SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
Zavod za elektroniku, mikroelektroniku,
računalne i inteligentne sustave

Interni materijal za predavanja iz predmeta

# Operacijski sustavi

Autor: Leonardo Jelenković

Ak. god. 2024./2025.

Zagreb, 2024.

# Sadržaj

# Prije uvoda

Materijali ove skripte su dijelom sažeci (natuknice) sadržaja knjige *Operacijski sustavi* autora *Lea Budina i ostalih* te prate isti slijed izlaganja. Neke teme su prikazane malo drukčije, neke proširene, neke izostavljene, uz uglavnom druge primjere zadataka.

U okviru ovog predmeta razmatraju se mehanizmi operacijskog sustava – algoritmi, postupci (i slično) koji se koriste da bi se ostvarila neka operacija te kako (kojim sučeljem) se ona može iskoristiti iz programa. Uglavnom se ne razmatra korištenje operacijskog sustava od strane korisnika za neke (uobičajene) operacije (npr. kako pronaći neku datoteku, kako promijeniti neku postavku) – pretpostavlja se da slušači ovog predmeta to već znaju ili mogu samostalno savladati. Izuzetak je ovaj kratki uvod u kojem se u vrlo kratkim crticama opisuje funkcionalno ponašanje računala.

**Podaci i programi pohranjeni su:**

- na disku: HDD, SSD, USB ključić, CD/DVD i sl.
  - takozvani perzistentni spremnici (trajni, čuvaju podatke i kad se računalo ugasi)
  - na njih su pohranjeni: operacijski sustav, programi, podaci, multimedija, ...
- u radnom spremniku (memoriji, RAM)
  - dijelovi OS-a, pokrenuti program i njihovi podaci
  - koristi se samo kada računalo radi (nije ugašeno)
  - kad se računalo ugasi ti se podaci gube (ako nisu pohranjeni na disk)
- u ROM-u – memoriji vrlo mala kapaciteta, koja se samo koristi prilikom pokretanja računala
  - BIOS/UEFI za osobna računala, radne stanice i poslužitelje
    - BIOS – *Basic Input/Output System* (starija računala)
    - UEFI – *Unified Extensible Firmware Interface* (novija računala)
  - "program pokretač" (engl. *bootloader*) kod drugih sustava (ugrađenih)

**Što se dogodi kad se pokrene računalo?**

1. Najprije se pokreću programi iz ROM-a (BIOS/UEFI) koji započinju s inicijalizacijom sklopovlja (procesora, memorije, ...)
2. Nakon toga program iz BIOS/UEFI s diska učitava drugi program pokretač (engl. *bootloader*) koji preuzima upravljanje te učitava operacijski sustav (ili taj program učita drugi koji to napravi do kraja)
3. Operacijski sustav učitava i priprema svoju jezgru (upravljanje procesima, datotečnim podsustavom, mrežnim podsustavom, korisničko sučelje, ...).
4. Pokreću se dodatne usluge (servisi – procesi u "pozadini") koji su dio ili proširenje operacijskog sustava kao što su sigurnosna stijena (engl. *firewall*), antivirusna zaštita, ažuriranje sustava, upravljanje elementima sustava, ostale usluge (vođenje dnevnika o događajima u sustavu, praćenje promjena priključenih naprava, ...).
5. Pokreću se usluge koje su dio korisničkog proširenja sustava kao što su:

- programi za prilagodbu ponašanja dijelova sustava (zaslon, zvuk, posebne naprave, provjera ažuriranja upravljačkih programa, ...)
- pohrana podataka u oblaku (npr. Dropbox, OneDrive, Google Drive)
- komunikacijski alati (npr. Skype, Viber, WhatsApp, . . . )
- ostale usluge

**Korištenje računala od strane korisnika**

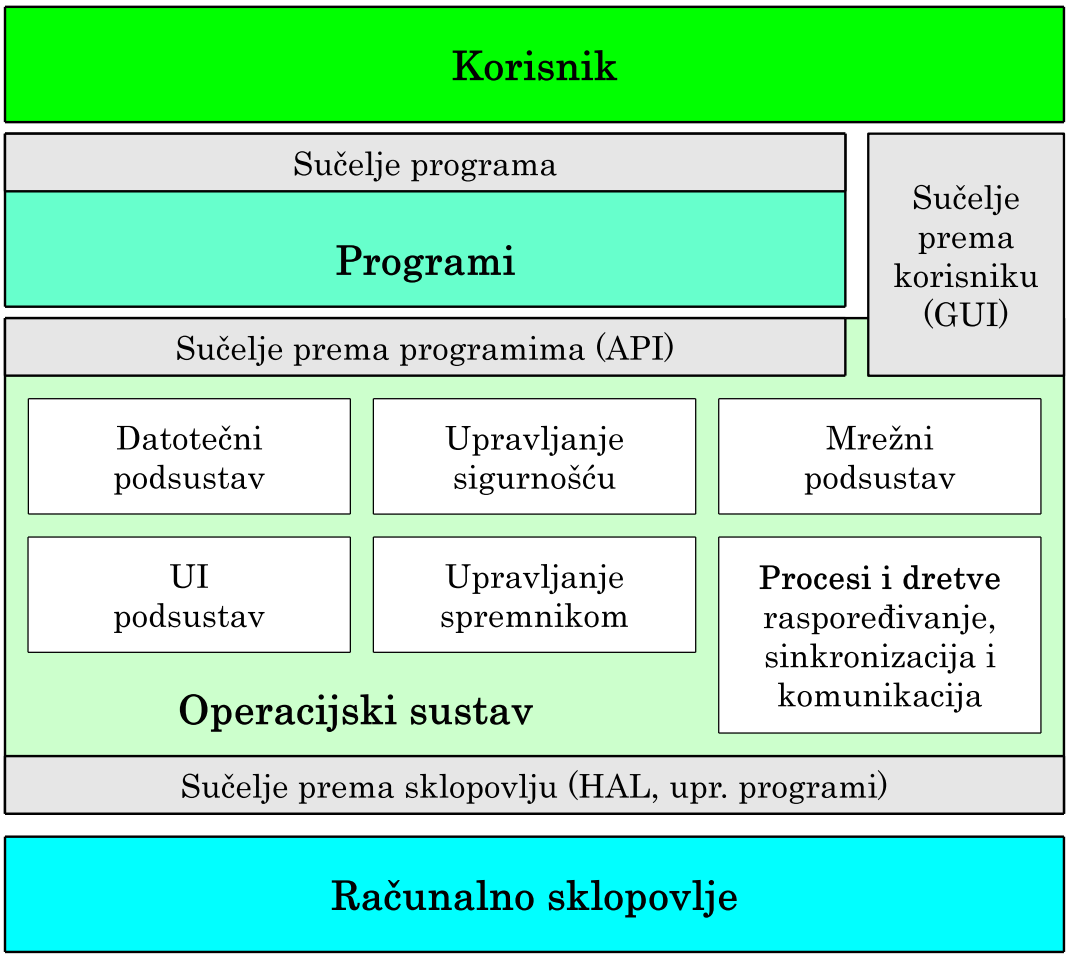
1. Korisnik preko sučelja operacijskog sustava (najčešće grafičkog):
   - pokreće programe (koji time postaju procesi)
   - zaustavlja procese
   - zadaje naredbe operacijskom sustavu (gašenje, stavljanje u stanje pripravnosti, promjena postavki izgleda i potrošnje i slično)
   - dodaje (*instalira*) nove programe, miče programe koji više nisu potrebni
   - ...
2. Svaki program preko svojeg sučelja omogućava korisniku rad s tim programom.
   - radi jednostavnosti korištenja uobičajeno program nude ista/slična sučelja za obavljanje istih operacija (npr. na jednaki način zadaju otvaranje datoteke)

# 1. UVOD

## 1.1. Računalni sustav

U današnje doba postoji mnoštvo različitih računala i sustava temeljenih na njima kao što su osobna i prijenosna računala, radne stanice, poslužitelji, pametni telefoni i slični uređaji, mikroupravljači. Operacijski sustavi (OS) koji se u njima koriste se također međusobno razlikuju. Međutim, temeljni elementi svih sustava se zasnivaju na istim osnovnim načelima. Iako je sustav koji se ovdje razmatra najbliži osobnom računalu, većina navedenog vrijedi i za ostale.

*Računalni sustav* se sastoji od sklopovlja (*hardvera* – računala), programske potpore (operacijski sustav i programi) te korisnika, kako je prikazano na slici 1.1.




**Slika 1.1. Računalni sustav, komponente (podsustavi) OS-a**

**Korisnici** koriste sustav radi svojih potreba, npr. obrada teksta, rad na Internetu, igre, multimedija i slično.

**Programi** su napisani da nude neke korisne operacije korisnicima. Obične operacije izvode se izravnim izvođenjem instrukcija programa na procesoru. Operacije koje zahtjevaju korištenje sklopovlja ili drugih zajedničkih sredstava izvode se tako da program pozove funkciju operacijskog sustava te ju OS obavi za program.

**Operacijski sustav** upravlja sustavom prema zahtjevima korisnika i programa. Omogućuje korisnicima i programima jednostavno korištenje sredstava sustava kroz odgovarajuća sučelja. Korisnici i programi stoga ne moraju poznavati (složene) detalje sklopovlja koje se koristi radi izvedbe njihovih naredbi.

Među **slojevima** se nalazi **sučelje**

- sučelje definira način korištenja/komunikacije među slojevima

Podsustavi OS-a imaju svoju ulogu, ali nisu posve razdvojeni kao na slici 1.1.

## 1.2. Operacijski sustav

**DEFINICIJA:** *Operacijski sustav* je skup osnovnih programa koji:

- omogućuju izvođenje radnih zahvata na računalu
- omogućuju izvođenje operacija računala

**Svrha/uloga OS-a**

- olakšavanje korištenja računala (skriva detalje)
- omogućava učinkovito korištenje svih dijelova računala (ima upravljačke programe)
- omogućava višeprogramski rad – najbitnija uloga, povećava učinkovitost sustava

"**OS olakšava korištenje računala**"

- skriva nepotrebne detalje od korisnika i programa
  - korisnici koriste korisničko sučelje koje nude programi i OS
  - programi koriste API koje nudi OS
- upravljački programi ("drajveri") znaju kako sa sklopovljem
  - OS ih koristi
  - jedan upravljački program radi samo za pojedinu komponentu računala
  - na različitim računalima koriste se različiti upravljački programi
- sklopovski različita računala se na jednaki način koriste s istim OS-om
  - isto korisničko sučelje
  - isto sučelje prema programima (API)
  - OS je skoro identičan, samo se koriste drugi upravljački programi

## 1.3. O mjernim jedinicama

S obzirom na to da računalo radi u binarnom sustavu, sklopovlje je podređeno takvom načinu rada. To uključuje i organizaciju spremnika kao i strukture podataka. Baza binarnog sustava jest dva te su uobičajene jedinice oktet/bajt B $= 2^3$ bitova, i veće KB $= 2^{10}$ B $= 1024$ B, MB $= 2^{20}$ B $= 2^{10}$ KB, GB $= 2^{30}$ B $= 2^{10}$ MB, itd.

Navedene jedinice su u suprotnosti sa SI sustavom koji koristi potencije broja 10 (k $= 10^3$, M $= 10^6$, G $= 10^9$ ...).

Stoga su smišljene nove oznake s dodatkom slova 'i': KiB $= 2^{10}$ B, MiB $= 2^{20}$ B, ...

Međutim, vrlo često i u operacijskim sustavima i u literaturi se i dalje koriste "stare" oznake (bez 'i'), ali podrazumijevajući potencije od 2 a ne 10. Isto je i s ovom skriptom. Izuzetak su brzine prijenosa, gdje je uobičajeno (i u računalnom okruženju) koristiti potencije broja 10.

Kada se te jedinice promatraju u nekom okruženju prvo treba ustanoviti koje jedinice se tamo

koriste. Npr. na “Windows” operacijskim sustavima koriste se navedene jedinice u potencijama broja 2 (KB, MB, ...), dok na Linux operacijskim sustavima koriste SI jedinice s potencijama broja 10 (kB, MB, ...). Proizvođači diskova koriste SI sustav jer u tom sustavu se manja vrijednost “prikazuje” većom oznakom. Npr. ako proizvođač kaže da disk ima kapacitet od 1 TB, on misli na $10^{12}$ B, što je $10^{12}/2^{40}$ TiB $= 0,909$ TiB, odnosno, $10^{12}/2^{30}$ GiB $= 931,3$ GiB.

Primjeri (kako interpretirati brojke u ovoj skripti):

- $5$ KB $= 5 \cdot 2^{10}$ B $= 5 \cdot 1024$ B
- $5$ MB $= 5 \cdot 2^{20}$ B $= 5 \cdot 1024 \cdot 1024$ B
- $5$ GB $= 5 \cdot 2^{30}$ B $= 5 \cdot 1024 \cdot 1024 \cdot 1024$ B
- $5$ kbita/s $= 5 \cdot 10^{3}$ bita/s $= 5 \cdot 1000$ bita/s
- $5$ Mbita/s $= 5 \cdot 10^{6}$ bita/s $= 5 \cdot 1000000$ bita/s

## Pitanja za vježbu 1

1. Što je to "računalni sustav"? Od čega se sastoji?
2. Što je to "operacijski sustav"? Koja je njegova uloga u računalnom sustavu?
3. Što je to "sučelje"? Koja sučelja susrećemo u računalnom sustavu?
4. Navesti osnovne elemente (podsustave) operacijskog sustava.

# 2. MODEL JEDNOSTAVNOG RAČUNALA

## 2.1. Sabirnički model računala

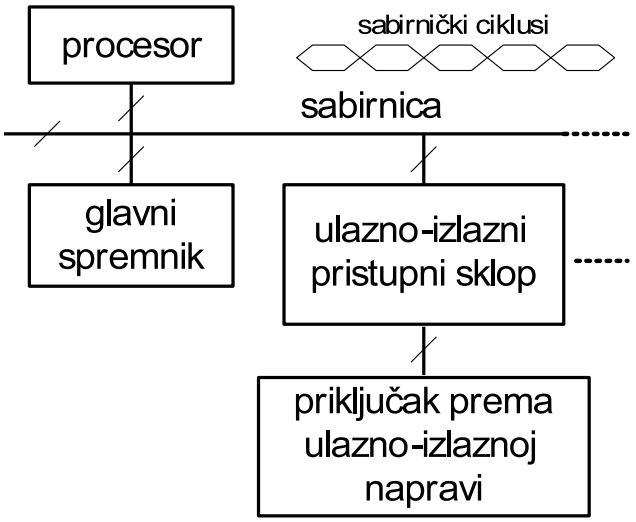
U sabirničkom modelu, računalo se (iznutra) sastoji od procesora, spremnika, ulazno-izlaznih naprava te sabirnice.

*Procesor* je osnovni element sustava, on izvodi instrukciju za instrukcijom i time omogućava provođenje korisniku potrebnih operacija.

*Spremnik* (radni spremnik, memorija) služi za privremenu pohranu instrukcija, podataka i rezultata. Procesor neprestano dohvaća i pohranjuje podatke iz spremnika pri svom radu. Stoga je spremnik, nakon procesora, najbitniji element sustava (i najbrži, nakon procesora).

*Ulazno-izlazne naprave* koriste međusklop (pristupni sklop, kontroler) za povezivanje u računalo. Više o napravama u idućem poglavlju.

*Sabirnica* omogućuje povezivanje elemenata računala – u sabirničkom modelu svi se elementi spajaju na sabirnicu i koriste ju za međusobnu komunikaciju.




**Slika 2.1. Sabirnički model računala**

*Sabirnički ciklus*

- prijenos jednog podatka između dva sklopa (procesor – spremnik, procesor – pristupni sklop, pristupni sklop – spremnik)

Sabirnicom upravlja procesor

1. postavlja adresu na adresni dio sabirnice
2. postavlja podatak (kada se on zapisuje u spremnik ili UI) na podatkovni dio
3. postavlja upravljačke signale

**Primjer 2.1. Trajanje sabirničkog ciklusa**

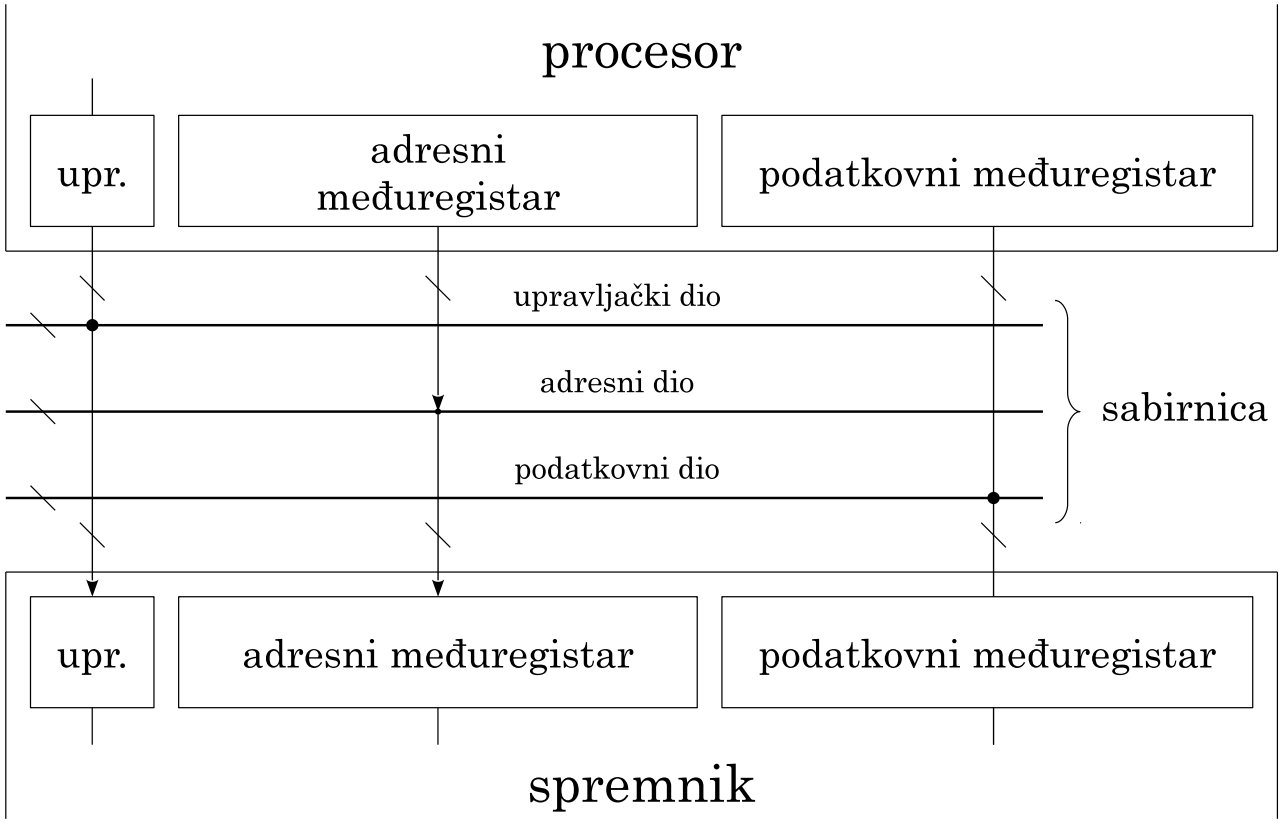
Neka je trajanje jednog sabirničkog ciklusa $T_B = 1$ ns, tada:

- frekvencija rada sabirnice jest $\frac{1}{T_B} = 1$ GHz
- ako je širina sabirnice 64 bita tada je propusnost sabirnice: $64 \cdot 1$ GHz $= 64$ Gbita/s (G je u ovom slučaju (za brzine) $10^9$ a ne $2^{30}$)

## 2.2. Kratki opis komponenata računala

U ovom poglavlju su razmotrene samo osnovne komponente: procesor, spremnik i sabirnicu.

### 2.2.1. Sabirnica

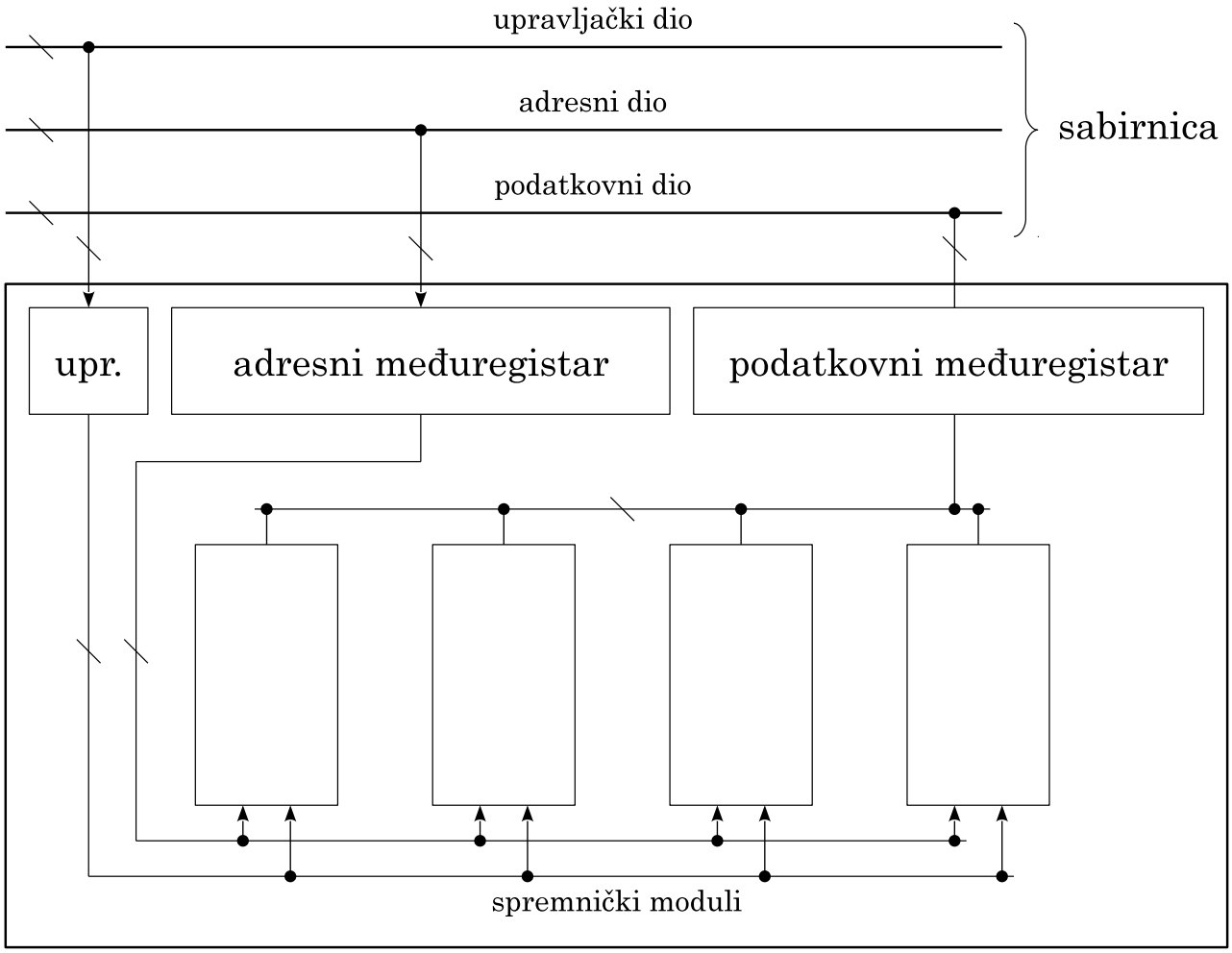



**Slika 2.2. Model sabirnice**

Sabirnica se sastoji od tri dijela:

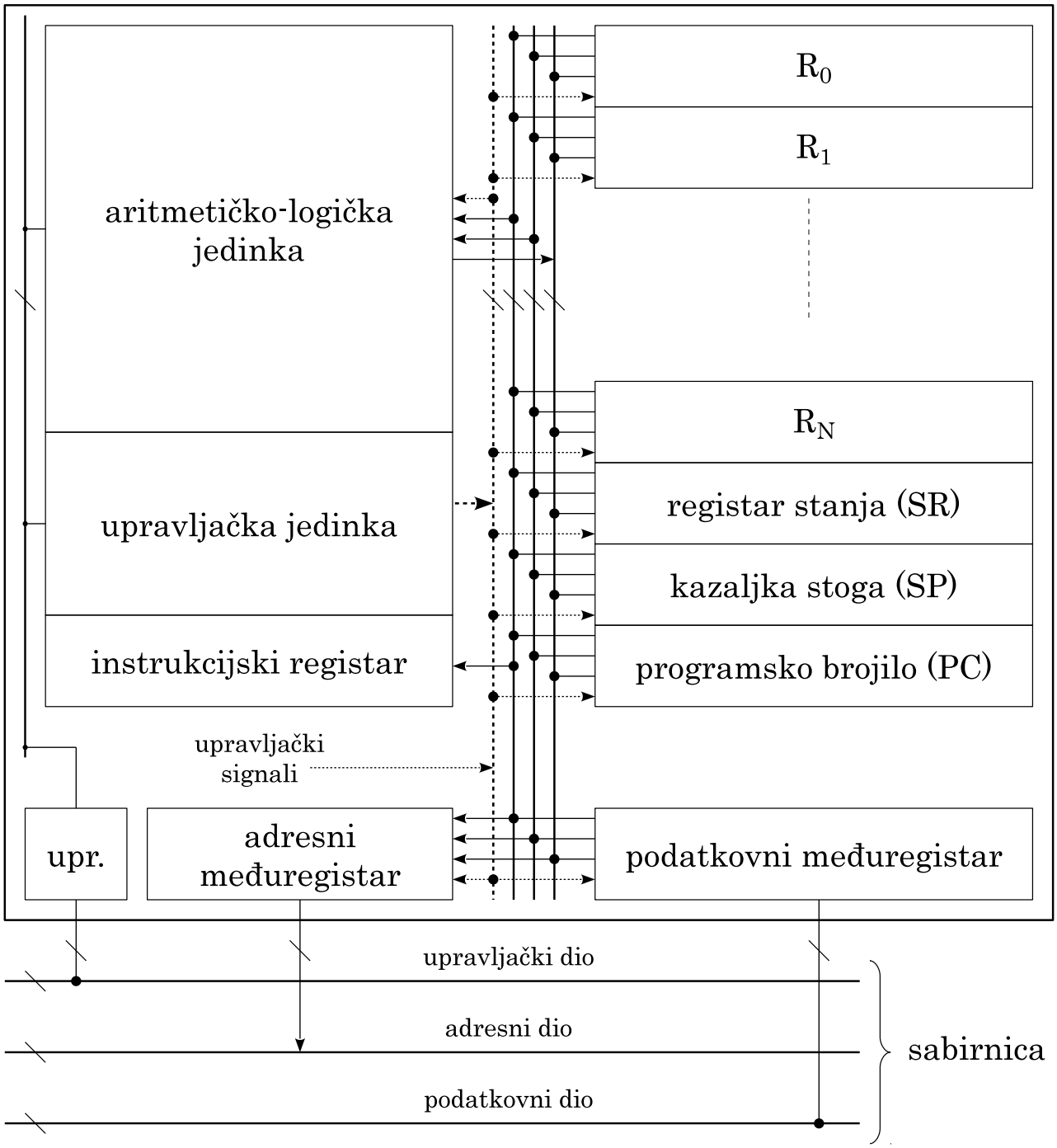
- adresnog dijela
- podatkovnog dijela
- upravljačkog dijela (npr. signali piši ili čitaj, BREQ, BACK, prekidi, ...)

### 2.2.2. Spremnik




**Slika 2.3. Model spremnika**

### 2.2.3. Procesor




**Slika 2.4. Model procesora**

Procesor se sastoji od elemenata:

- aritmetičko-logička (AL) jedinka
- registri opće namjene (npr. R0-R7)
- programsko brojilo PC
- kazaljka stoga SP
- registar stanja SR
- *upravljačka jedinka*
  - *instrukcijski registar*
  - *upravljački signali*
- *adresni međuregistar*
- *podatkovni međuregistar*

*Koso označeni elementi* nisu izravno dostupni programeru. Ostali elementi jesu i čine "programerski model" procesora.

Procesor se može opisati kao automat koji ciklički obavlja slijedeće operacije:

**Isječak kôda 2.1. Opis rada procesora**

```
ponavljaj {
    dohvati instrukciju na koju pokazuje PC
    povećaj PC tako da pokazuje na iduću instrukciju
    dekodiraj instrukciju
    obavi operaciju zadanu instrukcijskim kodom
        ( ovisi o instrukciji, npr.\ za AL instrukciju može biti:
          dohvati operande, obavi AL, spremi rezultat )
}
dok je procesor uključen
```

**Instrukcije se mogu podijeliti na instrukcije za:**

- premještanje sadržaja
- obavljanje AL operacija
- programske skokove i grananja
- posebna upravljačka djelovanja (npr. zabrana prekida)

**Instrukcija (u strojnom obliku – niz bitova) se sastoji od:**

- operacijskog koda (“koja operacija”) i
- adresnog dijela (operandi, adrese)

**Posebno zanimljive instrukcije (s aspekta upravljačkog djelovanja OSa)**

- ostvarenje instrukcija skoka: `adresa skoka => PC`
- poziv potprograma: `PC => stog; adresa potprograma => PC`
- povratak iz potprograma: `stog => PC`

[dodatno]

ARM procesori koriste poseban registar (LR=R14, *link register*) za pohranu povratne adrese; međutim, radi ugnježđavanja programski je potrebno na početku funkcije pohraniti taj registar na stog (čime se postiže sličan učinak kao i gornji prikaz poziva i povratka iz potprograma koji je uobičajen za većinu arhitektura).

U nastavku se koristi asembler sličan ARM-ovom, uz neke razlike (primjerice poziv i povratak iz potprograma nije kao kod ARM-a već uobičajeni sa CALL i RET).

## 2.3. Instrukcijska dretva

Pojam "dretva" – postolarski konac; iz "Čudnovate Zgode Šegrta Hlapića" Ivane Brlić Mažuranić: [...] *U torbu metne jedan modar rubac, pa jedno šilo, malo dretve i nekoliko komadića kože. Hlapić je, naime, bio pravi mali majstor, a postolar ne može da bude bez šila i dretve, kao ni vojnik bez puške.* [...]

Razlikujemo niz instrukcija (program) i izvođenje niza instrukcija (dretva).

### Primjer 2.2. Primjer programa s grananjem i pozivima potprograma

Program (učitan u memoriju):

```
; Računanje faktorijela
; zadano: N, pretpostavlja se N > 0 !
; rezultat spremiti u: REZ
100         LDR R1, N   ; učitaj N u R0
104         LDR R0, N   ; R0 akumulira rezultat (umnožak)
108 PETLJA: SUB R1, 1   ; R1 = R1 - 1
112         CMP R1, 1   ; usporedi R1 i 1
116         BLE KRAJ    ; ako je R1 <= 1 skoči na KRAJ
120         PUSH R1     ; stavi R1 na stog - parametar funkcije
124         CALL MNOZI
128         ADD SP, 4   ; makni R1 s vrha stoga
132         B PETLJA    ; skoči na PETLJA
136 KRAJ:   STR R0, REZ ; spremi R0 u REZ
            ...
; potprogram
200 MNOZI:  LDR R2, [SP+4]  ; dohvati parametar
204         MUL R0, R2      ; R0 = R0 * R2;
208         RET
```

Program ima 13 instrukcija. Međutim, pri izvođenju za različite `N`-ove dretva će obaviti različiti broj instrukcija.

Za `N=3` samo će se jednom pozvati potprogram `MNOZI`, tj. ukupno će obaviti: $2+(7+3)+4 = 16$ instrukcija.
Za `N=4`: $2 + 2 * (7 + 3) + 4 = 26$ instrukcija.
Za `N=100`: $2 + 98 * (7 + 3) + 4 = 986$ instrukcija.

Primjer izvođenja za `N=4` (dretva)

```
100         LDR R1, 4
104         LDR R0, 4
108 PETLJA: SUB R1, 1 ; R1 = 3 nakon ove instrukcije
112         CMP R1, 1 ;
116         BLE KRAJ  ; nije ispunjen uvjet skoka
120         PUSH R1
124         CALL MNOZI
200 MNOZI:  LDR R2, [SP+4] ; R2 = 3
204         MUL R0, R2     ; R0 = 4*3 = 12
208         RET
128         ADD SP, 4
132         B PETLJA
108 PETLJA: SUB R1, 1 ; R1 = 2
112         CMP R1, 1
116         BLE KRAJ  ; nije ispunjen uvjet skoka
120         PUSH R1
124         CALL MNOZI
```

```
200 MNOZI:  LDR R2, [SP+4] ; R2 = 2
204         MUL R0, R2     ; R0 = 12*2 = 24
208         RET
128         ADD SP, 4
132         B PETLJA
108 PETLJA: SUB R1, 1 ; R1 = 1
112         CMP R1, 1
116         BLE KRAJ  ; ispunjen je uvjet skoka
136 KRAJ:   STR R0, REZ
```

---

**Program**

- niz instrukcija (i podataka) koji opisuju kako nešto (korisno) napraviti
- program = slijed instrukcija u spremniku (“prostorno povezanih”)
- nalazi se u datoteci, na disku, ili u spremniku

**Proces**

- nastaje pokretanjem programa
- traži i zauzima sredstva sustava (spremnik, procesorsko vrijeme, naprave)

Izvođenjem programa (u procesu) instrukcije se ne izvode isključivo slijedno – zbog skokova i poziva potprograma

Slijed instrukcija kako ih procesor izvodi („vezane“ vremenom izvođenja) nazivamo *instrukcijska dretva* ili samo kraće *dretva* (engl. *thread*).

**Dretva**

- dretva = slijed instrukcija u izvođenju (“povezanih vremenom izvođenja”)
- dretva izvodi instrukcije programa, ona radi taj koristan posao

U nekom trenutku stanje dretve je određeno:

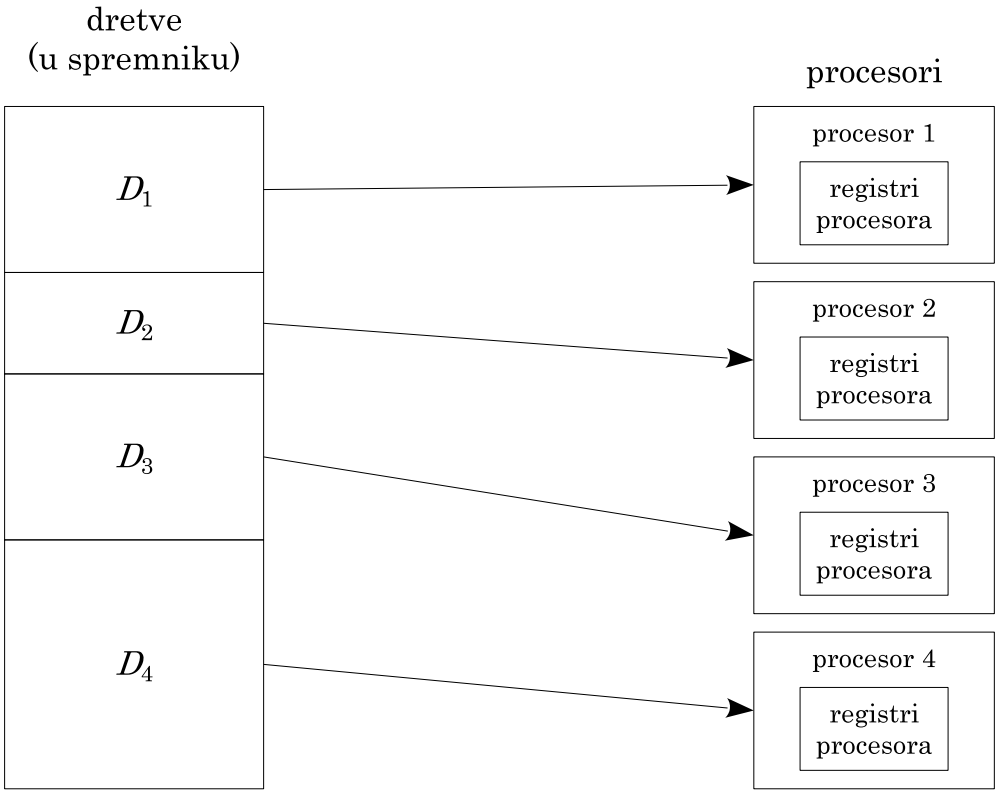
- instrukcijama, podacima koje koristi, sadržajem stoga → sve u spremniku
- sadržajima u registrima procesora → *kontekstom dretve*

**Kako izvoditi više dretvi na jednom procesoru ili na manjem broju procesora od dretvi?**

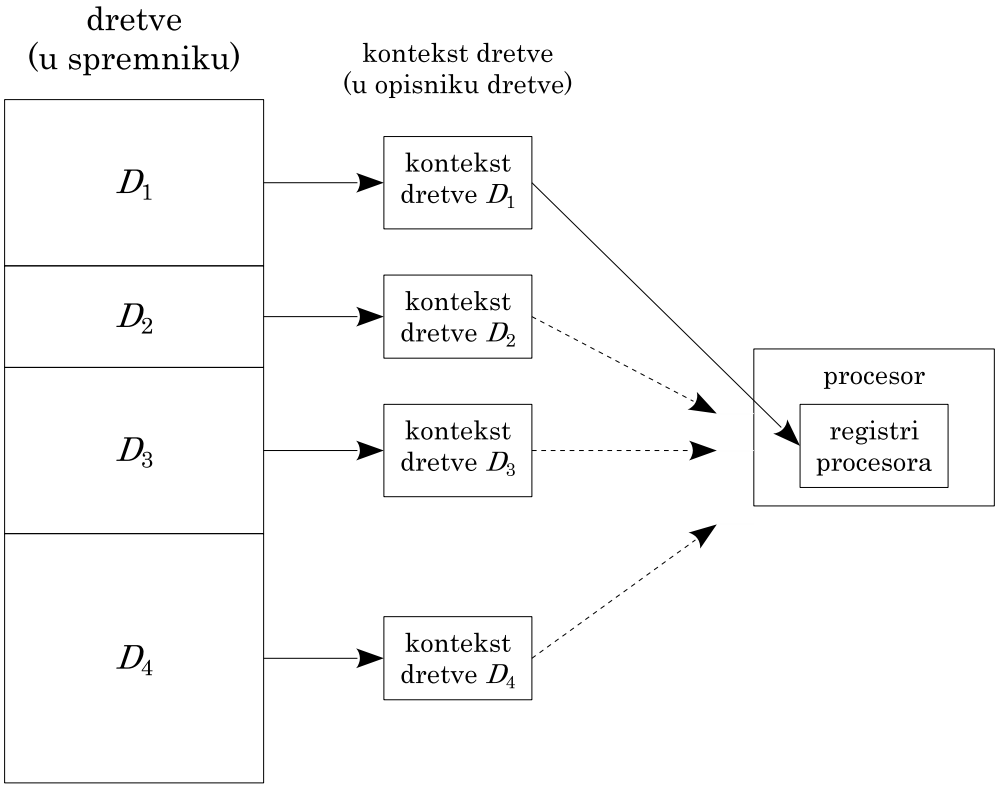
- ideja:
  1. “zamrznuti” dretvu koja se trenutno izvodi; maknuti ju u stranu;
  2. odabrati/uzeti drugu “zamrznutu” dretvu;
  3. “odmrznuti” tu dretvu i nastaviti s njenim radom
- kako to napraviti:
  1. “zamrznuti”: zaustaviti dretvu i pohraniti kontekst te dretve u spremnik
  2. “odmrznuti”: obnoviti kontekst dretve iz spremnika i nastaviti s njenim radom

## 2.4. Višedretveni rad

1. kada ima dovoljno procesora – svaka dretva se može izvoditi na zasebnom procesoru
2. kada nema dovoljno procesora – dretve dijele procesore naizmjeničnim radom




**Slika 2.5. Višedretvenost na višeprocesorskom sustavu**




**Slika 2.6. Višedretvenost na jednoprocesorskom sustavu**

Višedretvenost se (na jednoprocesorskom sustavu) ostvaruje tako da se u nekom odabranom trenutku jedna dretva “zamjeni” drugom.

Postupak zamjene jedne dretve drugom:

1. prekida se izvođenje trenutno aktivne dretve (prekidom, opisano u idućem poglavlju)
2. sprema se kontekst aktivne dretve (trenutni sadržaji registara) u za to predviđeno mjestu u spremniku (u opisnik te dretve)
3. odabire se nova aktivna dretva
4. obnavlja se kontekst novoodabrane dretve (nove aktivne) te
5. aktivna dretva nastavlja s radom

U nastavku (idućim poglavljima) koristi se prikazani model jednostavnog računala, koji se postupno nadograđuje potrebnim sklopovljem i funkcionalnošću.

## Pitanja za vježbu 2

1. Skicirati procesor – njegove osnovne dijelove, registre.
2. Kako se koristi sabirnica?
3. Što procesor trajno radi?
4. Kako se ostvaruju instrukcije "za skok", "za poziv potprograma", "za povratak iz potprograma"?
5. Što predstavljaju pojmovi: program, proces, dretva?
6. Kako se ostvaruje višedretveni rad? Što je to kontekst dretve?

# 3. OBAVLJANJE ULAZNO-IZLAZNIH OPERACIJA, PREKIDNI RAD

U ovom poglavlju se razmatra upravljanje UI napravama s aspekta OS-a. OS treba upravljati napravama i preko nekog sučelja omogućiti i programima da komunikaciju s napravama.

Kako programi koriste naprave (info)?

- pristupa im se kao datotekama, uz dodatne opcije i potrebne provjere povratnih vrijednosti
- u C-u: read/write/scanf/getch, select/poll, fcntl, ...

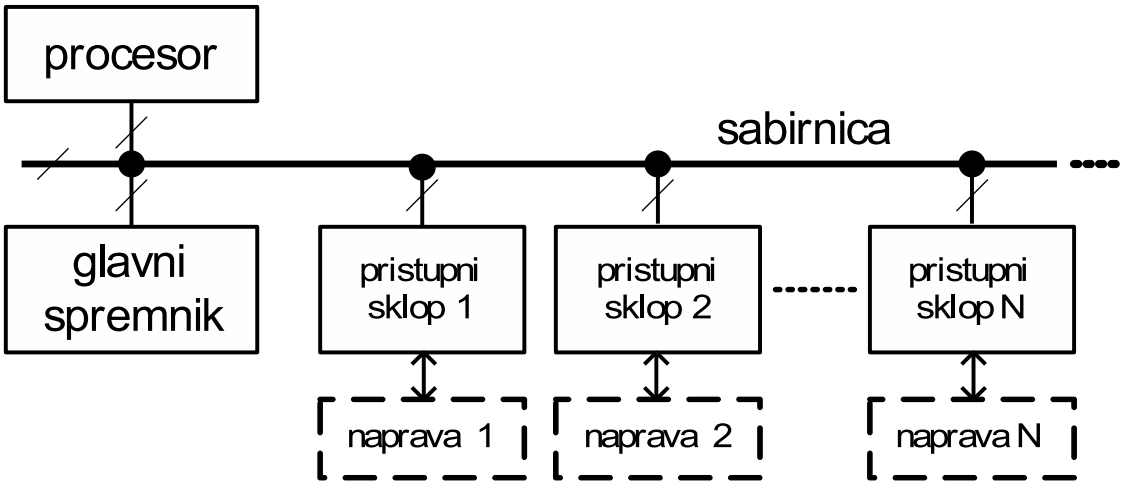
## 3.1. Spajanje naprava u računalo

Naprave koje se spajaju u računalo imaju različita svojstva

- način rada
  - znak-po-znak ili u bloku stalne veličine
  - pristup preko adrese ili korištenje posebnih instrukcija
- brzina prijenosa podataka
- primjeri naprava: zaslon, tipkovnica, miš, zvučnici, disk, mreža, grafička kartica, ...
- u idućim razmatranjima sve osim procesora, spremnika i sabirnice su naprave

### Pristupni sklop naprava

Zbog različitih svojstava naprave se ne spajaju izravno na (glavnu) sabirnicu, već se spajaju preko međusklopa – *pristupnog sklopa*.




