Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Operacijski sistemi

21. 2. 2020
–

Xxxxxx xxxx
Mentor: Peter Peer
Peter.Peer@fri.uni-lj.si
Govorilne ure:

Vsebina

Glavni pojmi

Proces

Upravljanje s pomnilnikom

Razporejanje procesov

Datoteke

Zgodovinski razvoj

Jedro

Zamenjava konteksta

Procesni model - ima lahko kopico stanj. 2stanja 3...,

Procesna slika

Niti

Multiprogramiranje

Multiprocesiranje – imama več jeder

Sočasnost

Vzajemno izključevanje

Semafor

Navidezni pomnilnik

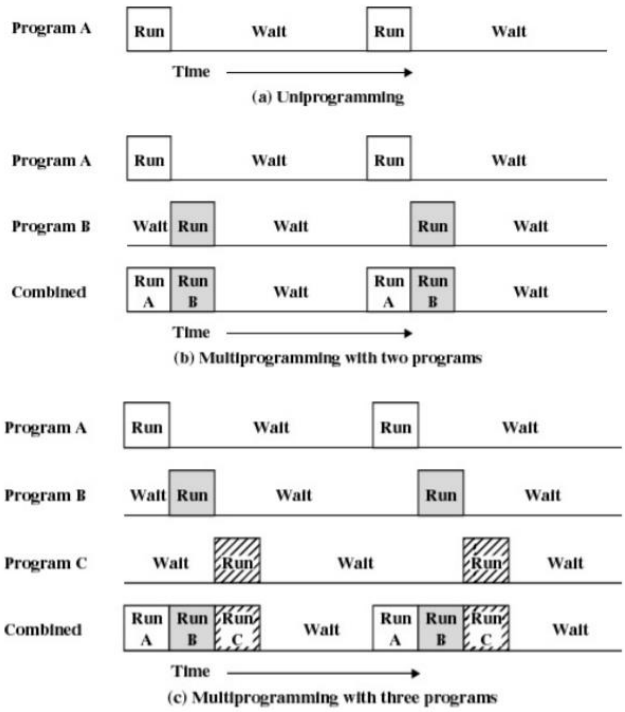
Upravljanje V/I

Multiprogramiranje se je uvedel, ker je bil procesor zelo malo uporabljen. S komunikacijo z trdim diskom je bila izkoriščenost 3,2%. Zato se uvede multiprogramiranje kateri cilj je da izkorišča čim več procesorske moči in se zato izvaja več programov skproti.

Je osnovna oblika vzporedne obdelabe programov, v kateri se več programov izvaja hkrati na enem procesorju.

Zaporedje izvajanje programov je odvisno od njihove prioritete. Izvaja vedno tistega z najvišjo prioriteto.

Uniprogramiranje je izkorišča virov kot bi jih lahko.



Glavni stebri delovanja računalniške arhiktekture (operacijskega sistema)

- Procesi in niti
- Upravljanje s pomnilnikom
- Razporejanje procesov
- Komunikacija z zunanjimi napravam (disk, V/I enote)

Operacijski sistem

- Je vmesnik med aplikacijami in strojno opremo
- Je program, ki nadzira delovanje programov

Namen operacijskega sistema

- Udobnost

- Učinkovitost
- Zmožnost razvoja

Monitor

Zagotavlja enako funkcionalnost kot semafor, samo da ga je lažje nadzarovati.

Enostavni sveženjski sistem

- Program, ki nadzoruje izvajojoče se aplikacije
- Ko aplikacija zaključi izvajanje se nadaljuje izvajanje monitorja
- Je rezidenčen v glavnem pomnilniku in vedno pripravljen na izvajanje

Željeni strojne lasnosti monotorja

- Zaščita pomnilnika
- Ura
- Priviligirani ukazi
- Prekinitve

Razvoj OS skozi leta

Bistvena problema (oba povezana s časom)

- Uporabnik sam **rezervira** potreben čas
- **Nameščanje** je vključevalo nalaganje gonilnika

Procesi

Osnovna naloga OS je upravljanje s procesi.

Kaj zahtevamo od operacijskega sistema?

- Ustvarjaj procese
- Prepletaj izvajanje procesov
- Zgodovina: maksimiziraj izrabo, minimiziraj odzivni čas
- Zasegaj vire na zahtevo procesov
- Zagotovi podporo za medprocesno komunikacijo. (da ne pride do bedarij)

Kaj je proces?

Proces je izvajanje individualnega programa (je izvajajujoči se program).

