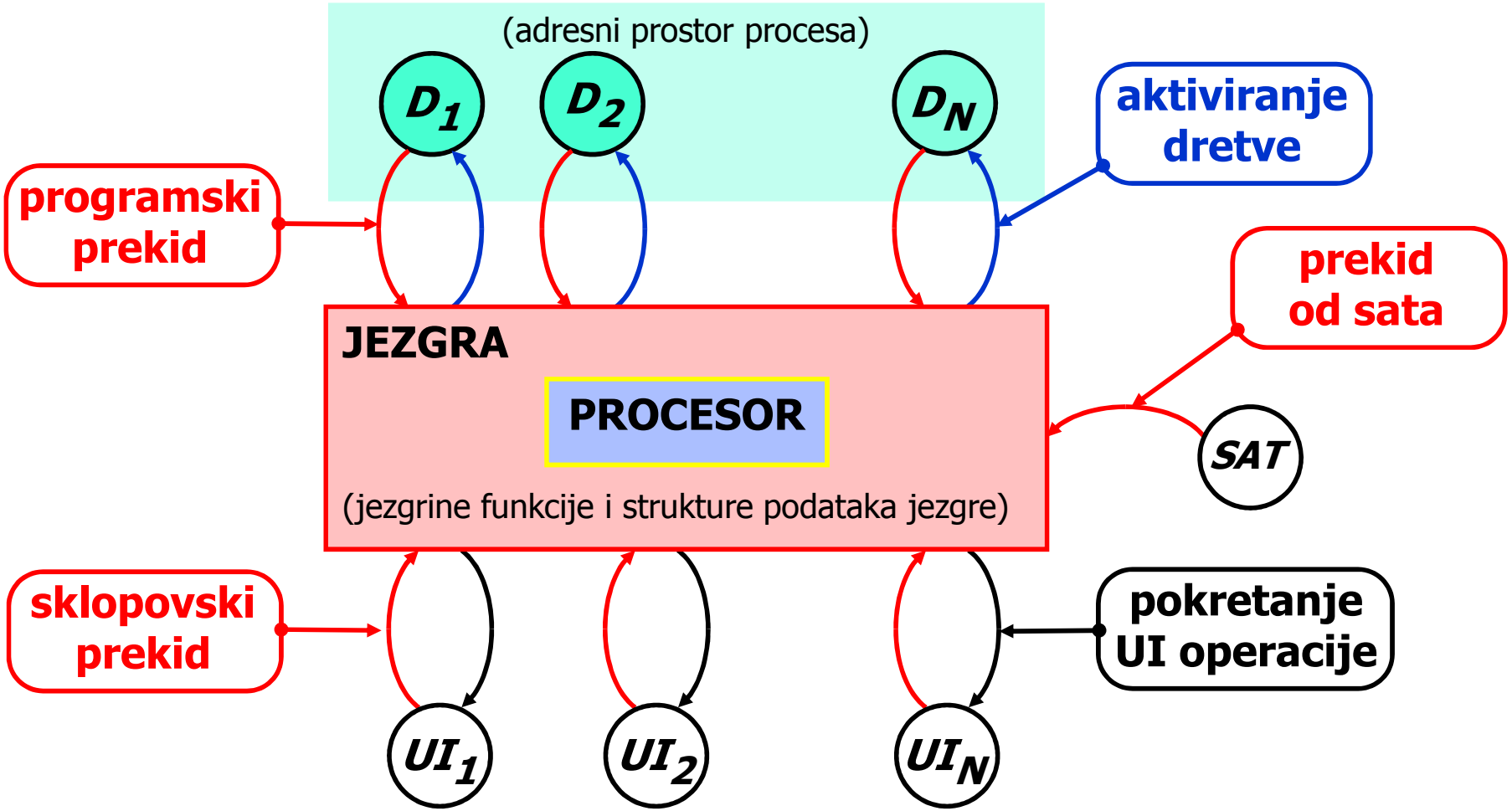
Osnovni koncepti operacijskih sustava

Jezgra operacijskog sustava

Jezgra OS-a

- poziva se prekidima
- sastoji se od jezgrinih funkcija i struktura podataka
- jezgrini podaci i kod su od korisničkih dretvi zaštićeni mehanizmima zaštite spremnika te načinima rada procesora
- osnovne zadaće jezgre:
  - upravljanje dretvama (raspoređivanje, sinkronizacija i komunikacija)
  - upravljanje sredstvima sustava (spremnik, UI, procesorsko vrijeme, …)



Poziv jezgrine funkcije — ulazak u jezgru — zbiva se kada se dogodi prekid.
Izlazak iz jezgre svodi se na pokretanje jedne od dretvi, pri čemu procesor mora biti vraćen u korisnički način rada.

