

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

Interni materijal za predavanja iz predmeta

Operacijski sustavi

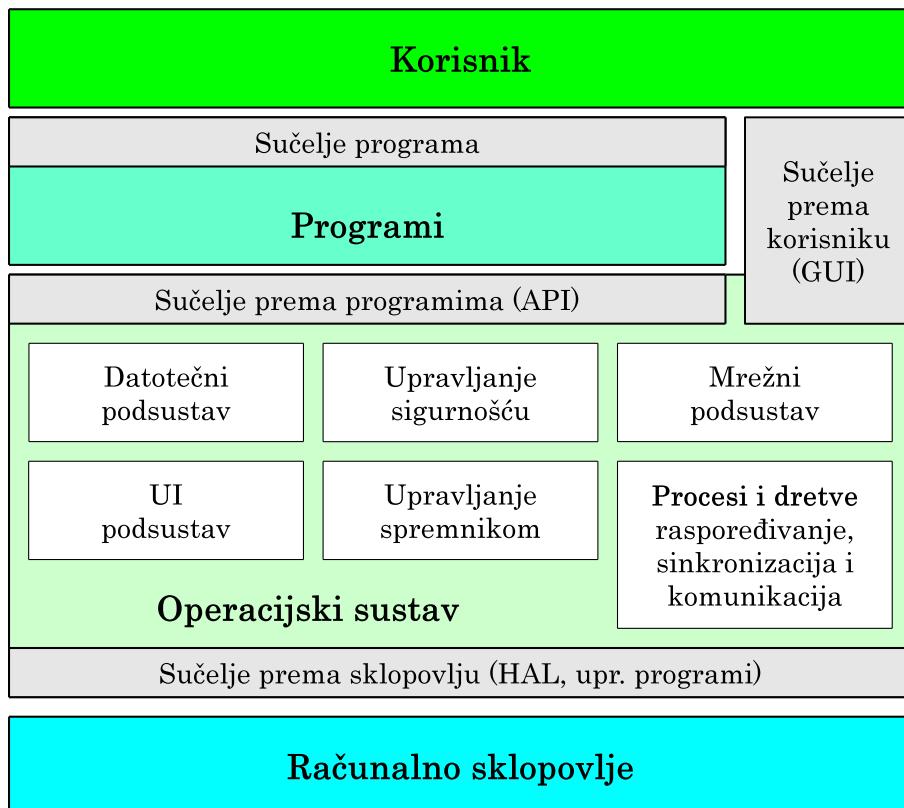
Autor: Leonardo Jelenković

Ak. god. 2014./2015.

Zagreb, 2014.

1. Uvod

Računalni sustav se sastoji od: korisnika, programa, operacijskog sustava te sklopoljja.



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

Među slojevima nalazi se **sučelje**

- definira način korištenja/komunikacije među slojevima

Operacijski sustav je skup osnovnih programa koji:

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

Svrha/uloga OS-a

- olakšavanje uporabe računala (skriva detalje)
- djelotvorno (učinkovito) korištenje svih dijelova računala (ima upravljačke programe)
- višeprogramska rad
- OS doprinosi učinkovitosti sustava – zato je i uveden!

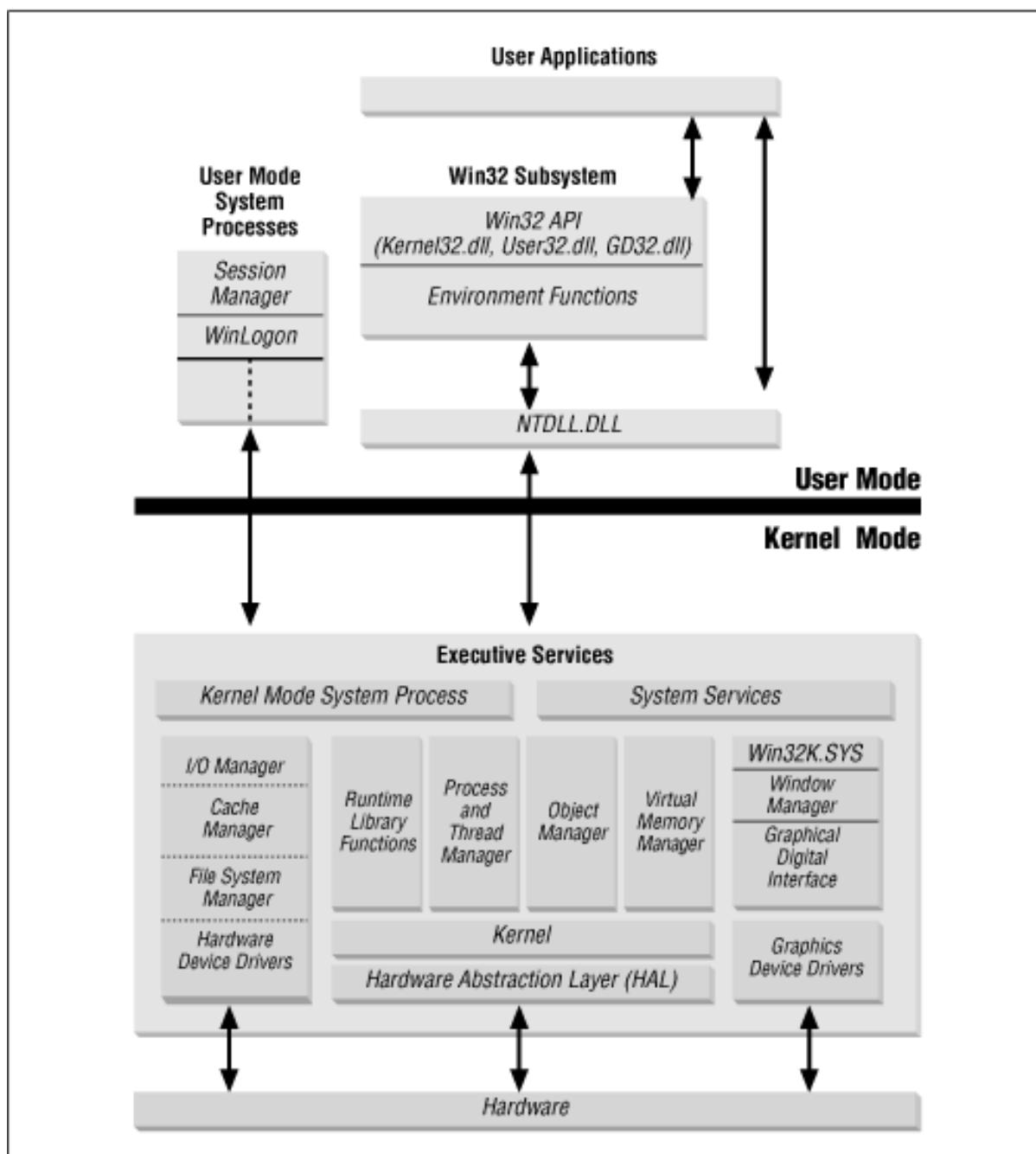
OS je jako složen sustav, zato se osim na slojeve prema prethodnoj slici dijeli i na na podsustave (koristi se načelo “podijeli i vladaj”).

1.1. Podsustavi, slojevi (info)

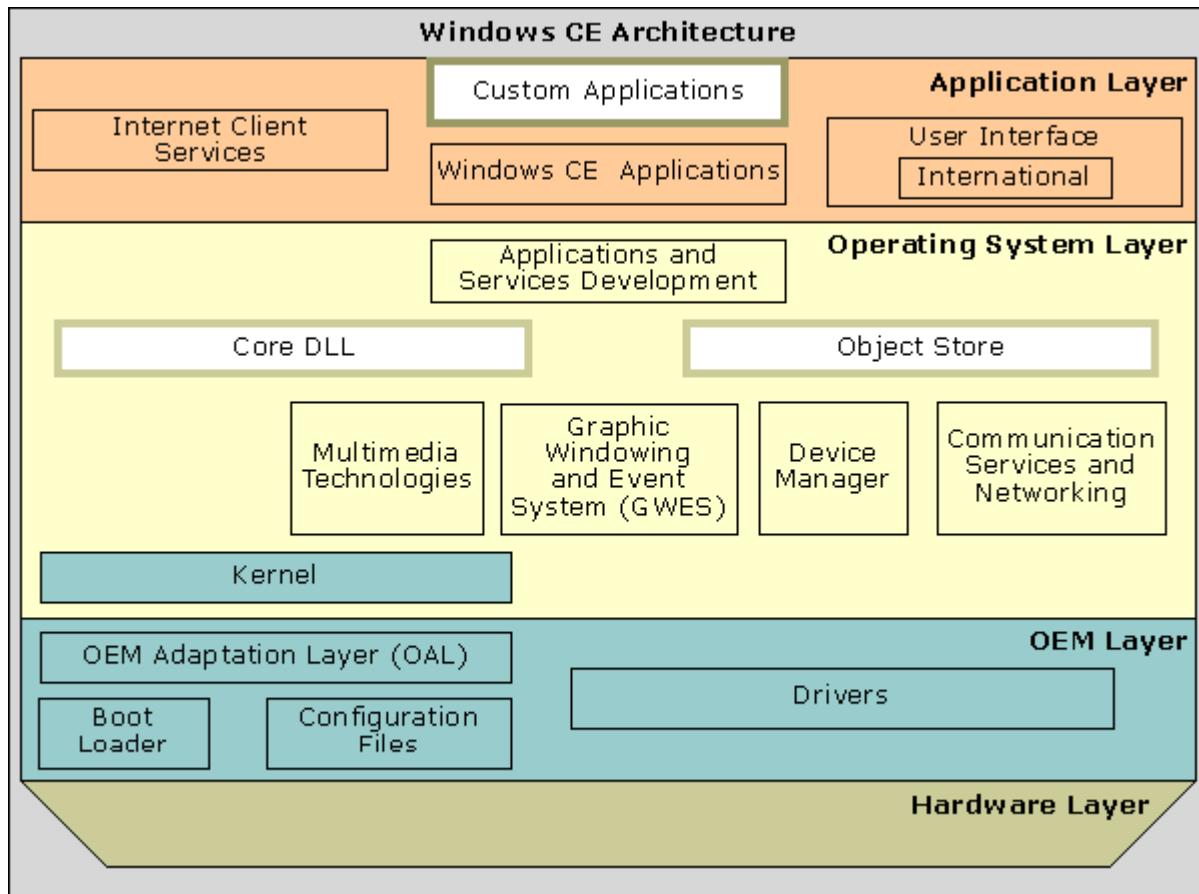
OS se može podijeliti i na slojeve (opc., arhitektura OS-a):

- sustavske funkcije i sučelje prema programima (API)
- jezgra operacijskog sustava (s većinom podsustava)
- apstrakcija sklopolja (HAL – hardware abstraction layer)

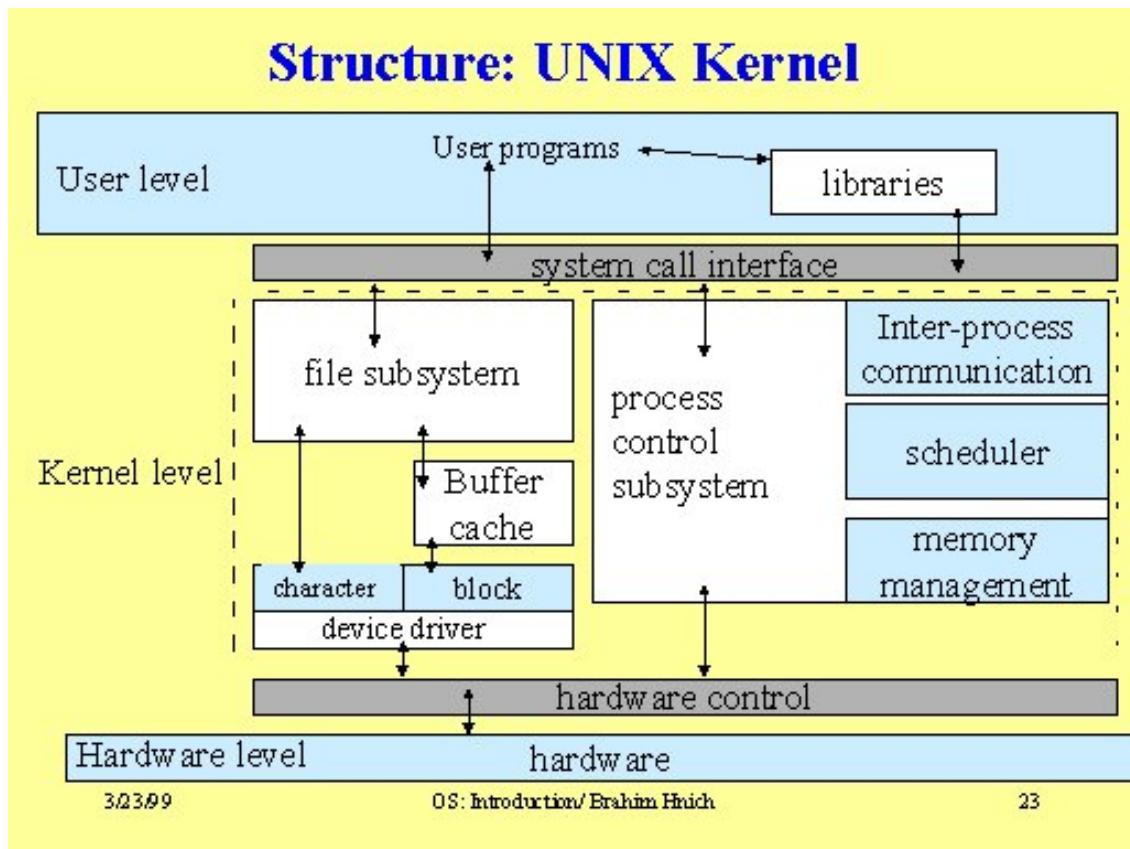
Primjeri podjela OS-a na slojeve/podsustave su u nastavku.



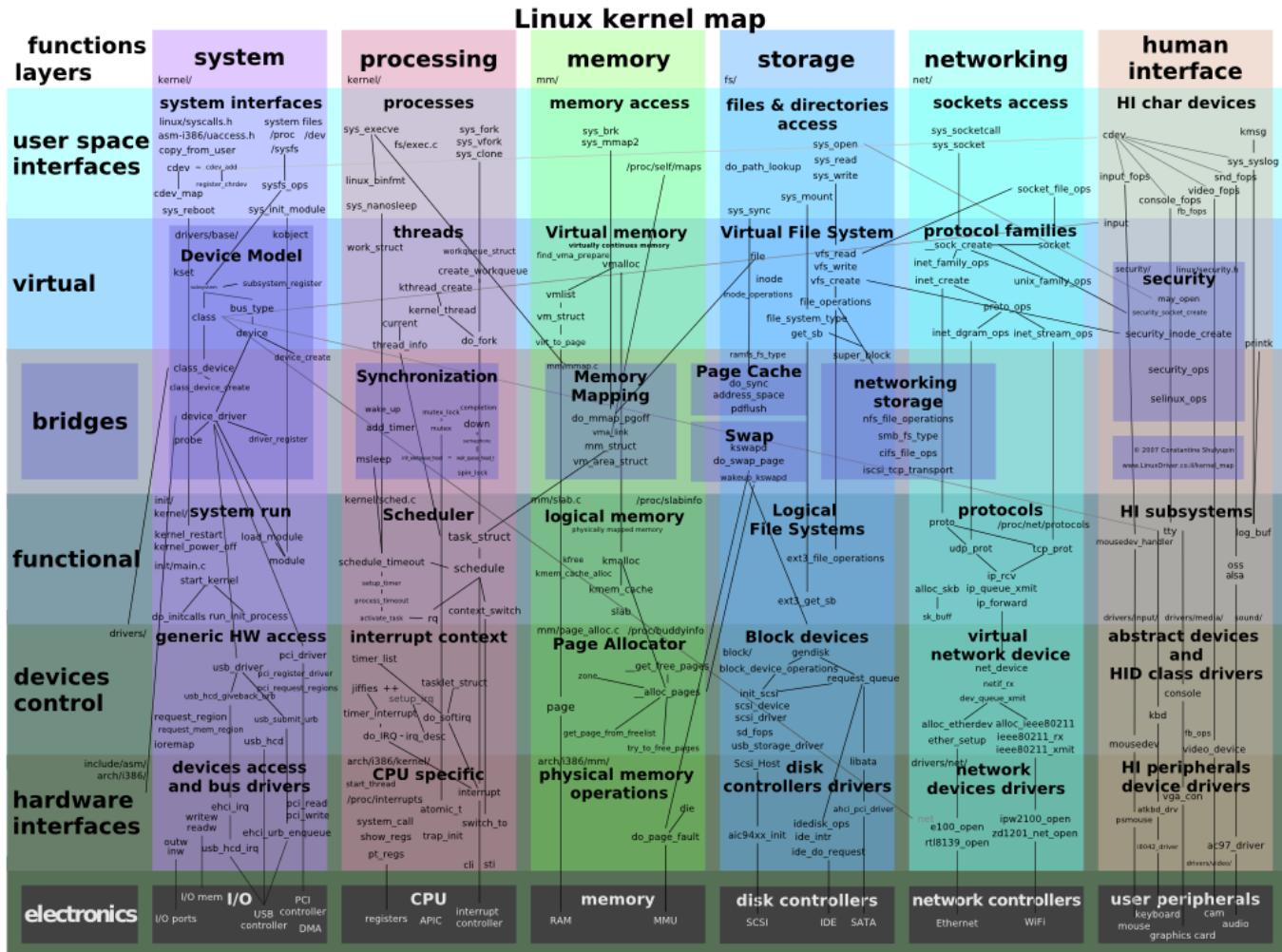
Slika 1.2. Arhitektura operacijskog sustava Windows*



Slika 1.3. Arhitektura operacijskog sustava Windows CE

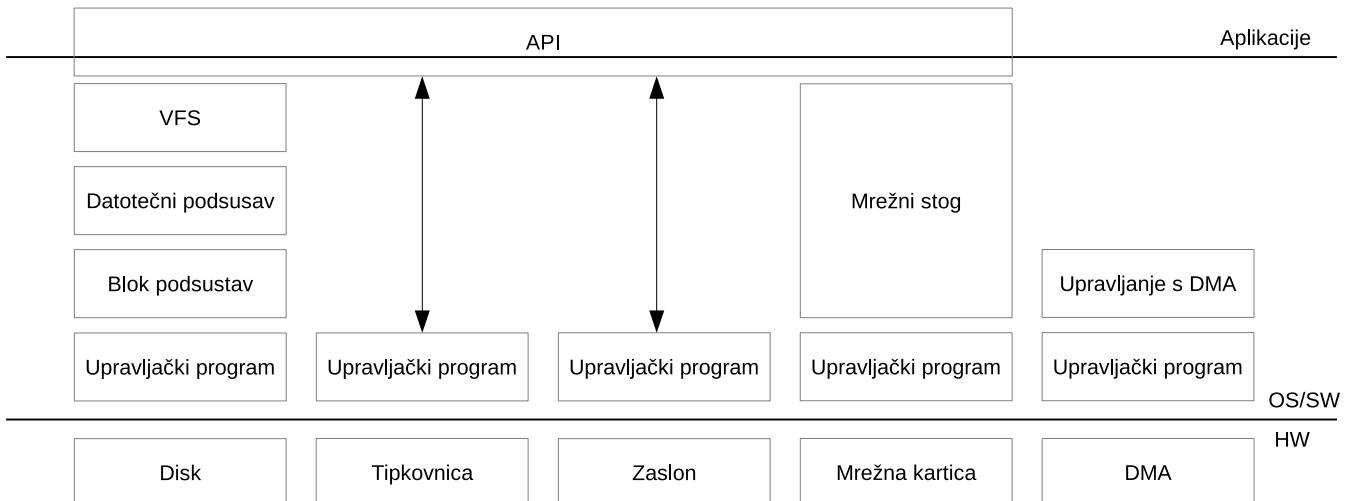


Slika 1.4. Arhitektura operacijskog sustava UNIX



Slika 1.5. Arhitektura operacijskog sustava Linux

Kao što je vidljivo iz prethodnih primjera, OS je vrlo složen sustav kod kojeg su i podsustavi složeni. U okviru ovog predmeta razmatrati će se jednostavniji sustav, sustav koji bi se mogao prikazati slikom



Slika 1.6. Pojednostavljena arhitekura

1.2. Kratka povijest operacijskih sustava (info)

| Godina | Operacijski sustavi |
|---------|---|
| ~1960 | Multics, IBM System/360 Operating System, CP/CMS |
| 1969. | UNIX (Unics) (AT&T) (Ritchie, Kernighan + ostali) (UNiplexed Information and Computing Service) |
| 1973. | UNIX prepisan u C, dalje se intenzivnije razvija i grana |
| 1980-te | Mac OS, MS-DOS, OS/2 |
| 1990-te | MS-Win 3.11 (Windows for workgroups) Windows 95, Windows NT, Windows 98 Linux (1991. verzija 0.01); kasnije i Linux distribucije: Slackware, Redhat, Debian, Ubuntu... FreeBSD |
| 2000-te | Windows (2000, ME, XP, 2003, Vista, 2008) Linux (jezgra) 2.4 – 2001, 2.6 – 2003, 3.0 – 2011 Mac OS X Android, iOS, Windows Phone |
| 2010-te | (verzije provjerene 17. 9. 2013.) Windows 7 – 2009, Windows 8 – 2012, (8.1 – 10/2013) Linux (jezgra) 3.0 – 2011 (z.v. 3.10.xx – 09/2013) OS X (Mac) (z.v. 10.9 – 2013) Android (z.v. 4.3 – 7/2013, 4.4 uskoro) iOS (z.v. 6.1 – 03/2013, 7.0 – 09/2013) Windows Phone (z.v. 8 – 10/2012) |

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

Dijelovi računala

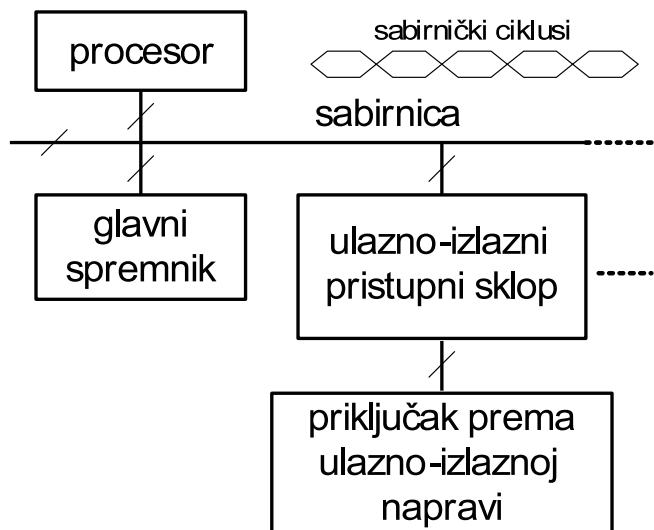
- funkcionalni model: ulazni dio, izlazni dio, radni spremnik, AL jedinka, upravljačka jedinka
- sabirnički model: procesor, sabirnica, spremnik, UI pristupni sklopovi

[dodatak]

Von Neumannov model računala – instrukcije i podaci u istom spremniku (dohvaćaju se preko zajedničke sabirnice)

Druga arhitektura = Harvardska arhitektura – instrukcije odvojene od podataka i dohvaćaju se za sebnom sabirnicom, dok se podaci dohvaćaju svojom. L1 priručni spremnik procesora je uglavnom izведен Harvardskom arhitekurom (podijeljen na dio za instrukcije i dio za podatke).

2.1. Sabirnički model računala



Slika 2.1. Sabirnički model računala

Sabirnički ciklus

- prijenos jednog podatka od procesora do spremnika (ili pristupnog sklopa) ili obratno (do procesora)

[dodatak]

Procesor upravlja radom sabirnice (u ovakovom modelu)

1. postavlja adresu
2. postavlja podatak (kada se on zapisuje u spremnik ili UI)
3. postavlja upravljačke signale

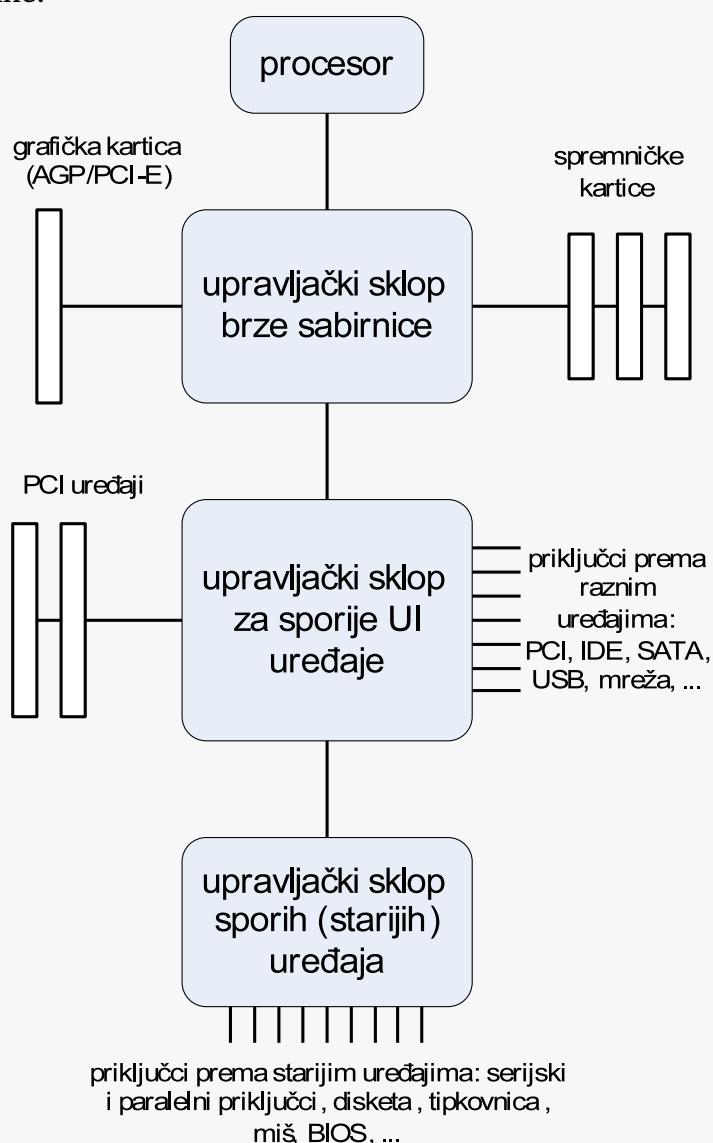
Primjer 2.1. Trajanje sabirničkog ciklusa

Neka je $T_B = 10 \text{ ns}$ trajanje jednog sabirničkog ciklusa, tada:

- frekvencija rada sabirnice jest $\frac{1}{T_B} = 100 \text{ MHz}$
- ako je širina sabirnice 32 bita tada je propusnost sabirnice: $32 \cdot 100 \text{ MHz} = 3200 \text{ Mbita/s}$ (M je u ovom slučaju (za brzine) 10^6 a ne 2^{20})

Primjer 2.2. Hijerarhijsko povezivanje sabirnica

Zbog različitih brzina komponenata računalnog sustava, stvarne arhitekture imaju više sabirnica kojima su one povezane.

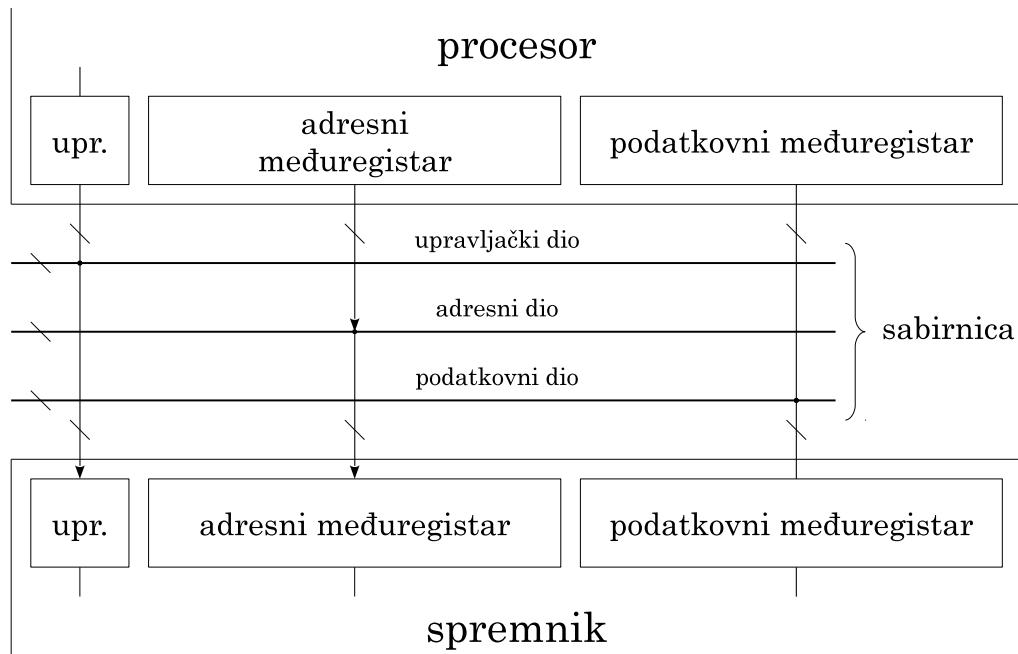


Slika 2.2. Moderni model računala

2.2. Kratki opis komponenata računala

Razmotrimo prvo samo najosnovnije komponente: procesor, spremnici i sabirnicu (ostale ćemo kasnije).

2.2.1. Sabirnica

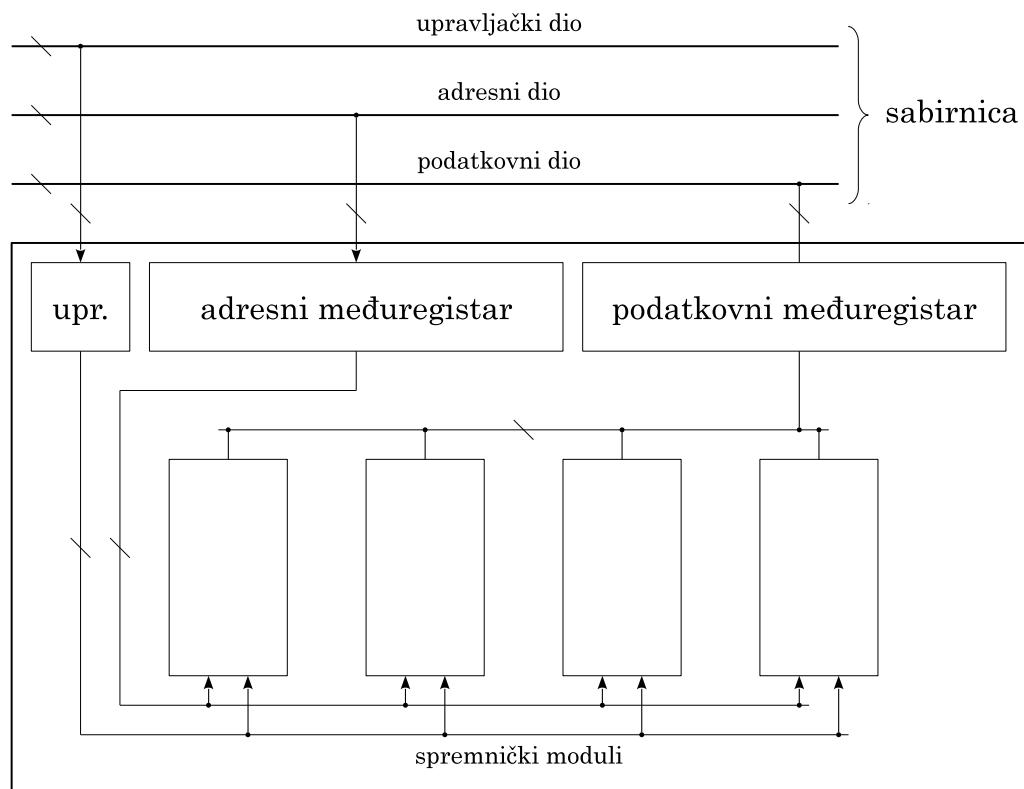


Slika 2.3. Model sabirnice

Sabirnica se sastoji od tri dijela:

- adresni dio
- podatkovni dio
- upravljački dio (npr. piši ili čitaj, BREQ, BACK, prekidi, ...)

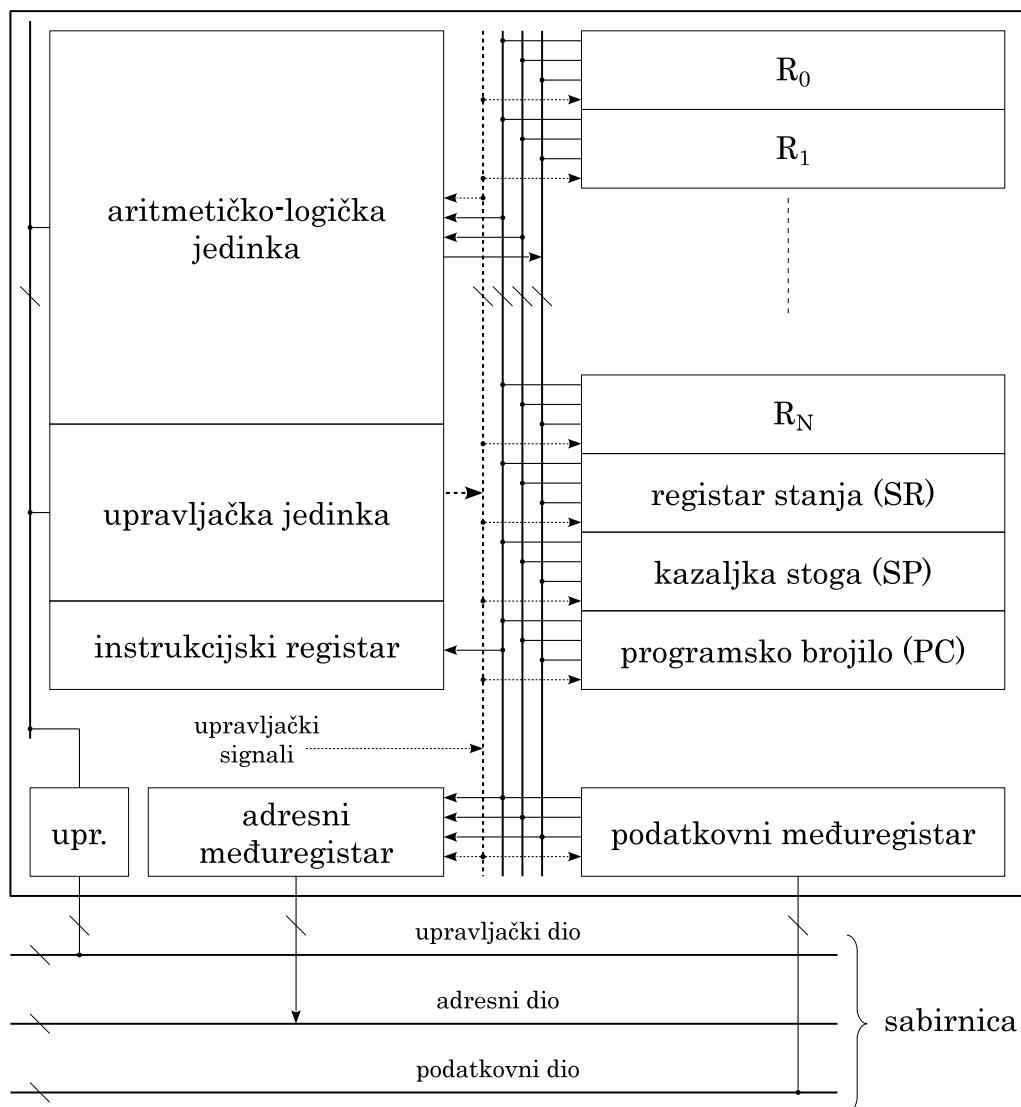
2.2.2. Spremnik



Slika 2.4. Model spremnika

Spremnik spaja se na sabirnicu s svojim međuregistrima i poslužuje zahtjeve za čitanje i pisanje.

2.2.3. Procesor



Slika 2.5. Model procesora

Procesor se sastoji od elemenata:

- aritmetičko-logička (AL) jedinka
- registri opće namjene (npr. R_0-R_7)
- 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 promatrati kao automat:

```

ponavljaj {
    dohvati instrukciju na koju pokazuje PC;
    povećaj PC tako da pokazuje na iduću instrukciju;
    dekodiraj instrukciju;
    obavi operaciju zadalu 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

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

[dodatakno]

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).

2.3. Instrukcijska dretva

Primjer 2.3. Primjer programa s grananjem i pozivima potprograma

```
; Računanje faktorijela
; zadano: N, prepostavlja se N > 0 !
; rezultat spremiti u: REZ

LDR R1, N      ; učitaj N u R0
LDR R0, N      ; R0 akumulira rezultat (umnožak)

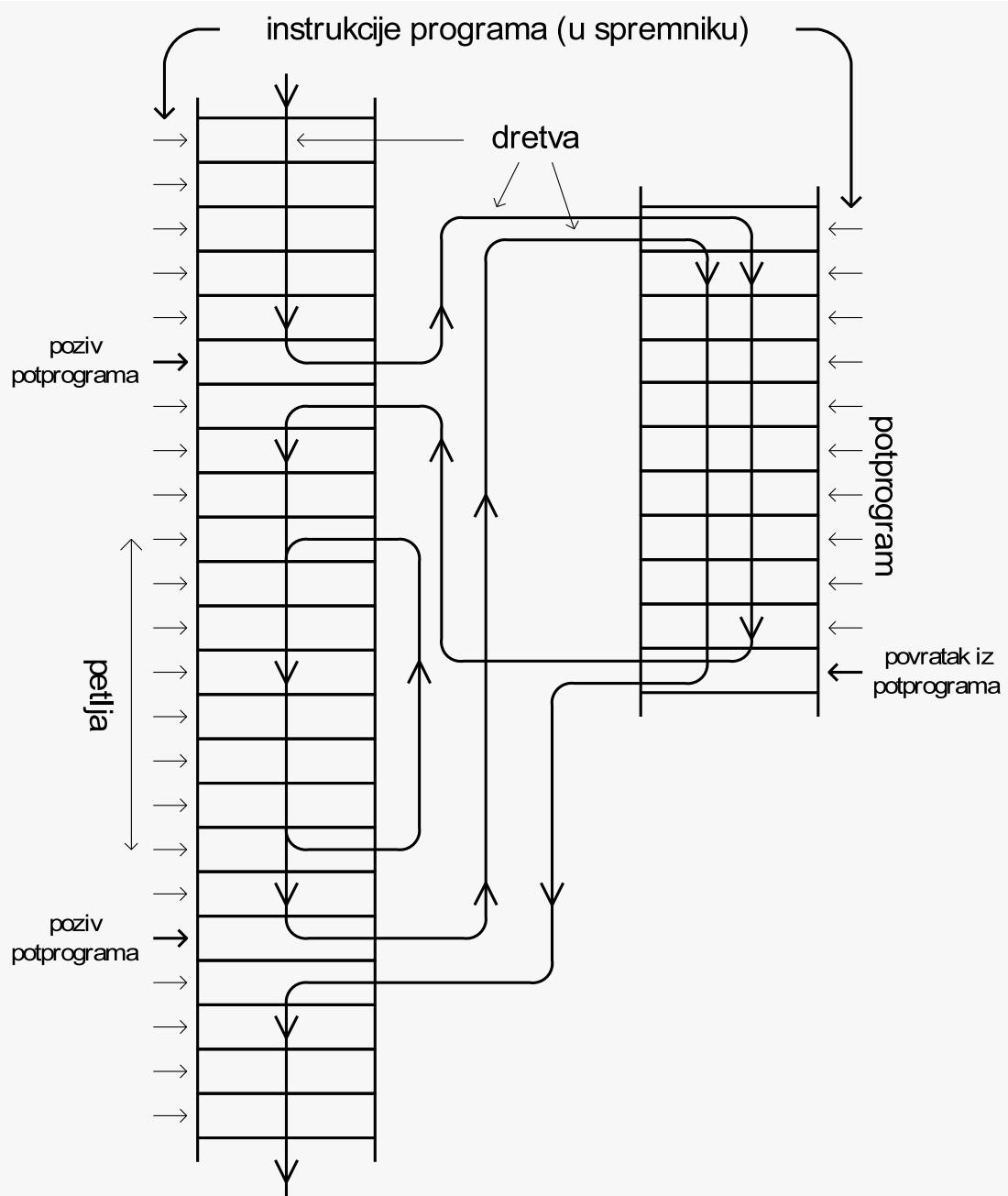
PETLJA:
    SUB R1, 1    ; R1 = R1 - 1
    CMP R1, 1    ; usporedi R1 i 1
    BEQ KRAJ     ; ako je R1 == 1 skoči na KRAJ

    PUSH R1      ; stavi R1 na stog - parametar funkcije
    CALL MNOZI
    ADD SP, 4    ; makni R1 s vrha stoga

    B PETLJA     ; skoči na PETLJA

KRAJ:
    STR R0, REZ ; spremi R0 u REZ
    ...

