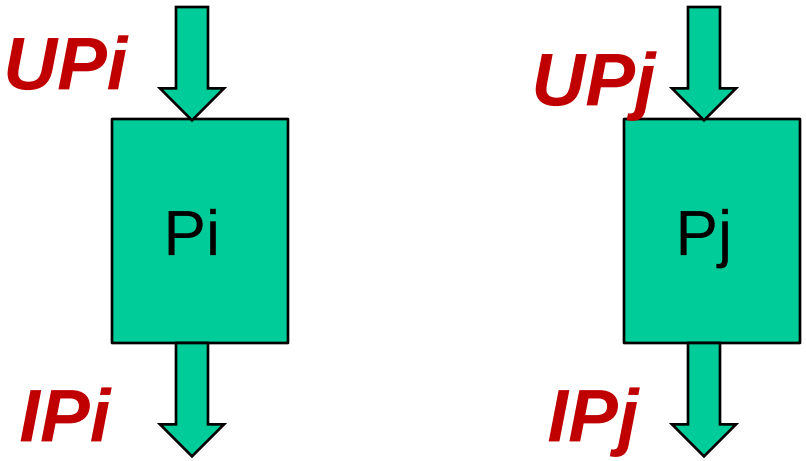
R(*Si*) = *{a1, a2, …, am}* - skup čitanja za Si je skup svih promenljivih čije se vrednosti koriste tokom izvršavanja iskaza Si.

W(*Si*) = *{b1, b2, …, bn}* - skup upisa za Si je skup svih promenljivih čije se vrednosti menjaju tokom izvršavanja iskaza Si.

- Tri uslova moraju da budu zadovoljena za dva uzastopna iskaza **S1** i **S2** da bi oni mogli da se izvršavaju konkurentno i da pri tom daju isti rezultat kao i kod sekvencijalnog izvršenja.
- Ovi uslovi su poznati kao Berštajnovi uslovi.

1. $R(S1) \cap W(S2) = \{\}$
2. $W(S1) \cap R(S2) = \{\}$
3. $W(S1) \cap W(S2) = \{\}$