Za vsaj program, ki ga želimo izvršiti, se ustvari proces

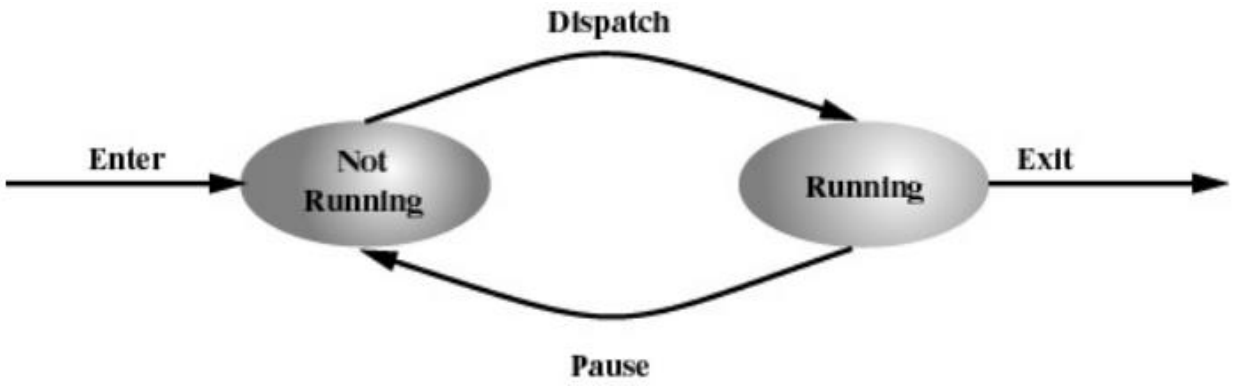
Proces je predstavljen s podatkovno strukturo v pomnilniku

Njeni glavni trije gradniki so:

- Koda (program)
- Vhodni podatki
- PCB process control blok (nadzorni blok procesa)

Enostaven procesni model dveh stanj

- Proces je lahko v enem izmed 2 stanj:
    - Izvajanje
    - Čakanje



Razlogi za zaključek procesa

- Uporabnik se odjavi
- Izhod iz aplikacije
- Napake pri izvrševanju
- Zaključek starševskega procesa
- Prekoračitev časovnega okvirja

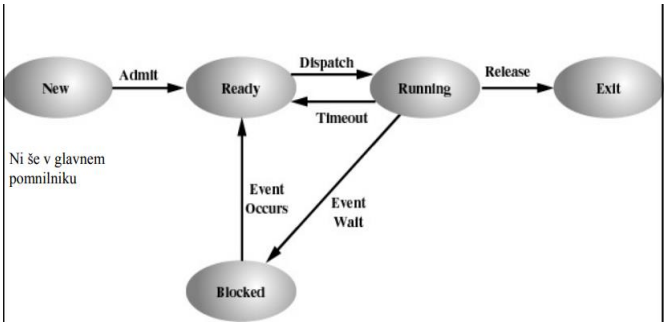
Problem procesnega modela dveh stanj

- Proces čaka
    - Lahko je pripravljen na izvajanje ali pa je:
        - Blokiran
            - Čaka na V/I

- Nemoremo vedno izbrati procesa, ki je prvi v vrsti, saj mogoče ni pripravljen na izvajanje.

Procesni model petih stanj

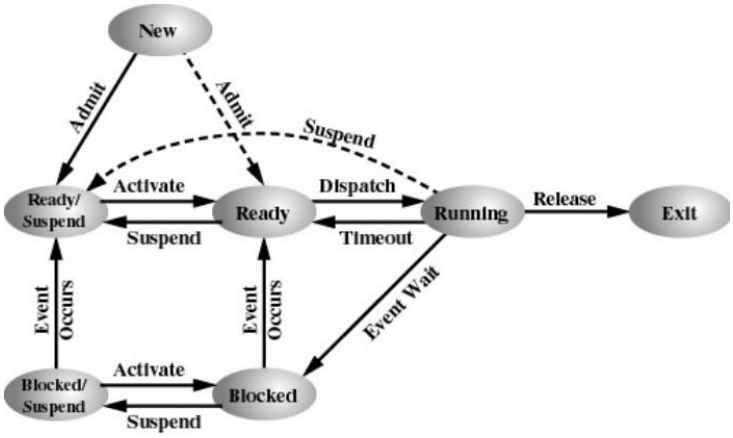
- Nov
- Pripravljen
- Izvajanje
- Blokiran
- Izhod



- Ker je procesor veliko hitrejši od V/I in če bi vsi procesi čakalo na njih bi nam hitro zmankalo pomnilnika (še posebej če nimamo navideznega)
- **Prestavi takšne procese na disk**, da se sprosti glavni pomnilnik za nove ali začasno odstranjenje procese.
- ko ga prestavimo na disk se stanje **blokiran** spremeni v stanje **začasno odstranjen** (novo stanje)
- **Pomankljivost 6 stanj:**
    - Začasno odstranjen proces gre v stanje pripravljenosti, pri tem pa ni nujno da je pripravljen na izvajanje.
        - Zato še eno novo stanje: **pripravljen ustavljen**

7 procesnih stanj

- Izvajanje
- Pripravljen
- Blokiran
- Nov
- Izhod
- Blokiran ustavljen (začasno odstranjen)
- Pripravljen ustavljen