MNOZI:
    LDR R2, [SP+4] ; dohvati parametar
    MUL R0, R2; R0 = R0 * R2;
    RET
```



Slika 2.6. Instrukcijska dretva

Korištenje stoga na prikazani način omogućava gniježđenje poziva i rekurziju.

Program

- niz instrukcija (i podataka) koji opisuju kako nešto 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)

[dodatno]

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* (*thread*).

Dretva

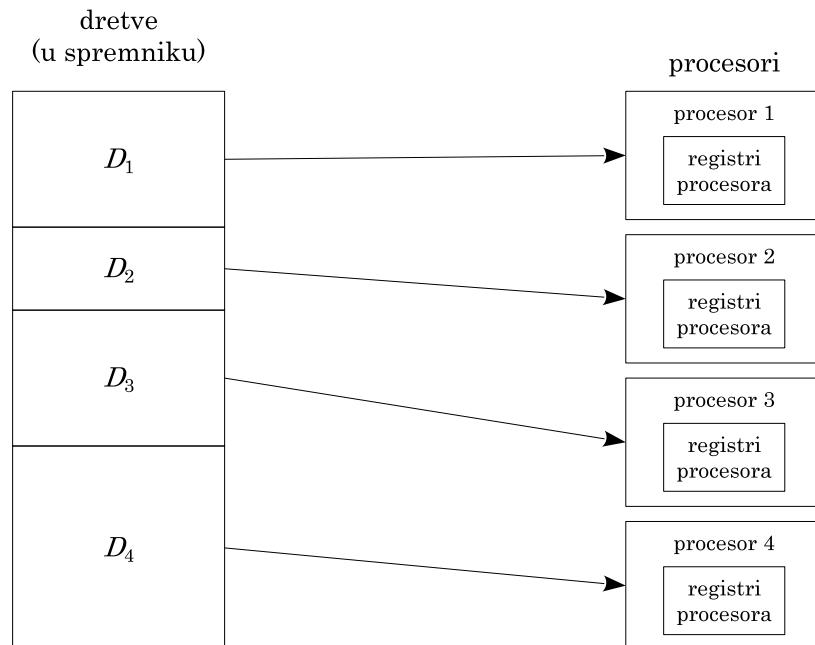
- dretva = slijed instrukcija u izvođenju ("povezanih vremenom izvođenja")

Dretva za svoje izvođenje treba:

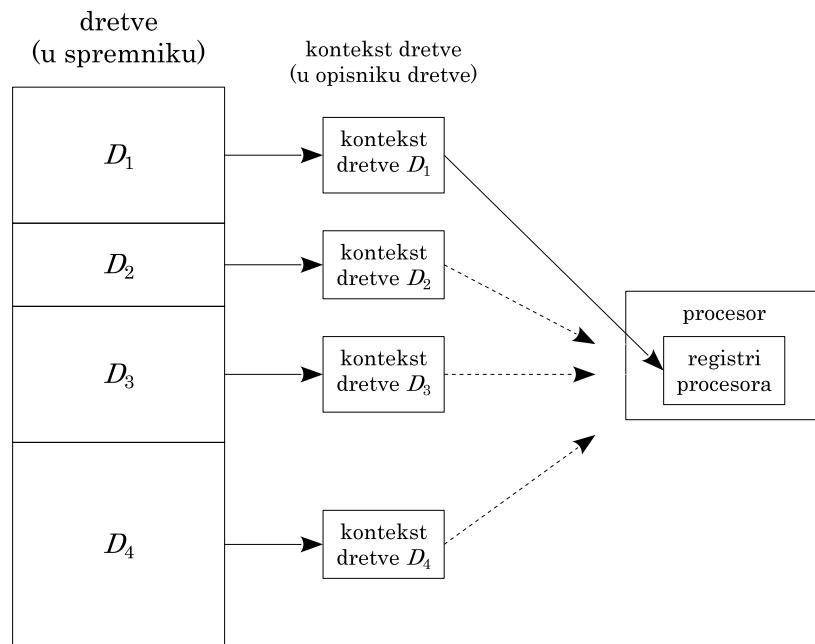
- instrukcije, podatke, stog => sve u spremniku
- registre procesora => u procesoru = *kontekst dretve*

Kako ostvariti višedretveni rad?

1. kada imamo dovoljno procesora – svaka dretva na svom procesoru
2. kada nemamo dovoljno procesora – naizmjenični rad



Slika 2.7. Višedretvenost na višeprocesorskom sustavu



Slika 2.8. 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:

- prekida se izvođenje aktivne dretve
- sprema se kontekst aktivne dretve (u za to predviđeno mjestu u spremniku = opisnik dr.)
- odabire se nova aktivna dretva

- obnavlja se kontekst novoodabrane dretve te
- ona nastavlja s radom

Primjer 2.4. Zamjena dretve (info)

Kod prve dretve (ili u obradi prekida)

```
.
.
.
CALL Rasporedi
.
```

Funkcija sustava – Rasporedi

Rasporedi :

```
SPREMI_SVE_REGISTRE_OSIM_PC kontekst_aktivne
CALL Odaberi_novu_aktivnu_dretvu //složeno možda!
OBNOVI_SVE_REGISTRE_OSIM_PC kontekst_aktivne //promjena stoga!
RET
```

Programsko brojilo se ne pohranjuje! Nije potrebno.

Varijabla `kontekst_aktivne` mora pokazivati na spremnički prostor na kojem se nalazi spremljjen kontekst dretve koju želimo aktivirati (te prije njenog micanja s procesora tamo spremamo njen kontekst)

Operacija (funkcija) `ODABERI_NOVU_AKTIVNU_DRETVU` mijenja tu varijablu tako da pokazuje na novu aktivnu dretvu

U nastavku (idućim poglavljima) koristimo prikazani model jednostavnog računala, nadograđujemo ga potrebnim sklopoljjem i slično.

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. Što je to kontekst dretve? Kako se ostvaruje višedretveni rad?

3. Obavljanje ulazno-izlaznih operacija, prekidni rad

3.1. Spajanje naprava u računalo

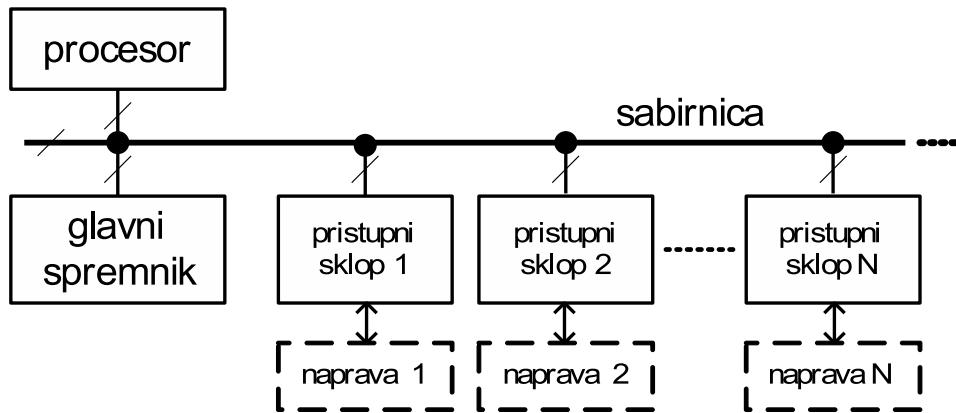
Naprave koje se spajaju u računalo imaju različita svojstva

- način rada
- brzina

Primjeri naprava: zaslon (monitor), tipkovnica, miš, zvučnici, disk, mreža, grafička kartica, ...

Sve osim procesora i spremnika (i sabirnice) su naprave (u našim razmatranjima rada sustava i upravljanja).

Zato se naprave ne spajaju izravno na (glavnu) sabirnicu, već se spajaju preko međusklopa – *pristupnog sklopa*



Slika 3.1. Spajanje UI naprava na sabirnicu

Usporedba vremenskih svojstava raznih naprava (info)

Zbog različitih potreba razne naprave rade različitim frekvencijama.

Primjerice, prema "van" potrebe su:

- tipkovnica i miš: 125-1000 Hz
- audio podsustav (izlaz): 44, 48, 96, ... kHz
- disk: čitanje jednog podatka ~ 10 ms za HDD (~ 0.1 ms za SSD); prijenos kompaktno smještenih podataka ~ 100 MB/s (i više)
- mrežna kartica: npr. 1 Gbit/s \Rightarrow 125 MB/s; odziv mreže od $\sim 0.1 \mu s$ do ~ 100 ms
- grafička kartica: npr. za 1920x1080 sa 60 Hz \Rightarrow ~ 124 MHz

Tablica 3.1. Okvirna vremena dohvata jednog podatka (info)

| operacija | trajanje | skalirano |
|-------------------------|-----------------|------------------|
| ciklus procesora | 0.3 ns | 1 s |
| dohvat registra | 0.3 ns | 1 s |
| priručni spremnik L1 | 0.9 ns | 3 s |
| priručni spremnik L3 | 12.9 ns | 43 s |
| radni spremnik | 120 ns | 6 min |
| SSD | 50 μ s | 2 dana |
| HDD | 1 ms | 1 mjesec |
| ponovno pokretanje OS-a | 3 min | 20000 godina |

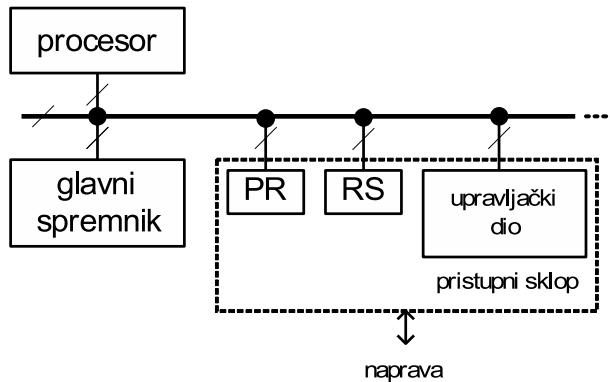
Uz podatke iz tablice 3.1. treba dodati da su to najgora vremena potrebna za dohvat. Naime, mnoge su naprave optimirane za dohvat skupa podataka, a ne samo jednog, tome je prilagođeno i korištenje naprava kroz međuspremnik. Stoga su njihova svojstva znatno bolja kada se gleda u "prosječnim" vremenima. Naime, iako se možda koriste podaci iz s diska prosječno vrijeme pristupa tim podacima može biti u rangu L1 priručnog spremnika ili i brže kada se koriste registri i kada je obrada nešto dulja. Na tom se mogućnošću zasniva i upravljanje spremnikom metodom straničenja (o tome više u 8. poglavlj).

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



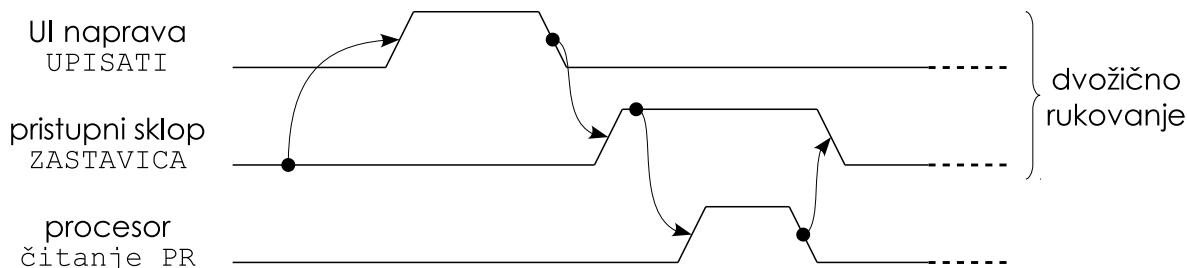
Slika 3.2. Pristupni sklop UI naprave

PR – podatkovni registar, služi za prijenos podataka

RS – registar stanja, sadrži ZASTAVICA-u koja pokazuje je li pristupni sklop spreman za komunikaciju s procesorom (ZASTAVICA==1) ili nije (ZASTAVICA==0)

Procesor u petlji čita RS dok ZASTAVICA ne postane jednaka 1

Brisanje ZASTAVICE po komunikaciji s procesorom može biti ostvareno u pristupnom sklopu (npr. čitanje/pisanje po PR se može detektirati te obrisati zastavicu) ili to može procesor napraviti upisom vrijednosti u RS.



Slika 3.3. Primjer sinkronizacije između naprave, pristupnog sklopa i procesora

Isječak kôda 3.1. Primjer čitanja jednog podatka s naprave

```

...
ADR R0, RS           //adresa registra stanja u R0
ADR R2, PR           //adresa podatkovnog registra u R2

petlja:
    LDR R1, [R0]      ;pročitaj registar stanja s ZASTAVICA-om
    CMP R1, 0          |
    BEQ petlja         |   \  radno
                           |   | čekanje
                           |
    LDR R1, [R2]      // pročitaj znak iz pristupnog sklopa u registar R1
    ...//obrada pročitanog podatka, npr. samo spremanje u spremnik

```

Svojstva radnog čekanja kao načina upravljanja UI:

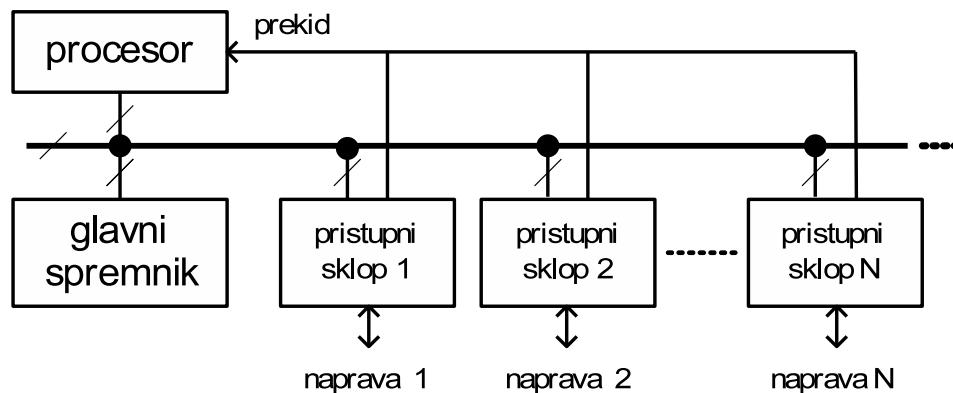
1. prednost: sklopolje je vrlo jednostavno
2. nedostatak: procesor ne radi produktivno, nema koristi od tisuća iteracija petlje
 - a) drukčijim programom to se može ponekad ublažiti, npr. pojedini sklop se provjerava periodički, provjerava se više sklopova ("prozivanje"), ... (nije uvjek primjenjivo)

3.3. Prekidni rad

Ideja: naprava sama, kada postane spremna traži obradu slanjem signala – prekidnog signala (električnog signala preko vodiča) procesoru.

3.3.1. Prekidni rad bez sklopa za prihvatanje prekida

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



Slika 3.4. 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 prihvati prekidni signal (za 1.) ili ga privremeno ignorirati (za 2.).

Prihvatanje signalata 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 drugčije ne definira)

Načini rada procesora

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

Postupak prihvata prekida od strane procesora

(što 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) na kraju instrukcije:
 - + ako su u procesoru prekidi omogućeni on provjerava je li prekidni signal postavljen
 - + ako je prekidni signal postavljen tada **procesor** prihvata prekid u slijedećim 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 i koristi taj stog
 3. na stog (sustavski) pohrani programsko brojilo (PC) i registar stanja (SR) (tzv. minimalni kontekst)
 4. u PC stavi adresu "prekidnog potprograma"
 - gdje je ta adresa zapisana? ugrađena u procesoru, ili preko tablice, npr. IDT (IDTR!)

Navedeno je **ugrađeno ponašanje procesora**.

Opis rada procesora (dohvati/dekodiraj/izvedi) treba proširiti tim dijelom na kraju instrukcije.

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

```
Prekidni potprogram
{
    pohrani kontekst na sustavski stog (osim PC i SR koji su već тамо);
    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; // poziv funkcije upravljačkog programa

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

```
instrukcija: vradi_se_u_prekinutu_dretvu
{
    obnovi PC i SR sa sustavskog stoga;
    prebaci (vradi) 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).

Kako uobičajeno izgleda obrada prekida naprave ("prava obrada")?

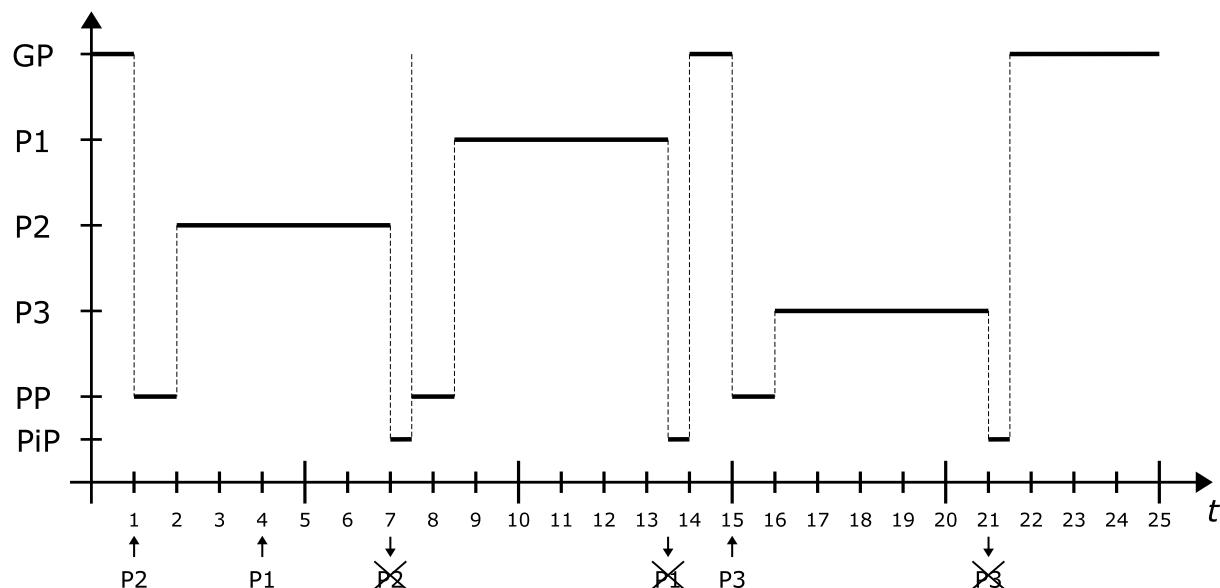
```
obrada_prekida_naprave_I
{
    dohvati PR_I;
    obradi podatak; // "svašta" ili samo pohrana negdje u spremnik
    // ili ako se radi o izlaznoj napravi samo: pohrani nešto u PR_I
}
```

Zadatak: 3.1. Primjer obrade niza zahtjeva za prekid

U nekom sustavu javljaju se prekidi: P1 u 1. ms, P2 u 4. ms te P3 u 15. ms. Obrada svakog prekida traje 5 ms. Postupak prihvata prekida (PP, od reakcije procesora na prekid do poziva funkcije za obradu prekida) neka traje 1 ms. Povratak iz prekida (PIP, nakon dovršetka obrade – obnova konteksta i povratak u dretvu) neka traje 0,5 ms.

- Grafički prikazati rad procesora u glavnom programu (GP), obradama P1, P2 i P3 te kućanskim poslovima PP i PIP.
- Koliko se ukupno vremena potroši na kućanske poslove?
- Koliko se odgađa obrada GP zbog ta tri prekida?

a)



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

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

Svojstva upravljanja napravama prekidom, bez sklopa za prihvatanje prekida

- radi jednostavno, nije potrebno složeno sklopovlje
- problem: za vrijeme jedne obrade svi ostali zahtjevi MORAJU pričekati kraj obrade prethodnog prekida
 - u nekim sustavima to može biti veliki nedostatak jer nisu svi prekidi istog značaja, neki su bitniji (prioritetniji) od drugih
 - ako želimo obradu prema prioritetima prekida tada to možemo riješiti:
 - * programski (popraviti gornje programe) ili
 - * sklopovski
 - u nekim se sustavima podrazumijeva da obrade prekida traju vrlo kratko te je i ovakav način prihvata i obrade prekida dovoljno dobar (većina sustava za rad u stvarnom vremenu!)

3.3.2. Obrada prekida prema prioritetima, bez sklopa za prihvatanje prekida

Ideja programskog rješenja:

- sama obrada prekida („čista obrada“) neka se obavlja s dozvoljenim prekidanjem
- na svaki zahtjev za prekid pozvat proceduru koja će ustanoviti prioritet zahtjeva i usporediti ga s trenutnim poslom:
 - ako je zahtjev prioritetniji ide se odmah na njega
 - u protivnom, samo se zabilježi taj zahtjev te se nastavlja s prekinutom obradom
- prekidima (napravama) treba dodijeliti prioritete
 - u nastavku je prioritet određen brojem naprave, veći broj \Rightarrow veći prioritet

Podatkovna struktura:

- TEKUĆI_PRIORITET – prioritet tekućeg posla, 0 za dretvu, 1 za obradu prekida naprave rednog broja (prioriteta) 1
- OZNAKA_ČEKANJA[N] – polje od N elemenata (N je najveći prioritet)
 - označava da li dotočna naprava čeka na početak obrade (zbog prioritetnijih pr.)
- KON[N] – rezervirano mjesto u spremniku za pohranu konteksta dretve pri obradi pojedinog prekida; uz kontekst spremi se i tekući prioritet

```

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

    ponavljam {
        max = 0; //indeks najprioritetnije naprave koja čeka na obradu
        za i = TEKUĆI_PRIORITET + 1 do N radi
            ako je ( OZNAKA_ČEKANJA[i] != 0 )
                max = i;

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

            omogući prekidanje;
            obrada prekida naprave (max);
            zabrani prekidanje;

            sa KON[max] obnovi TEKUĆI_PRIORITET, a kontekst stavi na sustavski stog;
        }
    }
    dok je ( max > TEKUĆI_PRIORITET );

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

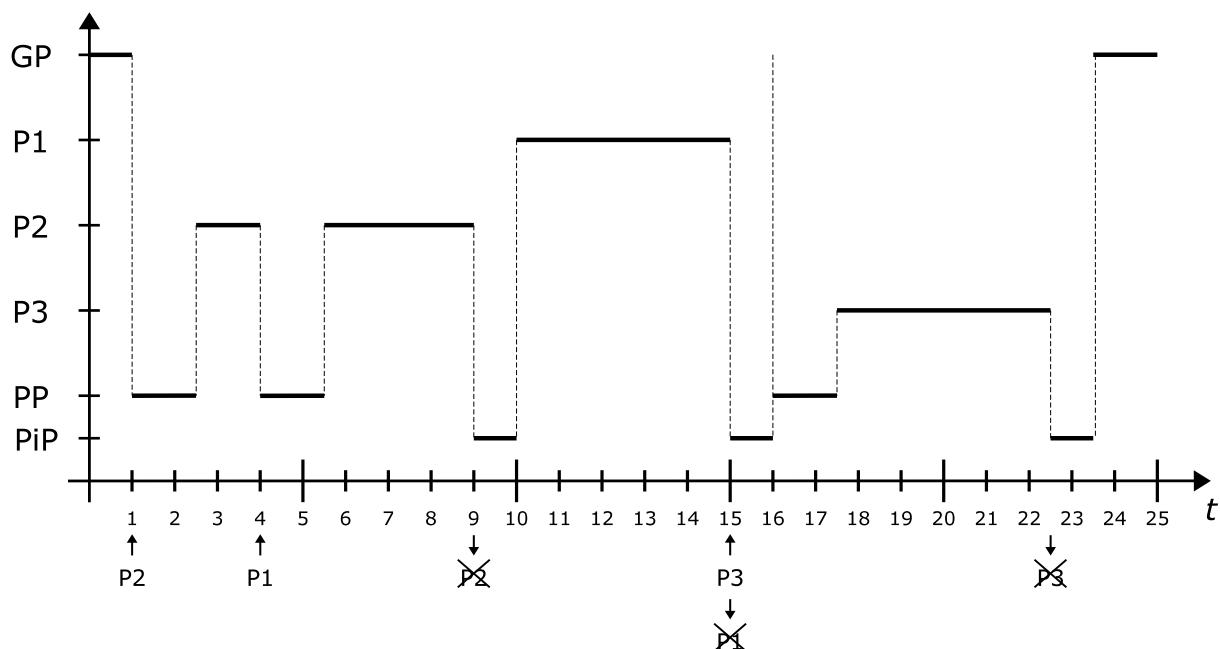
```

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

U nekom sustavu koji nema sklopo za prihvatanje 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, od reakcije procesora na prekid do poziva funkcije za obradu prekida) neka traje 1,5 ms. Povratak iz prekida (PIP, nakon dovršetka obrade – obnova konteksta i povratak u dretvu) neka traje 1 ms.

- Grafički prikazati rad procesora u glavnom programu (GP), obradama P1, P2 i P3 te kućanskim poslovima PP i PIP.
- Koliko se ukupno vremena potroši na kućanske poslove?
- Koliko se odgada GP zbog ta tri prekida?

a)



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

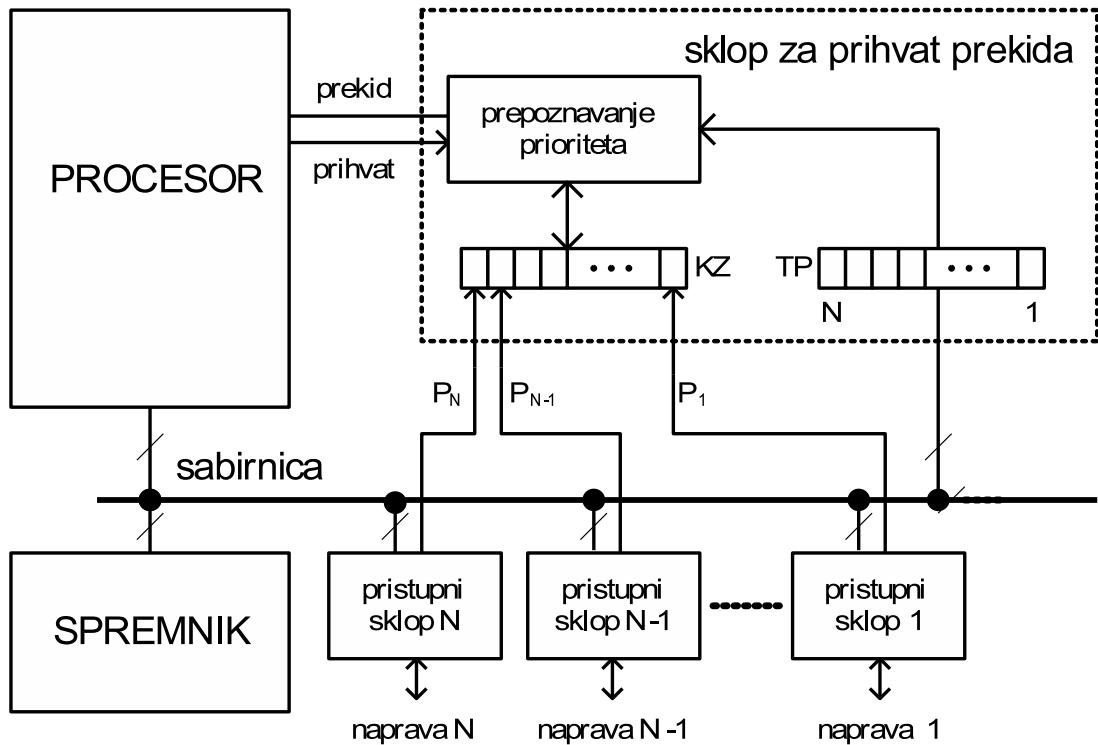
c) GP prekinut u 1. ms, nastavlja u 23,5 ⇒ 22,5 ms je odgođen

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? korištenjem dodatnog sklopovlja!

3.3.3. Obrada prekida korištenjem sklopa za prihvatanje prekida

Sustav sa sklopom za prihvatanje prekida (to više nije sustav s "minimalnim sklopoljem" kao prije)



Slika 3.5. Sklop za prihvatanje prekida

KZ = kopije ZASTAVICA pristupnih sklopova, naprave koje čekaju na obradu postavljaju i brišu odgovarajuće bitove KZ -a

TP = tekući prioritet; bit na mjestu i je postavljen ako se zahtjev prioriteta i započeo obrađivati; indeks najznačajnijeg postavljenog bita TP -a označava tekući prioritet

Sklop za prepoznavanje prioriteta uspoređuje KZ i TP te ako postoji zahtjev većeg prioriteta od tekućeg generira zahtjev za prekid prema procesoru ($\text{msb}(KZ) > \text{msb}(TP)$, gdje msb označava funkciju koja izračunava indeks najznačajnijeg postavljenog bita).

Prekidni potprogram za prihvatanje i obradu prekida bi u ovom slučaju mogao izgledati:

```

prekidni potprogram {
    pohrani kontekst na sustavski stog (osim PC i SR);
    odredi uzrok prekida (dohvati KZ) => i;
    signaliziraj napravi da je njen prekid prihvaćen (te da spusti prekidni signal);
    TP[i] = 1;

    dozvoli prekidanje;
    obradi prekid (i);
    zabrani prekidanje;

    TP[i] = 0
    obnovi kontekst sa sustavskog stoga (osim PC i SR);
    vrati_se_u_prekinutu_dretvu;
}

```

Svojstva:

- + sa sklopom za prihvat prekida izbjegavaju se nepotrebna prekidanja => propuštaju se samo prioritetniji zahtjevi!
- + manje "kućanskih poslova" (kraće traje prihvat prekida)
- potreban je sklop

U idućim poglavljima će se podrazumijevati da sustav posjeduje sklop za prihvat prekida, koji će od sada biti dio procesora.

Zahtjevi za prekid iz zadataka 3.1. i 3.2. bi se korištenjem sklopa obradili redoslijedom kao u 3.1. zbog toga što se u tim primjerima ne pojavljuje prekid veće prioriteta koji bi prekinuo obradu prekida manjeg prioriteta. Međutim, obzirom da se kod sustava koji imaju sklop ne radi prozivanje naprava već se izvor prekida očitava iz registra sklopa za prihvat prekida (KZ), sam postupak prihvata prekida je nešto brži (npr. mogli bi postaviti 0,5 ms za prihvat te 0,5 ms za povratak iz prekida).

Zadatak: 3.3. (ispitni zadatak)

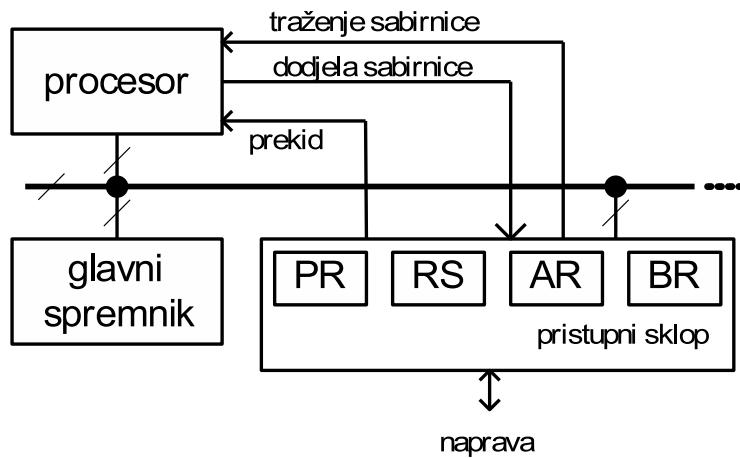
U nekom sustavu javljaju se prekidi P1 u 3. ms, P2 u 1. ms te P3 u 4. ms. Prioritet prekida određen je brojem (P3 ima najveći prioritet). Obrada svakog prekida traje po 3 ms. Grafički prikazati aktivnosti procesora u glavnom programu (GP), procedurama za obradu prekida (Pi) te procedurama za prihvat 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 vremena prihvata prekida (PP) od 1 ms (*uključuje potragu za izvorom prekida – zahtjevi većeg prioriteta se prvi prihvaćaju*) te vremenu 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 vrijeme 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*)
- d) **[sa sklopom za prihvat prekida]** u sustavu sa sklopom za prihvat prekida uz vrijeme prihvata prekida (*pohrana konteksta prekinute dretve*) od 0,5 ms (PP) te vrijeme povratka iz prekida (*obnove konteksta*) od 0,5 ms (PiP)

Dodatna "nepisana" pravila za rješavanje:

- ako u istom trenutku neka obrada završava i pojavljuje se novi prekid prepostavljamo da je ipak najprije završila obrada, a potom se dogodio prekid
 - ako se u istom trenutku pojavi više prekida: za svaki prekid poziva se zasebni PiP, s time da svaki put prvo za zahtjev najvećeg prioriteta (koriste se ispitni lanci)
-

3.4. Korištenje UI naprava sa sklopovima za neposredni pristup spremniku



Slika 3.6. Sklop s izravnim pristupom spremniku

- PR – podatkovni registar
- RS – registar stanja
- AR – adresni registar – od kuda/kamo se učitavaju/pohranjuju podaci
- BR – brojilo podataka – koliko podataka još za prenijeti
- UI pristupni sklop sam prenosi podatak iz PR u spremnik ili obratno
- Za korištenje sabirnice traži dozvolu procesora
- Po zahtjevu procesor prepušta idući sabirnički ciklus zahtjevu (prepostavka)

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

```
dok je ( BR > 0 ) {
    čekaj na podatak VJ; //ili na spremnost VJ za prihvatanje 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 sabirnički ciklus

Ovakav sklop je pogodan kada treba prenijeti veću količinu podataka, inače samo "programiranje" postaje utjecajno (nije samo jedan sab. ciklus)

Kada se prenesu svi podaci sklop izaziva prekid te ga procesor (eventualno) opet inicijalizira.

Primjer 3.1. 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
 - $1000 \text{ (zn. u 1 s)} * 10 \text{ (instr.)} / 10\ 000\ 000 \text{ (instr./s)} = 0,01\% \text{ korisnog rada}$
 - (ili $10 / 10\ 000 = 0,01\%$ korisnog rada)
- prekidi: ~ 200 instr. po prekidu => $1000 * 200 / 10\ 000\ 000 = 2\% => 98\% \text{ ostane!}$
 - (ili $200 / 10\ 000 = 2\%$, 98% ostaje)
- DMA: $1000 / 10\ 000\ 000 = 0,01\% => 99,99\% \text{ ostane! (ne računajući obradu prekida!)}$
 - (ili $1 / 10\ 000 = 0,01\%$, 99.99% ostaje)

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

- radno čekanje: $10\ 000\ 000 / 10 = 1000\ 000 \text{ znakova/s}$ uz 100% opt. procesora
 - prekidi: $10\ 000\ 000 / 200 = 50\ 000 \text{ znakova/s}$ uz 100% opt. procesora
 - DMA: $10\ 000\ 000 / 2 = 5000\ 000 \text{ znakova/s}$ uz 50% opt. procesora (svaki 2.s.c.)
-

3.5. Prekidi generirani unutar procesora, poziv sustavskog potprogramma

a) Pri radu, procesor izaziva neke greške:

- dijeljenje s nulom
- nepostojeća adresa (operanda, instrukcije)
- nepostojeći operacijski kod
- instrukcija se ne može izvesti s trenutnim ovlastima
- ...

Takvu dretvu treba prekinuti (zaustaviti)

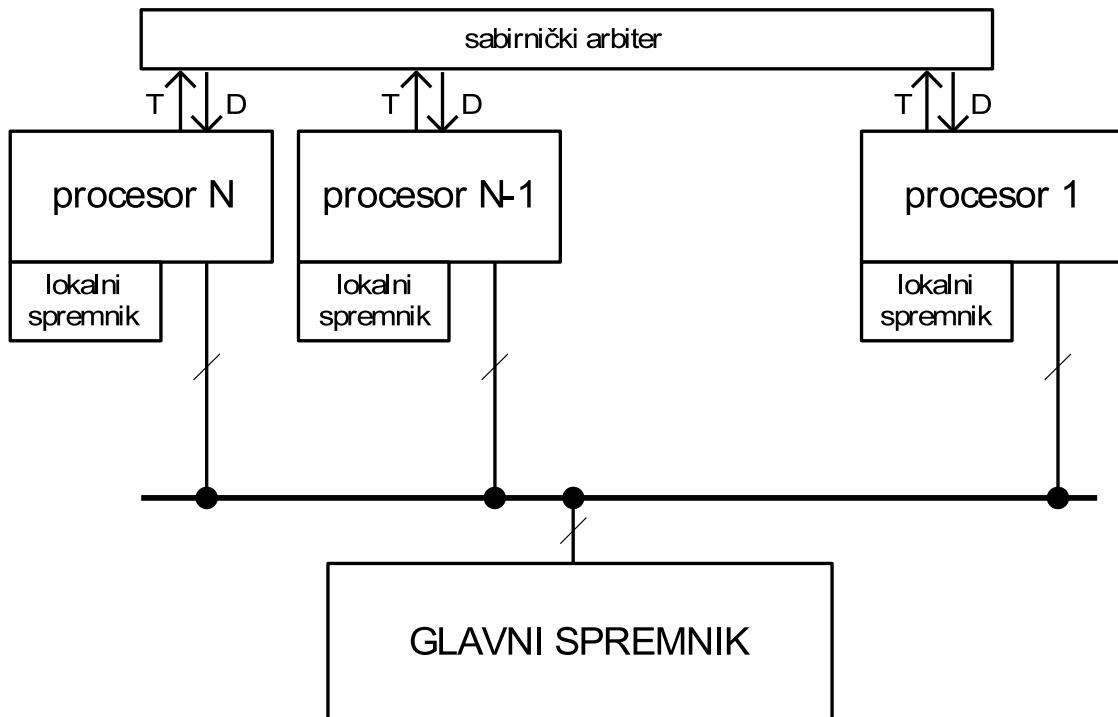
- procesor sam izaziva prekid; u obradi prekida dretva se prekida i miče iz sustava

b) Kako dretva obavlja operacije koje zahtijevaju više privilegije?

- ona namjerno izaziva prekid – *programska prekid* te se poziva *jezgrina funkcija*
- jezgrine funkcije = unaprijed pripremljena, dio sustava, dretva ju ne može promijeniti da izvodi njen kod
- mehanizam zaštite je ugrađen preko programskog prekida
- ako dretva nešto želi ona to traži od jezgre (o tome kasnije)

3.6. Višeprocesorski (sabirnički povezani) sustavi

Ideja – proširiti DMA pristup, svaki DMA sklop procesor, a “stari” procesor arbiter prema sabirnici



Slika 3.7. 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)

[dodatak]

Pojmovi:

- simetrični više procesorski sustavi – SMP
- homogeni — II —
- NUMA – Non-Uniform Memory Access

3.7. Prekidi u "stvarnim sustavima"? (info)

Primjer Intelove arhitektura (pojednostavljen)

a) prihvatanje prekida od strane procesora

- obrada prekida se programira preko zasebne tablice – IDT (interrupt description table)
- u registar procesora IDTR se stavlja "adresa" te tablice
- kad se dogodi prekid, uz prekid dolazi i informacija o uzroku – broj prekida = N
- broj prekida se koristi kao indeks za IDT, uzima se N -ti redak i tamo piše koju funkciju treba pozvati za obradu tog prekida

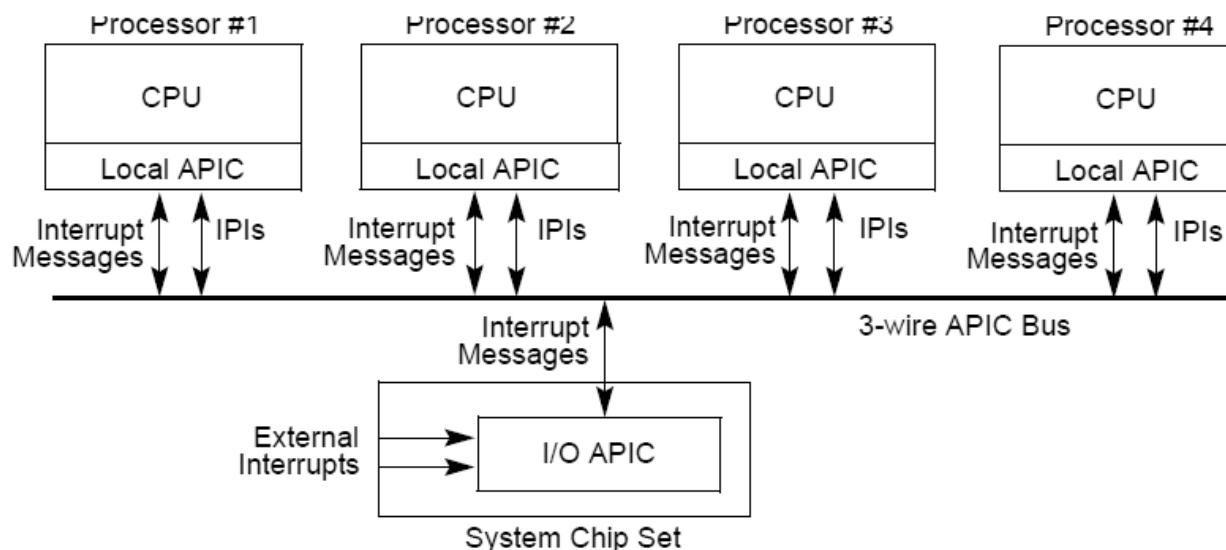
b) izvori prekida – "proslijedivanje prekida do procesora" (novije APIC sučelje)

(APIC – advanced programmable interrupt controller)

- uz procesor nalazi se lokalni APIC sklop (Local APIC) koji:
 - prima i proslijedi "lokalne" prekide procesoru:
 - * lokalno spojeni uređaji, prekidi lokalnog brojila (sata), greške, ...
 - prima poruke o prekidima uređaja spojenih na I/O APIC
 - prima/šalje poruke od/prema lokalnih APIC-a drugih procesora (u višeproc. sustavima)
- u sustavu postoji I/O APIC sklop na koji su spojeni "vanjski" izvori prekida
 - sa lokalnim APIC-ima komunicira preko sabirnice (šalje poruke o prekidima)
 - može se programirati: koje prekide proslijedi i kome

Upravljanje prekidima naprava – zabrana prihvata prekida:

- na razini procesora
- na razini lokalnog APIC-a
- na razini I/O APIC-a
- na razini naprave (nju se isto može programirati da ne generira zahtjeve za prekid)



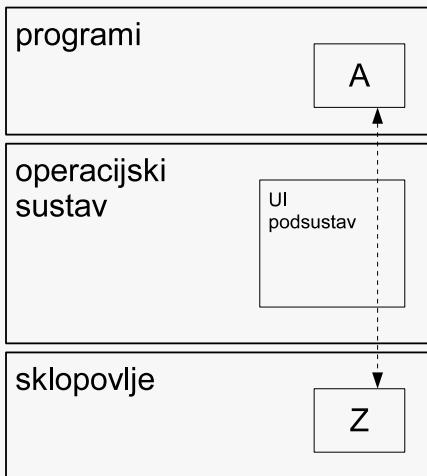
Prihvatanje prekida sa strane operacijskih sustava

- podjela obrade u dva dijela:
 - prvi, odmah po primjeku prekida, kraći, uz zabranjeno prekidanje
 - drugi, duži, naknadno
- Windows: ISR (Interrupt Service Routine) + IST (Interrupt Service Thread)
- Linux: Top half, Bottom half (zapravo malo složenije u novijim, ali se svodi na slično)

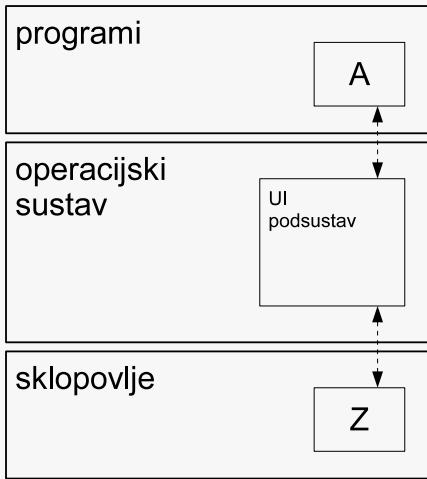
3.8. "Upravljački programi" (engl. *device drivers*) (info)

- predstavljaju skup funkcija koje omogućuju korištenje pojedinog sklopoljja
- primjerice za neku jednostavnu UI napravu upravljački program bi minimalno trebao imati funkcije:
 - inicijaliziraj
 - * početna inicijalizacija naprave (pisanje u upravljačke registre, čitanje registara stanja, ...)
 - status
 - * dohvati stanja naprave (ima li novi podatak, može li se slati, ...)
 - pošalji
 - * slanje podataka prema napravi
 - pročitaj
 - * čitanje podataka od naprave
- OS pri inicijalizaciji upravljačkog programa za neku napravu treba registrirati prekid koji naprava generira te u obradi tog prekida pozvati funkcije upravljačkog programa
 - primjerice, OS treba imati funkciju `registriraj_prekid (id_prekida, funkcija)`
 - u registriranoj funkciji (`funkcija`) treba na osnovi dobivenih podataka (koje daje prekidni podsustav) pogledati što se dogodilo s napravom koja je izazvala prekid te pozvati odgovarajuću proceduru

Primjer 3.2. Primjer korištenja naprave kroz upravljačke programe (idejno)



kako?

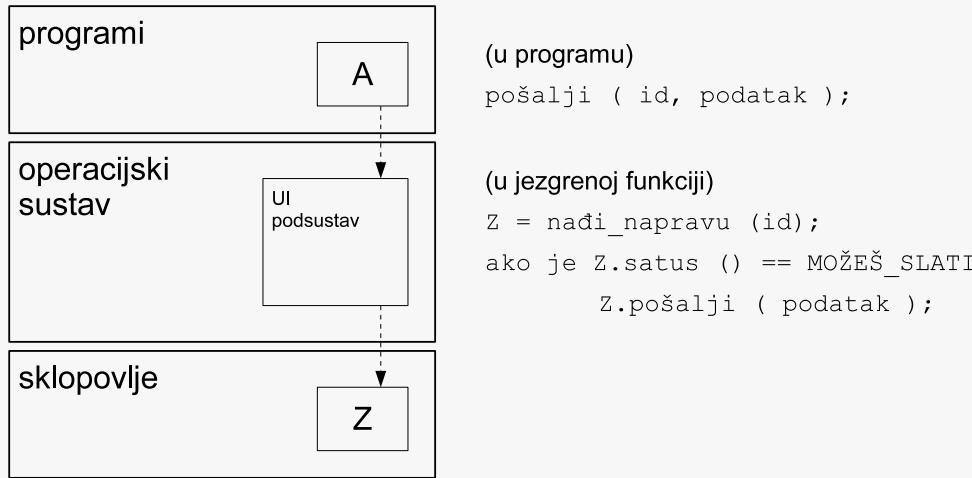


"A" koristi sučelje OS-a:
pošalji (id, podatak);
pročitaj (id, podatak);

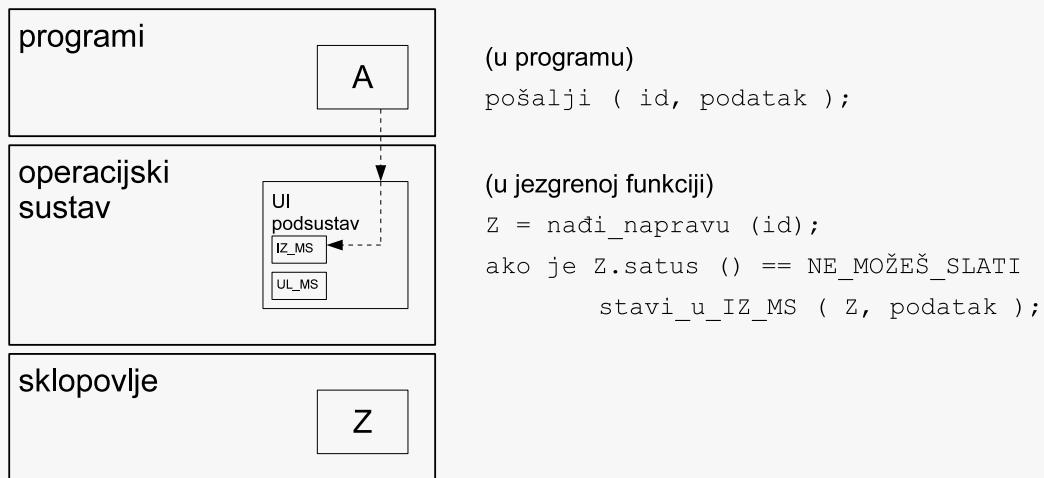
kako?

OS koristi upravljački program naprave "Z":
Z.pošalji (podatak);
Z.pročitaj (podatak);
Z.satus ();

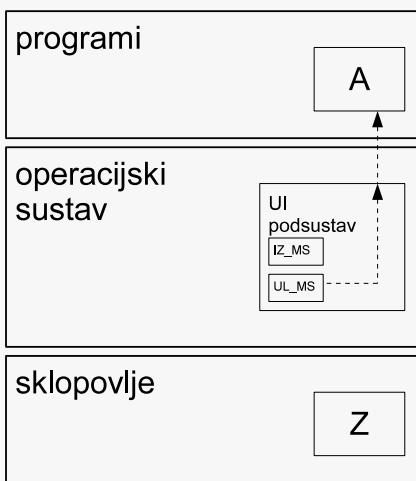
Slanje prema napravi (1)



Slanje prema napravi (2)



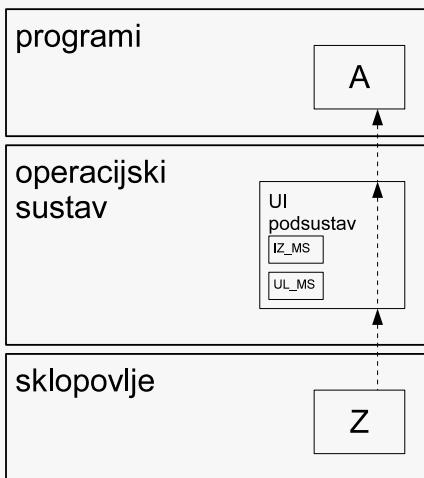
Čitanje s naprave (1)



(u programu)
`pročitaj (id, podatak);`

(u jezgrenoj funkciji)
`Z = nađi_napravu (id);`
`ako je UL_MS(Z) neprazan`
`pročitaj_UL_MS (Z, podatak);`

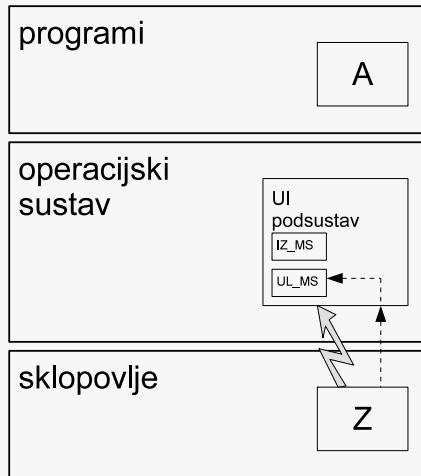
Čitanje s naprave (2)



(u programu)
`pročitaj (id, podatak);`

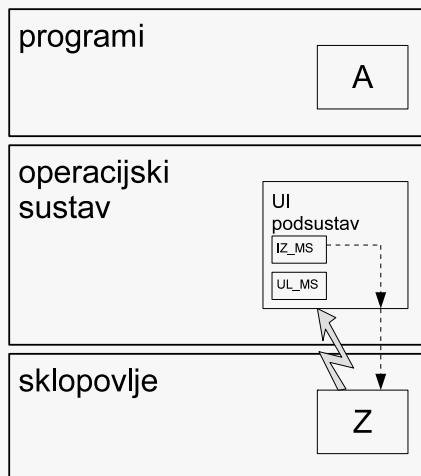
(u jezgrenoj funkciji)
`...`
`inače ako je Z.status () == IMA_PODATAKA`
`Z.pročitaj (podatak);`
`inače`
`vrati 0; //nema podataka`

Prekid naprave Z (1)



```
(u obradi prekida naprave)
Z = nađi_napravu (id_prekida);
ako je Z.status () == IMA_PODATAKA
    Z.pročitaj ( UL_MS );
```

Prekid naprave Z (2)



```
(u obradi prekida naprave)
...
inače ako je Z.status () == SVE_POSLANO
    ako je IZ_MS neprazan
        Z.pošalji ( IZ_MS );
    //inače: nema ništa za slanje
    //inače: prekid iz drugih razloga ...
```

Isječak kôda 3.2. Primjer prekidnog podsustava i korištenja upravljačkog programa

```

/* prihvati prekida vanjskih naprava ("prekidni podsustav") */
Prekidni_potprogram () /* slično prethodno opisanim načinima prihvata */
{
    pohrani_kontekst;
    idp = ustanovi_uzrok_prekida ();
    za svaku napravu "upp" koja je registrirana za prekid "idp"
    {
        status = upp.status ();
        ako je ( status == PRISTIGLI_NOVI_PODACI ) tada
        {
            upp.pročitaj ( UL_MS );
            (pogledaj kome podaci trebaju ...)
        }
        inače ako je ( status == MEĐUSPREMNIK_PRAZAN ) tada
        {
            ako je ( IZ_MS neprazan )
                upp.pošalji ( IZ_MS );
            }
            inače ako je ...
        }
        povratak_iz_prekida;
    }
    /* dodavanje naprave s upravljačkim programom */
    os_dodaj_napravu ( upravljački_program upp )
    {
        upp.inicijaliziraj ();
        registriraj_prekid ( dohvati_id_prekida ( upp ), upp );
    }
}

```

Prekidni podsustav je dio operacijskog sustava. Program može izravno tražiti komunikaciju s nekom napravom (preko OS-a), ali i tražiti 'da bude obaviješten' kad se s napravom nešto dogodi. Kao reakciju na prekid naprave operacijski sustav može poslati poruku/signal nekom procesu. Primjerice, pritisak na tipku, pomak miša i sl. operacijski sustav može proslijediti programu (prozoru u fokusu) kao poruku.

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 prihvata prekide (postupak prihvata)?
6. Što je to sklop za prihvat prekida? Koje prednosti donosi njegovo korištenje?
7. Na koje sve načine se mogu prihvataći i obrađivati prekidi?
8. Što su to "upravljački programi" (engl. *drivers*)?
9. Koji se problemi javljaju pri/zbog obrade prekida? (zabrana prekidanja, prioriteti)
10. Koji su sve izvori prekida? (izvan procesora, prekidi izazvani u procesoru-koji?)
11. Skicirati ostvarenje višeprocesorskog sustava. Kako se upravlja zajedničkom sabirnicom u takvom sustavu? Čemu služe priručni spremnici uz procesor?
12. 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)
 - d) sa sklopopom za prihvat prekida, uz trajanje prihvata prekida (PP) od 0,5 ms te 0,5 ms za povratak iz prekida (PiP).
- Odrediti stanje sustava i vrijednosti korištenih struktura podataka u $t=8.5$ ms.
13. 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 33 MHz.