**Slika 3.1. Spajanje UI naprava na sabirnicu**

Pristupni sklop:

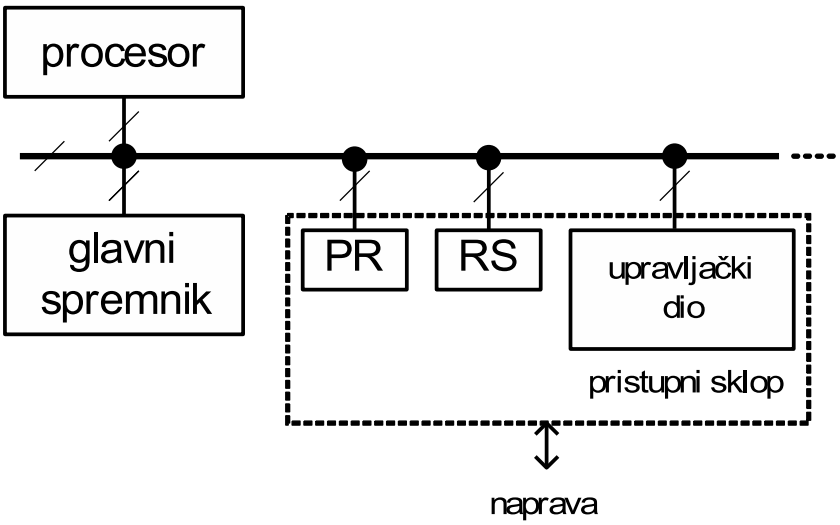
- "zna" komunicirati sa sabirnicom (njenim protokolima)
- "zna" komunicirati s napravom (njenim protokolima, primjerice USB, SATA, Ethernet, HDMI, analogni audio izlaz, S/PDIF ...)

Naprave se mogu koristiti na nekoliko načina:

1. radnim čekanjem
2. prekidima
3. izravnim pristupom spremniku

## 3.2. Korištenje UI naprava radnim čekanjem

Procesor “aktivno” (u programskoj petlji) čeka da naprava postane spremna za komunikaciju




**Slika 3.2. Pristupni sklop UI naprave**

Elementi pristupnog sklopa i njegovo ponašanje

- PR – podatkovni registar, služi za prijenos podataka
- RS – registar stanja, sadrži zastavicu ZASTAVICA koja pokazuje je li pristupni sklop spreman za komunikaciju s procesorom (ZASTAVICA==1) ili nije (ZASTAVICA==0)
- Upravljački dio s jedne strane osluškuje sabirnicu i radi odgovarajuću akciju kad detektira adresu PR-a ili RS-a, a s druge strane upravlja komunikacijom prema priključenoj napravi.
- Procesor u petlji čita RS dok ZASTAVICA ne postane jednaka 1.

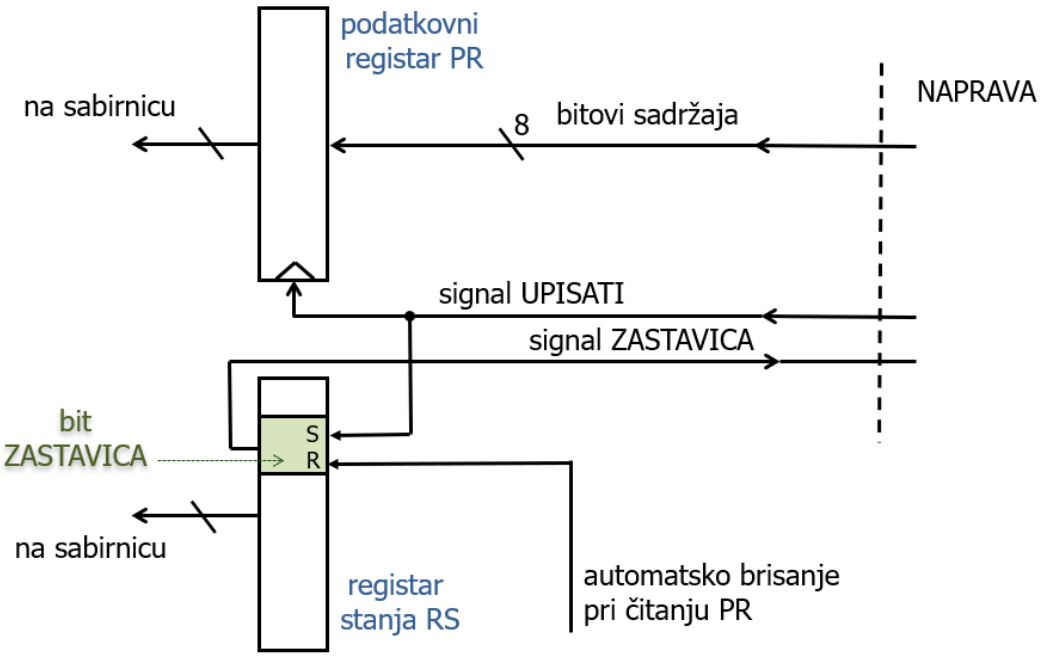
**Isječak kôda 3.1. Primjer čitanja jednog podatka s naprave - u pseudokodu**

```
    dok je ZASTAVICA == 0 radi  //radno čekanje
        ;                       //u petlji čita RS i ispituje bit ZASTAVICA
    pročitaj PR
```

**Isječak kôda 3.2. Primjer čitanja jednog podatka s naprave - u asembleru**

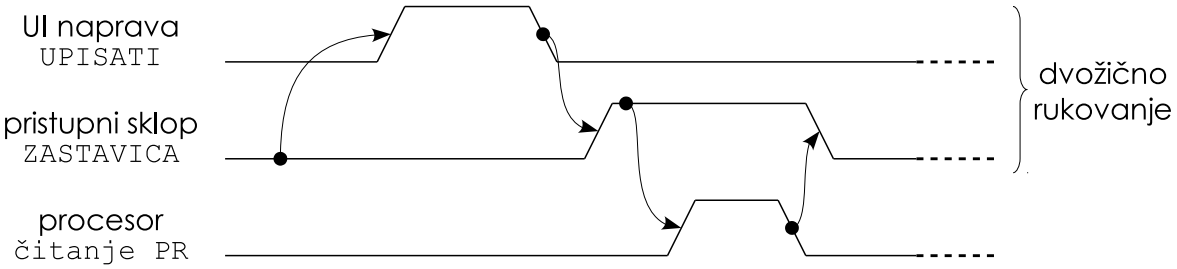
```
    ADR R0, RS      ; adresa registra stanja u R0
    ADR R2, PR      ; adresa podatkovnog registra u R2
petlja:                                                  +\
    LDR R1, [R0]    ; pročitaj registar stanja (sa zastavicom) +| radno
    CMP R1, 0                                                  +| čekanje
    BEQ petlja                                               +/
    LDR R1, [R2]    ; pročitaj znak iz pristupnog sklopa u registar R1
    ... ; obrada pročitanog podatka, npr. samo spremanje u spremnik
```

Brisanje ZASTAVICE po komunikaciji s procesorom može biti ostvareno u pristupnom sklopu (kao na slici 3.3.) ili to može procesor napraviti upisom vrijednosti u RS.




**Slika 3.3. Primjer ostvarenja pristupnog sklopa**

Signali UPISATI i ZASTAVICA se u takvom slučaju koriste za dvožično rukovanje.




**Slika 3.4. Primjer sinkronizacije između naprave, pristupnog sklopa i procesora**

**Svojstva radnog čekanja kao načina upravljanja UI**:

+ prednost: sklopovlje je vrlo jednostavno
− nedostatak: procesor ne radi produktivno, nema koristi od tisuća iteracija petlje
  – drukčijim programom se to ponekad može ublažiti, npr. pojedini sklop se provjerava periodički, provjerava se više sklopova (“prozivanje”), . . .

### Primjer 3.1. Upravljanje temperaturom

Neka postoji jednostavno računalo koje treba očitavati temperaturu od senzora. Međutim, samo očitavanje traje te se pojava novog učitanja označava zastavicom u statusnom registru. Program koji bi čekao na očitanje te potom očitao temperaturu i upalio grijanje ili hlađenje kada je to potrebno, bi izgledao kao u nastavku.

```
program() {
    ponavljaj {
        status = NEMA_OČITANJA
        dok je status == NEMA_OČITANJA radi
            status = dohvati_statusni_registar()

        hlađenje = grijanje = UGAŠENO
        t = dohvati_temperaturu_senzora()
        ako je t < T1 onda
            grijanje = UPALJENO
        ako je T > T2 onda
            hlađenje = UPALJENO
        postavi_hlađenje(hlađenje)
        postavi_grijanje(grijanje)
    }
}
```

### Primjer 3.2. Periodička provjera

Upravljanje se može obavljati kao periodički posao, dodan u nešto drugo. Na taj način se istim postupkom može i upravljati klimom i raditi nešto drugo korisno.

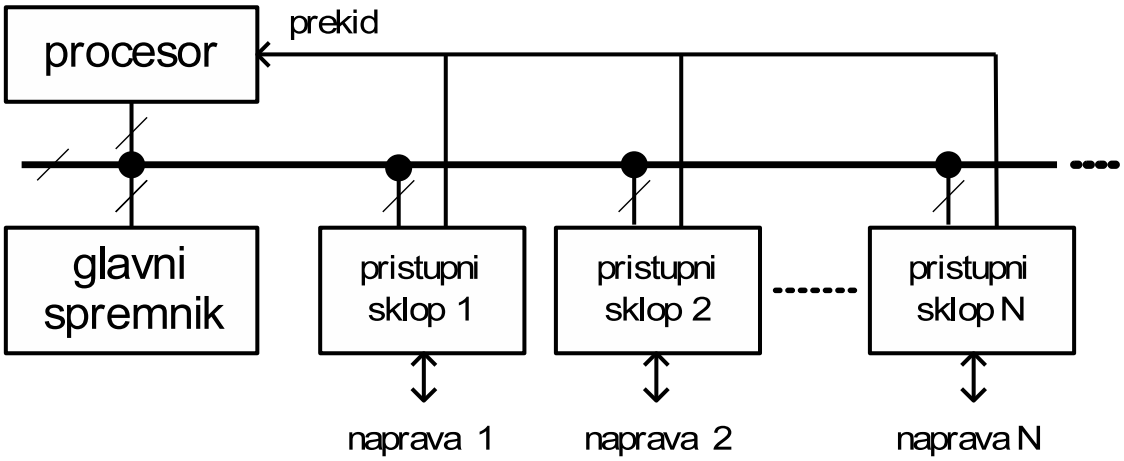
```
funkcija provjeri_temperaturu() {
    status = dohvati_statusni_registar()
    ako je status == NOVO_OČITANJE radi
        hlađenje = grijanje = UGAŠENO
        t = dohvati_temperaturu_senzora()
        ... //ostalo isto kao i u prethodnom primjeru
    }
}
program() {
    ponavljaj {
        ... (nešto drugo)
        provjeri_temperaturu()
        ... (nešto drugo)
    }
}
```

## 3.3. Prekidni rad

Ideja: kada nema podataka od naprave, procesor radi neki drugi manje bitan posao. Kada dođe novi podatak od naprave, naprava sama traži obradu slanjem signala – prekidnog signala do procesora (električnog signala preko žice). Npr. bit ZASTAVICA iz registra stanja pristupnog sklopa postaje signal za prekid.

### 3.3.1. Prekidni rad bez sklopa za prihvat prekida ("bez prioriteta")

Svi zahtjevi za prekide izravno dolaze procesoru preko zajedničkog vodiča (npr. spojeni ILI).




**Slika 3.5. Skica najjednostavnijeg sustava za prekidni rad**

Kako se procesor treba ponašati kad dobije prekidni signal?

Signal može doći u bilo kojem trenutku:

1. dok se izvodi neka dretva
2. dok se obrađuje prijašnji prekid

Procesor može prihvatiti prekidni signal (za 1.) ili ga privremeno ignorirati (za 2.)

Prihvat signala mora popratiti nekoliko akcija:

- mora se omogućiti kasniji nastavak rada prekinute dretve
  - mora se spremiti kontekst prekinute dretve
- mora se spriječiti daljnje prekidanje, dok se prekid ne obradi (ili dok se drukčije ne definira)

#### Načini rada procesora

- Korisnički način rada
  - manje privilegirani, neke stvari nisu dostupne (instrukcije, registri, spremničke lokacije)
- Prekidni načina rada – jezgrin/sustavski način rada
  - privilegirani način rada
  - dozvoljene su sve operacije (instrukcije)
  - dostupni su svi registri/podaci/resursi

## Postupak prihvata prekida od strane procesora (ispitno pitanje)

Operacije koje radi procesor kada nema sklop za prihvat prekida u slučaju zahtjeva za prekidom

a) početno stanje: procesor izvodi neku instrukciju

b) pojavljuje se prekidni signal

c) procesor dovršava trenutnu instrukciju (regularno, ona se ne prekida)

d) po dovršetku instrukcije (na kraju):

+ ako su u procesoru prekidi omogućeni on provjerava je li prekidni signal postavljen

+ ako je prekidni signal postavljen tada **procesor** prihvaća prekid u sljedećim koracima ("postupak prihvata prekida" u užem smislu):

1. zabrani daljnje prekidanje
2. prebaci se u prekidni način rada (jezgrin, sustavski)
   * adresiraj sustavski dio spremnika, aktiviraj sustavku kazaljku stoga
3. na stog (sustavski) pohrani programsko brojilo (PC) i registar stanja (SR) (tzv. minimalni kontekst)
4. u PC stavi adresu “prekidnog potprograma”
   * ta je adresa najčešće ili ugrađena u procesor (bira se prema vrsti prekida), ili se koristi tablica za odabir adrese (i koristi se prekidni broj)

Navedeno je **ugrađeno ponašanje procesora**.

Opis rada procesora (kod sa 2.1.) treba proširiti tim operacijama.

**Isječak kôda 3.3. Opis rada procesora s podrškom za prekide**

```
ponavljaj {
   dohvati instrukciju na koju pokazuje PC
   povećaj PC tako da pokazuje na iduću instrukciju
   dekodiraj instrukciju
   obavi operaciju zadanu instrukcijskim kodom

   ako su prekidi omogućeni i prekidni signal je postavljen tada
      zabrani daljnje prekidanje
      prebaci se u prekidni način rada
      na stog pohrani programsko brojilo i registar stanja
      u programsko brojilo stavi adresu prekidnog potprograma
}
dok je procesor uključen
```

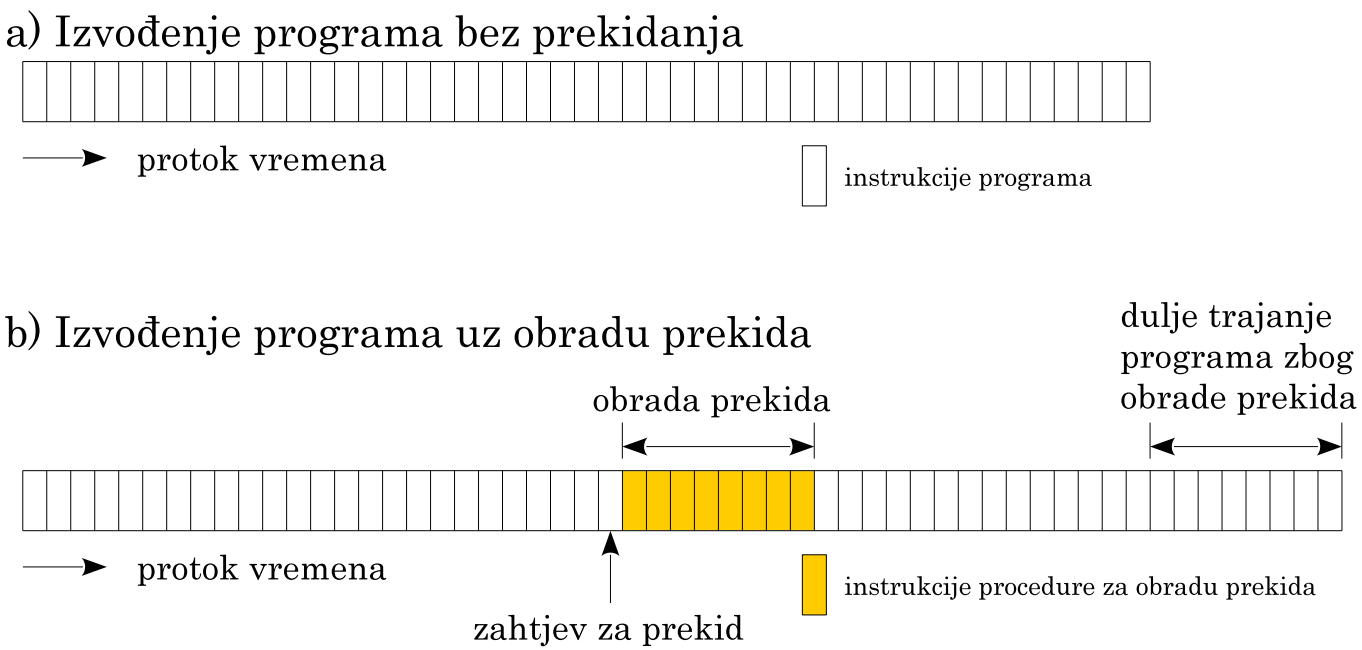
Prekidni potprogram, koji se počinje izvoditi nakon točke d), može izgledati ovako:

```
Prekidni potprogram
{
   pohrani kontekst na sustavski stog (osim PC i SR koji su već tamo)
   ispitnim lancem odredi uzrok prekida (ispitujući RS pristupnih sklopova) => I
   signaliziraj napravi da je njen prekid prihvaćen te da spusti prekidni signal

   obradi prekid naprave I //npr. poziv funkcije upravljačkog programa

   obnovi kontekst sa sustavskog stoga (osim PC i SR)
   vrati_se_u_prekinutu_dretvu //jedna instrukcija
}
```

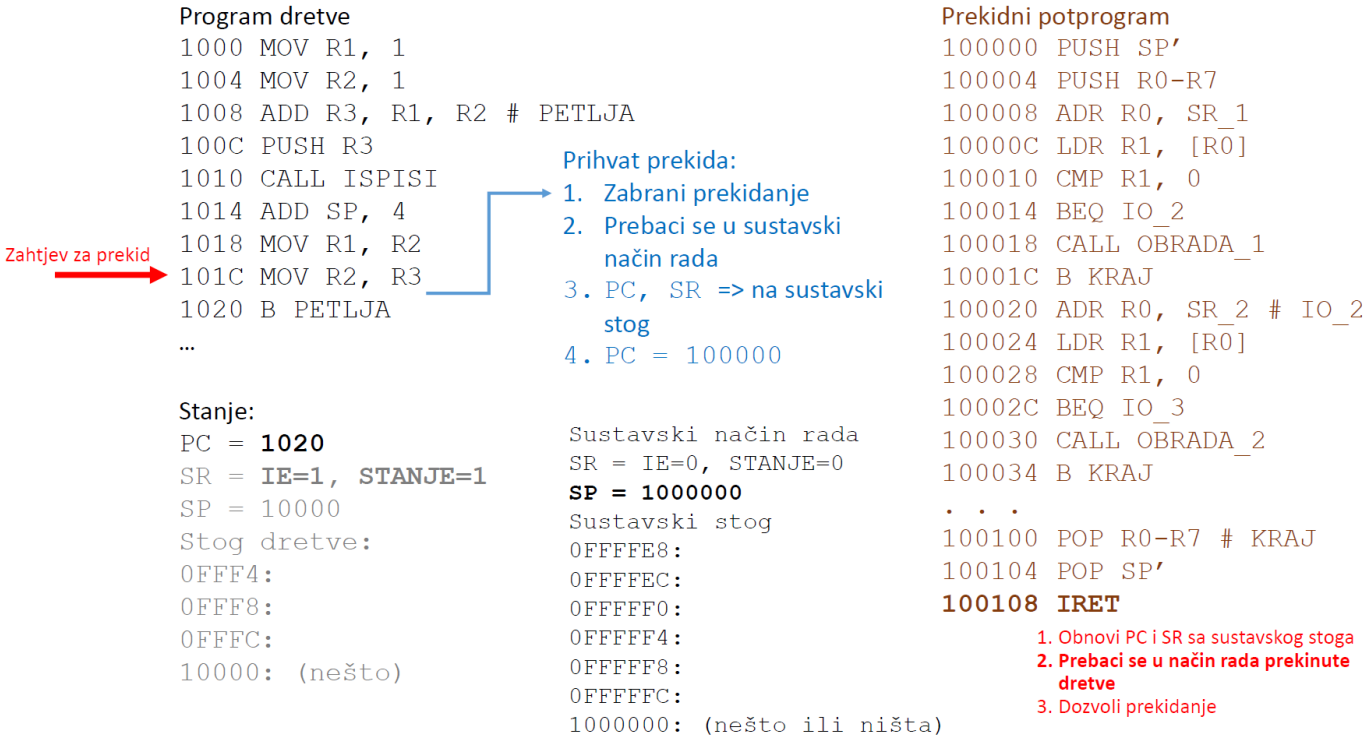
```
instrukcija: vrati_se_u_prekinutu_dretvu
{
   obnovi PC i SR sa sustavskog stoga
   prebaci (vrati) se u način rada prekinute dretve (definiran u SR)
   dozvoli prekidanje
}
```

Obnavljanje registra stanja vraća dretvu u prethodni način rada (korisnički, ako se vraćamo u dretvu ili sustavski ako se vraćamo u obradu nekog prekinutog prekida).

Obrada prekida naprave može uključivati razne operacije, od samog kopiranja podatka u radni spremnik do složenih operacija (npr. aktivaciju neke dretve).




**Slika 3.6. Utjecaj prekida na odgodu programa**




**Slika 3.7. Primjer prihvata prekida**

## Zadatak 3.1. Primjer obrade niza zahtjeva za prekid

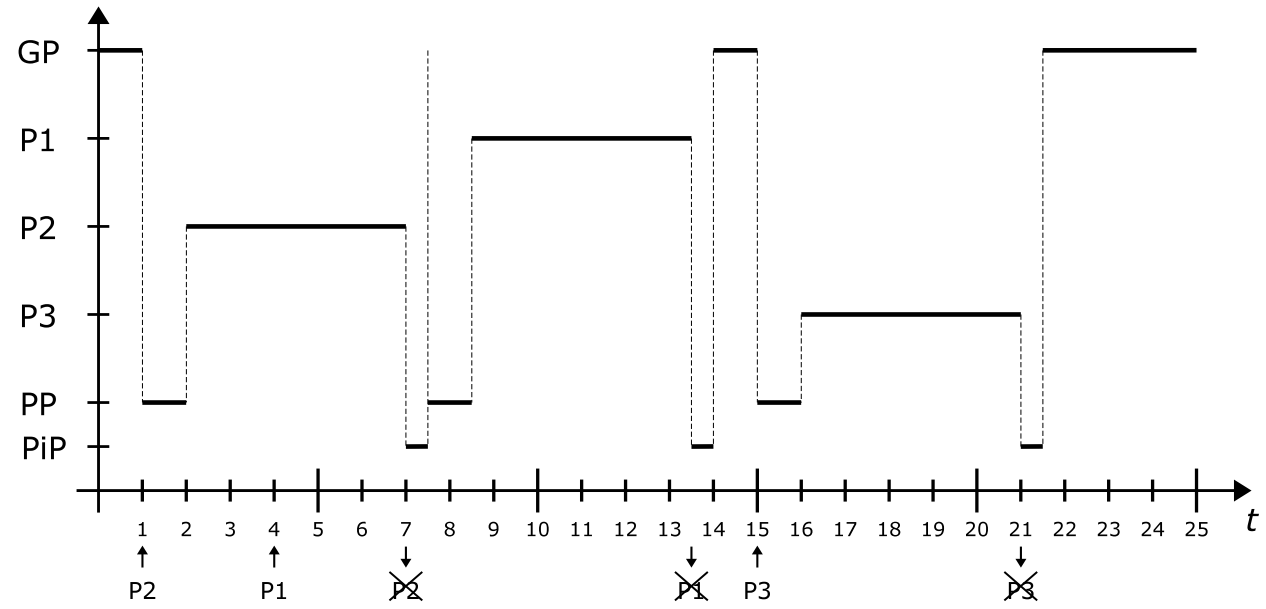
U nekom sustavu javljaju se prekidi: P1 u 4. ms, P2 u 1. ms te P3 u 15. ms. Obrada svakog prekida traje 5 ms (koristan dio obrade). Postupak prihvata prekida (PP, spremanje konteksta prekinute dretve, ispitni lanac) neka traje 1 ms. Povratak iz prekida (PIP, obnova konteksta i povratak u dretvu) neka traje 0,5 ms.

a) Grafički prikazati rad procesora u glavnom programu (GP), obradama P1, P2 i P3 te kućanskim poslovima PP i PIP.

b) Koliko se ukupno vremena potroši na kućanske poslove (PIP + PP)?

c) Koliko se odgađa obrada GP zbog ta tri prekida?

a)




b) 3 prekida puta (PP + PIP) $= 3 * 1,5 = 4,5$ ms

c) u intervalu $1 - 21,5$ GP je radio 1 ms; odgođen je $21,5 - 1 - 1 = 19,5$ ms

---

### Svojstva upravljanja napravama prekidom, bez sklopa za prihvat prekida

$+$ radi, jednostavno sklopovlje i programska potpora

$-$ potrebno je dodatno sklopovlje (procesor s potporom za prihvat prekida)

- problem: za vrijeme jedne obrade jednog prekida svi zahtjevi koji se tada pojave MORAJU pričekati kraj obrade prethodnog
- u nekim se sustavima (za rad u stvarnom vremenu) podrazumijeva da obrade prekida traju vrlo kratko te je kod njih i ovakav način prihvata i obrade prekida dovoljno dobar
- obrade prekida koji čekaju na obradu ne ide prema redoslijedu zahtjeva već prema redoslijedu provjere u ispitnom lancu u prekidnom potprogramu – na taj način ipak postoji neki "prioritet", ali ne takav koji omogućuje prekidanje obrade prekida manjih prioriteta zahtjevima veće prioriteta – više o tome u nastavku

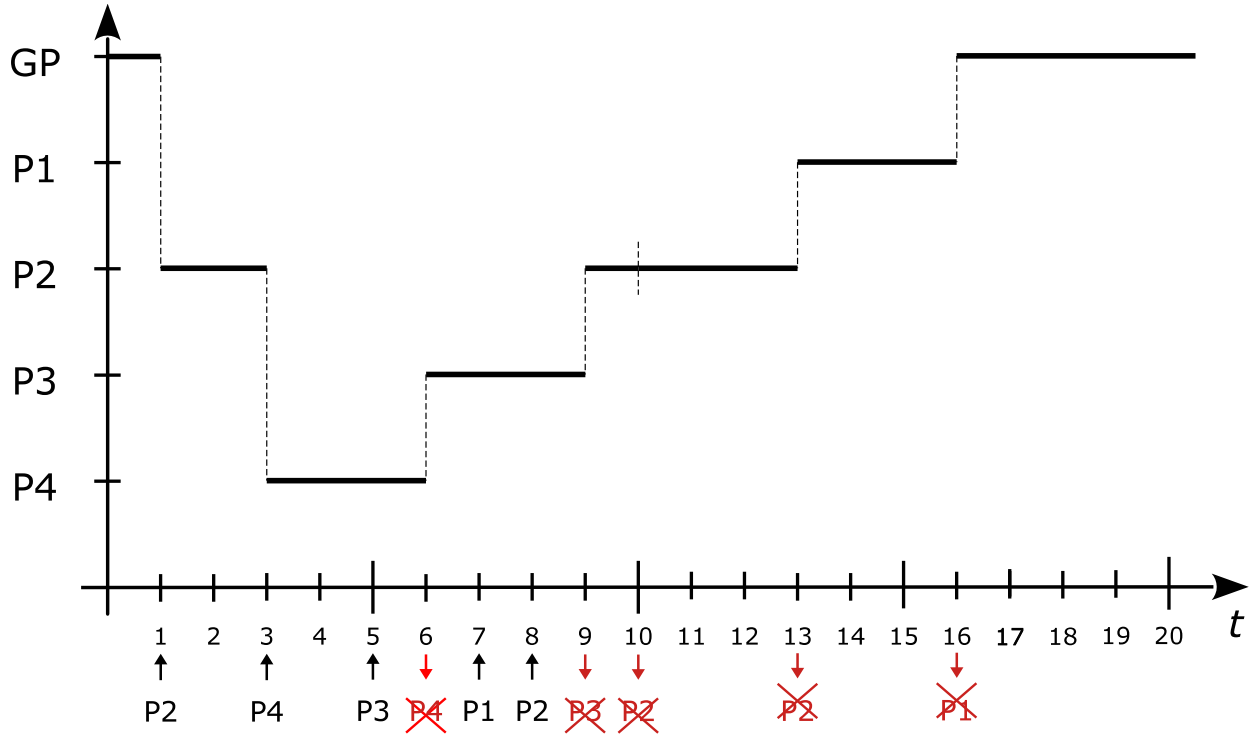
### 3.3.2. Poželjni (idealan) način prihvata prekida

**Obrada prema prioritetu**

- Ako se za vrijeme obrade jednog prekida pojavi novi zahtjev za prekida čija obrada bi trebala biti što prije gotova, taj će novi zahtjev morati čekati da prethodna završi.
- Zapravo bi željeli zahtjevima dodijeliti prioritet, pa da se obrađuju prema njemu, prvo oni zahtjevi većeg prioriteta.
- Kada zahtjev većeg prioriteta dođe, on treba privremeno prekinuti obradu zahtjeva manjeg prioriteta, da bi krenula obrada ovog zahtjeva, a po završetku onda nastaviti prethodno prekinutu obradu ili možda čak započeti nova koja je čekala jer je bila manjeg prioriteta od ove u obradi ali većeg od prekinute.
- Arhitekt sustava treba pridijeliti prioritete zahtjevima, prema UI napravama koje ih izazivaju. U stvarnosti se to ostvaruje sklopovima, odnosno, dovođenjem zahtjeva za prekid do jednog od ulaza sklopa za prihvat prekida te programiranjem takvog sklopa da taj ulaz ima takav prioritet.
- U pojednostavljenom prikazu svakoj napravi ćemo dodijeliti identifikacijski broj (redni broj) koji će ujedno označavati i prioritet. Naprava X ima prioritet X. Neka veći broj označava bitnije naprave, tj. “veći prioritet”
- obradu prema prioritetima prekida možemo ostvariti: *programski* ili *sklopovski*

**Kućanski poslovi**

- u postupku prihvata prekida obavljaju se razne: prihvat prekida od strane procesora te operacije u prekidnom potprogramu – sve su one neophodne u pojedinom načinu prihvata prekida
- ali samo "`obradi prekid naprave I`" radi koristan posao
- sve ostalo su “kućanski poslovi” te bi željeli da traju što kraće, tj. u idelnom scenariju kao da ih nema




**Slika 3.8. Primjer idealnog prihvata prekida**

### 3.3.3. Obrada prekida prema prioritetima, bez sklopa za prihvat prekida

Ideja programskog rješenja:

- sama obrada prekida (korisna/čista obrada) neka se obavlja s dozvoljenim prekidanjem
- na svaki zahtjev za prekid pozvati proceduru koja će ustanoviti prioritet zahtjeva i usporediti ga s trenutnim poslom:
  - ako je novi zahtjev prioritetniji odmah započinje njegova obrada (prekinuti posao se kasnije nastavlja)
  - u protivnom, novi zahtjev se samo zabilježi te se nastavlja s prekinutom obradom (novi zahtjev će doći na red kasnije)
- prekidima (napravama) treba dodijeliti prioritete
  - u nastavku je (radi jednostavnosti algoritma) prioritet određen brojem naprave, veći broj $\Rightarrow$ veći prioritet

Podatkovna struktura rješenja:

- `TEKUĆI_PRIORITET` – prioritet tekućeg posla, 0 kad se izvodi obična dretva ("glavni program"), broj `I` za obradu prekida naprave rednog broja `I` (njena prioriteta)
- `OZNAKA_ČEKANJA[]` – polje od `N` elemenata (`N` je najveći prioritet)
  - `OZNAKA_ČEKANJA[I]` označava da li naprava `I` čeka na početak obrade ili ne
- `KON[N]` – rezervirano mjesto u spremniku za pohranu konteksta dretve pri obradi pojedinog prekida; uz kontekst dretve sprema se i tekući prioritet

**Isječak kôda 3.4. Prihvat prekida prema prioritetima (bez sklopa)**

```
Prekidni potprogram  //prihvat prekida
{
   spremi kontekst na sustavski stog (osim PC i SR)
   ispitnim lancem utvrdi uzrok prekida, tj. indeks I
   OZNAKA_ČEKANJA[I] = 1
   signaliziraj napravi da je njen prekid registriran te da spusti prekidni signal

   dok je (I > TEKUĆI_PRIORITET) //u obradu prekida prioriteta "I" ?
      OZNAKA_ČEKANJA[I] = 0
      pohrani TEKUĆI_PRIORITET i kontekst sa sustavskog stoga u KON[I]
      TEKUĆI_PRIORITET = I

      omogući prekidanje
      obrada prekida naprave (I)
      zabrani prekidanje

      iz KON[I] obnovi TEKUĆI_PRIORITET, a kontekst stavi na sustavski stog

      I = max { J | za sve J za koje vrijedi: OZNAKA_ČEKANJA[J] != 0 }
      //od naprava koje čekaju na početak obrade, naprava I ima najveći prioritet
      //npr. I=0; za j=0 do N radi: ako je OZNAKA_ČEKANJA[J]!=0 onda I=J;
   }

   obnovi kontekst sa sustavskog stoga (osim PC i SR)
   vrati_se_u_prekinutu_dretvu
}
```

## Zadatak 3.2. Primjer obrade niza zahtjeva za prekid, uz prioritete

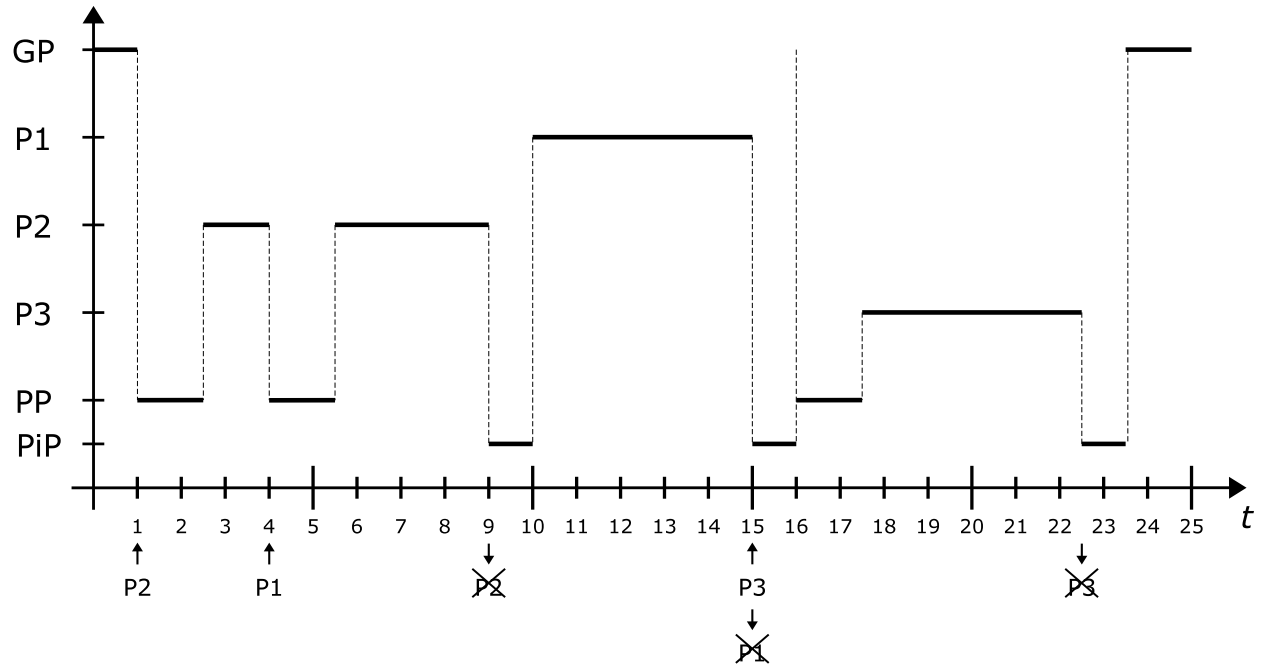
U nekom sustavu koji nema sklop za prihvat prekida, ali ima programski rješenu obradu prekida prema prioritetima, javljaju se prekidi: P1 u 4. ms, P2 u 1. ms te P3 u 15. ms. Prioriteti prekida zadani su brojem: P1 ima najmanji, a P3 najveći prioritet. Obrada svakog prekida traje 5 ms. Postupak prihvata prekida (PP) neka traje 1,5 ms. Povratak iz prekida (PIP) neka traje 1 ms.

a) Grafički prikazati rad procesora u glavnom programu (GP), obradama P1, P2 i P3 te kućanskim poslovima PP i PIP.

b) Koliko se ukupno vremena potroši na kućanske poslove?

c) Koliko se odgađa obrada GP zbog ta tri prekida?

a)




b) 3 prekida puta (PPP + PIP) $= 3 * (1,5 + 1) = 7,5$ ms

c) GP prekinut u $1.$ ms, nastavlja u $23,5 \Rightarrow 22,5$ ms je odgođen

---

**Svojstva prihvata i obrade prekida prema prioritetima, bez posebnog sklopa**

+ prekidi se obrađuju u skladu s prioritetima
+ nije potrebno posebno sklopovlje za to
− "mali nedostatak" – svaki prekid uzrokuje "kućanske poslove" tj. poziv prekidnog potprograma, čak i prekidi manjeg prioriteta
  – kako riješiti taj nedostatak? jedino korištenjem dodatnog sklopovlja

## Zadatak 3.3. (ispitni zadatak)

U nekom sustavu javljaju se zahtjevi za prekid: P1 u 3. ms, P2 u 1. ms te P3 u 4. ms. Prioritet prekida određen je brojem (P3 ima najveći prioritet). Obrada svakog prekida traje po 4 ms. Grafički prikazati aktivnosti procesora u glavnom programu (GP), procedurama za obradu prekida (Pi) te procedurama za prihvat prekida (PP) i povratak iz prekida (PiP) i to:

a) u idealnom slučaju *(prekidi se obrađuju prema prioritetu, trajanje kućanskih poslova se zanemaruje)*

b) bez sklopa za prihvat prekida *(u sustavu koji nema sklop za prihvat prekida u kojem se po prihvatu nekog prekida on obrađuje do kraja – nema ni programske ni sklopovske potpore za obradu prekida prema prioritetima)*, uz trajanje prihvata prekida (PP) od 1 ms *(uključuje potragu za izvorom prekida – zahtjevi većeg prioriteta se prvi prihvaćaju)* te trajenje povratka iz prekida (PiP) od 0,5 ms *(obnova konteksta prekinute dretve)*

c) bez sklopa ali s programskom potporom *(u sustavu koji nema sklop za prihvat prekida, ali se programski određuje prioritet prekida – obrada prekida se odvija s dozvoljenim prekidanjem)*, uz trajanje procedure za prihvat prekida i određivanje prioriteta prekida od 1,5 ms (PP), te 1 ms za povratak iz prekida (PiP) *(na dovršetku obrade prekida ponovno treba pogledati ima li još nešto za obraditi)*

U trenutku $t_x$ prikazati stanje korištene podatkovne strukture (za c)).

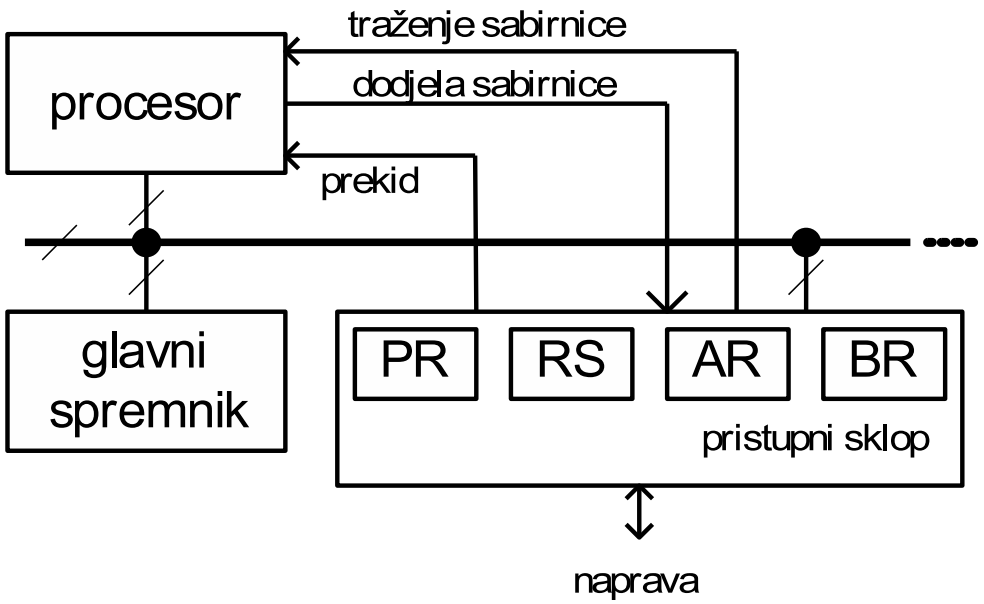
Dodatna pravila za rješavanje:

- ako u istom trenutku neka obrada završava i pojavljuje se novi zahtjev za prekid pretpostavljamo da je ipak najprije završila obrada, a potom se dogodio zahtjev za prekid
- ako u istom trenutku više naprava generira zahtjev za prekid najprije se prihvaća onaj najvećeg prioriteta – ostale i dalje imaju postavljen zahtjev za prekid (na prekidnom ulazu procesora ili sklopa za prihvat prekida)

---

## 3.4. Korištenje sklopova s izravnim pristupom spremniku

- pri korištenju prekida za upravljanje UI napravama dosta je neproduktivnog rada - kućanski poslovi spremanja/obnove konteksta, prozivanja naprava, ...
- koristan posao je često samo prijenos podatka u spremnik ili iz spremnika
- može li prenošenje u spremnik ili iz njega prema UI napravama biti učinkovitije?
- može, proširenjem sklopovlja dodatnim mogućnostima




**Slika 3.9. Pristupni sklop s izravnim pristupom spremniku**

Elementi i rad pristupnog sklopa s izravnim pristupom spremniku (*direct memory access – DMA*)

- PR, RS – podatkovni registar i registar stanja (kao i prije)
- AR – adresni registar – od kuda/kamo se učitavaju/pohranjuju podaci
- BR – brojilo podataka – koliko podataka još za prenijeti
- kad UI pristupni sklop ima novi podatak ili može preuzeti novi, on:
  - traži upravljanje sabirnicom za jedan ciklus
  - kad dobije upravljanje, prenosi podatak iz PR u spremnik ili obratno
- procesor prati zahtjeve tek kada on upravlja sabirnicom
- po zahtjevu, procesor prepušta **idući** sabirnički ciklus zahtjevu

Pristupni sklop po inicijalizaciji (nakon učitavanja AR i BR od strane procesora) radi sljedeće:

```
dok je (BR > 0) {
    čekaj na podatak VJ //ili na spremnost VJ za prihvat novog podatka
    zatraži sabirnicu i čekaj na dodjelu sabirnice //dvožično rukovanje!
    {AR, PR} na sabirnicu (+čitaj/piši signal)
    AR++
    BR--
}
postavi signal PREKID
```

Po jednom prijenosu procesor “izgubi” samo jedan ciklus na sabirnici

Ovakav sklop je pogodan kada treba prenijeti veću količinu podataka, inače je jednak običnome Kada se prenesu svi podaci sklop izaziva prekid te ga procesor (eventualno) opet programira.