Naslovni blok procesa – PCB

Vsebuje informacije procesa

1. oznake procesa (id)

2. informacije o stanju procesorja
    a. vidni registri (programsko dostopni registri)
    b. nadzorni in statusni registri (PC, CC, PSW)
    c. sistemski skladovni kazalec (last in frist out)
3. informacije za nadzor procesa
    a. informacije za razporejanje
    b. strukturiranje podatkov / povezanost procesov (glede na njihovo stanje in prioriteto)
    c. medprocesna komunikacija
    d. privilegiji procesa
    e. upravljanje s pomnilnikom
    f. lastništvo in koriščenost virov

Kako OS uporablja te strukture za nadzor procesov?

1. Kako ustvarimo proces?
    a. Ustvarimo indetifikator
    b. Postavimo proces v ustrezno vrsto
    c. Vzdržuj potrebne podatkovne strukture
2. Kdaj preklopimo med procesi?
    a. Če pride do prekiniteve - napaka pri dostopu do pomnilnika
    b. Past – nimam pravic do datoteke
    c. Nadzorni klici – odpiranje datoteke
3. Kaj se zgodi ob preklopu?
    a. Context switch – je proces shranjevanja stanja procesa ali niti, za nadalno uporabo. To je način, da si lahko več procesov deli eno jedro.
4. Kako ga lahko izvajamo?

Mikrojedro

Vsebuje osnovne funkcionalnosti, preko katerih komunicirajo vsi ostali deli (modluli).

Jedro mora zagotoviti:

- Preverjanja
- Predajanje sporočil
- Komunikacija s strojno opremo

Upravljanje pomnilnika:

Zagotovljamo okvir navidezne strani v fizično.

Aplikacija pošlje zahtevo (page fault), da dobi paket nazaj zahtevane strani.

Mikrojedro ve kam more naprej poslati zahtevo (to je njegovo delo) in to pošlje v pager. Pager komunicira s shemo navideznega pomnilnika.

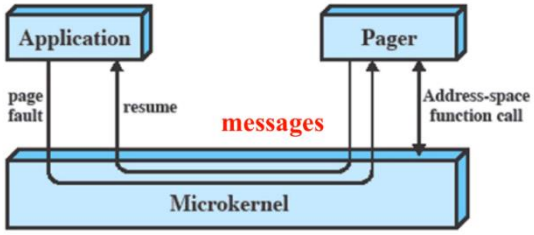


Figure 4.11 Page Fault Processing

Mikrojedro mora zagotoviti preverjanje in predajanje sporočil ter preslikavo strani/dostop do *hw*!

Prednosti mikrojedrne organiziranosti OS

- API – programski vmesnik
- Razširljivost – omogoča dodajanje novih storitev
- Fleksibilnost – odstrani nepotrebne funkcionalnosti, doda potrebne
- Prenosljivost
  - Prilogatitev mikrojedra ter njegovih 3 glavnih funkcionalostih
- Zanesljivost
  - Mikrojedra so majhna ter jih zato lahko zanesljivo testiramo
- Podpora porazdeljenemu sistemu
  - Da lahko pošlje ukaze kamorkoli naprej. Aplikacijo ne zanima kam pošlje ukaz, samo da dobi nazaj odgovor. To naslavlja mikrojedro
- Objektno usemrjeni OS – vsi moderni OS imajo implementacije objektov z jano definiramimi vmesniki, ki jih lahko sestavljamo kot legokocke

Organizacija procesov v Windowsih

Proces je zagotovo objekt. Vsaka stvar v Windowsih je prikazana kot objekt.

Dosegljivi (avaible) objekti:

- Za svoje delovanje potrebuje datoteko
- Section Z je deljene naslovni prostor

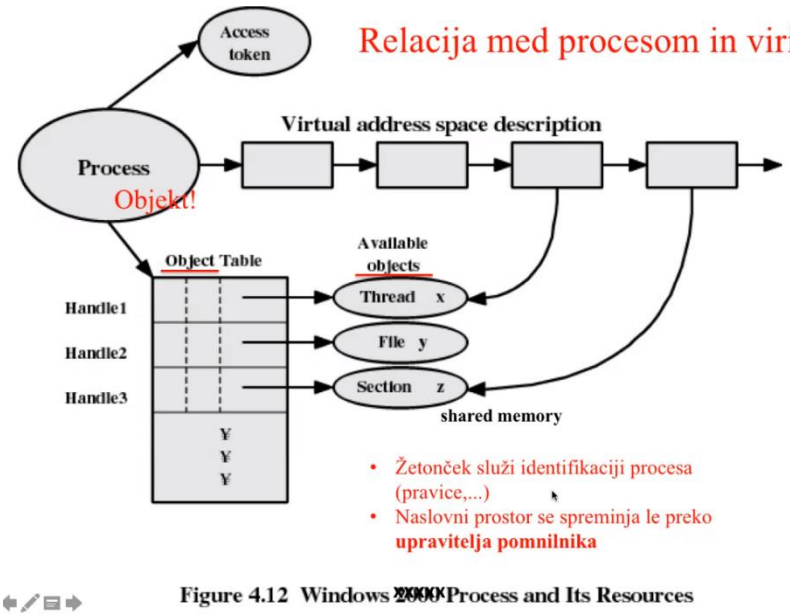


Figure 4.12 Windows 2000 Process and Its Resources

Acces token

- Tokeni so žetončki. V tem primeru varnostni žetončki
- Windows nam dodeli žetonček (kakššne imamo pravice) ta žetonček se predaja med vsemi procesi.
- so tudi objekti

Object table

1. Je zbirnik kazalcev, ki kaže na procese, ki jih pozna

Niti

Več jedrna arhitektura, simetrično multiprocesiranje in mikdorjedra.

Proces : nit

Proces ločimo od niti z vprašanjem kaj razporeja razporejevalnik (vzame funkcionalnosti in jo da na procesor). Nit je najmanjša enota izvajanja, en proces ima več niti, razporejamo v bistvu niti na procesor. Vire si pa lastijo posamezni procesi, namreč niti si same po sebi ne lastijo virov, vendar dostopajo do virov v procesu, kjer se nahajajo.

Prednosti niti

- **Hitrost ustvarjanja niti**
  - Ob ustvarjanju procesa – čas ustvarjanja nove nove niti je 70x hitrejši od ustvarjanje novega procesa
- **Hitrost zaključevanja niti**
  - Veliko hitreje kot procese
- **Preklopen čas**
  - Preklopi med procesi (context swich)
    - Vsebije veliko časovno potratnih korakov

- preklop med nitmi
  - Le delček navezovanja na nit se zamenja, večina procesa pa ostane – zato je hitrejši
- **Deljenje virov medsebojna komunikacija**
  - Preklop med procesi potrebuje preklop na večjederni način izvajanja, pri nitih tega ni potrebno

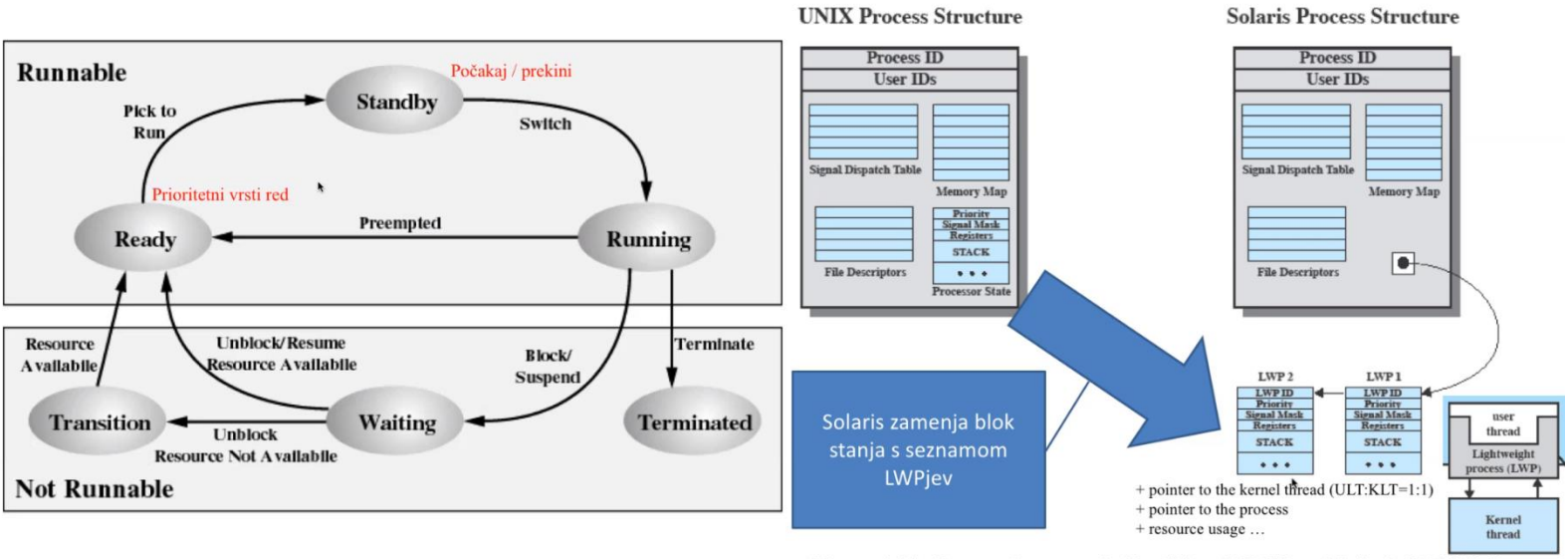


Figure 4.16 Process Structure in Traditional UNIX and Solaris [LEWI96]

Asd

Kakšna je učinkovitost multinitne aplikacije v večjedernem sistemu?