4. Međusobno isključivanje u višedretvenim sustavima

4.1. Osnovni pojmovi

4.1.1. 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.

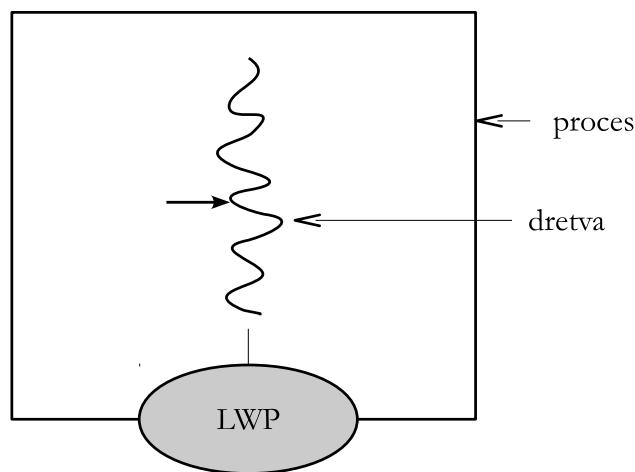
Dretva je niz instrukcija koji se izvodi.

Proces se sastoji od *barem* jedne dretve

[dodatak]

Prije, kada se proces sastojao samo od *samo* jedne dretve, pojam dretve se nije ni koristio.

Nove dretve u procesu stvaraju se izvođenjem dretve i pozivom odgovarajućih funkcija OS-a.

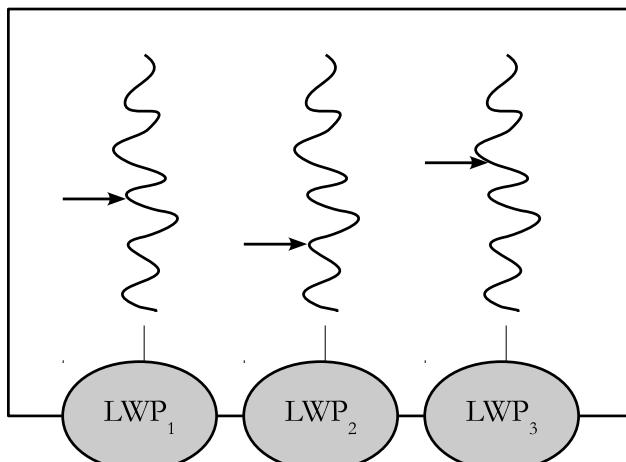


Slika 4.1. Tradicionalni proces

LWP:

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

Danas: proces se sastoji od *barem* jedne dretve

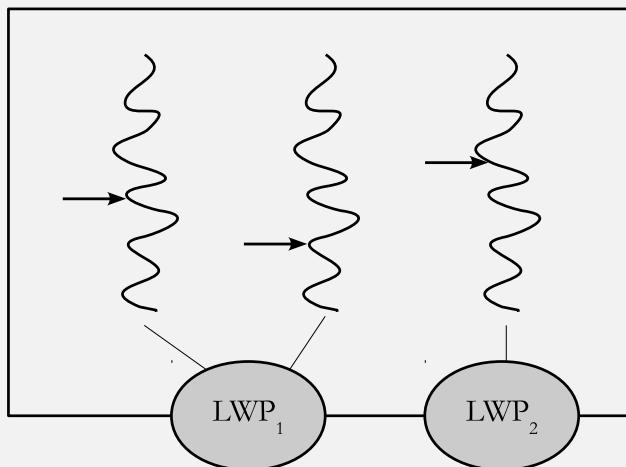


Slika 4.2. Moderan proces

Standardno, OS vidi i upravlja svim dretvama.

[dodatak]

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



Slika 4.3. Proces s vlastitim upravljanjem dretvi

Skup zauzetih sredstava je isti za sve dretve istog procesa.

- postoji zajednički spremnik
 - cijeli adresni prostor 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 OS-a

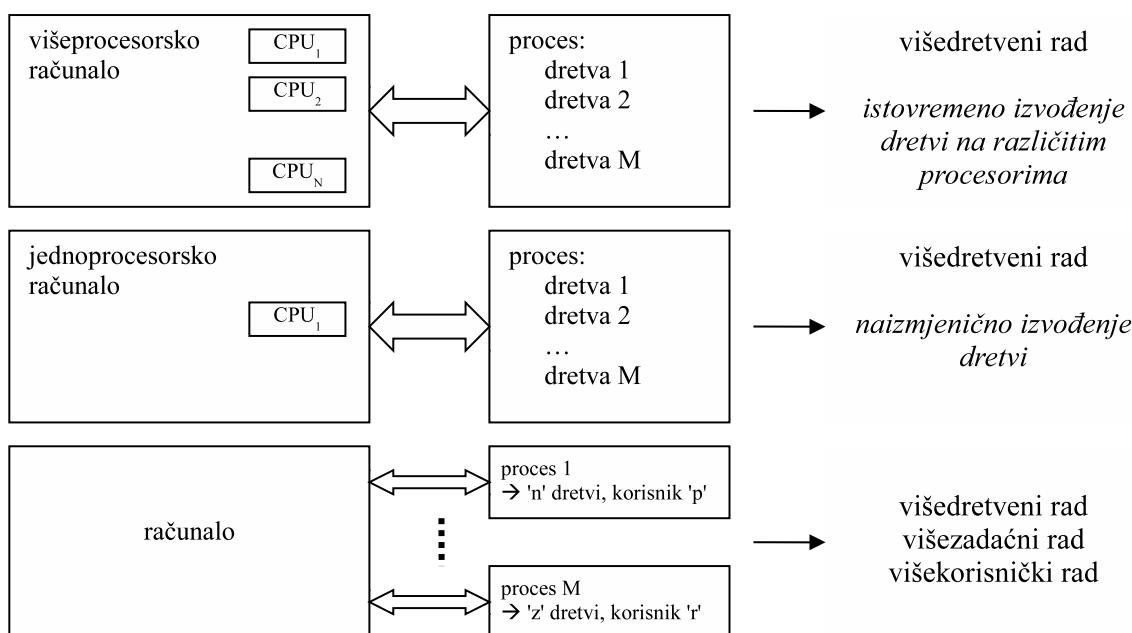
4.1.2. Višedretvenost (info)

Višedretveni rad – izvođenje više dretvi u paraleli (ne nužno i istovremeno)

Višezadačni rad – više zadataka (različitih problema/programa) odjednom. Svaki zadatak izvodi se u zasebnoj dretvi (ili i više njih ako je rastavljen na podzadatake)

Kod *višeprocesorskih računala* više dretvi se može paralelno izvoditi na različitim procesorima

Kod *jednoprocesorskih računala* (kao i kod višeprocesorskih kada imamo više dretvi od procesora) višedretveni rad ostvaruje se *naizmjeničnim radom dretvi* na dostupnim procesorima



Slika 4.4. Usporedba raznih višedretvenih sustava

Današnji (moderni) operacijski sustavi su i višedretveni i višezadačni i višekorisnički (pogledati procese na UNIX/Win32 sustavima).

4.1.3. Zašto koristiti višedretvenost? (info)

Nedostaci:

- višedretvenost je skupa za OS, podrška je jako složena!!! (kao što će se i vidjeti u nastavku)
- višedretveno programiranje je složeno i podložno greškama
- potrebni su mehanizmi sinkronizacije i komunikacije

Korist:

- višezadaćnosti – više poslova paralelno
- paralelno koristiti elemente računala
 - iskoristiti višeprocesorske sustave – intenzivni računalni problemi (CPU) koji se daju rastaviti na (bar djelomično) neovisne dijelove
 - dok jedna dretva čeka dovršetak UI operacije (naprava “radi/izvodi” dretvu), procesor izvodi drugu dretvu
- asinkrono upravljanje događajima, npr. Web poslužitelj na novi zahtjev stvara novu dretvu koja će poslužiti zahtjev (ili bira postojeću) (obrada prekida: IST, bottom half)
- složeni (kompleksni sustavi) – principom “podijeli i vladaj” smanjujemo složenost odvajanjem zasebnih aktivnosti u zasebne dretve (ali oprezno!)

4.2. Višedretveno ostvarenje zadatka – zadatak i podzadaci

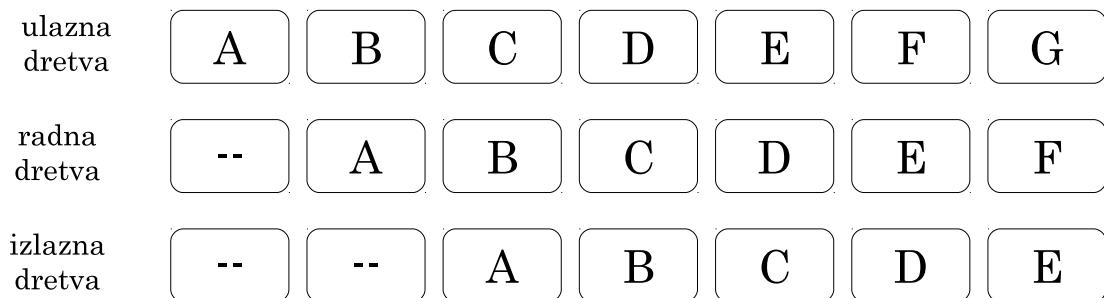
Svaki se zadatak, barem i "umjetno" može podijeliti na podzadatke, ako to nije očito iz strukture zadatka (dijelovi, paralelna obrada, ...)

Zadatak se može podijeliti na (jedna od mogućih podijela):

- 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 principom cjevovodnog rada



Slika 4.5. Princip cjevovodnog rada dretvi

Što ako poslovi pojedinih dretvi ne traju jednak? Dretve je potrebno *sinkronizirati!* Npr. ulazna dretva treba prije predaje podataka radnoj dretvi pričekati da radna dretva dovrši započetu obradu (rad na prethodno predanim podacima).

Mnogi zadaci se mogu (smisleno) rastaviti na podzadatke (npr. $Z : Z_1 \rightarrow Z_2 \rightarrow \dots \rightarrow Z_N$). Ipak, i u takvim slučajevima potrebno je sinkronizirati zadatke:

- uređiti slijed njihova izvođenja – tko prije a tko poslije
- osigurati pojedinačno korištenje zajedničkih sredstava

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

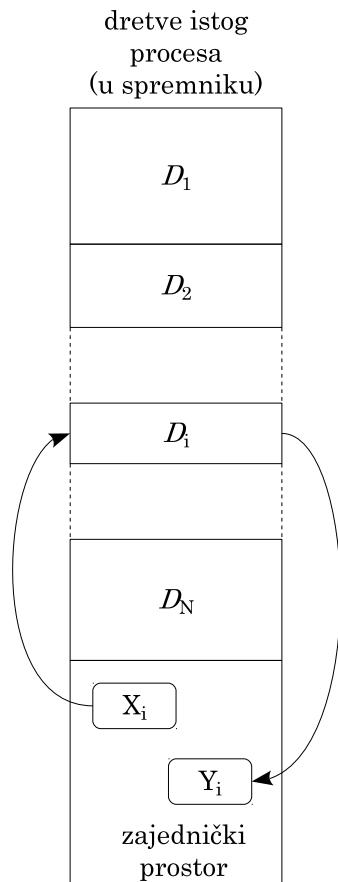
Problem nastaje kada jedna dretva treba čekati na drugu

- Kako to ostvariti?
- Kako koristiti iste podatke uz očuvanje konzistentnosti (da ne prepisuju podatke)?
- Potrebna je SINKRONIZACIJA
 - najjednostavniji oblik sinkronizacije je *međusobno isključivanje*
- Koje je dretve potrebno sinkronizirati?
- Kako ostvariti sinkronizaciju?

4.3. Model višedretvenosti, nezavisnost dretvi

Prepostavke:

- 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 4.6. 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 funkcijски opisati kao preslikavanje: $Y_i = f(X_i)$

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

Dva su podzadataka *nezavisna* ako nemaju nikakvih zajedničkih spremničkih lokacija ili imaju pre-sjeka samo u domenama.

Uvjet nezavisnosti podzadataka odnosno dretvi D_i i D_j :

$$(X_i \cap Y_j) \cup (X_j \cap Y_i) \cup (Y_i \cap Y_j) = \emptyset \quad (4.1.)$$

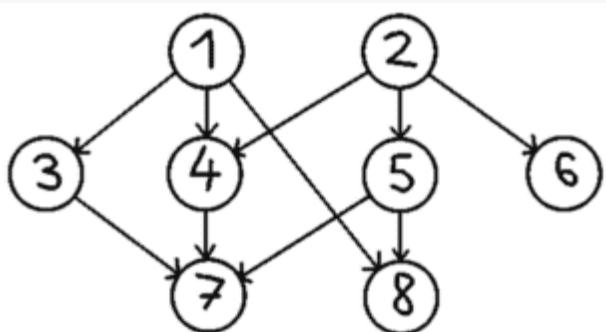
Ako su podzadaci *zavisni* tada se mora utvrditi redoslijed izvođenja njihovih dretvi.

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

Primjer 4.1. Sustav zadataka zadan u obliku usmjerenog grafa

Ako se zadatak može rastaviti na lanac podzadataka $Z_1 \rightarrow Z_2 \rightarrow \dots \rightarrow Z_N$, 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 podzadataka se možda može prikazati kao:



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

Za svaki zadatak može se napraviti usporedba ovisnosti sa svim ostalima

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 samo osigurati da takva korištenja budu pojedinačna – da se promjene ne obavljaju paralelno. Više o tome kasnije (u odjeljku o međusobnom isključivanju).

Zadatak: 4.1. (ispitni zadatak)

Sustav zadataka je zadan u obliku lanca (kada se zadaci izvode ovim redom rezultat će biti ispravan): $Z_1 \rightarrow Z_2 \rightarrow Z_3 \rightarrow Z_4 \rightarrow Z_5 \rightarrow Z_6 \rightarrow Z_7$ Zadaci imaju domene (D) i kodomene (K) prema tablici:

Tablica 4.1. Domene i kodomene za zadatke

| | Z_1 | Z_2 | Z_3 | Z_4 | Z_5 | Z_6 | Z_7 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| M_1 | D | D | D | | | K | |
| M_2 | K | | | K | | D | D |
| M_3 | | K | | D | K | | K |
| M_4 | | | K | | D | | |
| M_5 | | | | | | | |

[dodatno]

Da li može biti prazan redak (kao M5) – može! Da li može biti prazan stupac – može!

Da li može biti u istom stupcu više D i/ili više K – može!!

Da li može biti u nekom stupcu samo K ili samo D – pogađate: može!!

Odrediti maksimalno paralelni sustav zadataka, uzimajući u obzir njihov međusobni odnos u lancu i domene i kodomene.

4.4. Međusobno isključivanje

MI = najjednostavniji/osnovni mehanizam sinkronizacije

Pristup zajedničkim sredstvima (npr. varijablama) treba zaštiti.

Odsječke koda koji koriste takva zajednička sredstva nazivamo *kritičnim odsječcima*.

[dodatak]

Npr. ako više dretvi povećava neki brojač kodom `brojač=brojač+dodano`; rezultat može biti krivi zato jer su potrebne četiri instrukcije (dohvati `brojač`, dohvati `dodano`, zbroji, pohrani) koje mogu biti prekinute ili izvođene paralelno.

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 ćemo razmatrati cikličke dretve, koje imaju samo jedan kritičan odsječak prema:

```
ciklička_dretva
{
    ponavljam {
        kritični odsječak;
        nekritični odsječak;
    }
    do zauvijek;
}
```

Primjer 4.2. Poslužiteljska dretva

U nekom poslužitelju dretve koje poslužuju zahtjeve će raditi otprilike slijedeće:

```
dretva_poslužitelja //jedna od "radnih dretvi"
{
    ponavljam {
        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
{
    ponavljam {
        uđi_u_KO();
        kritični odsječak;
        izađi_iz_KO();

        nekritični odsječak;
    }
    do zauvijek;
}
```

Kako ostvariti te funkcije (`uđi_u_KO` i `izađi_iz_KO`)?

Zahtjevi na algoritme međusobnog isključivanja (ispitno pitanje)

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

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

4.5. Potraga za algoritmima međusobnog isključivanja

Zašto ga "tražimo" kad znamo koji valjaju? Da usput pokažemo probleme višedretvenih sustava!

Opće pretpostavke:

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

Pronađimo prvo algoritam koji radi za dvije dretve, jer ako ne radi za dvije neće ni za više!

Koristiti ćemo radno čekanje jer trenutno ne znamo za bolje rješenje.

4.5.1. 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 promjeni

```
uđi_u_KO ()
{
    dok je ( ZASTAVICA == 1 )
        ; //ništa, ponovno pročitaj ZASTAVICU;
    ZASTAVICA = 1;
}
```

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

U asembleru uđi_u_KO:

```
uđi_u_KO:
    ADR R0, ZASTAVICA;
petlja:
    LDR R1, [R0] //problem ako ovo ide paralelno, svi čitaju 0!
    CMP R1, 1
    BEQ petlja
    STR 1, [R0] //tek se ovdje ZASTAVICA mijenja
    RET
```

Problemi:

- ako idu paralelno, u uzastopnim sabirničkim ciklusima, obje čitaju 0 i obje ulaze u KO;
- nije ispunjen osnovni uvjet (1) (ostali jesu, ali moraju biti svi)

4.5.2. Drugi pokušaj (PRAVO)

Koristiti varijablu PRAVO koja može imati vrijednost 0 ili 1 što je indeks dretve koja iduća može ući u KO ($I = 1 - J$; $J = 1 - I$;

```
uđi_u_KO (I)
{
    dok je ( PRAVO != I )
        ; //ništa, ponovno pročitaj PRAVO;
}
```

```
izađi_iz_KO (I)
{
    PRAVO = 1 - I;
}
```

Problemi:

- ulazak u KO je strogo naizmjeničan
- uvjeti 2 i 3 nisu ispunjeni!

4.5.3. Treći pokušaj (`ZASTAVICA[]`)

Koristiti dvije varijable `ZASTAVICA[I]` i `ZASTAVICA[J]` koje mogu imati vrijednost 0 ili 1 što označava jesu li zadane dretve u KO ili nisu

```
udi_u_KO (I)
{
    J = 1 - I; //druga dretva

    dok je ( ZASTAVICA[J] != 0 )
        ;

    ZASTAVICA[I] = 1;
}
```

```
izađi_iz_KO (I)
{
    ZASTAVICA[I] = 0;
}
```

Problem:

- u paralelnom radu obje mogu pročitati 0 u suprotnim zastavicama i uči u KO

4.5.4. Četvrti pokušaj (`ZASTAVICA[]`)

Zastavicu postaviti prije provjere suprotne zastavice

```
udi_u_KO (I)
{
    J = 1 - I;

    ZASTAVICA[I] = 1;

    dok je ( ZASTAVICA[J] != 0 )
        ;
}
```

```
izađi_iz_KO (I)
{
    ZASTAVICA[I] = 0;
}
```

Problem:

- u paralelnom radu obje mogu prvo postaviti svoje zastavice i onda u idućoj petlji beskonačno radno čekati jer se zastavice neće spustiti – nikad neće uči u KO (4) – potpuni zastoj

4.5.5. Peti pokušaj (`ZASTAVICA[]`)

Privremeno spustiti zastavicu dok je ona druga podignuta

```
udi_u_KO (I)
{
    J = 1 - I;

    ZASTAVICA[I] = 1;
    dok je ( ZASTAVICA[J] != 0 ) {
        ZASTAVICA[I] = 0;
        dok je ( ZASTAVICA[J] != 0 )
            ;
        ZASTAVICA[I] = 1;
    }
}
```

```
izađi_iz_KO (I)
{
    ZASTAVICA[I] = 0;
}
```

Problem:

- u paralelnom radu obje dretve mogu sinkrono podizati i spuštati zastavice i nikad ne uči u KO (4), mada dovoljan je mali “poremećaj” pa da jedna “prođe”.

4.5.6. Dekkerov algoritam (ispravan, radi za dvije dretve)

- rješenje koje je ponudio nizozemski matematičar T. Dekker, a opisao ga je E.W.Dijkstra (1959)
- kombinacija drugog i petog pokušaja

```
uđi_u_KO (I)
{
    J = 1 - I;
    ZASTAVICA[I] = 1;

    dok je ( ZASTAVICA[J] != 0 ) {
        ako je ( PRAVO == J ) {
            ZASTAVICA[I] = 0;
            dok je ( PRAVO == J )
                ;
            ZASTAVICA[I] = 1;
        }
    }
    //neće izaći iz petlje dok se ZASTAVICA[J] ne spusti!
}
```

```
izađi_iz_KO (I)
{
    ZASTAVICA[I] = 0;
    PRAVO = 1 - I;
}
```

Pojednostavljenje: Petersonov algoritam

```
uđi_u_KO (I)
{
    J = 1 - I;
    ZASTAVICA[I] = 1;
    PRAVO = 1 - I;

    dok je ( ZASTAVICA[J] != 0 && PRAVO == J )
        ;
}
```

```
izađi_iz_KO (I)
{
    ZASTAVICA[I] = 0;
}
```

Prednosti Petersonova algoritma:

- kraći i brži (potrebno manje instrukcija)
- ne ovisi o početnoj vrijednosti varijable PRAVO

Dekkerov i Petersonov algoritam rade za sustave sa samo dvije dretve

Proširenje tog algoritma napravio je Lamport svojim “pekarskim” algoritmom

4.5.7. 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 ogradije
- zajednički podaci:
 - BROJ[N] – dodijeljeni broj (0 kada dretva ne traži ulaz u KO)
 - ULAZ[N] – štiti se dodjela broja
 - ZADNJI – zadnji dodijeljeni broj (najveći do sada)
 - početne vrijednosti varijabli = 0;

```

udi_u_KO (I)
{
    //uzimanje broja
    ULAZ[I] = 1;
    BROJ[I] = ZADNJI;
    ZADNJI++;
    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 proizvoljnem broju procesora!
- – potrebna poveća struktura podataka
- – radno čekanje (kao i Dekkerov i Petersonov)

4.6. Sklopovska potpora međusobnom isključivanju

Zabrana prekidanja – za jednoprocesorske sustave

U jednoprocesorskim sustavima bi mogli koristiti zabranu i dovolu prekidanja:

- `udi_u_KO() == onemoguci_prekidanje`
- `izaди_iz_KO() == omoguci_prekidanje`

Problemi:

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

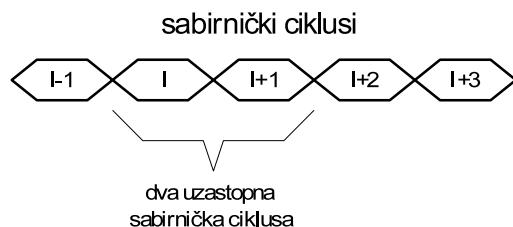
Drugi načini?

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

Ako se to sklopovali 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!

Sklopovske podrške u obliku instrukcija `TAS`, `SWP`, `INC`, `CAS`

4.6.1. Ostvarenje MI sa 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 drugom ciklusu na tu adresu spremi vrijednost 1

[dodatno]

Nekakav ekvivalent operacija te instrukcije bio bi:

```
TAS (adresa) = {
    //prvi dio instrukcije koristi 1. sabirnički ciklus
    LDR X, [adresa]
    CMP X, 0      //proširenje gornje instrukcije; samo pogleda je li 0

    //drugi dio instrukcije koristi 2. sabirnički ciklus (odmah iza 1.)
    STR 1, [adresa]
}
```

X je privremeni registar, a prva i treća naredba koriste dva uzastopna sabirnička ciklusa

Ali razlika je u tome što je to JEDNA instrukcija!

Rješenje MI sa TAS u asembleru:

| |
|---|
| <u>uđi_u_KO:</u> petlja: TAS zastavica BNE petlja //kada nije 0 čekaj RET |
|---|

| |
|--|
| <u>izađi_iz_KO:</u> ADR R1, zastavica STR 0, [R1] RET |
|--|

Neka u pseudokodu **TAS** adresa vraća pročitanu vrijednost (radi jednostavnosti i kraćeg zapisa):

| |
|--|
| <u>uđi_u_KO ()</u> { dok je (TAS (zastavica) == 1) ; // radno čekanje } |
|--|

| |
|--|
| <u>izađi_iz_KO ()</u> { zastavica = 0; } |
|--|

4.6.2. Ostvarenje MI sa *Zamijeni – SWP* ili *XCHG – sxchange* (info)

Neka postoji **instrukcija** `SWP R1, R2, [R3]` koja koristi dva uzastopna sabirnička ciklusa tako da:

- u prvom ciklusu pročita vrijednost sa adrese zadane s `R3` u registar `R2`
- u drugom ciklusu na tu adresu spremi vrijednost iz registra `R1`

[dodatakno]

Nekakav ekvivalent operacija te instrukcije bio bi:

`SWP R1, R2, [R3] = { LDR R2, [R2]; STR R1, [R3]; }`

Ali razlika je u tome što je to JEDNA instrukcija!

Rješenje MI sa SWP u asembleru:

```
uđi_u_KO:
    ADR R3, zastavica
    LDR R1, 1
petlja:
    SWP R1, R2, [R3]
    CMP R2, 1
    BEQ petlja
    RET
```

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

Neka u psedudokodu funkcija `Zamijeni (adresa, var)` radi zamjenu vrijednosti varijabli `adresa` i registra `reg` u dva uzastopna sabirnička ciklusa.

```
uđi_u_KO ()
{
    reg = 1;
    ponavljam {
        Zamijeni ( zastavica, reg );
    }
    dok je ( reg == 1 );
}
```

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

4.6.3. Ostvarenje MI sa *Usporedi_i_Zamijeni – CAS* (compare-and-swap) (info)

U novijim procesorima postoji bolja inačica prethodnih instrukcija koja ne koristi drugi sabirnički ciklus ako nije potreban. Primjerice, ako smo već u prvom dijelu izvođenja instrukcije `TAS` ustanovili da je zastavica postavljena, drugi sabirnički ciklus ništa ne mijenja – zapisuje broj 1 preko broja 1. Stoga ima smisla taj drugi dio operacije izostaviti.

Naredba `CAS R1, R2, [R3]` u prvom sabirničkom ciklusu dohvata vrijednost s adrese zadane u `R3` te uspoređuje tu vrijednost s onom u registru `R1`. Ako su vrijednosti jednake, tada koristi drugi sabirnički ciklus (uzastopni) i na istu adresu zapisuje vrijednost iz registra `R2`. Instrukcija pritom postavlja i odgovarajuće zastavice u statusnom registru (primjerice zastavicu Z).

4.6.4. Problemi prikazanih mehanizama međusobnog isključivanja

- radno čekanje – osnovni problem svih prikazanih, sa i bez sklopovske potpore
- nepoštivanje redoslijeda zahtjeva – “problem” algoritama ostvarenih sklopovskom potporom: 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 ne, kada će se dretva maknuti s procesora (da ne troši procesorsko vrijeme na neproduktivnu petlju).

Pitanja za vježbu 4

1. Što je to proces, a što dretva? Kako se stvara novi proces (na UNIX-u), a kako nova dretva?
2. Što je zajedničko dretvama istog procesa?
3. Zašto se koristi višedretvenost? Koje su prednosti (mogućnosti) sustava koji podržavaju višedretvenost?
4. Kada su dva zadatka međusobno zavisna, a kada nezavisna? Što sa zavisnim zadacima, kako izvoditi njihove dretve?
5. Odrediti maksimalno paralelan sustav zadataka uzimajući u obzir njihov međusobni odnos u lancu te domene i kodomene. . .
6. Što je to *kritični odsječak* i *međusobno isključivanje*?
7. Kako se međusobno isključivanje može ostvariti u jednoprocесorskim a kako u višeprocесorskim sustavima?
8. Koji problem može nastati ako više dretvi obavlja operaciju: $A = A + 1$ nad zajedničkom varijablom A ?
9. Navesti zahtjeve (4) na algoritme međusobnog isključivanja.
10. Čemu služi Dekkerov algoritam, a čemu Lamportov?
11. Kako iskoristiti sklopovsku potporu međusobnom isključivanju korištenjem instrukcija TAS, SWP i sličnih?
12. Koji su problemi različitih načina ostvarenja međusobnog isključivanja?
13. Za zadani algoritam međusobnog isključivanja provjeriti je li zadovoljava sve zahtjeve prema takvim algoritmima. . .

5. Jezgra operacijskog sustava

Potreba za jezgrom (za OS)?

Do sada je primijećeno:

- Prekidi
 - upravljanje UI
 - prekidni – sustavski – jezgrin način rada procesora
 - operacije koje se izvode u prekidu su posebne => dio jezgre OSa!
 - programski prekid: program poziva te “posebne funkcije”
- Sinkronizacija
 - programska i sklopovska imaju nedostatke: radno čekanje, nepoštivanje redoslijeda

Potreba skupa funkcija – jezgrinih funkcija – koje će se pozivati mehanizmom *prekida*

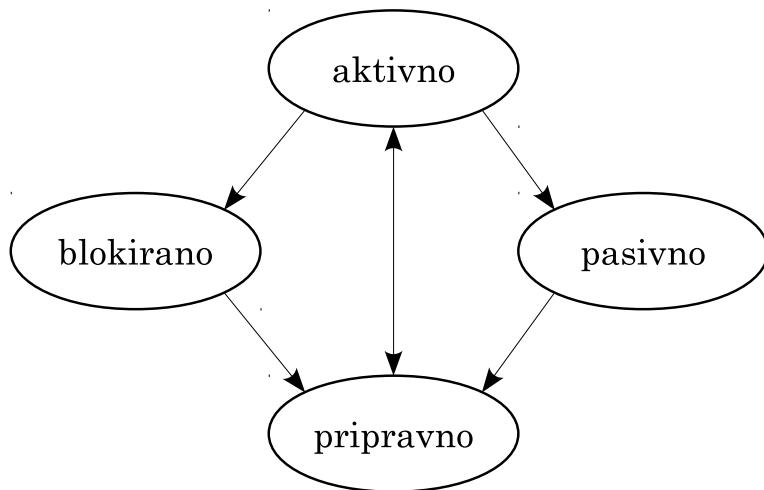
- privilegirani način rada
- korištenje zaštićene strukture podataka – za upravljanje sustavom
- za upravljanje UI napravama (i za UI naprave, međuspremniči, ...)
- upravljanje dretvama

Prepostavke idućih razmatranja za ostvarenje jezgre:

- jednoprocesorski sustav
- 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 se neće razmatrati procesi, neka su sve dretve sustava u jednom zajedničkom procesu.

Dretve mogu biti u različitim stanjima prema slici 5.3.



Slika 5.1. Moguća stanja dretve

Jezgra OS-a se sastoji od: strukture podataka jezgre i jezgrinih funkcija

5.1. Strukture podataka jezgre

Struktura podataka jezgre se sastoji od sljedećih elemenata:

- *opisnici UI naprava*
 - kazaljke na funkcije upravljačkog programa
 - međuspremniči
 - lista za dretve koje čekaju na napravu
- *opisnici dretvi:*
 - id
 - podaci za raspoređivanje: način raspoređivanja, prioritet, ...
 - stanje dretve
 - opis spremničkog prostora (gdje su instrukcije, podaci, stog)
 - zadano_kašnjenje – za ostvarivanje kašnjenja
 - kontekst – mjesto za pohranu konteksta dretve
- *lista stanja dretvi* – liste za opisnike dretvi
 - Aktivna_D
 - Pripravne_D
 - blokirane dretve:
 - * UI[i] – liste dretvi koje čekaju na napravu i
 - * Odgođene_D
 - * BSEM[i] – binarni semafori
 - * OSEM[j] – opći semafori
 - Postojeće_D – lista u kojoj se nalaze sve dretve
 - * ako se dretva nalazi samo u ovoj listi, dretva je *pasivna*

Liste: jednostruko povezane, dvostruko povezane

Liste dretvi mogu biti uređene (složene):

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

Za svaku listu potrebno je:

- kazaljka na prvu dretvu u listi (npr. `prva`)
- (opc.) kazaljka na zadnju dretvu u listi (npr. `zadnja`) – umetanje na kraj tada je složenosti $O(1)$!

Jezgra izvodi dretvu za dretvom:

- kada se trenutno aktivna blokira, uzima se prva iz reda pripravnih
- u obradama prekida se neke blokirane dretve mogu odblokirati i staviti među pripravne
- složenije načine odabira aktivne dretve ćemo obraditi naknadno (u 7. poglavljju)

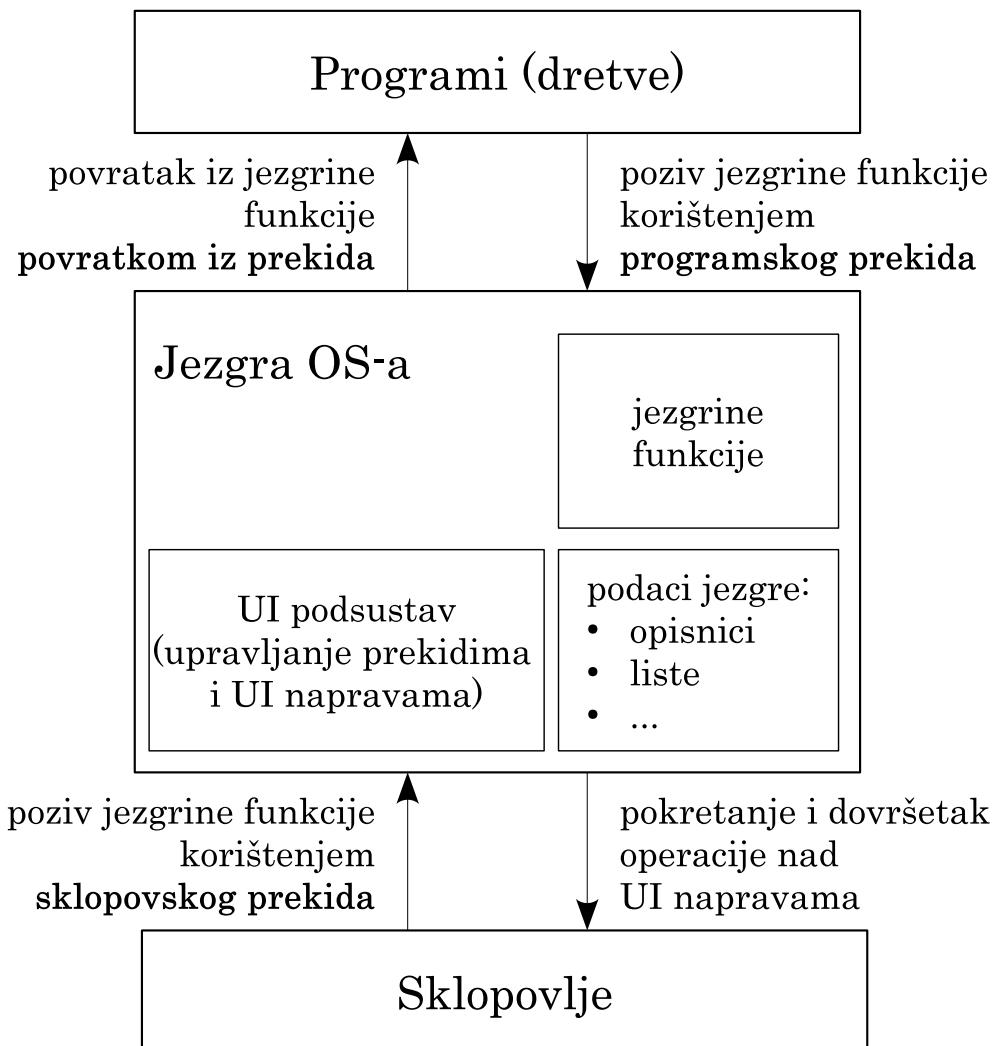
Latentna dretva (engl. *idle thread*)

- Što ako nema niti jedne pripravne dretve?
- Zbog takve mogućnosti u sustav se dodaje jedna neproduktivna dretva – latentna dretva koja će trošiti procesorsko vrijeme u takvim slučajevima.
- Takva dretva ima najmanji prioritet, tj. izvodi se jedino ako nema niti druge dretve!

5.2. Jezgrine funkcije

Jezgrine funkcije pozivaju se mehanizmom prekida i to:

- sklo povskim prekidom (zahtjev UI naprave ili satnog mehanizma)
- programskim prekidom (zahtjev iz programa)



Slika 5.2. Mehanizam poziva jezginih funkcija

5.2.1. Kako se pozivaju j-funkcije iz programa? (info)

U kodu programa:

```
...
J_Funkcija ( parametri ); //npr. printf ("Hello\n");
...
```

Sama funkcija (“omotač”) koja poziva j-funkciju:

```
J_Funkcija ( parametri )
{
    dodatni poslovi prije poziva jezgre;
    pohrani za poziv jezgre ( J_FUNKCIJA_ID, parametri );
    izazovi programski prekid;
    vradi = dohvati_povratnu_vrijednost_j_funkcije();
    dodatni poslovi nakon poziva jezgre;
}
```

U obradi prekida:

```
...
ako je ( uzrok prekida programski prekid, tj. poziv j-funkcije ) {
    {id, parametri} = dohvati ID tražene jezgrine funkcije te parametre;
    JF[id] ( parametri ); //J_FUNKCIJA(parametri);
}
inače ...
```

Opći izgled jezgrinih funkcija

```
j-funkcija J_FUNKCIJA (p)
{
    pohrani kontekst u opisnik Aktivna_D;
    operacija jezgrine funkcije; //može uzrokovati i promjenu aktivne dretve
    obnovi kontekst iz opisnika Aktivna_D;
}
```

U nastavku je sama j-funkcija pisana VELIKIM_SLOVIMA, a poziv iz programa Malim_Slovima:

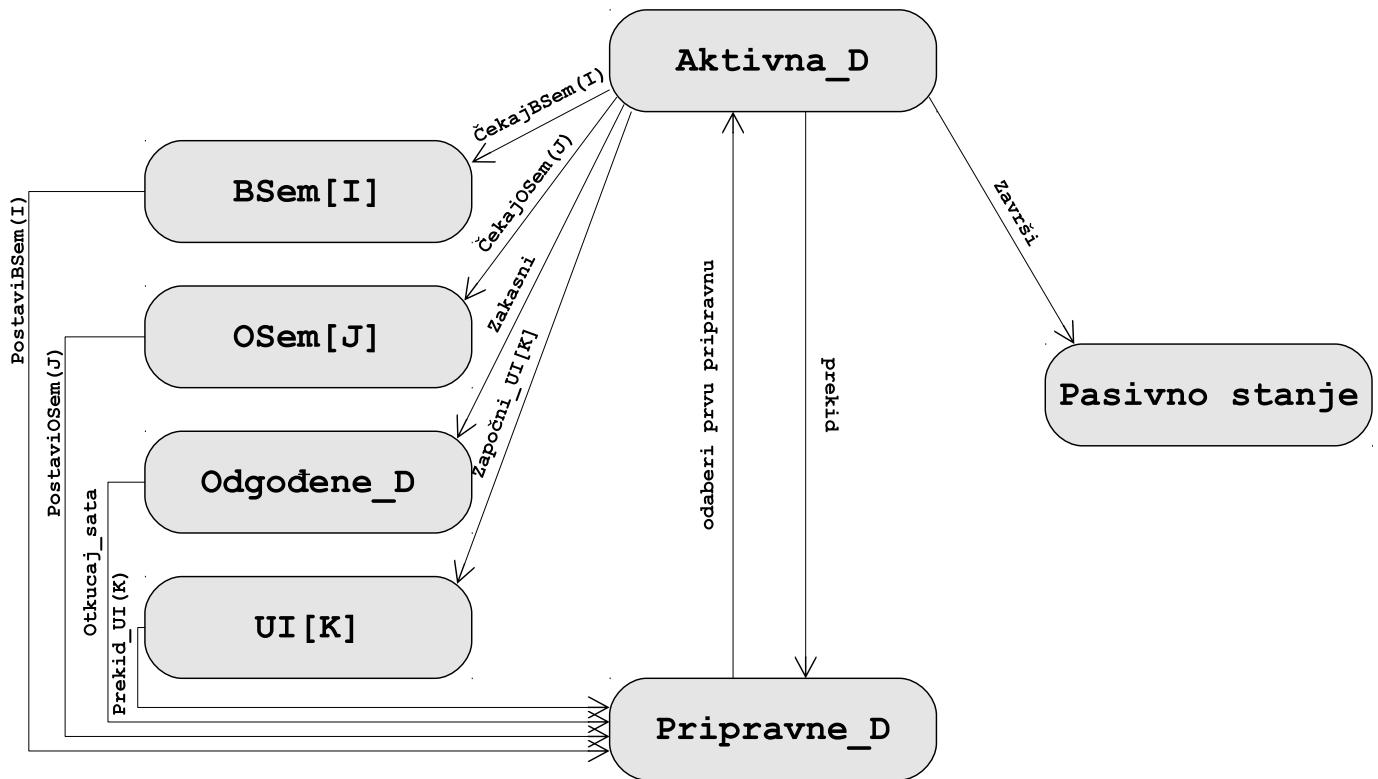
```
J_FUNKCIJA <==> J_Funkcija
```

Prije poziva na stog je pohranjen kontekst prekinute dretve.

Izlazak pretpostavlja vraćanje u dretvu čiji je opisnik u redu Aktivna_D

Pretpostavlja se da obnovi kontekst iz opisnika Aktivna_D, osim obnove svih registara procesora izvodi i instrukciju: vratiti_se_u_prekinutu_dretvu (obnovi PC i SR sa stoga, prebací se u način rada prekinute dretve, dozvoli prekidanje)

5.3. Moguća stanja dretvi u jednostavnom modelu sustava



Slika 5.3. Moguća stanja dretve, jezgrine funkcije

Pasivne_D 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).

Sve j-funkcije osim Otkucaj_sata i Prekid_UI (koje se pozivaju sklopovskim prekidima) poziva Aktivna_D !

Tablica 5.1. Primjer stanja sustava u nekom trenutku

| Red (liste opisnika) | Popis dretvi (id) |
|----------------------|-------------------|
| Aktivna_D | 7 |
| Pripravne_D | 8 5 0 |
| BSEM[1] | 4 |
| OSEM[1] | 1 |
| UI[1] | 6 |
| UI[2] | 3 |
| Odgodene_D | 2 |

5.4. Obavljanje UI operacija

UI operacije obavljaju se preko jezgrinih funkcija:

- `ZAPOČNI_UI (K, parametri)` – poziva ju dretva (ili jezgra)
 - UI operacija traje (nije trenutna) pa će dretva čekati na njen kraj
- `PREKID_UI (K)` – poziva se na prekid naprave K

```
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;
    stavi_u_red ( makni_prvu_iz_reda ( Pripravne_D ), Aktivna_D );

    obnovi kontekst iz opisnika Aktivna_D;
}
```

U knjizi su iste operacije ostvarenje/napisane malo drugačije. Međutim, funkcionalnost je identična.

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

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

    obnovi kontekst iz opisnika Aktivna_D;
}
```

Pomoćne funkcije:

- `stavi_u_red (opisnik_dretve, opisnik_reda)` – stavlja zadani opisnik u zadani red poštivajući uređenje reda:
 - ako je red složen po redu prispijeća, novi opisnik ide na kraj reda
 - ako je red složen po prioritetu, novi opisnik ide na odgovarajuće mjesto (prva dretva u redu ima najveći prioritet, zadnja najmanji)
- `makni_prvu_iz_reda (opisnik_reda)` – miče prvi opisnik iz reda i vraća ga (kazaljku/referencu na njega)

5.5. Ostvarivanje kašnjenja

Koristi se prekid sata koji u pravilnim intervalima otkucava (kvant vremena), tj. 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 `Zadano_kašnjenje` element opisnika dretve

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

    stavi_u_red_odgođenih ( makni_prvu_iz_reda ( Aktivna_D ), M, Odgođene_D );
    stavi_u_red ( makni_prvu_iz_reda ( Pripravne_D ), Aktivna_D );

    obnovi kontekst iz opisnika Aktivna_D;
}
```

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

    /* neki rasporedišivači pozivaju se iz ove funkcije (info) */
    //rasporedišvanje_dretvi_podjelom_vremena ();

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

    obnovi kontekst iz opisnika Aktivna_D;
}

//primjer rasporedišvanja podjelom vremena (info)
rasporedišvanje_dretvi_podjelom_vremena ()
{
    ako je ( Aktivna_D se raspoređuje podjelom vremena ) {
        Aktivna_D->Otkucaji--;
        ako je ( Aktivna_D->Otkucaji == 0 ) {
            Aktivna_D->Otkucaji = Aktivna_D->Otkucaji_max;
            //stavi dretvu na kraj reda
            stavi_u_red ( makni_prvu_iz_reda ( Aktivna_D ), Pripravne_D );
            stavi_u_red ( makni_prvu_iz_reda ( Pripravne_D ), Aktivna_D );
        }
    }
}
```

5.6. Semafori

Semafori služe za sinkronizaciju dretvi.

Postoji nekoliko inačica semafora, koji se koriste za razne potrebe. U nastavku su prikazani oni najvažniji.

5.7. Binarni semafor

- binarni semafor ima dva stanja: prolazno i neprolazno
- zato je prikladan za međusobno isključivanje, ali i za druge potrebe

Struktura podataka potrebna za semafor:

- `.v` – vrijednost semafora
 - kada je `.v == 0` semafor je *neprolazan*
 - kada je `.v == 1` semafor je *prolazan*
- `.red` – blokirane dretve nad semaforom (lista opisnika tih dretvi)

Jezgrine funkcije:

- `ČEKAJ_BSEM (I) (ili ISPITATI_BSEM)`
- `POSTAVI_BSEM(I) (ili POSTAVITI_BSEM)`

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

    ako ( BSEM[S].v == 1 ) {
        BSEM[S].v = 0;
    }
    inače {
        stavi_u_red ( makni_prvu_iz_reda ( Aktivna_D ), BSEM[S] );
        stavi_u_red ( makni_prvu_iz_reda ( Pripravne_D ), Aktivna_D );
    }

    obnovi kontekst iz opisnika Aktivna_D;
}
```

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

    ako ( red BSEM[S] nije prazan ) {
        stavi_u_red ( makni_prvu_iz_reda ( Aktivna_D ), Pripravne_D );
        stavi_u_red ( makni_prvu_iz_reda ( BSEM[S] ), Pripravne_D );
        stavi_u_red ( makni_prvu_iz_reda ( Pripravne_D ), Aktivna_D );
    }
    inače {
        BSEM[S].v = 1;
    }

    obnovi kontekst iz opisnika Aktivna_D;
}
```

Pri oslobođanju dretve može se dogoditi da oslobođena dretva ima veći prioritet od dretve koja ju je oslobođila (trenutno aktivna dretva): zato se obje pomiču u red pripravnih – ako su liste uređene

prema prioritetu, na prvo će mjesto doći najprioritetnija.

Npr. ako je prva u redu `BSEM[1]` dretva 4 prioriteta 4, a pozivajuća dretva ima prioritet 3 onda po pozivu j.f. aktivna dretva je dretva 4.

Primjer 5.1. Primjer ostvarenja kritičnog odsječka binarnim semaforom

```
dretva () {
    ponavljam {
        Čekaj_BSEM(I);
        kritični odsječak;
        Postavi_BSEM(I);
        nekriticni odsječak;
    }
    do zauvijek
}
```

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

Primjer 5.2.

Primjer: sinkronizirati ulaznu, radnu i izlaznu dretvu s četiri binarna semafora.

Ulagna dretva:

```
while(1)
{
    dohvati podatke;

    pošalji radnoj d.;

}
```

Radna dretva:

```
while(1)
{
    primi podatke;

    obradi podatke;

    posalji izlaznoj d.;

}
```

Izlazna dretva:

```
while(1)
{
    primi_rezultate;

    pohrani rezultate;
}
```

5.8. Opći semafor

Opći semafor ima više stanja: jedno neprolazno i više prolaznih

Zato je prikladan za brojanje događaja i sredstava i 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 onima koja se koriste u stvarnim sustavima. Označimo takav semafor sa OSEM.