### Primjer 3.3. Usporedba načina korištenja UI jedinice

a) neka procesor ima 10 MIPS-a te neka u sekundi prosječno treba prenijeti 1000 znakova

- radno čekanje => 100% procesorskog vremena, 10 korisnih instrukcija po prijenosu
  - 1000 (zn. u 1 s) * 10 (instr.) / 10 000 000 (instr./s) = 0,1% korisnog rada
  - (ili 10 / 10 000 = 0,1% korisnog rada)
- prekidi: ∼200 instr. po prekidu => 1000 * 200 / 10 000 000 = 2% => 98% ostane!
  - (ili 200 / 10 000 = 2%, 98% ostaje)
- DMA: 1000 / 10 000 000 = 0,01 % => 99,99 % ostane!
  - (ili 1 / 10 000 = 0.01%, 99.99% ostaje)
  - nakon prijenosa bloka znakova izaziva se prekid; npr. kada je blok 1000 znakova onda bi nakon 1000. znaka bio prekid i njegova obrada s 200 instrukcija bi neznatno povećala opterećenje s 0,01 na 0,012

b) isti procesor, maksimalna brzina prijenosa (jako brza UI)

- radno čekanje: 10 000 000 / 10 = 1000 000 znakova/s uz 100% opt. procesora
- prekidi: 10 000 000 / 200 = 50 000 znakova/s uz 100% opt. procesora
- DMA (svaki 2. sabirnički ciklus): 10 000 000 / 2 = 5 000 000 znakova/s uz 50% opt. procesora

## 3.5. Usporedba načina upravljanja UI napravama

1. Radno čekanje
   - u petlji se preko zastavica provjerava mogućnost slanja/primanja podataka

   \+ jednostavno sklopovlje (jeftino)

   − neefikasno korištenje procesora (radno čekanje)
2. Prekidi
   - ideja – biti efikasniji od radnog čekanja omogućujući procesoru da izvodi neki drugi posao u međuvremenu, ali da ipak brzo reagira na događaj
   - pri prihvatu prekida potrebno je spremiti kontekst prekinute dretve, a pri povratku obnoviti kontekst neke dretve (prekinute ili neke druge), tj. postoji cijena, dodatni poslovi (overhead) ovog pristupa ("kućanski poslovi")
   - nekoliko načina prihvata prekida

   2.1. bez prekidanja započete obrade (bez sklopa, bez prioriteta)
      - kad se ustanovi tko traži prekid (ispitnim lancem) krenuti u njegovu obradu i dok ona ne završi ne prihvaćati nove zahtjeve

      \+ ne treba sklop za prihvat prekida

      − ali bitni događaji (zahtjevi za prekid) mogu duže (predugo) čekati

   2.2. programski prihvat prekida prema prioritetu (bez sklopa)
      - uz dodatnu strukturu podataka pratiti što se događa i na novi zahtjev pravilno reagirati – samo zapisati novi zahtjev, ako je manjeg prioriteta od onog što procesor radi, ili odmah i započeti obradu

      \+ ne treba sklop a ipak se prekidi obrađuju prema prioritetu

      − malo više kućanskog posla
3. Izravan pristup spremniku
   - pristupni sklop sam prenosi podatke u/iz memorije (bez prekida, troši po jedan sabirnički ciklus za prijenos jednog podatka)
   - kod naprava koje šalju/primaju više podataka, a tek kad su svi poslani/primljeni ide neka akcija (tek tada pristupni sklop izaziva prekid); npr. disk, mreža

   \+ efikasniji sustavi (uz gornju pretpostavku)

   − složeniji pristupni sklop i procesor

   − nisu za jednostavne naprave (npr. za tipkovnicu, miša, ...)

## 3.6. Prekidi generirani unutar procesora, poziv jezgre

1. Pri radu, procesor može izazvati razne greške kao što su:
   - dijeljenje s nulom,
   - nepostojeća adresa (operanda, instrukcije),
   - nepostojeći operacijski kod,

- instrukcija se ne može izvesti s trenutnim ovlastima
- ...

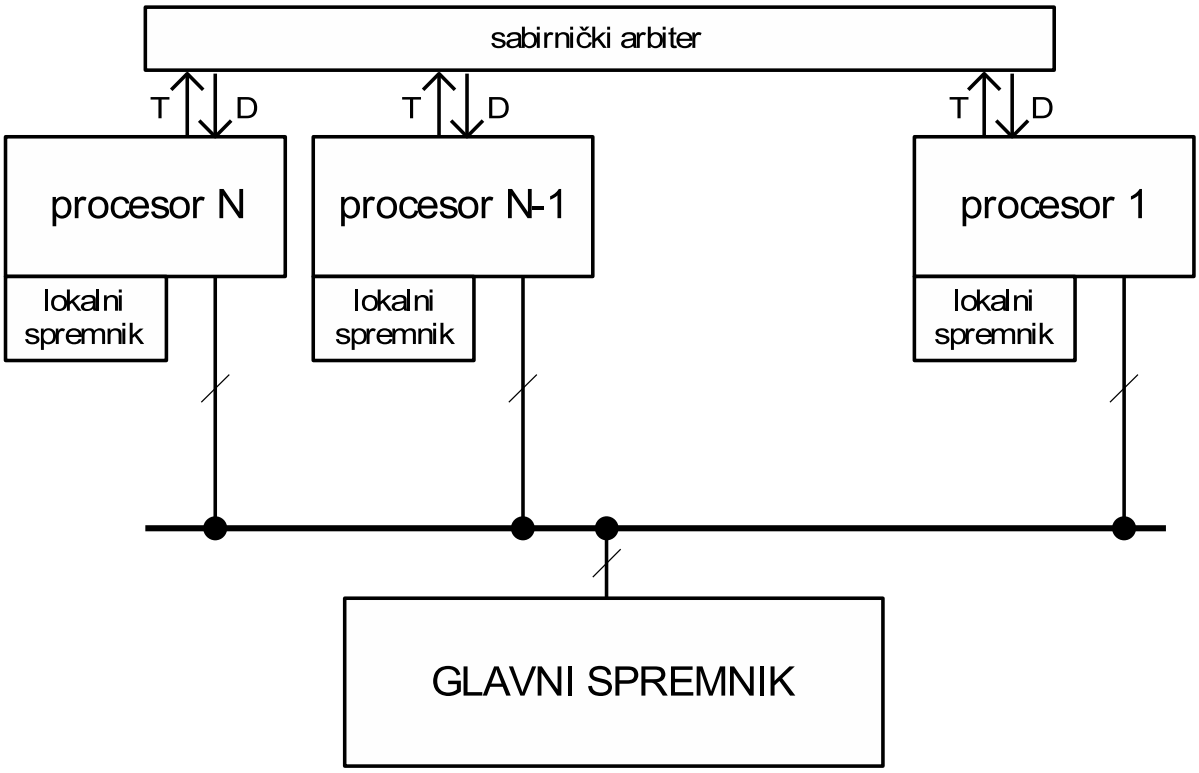
Dretvu koja je izazvala takvu grešku treba zaustaviti (ne može se oporaviti od takve greške). Mehanizam prekida je prikladan za takvu operaciju: procesor sam izaziva prekid, a u obradi operacijski sustav prekida dretvu i miče ju iz sustava.

2. Kako dretva obavlja operacije koje zahtijevaju više privilegije?

- ona namjerno izaziva prekid – *programski prekid* te se poziva *jezgrina funkcija*
- jezgrine funkcija = unaprijed pripremljena, dio sustava
- mehanizam zaštite je ugrađen preko programskog prekida
- ako dretva nešto želi što ne može/smije sama napraviti ona to traži od jezgre operacijskog sustava (o tome u 5. poglavlju)

## 3.7. Višeprocesorski (sabirnički povezani) sustavi

Ideja: proširiti DMA pristup, svaki DMA sklop zamijeniti procesorom, a "stari" procesor arbiterom (upravljačem) sabirnice




**Slika 3.10. Višeprocesorski sustav**

T – traženje sabirnice (BREQ – Bus Request)

D – dodjela sabirnice (BACK – Bus Acknowledge)

- sabirnicu dijele svi procesori
- lokalni (priručni) spremnik se koristi da se smanji potreba za sabirnicom
- ovo je programski model višeprocesorskog sustava (i prije i sada i vjerojatno u bližoj budućnosti)

# Pitanja za vježbu 3

1. Navesti načine upravljanja ulazno-izlaznim napravama u računalnom sustavu (programska izvedba). Vrlo kratko opisati svaki od načina.
2. Čemu služi pristupni sklop? Od kojih se elemenata minimalno sastoji? Što je to “dvožično rukovanje”?
3. Što su to i čemu služe "prekidi"?
4. Zašto su potrebni različiti načini rada procesora (korisnički i sustavni/prekidni/nadgledni)?
5. Kako procesor prihvaća prekide (postupak prihvata)?
6. Koji su sve izvori prekida? (izvan procesora, prekidi izazvani u procesoru-koji?)
7. Skicirati ostvarenje višeprocesorskog sustava. Kako se upravlja zajedničkom sabirnicom u takvom sustavu? Čemu služe priručni spremnici uz procesor?
8. U nekom sustavu javljaju se prekidi P1 u 5. i 9. ms, P2 u 2. ms te P3 u 4. i 11. ms. Prioritet prekida određen je brojem (P3 ima najveći prioritet). Obrada svakog prekida traje po 2 ms. Grafički prikazati aktivnosti procesora u glavnom programu (GP), procedurama za obradu prekida (Pi) te procedurama za prihvat prekida (PP) i povratak iz prekida (PiP) i to:
   a) u idealnom slučaju
   b) bez sklopa za prihvat prekida, obrada uz zabranjeno prekidanje, uz trajanje prihvata prekida (PP) od 1 ms te 0,5 ms za povratak iz prekida (PiP)
   c) bez sklopa ali s programskom potporom, uz trajanje prihvata prekida (PP) od 1,5 ms te 1 ms za povratak iz prekida (PiP)

   Odrediti stanje sustava i vrijednosti korištenih struktura podataka u t=8.5 ms.
9. U nekom sustavu javljaju se prekidi P1 u 0. ms, P3 u 4. ms, P2 se javlja u 6. ms. Prioritet prekida određen je brojem (P3 ima najveći prioritet). Obrada svakog prekida traje po 4 ms. Grafički prikazati aktivnosti procesora u glavnom programu (GP), procedurama za obradu prekida (Pi) te procedurama za prihvat prekida (PP) i povratak iz prekida (PiP) uz trajanje prihvata prekida od 0,5 ms (PP) te trajanje povratka iz prekida od 0,5 ms (PiP).
10. Usporediti svojstva sustava za upravljanje UI napravama korištenjem radnog čekanja, prekida te metode izravnog pristupa spremniku za sustav kod kojeg preko jedne UI naprave prosječno dolazi novi podatak svakih 0,1 ms, a sabirnica radi na 25 MHz.

# 4. MEĐUSOBNO ISKLJUČIVANJE U VIŠEDRETVENIM SUSTAVIMA

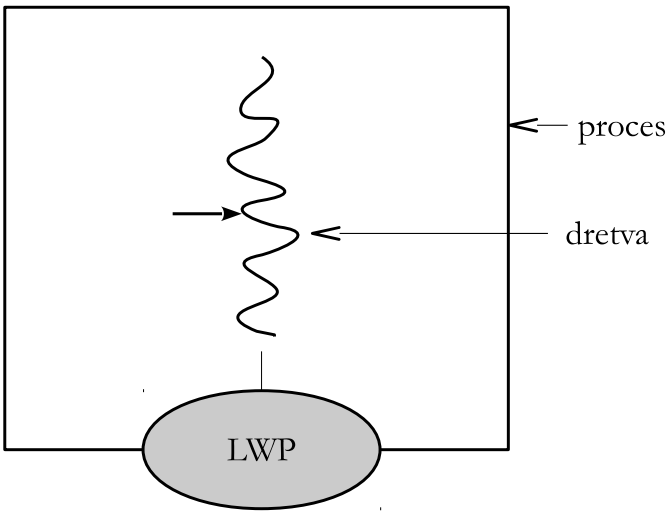
## 4.1. Osnovni pojmovi – program, proces, dretva

*Program* je statični niz instrukcija, nešto što je pohranjeno na papiru, disketi, memoriji

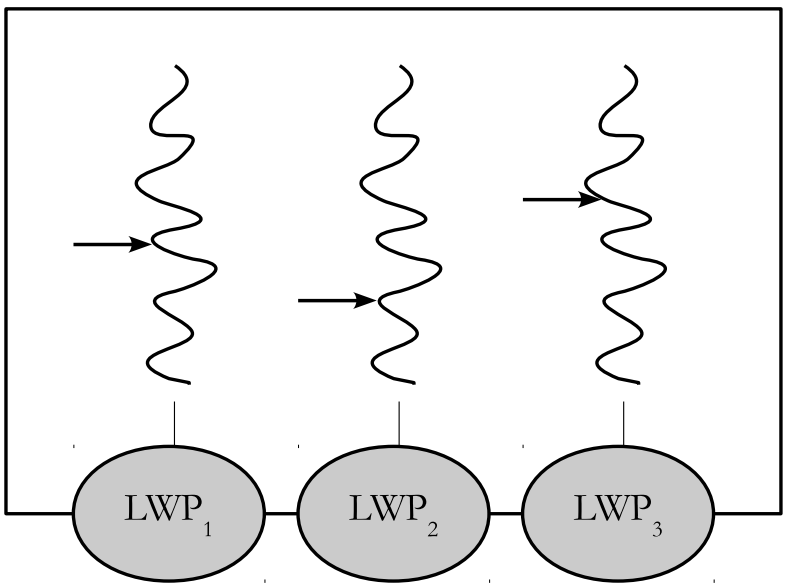
*Proces* je:

- skup računalnih resursa koji omogućuju izvođenje programa ili
- okolina u kojoj se program izvodi ili
- "sve što je potrebno" za izvođenje programa.

*Dretva* je niz instrukcija koji se izvodi. Proces se sastoji od *barem* jedne dretve




**Slika 4.1. Tradicionalni proces**



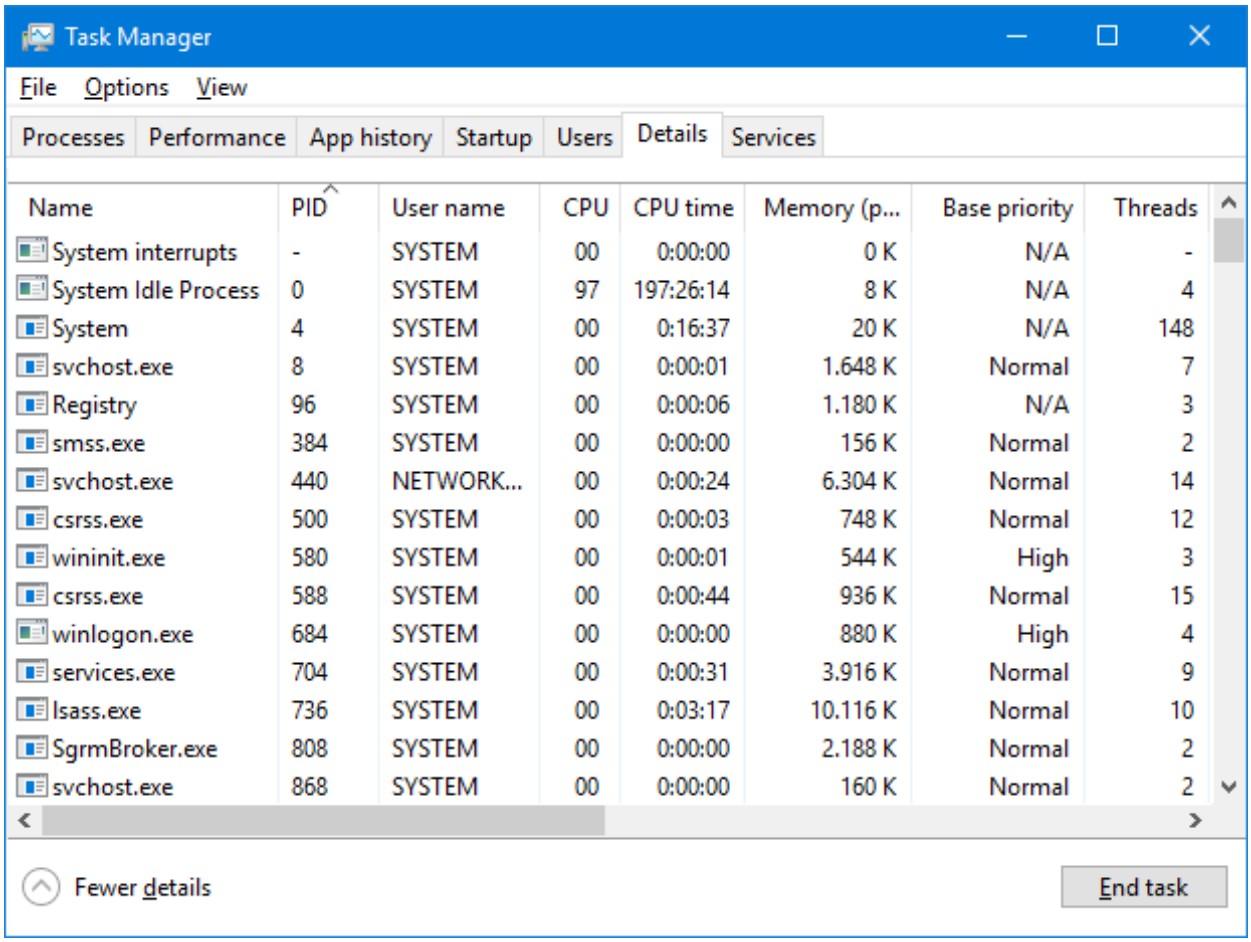

**Slika 4.2. Moderan proces**

LWP:

- laki (engl. *lightweight*) proces, virtualni procesor, ono što OS vidi kao dretvu procesa
- u većini slučajeva LWP je isto što i dretva procesa

Danas: proces se sastoji od *barem* jedne dretve

Uobičajeno je da OS vidi i upravlja svim dretvama. Međutim ima i iznimaka kada se nekim dretvama upravlja unutar procesa – OS ih ne vidi sve (npr. *fiber*).




**Slika 4.3. Primjer popisa dijela procesa s raznim parametrima i brojem dretvi u zadnjem stupcu**

Skup zauzetih sredstava je isti za sve dretve istog procesa.

- postoji zajednički spremnik (proces)
  - cijeli adresni prostor (proces) je “zajednički spremnik” (programski gledano, najčešće se to osjeti u korištenju globalnih varijabli koje su “globalne” za sve dretve)
- komunikacija među dretvama je znatno brža (koriste se globalne varijable)
- komunikacija među dretvama istog procesa može se odvijati i bez uplitanja OSa

## 4.2. Višedretveno ostvarenje zadatka – zadatak i podzadaci

Svaki se zadatak, barem i "umjetno" može podijeliti na podzadatke, ako to nije očito iz strukture zadatka (dijelovi, paralelna obrada, ...)

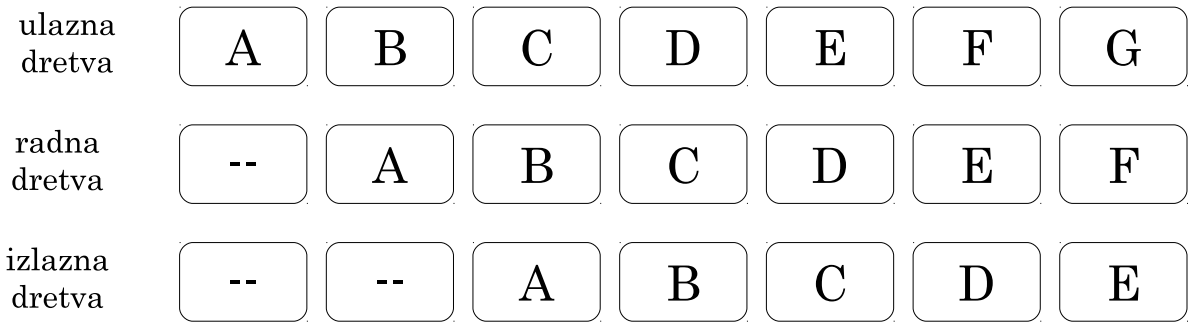
Jedna od mogućih podjela zadatka na podzadatke analogna protočnoj strukturi procesora je prikazana u nastavku.

Zadatak se može podijeliti na:

- podzadatak za čitanje ulaznih podataka – ulazna dretva
- podzadatak za obradu – radna dretva
- podzadatak za obavljanje izlaznih operacija – izlazna dretva

Navedena podjela može doprinijeti učinkovitosti sustava jer paralelno rade: ulazna jedinica, procesor i izlazna jedinica.

Korištenjem navedene podjele, zadatak se može izvoditi načelom cjevovodnog rada




**Slika 4.4. Načelo cjevovodnog rada dretvi**

Problem: razmjena podatka između dretvi kada dretvama treba različito vrijeme za odradu posla. Npr. radna dretva treba prije preuzimanja podataka od ulazne dretve pričekati da ulazna dovrši dohvat tih podataka. Isto tako ulazna dretva treba prije predaje podataka radnoj dretvi pričekati da radna dretva dovrši započetu obradu (rad na prethodno predanim podacima). I slično.

Dretve je potrebno uskladiti, tj. *sinkronizirati*!

Mnogi zadaci se mogu (smisleno) rastaviti na podzadatke, npr.: $Z_1 \rightarrow Z_2 \rightarrow \cdots \rightarrow Z_N$.

Ipak, i u takvim slučajevima potrebno je sinkronizirati zadatke – urediti slijed njihova izvođenja – tko prije a tko poslije.

Problemi sinkronizacije:

- Kako ostvariti mehanizme sinkronizacije?
- Koje je dretve potrebno sinkronizirati?

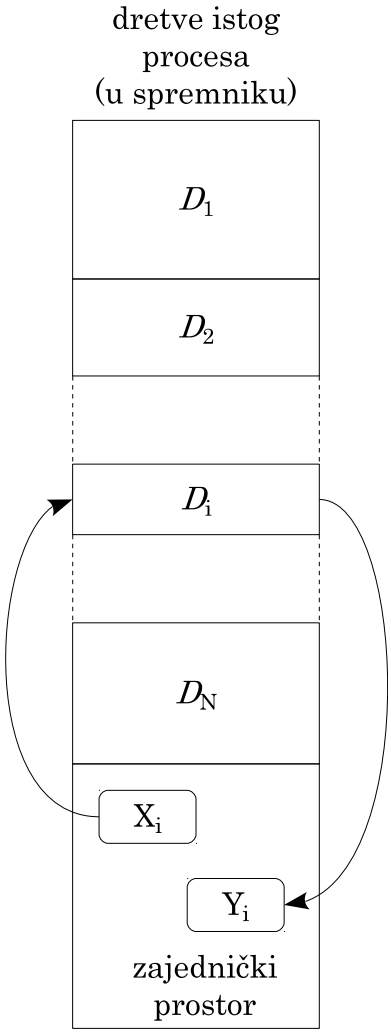
Postoje različiti oblici sinkronizacije. Za prethodne probleme potreban je mehanizam koji će jednu dretvu zaustaviti dok joj druga ne signalizira da može nastaviti. Jedan od sinkronizacijskih mehanizama za takvu sinkronizaciju jest semafor. Semafor se razmatra u idućem poglavlju, dok je ovdje prikazan jednostavniji sinkronizacijski mehanizam: *međusobno isključivanje*.

Svaki podzadatak se izvodi u svojoj dretvi. U nastavku se ponegdje umjesto podjele *zadatak⇒podzadaci* koristi *sustav zadataka⇒zadatak*, ali je načelo jednako.

## 4.3. Model višedretvenosti, nezavisnost dretvi

Pretpostavke:

- sve su dretve unutar istog procesa – dijele njegov spremnički prostor
- svaka dretva ima skup instrukcija, skup podataka te vlastiti stog
- postoji zajednički spremnički prostor koji dretve koriste pri rješavanju zadatka




**Slika 4.5. Dretva, domena i kodomena**

Svaki zadatak ima svoju:

- domenu $X_i$ (iz koje samo čita) te
- kodomenu $Y_i$ (koju mijenja)

tj. zadatak se može funkcijski opisati kao preslikavanje: $Y_i = f(X_i)$

**Kada se dva zadatka (dretve koje ih izvode) mogu izvoditi paralelno, a kada ne?**

Dva su podzadatka *nezavisna* ako nemaju nikakvih zajedničkih spremničkih lokacija ili imaju presjeka samo u domenama.

**Uvjet nezavisnosti podzadataka, odnosno dretvi $D_i$ i $D_j$:**

$$(X_i \cap Y_j) \cup (X_j \cap Y_i) \cup (Y_i \cap Y_j) = \phi \tag{4.1.}$$

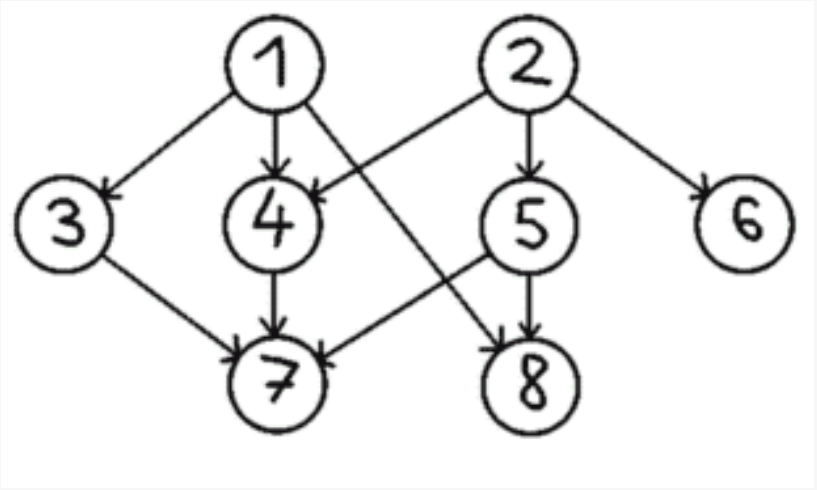
Ako su podzadaci *zavisni* tada se mora utvrditi redoslijed izvođenja njihovih dretvi.

Ako su podzadaci *nezavisni* tada se njihove dretve mogu se izvoditi *proizvoljnim* redoslijedom, pa i *paralelno*!

### Primjer 4.1. Sustav zadataka zadan u obliku usmjerena grafa

Ako se neki posao može rastaviti na lanac zadataka $Z_1 \rightarrow Z_2 \rightarrow \cdots \rightarrow Z_N$, tada se razmatranjem njihovih domena i kodomena mogu ustanoviti zavisni i nezavisni zadaci te se sustav može prikazati usmjerenim grafom koji to uzima u obzir.

Npr. neki sustav zadataka se možda može prikazati kao na slici:




**Slika 4.6. Primjer sustava zadataka**

Na prikazanom primjeru, zadatak 1 se mora obaviti prije zadataka: 3, 4, 8 (i 7 tranzitivno).

Za svaki zadatak se može napraviti usporedba ovisnosti sa svim ostalim zadacima.

Zadaci na istim putovima su zavisni – mora se poštivati redoslijed izvođenja.

Zadaci na različitim putovima su nezavisni – mogu se izvoditi paralelno

---

Često kada dretve koriste zajednička sredstva nije potrebno utvrđivati redoslijed njihova izvođenja već je dovoljno da se osigura da ta sredstva ne koriste u istom trenutku – da se promjene nad njima ne obavljaju paralelno. Npr. ako dvije dretve žele povećati istu varijablu za jedan dovoljno ih je spriječiti da to ne rade istovremeno, redoslijed nije bitan jer je u oba slučaja ispravan.

## 4.4. Problem paralelnog korištenja zajedničkih varijabli (engl. *race condition*)

Pristup zajedničkim sredstvima (npr. varijablama) treba zaštititi jer se paralelnim radom može doći do krivog rezultata.

**Primjer 4.2. Primjer problema s paralelnim radom dretvi**

Neka u sustavu postoji više dretvi koje koriste zajedničke strukture podataka. Jedna od tih struktura je varijabla `brojilo` koja označava broj poruka u međuspremniku. Ta se varijabla negdje povećava, a negdje smanjuje, kao u odsječku:

```
...
ako je (brojilo > 0) tada {
    uzmi poruku
    brojilo = brojilo - 1
}
...
```

Problem kod paralelnog korištenja varijable `brojilo` prema gornjem kodu jest da se od provjere vrijednosti (`ako je`) do akcije (kada je uvjet bio ispunjen) zbog stanja te vrijednosti u sustavu svašta može dogoditi.

Za gornji primjer se može dogoditi da dvije dretve paralelno provjere vrijednost brojila. U slučaju da je njegova vrijednost bila jedan, obje bi mogle ući u `ako je` dio koda te pokušati uzeti poruku. Kako brojilo broji poruke, samo je jedna poruka na raspolaganju te će jedna od dretvi (ona druga) napraviti grešku u dijelu `uzmi poruku` (ovisno o izvedbi reda, dohvatit će smeće ili staru poruku i pritom unijeti grešku u tu strukturu podataka).

Rješenje navedena problema ostvaruje se tako da se dio koda koji koristi zajedničke varijable zaštiti od istovremenog korištenja. Npr. prije `ako je` staviti ogradu koja će zabraniti ulazak više od jedne dretve u kod iza nje.

```
...
ulaz_u_kritični_dio_koda
ako je (brojilo > 0) tada {
    uzmi poruku
    brojilo = brojilo - 1
}
izlaz_iz_kritična_dijela_koda
...
```

Često nam se čini da je scenarij u kojem će se dogoditi ovakav problem vrlo malo vjerojatan – puno toga se mora poklopiti između različitih dretvi. Međutim, dretve mogu paralelno obavljati jako puno ponavljanja problematična koda (u petlji) te vjerojatnost pojave problematična scenarija značajno naraste – najčešće je vjerojatnije da će se on dogoditi nego da neće.

## 4.5. Međusobno isključivanje

Međusobno isključivanje (MI) je najjednostavniji mehanizam sinkronizacije.

Odsječke koda koji koriste zajednička sredstva nazivamo *kritičnim odsječcima* i njih treba zaštititi sinkronizacijskim mehanizmom međusobnog isključivanja.

Izvorni kod dretve se stoga može podijeliti na kritične odsječke i nekritične odsječke, primjerice:

```
dretva_x
{
    ...
    nekritični odsječak
    kritični odsječak
    nekritični odsječak
    kritični odsječak
    ...
}
```

Radi jednostavnosti u nastavku se razmatraju cikličke dretve koje imaju jedan kritičan odsječak

```
ciklička_dretva
{
    ponavljaj {
        kritični odsječak
        nekritični odsječak
    }
    do zauvijek
}
```

### Primjer 4.3. Poslužiteljska dretva

U nekom poslužitelju dretve koje poslužuju zahtjeve mogu se modelirati sljedećim kodom:

```
dretva_poslužitelja //jedna od "radnih dretvi"
{
    ponavljaj {
        uzmi_idući_zahtjev_iz_reda // kritični odsječak
        obradi_zahtjev_i_vrati_rezultat //nekritični odsječak
    }
    do kraja_rada_poslužitelja
}
```

**Kako ostvariti kritični odsječak?**

- koristiti mehanizme međusobnog isključivanja – funkcije `uđi_u_KO` i `izađi_iz_KO`

```
ciklička_dretva
{
    ponavljaj {
        uđi_u_KO()
        kritični odsječak
        izađi_iz_KO()
        nekritični odsječak
    }
    do zauvijek
}
```

Kako ostvariti te funkcije (`uđi_u_KO` i `izađi_iz_KO`)? Koja svojstva moraju one imati?

**Zahtjevi na algoritme međusobnog isključivanja (ispitno pitanje)**

1. U kritičnom odsječku u svakom trenutku smije biti najviše jedna dretva.
2. Mehanizam međusobnog isključivanja mora djelovati i u uvjetima kada su brzine izvođenja dretvi proizvoljne.
3. Kada neka od dretvi zastane u svom nekritičnom dijelu ona ne smije spriječiti ulazak druge dretve u svoj kritični odsječak.
4. Izbor jedne od dretvi koja smije ući u kritični odsječak treba obaviti u konačnom vremenu.

Algoritam mora vrijediti i za jednoprocesorske i za više procesorske sustave (tj. za sve sustave u kojima se želi primijeniti).

## 4.6. Potraga za algoritmima međusobnog isključivanja

ZAŠTO se “traže” kad se zna koji valjaju?
ZATO da se usput pokažu problemi višedretvenih sustava!

Opće pretpostavke:

- višeprocesorski sustav (barem 2 procesora)
- dretve su u istom procesu (dijele adresni prostor)

Neka se prvo "pronađu" algoritmi koji rade za dvije dretve. Ako algoritam ne radi za dvije dretve neće ni za više!

Koristit će se radno čekanje jer trenutno nije prikazano bolje rješenje (preko OS-a).

### 4.6.1. Prvi pokušaj (ZASTAVICA)

Koristi se zajednička varijabla `ZASTAVICA`

- kada je `ZASTAVICA == 0`, nitko nije u KO
- kada je `ZASTAVICA == 1`, jedna dretva je u KO te druga neće ući već će radno čekati da se ta varijabla promijeni

```
uđi_u_KO ()
{
    ponavljaj
        pročitaj varijablu ZASTAVICA
    sve dok je (ZASTAVICA == 1)

    ZASTAVICA = 1
}
```

```
izađi_iz_KO ()
{
    ZASTAVICA = 0
}
```

U kodu (ili pseduokodu) svako korištenje varijable podrazumijeva da se ona mora učitati iz spremnika. Stoga će u nastavku biti izostavljen dio `pročitaj varijablu` i svako pojavljivanje varijable će podrazumijevati da se ta varijabla prvo mora dohvatiti.

```
uđi_u_KO ()
{
    dok je (ZASTAVICA == 1)
        ;

    ZASTAVICA = 1
}
```

```
izađi_iz_KO ()
{
    ZASTAVICA = 0
}
```

U asembleru `uđi_u_KO`:

```
1 uđi_u_KO:
2     ADR R0, ZASTAVICA
3 petlja:
4     LDR R1, [R0] //pročitaj varijablu ZASTAVICA
5     CMP R1, 1    //ZASTAVICA == 1 ?
6     BEQ petlja
7     STR 1, [R0]  //ZASTAVICA = 1
8     RET
```

Problemi:

- ako dretve rade paralelno, u uzastopnim sabirničkim ciklusima mogu izvesti instrukciju s linije 4 – obje mogu pročitati 0 i obje ući u KO
- iako na prvi pogled izleda malo vjerojatno da će dretve ovaj kod izvoditi baš paralelno, treba uzeti u obzir da takve petlje procesor može izvesti milijune (i više) puta u sekundi – i vrlo mala vjerojatnost jednog događaja se ovako znatno poveća – stoga je često vjerojatnost da će se ovo dogoditi veća nego da se neće dogoditi
- nije ispunjen osnovni uvjet (1) (ostali jesu, ali moraju biti svi)

### 4.6.2. Lamportov algoritam međusobnog isključivanja

- drugo ime: *pekarski algoritam*
- svaka dretva prije ulaska u KO dobije svoj broj, koji je za 1 veći od najvećeg do sada dodijeljenog
- u KO ulazi dretva s najmanjim brojem!
- što ako dvije dretve dobiju isti broj? onda se gleda i indeks dretve
- postupak dobivanja broja je također neki oblik KO pa se i on ograđuje s `ULAZ[]`
- zajednički podaci:
  - `BROJ[]` – dodijeljeni brojevi (0 kada dretva ne traži ulaz u KO)
  - `ULAZ[]` – štiti se dodjela broja
  - početne vrijednosti varijabli su nule

```
uđi_u_KO (I)
{
    //uzimanje broja
    ULAZ[I] = 1
    BROJ[I] = max(BROJ[1], ..., BROJ[N]) + 1
    ULAZ[I] = 0

    //provjera i čekanje na dretve s manjim brojem
    za J=1 do N
    {
        dok je (ULAZ[J] == 1)
            ; //čeka se da dretva J dobije broj, ako je u postupku dobivanja

        dok je (BROJ[J] != 0 &&
                (BROJ[J] < BROJ[I] || (BROJ[J] == BROJ[I] && J < I)))
            ; //čekaj ako J ima prednost
    }
}

izađi_iz_KO (I)
{
    BROJ[I] = 0
}
```

Svojstva Lamportova algoritma

+ **radi** za proizvoljan broj dretvi na proizvoljnom broju procesora!
− radno čekanje (kao i Dekkerov i Petersonov)
− (manji nedostatak) potrebna poveća struktura podataka

## 4.7. Sklopovska potpora međusobnom isključivanju

**Zabrana prekidanja – samo za jednoprocesorske sustave**

Dretvu (i u KO) može prekinuti samo prekid (i u prekidu dretva može biti zamijenjena nekom drugom). Ako se prekid zabrani – ako dretva u svom izvođenju zabrani prekidanje, onda će ona raditi dok ta ista dretva ne dozvoli prekidanje.

U jednoprocesorskim sustavima bi mogli koristiti zabranu i dovolu prekidanja za ostvarenje kritična odsječka:

- `uđi_u_KO()` == `onemogući_prekidanje`
- `izađi_iz_KO()` == `omogući_prekidanje`

Problemi:

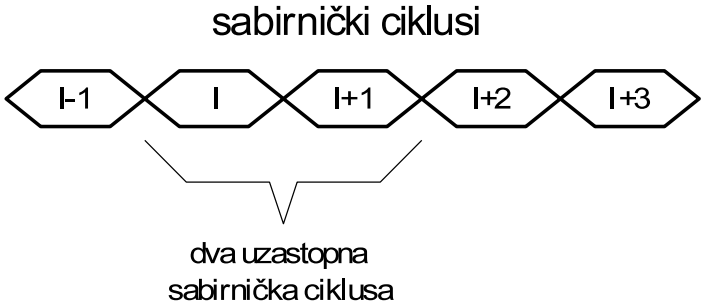
- radi samo u jednoprocesorskim sustavima
- traži privilegirani rad – zabrana i dozvola prekidanja su privilegirane operacije

**Drugi načini?**

Problem prvog pokušaja sa samo jednom zastavicom je bio u tome što se provjera zastavice i njeno postavljanje moglo prekinuti drugom dretvom (zajedničkim korištenjem sabirnice ili prekidom), koja je također, paralelno mogla provjeriti vrijednost zastavice.

Ako se to sklopovski onemogući, onda bi algoritam bio dobar! Kako?

**Korištenje dva uzastopna sabirnička ciklusa**




**Slika 4.7. Korištenje dva uzastopna sabirnička ciklusa**

U prvom ciklusu (I) se čita vrijednost zastavice, a u drugom (I+1) se zastavica postavlja u 1!

### 4.7.1. Ostvarenje MI s instrukcijom *Ispitaj_i_postavi – TAS* (*test-and-set*)

Neka postoji **instrukcija** `TAS adresa` koja koristi dva **uzastopna** sabirnička ciklusa tako da:

1. u prvom sabirničkom ciklusu pročita vrijednost sa zadane adrese i postavlja zastavice registra stanja (npr. zastavicu Z (zero), kao da uspoređuje s nulom)
2. u sljedećem sabirničkom ciklusu na tu adresu sprema vrijednost 1

Rješenje MI s TAS u asembleru:

```
uđi_u_KO:
petlja:
    TAS zastavica
    BNE petlja //dok nije 0 ponovi
    RET
```

```
izađi_iz_KO:
    ADR R1, zastavica
    STR 0, [R1]
    RET
```

U pseudokodu neka `TAS(adresa)` označava izvođenje TAS instrukcije, ali neka i vraća pročitanu vrijednost (radi jednostavnosti i kraćeg zapisa)

```
uđi_u_KO ()
{
    dok je (TAS (zastavica) == 1)
        ;
}
```

```
izađi_iz_KO ()
{
    zastavica = 0
}
```

## 4.8. Problemi prikazanih mehanizama međusobnog isključivanja

Osnovni problem svih prikazanih algoritama, sa i bez sklopovske potpore je radno čekanje – neefikasno korištenje procesora

Dodatno:

- Dekkerov i Petersonov algoritam rade samo za dvije dretve (Lamportov te ostali sa sklopovskom potporom rade za proizvoljan broj dretvi)
- manji problem algoritama ostvarenih sklopovskom potporom je nepoštivanje redoslijeda zahtijeva za ulaz u KO: prva dretva koja naleti u svom radnom čekanju na spuštenu zastavicu ulazi u KO – to može biti i zadnja dretva koja je došla do petlje, a ne ona koja je najduže čekala!

Da bi se riješili ovi problemi mehanizme sinkronizacije treba drukčije riješiti – korištenjem jezgrinih funkcija, gdje će se u kontroliranom okruženju dretva pustiti u KO ili neće, kada će se dretva maknuti s procesora (da ne troši procesorsko vrijeme na neproduktivnu petlju).

### Pitanja za vježbu 4

1. Kada su dva zadatka međusobno zavisna, a kada nezavisna? Što sa zavisnim zadacima, kako izvoditi njihove dretve?
2. Što je to *kritični odsječak* i *međusobno isključivanje*?
3. Kako se međusobno isključivanje može ostvariti u jednoprocesorskim a kako u višeprocesorskim sustavima?
4. Koji problem može nastati ako više dretvi obavlja operaciju: $A = A + 1$ nad zajedničkom varijablom $A$?
5. Navesti zahtjeve (4) na algoritme međusobnog isključivanja.
6. Čemu služi Lamportov algoritam? Opisati njegov rad.
7. Kako iskoristiti sklopovsku potporu međusobnom isključivanju korištenjem instrukcija TAS, SWP i sličnih?
8. Koji su problemi različitih načina ostvarenja međusobnog isključivanja?