- Amdahlov zakon
- Lahko ga zračunamo z enačno. Nemoremo vsega pohitriti

Sočasnost in smrtni objem 6. Teden

- **tekmovanje procesov za vire** – kjer je več niti ali procesov bere ali piše isto podatkovno strukturo in končni rezultat je odvisen od časa zaključitve procesov
- **vzajemno izključevanje** – kadar je en proces v kritični sekciji izvajanja, ki dostopa do deljenih virov. In v kritični sekciji je lahko samo 1 proces
- **Kritična območje** - sekcija kode znotraj procesa, ki potrebuje dostop do deljenega naslovnega prostora, ki ne sme biti dostopana, med tem ko nek drug proces dostopa do tega dela kode.

    - Če je nek proces v kritičnem območju se morajo vsi drugi procesi tukaj ustaviti in počakat, da ta proces pride iz kritičnega območja
- **Nepreklinljivost/avtomičnost** – vedno, ko pride do izvedbe, se v ukaznem ciklu ne more prekiniti – vedno se izvrši do konca
- za sodelovanje procesov ob delitvi virov?

  - **cel sklop** programskih stavkov se mora izvršiti (podajo kritično območje)

- **Sodelovanje procesov preko komunikacije** – cel sklop programskiv stavkov se mora izvršiti, kritično območje nad katerim je atomično izvajanje

Kaj bi dali v kritično območje? Tisto kar more biti nepreklinljivo, da zagotavjlaš določene pogoje

Kako zagotovino vzajemno izključevanje?

- Algoritmi v naših programih (Petersonov)
- Algoritmi v programskih jezikih
- Omogočanje prekinitve
- Neprekinljivi strojni ukazi
- Katere 2 strojna ukaza poznamo:
    - Compare in swap. Ter exchange

Smrtni objem

Je situacije v kateri dva ali več procesa ne moreta nadaljvenati, ker čakata, da se eden izmed njiu konča.

Potencialni smrtni objem

Noben proces še ne preprečuje drugemu, da bi nadaljeval, vendar lahko vidimo da no te tega prišlo, če ne bomo ukrepali.

Katere lastnosti za zagotavlanje sočasnosti mora imeti sistem, da lahko pride do smrtnega objema

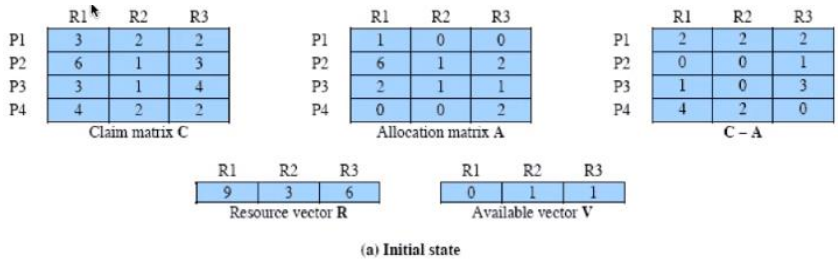
Prve tri so pogoji za potencialni smrtni objem. Vsi štirje skupaj so zahteve za dejanski smrtni objem.

1. **Vzajemno izključevanje**

2. **Drži in čakaj** (dobim vir in čakam naslednjega)
3. **Brez spostitve** (ko imam vir v lasti, ga ne bom sprostil dokler ga ne bom izvedel do konca)
4. **Ciklično čakanje**

Kako reševati problem smrtnega objema?

1. Kako bi ga **preprečili?:**
    a. ~~**Preprečevanja vzajemnega izključevanja?**~~ NE
    b. **Drži in čakaj**? DA. Obdrži tiste procese ki potrebuje iste vire, če jih dobi je ok če ne ga blokiramo.
    c. **Brez sprostitve?** Če proces ne dobi vseh potrebnih virov, ostale sprosti. Če potrebuje 3, dobi pa 2. 2 sprosti in ponovno zahteva vse 3.
    d. **Ciklično čakanje?** Definiramo linearno vrsto, kjer lahko vedno zahtevanjo vire, ki so vedno desno od že zaseženega vira. In tako dodeljujemo vire in preprečujemo smrtni objem.
2. Kako bi se mu **izognili?**
    a. Če gledamo cel proces kot enoto.Ali je zagon procesa problematičen:
        i. Maksimalno št. Zahtev trenutnih procesov + zahteve novega procesa < število vseh virov
    b. Ali je dodelitev vira problematična
        i. Bančniški algoritem (to je simulacija)
    c. Kako bi ga **zaznali?** Z algoritmom za zaznavanje (nadgrajen bančniški algoritem) se dejansko izvaja, če prideo do smrtnega se rollbacka



