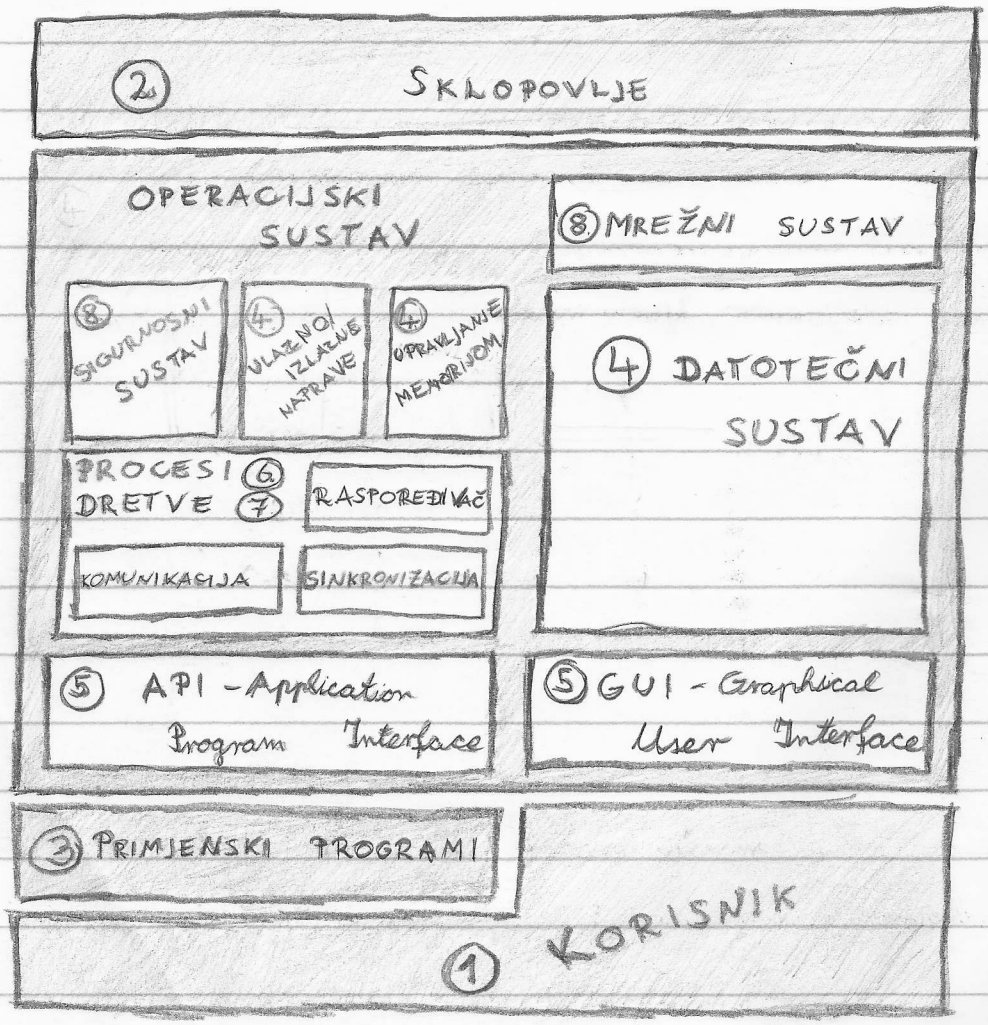
OPERACIJSKI SUSTAV

⇒ skup programa koji omogućavaju obavljanje raznih zadataka na računalu

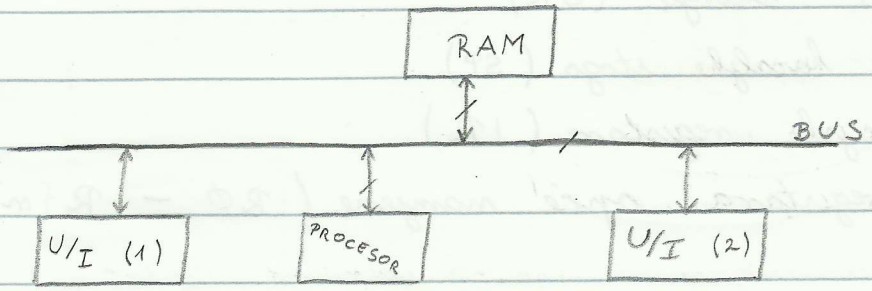
⇒ zadaci operacijskog sustava:
- ↳ višeprogramski rad
- ↳ suptilnost / prikrivenost
- ↳ jednostavnost rada i efikasno korištenje hardvera-a

⇒ bitno svojstvo i funkcionalnost svakog operacijskog sustava je rad s datotekama

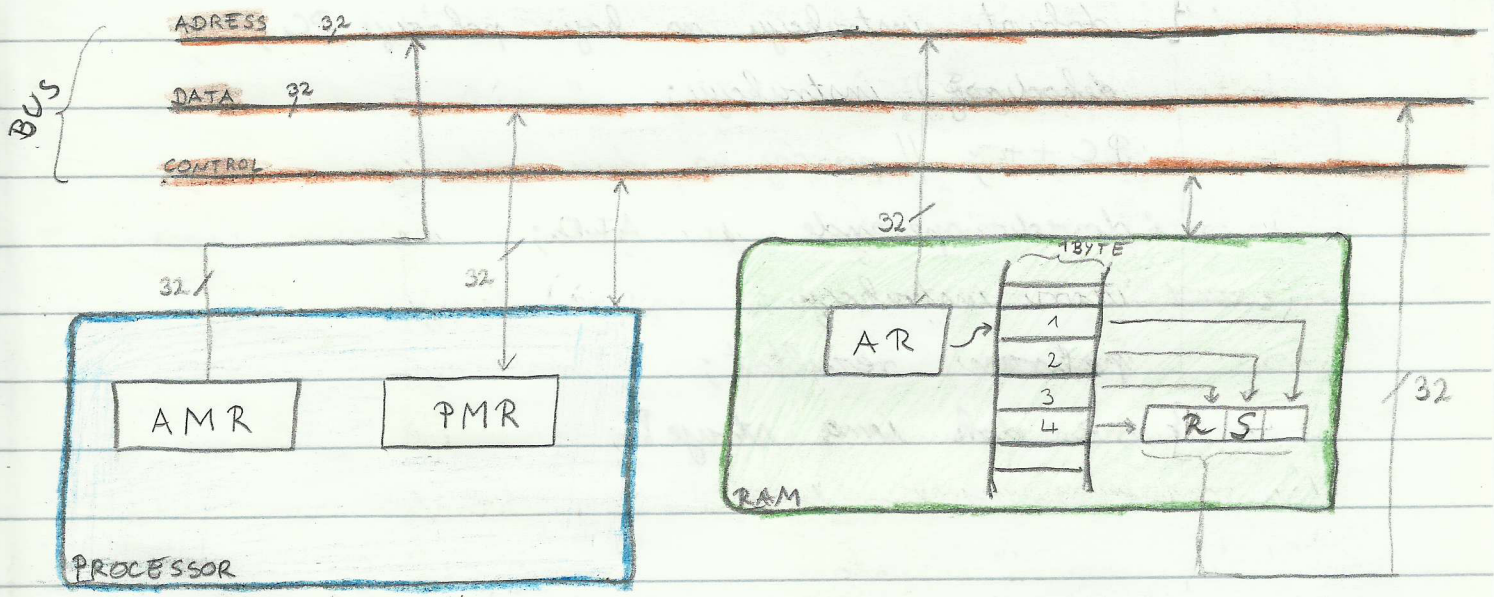


MODEL JEDNOSTAVNOG RAČUNALA

⇒ Von Neumannov model računala:



⇒ rudimentarno računalo - dalje pojednostavljeni model računala koj sadrži samo procesor i spremnik (RAM)



PMR - podatkovni međuregistar

AMR - adresni međuregistar

RS - registar sadržaja

AR - adresni registar

⇒ VRSTE REGISTARA U PROCESORU:

a) programsko brojilo (PC)
b) adresni međuregistar (AMR)
c) podatkovni međuregistar (PMR)
d) registar stanja (SR)
e) registar kazaljke stoga (SP)
f) instrukcijski registar (IR)
g) skup registara opće namjene (R0 – R[n])

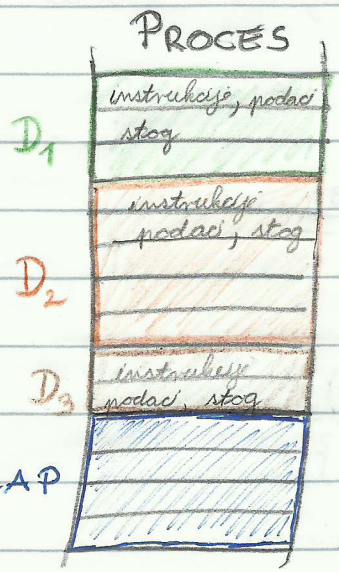
⇒ neke programske rutine ćemo morati programirati u asembleru

⇒ ŠTO RADI PROCESOR:

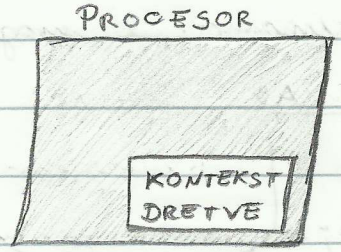
```
ponavljaj
  { dohvati instrukciju na koju pokazuje PC;
    dekodiraj instrukciju;
    PC++; // povećaj na iduću instrukciju
    dovedi operande u ALU;
    izvrši instrukciju;
    pohrani rezultat;
  } sve dok ima struje!;
```

⇒ PROGRAM - statični niz instrukcija (npr. napisan na papiru)
⇒ DRETVA - program u izvođenju
⇒ PROCES - skup resursa (memorija, datoteke, ...) da bi program mogao raditi

⇒ proces se sastoj od najmanje jedne dretve



} zajednički adresni prostor



⇒ procesor može izvršavati samo jednu dretvu u određenom trenutku

⇒ procesor za svaku dretvu mora znati gdje su:

a) instrukcije

b) podaci

c) stog

⇒ kontekst dretve - sadržaj svih registara dretve

- potrebno procesoru kako bi znao nastaviti od kraja posljednjeg izvođenja dretve

⇒ DRETVA (druga definicija) - niz instrukcija koje procesor izvodi

- ona se događa u vremenu

⇒ VIŠEZADAĆNOST (multitasking) - više niova instrukcija se želi izvršavat odjednom

⇒ višedretveni rad se može provest i na jednoprocesorskom računalu na način da procesor naizmjence izvodi sve dretve

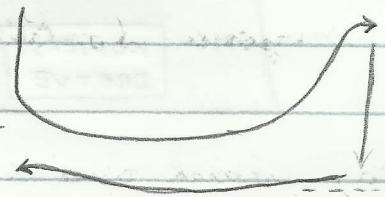
↳ npr. imamo 2 programa:

DRETVA A:

```
a = 3*3;
z = a - 4;
-----------
z = a++;
a = a--;
```

DRETVA B:

```
f = 2;
e = 8;
if ( f == 2)
{ e++; }
e++;
```



⇒ KONTEKST DRETVE:

↳ prilikom prekida, kontekst dretve se mora spremit negdje u memoriji kako bi se po povratku dretva nastavila izvodit

↳ to izmjenjivanje se naziva „kućanski poslovi" (housekeeping)

↳ time se postiže višeprogramski rad

↳ onaj program koji nešto čeka, miče se s procesora i uzima se onaj kontekst koji je bitniji i koji nešto mora radit

↳ ako se izmjena konteksta dretve vrši pravilno, svaka dretva ima privid kao da ima svoj vlastiti procesor

↳ izmjena konteksta se u modernim operacijskim sustavima vrši svakih 20 ms (otprilike)

ULAZNO-IZLAZNE OPERACIJE I PREKIDNI RAD

⇒ kako bi OS mogao prekinuti neki program, procesor mora imati mogućnost prekida

↳ isključivo prekid izaziva izmjenu konteksta

⇒ prekidi su npr. pomak miša, pritisak tipke, ...

⇒ na adresama kamo procesor skače prilikom prekida nalazi se kod operacijskog sustava

↳ operacijski sustav aktivira se samo prilikom prekida

↳ na adrese operacijskog sustava, korisnik nema pristupa (u normalnom „user" modu)

⇒ ULAZNO-IZLAZNE NAPRAVE:

↳ brzine ulazno-izlaznih naprava su različite, pa nam treba sinkronizacija s procesorom

↳ svaka U/I naprava ima svoj UPRAVLJAČKI SKLOP

↳ U/I naprave se na sabirnicu spajaju preko prestupnog sklopa (npr. USB) - on se prilagođen i sabirnici i ulazno-izlaznoj napravi

↳ PRIJENOS PODATAKA:

a) pojedinačno („byte po byte")

b) skupno

⇒ POJEDINAČNI PRIJENOS:

↳ nama beruvjetni način rada nije bitan, te demo proučavati uvjetni i prekidni način rada

↳ struktura PRISTUPNOG SKLOPA:

a) podatkovni registar

b) registar stanja

c) upravljanje

↳ koraci prijenosa:

1) prvo iz polazišta u registar procesora

2) iz registra procesora na odredište

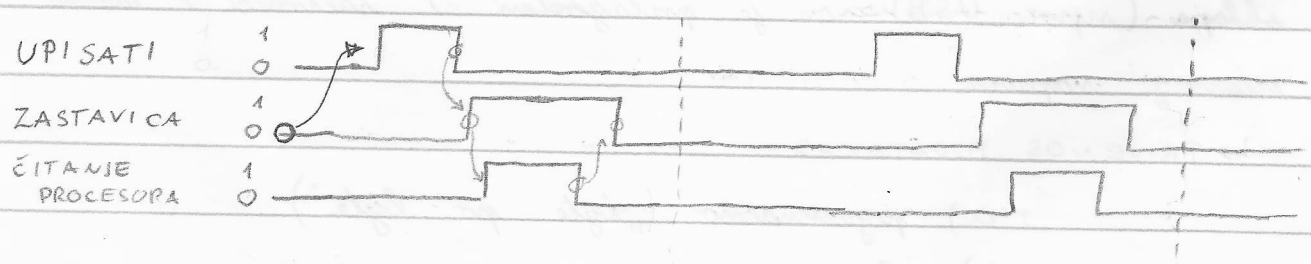
⇒ <u>prijenos znakova radnim čekanjem:</u>

↳ uvjetni prijenos

↳ procesor mora čekat dok naprava nije spremna za prijenos - to je prijenos prema napravi (provjerava se registar stanja za spremnost - IZLAZNA ZASTAVICA)

↳ procesor mora čekat dok naprava nije postavila podatak - to je prijenos iz naprave - ULAZNA ZASTAVICA (također u registru stanja (RS))

↳ ovakva sinkronizacija se može realizirati SR-bistabilom i pomoću dvije žice, pa se naziva DVOŽIČNO RUKOVANJE („two-wire handshake")



⇒ za realizaciju sinkronizacije, nema drugog načina nego da procesor stalno čita zastavicu i provjerava ulazno stanje (spremnost) - procesor se vrti u petlji !

↳ to se stoga naziva RADNO ČEKANJE

↳ ovaj način rada ima velik nedostatak, a to je beskorisno trošenje procesorskog vremena

↳ na ovakav način mi nećemo realizirat gotova rješenja !

⇒ prijenos znakova prekidom:

↳ prekidni prijenos

↳ PREKIDNI SIGNAL - naprava šalje signal, prebacuje procesor u sustavski način rada - poziva se jezgra procesora (SYSTEM MODE, KERNEL MODE, ...)

↳ do prekidnog signala, radna dretva se izvršavala u korisničkom načinu rada (USER MODE)

↳ NAPOMENA: barem dva načina rada su uvjet za naš (i današnji) operacijski sustav

⇒ prijelaz u sustavski način rada:

1) onemogućavanje prekida

2) adresira se sustavski dio memorije (memorija također ima sustavski dio)

3) adresira se sustavski stog

4) PC, RS se spremaju na sustavski stog

5) u PC se upisuje adresa prekidnog potprograma (definirana samom arhitekturom procesora)

⇒ ZAŠTITA OS-a: sustavski dio spremnika je nedostupan korisničkim (user) dretvama

↳ prva operacija nakon prekida je pohrana konteksta na sustavski stog (za nas će to biti instrukcija „POHRANI KONTEKST")

↳ druga operacija je „OBRADI PREKID" - poslužuje se prekidna jedinica (il nešto drugo ...)

↳ treća operacija je „ZAVRŠETAK OBRADE" - pošaljemo sve podatke i javimo kraj

↳ četvrto je: nastavak prekinute dretve, ali prvo treba „OBNOVI KONTEKST" (obnavlja sve osim PC i RS) te slijedi instrukcija: „VRATI SE U PREKINUTU DRETVU" (aktivira način rada i adresni prostor prekinute dretve, a potom obnavlja PC i RS)

PREKIDNI PODSUSTAV

⇒ ako imamo više U/I naprava, lako je moguće da prekidi mogu doći od više naprava

↳ jedno rješenje: SAMO 1 PREKIDNI SIGNAL (radi se pomoću jednog signala spojenih koji je rezultat ILI operacije svih naprava)

⇒ nadalje, prilikom prekida:

1) PROZIVANJE (POLLING)

↳ ispitivanje registra stanja svakog sklopa **ispitnim lancem** - ispituju se po prioritetu

2) POSLUŽIVANJE

↳ <u>nedostatak</u>: nema prekidanja obrade prekida (npr. ako hard-disk ima aktivni prekid, miš će nam biti blokiran jer ne može izazvat prekid)

↳ imamo samo jednu riječ, pa trebamo programsko rješenje

⇒ PODSUSTAV S NAJJEDNOSTAVNIJIM SKLOPOVLJEM:

↳ najjednostavnije sklopovlje znači da imamo samo jednu riječ za prekid

↳ <u>željeno ponašanje</u>:

1) kućanski poslovi - neprekidivi
2) obrada prekida - prekidiva
3) prekid se prihvaća ako je važniji od obrade trenutnog
4) po završetku obrade, obrađuju se prekidi na čekanju ili se pokreće korisnička dretva

↳ **realizacija:**

T_P ⇒ tekući prioritet

K_Z[N] ⇒ polje „kopija-zastavica"

KON[N] ⇒ mj. zapisa konteksta

↳ **nedostatak:**

↳ puno kućanskih poslova - za svaku pojavu prekida spremamo i obnavljamo kontekst (ovo se može riješiti softverski već samo sklopovski)

⇒ PODSUSTAV SA SKLOPOM ZA PRIHVAT PREKIDA:

↳ prekidni signal od naprava dolazi na sklop za prihvat prekida

↳ taj sklop propušta procesoru samo prekide višeg prioriteta

↳ procesor mora imati i signal za prihvat prekida kojem javlja sklopu da se može prihvatiti prekid

↳ **realizacija sklopa:**

registar T_P ⇒ tekući prioritet

registar K_Z ⇒ polje „kopija-zastavice"

ADRESE prekidnih potprograma

↳ **sklop radi:**

- propušta prekid razine veće od T_P
- briše zastavicu u K_Z
- na temelju dobivenog signala IACK (PRIHVAT) stavlja adresu prekidnog programa na sabirnicu

↳ promjene u procesoru:

a) prilikom prihvaćanja prekida postavlja signal PRIHVAT (IACK)

b) pohrana konteksta može postati dio procesora (ovo nije nužno)

↳ obrada prekida razine I:

```
pohrani T_P na sustavski stog;
T_P[I] = 1;
omogući prekidanje;
obrada prekida I-te razine;
onemogući prekide;
obnovi sa stoga registar T_P u prekidni sklop;
obnovi kontekst sa stoga;
vrati se u prekinuto dretvu;
```

⇒ NAPOMENA: bolje od sklopovskog prihvata prekida ne možemo - sama mala poboljšanja su prisutna u današnjim procesorima

PREKIDI UNUTAR PROCESORA

⇒ primjer kada nam je ovo potrebno je kada procesor pročita neku „neispravnu instrukciju", „neispravnu adresu" ili dijeljenje s nulom (ima još slučajeva)

⇒ danas, sklop za prihvat prekida može biti dio procesora

⇒ prekid mogu biti izazvani i od strane drivera

↳ to je kada npr. želimo nešto ispisati na ekranu, želimo dodatnu memoriju

⇒ POZIV JEZGRE OS-a:

↳ unutar programa, funkcije jezgre OS-a pozivaju se programskim prekidom (SOFTWARE INTERRUPT)

↳ svaki prekid (i jedino prekid) aktivira jezgru OS-a

ZAD: U sustavu se pojavljuju prekidi P1 : 3ms, P2 : 1ms, P3 : 4ms. Prioritet je odreden brojem (P3 je najveći), a obrada svakog prekida traje 3ms. Grafički prikaži aktivnost procesora u glavnom programu (GP), prekidu (P_i) te procedurama za prihvat (PP) i procedurama za povrat (P_iP) iz prekida i to:

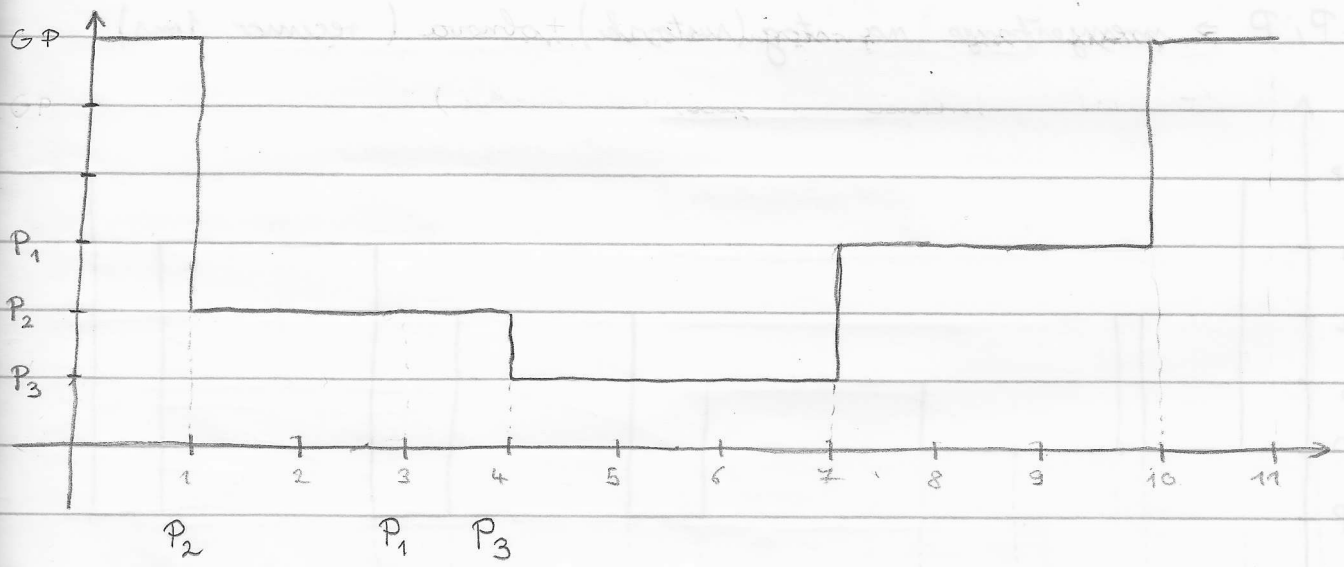
a) u idealnom slučaju

b) bez sklopa za prihvat prekida (obrada uz zabranjene prekidove)

c) bez sklopa, ali uz programsku potporu

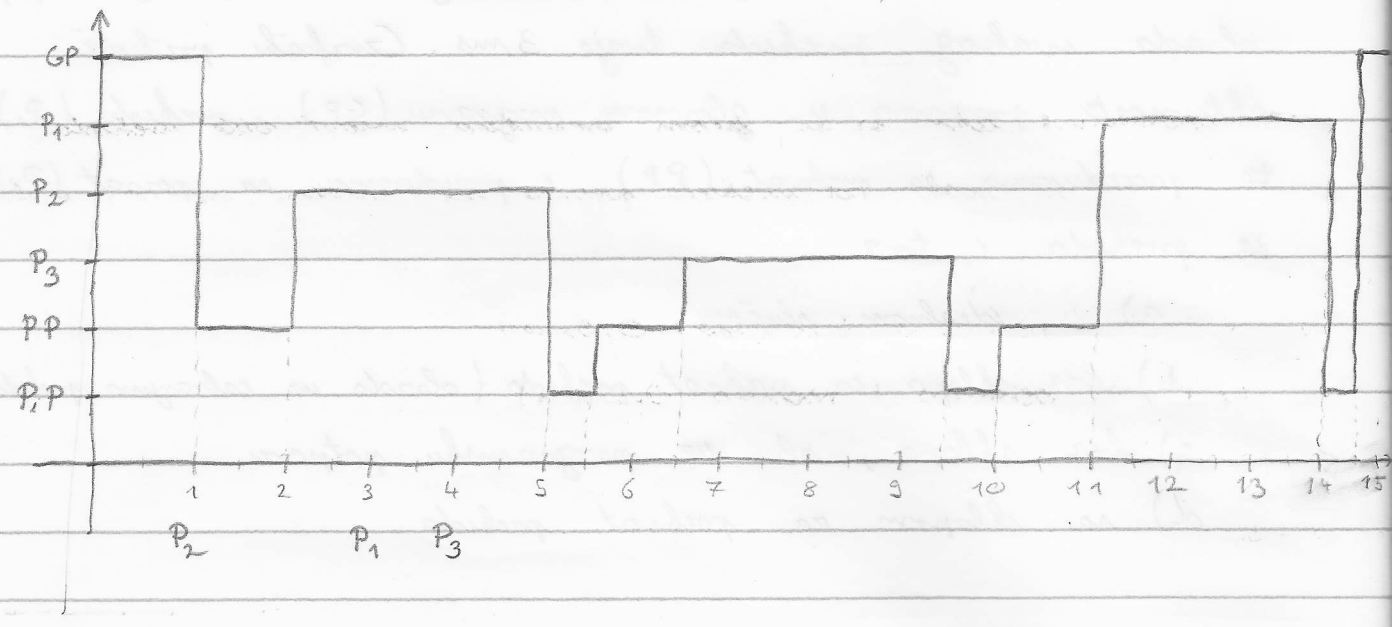
d) sa sklopom za prihvat prekida

a) Prekidi se obrađuju prema prioritetu - zanemaruje se trajanje kućanskih poslova



b) PP ⇒ pohrana konteksta i prosvanje (recimo 1 ms).

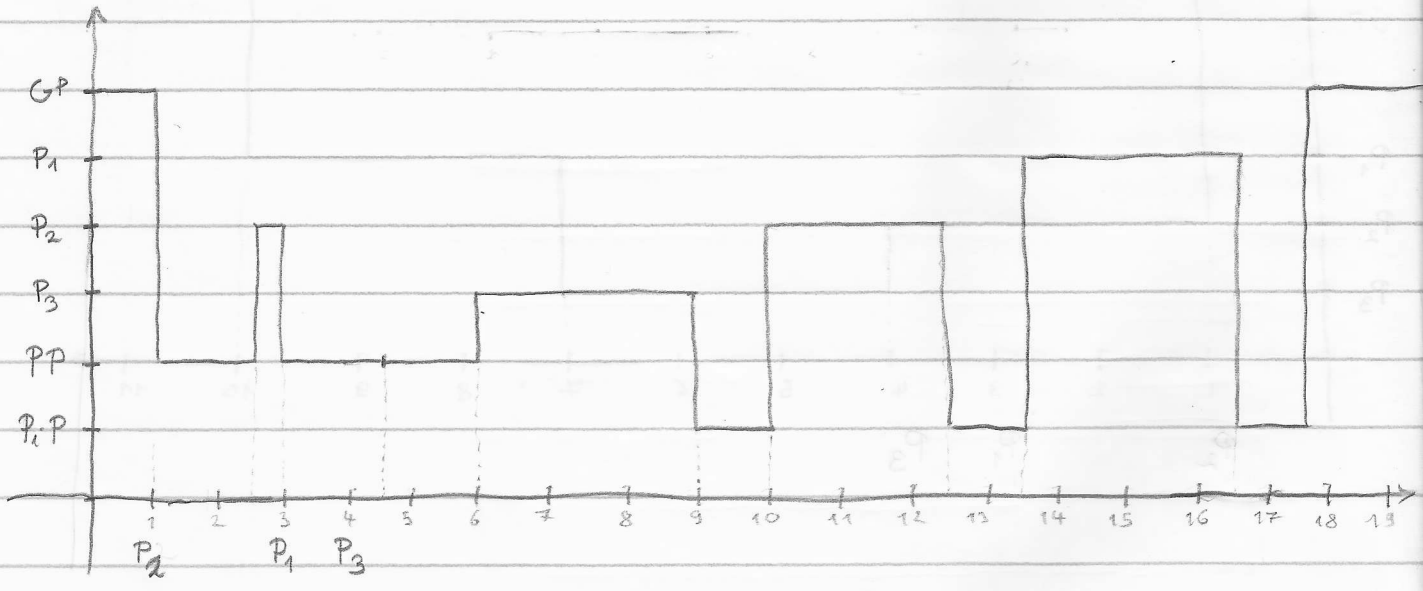
P_iP ⇒ obnova konteksta (recimo 0.5 ms)



c) Obrada prekida je sada prekidiva

PP ⇒ pohrana + prosvanje + prenyeštanje u KON(J) (recimo 1.5 ms)

P_iP ⇒ prenyeštanje na stog (sustavski) + obnova (recimo 1 ms)



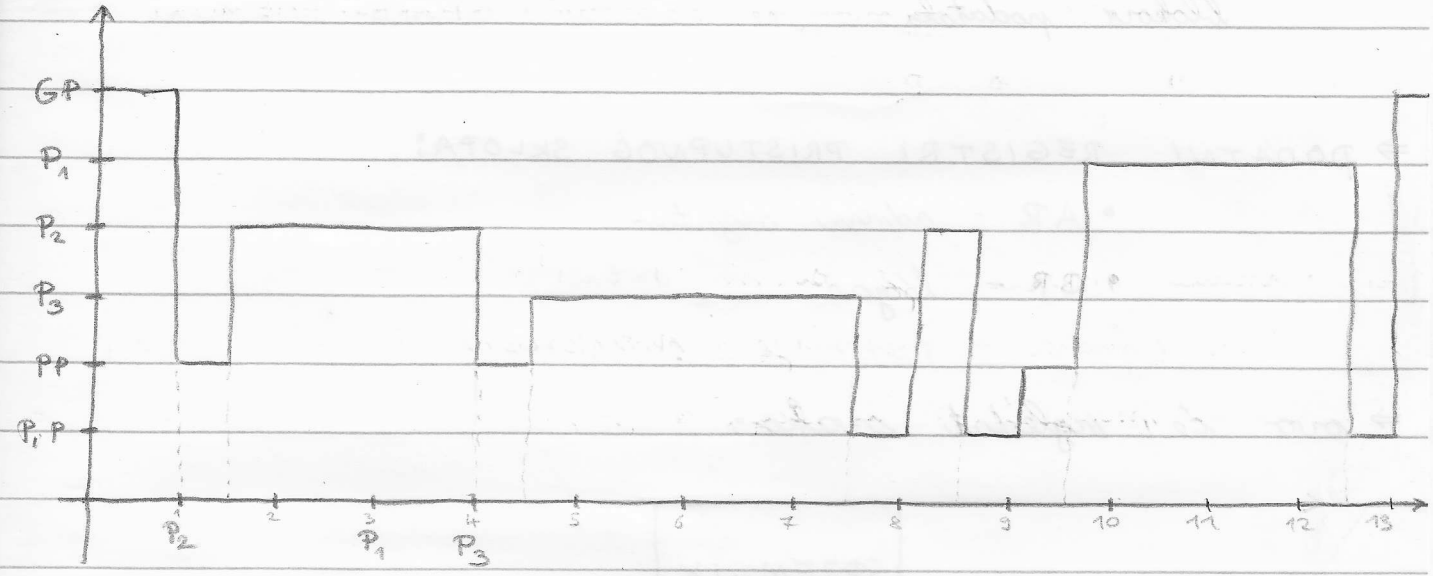
↳ napomena: ako u istom trenutku obrada razvoja i javlja se prekid, pretpostavljamo da se najprije razvrša

obrada, a tek onda prekidni signal ima utjecaj (stoga, prvo se „vratimo" iz prekida, a tek onda idemo u novi prekid)

↳ napomena: ako se u istom trenutku pojavi više prekida na svaki od prekida potrebno se posebno prihvat prekida

d) PP ⇒ pohrana (recimo 0.5 ms)

P1 P ⇒ obnova (recimo 0.5 ms)



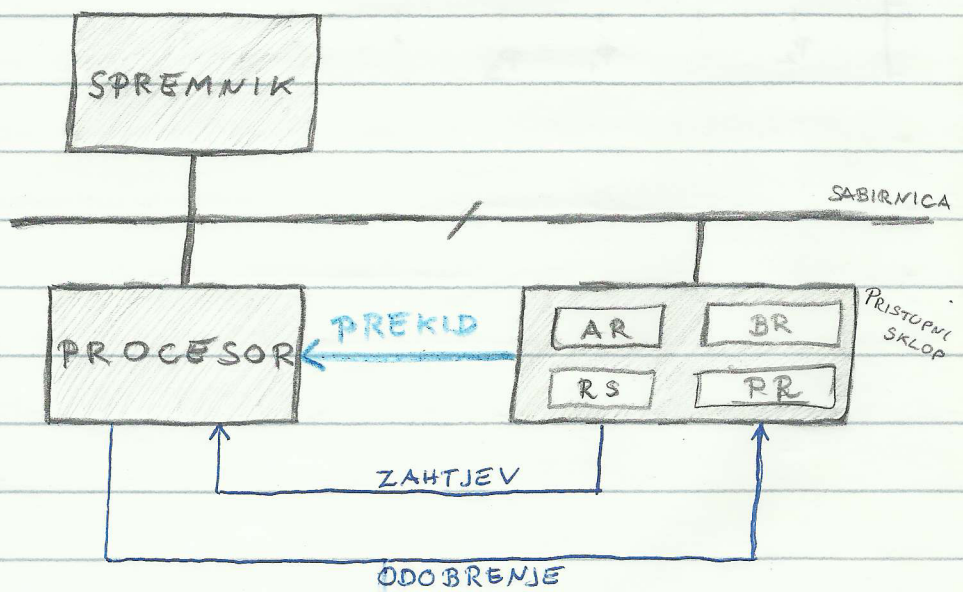
NEPOSREDNI PRISTUP SPREMNIKU

⇒ english naziv: direct memory access (DMA)

⇒ ideja je da pristupni sklop olakša prijenos podataka kako bi se procesor rasteretio jednog jednostavnog posla
↳ ovo je osobito korisno kada se prenose veliki blokovi podataka

⇒ DODATNI REGISTRI PRISTUPNOG SKLOPA:
- AR - adresni registar
- BR - brojač

⇒ ovo će izgledati ovako:



⇒ IZMJENE U PROCESORU:

↳ mora podržavat odobravanje kontrole sabirnice drugom uredaju

↳ za to nam trebaju dva signala (žice):

1) zahtjev (BREQ - Bus request)

2) odobrenje (BACK - Bus acknowledged)

⇒ ovakav način prijenosa podataka je drastično ubrzanje na standardne prekidne prijenose

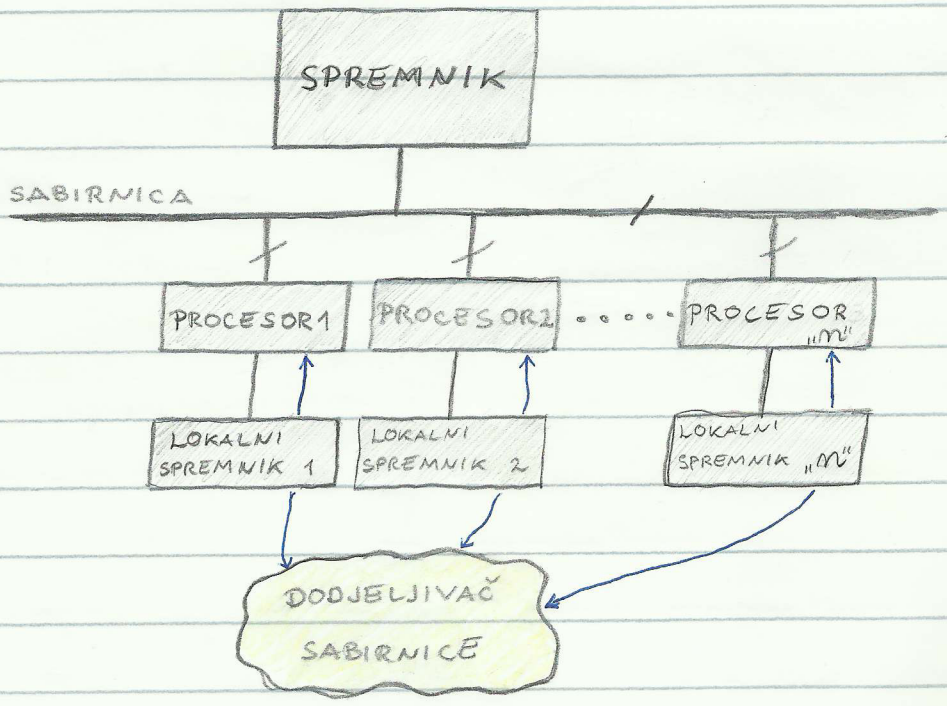
ČVRSTO POVEZANI VIŠEPROCESORSKI SUSTAV

⇒ postavlja se pitanje: kako napraviti operacijski sustav za višeprocesorsko računalo?

⇒ PRETPOSTAVKE (zbog jednostavnosti):

a) više procesora, ali zajednički spremnik i sabirnica
↳ ovo je nazvano SHARED MEMORY

b) struktura podataka OS-a je smještena cijela u zajedničkom spremniku



⇒ za ovakav rad je nužan dodjeljivač sabirnice (BUS ARBITER)
↳ dodjela sabirnice vrši se ciklički.

⇒ HOMOGENI SUSTAV: svi procesori su jednaki (ravnopravni)

↳ posljedica toga je da se bilo koja dretva može izvodit na bilo kojem procesoru

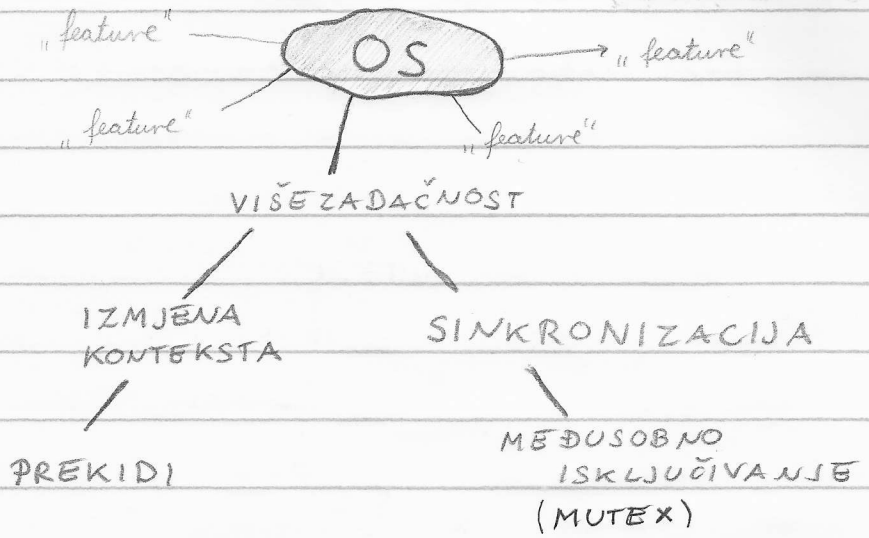
↳ napomena: nastoj se dretvu vratit na procesor na kojem se je već izvršavala kako ne bi premještal podatke iz CACHE-memoriji tog procesora na drugi procesor

MEĐUSOBNO ISKLJUČIVANJE

⇒ osnovna funkcionalnost koju želimo u našem OS-u je **višezadaćnost** - za to su nam bile potrebne izmjena konteksta i prekid

↳ da bi višezadaćnost bila moguća, treba nam **sinkronizacija** (dretvi i procesa)

↳ međusobno isključivanje je osnovni problem sinkronizacije



⇒ PROCES ⇒ skup računalnih sredstava koji omogućuju izvođenje programa

⇒ nastaje pokretanjem programa

⇒ u početku se odvija / izvodi kao jedna, ali kasnije može i kao više dretvi

↳ razlaganje na više dretvi se danas koristi za ubrzanje, ali se i prije to koristilo jer je program semantički jasniji i čitljiviji

↳ stoga, program je lakše napisati u više dretvi

⇒ unutar procesa stoga imamo više dretvi i one dijele sva sredstva koje je OS dodijelio tom procesu

⇒ različiti procesi su potpuno odvojeni jedan od drugoga te jedan na drugoga ne mogu utjecati

↳ to je namjerna zaštitna mjera OS-a

↳ komunikacija između procesa moguća je jedino uz pomoć OS-a (npr. komunikacija preko datoteke)

VIŠEDRETVENI PROGRAMSKI MODEL

⇒ primjer podjele rada:

↳ na tri dretve: ulazna, radna, izlazna

⇒ dretve se mogu izvoditi istodobno, ali je potrebna sinkronizacija

↳ npr. ulazna dretva ne smije poslati novi podatak radnoj dretvi prije nego je radnj podatak bio preuzet od strane radne dretve

⇒ MEĐUOVISNOST DRETVI (radataka):

↳ dretveni adresni prostor (zasebno za svaku dretvu):

- dio za instrukcije
- stog
- lokalni podaci

↳ procesni adresni prostor:

- dio za dretvene prostore
- zajednički dio (npr. globalne varijable)

↳ zbog ovog dijela je potrebna sinkronizacija

⇒ sinkronizaciju dretvi mora omogućiti operacijski sustav

⇒ **UVJET NEZAVISNOSTI DRETVI:**

↳ svaka dretva D_i ima domenu X_i i kodomenu Y_i

↳ imamo dvije dretve:

„D_i" i „D_j"

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

↳ „ako postoji ikakvo preklapanje u parovima dviju dretvi, one su ravisne"

↳ ako su dretve neravisne, svejedno je koja se izvodi prije a koja poslije te da li se one izvode istodobno

↳ ravisne dretve moraju imati uredan redoslijed

⇒ **SUSTAV DRETVI:**

↳ za opis redoslijeda izvršavanja dretvi koristi se **usmjeren graf**

↳ dretve na istom putu su međusobno ravisne

↳ CIKLIČKE DRETVE – dretve koje se trajno ponavljaju

↳ analogno definiramo ciklički sustav dretvi

⇒ **UPRAVLJANJE DRETVAMA (OS mora ponuditi):**

- pokretanje
- raustavljanje
- sinkronizacija (redoslijed, više dretvi odjednom)
- razmjena podataka (globalne varijable, redovi poruka)

ZAD: Sustav radataka je radan u obliku lanca:

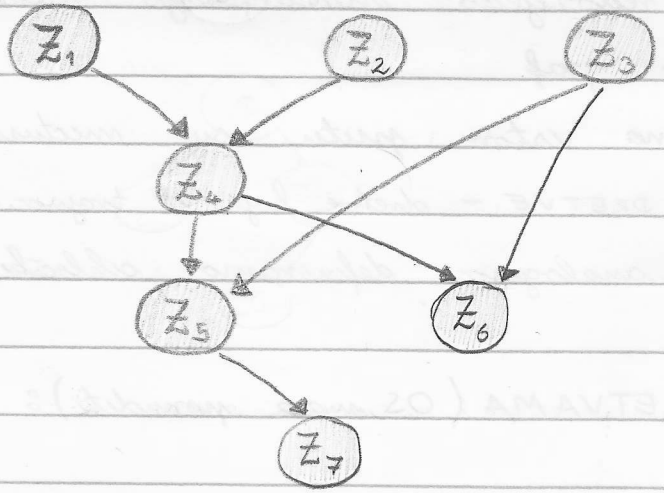
$$Z_1 \rightarrow Z_2 \rightarrow Z_3 \rightarrow \dots \rightarrow Z_7$$

Zadaci imaju domene i kodomene prema tablici:

	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							

Odredi maksimalno paralelni sustav radataka uzimajući u obzir njihov odnos u lancu?