# 5. JEZGRA OPERACIJSKOG SUSTAVA

Što je jezgra OS-a?

- osnovni, najbitniji dijelovi bez kojih OS ne bi radio
- OS ima i druge dijelove (pomoćne programe, usluge) koji koriste jezgru

Potreba za jezgrom?

- Upravljanje dretvama – ostvarivanje višedretvenosti
- Upravljanje UI napravama
  - treba biti kontrolirano – ne prepušteno dretvama
  - prekidi su dobar mehanizam za prelazak u "kontrolirani način rada"
  - operacije koje se izvode u prekidu su posebne => dio jezgre OSa!
  - programski prekid: dretva preko njega poziva te "posebne funkcije" – jezgrine funkcije
- Sinkronizacija
  - nedostaci sinkronizacijskih mehanizama prikazanih u 4. poglavlju: radno čekanje, nepoštivanje redoslijeda ulaska u KO
  - treba izbijeći radno čekanje, ali i poštovati redoslijed zahtjeva
  - ipak, treba uzeti u obzir da ništa "nije besplatno", sve ima svoju cijenu (složenosti, dodatne kućanske poslove, strukture podataka, ...)

Jezgrine funkcije je potrebno pozivati mehanizmom *prekida* jer oni omogućuju:

- kritični odsječak na jednoprocesorskim sustavima (o proširenju za višeprocesorske kasnije)
- privilegirani način rada potreban za
  - korištenje zaštićene strukture podataka u sustavskom dijelu spremnika
  - upravljanje UI napravama
- upravljanje dretvama (manipulaciju opisnicima dretvi)

**Jezgra OS-a** se sastoji od:

- strukture podataka jezgre (opisnici, liste, međuspremnici, ...)
- jezgrinih funkcija

Tipovi prekida koji se koriste za pozive jezgrinih funkcija:

- sklopovski prekid (zahtjev UI naprava i satnog mehanizma)
- programski prekid (zahtjev iz programa)

**Pretpostavke idućih razmatranja za ostvarenje jezgre**:

- jednoprocesorski sustav (kasnije je dano i proširenje za višeprocesorske sustave)
- korisnički i jezgrin način rada podržani od strane procesora
- sve je u spremniku – neće se (za sada) razmatrati učitavanje i pokretanje programa
- jezgrine funkcije pozivaju se sklopovskim i programskim prekidom

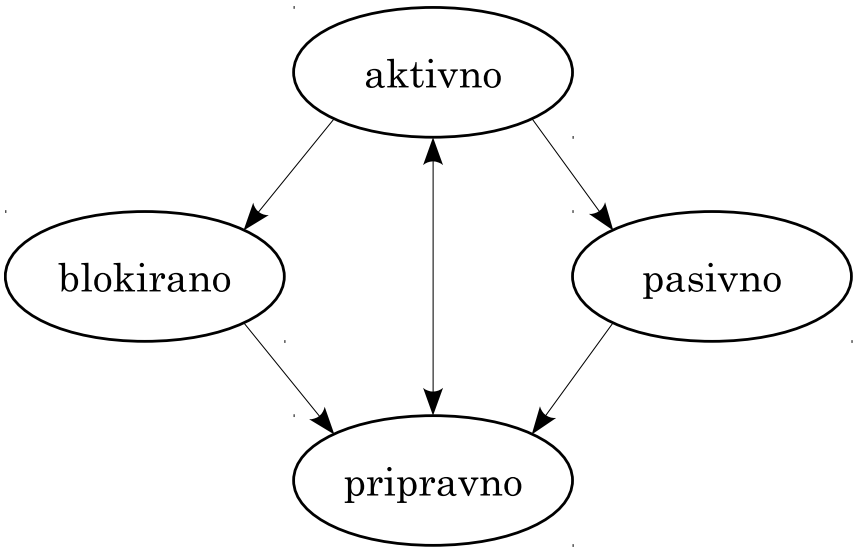
- postoji satni mehanizam koji periodički izaziva sklopovski prekid sata

Za sada (do 8. poglavlja) se neće razmatrati procesi, neka su sve dretve sustava u jednom zajedničkom procesu.

Pozivi jezgre ilustrirani su slikom 5.1.


**Slika 5.1. Mehanizam poziva jezginih funkcija**

Dretve mogu biti u različitim stanjima prema slici 5.2.




**Slika 5.2. Moguća stanja dretve**

## 5.1. Strukture podataka jezgre

Struktura podataka jezgre se sastoji od sljedećih elemenata:

- *opisnici dretvi*:
  - id (identifikacijski broj dretve)
  - podaci za raspoređivanje: način raspoređivanja, prioritet, ...
  - stanje dretve (aktivno, pripravno, ...)
  - opis spremničkog prostora dretve (gdje su instrukcije, podaci, stog)
  - `zadano_kašnjenje` – za ostvarivanje kašnjenja
  - kontekst – mjesto za pohranu konteksta dretve
  - kazaljka za liste prema stanjima (`Aktivna_D`, `Pripravne_D`, ...)
  - kazaljka za listu `Postojeće_D`
- *opisnici UI naprava* (info)
  - kazaljke na funkcije upravljačkog programa
  - međuspremnici i druge strukture podataka
  - lista za dretve koje čekaju na dovršetak svoje operacije nad napravom
- *liste stanja dretvi* – liste za opisnike dretvi
  - `Aktivna_D` – dretva koja se trenutno izvodi na procesoru
  - `Pripravne_D` – dretve koje se mogu izvoditi (aktivna se bira među njima)
  - blokirane dretve:
    - `UI[]` – liste dretvi koje čekaju na neku napravu
    - `Odgođene_D` – dretve koje su tražile odgodu
    - `BSEM[]` – binarni semafori
    - `OSEM[]` – opći semafori
  - `Postojeće_D` – lista u kojoj se nalaze sve dretve
    - ako se dretva nalazi samo u ovoj listi, dretva je *pasivna*

Liste dretvi mogu biti uređene (složene):

- prema redu prispijeća u listu
- prema prioritetu dretvi
- prema posebnim kriterijima (npr. vremenu odgode)

Za svaku listu je potrebno:

- kazaljka na prvu dretvu u listi (npr. `prva`)
- opcionalno/poželjno: kazaljka na zadnju dretvu u listi (npr. `zadnja`) – složenost umetanja na kraj je tada O(1)!

Jezgra upravlja sustavom – izvodi dretvu za dretvom

- kada se trenutno aktivna dretva blokira, uzima se prva dretva iz reda pripravnih
- u obradama prekida se neke blokirane dretve mogu odblokirati i staviti među pripravne

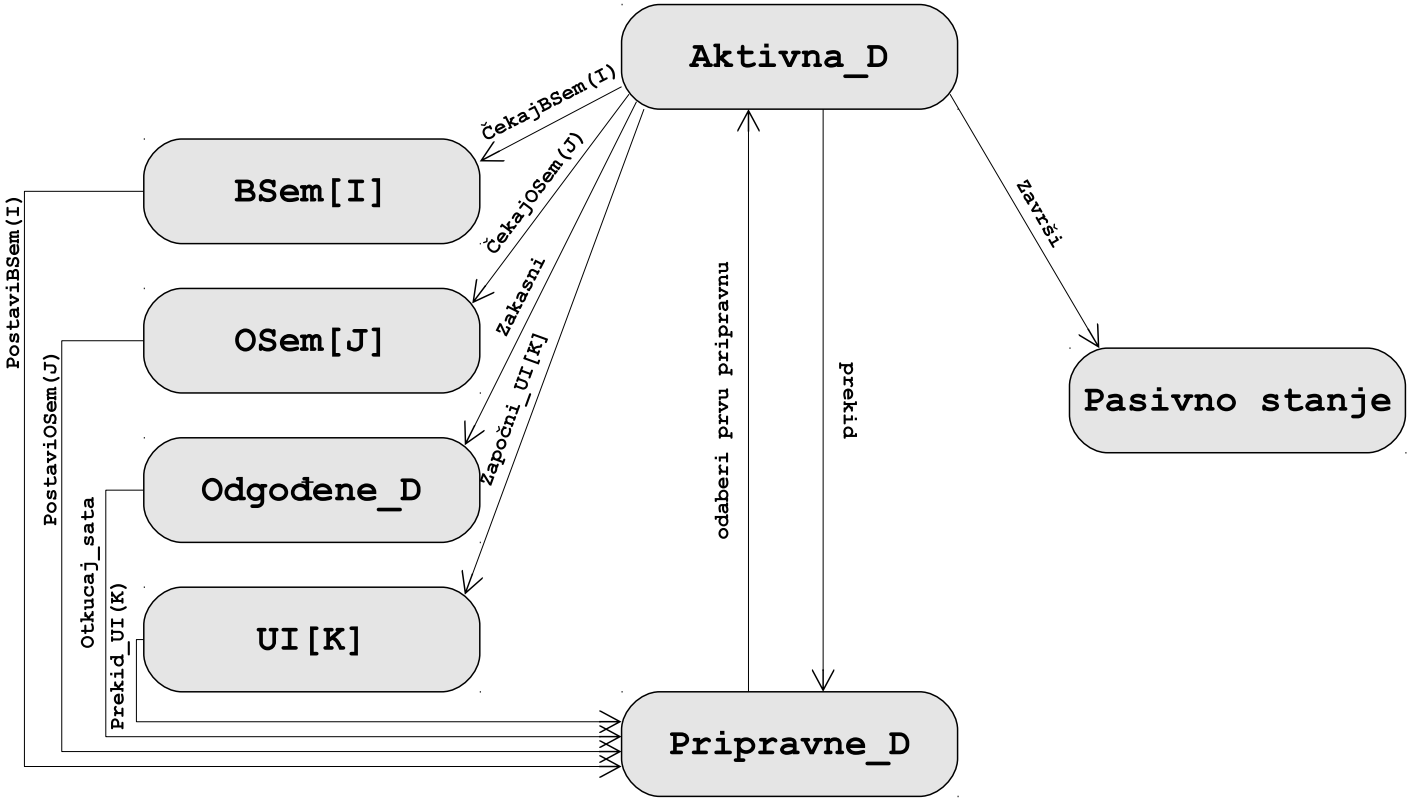
Latentna dretva (engl. *idle thread*)

- Dodatna, pomoćna dretva koja se izvodi kad nema niti jedne druge dretve u sustavu.
- Njen je zadatak dati procesoru nešto da radi (on mora nešto raditi).
- Takva dretva ima najmanji prioritet, tj. izvodi se jedino kada nema niti jedne druge dretve!
- U nastavku se podrazumijeva da takva dretva postoji, ali se ne spominje niti prikazuje.

## 5.2. Jezgrine funkcije

### 5.2.1. Moguća stanja dretvi u jednostavnom modelu sustava (ispitno pitanje)

Slika 5.3. prikazuje jednostavni model jezgre prikazan u ovom poglavlju sa svim mogućim stanjima i jezgrinim funkcijama koje uzrokuju promjene stanja.




**Slika 5.3. Moguća stanja dretve, jezgrine funkcije**

Pasivno stanje je pseudo stanje – nema zasebne liste za to. Dretva je u tom stanju ako nije ni u jednoj drugoj listi, tj. nalazi se samo u listi `Postojeće_D` (pomoćnoj listi gdje se nalaze sve dretve).

Osim j-funkcija `Otkucaj_sata` i `Prekid_UI` (koje se pozivaju sklopovskim prekidima) sve ostale poziva aktivna dretva (`Aktivna_D`) !

### 5.2.2. Razlike u ovdje prikazanom ostvarenju jezgre prema prikazanome u knjizi

Neke operacije su ovdje ostvarene na malo drukčiji način nego u knjizi. Međutim, oba pristupa pri rješavanju zadataka su ispravna u ispitima iako mogu ponekad dati različita rješenja. U nastavku su detaljnije prikazane razlike.

Iskazivanje operacija nad strukturama jezgre (opisnicima, listama)

- u knjizi – tekstom: `Premjesti opisnik iz reda Aktivna_D u red Bsem[I]`
- ovdje – funkcijama: `stavi_u_red(makni_prvu_iz_reda(Aktivna_D), BSEM[I])`

*U knjizi – jezgrine funkcije u kojima se propuštaju/odblokiraju dretve*

- u svim jezgrinim funkcijama koje potencijalno mogu odblokirati neku dretvu:
  1. najprije se aktivna prebacuje u red pripravnih
  2. potom se gleda stanje sustava i možda odblokira neka dretva – prebaci u red pripravnih
  3. na kraju se prva iz reda pripravnih uzme za aktivnu
- ovisno o načinu raspoređivanja (red prispijeća, prioritet), aktivna dretva nakon jezgrine funkcije može biti:
  - dretva koja je pozvala jezgrinu funkciju (i prije bila aktivna)
  - odblokirana dretva
  - dretva koja je bila prva u redu pripravnih prije poziva "ove" jezgrine funkcije

*Ovdje – jezgrine funkcije u kojima se propuštaju/odblokiraju dretve*

- ovdje se malo "optimiralo" rad takvih jezgrinih funkcija te se promjena aktivne događa samo po potrebi, tj. *raspoređivač* se poziva samo ako se neka dretva blokirala ili odblokirala
- ako se u obavljanju jezgrine funkcije aktivna blokira ili neka druga dretva odblokira sa:
  - `stavi_u_red(makni_prvu_iz_reda(Aktivna_D), neki_red)`
  - `stavi_u_red(makni_prvu_iz_reda(neki_red), Pripravne_D)`
- tada prije kraja jezgrine funkcije mora se pozvati:
  - `odaberi_aktivnu_dretvu()`
- operacija `odaberi_aktivnu_dretvu()` mogla bi se opisati sljedećim kodom:

```
odaberi_aktivnu_dretvu() //schedule()
{
    ako je (Aktivna_D.stanje != AKTIVNO) { //dretva je maknuta u neki drugi red
        stavi_u_red(makni_prvu_iz_reda(Pripravne_D), Aktivna_D)
    }
    inače ako je (Raspoređivanje == PREMA_PRIORITETU) {
        prva_pripravna = dohvati_prvu_iz_reda(Pripravne_D)
        ako je (Aktivna_D.prioritet < prva_pripravna.prioritet) {
            stavi_u_red(makni_prvu_iz_reda(Aktivna_D), Pripravne_D)
            stavi_u_red(makni_prvu_iz_reda(Pripravne_D), Aktivna_D)
        }
    }
}
```

- pretpostavlja se da funkcija `stavi_u_red` koristi zadano uređenje za dotični red; npr. za red pripravnih to može biti prema prispijeću ili prioritetu

*Problematične funkcije*

- funkcije koje mogu generirati različita stanja zbog navedenih promjena su `PREKID_UI()`, `OTKUCAJ_SATA()`, `POSTAVI_BSEM()` i `POSTAVI_OSEM()`

*Razlike u stanju nakon jezgrinih funkcija*

- kada je isto stanje nakon funkcija?
  - ako se dretve raspoređuju prema prioritetu i sve dretve imaju različit prioritet
    - nakon jezgrine funkcije aktivna je ona s najvećim prioritetom, a red pripravnih složen prema prioritetu
  - ili je red pripravnih bio prazan i a) raspoređivanje je po redu prispijeća, ili b) odblokirana dretva ima isti ili manji prioritet od aktivne
    - u knjizi je tada aktivna najprije stavljena u red pripravnih (prva i jedina u tom redu) potom je odblokirana stavljena u red pripravnih iza nje te konačno se uzima prva iz reda pripravnih – ona koja je i prije bila aktivna
- kada stanje nakon funkcija ne mora biti isto?
  - red pripravnih nije bio prazan prije poziva jezgrine funkcije i
    - dretve se raspoređuju po redu prispijeća, ili
    - dretve se raspoređuju prema prioritetu i prva u redu pripravnih ima isti prioritet kao i aktivna a odblokirana nema veći prioritet
  - razlika u ponašanju:
    - u knjizi: nakon poziva aktivna će biti ona koja je prije poziva bila prva u redu pripravnih
    - ovdje: aktivna će ostati ona koja je pozvala jezgrinu funkciju, a odblokirana dretva će biti zadnja u redu pripravnih (iza onih s istim prioritetom, ispred onih s manjim)
  - primjer: neka su svi redovi složeni po redu prispijeća
    - početno stanje: {Aktivna=$D_5$, Pripravne=$\{D_3, D_6, D_2\}$, red BSEM[S]=$\{D_4, D_1\}$};
    - aktivna poziva `PostaviBSEM(S)` (koja oslobađa prvu iz reda semafora)
    - stanje nakon prema knjizi:

      {Aktivna=$D_3$, Pripravne=$\{D_6, D_2, D_5, D_4\}$, red BSEM[S]=$\{D_1\}$};
    - stanje nakon prema ovdje prikazanom ostvarenju:

      {Aktivna=$D_5$, Pripravne=$\{D_3, D_6, D_2, D_4\}$, red BSEM[S]=$\{D_1\}$};
- VAŽNO: oba pristupa pri rješavanju zadataka (prikazana ovdje i prikazana u knjizi) su ispravna u ispitima, iako mogu dati različita rješenja!

### 5.2.3. Obavljanje ulazno-izlaznih operacija

Pretpostavke jednostavnog modela jezgre:

- UI operacije se obavljaju radi zahtjeva dretvi – one pokreću operaciju (preko jezgre)
- dovršetak UI operacije naprava javlja mehanizmom prekida
- UI naprave se koriste na "sinkroni način" – dretva čeka kraj operacije prije nastavka rada
- zahtjeve same jezgre prema UI napravama u ovom prikazu zanemarujemo

UI operacije obavljaju se preko jezgrinih funkcija:

- `ZAPOČNI_UI(K, parametri)` – poziva ju dretva
  - UI operacija traje (nije trenutna) pa će dretva čekati na njen kraj
- `PREKID_UI(K)` – poziva se na prekid naprave K
  - naprava K obavlja operacije redom kojim su joj zadane
  - kad je gotova s operacijom ona izaziva prekid

```
j-funkcija ZAPOČNI_UI(K, parametri)
{
    pohrani kontekst u opisnik Aktivna_D

    stavi_u_red(makni_prvu_iz_reda(Aktivna_D), UI[K])
    pokreni UI operaciju na napravi K
    odaberi_aktivnu_dretvu() //schedule()

    obnovi kontekst iz opisnika Aktivna_D
}
```

```
j-funkcija PREKID_UI(K)
{
    pohrani kontekst u opisnik Aktivna_D

    dovrši UI operaciju na napravi K
    stavi_u_red(makni_prvu_iz_reda(UI[K]), Pripravne_D)
    odaberi_aktivnu_dretvu()

    obnovi kontekst iz opisnika Aktivna_D
}
```

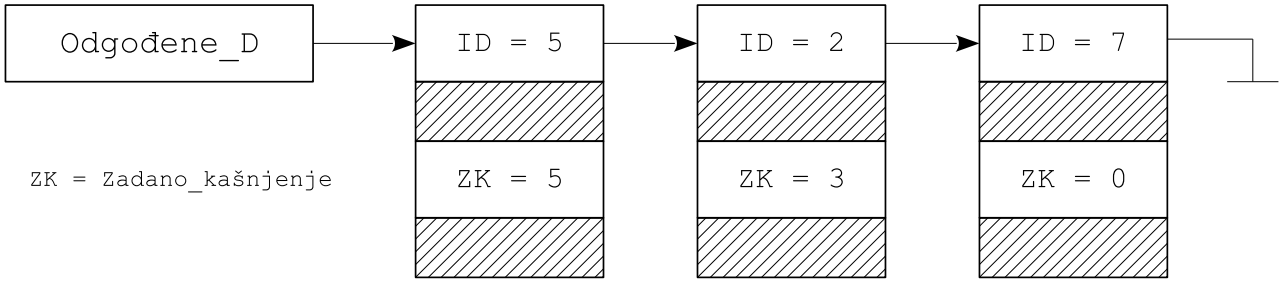
(info) Druga mogućnost ostvarenja UI operacija jest asinkroni rad: dretva ne čeka na dovršetak UI operacije već naknadno provjerava status (OS ažurira status u prekidima naprave)

### 5.2.4. Ostvarivanje kašnjenja

Za ostvarenje kašnjenja koristi se prekid sata koji u pravilnim intervalima izaziva prekid.

Postoji jedan red (lista opisnika) za odgođene dretve: `Odgođene_D`

- lista je složena prema vremenima odgode dretvi
- prva dretva u listi ima broj otkucaja sata koje još mora čekati
- svaka iduća dretva ima relativan broj u odnosu na prethodnu
- koristi se element `Zadano_kašnjenje` u opisniku dretve




**Slika 5.4. Primjer liste `Odgođene_dretve`**

Opis primjera sa slike 5.4.

- dretva D5 treba čekati još 5 otkucaja sata
- dretva D2 treba čekati da se prethodna dretva makne iz reda (5 otkucaja) te još 3 otkucaja nakon toga (ukupno još 8)
- dretva D7 treba čekati da se D5 i D2 maknu, pa se i ona oslobađa iz reda (jer treba čekati isto kao i D2)

Jazgrine funkcije za odgodu su: `ZAKASNI(M)` koju poziva dretve te `OTKUCAJ_SATA()` koja se poziva na prekid satnog mehanizma (brojila).

```
j-funkcija ZAKASNI(M)
{
  pohrani kontekst u opisnik Aktivna_D

  stavi_u_red_odgođenih(makni_prvu_iz_reda(Aktivna_D), M, Odgođene_D)
  odaberi_aktivnu_dretvu()

  obnovi kontekst iz opisnika Aktivna_D
}
```

```
j-funkcija OTKUCAJ_SATA()
{
  pohrani kontekst u opisnik Aktivna_D

  ako (Odgođene_D->prva != prazno) {
    Odgođene_D->prva.Zadano_kašnjenje--
    ako je (Odgođene_D->prva.Zadano_kašnjenje == 0) {
      dok je (Odgođene_D->prva != prazno && Odgođene_D->prva.Zadano_kašnjenje == 0)
        stavi_u_red(makni_prvu_iz_reda(Odgođene_D), Pripravne_D)

      odaberi_aktivnu_dretvu()
    }
  }
  obnovi kontekst iz opisnika Aktivna_D
}
```

## 5.3. Semafori

Semafori služe za sinkronizaciju dretvi.

Za razliku od semafora na raskršću koji mijenjaju stanje protokom vremena, sinkronizacijski semafori se mijenjaju preko poziva jezgrinih funkcija – koje pozivaju dretve.

Postoji nekoliko inačica semafora koji se koriste za razne potrebe. U nastavku je prikazan opći semafor koji najbolje odražava semafore koje pružaju operacijski sustavi.

### 5.3.1. Opći semafor

Opći semafor ima više stanja: jedno neprolazno i više prolaznih.

Zato je prikladan za brojanje događaja i sredstava te slične sinkronizacije.

Struktura za semafor:

- `.v` – vrijednost semafora
  - kada je `.v == 0` semafor je *neprolazan*
  - kada je `.v > 0` semafor je *prolazan*
- `.red` – blokirane dretve nad semaforom (lista opisnika tih dretvi)

Opći semafor se može ostvariti na razne načine. Mi ćemo koristiti ostvarenje koje je nabliže onome koji se koristi u stvarnim sustavima. Označimo takav semafor s OSEM.

```
j-funkcija ČEKAJ_OSEM(S)
{
    pohrani kontekst u opisnik Aktivna_D

    ako je (OSEM[S].v > 0) {
        OSEM[S].v--
    }
    inače {
        stavi_u_red(makni_prvu_iz_reda(Aktivna_D), OSEM[S])
        odaberi_aktivnu_dretvu()
    }

    obnovi kontekst iz opisnika Aktivna_D
}

j-funkcija POSTAVI_OSEM(S)
{
    pohrani kontekst u opisnik Aktivna_D

    ako je (red OSEM[S] nije prazan) {
        stavi_u_red(makni_prvu_iz_reda(OSEM[S]), Pripravne_D)
        odaberi_aktivnu_dretvu()
    }
    inače {
        OSEM[S].v++
    }

    obnovi kontekst iz opisnika Aktivna_D
}
```

Vrijednost `OSEM[I].v` nikako ne može postati negativna.

Vrijednost `OSEM[I].v` (kao i vrijednost `.v` ostalih semafora) može se mijenjati isključivo preko

poziva jezginih funkcija. U programu dretve se `OSEM[I].v` ne može izravno koristiti (nije dohvatljiv izravno)!

### Primjer 5.1. Primjer ostvarenja kritičnog odsječka semaforom

```
dretva () {
    ponavljaj {
        Čekaj_OSEM(1)
        kritični odsječak
        Postavi_OSEM(1)
        nekritični odsječak
    }
    do zauvijek
}
```

Početna vrijednost semafora `OSEM[1]` (prije pokretanja ovakvih dretvi) treba biti 1.

### Primjer 5.2. Utjecaj jezgrinih funkcija na stanje dretvi

Pokažimo kako će se mijenjati stanje sustava pozivom jezgrinih funkcija u određenim trenucima (zadano ispod). Programske prekide uvijek izaziva trenutno aktivna dretva.

Pozivi jezgrinih funkcija:

1. u stanju 0: `Prekid_UI(1)`
2. u stanju 1: `Započni_UI(2)`
3. u stanju 2: `Otkucaj_sata`
4. u stanju 3: `Zakasni(3)`
5. u stanju 4: `Otkucaj_sata`
6. u stanju 5: `PostaviOSEM(1)`
7. u stanju 6: `ČekajOSEM(2)`

**Tablica 5.1. Stanja sustava kad se dretve raspoređuju prema prioritetu – samo je red pripravnih dretvi složen prema prioritetu, dok su svi ostali po redu prispijeća (osim reda odgođenih)**

| Red / stanje | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| `Aktivna_D` | 8 | 8 | 7 | 7 | 6 | 6 | 6 | 5 |
| `Pripravne_D` | 7 5 | 7 6 5 | 6 5 | 6 5 | 5 | 5 2 | 5 4 2 | 4 2 |
| `OSEM[1]` | 4 11 | 4 11 | 4 11 | 4 11 | 4 11 | 4 11 | 11 | 11 |
| `OSEM[2]` | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 6 |
| `UI[1]` | 6 | – | – | – | – | – | – | – |
| `UI[2]` | 3 12 | 3 12 | 3 12 8 | 3 12 8 | 3 12 8 | 3 12 8 | 3 12 8 | 3 12 8 |
| `Odgođene_D` | $2^2$ $9^5$ | $2^2$ $9^5$ | $2^2$ $9^5$ | $2^1$ $9^5$ | $2^1$ $7^2$ $9^3$ | $7^2$ $9^3$ | $7^2$ $9^3$ | $7^2$ $9^3$ |

## 5.4. Izvedba jezgrinih funkcija za višeprocesorske sustave

- Zabrana prekida nije dovoljna za višeprocesorske sustave kao mehanizam međusobnog isključivanja jezgrinih funkcija, jer je zabrana lokalna za pojedini procesor
- Kako ostvariti MI u takvim sustavima?
  - mora se dodati radno čekanje
  - neka postoji instrukcija `TAS` kako je prikazano u 4. poglavlju
  - neka se za svaki procesor definira
    * lista aktivnih dretvi: `Aktivna_D[N]`
    * lista pripravnih dretvi: `Pripravne_D[N]`
  - neka se definira varijabla `OGRADA_JEZGRE` (zastavica za radno čekanje)

### 5.4.1. Opći semafor

```
j-funkcija ČEKAJ_OSEM(S)
{
  //neka je M indeks procesora, npr. iz opisnika dreve ili nekog registra procesora
  pohrani kontekst u opisnik Aktivna_D[M]
  //Aktivna_D[M] su lokalni podaci, nije ih potrebno štititi

  dok je (TAS (OGRADA_JEZGRE) == 1)
    ; // radno čekanje

  ako je (OSEM[S].v > 0) {
    OSEM[S].v--
  }
  inače {
    stavi_u_red(makni_prvu_iz_reda(Aktivna_D[M]), OSEM[S])
    odaberi_aktivnu_dretvu_na_procesoru(M)
  }
  OGRADA_JEZGRE = 0

  obnovi kontekst iz opisnika Aktivna_D[M]
}
j-funkcija POSTAVI_OSEM(S)
{
  //M indeks procesora
  pohrani kontekst u opisnik Aktivna_D[M]

  dok je (TAS (OGRADA_JEZGRE) == 1)
    ; // radno čekanje

  ako je (red OSEM[S] nije prazan) {
    stavi_u_red(makni_prvu_iz_reda(OSEM[I]), Pripravne_D[L])
    //L - iz opisnika dretve
    odaberi_aktivnu_dretvu_na_procesoru(M)
  }
  inače {
    OSEM[S].v++
  }
  OGRADA_JEZGRE = 0

  obnovi kontekst iz opisnika Aktivna_D[M]
}
```

Koristi se *radno čekanje*:

- ali samo do ulaska u jezgrinu funkciju (onaj bitan dio)
- s obzirom na to da su jezgrine funkcije kratke to nije dugo čekanje

## Pitanja za vježbu 5

1. Što je to *jezgra operacijskog sustava*?
2. Kako se *ulazi* u jezgru?
3. Od čega se jezgra sastoji?
4. U kojim se stanjima može naći dretva?
5. Navesti strukturu podataka jezgre za *jednostavni model jezgre* prikazan na predavanjima.
6. Što su to jezgrine funkcije? Navesti nekoliko njih (rad s UI jedinicama, semafori).
7. Što je to semafor? Koja struktura podataka jezgre je potrebna za njegovo ostvarenje? Skicirati funkciju `Čekaj_OSEM(I)/Postavi_OSEM(I)`.
8. Navesti primjer korištenja jezgrinih funkcija za ostvarenje međusobnog isključivanja (kritičnog odsječka).
9. Kako treba proširiti jezgrine funkcije za primjenu u višeprocesorskim sustavima?
10. Neki sustav se u promatranom trenutku sastoji od 5 dretvi u stanjima: D1 je aktivna, D2 u redu pripravnih, D3 u redu semafor OSEM[1], D4 i D5 u redu odgođenih (D4 treba čekati još jedan kvant vremena, a D5 ukupno još 5). Grafički prikazati stanje sustava (liste s opisnicima). Prikazati stanje sustava:
    a) nakon što dretva D1 pozove `Postavi_OSEM(1)`
    b) nakon što se obradi prekid sata.

# 6. MEĐUDRETVENA KOMUNIKACIJA I KONCEPCIJA MONITORA

U ovom se poglavlju prikazuju mnogi primjeri sinkronizacije dretvi korištenjem semafora i monitora.

Pri osmišljavanju sinkronizacije potrebno je voditi računa o nekoliko aspekata i mogućih problema.

### Što znači "ispravno sinkronizirati dretve/procese/zadatke"?

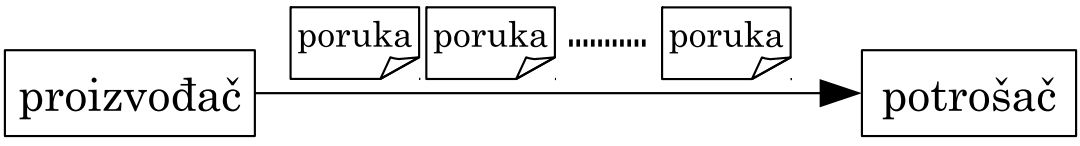
- Redoslijed izvođenja mora biti istovjetan opisu u tekstu zadatka.
- Međusobno isključivo treba koristiti sredstava za koje se to zahtijeva ili je iz zadatka očito da tako treba.
- Nema mogućnosti potpunog zastoja.
- Nema mogućnosti radnog čekanja.
- Navesti početne vrijednosti svih varijabli i semafora!

U nastavku je najprije prikazan primjer komunikacije između dretve proizvođača i dretve potrošača. Potom su prikazani problemi koji mogu nastati korištenjem semafora te kako se neki od njih mogu izbjeći korištenjem monitora.

## 6.1. Primjer sinkronizacije semaforima: proizvođač i potrošač

**Zadatak: Ostvariti komunikaciju između jednog proizvođača i jednog potrošača**

Spremnik veličine jedne poruke, kakav smo već susretali kod ulazne, radne i izlazne dretve poprilično ograničava rad dretvi. U mnogim slučajevima u kojima dretve komuniciraju, poslovi koje takve dretve rade u prosjeku jednako traju. Međutim, za pojedinu poruku će ponekad jednoj strani trebati više vremena, a ponekad drugoj. Stoga ima smisla dodati međuspremnike većeg kapaciteta od jedne poruke koji će ublažiti razlike u obradi pojedinačnih poruka i omogućiti veću dinamičnost u izvođenju dretvi. U nastavku je stoga krenuto od pretpostavke da u međuspremnik stane više poruka.




**Slika 6.1. Proizvođač i potrošač**

### 6.1.1. Načelni pseudokod rješenja

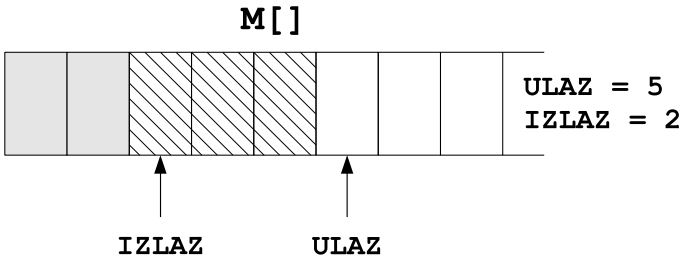
```
proizvođač
{
    ponavljaj {
        P = stvori poruku ()
        pošalji poruku(P)
    }
    do zauvijek
}
```

```
potrošač
{
    ponavljaj {
        čekaj na poruku
        R = uzmi poruku()
        obradi poruku (R)
    }
    do zauvijek
}
```

Kako ostvariti `pošalji poruku(P)`, `čekaj na poruku` i `uzmi poruku()`?

Potrebno je koristiti neke sinkronizacijske mehanizme i zajednički spremnik. Prikažimo to na primjerima u nastavku.

### 6.1.2. Korištenje neograničenog međuspremnika i radno čekanje




**Slika 6.2. Neograničeni međuspremnik**

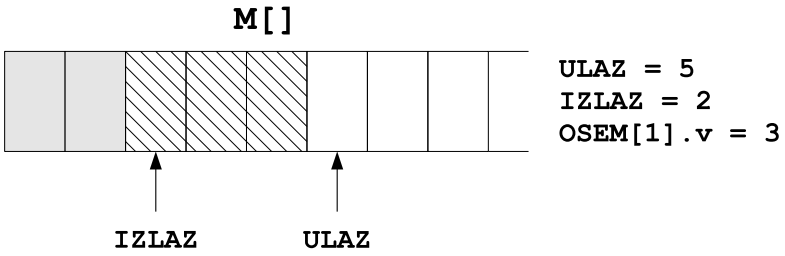
- `M[]` – međuspremnik neograničenje veličine
- `ULAZ` – indeks prvog praznog mjesta u međuspremniku, početno 0
- `IZLAZ` – indeks prvog nepročitanog mjesta u međuspremniku, početno 0

```
proizvođač
{
    ponavljaj {
        P = stvori poruku()
        M[ULAZ] = P
        ULAZ++
    }
    do zauvijek
}
```

```
potrošač
{
    ponavljaj {
        dok je (IZLAZ >= ULAZ) \\radno
            ;                  \\čekanje
        R = M[IZLAZ]
        IZLAZ++
        obradi poruku(R)
    }
    do zauvijek
}
```

Nedostaci: radno čekanje, neograničeni međuspremnik

### 6.1.3. Korištenje neograničenog međuspremnika i jednog semafora




**Slika 6.3. Neograničeni međuspremnik sa semaforima**

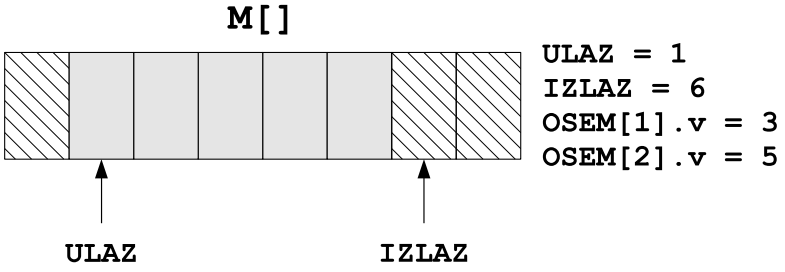
- `OSEM[1]` – broji poruke u međuspremniku, početno 0

```
proizvođač
{
    ponavljaj {
        P = stvori poruku()
        M[ULAZ] = P
        ULAZ++
        Postavi_OSEM(1)
    }
    do zauvijek
}
```

```
potrošač
{
    ponavljaj {
        Čekaj_OSEM(1)
        R = M[IZLAZ]
        IZLAZ++
        obradi poruku(R)
    }
    do zauvijek
}
```

Nedostatak: neograničeni međuspremnik

### 6.1.4. Korištenje ograničenog međuspremnika i dva semafora




**Slika 6.4. Ograničeni međuspremnik**

- `M[N]` – međuspremnik s `N` mjesta za poruke
- `ULAZ` i `IZLAZ` se povećavaju po modulu `N`
- `OSEM[1]` – broji poruke u međuspremniku, početno 0
- `OSEM[2]` – broji prazna mjesta u međuspremniku, početno `N` (veličina međuspremnika)

```
proizvođač
{
    ponavljaj {
        P = stvori poruku()
        Čekaj_OSEM(2)
        M[ULAZ] = P
        ULAZ = (ULAZ + 1) MOD N
        Postavi_OSEM(1)
    }
    do zauvijek
}
```

```
potrošač
{
    ponavljaj {
        Čekaj_OSEM(1)
        R = M[IZLAZ]
        IZLAZ = (IZLAZ + 1) MOD N
        Postavi_OSEM(2)
        obradi poruku(R)
    }
    do zauvijek
}
```

Varijablu ULAZ koristi samo proizvođač, dok varijablu IZLAZ samo potrošač. Međutim, obje dretve koriste međuspremnik M. Ipak, vrijednosti u varijablama ULAZ i IZLAZ te semafori OSEM[1] i OSEM[2] će spriječiti da obje dretve istovremeno pokušaju koristiti isti element međuspremnika.

Kada bi imali više proizvođača ili više potrošača, onda bi trebali dodati binarne semafore za zaštitu od istovremenog korištenja tih varijabli i istih dijelova međuspremnika.

### 6.1.5. Više proizvođača i jedan potrošač

Potrebno je dodati jedan binarni semafor za proizvođače da se spriječi istovremeni pokušaj stavljanja u međuspremnik na isto mjesto te promjenu varijable ULAZ.

```
proizvođač
{
    ponavljaj {
        P = stvori poruku()
        Čekaj_OSEM(2)
        Čekaj_BSEM(1)
        M[ULAZ] = P
        ULAZ = (ULAZ + 1) MOD N
        Postavi_BSEM(1)
        Postavi_OSEM(1)
    }
    do zauvijek
}
```

```
potrošač
{
    ponavljaj {
        Čekaj_OSEM(1)
        P = M[IZLAZ]
        IZLAZ = (IZLAZ + 1) MOD N
        Postavi_OSEM(2)
        obradi poruku(P)
    }
    do zauvijek
}
```

### 6.1.6. Više proizvođača i više potrošača

Potrebno je dodati dva binarna semafora: jedan za proizvođače da se spriječi istovremeni pokušaj stavljanja u međuspremnik i promjena varijable ULAZ, te jedan za potrošače da se spriječi pokušaj istovremenog uzimanja poruke iz međuspremnika i mijenjanja varijable IZLAZ.

```
proizvođač
{
    ponavljaj {
        P = stvori poruku()
        Čekaj_OSEM(2)
        Čekaj_BSEM(1)
        M[ULAZ] = P
        ULAZ = (ULAZ + 1) MOD N
        Postavi_BSEM(1)
        Postavi_OSEM(1)
    }
    do zauvijek
}
```

```
potrošač
{
    ponavljaj {
        Čekaj_OSEM(1)
        Čekaj_BSEM(2)
        P = M[IZLAZ]
        IZLAZ = (IZLAZ + 1) MOD N
        Postavi_BSEM(2)
        Postavi_OSEM(2)
        obradi poruku(P)
    }
    do zauvijek
}
```

## Zadatak 6.1. Ping-pong dretve

Simulirati rad dretvi *ping* i dretvi *pong*. Dretve se nasumično pojavljuju u sustavu (stvaraju ih neke druge dretve) i u svom radu samo ispisuju poruku: dretve *ping* ispisuju `ping` dok dretve *pong* ispisuju `pong`. Sinkronizirati rad dretvi tako da:

a) ispis bude pojedinačan (unutar kritičnog odsječka)

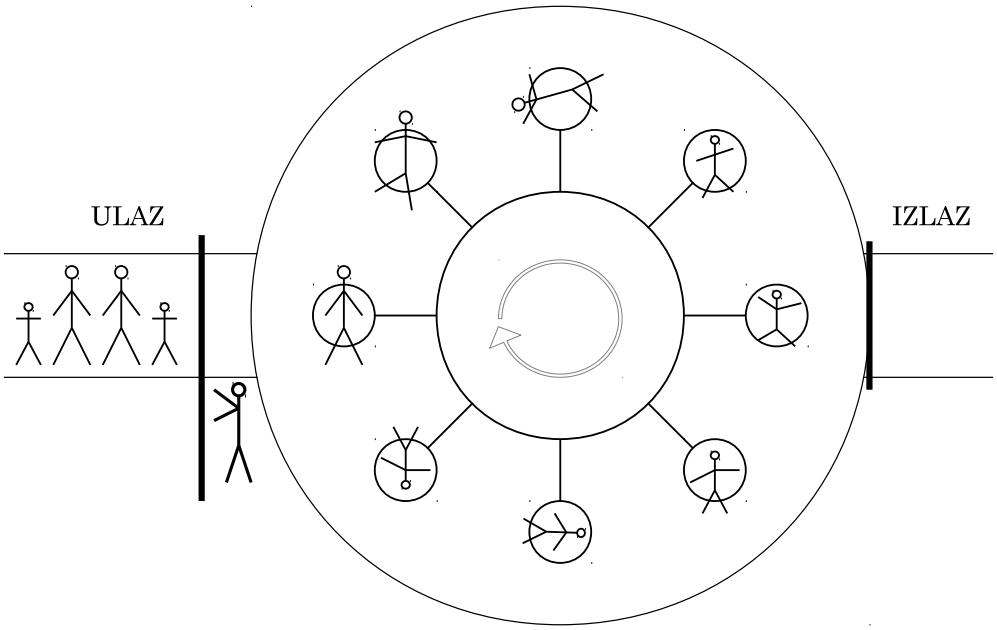
b) dretve *ping* i *pong* naizmjence ispisuju poruke (ispis: `ping pong ping pong ...`)

c) dretve *ping* i *pong* ispisuju poruke tako da se uvijek pojavljuju dva `ping`-a prije svakog `pong`-a (ispis: `ping ping pong ping ping pong ...`)

---

## Zadatak 6.2. Vrtuljak

Modelirati vrtuljak s dva tip dretvi: dretvama *posjetitelj* (koje predstavljaju posjetitelje koji žele na vožnju) te dretvom *vrtuljak* koja predstavlja sam vrtuljak (upravljanje vrtuljkom). Dretvama *posjetitelj* se ne smije dozvoliti ukrcati na vrtuljak prije nego li prethodna grupa ode te kada više nema praznih mjesta (`N`), a pokretanje vrtuljka napraviti tek kada je pun.