(a) Initial state

C: matrika zahtev – katere vire zahteva proces

A: kateri viri so uporabljeni v sistemu

C-A: prvi kvadrat 3 – 1 = 2. Iz prve matrike odštejemo drugo

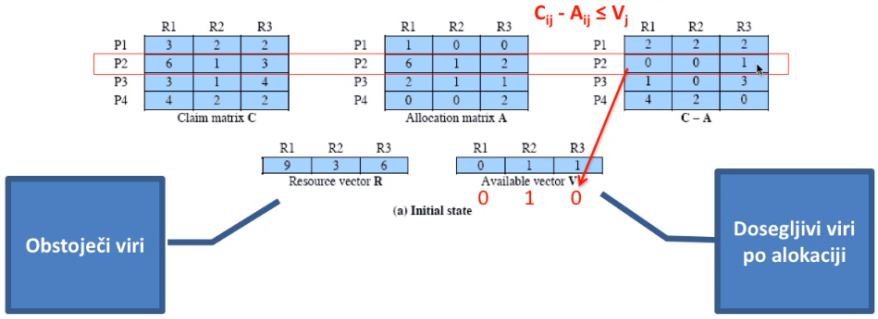
R- obstoječi viri v sistemu: seštejemo vrstice: R1-> 1+6+2+0 = 9,..

V: Pogledamo vire(R) seštejemo vrstice pri A in odštejemo

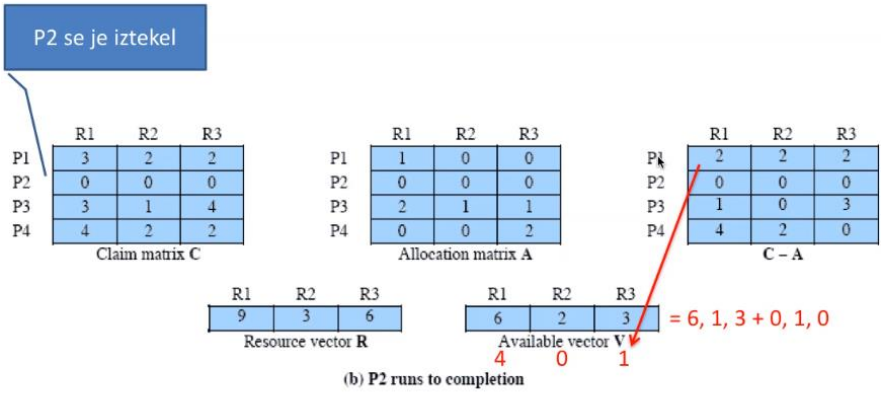
R1: 1+6+2+0 = 9 -> 9(R) – 9(A) = 0

R2: 0+1+1+0 = 2 -> 3(R) – 2(A) = 1

Se lahko katerikoli od procesov izvede do konca z dosegljivimi viri? -> $C_{ij} - A_{ij} \leq V_j$

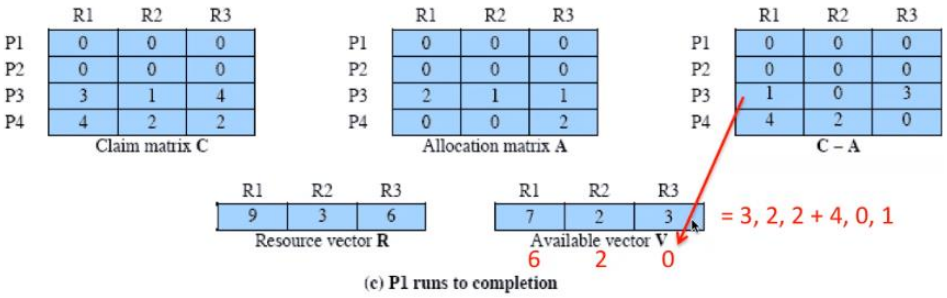


Če ti dam vir, ali je to stanje VARNO?

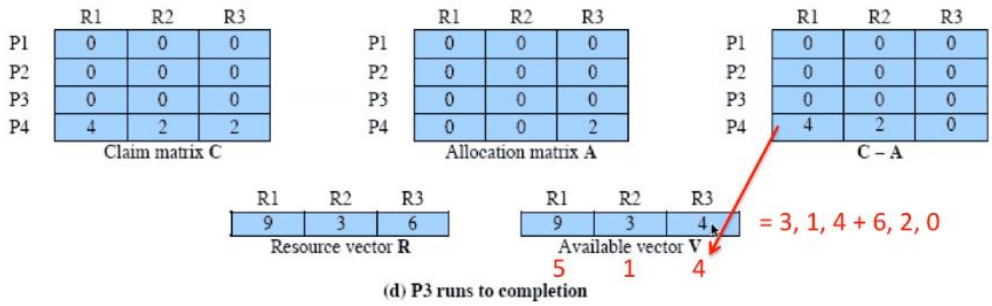


Se lahko sedaj izteče še kateri proces?