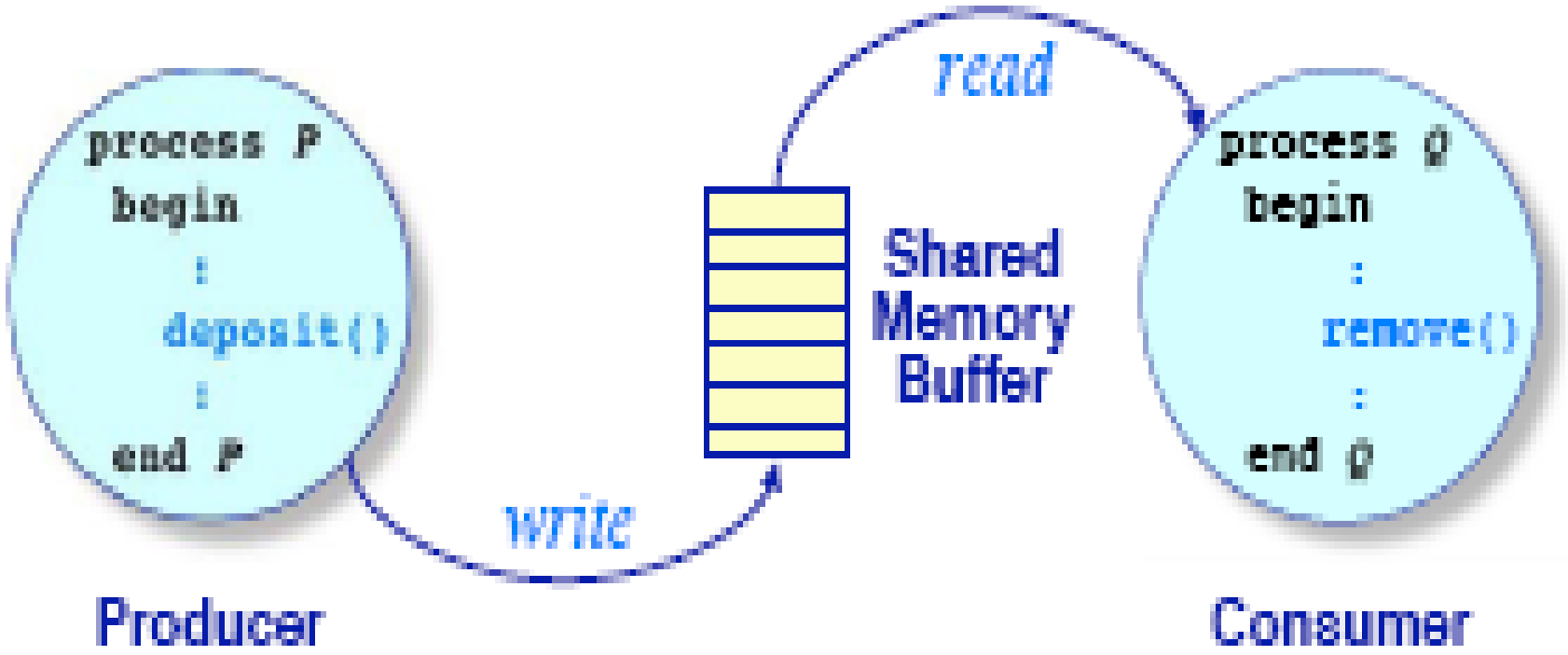
4.1-Berštajnova pravila



$$IPi \cap (Upj \cup IPj) = \{\}$$

$$IPj \cap (Upi \cup IPi) = \{\}$$

4.1-Problem proizvođač-potrošač




```
item buffer [N]      // definicija veličine bafera
int in = 0;          // prvo slobodno mesto
int out = 0;         // prvo zauzeto mesto
int brojač =0;       // broj zauzetih elementata
```

4.1-Problem proizvođač-potrošač

Proizvođač

```
item sledeći_proizveden;
while(1) {
  /*proizvođač proizvodi*/
  while (brojač == N)
  /*bafer je pun, proizvođač čeka
    da potrošač nešto uzme */
  buffer[in]=sledeći_proizveden;
  in = (in+1) % N;
  brojač++;
}
```

Potrošač

```
item sledeći_potrošen;
while(1) {
  while (brojač == 0)
  /*bafer je prazan, potrošač čeka
    da proizvođač nešto proizvede*/
  sledeći_potrošen= buffer[out];
  out = (out+1) % N;
  brojač--;
/*potrošač uzima proizvod*/
}
```

4.1-Problem proizvođač-potrošač

operacija brojač ++:
registar1 = brojač
registar1 = registar1 + 1
brojač = registar1

operacija brojač--:
registar2 = brojač
registar2 = registar2 - 1
brojač = registar2

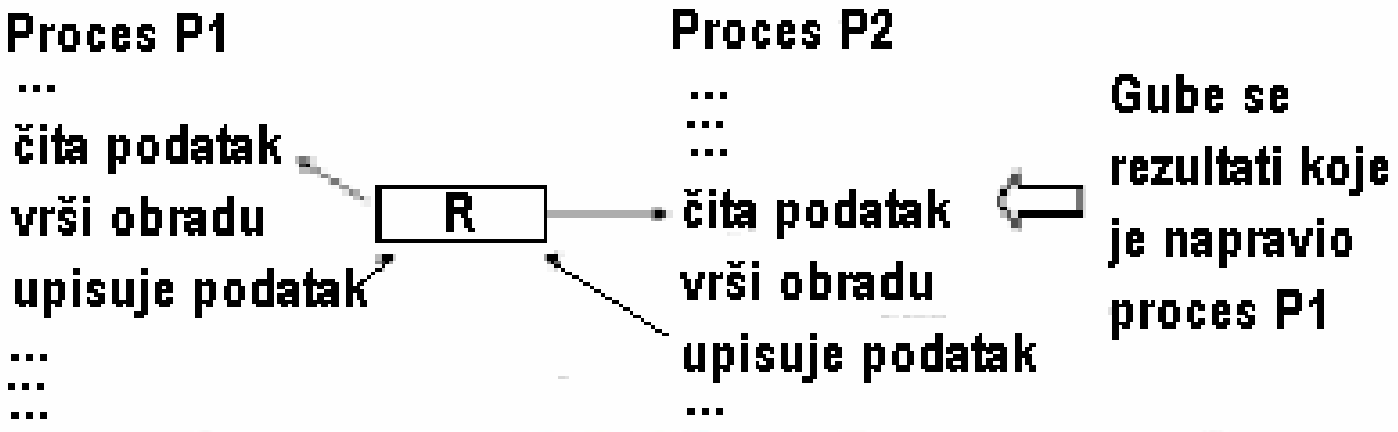
	izvršava	**naredba**	**vrednosti**
1.	proizvođač	registar1 = *brojač*;	registar1 = 5
2.	proizvođač	registar1 = registar1 + 1;	registar1 = 6
3.	potrošač	registar2 = *brojač*;	registar2 = 5
4.	potrošač	registar2 = registar2 - 1;	registar2 = 4
5.	proizvođač	*brojač* = registar1;	***brojač* = 6**
6.	potrošač	*brojač* = registar2 ;	***brojač* = 4**