---

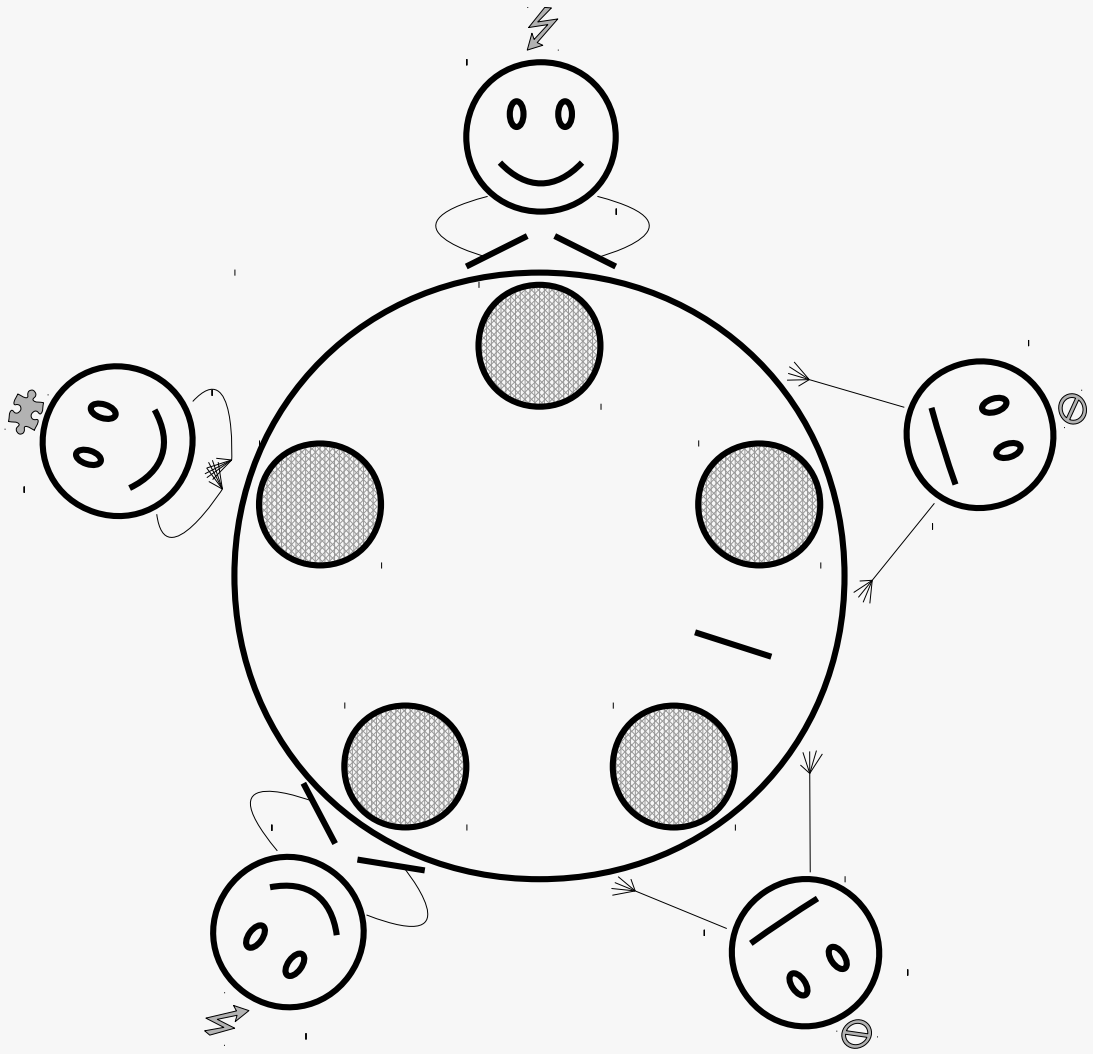
## 6.2. Problemi sa semaforima

### Primjer 6.1. Problem pet filozofa

Pet filozofa sjedi za jednim okruglim stolom. Ispred svakog filozofa je zdjela s hranom (koju konobari nadopunjavaju čim se isprazni). Na stolu se nalazi pet štapića, po jedan između svaka dvije zdjele. Svaki filozof cijelo vrijeme ponavlja sljedeća korake:

1. razmišlja
2. uzima lijevi i desni štapić ako su slobodni, inače čeka da se oslobode
3. jede
4. opere štapiće i vraća ih na stol, lijevo i desno od svoje zdjele

S obzirom da nema dovoljno štapića, neće svi filozofi moći istovremeno jesti.



**Slika 6.5. Problem pet filozofa**

Na slici 6.5. dva filozofa jedu (na vrhu i dole lijevo), jedan razmišlja (lijevo u sredini) te dva su gladna i čekaju da se oslobode štapići (desno u sredini i desno dole).

U simulaciji filozofa dretvama, sinkronizacija semaforima kod koje bi svaki štapić simulirali jednim semaforom mogla bi izgledati kao u nastavku.

```
dretva Filozof(I)
{
    L = I; D = (I + 1) % 5
    ponavljaj {
        misli
        ČekajBSEM(L)       //uzmi_štapić(L)
        ČekajBSEM(D)       //uzmi_štapić(D)
        jedi
        PostaviBSEM(L)     //vrati_štapić(L)
        PostaviBSEM(D)     //vrati_štapić(D)
    }
    do ZAUVIJEK
}
```

Problem s navedenim rješenjem jest u slučaju da sve dretve paralelno rade te paralelno pozovu operaciju `ČekajBSEM(L)` (`L` je različit za svaku dretvu). Obzirom da bi u tom trenutku svi štapići još bili na stolu, sve bi dretve prošle kroz prvi poziv "čekaj" sa semaforima. Međutim, tada bi svi semafori (svih pet) bilo neprolazno i sve bi dretve zapele na drugom "čekaj" pozivu – nastao bi *potpuni zastoj*.

Na prvi pogled navedeni scenarij izgleda malo vjerojatan. Međutim, ako dretve petlju obavljaju jako brzo, jako puno puta, paralelno, vjerojatnost postaje jako velika.

---

Potpuni zastoj se najčešće rješava korištenjem drugih sinkronizacijskih algoritama (npr. monitorima), ali oni neće biti prikazani u okviru ovog predmeta.

## Pitanja za vježbu 6

1. Sinkronizirati više proizvođača i više potrošača koji komuniciraju preko ograničenog međuspremnika korištenjem semafora (i dodatno potrebnih varijabli). Ako se u međuspremniku kapaciteta N poruka u promatranom trenutku nađe M poruka, koje su vrijednosti korištenih općih semafora?
2. Sinkronizirati dvije dretve tako da one neizmjence obavljaju svoje kritične odsječke.
3. Što je to “potpuni zastoj”?

---

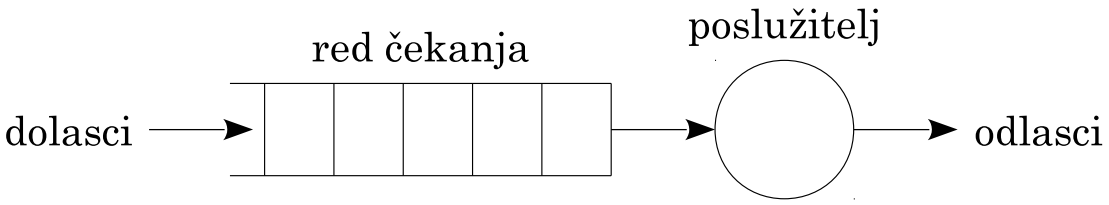
# 7. ANALIZA VREMENSKIH SVOJSTAVA

Ili: kako raspoređivati dretve?

## 7.1. Raspoređivanje po redu prispijeća

Osnovno načelo: dretva koja je najdulje u redu pripravnih bit će prva odabrana za aktivnu.

FIFO - First-in-first-out




**Slika 7.1. Model poslužitelja**

Svojstva:

- vrlo jednostavno za ostvarenje
- problem za sustav s dugim trajanjem izvođenja dretvi – ostale dretve puno čekaju na svoj red

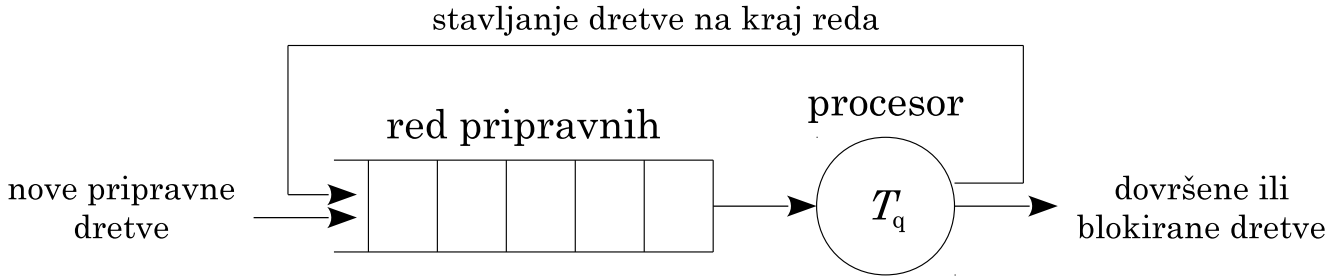
## 7.2. Raspoređivanje prema prioritetu

- vrlo jednostavno za ostvarenje
- najčešći raspoređivač u sustavima gdje su poznate važnosti pojedinih poslova – dretvi
- dretvama treba postaviti odgovarajući prioritet, važnijima veći od manje važnih

## 7.3. Kružno posluživanje (posluživanje podjelom vremena)

Engleski termini: Round-Robin (RR), time-share

Neka različite poslove izvode različite dretve. Posluživanje poslova stoga možemo nazvati posluživanje dretvi, odnosno raspoređivanje dretvi.




- Raspoređivanje pravednom podjelom procesorskog vremena (“svima jednako”)
- Svaki posao dobije kvant vremena $T_q$

  - ako ne završi u tom kvantu vraća se na kraj reda
- Poslovi se prekidaju u izvođenju!
  - npr. koristi se satni mehanizam za izazivanje periodičkih prekida.
- Pravednije posluživanje naspram kraćih poslova.

## 7.4. Raspoređivanje dretvi u operacijskim sustavima

Osnovne (teorijske) strategije raspoređivanja:

- raspoređivanje po redu prispijeća (engl. *first-in-first-out – FIFO*)
- raspoređivanje prema prioritetu
- raspoređivanje podjelom vremena (engl. *time-share, round-robin – RR*)

### Zadatak 7.1. Osnovne strategije raspoređivanja

U nekom sustavu javljaju se poslovi/dretve A, B, C i D u trenucima 3.5, 0, 2,5 i 6,5 respektivno s trajanjima obrade 5, 5, 3 i 2 (respektivno). Pokazati izvođenje dretvi (raspoređivanje) ako se koristi:

a) raspoređivanje po redu prispijeća

b) raspoređivanje prema prioritetu uz $p_A > p_B > p_C > p_D$

c) kružno raspoređivanje s $t_q = 1$ (jedinica vremena)

---

Načini raspoređivanja dretvi u operacijskim sustavima često koriste kombinacije gornjih strategija, ali i ovise o tipovima dretvi. Naime, različiti su zahtjevi pojedinih tipova dretvi. Primjeri tipova dretvi:

- dretve koje preko U/I naprava upravljaju nekim procesima (industrija, automobili, zgrade, ...) – kritične dretve koje moraju brzo/pravovremeno reagirati/slati naredbe
- dretve koje obavljaju neke dugotrajne proračune – ako ih se malo i odgodi "neće to primijetiti"
- dretve koje upravljaju sučeljem programa – korisničko iskustvo će biti bolje ako te dretve što prije dođu na red za izvođenje, a uglavnom su vrlo brzo gotove sa svojim poslom u tom trenutku ("interaktivne dretve")

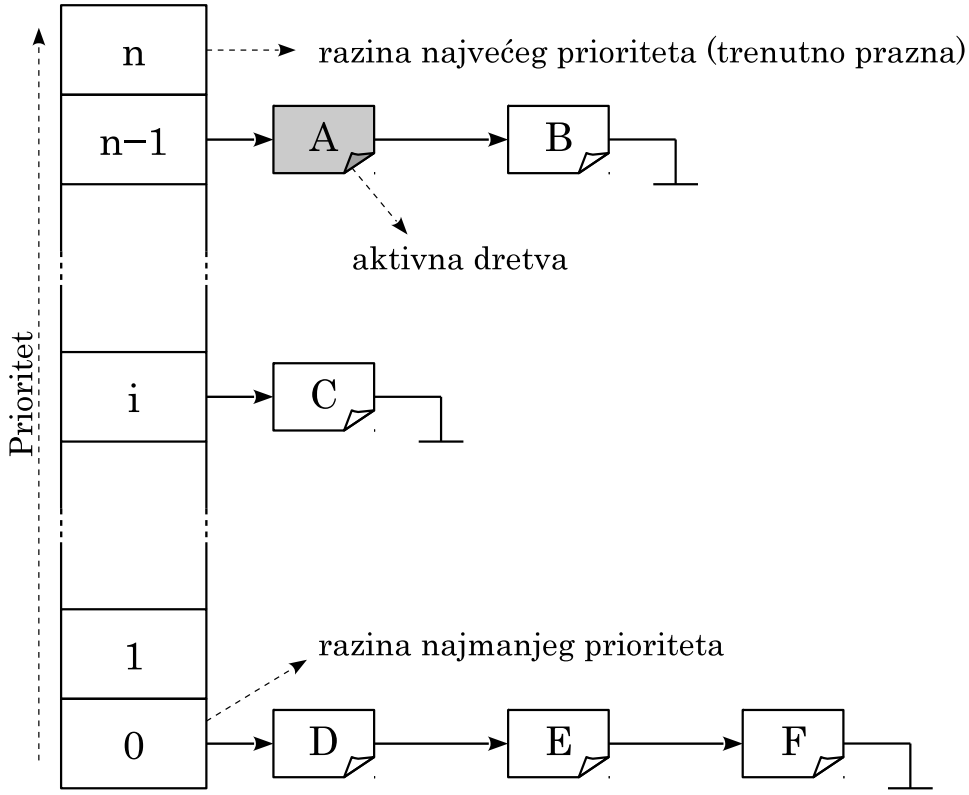
Obzirom na navedeno, dretve u kontekstu raspoređivanja unutar operacijskih sustava se dijele na vremenski kritične i vremenski nekritične (normalne). Raspoređivanje jednih i drugih nije jednako. (Više o raspoređivanju)

### 7.4.1. Vremenski kritične dretve (engl. *real-time*)

**Prioritetno raspoređivanje**

- **osnovni kriterij** = prioritet – UVIJEK je aktivna ona dretva s najvećim prioritetom
- kad se u sustavu pojavi nova dretva B s većim prioritetom od prioriteta trenutno aktivne dretve A, dretva B postaje aktivna – istisne dretvu A koja će nastaviti s radom tek nakon što se dretva B makne (završi ili blokira)

- kada osnovni kriterij ne daje samo jednu dretvu (postoji više dretvi ista najveća prioriteta) tada se za odabir jedne od njih koristi **dodatni** kriterij najčešće FIFO ili RR:
  - prema redu prispijeća – FIFO: aktivna dretva se ne mijenja sve se ne završi ili blokira (ili dođe nova dretva većeg prioriteta)
  - podjelom vremena – RR: aktivna se izvodi zadani kvant vremena, potom se miče iza svih dretvi ista prioriteta u redu pripravnih dretvi i uzima iduća pripravna (ista prioriteta)
  - dodatni kriterij se najčešće definira za svaku dretvu zasebno!
- s obzirom na to da se uvijek najprije gleda prioritet, način raspoređivanja (postavljen pojedinoj dretvi) dobiva ima po dodatnom kriteriju; nazivi prema istoimenim UNIX raspoređivačima:
  - SCHED_FIFO (prioritet pa red prispijeća)
  - SCHED_RR (prioritet pa podjela vremena)
- slika 7.2. prikazuje primjer skupa pripravnih dretvi u kojemu su dretve složene u različite redove, u skladu s njihovim prioritetima




**Slika 7.2. Sustav pripravnih dretvi raspodijeljen prema prioritetu**

- dretve A i B imaju u tom trenutku najveći prioritet $p_A = p_B = n - 1$
- prva dretva u redu je dretva A => aktivna dretva
- ako je drugi kriterij za raspoređivanje dretve A FIFO, ona će se izvoditi dok ne završi ili više ne bude u pripravnom stanju
- ako je drugi kriterij za raspoređivanje dretve A RR, ona će se izvoditi dok ne završi ili više ne bude u pripravnom stanju ili dok ne istekne kvant vremena koji raspoređivač koristi
- korištenje strukture podataka pripravnih dretvi kao na slici 7.2., s jednom listom za svaku razinu prioriteta omogućuje složenost O(1):
  - uz dodatnu kazaljku na zadnji element liste, dodavanje dretve u red pripravnih se radi u

jednom koraku

  - uz bitmapu u kojoj jedinice označavaju neprazne liste, odabir naprioritetnije se također obavlja u jednom koraku (jednom instrukcijom se iz bitmape dobije indeks najviše postavljene jedinice, tj. indeks najprioritetnijeg nepraznog reda pripravnih dretvi)

- u višeprocesorskim sustavima za svaki procesor se koristi zasebni red pripravnih
  - razlog je efikasnost iz dva aspekta
    1. korištenje priručnog spremnika – dretva se vraća na isti procesor u čijem priručnom spremniku su možda još uvijek neki njeni podaci – ne treba ih dohvaćati iz memorije (na što bi se inače potrošilo vrijeme)
    2. smanjena potrebi zaključavanja reda – svaki procesor ima svoj red pa pri raspoređivanju uglavnom zaključava samo svoj red
  - problemi kad se koriste prioritetni raspoređivači: moglo bi se dogoditi da u nekom redu pripravnih bude dretva veće prioriteta od neke druge koja se izvodi na drugom procesoru; to treba spriječiti (push-pull postupci – gurnuti takvu dretvu drugom procesoru, uzeti takvu dretvu iz reda drugog procesora)

### 7.4.2. "Normalne" dretve – nekritične dretve

- osnovna ideja: raspoređivanje podjelom vremena

a) pravedna podjela procesorskog vremena

b) prioritet određuje udio vremena koji će dretva dobiti (uglavnom)

c) tip dretve određuje udio vremena koji će dretva dobiti

- heuristika koja se nastoji primijeniti na raspoređivanje nekritičnih dretvi (podjela vremena tipa c)) opisuje se algoritmom naziva *višerazinsko raspoređivanje s povratnom vezom* (engl. *multilevel feedback queue – MFQ*).

**MFQ algoritam (info)**

- algoritam nastoji:
  - dati prednost dretvama s kratkim poslovima
  - dati prednost dretvama koje koriste ulazno-izlazne naprave
  - na osnovi rada dretve ustanoviti u koju skupinu dretva pripada te ju prema tome dalje raspoređivati
- koristi nekoliko FIFO redova različita prioriteta
- aktivna dretva je prva dretva iz nepraznog reda najveća prioriteta
- dretva dobiva kvant vremena
- ako dretva završi prije isteka kvanta nestaje iz sustava
- ako se dretva blokira prije isteka kvanta privremeno nestaje iz domene raspoređivača (nije među pripravnima); pri odblokiranju, takva dretva se vraća u isti red iz kojeg je i otišla ili čak i viši red (dretva zadržava prioritet ili joj se on i povećava)
- ako je dretva protrošila cijeli kvant, onda ju raspoređivač pri isteku kvanta miče na kraj prvog idućeg reda (manjeg prioriteta)

- u redu najmanjeg prioriteta dretve se poslužuju kružno (svakoj kvant vremena)
- pri pojavi nove dretve ona ide u red najvećeg prioriteta (na kraj)
- sustav redova identičan je onome sa slike 7.2.
- MFQ algoritam brzo procjenjuje/klasificira dretvu:
  - je li procesorski zahtjevna
  - je li kratka ("interaktivna", uglavnom koristi UI)
- ... te se dalje prema njoj odnosi prema tome
  - kratke dretve dobivaju prednost jer brzo izlaze iz sustava
  - zahtjevne dijele procesorsko vrijeme podjelom vremena
- operacijski sustavi (danas) nemaju raspoređivače ostvarene po navedenom algoritmu, ali nastoje (s drugim algoritmima) ostvariti slična načela

### 7.4.3. Raspoređivanje u Linuxu

- za korisničke dretve Linux ima tri raspoređivača:
  1. prema roku; DEADLINE – SCHED_DEADLINE
  2. prema prioritetu: REALTIME – SCHED_FIFO, SCHED_RR
  3. obične dreteve: CFS/EEVDF – SCHED_OTHER

**Raspoređivanje dretvi za rad u stvarnom vremenu (engl. *real time*)**

- strategije: SCHED_FIFO, SCHED_RR (striktno)
  - prioriteti od 0 do 99 (veći broj veći prioritet)
- strategija: SCHED_DEADLINE – raspoređivanje prema krajnjim trenucima završetaka (info)
- ove dretve uvijek imaju prednost pred običnim dretvama!

**Raspoređivanje "običnih" dretvi**

- strategije: SCHED_OTHER i slični (_IDLE, _BATCH),
- prioriteti "službeno" od 100 do 139, ali se ne koriste
- koristi se "razina dobrote" (engl. *nice*): -20 do 19
  - manji broj veća dobrota (ekvivalent prioritetu)
  - negativne vrijednosti može postavljati samo povlašteni korisnik
  - pri pokretanju obična procesa njegova dobrota je 0
- cilj raspoređivača jest "pravedno" podijeliti procesorsko vrijeme
- dretve veće dobrote (prioriteta) dobivaju više procesorskog vremena
  - oko 10-15% po prioritetu
  - npr. dretva dobrote 0 treba dobiti oko $1{,}15^5$ više procesorskog vremena od dretve dobrote 5

**Algoritam CFS: potpuno pravedan raspoređivač (engl. *Completely Fair Scheduler*)**

(info)

- koristi se od jezgre 2.6.23 (2007.)
- koristi razliku između:
  - izračunatog vremena koje pripada pojedinoj dretvi s obzirom na njenu dobrotu
  - stvarno dodijeljenog procesorskog vremena pojedinoj dretvi
- razlika određuje položaj opisnika dretve u stablu (crveno-crna stabla)
- aktivna dretva je dretva kojoj sustav najviše duguje ("najlijevija" dretva u stablu)
- od jezgre 6.6. (2023.) koristi se raspoređivanje prema roku spremnih zadataka (EEVDF), ali on je sličan CFS-u, tj. nešto malo složeniji i nije ovdje opisan (u skraćenoj inačici skripte)

### 7.4.4. Raspoređivanje u Windowsima

- koristi se prioritetno raspoređivanje s podjelom vremena (slično SCHED_RR)
- prioritet se formira na osnovu: prioritetne klase procesa i prioritetne razine dretvi
- raspoređivanje dretvi za rad u stvarnom vremenu (engl. *real time*):
  - strategija: REALTIME_PRIORITY_CLASS uz kružno posluživanje (identično sa SCHED_RR)
  - prioriteti od 16 do 31 (veći broj veći prioritet)
- raspoređivanje "običnih" dretvi:
  - strategija: kružno (slično kao i SCHED_RR) uz iznimke:
    - radi rješavanja problema izgladnjivanja (da i dretve manjeg prioriteta ipak nešto povremeno rade)
    - dinamičkog dodavanja prioriteta procesima u fokusu (engl. *foreground*)
    - radi osiguravanja kvalitete usluge (ne samo na razini procesora)
  - prioriteti od 0 do 15 (veći broj veći prioritet)

### Zadatak 7.2. Raspoređivanje prema SCHED_FIFO, SCHED_RR i SCHED_OTHER

U nekom sustavu javljaju se poslovi/dretve A, B, C i D u trenucima 3.5, 0, 2,5 i 6,5 respektivno s trajanjima obrade 5, 5, 3 i 2 (respektivno). Pokazati rad poslužitelja ako se koristi:

d) raspoređivanje prema SCHED_FIFO uz $p_A > p_B = p_C = p_D$

e) raspoređivanje prema SCHED_RR uz $p_A > p_B = p_C = p_D$ i $t_q = 1$

f) raspoređivanje prema SCHED_OTHER uz kvantove po dretvama $t_{q_A} = 2$, $t_{q_B} = t_{q_C} = t_{q_D} = 1$ (npr. za $d_A = 5, d_B = d_C = d_D = 10$ – razlika od 5 u dobroti: $1{,}15^5 \approx 2$)

---

### Zadatak 7.3. Raspoređivanje na dvoprocesorskom računalu

U nekom sustavu javljaju se poslovi/dretve A, B, C i D u trenucima 3.5, 0, 2,5 i 6,5 respektivno s trajanjima obrade 5, 5, 3 i 2 (respektivno). Pokazati rad **dvoprocesorskog** poslužitelja ako se koristi:

g) raspoređivanje prema SCHED_FIFO uz $p_A > p_B = p_C = p_D$

h) raspoređivanje prema SCHED_RR uz $p_A > p_B = p_C = p_D$ i $t_q = 1$

i) raspoređivanje prema SCHED_OTHER uz kvantove po dretvama $t_{q_A} = 2$, $t_{q_B} = t_{q_C} = t_{q_D} = 1$ (npr. za $d_A = 5, d_B = d_C = d_D = 10$ – razlika od 5 u dobroti: $1{,}15^5 \approx 2$)

---

## Pitanja za vježbu 7

1. Opisati osnovne principe raspoređivanja dretvi:
   a) po redu prispijeća (engl. *First-In-First-Out – FIFO*)
   b) podjelom vremena (engl. *Round-Robin – RR*)

   Navesti prednosti i nedostatke navedenih principa raspoređivanja.
2. Opisati principe raspoređivanja dretvi koji se koriste u današnjim operacijskim sustavima (Windows, Linux, SCHED_FIFO, SCHED_RR, SCHED_OTHER).

# 8. UPRAVLJANJE SPREMNIČKIM PROSTOROM

Osim riječi *spremnik*, tj. *radni spremnik*, jednakovrijedna je i riječ *memorija*, koja se udomaćila u hrvatskom jeziku.

## 8.1. Uvod

### 8.1.1. Veličine radnog spremnika, što se sve nalazi u njemu

Veličina spremnika (RAM) za pojedine namjene (okvirno) (info)

- poslužitelj: 16 GB+
- osobno računalo: 4 GB+
- pametni telefoni i tableti: 512 MB+
- ugrađeni sustavi: obično u rasponu od 2 KB do desetke MB

Adresiranje spremničke lokacije – širina sabirnice i moguće veličine spremnika:

- 16 bita $\Rightarrow$ 64 KB = 65536 B
- 32 bita $\Rightarrow$ 4 GB $\approx 4 \cdot 10^9$ B
- 64 bita $\Rightarrow$ 16 EB (exa-) $\approx 2 \cdot 10^{19}$ B
  - ovo je previše i za današnje sustave
  - x64 procesori koriste 48 bita i teoretski mogu koristiti 256 TB spremnika

Pristup spremniku (od strane procesora)

- pri dohvatu instrukcija
- pri dohvatu i pohrani operanada (podataka)
- pri korištenju stoga

Što sve ide u spremnik?

- jezgra:
  - jezgrine funkcije (npr. `Započni_UI`, `ČekajBSEM`, ...)
  - strukture podataka (opisnici dretvi, procesa, naprava, liste, ...)
- programi:
  - instrukcije
  - podaci
  - gomila (heap, za malloc/free)
  - stog (za svaku dretvu zaseban)

### 8.1.2. Osnovne ideje upravljanja spremnikom

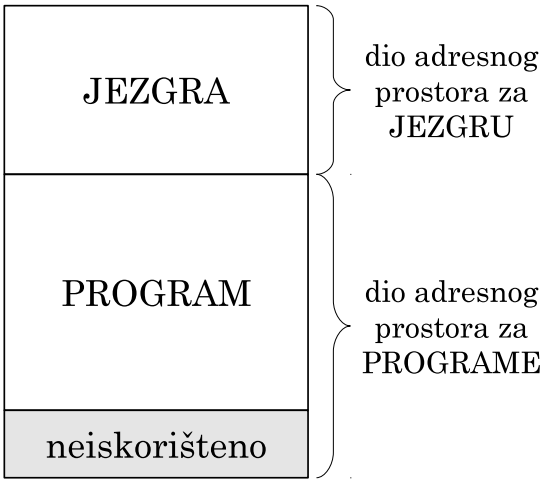
Pojmovi program i proces u ovom kontekstu

- pod pojmom proces obično podrazumjevamo i odvojeni adresni prostor od ostalih procesa
- ipak, ovdje se koristi i za načine upravljanja koji to nemaju
- *program* se odnosi na niz instrukcija i podataka čijim pokretanjem nastaje proces (OS pokreće program i nastaje proces)

Zahtjevi od programa i OS-a prema upravljanju spremnikom

- da sve "potrebno" za rad stane u spremnik
- zaštita "da nitko nikome ne smeta", tj. od grešaka u programima i zlonamjernih programa
  - da se zaštiti jezgra od procesa, proces od drugog procesa
- učinkovito koristiti radni spremnik i po potrebi pomoćne spremnike
- dati preporuke programerima kako učinkovito koristiti spremnik

**Najjednostavniji način upravljanja: samo OS i jedan proces (slika 8.1.)**




**Slika 8.1. Upravljanje spremnikom u sustavima s jednim procesom**

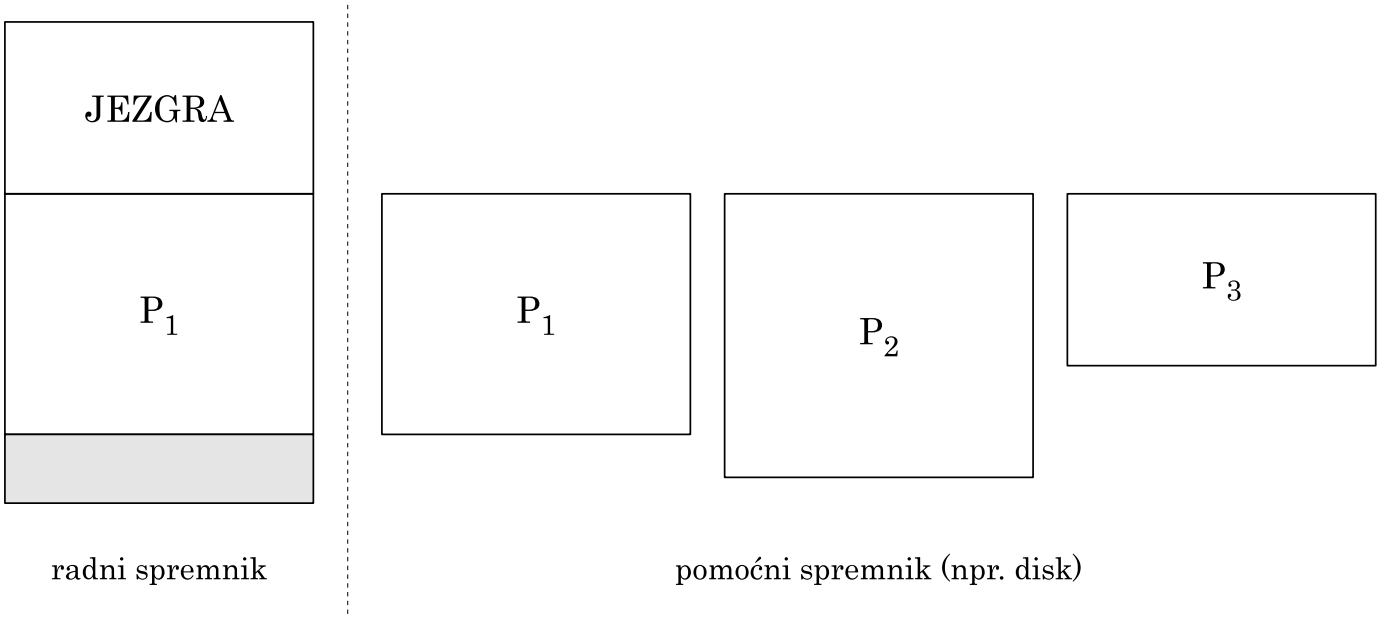
- ovo je dovoljno samo za vrlo jednostavne sustave
  - međutim, takvih sustav ima jako puno – većina ugrađenih sustava je jednostavno i ovo je dovoljno

**Više procesa – svi u radnom spremniku**

- ovo je "idealno", ali potencijalno traži puno spremnika (stoga i skupo rješenje)
- koristi se tamo gdje su procesi mali i svi stanu u spremnik (ugrađeni sustavi)

**Više procesa – jedan u radnom spremniku, ostali na pomoćnom (slika 8.2.)**

- na pomoćnom spremniku su pripremljeni svi procesi
- po jedan se učitava u radni spremni i izvodi
- pri zamjeni procesa radi se "velika" zamjena konteksta:
  1. proces iz radnog spremnika miče se na pomoćni spremnik
  2. drugi proces se učitava s pomoćnog u radni spremnik

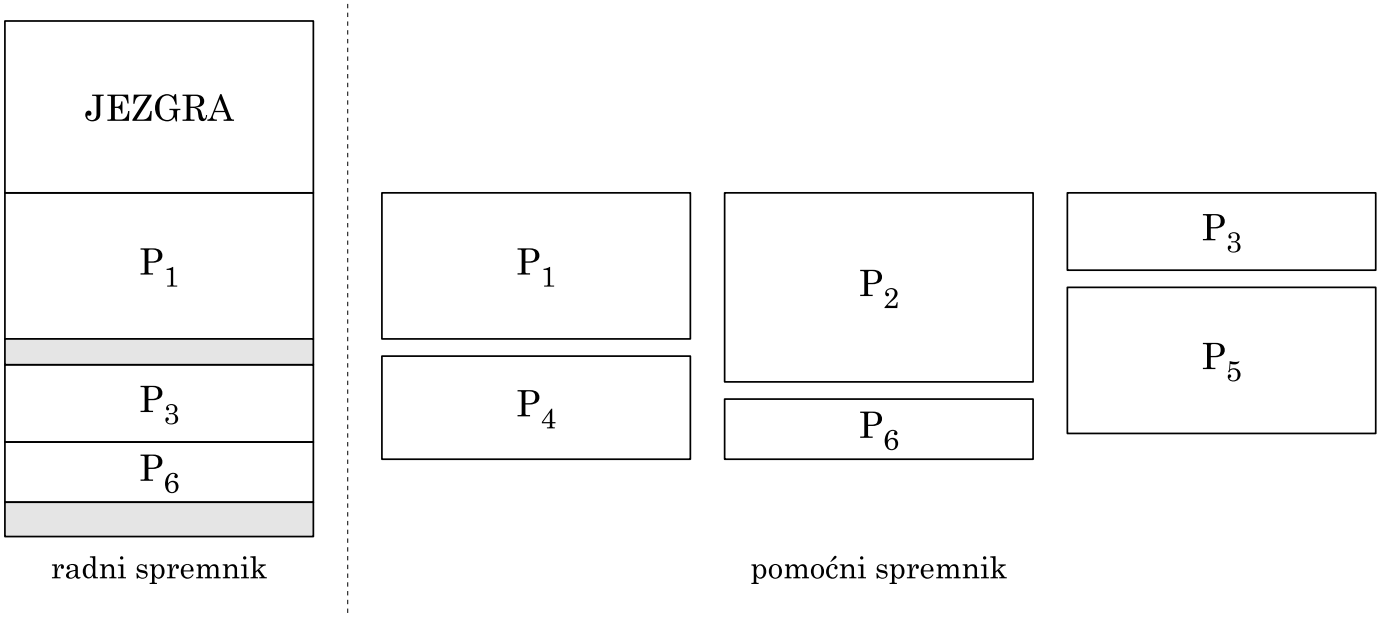



**Slika 8.2. Korištenje pomoćnog spremnika za "veliku zamjenu konteksta"**

**Svojstva pomoćnog spremnika – diska**

- disk se detaljnije razmatra u 9. poglavlju, sada samo osnovna svojstva
- vremenska svojstva diska:
  - vrijeme pristupa koda HDD-a (čitanje jednog bloka/sektora): red veličine ≈ 10 ms!
  - vrijeme pristupa koda SSD-a (čitanje jednog bloka/sektora): red veličine ≈ 0,05 ms!
  - čitanje procesa s diska od ≈ 1 ms do 100 ms (i više)!
- spremniku se pristupa znatno brže ≈ red veličine nekoliko ns (bar 1000 puta brže čak i uz SSD)
- disk je prespor da bi navedeni gornji način (prema slici 8.2.) bio zadovoljavajući i učinkovit

**Učinkovito korištenje pomoćnog spremnika za upravljanje spremnikom (slika 8.3.)**



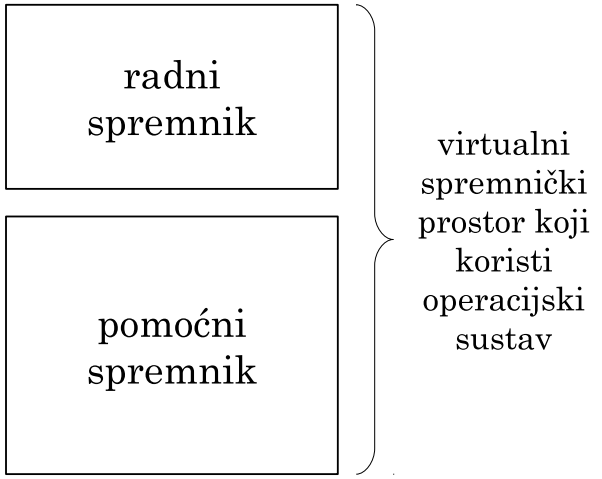

**Slika 8.3. Primjer sustava s više procesa u radnom spremniku**

- u radnom spremniku treba biti više od jednog procesa (ako svi ne stanu)

- koristiti pomoćni spremnik za privremenu pohranu svih procesa
  - svi se početno pripremaju na pomoćnom disku, a neki učitavaju u radni spremnik
  - za učitavanje/spremanje procesa koristiti sklopove s izravnim pristupom spremniku (DMA)
- zamjena jednog procesa (spremanje prvog pa učitavanje drugog) – velika zamjena konteksta
- za vrijeme obavljanja zamjene (korištenjem DMA načina) procesor može izvoditi neki drugi proces koji je u radnom spremniku – ne gubi se to vrijeme, ovakav sustav je učinkovit

### Virtualni spremnički prostor

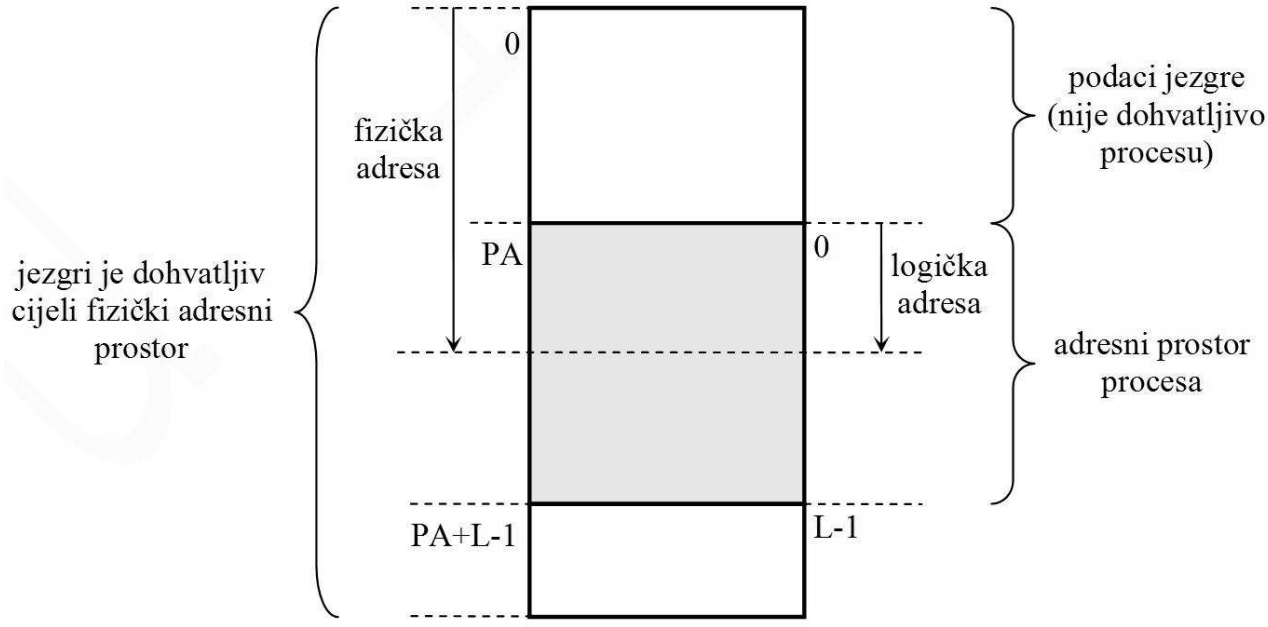
U sustavima koji koriste pomoćni spremnik za upravljanje spremničkim prostorom koristimo pojam virtualni spremnički prostor OS-a (engl. *virtual memory*) koji se sastoji od radnog spremnika i pomoćnog spremnika.