Izbran je bil P1 …



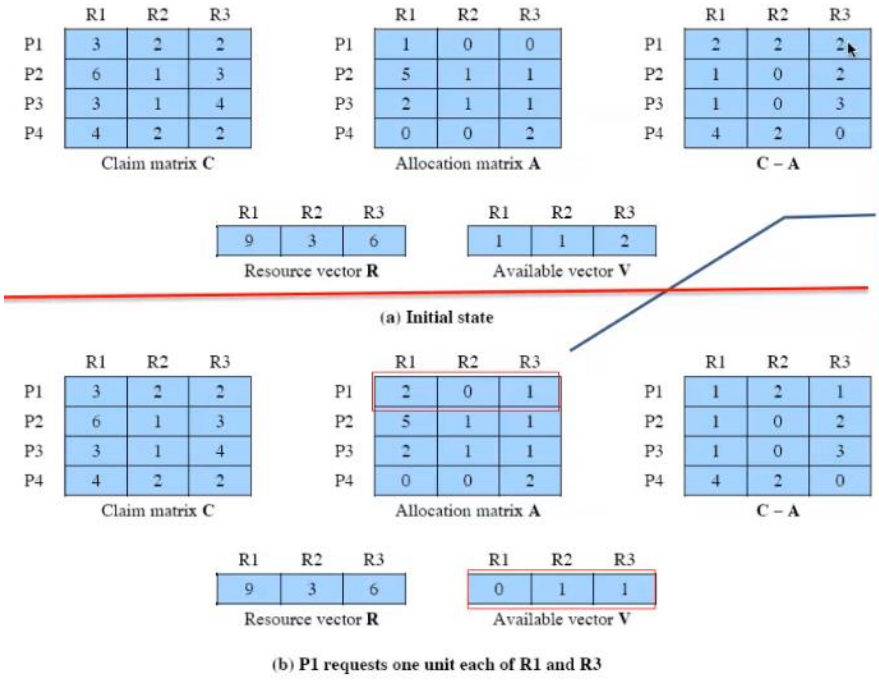
Izberemo P3 …



(d) P3 runs to completion

Vse procese smo izvršili do konca, zato je varno ker ne pride do smrtnega objema.

Malo spremenimo:



Tokrat naj P1 zahteva (in v **simulaciji** dobi) dva **dodatna** vira: R1 in R3.

Je to varno stanje?

(b) P1 requests one unit each of R1 and R3

Avaible vector je pri R1 sedaj 0. Kar pomeni da pride v smrtni objem. Zato išče novo rešitev, saj te ne bo izvedel.

Upravlanje s pomnilnikom

Glavna naloga je da sočasno teče več procesov, ki so pripravljeni na izvajanje

Kje mora biti proces da se lahko izvaja?

- V glavnem pomnilniku

V kakšnem stanju mora biti da se lahko izvaja?

- ready

Kakšna je delitev naslovnega prostora?

1. V primeru uniprogramiranja?
    a. Razdeljen na dva dela- za monitor in uporabniški program
2. V primeru multiprogramiranja?
    a.

Glavni pojmi

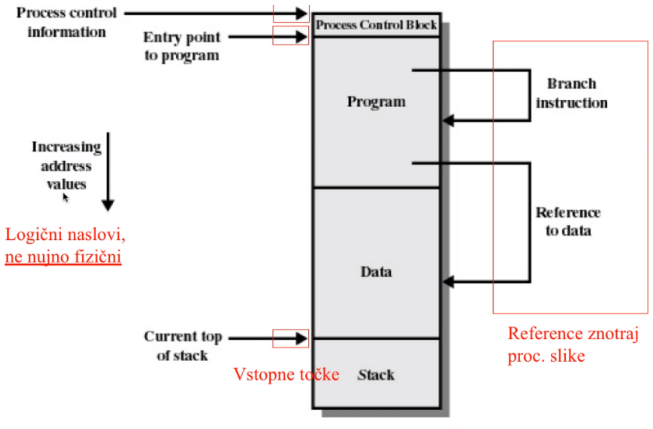
Okvir - blok **glavnega** spomina fiksne dolžine

Stran - blok sekundarnega spomina (disk) fiksne dolžine

Segment - blok sekundarnega spomina spremenljive dolžive

Katerim zadevam mora zagotoviti?