Podatkovne strukture jezgre

- za upravljanje dretvama:
  - opisnici dretvi:
    - identifikacijski broj dretve
    - prioritet, način raspoređivanja, …
    - opis spremničkih lokacija (stog, privatni podaci dretve, …)
    - mjesto za spremanje konteksta
    - …
  - stanja dretve – liste dretvi:
    - aktivna dretva – trenutno se izvodi
    - pripravne dretve
    - blokirane dretve – odgođene, blokirane na semaforu, UI, …
    - pasivne dretve – dretve koje su završile s radom

Podatkovne strukture jezgre

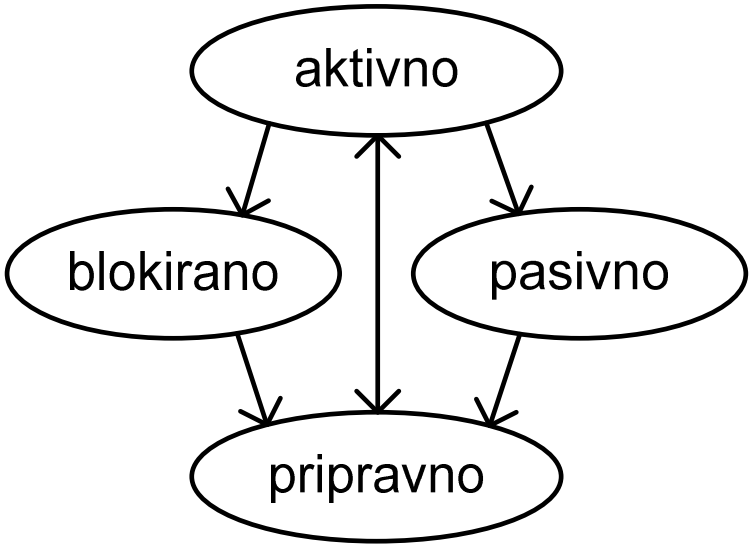
- za upravljanje procesima:
  - opisnici procesa:
    - spremničke lokacije - kod, podaci, stogovi, opisnici korištenog adresnog prostora
    - opisnici korištenih sredstava
      - UI naprave
      - sinkronizacijski i komunikacijski mehanizmi
      - datoteke
      - …
    - podaci o korisniku, procesu roditelju, …
    - prioritet, parametri raspoređivanja
    - lista dretvi
    - …
- ostala sredstva sustava – spremnik, UI, datotečni sustav, mrežni podsustav, …
  - spremničke lokacije, međuspremnici, liste blokiranih dretvi

Jezgrine funkcije

- pozivaju se mehanizmom prekida
- dok se obrađuju zabranjeno je prekidanje
- tipični scenarij obrade:
  - prekidni signal (ili instrukcija)
  - prihvat prekida – automatsko ponašanje procesora:
    - zabrana daljnjeg prekidanja
    - promjena načina rada procesora – prekidni način rada
    - pohranjuje se minimalni kontekst na stog (programsko br.)
    - skok u obradu prekida (adresa se postavlja u prog. br.)
  - *u proceduri za obradu prekida*:
    - sprema se puni kontekst prekinute dretve
    - utvrđuje se uzrok prekida i poziva odgovarajuća ***jezgrina funkcija***
    - po njenom završetku obnavlja se kontekst prekinute dretve (ili druge koja je aktivirana u obradi) te se nastavlja s njenim izvođenjem

Stanje dretvi u sustavu

- aktivno
- pripravno
- pasivno
- blokirano:
  - sinkronizacija
    - npr. semafor
  - UI naprava



Primjer jezgrine funkcije – binarni semafor

- Jednostavni sinkronizacijski mehanizam
- Potrebna struktura podataka za svaki semafor **`Sem[id]:`**
  - **`vrijednost`** – trenutna vrijednost: 0 ili 1
  - **`red`** – red za blokirane dretve

```
j-funkcija CekajBSem(id)
{
   ako (Sem[id].vrijednost == 1) {
        Sem[id].vrijednost = 0;
   }
   inace {
        Stavi_u_red(Aktivna_D, Sem[id].red);
        Aktivna_D = UzmiPrvu(Pripravne_D);
   }
}
```