Opći semafor – OSEM

- druga imena: brojački semafor, brojilo događaja
- po ostverenju (jezgrinim funkcijama) jako slično BSEM-u, uz `++ umjesto =1` i `-- umjesto =0`

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

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

    obnovi kontekst iz opisnika Aktivna_D;
}

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

    ako ( red OSEM[S] nije prazan ) {
        stavi_u_red ( makni_prvu_iz_reda ( Aktivna_D ), Pripravne_D );
        stavi_u_red ( makni_prvu_iz_reda ( OSEM[S] ), Pripravne_D );
        stavi_u_red ( makni_prvu_iz_reda ( Pripravne_D ), Aktivna_D );
    }
    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) nije izravno dohvatljiva programima – oni je ne mogu promijeniti osim preko jezgrinih funkcija `Čekaj_OSEM/Postavi_OSEM !!!`

Primjer 5.3. Dijkstrin semafor (info)

Osim općeg semafora OSEM postoje i druge zamisli za ostvarenje semafora.

Dijkstrin semafor (označimo ga s OS) jest “jednokratni” semafor pridijeljen dretvi, na koji samo jedna dretva može čekati. Svojstvo ovog semafora jest da može poprimiti i negativne vrijednosti. Ostvarenje takvog semafor prikazano je u nastavku.

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

    OS[S].v--;
    ako ( OS[S].v < 0 ) {
        stavi_u_red ( makni_prvu_iz_reda ( Aktivna_D ), OS[S] );
        stavi_u_red ( makni_prvu_iz_reda ( Pripravne_D ), Aktivna_D );
    }

    obnovi kontekst iz opisnika Aktivna_D;
}

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

    OS[S].v++;
    ako (OS[S].v >= 0 && red OS[S] nije prazan ) {
        stavi_u_red ( makni_prvu_iz_reda ( Aktivna_D ), Pripravne_D );
        stavi_u_red ( makni_prvu_iz_reda ( OS[S] ), Pripravne_D );
        stavi_u_red ( makni_prvu_iz_reda ( Pripravne_D ), Aktivna_D );
    }

    obnovi kontekst iz opisnika Aktivna_D;
}
```

Binarni semafor i opći semafor BITNO se razlikuju po načinu rada. Pozivom `Postavi_OSEM(I)`, ukoliko nema blokiranih dretvi u redu tog semafora, vrijednost semafora se povećava za 1. Ako je trenutna vrijednost binarnog semafora već jednaka 1, pozivom `Postavi_BSEM(I)` se ništa neće promijeniti – vrijednost će i nakon poziva biti jednaka 1.

Primjer 5.4. Primjer ostvarenja kritičnog odsječka semaforom

```
dretva () {
    ponavljam {
        Čekaj_OSEM(I);
        kritični odsječak;
        Postavi_OSEM(I);

        nekritični odsječak;
    }
    do zauvijek
}
```

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

Primjer 5.5.

Primjer: sinkronizirati ulaznu, radnu i izlaznu dretvu s četiri semafora.

Ulagna dretva:

```
while(1)
{
    dohvati podatke;

    pošalji radnoj d.;

}
```

Radna dretva:

```
while(1)
{
    primi podatke;

    obradi podatke;

    posalji izlaznoj d.;

}
```

Izlazna dretva:

```
while(1)
{
    primi_rezultate;

    pohrani rezultate;

}
```

Primjer 5.6. Web poslužitelj s glavnom dretvom i više radnih dretvi (i)

```

glavna_dretva
{
    ponavljam {
        C = čekaj_spoj_klijenta();
        Čekaj_OSEM(k);
        stavi_zahajev_u_red (C);
        Postavi_OSEM(k);
        Postavi_OSEM(i);
    }
    do kraja_rada;
}

radna_dretva
{
    ponavljam {
        Čekaj_OSEM(i);
        Čekaj_BSEM(k);
        uzmi_idući_zahajev_iz_reda;
        Postavi_BSEM(k);
        obradi_zahajev_i_vrati_rezultat;
    }
    do kraja_rada;
}

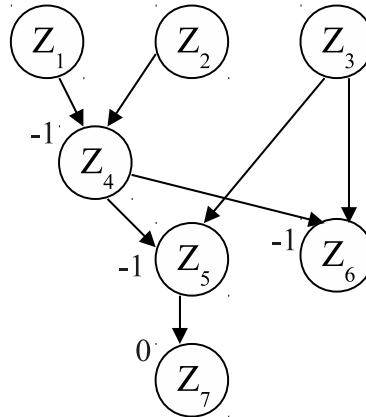
```

Početne vrijednosti: OSEM[k].v = 1, OSEM[i].v = 0

Zadatak: 5.1. (ispitni zadatak)

Za ostvarenje sustava dretvi prema slici (rješenje zadatka 4.1) koriste se binarni/opći semafori.

- Koliko semafora je potrebno za sinkronizaciju?
- Koje su početne vrijednosti semafora?
- Neka je T_i tekst zadatka i . Proširiti svaki zadatak T_i u T'_i s minimalnim brojem potrebnih procedura `čekaj_OSEM(j)` i `Postavi_OSEM(j)`.



5.9. Izvedba jezgrinih funkcija za višeprocesorske sustave

- Zabrana prekida nije dovoljna za višeprocesorske sustave kao mehanizam međusobnog isključivanja jezgrinih funkcija
- 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)

5.9.1. 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 ( OSEM[S].v == 1 ) {
        OSEM[S].v = 0;
    }
    inače {
        stavi_u_red ( makni_prvu_iz_reda ( Aktivna_D[M] ), OSEM[S] );
        stavi_u_red ( makni_prvu_iz_reda ( Pripravne_D[M] ), Aktivna_D[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 ( red OSEM[S] nije prazan ) {
        stavi_u_red ( makni_prvu_iz_reda ( Aktivna_D[M] ), Pripravne_D[M] );
        stavi_u_red ( makni_prvu_iz_reda ( OSEM[I] ), Pripravne_D[L] );
        //L - iz opisnika dretve
        stavi_u_red ( makni_prvu_iz_reda ( Pripravne_D[M] ), Aktivna_D[M] );
        //signaliziraj prekid procesoru L da napravi raspoređivanje
    }
    inače {
        OSEM[S].v = 1;
    }

    OGRADA_JEZGRE = 0;

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

[dodatak]

Kada nekom drugom procesoru stavljamo dretvu u red pripravnih on bi trebao pogledati je li sad to dretva najvećeg prioriteta kod njega (je li prioritet te dretve veći od aktivne)

Koristi se *radno čekanje*:

- ali samo do ulaska u jezgrinu funkciju (onaj bitan dio)!
- obzirom da su one kratke to nije dugo čekanje!

5.10. Primjeri sinkronizacije binarnim semaforima

Zadatak: 5.2. Problem pušača cigareta

(Ograda – ne reklamiramo cigarete (naprotiv), ali to je ime algoritma – Patil, 1971)

Zamislimo sustav s tri dretve pušača i jednom dretvom trgovcem. Svaki pušač neprestano savija cigarete i puši. Kako bi se savila i popušila cigareta potrebno je imati tri sastojka: duhan, papir i šibice. Jedan pušač ima u neograničenim količinama samo papir, drugi samo duhan, a treći samo šibice. Trgovac ima sva tri sastojka u neograničenim količinama. Trgovac nasumice stavlja na stol dva različita sastojka. Pušač koji ima treći sastojak uzima sastojke sa stola, signalizira trgovcu, savija cigaretu i puši. Trgovac stavlja nova dva sastojka na stol i ciklus se ponavlja. Na početku stol je prazan. Napisati dretve pušača i trgovca tako da se one međusobno ispravno sinkroniziraju s pomoću dva binarna semafora. Napisati početne vrijednosti semafora.

Zadatak: 5.3. Prioritetni redovi i semafori (kraći)

U jednoprocесорском računalu pokrenut je sustav dretvi D_1 , D_2 i D_3 s prioritetima 1, 2 i 3 respektivno. Najviši prioritet je 3. Svi zadaci koje obavljaju dretve su istog oblika D_x . Red pripravnih dretvi i red semafora su prioritetni. Aktivna je dretva koja je prva u redu pripravnih (nema posebnog reda aktivnih dretvi). Prije pokretanja sustava dretvi semafor s je bio zatvoren. Nakon nekog vremena sve dretve se nađu u redu semafora s . Ako se tada pozove procedura `Postavi_BSEM(s)`, što će se ispisati na zaslonu?

```
Dretva Dx{
    dok je(1) {
        Čekaј_BSEM(S);
        piši(Px);      a
        Postavi_BSEM(S);-----
        piši(Zx);      b
    }
}
```

5.11. Operacije stvaranja dretvi i semafora (info)

Stvaranje novih dretvi kao i drugih jezgrinih objekata (npr. semafora) traži podsustav za upravljanje spremnikom. Stoga te operacije nisu uključene u prikaz jezgrinih funkcija. Ipak, u dosta općenitom obliku u ovom su potpoglavlju navedene skice i takvih operacija.

```
j-funkcija STVORI_DRETVU ( početna_funkcija, parametar )
{
    pohrani kontekst u opisnik Aktivna_D;

    stog = rezerviraj dio spremnika za stog;
    opisnik = stvori novi opisnik dretve ( početna_funkcija, parametar, stog );
    stavi_u_red2 ( opisnik, Postojeće_D );

    stavi_u_red ( makni_prvu_iz_reda ( Aktivna_D ), Pripravne_D );
    stavi_u_red ( opisnik, Pripravne_D );
    stavi_u_red ( makni_prvu_iz_reda ( Pripravne_D ), Aktivna_D );

    obnovi kontekst iz opisnika Aktivna_D;
}

j-funkcija ZAVRŠI_DRETVU ()
{
    osloboodi stog kojeg je Aktivna_D koristila;

    makni_iz_reda ( Aktivna_D, Postojeće_D );
    osloboodi opisnik Aktivna_D;
    stavi_u_red ( makni_prvu_iz_reda ( Pripravne_D ), Aktivna_D );

    obnovi kontekst iz opisnika Aktivna_D;
}

j-funkcija STVORI_BSEM ( početna_vrijednost )
{
    pohrani kontekst u opisnik Aktivna_D;

    opisnik = stvori novi opisnik za binarni semafor;
    I = dodaj "opisnik" u polje binarnih semafora;
    BSEM[I].v = početna_vrijednost;
    postavi "I" kao povratnu vrijednost ove jezgrine funkcije;

    obnovi kontekst iz opisnika Aktivna_D;
}

j-funkcija UNIŠTI_BSEM ( I )
{
    pohrani kontekst u opisnik Aktivna_D;

    opisnik = BSEM[I];
    makni opisnik "I" iz polja binarnih semafora;
    osloboodi "opisnik";

    obnovi kontekst iz opisnika Aktivna_D;
}
```

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. Navesti primjer korištenja jezgrinih funkcija za ostvarenje međusobnog isključivanja (kritičnog odsječka).
8. Što je to semafor? Koja struktura podataka jezgre je potrebna za njegovo ostvarenje? Skicirati funkciju `Čekaj_OSEM(I)/Postavi_OSEM(I)`.
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 i nakon:
 - a) što dretva D1 pozove `Postavi_OSEM(1)`
 - b) što se obradi prekid sata.
11. Sinkronizirati sustav zadataka prikazan grafom (...) binarnim/općim semaforima. Naznačiti početne vrijednosti svih semafora.
12. Proizvodnja nekog elementa odvija se na traci na kojoj postoje tri robota koji svaki rade svoj dio posla, naprije R1, potom R2 te na kraju R3 (nakon tog proizvod se miče na drugu traku i odlazi iz sustava). Pomak trake obavlja se tek kad su sva tri robota završila sa svojim poslom nad zasebnim elementima. Rad pojedinih robota upravljan dretvama `r1()`, `r2()` i `r3()` (`r1` upravlja s R1, `r2` s R2 te `r3` s R3). Ako sa `posao_r1()` označimo sam rad na objektom koji radi R1 (za R2 i R3 slično) te sa `pomak()` aktivaciju trake i pomak za jedno mjesto, napisati pseudokod za dretve `r1`, `r2` i `r3`. Za sinkronizaciju koristiti binarne/opće semafore te po potrebi i dodatne varijable. Neka je i u početnom stanju ispred svakog robota dio nad kojim on treba obaviti svoje operacije.
13. Za sustav dretvi različita prioriteta koje izvode zadani kod (...), prikazati rad do idućih 15 ispisa.

6. Međudretvena komunikacija i koncepcija monitora

U ovom se poglavlju prikazuju mnogi primjeri sinkronizacije dretvi. Kako ustanoviti je li ta sinkronizacija ispravno napravljena?

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 beskonačne petlje.
- Nema mogućnosti potpunog zastoja.
- Nema radnog čekanja.
- Početne vrijednosti svih varijabli i semafora su navedene!

Problemi sinkronizacije semaforima će biti prikazani na primjeru problema *proizvođača i potrošača*

6.1. Problem proizvođača i potrošača

Zadatak: Ostvariti komunikaciju između jednog proizvođača i jednog potrošača

Načelni pseudokod rješenja:

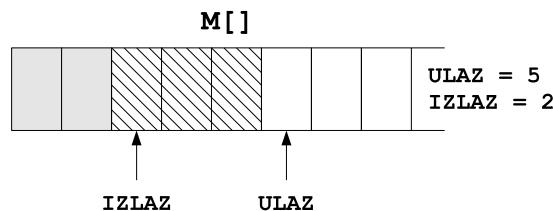
```
proizvođač
{
    ponavljam {
        P = stvori poruku ();
        pošalji poruku (P);
    }
    do zauvijek
}
```

```
potrošač
{
    ponavljam {
        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.

6.1.1. Korištenje neograničenog međuspremnika i radno čekanje



Slika 6.1. Neograničeni međuspremnik

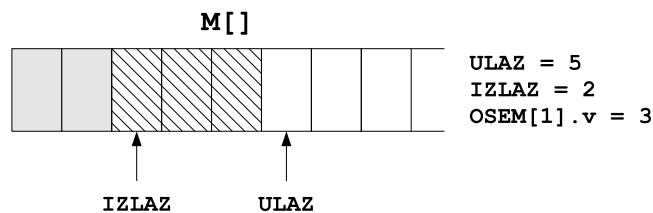
- **M[]** – međuspremnik neograničenje veličine
- **ULAZ** – indeks prvog praznog mesta u međuspremniku, početno 0
- **IZLAZ** – indeks prvog nepročitanog mesta u međuspremniku, početno 0

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

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

Problemi: radno čekanje, neograničeni međuspremnik

6.1.2. Korištenje neograničenog međuspremnika i jednog semafora



Slika 6.2. Neograničeni međuspremnik sa semaforima

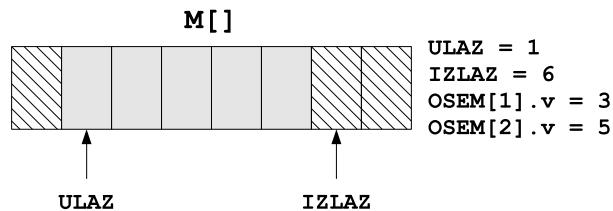
- OSEM[1] – broji poruke u međuspremniku, početno 0

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

```
potrošač
{
    ponavlja {
        Čekaј_OSEM(1);
        R = M[IZLAZ];
        IZLAZ++;
        obradi poruku (R);
    }
    do zauvijek
}
```

Problem: neograničeni međuspremnik

6.1.3. Korištenje ograničenog međuspremnika i dva semafora



Slika 6.3. Ograničeni međuspremnik

- $M[N]$ – međuspremnik sa N mesta 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
{
    ponavljaј {
        P = stvori poruku ();
        Čekaj_OSEM(2);
        M[ULAZ] = P;
        ULAZ = (ULAZ + 1) MOD N;
        Postavi_OSEM(1);
    }
    do zauvijek
}
```

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

Što ako ima više proizvođača (ili potrošača)?

6.1.4. Više proizvođača i jedan potrošač

Potrebno je dodati binarni semafor da proizvođači ne bi istovremeno mijenjali M i $ULAZ$.

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

6.1.5. Više proizvođača i više potrošača

```
proizvođač
{
    ponavlja {
        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
{
    ponavlja {
        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
}
```

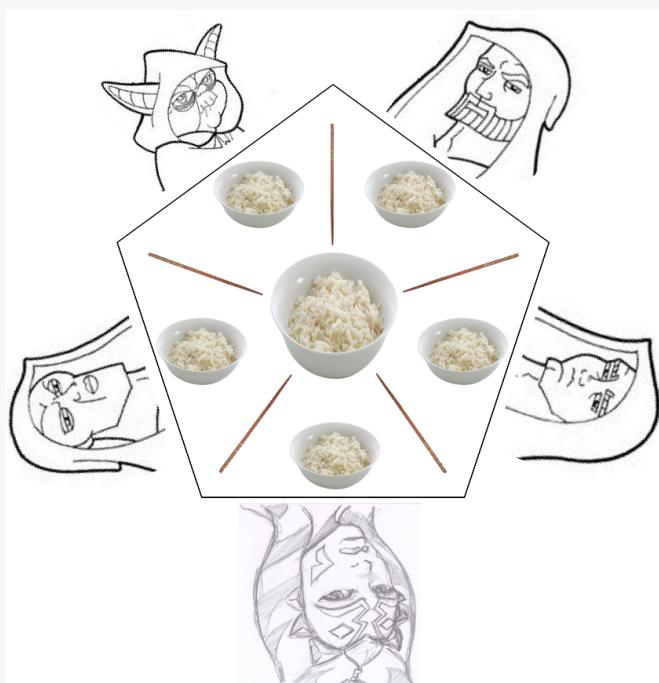
6.2. Problemi sa semaforima

Primjer 6.1. Problem pet filozofa

Pet filozofa sjedi za jednim stolom. Svaki od njih cijelo vrijeme ponavlja sljedeća tri koraka:

1. razmišlja neko vrijeme;
2. uzima lijevi i desni štapić (lijevo i desno od njegova tanjura s rižom) ako su slobodni (inače čeka da se oslobole)
3. jede

Obzirom da nema dovoljno štapića, neki će filozofi morati čekati na druge.
(Prije svake uporave štapići se operu (npr. uz pomoć "sile" :)).)



Slika 6.4. Problem pet filozofa

Pokušaj rješenja s pet binarnih semafora = po jedan za svaki štapić:

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

Problem: ako svi istovremeno uzmu svoj lijevi štapić – desni nije slobodan i svi stanu.

Sličan problem bi imali za problem pušača kad bi svako sredstvo štitili binarnim semaforom.

Primjer 6.2. Dvije dretve i dva semafora

```

dretva prva ()
{
    ponavljam {
        ...
        Čekaj_BSEM(I);
        ...
        Čekaj_BSEM(J); //stane
        ...
        Postavi_BSEM(J);
        ...
        Postavi_BSEM(I);
        ...
        ...
    }
    do ZAUVIJEK;
}

dretva druga ()
{
    ponavljam {
        ...
        ...
        Čekaj_BSEM(J);
        ...
        Čekaj_BSEM(I); //stane
        ...
        Postavi_BSEM(I);
        ...
        Postavi_BSEM(J);
        ...
    }
    do ZAUVIJEK;
}

```

Oba primjera prikazuju *potpuni zastoj*

6.2.1. Potpuni zastoj

Nužni uvjeti za mogućnost nastanka potpunog zastoja:

1. bar dvije dretve i bar dva sredstva koje te dretve obje koriste
2. sredstvo smije koristiti samo jedna dretva (međusobno isključivo)
3. dretvi se sredstvo ne može oduzeti, ona ga sama otpušta kada joj više ne treba
4. dretva drži dodijeljeno sredstvo dok traži dodatno sredstvo

Može li se neki uvjet maknuti?

Prva tri nema smisla – mijenja se logika programa.

Četvrти uvjet se može promijeniti drukčijim programiranjem:

- neka dretva zauzima potrebna sredstva odjednom, ne pojedinačno

Dosadašnji modeli semafora nam to ne dozvoljavaju!

[dodatačno]

Za razliku od našeg jednostavnog modela jezgre, današnji operacijski sustavi imaju i takva sučelja za rad atomarnih operacija nad skupom semafora, primjerice:

- semop na *UNIX*-u
- WaitForMultipleObjects na Win*

Međutim, ponekad (često) ni sa takvim sučeljem nije moguće riješiti problem.

6.3. Monitori

Problemi sa semaforima (npr. potpuni zastoj) nastaju u sustavima s više dretvi i više sredstava ili složenijim problemima sinkronizacije gdje semafori nisu dovoljni.

C.A.R. Hoare je 70'tih predložio drukčiji mehanizam sinkronizacije:

- sve kritične radnje koje uključuju zajednička sredstva obavljati u kontroliranom okruženju – monitorskim funkcijama – u monitoru
- monitorske funkcije su vrlo slične kritičnim odsjećcima (zapravo to i jesu) i jezgrnim funkcijama, ali NISU jezgrine funkcije već korisničke
- monitor ima proširenu funkcionalnost (dodatne operacije osim ‘uđi’ i ‘izadji’)
- u monitoru se mijenjaju i ispituju variable koje opisuju stanje sustava
- ako stanje sustava nije povoljno za dretvu da ona nastavi s radom (npr. nema lijevog i desnog štapića) dretva se blokira i privremeno napušta monitor – jednom jezgrinom funkcijom!
- blokirana dretva odblokira druga jezgrnim funkcijama (Oslobodi_iz_reda... objašnjena naknadno)
- u monitorskim funkcijama je uvijek najviše jedna dretva aktivna (blokirane ne brojimo)

Npr. problem pet filozofa bi mogli IDEJNO riješiti monitorom:

```
dretva Filozof (I)
{
    ponavljam {
        uzmi_štapiće (I); // monitorska funkcija
        jedi;
        vrati_štapiće (I); // monitorska funkcija
        misli;
    }
    do zauvijek;
}

m-funkcija uzmi_štapiće (I)
{
    dok (oba štapića oko filozofa I nisu slobodna)
        blokiraj dretvu;

    uzmi oba štapića oko filozofa I;
}

m-funkcija vrati_štapiće (I)
{
    vrati oba štapića oko filozofa I;
    propusti desnog filozofa ako čeka;
    propusti lijevog filozofa ako čeka;
}
```

6.3.1. Podrška jezgre (jezgrine funkcije) za ostvarenje monitora

U nastavku su navedene jezgrine funkcije za ostvarenje monitora (u prva dva izdanja knjige je to “svremenih monitor” – ali mi samo njega koristimo!).

(Ne traži se reprodukcija koda, već “samo” razumijevanje što koja funkcija radi.)

```
j-funkcija Uđi_u_monitor (M)    //mutex_lock
{
    pohrani kontekst u opisnik Aktivna_D;

    ako ( Monitor[M].v == 1 ) {
        Monitor[M].v = 0;
    }
    inače {
        stavi_u_red ( makni_prvu_iz_reda ( Aktivna_D ), Monitor[M] );
        stavi_u_red ( makni_prvu_iz_reda ( Pripravne_D ), Aktivna_D );
    }

    obnovi kontekst iz opisnika Aktivna_D;
}
```

```
j-funkcija Izadi_iz_monitora (M)    //mutex_unlock
{
    pohrani kontekst u opisnik Aktivna_D;

    ako ( red Monitor[M] nije prazan )
    {
        stavi_u_red ( makni_prvu_iz_reda ( Aktivna_D ), Pripravne_D );
        stavi_u_red ( makni_prvu_iz_reda ( Monitor[M] ), Pripravne_D );
        stavi_u_red ( makni_prvu_iz_reda ( Pripravne_D ), Aktivna_D );
    }
    inače {
        Monitor[M].v = 1;
    }

    obnovi kontekst iz opisnika Aktivna_D;
}
```

[dodatno]

Funkcije `Uđi_u_monitor` i `Izadi_iz_monitora` su identične funkcijama za rad s binarnim semaforima (čekaj_BSEM i Postavi_BSEM).

Međutim, nad monitorom se obavljaju i druge operacije; uz njega su povezane i druge strukture podataka.

Također, ulazak u monitor vezuje dretvu s monitorom (ovdje to nismo radili radi jednostavnosti, ali u “stvarnim sustavima” se to provjerava).

```
funkcija Čekaj_u_redu_uvjeta (M, R)      //cond_wait
{
    Uvrsti_u_red_uvjeta (M, R);
    Uđi_u_monitor (M);
}
```

```
j-funkcija Uvrsti_u_red_uvjeta (R, M) //nema ekv. u sustavima
{
    pohrani kontekst u opisnik Aktivna_D;

    stavi_u_red ( makni_pruv_iz_reda ( Aktivna_D ), Red_uvjeta[R] );
    ako ( red Monitor[M] nije prazan ) {
        stavi_u_red ( makni_pruv_iz_reda ( Monitor[M] ), Pripravne_D );
    }
    inače {
        Monitor[M].v = 1;
    }
    stavi_u_red ( makni_pruv_iz_reda ( Pripravne_D ), Aktivna_D );

    obnovi kontekst iz opisnika Aktivna_D;
}
```

```
j-funkcija Oslobodi_iz_reda_uvjeta (R)      //cond_signal
{
    pohrani kontekst u opisnik Aktivna_D;

    ako ( red Red_uvjeta[R] nije prazan ) {
        stavi_u_red ( makni_pruv_iz_reda ( Aktivna_D ), Pripravne_D );
        stavi_u_red ( makni_pruv_iz_reda ( Red_uvjeta[R] ), Pripravne_D );
        stavi_u_red ( makni_pruv_iz_reda ( Pripravne_D ), Aktivna_D );
    }

    obnovi kontekst iz opisnika Aktivna_D;
}
```

```
j-funkcija Oslobodi_sve_iz_reda_uvjeta (R) //cond_broadcast
{
    pohrani kontekst u opisnik Aktivna_D;

    ako ( red Red_uvjeta[R] nije prazan ) {
        stavi_u_red ( makni_pruv_iz_reda ( Aktivna_D ), Pripravne_D );
        dok je ( red Red_uvjeta[R] neprazan )
            stavi_u_red ( makni_pruv_iz_reda ( Red_uvjeta[R] ), Pripravne_D );
        stavi_u_red ( makni_pruv_iz_reda ( Pripravne_D ), Aktivna_D );
    }

    obnovi kontekst iz opisnika Aktivna_D;
}
```

Tablica 6.1. Ekvivalencija poziva sa pozivima stvarnih sustava (info)

| Funkcija | POSIX | Win32 (Vista+) |
|-----------------------------|------------------------|--------------------------|
| Uđi_u_monitor | pthread_mutex_lock | EnterCriticalSection |
| Izadi_iz_monitora | pthread_mutex_unlock | LeaveCriticalSection |
| Čekaj_u_redu_uvjeta | pthread_cond_wait | SleepConditionVariableCS |
| Oslobodi_iz_reda_uvjeta | pthread_cond_signal | WakeConditionVariable |
| Oslobodi_sve_iz_reda_uvjeta | pthread_cond_broadcast | WakeAllConditionVariable |

6.4. Primjeri sinkronizacije semaforima i monitorima

Zadatak: Problem pet filozofa

```
dretva Filozof (I)
{
    ponavljam {
        uzmi_štapiće (I);
        jedi;
        vrati_štapiće (I);
        misli;
    }
    do zauvijek;
}
```

štapić[i] = 1 – i-ti štapić slobodan (početna vrijednost);
 štapić[i] = 0 – i-ti štapić zauzet

```
m-funkcija uzmi_štapiće (I)
{
    L = I; D = ( I + 1 ) MOD 5;
    Uđi_u_monitor (m);
    dok( štapić[L] + štapić[D] < 2 )
        Čekaj_u_redu (red[I], m);
    štapić[L] = 0;
    štapić[D] = 0;
    Izadi_iz_monitora (m);
}
```

```
m-funkcija vrati_štapiće (I)
{
    L = I; D = ( I + 1 ) MOD 5;
    Uđi_u_monitor (m);
    štapić[L] = 1;
    štapić[D] = 1;
    Oslobodi_iz_reda_uvjeta(red[(L-1)MOD 5],m);
    Oslobodi_iz_reda_uvjeta(red[D], m);
    Izadi_iz_monitora (m);
}
```

Zadatak: Problem pušača

```
dretva Trgovac()
{
    ponavljam {
        (s1, s2) = nasumice odaberi ...;

        Uđi_u_monitor (m);

        dok ( stol_prazan != 1 )
            Čekaj_u_redu_uvjeta (m, red
                [0]);

        stavi_sastojke_na_stol (s1, s2);
        Oslobodi_iz_reda_uvjeta (red[1]);
        Oslobodi_iz_reda_uvjeta (red[2]);
        Oslobodi_iz_reda_uvjeta (red[3]);
        Izadi_iz_monitora (m);
    }
    do ZAUVIJEK;
}
```

```
dretva Pušač(p)
{
    (r1, r2)=sastojci_koje_pušač_nema (p);

    ponavljam {
        Uđi_u_monitor (m);

        dok(na_stolu_sastojci(r1,r2)!=DA)
            Čekaj_u_redu_uvjeta (m, red[p
                ]);

        uzmi_sastojke(r1, r2);
        stol_prazan = 1;
        Oslobodi_iz_reda_uvjeta (red[0]);
        Izadi_iz_monitora (m);

        smotaj, zapali i puši;
    }
    do ZAUVIJEK;
}
```

Početne vrijednosti: stol_prazan = 1

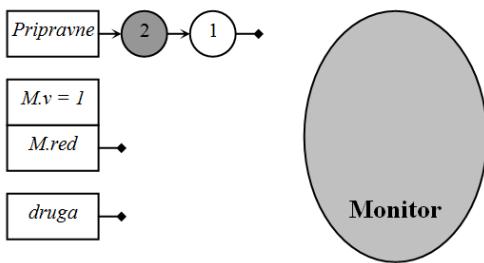
Zadatak: Stanje strukture podataka na jednom primjeru

U sustavu se nalaze dvije dretve koje obavljaju kod prema pseudokodu ispod. Nakon stvaranja dretvi one se nalaze u redu pripravnih prema [1]. Redovi su složeni po redu prispjeća. Stanja sustava za vrijeme rada tih dretvi prikazana su u nastavku. Koriste se monitori s labosa.

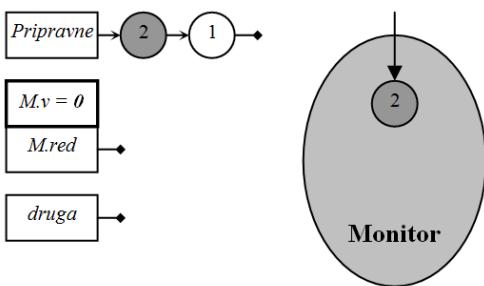
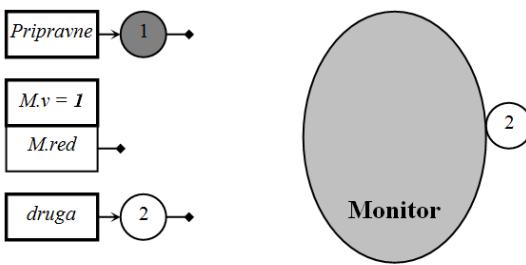
```
Dretval {
    mutex_lock(M);
    ispiši("Prva");
    prva = 1;
    cond_signal(druga, M);
    mutex_unlock(M);
}

Dretva2 {
    mutex_lock(M);
    dok je (prva == 0)
        cond_wait(druga, M);
    ispiši("Druga");
    mutex_unlock(M);
}
```

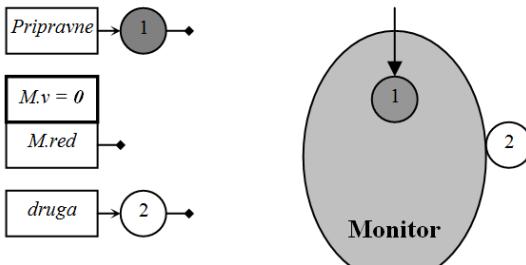
[1] Početno stanje



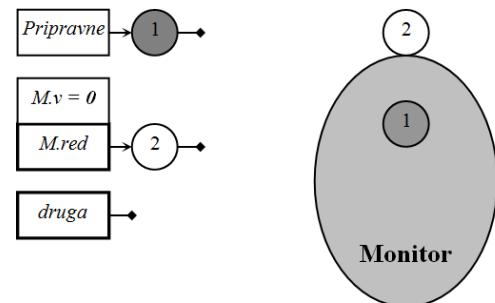
[2] (dretva 2) mutex_lock(M);

[3] (dretva 2) dok je (prva == 0)
cond_wait(druga, M);

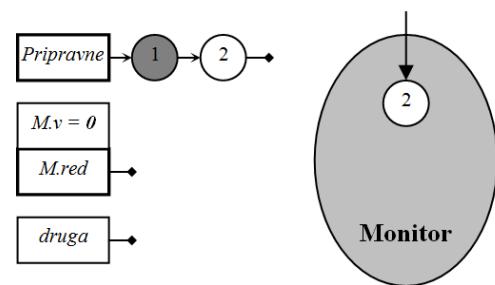
[4] (dretva 1) mutex_lock(M);



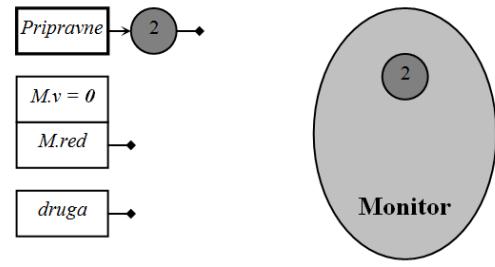
[5] (dretva 1) ispiši("Prva");
prva = 1;
cond_signal(druga, M);



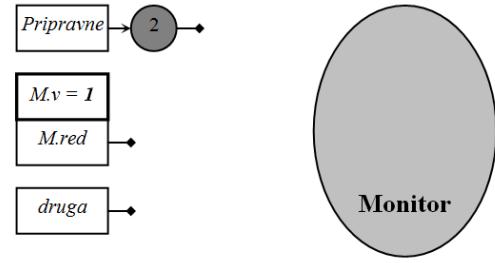
[6] (dretva 1) mutex_unlock(M);



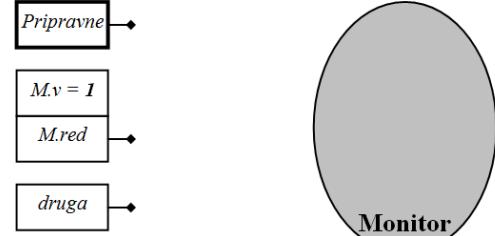
[7] Dretva 1 završava



[8] (dretva 2): ispiši("Druga");
mutex_unlock(M);



[9] Dretva 2 završava



Zadatak: 6.3. Problem starog mosta

Stari most je uski most i stoga postavlja ograničenja na promet. Na njemu istovremeno smiju biti najviše tri automobila koja voze u istom smjeru. Simulirati automobile dretvom Auto koja obavlja niže navedene radnje. Napisati pseudokod monitorskih funkcija Popni_se_na_most (smjer) i Siđi_s_mosta () .