**Slika 8.4. Virtualni spremnički prostor OS**

### Fizičke i logičke adrese

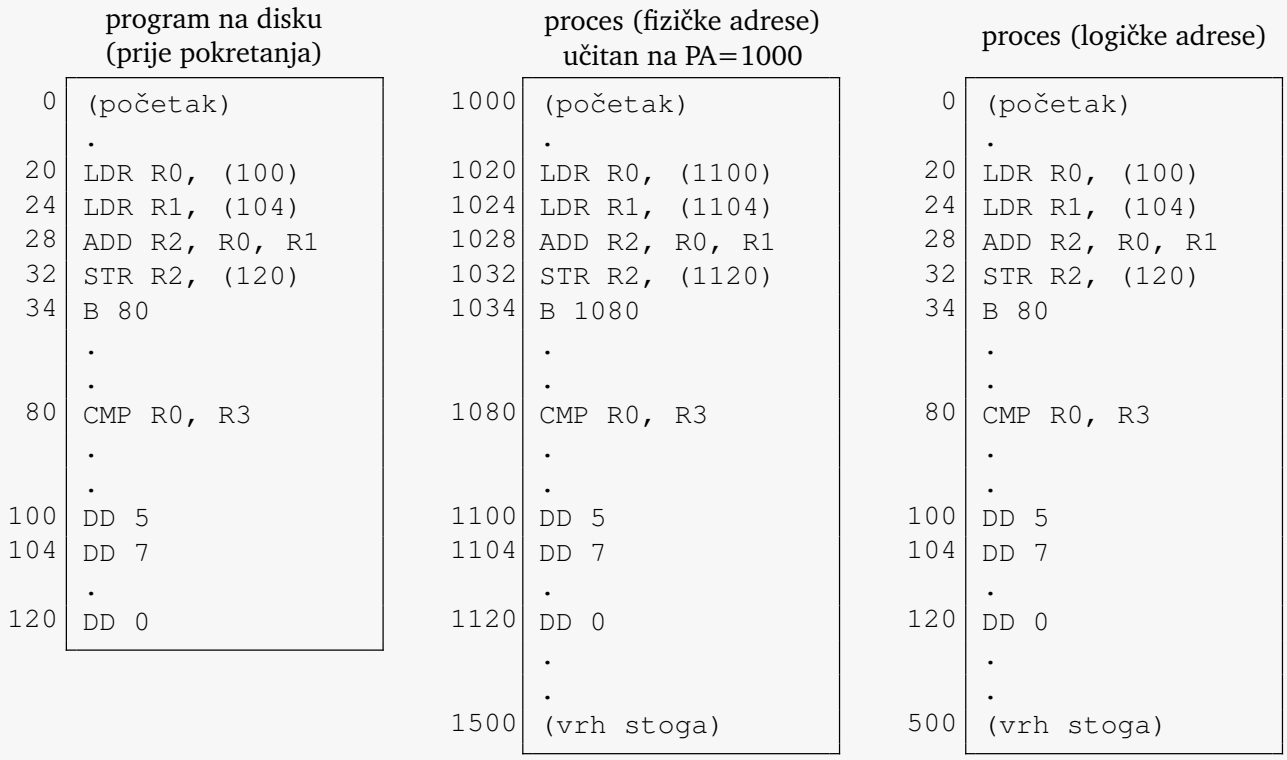
- apsolutne adrese = fizičke adrese → adrese koje idu na sabirnicu
- relativne adrese = logičke adrese → koriste se u procesu




**Slika 8.5. Fizička i logička adresa**

## Primjer 8.1. Fizička i logička adresa

Odnos fizičke i logičke adrese procesa ovisi gdje se on nalazi.




### Procesni informacijski blok

- za upravljanje procesom potreban je i opisnik spremničkog prostora procesa
  - sadržaj opisnika ovisan je o algoritmu upravljanja spremnikom
  - opisnik mora opisivati korišteni dio radnog spremnika, ali i korišteni dio pomoćnog spremnika od strane procesa
- za upravljanje procesa, osim opisnika spremničkog prostora procesa potrebni su i opisnici dretvi, opisnici korištenih datoteka i slično
- skup potrebnih informacija za upravljanje procesom (koji uključuje i sve navedeno) naziva se *procesni informacijski blok*

### Načini upravljanja spremnikom koji koriste pomoćni spremnik

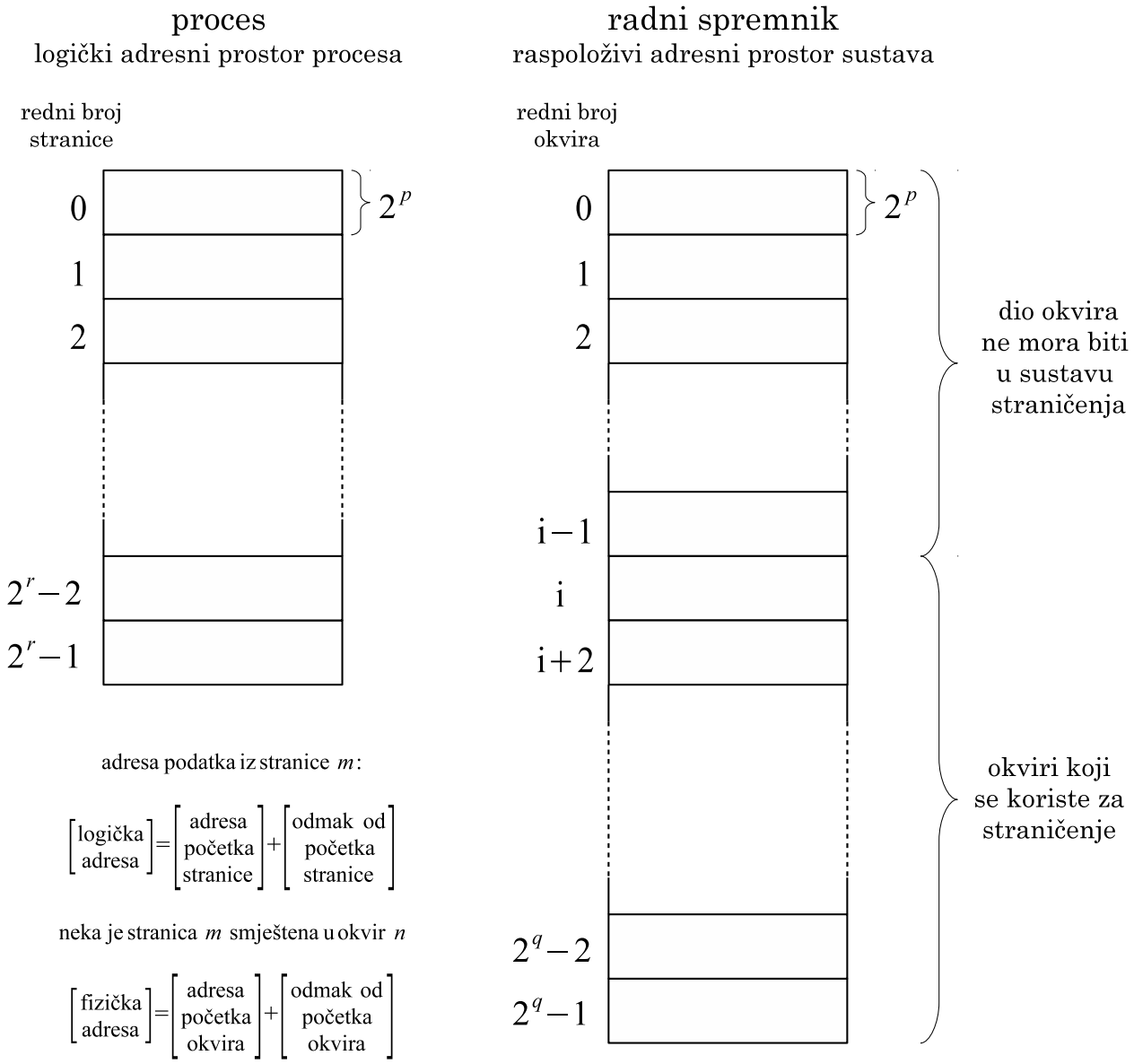
1. statičko upravljanje: statički podijeliti spremnik na particije
2. dinamičko upravljanje: dinamički dijeliti spremnik prema potrebama
3. straničenje

## 8.2. Straničenje

### 8.2.1. Stranice, okviri, tablica prevođenja

Osnovne ideje:

- proces se dijeli na *stranice*
- radni spremnik se dijeli na *okvire*
- jedna stranica stane u jedan okvir (istih su veličina, uobičajeno 4 KB)
- pretvorba adresa obavlja se dodatnim *sklopom* uz odgovarajuću *strukturu podataka* – tablicu prevođenja, koju održavaju jezgrine funkcije
- u radnom spremniku nalaze se samo trenutno potrebne stranice procesa
- u pomoćnom spremniku nalaze se sve stranice procesa (u ovom modelu; u stvarnim sustavima se samo stranice procesa koje nisu u radnom spremniku nalaze u pomoćnom spremniku)
- stranice se učitavaju prema potrebi – kad su potrebne za rad procesa

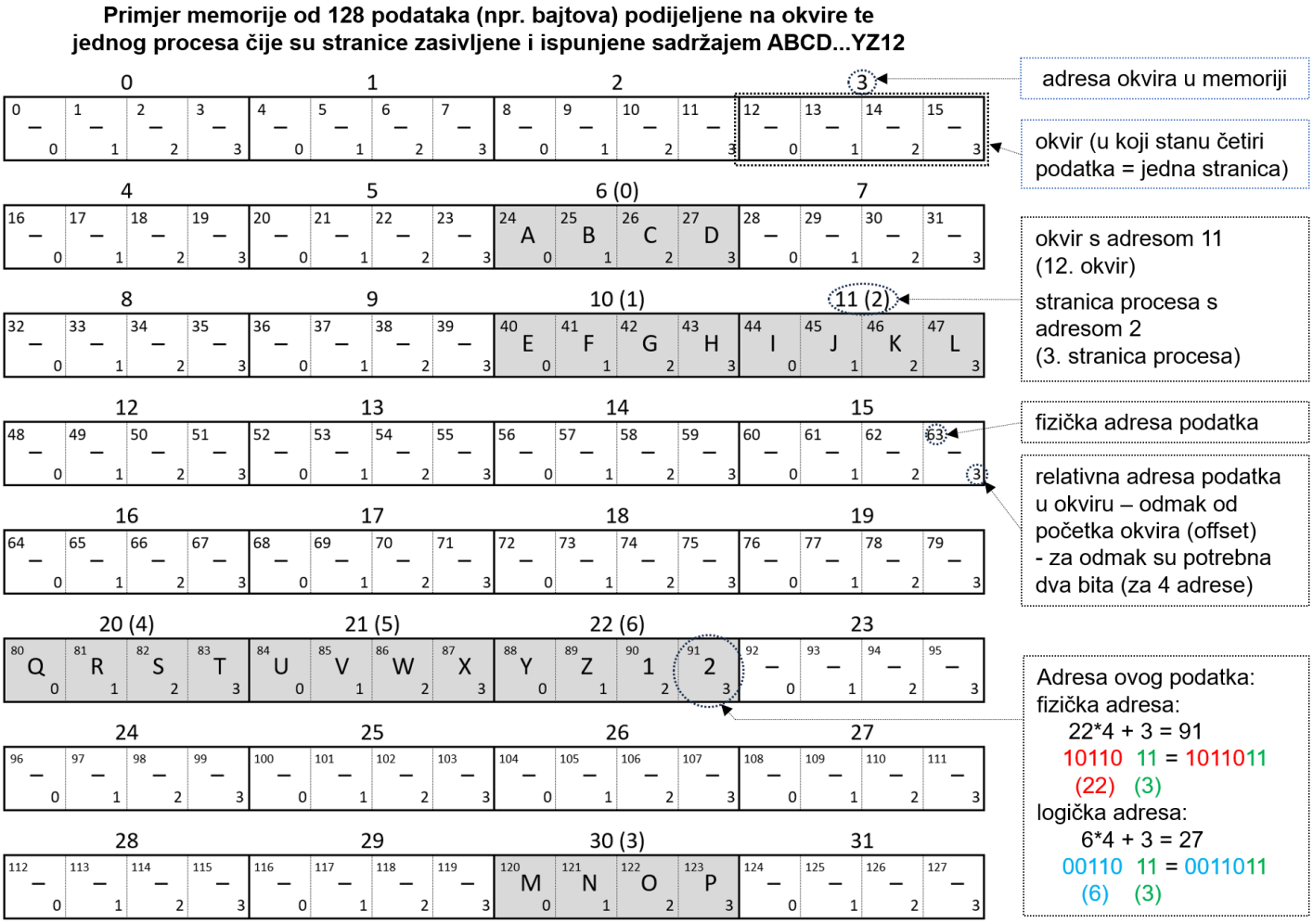



**Slika 8.6. Podjela na stranice i okvire**

Veličina stranice mora biti potencija broja 2 da bi se adresa (logička i fizička) mogla podijeliti

na dva dijela

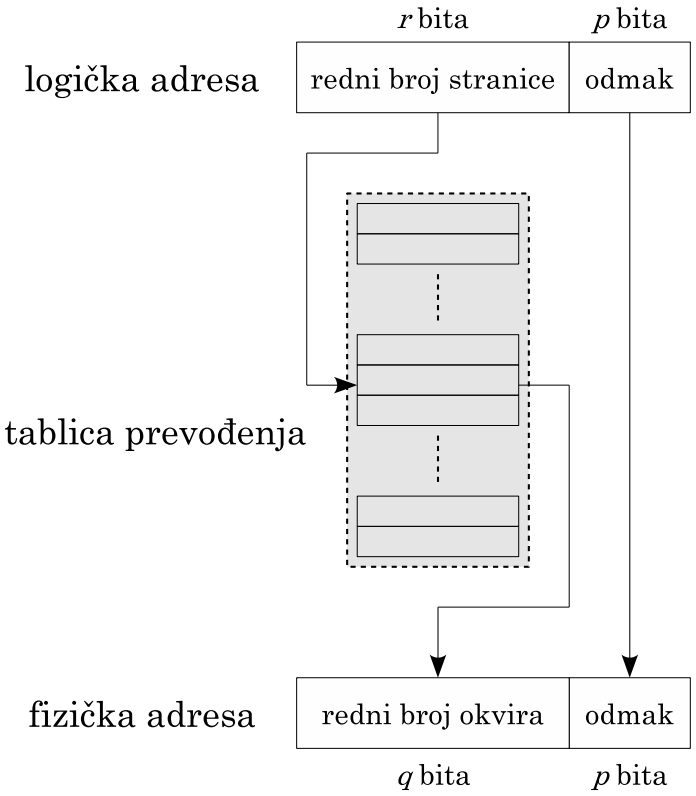
- logička adresa:
  - *redni broj stranice* (indeks stranice) – viših $r$ bita adrese
  - *odmak* od početka stranice – nižih $p$ bita adrese
- fizička adresa:
  - *redni broj okvira* (indeks okvira) – viših $q$ bita adrese
  - *odmak* od početka okvira – viših $p$ bita adrese




**Slika 8.7. Ilustrativni primjer s malim spremnikom i malim okvirima**

**Tablica prevođenja**

- za svaku stranicu procesa postoji jedan opisnik – jedan redak u tablici prevođenja
- zapisana je u opisniku procesa, za svaki proces treba zasebna tablica
- izgrađuje ju i održava OS, koristi sklop za pretvorbu adresa



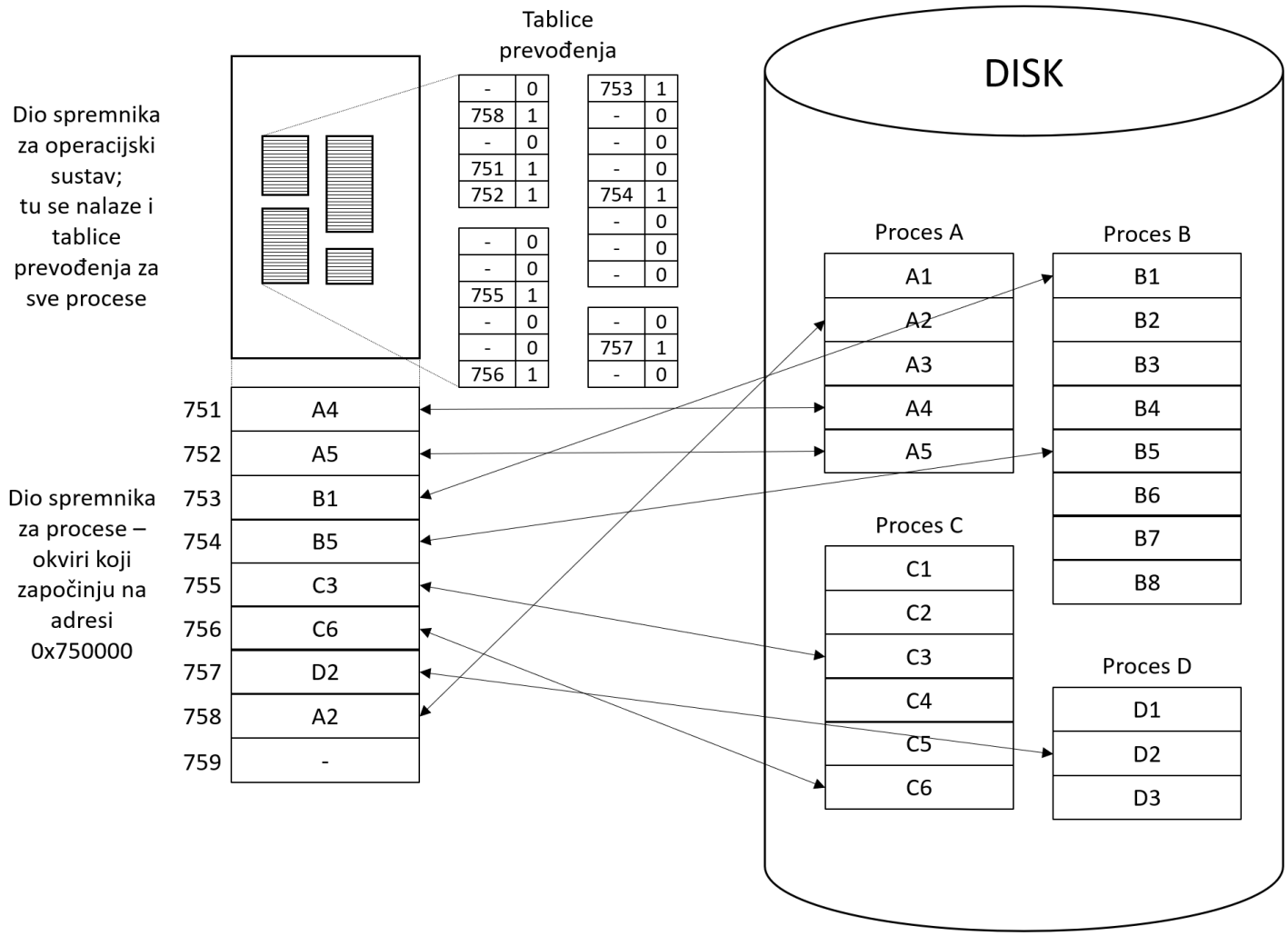

**Slika 8.8. Pretvorba adrese kod straničenja**

- svaki opisnik stranice se sastoji od dva dijela:
  - adrese okvira u kojem se stranica nalazi (ako se nalazi)
  - zastavice – detalji o stranici
    - bita prisutnosti: nalazi li se stranica u radnom spremniku (1) ili ne (0)
    - bitova koji označavaju pristup stranici, promjenu sadržaja, zaštitu od promjene/pristupa i sl. (zastavice su detaljnije opisane kasnije)

Pretvorba logička $\Longrightarrow$ fizička:

- *odmak* se prekopira
- viši bitovi logičke adrese – *redni broj stranice* – koristi se kao indeks u tablici prevođenja iz koje se dohvaća opisnik te stranice
- u opisniku stranice piše *redni broj okvira* u kojem se stranica nalazi (kada je bit prisutnosti postavljen, ako nije sklop izaziva prekid zbog promašaja – o tome kasnije)

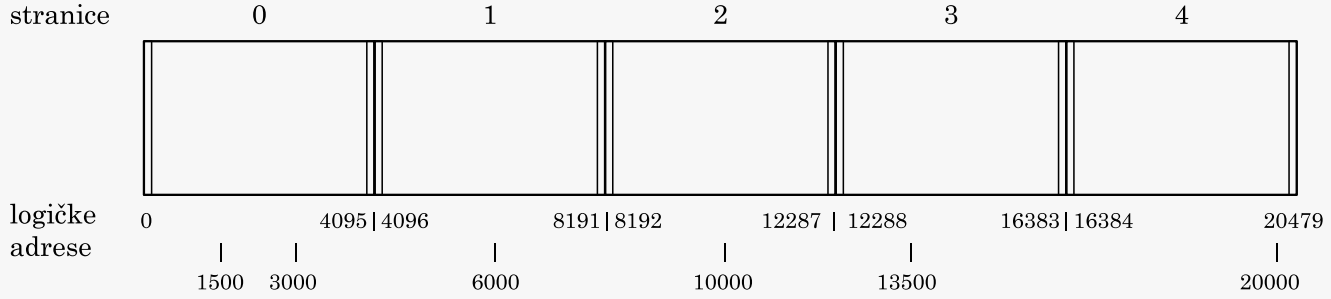
Prevođenje adresa mora biti obavljeno sklopom – inače bi sustav bio prespor




**Slika 8.9. Ilustrativni primjer straničenja sa svim elementima i njihovim smještajem**

## Primjer 8.2. Stranice, okviri, logičke i fizičke adrese

Neki proces se sastoji od pet stranica, svaka veličine 4096B prema slici 8.10. Uz tablicu prevođenja 8.1. pokazati kako će se prevesti izdvojene logičke adrese (1500, 3000, . . . ).




**Slika 8.10. Proces, u logičkom adresnom prostoru**

**Tablica 8.1. Tablica prevođenja**

| stranica | indeks okvira | bit prisutnosti |
|---|---|---|
| 0 | 0x108B | 1 |
| 1 | | 0 |
| 2 | 0x3039 | 1 |
| 3 | | 0 |
| 4 | 0x673 | 1 |

Pri pretvorbi iz logičke u fizičku, adresa se najprije dijeli na dva dijela, gornji dio koji označava redni broj stranice procesa te donji dio koje označava odmak od početka stranice. Obzirom da je stranica velika 4 KB, za odmak je potrebno 12 najnižih bita adrese, što se u heksadekadskom zapisu zapisuje s tri znamenke. U nastavku će se adrese najprije prikazati u heksadekadskom obliku, a da bi se ovo rastavljanje pojednostavilo.

Npr. logička adresa 10000 = 0x2710 se rastavlja na {0x2, 0x710} te preko tablice prevođenja prevodi u fizičku adresu {0x3039, 0x710}, tj. 0x3039710.

**Tablica 8.2. Preslikavanje adresa**

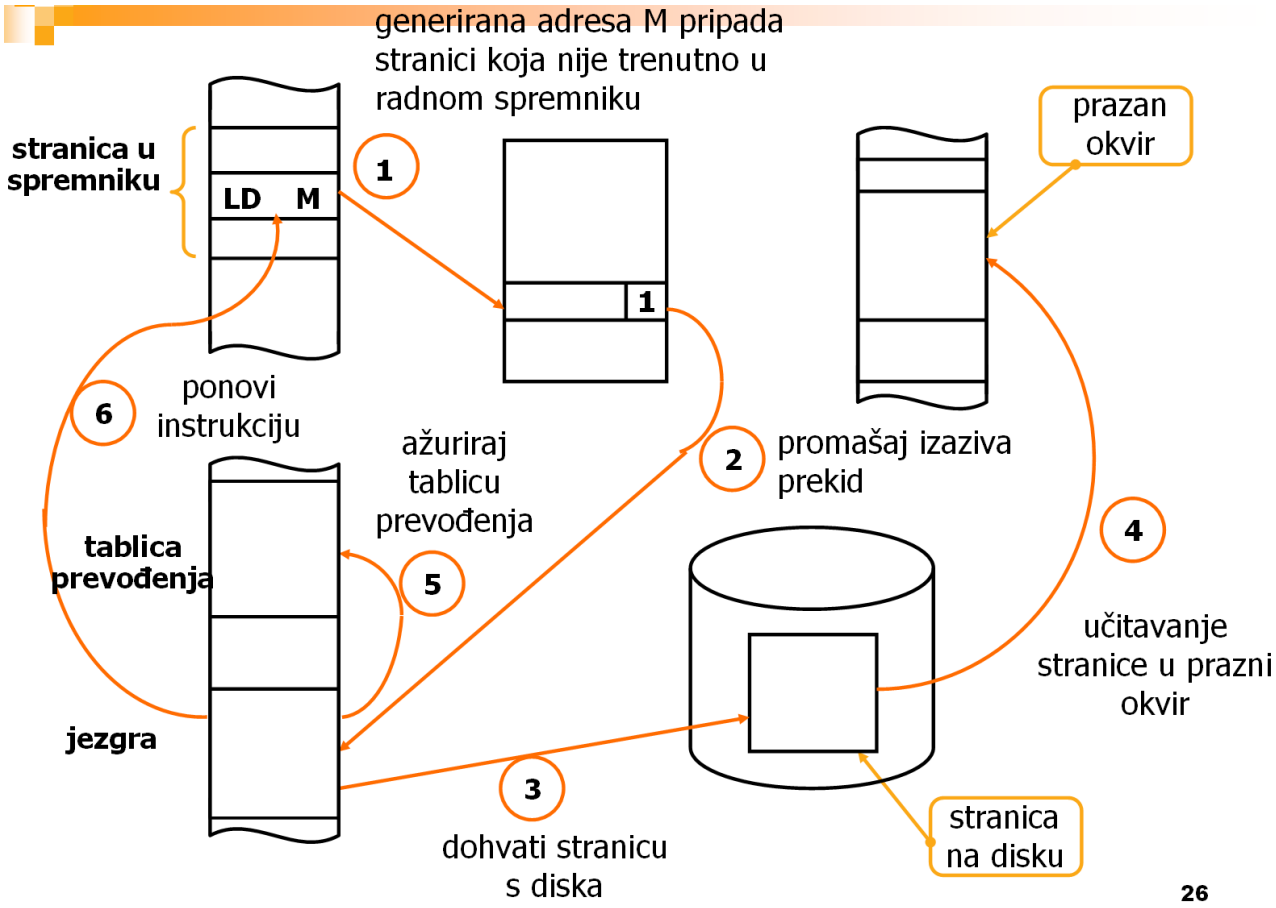
| logička adresa | | | fizička adresa | |
|---|---|---|---|---|
| LA | hex(LA) | {stranica, odmak} | {okvir, odmak} | hex(FA) |
| 1500 | 0x05DC | {0x0, 0x5DC} | {0x108B, 0x5DC} | 0x108B5DC |
| 3000 | 0x0BB8 | {0x0, 0xBB8} | {0x108B, 0xBB8} | 0x108BBB8 |
| 6000 | 0x1770 | {0x1, 0x770} | promašaj (prekid) | |
| 10000 | 0x2710 | {0x2, 0x710} | {0x3039, 0x710} | 0x3039710 |
| 13500 | 0x34BC | {0x3, 0x4BC} | promašaj (prekid) | |
| 20000 | 0x4E20 | {0x4, 0xE20} | {0x673, 0xE20} | 0x673E20 |

Stranice 1 i 3 nisu radnom spremniku pa se adrese 6000 i 13500 ne mogu pretvoriti u fizičke (sklop će izazvati prekid u pokušaju pretvorbe).

### 8.2.2. Straničenje na zahtjev

Što kada proces generira adresu za stranicu koja nije u radnom spremniku?

- kažemo da se dogodio *promašaj* – sklop za upravljanje spremnikom izaziva *prekid*
- u obradi prekida OS dohvaća stranicu s pomoćnog spremnika i stavlja ju u radni




**Slika 8.11. Operacije po promašaju**

*Straničenje na zahtjev* (engl. *demand paging*) je način upravljanja spremnikom kod kojeg se početno samo mali dio procesa učita u radni spremnik, a ostale stranice se učitavaju tek na zahtjev – pri promašajima.

Po dohvatu stranice, instrukcija koja je izazvala promašaj mora se *ponoviti*

- procesor mora imati pomoćne registre koji pohranjuju međurezultate, a koji se mogu odbaciti ako se dogodi promašaj
- npr. neka postoji instrukcija `DIV a,b,d,r` koja cjelobrojno dijeli `a` i `b`, kvocijent sprema u `d`, a ostatak u `r`; ako bi se promašaj dogodio pri spremanju ostatka `r` (u spremnik), niti `d` se ne smije pohraniti, već odbaciti – obzirom da će se nakon dohvata te stranice instrukcija ponoviti, sustav treba dovesti u stanje u kojem je bilo i prije prvog pokretanja instrukcije (npr. `d` je isto što i `a` i/ili `b`)

### 8.2.3. Usporenje rada procesa zbog straničenja

- pretpostavimo da se za pomoćni spremnik koristi tvrdi disk i da nam treba oko 10 ms za dohvat stranice s diska.
- promašaj će usporiti rad procesa, tj. odgoditi njegovo izvođenje za to vrijeme dok se stranica ne dohvati

**Primjer 8.3. Usporenje procesa**

Pretpostavimo da sabirnički ciklus traje $T_B = 10$ ns, te da dohvat stranice s diska traje $T_D = 10$ ms. Ako na svakih $N$ instrukcija (sabirničkih ciklusa) imamo jedan promašaj, koliko će se proces usporiti (u postocima)?

Prosječno trajanje sabirničkog ciklusa može izraziti sa: $\overline{T}_B = \frac{(N-1) \cdot T_B + T_D}{N}$

**Tablica 8.3. Usporenje rada procesa zbog straničenja**

| $N$ | $10^3$ | $10^4$ | $10^5$ | $10^6$ | $10^7$ |
|---|---|---|---|---|---|
| $\overline{T}_B$ | 10,01 $\mu$s | 1,01 $\mu$s | 110 ns | 20 ns | 11 ns |
| $\overline{T}_B/T_B$ | 1001 | 101 | 11 | 2 | 1,1 |

U stvarnosti je usporenje još i manje jer se pri promašaju uglavnom ne dohvaća samo jedna stranica već više njih.

Uz modernije računalo, uz $T_B$ =1 ns i $T_D = 10$ $\mu$s rezultat $\overline{T}_B/T_B = 1,1$ ("prihvatljivo usporenje" od 10%) postiže se za $N = 10^6$.

### 8.2.4. Strategije zamjene stranica

Što ako u obradi promašaja nema praznih okvira kamo bi učitali stranicu?

- treba odabrati jedan okvir i isprazniti ga – KAKO?
- koji se okvir "isplati" isprazniti?
- iskoristiti svojstvo "prostorno-vremenske lokalnosti" procesa
  - sukcesivni zahtjevi procesa prema spremniku su većinom za lokacije bliske prethodnim zahtjevima
    - instrukcije koje se izvode su blizu jedna drugoj, jedna iza druge
    - podaci nad kojima instrukcije nešto rade su većinom također blizu jedni drugima
    - stog je kompaktan
- načini korištenja tog svojstva:
  - korištenje zastavica A i D iz opisnika stranice
    - zastavica A – "nedavno" korištene stranice (njih pokušaj ostaviti)
    - zastavica D – "čiste" i "nečiste" stranice, trošak njihove zamjene nije isti
  - teorijske strategije: FIFO, LRU, LFU, OPT
  - satni algoritam (ono što se koristi)

LRU – Least-Recently-Used

- izbaciti stranicu koja se najdulje nije koristila
- jedina koja je donekle ostvariva i nudi najveću učinkovitost (ne računajući OPT)

### 8.2.5. Prostorno-vremenska lokalnost

- koristiti podatke slijedno, ne "šarati" po spremniku
- smanjuje se broj promašaja
- povećava iskoristivost priručnih spremnika
- dobitak na učinkovitosti može biti vrlo velik (za nekoliko reda veličine!)

**Primjer 8.4. Inicijalizacija velike matrice**

Incijalizacija matrice $A[N][N]$ po recima ili stupcima? Neka jedan redak matrice stane u jednu stranicu i neka na raspolaganju stoji samo jedan okvir za podatke matrice.

```
za i = 1 do N radi
{
    za j = 1 do N radi
    {
        A[i][j] = 0; ili A[j][i] = 0;   !!!
    }
}
```

Inicijalizacija po retcima $= N$ promašaja – nakon promašaja zbog dohvata prvog elementa $A[i][0] = 0$ svi ostali upisi su pogodci ($A[i][j > 0] = 0$). Ovo je ujedno i minimalan broj promašaja – veći broj okvira ne bi pomogao, matricu u konačnici treba dohvatiti (postaviti svugdje nule), a ona je velika $N$ stranica.

Inicijalizacija po stupcima $= N^2$ promašaja – svaki susjedni zahtjev je za različiti redak što je ujedno i različita stranica, svaki stupac generira $N$ promašaja. Broj promašaja se ovdje ne bi smanjio ni kada bismo imali više okvira za matricu (osim kada je broj okvira blizu $N$ ili veći). Naime, nakon što sve okvire iskoristimo za nekoliko redaka, morali bi jedan okvir isprazniti za idući redak. Ali taj redak nam treba kasnije i trebat će ga ponovno dohvaćati – svaki se redak dohvaća $N$ puta!

**Zadatak 8.1. Upravljanje spremnikom**

U nekom sustavu trebaju se obaviti četiri procesa: P1, P2, P3 i P4 koji su već pripremljeni na pomoćnom spremniku i zauzimaju redom 5 MB, 8 MB, 3 MB, 10 MB. Događaji pokretanja i završetka procesa znani su unaprijed i mogu se iskazati sljedećim nizom događaja: P1 pokrenut; P2 pokrenut; P1 završava; P3 pokrenut; P4 pokrenut; P3 završava; P2 završava; P4 završava. Sustav na raspolaganju ima 20 MB spremnika rezervirana za korisničke procese. Prikazati stanje radnog spremnika ako se za upravljanje spremnikom koristi straničenje, uz veličinu stranice od 1 MB.

### 8.2.6. Zaključne napomene o straničenju

Prednosti:

- nema fragmentacije
- zaštita jezgre i procesa
- pokretanje i velikih procesa – učitavaju se samo trenutno potrebne stranice

- podržano sklopovljem i operacijskim sustavom
- transparentno za program (ali dobar program može biti učinkovitiji)
- ostvarenje dijeljenog spremnika između procesa – tablice prevođenja oba procesa pokazuju na iste stranice
- dupliciranje procesa (fork) – nije potrebno fizički kopirati dijelove koji se samo čitaju

Nedostaci:

- potreban je složeni sklop
- moguće usporenje zbog promašaja
  - promašaj može odgoditi izvođenje procesa što u nekim okruženjima može izazvati nedopušteno kašnjenje (npr. u slanju upravljačkih naredbi)

**Što arhitekt/programer treba/može napraviti?**

- teoretski ništa – upravljanje je transparentno – potpuno rješeno sklopovljem i OS-om
- međutim, korištenjem načela "prostorno-vremenske lokalnosti" smanjuje se broj promašaja i povećava učinkovitost
  - vrijedi općenito, ne samo radi straničenja
  - putevi podataka u računalu
    - [ disk ] $\Leftrightarrow$ [ radni spremnik ] $\Leftrightarrow$ [ procesor ]
    - [ disk ]: medij (magnetske ploče i sl.) $\Leftrightarrow$ međuspremnik diska
    - [ procesor ]: priručni spremnik procesora (L3 $\Leftrightarrow$ L2 $\Leftrightarrow$ L1) $\Leftrightarrow$ registri procesora
- korištenje u sustavima za rad u stvarnom vremenu (RT)
  - zaključati stranice kritičnih procesa u radni spremnik (preko sučelja OS-a)
    - `mlock/mlockall`, `VirtualLock`, `SetProcessWorkingSetSize`

**O korištenju pomoćnog spremnika**

- u prikazanome modelu OS pri pokretanju programa priprema proces najprije na pomoćnom spremniku, a onda ga učitava u radni
- u stvarnim sustavima se pomoćni spremnik koristi tek po potrebi
  - na pomoćnom spremniku se ne mora nalaziti proces
  - na pomoćnom spremniku može biti i samo dio procesa (po potrebi)

**Dijeljene biblioteke**

- za obavljanje zadanog posla mnogi programi koriste već pripremljene dijelove koda za pojedine operacije = operacije iz pojedinih *biblioteka*
- biblioteke se ne moraju ugrađivati u datoteku s programom ako su one prisutne (instalirane) na operacijskom sustavu (engl. *shared libraries*)
- dijelovi biblioteka se mogu dinamički učitavati pri pokretanju programa
- dijelovi biblioteka koji sadrže samo kod (instrukcije) mogu se učitati u stranice koje se dijele među procesima koji ih koriste – nije potrebno da se za svaki proces ponovno učitaju ti dijelovi

- primjeri:
  - .dll datoteke na Windows sustavima (engl. *dynamic-link library*)
  - .so datoteke na UUNIX sustavima (engl. *dynamically linked shared object libraries*)

## Pitanja za vježbu 8

1. Kada, iz kojih razloga, procesor pristupa spremniku?
2. Koliko adresnog prostora može adresirati sustav koji koristi 36-bitovnu adresnu sabirnicu?
3. Od čega se sastoji *virtualni spremnički prostor* koji koristi operacijski sustav?
4. Navesti dobra i loša svojstva algoritama:
   - upravljanje spremnikom straničenjem.
5. Što su to fizičke a što logičke adrese?
6. Objasniti pojmove: stranica, okvir, tablica prevođenja u kontekstu straničenja.
7. Čemu služi i od čega se sastoji tablica prevođenja?
8. Što je to “promašaj” u kontekstu straničenja?
9. Opisati upravljanje spremnikom metodom “straničenje na zahtjev”.
10. Što je to “prostorno vremenska lokalnost” i kako ona utječe na učinkovitost sustava?

# 9. DATOTEČNI SUSTAV

Datotečni sustavi (engl. *File Systems – FS*) su uglavnom ostvareni na diskovima, pa se prije razmatranja samih datotečnih sustava razmatraju diskovi.

Iako se koristi pojam *diskovi* u ovu kategoriju su uključeni i spremnici podataka koji nisu diskovi u stvarnosti, npr. SSD disk nije disk iako se tako zove. Osim na diskovima, datotečni sustavi su i na CD-u/DVD/*, USB ključiću, memorijskim karticama.

## 9.1. Diskovi

Uloga diska u računalnom sustavu:

- kao skladište za trajno spremanje podataka (i kada se računalo ugasi)
- kao pomoćni spremnik pri upravljanju spremnikom

### 9.1.1. Svojstva HDD i SSD diskova

- tvrdi disk – HDD (engl. *hard disk drive, hard disk, hard drive*)
- SSD disk – SSD (engl. *solid-state drive*)
- interno su potpuno različiti dok prema van (OS-u) daju isto (slično) sučelje
- u idućem poglavlju je detaljnije razmatran HDD obzirom na njegovu složenu građu
- u ovom su poglavlju samo ukratko navedena svojstva oba tipa diskova

**Svojstva tvrdog diska – HDD**

- diskovi su elektro-mehaničke naprave
- podaci su pohranjeni na magnetiziranim pločama (koje se vrte sa 5000-15000 okr/min)
- ručica s glavama se pomiče po dijagonali i čita/piše s pojedine ploče (površine)
- organizacija podataka:
  1. staza – podaci na jednoj površini jednako udaljeni od osi rotacije – mogu se pročitati/zapisati bez pomaka glave (samo s rotacijom ploče)
  2. sektor – staza se dijeli na manje jedinice – sektore, tipično velike 512 B (ili 4 KB za diskove za spremanje podataka, video nadzor i slično)
  3. cilindar – staze na različitim pločama jedako udaljene od osi rotacije – mogu se dohvatiti bez pomicanja glave, samo aktivacijom druge glave za drugu površinu

  – jedinica podataka je sektor

  – "adresa sektora": {broj površine (glave), broj staze na površini, broj sektora na stazi}

  – izvorno CHS *Cylinder-Head-Sector* (C=broj staze, H=broj površine)

- disk je spor zbog mehaničkih djelova
- ublažavanje sporoće – podatke kompaktno smjestiti, dok se ne pohrani sva datoteka koristiti

redom:

1. uzastopne ili sve sektore iste staze
2. ostale staze istog cilindra (bez pomaka glave)
3. susjedne cilindre (što manji pomak glave)

  - OS nastoji tako smjestiti datoteke, ali kad je disk dosta popunjen to više nije moguće i javlja se fragmentacija: različiti dijelovi datoteke su na različitim mjestima diska – čitanje/pisanje se tada usporava
  - defragmentacija nastoji premjestiti sadržaje tako da budu kompaktnije smješteni
  - elektronika diska "skriva" internu arhitekturu (broj površina, staza, sektora po stazi) i prema OS-u sve sektore prikazuje kao linearno polje sektora
  - u takvom linearnom polju susjedni sektori jesu i susjedni na disku (osim iznimno kad se prelazi na stazu na drugom cilindru ili susjednu stazu, što je rijetko)

- OS poslužuje skupinu zahtjeva i optimira njihovo posluživanje radi manjeg pomaka glave
- uobičajena svojstva diskova:
  - kapaciteti: od nekoliko stotina GB do desetke TB (nekoliko TB je uobičajeno)
  - brzine čitanja/pisanja kompaktno smještenih podataka: 100-200 MB/s
  - čitanje/pisanje nasumičnog bloka: 5-15 ms !
  - priključak SATA (Serial ATA; ATA: AT Attachment; AT: Advanced Technology (u Ĩ986.))
  - komunikacija procesor – disk kontroler: AHCI *(Advanced Host Controller Interface)*
    - optimiran za rad s diskovima preko SATA ožičenja
    - half-duplex – prijenos samo u jednom smjeru u jednom trenutku
- OS, tj. datotečni sustavi obično rade s *blokom* podataka koji se sastoji od susjednih sektora (uobičajeni blok je 4 KB – osam uzastopnih sektora od 512 B)

**Svojstva SSD-a**

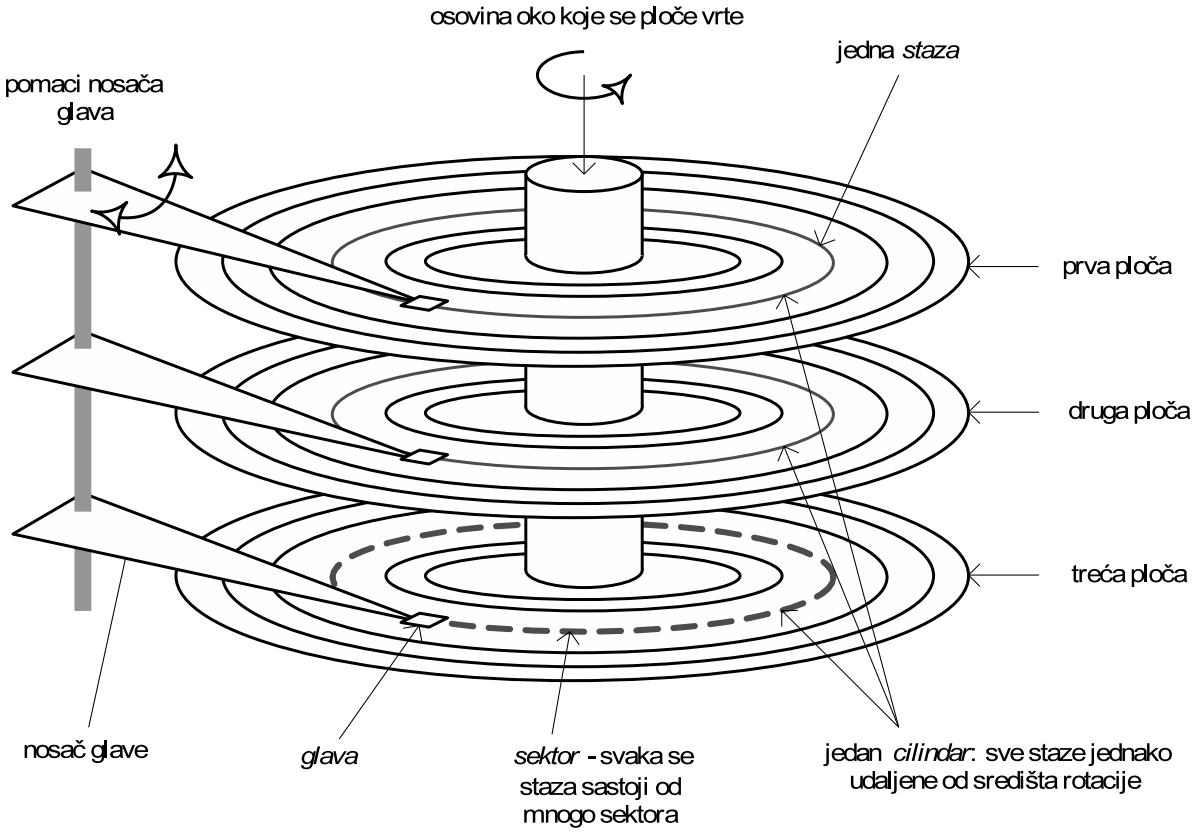
- poluvodički "diskovi" – koriste se poluvodičke ćelije za spremanje naboja
- ćelija: može ovisno o tehnologiji spremiti od 1 do 4 bita
  - 1-bit: SLC–*Single Level Cells* – najbolji ali i najskuplji (za poslužitelje)
  - 2-3 bita: MLC–*Multi Level Cells* – jefitinji (za obične korisnike)
  - 4-bita: QLC–*Quad-bit Cells* – jefitinji (za obične korisnike)
  - 3D XPoint: novija tehnologija – umjesto naboja "sprema" otpor (postavlja otpor)
- jedinica podataka je sektor/blok (kao i za HDD)
- adresiranje je linearno – nema površina, staza, sektora
- dohvat je efikasniji ako se radi u blokovima
  - dohvat jednog nasumičnog sektora može trajati i do 0,1 ms
  - dohvat susjednih sektora (već dohvaćenom) je puno brži
  - stoga se dohvaća blok, a ne pojedinačini sektor

  - npr. za dohvat nasumičnog bloka od 4 KB (8 sektora) trajanje je neznatno veće od dohvata jednog nasumičnog sektora
- pisanje zahtjeva dvije operacije: brisanje ćelije, pa tek onda pisanje
- pisanje nije sporije ako se radi u blokovima
- mješoviti zahtjevi čitanje/pisanje za nasumičnim blokovima znatno usporavaju rad
- uobičajena svojstva SSD-ova:
  - kapaciteti: od nekoliko stotina GB do nekoliko TB
  - brzine čitanja/pisanja kompaktno smještenih podataka:
    * preko SATA sučelja (AHCI): 200-550 MB/s
    * preko M.2 ili PCIe (NVMe): 1000 do 7000 MB/s
  - čitanje/pisanje nasumičnog bloka: 0,1 ms ! znatno sporije od gornjih brzina kad se čitaju/pišu nasumični mali dijelovi (<4KB)
  - komunikacija procesor – disk kontroler: NVMe *(Non-Volatile Memory Host Controller Interface Specification)*
    * optimiran za rad s SSD-ovima preko M.2 i PCIe
    * full-duplex – prijenos u oba smjera istovremeno
    * više redova za zahtjeve od AHCI-ja

### 9.1.2. Fizička svojstva tvrdih diskova (HDD) (info)

HDD je elektromehanička naprava. Sastoji se od:

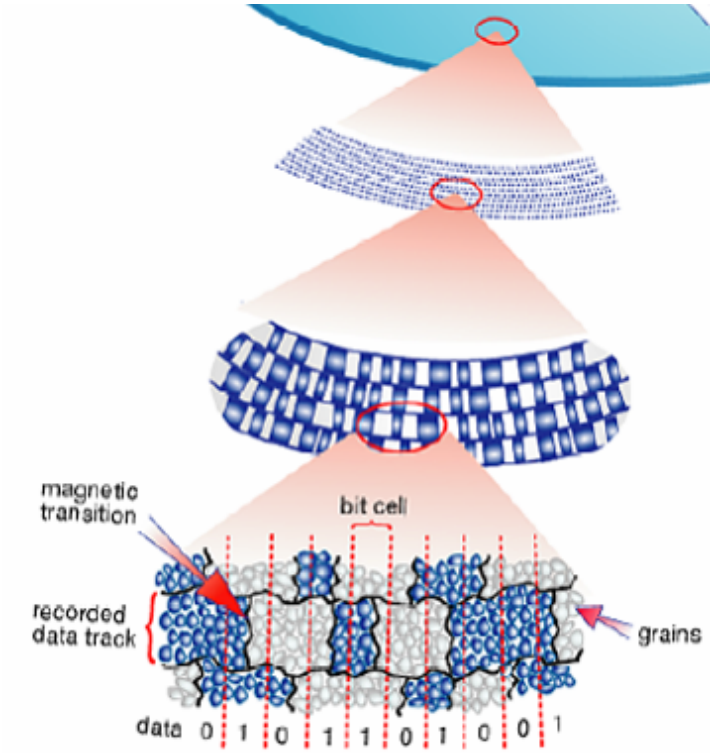
- pokretnih djelova:
  - magnetiziranih ploča
  - pokretnih glava
  - elektromotora koji pokreću ploče (vrte ih)
  - elektromotora koji pokreću glave (od ruba prema centru rotacije i obratno)
- nepokretnog dijela: upravljački sklop




**Slika 9.1. Shematski prikaz mehaničkog dijela diska**

### Magnetska ploča pod povećalom (info)

Podaci se na magnetskim pločama pohranjuju korištenjem različite polarizacije vrlo malih površina ploča. Najjednostavniji pristup bi bio da je jedinica predstavljena jednim smjerom a nula drugim. Međutim, diskovi koriste različite načine kodiranja. Slika 9.2. prikazuje jedan takav primjer kod kojeg je jedinica predstavljena promjenom stanja.