4.2-Kritična sekcija

- Kada više procesa pristupa istim podacima i modifikuje ih konkurentno (paralelno), krajnja vrednost zajedničkih podataka zavisi od sekvence (redosleda) instrukcija procesa koje tim podacima pristupaju
- Taj pojam poznat je pod terminom - **stanje trke**.
- Mora se obezbediti mehanizam sinhronizacije tog dela procesa gde se pristupa zajedničkom resursu i koji se naziva **kritična sekcija** ili **oblast**
- On treba da definiše pristup zajedničkom resursu isključivo jednom od konkurentnih procesa u jednom trenutku
- Kritična oblast ili kritična sekcija predstavlja deo programskog koda (redosled naredbi) procesa (program ili deo programa u stanju izvršavanja) gde se pristupa zajedničkim podacima (memorijske promenljive, tabele, datoteke) ili se vrši njihovo modifikovanje.
- **Neprekidiv niz operacija** (kao jedna primitivna operacija)
- **Proces ne sme biti prekinut dok se nalazi u svojoj kritičnoj oblasti.**

Osnovni zadatak OS je da dozvoljava samo jednom procesu da bude u svojoj kritičnoj sekciji

4.2-Kritična sekcija



4.2-Kritična sekcija - realizacija

1. softverski (programer),
2. hardverski,
3. tehnika semafora,
4. operativni sistem
5. višim programskim strukturama za sinhronizaciju

4.2 Softverska realizacija

1. **Međusobno isključivanje** (*Mutal Exclusion*): Ako se jedan proces nađe u kritičnoj sekciji tada ni jedan drugi proces ne sme biti u svojoj kritičnoj sekciji.

2. **Napredovanje** (*Progress*): Ako proces nije u svojoj kritičnoj sekciji, a postoje procesi koji čekaju da uđu u svoju kritičnu sekciju, tada samo ti procesi mogu učestvovati u donošenju odluke koji proces će dobiti pravo da prvi uđe u kritičnu sekciju, tj. procesi koji nisu u kritičnoj sekciji ne smeju sprečiti druge procese da uđu u svoje kritične sekcije.

3. **Ograničeno čekanje** (*Bounded Waiting*):
   - Mora postojati vremensko ograničenje unutar koga se mora omogućiti procesu koji čeka da uđe u svoju kritičnu sekciju da to učini – ne sme neograničeno dugo da čeka.
   - Proces ne sme neograničeno dugo ostati u svojoj kritičnoj sekciji

4.2-Algoritam striktne alternacije

Proces P0

```
do {
   while (turn != 0) ;
   /*kritična sekcija*/
   turn=1;
   /* ostatak programa */
} while (1);
```

Proces P1

```
do {
   while (turn != 1) ;
   /*kritična sekcija*/
   turn=0;
   /* ostatak programa */
} while (1);
```

- Procesi se izvršavaju strogo u poretku **P0, P1, P0, P1** ....
- Nedostatak jer se koristi tehnika prozivanja (polling)
- Stalno se troši procesorsko vreme
- Javlja se problem ako se desi da su vremena trajanja kod dva procesa neujednačena: brzi i spori procesi