```
Dretva Auto (smjerA) // smjerA = 0 ili 1
{
    Popni_se_na_most (smjerA);
    prijedî_most;
    Siđi_s_mosta(smjerA);
}
```

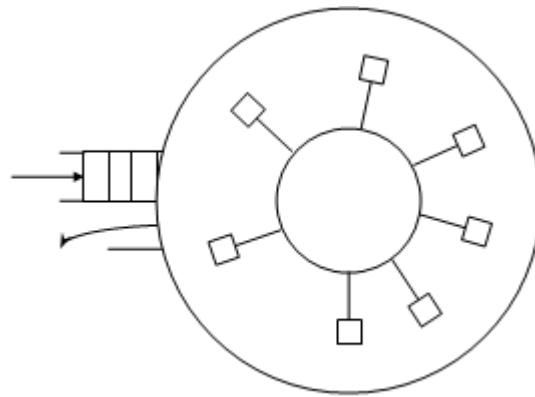
Zadatak: Ping-pong dretve

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:

- ispis bude pojedinačan (unutar kritičnog odsječka)
- dretve *ping* i *pong* naizmjence ispisuju poruke (ispis će biti: *ping pong ping pong ...*)
- dretve *ping* i *pong* ispisuju poruke tako da se uvijek pojavljuju barem dva *ping*-a prije svakog *pong*-a (npr. ispis će biti: *ping ping ping pong ping ping ...*)

Zadatak: 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 mesta (*BR_MJESTA*), a pokretanje vrtuljka napraviti tek kada je pun.



Zadatak: Parking

Neki parking (npr. FER-ov) ima dvije rampe: jednu za ulaz i drugu za izlaz. Automobili ulaze na parking uz pomoć daljinskog uređaja. Svaki uređaj ima jedinstveni broj za koji u bazi sustava postoji definirano stanje: 0 – auto nije na parkingu i 1 – auto je na parkingu. Ulazak je automobilima moguć samo u stanju 0, a izlazak samo u stanju 1. Automobil ne može ući ako je na parkingu već MAX

automobila. Simulirati sustav dretvama, tj. napisati funkcije `ulaz(id)` i `izlaz(id)` koje pozivaju auti s brojem uređaja `id`.

Ponekad se greškom ne evidentira ulazak ili izlazak automobila te se ne promijeni stanje uređaja i broj mjesta. Ako je potrebno, portir ima pristup bazi i može promijeniti stanje automobila i broj mjesta. Dodati monitorske funkcije `portir_postavi(id)` i `portir_obriši(id)` koje poziva portir, a koje postavljaju odgovarajuće stanje uređaja i broj mjesta.

Zadatak: H_2O

Sinkronizirati dretve vodika i kisika koje stvaraju molekulu vode H_2O .

Zadatak: Vukovi i ovce

U odvojenim dijelovima zoološkog vrta stanuju vukovi i ovce. Jedino što ove dvije skupine životinja dijele jest pojilište. Propuštanje u pojilište obavlja se kontroliranim vratima, tako da sustav uvijek može kontrolirati tko je na pojilištu te sukladno tome pustiti iduću životinju ili ne, ako bi mogao nastati problem. Problem koji treba spriječiti jest da se na pojilištu istovremeno pojave i ovce i vukovi (bar jedan vuk i bar jedna ovca). Osmisliti dretve koje simuliraju ovce i vukove koji dolaze na pojilište radi "operacije" `pije vodu`, tj. dodati sinkronizaciju i prije i poslije te operacije u pseudokod `žedna_životinja (tip)`. Sinkronizaciju riješiti tako da se problem izgladnjivanja ublaži. Za sinkronizaciju koristiti monitore.

Zadatak: Čitači i pisači

Riješiti problem sinkronizacije dretvi čitača i pisača (napisati pseudokod funkcija `pisac()` i `čitač()`) korištenjem semafora i/ili monitora te dodatnih varijabli (po potrebi). Prepostaviti da pisači u kritičnom odsječku pišu funkcijom `piši(X)`, a čitači čitaju funkcijom `čitaj(Y)`. Kada neki čitač čita niti jedan pisač ne smije pisati, dok istovremeno drugi čitači mogu čitati. Kada neki od pisača piše svi ostali moraju čekati (i čitači i pisači).

Zadatak: Besplatno piće na sajmu

Sustav kojeg treba simulirati dretvama sastoji se od više *konobara* i više *posjetitelja*. Svaki konobar obavlja ciklički posao: uzima čistu praznu čašu; puni je ili čajem ili kuhanim vinom ili pivom (svaki konobar uvijek puni isto); stavlja punu čašu na šank te ponavlja posao. Postoje tri aparata za punjenje pića koji mogu raditi paralelno: jedan za čaj, drugi za kuhano vino a treći za pivo. Svaki aparat se koristi pojedinačno (npr. dok se jedan čaj ne natoči ostali konobari koji žele natočiti čaj čekaju). Stavljanje pića na zajednički stol također treba obaviti pojedinačno. Posjetitelji se kao i konobari dijele po tipu na one koji hoće čaj, one koji hoće vino te one koji hoće pivo. Više posjetitelja može paralelno uzimati piće sa šanka. Posjetitelj koji želi piće kojeg trenutno nema na stolu čeka (ali ne sprječava druge da uzmu svoje piće). Po uzimanju pića posjetitelj se miče od stola (nestaje iz simuliranog sustava).

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"?
4. Prikazati primjer nastanka potpunog zastoja u sustavu koji koristi semafore za sinkornizaciju.
5. Navesti nužne uvjete za nastanak potpunog zastoja.
6. Opisati koncept monitora.
7. Navesti jezgrine funkcije potrebne za ostvarenje monitora.
8. Koje su tipične operacije koje se izvode unutar monitorske funkcije?
9. Monitorima sinkronizirati dretve koje simuliraju rad "pet filozofa".
10. Koje sve aspekte uključuje "ispravna sinkronizacija"?
11. Što je to "izgladnjivanje" u kontekstu sinkronizacije i korištenja zajedničkih sredstava?
12. U nekom hipotetskom sustavu na predavanja dolazi N studenata. Svaki student pri ulasku u dvoranu treba se prijaviti preko zasebnog uređaja (prijava ide slijedno, student po student). Nakon što uđe N studenata u dvoranu može ući predavač. Po završetku sata, studenti odlaze i odjavljuju se (preko istog uređaja). Tek kad zadnji student izđe, izlazi i predavač. Napisati dretve student() i predavač() koji koriste semafore/monitore za sinkornizaciju na gore opisani način.

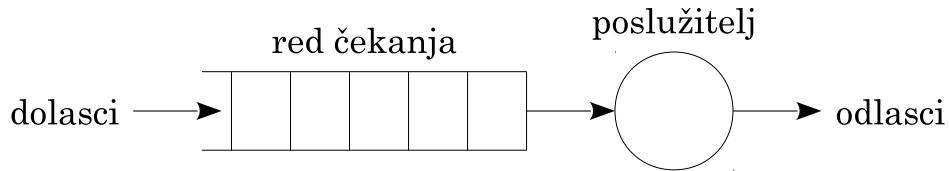
7. Analiza vremenskih svojstava računalnog sustava

Ili: kako raspoređivati dretve?

Da bi to mogli odrediti trebamo analizirati dinamičko ponašanje računalnog sustava. Kako?

- simulacijom
- praćenjem stvarnog sustava
- korištenjem modela \Leftarrow ovako ćemo mi

7.1. Deterministički sustav



Slika 7.1. Model poslužitelja

- svi događaji su poznati ili predvidljivi
- neki posao se u sustavu pojavljuje u trenutku t_d a iz njega odlazi u t_n
- vrijeme zadržavanje posla u sustavu je:

$$T = t_n - t_d \quad (7.1.)$$

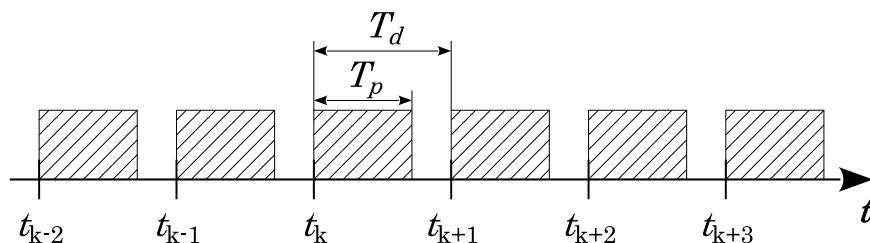
- kad dođe u sustav, poslužitelj može biti slobodan ili zauzet
 - ako je poslužitelj zauzet novi posao čeka u redu
- vrijeme čekanja u redu označavamo sa: T_r
- vrijeme posluživanja posla označimo sa: T_p
- vrijeme zadržavanja posla u sustavu označimo sa: T

$$T = T_r + T_p \quad (7.2.)$$

- ako je red uvijek prazan:

$$T_p = T \quad (7.3.)$$

Ako su svi poslovi isti i periodički dolaze s periodom T_d (mogli bi reći i da se jedan posao ponavlja s tom periodom) tada takve poslove požemo prikazati slikom 7.2.



Slika 7.2. Periodički poslovi

Da se poslovi ne bi gomilali mora biti:

$$T_p \leq T_d \quad (7.4.)$$

Iskoristivost procesora:

$$\rho = \frac{T_p}{T_d} \quad (7.5.)$$

ili u postocima:

$$\eta = \frac{T_p}{T_d} \times 100\% \quad (7.6.)$$

Recipročna vrijednost periode dolaska novih poslova T_d je:

$$\alpha = \frac{1}{T_d} \quad (7.7.)$$

- α – broj dolazaka novih poslova u jedinici vremena
- $\frac{1}{\alpha}$ – vrijeme (interval) između dva dolaska

Slično, recipročna vrijednost od T_p je:

$$\beta = \frac{1}{T_p} \quad (7.8.)$$

- β – broj poslova koje bi poslužitelj mogao obraditi u jedinici vremena
- $\frac{1}{\beta}$ – vrijeme obrade (posluživanja)

Očito je:

$$\rho = \frac{\alpha}{\beta} \quad (7.9.)$$

ρ – iskoristivost poslužitelja (opterećenje)

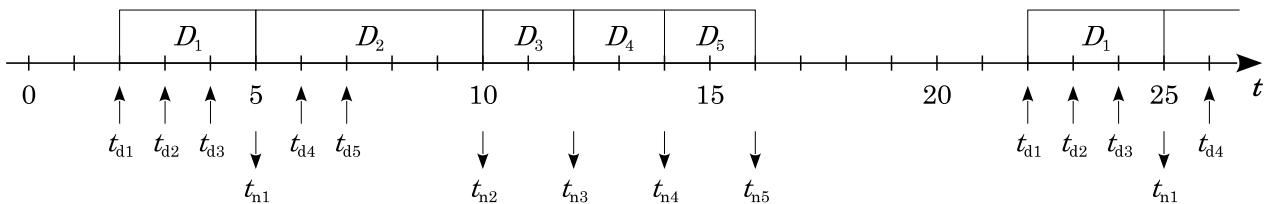
Kada bi T_p bilo jednako T_d poslužitelj bi imao 100% iskoristivost. To je dozvoljeno samo u determinističkim slučajevima!!!

Zadatak: Primjer 7.1. (ispitni zadatak)

Prepostavimo da sustav obraduje pet poslova koji u njega dolaze periodno s periodom od 20 jedinica vremena. Trenuci dolazaka u prvoj periodi koja započinje s $t = 0$ i trajanje poslova navedeni su u tablici.

| Posao | t_d | T_p |
|-------|-------|-------|
| D_1 | 2 | 3 |
| D_2 | 3 | 5 |
| D_3 | 4 | 2 |
| D_4 | 6 | 2 |
| D_5 | 7 | 2 |

Ta će se skupina poslova ponavljati s periodom od 20 jedinica vremena.



Vremensko ponašanje može se opisati tablicom:

| Posao | t_d | T_p | t_n | T | T_r |
|-------|-------|-------|-------|-----|-------|
| D_1 | 2 | 3 | 5 | 3 | 0 |
| D_2 | 3 | 5 | 10 | 7 | 2 |
| D_3 | 4 | 2 | 12 | 8 | 6 |
| D_4 | 6 | 2 | 14 | 8 | 6 |
| D_5 | 7 | 2 | 16 | 9 | 7 |

Oznake:

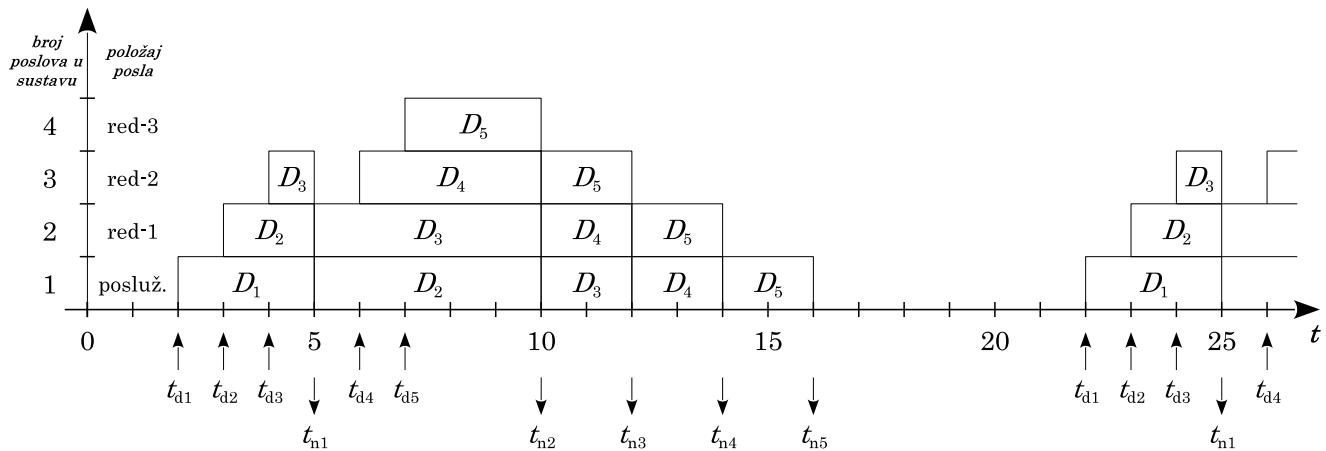
- \bar{T} – prosječno zadržavanje poslova u sustavu
- \bar{T}_r – prosječno čekanje u redu
- \bar{T}_p – prosječno trajanje posluživanja

$$\bar{T} = \frac{3 + 7 + 8 + 8 + 9}{5} = 7 \quad (7.10.)$$

$$\bar{T}_p = \frac{3 + 5 + 2 + 2 + 2}{5} = 2.8 \quad (7.11.)$$

$$\bar{T}_r = \bar{T} - \bar{T}_p = 4.2 \quad (7.12.)$$

Prosječan broj poslova u sustavu: $\bar{n} = ?$



Slika 7.3. Broj poslova u sustavu

$\bar{n} = ?$ se može izračunati kao integral površine podijeljen vremenom periode (20):

$$\bar{n} = \frac{35}{20} = 1,75 \quad (7.13.)$$

površina = suma zadržavanja svih poslova u sustavu

Iz slike se \bar{n} može izračunati prema:

$$\bar{n} = \frac{T_1 + T_2 + T_3 + T_4 + T_5}{20} = \frac{5 \times \bar{T}}{20} = \frac{5}{20} \times \bar{T} = 1,75 \quad (7.14.)$$

Obzirom da je broj poslova u jedinici vremena $\alpha = \frac{5}{20}$ slijedi:

$$\bar{n} = \alpha \times \bar{T} \implies \text{Littleovo pravilo} \quad (7.15.)$$

Littleovo pravilo vrijedi općenito, ne samo za determinističke sustave!

Intuitivni dokaz (statistički gledano):

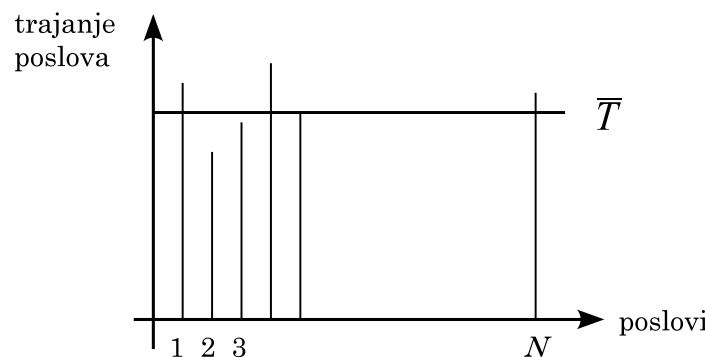
Promatrajmo vremenski interval \bar{T} . U njemu dođe $\alpha \cdot \bar{T}$ novih poslova (α u jedinici vremena). Na početku tog intervala u sustavu su samo poslovi koji su prije došli, a na kraju samo oni koji su došli za vrijeme tog intervala (jer statistički se poslovi zadržavaju \bar{T} pa su svi koji su došli prije i izašli tijekom \bar{T}). Dakle, na kraju intervala u sustavu imamo $\alpha \cdot \bar{T}$ poslova. Ako u tom trenutku imamo toliko poslova zašto bi u nekom drugom trenutku to bilo različito (statistički gledano)?

Pokušaj dokaza Littleova pravila (informativno)

Promatrajmo veći vremenski interval $T \gg \bar{T}$

U tom intervalu poslužitelj obradi N poslova. Ako je T dovoljno velik, onda se rubni slučajevi mogu zanemariti (poslovi koji su počeli prije intervala i dovršili izvođenje u intervalu te poslovi koji su započeli u intervalu a dovršili iza njega). Razmotrimo detaljnije tih N poslova.

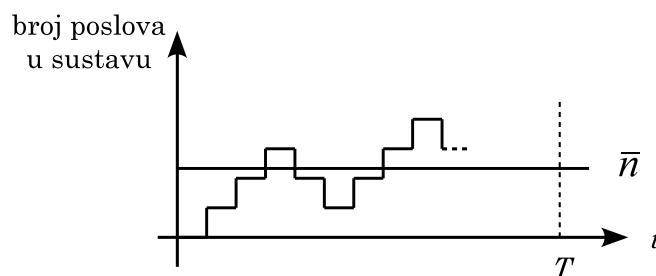
Graf zadržavanja poslova u sustavu prema pojedinom poslu (poslovi su numerirani od 1 do N) neka izgleda:



Iz slike slijedi:

$$\bar{T} = \frac{\sum T_i}{N}, \quad \alpha = \frac{N}{T} \quad (7.16.)$$

Isti se poslovi s njihovim trajanjima mogu pokazati i u grafu koji pokazuje broj poslova u sustavu u nekom trenutku. Karikirano, takva slika izgleda kao u nastavku



To su isti poslovi kao i na prijašnjem grafu samo "polegnuti" od trenutka pojave do napuštanja sustava.

Srednji broj poslova se može izračunati kao:

$$\bar{n} = \frac{\text{površina}}{\text{period}} = \frac{\sum T_i}{T} \quad (7.17.)$$

Ako se to raspiše (pomnoži s "1" i iskoriste prijašnje formule):

$$\bar{n} = \frac{\sum T_i}{T} = \frac{\sum T_i}{T} \times \frac{N}{N} = \frac{\sum T_i}{N} \times \frac{N}{T} = \bar{T} \times \alpha = \alpha \times \bar{T} \implies \text{Littleovo pravilo} \quad (7.18.)$$

7.2. Nedeterministički sustav

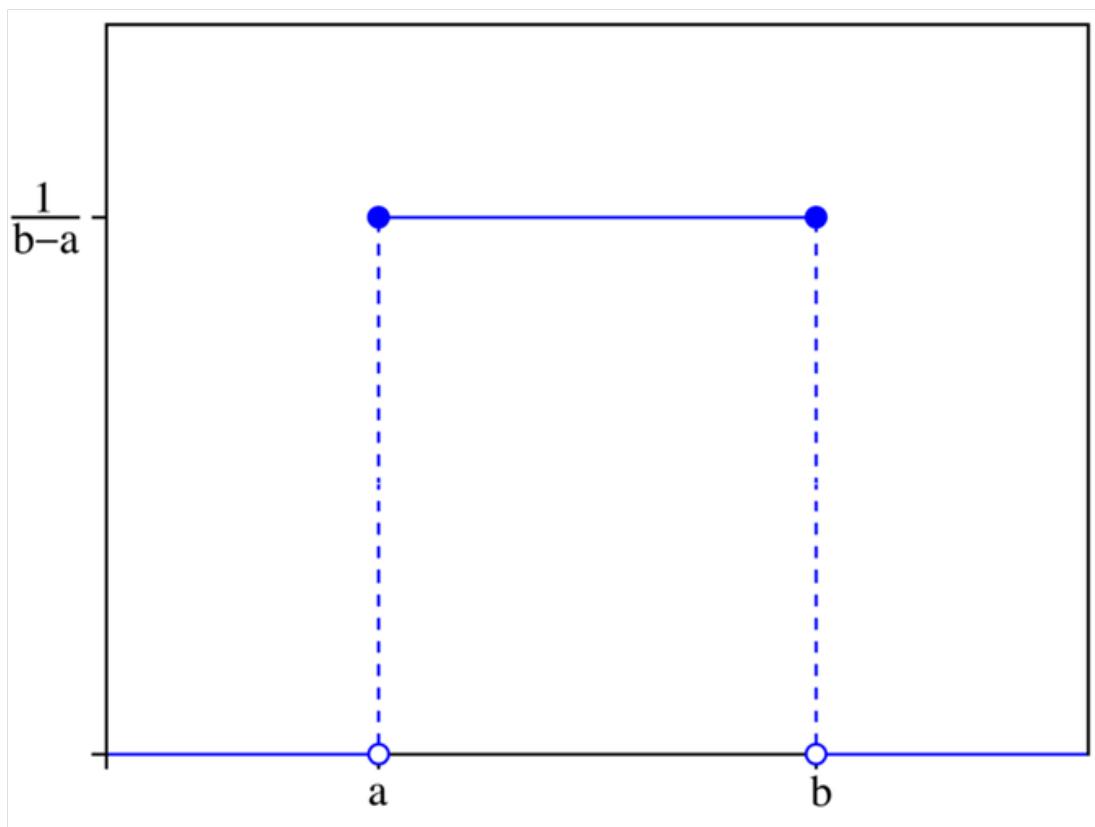
Pretpostavke:

- dolasci se podvrgavaju Poissonovoj razdiobi s parametrom (očekivanjem) α
 - α je prosječan broj dolazaka novih poslova u jedinici vremena
 - $\frac{1}{\alpha}$ je prosječno vrijeme između dolaska dva posla
- trajanje obrade podvrgava se eksponencijalnoj razdiobi s parametrom (očekivanjem) $\frac{1}{\beta}$
 - $\frac{1}{\beta}$ je prosječno trajanje obrade jednog posla
 - β je prosječan broj poslova poslužitelj može obraditi u jedinici vremena

Zašto te razdiobe?

- zato jer modeliraju stvarne sustave
- zato jer se inače koriste

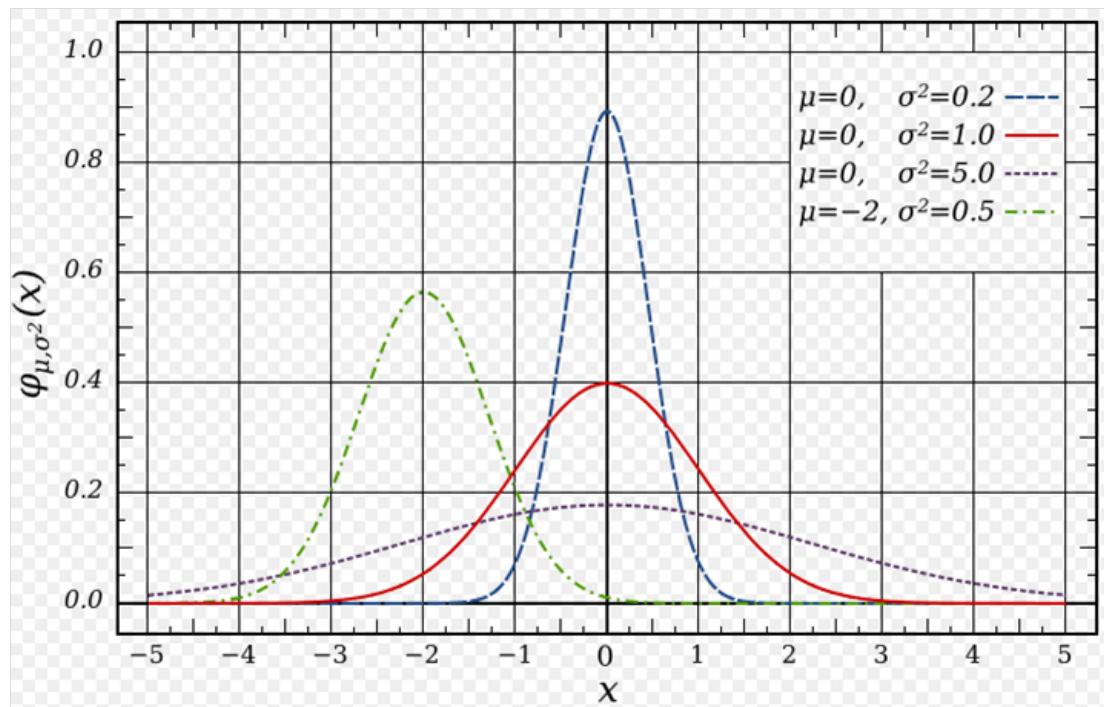
7.2.1. O razdiobama



Slika 7.4. Uniformna razdioba

Primjeri uniformne razdiobe:

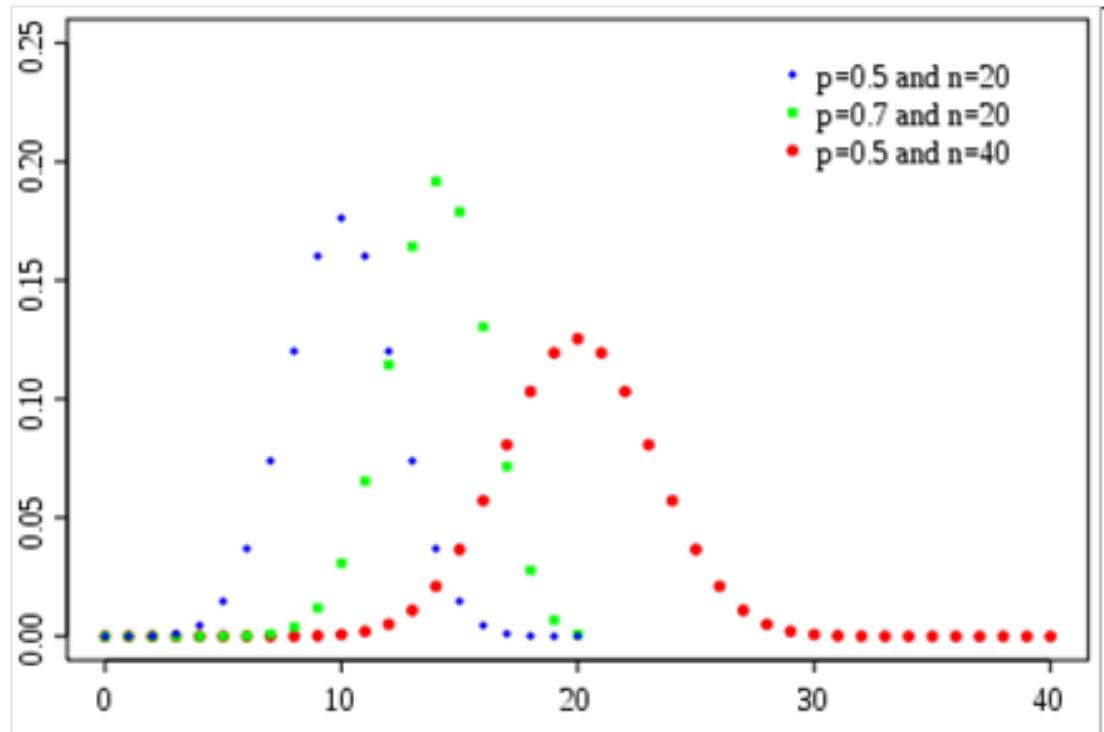
- jedno bacanje kocke
- generator slučajnih brojeva



Slika 7.5. Gaussova razdioba

Primjeri Gaussove razdiobe:

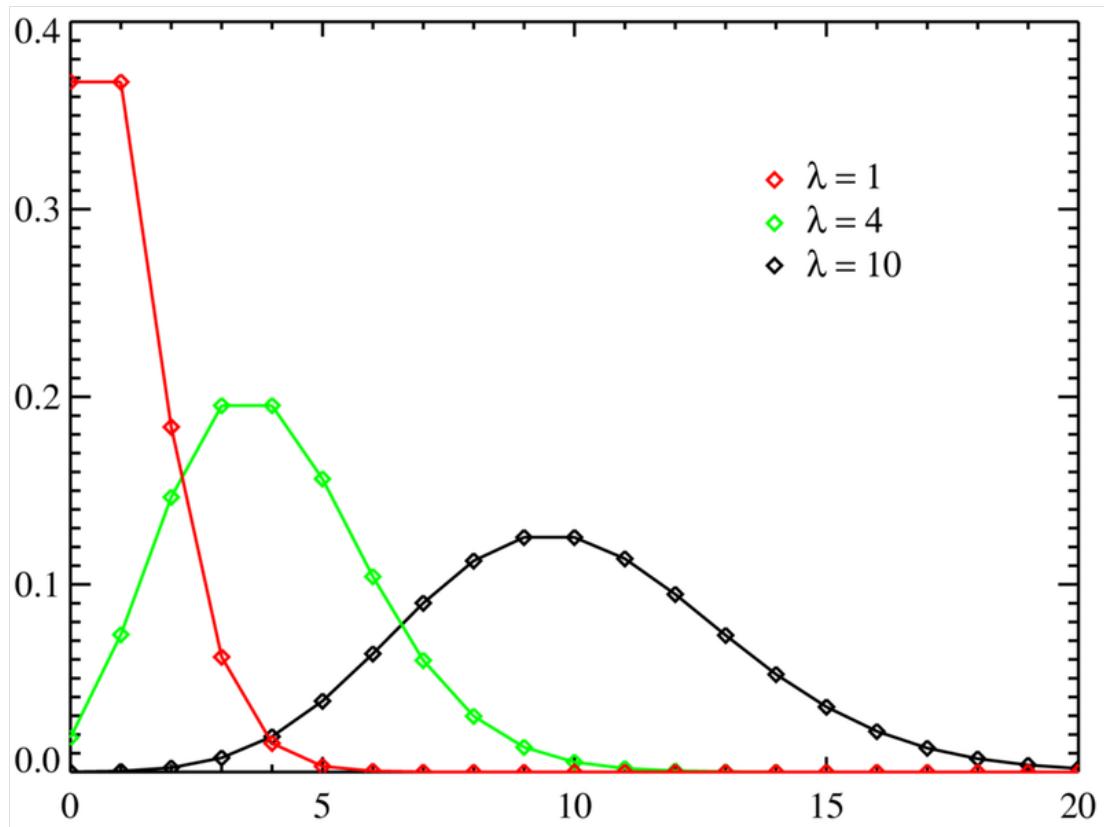
- raspodijela bodova na predmetu, broj pojedinih ocjena
- visina/težina/* skupine ljudi



Slika 7.6. Binomna razdioba

Primjeri binomne razdiobe:

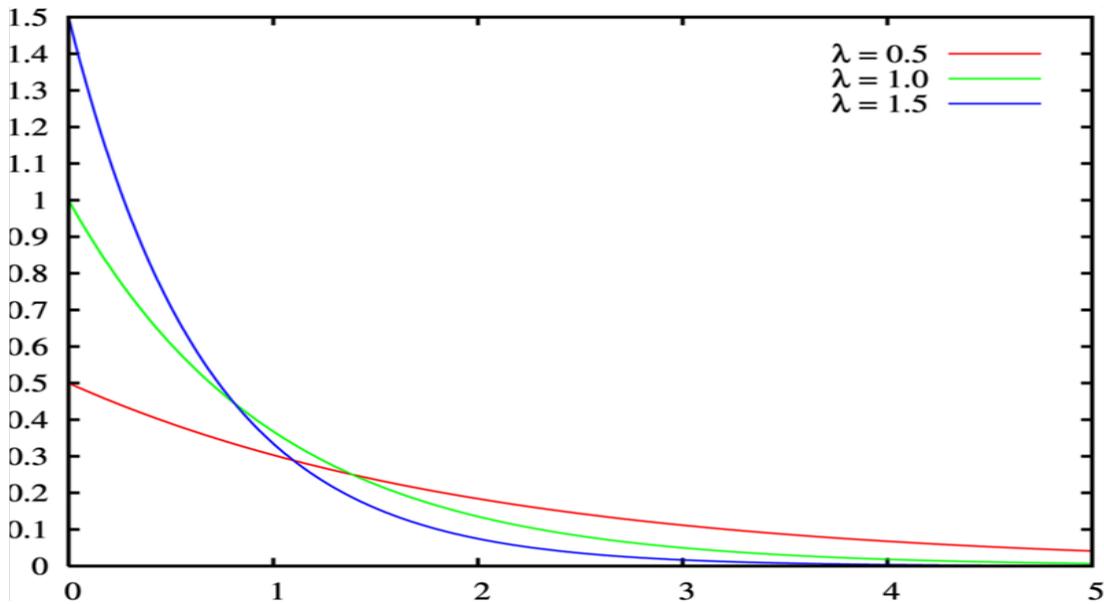
- vjerojatnost n događaja (a svaki događaj je nezavisan i ima poznatu vjerojatnost pojave)



Slika 7.7. Poissonova razdioba

Primjeri Poissonove razdiobe:

- broj novih klijenata u banci/pošti (u nekom intervalu)
- broj zahtjeva nekom web poslužitelju (u nekom intervalu)



Slika 7.8. Eksponencijalna razdioba

Primjeri eksponencijalne razdiobe:

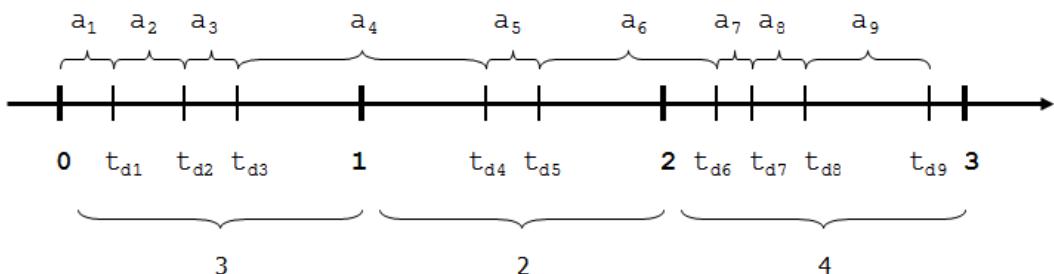
- trajanje posluživanja nekog klijenta/zahtjeva
- razmak između dva događaja (dva dolaska zahtjeva)

7.2.2. Modeliranje dolazaka

Modeliranje dolazaka

("d" u indeksu označava "dolazak")

$a_1, a_2, a_3, a_4, \dots \Rightarrow \text{eksponencijalna razdioba! } (\alpha)$



broj događaja u jedinici vremena \Rightarrow Poissonova razdioba (α)

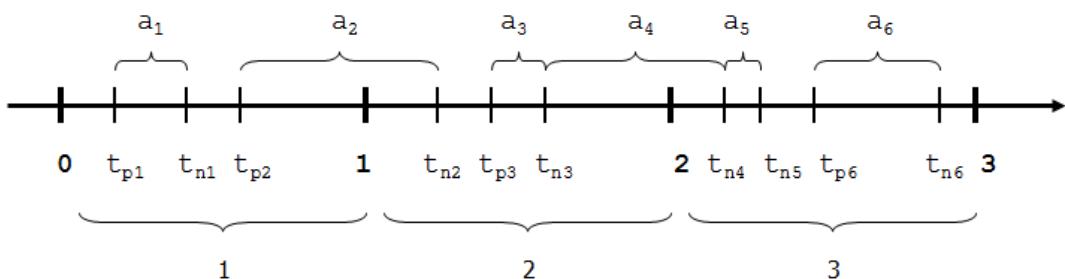
Slika 7.9. Modeliranje broja dolazaka poslova Poissonovom razdiobom

7.2.3. Modeliranje obrade

Modeliranje obrade

("p" u indeksu označava "početak obrade", a "n" označava "napuštanje sustava")

$a_1, a_2, a_3, a_4, \dots \Rightarrow \text{eksponencijalna razdioba! } (\beta)$



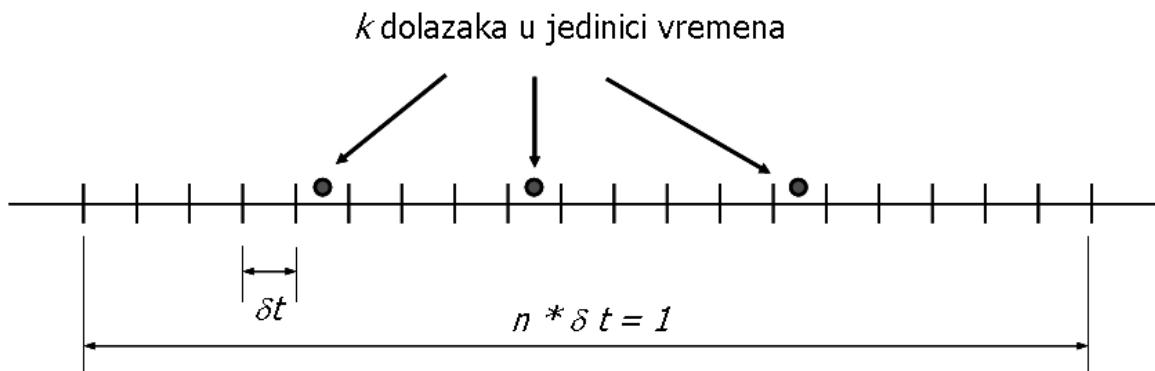
broj odlazaka u jedinici vremena \Rightarrow ? (nije ni bitno za "nas")

Slika 7.10. Modeliranje trajanja obrade eksponencijalnom razdiobom

7.3. Poissonova razdioba (izvod iz binomne) (info)

Razmatramo jedinični vremenski interval (jedna sekunda).

Podijelimo ga na n odsječaka.



Neka je vjerojatnost događaja u jednom odsječku: p

Suprotna vjerojatnost (odsustvo događaja) je: $1 - p = q$

Vjerojatnost da se dogodilo k događaja u jedinici vremena je:

$$b(k, n, p) = \binom{n}{k} p^k q^{n-k} \quad (7.19.)$$

- $n, p \Rightarrow$ parametri razdiobe
- $k \Rightarrow$ slučajna varijabla

Primjer 7.1. Dobivanje 2 šestice u 10 bacanja kocke

$$b\left(2, 10, \frac{1}{6}\right) = \binom{10}{2} \left(\frac{1}{6}\right)^2 \left(\frac{5}{6}\right)^{10-2} = 0,29 \quad (7.20.)$$

Vjerojatnost da se nije dogodio niti jedan događaj (u jedinici vremena):

$$b(0, n, p) = q^n = (1 - p)^n \quad (7.21.)$$

Vjerojatnost da se dogodio barem jedan događaj (u jedinici vremena):

$$b(k > 0, n, p) = 1 - q^n \quad (7.22.)$$

Poissonova razdioba dobiva se kada $n \rightarrow \infty$ i $p \rightarrow 0$, ali tako da umnožak $n \cdot p$ bude konačna vrijednost: $\lambda = n \cdot p$:

$$p(0, \lambda) = b(0, n, p) = q^n = (1 - p)^n = \left(1 - \frac{\lambda}{n}\right)^n = \left[\left(1 + \frac{1}{-\frac{n}{\lambda}}\right)^{-\frac{n}{\lambda}}\right]^{-\lambda} \quad (7.23.)$$

Kada $n \rightarrow \infty$ tada:

$$p(0, \lambda) = e^{-\lambda} \quad (7.24.)$$

Kako izračunati $p(k, \lambda)$ – vjerojatnost k događaja u jedinici vremena?

Opet krenemo od binomne razdiobe i omjera:

$$\frac{b(k, n, p)}{b(k-1, n, p)} = \frac{\binom{n}{k} p^k q^{n-k}}{\binom{n}{k-1} p^{k-1} q^{n-(k-1)}} = \frac{n - (k-1)}{k} \frac{p}{q} = \frac{n}{k} \frac{p}{q} - \frac{k-1}{k} \frac{p}{q} \quad (7.25.)$$

Prelaskom na limese: $n \rightarrow \infty$, $p \rightarrow 0$, $q \rightarrow 1$, $n \cdot p \rightarrow \lambda$:

$$\frac{p(k, \lambda)}{p(k-1, \lambda)} = \frac{\lambda}{k \cdot 1} - 0 = \frac{\lambda}{k} \rightarrow p(k, \lambda) = \frac{\lambda}{k} \cdot p(k-1, \lambda) \quad (7.26.)$$

te indukcijom

$$\begin{aligned} p(1, \lambda) &= \frac{\lambda}{1} \cdot p(0, \lambda) = \frac{\lambda}{1} e^{-\lambda} \\ p(2, \lambda) &= \frac{\lambda^2}{2} e^{-\lambda} \\ p(3, \lambda) &= \frac{\lambda^3}{3!} e^{-\lambda} \\ &\vdots \end{aligned} \quad (7.27.)$$

$$p(k, \lambda) = \frac{\lambda^k}{k!} e^{-\lambda} \quad (7.28.)$$

Što je λ ? Očito $n \cdot p$, ali ovisi kako dijelimo jedinični interval! A n i p koristimo samo pri izvodu, ne i kasnije pri korištenju.

Izračunajmo očekivanje slučajne varijable k (očekivanje je slično prosječnoj vrijednosti):

$$\begin{aligned} E(k) &= \sum_{k=0}^{\infty} k \cdot p(k, \lambda) = \sum_{k=1}^{\infty} k \cdot \frac{\lambda^k}{k!} \cdot e^{-\lambda} = e^{-\lambda} \sum_{k=1}^{\infty} \frac{\lambda \cdot \lambda^{k-1}}{(k-1)!} \\ &= e^{-\lambda} \cdot \lambda \sum_{k=1}^{\infty} \frac{\lambda^{k-1}}{(k-1)!} = e^{-\lambda} \cdot \lambda \cdot e^{\lambda} = \lambda \end{aligned} \quad (7.29.)$$

Vrijednost λ je prosječna vrijednost slučajne varijable k . Npr. prosječni broj dolazaka novih poslova u jedinici vremena.

Kako izračunati vjerojatnost za proizvoljni vremenski period (ne samo jedinični)?

Isti izvod, samo umjesto n staviti $t \cdot n$ te se dobiva:

$$p(k(t), \lambda) = \frac{(\lambda t)^k}{k!} e^{-\lambda t} \quad (7.30.)$$

Koja je vjerojatnost da u intervalu t nema ni jednog događaja?

$$p(k(t) = 0, \lambda) = e^{-\lambda t} \quad (7.31.)$$

Da ima bar jedan?

$$p(k(t) > 0, \lambda) = 1 - e^{-\lambda t} \quad (7.32.)$$

Napomena: Poissonova razdioba je diskretna, k je cijeli broj!

Primjer 7.2.

U jednoj minuti prosječno padne 100 kapi kiše na površinu stola. Koja je vjerojatnost da u sljedećoj sekundi na stol padnu dvije kapi?

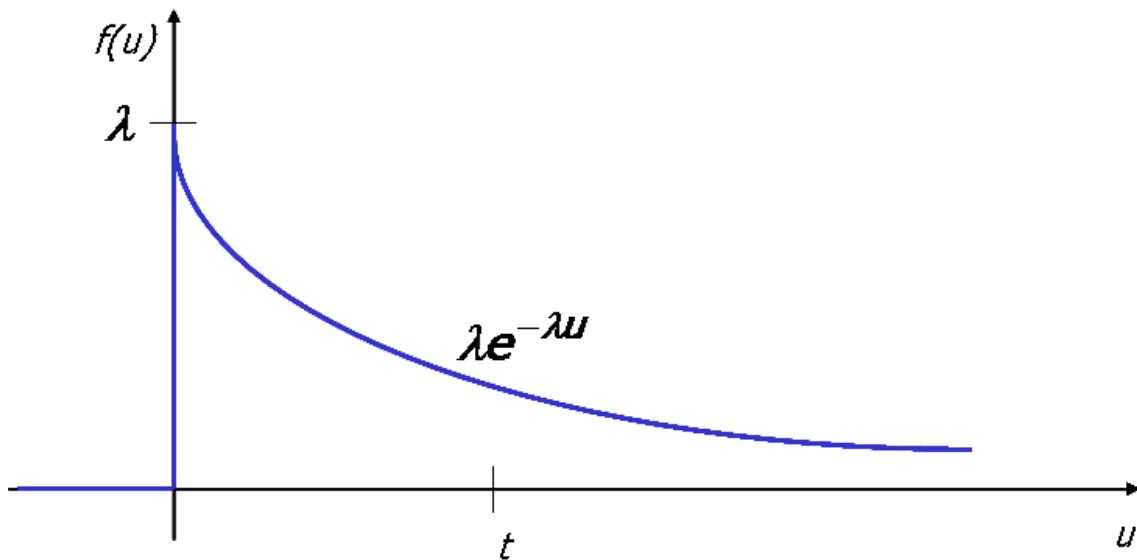
$$p\left(k(1) = 2, \lambda = \frac{100}{60}\right) = \frac{\lambda^2}{2!} \cdot e^{-\lambda} = 26,2\% \quad (7.33.)$$

Vjerojatnosti idu redom (u %) (k(1), za k iz {0-5}) p= {19; 31; 26; 15; 6; 2}

7.4. Eksponencijalna razdioba

Razdioba je zadana funkcijom gustoće vjerojatnosti:

$$f(u) = \begin{cases} 0, & u < 0 \\ \lambda e^{-\lambda u}, & u \geq 0 \end{cases} \quad (7.34.)$$



(opet λ , ali samo u modelu, kasnije ćemo uz tu razdiobu vezivati β)

Eksponencijalna razdioba je kontinuirana razdioba! Parametar razdiobe može poprimiti realnu vrijednost.

Vjerojatnost da slučajna varijabla T poprimi jednu diskretnu vrijednost je 0.

Računa se vjerojatnost da slučajna varijabla poprimi vrijednost iz intervala $[a, b]$ kao integral gustoće vjerojatnosti od a do b .

$$p(a < T < b) = \int_a^b f(u) du \quad (7.35.)$$

Eksponencijalnom razdiobom modeliramo razmake između događa kao i trajanje obrade.

Povezanost eksponencijalne i Poissonove razdiobe!

Vjerojatnost da je vrijeme između dva događaja veće od t :

$$p(t < T < \infty, \lambda) = \int_t^\infty \lambda e^{-\lambda u} du = \lambda \int_t^\infty e^{-\lambda u} du = \lambda \left(-\frac{1}{\lambda} e^{-\lambda u} \right) \Big|_t^\infty = e^{-\lambda t} \quad (7.36.)$$

To smo već imali, to je vjerojatnost da se u intervalu t nije dogodio ni jedan događaj!

I obratno, vjerojatnost da se dogodio barem jedan događaj jest vjerojatnost da je razmak između dva događaja manje od t :

$$p(T < t, \lambda) = \int_0^t \lambda e^{-\lambda u} du = -e^{-\lambda u} \Big|_0^t = 1 - e^{-\lambda t} \quad (7.37.)$$

Koje je značenje parametra λ ? Koristimo isti princip, računamo očekivanje sl.var. T :

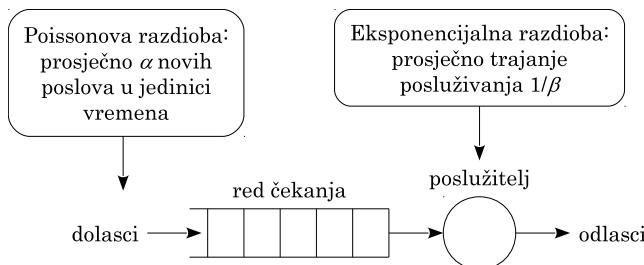
$$E(T) = \int_0^\infty u \cdot \lambda \cdot e^{-\lambda u} du = (\text{supst. } v = \lambda u) = \frac{1}{\lambda} \int_0^\infty v \cdot e^{-v} dv = \frac{1}{\lambda} \quad (7.38.)$$

$\frac{1}{\lambda}$ – prosječno vrijeme između dva događaja!

λ – prosječan broj događaja u jedinici vremena.

7.4.1. Modeliranje poslužitelja s Poissonovom razdiobom dolaska novih poslova i eksponencijalnom razdiobom trajanja obrade

$\frac{1}{\lambda}$ je prosječno trajanje obrade, a λ je prosječan broj poslova koje poslužitelj može obaviti u jedinici vremena (prethodno označen s β).



Imamo razdiobe, α koji modelira dolaske i β koji modelira obradu.

Znamo izračunati opterećenje poslužitelja $\rho = \frac{\alpha}{\beta}$, vjerojatnosti dolaska i odlaska u nekom intervalu.

Ne znamo:

- \bar{T} – koliko poslovi prosječno čekaju
- \bar{n} – koliki je prosječan broj poslova u sustavu
- tj. koja je kvaliteta usluge (koliko se čeka na uslugu) i koliki mora biti red?

Littleova formula $\bar{n} = \alpha T$ povezuje ova dva parametra pa je dovoljno izračunati jedan.

7.4.2. Izračun (izvod) prosječnog broja poslova u sustavu (info)

Ako sa i označimo slučajnu varijablu koja označava *broj poslova u sustavu* u nekom trenutku, tada očekivanje te varijable $E(i)$ je prosječan broj poslova u sustavu, tj. \bar{n} .

$$E(i) = \sum_{i=0}^{\infty} i \cdot p_i \quad (7.39.)$$

Označimo s $p_i(t)$ vjerojatnost da se u nekom trenutku u sustavu nalazi i poslova. Slučajna varijabla je i .

Trebaju nam dakle vjerojatnosti p_i da se u sustavu nalazi i poslova (u trenutku t).

Prepostavimo da je sustav u stohastičkoj ravnoteži:

- sve vjerojatnosti konstantne (vjerojatnost događaja)
- ako dovoljno dugo promatramo sustav to je ispunjeno

(Prije smo imali sličnu oznaku ali za vjerojatnost da se u intervalu t pojavilo novih k poslova!)

Za sustav u stohastičkoj ravnoteži vrijedi $p_i(t) = \text{konstantno}!$

Promatrajmo sustav u nekom stanju i . Kolike su vjerojatnosti da će se u intervalu Δt nešto dogoditi ili da se neće ništa dogoditi (doći novi poslovi ili otići/završiti neki)?

Vjerojatnost da neće doći novi poslovi je:

$$p(k(\Delta t) = 0, \alpha) = e^{-\alpha \Delta t} \quad (7.40.)$$

Ako se to razvije u Taylorov red:

$$p(k(\Delta t) = 0, \alpha) = e^{-\alpha \Delta t} = 1 - \alpha \Delta t + \frac{(\alpha \Delta t)^2}{2!} - \frac{(\alpha \Delta t)^3}{3!} + \dots \approx 1 - \alpha \Delta t \quad (7.41.)$$

Suprotna vjerojatnost, da će se dogoditi barem jedan događaj može se aproksimirati sa:

$$p(k(\Delta t) > 0, \alpha) = 1 - e^{-\alpha \Delta t} \approx \alpha \Delta t \quad (7.42.)$$

Prepostavimo da je Δt tako mali da je to vjerojatnost upravo jednog događaja.

Vjerojatnost da nema niti jednog odlaska u intervalu Δt (vjerojatnost da je trajanje obrade veće od Δt , vrijedi samo za stanja $i > 0$, tj. imamo bar 1 posao u sustavu) = vjerojatnost da je vrijeme između dva događaja veća od t :

$$p(t < T < \infty, \beta) = e^{-\beta \Delta t} = 1 - \beta \Delta t + \frac{(\beta \Delta t)^2}{2!} - \frac{(\beta \Delta t)^3}{3!} + \dots \approx 1 - \beta \Delta t \quad (7.43.)$$

Suprotna vjerojatnost, da će se barem jedan posao napustiti (barem jedan odlazak) sustav je prema tome:

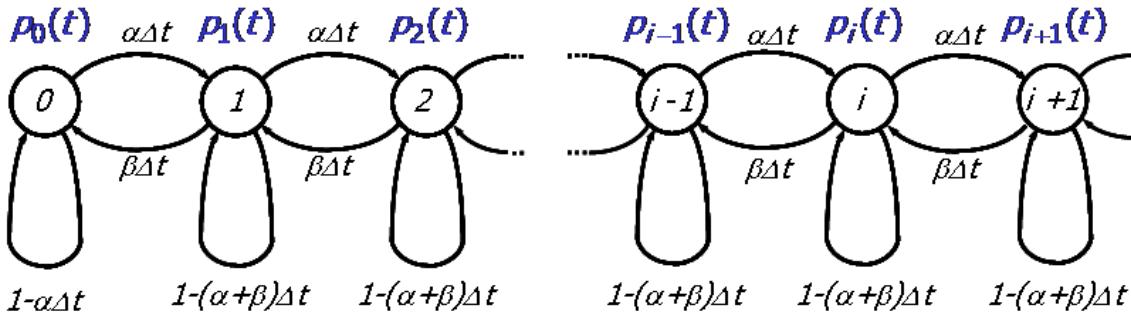
$$p(T < t, \beta) = 1 - e^{-\beta \Delta t} \approx \beta \Delta t \quad (7.44.)$$

Prepostavimo da je Δt tako mali da je to vjerojatnost upravo jednog odlaska.

Dakle, za vrlo mali interval Δt vjerojatnost: (pitamo!)

- jednog dolaska: $\alpha\Delta t$
- jednog odlaska: $\beta\Delta t$
- vjerojatnost jednog dolaska i jednog odlaska: $\alpha\Delta t \cdot \beta\Delta t = \alpha\beta\Delta t^2 \approx 0$ (zanemarujemo)
- vjerojatnost da nema ni dolaska ni odlaska: $(1 - \alpha\Delta t)(1 - \beta\Delta t) \approx 1 - (\alpha + \beta)\Delta t$
- vjerojatnosti višestrukih dolazaka i/ili odlazaka u Δt zanemarujemo

Markovljev lanac



Za trenutak $t + \Delta t$, za čvor i (stanje i) iz grafa slijedi:

$$p_i(t + \Delta t) = p_i(t) \cdot [1 - (\alpha + \beta)\Delta t] + p_{i-1}(t) \cdot \alpha \cdot \Delta t + p_{i+1}(t) \cdot \beta \cdot \Delta t \quad (7.45.)$$

Iznimka je stanje 0:

$$p_0(t + \Delta t) = (1 - \alpha\Delta t) \cdot p_0(t) + \beta \cdot \Delta t \cdot p_1(t) \quad (7.46.)$$

Sredimo prije puštanja Δt u 0:

$$\begin{aligned} \frac{p_i(t + \Delta t) - p_i(t)}{\Delta t} &= -(\alpha + \beta) \cdot p_i(t) + \alpha \cdot p_{i-1}(t) + \beta \cdot p_{i+1}(t) \\ \frac{p_0(t + \Delta t) - p_0(t)}{\Delta t} &= -\alpha \cdot p_0(t) + \beta \cdot p_1(t) \end{aligned} \quad (7.47.)$$

Kada $\Delta t \rightarrow 0$ tada je lijeve strane derivacija vjerojatnosti, ali obzirom da su vjerojatnosti konsantne to je 0! Slijedi:

$$\begin{aligned} 0 &= -(\alpha + \beta) \cdot p_i(t) + \alpha \cdot p_{i-1}(t) + \beta \cdot p_{i+1}(t) \\ 0 &= -\alpha \cdot p_0(t) + \beta \cdot p_1(t) \end{aligned} \quad (7.48.)$$

odnosno,

$$\begin{aligned} p_1(t) &= \frac{\alpha}{\beta} \cdot p_0(t) = \rho \cdot p_0(t) \\ p_{i+1}(t) &= (1 + \frac{\alpha}{\beta}) \cdot p_i(t) - \frac{\alpha}{\beta} \cdot p_{i-1}(t) = (1 + \rho) \cdot p_i(t) - \rho \cdot p_{i-1}(t) \end{aligned} \quad (7.49.)$$

Uvrštavanjem redom dobiva se konačna formula: (skraćeno: p_i umjesto $p_i(t)$)

$$\begin{aligned} p_1 &= \rho \cdot p_0 \\ p_2 &= (1 + \rho) \cdot p_1 - \rho \cdot p_0 = (1 + \rho) \cdot \rho \cdot p_0 - \rho \cdot p_0 = \rho^2 \cdot p_0 \\ p_3 &= (1 + \rho) \cdot p_2 - \rho \cdot p_1 = (1 + \rho) \cdot \rho^2 \cdot p_0 - \rho \cdot \rho \cdot p_0 = \rho^3 \cdot p_0 \\ &\vdots \\ p_i &= \rho^i \cdot p_0 \end{aligned} \quad (7.50.)$$

Da bi imali sve određeno treba nam samo p_0 .

Njega možemo izračunati iz zakonitosti da zbroj svih vjerojatnosti bude 1 (vjerojatnost da je sustav u nekom stanju je 1).

$$\sum_{i=0}^{\infty} p_i = \sum_{i=0}^{\infty} \rho^i \cdot p_0 = p_0 \sum_{i=0}^{\infty} \rho^i = p_0 \cdot \frac{1}{1-\rho} = 1 \Rightarrow p_0 = 1 - \rho \quad (7.51.)$$

te konačna formula za vjerojatnost da u sustavu ima i poslova:

$$\boxed{p_i = (1 - \rho)\rho^i} \quad \text{pišemo i: } \boxed{p(i = N) = (1 - \rho)\rho^N} \quad (7.52.)$$

Očekivanje slučajne varijable i je zapravo prosječan broj poslova \bar{n} :

$$\bar{n} = E(i) = \sum_{i=0}^{\infty} i \cdot p_i = \sum_{i=0}^{\infty} i \cdot (1 - \rho)\rho^i = (1 - \rho) \sum_{i=0}^{\infty} i \cdot \rho^i = (1 - \rho) \frac{\rho}{(1 - \rho)^2} = \frac{\rho}{1 - \rho} \quad (7.53.)$$

$$\boxed{\bar{n} = \frac{\rho}{1 - \rho} = \frac{\alpha}{\beta - \alpha}} \quad (7.54.)$$

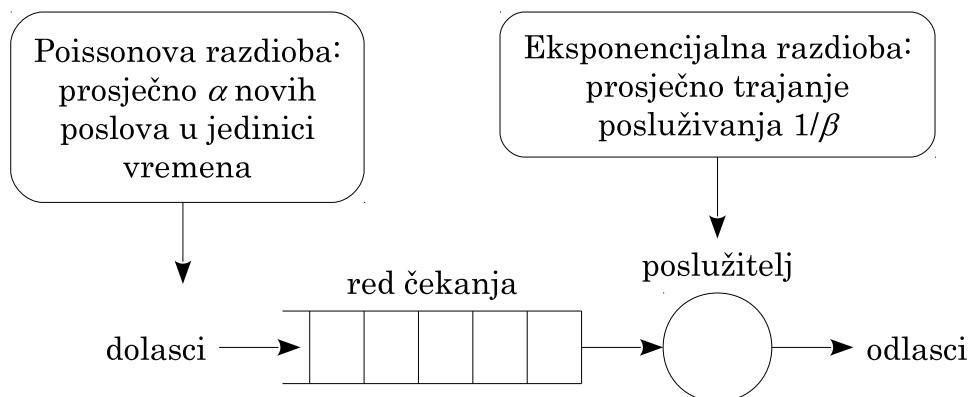
Te prema Littleovoju formuli slijedi i:

$$\boxed{\bar{T} = \frac{\bar{n}}{\alpha} = \frac{1}{\beta - \alpha}} \quad (7.55.)$$

7.5. Posluživanje s Poissonovom i eksponencijalnom razdiobom

[dodatno]

U literaturi se ovakav sustav obilježava sa: M/M/1, gdje prva oznaka obilježava dolaske, druga obradu, a treća broj poslužitelja. Obzirom da su Poissonova i eksponencijalna razdioba povezane i spadaju u tzv. klasu Markovljevih procesa dolaska označavaju se s M.



Slika 7.11. Model poslužitelja u nedeterminističkom sustavu

Poslovi dolaze u sustav s Poissonovom razdiobom:

- α – prosječan broj dolazaka novih poslova u sustav u jedinici vremena
- $\frac{1}{\alpha}$ – prosječno vrijeme između dva dolaska
- (vremena između dva dolaska podliježe eksponencijalnoj razdiobi s parametrom $\frac{1}{\alpha}$)

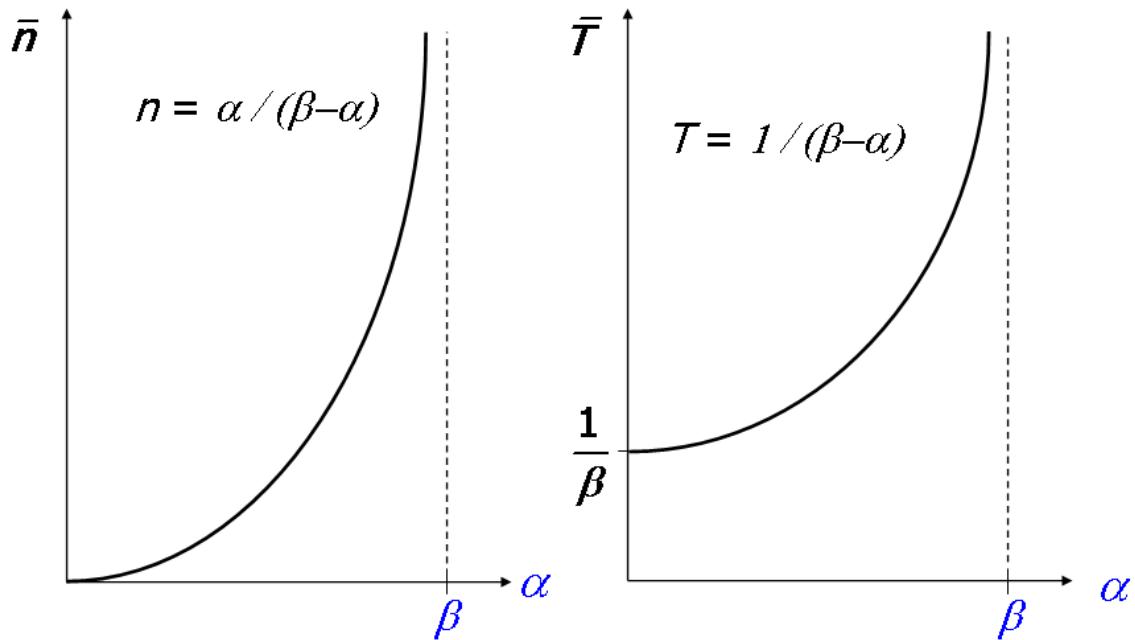
Vjerovatnost da u intervalu t dođe k novih poslova:

$$p(k(t), \lambda) = \frac{(\lambda t)^k}{k!} \cdot e^{-\lambda t} \quad (7.56.)$$

Trajanje obrade podliježe eksponencijalnoj razdiobi

- $\frac{1}{\beta}$ – prosječno trajanje obrade/posluživanja (za zadane poslove)
- β – prosječan broj poslova koje poslužitelj može obaviti u jedinici vremena
– sposobnost poslužitelja da obavi prosječno β poslova u jedinici vremena
- \bar{T} – prosječno trajanje zadržavanja posla u sustavu
- \bar{n} – prosječan broj poslova u sustavu
- $\bar{n} = \alpha \cdot \bar{T}$ – Littleova formula
- $\rho = \frac{\alpha}{\beta}$ – faktor iskorištenja, prosječno iskorištenje procesora/poslužitelja
- $\bar{n} = \frac{\rho}{1 - \rho} = \frac{\alpha}{\beta - \alpha} \implies \bar{T} = \frac{1}{\beta - \alpha}$
- $p(i = N) = (1 - \rho)\rho^N$ – vjerovatnost da u sustavu ima N poslova
- $p(i > N) = 1 - p(i \leq N) = \rho^{N+1}$ – vjerovatnost da u sustavu ima više od N poslova

Iz prethodnih se formula vidi da se ne smije dopustiti 100% opterećenje jer tada nazivnik ide u beskonačnost. Povećanjem opterećenja red raste i prosječno zadržavanje također.



β je konstantan za graf (ne mijenjamo poslužitelj, nego mu samo dajemo više posla).

Zadatak: Primjer 7.2.

Prepostavimo da proizvođač i potrošač komuniciraju preko ograničenog spremnika koji se sastoji od N pretinaca. U sustavu se tada može nalaziti najviše $M = N + 1$ poruka (N poruka nalazi se u redu, a jednu troši potrošač). Prepostavimo, nadalje, da proizvođač proizvodi poruke tako da su događaji njihovih stavljanja u spremnik podvrugnuti Poissonovoj razdiobi, te da potrošač na obradu poruka troši vremena podvrugnuta eksponencijalnoj razdiobi. Zanima nas koliko međuspremnik mora imati pretinaca da, uz dani faktor ρ , vjerojatnost blokiranja proizvođača bude manja od neke zadane vrijednosti.

Preformulirano: kolika je vjerojatnost da će u sustavu biti manje od M poruka (tada ne dolazi do blokiranja proizvođača), tj. $p(i < M) = p(i \leq N) = ?$

Prema formulama:

$$p(i \leq N) = \sum_{i=0}^N p_i = \sum_{i=0}^N (1 - \rho) \cdot \rho^i = (1 - \rho) \sum_{i=0}^N \rho^i = (1 - \rho) \frac{1 - \rho^{N+1}}{1 - \rho} = 1 - \rho^{N+1} \quad (7.57.)$$

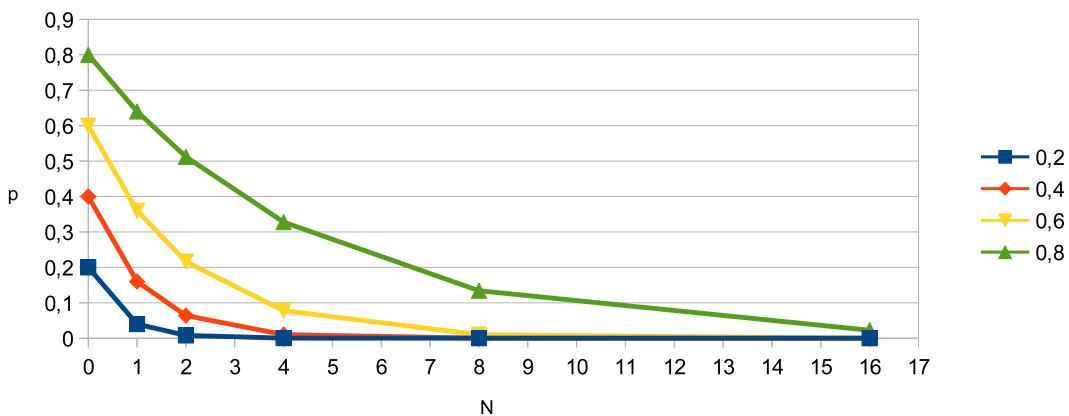
Suprotna vjerojatnost, vjerojatnost blokiranja je:

$$p(i > N) = \rho^{N+1} \quad (7.58.)$$

Brojke (vjerojatnost blokiranja):

| N | 0,2 | 0,4 | 0,6 | 0,8 |
|----|----------|----------|---------|---------|
| 0 | 0,20000 | 0,40000 | 0,60000 | 0,80000 |
| 1 | 0,04000 | 0,16000 | 0,36000 | 0,64000 |
| 2 | 0,00800 | 0,06400 | 0,21600 | 0,51200 |
| 4 | 0,00032 | 0,01024 | 0,07776 | 0,32768 |
| 8 | 5,12E-07 | 0,00026 | 0,01008 | 0,13422 |
| 16 | 1,31E-12 | 1,72E-07 | 0,00017 | 0,02252 |

Vjerojatnost blokiranja značajno raste s opterećenjem!



Zadatak: Za domaću zadaću

U nekom sustavu poslovi se javljaju periodički svakih 30 ms i to: P1 u 3. ms, P2 u 5., P3 u 6., P4 u 10., P5 u 20. i P6 u 23. milisekundi (glezano prema početku periode). Svi poslovi traju isto, po 4 ms. Odrediti prosječno zadržavanje poslova u sustavu i prosječan broj poslova u sustavu.

Zadatak: 7.1.

Zahtjevi za obradu podliježu Poissonovoj razdiobi s $\alpha = 2s^{-1}$, a vrijeme obrade ima eksponencijalnu razdiobu. Mjeranjem je ustanovljeno prosječno vrijeme zadržavanja posla u sustavu $\bar{T} = 0,5s$. Kolika je vjerojatnost da se u sustavu nađe više od 5 poslova?

Dodatno:

- Kolika je vjerojatnost da u sustavu bude bude između 2 i 4 (2, 3 ili 4) poslova?
- Što ako se poslužitelj ubrza za 30%?

Zadatak: 7.2.

Za neki Web sustav s jednim poslužiteljem prosječan broj zahtjeva u minuti je 100, dok je snaga poslužitelja znatno veća, on ih može obraditi 300 u minuti (prosječno). Koliki se najveći postotak poslužiteljskog vremena može rezervirati za druge usluge, a da klijenti i dalje ne čekaju više od dvije sekunde na svoje zahtjeve (prosječno)? (Prepostaviti da to neće utjecati na razdiobe. Npr. da će se vrijeme za te druge poslove dati u vrlo kratkim intervalima.)

Zadatak: 7.3.

U nekom je poslužiteljskom centru napravljena analiza rada poslužitelja. Ustanovljeno je da tri poslužitelja rade s prilično malim opterećenjem. Poslužitelj P1 prosječno dobiva 70 zahtjeva u minuti i njegova prosječna iskoristivost je 20 %, poslužitelj P2 dobiva 200 zahtjeva u minuti s prosječnim opterećenjem od 30 %, dok poslužitelj P3 s prosječno 150 zahtjeva u minuti radi tek s 10 % opterećenja. Poslužitelj P3 je procesorski najjači, 50% jači od P1 te 100% jači od P2. Izračunati kvalitetu usluge (prosječno vrijeme zadržavanja zahtjeva u sustavu) ukoliko bi se svi poslovi preselili na poslužitelj P3.

Zadatak: 7.4.

Za neki Web sustav s jednim poslužiteljem prosječan broj zahtjeva u sekundi je 100 (dolazak zahtjeva podliježe Poissonovoj razdiobi). Poslužitelj obrađuje tri tipa zahtjeva: Z_1 , Z_2 i Z_3 . Obrada zahtjeva podliježe eksponencijalnim razdiobama. Za zahtjeve tipa Z_1 obrada prosječno traje 5 ms, za Z_2 8 ms te za Z_3 10 ms. Ukoliko je postotak zahtjeva za Z_1 30%, za Z_2 40% te za Z_3 30% odrediti prosječnu kvalitetu usluga koje poslužitelj pruža, tj. odrediti prosječno vrijeme zadržavanja zahtjeva u sustavu te vjerojatnost da se u sustavu nalazi više od 10 zahtjeva.

Zadatak: Za vježbu

U nekom sustavu imamo dva poslužitelja P_1 i P_2 i dva tipa poslova Z_1 i Z_2 koje oni obrađuju (P_1 - Z_1 , P_2 - Z_2). Poslovi Z_1 prosječno dolaze sa 30 poslova u minuti, dok poslovi Z_2 dolaze sa 90 poslova u minuti. P_1 radi s 30% opterećenjem, a P_2 s 60%.

(ZAD) Kada bi zamjenili poslužitelje, tj. kada bi P_1 obrađivao poslove Z_2 (umjesto Z_1), opterećenje mu iznosi na 80%. Koje bi bilo opterećenje poslužitelja P_2 ako bi on obrađivao poslove Z_1 ?

Druge inačice zadatka:

(ZAD) Ako sve poslove obrađuje P1 opterećenje mu naraste na 60%. Koliko bi bilo opterećenje P2 ako on obrađuje sve poslove?

(ZAD) Ako sve poslove obrađuje P1 prosječno vrijeme zadržavanja poslova u sustavu naraste na 2 s. Koliko bi prosječno vrijeme zadržavanja poslova u sustavu ako sve sve poslove obrađuje P2?

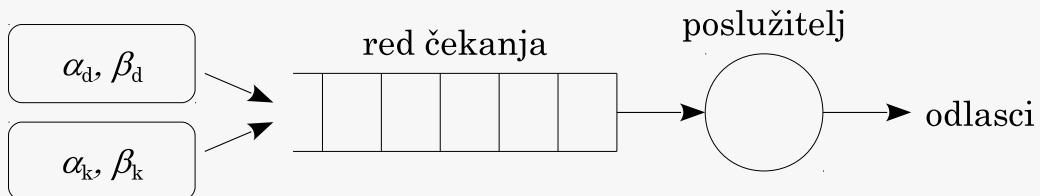
7.6. Osnovni načini dodjeljivanja procesora dretvama – raspoređivanje dretvi

7.6.1. Dodjeljivanje po redu prispijeća

Najjednostavnije. Ali ima problema. Analizirajmo (prvo) statistički.

Primjer 7.3. Mješanje kratkih i dugih poslova

Neka u sustav dolaze dvije skupine zadataka: kratki s indeksom k te dugi s indeksom d . Kakva je statistika za tu mješavinu? Kako to izgleda za pojedinačni zadatak?



Koliki je ukupni α ? Zbroj!

$$\alpha = \alpha_k + \alpha_d \quad (7.59.)$$

Što je s β ? Bolje razmišljati i $\frac{1}{\beta}$ (prosječnom trajanju posluživanja mješavine poslova).

$$\frac{1}{\beta} = \frac{\alpha_k}{\alpha_k + \alpha_d} \cdot \frac{1}{\beta_k} + \frac{\alpha_d}{\alpha_k + \alpha_d} \cdot \frac{1}{\beta_d} \quad (7.60.)$$

Ipak jednostavnije je razmišljati o ukupnom opterećenju koje je suma opterećenja koje generiraju kratki i dugi poslovi:

$$\rho = \rho_k + \rho_d \quad (7.61.)$$

te β računati prema:

$$\beta = \frac{\alpha}{\rho} \quad (7.62.)$$

Zadatak: 7.3.

Promotrimo dvije skupine poslova koje možemo nazvati kratkim i dugim poslovima. Svojstva tih dvaju skupina možemo prikazati tablično (dolasci Poisson, obrada eksp. razd.):

| | kratki | dugi | mješavina |
|-------------------|--------|------|-----------|
| α | 10 | 0,01 | 10,01 |
| β | 50 | 0,02 | 14,3 |
| $\frac{1}{\beta}$ | 0,02 | 50 | 0,6993 |
| ρ | 0,2 | 0,5 | 0,7 |
| \bar{n} | 0,25 | 1 | 2,23 |
| \bar{T} | 0,025 | 100 | 0,233 |

- Ovo je statistika za mješavinu.
- Kad ih pomiješamo i dalje će obrada kratkog trajati 0,02 a dugog 50 (prosječno!).
- Sama statistika je značajno pogoršanje za kratke poslove! Za red veličine (statistički).
- Najgori slučaj? Kratki posao koji dolazi neposredno iza dugog: on mora čekat da dugi završi, tj. mora čekati i više od 50 jedinica vremena!!!

Analizom prethodnih primjera ustanovljeno je da raspoređivanje po redu prispjeća nije dobro za kratke poslove.

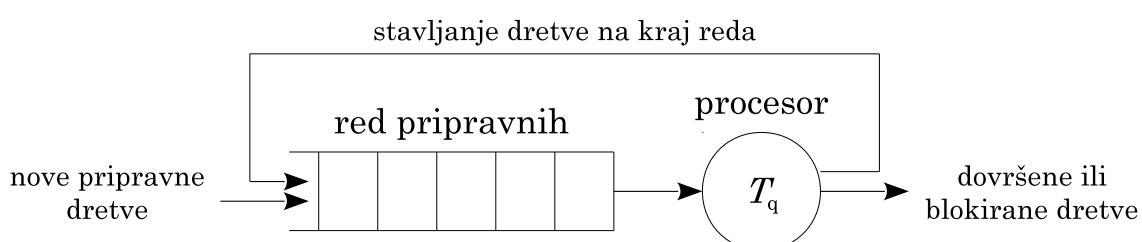
Kako onda raspoređivati?

Koristiti prioritete? Dati kratkim zadacima veći prioritet? To bi bilo nepravedno prema dugim poslovima koji bi se stalno odgađali.

Druge rješenje = podjela poslužiteljskog vremena = kružno posluživanje!

7.6.2. Kružno posluživanje (posluživanje podjelom vremena)

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



- 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.

7.6.3. Usporedba dodjeljivanja: po redu prispjeća i kružno

Za primjer, uzmimo iste podatke kao i za zadatak 7.3. ali sada neka to bude *deterministički sustav*, gdje su vremena absolutna a ne prosječna (kratki dolaze svakih 100 ms, dugi svakih 100 s, obrada kratkih traje 20 ms a dugih 50 s).

Primjer 7.4. Raspoređivanje po redu prispjeća

α i ρ su isti kao i prije (isto razmišljanje i dalje vrijedi!):

$$\alpha = \alpha_k + \alpha_d = 10 + 0,01 = 10,01 \quad (7.63.)$$

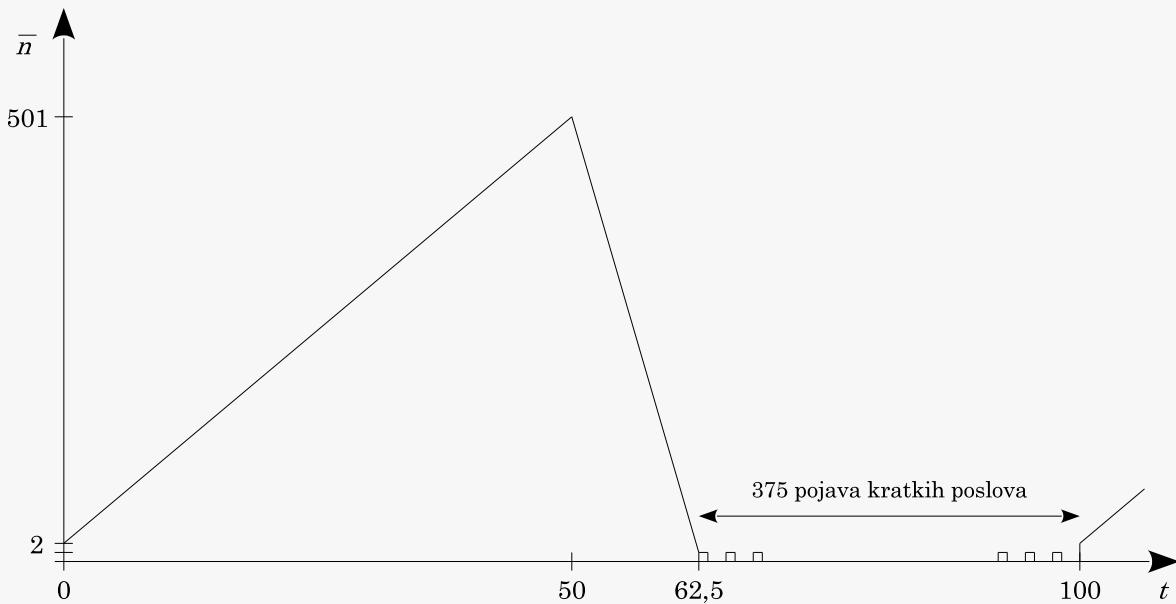
$$\rho = \rho_k + \rho_d = \frac{10}{50} + \frac{0,01}{0,02} = 0,7 \quad (7.64.)$$

Koliki su \bar{n} i \bar{T} (stvarni po zadacima te u prosjeku)?

Razmatrajmo pojedinačne zadatke.

Prepostavimo da je najprije došao dugi posao i odmah nakon njega kratki.

Obratno razmatranje gotovo da samo pomiče vremensku skalu i neznatno usporava taj dugi posao (20ms).



Slika 7.12. Broj poslova u redu poslužitelja

U $t = 0$ kreće dugi posao i prvi kratki ide u red (2 ukupno).

Obrada dugog traje 50 s te za to vrijeme dolaze i kratki poslovi (10 u sekundi) i gomilaju se u redu.

U 50. sekundi kada dugi posao završava u sustavu se nagomila 500 kratkih poslova (u redu).

Nakon toga broj poslova pada: poslužitelj obrađuje $1/0,02=50$ poslova u sekundi, ali poslovi i dalje dolaze sa 10 u sekundi = broj poslova pada s 40 u sekundi.

$$\frac{500}{40} = 12,5 \Rightarrow \text{za } 12,5 \text{ sekundi red se isprazni (u } 50 + 12,5 = 62,5 \text{ s).}$$

Nakon toga red je prazan i svaki posao koji dođe dolazi izravno do poslužitelja.

Prosječan broj poslova u sustavu \bar{n} može se približno izračunati prema:

$$\bar{n} = \frac{\text{površina}}{\text{vrijeme}} \approx \frac{\frac{62,5 \cdot 500}{2} + 375 \cdot 0,02 \cdot 1}{100} = 156,325 \quad (7.65.)$$

Opaska: računamo kao da je kontinuirano iako je diskretno!

Prema Littleovu pravilu:

$$\bar{T} = \frac{\bar{n}}{\alpha} \approx \frac{156,325}{10,01} = 15,62 \text{ s} \quad (7.66.)$$

Navedeno je statistički. Što možemo reći o zasebnim poslovima?

Skicirati graf za T_k (izgled: $| \backslash _ _ _ |$).

Prvi kratki se u sustavu zadržava 50 s !!!

Primjer 7.5. Kružno dodjeljivanje

Neka je kvant vremena 0,01 s (10 ms).

Kratki poslovi trebaju $\frac{0,02}{0,01} = 2$ kvanta, dugi $\frac{50}{0,01} = 5000$ kvantova

Neka dugi opet dođe prvi.

Redoslijed izvođenja: D K D K D D D ... D K D K D D D ...

Dugi se zadržava: $50 \cdot \frac{10}{8} = 62,5 \text{ s}$

Kratki:

- prvi: $4 \cdot 0,01$
- dalnjih 624: $3 \cdot 0,01$
- zadnjih 375: $2 \cdot 0,01$

$$\bar{T} = 62,5 \cdot \frac{1}{1001} + 0,04 \cdot \frac{1}{1001} + 0,03 \cdot \frac{624}{1001} + 0,02 \cdot \frac{375}{1001} = 0,0887 \text{ s} \quad (7.67.)$$

$$\bar{n} = \alpha \cdot \bar{T} = 10,01 \cdot 0,0887 = 0,887 \quad (7.68.)$$

Iz primjera je očito je RR bolje raspoređivanje.

Ali: treba prekidati obrade (satni mehanizam) + troši se vrijeme na zamjenu dretvi.

Ako su zamjene rijetke onda se vrijeme zamjena dretvi (zamjena konteksta) može zanemariti, ali ako su česte one mogu značajno utjecati na učinkovitost sustava!

7.7. Raspoređivanje dretvi u operacijskim sustavima

Osnovne (teorijske) strategije raspoređivanja:

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

U operacijskim sustavima se kombiniraju gornje strategije, prema zahtjevima sustava.

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

- osnovni kriterij – raspoređivanje prema prioritetu
- ako osnovni kriterij ne daje samo jednu dretvu (imamo više dretvi ista najveća prioriteta) tada se za odabir jedne dretve koristi 2. kriterij:
 - prema redu prispjeća – FIFO
 - podjelom vremena – RR

“Normalne” dretve:

- raspoređivanje podjelom vremena
 - pravedna podjela procesorskog vremena
 - prioritet određuje udio vremena koji će dretva dobiti (uglavnom)

Linux (info)

- 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)
- raspoređivanje “običnih” dretvi:
 - strategije: *SCHED_OTHER* i slični (*_IDLE*, *_BATCH*),
 - prioriteti od 100 do 139 (nice od -20 do 19/20)
 - “pravedno” podijeliti procesorsko vrijeme
 - dretve većeg prioriteta dobivaju više procesorskog vremena
 - * oko 10-15% po prioritetu

Windows (info)

- raspoređivanje dretvi za rad u stvarnom vremenu (engl. *Real Time*):
 - strategija: kružno uz *REALTIME_PRIORITY_CLASS* (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
 - iznimke:
 - * radi rješavanja problema izgladnjivanja (procesorskog)
 - * 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)
- prioritet se formira na osnovu:
 - prioritetne klase procesa (IDLE, BelowNormal, N, AboveN, High, RT)
 - prioritetne razine dretve (IDLE, Lowest, BN, N, AN, Highest, RT)

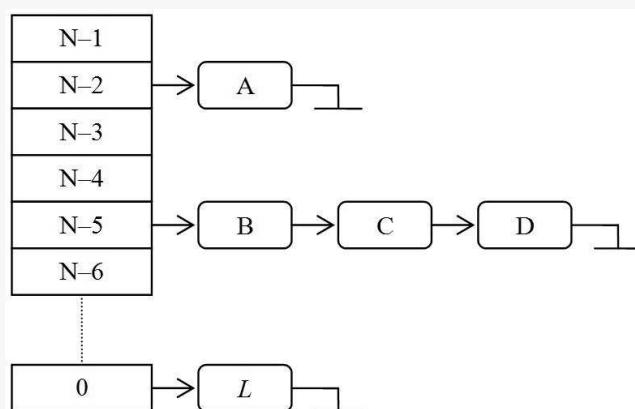
Zadatak: 7.5. Raspoređivanje samo prema jednom kriteriju

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

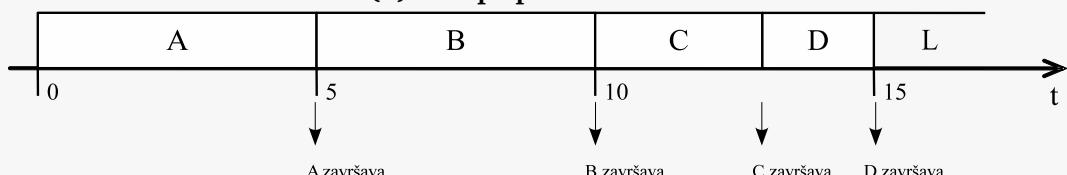
- raspoređivanje po redu prispjeća
- raspoređivanje prema prioritetu uz $p_A > p_B > p_C > p_D$
- kružno raspoređivanje sa $t_q = 1$ (jedinica vremena)

Primjer 7.6. Raspoređivanja “u stvarnim sustavima” (info)

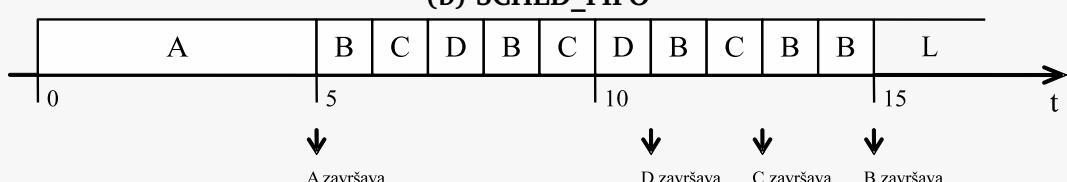
Prikazati rad poslužitelja tijekom posluživanja zadanih dretvi korištenjem raspoređivanja prema SCHED_FIFO, SCHED_RR te SCHED_OTHER, ako je stanje nekog sustava prikazano pripravnim dretvama A, B, C i D različita prioriteta prema slici:



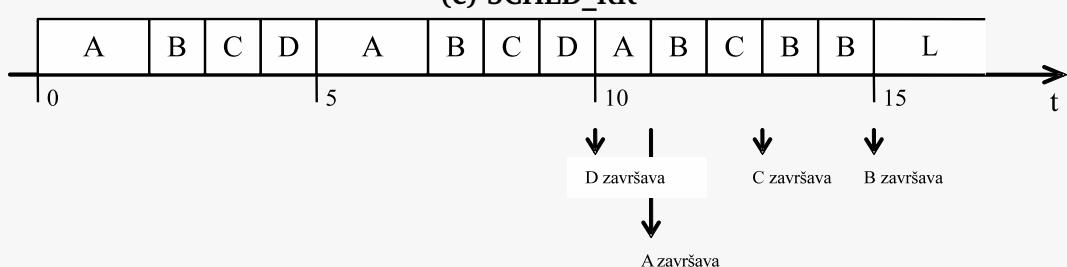
(a) Red pripravnih dretvi



(b) SCHED_FIFO



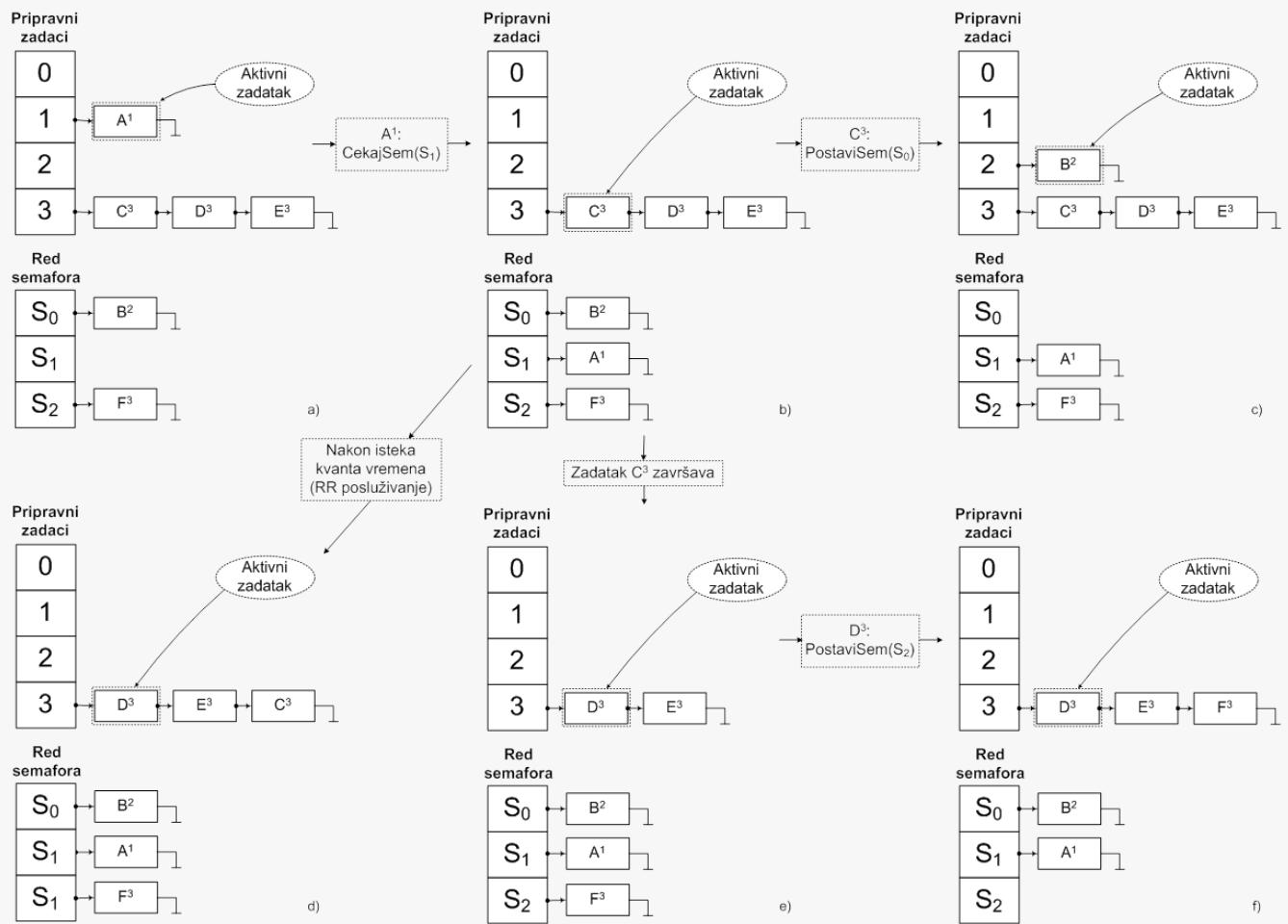
(c) SCHED_RR



(d) SCHED_OTHER

Slika 7.13. Rad raznih raspoređivača

Primjer 7.7. Primjeri odluka raspoređivanja (info)



Slika 7.14. Primjeri raspoređivanja

Pitanja za vježbu 7

1. Opisati značenja veličina: α , $1/\alpha$, β , $1/\beta$, \bar{T} , \bar{n} i ρ u kontekstu sustava s jednim poslužiteljem:
 - a) kada se radi o determinističkom sustavu
 - b) kada se radi o nedeterminističkom sustavu s Poissonovom razdiobom dolazaka i eksponentijalnom razdiobom trajanja obrada.
Kojim su formulama povezane zadane veličine?
2. Napisati i pojasniti Littleovo pravilo.
3. Navesti formulu za izračun vjerojatnosti pojave k događaja u vremenskom intervalu t , ako se koristi Poissonova razdioba za modeliranje dolazaka uz parametar α (prosječan broj dolazaka novih poslova u jedinici vremena).
4. Nacrtati Markovljev lanac koji prikazuje moguća stanja sustava u odnosu na trenutni broj poslova u sustavu ako se promatra jako mali interval Δt u kojem se mogu dogoditi promjene (doći novi posao ili neki posao napustiti sustav). Označiti vjerojatnosti prijelaza iz jednog stanja u drugo. Korištenjem grafa izraziti vjerojatnost da sustav u trenutku $t + \Delta t$ bude u stanju i (tj. $p_i(t + \Delta t)$).
5. Koja formula povezuje \bar{T} sa α i β ako se koriste Poissonova i eksponentijalna razdioba za modeliranje dolazaka i trajanja obrade?
6. Zašto kod nedeterminističkog sustava nije dozvoljeno opteretiti poslužitelj sa 100% opterećenja?
7. Ako je zadano prosječno opterećenje poslužitelja sa ρ , izračunati vjerojatnost (navesti formule) da u nekom trenutku u sustavu ima:
 - a) N poslova
 - b) više od N poslova.
8. 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.
9. Opisati principe raspoređivanja dretvi koji se koriste u današnjim operacijskim sustavima (Windows, Linux, SCHED_FIFO, SCHED_RR, SCHED_OTHER).

8. Upravljanje spremničkim prostorom

8.1. Uvod

Kako i kada procesor pristupa spremniku?

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

[dodatakno]

Veličina spremnika (RAM) za pojedine namjene (okvirno)

- poslužitelj: 8 GB +
- osobno računalo: 2 GB +
- pametni telefoni i tableti: 512 MB +
- pametni televizori ?
- obični televizori, obični telefoni, ... ?

Adresiranje spremničke lokacije – širina sabirnice i moguće veličine spremnika:

- 16 bita $\Rightarrow 64 \text{ KB} = 65536 \text{ B}$
- 32 bita $\Rightarrow 4 \text{ GB} \approx 4 \cdot 10^9 \text{ B}$
- 64 bita $\Rightarrow 16 \text{ EB} (\text{exa-}) \approx 2 \cdot 10^{19} \text{ B}$

Što sve ide u spremnik?

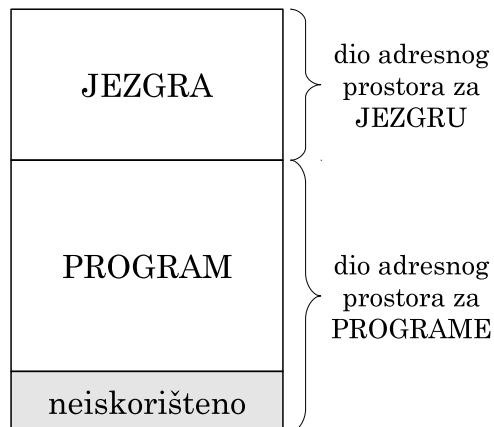
- jezgra:
 - jezgrine funkcije
 - strukture podataka
- programi:
 - instrukcije
 - podaci
 - gomila (heap, za malloc/free)
 - stog (za svaku dretvu zaseban)

8.1.1. Osnovne ideje upravljanja spremnikom

Koje su operacije OS-a potrebne i kako ih ostvariti?

- sve “potrebno” moći učitati u spremnik
- da sve “potrebno” stane u spremnik
- da nitko nikome ne smeta (proces procesu i jezgri)
 - da se zaštiti jezgra od procesa te proces jedan od drugoga, i to od grešaka i zlonamjernih napada
- efikasno koristiti glavni spremnik i ostale (pomoćne) spremnike, po potrebi
- dati preporuke za korištenje spremnika za programere

8.1.1.1. Najjednostavniji način: samo OS i jedan program



Slika 8.1. Upravljanje spremnikom u sustavima s jednom programom

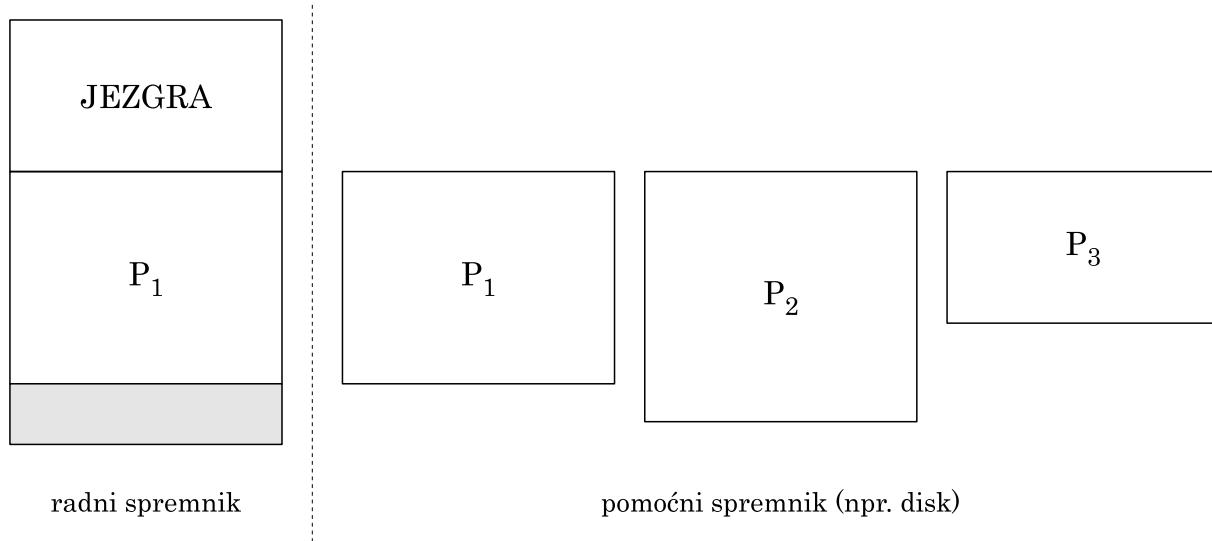
- samo za vrlo jednostavne sisteme je ovo dovoljno
 - međutim, takvih sistema ima puno – dio ugrađenih sustava je ovakvo

8.1.1.2. Više programa – svi u radnom spremniku

- “idealno”, ali treba puno spremnika
- koristi se tamo gdje su programi mali i svi stanu u spremnik (ugrađeni sistemi)

8.1.1.3. Više programa – jedan u radnom spremniku, ostali na pomoćnom

- “velika” zamjena konteksta
- koristiti pomoćni spremnik



Slika 8.2. Korištenje pomoćnog spremnika za “veliku zamjenu konteksta”

Svojstava pomoćnog spremnika – diska

- disk detaljnije u 9. poglavlju, sada samo osnovna svojstva
- vremenska svojstva diska:
 - vrijeme pristupa koda HDD-a (čitanje jednog bloka/sektora): red veličine $\approx 10 \text{ ms}$!
 - vrijeme pristupa koda SSD-a (čitanje jednog bloka/sektora): red veličine $\approx 0,1 \text{ ms}$!

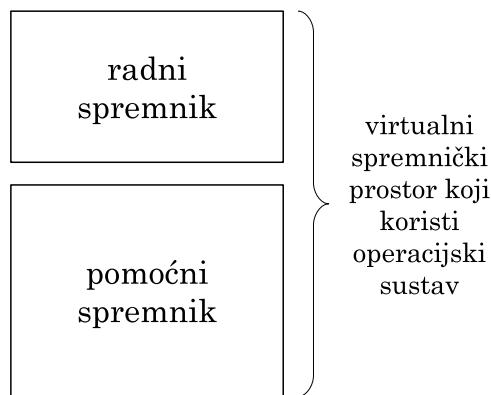
- čitanje procesa s diska od ≈ 1 ms do 100 ms (i više)!
- spremniku se pristupa znatno brže \approx red veličine ns $\Rightarrow \approx 10^5$ brže!
- disk je prespor da bi navedeni gornji način bio zadovoljavajući i učinkovit

Učinkovito korištenje pomoćnog spremnika za upravljanje spremnikom

- u spremniku trebaju 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
- zamjenu procesa obaviti DMA sklopovima (za minimalni gubitak vremena)
- za vrijeme obavljanja zamjene procesor će izvoditi neki drugi proces koji je u radnom spremniku!

Virtualni spremnički prostor

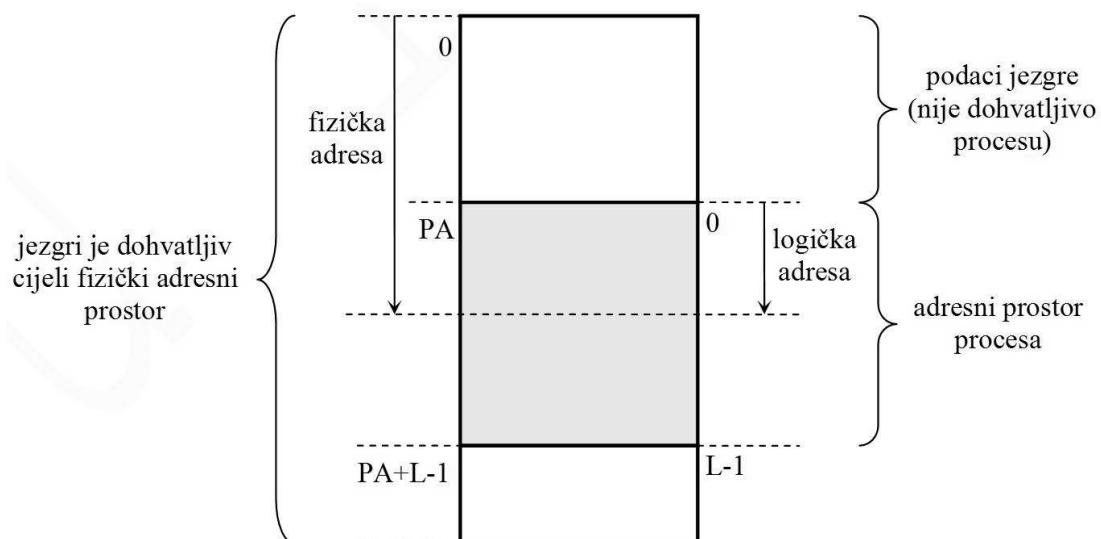
U sustavima koji koriste pomoćni spremnik za upravljanje spremničkim prostorom koristimo pojam virtualni spremnički prostor OS-a (engl. *virtual memory*) koji se sastoji od glavnog spremnika i pomoćnog spremnika.



Slika 8.3. Virtualni spremnički prostor OS

Fizičke i logičke adrese

- apsolutne adrese = fizičke adrese
- relativne adrese = logičke adrese



Slika 8.4. Fizička i logička adresa

Primjer 8.1. Fizička i logička adresa

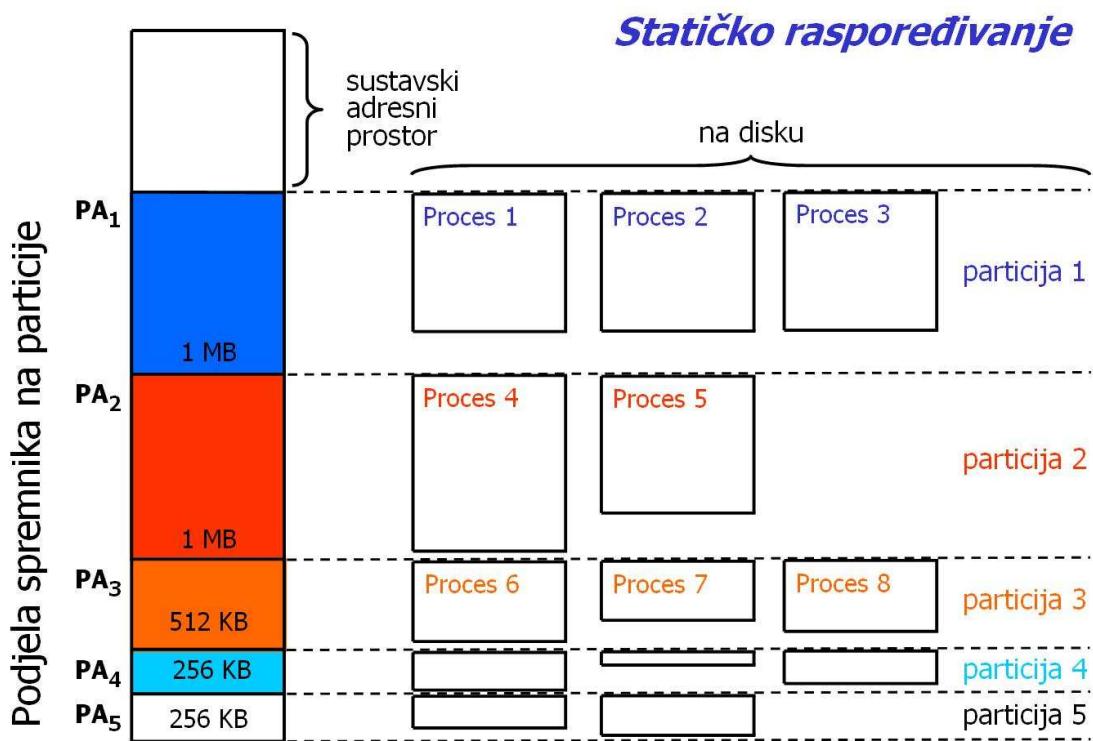
Odnos fizičke i logičke adrese programa ovisi gdje se on nalazi.

| program na disku (prije pokretanja) | proces (fizičke adrese) učitan na PA=1000 | proces (logičke adrese) |
|--|--|-------------------------|
| 0 (početak) | 1000 (početak) | 0 (početak) |
| . | . | . |
| 20 LDR R0, (100) | 1020 LDR R0, (1100) | 20 LDR R0, (100) |
| 24 LDR R1, (104) | 1024 LDR R1, (1104) | 24 LDR R1, (104) |
| 28 ADD R2, R0, R1 | 1028 ADD R2, R0, R1 | 28 ADD R2, R0, R1 |
| 32 STR R2, (120) | 1032 STR R2, (1120) | 32 STR R2, (120) |
| 34 B 80 | 1034 B 1080 | 34 B 80 |
| . | . | . |
| . | . | . |
| 80 CMP R0, R3 | 1080 CMP R0, R3 | 80 CMP R0, R3 |
| . | . | . |
| . | . | . |
| 100 DD 5 | 1100 DD 5 | 100 DD 5 |
| 104 DD 7 | 1104 DD 7 | 104 DD 7 |
| . | . | . |
| 120 DD 0 | 1120 DD 0 | 120 DD 0 |
| | 1500 (vrh stoga) | 500 (vrh stoga) |

Načini upravljanja spremnikom koji koriste pomoći spremnik

- statičko upravljanje: statički podijeliti spremnik na particije
- dinamičko upravljanje: dinamički dijeliti spremnik prema potrebama
- straničenje

8.2. Statičko upravljanje spremnikom



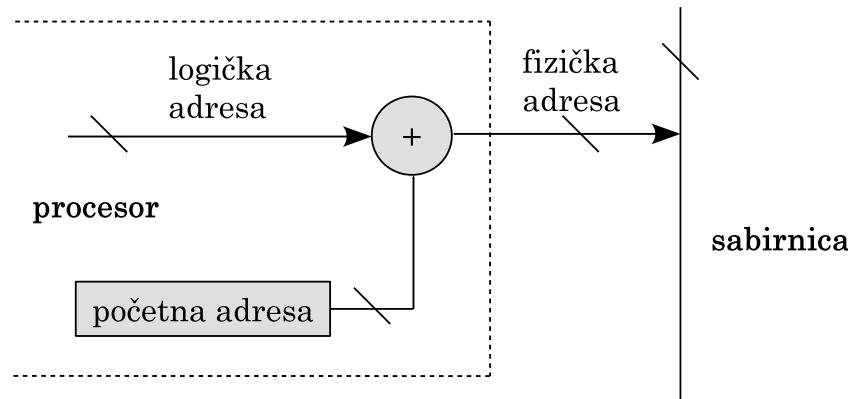
- dio spremnika namjenjenog za programe se u početku podijeli na *particije* unaprijed zadanih veličina.
- prilikom pokretanja programa on se najprije priprema za određenu particiju na pomoćnom spremniku
 - priprema se u absolutnim adresama (za tu particiju)!
 - odabir particije ide na osnovu potrebne veličine procesa i trenutnog “zauzeća” svih particija (kada postoji mogućnost odabira)
- OS odabire koje će programe učitati u njihove particije (samo po jedan program u pojedinu particiju)
- kada se program blokira, OS pokreće postupak njegove zamjene, a u međuvremenu (dok se zamjena ne obavi pod utjecajem DMA sklopova) OS odabire neku dretvu iz procesa koji se nalaze u radnom spremniku (u nekoj od drugih particija)

Svojstva statičkog upravljanja

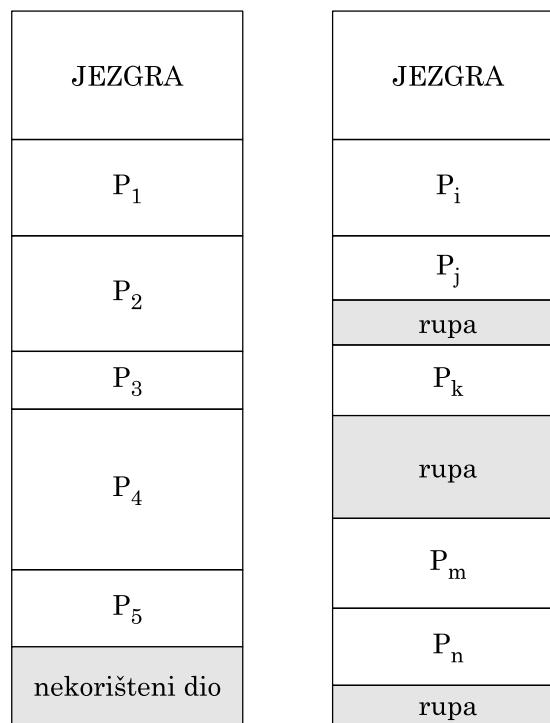
- + jednostavan model (za jednostavne primjene)
- unutarnja fragmentacija (dio particije koji proces ne koristi ne može se iskoristiti od strane drugih procesa)
- vanjska fragmentacija (ukoliko svi procesi pripremljeni za jednu particiju su u nekom trenutku blokirani, tu particiju se ne može iskoristiti za procese koji su pripremljeni za druge particije – program se može izvoditi samo u particiji za koju je pripremljen)
- nema zaštite
 - greška u jednom procesu može uzrokovati grešku u drugom ili i u jezgri
 - npr. zbog krive vrijednosti indeksa izlazi se iz okvira procesa i mijenja se podatak drugog procesa (što se može manifestirati kao greška u oba procesa)
- veliki programi se ne mogu pokretati
- (manji nedostatak) prilikom pokretanja program treba prvo na pomoćnom spremniku pripremiti u fizičkim adresama za odabranu particiju (zbrajanjem logičke adrese s adresom particije za koju se priprema)

8.3. Dinamičko upravljanje spremnikom

- program se priprema za logičke adrese (ostaje u logičkim adresama i u spremniku)
- koristi se sklopovska pretvorba logičke adrese u fizičku, i to u trenutku kad adresa izlazi iz procesora
- dovoljno je zbrajalo koje zbraja logičku adresu (LA) generiranu u procesoru sa adresom početka segmenta gdje je program smješten (taj registar postaje dio konteksta dretve).



Slika 8.5. Sklop za pretvorbu logičke adrese u fizičku



Slika 8.6. Primjer stanja na početku rada te kasnije

Problem fragmentacije:

- fragmentacije se ne možemo riješiti, možemo ju samo kontrolirati:
 - pri dodjeli uzeti najmanju rupu koja je dovoljno velika za zahtjev
 - pri oslobođanju bloka, nastalu rupu spojiti sa susjednim rupama
 - ako treba i zaustaviti sustav i napraviti "defragmentaciju"

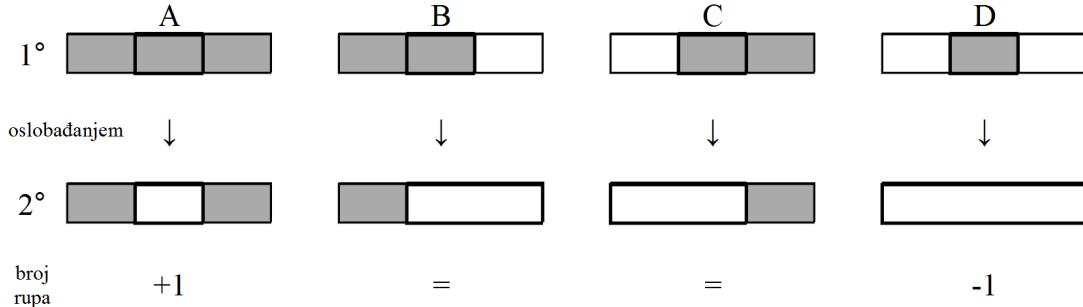
Knuthovo pravilo:

- u stabilnom sustavu broj rupa jednak je 50% broja punih blokova, tj. 1/3 svih blokova su rupe!

Knuthovo pravilo – dokaz

Sustav se nalazi u stohastičkoj ravnoteži – vjerojatnost zahtjeva za memorijom p_Z jednaka je vjerojatnosti zahtjeva za oslobođanje p_O . tj. $p_Z = p_O$. (Pokušaj obrazloženja: ako to ne bi bilo tako došlo bi do gomilanja zahtjeva, tj. novih programa i sustav bi bio neodrživ)

U sustavu imamo 4 tipa punih blokova (puni blokovi su zasjenjeni):



Iz 1° se može izračunati broj punih blokova i broj rupa (a broj punih blokova tipa A, b ...):

- broj punih blokova m i broj rupa n :

$$\begin{aligned} m &= a + b + c + d \\ n &= \frac{b + c + 2d}{2} \end{aligned} \tag{8.1.}$$

Svaka rupa koju blok B vidi kao desnu mora biti ili lijeva od C ili D. Zato se broj rupa dijeli sa dva. Isto tako, ako zanemarimo rubne slučajeve, nakon jedne rupe koju B vidi (desno) mora se pojavitи blok tipa C koji tu rupu vidi s lijeve strane, ili blok D – ali onda se konačno nakon D mora pojavitи blok C koji njegovu desnu rupu vidi kao lijevu!

Tj. vrijedi:

$$\begin{aligned} b &= c \implies m = a + 2b + d \\ n &= \frac{b + b + 2d}{2} = b + d \end{aligned} \tag{8.2.}$$

Vjerojatnost povećanja broja rupa možemo izračunati (prema gornjoj slici) kao vjerojatnost oslobođanja uz uvjet da oslobođeni blok bude tipa A, tj.:

$$p_+ = p_O \cdot \frac{a}{m} \tag{8.3.}$$

Vjerojatnost smanjenja broja rupa možemo izračunati (prema gornjoj slici) kao vjerojatnost oslobođanja uz uvjet da oslobođeni blok bude tipa D, tj.:

$$p_- = p_O \cdot \frac{d}{m} \tag{8.4.}$$

(zanemarujuemo mogućnost zahtjeva koji bi potpuno popunio neku rupu)

U stanju stohastičke ravnoteže te dvije vjerojatnosti moraju biti jednake:

$$p_+ = p_- \implies p_O \cdot \frac{a}{m} = p_O \cdot \frac{d}{m} \implies a = d \implies \begin{aligned} m &= 2(a + b) \\ n &= a + b \end{aligned} \implies n = \frac{m}{2} \tag{8.5.}$$

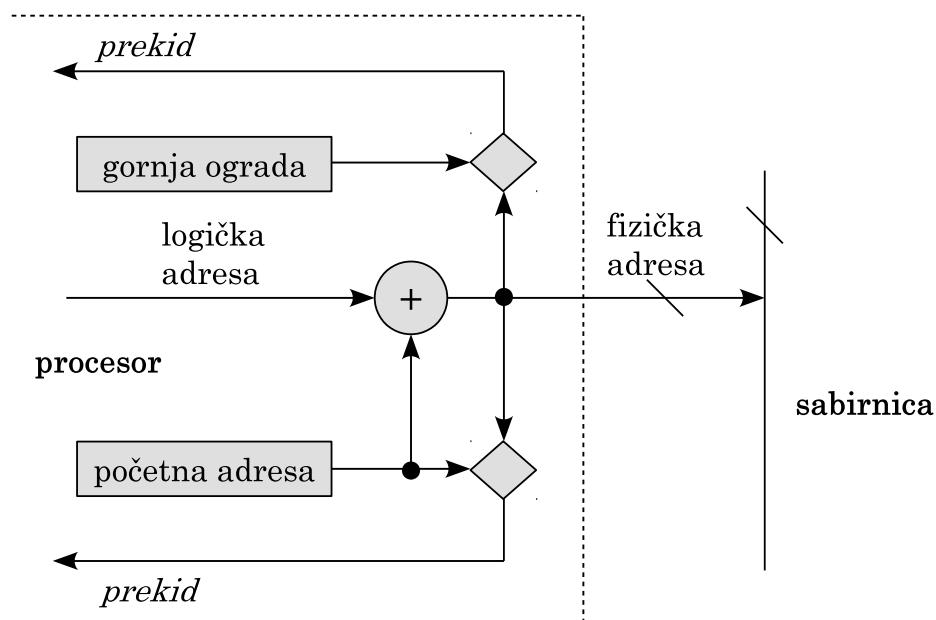
Knuthovo 50% pravilo: $n = \frac{m}{2}$ tj. trećina svih blokova su slobodni (rupe)

Knuthovo pravilo pokazuje problem rupa odnosno rascjepkanog slobodnog prostora koji se javlja u mnogim slučajevima.

Sustavi "zahvaćeni" Knutovim pravilom (info)

- dinamičko upravljanje spremnikom (na više razina)
 - dodjela prostora za procese (na opisani način "dinamičkog upravljanja spremnikom")
 - dodjela prostora za potrebe jezgre (opisnici, međuspremnići i sl.)
 - upravljanje prostorom i kod drugih načina upravljanja (upravljanje okvirima)
 - gomila (heap) – dodjela prostora unutar procesa (malloc/free)
- upravljanje datotečnim sustavom (o njemu više kasnije)
 - zauzeti i slobodni dijelovi diska
 - zauzeti i slobodni elementi tablica raznih opisnika

Zaštita spremničkog prostora



- uz zbrajalo mogu se dodati dva komparatora koja će spriječiti da program izade iz svog dodijelenog segmenta (slika): donja ograda = PA, gornja ograda
 - dobiva se sklopovska zaštita spremnika (memory protection unit – MPU)
 - koristi se u jednostavnijim procesorima jer je jednostavno za ostvariti

Svojstva dinamičkog upravljanja spremnikom

- + program ostaje u logičkim adresama
- + zaštita spremnika (uz MPU)
- treba (jednostavna) sklopovska potpora (kod statičkog nije potrebna!)
- fragmentacija
- ne mogu se pokretati programi koji ne stanu u radni spremnik!

U odnosu na statičko upravljanje:

- + fleksibilnije, nije potrebno unaprijed podijeliti spremnik
- + prosječno veća iskoristivost spremnika
- + nema vanjske fragmentacije, ali ima fragmentacije
- + program se može izvoditi s bilo kojeg dijela spremnika; jednom se može učitati na jednu lokaciju, a potom (nakon što je izbačen iz spremnika) na drugu itd.
- (info) u početku je potrebno znati potrebe procesa za spremnikom i toliko se prostora zauzme – kasnije je teško povećavati ili smanjivati adresni prostor (vrijedi i za statičko i dinamičko upravljanje spremnikom podjednako)

Statičko i dinamičko upravljanje u praksi? (info)

- u praksi se statičko/dinamičko uglavnom koristi kod vrlo jednostavnih sustava (ugrađenih) koji ne trebaju/koriste pomoćni spremnik
- imaju suviše nedostataka da se koriste u složenijim sustavima (obzirom da postoji bolji način – straničenje)
- u okviru ovog poglavlja prikazani su uz korištenje pomoćnog spremnika uglavnom zato da se vide njihovi nedostaci kao metoda upravljanja uza opće sustave

Kako riješiti problem velikih programa?

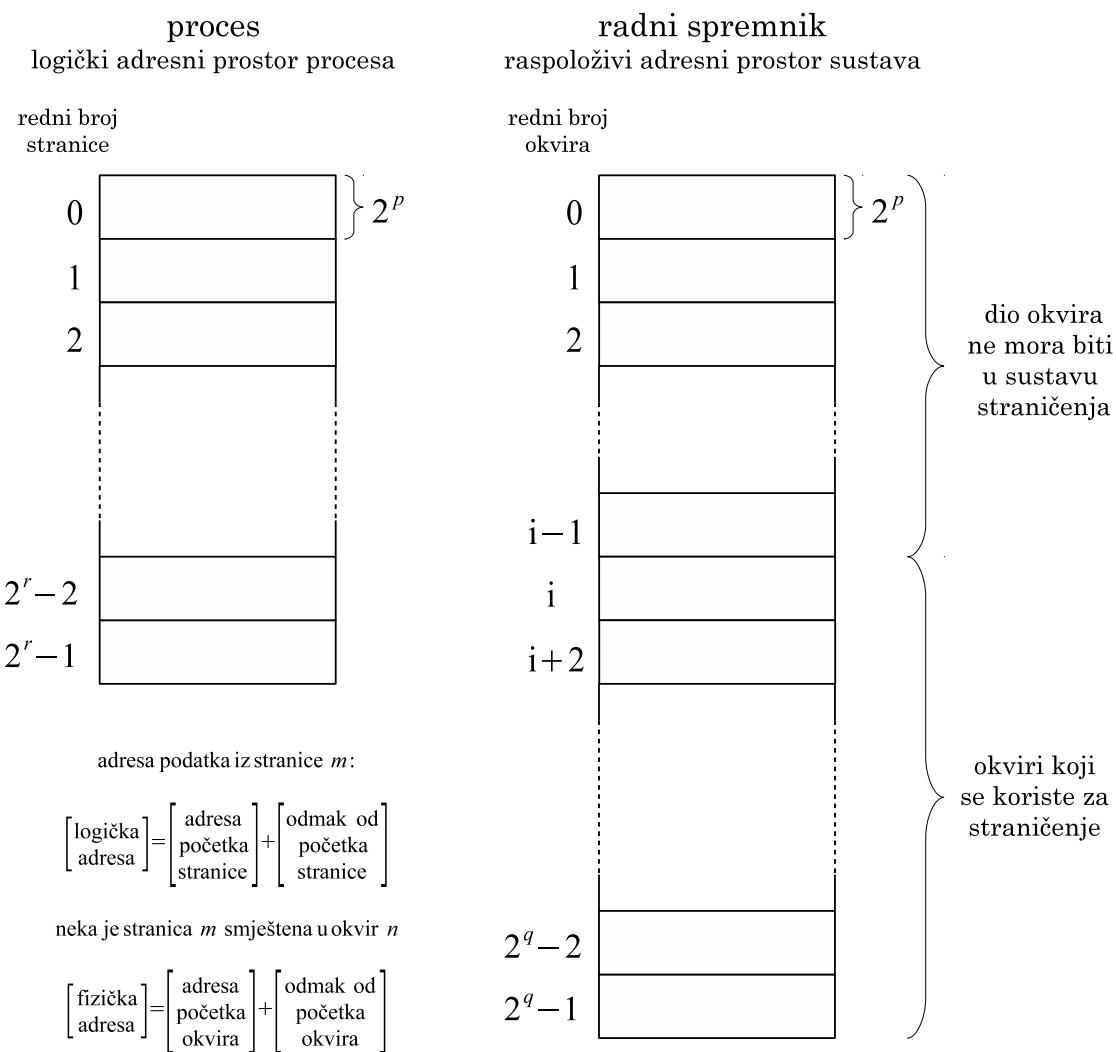
- osnovna ideja: učitavati samo one dijelove programa koji su trenutno potrebni, a ostale ostaviti na pomoćnom spremniku; ali kako?
 - Primjer – preklopni rad
 - * pod utjecajem programa zatražiti zamjenu djelova procesa (jedan dio van iz spremnika na pomoćni spremnik, a drugi u spremnik)
 - * problem: presloženo, zahtijeva od programera da tako izgradi program i predviđa potrebe kada što izbaciti i ubaciti ...
- dodatna ideja: podijeliti spremnik i proces na male ali jednake dijelove i dodijeliti neke djelove spremnika djelovima procesa; pretvorbu adresa raditi sklopovljem \Rightarrow straničenje

8.4. Straničenje

8.4.1. 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, npr. 4 KB)
- pretvorba adresa obavlja se dodatnim *sklopom* uz odgovarajuću *strukturu podataka* koju održavaju jezgrine funkcije
- u radnom spremniku nalaze se samo trenutno potrebne stranice procesa
- u pomoćnom spremniku nalaze se sve stranice procesa
- stranice se učitavaju prema potrebi



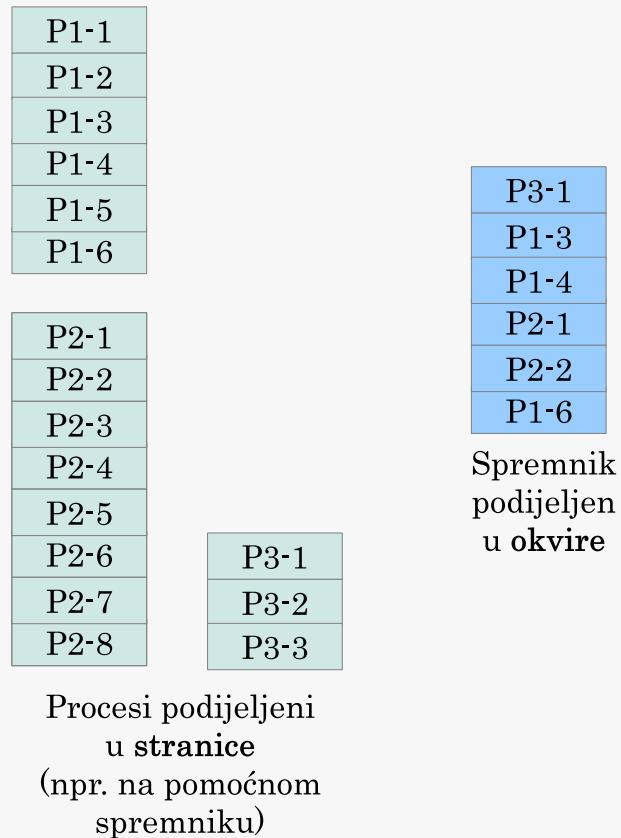
Slika 8.7. Podjela na stranice i okvire

Tablica prevođenja – opis gdje se koja stranica procesa nalazi

- zapisana u opisniku procesa
- za svaki proces zasebna tablica
- izgrađuje ju i održava OS, koristi sklop (i ažurira zastavice)
- za svaku stranicu postoji jedan opisnik – jedan redak u tablici prevođenja

- svaki opisnik 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)

Primjer 8.2. Primjer straničenja s tri procesa



Slika 8.8. Logički adresni prostori procesa te radni spremnik

Tablica 8.1. Tablice prevodenja

Za proces P1

| | |
|---|---|
| | 0 |
| | 0 |
| 2 | 1 |
| 3 | 1 |
| | 0 |
| 6 | 1 |

Za proces P2

| | |
|---|---|
| 4 | 1 |
| 5 | 1 |
| | 0 |
| | 0 |
| | 0 |
| | 0 |
| | 0 |
| | 0 |

Za proces P3

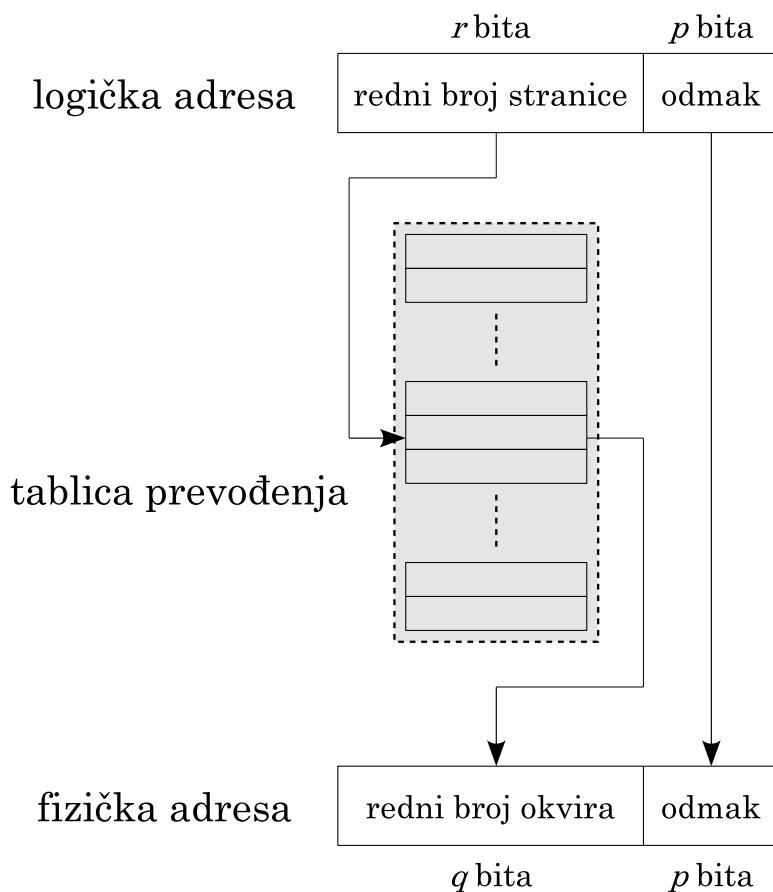
| | |
|---|---|
| 1 | 1 |
| | 0 |
| | 0 |

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

Pri pretvorbi logička \Rightarrow fizička:

- *odmak* se prekopira
- tablica prevodenja se koristi za pretvorbu *rednog broja stranice* u *redni broj okvira* (ako je stranica u radnom spremniku)

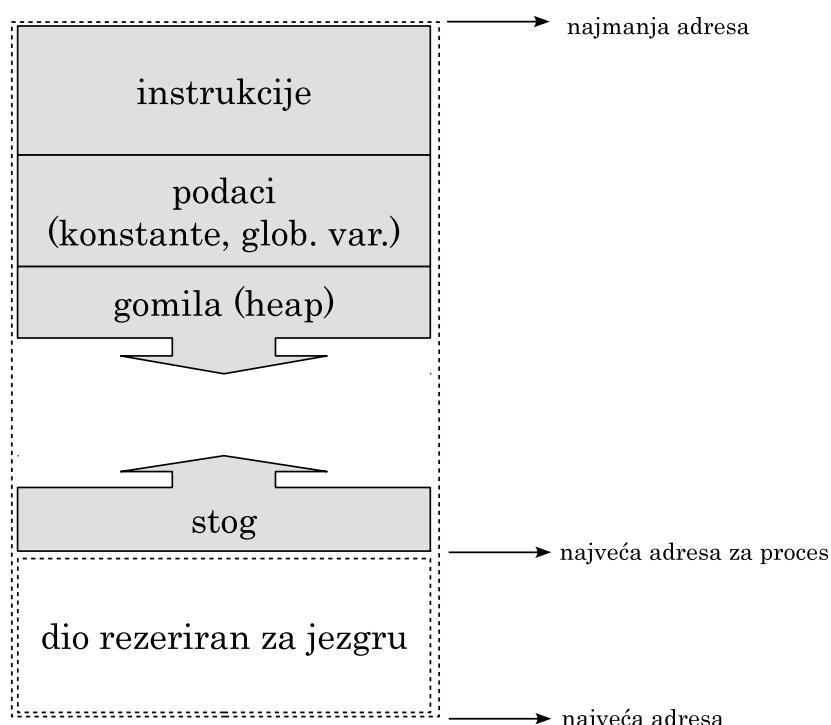


Slika 8.9. Pretvorba adrese kod straničenja

Prevođenje treba biti obavljeno sklopom – inače bi sustav bio prespor

8.4.2. Struktura i organizacija logičkog adresnog prostora procesa (info)

- kada bi tablica prevodenja bila linearna i za svaki proces opisivala cijeli mogući adresni prostor tog procesa, onda bi bila jako velika (zauzimala bi znatan dio spremnika)
 - npr. za 4 GB adresnog prostora i stranice veličine 4 KB treba 2^{20} redaka!
- takva linearна tablica nije potrebna jer većini procesa ne treba cijeli adresni prostor (npr. pogledati zauzeća spremničkog prostora procesa u task manageru)
- ipak, radi fleksibilnosti koristi se početak i kraj adresnog prostora
 - na početku su instrukcije i podaci
 - iza njih raste gomila (koristi se kod malloc/free, new/delete operacija)
 - na kraju je stog koji raste prema manjim adresama



Slika 8.10. Struktura procesa (logički adresni prostor)

Dio procesa može opisivati (mapirati) dio same jezgre

- na Windowsima i Linuxu dio jezgre je mapiran u zadnjem dijelu adresnog prostora (npr. zadnja 2 GB kod 32-bitovnih Win32 sustava, od 2^{47} na 64-bitovnim)
- taj dio ne može koristiti proces iz svojih dretvi, već je dohvatljiv samo u jezgrinim funkcijama (dretve nemaju privilegije za korištenje tih stranica – to se može postaviti u opisniku tih stranica)
- npr. u taj dio se može smjestiti kod za prihvat prekida