**Slika 9.2. Magnetski materijal pod povećalom**[1]

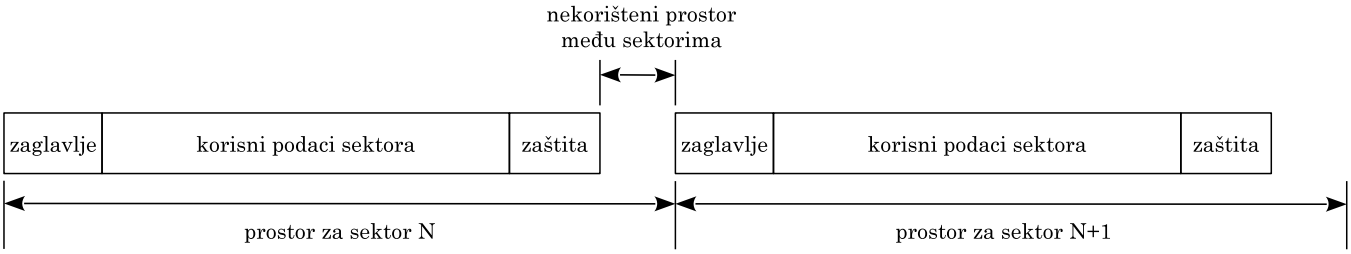
[1]Izvori: http://www.cs.virginia.edu/~gurumurthi/courses/HDD_Basics.ppt,

### Čitanje, pisanje, dodatni i zaštitni bitovi (info)

Podaci se čitaju preko glave koja je zapravo jednostavan strujni krug. Pri čitanju, svaka će magnetizirana površina inducirati napon koji se onda može interpretirati kao nula ili jedinica. Međutim, zbog velike gustoće zapisa, očitani signal ipak nije lijepi "pravokutni" već ga treba obraditi da bi dobili stvarno stanje (susjedni bitovi donekle utječu na očitanje trenutne "male površine". Nadalje, s obzirom na te i druge specifičnosti, koriste se posebni kodovi (npr. RLL) za zapis podataka.

Pisanje se ostvaruje tako da se kroz glavu propusti struja određena smjera u trenutku kada je glava točno iznad željenog područja. Tada će se magnetski materijal ispod glave polarizirati u željenome smjeru (i time zapisati željeni sadržaj).

Jedna staza se sastoji od mnogo sektora. Osim korisnih podataka pohranjenih u sektoru, uz sam sektor moraju biti i dodatni podaci. Primjerice, zaglavlje koje uključuje identifikaciju sektora, identifikaciju staze, podnožje sa zaštitnim bitovima (da se može detektirati i možda ispraviti greška u čitanju). Također se između dva sektora može ostaviti i malo prazna prostora ... Proizvođači diskova nastoje smanjiti dodatno trošenje prostora na ostale podatke tako da mogu pohraniti više korisnih podataka.




**Slika 9.3. Sektor i popratni podaci na stazi**

S obzirom na to da je staza kružnica te da su kružnice dalje od centralne osi rotacije dužeg opsega, na njima je moguće pohraniti više podataka (bitova) nego na onim stazama bliže centru. Stoga se na udaljenijim stazama nalazi i veći broj sektora (te je i čitanje podataka brže s obzirom na konstantnu brzinu vrtnje).

U nastavku razmatranja diskova (prvenstveno u zadacima), koristit će se jednostavan model diska kod kojeg se zanemaruju ostali podaci na stazi, osim korisnih bitova pripadnih sektora. Također, pretpostavit će se da svaka staza ima jednaki broj sektora.

### Podatkovna jedinica koju disk daje na zahtjev je “sektor”

Adresa sektora (“fizička adresa”) (engl. *cylinder-head-sector – CHS*) sastoji se od:

1. rednog broja površine
2. rednog broja staze
3. rednog broja sektora

Upravljački sklop diskovne jedinice

- upravlja mehaničkim i elektroničkim dijelovima
- ima procesor (očitani signali nisu lijepi uglati)
- spaja se na sabirnicu

http://www.hitachigst.com/hdd/research/storage/pm/index.html

- ima međuspremnik
- pretvara “linearnu” ili “logičku” adresu u CHS i obratno

Logička adresa sektora (engl. *logical block addresing – LBA*)

- svi sektori su predstavljeni kao jedno polje
- jednostavniji prikaz i upravljanje diskom za OS
- pretvaranje adresa radi sklop diskovne jedinice

Svojstva današnjih diskova (okvirne vrijednosti)

- gustoća zapisa do 1,34 Tbita/in$^2$ ($\approx$2 Gbita/mm$^2$)
- kapaciteti: od $\approx$ 100 GB do 20 TB
- promjeri ploča: 3,5"; 2,5"; 1,8"; 1"
- brzine okretanja: 5000 do 15000 okr/min; 5400; 5900; 7200, 10000
- brzina prijenosa (kompaktno smještenih podataka): 100 – 200 MB/s
- vrijeme pristupa (od zahtjeva do posluživanja): 2 do 15 ms ($\approx$ 8 ms)
  - prosječno pomicanje glave za “slučajni zahtjev”
  - računa se kao pomicanje glave za 1/3 staze
- veličine sektora: 512 B, 4 kB

### 9.1.3. Vremenska svojstva diskova

Obzirom na mehaničke dijelove disk je jako spor. Stoga se kroz optimiranje smještaja podataka i zahtjeva za pristup pokušava ubrzati njegovo korištenje.

Datoteke se stoga nastoje pohraniti u uzastopne sektore i staze, da se pri njenom korištenju što manje pomiče glava. Ako datoteke nisu ovako kompaktno smještene (disk je "fragmentiran"), učinkovitost rada s tim datotekama je značajno manja.

## 9.2. Datotečni sustav

CHS $\Rightarrow$ LBA

- adresiranje korištenjem numeracije staza/ploča/sektora (engl. *cylinder-head-sector – CHS*) je kompliciran na novijim diskovima (različiti broj sektora na stazama i sl.)
- noviji diskovi (tj. svi) najčešće i ne nude (prave) informacije o broju staza/ploča/sektora
- oni nude sučelje za korištenje "polja sektora" (engl. *logical block addressing – LBA*)
- sektori su dostupni preko samo jednog broja = rednog broja sektora.
- elektronika pretvara LBA $\Leftrightarrow$ CHS

Blok (nakupina sektora, klaster)

- veličina sektora je svojstvo diska (ne može se mijenjati npr. formatiranjem)
- veličine sektora: 512 B (uobičajeno), 4 KB (noviji diskovi)
- datotečni sustav definira novu jedinicu podataka = blok (engl. *cluster*)
- blok je niz uzastopnih sektora (nakupina sektora)
- niz čini 1 ili 2 ili 4 . . . ili $2^n$ uzastopnih sektora
- što je blok veći potrebna struktura podataka za opis je manja, ali je gubitak zbog fragmentacije veći

### 9.2.1. Datoteke

- Podaci na disku su organizirani u datoteke
- Datoteka: skup povezanih informacija koje čine cjelinu (logičke tvorevine)
- Datoteke obično sadrže:
  - programe, npr.: .exe; .out; .bat; .sh; .dll; .so; .jar; . . .
  - podatke, npr.
    - dokumente (word, tekstualne datoteke, HTML, . . . )
    - multimediju (slike, video, muziku, . . . )
    - postavke programa i sustava
  - ostalo
    - privatne podatke OS-a (dijelove pomoćnog spremnika)
    - privatne podatke datotečnog sustava
- formati datoteka:
  - binarna (.exe, .doc, .dll, .zip, .mp3)
  - tekstualna (.txt, .html, .pls, .srt, .bat) => ASCII ili sličan format (npr. UTF-8)
- OS se učitava iz datoteka!
- Sve mora biti na disku u datotekama (osim za ugrađene sustave koji ne moraju imati disk)

### 9.2.2. Datotečni sustav

Pojam "datotečni sustav" koristimo:

- za oznaku tipa datotečnog sustava (NTFS, FAT, UDF, ISO 9660, . . . )
- za dio operacijskog sustava – točnije bi bilo reći "datotečni podsustav"
- za neki konkretni datotečni sustav (na primjeru ili stvarnom računalu, "na tom disku")

Iz konteksta je uvijek jasno na što se odnosi pojam.

Datotečni sustav daje odgovore na pitanja:

- kako su datoteke smještene na disku
  - fizičko smještanje: gdje, u kojim sektorima/blokovima, kojim redoslijedom
  - logičko smještanje: kako su datoteke organizirane, grupirane, kako im se pristupa, pronalazi, . . . ?
  - atributi: tko im smije pristupiti, sigurnost, učinkovito korištenje diska (fragmentacija), . . . ?
- koji dijelovi diska su slobodni

Datotečni sustav definira kako smjestiti podatke na disk i kako do njih doći

Disk se može i podijeliti u više particija (svezaka)

- svaka particija je zasebni datotečni sustav
  - npr. part1: blokovi 0-10000, part2: blokovi 10001-20000

Datotečna tablica (engl. *file table*)

- tablica sadrži:
  - podatke koji definiraju disk, slobodni prostor (ponekad su ove informacije u zasebnim strukturama izvan tablice)
  - opisnike datoteka
- svaka datoteka ima svoj opisnik u datotečnoj tablici
- datoteke se nastoje spremiti u kontinuirani dio
  - smanjenje fragmentacije – datoteka se brže učitava

OS koristi datotečni sustav – preko datotečnog podsustava – za operacije:

- stvori, obriši, preimenuj, premjesti datoteku ili direktorij
- otvori datoteku, čitaj, piši, pomakni kazaljku, . . .

### 9.2.3. Opisnik datoteke

- svaka datoteka ima svoj opisnik u datotečnoj tablici
- osnovni dijelovi opisnika:
  - ime datoteke
  - direktorij gdje je datoteka smještena (u logičkoj org. diska)
  - tip datoteke

  - veličina datoteke
  - vrijeme stvaranja, zadnje promjene, zadnjeg korištenja
  - podaci o “vlasniku” (kojem korisniku pripada), prava pristupa
  - ...
  - opis smještaja na disku (u kojim blokovima)

### 9.2.4. Direktoriji

- datoteke su logički organizirane preko stabla direktorija
- direktoriji su logička tvorevina – povezuju datoteke iste namjene, istog korisnika i slično
- Windows pristup:
  - svaka particija ima svoje ime (C:, D:, E:, ...)
  - particija na kojoj je OS (načešće C:) naziva se sustavska
  - uobičajeni direktoriji i njihov sadržaj:
    * `C:\Windows\` – operacijski sustav
    * `C:\Program Files\` – programi
    * `C:\Program Files (x86)\` – 32-bitni programi na 64-bitovnom OS-u
    * `C:\ProgramData` – postavke programa
    * `C:\Users\korisničko_ime` – korisnički direktoriji, postavke i podaci
    * `C:\pagefile.sys` – pomoćni spremnik za straničenje
  - druge particije, CD/DVD i sl.: svaki ima svoju oznaku (D:, E:, ...)
- UNIX pristup:
  - `/` – početni direktorij (korijen, *root*)
  - `/home/korisničko_ime` – korisnički direktoriji (postavke i podaci)
  - `/etc/` – većina postavki sustava
  - `/bin/`, `/sbin/`, `/usr/bin/`, `/usr/local/bin/` – OS i programi
  - i još puno njih sa svojim posebnim funkcijama
  - pogledati: http://en.wikipedia.org/wiki/Unix_filesystem
  - particije:
    * tipično (najjednostavnije)
      · jedna particija za `/` (i sve na njoj)
      · jedna particija za `swap` (pomoćni spremnik za straničenje, opcionalno)
    * “naprednije” postavke s više particija, npr.:
      · jedna particija za `/home`
      · jedna particija za `/boot`
      · jedna particija za `/` (sve ostalo)

        - jedna particija za `swap` (pomoćni spremnik za straničenje)
    - druge particije, CD/DVD i sl.:
        - spajaju se na neku "točku" datotečnog sustava (engl. *mount point*)

Na jednoj particiji (jednom datotečnom sustavu) nalaze se blokovi sa sadržajima:

- "opisnik" particije (veličina i broj blokova, ...)
- datotečna tablica (opisnici datoteka i direktorija)
- opisnici slobodnog prostora
- blokovi sa sadržajem datoteka
- slobodni blokovi

## 9.3. Primjeri datotečnih sustava

### 9.3.1. NTFS

- NTFS – skraćenica od *New Technology File System*
- NTFS sadrži datotečnu tablicu koja se zove *Master File Table* – MFT (*glavna tablica datoteka*)
  - svaka datoteka ima opisnik u MFT, pa i sama MFT
  - u opisniku se nalaze podaci o datoteci
- numeriranje blokova u NTFS-u:
  - LCN – Linear Cluster Number
    - logička adresa bloka na particiji
    - particija se dijeli u blokove, linearno numerirane, počevši s LCN=0
  - VCN – Virtual Cluster Number
    - logička adresa bloka datoteke
    - svaka se datoteka sastoji od skupine blokova (osim onih vrlo malih, čiji je sadržaj pohranjen u samom opisniku)
    - VCN predstavlja adresu bloka unutar datoteke
      - prvi dio datoteke je u bloku s VCN=0, drugi u VCN=1, itd.
  - Povezivanje VCN-a u LCN definirano je u opisniku datoteke
  - datoteka koja je kompaktno smještena na disku ima samo jedan zapis u tablici (gdje je prvi blok i koliko ih ukupno ima)
- jako male datoteke (manje od ∼900 B) pohranjuju se u sam opisnik, ne zauzimaju dodatne blokove na disku

## Primjer 9.1. Datoteka na disku

Zadana je datoteka sa sadržajem i prikazom svih blokova na disku.

**Tablica 9.1. Datoteka – logički prikaz**

| blok | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| sadržaj | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |

**Tablica 9.2. Datotečni sustav (blokovi particije)**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 17 | 18 C | 19 D | 20 E | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
| 33 | 34 | 35 | 36 | 37 A | 38 B | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 K | 52 L | 53 M | 54 N | 55 O | 56 |
| 57 F | 58 G | 59 H | 60 I | 61 J | 62 | 63 | 64 |

## Primjer 9.2. NTFS

Opis smještaja datoteke za primjer 9.1. prema NTFS pravilima:

| VCN | LCN | # |
|---|---|---|
| 1 | 37 | 2 |
| 3 | 18 | 3 |
| 6 | 57 | 5 |
| 11 | 51 | 5 |

# – broj uzastopnih blokova – nakupina blokova

Prvi blok datoteke (VCN=1) nalazi se u 37. bloku particije (LCN=37). S obzirom na to da datotečni sustav nastoji datoteke održati kompaktnima, jedan red tablice može opisati i više **uzastopnih** blokova. Koliko ih opisuje kazuje nam zadnji stupac. Prvi red tako opisuje blok 1 i blok 2, s time da se blok 2 (VCN=2) nalazi u bloku LCN=38.

**Primjer 9.3. NTFS (2)**

Neka datoteka pohranjena je kompaktno po dijelovima u blokovima:

1. $526 - 587$
2. $124 - 225$
3. $432 - 449$.

Prikazati sadržaj dijela opisnika te datoteke koji opisuje njen smještaj, ako se radi o datotečnom sustavu NTFS.

Rješenje:

1. dio: $526 - 578 \Rightarrow 578 - 526 + 1 = 62$ bloka (uključen je i $526.$ i $578.$!)
2. dio: $124 - 225 \Rightarrow 225 - 124 + 1 = 102$ bloka
3. dio: $432 - 449 \Rightarrow 449 - 432 + 1 = 18$ blokova

| VCN | LCN | # |
|---|---|---|
| 1 | 526 | 62 |
| 63 | 124 | 102 |
| 165 | 432 | 18 |

## Zadatak 9.1. NTFS (3)

Neka datoteka je smještena na disku po dijelovima: prvi MB je kompaktno smješten počevši od 725. bloka diska, druga dva MB su kompaktno smještena počevši od bloka 2001. Ako je veličina bloka 4 KB, prikazati dio sadržaj opisnika datoteke, koji opisuje smještaj datoteke na disku. U kojem se bloku na disku nalazi bajt s adresom 2000000 unutar datoteke?

Rješenje:

Zad. 9.6. NTFS

prvi MB => 725. bloka diska

druga dva MB => 2001.

vel. bloka = 4 KB

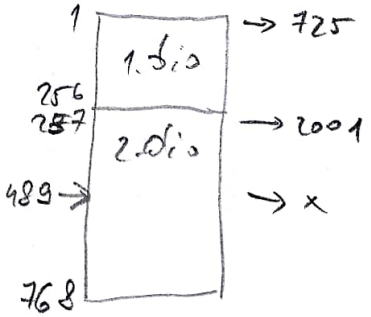
| VCN | LCN | # |
|---|---|---|
| 1 | 725 | 256 |
| 257 | 2001 | 512 |

$\frac{1\,MB}{4\,KB} = \frac{1024 \cdot KB}{4 \cdot KB} = 256$ blokova

2 MB => 512 blokova

$\frac{2000000}{4\,KB} = \frac{2000000}{4096} = 488{,}28$ => bajt 2000000 je u bloku dat 489

gdje je na disku blok 489 ?




489 - 257 = x - 2001

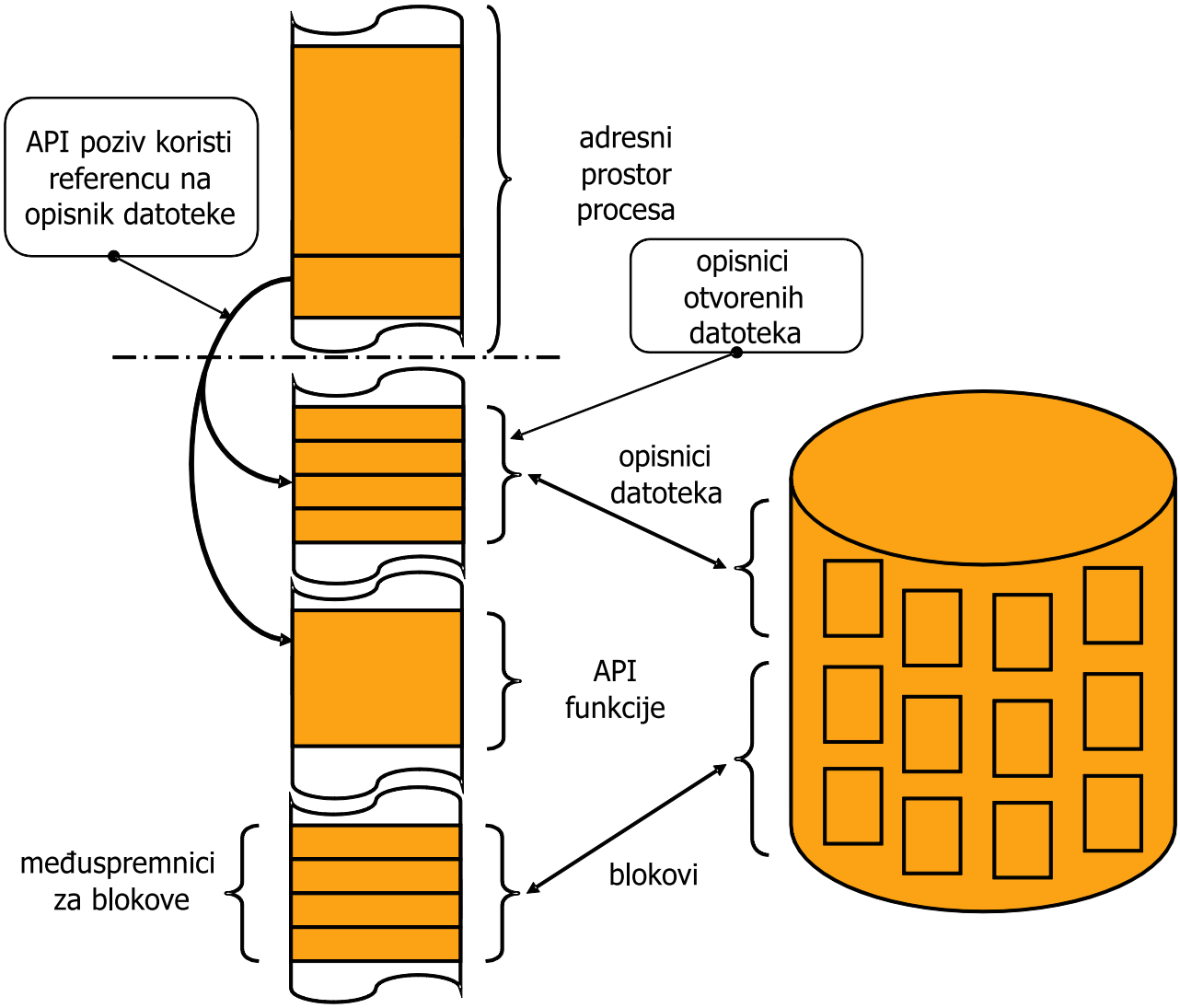
x = 2001 + 489 - 257

= 2001 + 222

= 2223

# 9.4. Datotečni podsustav operacijskog sustava

- Operacijski sustav treba omogućiti korištenje datotečnog sustava – kroz datotečni podsustav
- OS zato:
  - kopira datotečnu tablicu u radni spremnik (ili samo dijelove koje koristi)
  - za svaku datoteku koja se koristi stvara kopiju opisnika i proširuje ga:
    * kazaljkom
    * međuspremnicima (za brži rad)
    * ...
- Pri radu s datotekama (čitanje/zapisivanje) koristi se datotečna kazaljka (engl. *file pointer*)
  - prilikom "otvaranja datoteke" kazaljka pokazuje na početak datoteke
  - čitanjem i pisanjem se kazaljka pomiče prema naprijed




- Korištenje datoteka
  - OS pruža sučelje za korištenje datoteka
  - uobičajena sučelja uključuju:
    * `int open (char *filename, int access, int perm);`
    * `int close (int handle);`

```
* int read (int handle, void *buffer, int nbyte);
* int write (int handle, void *buffer, int nbyte);
* int lseek (int fildes, int offset, int whence);
```

Primjer programa:

```
include <stdio.h>

int main()
{
    FILE *fp;

    fp = fopen("hello.txt", "w");
    fwrite("Hello world!\n", 13, 1, fp);
    fclose(fp);

    return 0;
}
```

## 9.5. Uloga međuspremnika u povećanju učinkovitosti (sažetak)

Korištenje međuspremnika (engl. *cache*) u povećanju učinkovitosti (sažetak)

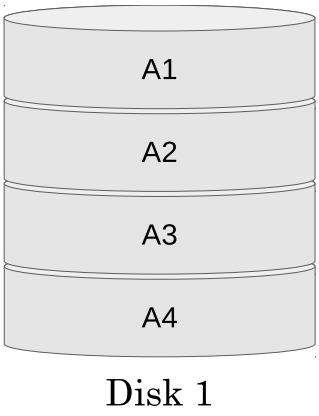
- ideja: korištenjem međuspremnika postići brzinu pristupa jednaki najbržoj komponenti spremnika (npr. brzini L1 međuspremnika procesora!)
- [disk: mag. ploče⇔međuspremnik] $\Longleftrightarrow$ [spremnik] $\Longleftrightarrow$ [procesor: L3⇔L2⇔L1⇔registri]
- straničenje:
  - TLB (pamti opisnike zadnjih korištenih stranica)
  - u radnom spremniku samo “potrebne” stranice
- korištenje diska (i drugih UI naprava)
  - čitanje malo više podataka od traženih (susjedni blokovi će možda trebati)
  - zadržavanje korištenih blokova jedno vrijeme (možda će opet trebati)
- uloga “arhitekta” programske potpore je značajna
  - hijerarhijska organizacija spremnika MOŽE biti vrlo učinkovito iskorištena (idealno se efektivna brzina svodi na najbrži spremnik = L1; tome se možemo jako približiti!)
  - isto tako loše posloženi sustavi mogu biti vrlo spori (i na najbržem sklopovlju) upravo zbog lošeg korištenja priručnih spremnika, tj. zbog "šaranja" po spremniku

## 9.6. Zalihost u višediskovnim sustavima

### 9.6.1. Potreba za višediskovnim sustavima

- veći kapaciteti
- veće brzine čitanja i/ili pisanja
- zaštita u slučaju kvara jednog ili više diskova

U sustavima sa samo jednim diskom, logička organizacija jedinica podataka, nazovimo ih blokovima, jednaka je fizičkoj – susjedni blokovi su susjedni i na disku. Slika 9.4. prikazuje takav disk i numeraciju blokova (termin bloka ovdje ne mora biti povezan s istim terminom korištenim u opisu datotečna sustava, gdje je on označavao nakupinu sektora – *claster*).




**Slika 9.4. Organizacija podataka na jednom disku**

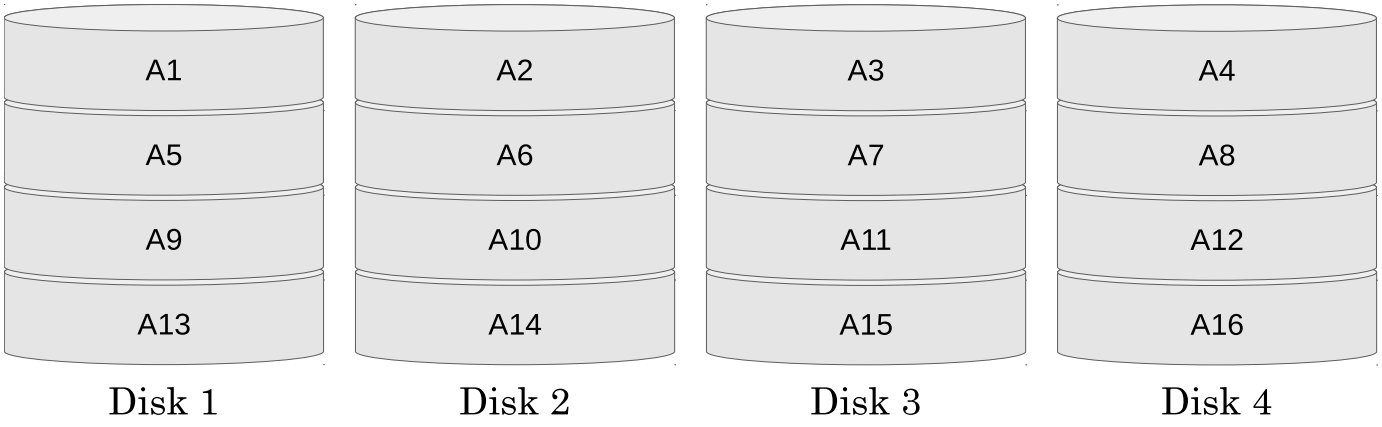
Višediskovni sustavi mogu se koristiti na način da svaki disk ostaje zasebna logička cjelina ili da više diskova zajedno čini logičku cjelinu (npr. da se sadržaj jednog bloka raspodijeli na sve diskove).

Uobičajene (standardne) višediskovne konfiguracije (podržane i sklopovljem i operacijskim sustavima) nose prefix RAID

- RAID – višediskovni zalihosni spremnici (engl. *redundant array of independent disks*)
- prednosti (nekih konfiguracija) RAID-a:
  - veća učinkovitost istovremenim korištenjem (radom) više diskova
  - veća pouzdanost dodavanjem zaštitnih bitova (za oporavak u slučaju kvara)
- uobičajeni sustavi: RAID 0, RAID 1, RAID 5, RAID 6, RAID 0+1, RAID 10, RAID 51, ...

### 9.6.2. RAID 0 – sustav bez zalihe

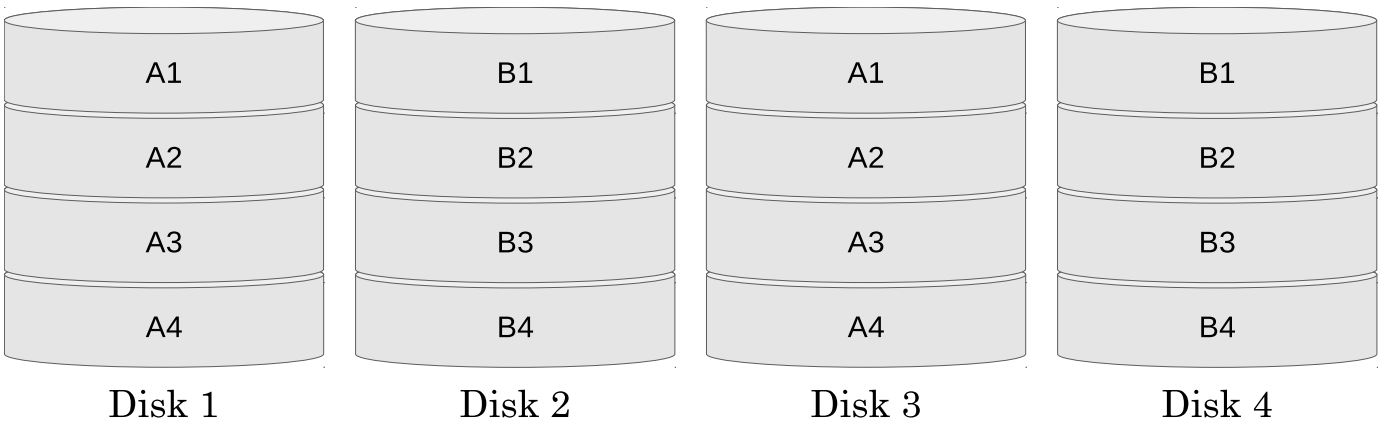
- na različitim diskovima nalaze se različiti dijelovi bloka
- nema dodatne zaštite – ako se bilo koji disk pokvari podaci su izgubljeni
- kapacitet sustava jednak je $N \cdot M$, gdje je $N$ ukupan broj diskova, a $M$ kapacitet pojedinog diska
- veće performanse pri čitanju/pisanju – diskovi paralelno rade nad uzastopnim blokovima




**Slika 9.5. Primjer RAID 0 sustava**

### 9.6.3. RAID 1 – zrcaljenje podataka

- podaci se dupliciraju, svaki disk ima svoju kopiju/par na drugom disku
- svaki par diskova ostaje zasebna logička cjelina
- kapacitet sustava jednak je $N/2 \cdot M$, gdje je $N$ ukupan broj diskova, a $M$ kapacitet pojedinog diska
- dopušteni su kvarovi i više diskova, ako nisu od istog para (inače samo jedan)
- u slučaju kvara, pri zamjeni podaci se mogu obnoviti od uparenog diska
- veće performanse pri čitanju – uzastopni blokovi se mogu čitati s različitih diskova paralelno
- uobičajeno se RAID 1 koristi u kombinaciji s RAID 0 (RAID 0 diskovi se zrcale)



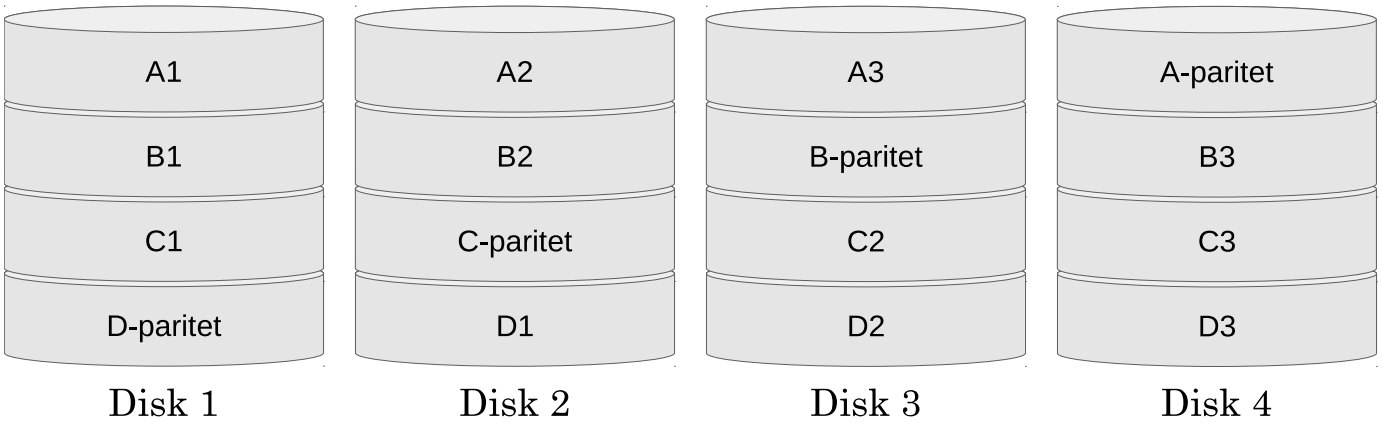

**Slika 9.6. Primjer RAID 1 sustava**

### 9.6.4. RAID 5 – raspodijeljeni paritet

- podaci su raspodijeljeni na diskovima uz dodatak paritetne zaštite
- svaki blok podataka podijeli se na $N - 1$ dijelova te se za njih računa paritetna zaštita te tako pohrani raspodijeljeno na svih $N$ diskova
- paritet je za svaki idući blok na drugom disku – kad bi odvojili jedan disk samo za paritet, onda bi taj disk postao usko grlo – svaki zapis na bilo koji disk bi zahtijevao i zapis izmijenjenog pariteta na paritetni disk
- kapacitet sustava jednak je $(N - 1) \cdot M$, gdje je $N$ ukupan broj diskova, a $M$ kapacitet

pojedinog diska

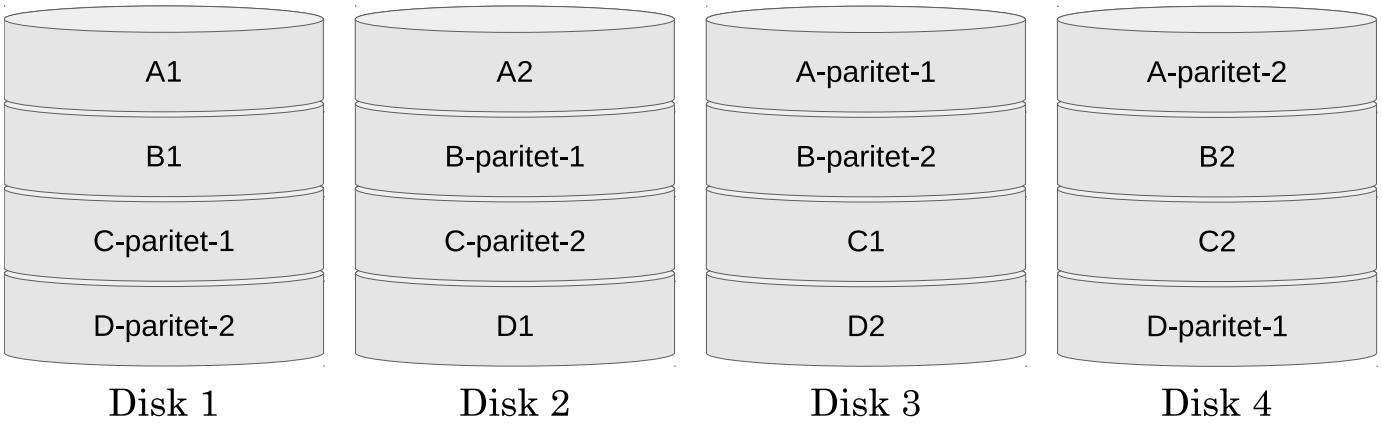
- dopušten je kvar jednog diska
- u slučaju kvara, pri zamjeni podaci se mogu obnoviti od ostalih diskova (svi sudjeluju u obnovi)
- veće performanse pri čitanju/pisanju – diskovi paralelno rade nad uzastopnim blokovima




**Slika 9.7. Primjer RAID 5 sustava**

### 9.6.5. RAID 6 – dvostruki paritet

- sličan RAID 5 sustavu, uz dodatni blok pariteta (kodiranje prema Reed–Solomonu)
- paritetna zaštita je uvijek na dva različita diska (raspodijeljeno po svim diskovima za svaki idući blok)
- kapacitet sustava jednak je $(N-2)\cdot M$, gdje je $N$ ukupan broj diskova, a $M$ kapacitet pojedinog diska
- dopušten je kvar dva diska
- u slučaju kvara, pri zamjeni pokvarenog diska s ispravnim, podaci se mogu obnoviti od ostalih diskova
- veće performanse pri čitanju/pisanju – diskovi paralelno rade nad uzastopnim blokovima




**Slika 9.8. Primjer RAID 6 sustava**

Osim navedenih postoje i kombinacije: RAID 0+1, RAID 1+0 (RAID 10), RAID 51, ... koje u prvoj razini imaju jednu organizaciju, a u drugoj drugu. Primjerice RAID 0+1 u prvoj razini

ima RAID 0, a u drugoj zrcali te podatke. Broj diskova u razinama može biti različit ako su različita kapaciteta. Npr. u prvoj razini može biti četiri diska po 3 TB a u drugoj tri diska po 4 TB – zrcale se podaci ne diskovi.

Tablica 9.3. uspoređuje navedene sustave prema nekim svojstvima.

**Tablica 9.3. Usporedba RAID sustava koji ima ukupno $N$ diskova svaki kapaciteta $M$**

| svojstvo | RAID 0 | RAID 1 | RAID 5 | RAID 6 |
|---|---|---|---|---|
| kapacitet za podatke | $N \cdot M$ | $N/2 \cdot M$ | $(N-1) \cdot M$ | $(N-2) \cdot M$ |
| iskoristivost spremnika | 100 % | 50 % | $(N-1)/N \cdot$ 100 % | $(N-2)/N \cdot$ 100 % |
| minimalni broj diskova | 2 | 2 | 3 | 4 |
| dopušteni broj diskova u kvaru | 0 | 1+ | 1 | 2 |

RAID sustav je moguće ostvariti uz potporu dodatnih sklopova (RAID upravljača) ili uz pomoć operacijskog sustava (programski RAID).