4.2-Bez stroge alternacije

- Uvođenje binearne promenljive (***flag***) koja označava spremnost nekog procesa da želi da uđe u svoji kritičnu sekciju

Proces Pi

```
int flag[2]
flag[0]= flag[1]=0;
do {
   flag[ i ]=1;
   while (flag[ j ]);
   /*kritična sekcija*/
   flag[ i ] =0;
   /* ostatak programa */
} while (1);
```

Problem:

P0 postavlja flag[0]=1, a posle toga se kontrola dodeljuje procesu P1 koji to isto uradi, tj. P1 postavlja flag[1]=1

Rezultat:

Oba procesa ulaze u petlju jer ni jedan ne može da uđe u svoju kritičnu sekciju jer su obe promenljive postavljene na vrednost 1

4.2-Dekker-Petersonov algoritam

- Kombinacija prethodna dva rešenja (*turn* i *flag* promenljive)
- Proces **Pj** želi da uđe u svoju kritičnu sekciju **flag[j] =1**
- Proces **Pj** može da uđe u svoju kritičnu sekciju samo ako je **turn=j**

Proces Pi

```
do {
   flag[i]= true;
   turn = j;
   while(flag[ j ] && turn=j);
   /*kritična sekcija*/
   flag[i] =0;
   /* ostatak programa */
} while (1);
```

Svi uslovi su zadovoljeni:

- ✓ Poštuje se međusobno isključivanje
- ✓ Nema striktne alternacije
- ✓ Nema zaglavljivanja u beskonačnoj petlji
- ✓ Pekarski algoritam (*bakery algorithm*) rešava ovaj problem za n procesa

4.2 Softverska realizacija

- Teškoće koje nastaju u softverskoj realizaciji algoritama za upravljanje kritičnim sekcijama ogledaju se u:

1. stalnom testiranju promenljivih ili stanja čekanja, što dosta utiće na potrošnju procesorskog vremena,
2. svi detalji implementacije direktno zavise od programera i mogućnost greške je uvek prisutna,
3. ne postoji način da se nametne protokol koji zavisi od kooperacije, programer može da izostavi neki deo,
4. ovi protokoli su suviše komplikovani.

4.2-Hardverska realiz.kritične sekcije

- Hardverska implementacija – procesorska instrukcija koja je nedeljiva (atomičan način izvršavanja) - ne može se prekinuti tokom izvršavanja
- Naredba **TestAndSet (lock)**: proces može da pročita vrednost promenljive lock (0 – dozvoljen, 1 – nije dozvoljen ulazak u kritičnu sekciju) i postavlja vrednost promenljive lock na 1
- Promenljivoj lock se inicijalno dedeljuje vrednost 0. PRVI proces (npr. Pi) koji želi da uđe u kritičnu sekciju naredbom TestAndSet (lock) proverava vrednost promenljive lock i ako je 0, postavlja vrednost lock = 1.
- Drugi procesi ne mogu ući u kritičnu sekciju (lock = 1) sve dok proces Pi ne izađe iz kritične sekcije i postavi vrednost promenljivoj lock = 0.

```
Int lock=0;
do {
   while
(TestAndSet(lock);
   /*kritična sekcija*/
   lock =0;
   /* ostatak programa */
} while (1);
```

4.2-Tehnika semafora

- Jednostavan koncept koji je moguće primeniti na kompleksnije slučajeve tj. za veći broj procesa
- Poseduje mehanizam za programiranje međusobnog isključenja i sinhronizaciju aktivnosti procesa.
- Semafor je celobrojna nenegativna promenljiva koja štiti neki resurs.