```
#include <stdio.h>
int a = 1;
int main() {
    int b = 2;
    printf ("gl. var. => %p\n", &a );
    printf ("lok.var. => %p\n", &b );
    printf ("f. main  => %p\n", main );
    return 0;
}
```

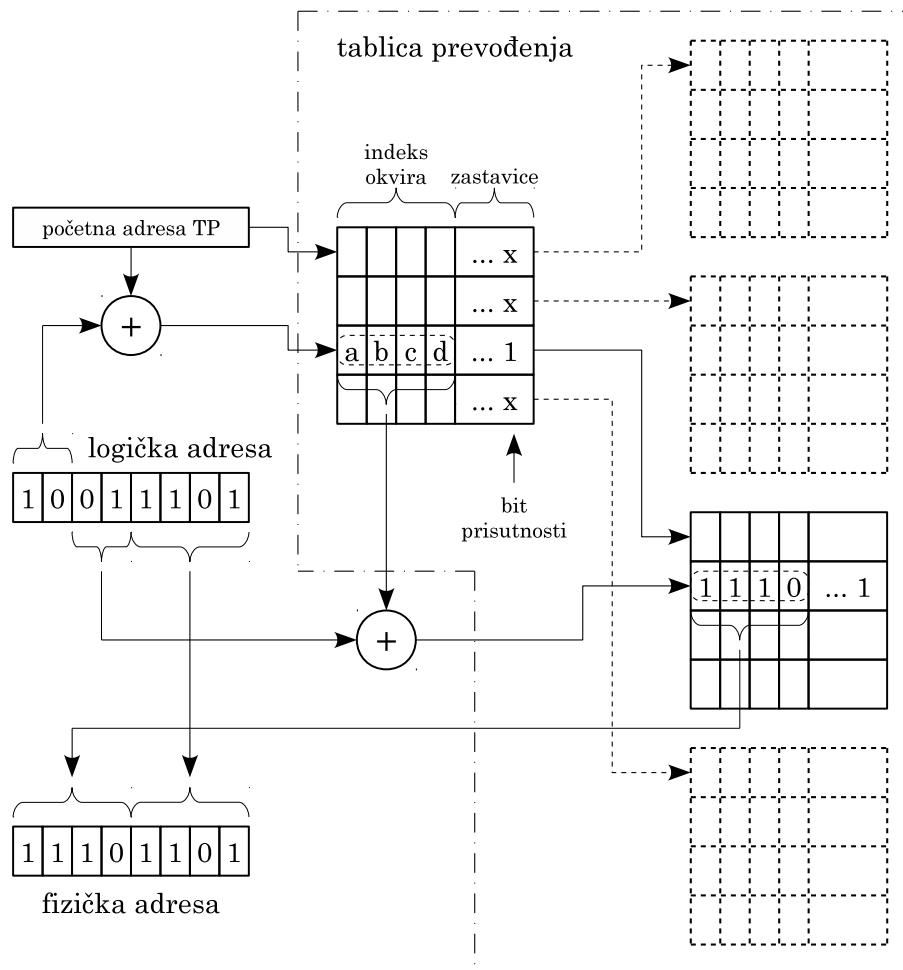
```
# primjer pokretanja na 32-bitovnom rač.
gl. var. => 0x804a014
lok.var. => 0xbfb4f1bc
f. main  => 0x80483e4