### 9.6.6. Vrijeme rada sustava do kvara (info)

U sustavima s većim brojem diskova veća je i vjerojatnost kvara. U tom konteksu spominju se pojmovi:

- $MTTF$ – srednje vrijeme rada diska do kvara (engl. *mean time to failure*)
- $MTTR$ – srednje vrijeme trajanja popravka kvara ili zamjene diska (engl. *mean time to repair*)
- $MTTF_S$ – srednje vrijeme rada sustava do kvara (gubitka podataka)

U sustavu s $N$ diskova istih svojstava i s istim $MTTF$, $N$ puta je veća vjerojatnost kvara te je vrijeme do kvara jednog (bilo kojeg) diska $N$ puta manje. Za sustave kod kojih i kvar samo jednog diska onemogućava rad sustava (npr. RAID 0), vrijeme do kvara sustava se stoga može izraziti formulom (9.1.).

$$MTTF_S = \frac{1}{N} \cdot MTTF \tag{9.1.}$$

Primjerice, za sustav od 5 diskova koji svaki imaju $MTTF = 5$ godina, očekivano vrijeme rada bez ijednog kvara je $MTTF_S = 1/5 \cdot 5$ godina $= 1$ godina.

Ako kvar jednog diska nije kritičan (npr. RAID 5) te se neispravni disk zamijeni u prosječno $MTTR$ vremena, tada je očekivano vrijeme do kvara sustava (kad se ipak preklope dva kvara diskova) zadano formulom (9.3.).

$$MTTF_S = \frac{1}{N(N-1)} \cdot \frac{MTTF^2}{MTTR} + \frac{2N-1}{N(N-1)} \cdot MTTF \tag{9.2.}$$

Sustav kod kojeg se jedan disk može zamijeniti ispravnim može raditi znatno duže od očekivanog rada pojedinog diska. Kritično je vrijeme potrebno za zamjenu, jer ako se u tom intervalu dogodi kvar još jednog diska tada se podaci gube (sustav se od dvostruka kvara ne može oporaviti).

Primjerice, za sustav od 5 diskova koji svaki imaju $MTTF = 5$ godina, uz vrijeme popravka $MTTR = 7$ dana, očekivano vrijeme rada bez dvostruka kvara je:

$$MTTF_S = \frac{1}{5(5-1)} \cdot \frac{5^2}{7/365} + \frac{2 \cdot 5 - 1}{5(5-1)} \cdot 5 = 65,18 + 2,25 = 67,43 \text{ godina} \tag{9.3.}$$

Kada bi vrijeme popravka bilo 15 dana, $MTTF_S$ je skoro dvostruko kraće.

## Pitanja za vježbu 9

1. Od kojih se komponenata sastoji disk (HDD)?
2. Kako se na disk pohranjuju podaci? Koji se materijali i principi koriste?
3. Kako se iskazuje adresa jednog sektora (kako se sektor jedinstveno identificira)?
4. Što je to “cilindar”?
5. Koje operacije obavlja upravljački sklop diskovne jedinice?
6. Opisati strategije posluživanja: FCFS, SSTF, SCAN, LOOK, C-SCAN, C-LOOK.
7. Što je to “datoteka”?
8. Što je to datotečni sustav? Što sadrži?
9. Navesti elemente opisnika datoteke.
10. Opisati kako se opisuje smještaj datoteke na NTFS te UNIX i-node sustavima.
11. Opisati svojstva RAID 0, RAID 1, RAID 5 i RAID 6 sustava (kapacitet, otpornost na kvarove).
12. Ako u nekom sustavu kvar i samo jednog diska označava gubitak podataka, koje je očekivano vrijeme rada sustava do gubitka podataka? Za svaki do N diskova znan je MTTF.
13. Ako u nekom sustavu se kvar samo jednog diska može tolerirati, ali ne i dva istovremeno, koje je očekivano vrijeme rada sustava do gubitka podataka? Za svaki do N diskova znan je MTTF. U slučaju kvara, disk u kvaru se zamjenjuje za MTTR vremena.

# 10. KOMUNIKACIJA IZMEĐU PROCESA

## 10.1. Međudretvena komunikacija unutar istog računalnog sustava

### 10.1.1. Dijeljeni spremnik

U dosadašnjim razmatranjima i primjerima pretpostavljalo se da su razmatrane dretve dio istog procesa te su mogle komunicirati preko sredstava tog procesa: varijabli koje se u njemu nalaze.

Slično bi mogle komunicirati i dretve različita procesa ako prethodno zatraže od operacijska sustava da dio adresnog prostora procesa bude zajednički za različite procese.

Prednost dijeljenog spremnika jest u vrlo velikoj brzini pristupa i mogućnosti razmjene velikog skupa podataka. Međutim, problem je što dijeljeni spremnik nije dovoljan: potrebno je uskladiti pristup zajedničkim resursima korištenjem nekih sinkronizacijskih mehanizama.

### 10.1.2. Redovi poruka

U primjeru proizvođača i potrošača pojavila se potreba jednosmjerne komunikacije: proizvođač šalje poruku potrošaču. U navedenom primjeru problem je riješen korištenjem zajedničkog spremnika i nekoliko semafora. Međutim, s obzirom na to da se isti problem javlja često, operacijski sustavi najčešće imaju potporu i za taj način komunikacije preko mehanizma *reda poruka*.

Osnovno sučelje za rad s redovima poruka uključuje:

- stvaranje reda poruka (npr. `mq=mq_open("/q23",O_RDWR|O_CREAT,00600,NULL)`)
- slanje poruke u red (npr. `mq_send(mq,msg_ptr,msg_len,msg_prio)`)
- čitanje poruke iz reda (npr. `mlen=mq_receive(mq,msg_ptr,MAXSZ, &msg_prio)`)

Poruka o ovom kontekstu je jedna kratka cjelovita informacija. Može imati i dodatne atribute, kao što je tip ili prioritet.

Poruke u redu poruka su najčešće posložene po redu prispijeća. Operacija čitanja poruka će vratiti najstariju poruku iz reda. Izuzetak se zbiva kada se zatraži poruka posebna tipa ili prioriteta. Navedeno POSIX sučelje poruke označava prioritetima, pa će poruke u redu najprije biti složene prema prioritetu a tek onda po redu prispijeća (pri čitanju prvo će se uzeti poruka najveća prioriteta).

### 10.1.3. Cjevovodi

Pri prenošenju veće količine podataka umjesto poruka mogu se koristiti *cjevovodi*. Za razliku od komunikacije porukama, kod komunikacije cjevovodom ne radi se razgraničenje između dijelova poslanih/primljenih podataka. Dok je kod komunikacije porukama svako slanje bilo slanje jedne poruke, slanje podataka u cjevovod predstavlja nadovezivanje na prethodno poslane podatke. Ne postoji vidljiva granica između podataka u cjevovodu s obzirom na operaciju slanja.

Primjerice, ako su u cjevovod upisivane vrijednosti u dva navrata: prvi put 10, 20, 30, a drugi put 1, 2, 3, u cjevovodu će biti niz vrijednosti 10, 20, 30, 1, 2, 3. Nikakve dodatne informacije o pojedinačnim upisima nema.

Cjevovod može biti imenovani – vidljiv u datotečnom sustavu (npr. stvoren iz programa s `mkfifo("/tmp/pipe_34", S_IWUSR|S_IRUSR)`) ili neimenovan (stvoren s `pipe(fds)`) kada ga mogu koristiti samo procesi "u srodstvu" (proces roditelj i njegova djeca/unuci, stvoreni s `fork()`).

Slanje podataka u cjevovod i čitanje iz cjevovoda su operacije jednake onima za rad s datotekama te se koriste i ista sučelja. Međutim, za razliku od datoteka, čitanje iz cjevovoda miče te podatke iz cjevovoda (oni nestaju iz njega).

**Tablica 10.1. Usporedba komunikacijskih mehanizama**

| **mehanizam** | **dobra svojstva** | **nedostaci** |
|---|---|---|
| dijeljeni spremnik | brzina pristupa, veličina podataka | potrebna dodatna sinkronizacija |
| redovi poruka | jednostavno sučelje, kratke poruke | samo za kraće poruke |
| cjevovodi | jednostavno sučelje, veće poruke | nije učinkovito za kratke poruke |

## 10.2. Komunikacija u raspodijeljenom sustavu

Mehanizmi prikazani u prethodnom potpoglavlju namijenjeni su za komunikaciju među dretvama istog računalnog sustava, dretvama unutar istog operacijskog sustava. Slični mehanizmi postoje i za komunikaciju među dretvama koje se nalaze na različitim sustavima. Ti mehanizmi, međutim, uključuju i uspostavu komunikacijskog kanala između ta dva sustava, bilo izravno ili neizravno (korištenjem dodatnih usluga operacijska sustava).

U današnjim računalnim sustavima dominantan protokol za povezivanje računala jest protokolni slog Internet (kraće samo Internet) čiji su najbitniji slojevi IP (*Internet Protocol*, mrežni sloj) te TCP (*Transmission Control Protocol*, prijenosni sloj). Stoga se često umjesto Internet koristi i kratica TCP/IP za oznaku istoga.

Proces koji želi usluge mrežnog podsustava operacijska sustava, mora najprije zatražiti spajanje s mrežnim podsustavom (uspostaviti priključnicu – *socket*) te onda preko te veze slati i primati podatke. Pri slanju potrebno je i definirati adresu odredišta (IP adresu i broj priključnice (*port*) na koju je spojen udaljeni proces na svom računalu). Primjerice, pri dohvatu Web stranica potrebno je poznavati adresu Web poslužitelja (npr. www.fer.unizg.hr) te broj priključnice (ako nije zadano pretpostavljena vrijednost za Web poslužitelje je 80). Adresa odredišna računala jest broj (IPv4 ili IPv6 adresa). Međutim, nama ljudima je lakše upamtiti neko ime koje je građeno s određenom logikom. Pretvorbu iz imena u broj obavlja usluga DNS (*Domain Name System*) korištenjem prikladno povezanih poslužitelja.

Sa stanovišta procesa i njegova korištenja mrežnog podsustava, pri korištenju TCP-a koristi se mehanizam sličan dvostrukom cjevovodu: ono što jedna strana upiše u TCP vezu druga će pročitati i obratno. Sve potrebne poslove pakiranja poruke u pakete, zaštita, slanje, prosljeđivanje na odgovarajući izlaz i slično obavlja mrežni podsustav operacijska sustava.

Ako se želi komunikacija slična razmjeni poruka (razmjenjuju se manji podaci) onda se može koristiti UDP (User Datagram Protocol) koji je nešto jednostavniji protokol od TCP-a, ali ne osigurava isporuku. Svaka se poruka zasebno treba adresirati (IP adresa i broj priključnice) i kao takva poslati.

Obzirom na različitost u svojstvima TCP-a i UDP-a, različiti primjenski protokoli se oslanjaju na jedan ili drugi. Primjerice, HTTP koristi TCP kao prijenosni protokol dok DNS koristi UDP.

## Pitanja za vježbu 10

1. Usporediti komunikaciju korištenjem dijeljenog spremnika, reda poruka i cjevovoda.
2. Od čega se sastoji "adresa" udaljenog programa?
3. Kada se koristi TCP a kada UDP?

# 11. VIRTUALIZACIJA

## 11.1. Uvod

Virtualizacijom se unutar stvarna sustava (operacijska sustava, sklopovlja) stvara okruženje (virtualno računalo) sličnih ili različitih svojstava od stvarna sustava, prema potrebama.

Moguće prednosti virtualizacije:

- virtualno okruženje bolje odgovara potrebama
  - sklopovlje u virtualnom sustavu može biti i različito od postojećeg, stvarnog
  - programska potpora može biti različita (npr. operacijski sustav, programi, datotečni sustav, ...)
  - kod projektiranja drugih sustava, simulacija takvih sustava kroz virtualizaciju omogućava jednostavniji razvoj (lakše povezivanje s alatima za ispitivanje ispravnosti, pronalaženje i ispravljanje pogrešaka)
- bolja iskorištenost sklopovlja
  - na istom sklopovlju može se pokrenuti više istih/različitih virtualnih sustava
  - umjesto N stvarnih računala/poslužitelja koristi se jedno (možda snažnije)
  - smanjeni troškovi održavanja, potrošnje, nadogradnje
  - proširivost
    - jednostavno dodavanje novih virtualnih okolina/računala
    - modularno sklopovlje se često može nadograditi (npr. poslužitelj koji se koristi za virtualizaciju se može proširiti dodatnim diskovima ili dodatnim poslužiteljima s kojima će tvoriti logičku cjelinu radi ostvarenja još jačeg sklopovlja koja je podloga virtualizaciji – dodatno sklopovlje se najčešće može samo ugraditi, a virtualizacijski programi će ga automatski uklopiti u postojeći sustav)
  - zastarijeli poslužitelji se mijenjaju jednim novim, koji preuzima usluge prethodnih korištenjem virtualizacije (i u novom sustavu mogu postojati jednaki sustavi, samo što su oni sada virtualni, ali i dalje jednako funkcionalni ili čak i bolji)
- izolacija
  - izolacijom se štite sustavi izvan virtualnog od sustava u virtualnom okruženju, i obratno
  - dostupnost samo dijela sustava u virtualnom okruženju
    - neki se podsustavi operacijska sustava mogu i izostaviti u virtualnom okruženju ili ograničiti pristup samo dijelu nekih podsustava/operacija
  - razvoj novih sustava/programa/operacija u zaštićenom okruženju bez rizika za ostatak sustava

### 11.1.1. Uobičajeni načini virtualizacije

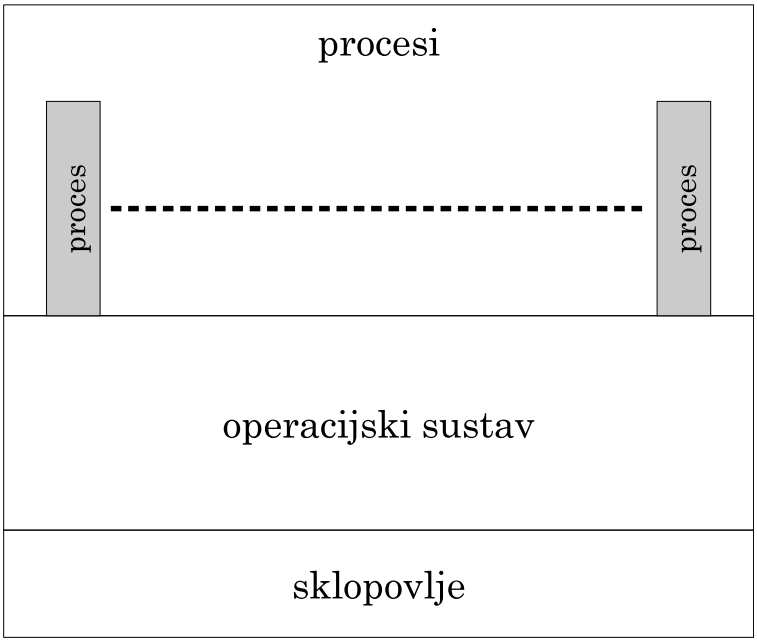
Osnovni pojmovi:

- domaćin (engl. *host*) – operacijski sustav (OS) unutar kojeg se ostvaruje virtualno okruženje, ili hipervizor ako nema OS-a
- gost (engl. *guest*) – OS koji se pokreće u virtualnom okruženju
- virtualni stroj (VS), virtualno računalo – okruženje u kojem se pokreće gost, a s kojim upravlja hipervizor
- upravljač virtualnim strojevima – hipervizor (engl. *hypervisor*) – program na domaćinu koji stvara virtualno okruženje u kojem se mogu pokretati gostujući OS-evi

Uobičajeni načini virtualizacije

1. korištenjem operacijskog sustava domaćina
   a) procesi (ostvareni mehanizmima upravljanja spremnikom kao što je to straničenje)
   b) virtualno okruženje za pokretanje jednog procesa (npr. Java virtualni stroj – *Java VM*)
   c) virtualno okruženje za pokretanje skupa procesa (npr. *Linux Containers*)
   d) virtualno okruženje za pokretanje operacijskog sustava (npr. *VMware*, *VirtualBox*)
2. korištenjem računala domaćina
   - virtualizacijski program (engl. *hypervisor*) nalazi se nad samim sklopovljem
   - primjeri: VMware ESX/ESXi, Xen Project, Oracle VM Server, Microsoft Hyper-V

## 11.2. Sustav bez virtualizacije – procesi u operacijskom sustavu

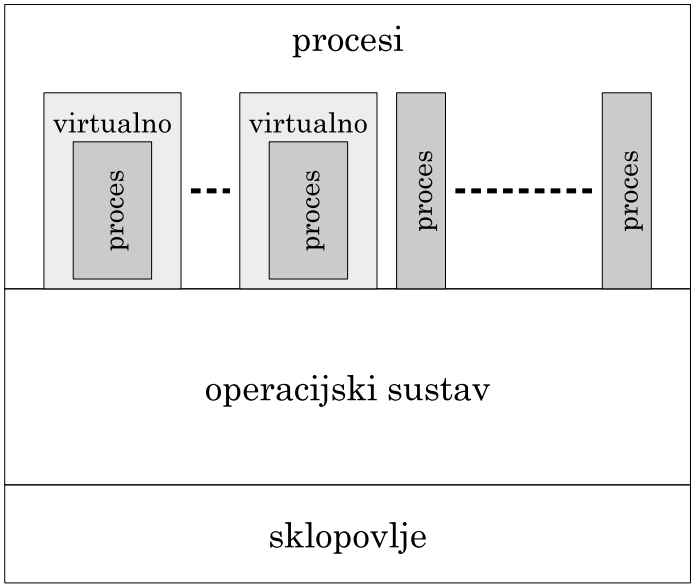



**Slika 11.1. Sustav bez virtualizacije**

- pokretanjem programa stvara se proces
- proces ima izolirani adresni prostor od ostalih procesa i operacijska sustava (straničenje to omogućava)

- svi procesi unutar istog operacijskog sustava dijele sam operacijski sustav
- zajednički elementi unutar jednog operacijskog sustava za procese:
  - datotečni (pod)sustav (i povezani objekti)
  - mrežni podsustav
  - ulazno-izlazne naprave (grafički podsustav, ...)
  - zajednički spremnik, redovi poruka, cjevovodi, ...
- jedan proces svojim akcijama neizravno, preko zajedničkih elemenata, može utjecati na druge procese

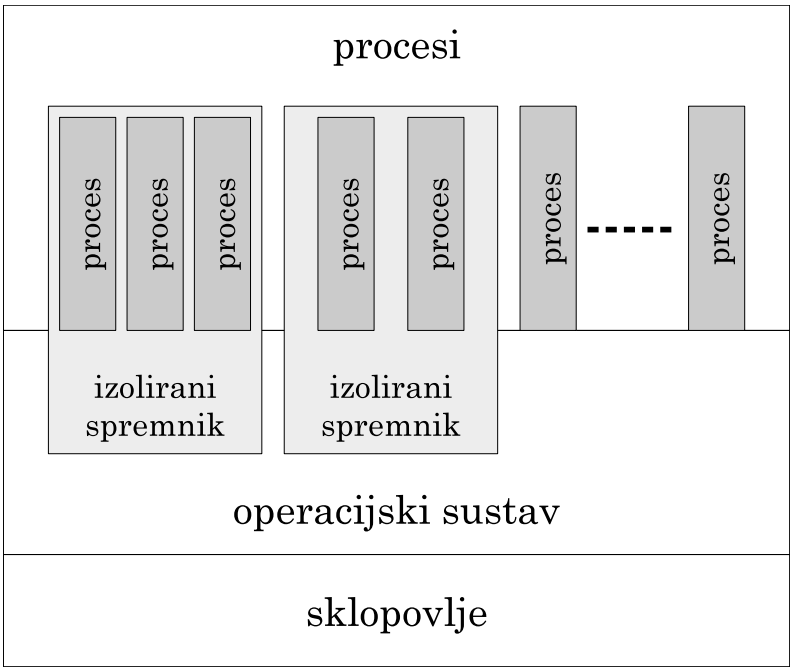
## 11.3. Virtualno okruženje za jedan proces




**Slika 11.2. Virtualizacija na razini aplikacije**

- programi koji nisu pripremljeni u strojnom obliku za operacijski sustav i računalo na kojem se izvode
- stvara se virtualno okruženje samo za taj program
- primjeri: Java programi, programi koji se interpretiraju (npr. skripte, programi u Python-u, Perl-u, JavaScript-u, ...)
- virtualno okruženje može imati ograničeni pristup sredstavima sustava (disku, mreži, napravama, ...)
- virtualno okruženje je u takvu sustavu samo jedan dodatni proces

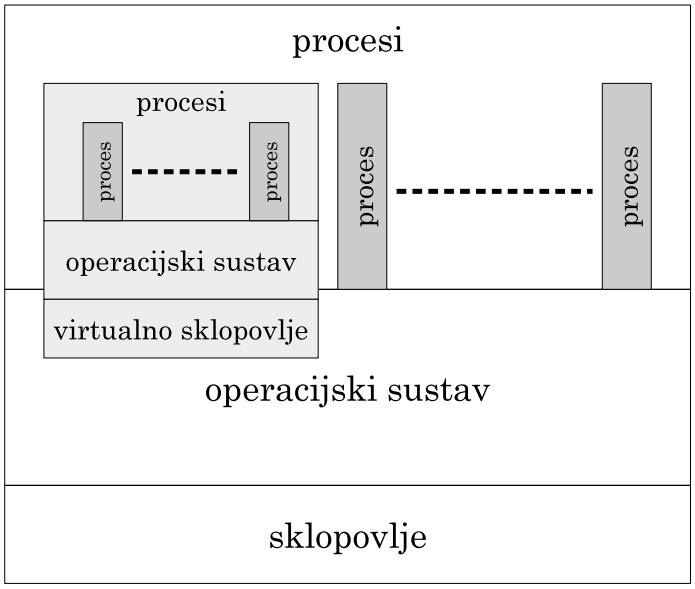
## 11.4. Virtualno okruženje za skup procesa – spremnici




**Slika 11.3. Virtualizacija na razini spremnika**

- podržavaju operacijski sustavi temeljeni na Linux jezgri (engl. *Linux Containers*)
- skup sredstava sustava se izolira/duplicira (disk, mreža, radni spremnik, procesori, ...)
- skup takvih sredstava nazovimo *spremnikom* (engl. *container*)
  - u kontekstu virtualizacije termin spremnik se ne odnosi na radni spremnik – memoriju računala, već na opisano virtualno okruženje
- u takvom okruženju pokreću se novi procesi
- procesi (preko jezgre) koriste pravo sklopovlje, samo su pojedina sredstva zasebna i ograđena – virtualizacija se ostvaruje pri korištenju jezgre za pristup takvim sredstvima
- temeljni operacijski sustav je jednak, ali korištenje sredstava je izolirano
- u svakom spremniku se može pokrenuti zasebni niz usluga
  - web poslužitelj, poslužitelj elektroničke pošte, korisnici, sve može biti u zasebnim spremnicima
  - različiti web poslužitelji mogu biti u različitim spremnicima (npr. pružatelji usluga iznajmljivanja web poslužitelja, mogu za svakog klijenta napraviti zasebni spremnik i u njemu dati prostor za web poslužitelj)
  - razne usluge u različitim spremnicima su izolirane – ne utječu jedne na drugu s aspekta sigurnosti
  - svaki spremnik može imati dodijeljeno i procesorsko vrijeme (npr. broj procesora, postotak procesorskog vremena), prostor u memoriji, vlastiti datotečni sustav koje je bar u nekim dijelovima različiti od ostalih, svoj mrežni stog (adrese, način prosljeđivanja i slično), ...

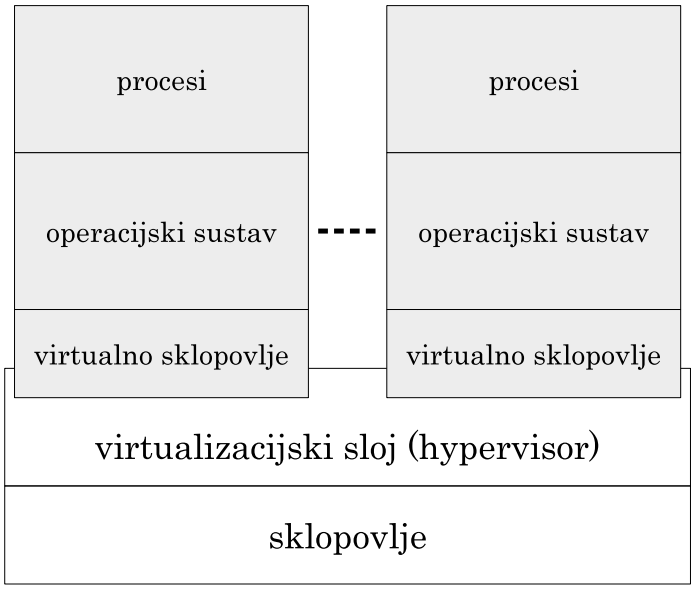
## 11.5. Virtualno okruženje za operacijski sustav ("tip-2")




**Slika 11.4. Virtualizacija na razini operacijskog sustava**

- nad operacijskim sustavom domaćina ostvaruje se virtualno okruženje – virtualno računalo u kojem se pokreće gostujući operacijski sustav
- domaćin i gost mogu biti različiti operacijski sustavi (npr. domaćin je Windows 10, a gost Ubuntu)
- domaćin osigurava pristup sklopovlju gostu korištenjem raznih mehanizama (stvaranjem virtualnih uređaja, dozvolom izravna pristupa nekom sklopovlju i slično)
- studenti koji na svom računalu nemaju neki Linux/UNIX sustav su preko odgovarajućih programa (VMware, VirtualBox) uspostavili virtualno računalo za laboratorijske vježbe za ovaj predmet

## 11.6. Virtualno okruženje na razini sklopovlja ("tip-1")




**Slika 11.5. Virtualizacija na razini sklopovlja**

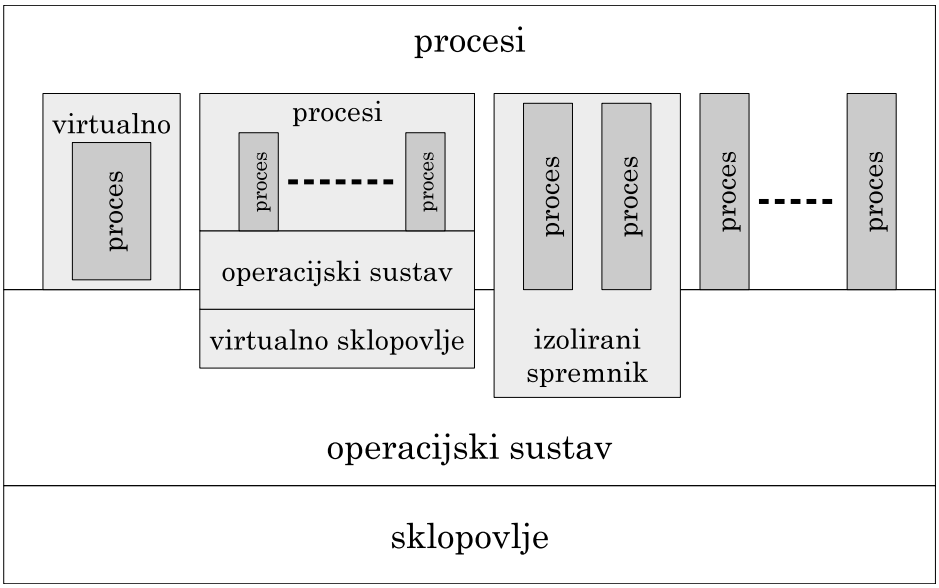
- virtualizacijski program (engl. *hypervisor*) nalazi se nad samim sklopovljem
- virtualizacijski program predstavlja tanak sloj između sklopovlja i operacijskih sustava

- iako u virtualnom okruženju, učinkovitost operacijskih sustava znatno je veća nego kad se za virtualizaciju koristi operacijski sustav domaćina
- virtualizacijski program omogućuje pokretanje više sustava iznad sebe, upravlja dodijeljenim sredstvima, stvara nova virtualna okruženja, ...
- ukupno virtualno dodijeljena sredstva (procesori, memorija) su najčešće i puno veća od dostupnih sredstava – hipervizor dinamički dodijeljuje sredstva, prema potrebama – ne koriste svi virtualni strojevi sve resurse u isto vrijeme (sjetite se straničenja, isto načelo se ovdje primjenjuje i za ostale resurse)
- primjeri: VMware ESX/ESXi, Xen Project, Oracle VM Server, Microsoft Hyper-V
- najčešće se koristi u poslužiteljskim centrima (podatkovnim centrima, sustavima koji pružaju infrastrukturu za računarstvo u oblaku)
  - virtualno računalo ovog tipa se može zakupiti (npr. Amazon, Google, Microsoft Azure)
- virtualizacijski program mora imati mehanizme slične onima koje ima i operacijski sustav
  - virtualnim računalima treba dodijeliti procesorsko vrijeme, memoriju, ...
  - virtualizacijski program zato često uključuje i jezgru OS s kojom je čvrsto povezan

- kao dodatni način virtualizacije ("tip-0") je virtualizacija kojim upravlja sklopovlje ("firmware")
  - najjednostavniji oblik virtualizacije kod koje se statički dodijele resursi sustava nekom "virtualnom" računalu
  - virtualno računalo samo upravlja dodijeljenim resursima, a uloga hipervizora jest samo "mapiranje" dodijeljenih resursa virtualnom računalu (da ono vidi samo te dodijeljene resurse)
  - s aspekta performansi ovo je najbolje (gotovo da i nema virtualizacije), ali s aspekta učinkovitosti iskorištenja sredstava sustava najlošije

## 11.7. Korištenje različitih oblika virtualizacije istovremeno

Navedeni oblici se mogu kombinirati, koristiti paralelno ili čak jedan ugraditi u drugi.




**Slika 11.6. Paralelno korištenje različitih tipova virtualizacije**

## 11.8. Problemi ostvarenja virtualizacije

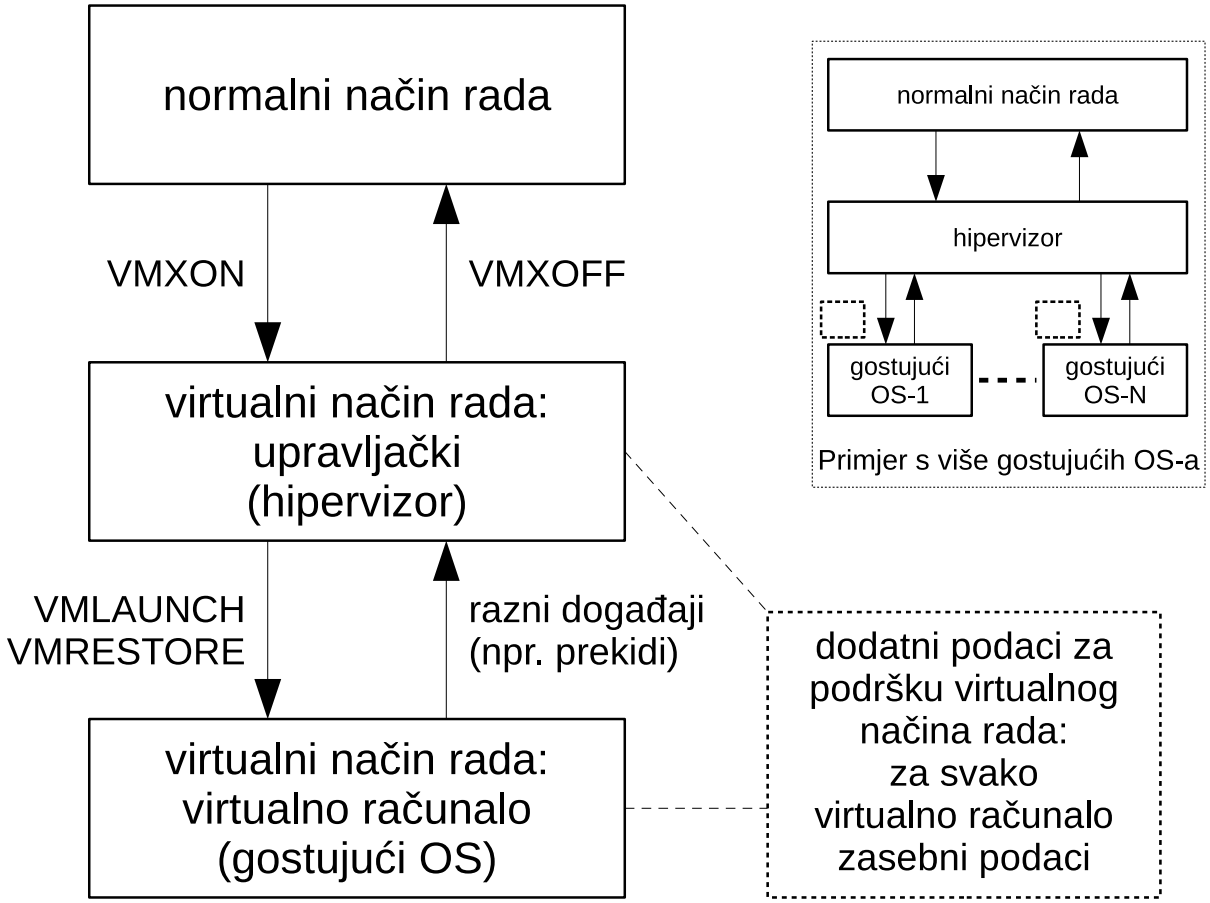
- osnovni problem: gostujuće računalo se izvodi u korisničkom načinu rada
- u gostujućem računalu se OS (njegova jezgra) želi ostvariti korištenjem privilegiranog načina rada
  - računalo domaćin ne smije dozvoliti da gost bude u privilegiranom načinu rada jer onda može kompromitirati i domaćina (i ostale programe/podatke na njemu)
  - pokušaj ulaska u privilegirani način rada iz program na gostu, kao i pokušaj izvođenja privilegiranih načina rada može se detektirati u računalu (operacijskom sustavu) domaćina (kao prekid zbog nedozvoljenih operacija)
    - način "uhvati-i-emuliraj"
    - u obradi takvih događaja, uz provjeru da se zaista radi o procesu koji omogućuje virtualno računalo, "problematične instrukcije/operacije" gostujućeg računala se ručno odrađuju – simulira se njihov rad
  - problem: neke instrukcije se drukčije izvode ovisno o načinu rada
    - npr. instrukcija POPF će u korisničkom načinu rada sa stoga obnoviti samo "obične zastavice" u registar stanja, dok će u privilegiranom načina rada obnoviti sve zastavice (i one koje utječu na prihvat prekida i na promjenu načina rada procesora!)
    - takva instrukcija neće izazvati prekid u korisničkom načinu rada procesora, neće ju se na taj način moći ispraviti
    - stoga se pri zahtjevu za prelazak u jezgrin način rada u gostujućem računalu, što se neće dogoditi već treba simulirati, prije nastavka rada treba "proći" krozi instrukcije koje bi se obavile te po potrebi napraviti korekcije
      - binarna translacija
      - jedna instrukcija se zamjenjuje drugom ili čak i nizom instrukcija
- problem s upravljanjem spremnikom (straničenje!), rad s UI napravama, ...
- mnogi problemi se mogu riješiti ako procesor ima sklopovsku potporu za virtualizaciju

### 11.8.1. Sklopovska podrška virtualizaciji

- danas većina procesora ima nekakvu podršku virtualizaciji
- podrška virtualizaciji kod Intelovih procesora:
  - dodatni načini rada procesora i dodatne instrukcije i strukture podataka
  - osim dva normalnog načina rada (korisnički i privilegirani), procesor dodatno ima i dva "virtualna načina rada" za podršku virtualizacije
    1. upravljački virtualni način rada - za rad virtualizacijskog programa (hypervisor)
    2. virtualni način rada za gosta
       - u ovom načinu se lakše upravlja virtualim korisničkim i jezgrinim načinom rada
       - neke stvari procesor može sam napraviti ispravno, a one za koje to ne može prekida izvođenje i prepušta upravljaču virtualnog načina rada da to odradi za njega
  - posebnim instrukcijama se ulazi u te načine rada i pokreće virtualna računala (slika

11.7.)

– dodatna struktura podataka omogućuje pohrane/obnove ključnih podataka da virtualno računalo ne bi utjecalo na računalo domaćina i ostala virtualna računala
– slično upravljanju procesima u OS-u, samo što se ovdje radi o virtualnim računalima kojima upravlja upravljački program (hipervizor)




**Slika 11.7.**

# 12. POSLOVI ODRŽAVANJA OPERACIJSKIH SUSTAVA

## 12.1. Instalacija operacijskog sustava

- Instalacija operacijskog sustava je danas uglavnom vrlo jednostavna
  - instalacijski postupak sam dohvati informacije o sklopovlju, mreži
  - potrebni podaci:
    - podaci o korisniku, jeziku, vremenskoj zoni, regionalnim postavkama, tipkovnici
    - mrežne postavke, ako je potrebno (spajanje na bežičnu mrežu, ako nema DHCP-a onda statičke IP adrese, spajanje PPPoE i sl.)
- Mogući problemi: OS nema upravljačke programe za sve naprave u računalu
  - tada treba identificirati što ne radi (npr. Device Manager)
  - korištenjem upravljački programa koji su došli s računalom (najčešće na CD-u) ili preko stranica proizvođača dohvatiti odgovarajuće programe
    - paziti na ispravan upravljačkog programa (obično isti CD sadrži upravljačke programe za slične proizvode istog proizvođača)
    - paziti na ispravan odabir OS-a, 32-bita/64-bita

## 12.2. Održavanje

- provjeriti postoji li antivirusni alat, sigurnosna stijena, je li sustav ažuriran (uključujući OS, antivirusne alate i sigurnosnu stijenu)
- instalirati **antivirusni alat** ako već nije (npr. win 10 imaju)
- postaviti **sigurnosnu stijenu** (engl. *firewall*), ako nije automatski postavljena (uglavnom je)
- redovito **ažurirati računalo zakrpama** koje nudi OS
  - zakrpama se ispravljaju do tada uočene greške i propusti
  - noviji sustavi to rade automatski
    - npr. win 10 pri restartu naprave potrebno (ponekad i sami restartaju računalo)
    - Linux distribucije uglavnom pitaju kada žele ažurirati
- **sigurnosna kopija podataka** (engl. *backup*)
  - podatke (ne OS, programe) bar povremeno kopirati na druge medije
  - ako podaci nisu vrlo tajni, moguće je koristiti usluge pohrane u oblaku (npr. Dropbox, Google Drive, OneDrive, ...)
    - podaci se tamo (interno) pohranjuju na način da postoji bar još jedna kopija (da se i u slučaju kvara nekog diska kod njih i dalje može doći do podataka)
    - u tom slučaju nije neophodno i ručno raditi kopiju (ali je preporučeno, samo u većim razmacima)

  - u “produkcijskim sustavima” izrada sigurnosne kopije mora biti automatizirana (mora postojati i procedura, svako koliko, što se trajno čuva, što se prepiše novim, ...)
- kopija kritičnih datoteka OS-a:
  - "system restore" (kod Windows OS-eva)
    - OS to radi automatski prije nekih većih promjena (npr. ažuriranja)
    - mogu se napraviti i ručno prije nego što korisnik napravi promjene (npr. želite ručno nadograditi upravljačke programe bitne naprave)
    - uobičajeno je da OS na disku rezervira 10% prostora za te pohrane
    - u slučaju problema (pri pokretanju ili nešto ne radi) može se vratiti stanje sustava (kritična dijela sustava) kako je bilo u nekom prethodnom trenutku
- programi koji su trebali samo jednom ili u nekom intervalu i više ne trebaju niti se predviđa da će trebati – preporuka je da ih se makne (deinstalira)
  - to se pogotovo odnosi na igre na kućnim računalima (igre mogu zauzimati dosta prostora na disku!)
- pri instalaciji programa (posebice onih skinutih s Interneta), pažljivo gledati postavke
  - mnogi takvi programi dodatno žele instalirati neki dodatak preglednicima (a oni mogu biti vrlo opasni ili "samo" dosadni)
  - mnogi dodaci takvih programa najčešće nisu potrebni (npr. neki servis koji će se pokretati s OS-om; takvih servisa se jako brzo nakupi mnogo)
- povremeno provjeriti stanje diska: kad se on prepuni (kad je malo praznog prostora) performanse sustava mogu značajno opasti (sustav je sporiji)
- povremeno pogledati dnevnike sustava (zapisnike o događajima, *logove*)
- **zaporke** (lozinke) generirati tako da nisu jednostavne za odgonetanje svima osim vama (npr. ne koristiti samo brojke, ne samo znakove, ne kraće od 8 znakova)
  - dobra zaporka ima barem po jedan element od svih sljedećih:
    - veliko slovo (A-Z)
    - malo slovo (a-z)
    - broj (0-9)
    - interpunkcijski znak (. , ; : ! ? < > # $ % & ( ) [ ] { } | = + – * / _ ’ " \ ∼)
  - izbjegavati korištenje naših grafema, jer pri prijavi s drugog računala na kojem možda nisu na istom mjestu može biti problema
  - ne stavljati svoja imena ili imena osoba s kojima smo povezani, ne stavljati datume rođenja, OIB ili neki drugi broj ili riječ koju bi napadač mogao jednostavno doznati o vama
  - jedna mogućnost: staviti kraticu koja ima nekog smisla vama (od stiha neke pjesme, neke uzrečice, nešto što ste sami smislili) – tako se teže zaboravi
    - npr. od uzrečice: “Ako nije bilo backupirano, onda nije bilo važno.”
      - uzeti samo početna slova riječi: Anbb,onbv
      - interpunkcijski znak već postoji (zarez)
      - veliko slovo već postoji (A)

- dodati neki broj (npr. 3 – automatizirani backup se obično radi preko noći): Anbb,onbv3
- dobivena lozinka se lako pamti – ne po slovima nego po uzrečici i logici dodataka
- dobivena lozinka se teško pogađa od strane drugih osoba i računalnih programa "razbijača" zaporki

## 12.3. Otklanjanje problema

- Kako otkriti da nešto ne radi dobro?
  1. Računalo se neće pokrenuti
     - pri pokretanju aktivirati poseban način rada (npr. preko ESC, F2, F8 ili slično)
       - u takvu načinu rada detektirati i popraviti računalo
     - ako se računalo neće ni u tom načinu rada pokrenuti, koristiti dodatne medije za pokretanje (instalacijski ili nekakav "repair" disk)
  2. Računalo se pokreće, ali:
     - računalo je sporo (a nije problem u procesorskoj snazi, količini memorije, disku)
     - otvaraju se stranice koje nismo zahtijevali
     - ne pokreću se programi koje smo željeli pokrenuti i dali te naredbe preko sučelja OS-a
     - računalo se često zaustavlja, smrzava, pojavljuje se "plavi ekran" ("Blue Screen of Death", BSoD)
- Mogući uzroci problema i neke ideje za rješavanje
  - Virusi/trojanci/*:
    - instalacija program za micanje tog virusa
    - ponekad je potrebno to napraviti u posebnu načinu rada (engl. *safe mode*)
    - poželjno se je odspojiti od Interneta za taj proces (najbolje je kad bi imali drugo računalo s kojim se može dohvatiti takav program i preko USB ključića ili CD-a prenijeti na zaraženo računalo)
  - Greške OS-a/upravljačkih programa:
    - "system restore" / "recovery" ili nanovo instalirati upravljačke programe
  - Sklopovska greška (npr. greška u memoriji)
    - otkriti gdje je greška i zamijeniti taj sklop (ako je moguće)