NAPOMENA: rješenje traži minimalan broj međuovisnosti



PROBLEM MEĐUSOBNOG ISKLJUČIVANJA

⇒ npr. prodaja ulaznica - samo jedna blagajna u jednom trenutku smije prodavat ulaznice za odredena mjesta

⇒ KRITIČNI ODSJEČAK (Critical section):

↳ niz instrukcija u kojemu dretva pristupa nekom zajedničkom sredstvu

⇒ zajedničko sredstvo smije se koristit samo međusobno isključivo (pojedinačno)

⇒ NEKRITIČNI ODSJEČAK:

↳ sve instrukcije dretve koje nisu u kritičnom odsječku

⇒ MEĐUSOBNO ISKLJUČIVANJE:

↳ sinkronizacijski mehanizam koji onemogućava pojedinačni pristup nekom sredstvu (najčešće zajedničkom)

⇒ realizacije međusobnog isključivanja mogu bit softverska i hardverska

⇒ <u>realizacija</u>:

- jednoprocesorski sustav - zabrana prekida tijekom kritičnog odsječka
  ↳ napomena: ovo nije moguće napravit u korisničkim dretvama

• višeprocesorski sustav - ralrana prekidanja ne rješava problem

⇒ mehanizam međusobnog isključivanja je nužan za ostvarivanje višeprocesorskog operacijskog sustava

⇒ ovaj problem uočen je i rješen 1959. godine

MEĐUSOBNO ISKLJUČIVANJE

DVIJE DRETVE

⇒ naš sustav ima sljedeće:

- dvije dretve
- višeprocesorski sustav
- sklopka dodjela sabirnice

⇒ dretve su neravnopravne, al koriste razjedničko sredstvo

⇒ oblik dretve:

```
while (1) {
    kritični odsječak (K.O.);
    nekritični odsječak (N.K.O.);
}
```

⇒ UVJETI RJEŠENJA:

a) samo jedna dretva smije se nalaziti u K.O.

b) mehanizam mora djelovati i uz proizvoljne brzine dretvi

c) ostanak neke dretve u N.K.O. ne smije drugoj dretvi ometati ulazak u K.O.

d) odabir dretve koja će izvršavati K.O. mora se obaviti u konačnom vremenu

⇒ ove uvjete nije posve jednostavno zadovoljiti

⇒ PRVO RJEŠENJE:

↳ matematičar „Dekker", 1959.

↳ imamo: ZAST[2] = Ø // ZAST[0] = Ø, ZAST[1] = Ø

PRAVO = {Ø, 1} // može biti nula ili jedan

```
DOK JE (1) {
    ZAST[i] = 1;
    DOK JE (ZAST[j] != Ø) {
        AKO (PRAVO != j) {
            ZAST[I] = Ø;
            DOK JE (PRAVO != I);
            ZAST[I] = 1;
        }
    }
    K.O.;
    PRAVO = J;
    ZAST[I] = Ø;
    N.K.O.;
}
```

⇒ najbolji „stress-test" algoritama:

a) jedna dretva je u K.O., a druga čeka

b) obje dretve istovremeno čeka u K.O.

c) jedna čeka u K.O., a druga nije u K.O. nego je negdje drugdje (npr. u N.K.O.)

⇒ JEDNOSTAVNIJE RJEŠENJE:

↳ „Peterson", 1981.

↳ imamo: ZAST[2] = Φ

PRAVO = {Φ, 1}

```
DOK JE(1) {
    ZAST[i] = 1;
    PRAVO = j;
    DOK JE((PRAVO == J) && (ZAST[j] == 1));
    K.O.;
    ZAST[i] = Φ;
    N.K.O.;
}
```

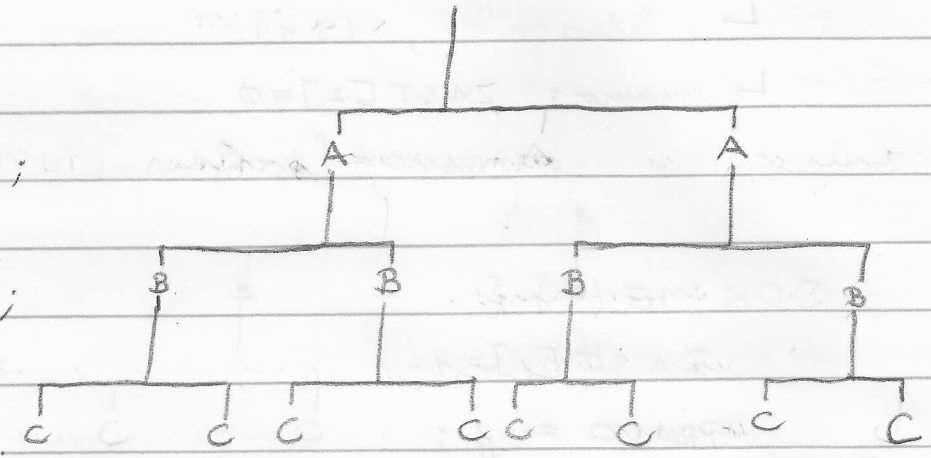
⇒ usporedba rješenja:

↳ Dekkerovo rješenje ovisi o početnoj vrijednosti varijable PRAVO

↳ Petersonovo rješenje ne ovisi o početnoj vrijednosti varijable PRAVO te daje prednost broju dretvi

PR: Imamo sljedeći odsječak:

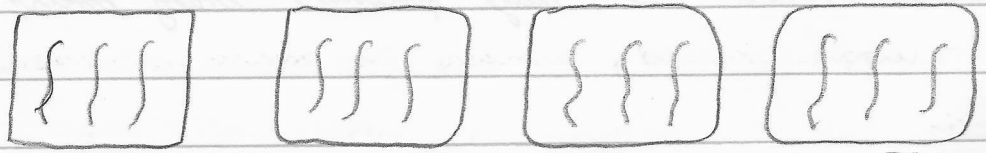
```
....
fork();
printf("A");
fork();
printf("B");
fork();
printf("C")
...
```



PR: Imamo sljedeći odsječak:

```
....
fork();
fork();
pthread_create(..., ..., fja, ...);
pthread_create(..., ..., fja, ...);
....
```

↳ koliko dretvi imamo na kraju?



MEĐUSOBNO ISKLJUČIVANJE VIŠE DRETVI

⇒ rješenje je **Lamportov protokol** (1974.)

⇒ osnovne informacije:

- niz **broj[j]**; // redni broj dolaska dretve
- **radnj_broj**; // najveći dodijeljen broj
- niz **ulaz[j]**; // rastavca koja označava da dretva uzima broj

⇒ uzimanje broja - mora biti isto za sve dretve te se radi na način:

```
ulaz[i] = 1;
broj[i] = radnj_broj + 1;
radnj_broj = broj[i];
ulaz[i] = 0;
```

⇒ u slučaju istoga broja, dretva s manjim indeksom ima prednost

↳ ovo je dogovor koj se koristi za razrješavanje rijetkih konflikata kada dretve u istom trenutku žele uzimat broj i završe s istim brojem

⇒ uvedimo i pomoćnu operaciju: USPOREDBA PAROVA

$$(a, c) < (b, d) \iff (a < b) \vee [(a == b) \wedge (c < d)]$$

SKLOPOVSKO

MEĐUSOBNO ISKLJUČIVANJE

⇒ na jednoprocesorskom sustavu, najlakša zaštita kritičnog odsječka je ZABRANA PREKIDA

⇒ <u>na višeprocesorskom sustavu:</u>

↳ UVJET: postojanje instrukcije koja obavlja dva nedjeljiva sabirnička ciklusa
(npr. možemo čitat i pisat)

PR: Atomarna naredba TAS

↳ TAS → „Test and Set"

↳ čita sadržaj memorijske lokacije i na tu istu lokaciju se upisuje 1

↳ koristi nam za provjeru možemo li ući u kritični odsječak

↳ oblik naredbe: TAS <varijabla>

↳ sa ovakvom potporom atomarnih naredbi, dovoljan nam je jedan bit za provjeru je li netko u kritičnom odsječku

ZAKLJUČAK MEĐUSOBNOG ISKLJUČIVANJA

⇒ dretva koja čeka na k. O., treba se makmut s procesora

⇒ korisničke dretve ne implementiraju algoritme međusobnog isključivanja - to implementira OS

⇒ <u>prije ulaska u k. O.:</u>

↳ rove se funkcija_za_ulazak jezgre OS-a

↳ ovisno o dopuštenju OS-a, dretva nastavlja dalje il se stavlja na čekanje

⇒ <u>nakon izlaska iz k. O.:</u>

↳ rove se funkcija_za_izlazak jezgre OS-a

↳ OS zada može pokrenut drugu dretvu koja želi ući u k. O.

Pr: Imamo više dretvi sa sredstvima: DAT, VAR!

D1

čitaj DAT;
čitaj VAR;
VAR = F1(...); } K.O.1
računaj;
DAT = F2(...); } K.O.2
računaj;
VAR = F1(...); } K.O.1

D2

čitaj VAR;
čitaj DAT;
DAT = F2(...); } K.O.2
računaj;
VAR = F1(...); } K.O.1

↳ primjetimo da ako imamo dva različita sredstva, imamo dva različita tipa kritičkog odsječka (za varijablu - K.O.1; za datoteku - K.O.2)

↳ moramo imati dva algoritma provjere - svaki algoritam ima svoju tablicu za svoja sredstva

JEŽGRA OPERACIJSKOG SUSTAVA

⇒ jezgra operacijskog sustava je zapravo **skup funkcija** i pratećih **struktura podataka**

⇒ ULOGE JEZGRE:

- stvoriti okruženje za izvođenje programa
- omogućiti višedretveni rad
- sinkronizacija dretvi (npr. međusobno isključivanje)
- rad s ulazno / izlaznim napravama

⇒ prije gradenja našeg OS-a, navedimo pretpostavke:

a) jednoprocesorski sustav (zbog jednostavnosti)

b) postoji U/I naprave te prekidni rad

c) u sustavu djeluje sklopovski sat koji izaziva periodične prekide u vremenu T_S (koristit ćemo ga za izmjenu dretvi)

⇒ jezgrine funkcije pozivaju se programskim prekidom iz različitih razloga (npr. U/I operacija, sinkronizacija)

⇒ ostali prekidi dolaze od sklopovskog sata i nekih U/I naprava

⇒ ZADAĆA JEZGRINIH FUNKCIJA - što prije obaviti radni posao i time što prije pokrenuti korisničku dretvu

STANJA DRETVI

⇒ operacijski sustav mora znati za svaku dretvu na što ona čeka kako bi mogao efikasno ostvariti višedretveni rad

⇒ za svaku dretvu, OS pohranjuje OPISNIK DRETVE koji sadrži:

- ID procesa
- ID dretve
- stanje
- prioritet
- početna adresa
- ⋮
- KONTEKST – najviše memorije i mjesta

⇒ opisnici dretvi se nalaze u listama

↳ svako moguće stanje dretve ima posebnu listu

⇒ STANA DRETVI (liste):

a) aktivne
b) pripravne
c) blokirane

1) AKTIVNA DRETVA:

⇒ ona dretva koja se izvodi

⇒ u našem jednoprocesorskom sustavu postoji samo jedna aktivna dretva

⇒ kada dođe prekid, OS zna da je to dretva koja je bila prekinuta i njezin kontekst je spremljen u „aktivnu" listu na sustavskom stogu

2) PRIPRAVNA DRETVA:

⇒ dretve koje su spremne za izvođenje

⇒ te dretve ne čekaju na ništa drugo, osim da se procesor oslobodi

⇒ svaka aktivna dretva je prvo morala biti pripravna

⇒ pripravne dretve se stavljaju u RED:

↳ realizacija reda:

a) FIFO načelo

b) prioritetni red

↳ <u>višeslojna realizacija</u> - postoj FIFO red za svaki poseban prioritet

⇒ unutar reda pripravnih, OS može raditi podjelu vremenu procesora

⇒ LATENTNA DRETVA - dretva koja je uvijek pripravna

- stavlja se na procesor kada je OS napravio svoje, a korisnik nije ratrazio ništa

- „idle" dretva

⇒ AKTIVIRANJE DRETVE:

↳ opisnik se premješta iz reda pripravnih u red aktivnih i iz opisnika obnavljamo kontekst dretve (stavljamo ju na procesor)

↳ naravno, uvijek se prilikom aktiviranja dretve uzima ona prva iz reda pripravnih dretvi

3) BLOKIRANA DRETVA:

⇒ dretve koje čekaju na ispunjenje nekog uvjeta (nakon ispunjenja svih uvjeta, dretva može postati pripravna)

⇒ uzrod čekanja:

a) binarni semafor

b) opći semafor

c) odgođeno stanje

d) čekanje na U/I napravu

UZROČNICI BLOKIRANJA

a) BINARNI SEMAFOR:

⇒ sinkronizacijski mehanizam koji omogućava međusobno isključivanje

⇒ podatkovna struktura:

- BSEM[i].v – vrijednost (0, 1)
- red opisnika dretvi blokiranih na tom semaforu

⇒ ako je: v == 1

↳ semafor je prolazan

↳ dretva može proći, ali vrijednost postaje nula

⇒ ako je: v == 0

↳ semafor je neprolazan

↳ dretva ne može proći semafor i postaje blokirana

⇒ stanja binarnog semafora:

- v == 1 → prolazan
- v == 0 → neprolazan, nema blokiranih dretvi (v == 0, red prazan)
- v == 0 → neprolazan, ima blokiranih dretvi (v == 0, red neprazan)

⇒ upotreba semafora:

1) ispitivanje semafora

↳ funkcija čekaj_bsem(ispitaj_bsem[i])

↳ to može učiniti jedino aktivna dretva

↳ ona pita: „mogu li proći?"

↳ dva slučaja:

a) $v == 1$ → dretva ostaje aktivna

b) $v == 0$ → dretva se smješta u red čekanja na tom semaforu (postaje blokirana)

2) oslobađanje blokiranih dretvi

↳ dretva ne može sama sebe deblokirat

↳ samo aktivna dretva koja raktjera postavljanje semafora može osloboditi neku od dretvi iz reda toga semafora

↳ time, ta dretva ima funkciju:

postavi_bsem[i]

↳ ako nema blokiranih dretvi: $v = 1$

↳ ako ima blokiranih dretvi:

- prva dretva iz reda blokiranih prelazi se u red pripravnih
- semafor ostaje neprolazan

↳ ovakov semafor možemo koristit za međusobno isključivanje:

- na početku kritičnog odsječka stavimo funkciju: čekaj_bsem(...);
- na kraju kritičnog odsječka stavimo funkciju: postavi_bsem(...);

PR: Sinkroniziraj ULAZNU, RADNU i IZLAZNU dretvu.

ULAZNA

```
dok je (1)
    čekaj_bsem[1];        1.
    dohvat podatak;
    pošalj radnoj dretvi;
    postavi_bsem[3];      6.
```

RADNA

```
dok je (1)
    čekaj_bsem[3];        5.
    čekaj_bsem[2];        3.
    prim podatak;
    obradi podatak;
    pošalj izlaznoj dretvi;
    postavi_bsem[1];      2.
    postavi_bsem[4];      8.
```

IZLAZNA

```
dok je (1)
    čekaj_bsem[4];        7.
    prim rezultat;
    prikaži rezultat;
    postavi_bsem[2];      4.
```

⇒ početna stanja:

```
bsem[1].v = 1;
bsem[2].v = 1;
bsem[3].v = 0;
bsem[4].v = 0;
```

↳ ovime smo dobili da se naše dretve izvode točno definiranim redoslijedom neovisno o brzini pojedine dretve

MONITORI

⇒ nastali su za sinkronizaciju dretvi jer su semafori zakazali u nekim slučajevima

PR: Problem proizvođača i potrošača

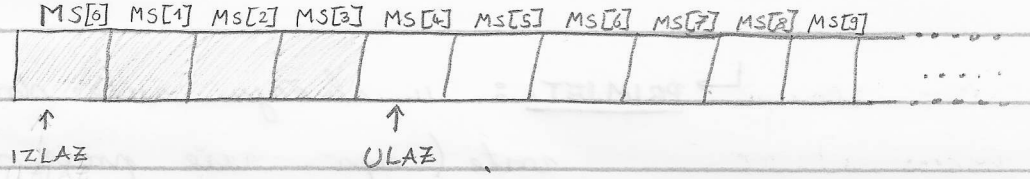
PROIZVOĐAČ - ciklička dretva koja generira podatke i šalje ih potrošaču

POTROŠAČ - ciklička dretva koja čeka na podatke i konzumira ih

↳ problem je nagomilavanje poruka

↳ PRETPOSTAVKE:

a) neograničen spremnik (oznaka MS[i])



↳ sinkronizacija: 1 opći semafor koji broj poruke u međuspremniku

↳ početno: UL = 0 , IZ = 0, OSEM(1).V = 0

PROIZVOĐAČ

```
DOK JE (1)
    proizvedi podatak;
    MS[UL++] = podatak;
    postavi_osem(1);
```

POTROŠAČ

```
DOK JE (1)
    čekaj_osem(1);
    podatak = MS[IZ++];
    potroši podatak;
```

b) ograničeni spremnik

↳ sinkronizacija: dva opća semafora
(jedan za broj poruka, a drugi
za broj praznih pretinaca)

↳ početno: osem(1).v = 0, osem(2).v = N

PROIZVOĐAČ

```
DOKJE(1)
    proizvedi podatak;
    čekaj_osem(2);
    MS[UL] = podatak;
    UL = (UL+1) % N;
    postavi_osem(1);
```

K.O.