# primjer pokretanja na 64-bitovnom rač.
gl. var. => 0x601020
lok.var. => 0x7fff1164fb1c
f. main  => 0x4004f4
```

8.4.3. Hijerarhijska organizacija tablice prevođenja

Jedan od načina organizacije tablice prevođenja koji ne zahtijeva opise svih mogućih stranica je *hijerarhijska organizacija*

- tablicu podijeliti na manje tablice
- dodati tablicu koja opisuje skup manjih tablica (ili sve u nižoj razini)
- u slučaju više razina u hijerarhiji (više od dvije) dodati i tablicu iznad
- tablice u najnižoj razini koje opisuju stranice koje se ne koriste nisu ni potrebne!

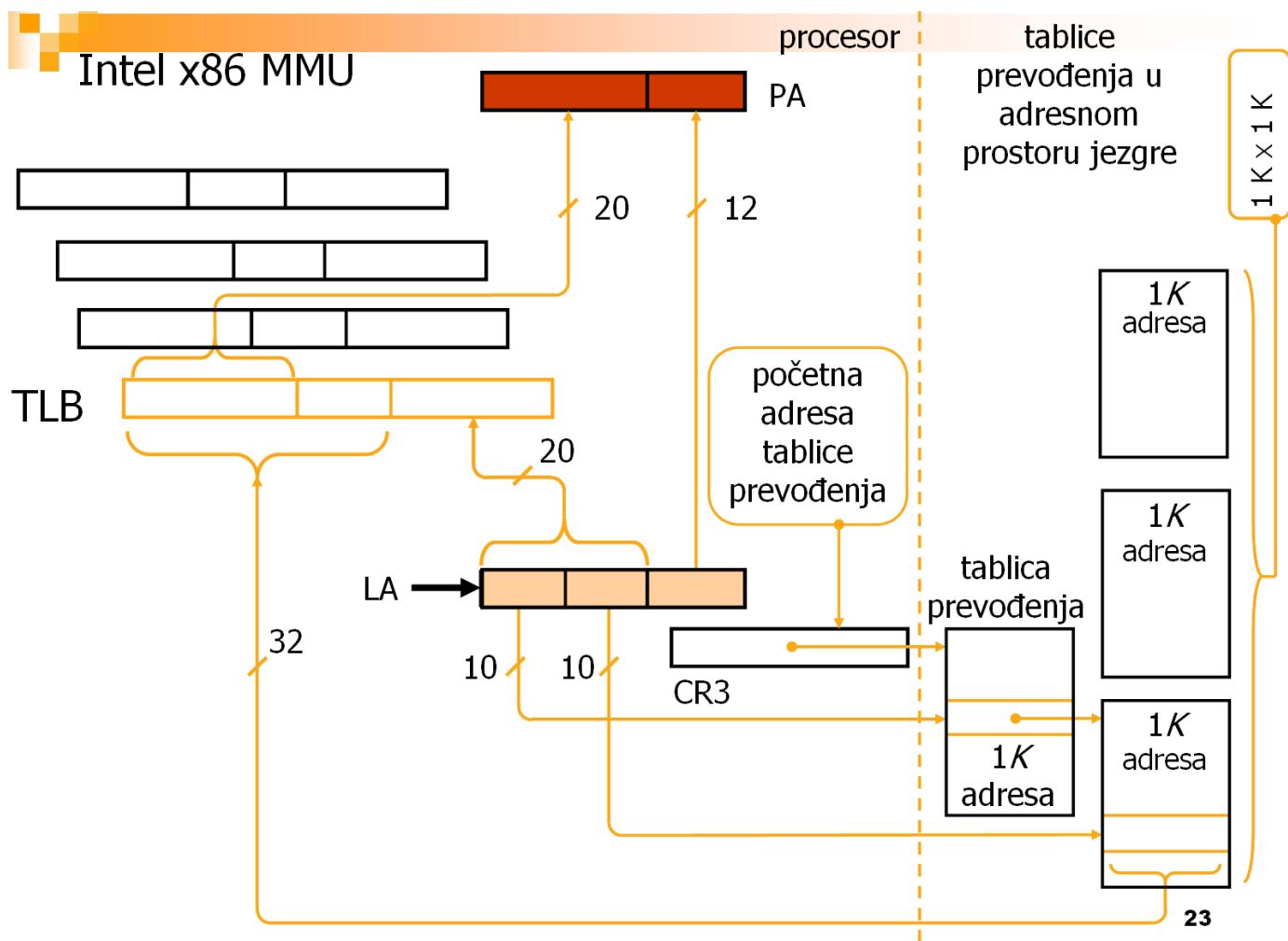


Slika 8.11. Primjer dvorazinske organizacije

Hijerarhijska organizacija omogućava korištenje značajno manje strukture podataka za opis tablice prevođenja za manje procese.

Za gornji primjer, ako bi npr. cijeli proces stao u jednu stranicu na logičkoj adresi koja započinje na 10010000b tada bi bile potrebna samo dva dijela tablice: početni dio (u 1. razini hijerarhije) te 3. dio u drugoj razini.

Na sličan način je i u stvarnim sustavima (32-bitovnim arhitekturama) potrebno tek nekoliko dijelova: početni dio u hijerarhiji te nekoliko dijelova u drugoj razini (za opis instrukcija i podataka na početku, te za opis stoga na kraju)



Slika 8.12. Primjer dvorazinske organizacije kod x86 arhitekture

Međuspremnik za opisnike (engl. *translation lookaside buffer* – TLB)

- prevođenje logičke adrese u fizičku zahtjeva prisutnost opisnika te stranice
- dohvati opisnika ide u dva koraka – dva dohvata iz spremnika
 - prvo se dohvaća adresa tablice u drugoj razini (iz tablice u prvoj)
 - iz tablice u drugoj razini dohvaća se opisnik
- za jedan dohvati podataka za proces potrebna su tri pristupa spremniku!
- da to ne bude tako (sporo) služi TLB koji sadrži prethodno dohvaćene opisnike
- tek ako opisnik već nije u TLB on se mora dohvatiti iz spremnika

Opisnik stranice:

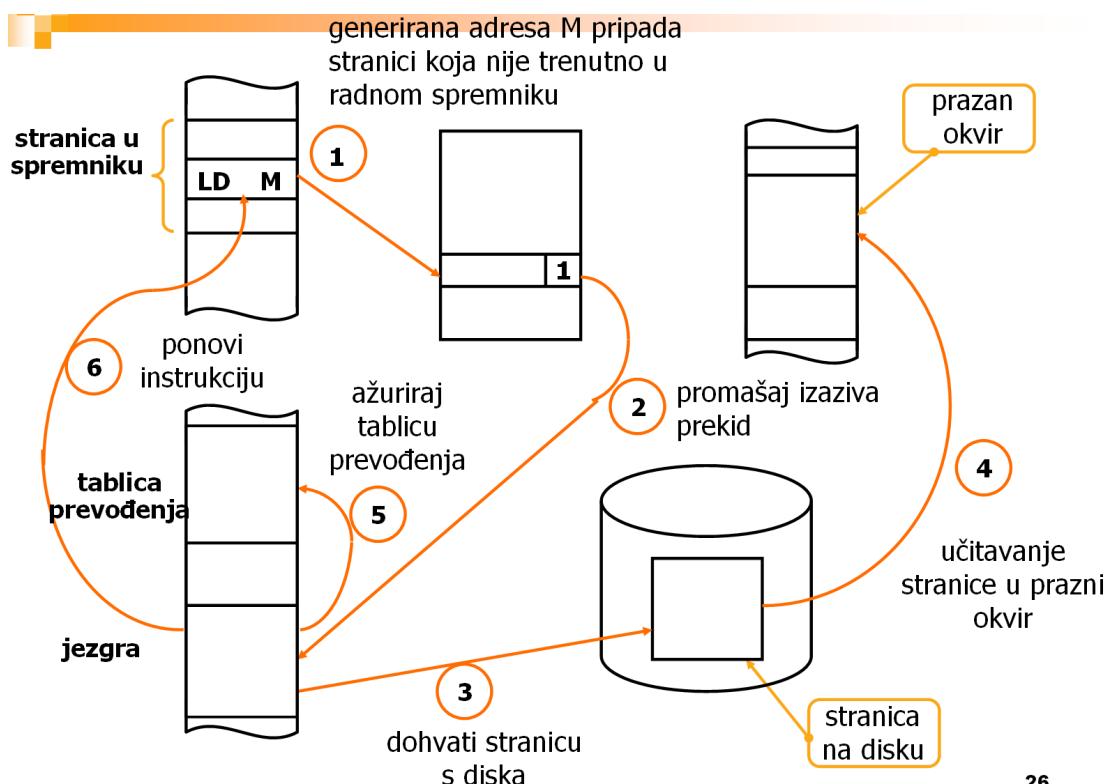
- bitovi 31-12 – određivanje indeksa okvira
- bitovi 11-0 su zastavice:
 - V – bit prisutnosti (engl. *validity bit*) (je li stranica u radnom spremniku)
 - A – stranica je korištena (engl. *accessed*)
 - D – “prljava” (engl. *dirty*) (stranica je mijenjana)
 - W – zaštita od promjene (engl. *write protect*)
 - O – stranica je za OS ne za proces
 - Wt – “write through” (svaka promjena stranice pokreće njenu pohranu i na pomoći spremnik tako da i u slučaju nestanka napajanja sustav ostaje zapamćen na pomoći spremniku)
 - Gl – globalna (npr. za ostvarenje dijeljenog spremnika)

8.4.4. Straničenje na zahtjev

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

- sklop za upravljanje spremnikom izaziva *prekid*
- u obradi prekida dohvata se stranica i stavlja u radni spremnik
- kažemo da se dogodio *promašaj*

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



26

Slika 8.13. Operacije po promašaju

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. kada bi imali instrukciju `DIV a, b, d, r` koja cijelobrojno dijeli `a` i `b`, kvocient spremi u `d`, a ostatak u `r`, ako bi se promašaj dogodio pri spremanju ostatka `r` (u spremnik), cijela instrukcija se ne smije ponoviti (npr. zato jer `d` može biti jednak `b-a`)

8.4.5. Usporenje rada programa zbog straničenja

Promašaji će usporiti rad programa (svaki za ≈ 10 ms)!

Primjer 8.3. Usporenje programa

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

Prosječno trajanje sabirničkog ciklusa može izraziti sa:

$$\bar{T}_B = \frac{(N - 1) \cdot T_B + T_D}{N} \text{ s} \quad (8.6.)$$

Tablica 8.2. Usporenje rada programa zbog straničenja

| N | 10^3 | 10^4 | 10^5 | 10^6 | 10^7 |
|-----------------|---------------|--------------|--------|--------|--------|
| \bar{T}_B | 10,01 μ s | 1,01 μ s | 110 ns | 20 ns | 11 ns |
| \bar{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.

8.4.6. 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” programa
 - instrukcije koje se izvode su blizu jedna drugoj (u spremniku)
 - podaci nad kojima intrukcije nešto rade su većinom takođe blizu jedni drugima
 - stog je kompaktan
- načini:
 - korištenje zastavica A i D iz opisnika stranice
 - * zastavica A – “nedavno” korištene stranice (engl. *accessed*)
 - * zastavica D – “čiste” i “nečiste” stranice (engl. *dirty*), trošak njihove zamjene nije isti
 - teorijske strategije: FIFO, LRU, LFU, OPT
 - satni algoritam (ono što se koristi)

8.4.6.1. Teorijske strategije zamjene stranica

FIFO – First-In-First-Out

- izbaciti najstariju stranicu – onu koja je najduže u radnom spremniku

LRU – Least-Recently-Used

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

LFU – Least-Frequently-Used

- izbaciti stranicu koja se najmanje puta koristila

OPT – optimalna strategija

- izbaciti stranicu koja se najduže neće koristiti (u budućnosti)
- nije ostvariva, ali može služiti za usporedbu

Navedene strategije su presložene za ostvarenje.

8.4.6.2. Satni algoritam (engl. *clock algorithm, second chance algorithm*)

- inačica LRU
- koristi se (UNIX, Windows) (prilagođena)
- princip rada:
 - zastavica A u opisniku stranice koja je u radnom spremniku se postavlja u 1 pri korištenju te stranice (sam sklop postavlja tu zastavicu u 1 kad se iz nje čita ili u nju piše)
 - u simulaciji zastavicu A se može postaviti u 1 odmah pri učitavanju nove stranice
 - sve stranice u radnom spremniku (u okvirima) su u jednoj kružnoj listi
 - pri traženju stranice za izbacivanje lista se obilazi kružno – posebnom kazaljkom
 - * ako stranica na koju pokazuje kazaljka ima zastavicu $A == 0$
 - . stranica se izbacuje (zamjenjuje potrebnom)
 - . kazaljka se ne pomiče!
 - * ako stranica na koju pokazuje kazaljka ima zastavicu $A == 1$
 - . zastavica A se postavlja u nulu ($A=0$)
 - . kazaljka se pomiče na iduću stranicu u listi

Zadatak: 8.1. Teorijske strategije

U sustavu sa straničenjem program veličine 400 riječi (1-400) generira slijed adresa: 23, 47, 333, 81, 105, 1, 400, 157, 30, 209, 289, 149, 360. Program ima na raspolaganju 200 riječi radnog spremnika. Napisati niz referenciranja stranica veličine 50 riječi. Koliki je postotak promašaja za sve četiri navedene strategije izbacivanja stranica?

| | | | | | | | | | | | | | |
|----------|----|----|-----|----|-----|---|-----|-----|----|-----|-----|-----|-----|
| adresa | 23 | 47 | 333 | 81 | 105 | 1 | 400 | 157 | 30 | 209 | 289 | 149 | 360 |
| stranica | — | — | — | — | — | — | — | — | — | — | — | — | — |
| FIFO | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| LRU | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| LFU | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| OPT | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| Satni | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| Zast. A | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |

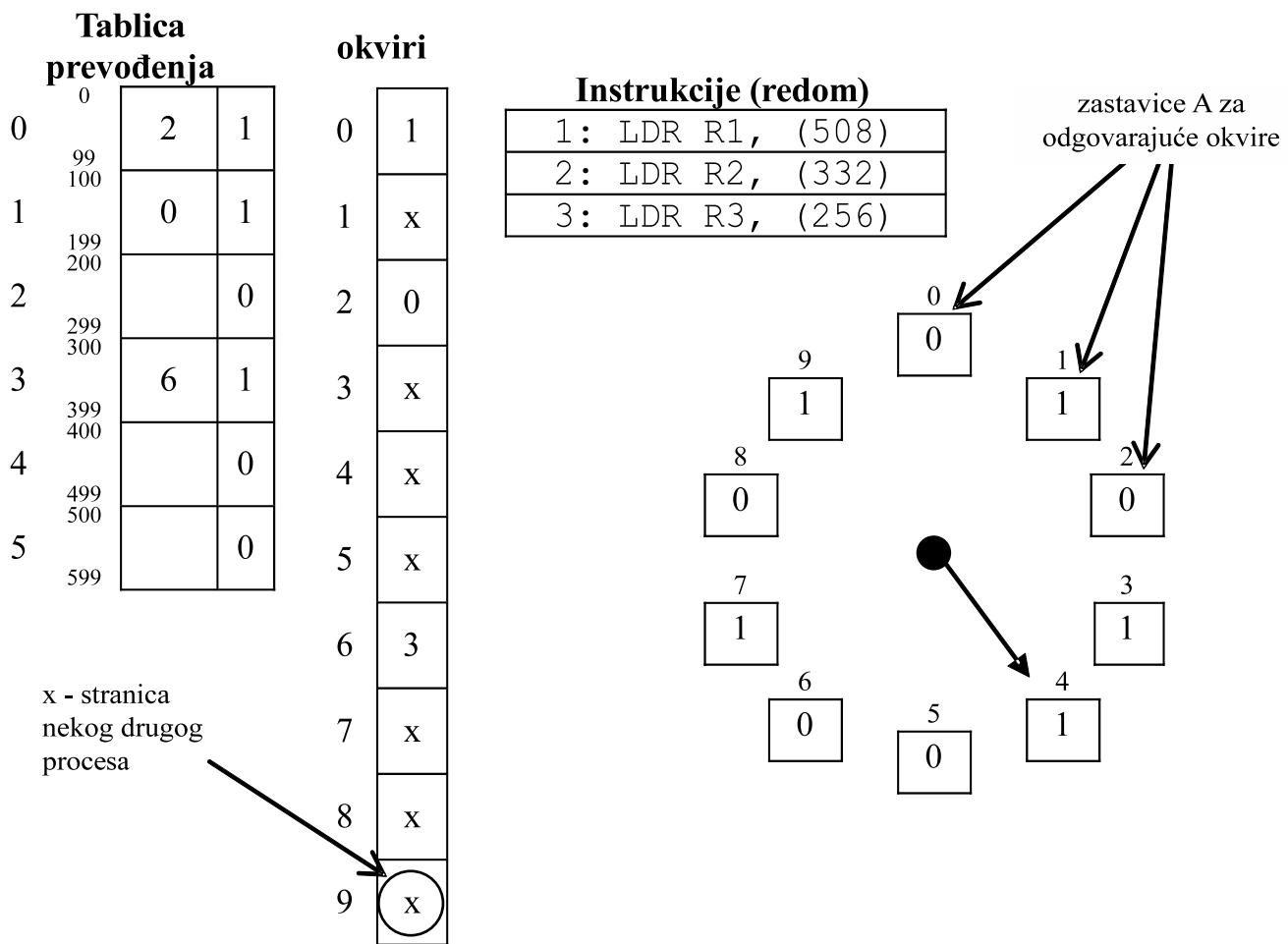
Dodatak: Prikazati trenutni izgled tablice prevođenja na kraju primjene LFU strategije

Dodatak: Prikazati trenutni izgled tablice prevođenja na kraju primjene LFU strategije

| indeks | ind. okvira | bit pris. |
|--------|-------------|-----------|
| 1 | | |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |
| 8 | | |

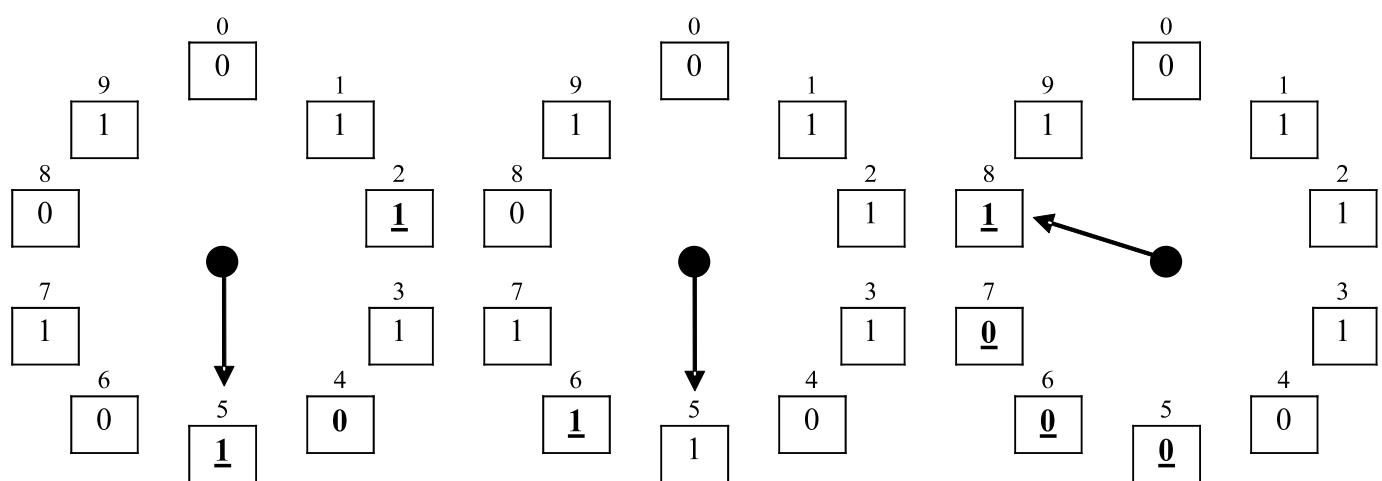
Zadatak: Satni algoritam

Za neki sustav koji koristi straničenje, zadana je tablica prevodenja za jedan proces, stanje okvira sustava, stanje kazaljke i zastavica A (za upravljanje metodom satnog algoritma). Ako zadani proces treba napraviti zadane instrukcije, kako će se sustav mijenjati? Neka se instrukcije nalaze u 0. stranici.



x - stranica
nekog drugog
procesa

Rješenje:



a) nakon 1. instr.

b) nakon 2. instr.

c) nakon 3. instr.

Objašnjenje: Dohvat prve instrukcije izazvat će pogodak (0. stranica je u 2. okviru). Izvođenjem prve instrukcije dogoditi će se prekid zbog promašaja podataka (traži se stranica 5 programa). Zastavica četvrtog okvira A(4) postaviti će se u 0 te će se kazaljka pomaknuti. Zastavica A(5) je 0 te će se taj okvir oslobođiti i u njega staviti 5. stranica procesa. Tada se može obaviti prva instrukcija. Njenim izvođenjem (čitanjem iz 5. okvira) postavlja se zastavica A(5) u 1 (sl. a)).

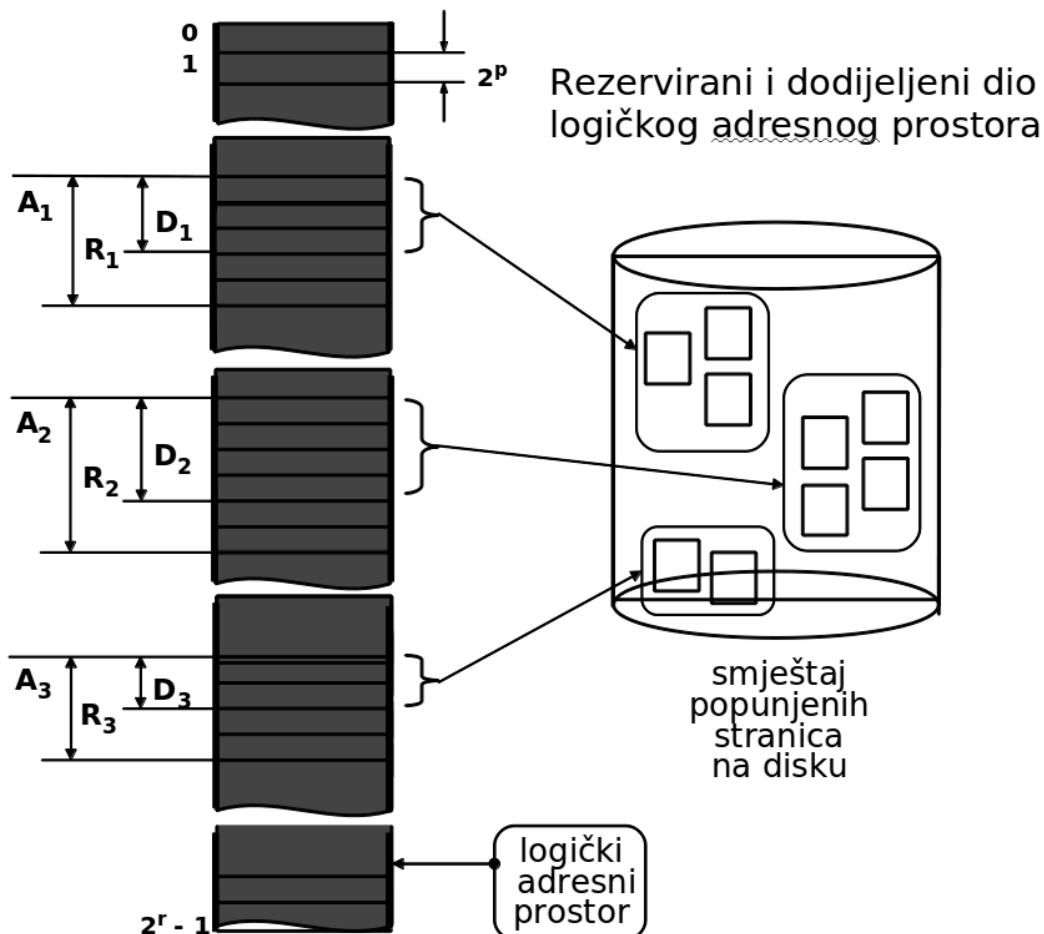
Druga instrukcija traži 3. stranicu koja se nalazi u okviru 6. Njenim izvođenjem (čitanjem podatka iz 6. okvira) postaviti će se zastavica A(6) u 1 (sl. b)) (kazaljka se ne miče).

Treća instrukcija traži podatak iz 2. stranice koja nije u radnom spremniku, pa će se kazaljka pomaknuti, najprije na 6. mjesto, pa na 7. (pritom postavlja A(5), A(6) i A(7) u nulu) i tek na 8. pronalazi $A(8)=0$, izbacuje stranicu koja se tu nalazi i učitava stranicu 2 procesa. Nakon toga može se izvesti instrukcija 3. Izvođenjem 3. instrukcije postavlja se zastavica A(8) u 1 (sl. c)).

8.4.7. Rezervirani i dodijeljeni dijelovi procesa

Opisnik spremničkog prostora procesa – *informacijski blok* sastoji se:

- tablice prevodenja
- opisa gdje je proces smješten na pomoćnom spremniku
- dodatnog opisa:
 - koji su dijelovi procesa dodijeljeni – D (i opisani tablicom prevodenja)
 - koji su dijelovi procesa rezervirani – A (a još nisu posve opisani i tablicom prevodenja)



Slika 8.14. Rezervirani i dodijeljeni dijelovi adresnog prostora procesa

Navedeni opisi rezerviranog i dodijeljenog prostora omogućavaju:

- stvaranje potrebnih stranica kada one postanu neophodne (uz popunjavanje tablice prevodenja)
- detekciju greške (engl. *segmentation fault*) te prekid procesa
 - primjerice, ako je tražena adresa (u nekoj instrukciji) izazvala prekid zbog toga što stranica nije u radnom spremniku ili nije čak ni opisana u tablici prevodenja, pregledom navedene strukture podataka može se ustanoviti je li tražena adresa unutar dodijeljenog prostora procesa ili nije
 - * ako jest, onda se stvara takva stranica i opisuje u tablici prevodenja
 - * ako nije, prekida se proces jer je izazvao kritičnu grešku (kako se oporaviti od nje?)

8.4.8. Upravljanje okvirima

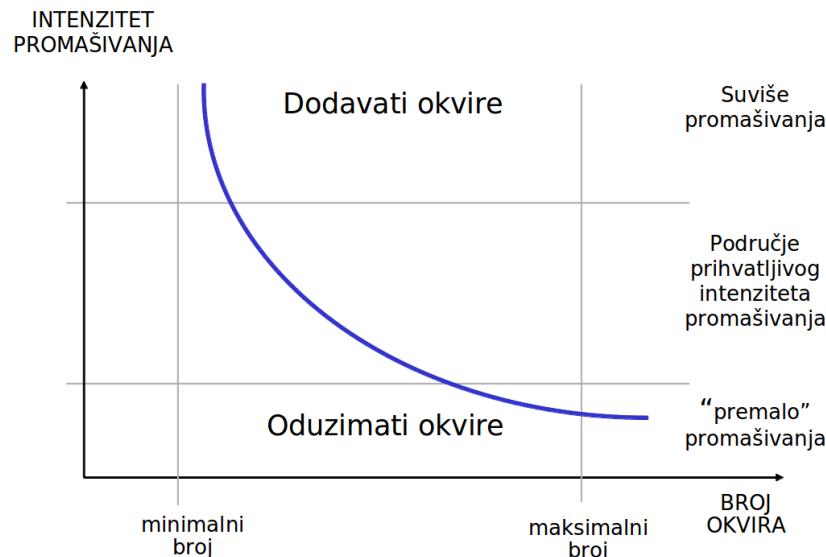
OS mora voditi evidenciju o svim okvirima

Stanja u kojima se okviri mogu naći:

- aktivno – sadrži stranicu koja se koristi
- slobodno – trenutno se ne koristi, ali ima neki sadržaj – potrebno ga je obrisati prije dodjele (osim ako se ne prepisuje drugom stranicom)
- slobodno s obrisanim sadržajem – spremna za dodjelu
- neispravno – greška u tom djelu spremnika te se oni ne koriste

8.4.8.1. Dodjela okvira

- dio okvira koristi OS za svoje potrebe (strukture podataka i međuspremnike)
- koliko okvira dati procesima?
 - koliko traže (možda, ako ima toliko slobodnih okvira)
 - fiksni broj (npr. svima isto)
 - prema postotku promašaja: (slika)



Slika 8.15. Dodjela okvira prema intenzitetu promašaja

OS ima i sučelje za programe koji preko njega mogu utjecati na straničenje

- može zatražiti da neke stranice (dijelovi procesa) ostanu uvijek u radnom spremniku
- uglavnom su to samo preporuke (koje OS ne mora poštovati)
- primjeri:
 - POSIX: `int mlock (const void * addr, size_t len);`
 - POSIX: `int mlockall (int flags);`
 - Win32: `bool VirtualLock (void *address, void *size);`
 - Win32: `bool SetProcessWorkingSetSize (void *process, size_t min, size_t max);`

8.4.9. 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

Inicijalizacija matrice $A[N][N]$ po recima ili stupcima? Neka redak matrice stane u jednu stranicu.