Primjer jezgrine funkcije – binarni semafor

```
j-funkcija PostaviBSem(id)
{
   ako (Sem[id].red neprazan) {
       Stavi_u_red(Aktivna_D, Pripravne_D);
       prva = UzmiPrvu(Sem[id].red);
       Stavi_u_red(prva, Pripravne_D);
       Aktivna_D = UzmiPrvu(Pripravne_D);
   }
   inace {
       Sem[id].vrijednost = 1;
   }
}
```

- Prikazani mehanizam sadržava samo osnovnu funkcionalnost

Jezgrine funkcije

- Većina jezgrinih funkcija koristi iste principe kao u prikazanom primjeru:
  - sinkronizacijske funkcije
  - upravljanje vremenom
  - UI
  - …

Podrška za višeprocesorske sustave

- struktura podataka jezgre mora se nalaziti u dijeljenom spremniku (zajedničkom za sve procesore)
- ostvarenje koncepta kritičnog odsječka za jezgrine funkcije mora se ojačati, jer korištenje prekida nije dovoljno
- "Ispitaj i postavi" (*Test and Set* – TAS) i slične instrukcije koriste se pri zaključavanju jezgre (*spinlock*)

```
zakljucaj_j: TAS ograda_jezgre, reg;
             ako reg == 1 skoci_na zakljucaj_j;
```

- instrukcija TAS koristi dva uzastopna sabirnička ciklusa:
  - u prvom pročita sadržaj zadane lokacije u registar
  - u drugom na tu lokaciju spremi jedinicu
- "radno čekanje" pri ulasku u jezgru je neizbježno u višeprocesorskim sustavima

Primjer proširenja za višeprocesorske sustave

```
j-funkcija PostaviBSem(id)
{
   zakljucaj_j: TAS ograda_jezgre, reg;
                ako reg == 1 skoci_na zakljucaj_j;

   ako (Sem[id].red neprazan) {
       Stavi_u_red(Aktivna_D[P], Pripravne_D);
       prva = UzmiPrvu(Sem[id].red);
       Stavi_u_red(prva, Pripravne_D);
       Aktivna_D[P] = UzmiPrvu(Pripravne_D);
   }
   inace {
       Sem[id].vrijednost = 1;
   }

   ograda_jezgre = 0;
}
```

Uobičajeni postupci u izgradnji jezgre

- pripravne dretve složene su u višerazinske redove (liste), za svaku razinu prioriteta po jedan red (lista)

- u današnjim višeprocesorskim sustavima redovi pripravnih dretvi su podijeljeni po procesorima
  - svaki procesor ima svoj red pripravnih dretvi

  - razlog je iskorištenje priručnog spremnika uz procesor, koji je višestruko brži od glavnog spremnika
    - dretve koje se izvode na istom procesoru i nakon zamjene drugim, pri povratku na procesor mogu u priručnom spremniku zateći svoje podatke (“hot-cache”)

  - ipak, povremeno treba uskladiti te redove da sve dretve dobiju približno isto procesorsko vrijeme

Troškovi jezgre (overhead)

- Jezgrine su funkcije građene da traju vrlo kratko
  - tj. trebale bi biti takve

- Ipak, ako se *vrlo često* pozivaju
  - tada implicitni troškovi poziva:
    - spremanje i obnova *konteksta* dretve
    - promjena načina rada procesora (nije uvijek zanemarivo!)
  - mogu činiti i značajan dio procesorskog vremena

Osnovni koncepti operacijskih sustava

Upravljanje dretvama:
sinkronizacija

Potreba za sinkronizacijom

- Puno zadataka – malo sredstava
  - samo ograničeni broj zadataka može istovremeno koristiti dostupna sredstva (UI, objekte, …)
  - najčešće “ograničeni broj” je jednak jedan, tj. samo jedan zadatak može koristiti sredstvo, dok svi ostali moraju čekati (biti blokirani)
- Jedan zadatak koji koristi više dretvi
  - dretve koriste zajedničke objekte – korištenje zajedničkog objekta može biti kritična operaciju koju treba zaštititi od istovremenog korištenja
  - dretve koje surađuju u izvođenju zadanog posla mogu trebati neki mehanizam sinkronizacije (npr. pri dodjeli elemenata za obradu i sl.)
  - dretve mogu koristiti mehanizam cjevovoda za ostvarenje komunikacije (rezultati prethodne dretve prosljeđuju se idućoj)