  Vrednost semafora S=0 => resurs zauzet

  Vrednost semafora S>0 => resurs slobodan
- Svaki semafor ima svoju početnu vrednost i mogu se izvršiti samo dve nedeljive operacije (primitive): signal (s) i wait (s)
- Operacije signal i wait se ne mogu podeliti u više ciklusa
- Dva ili više procesa ne mogu istovremeno izvršavati ove operacije nad istim semaforom
- Funkcije operacija signal (s) i wait (s) :
  - **wait(S)**: vrednost semafora S>0 (resurs slobodan) i vrednost S se umanjuje za 1; vrednost semafora S=0 (resurs zauzet) proces mora da čeka sve dok S ne postane veće od nule i tada se vrednost semafora S umanjuje za jedan
  - **signal(S)**: vrednost semafora se uvećava za jedan

4.2-Tehnika semafora

Primer - binarni semafor mutex čija je inicijalna vrednost 1

```
semaphore mutex;    /* inicijalno mutex=1 */


do {
wait(mutex) ;
      /* kritična sekcija */
signal(mutex) ;
      /* ostatak koda */
       }  while (1) ;
```

4.2-Tehnika semafora

Pogodnosti semafora:

- jednostavan i efikasan koncept
- generalan koncept niskog nivoa – pomoću njega se mogu implementirati mnogi drugi, apstraktniji koncepti za sinhronizaciju

Loše strane semafora:

- suviše jednostavan koncept niskog nivoa – nije logički povezan sa konceptima bližim domenu problema (resurs, kritična sekcija, ...)
- kod složenijih programa lako postaje glomazan, nepregledan, težak za razumevanje, proveru i održavanje jer su operacije nad semaforima rasute
- podložan je greškama – mora se paziti na uparenost i redosled operacija *wait* i *signal*

4.2-Pomoću OS

- Nedostaci softverske i hardverske realizacije zaštite kritične sekcije semaforima su:
  - ignorisanje prioriteta procesa – proces najvišeg prioriteta ulazi u kritičnu sekciju posle mnoštva manje prioritetnih procesa.
  - proces može biti zauzet čekanjem, proverava vrednost neke promenljive kako bi saznao da li može da uđe u svoju kritičnu sekciju – ne radi ništa korisno a troši procesorske vreme.

Rešenje:

- za svaki semafor (resurs) uvesti red čekanja – semaforski red.
- svi procesi koji izvršavaju operaciju **wait** nad semaforom i ako je vrednost semafora S<=0, pomoću sistemskog poziva **sleep** se blokiraju i prevode u semaforski red (red za resurs).
- procesor se oslobađa i predaje nekom drugom procesu da radi nešto
- proces nastavlja svoje izvršenje nakon sistemskog poziva **wakeup** koji ukida blokadu procesa.
- Blokada se ukida ako je proces prvi u semaforskom redu ili ima najviši prioritet u semaforskom redu.

4.2-Višeprogramske strukture

- **Kritični region** zahteva promenljivu koja je zajednička za veći broj procesa i njoj se može pristupiti samo unutar kritičnog regiona.
- Kritični region štite od grubih, nehotičnih programerskih grešaka u sihronizaciji procesa
- **Monitori** omogućavaju programeru da svaki resurs posmatra kao jednu vrstu objekta.

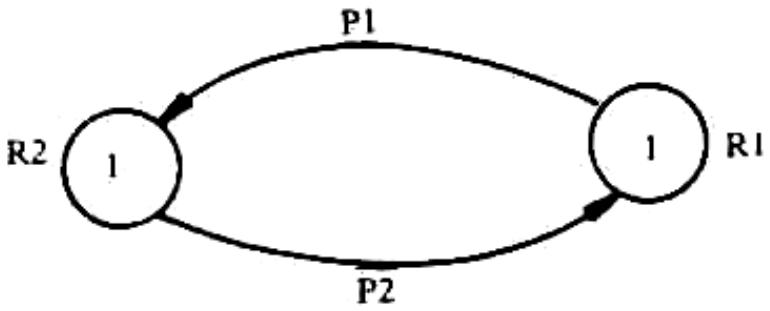
Monitor se sastoji od:

- promenljivih koje opisuju resurs (njihove vrednosti definišu resurs)
- skupa procedura i funkcija kojima se pristupa objektu (promenljivima monitora)
- dela programskog koda koji inicijalizuje objekte i koji se izvršava samo jednom, prilikom stvaranja objekata

4.3-Sistemski model i osobine zastoja

- U višeprocesnom okruženju, više procesa sa takmiči za konačan broj resursa.
- Kad proces zahteva resurs a resurs nije raspoloživ, proces ulazi u stanje čekanja (WAIT) i blokira se.
- Blokirani proces može ostati zauvek u tom stanju ukoliko potreban resurs ostane neraspoloživ, ti.:

1. Resurs je prethodno dodeljen drugom procesu na korišćenje.
2. Drugi proces je prešao u stanje čekanja jer mu je potreban još jedan resurs koji mu nije raspoloživ.