POTROŠAČ

```
DOKJE(1)
    čekaj_osem(1);
    podatak = MS[IZ];
    IZ = (IZ+1) % N;
    postavi_osem(2);
    potroši podatak;
```

↳ **PRIMJETI**: u slučaju više dretvi iste vrste (npr. više proizvođača), trebamo uvesti binarni semafor za zaštitu kritičnog odsječka

c) **red poruka** - sada nemamo statički rezervirani prostor, već dinamički

↳ ideja je da više parova proizvođača i potrošača imaju zasebne redove poruka

↳ koristi se zajednička dinamička skladište poruka

↳ PODACI ZA PAR proizvođač/potrošač:

- red_poruka[i]
- jedan osem[i] za broj poruka u redu poruka (početno je nula)

↳ ZAJEDNIČKI PODACI S

- lista skladište
- osem[s] → broj pretinaca (početno je N)
- jedan bsem za raštetu pristupa skladištu i redarna poruka (početno je 1)

PROIZVOĐAČ

```
DOKJE(1)
    proizvedi podatak;
    čekaj_osem(s);
    čekaj_bsem(k);
    dohvati pretinac sa skladišta;
    smjesti u red_poruka[i];
    postavi_bsem(k);
    postavi_osem(i);
```

POTROŠAČ

```
DOKJE(1)
    čekaj_osem(i);
    čekaj_bsem(k);
    podatak = poruka iz red_poruka[i];
    vrati pretinac u skladište;
    postavi_bsem(k);
    postavi_osem(s);
    potroši podatak;
```

ZAD: Imamo sljedeću strukturu po FIFO načelu:

AKTIVNA DRETVA	PRIPRAVNA DRETVA	BSEM[1]	OSEM[1]	UI[1]	UI[2]	ODGOĐENA DRETVA	(J_FJA)
7	8, 5	4, 11	1	6	3, 12	2(2), 9(5)	prekid_UI(1)
8	5, 7, 6	4, 11	1	–	3, 12	2(2), 9(5)	započni_UI(2)
5	7, 6	4, 11	1	–	3, 12, 8	2(2), 9(5)	otkucaj_sata()
7	6, 5	4, 11	1	–	3, 12, 8	2(1), 9(5)	zakasni(3)
6	5	4, 11	1	–	3, 12, 8	2(1), 7(2), 9(3)	otkucaj_sata()
5	6, 2	4, 11	1	–	3, 12, 8	7(2), 9(3)	postavi_bsem(1)
6	2, 5, 4	11	1	–	3, 12, 8	7(2), 9(3)	čekaj_osem(1)
2	5, 4	11	1, 6	–	3, 12, 8	7(2), 9(3)	

POTPUNI ZASTOJ

⇒ može se dogoditi ako se barem dvije dretve nadmeću za najmanje dva sredstva

⇒ UVJETI ZA POTPUNI ZASTOJ:

a) barem dvije dretve koriste barem dva sredstva

b) sredstva se koriste međusobno isključivo

c) dretva sama otpušta sredstvo nakon korištenja

d) dretva drži sredstva zauzeta dok čeka na dodjelu dodatnog sredstva

⇒ u praksi, najčešće se odstranjuje 4 uvjet tako da se sva sredstva dodjeljuju u isti mah (odjednom)

↳ ovo nije moguće sa semaforima jer se ispitivanjem semafora sredstvo automatski zauzima te se ispituje samo jedno sredstvo

↳ zbog toga ćemo trebati monitore

PROBLEM SEMAFORA:

↳ nije moguće ispitati više uvjeta, a potom zauzeti sredstva

PR: Problem trgovca i pušača!

TRGOVAC

```
DOKJE(1)
    čekaj_bsem(stol_praran);
    stavi sastojke;
    postavi_bsem(duhan);
    postavi_bsem(šibice);
    postavi_bsem(papir);
```

PUŠAČ_ŠIBICE

```
DOKJE(1)
    cekaj_bsem(duhan);
    cekaj_bsem(papir);
    uzmi sastojke;
    postavi_bsem(stol_praran);
    ⋮
```

↳ pušač s papirom i duhanom su analogni pušaču sa šibicama:

PUŠAČ_DUHAN

```
DOKJE(1)
    ⋮
    cekaj_bsem(šibice);
    cekaj_bsem(papir);
    ⋮
```

PUŠAČ_PAPIR

```
DOKJE(1)
    ⋮
    cekaj_bsem(šibice);
    cekaj_bsem(duhan);
    ⋮
```

⇒ JEDNOSTAVNI UVJET: samo jedno sredstvo je potrebno za napredak dretve

⇒ SLOŽENI UVJET: više sredstva je potrebno da bi dretva napredovala (postoje neki logički operatori među sredstvima)

↳ za složene uvjete nam trebaju monitori

⇒ <u>SINKRONIZACIJA MONITOROM:</u>

↳ postoje dvije komponente:

a) MONITORSKE FUNKCIJE ⇒ pišemo ih korisnik za rješavanje određenog sinkronizacijskog problema

b) JEZGRINE FUNKCIJE ⇒ služe za pisanje monitorskih funkcija i pozivaju se samo unutar monitorskih funkcija

⇒ unutar monitorske funkcije dretva provjerava neki složeni uvjet te zauzima il otpušta sredstvo

⇒ u nekom trenutku, monitorsku funkciju smije izvoditi samo jedna dretva - monitorska funkcija je kritičan odsječak

↳ taj K.O. štitimo monitorskim semaforom

(on je vrlo sličan binarnom semaforu)

PR: Problem trgovca i pušača uz monitor ▽

a) IDEJNO RJEŠENJE:

PUŠAČ(i)

```
DOK JE (1)
    m-fja uzmi_sastojke();
    smotaj;
    ⋮
```

TRGOVAC

```
DOK JE (1)
    m-fja stavi-sastojke();
```

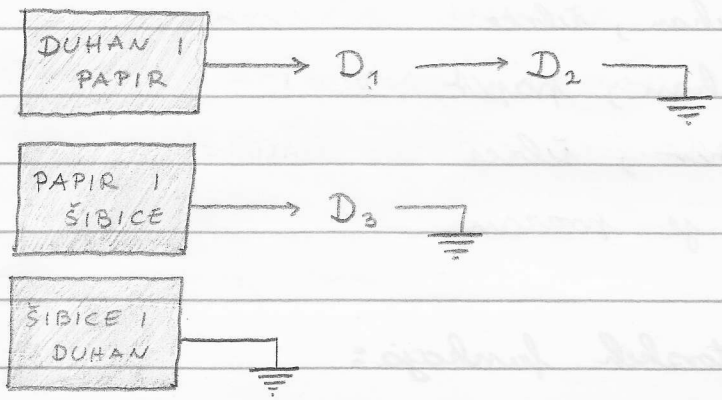
⇒ realizacija monitorskih funkcija:

```
m-fja uzmi_sastojke () {
    uđi-u-monitor();   // čekaj-bsem()
    ako je (duhan i papir na stolu) {
        uzmi duhan;
        uzmi papir;
    }
    ⋮
    izađi_iz_monitora(); // postavi-bsem()
}
```

```
m-fja stavi-sastojke() {
    uđi-u-monitor();   // čekaj-bsem()
    ako je (stol praran)
        stavi dva sastojka
    ⋮
    izađi_iz-monitora(); // postavi-bsem()
}
```

⇒ ako uvjet nije ispunjen u monitorskoj funkciji, dretva se blokira u red čekanja toga uvjeta, tj toga određenog monitora

↳ to primjerice izgleda ovako:



⇒ svaki složen uvjet dobiva svoju oznaku:

npr. "Λ" (DUHAN I ŠIBICE)

⇒ ponovno napomenimo:

↳ svaki uvjet ima svoj red čekanja

⇒ dretva koja se blokira u monitoru (na uvjetu) mora istovremeno otvoriti monitorski semafor

↳ takve jezgrine funkcije za sada nemamo i moramo ih dodati

⇒ druga dretva može ispuniti neki uvjet i obavijestiti dretvu koja čeka u redu toga uvjeta

PR: Problem trgovca i pušača uz monitor!

b) UZ JEZGRINE FUNKCIJE ZA MONITORE:

⇒ uvjeti: A → duhan, šibice
B → duhan, papir
C → papir, šibice
D → stol je praran

⇒ realizacija monitorskih funkcija:

```
m - fja  urni - sredstva() {
    uđi - u - monitor();
    dok nije (duhan i šibice na stolu)
        | čekaj - na - uvjet ("A");
    urni duhan i šibice;
    ispuni - uvjet ("D");
    izađi - iz - monitora();
    }
```

↳ NAPOMENA: u praksi se uvijek provjerava negativan uvjet jer je tako puno lakše realizirati stvar preko „while" petlje umjesto množtva „if"-ova

```
m_fja stavi_sredstva() {
    uđi_u_monitor();
    dok nije (stol je praran)
        čekaj_na_uvjet("D");
    stavi dva sastojka;
    ispuni_uvjet("oprano s stanjenim sastojcima");
    vrati_iz_monitora();
}
```

nova jezgrina funkcija

⇒ POTREBNE JEZGRINE FUNKCIJE:

a) ULAZAK u monitor

b) STAVLJANJE - stavljanje u red čekanja nekog uvjeta i propuštanje sljedeće dretve u monitor

c) oslobađanje druge dretve iz reda čekanja na uvjet (OSLOBAĐANJE)

d) IZLAZAK iz monitora

⇒ STRUKTURA PODATAKA MONITORA:

a) red čekanja na ulaz u monitor

b) redovi čekanja na uvjete

⇒ NAŠE JEZGRINE FUNKCIJE:

1) uđi_u_monitor (M)

↳ čekaj_lsem (M)

2) izađi_iz_monitora (M)

↳ pretavi_lsem (M)

3) čekaj_u_redu_uvjeta (M, K)

↳ uvrsti_u_red + uđi_u_monitor

4) oslobodi_iz_reda_uvjeta (K)

4a) oslobodi_sve_iz_reda_uvjeta (K)

PR: Problem 5 filozofa uz monitor!

⇒ uvjet: svaki filozof I čeka na uvjet I

↳ to znači: 1 → vilica 1, 2

2 → vilica 2, 3

⋮

5 → vilica 5, 1

⇒ pomoćne varijable:

LIJEVI → (I + 4) % 5

DESNI → (I + 1) % 5

FILOZOF(i)

```
DOK JE(1)
    misli;
    m-fja uzmi_vilice(i);
    jedi;
    m-fja vrati_vilice(i);
```

⇒ za rješenje, samo treba napisati monitorske funkcije:

```
uzmi_vilice(I){
    uđi_u_monitor(M);
    dok (nemam obje vilice)
        čekaj_u_redu_monitora(M, I);
    uzmi vilice;
    vrati_iz_monitora(M);
}
```

```
vrati_vilice(I){
    uđi_u_monitor(M);
    spusti vilice;
    oslobodi_iz_reda(LIJEVI);
    oslobodi iz reda(DESNI);
    izađi_iz_monitora(M);
}
```

PR: Problem pušača s monitorima (konkretno) ▽:

⇒ struktura podataka: $D = \emptyset$, $Š = \emptyset$, $P = \emptyset$

⇒ monitori: M

⇒ redovi uvjeta:

- A: D + Š
- B: D + P
- C: P + Š
- D: TRGOVAC

TRGOVAC

```
DOK JE (1)
    uđi_u_monitor(M);
    dok je ( D ≠ ∅ || Š ≠ ∅ || P ≠ ∅ )
        čekaj_u_redu_uvjeta(M, D);
    stavi 2 sastojka na stol;
    oslobodi_iz_reda_uvjeta(A);
    oslobodi_iz_reda_uvjeta(B);
    oslobodi_iz_reda_uvjeta(C);
    izađi_iz_monitora(M);
```

oslobodimo sve pušače, pa onda oni odlučuju koj mogu prći a koj ne - trgovac ne zna koj pušač što treba
↳ ovo će najčešće biti specificirano u zadatku (koja dretva što zna)

PUŠAČ(i)

```
DOKJE(1)
    uđi_u_monitor(M);
    dok je („moj uvjet nije radoslježen")   // npr. (D==∅ || Š==∅)
        čekaj_u_redu_uvjeta(M, „moj uvjet"); // npr. A
    uzmi dva tražena sastojka;              // npr. D=∅; Š=∅;
    oslobodi_iz_reda_uvjeta(D);
    izađi_iz_monitora;
    smotaj ...;
```

PR: Problem proizvođača i potrošača uz ograničen spremnik!

⇒ podaci: • MS[N], BROJ_M = N;

• IZ, UL = Ø

• POTROŠAČ_ČEKA = Ø;

• PROIZVOĐAČ_ČEKA = Ø

⇒ monitori: M

⇒ redovi uvjeta:

• 1: PROIZVOĐAČ

• 2: POTROŠAČ

```
pošalji_poruku(P) {
    uđi_u_monitor(M);
    dokle (BROJ_M == N)
        | čekaj_u_redu_uvjeta(M, 1);
    MS[UL] = P;
    UL = (UL + 1) % N;
    BROJ_M --;
    ako (POTROŠAČ_ČEKA == 1) {
        | oslobodi_iz_reda_uvjeta(2);
        | POTROŠAČ_ČEKA = Ø;
    }
    izađi_iz_monitora(M);
}
```

PROIZVOĐAČ_ČEKA = 1;

```
primiti_poruku(R); {
    uđi_u_monitor(M);
    dok je (BROJ_M == N) {
        POTROŠAČ_ČEKA = 1;
        čekaj_u_redu_uvjeta(M, 2);
    }
    R = MS[IZ];
    IZ = (IZ + 1) % N;
    BROJ_M++;
    ako (PROIZVOĐAČ_ČEKA == 1) {
        PROIZVOĐAČ_ČEKA = 0;
        oslobodi_iz_reda_uvjeta(1);
    }
    izađi_iz_monitora(M);
}
```

⇒ NAPOMENA: ovo neće omogućiti paralelizam više potrošača, već ako ih imamo više, može se desiti da samo jedan potrošač radi, a ostali su vječno blokirani

↳ prilagodba za više potrošača, proizvođača nije teška (povećavamo i smanjujemo broj potrošača i proizvođača na čekanju)

PRAVILA ZA MONITORSKE FUNKCIJE

⇒ monitorske funkcije izvode se u korisničkom načinu rada, ali su zaštićene monitorskim semaforom:

↳ na početku: uđi_u_monitor(M);

↳ na kraju: izađi_iz_monitora(M);

⇒ unutar funkcije provjerava se bilo koji složeni uvjet

⇒ ako uvjet nije zadovoljen, dretva se blokira na redu tog uvjeta i odključava monitor

⇒ funkcije „čekaj_u_redu_uvjeta(...)" i „oslobodi_iz_reda_uvjeta(...)" pozivaju se unutar monitorske funkcije

⇒ uvjet se uvijek provjerava sa DOKJE(...), a ne sa AKOJE(...) iz sigurnosti

ZAD: Microsoftovi i linuxovi programeri dijele restoran, te u jednom trenutku mogu u restoranu biti samo jedna vrsta programera. Algoritme programera:

```
PROGRAMER(vrsta)
    uđi(vrsta);
    jedi;
    izađi(vrsta);
```

a) Sinkroniziraj programere monitorom!

b) Rješi zadatak bez pojave izgladnjivanja tako da najviše N programera jedne vrste može ući zaredom u restoran dok na ulazak čekaju programeri druge vrste!

ANALIZA VREMENSKIH SVOJSTAVA RAČUNALNIH SUSTAVA

⇒ sustav možemo unaprijediti softwareski i hardwareski

⇒ česta pitanja u analizi:

- koliko je prosječno poslova u sustavu u nekom trenutku?
- koliko je prosječno radno čekanje u sustavu?
  ↳ odziv sustava
- kako poboljšati vremenska svojstva sustava?
- kako dodijeliti dretve procesoru?

⇒ OCJENA PONAŠANJA SUSTAVA:

a) matematički model

b) simulacija

c) mjerenje („stavi novi hardware, pa mjeri")
   ↳ skupo!

JEDNOSTAVAN MATEMATIČKI MODEL

⇒ PRETPOSTAVKE:

- sve pripravne dretve se nalaze u jednom redu čekanja
- FCFS → „first came, first served"
  ↳ tzv. RED PRISPJEĆA
- nema prekidanja dretvi u izvođenju

⇒ OSNOVNI POJMOVI:

- trenutak dolaska: t_D
- trenutak napuštanja: t_N
- zadržavanje: $T = t_N - t_D$
- čekanje u redu: T_R
- trajanje poslуživanja T_P

$$T = T_R + T_P$$

⇒ ULAZNE VRIJEDNOSTI:

- λ → broj dolazaka poslova u jedinici vremena
- β → broj poslova koje poslužitelj može obaviti u jedinici vremena („SNAGA POSLUŽITELJA")

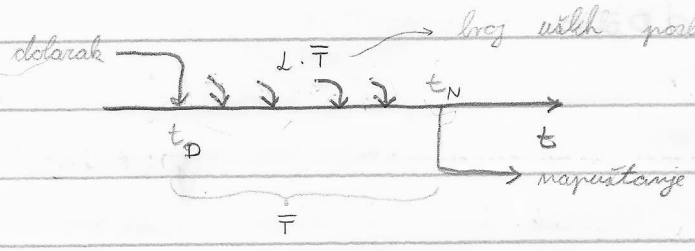
⇒ IZLAZNE VRIJEDNOSTI:

$$\boxed{\rho = \frac{\lambda}{\beta}} \quad (1)$$

- ρ → faktor iskorištenja poslužitelja $\rho \leq 1$
- $\bar{T}$ → prosječno zadržavanje poslova u sustavu
- $\bar{n}$ → prosječan broj poslova u sustavu

$$\boxed{\bar{n} = \lambda \cdot \bar{T}} \Rightarrow \text{LITTLEOVO PRAVILO} \quad (2)$$

⇒ intuitivna demonstracija Littleovog pravila:



↳ koliko ima poslova u sustavu u trenutku t_N?

- s obzirom na FCFS načelo, to su svi oni koji su ušli nakon t_D:

$$n(t_N) = \lambda \cdot \bar{T}$$

⇒ do sada smo se bavili sa **deterministickim** sustavom

↳ točno smo znali kada koj posao dolazi i koliko traje

⇒ sada prelazimo na **nedeterministički** sustav

↳ dolasci i trajanje obrade su slučajne varijable

↳ broj λ sada interpretiramo kao prosječan broj dolazaka u jedinici vremena

↳ broj β je prosječan broj poslova koj poslužitelj može obaviti u jedinici vremena

$\frac{1}{\beta}$ → prosječno trajanje posla

UVOD U RASPODJELE

⇒ SLUČAJNA VARIJABLA:

↳ varijabla koja poprima slučajne vrijednosti

(npr. broj na kocki)

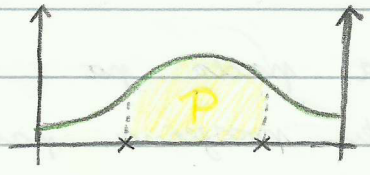
↳ OBLICI:

a) diskretne

b) kontinuirane

⇒ ponašanje slučajne varijable opisuje RASPODJELA

PR: vjerojatnost da čovjek stane na sredinu stepenice je najveća



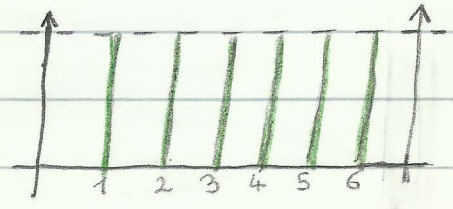
⇒ raspodjela je definirana funkcijom gustoće vjerojatnosti

⇒ vjerojatnost da slučajna varijabla poprimi neku vrijednost dobiva se kao površina ispod funkcije gustoće vjerojatnosti

⇒ za kontinuiranu slučajnu varijablu, vjerojatnost postoji samo za interval

⇒ za diskretne varijable, raspodjela definira vjerojatnost svake moguće vrijednosti

PR: bacanje kocke



PR: Bacamo kocku 10 puta. Koja je vjerojatnost da ćemo 3 puta dobiti šesticu?