```
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.

Inicijalizacija po stupcima = N^2 promašaja.

Zadatak: 8.3.

U sustavu s virtualnim spremnikom, veličina okvira je N riječi, a okviri se pune na zahtjev. Algoritam zamjene stranica je LRU. Poredak $A[N, N]$ je pohranjen po retcima (na susjednim lokacijama se mijenja desni indeks). Koliko promašaja će izazvati prikazani program ako za poredak A u radnom spremniku postoji:

- samo jedan okvir
- dva okvira
- tri okvira
- N okvira.

Program:

```
t = 0;
za i=1 do N-1
{
    za j=i+1 do N
    {
        t = t + A[i, j];
        t = t * A[j, i];
    }
}
```

Napomena: Zanemariti promašaje zbog dohvata instrukcija samog programa i pristupa pomoćnim varijablama. (Na primjer, neka je cijeli program u priručnoj memoriji za instrukcije, a pomoćne varijable i, j, t u registrima.)

Zadatak: 8.4. Upravljanje spremnikom

U nekom sustavu trebaju se obaviti četiri programa: 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 programa znani su unaprijed i mogu se iskazati slijedećim nizom događaja: P1 pokrenut; P2 pokrenut; P1 završava; P3 pokrenut; P4 pokrenut; P3 završava; P2 završava; P4 završava. Sustav na raspolaganju ima 20 MB memorije rezervirane za korisničke programe. Prikazati stanje radnog spremnika ako se koriste metode upravljanja spremnikom:

- a) statičko upravljanje s veličinom segmenta od 10 MB
 - b) dinamičko upravljanje
 - c) straničenje, uz veličinu stranice od 1 MB.
-

8.4.10. Zaključne napomene o straničenju

Prednosti:

- nema fragmentacije
- zaštita jezgre i procesa
- pokretanje i velikih programa – učitavaju se samo trenutno potrebne stranice
- podržano sklopoljem i operacijskim sustavom
- transparentno za program (ali dobar program može biti bolji ako koristi lokalnost)
- ostvarenje dijeljenog spremnika između procesa – tablice prevođenja oba procesa pokazuju na iste stranice
- duplicitanje procesa (fork) – nije potrebno fizički kopirati dijelove koji se samo čitaju

Nedostaci:

- potreban sklop, koji nije jednostavan
- moguće usporenje zbog promašaja (RT sustavi!)

Što arhitekt/programer treba/može napraviti?

- teoretski ništa – upravljanje je transparentno – potpuno rješeno sklopoljem 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 ($L_3 \Leftrightarrow L_2 \Leftrightarrow L_1$) \Leftrightarrow registri procesora
- korištenje u sustavima za rad u stvarnom vremenu (RT)
 - zaključati stranice u radni spremnik (sučelje OS-a)
 - * već navedeno sučelje `mlock/mlockall`, `VirtualLock`, `SetProcessWorkingSetSize`

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:
 - statičkog upravljanja spremnikom
 - dinamičkog upravljanja spremnikom
 - upravljanje spremnikom straničenjem.
5. Što su to fizičke a što logičke adrese?
6. Kod kojih algoritama upravljanja spremnikom je proces u fizičkim a kod kojih u logičkim adresama?
7. Objasniti pojmove: unutarnja i vanjska fragmentacija.
8. Koji se postupci koriste kod dinamičkog upravljanja spremnikom radi smanjenja fragmentacije?
9. Izvesti i objasniti Knuthovo 50% pravilo.
10. Nacrtati sklop koji se koristi za pretvorbu adresa kod dinamičkog upravljanja spremnikom.
Koja su proširenja tog sklopa potrebna da bi se dodala i zaštita?
11. Objasniti pojmove: stranica, okvir, tablica prevođenja u kontekstu straničenja.
12. Čemu služi i od čega se sastoji tablica prevođenja?
13. Zašto se koristi hijerarhijska organizacija tablice prevođenja?
14. Čemu služe zastavice V (bit prisutnosti), A (oznaka korištenja) te D (oznaka izmjene) u opisniku stranice?
15. Što je to "promašaj" u kontekstu straničenja?
16. Opisati upravljanje spremnikom metodom "straničenje na zahtjev".
17. Opisati strategije zamjene stranica: FIFO, LRU, LFU, OPT i satni algoritam.
18. Opisati mogućnosti upravljanja okvirima (načini dodjele).
19. Što je to "prostorno vremenska lokalnost" i kako ona utječe na učinkovitost sustava?
20. Kakva sučelja operacijski sustavi nude programima radi upravljanja straničenjem?

9. Datotečni sustav

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

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

Danas razlikujemo:

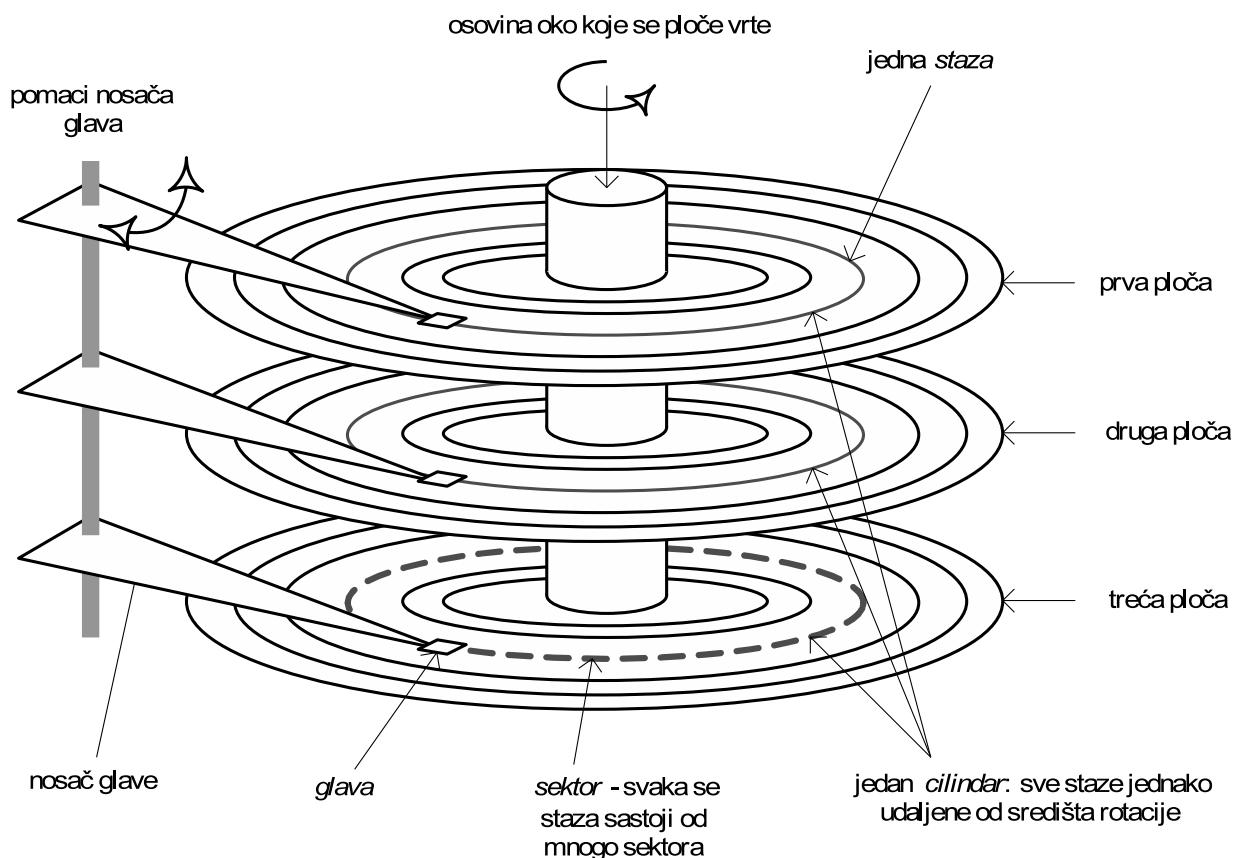
- HDD (engl. *Hard Disk Drive*) – tvrde diskove
- SSD (engl. *Solid State Drive*) – “nepomične” diskove
 - značajno brži – nemaju pokretnih djelova
 - noviji, do 10x skuplji od HDD-a (po TB)

U nastavku se detaljnije razmatraju samo tvrdi diskovi – HDD, tj. kad se spomene ‘disk’ misli se na HDD.

9.1.1. Fizička svojstva diskova

HDD je elektromehanička naprava. Sastoje 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



Podatkovna jedinica 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

- broj sektora na stazi može biti varijabilan, što je staza dalje od centra ona može imati više sektora
- uz osnovne informacije (podatke) koji su zapisani u sektoru, na disk, u "zaglavlje sektora" spremaju se i dodatne informacije (zaštitni bitovi)

Upravljački sklop diskovne jedinice

- upravlja mehaničkim i elektroničkim dijelovima
- ima procesor (očitani signali nisu lijepi uglati)
- spaja se na sabirnicu
- ima međuspremnik
- pretvara "linearnu" ili "logičku" adresu u CHS i obratno

Logička adresa sektora (engl. *logical block addressing* – 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, za ažurne provjeriti na webu, npr. wikipediji)

- kapaciteti: od ≈ 100 GB do 4 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): i do 100-tinjak MB/s; ≈ 70 -tak MB u prosjeku
- vrijeme pristupa (od zahtjeva do posluživanja): 2 do 15 ms (≈ 8 ms)
 - prosječno pomicanje glave za “slučajni zahtjev”
 - računa se kao pomicanje glave za 1/3 staze
- veličine sektora: 512 B, 4 kB
- SSD diskovi imaju znatno bolja svojstva
 - vrijeme pristupa: $\approx 0,1$ ms i manje
 - prijenos: 150-500 MB/s

9.1.2. Vremenska svojstva diskova

Komponente trajanja prijenosa podataka:

1. postavljanje glave na početak podataka (engl. *head position time*)
 - trajanje traženja staze, vrijeme postavljanja (engl. *seek time*)
 - obično zadano:
 - * formulom $T_D = \dots$ ili
 - * prosječnim trajanjem – vrijeme prijelaza preko 1/3 staze = *prosječno vrijeme traženja staze*
 - ovisi o početnom i konačnom položaju
 - sastoji se od vremena:
 - a) ubrzavanja ručice glave
 - b) kretanja konstantnom brzinom (maksimalnom)
 - c) usporavanja
 - d) finog pozicioniranja na stazu
 - prosječno rotacijsko kašnjenje (engl. *rotation latency*) = $\bar{T}_R = T_R/2$
 - 2. čitanje podataka
 - trajanje čitanja dijela staze ili cijele staze
 - po potrebi uzeti u obzir “faktor preplitanja” (info)
 - 3. prijenos podataka u radni spremnik
 - ovisno o disku ova akcija može ići paralelno sa:
 - postavljanjem glave na iduću stazu ili sektor
 - čitanjem idućih sektora
 - zadano u zadatku, ništa nije “prepostavljeno”

“Faktor preplitanja” (engl. *interleave factor*) (info)

- ako su dva sektora koja logički slijede jedan iza drugoga fizički smješteni na sektorima n te $n + k$ tada je “faktor preplitanja” za taj disk jednak k .

Zadatak: 8.1.

Disk sa samo jednom pločom, 2000 staza, 128 sektora po stazi vrti se brzinom 4800 okretaja u minuti. Veličina sektora je 512 bajtova. Upravljački sklop pročita jedan cijeli sektor u interni spremnik, a zatim ga prenosi u glavni spremnik. Prijenos u glavni spremnik odvija se brzinom od 10 Mbit/s, a za to vrijeme sklop ne može čitati s diska.

- Koliki treba biti faktor preplitanja (engl. *interleave factor*)?
- Koliko prosječno traje prebacivanje kompaktno smještene datoteke veličine 235 kB ako je vrijeme postavljanja 12 ms, vrijeme premještanja sa staze na stazu 2 ms?

Zadatak: 8.2.

Disk ima 100 sektora po stazi, 2000 staza, 4 ploče i vrti se brzinom 7200 okretaja u minuti (engl. *rpm – rotations per minute*). Veličina sektora je 1 kB. Podaci su zapisani na obje strane ploče (ukupno 8 površina). Upravljački sklop pročita jednu cijelu stazu u interni spremnik, a zatim je prenosi u glavni spremnik. Prijenos u glavni spremnik odvija se brzinom od 200 Mbit/s, a za to vrijeme sklop ne može čitati s diska (ali može pomicati glavu ako je potrebno za iduće zahtjeve).

- Koliki je kapacitet tog diska?
- Koliko prosječno traje prebacivanje kompaktno smještene datoteke veličine 1350 kB ako je vrijeme postavljanja 10 ms i vrijeme premještanja sa staze na stazu 1 ms?

Zadatak: 8.2.1

Program *A* veličine 5 MB i program *B* veličine 9200 kB kompaktno su smješteni na disku koji ima 128 sektora po stazi. Početni sektor i jednog i drugog programa su ujedno i prvi sektori na stazi na kojoj se nalaze. Veličina sektora je 1 kB, a disk se okreće brzinom 7200 rpm. Upravljački sklop pročita jednu cijelu stazu u interni spremnik, a zatim je prenosi u glavni spremnik. Prijenos u glavni spremnik odvija se brzinom od 400 Mb/s, a za to vrijeme sklop ne može čitati s diska. Vrijeme traženja staze je 10 ms, a vrijeme premještanja sa staze na stazu 1 ms. Koliko vremena protekne od istovremenog izdavanja naredbi za pokretanjem programa *A* i *B* pa do trenutka kada se oba programa izvode ako:

- se prvo program *A* u cijelosti učita u radnu memoriju, a zatim se učitava program *B*
- se u radni spremnik prvo prenosi jedna staza programa *A*, a zatim jedna staza programa *B* i tako dalje naizmjenično.
- Koji je način čitanja brži i zbog čega dolazi do razlike u izračunatim vremenima u slučajevima a) i b)?

Zadatak: 8.2.2

Ista fotografija kompaktno je smještena u dvije različite datoteke na disku u nekomprimiranom (128 MB) i komprimiranom formatu (896 kB). Disk ima 128 sektora po stazi, veličine sektora je 1 kB, a disk se okreće brzinom 7200 rpm. Početni sektor i jednog i drugog programa su ujedno i prvi sektori na stazi na kojoj se nalaze. Upravljački sklop pročita jednu cijelu stazu u interni spremnik, a zatim je prenosi u glavni spremnik. Prijenos u glavni spremnik odvija se brzinom od 100 Mb/s, a za to vrijeme sklop ne može čitati s diska. Vrijeme traženja staze je 10 ms, a vrijeme premještanja sa staze na stazu 1 ms. Ako procesor može dekomprimirati komprimiranu sliku brzinom od 5 Mb/s prikazuje li se brze slike koja se učita iz nekomprimirane ili komprimirane slike? Objasniti.

9.1.3. Posluživanje zahtjeva

- disk je SPOR pa je moguće koristiti postupke optimiranja dohvata
- jedan od oblika optimiranja je optimiranje nad skupom zahtjeva
- ne posluživati zahtjev po zahtjev redom, već iz skupine zahtjeva naći najbolji način redoslijeda posluživanja
 - optimiranje može raditi OS ali i upravljački sklop diska (nad dobivenim zahtjevima)

Strategije posluživanja:

- FCFS – redom prispjeća (engl. *First Come First Served*)
- SSTF – s najkraćim vremenom premještaja glave (engl. *Smallest Seek Time First*)
- SCAN – pregledavanje: ide u jednom smjeru do kraja (zadnje staze u tom smjeru) i staje na svakoj stazi na kojoj postoji zahtjev
- LOOK – pregledavanje: ide u jednom smjeru do zadnjeg zahtjeva u tom smjeru i staje na svakoj stazi na kojoj postoji zahtjev
- C-SCAN, C-LOOK – slične gornjim, samo što se posluživanje obavlja samo u jednom smjeru; kad se dođe do kraja ili zadnjeg zahtjeva, brzim potezom se glava postavlja na prvu stazu (ili prvi zahtjev za C-LOOK) s druge strane.

Zadatak: 9.1.

Disk s pokretnim glavama ima 100 staza (0 - 99). Neka se glava trenutno nalazi na stazi 29, s tim da je prije bila na stazi 8. Zahtjevi za pristup pojedinim stazama svrstani po redu prispjeća su: 3, 7, 40, 98, 71, 68, 70, 21, 5 i 49.

- a) Opisati svaku od navedenih strategija!
- b) Grafički prikazati kretanje glave prilikom obrade zahtjeva za sve strategije!
- c) Koliki je ukupan pomak glave pri izvršenju tih zahtjeva za strategije posluživanja: redom prispjeća (FCFS), s najkraćim vremenom premještaja glave (SSTF), pregledavanje (LOOK i SCAN) i kružno pregledavanje (C-LOOK i C-SCAN)?
- d) Kod kojih može doći do izgladnjivanja (beskonačnog odgađanja posluživanja nekih zahtjeva zbog posluživanja novo pristiglih zahtjeva)?

9.2. Datotečni sustav

CHS \Rightarrow LBA

- adresiranje korištenjem numeracije staza/ploča/sektora (engl. *cylinder-head-sector* – *CHS*) je komplikiran 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

- 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
- 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 pojma.

Datotečni sustav daje odgovore na pitanja:

- kako su datoteke smještene na disku

- fizičko smještanje: gdje, u kojim sektorima, 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 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 (Vista+)
 - * C:\Users\korisničko_ime – korisnički direktoriji, postavke i podaci (Vista+)
 - * C:\Documents and Settings\korisničko_ime – korisnički direktoriji (Windows XP–)
 - * C:\pagefile.sys – pomoćni spremnik za straničenje
 - druge particije, CD/DVD i sl.: svaki ima svoju oznaku (D:, E:, ...)
- UNIX pristup:
 - / – početni direktorij
 - /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) nalazi se dakle:

- “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 najmanjih koje su pohranjene 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

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 iz primjer 9.1. prema NTFS pravilima:

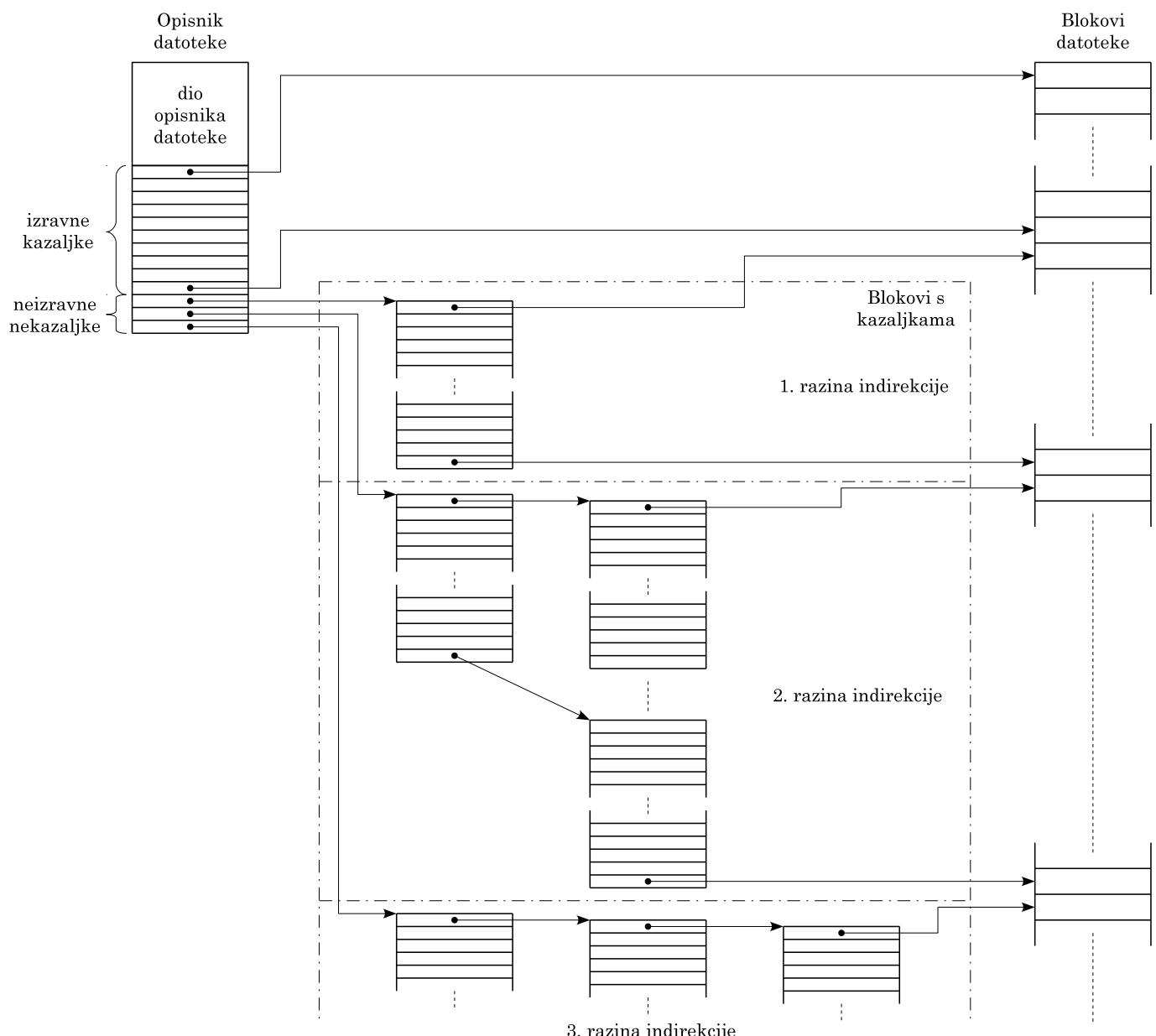
| VCN | LCN | # |
|-----|-----|---|
| 1 | 37 | 2 |
| 3 | 18 | 3 |
| 6 | 57 | 5 |
| 11 | 51 | 5 |

– broj uzastopnih blokova – nakupina blokova

Prvi blok datoteke (VCN=1) nalazi se u 37. bloku particije (LCN=37). Obzirom 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.

9.3.2. UNIX i-node

- Opisnik datoteke = i-node = index node
- Za opis blokova koriste se slijedeće kazaljke:
 - deset izravnih kazaljki
 - * kazaljke koje izravno pokazuju na blokove datoteke (LCN indeksi)
 - * npr. 5. kazaljka pokazuje na blok koji sadrži 5. blok datoteke
 - jedna jednostruko indirektna kazaljka
 - * kazaljka na blok sa kazaljkama na blokove datoteke
 - jedna dvostruko indirektna kazaljka
 - * kazaljka na blok sa kazaljkama na blokove sa kazaljkama na blokove datoteke
 - jedna trostruko indirektna kazaljka
 - * kazaljka na blok sa kazaljkama na blokove sa kazaljkama na blokove sa kazaljkama na blokove datoteke



Za datoteku iz primjera 9.1., uz pretpostavku 13 kazaljki, opis smještaja prema i-node sustavu: u opisniku:

| | | | | | | | | | | | | | |
|---------|----|----|----|----|----|----|----|----|----|----|---|---|---|
| opisnik | 37 | 38 | 18 | 19 | 20 | 57 | 58 | 59 | 60 | 61 | X | - | - |
|---------|----|----|----|----|----|----|----|----|----|----|---|---|---|

u jednom bloku – bloku X:

| | | | | | |
|----|----|----|----|----|------------------------------------|
| 51 | 52 | 53 | 54 | 55 | (ostatak kazaljki je neiskorišten) |
|----|----|----|----|----|------------------------------------|

Zadatak: 9.2

Ako je na UNIX datotečnom sustavu pohranjena datoteka veličine 555 kB koliko spremničkog prostora zauzimaju kazaljke za tu datoteku? Skicirajte organizaciju tih kazaljki. Veličina bloka je 1024 okteta (1kB), a veličina kazaljke 32 bita.

Zadatak: 9.2.1

Ako je na UNIX datotečnom sustavu pohranjena datoteka veličine 3GB koliko spremničkog prostora zauzimaju kazaljke za tu datoteku? Skicirajte organizaciju tih kazaljki. Veličina bloka je 4 kB, a veličina kazaljke 64 bita.

9.3.3. FAT (informativno)

- koristi:
 - “tablicu direktorija” (engl. *directory table*) i
 - “tablicu zauzeća” (engl. *file allocation table – FAT*)
- tablica direktorija sadrži:
 - opisnike datoteka i direktorija
 - * ime datoteke
 - * ostali atributi
 - * adresa prvog bloka datoteke (u kojem se bloku particije nalazi)
- tablica zauzeća je zajednička za sve datoteke (jedna za sve)
 - tablica ima onoliko zapisa koliko ima blokova na particiji
 - svaki zapis tablice (broj, 32-bitovni broj za FAT32):
 - * pokazuje na idući blok datoteke, ili
 - * -1 ako je ovo zadnji blok, ili
 - * 0 ako je blok slobodan (ne koristi se)

Primjer 9.3. FAT

Za datoteku iz primjera 9.1. u opisniku datoteke, koji se nalazi u tablici direktorija, piše samo adresa prvog bloka = 37.

Tablica zauzeća (engl. *file allocation table*):

| | | | | | | | |
|----|----|----|----|----|----|----|--|
| | | | | | | | |
| | | | | | | | |
| | 19 | 20 | 57 | | | | |
| | | | | | | | |
| | | | | | 38 | 18 | |
| | | | | | | | |
| | | 52 | 53 | 54 | 55 | -1 | |
| 58 | 59 | 60 | 61 | 51 | | | |

9.3.4. EXT2 (informativno)

- nastao po uzoru na MINIX/UNIX uz poboljšanja
- opisnik datoteke = inode (indeksni čvor)

Globalni podaci na particiji:

- superblock - svaka particija sadrži po jedan opisnik particije
- blokovi su grupirani u "grupe blokova" (iste veličine)
 - jednu grupu čine "susjedni" blokovi
- tablica sa opisom svih grupa blokova

Svaka grupa blokova ima:

- opisnik grupe - gdje su smješteni idući elementi (ovi ispod)
- bitmapa za opis zauzetih i slobodnih blokova
- bitmapa za opis zauzetih i slobodnih inode-a
- inode tablica za ovu grupu (tablica opisnika datoteka, direktorija, ...)
- kopija superblock-a - radi mogućnosti oporavka od kvara (ne moraju sve grupe ovo imati)
- blokovi za podatke datoteka (i dodatne opisnike)

Direktoriji:

- opisnik direktorija je također inode
- sadržaj blokova na koje inode (direktorija) pokazuju su parovi {inode, ime} koji predstavljaju sadržaj direktorija (tj. datoteke i direktorije u njemu)

Ideja grupiranja blokova u grupe:

- smanjiti problem fragmentacije držanjem povezanih podataka zajedno
- pokušati držati sadržaje jedne datoteke unutar grupe == blizu!
- sadržaj direktorija u jednoj grupi

Primjer za disk od 47 GB ($\approx 12 \cdot 10^6$ blokova): (`dumpe2fs -h /dev/*particija*`)

- vel. bloka = 4 KB
- vel. grupe = 32768 blokova
- broj inode-ova po grapi = 8192
- broj blokova za inode-ove = 512

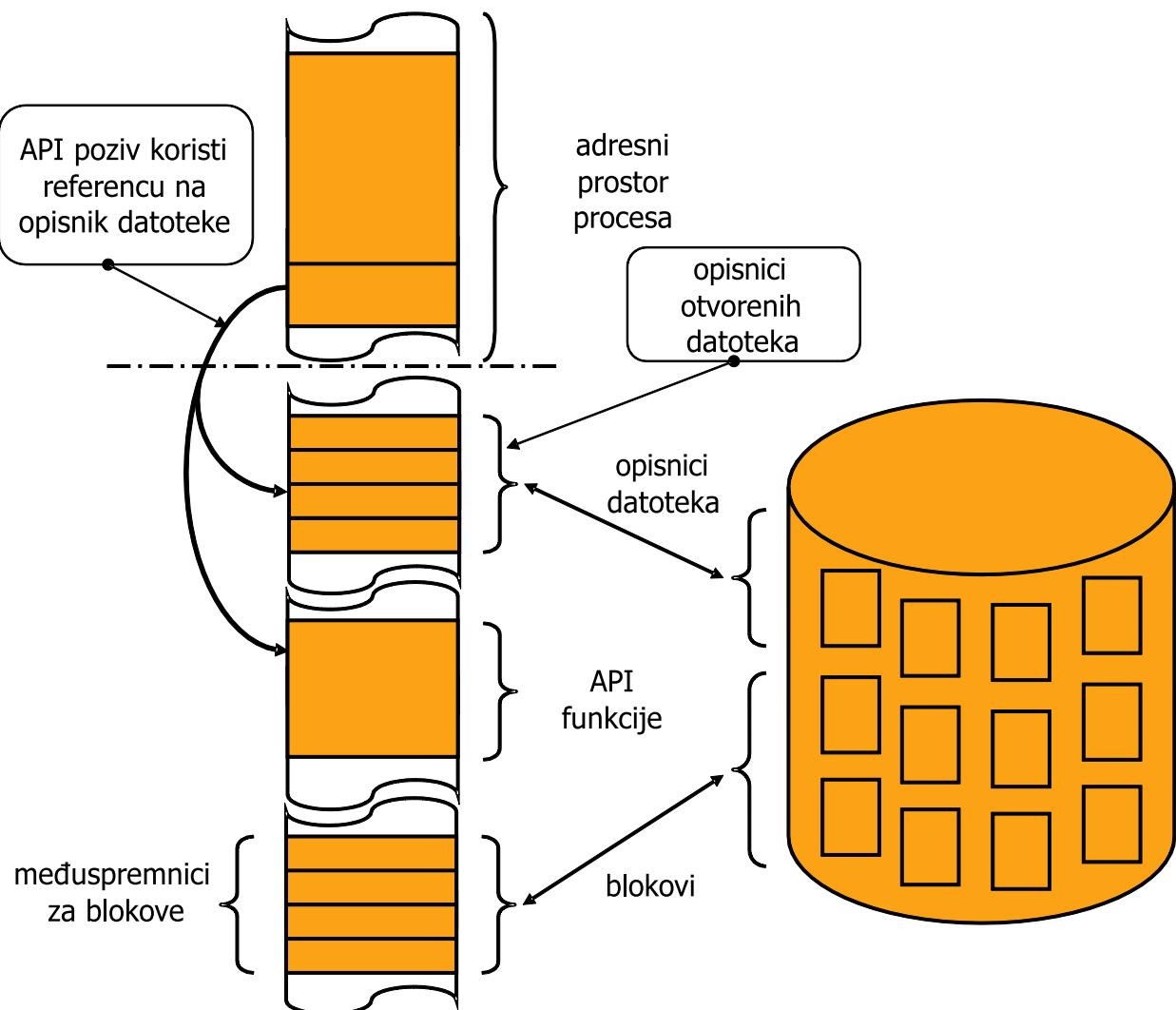
Dohvat sadržaja koje inode opisuje:

- 12 izravnih kazaljki + 3 indirektine (jednostruko, dvostruko i trostruko)

Noviji ext3 i ext4 su slični, barem po ovim opisanim svojstvima

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;
}
```

9.4.1. Jezgrine funkcije datotečna podsustava (informativno)

Načelno ostvarenje operacija datotečnog podsustava kroz jezgrine funkcije skicirano je u knjizi

- Stvoriti_datoteku (ime, atributi)
- Otvoriti_datoteku (ime, način_pristupa, id)
- Čitati_datoteku (id, logička_adresa, broj_bajtova)
- Stvoriti_datoteku (ime, atributi)
- pored navedenih potrebne su i operacije za zatvaranje datoteke, brisanje, premještanje, preimenovanje, pomicanje unutar datoteke, promjena veličine i slično

U načelnom rješenju su zanemarini neki problemi:

- korištenje podataka s diska traži:
 - započinjanje operacije
 - dovršavanje operacije nakon prekida naprave
- za obavljanje jedne operacije dohvata/pohrane može biti potrebno obaviti više operacija s diskom!
- u istom računalu može biti više uređaja s datotečnim sustavima
 - različiti uređaji (trebaju različite upravljačke programe)
 - različiti tipovi datotečnih sustava (npr. NTFS, FAT, UDF, ...)
- navedeno povećava složenost izvedbe takvih operacija
- zato se i sam datotečni podsustav najčešće ostvaruje u slojevima

9.5. Primjer slojevitog datotečnog podsustava (info)

- datotečni podsustav je složen
- zato se ostvaruje u slojevima
- podsjetnik na slojeve računalnog sustava
 - korisnik \iff programi \iff API \iff jezgra OS-a \iff sklopolje
- API sučelje prema programima
 - prilagodba jezgrinih funkcija
 - dodatno korištenje međuspremnika radi ubrzanja
 - neće se razmatrati ovdje (u sklopu dat. sust.) – razmatra se samo jezgra
- slojevi u jezgri i njihova svrha
 - VFS \iff LFS \iff BDEV \iff DEV
 - npr. pogledati mapu Linux jezgre na slici 1.5. na stranici 4 (pod *storage stupcem*)
- VFS – virtualni datotečni sustav
 - ostvaruje sučelje prema programima (ili API podsloju)
 - OS može koristiti više različitih datotečnih sustava, npr.
 - * na disku imamo NTFS ili ext2 ili slično
 - * USB štapić je možda FAT32
 - * DVD je u UDF formatu
 - * mrežnom disku pristupamo preko Samba-e ili NFS-a ili ?
 - * ...
 - kad se pozove `otvori_datoteku` iz programa to sučelje mora moći otvoriti datoteku bez obzira gdje se ona nalazila
 - * zato takvo sučelje (na vrhu) je "virtualno", "u ostvarenju" mora koristiti neko stvarno (npr. NTFS)
- LFS – logički datotečni sustav
 - "upravljački programi" koji znaju raditi s "konkretnim" tipom datotečnog sustava
 - * tu su npr. FAT32, NTFS, ISO*, UDF, Samba, ...
 - * "znaju" kako su podaci zapisani u opisniku, gdje su opisnici i sl.
 - isti tip datotečnog sustava može biti na raznim medijima
 - * npr. NTFS može biti i na disku i na USB štapiću i ...
 - * za svaki različiti uređaj potreban je različiti upravljački program
 - naprave koje služe za pohranu datotečnih sustava su spore (u usporedbi sa spremnikom)
 - * rad s njima svodi se na:
 - započinjanje operacije
 - dojava kraja operacije preko mehanizma prekida
 - * stoga bi operacije nad datotečnim sustavom bile slične opisanim UI operacijama (u 5. poglavljju)
 - * međutim, za jednu operaciju nad datotečnim sustavom često treba više UI operacija
 - primjerice: dohvati dijela datoteke (jedan poziv `read` ili `write`) može tražiti podatke iz više blokova datoteke
 - zahtjev jednog `read-a` može biti za više blokova
 - različite dretve (procesi) mogu imati zahtjeve nad istim diskom
 - posluživanje zahtjeva nije (općenito) FIFO
 - pristup: "blokiraj dretvu nad napravom + odblokiraj prvu u obradi prekida" ovdje nije dovoljan ni prikidan

- * jedan od načina rješavanja ovakva problema jest da se jezgrina dretva (koja izvodi jezgrinu funkciju) i sama tretira kao dretva (vezana uz dretvu koja je pozvala jezgrinu funkciju) te da se i ona može blokirati
 - poslije dovršetka prve operacije i odblokiranja takve dretve ona može započeti i drugu UI operaciju ...
 - za to je potreban i poseban kontekst takve dretve - jezgrin kontekst dretve
 - opisnik obične dretve tada treba proširiti opisnikom jezgrine dretve povezane s njom
 - o takvoj dretvi više u samom primjeru koda
- pojedini upravljački program LFS-a ne koristi izravno upravljački program naprave (npr. diska) već se koristi međusloj – blok naprave
- BDEV – blok naprave
 - posebnost naprava koje služe za pohranu datotečnih sustava jest što je jedinica podataka "blok"
 - * blok je veličine 512 B, 1 KB, 2 KB, 4 KB, ...
 - * operacije koje se traže nad takvim napravama su:
 - pročitaj blok
 - zapiši blok
 - * operacije nisu (općenito) odmah gotove!
 - naprava završetak zadane operacije javlja prekidom
 - u obradi prekida rade se dodatni poslovi
 - zato se uvodi dodatni sloj sloj "blok naprava" s dvije osnovne operacije:
 - * dohvati_blok te pohrani_blok
 - * dio tih operacija izvodi se po prihvatu prekida tih naprava
 - * zato je uz gornja dva sučelja potrebno i obradi_prekid_blok_naprave
 - blok naprava kao sloj može biti neovisna o uređaju i tipu datotečnog sustava
 - * s jedne strane koristi upravljački program naprave (preko sučelja)
 - * s druge strane daje sučelje prema LFS-u
 - zašto navedene operacije (ovo što sučelje "blok naprava" pruža) nisu izravno uključene u upravljački program naprave?
 - * da se iste operacije ne duplicitiraju u svakom upravljačkom programu
 - * da se smanji složenost upravljačkih programa
- DEV – sloj upravljačkog programa naprave (npr. diska)
 - "zna" komunicirati s napravom
 - * slati podatke
 - * čitati podatke
 - * dohvatiti status naprave i informacije o zadnjim posluženim zahtjevima
- primjer/skica ostvarenja u "dat-sust" (na webu)

9.6. 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 \Leftrightarrow međuspremnik] \Leftrightarrow [spremnik] \Leftrightarrow [procesor: L3 \Leftrightarrow L2 \Leftrightarrow L1 \Leftrightarrow 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

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 sektor (kako se sektor jedinstveno identificira)?
4. Što je to "cilindar"?
5. Koje operacije obavlja upravljački sklop diskovne jedinice?
6. Od kojih se komponenata sastoji vrijeme čitanja jednog sektora?
7. Opisati pojmove: vrijeme postavljanja, (prosječno) rotacijsko kašnjenje, faktor preplitanja.
8. Ako je brzina prijenosa podataka zadana s 100 Mb/s (ili Mbita/s) koliko je to b/s (bita/s)?
9. Opišite strategije posluživanja: FCFS, SSTF, SCAN, LOOK, C-SCAN, C-LOOK.
10. Što je to "datoteka"?
11. Što je to datotečni sustav? Što sadrži?
12. Navesti elemente opisnika datoteke.
13. Opisati kako se opisuje smještaj datoteke na NTFS te UNIX i-node sustavima.