Uobičajeni mehanizmi dostupni kroz OS

- Najefikasnija sinkronizacija postiže se korištenjem OS-a
  - ostali principi uključuju radno čekanje (koje je vrlo vrlo rijetko bolje rješenje)

- Ostvarenje kritičnog odsječka (međusobno isključivanje)
  - Korištenje instrukcija za zabranu i dozvolu prekida
  - Binarni semafori (mogu i ostali uz pažljivo korištenje)
  - Varijable međusobnog isključivanja (*mutex*)

- Brojački semafor – opći semafor
  - kada je broj sredstava ≥ 1

- Složenije sinkronizacije
  - korištenjem više semafora
  - korištenjem monitora
  - korištenje dodatnih varijabli (uz semafore ili monitor)

Zabrana/dozvola prekida

- Funkcionalna samo na jednoprocesorskim sustavima
- Operacija zabrane i dozvole prekidanje je privilegirana operacija (instrukcija)
  - program mora imati ovlasti za njeno izvođenje
- Mora se koristiti vrlo oprezno!
  - **ako dretva stane u kritičnom odsječku koji je zaštitila zabranom prekida događa se potpuno blokiranje sustava**
- Mehanizam ulaska i izlaska je vrlo jednostavan i efikasan kada se odgovarajuće koristi
  - samo za vrlo kratke kritične odsječke!

- Uglavnom se koristi samo u:
  - **jezgrinim funkcijama**
  - **ugrađenim i RT sustavima**

Primjer korištenja zabrane/dozvole prekida

```
.
.  (nekriticni odsjecak)
.
zabrani_prekidanje();

KRITICNI_ODSJECAK;          (samo jedna dretva može biti ovdje)

dozvoli_prekidanje();
.
.  (nekriticni odsjecak)
.
```

Binarni semafor

- **`CekajBSem(s_id)`**
  - *sinonimi*: ispitaj, zaključaj, dohvati
  - *operacija*: zaključaj objekt semafora
    - zaključaj samo ovaj objekt! nije globalno zaključavanje (kao mehanizam zabrane prekida)
    - *programerski pristup*: zaključavanje semafora osigurava pristup sredstvu koje semafor štiti (semafor ⇔ sredstvo)
    - ako je semafor već zaključan (od neke druge dretve):
      - dretva se blokira i miče u red zadanog semafora
- **`PostaviBSem(s_id)`**
  - *sinonimi*: otključaj, oslobodi, pošalji
  - *operacija*: otpusti objekt semafora
    - ako red semafora nije prazan (dretve čekaju u njegovu redu), tada prvu dretvu iz tog reda propusti u red pripravnih dretvi – toj dretvi *dodijeli* semafor
    - inače (prazan red), samo označi da je semafor prolazan

Primjer s binarnim semaforom za K.O.

```
.
.  (nekriticni odsjecak)
.
CekajBSem(s1);

KRITICNI_ODSJECAK;          (samo jedna dretva može biti ovdje)

PostaviBSem(s1);
.
.  (nekriticni odsjecak)
.
```

Primjer s binarnim semaforom za alternaciju

- Osim za primarnu namjenu međusobnog isključivanja, binarni se semafor može koristiti i za naizmjenično obavljanje dvije ili više dretvi

```
Dretva I:

dok(1) {
   CekajBSem(s1);

   na_redu_I();

   PostaviBSem(s2);
}
```

```
Dretva J:

dok(1) {
   CekajBSem(s2);

   na_redu_J();

   PostaviBSem(s1);
}
```

- Početno je samo jedan semafor postavljen u 1 (npr. **s1**)

Opći semafor (brojački semafor)

- Semafor se koristi za brojanje dostupnih sredstava
  - npr. broj poruka u redu, elemenata liste, …