$n = 10$

$k = 3$

$p = \frac{1}{6}$ ⇒ vjerojatnost da se željeni događaj dogodi

$q = \frac{5}{6}$ ⇒ vjerojatnost da se neželjeni događaj ne dogodi

$$\binom{10}{3} \cdot \left(\frac{1}{6}\right)^3 \cdot \left(\frac{5}{6}\right)^7 = 15.5\%$$

PR: Kiša pada po Poissonovoj raspodjeli. Ako u jednoj minuti prosječno padne 100 kapi, kolika je vjerojatnost da će u sljedećoj sekundi past točno dvije kapi?

$$p(k; \lambda) = \frac{\lambda^k}{k!} e^{-\lambda}$$

$\lambda = \frac{100}{60}$

$k = 2$

$p(2, \frac{5}{3}) = 26.2\%$

ANALIZA NEDETERMINISTIČKIH SUSTAVA

⇒ DOLASCI POSLOVA

↳ broj dolazaka : n

↳ prosječan broj dolazaka : λ

↳ poissonova raspodjela : k

⇒ TRAJANJE OBRADE

↳ kontinuirana slučajna varijabla

↳ prosječno trajanje obrade ($1/\beta$)

↳ eksponencijalna raspodjela - vjerojatnost da je trajanje obrade veće il manje od t

⇒ imamo jedan red (FCFS)

↳ trebamo izvesti $\bar{n}$, $\bar{T}$ uz poznat λ, β

$p_i(t)$ → vjerojatnost da se u sustavu nalazi „i" poslova u trenutku „t"

↳ pomoću oboga dolazimo do $\bar{n}$

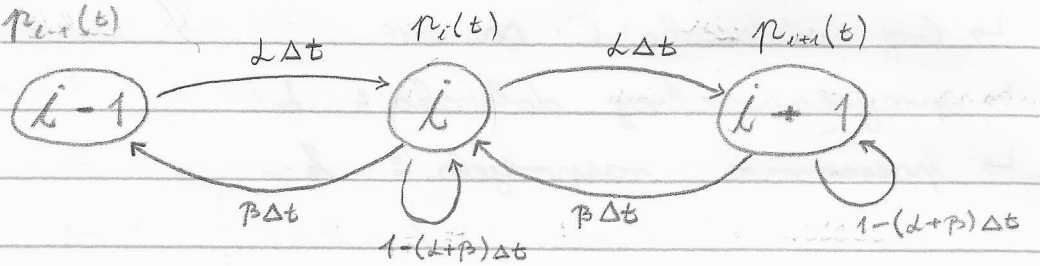
↳ gledamo sustav u intervalu ΔT (vrlo mal) te pomoću aproksimacija Poissonove jednadžbe dolazimo

VJEROJATNOST 1 DOLASKA: $\lambda \Delta T$

↳ na isti način dolazimo i VJEROJATNOST 1 ODLASKA: $\beta \Delta T$

⇒ MARKOVLJEV LANAC:

↳ prikaz našeg sustava u obliku neke vrste automata s konačnim brojem stanja



↳ svako stanje ima svoju vjerojatnost i s nekom vjerojatnosti automat mijenja stanje

↳ vjerojatnost ostanka u istom stanju je:

$$1-(\lambda+\beta)\Delta t$$

↳ zapitajmo se sljedeće:

→ koja je vjerojatnost da ćemo u trenutku $t+\Delta t$ imati točno „i" poslova?

↳ imamo 3 slučaja, da se nalazimo u svakom od 3 stanja, pa je konačno rješenje:

$$p_i(t+\Delta t) = p_{i-1}(t)\cdot\lambda\Delta t + p_{i+1}(t)\cdot\beta\Delta t + p_i(t)\left(1-(\lambda+\beta)\Delta t\right)$$

↳ sredimo ovu jednadžbu:

$$\frac{p_i(t+\Delta t)-p_i(t)}{\Delta t} = p_{i-1}(t)\lambda + p_{i+1}(t)\beta - p_i(t)(\lambda+\beta)$$

↳ pustimo na limes $\Delta t \to 0$, te dobijemo derivaciju, a mi se nalazimo u stohastičkom sustavu što znači da se stanje (prosječno) kroz vrijeme ne

mjenja, pa ovo možemo izjednačiti s nulom:

$$p_{i-1}(t)\,\lambda + p_{i+1}(t)\,\beta - p_i(t)\,(\lambda+\beta) = 0$$

↳ iz gornjih tvrdnji ćemo još neke zaključke izvodimo bitne formule:

$$\boxed{\bar{n} = \frac{\lambda}{\beta-\lambda}} \qquad \boxed{\bar{T} = \frac{1}{\beta-\lambda}} \qquad (3)$$

⇒ NAPOMENA: u nedeterminističkom sustavu faktor iskorištenja mora biti manji od jedan jer bi po formuli za $\bar{T}$, prosječno zadržavanje bilo beskonačno što je i slučaj u praksi

PR: Imamo podatke: $\lambda = 100\ min^{-1}$

a) $\beta_1 = 200\ min^{-1} \Rightarrow \bar{n} = 1$, $\bar{T} = 0{,}01\ min$

b) $\beta_2 = 120\ min^{-1} \Rightarrow \bar{n} = 5$, $\bar{T} = 0{,}05\ min$

c) $\beta_3 = 101\ min^{-1} \Rightarrow \bar{n} = 100$, $\bar{T} = 1\ min$

⇒ vjerojatnost da se u sustavu nalazi više od „N" poslova:

$$p(i>N) = \varrho^{N+1} \qquad (4)$$

⇒NAPOMENA: izrazi (3) i (4) vrijede samo uz Poissonove dolaske, eksponencijalno trajanje obrade i uvođenje redom prispijeća

ZAD: Zahtjevi za obradu podliježu Poissonovoj raspodjeli uz $\lambda = 2\ s^{-1}$, a trajanje obrade ima eksponencijalnu raspodjelu. Mjerenjem je ustanovljeno prosječno zadržavanje poslova u sustavu od 0,5 s. Kolika je vjerojatnost da se u sustavu nalazi više od pet poslova?

$\lambda = 2\ s^{-1}$

$\overline{T} = 0.5\ s$

$p(i > 5) = \rho^6$

$$\overline{T} = \frac{1}{\beta - \lambda} \Rightarrow \beta = \frac{1 + \lambda\overline{T}}{\overline{T}} = 4\ s^{-1}$$

$$\rho = \frac{\lambda}{\beta} = 0.5$$

$p(i > 5) = 0.015625$

↳ Koja je vjerojatnost da se u sustavu nalazi od dva do četiri posla?

$$p(i > 1) - p(i > 4) = \rho^2 - \rho^5 = 0.21875$$

⇒ sagledajmo slučaj kada u sustav dolaze RAZLIČITE SKUPINE POSLOVA - poslovi imaju različite parametre

Pr: Dolaze kratki i dugi poslovi:

KRATKI	DUGI
$\lambda_1 = 10\ s^{-1}$	$\lambda_2 = 5\ s^{-1}$
$\frac{1}{\beta_1} = 0{,}03\ s$	$\frac{1}{\beta_2} = 0{,}09\ s$

↳ trebamo pronaći neki rasjednički „λ":

$$\lambda = \lambda_1 + \lambda_2 = 15\ s^{-1}$$

↳ trebamo pronaći rasjednički „β":

$$\frac{1}{\beta} = \underbrace{\frac{5}{10+5} \cdot 0{,}09 + \frac{10}{10+5} \cdot 0{,}03}_{\text{koristimo težinski omjer}} = 0{,}05\ s$$

⇒ drugi način je da umjesto rasjedničkog „β" jednostavno računamo rasjedničko opterećenje „ϱ":

$$\left.\begin{aligned} \varrho_1 &= \frac{\lambda_1}{\beta_1} = 0{,}3 \\ \varrho_2 &= \frac{\lambda_2}{\beta_2} = 0{,}45 \end{aligned}\right\} \quad \varrho = \varrho_1 + \varrho_2 = 0{,}75$$

⇒ ponovimo ove formule:

$$\boxed{\begin{aligned} \lambda &= \lambda_1 + \lambda_2 \\ \varrho &= \varrho_1 + \varrho_2 \end{aligned}} \quad (5)$$

DODJELJIVANJE DRETVI PROCESORU

⇒ možemo li poboljšati parametre $\bar{n}$ i $\bar{T}$ bez da smanjujemo parametar p?

↳ možemo, jer dosadašnji rezultat vrijede samo za red prispijeća (FCFS), pa ako promijenimo to, možda možemo poboljšati svojstva sustava

KRUŽNO POSLUŽIVANJE (ROUND ROBIN - RR)

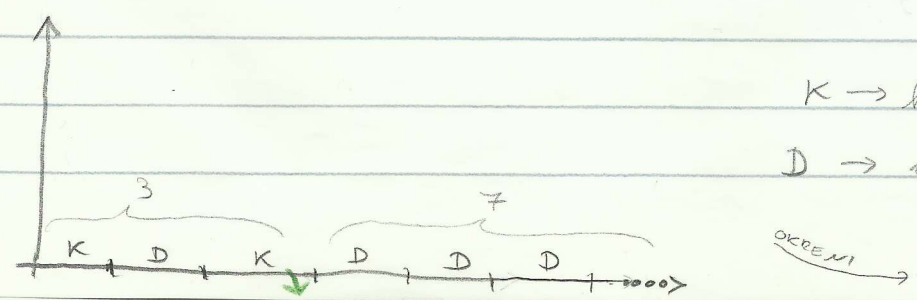
⇒ dretva se obrađuje u jednom kvantu vremena duljine T_q

⇒ nakon jednog kvanta, dretva se vraća na kraj reda pripravnih dretvi (ako u tom kvantu dretva nije završila)

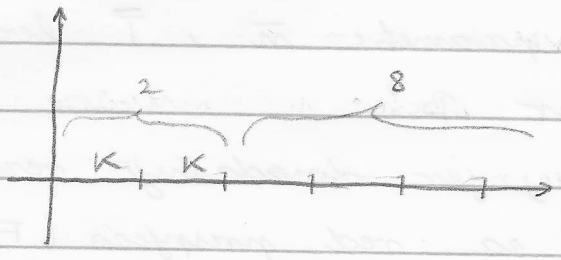
↳ pogledaj primjer sa prezentacija sa dugim i kratkim poslovima ▽

PR: Imamo mo poslove koj traju 0.02 s, a raditavaju se u sustavu $\bar{T} = 16.68$ s. Kroz jednog dugog posla trajanja 50 s koj rijetko dolaze u sustav (FCFS načelo) ▽

Primijenimo sada „Round Robin" ▽



↳ ova slika vrijedi za 625 kvanata, sve dok je u sustavu dugi posao, a nakon toga sustav će izgledati ovako:



↳ izračunajmo prosječno raspršivanje u sustavu:

$$\bar{T} = \frac{1 \cdot 62.5 + 625 \cdot 0.03 + 0.02 \cdot 375}{1001} = 0.0886$$

↳ prosječan broj poslova u sustavu:

$$\bar{m} = \frac{625(0.03 \cdot 2 + 0.07 \cdot 1) + 375(0.02 \cdot 1)}{100} = 0.8875$$

⇒ NAPOMENA: nedostatak kružnog poslužavanja može biti više izmjena konteksta, al ukolko je kvant oko 20ms, to se može zanemariti.

↳ stoga, Round Robin se koristi u velikoj većini današnjih sustava

⇒ RASPOREĐIVANJE DRETVI U STVARNIM SUSTAVIMA:

a) KRITIČNE DRETVE:

↳ izvršavaju se bez prekidanja

b) OSTALE DRETVE:

↳ ostale dretve su prekidive i raspoređuju se podjelom vremena

⇒ <u>varijante prioriteta dretvi:</u>

a) prioritetne dretve dolaze po više uzastopnih kvanata

b) postoje posebni redovi za različite razine prioriteta

PR: Raspoređivanje po jednom kriteriju ?

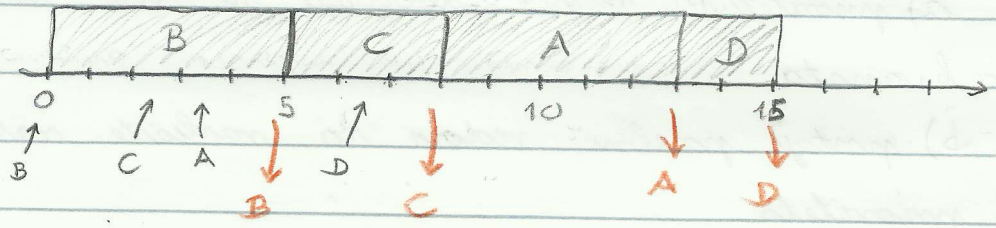
	A	B	C	D
p	5	5	3	2
t_0	3.5	0	2.5	6.5

⇒ prikaži rad ako se koristi: a) FCFS

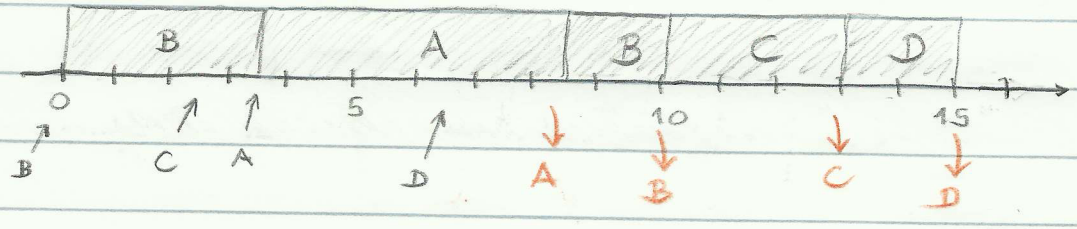
b) prema prioritetu $p_A > p_B > p_C > p_D$

c) Round Robin, $T_q = 1$

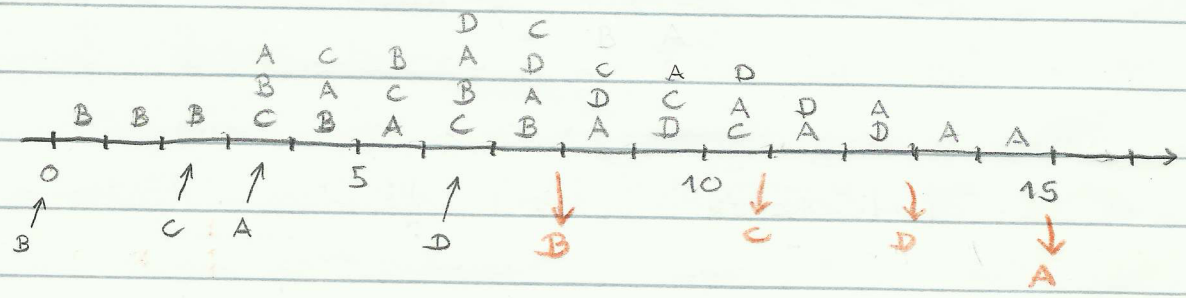
a)



b)



c)



PR: Višekriterijsko raspoređivanje!

	A	B	C	D
p	5	5	3	2
t_D	3.5	0	2.5	6.5

⇒ pokaži rad ako se:

a) raspoređuje FCFS

načelom uz: $p_A > p_B = p_C = p_D$

↳ to se naziva:

SCHED_FIFO

b) kružno raspoređivanje uz $T_q = 1$, $p_A > p_B = p_C = p_D$

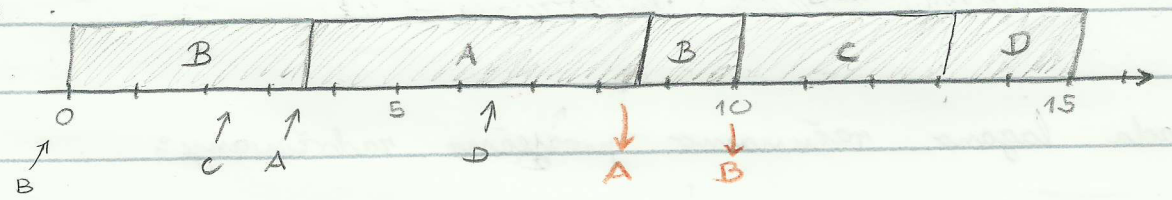
↳ to se naziva: SCHED_RR

c) kružno raspoređivanje uz različite duljine kvanta tako da je: $T_{qA} = 2$

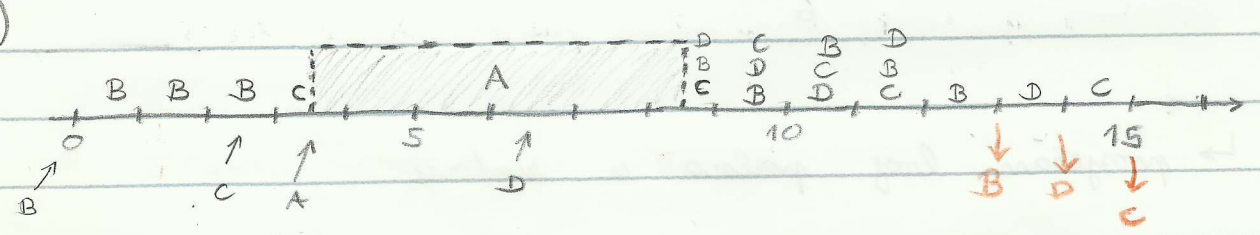
$T_{qB} = T_{qC} = T_{qD} = 1$

↳ to se naziva: SCHED_OTHER

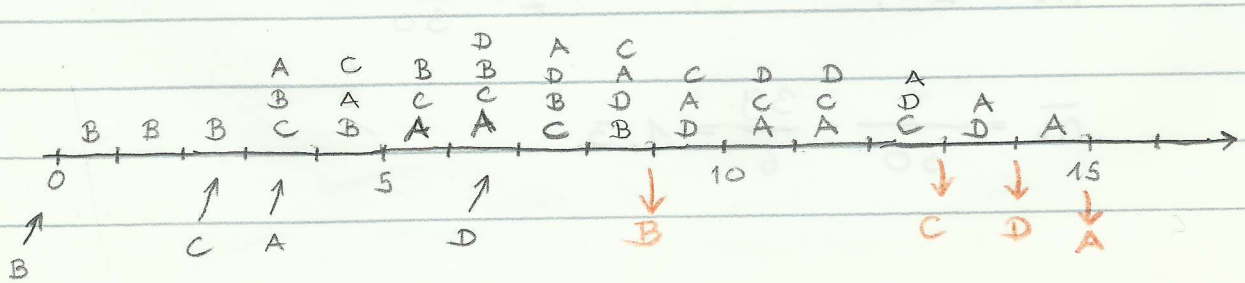
a)



b)



c)



ZAD: U deterministčkom sustavu poslovi se javljaju svakih 30 ms, to P_1 u 3 ms, P_2 u 5 ms, P_3 u 6 ms, P_4 u 10 ms, P_5 u 20 ms, P_6 u 21 ms. Svi poslovi traju 4 ms. Odredi prosječno zadržavanje i prosječan broj poslova u sustavu!

↳ radimo tablicu:

	t_D	T_P	t_N	T
P_1	3	4	7	4
P_2	5	4	11	6
P_3	6	4	15	9
P_4	10	4	19	9
P_5	20	4	24	4
P_6	21	4	28	7

↳ sada lagano računamo prosječno zadržavanje:

$$\overline{T} = \frac{39}{6} = 6.5 \text{ ms}$$

↳ prosječan broj poslova u sustavu

$$\overline{n} = \lambda \cdot \overline{T} \qquad \lambda = \frac{6}{30}$$

$$\overline{n} = \frac{6}{30} \cdot \frac{39}{6} = 1.3$$

Zad: Za neki web-sustav s jednim poslužiteljem prosječan broj zahtjeva u minuti je 100, a poslužitelj može obraditi 300 zahtjeva u minuti. Zahtjevi su po Poissonovoj raspodjeli, a trajanje je eksponencijalna raspodjela. Kolik se postotak poslužiteljeg vremena može odvojiti za druge usluge, a da klijenti ne čekaju u prosjeku više od 2 sekunde?

$\lambda = 100\ min^{-1}$

$\beta = 300\ min^{-1}$

$\overline{T}' = 2\ s$ → vrijeme nakon odvajanja snage procesora / poslužitelja

↳ trenutno, poslovi u prosjeku čekaju: $\overline{T} = \frac{1}{\beta - \lambda} = 0.3\ s$

↳ sad trebamo dobiti β' kako bi dobili omjer nove i stare snage poslužitelja:

$$\overline{T}' = \frac{1}{\beta' - \lambda} \Rightarrow \beta' = \frac{1}{\overline{T}'} + \lambda$$

$$\beta' = 130\ min^{-1}$$

↳ vrijeme koje možemo uzeti:

$$PS = 1 - \frac{\beta'}{\beta} = 56.6\%$$

ZAD: U sustavu s jednim poslužiteljem prosječan broj zahtjeva u sekundi je 100. Poslužitelj obrađuje 3 vrste zahtjeva:

Z_1: obrada 5 ms

Z_2: obrada 8 ms

Z_3: obrada 10 ms

Ako je udio zahtjeva Z_1 30%, Z_2 40% i Z_3 30%, odredi prosječno radbrivanje te vjerojatnost da je u sustavu više od 10 zahtjeva? Zahtjevi pristižu po Poissonovoj raspodjeli, a trajanja su po eksponencijalnoj raspodjeli?