Ove situacije se nazivaju ZASTOJ !

4.3-Sistemski model - primer

Primer: Sistem raspolaže sa 12 instanci jednog resursa. Trenutno su aktivna tri procesa sa stanjem prikazanim u tabeli.

Procesi	Max.zahteva	dodeljeno
P1	10	5
P2	4	2
P3	9	2

Redosled dodeljivanja < P2, P1, P3 > garantuje završetak sva tri procesa jer: proces P2: dobija 2 instance (jedna ostaje slobodna) i završava sa radom i oslobađa 4 istance (ostaje pet slobodnih); proces P1: dobija svih pet istanci (nema više slobodnih) i završava se radom i oslobadja 10 istanci; proces P3 dobija sedam istanci (tri ostaju slobodne) i završava. Pogrešnom dodelom resursa sistem može da dođe u nebezbedno stanje.

Procesi	Max.zahteva	Dodeljeno
P1	10	5
P2	4	2
P3	9	3

Sada jedino proces P2 može da nastavi, ali i kada završi oslobadja 4 instance. Kako proces P1 traži još pet, a proces P3 još šest instanci i oba (beznadežno) čekaju. Sistem je prema tome u nebezbednom stanju i može doći do zastoja

4.3-Uslovi za nastanak zastoja

1. Međusobno isključenje – samo jedan proces može koristiit resurs ili njegovu istancu
2. Nema pretpražnjenja (prekidanja) – resurs se ne može nasilno oduzeti i predati drugom procesu
3. Uslov zadržavanja resursa i čekanja na drugi (*hold and wait*)
4. Kružno čekanje – mora postojati skup procesa koji su kružno povezani (P0, P1, P2, ..., P0)

Zastoj može da nastupi samo ako su sva 4 uslova istovremeno ispunjena

4.3-Sistemski model i osobine zastoja

Zastoj se rešava na tri načina:

1. prevencijom ili izbegavanjem zastoja (koriste se metode da sistem nikada NE uđe u stanje zastoja)
2. detekcijom i oporavkom (koriste se metode koje dozvoljavaju sistemu da uđe u stanje zastoja, to stanje se detektuje i oporavlja sistem)
3. ignorisanje problema zastoja (pretvaraju se da problem zastoja ne postoji)

4.4-Metode uprav.zastojem-prevencija

- Međusobno isključenje – izbegavanje ovog uslova za sve deljive resurse
- Nema pretpražnjenja (prekidanja) - samo proces koji koristi resurs može taj resurs osloboditi, tj. resurs se ne može nasilno oduzeti od strane nekog procesa koji ne koristi taj resurs
- Uslov zadržavanja resursa i čekanja na drugi (*hold and wait*) - svaki proces mora da traži tj. alocira sve potrebne resurse pre početka izvršavanja ili da oslobodi sve resurse pre traženja nekog novog
- Kružno čekanje - svakom resursu se dodeli neki broj N iz skupa prirodnih brojeva a procesi mogu zahtevati resurse po strogo rastućem redosledu, tj. ako proces koristi resurs **Rj** on nakon toga može da zahteva samo resurs **Rx** gde važi pravilo da je x uvek mora da bude veće od j (**x>j**)

4.6 Detekcija i oporavak od zastoja

Dve najkorišćene metode za oporavak su:

1. Prekid izvršavanja procesa u zastoju
   - ✓ Prekid izvršenja svih zaglavljenih procesa
   - ✓ Prekid izvršenja procesa do razbijanja kružnog toka
2. Nasilno oduzimanje resursa od procesa u zastoju

U oba slučaja treba voditi računa o:

- ❑ Prioritetu procesa
- ❑ Koliko dugo radi proces i koliko mu je vremena ostalo do kraja
- ❑ Koje resurse koristi
- ❑ Vrsti procesa: interaktivni ili grupni

Hvala na pažnji !!!

Pitanja

? ? ?