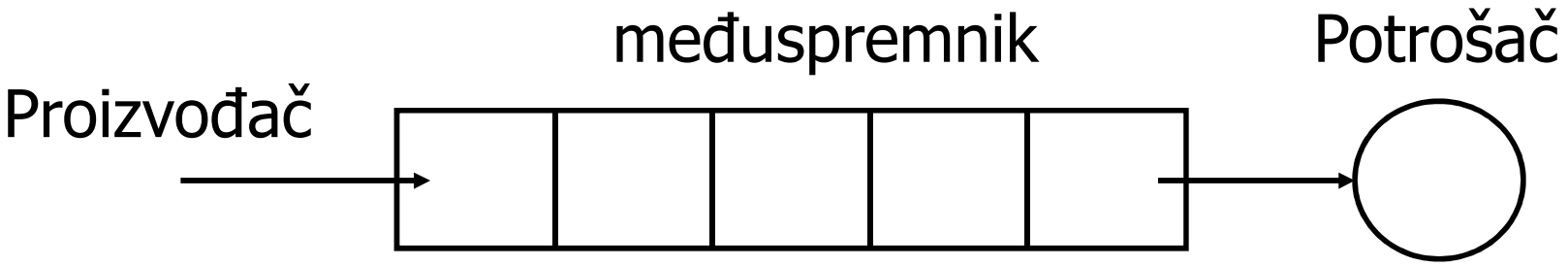
- *vrijednost* semafora:
  - ako je *vrijednost* = 0, semafor je neprolazan
    - dretve će se blokirati na takvom semaforu
  - ako je *vrijednost* > 0, semafor je prolazan
    - barem će jedna dretva proći takav semafor

- Npr. dretva potrošača koja uzima poruke iz međuspremnika, prije uzimanja ispituje semafor:

```
…
CekajSem(sm);          //blokiraj dretvu ako nema poruka
(uzmi iduću poruku iz međuspremnika)
…
```

Primjer: problem proizvođača i potrošača

- Problem proizvođača i potrošača prikazuje korištenje semafora koji komunicijraju preko ograničenog međuspremnika veličine N poruka
- Proizvođač “proizvodi” poruke i stavlja ih u međuspremnik
- Potrošač uzima poruke iz međuspremnika i “troši” ih
- Proizvođača treba blokirati ako je međuspremnik pun
- Potrošača treba blokirati ako je međuspremnik prazan



Primjer: problem proizvođača i potrošača

Proizvođač

```
dok(1) {
   P = proizvedi();

   CekajSem(prazan);

   StaviPoruku(P);

   PostaviSem(pun);
}
```

Potrošač

```
dok(1) {
   CekajSem(pun);

   R = UzmiPoruku();

   PostaviSem(prazan);

   potroši(R);
}
```

- Početne vrijednosti: `prazan=N; pun=0;`

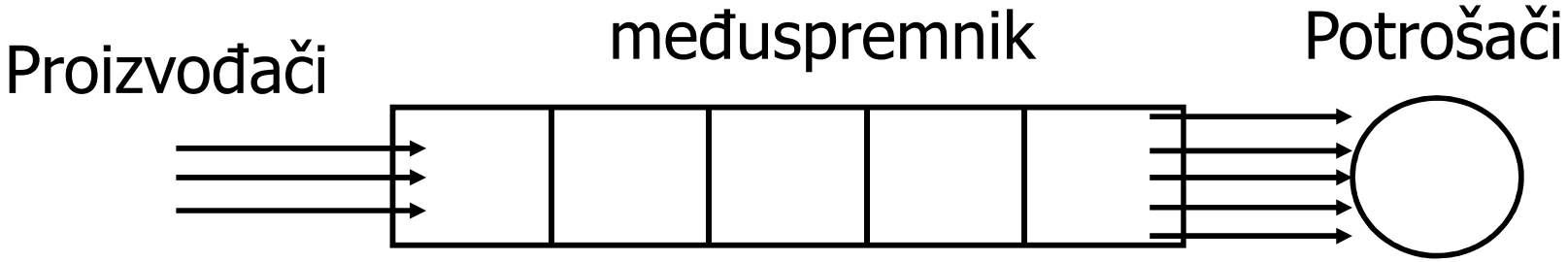
Problemi sa semaforima

- Semafori su najkorišteniji mehanizam za jednostavne slučajeve sinkronizacije dretvi
  - podržani od svih OS-a (čak i s više različitih sučelja)
  - jednostavni za korištenje