$\lambda = 100\ s^{-1}$

	Z_1	Z_2	Z_3
$\frac{1}{\beta_i}$	5	8	10
udio	30%	40%	30%

$\frac{1}{\beta_1} = 5\ ms$ $\quad \frac{1}{\beta_2} = 8\ ms$ $\quad \frac{1}{\beta_3} = 10\ ms$

$\frac{\lambda_1}{\lambda} = 0.3 = \frac{\lambda_3}{\lambda}$ $\quad \frac{\lambda_2}{\lambda} = 0.4$

ukupni β

$$\overline{T} = \frac{1}{\beta - \lambda}$$

↳ ukupni β računamo preko opterećenosti poslužitelja:

$$\rho = \frac{\lambda}{\beta} = \rho_1 + \rho_2 + \rho_3$$

$\rho_1 = \frac{\lambda_1}{\beta_1} = 0.15$ $\quad \rho_2 = \frac{\lambda_2}{\beta_2} = 0.32$

$\rho_2 = \frac{\lambda_3}{\beta_3} = 0.3$

$\rho = 0.77 \Rightarrow \beta = \frac{\lambda}{\rho}$

$$\overline{T} = \frac{1}{\frac{\lambda}{\beta} - \lambda} = 33.5\ ms$$

$p(i > 10) = \rho^{11} = 5.6\%$

ZADO: U nekom poslužiteljskom centru ustanovljeno je da tri poslužitelja rade s malim opterećenjem. Poslužitelj P_1 prosječno dobiva 70 zahtjeva u minuti uz iskorištenost 20%. Poslužitelj P_2 dobiva 200 zahtjeva u minuti uz iskorištenost 30%. Poslužitelj P_3 dobiva 150 zahtjeva u minuti uz iskorištenost 10%. P_3 je 50% jači od P_1 i 100% jači od P_2. Izračunaj kvalitetu usluge ako se svi poslovi presele na P_3? Zahtjevi su po Poissonovoj raspodjeli, a trajanja su po eksponencijalnoj raspodjeli!?

$\lambda_1 = 70\ min^{-1}$

$\lambda_2 = 200\ min^{-1}$

$\lambda_3 = 150\ min^{-1}$

$\overline{T} = ?$

$$\overline{T} = \frac{1}{\mu - \lambda}$$

↳ ukupni λ: $\lambda = 420\ min^{-1}$

↳ sada pomoću danih omjera (postotaka) treba dobiti μ:

$$\mu_{3,P_3} = \frac{\lambda_3}{f_3} = 1500\ min^{-3}$$

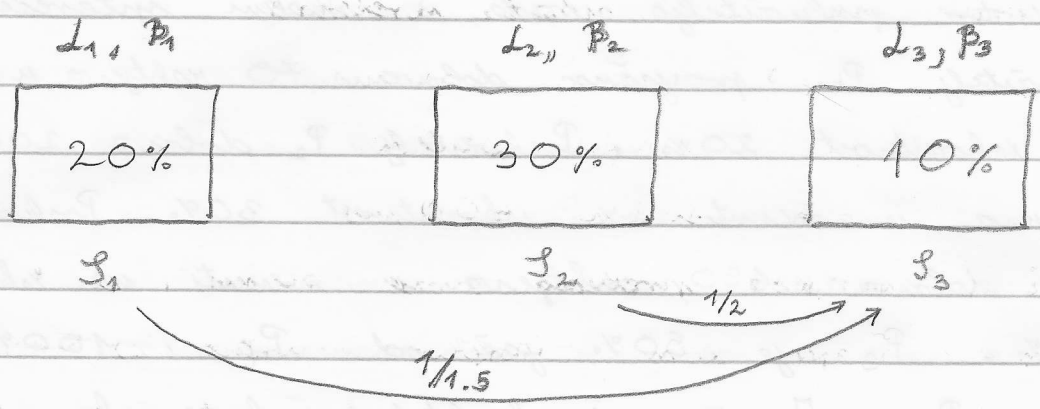
↑ μ za poslove P_3

⇒ ovo nije ukupni μ jer ne znamo jesu li svi poslovi istog trajanja (nije rečeno u zadatku, pa vjerojatno nisu)

↳ sada vidimo da je ovo krivi put jer nećemo moći dobiti kumulativni (ukupni) μ

OKRENI →

⇒ pokušajmo preko opterećenja poslužitelja:



$$\rho_3' = \rho_3 + \frac{1}{2}\rho_2 + \frac{1}{1.5}\rho_1 = 38.3\%$$

$$\beta_3' = \frac{\lambda}{\rho_3'}$$

↳ sada možemo računati kvalitetu sustava:

$$\bar{T} = \frac{1}{\frac{\lambda}{\beta_3'} - \lambda} = 0.08\dot{8}\ \Delta$$

ZAD: U sustavu bez sklopa za prihvat prekida javljaju se 3 različita prekida:

P_1 : svakih 150 ms

P_2 : svakih 350 ms

P_3 : svakih 400 ms

Obrada svakog prekida traje 50 ms, a kućanski poslovi traju 2 ms. Koliko postotno opterećenje stvaraju radarni prekidi, a koliko ostaje za druge programe?

⇒ NAPOMENA: primjeti da nam ovdje uopće nisu bitni prioriteti radarnih prekida jer sama računamo opterećenje, a ne raspoređivanje pojedinog prekida

⇒ nemamo sklop za prekid, pa se po svakoj pojavi, i po svakom izlasku iz prekida moraju obaviti kućanski poslovi

$$P_1 : \frac{50+2+2}{150} = \varphi_1$$

$$P_2 : \frac{50+2+2}{350} = \varphi_2$$

$$P_3 : \frac{50+2+2}{400} = \varphi_3$$

$$\varphi = \varphi_1 + \varphi_2 + \varphi_3$$

$$\varphi = 65\ \%$$

⇒ za druge programe stoga ostaje 35%

SPREMNIČKI I DISKOVNI PROSTOR

⇒ odgovaramo na pitanja:

a) gdje i kako se kontekst i obrtve smještaju?

b) kako rade hard-diskovi i način pohrane?

c) kako omogućit postojanje više procesa istovremeno u zajedničkom spremniku?

d) kako omogućit korištenje maksimalnog logičkog adresnog prostora unutar raspoloživog fizičkog prostora?

⇒ DANAŠNJA RJEŠENJA:

- STRANIČENJE („Paging")
- VIRTUALNA MEMORIJA

IZVOĐENJE PROGRAMA

⇒ program mora biti prepremljen u obliku strojnih instrukcija ⇐

↳ to rade compileri

⇒ adrese nastaju unutar procesora il DMA sklopa

↳ one nam određuju adrese instrukcija, stoga, nekih podataka ...

⇒ NAJJEDNOSTAVNIJE RJEŠENJE problema konflikta:

↳ u jednom trenutku samo jedan proces se nalazi u spremniku („sav RAM damo jednom programu")

↳ ostali procesi moraju biti pohranjeni na pomoćnom spremniku (DISKU)

↳ prekidi su ovdje također mogući, ali onda cijel RAM moramo prenještati na disk

↳ to nas košta vremenski, pa trebamo provući mogstva diskova

OSNOVNA SVOJSTVA DISKOVA

⇒ namjena diskova:
- pomoćni spremnik
- trajna pohrana podataka

⇒ KOMPONENTE DISKA:
- elektro - mehanički dio (pohrana podataka)
- upravljački sklop (razgovara s OS-om)

⇒ kod rada s diskom, pristupa se blokovima bajtova (jednom il više sektora)

⇒ ADRESIRANJE:
↳ OS adresira sektor jer on vidi disk kao linearni skup sektora
↳ upravljački sklop mora odrediti:
- broj ploče
- broj staze
- broj sektora na stazi

VREMENSKA SVOJSTVA DISKOVA

⇒ **UKUPNO TRAJANJE PRIJENOSA:**

↳ **trajanje postavljanja glave:**

- trajanje traženja staze (seek time):
  - ubrzavanje
  - kretanje maksimalnom brzinom
  - usporavanje
  - fino pozicioniranje
- rotacijsko kašnjenje ($T_R/2$)

↳ **trajanje prijenosa podataka:**

- trajanje čitanja dijela ili cijele staze te (po potrebi) trajanje premještanja glave na susjednu stazu (samim time treba opet ući rotacijsko kašnjenje)

⇒ VAŽNA NAPOMENA:

↳ ako je jedinica bajt (B), onda prefiksi K, M, G znače potencije broja dva: $2^{10}, 2^{20}, 2^{30}$

ZAD: Disk ima 100 sektora po staze veličine 1KB, te 2000 staza, 4 ploče, 7200 rpm, a podaci su raspisani na obje strane ploča. Upravljački sklop čita cijelu stazu u interni spremnik, a zatim radi prijenos preko sabirnice. Prijenos u glavni spremnik odvija se brzinom 200 Mb, a za to vrijeme sklop ne može čitati s diska.

a) Kolik je kapacitet diska?

b) Koliko traje prebacivanje kompaktno smještene datoteke od 135KB ako je trajanje traženja staze 10 ms, a trajanje premještanja na bliski stazu 1 ms?

a) $4 \cdot 2 \cdot 2000 \cdot 100 \cdot 1KB = 1600\,000\,KB \approx 1.525879\ GB$

b) 1) pronalaženje početka

- traženje staze: 10 ms
- rotacijsko kašnjenje: $\frac{60}{7200} \cdot \frac{1}{2}\ s$

} 14.17 ms

2) čitanje staze: $T_R = \frac{60}{7200}\ s = 8.3\ ms$

3) prijenos 100 KB podataka: $\frac{BR.\ BITOVA}{BRZINA} = \frac{100 \cdot 1024 \cdot 8}{200 \cdot 10^6} = 4.1\ ms$

4) čitanje sljedeće staze sa istog cilindra za ostalih 35 KB:

↳ imamo rotacijsko kašnjenje: 4.17 ms

↳ čitanje druge staze: 8.3 ms

↳ prijenos 35 KB podataka: $\frac{35 \cdot 1024 \cdot 8}{200 \cdot 10^6} = 1.4\ ms$

UKUPNO: 40.5 ms

DODJELJIVANJE SPREMNIKA

⇒ CILJEVI:
- smještanje više procesa istovremeno
- smještanje procesa koji je već od raspoloživog fizičkog spremnika

⇒ evidencija o procesima:
↳ za svaki proces postoj PROCESNI INFORMACIJSKI BLOK
- unutra se nalaze sve informacije povezane s blokom (kao opisnik dretve)

A) STATIČKO RASPOREĐIVANJE SPREMNIKA:
↳ <u>apsolutni oblik programa:</u>
- adrese u programu su stvarne lokacije u spremniku
- za izvođenje takvih programa mora biti poznata početna adresa

↳ <u>relativni oblik programa</u> (ovo rade compileri):
- sadrži relativne adrese
- pretpostavlja se da je početna adresa u svakom slučaju nula
- prije izvođenja program je potrebno pretvoriti u apsolutni oblik (to vremenski košta, pa se ovo danas ne radi)

↳ možemo problem riješiti ovako:

- izvođenje više procesa definira više PARTICIJA glavnog spremnika s različitim početnim adresama
- programi se prilagođavaju za određenu particiju (s određenom početnom adresom)

↳ ovo je STATIČKO raspoređivanje jer se program uvijek izvodi u onoj particiji za koju je pripremljen

⇒ FRAGMENTACIJA (kod RAM-a): pojava slobodnog prostora koji se ne može iskoristiti

↳ kod statičkog raspoređivanja imamo dvije vrste fragmentacije:

a) UNUTARNJA - neiskorišten dio unutar jedne particije
b) VANJSKA - nema procesa primjerenih za slobodne particije

B) DINAMIČKO RASPOREĐIVANJE SPREMNIKA

↳ IDEJA: pribrojaj početne adrese se dalje sklopovski u procesoru u dinamici (tijekom rada programa)

↳ na taj način program unijeh ostaje u relativnom obliku

↳ također, program vidi čitav logički adresni prostor (za 32-bita je to 4GB)

↳ program se sada može smjestiti na bilo koju početnu adresu (fizički adresni prostor - adrese koje program stvarno zauzme)

↳ hardverski rahtjevi (dodaci procesoru):

- ZBRAJALO - pribroja početnu adresu na logičke adrese
- BAZNI REGISTAR - pamti početnu adresu aktivnog procesa

↳ ovo je začetak „memory menagement unit"-a

↳ bazni registar postaje dio konteksta dretve (zajednički za isti proces)

⇒ PROBLEM: neovisnost procesnih adresnih prostora (kako zaštititi druge procese od aktivnog)

↳ rješenje sklopovsko:

- REGISTAR OGRADE: predstavlja gornju granicu procesnog adresnog prostora

↳ NAPOMENA: danas postoje posebni barem i ograđeni registri za različite segmente procesa

↳ to su: instrukcijski, stogovni, podatkovni

↳ FRAGMENTACIJA RADNOG SPREMNIKA:

↳ nastanak „rupa" između raspoređenih adresnih prostora

↳ smanjenje fragmentacije:

1.) oslobađanjem raspoređenog prostora rupe se spajaju

2.) traži se najbolje prilagođenje rupe procesu

3.) skupljanje otpadaka (GARBAGE COLLECTION)

↳ dijelovi se izvođenje i raspoređeni prostori se premještaju u kompaktni niz raspoređenih procesnih prostora

⇒ KNUTOVO 50%-tno PRAVILO:

↳ pitanje: koliki je broj rupa u dinamičkom raspoređivanju uz zadani broj procesa?

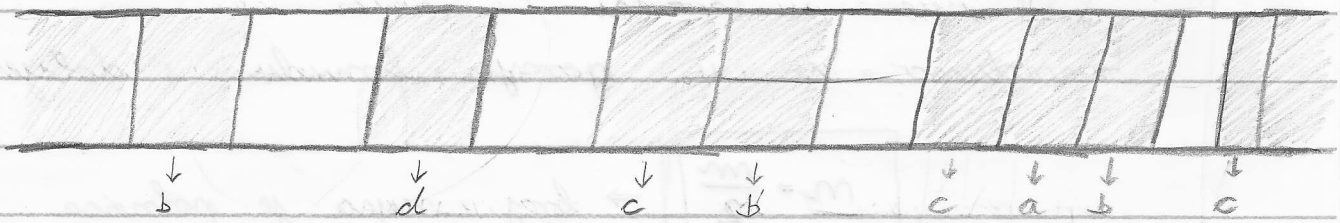
↳ PRETPOSTAVKE:

- $P_O = P_Z$ ⇒ vjerojatnost oslobađanja bloka jednaka je vjerojatnosti zauzimanja
- m → blokovi (broj)
- n → rupe (broj)

↳ nas zanima:

$$n = f(m) = ?$$

↳ skiciramo proizvoljni sadržaj memorije:



$$m = a + b + c + d$$

$$\left.\begin{array}{ll} \forall b \Rightarrow & 1 \text{ rupa} \\ \forall c \Rightarrow & 1 \text{ rupa} \\ \forall d \Rightarrow & 2 \text{ rupe} \end{array}\right\} \quad n = \frac{b + c + 2d}{2}$$

⇒ pretpostavimo da mora vrijediti:

- $b = c$

↳ gornja formula za „n" sada je:

$$n = \frac{2b + 2d}{2} = b + d$$

⇒ sada treba pogledati vjerojatnost:

1) povećanje broja rupa:

$$p(n++) = p_0 \cdot \frac{a}{m}$$

2) smanjenje broja rupa:

$$p(n--) = p_0 \cdot \frac{d}{m} + \overbrace{p_{\geq} \cdot q}^{\approx 0}$$

⇒ m se nalazimo u stohastičkoj ravnoteži, pa vrijedi:

$$p(n++) = p(n--)$$

OKRENI →

↳ iz ove jednakosti slijedi:

$$a = d$$

↳ vratimo se u gornju formulu i dobijemo:

$$\boxed{n = \frac{m}{2}} \Rightarrow \text{broj rupa je polovica broja procesa u memoriji}$$

(velik broj, pa tražimo bolje rješenje)

C) PREKLOPNO RASPOREĐIVANJE SPREMNIKA:

↳ IDEJA: program podijelimo na:

a) osnovni dio - uvijek u spremniku

b) zamjenjivi dijelovi - u spremniku po potrebi

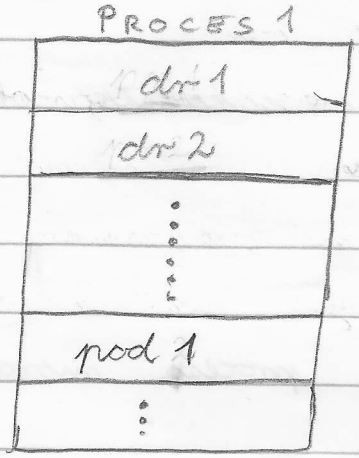
↳ ovo bi trebao raditi programer no današnji OS-ovi to mogu sami pomoću načina **strančenja**

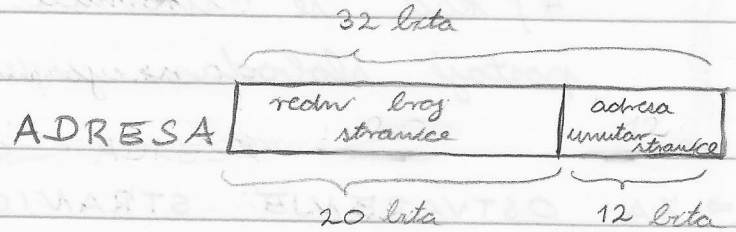
STRANIČENJE

⇒ današnje rješenje raspoređivanja spremnika koji se radi na način preklapanja

⇒ FAP – fizički adresni prostor (apsolutne adrese)
⇒ LAP – logički adresni prostor (relativne adrese)

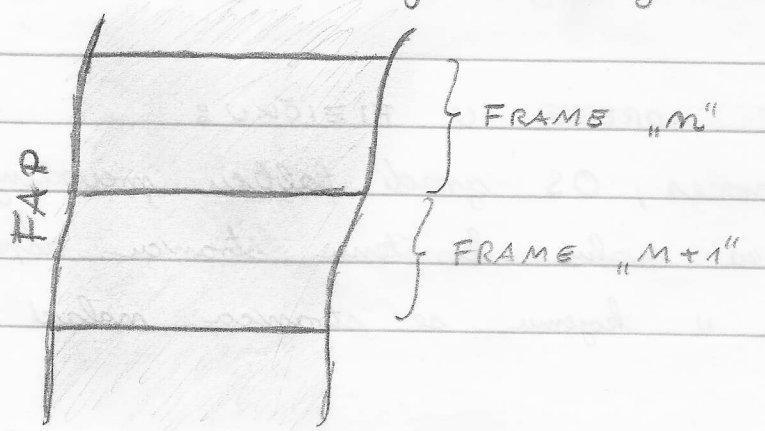
⇒ procese dijelimo na stranice:





⇒ LAP se dijeli na stranice
⇒ FAP se dijeli na okvire

⇒ veličina okvira jednaka je veličini stranice



⇒ IDEJE STRANIČENJA:

1) stranice LAP-a se automatski pohranjuju u okvire (kada ih proces ratrađ)
   ↳ ovo omogućuje postojanje više procesa
2) stranica se može staviti u bilo koji okvir
   ↳ ovo eliminira fragmentaciju (osim ranemarive fragmentacije unutar stranice)
3) stranica se može privremeno izbaciti i pohraniti na disk
   ↳ ovo omogućuje da LAP može biti veći od FAP-a
4) kad se stranica više ne koristi, okvir postaje slobodan

⇒ ZA OSTVARENJE STRANIČENJA:

a) za svaku stranicu svakog procesa moramo znati u kojem je okviru
b) pretvorba logičke u fizičku adresu mora biti rješena sklopovski u procesoru
c) koju stranicu privremeno izbaciti ako trebamo slobodan okvir?
   ↳ za ovo postoje razni algoritmi

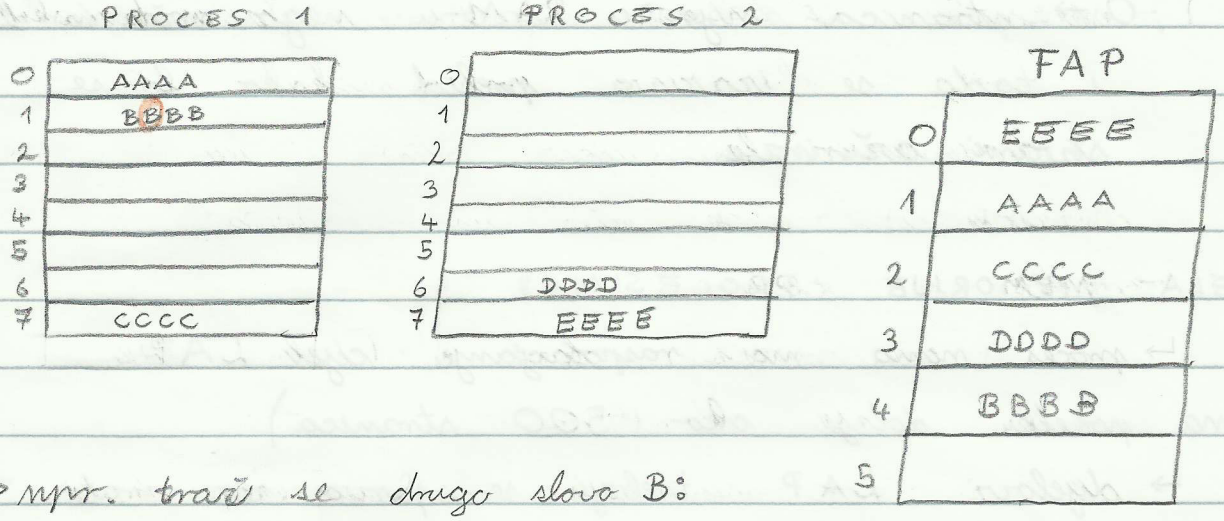
⇒ PRETVORBA LOGIČKE ADRESE U FIZIČKU:

↳ za svaki proces, OS gradi tablicu prevođenja
↳ u tablici se za svaku korištenu stranicu zapisuje adresa okvira u kojemu se stranica nalazi

PR: Imamo sljedeću strukturu:

L.A.P = 8 stranica s po 4 bajta (2^5 adresa)

F.A.P = 6 okvira

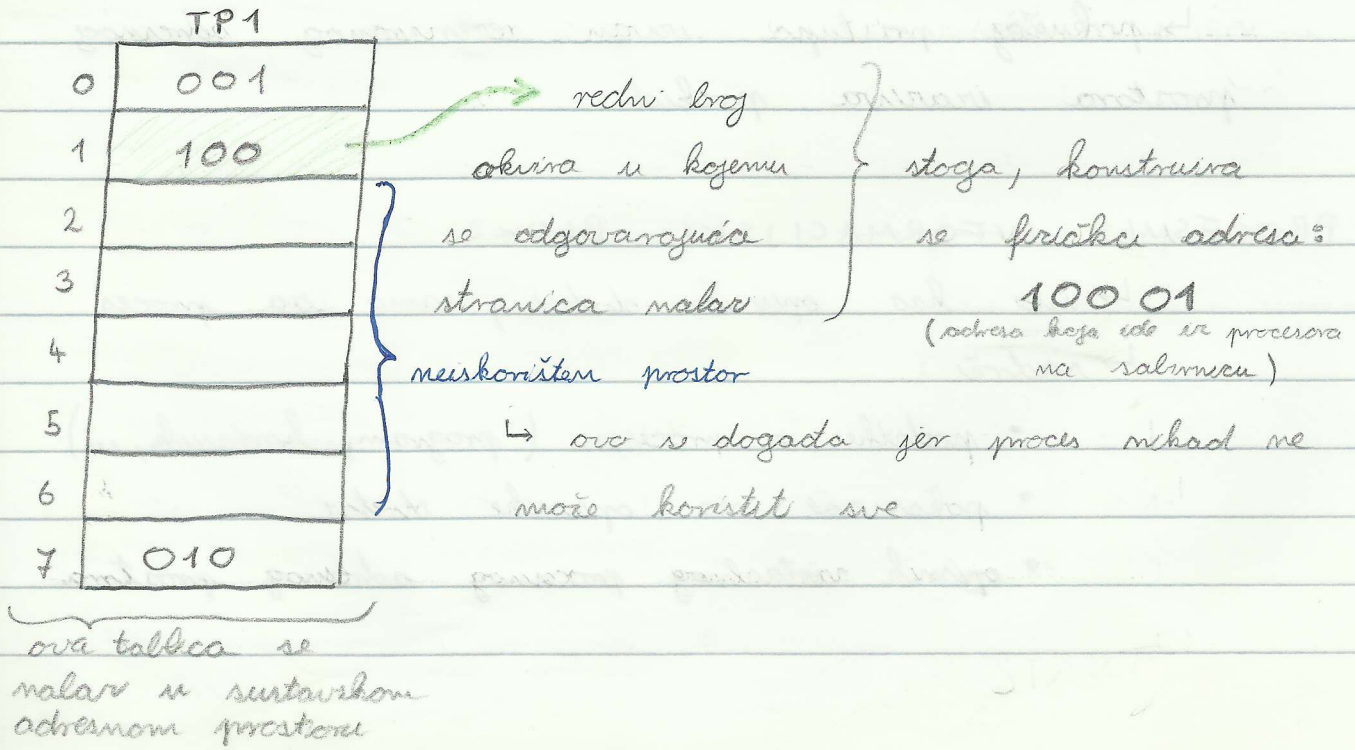


⇒ npr. traži se drugo slovo B:

↳ logička adresa: 00101

- 001 → redni broj stranice
- 01 → adresa unutar stranice

⇒ tablica prevođenja procesa 1:



⇒ u svakom retku tablice prevođenja nalazi se i **bit prisutnosti** koji označava je li stranica kopije u radnoj memoriji:

1 → stranica je u RAM-u

0 → stranica nije u RAM-u nego možda na disku i tada se izaziva **prekid** kako bi se stvari ažurirale

⇒ DODJELA MEMORIJE PROCESIMA:

↳ proces nema na raspolaganju cijel LAP (na početku dolije oko 500 stranica)

↳ dijelovi LAP trebaju se prvo rezervirati (to se rad funkcijama)

↳ samo rezervirani dijelovi LAP-a mogu se dodijeliti

↳ OS raspisuje tablicu rezerviranog adresnog prostora za svaki proces

↳ pokušaj pristupa izvan rezerviranog adresnog prostora izaziva prekid

⇒ PROCESNI INFORMACIJSKI BLOK:

↳ isto kao opisnik dretve, samo za proces

↳ sadrži:

- podatke o procesoru (program, kontrolka, ...)
- pokazivače na opisnike dretvi
- opisnik virtualnog procesnog adresnog prostora

↳ nadalje, opisnik virtualnog procesnog adresnog prostora sadrži:

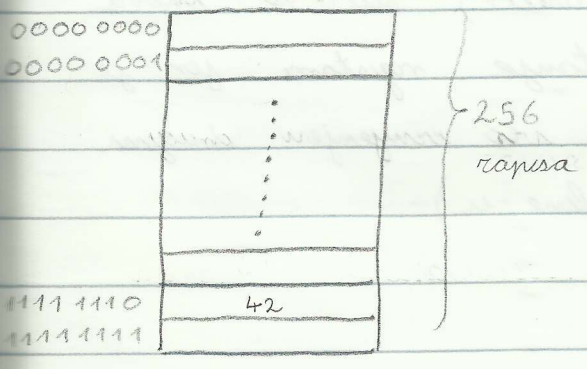
- tablica rezerviranog logičkog adresnog prostora
- smještaj dodijeljenih stranica na disku
- tablica prevođenja

⇒ VIŠERAZINSKA TABLICA PREVOĐENJA:

↳ uzmimo primjer logičke adrese:

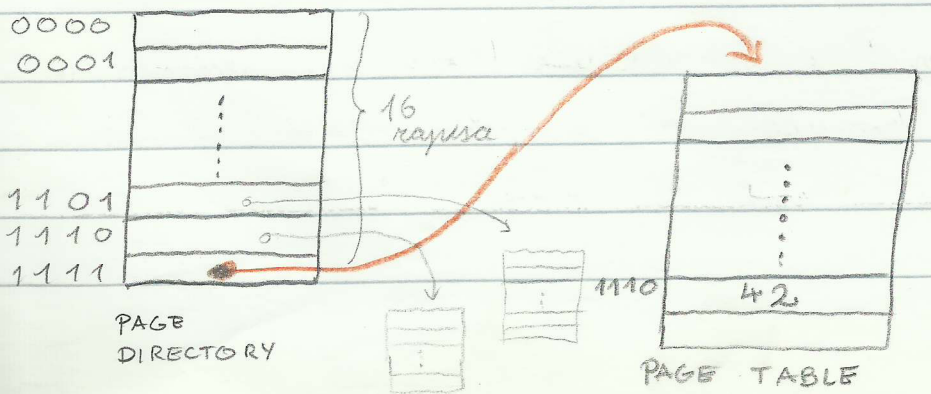
LA: 1111 1110 .....

↳ jednorazinska tablica prevođenja:



⇒ sada imamo 256 zapisa, a ustvari koristimo samo jedan ili dva zapisa (dvorazinska tablica prevođenja stoga štedi memoriju)

↳ iz razloga takvog slabog iskorištenja, podijelimo ovu tablicu na dvije od kojih svaka koristi 4 bita:



⇒ na prezentacijama pogledaj sklopovsku podršku pretvorbe logičke adrese u fizičku

⇒ STRANIČENJE NA ZAHTJEV:

↳ prilikom adresiranja nepostojeće stranice događa se prekid - nakon ove vrste prekida, prekinuta instrukcija se mora ponoviti jer se željeni podatak nije uspio učitati

↳ procesor koji podržavaju straničenje, imaju skrivene kopije svih registara (stanje svih registara na početku izvođenja instrukcije)

↳ ovo nam je potrebno zbog „pipelne" strukture (cijeli pipelne se pravi prilikom krivog adresiranja stranice, ali - se mora i obnoviti staro stanje registara jer je moguće da su ovi već izmijenjeni drugim naredbama u pipelne-u)

STRATEGIJE ZAMJENE STRANICA

⇒ čista stranica ⇒ stranica čija kopija već postoji na hard-disku, pa je prilikom mičanja s RAM-a ne treba spremati na disk

↳ u onih preostalih 12 bitova tablice prevođenja nalaze se BITOVI STANJA STRANICA:

- bit prisutnosti (V)
- bit čistoće (D) - nula ako je stranica čista
  - postavlja se hardverski
- bit pristupa (A) - postavlja se ako se stranica koristila
  - bit pristupa se periodički resetira
- bit dijeljenja (GL) - stranica je u shared-memoryju
  - više procesa može pristupiti stranici

⇒ STRATEGIJE ZAMJENE STRANICA: (teorijske strategije)

a) FIFO ⇒ izlacuje se stranica koja je najdulje u spremniku

b) LRU („Least Recently Used") ⇒ izlacuje se stranica koja se najdulje nije koristila

c) LFU („Least Frequently Used") ⇒ izlacuje se stranica koja se najmanje puta koristila

d) OPT („Optimalna strategija") ⇒ izlacuje se stranica koja se najdalje u budućnosti neće koristiti

ZAD: U sustavu sa straničenjem proces veličine 400 riječi generira slijed adresa:

1 1 7 2 3 1 8 4 1 5 6 3 8

23, 47, 333, 81, 105, 1, 400, 157, 30, 209, 289, 149, 360

Program ima na raspolaganju 200 riječi radne memorije. Nadi stanja okvira veličine 50 riječi za strategije:

a) FIFO

b) LRU

c) LFU

d) OPT

↳ pretpostavimo da imamo 4 okvira u RAM-u, a program koristi 8 stranica (broj stranice je označen plavom bojom)

	1	1	7	2	3	1	8	4	1	5	6	3	8
a) FIFO	1	1	1	1	1	1	[8]	8	8	8	[6]	6	6
	-	-	[7]	7	7	7	7	[4]	4	4	4	[3]	3
11 PROMAŠAJA	-	-	-	[2]	2	2	2	2	[1]	1	1	1	[8]
	-	-	-	-	[3]	3	3	3	3	[5]	5	5	5
b) LRU	1	1	1	1	1	1	1	1	1	1	1	1	[8]
	-	-	[7]	7	7	7	[8]	8	8	8	[6]	6	6
10 PROMAŠAJA	-	-	-	[2]	2	2	2	[4]	4	4	4	[3]	3
	-	-	-	-	[3]	3	3	3	3	[5]	5	5	5
c) LFU	1	1	1	1	1	1	1	1	1	1	1	1	1
	-	-	[7]	7	7	7	[8]	[4]	4	[5]	[6]	6	[8]
9 PROMAŠAJA	-	-	-	[2]	2	2	2	2	2	2	2	2	2
	-	-	-	-	[3]	3	3	3	3	3	3	3	3

(svejedno izbacujemo 7 il 2, ali u dogovoru izbacujemo onu koja je prva odonga

		1	1	7	2	3	1	8	4	1	5	6	3	8
d) OPT		[1]	1	1	1	1	1	1	1	1	[5]	[6]	6	6
8 PROMAŠAJA		-	-	[7]	7	7	7	[8]	8	8	8	8	8	8
		-	-	-	[2]	2	2	2	[4]	4	4	4	4	4
		-	-	-	-	[3]	3	3	3	3	3	3	3	3

↳ DODATNI ZADATAK:

- prikažite trenutni izgled tablice prevođenja na kraju primjene optimalne strategije zamjene stranica (uključi bit prisutnosti)

	ADRESA OKVIRA	BIT PRISUTNOSTI
1		0
2		0
3	4	1
4	3	1
5		0
6	1	1
7		0
8	2	1

⇒ NAPOMENA: primjet da je u svim strategijama isto ponašanje sve dok se svi okviri ne ispune sa stranicom (do isprekidane okomite crte)

⇒ NAPOMENA: u stvarnim sustavima se ove strategije ne koriste (FIFO je prespor, a ostale su prerahtjevne i skupe)

⇒ STRATEGIJE IZMJENE STRANICA: (stvarne implementacije)

a) **skloporska izvedba** - bit pristupa i bit čistoće se koriste za određivanje stranice za izbacivanje

b) **satni algoritam** („Clock algorithm", „Second chance")

↳ u osnovnom tipu se pojavljuje u današnjim operacijskim sustavima

↳ opis rada:

- stranice se obilaze u kružnoj listi
- ako je bit pristupa nula, stranica se izbacuje
- ako je bit pristupa jedan, on se postavlja na nula i kreće se na sljedeću stranicu

ZAD: Prikaži rad algoritma na raljenima iz prethodnog zadatka!

	1	1	7	2	3	1	8	4	1	5	6	3	8
CLOCK	[1]	1	1	1	1	1	1	1	1	1	1	1	1
	-	-	[7]	7	7	7	[8]	8	8	8	[6]	6	6
	-	-	-	[2]	2	2	2	[4]	4	4	4	[3]	3
	-	-	-	-	[3]	3	3	3	3	[5]	5	5	[8]
BIT PRISTUPA	1	1	0	0	0	1	0	0	1	1	0	0	0
	0	0	1	0	0	0	1	0	0	0	1	0	0
	0	0	0	1	0	0	0	1	1	0	0	1	0
	0	0	0	0	1	1	0	0	0	1	0	0	1

↳ primjeti da se algoritam ponaša kao FIFO, ako nemamo dodatnih informacija o ostalim stranicama (bitovi pristupa su svi u nuli)

RASPODJELA OKVIRA

⇒ operacijski sustav vodi evidenciju o korištenim okvirima
↳ za svaki okvir postoji OPISNIK OKVIRA
(on npr. sadrži stanje okvira)

⇒ STANJA OKVIRA:
- **aktivno** - dodijeljen nekom procesu
- **slobodno stanje** - nakon izlaćivanja stranice
- **obrisano stanje** („zeroed") - okvir je spreman za dodjeljivanje
- **prenošenje stranice na disk** - prilikom oslobađanja
- **stanje brisanje** - stranica se trenutno briše iz okvira
- **neispravan okvir** - ova stanja imaju samo napredni OS-ovi za slučaj da se dio RAM-a pokvari

⇒ broj okvira koji se dodjeljuje nekom procesu je početno ograničen kako neki proces neki ne zauzeo sve okvire

⇒ „WORKING SET" → okviri trenutno dodijeljeni nekom procesu
→ povećava se ili smanjuje u odnosu na na evidentiran broj promašaja

⇒ struktura podataka jezgre OS-a, trajno je smještena u rezervirane okvire

PROGRAMIRANJE UZ STRANIČENJE

⇒ API funkcijama moguće je utjecati na raspodjelu okvira i algoritama zamjene stranica

⇒ WINDOWS FUNKCIJA: virtual lock ()

↳ naputak OS-u da neku stranicu ostavi u RAM-u

⇒ varijable strukture podataka i algoritmi mogu imati veliki utjecaj na performanse

↳ npr. lokalnost podataka prilikom korištenja

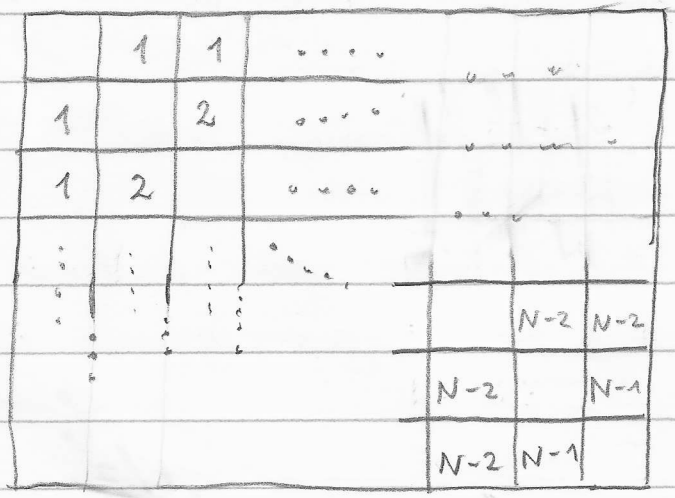
ZAD: U sustavu sa straničenjem, veličina okvira je N riječi, a straničenje se vrši prema LRU. Matrica A[N][N] smještena je po retcima. Koliko promašaja će izazvati zadani program ako za matricu A postoji:

a) 1 okvir
b) 2 okvira
c) 3 okvira
d) N okvira

ZADANI PROGRAM:

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

⇒ skica pristupa zadanog algoritma:



a)

ITER	$i=1$	$i=2$	.....	$i=N-1$
STRANICA/OKVIR	1 2 1 3 ... 1 N	2 3 2 4 ... 2 N	....	N-1 N
BROJ PROMAŠAJA	$2\cdot(N-1)$	$2(N-2)$	....	$2\cdot 1$

↳ ukupno: $2\cdot\underbrace{\left[(N-1)+(N-2)+\ldots+2+1\right]}_{\text{suma } N-1 \text{ prirodnih brojeva}}=\frac{(N-1)N}{2}\cdot 2=N(N-1)$

b)

ITER	$i=1$	$i=2$	...	$i=N-2$	$i=N-1$
STRANICA	1 2 1 3 ... 1 N	2 3 2 4 ... 2 N	...	N-2 N-1 N-2 N	N-1 N
OKVIRI	[1] 1 1 ... 1 – [2] [3] [N]	[2] 2 2 ... 2 N [3] [4] [N]	...	N-2 N-2 N-2 N N-1 N	N-1 N
BROJ PROMAŠAJA	N	N-1	...	3	1

↳ ukupno: $N+(N-1)+\ldots+3+1=\underbrace{\frac{N(N+1)}{2}}_{\text{suma prvih } N \text{ prirodnih brojeva}}-2$

c)

ITER	$i=1$		...	$i=N-4$	$i=N-3$
STRANICA	1 2 1 3 ... 1 N	2 3 2 4 ...	...	....N-4 N	N-3 N-2 N-3 N-1 N-3 N
OKVIRI	[1] 1 1 ... 1 – [2] 2 ... N-1 – – [3] [N]	1 [3] 3 [2] [2] 2 .... N N [4]	...	N-4 N N-1	N-4 [N-2] N-2 [N] N N [N-1] N-1 N-3 N-3 N-3 N-3
BROJ PROMAŠAJA	N	N-1	...		4

ITER	$i=N-2$	$i=N-1$
STRANICA	N-2 N-1 N-2 N	N-1 N
OKVIRI	N N [N-2] N-2 N-3 [N-1]	
BROJ PROMAŠAJA	2	0

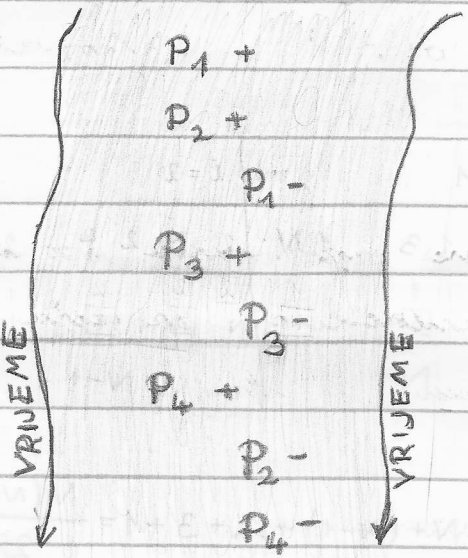
↳ ukupno: $\frac{N(N+1)}{2}-4$

d) jednostavno:
N promašaja

ZAD: Upravljanje radnim spremnikom !