- Međutim, ako problem sinkronizacije nije jednostavan, potrebno je više od jednog semafora!
  - ako u izvođenju dretve, ona traži više od jednog sredstva istovremeno, ona mora zauzeti više semafora
  - generiranje i razumijevanje takvog koda je sve osim jednostavnog
  - više semafora – veća vjerojatnost greške i pojave **potpunog zastoja**

“Proširenje” problema proizvođača i potrošača

- Ako se prethodni primjer proširi s više proizvođača i više potrošača (ne samo jednog od svake vrste)
  - proizvođači ne smiju paralelno stavljati poruke u međuspremnik
    - obzirom da upravljanje međuspremnikom traži i dodatne varijable, tu operaciju treba smatrati kao kritični odsječak
    - dodatni (binarni) semafor je potreban
  - slični su problemi i s potrošačima
    - potreban je dodatni (binarni) semafor



Proizvođači/potrošači – potpuni zastoj

- Neka se za korištenje međuspremnika koristi dodatni binarni semafor (i proizvođači i potrošači koriste isti)

Proizvođač

```
dok(1) {
   P = proizvedi();
   CekajSem(ms);
   CekajSem(prazan);
   StaviPoruku(P);
   PostaviSem(pun);
   PostaviSem(ms);
}
```

Potrošač

```
dok(1) {
   CekajSem(pun);
   CekajSem(s_buffer);
   R = UzmiPoruku();
   PostaviSem(s_buffer);
   SemSignal(prazan);
   potrosi(R);
}
```

- Kada se međuspremnik popuni, idući proizvođač će se blokirati na **prazan**, ali će pritom i semafor **ms** biti zaključan – potrošač neće moći uzeti poruku:
  - pojavljuje se **potpuni zastoj**

Tipični scenarij nastanka potpunog zastoja

- Dvije ili više dretvi treba dva ili više sredstva

```
Dretva I:

   ...
   CekajSem(s1);
   ...
   CekajSem(s2);
   ...
   ...
   PostaviSem(s1);
   ...
   PostaviSem(s2);
   ...
```

```
Dretva J:

   ...
   ...
   CekajSem(s2);
   ...
   CekajSem(s1);
   ...
   PostaviSem(s1);
   ...
   PostaviSem s2);
   ...
```

POTPUNI ZASTOJ!

Sprječavanje nastanka potpunog zastoja

- U nekim sustavima postoji mogućnost obavljanja operacija nad skupom semafora kao atomarne operacije
  - ako se barem jedna operacija ne može napraviti, dretva se blokira i niti jedna se operacija ne izvodi

- primjer (UNIX*):
  - `semop (id, polje_operacija, broj_op);`

- koristiti druge sinkronizacijske mehanizme
  - npr. **monitore** (ili slične)

Monitori

- operacije nad zajedničkim podacima obavljati u kontroliranom okruženju – u “monitorskim funkcijama”

- monitorske su funkcije kritični odsječci za koje vrijedi:
  - samo jedna dretva može biti unutra (aktivna ili pripravna)
  - dretva izvodi kritične operacije (za sustav/skup zadataka)
  - dretva može programski provjeriti jesu li tražena sredstva dostupna (ili drugi uvjeti za nastavak njena rada)
    - ako su sredstva dostupna – dretva ih zauzima i nastavlja
    - ako sredstva nisu dostupna – dretva se blokira i prividno napušta semafor

  - dretva u monitoru otpušta sredstva (mijenja sustav)
    - ako dretve čekaju na sredstva, oslobodi ih (ili samo prvu)
      - oslobođene dretve prije svog nastavka ipak moraju ponovno ući u monitor

Monitori

- monitor može biti podržan od strane programskog jezika (npr. ključna riječ *synchronized* u Javi), ili preko dodatnih biblioteka (npr. pthread.h)

- sučelje za ostvarenje monitora mora omogućavati:
  - sučelje za ulaz u monitor
  - sučelje za izlaz iz monitora
  - sučelje (i mehanizam) za blokiranje dretve u monitoru (i privremeno otpuštanje monitora)
  - sučelje (i mehanizam) za otpuštanje blokiranih dretvi

- u većini sustava monitori se implementiraju sa:
  - *varijablama međusobnog isključivanja (mutex*) i
  - *uvjetnim varijablama*

Varijable međusobnog isključivanja

- Mehanizam je vrlo sličan binarnom semaforu
- Međutim, binarni semafor je samo koncept
  - vrlo je rijetko implementiran
  - uglavnom se nudi samo opći semafor

- Sučelje varijabli međusobnog isključivanja:
  - `Zakljucaj(m_id)` (sinonim: *uđi_u_monitor/krit.odsj.*)
  - `Otkljucaj(m_id)` (sinonim: *izađi_iz_monitora/K.O.*)

- Razlika u odnosu na binarni semafor:
  - osmišljeno samo za sinkronizaciju **kritičnog odsječka**
  - dodatne funkcionalnosti kada se koristi sa uvjetnim varijablama (koncept monitora)

Uvjetne varijable

- Ponekad je potreban mehanizam koji će dretvi omogućiti da se blokira u redu (do nekog budućeg događaja)
  - dretva može ustanoviti (programskim ispitivanjem) da uvjeti za njen nastavak rada nisu ispunjeni (npr. provjerom stanja varijabli) te traži svoje blokiranje u nekom redu
  - tek kada se uvjeti **u budućnosti** poprave (druge ih dretve mijenjanju) tek se tada dretva može odblokirati (i to moraju napraviti druge dretve)
  - uvjeti se provjeravaju i mijenjaju korištenjem varijabli koje su zajedničke za više dretvi
    - provjera uvjeta i njegovo mijenjanje mora biti u kritičnom odsj.
    - blokiranje dretve u kritičnom odsječku dovelo bi do potpunog zastoja
    - ovakvo blokiranje stoga mora biti popraćeno (u istoj jezgrinoj funkciji) privremenim izlaskom iz kritična odsječka
- Opisani mehanizam naziva se *uvjetna varijabla*

Uvjetne varijable

- Uvjetne varijable mogu se teoretski koristiti i bez varijabli međusobnog isključivanja, ali vrlo rijetko tako nešto ima opravdanja (neće dovesti do problema)

- Sučelja uvjetnih varijabli:

  - **`Cekaj_u_redu(red_id, mi_id)`**
    - dretva se stavlja u red **`red_id`** i oslobađa se varijabla međusobnog isključivanja **`mi_id`**

  - **`Oslobodi_iz_reda(red_id)`**
    - oslobodi prvu dretvu iz reda **`red_id`**

  - **`Oslobodi_sve_iz_reda(red_id)`**
    - oslobodi sve dretve iz reda **`red_id`**

Tipična monitorska funkcija – "zauzmi"

```
m-funkcija zauzmi_sredstvo()
{
   Zakljucaj(m_id)

   dok (sva potreba sredstva nisu raspoloziva) //ne ako
        Cekaj_u_redu(red_id, m_id);

   //sredstva se označuju kao zauzeta – dodjeljuju se dretvi
   //ili se sredstva ovdje koriste, unutar monitora

   Otkljucaj(m_id);
}
```

Složeni uvjet

Tipična monitorska funkcija – “otpusti”

```
m-function otpusti_sredstva()
{
   Zakljucaj(m_id)

   //označi sredstva kao slobodna

   ako (dretve cekaju na sredstva)
       Oslobodi_iz_reda(red_id);

   (oslobađanje dretvi može se napraviti i bez provjere uvjeta ako se
   uvjeti u “zauzmi” monitorskoj funkciji provjeravaju “dok” petljom, a ne
   samo jednom sa “ako”;
   također može se koristiti i Oslobodi_sve_iz_reda(red_id);)

   Otkljucaj(m_id);
}
```

Primjer korištenja monitora

- Monitori se mogu koristiti za jednostavne i složene sinkronizacijske probleme
- Obzirom na jasniju predodžbu sinkronizacije (uvjeti se izravno ispituju provjerom varijabli), **monitori su preferirano sinkronizacijsko sredstvo**

- Opis problema za primjer
  - U nekom sustavu više dretvi čeka poruke
  - Poruke u sustav dolaze zajedničkim kanalom (napravom) te ih treba proslijediti prema odgovarajućoj dretvi
  - Prosljeđivanje se ne radi izravno:
    - zasebna dretva dohvaća poruke te obavještava ostale o tom događaju
    - po obavijesti dretve provjeravaju je li dotična poruka za njih
    - pretpostavka je da će uvijek samo jedna dretva uzeti poruku (poruka je samo njoj namijenjena!)