P_1: 5 MB
P_2: 8 MB
P_3: 3 MB
P_4: 10 MB

redoslijed pokretanja:
+ ⇒ pokretanje
- ⇒ raspletak



Na raspolaganju imamo 20 MB memorije, pa treba prikazati stanje memorije za raspoređivanje:

a) statičko (segment 10MB)
b) dinamičko
c) straničenje (stranica 1MB)

⇒ NAPOMENA: kod straničenja prvo dodjeljujemo obrisane okvire, a tek nakon toga brišemo i dodjeljujemo slobodne

DATOTEČNI PODSUSTAV

⇒ to je dio OS-a zadužen za organizaciju datoteka na vanjskim spremnicima

↳ OS zadaje čitanje nekog sektora i on (a ne disk ?) interpretira što se čita

⇒ ULOGE DATOTEKA:

- trajna pohrana podataka
- komunikacija između procesa (odviše spora komunikacija, ali se nekad koristi)

⇒ o formatu datoteka se ne brine OS već sam programer

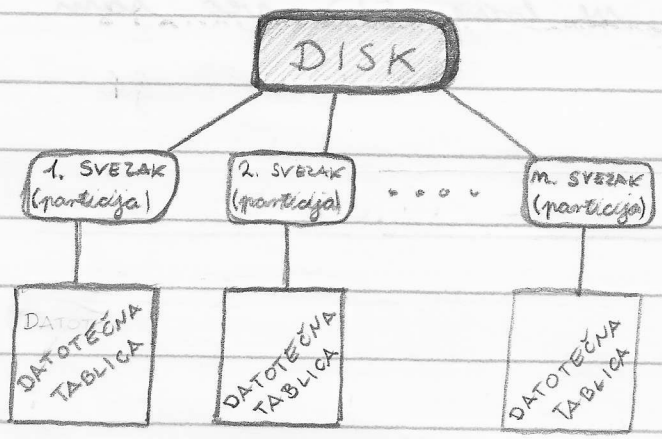
SMJEŠTANJE DATOTEKA NA DISKU

⇒ ovo je radaća operacijskog sustava

⇒ OPISNIK DATOTEKE:

- naziv, vrsta, vlasnik, prava otvaranja, vrijeme radnje promjene, ...
- opis smještaja datoteke na disku (po ovome se razlikuju file-sustavi znatno razlikuju)

⇒ ORGANIZACIJA DISKOVNOG PROSTORA:



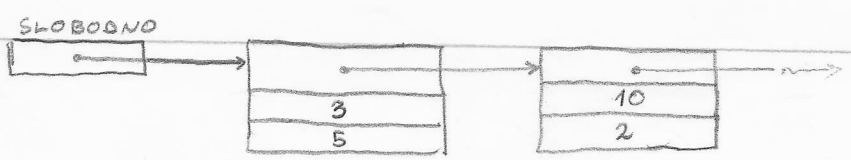
⇒ datotečna tablica:

- opis svih prostora particije u kojem se nalaze opisnici datoteke
- opis slobodnog prostora

⇒ opis slobodnog prostora može biti:

a) bitovni prikaz (0 - sektor zauzet, 1 - sektor slobodan)

b) lista slobodnih blokova:



⇒ datotečna tablica nalaz se uvijek na istom mjestu kako bi joj OS mogao uvijek lagano pristupiti

⇒ SMJEŠTAJ DATOTEKE:

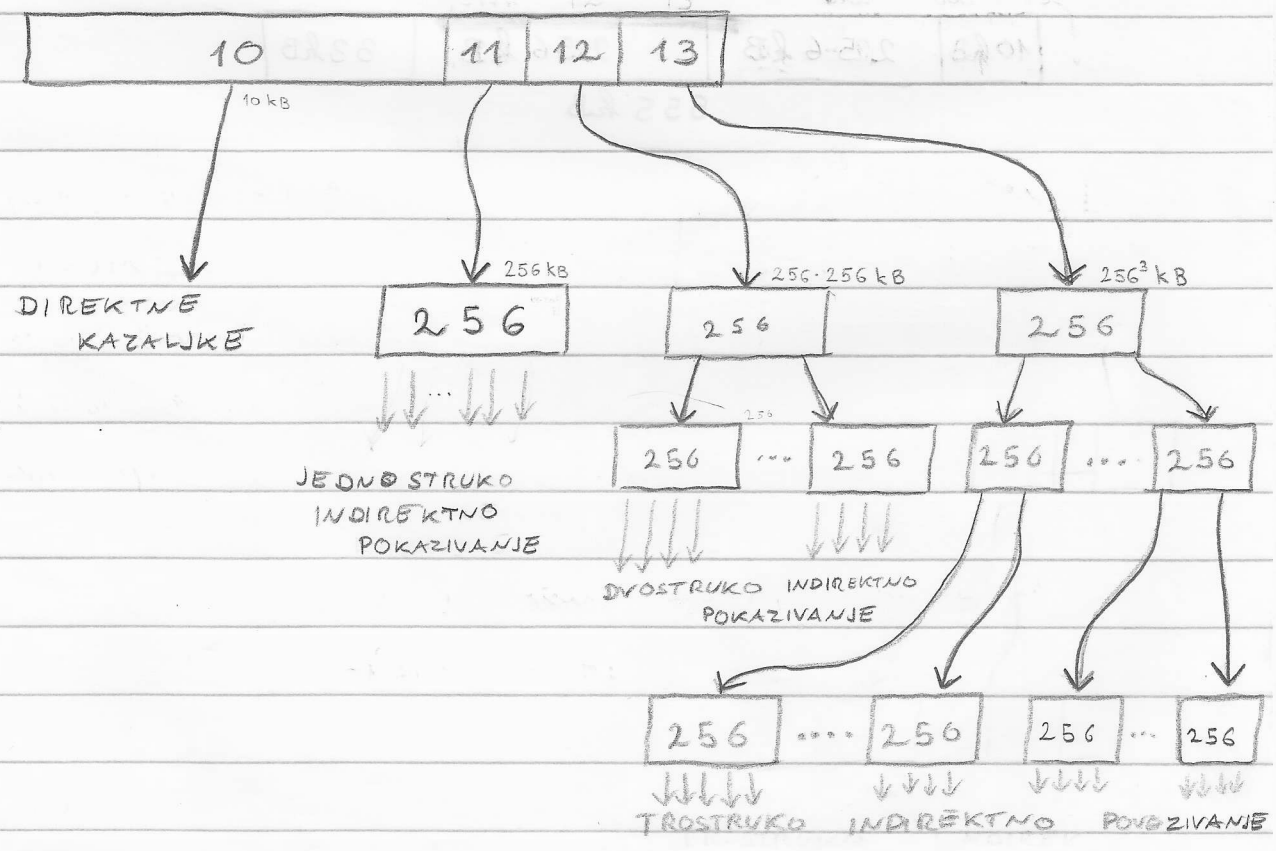
↳ datoteka se dijeli na blokove koji na disku mogu biti zapisani proizvoljnim redom

↳ za svaki blok datoteke postoji kazaljka na smještaj tog bloka na disku (slično tablici prevođenja)

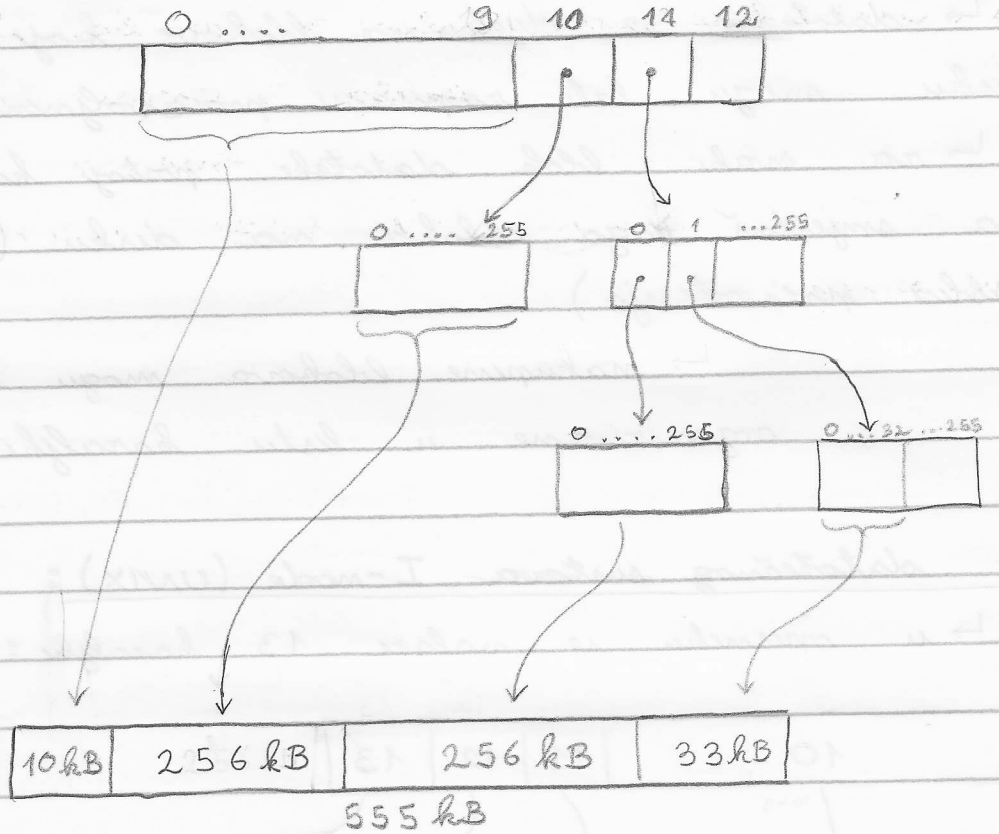
↳ nakupine blokova mogu biti organizirane u listu kazaljki

⇒ <u>primjer datotečnog sustava I-node (UNIX):</u>

↳ u opisniku se nalaze 13 kazaljki:



ZADO: U I-node datotečnom sustavu je pohranjena datoteka veličine 555 kB, koliko prostora zauzimaju karalyke za opis smještaja te datoteke. Veličina bloka je 1 kB, a karalyka je 32-bitna.



⇒ primjer datotečnog sustava NTFS:

↳ MFT (Master File Table) ⇒ sadrži opisnike datoteka

↳ LCN (Logical Cluster Number) ⇒ redni broj bloka na disku (to je zapravo fizička adresa)

↳ VCN (Virtual Cluster Number) ⇒ redni broj bloka unutar datoteke (to je zapravo logička adresa)

↳ pogledaj primjer rada NTFS sustava s prezentacije !

POSLUŽIVANJE ZAHTJEVA ZA PRISTUP DISKU

⇒ odredivanje redoslijeda pogonskih zahtjeva za pristup disku je bitna komponenta operacijskog sustava jer se pokazalo da FCFS nije najbolje rješenje

⇒ ciljevi poboljšati:

- PROPUSNOST („Throughput") ⇒ broj odradenih zahtjeva u jedinici vremena
- ODZIV („Response Time") ⇒ prosječno vrijeme čekanja na željeni podatak

⇒ najveći utrošak vremena predstavlja traženje željene staze (pomak glave)

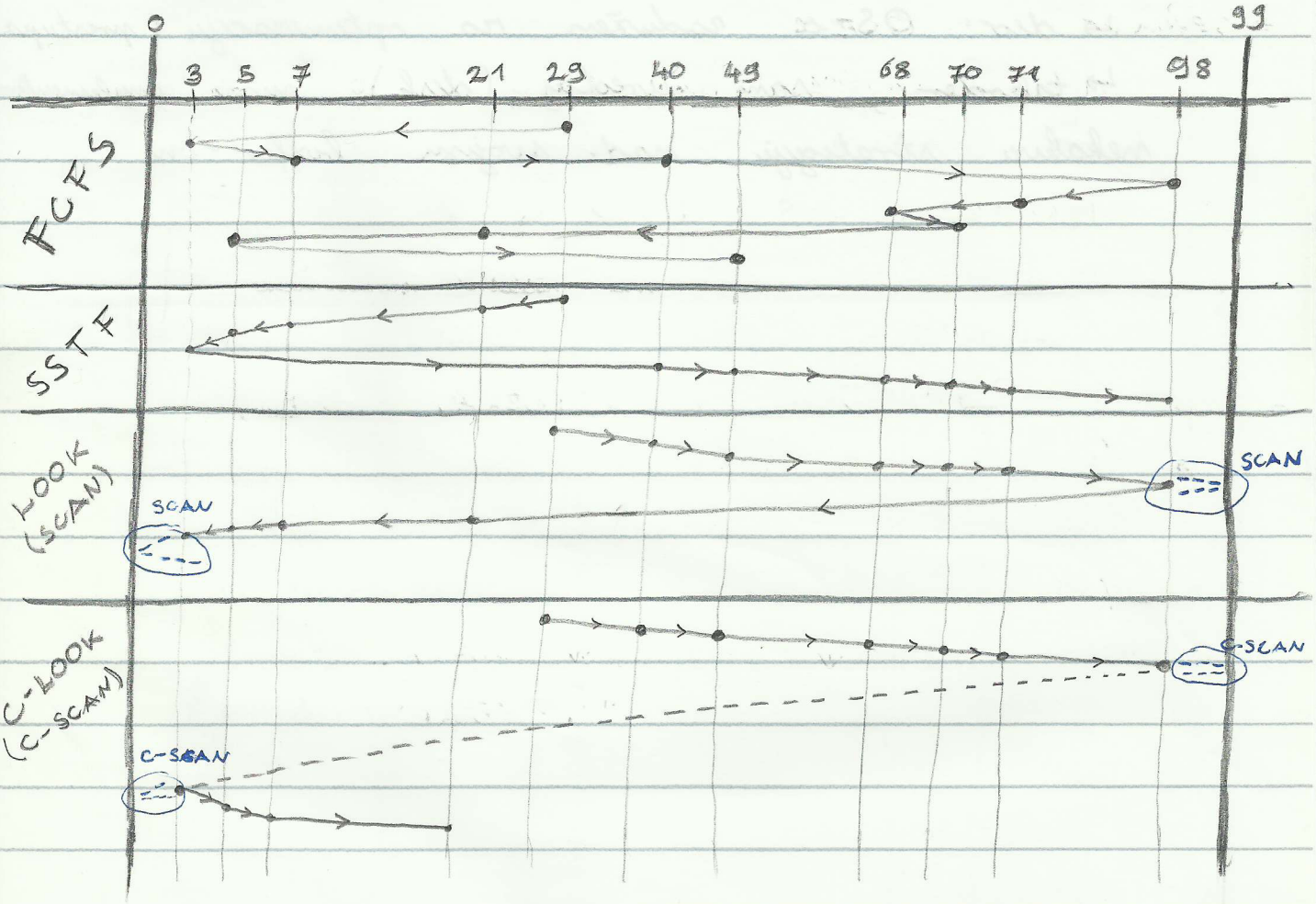
⇒ <u>strategije:</u>

- FCFS - poslužiranje redom prispjeća
- SSTF - „Shortest Seek Time First" (poslužiranje s najkraćim vremenom traženja)
- LOOK (SCAN) - „lift algoritam" (poslužuje algoritme ovisno o tome kamo ide)
- C-SCAN - „circular scan" (cirkularno look/scan poslužiranje)

ZAD 5 Disk ima 100 staza (0-99). Glava se trenutno nalazi na stazi 29 s tim da je prije bila na stazi 8. Zahtjevi za pristup pojedinim stazama po redu prispijeća su:

3, 7, 40, 98, 71, 68, 70, 21, 5, 49

Grafički prikaži kretanje glave na čitanje za sve 4 nabrojane strategije i izračunaj ukupni pomak glave za svaku strategiju!



⇒ primjeti da SSTF može uzrokovati izgladnjivanje

⇒ neki razlika između LOOK i SCAN strategija

⇒ U STVARNOJ IMPLEMENTACIJI:

↳ strategije mogu vratiti u obzir i ograničeni broj nasrtajućih zahtjeva

Pr: Poslužiranje za primjer zadatka pristupom SSTF pri tome promatrano 4 zahtjeva:

29 → 40 → 7 → 3 → 68 → ...

⇒ TKO IMPLEMENTIRA STRATEGIJU:

↳ dio OS-a zadužen za optimizaciju pristupa

↳ također, sam uređaj (disk) može implementirati nekakvu strategiju nad svojim buffer-om

PONAVLJANJE

ZAD: U sustav dolaze atomi vodika i kisika (dvoje) koji moraju čekati dok se u sustavu ne pojave svi atomi za molekulu vode. Tek kad se pojave sve potrebne čestice, atomi mogu početi funkciju za tvorbu vode. Sinkronizirati dretve:

a) monitorom b) semaforom

a) → trebamo monitorsk semafor (monitor M)

→ trebamo 2 reda uvjeta: O, H

→ trebamo dvije varijable BR_H, BR_O = 0

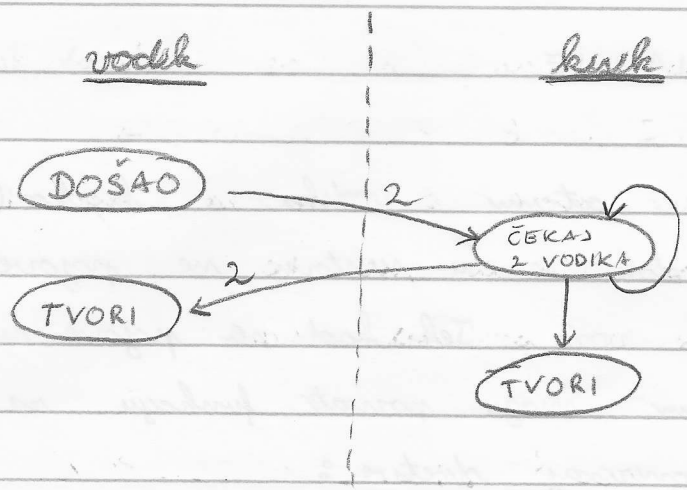
VODIK

```
M_LOCK(M);
BR_H++;
AKO (BR_H >= 2 && BR_O >= 1)
    C_SIGNAL(O);
    C_SIGNAL(H);
    BR_H -= 2;
    BR_O--;
INAČE
    C_WAIT(M,H);
M_UNLOCK(M);
```

KISIK

```
M_LOCK(M);
BR_O++;
AKO (BR_H >= 2)
    C_SIGNAL(H);
    C_SIGNAL(H);
    BR_H -= 2;
    BR_O--;
INAČE
    C_WAIT(M,O);
M_UNLOCK(M);
```

b)



⇒ SEMAFORI: OSEM: H, v=0

OSEM: H2, v=0

BSEM: 0; v=1

```
VODIK
POSTAVI_OSEM(H);
ČEKAJ_OSEM(H2);
TVORI;
```

```
KISIK
ČEKAJ_BSEM(0);
ČEKAJ_OSEM(H);
ČEKAJ_OSEM(H);
POSTAVI_OSEM(H2);
POSTAVI_OSEM(H2);
POSTAVI_BSEM(0);
TVORI;
```

ZADO: Na nekom web-poslužitelju, prosječan broj zahtjeva u minuti je 100, dok ih on u minuti može obraditi 300. Na ist poslužitelj se proputa drugi zahtjevi kojih dolazi 60 u minuti. Koliko bi bilo trajanje radroživanja samo za druge zahtjeve, ako je prosječno radroživanje za obje vrste zahtjeva jednako jednoj sekundi? (POISSON + EKSPONENCIJALNA)

$\lambda_1 = 100/min$

$\mu_1 = 300/min$

$\lambda_2 = 60/min$

$\bar{T} = 1s$

$\bar{T}_2 = ?$

$$\bar{T} = \frac{1}{\mu - \lambda}$$

$$\lambda = \lambda_1 + \lambda_2 = 160/min$$

$$\varrho = \varrho_1 + \varrho_2 \Rightarrow \frac{\lambda}{\mu} = \frac{\lambda_1}{\mu_1} + \frac{\lambda_2}{\mu_2}$$

$$\frac{\lambda_2}{\mu_2} = \frac{\lambda}{\mu} - \frac{\lambda_1}{\mu_1} \Rightarrow \mu_2 = \frac{\lambda_2}{\frac{\lambda}{\mu} - \frac{\lambda_1}{\mu_1}}$$

$$\mu = \frac{1}{T} + \lambda$$

$$\mu_2 = \frac{\lambda_2}{\frac{\lambda}{\frac{1}{T} + \lambda} - \frac{\lambda_1}{\mu_1}} = 152.3/min$$

$$\bar{T}_2 = \frac{1}{\mu_2 - \lambda_2} = 0.65s$$

Zad: Redoslijed referentnih stranica je:

2, 1, 2, 3, 5, 3, 4, 5, 6, 3, 2, 6

Prikaži stanja 3 raspoloživa okvira za:

a) algoritam clock

b) optimalnu strategiju

	2	1	2	3	5	3	4	5	6	3	2	6
CLOCK	[2]	2	2	2		[4]		4	[3]	3		
	–	[1]	1	[5]		5		5	5	[2]		
	–	–	[3]	3		3		6	6	6		
bit pristupa	→1	0	1	1	0	0	→1	→1	0	→1	0	0
	0	→1	→1	0	→1	→1	0	1	0	0	→1	1
	0	0	0	→1	0	1	0	0	→1	0	0	1
OPTIMALNO	[2]	2	2	2		[4]		[6]		6		
	–	[1]	1	[5]		5		5		[2]		
	–	–	[3]	3		3		3		3		