Primjer korištenja monitora

- Dretve sustava (za primjer):
  - dretva za dostavljanje: **`dostava`**
    - čeka nad uređajem – izvorom poruka
    - kada se poruka pojavi, dretva ostalima signalizira pojavu
  - dretve za obradu: **`obrada`**
    - svaka dretva čeka na zaseban tip poruke
    - nakon provjere zaglavlja pristigle poruke, dretva će poruku ili ignorirati ili uzeti (i signalizirati to dretvi za dostavljanje)
- Dretve su cikličke – ponavljaju navedene operacije do kraja, tj. dok kraj nije označen sa **`nije_kraj`** funkcijom (ili varijablom)

Primjer korištenja monitora

- Podatkovna struktura:
  - monitor:
    - **`m1`** – varijabla međusobnog isključivanja
    - **`c1`** i **`c2`** – uvjetne varijable
      - **`c1`** – red za dretve **`obrada`** koje čekaju na pojavu poruke (čekaju signalizaciju da je poruka pristigla koju daje **`dostava`**)
      - **`c2`** – red za dretvu **`dostava`** koja čeka signal da ja zadnja isporučena poruka preuzeta od neke dretve **`obrada`**
  - **`dodijeljena`** – zajednička varijabla (globalna) koja označava je li zadnja pristigla poruka preuzeta od neke dretve obrada ili još nije
    - ako nije postavljena, dretve će provjeravati zaglavlje poruke

Dretva za dostavljanje poruka

```
dostava ()
{
   dodijeljena = ISTINA;  // zajednička varijabla
   dok (nije_kraj()) {
        cekaj_na_poruku();  // čeka na ulaznoj napravi

        Zakljucaj(m1);
        dohvati_poruku_u_zajednicki_spremnik();
        dodijeljena = NEISTINA;
        Oslobodi_sve_iz_reda(c1);

        // čekaj dok neka dretva ne preuzme poruku
        dok (dodijeljena == NEISTINA)
              Cekaj_u_redu(c2, m1);
        Otkljucaj(m1);
   }
}
```

Dretve za obradu poruka

```
obrada_i()  //"i"-ta dretva obrade
{
   Zakljucaj(m1);
   dok (nije_kraj()) {
       ako (dodijeljena == NEISTINA &&
           (poruka pripada ovoj dretvi) ) {
             uzmi_poruku();
             dodijeljena = ISTINA;
             Oslobodi_iz_reda(c2);
             Otkljucaj(m1);
             obradi_poruku();
             Zakljucaj(m1);
       } inace
             Cekaj_u_redu(c1, m1);
   }
    Otkljucaj(m1);
}
```

Složeni uvjet

Isti problem sa semaforima?

```
dostava ()
{
   dodijeljena = ISTINA;  // zajednička varijabla
   dok (nije_kraj()) {
        cekaj_na_poruku;  // čeka na ulaznoj napravi

        dohvati_poruku_u_zajednicki_spremnik();
        dodijeljena = NEISTINA;

        za i = 1 do broj_dretvi
              PostaviSem(s1);

        // čekaj dok neka dretva ne preuzme poruku
        CekajSem(s2);
   }
}
```

Isti problem sa semaforima?

```
obrada_i()
{
   dok (nije_kraj()) {
        ako (dodijeljena == NEISTINA &&
             (poruka pripada ovoj dretvi) ) {
               uzmi_poruku();
               dodijeljena = ISTINA;
               PostaviSem(s2);
               obradi_poruku();
        } inace
               CekajSem(s1);
   }
}
```

- Problemi: mnogi; rješenja: mnoga; **dobra rješenja**?
  - ponekad se sinkronizacija može ispravno izvesti i sa semaforima ako se svakoj dretvi pridijeli vlastiti semafor
  - *monitorski* principi ponekad mogu *upaliti* i sa semaforima

Dodatni primjer sinkronizacije s monitorima

- Problem pet filozofa

- Primjeri zadataka:

http://www.cs.berkeley.edu/~kubitron/courses/cs162-F06/hand-outs/synch-problems.html

- i njihovih rješenja:

http://www.cs.berkeley.edu/~kubitron/courses/cs162-F06/hand-outs/synch-solutions.html

- Primjeri i objašnjenja:

http://www.zemris.fer.hr/~leonardo/unofficial/radovi/Sinkronizacija_MIPRO07.pdf