1. Sprememba lokacije procesa (relokacija) ->



Procesna slika

Figure 7.1 Addressing Requirements for a Process

2. Zaščita procesa
3. Delitev delov procesa
4. Logična predstavitev procesa
5. Fizična predstavitev procesa

Tehnike
(sistemskega) upravljanja s pomnilnikom torej pretoka

Razdeljevanje/ -> (nespremenljivo razdeljevanje)

Nespremenljivo razdeljevanje

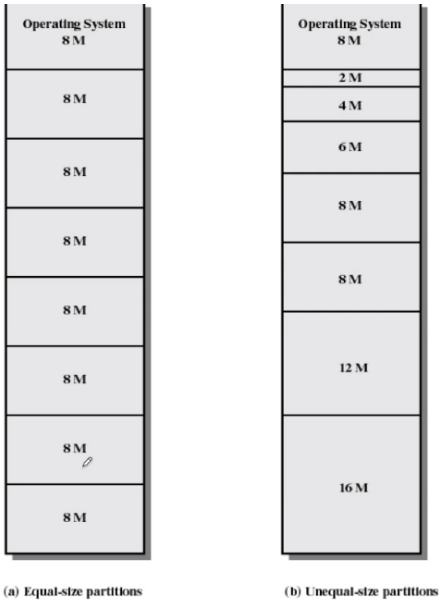
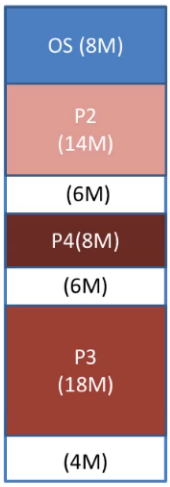


Figure 7.2 Example of Fixed Partitioning of a 64-Mbyte Memory

Dinamično razdeljevanje



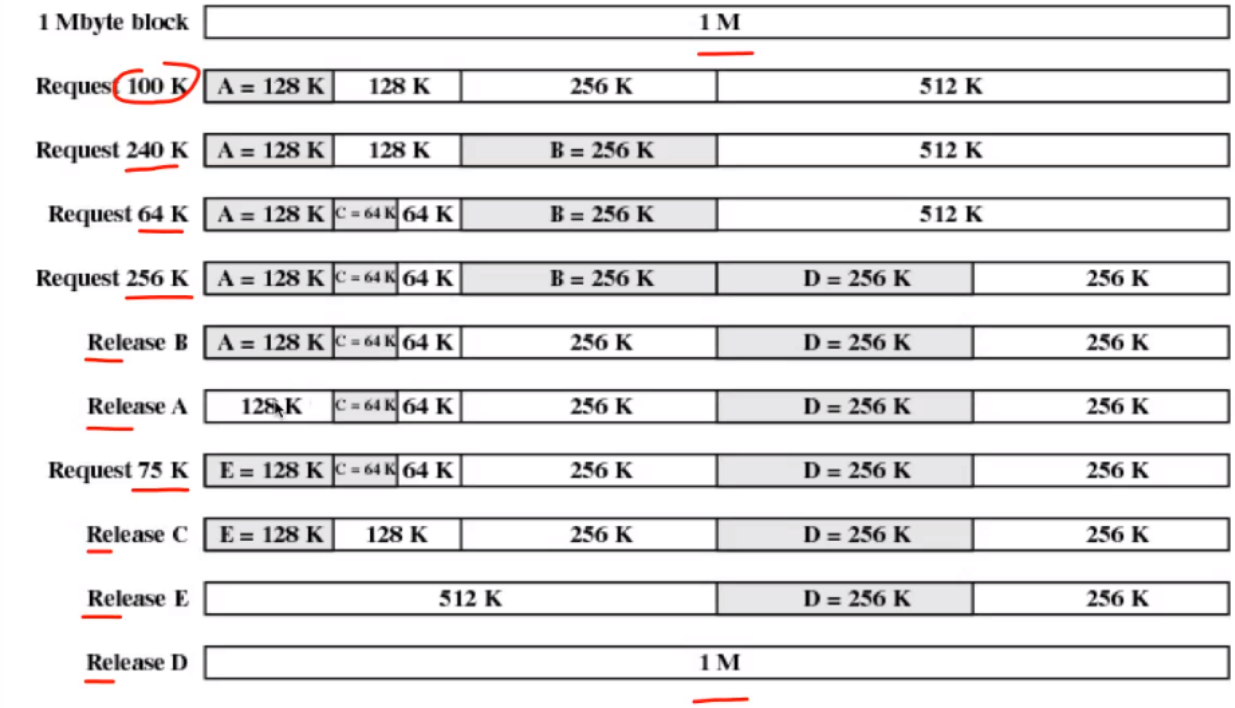
$6+6+4=16 \Rightarrow$ Prostor za še en proces

Dobra rešitev problema. Edin problem je da nastajajo ljukne ko gredo programi ven in notr. (**zunanja fragmentacija**) Ampak poznamo operacijo, ki potisne vse bloke skupaj. Rečemo ji zgoščenavnje. S tem se spremenijo fizični naslovi. Tem rečemo relokacija.

(Dinamično razdeljevanje)

Sistem prijateljev

Sistem prijateljev



$2^{U-1} < S \leq 2^U$

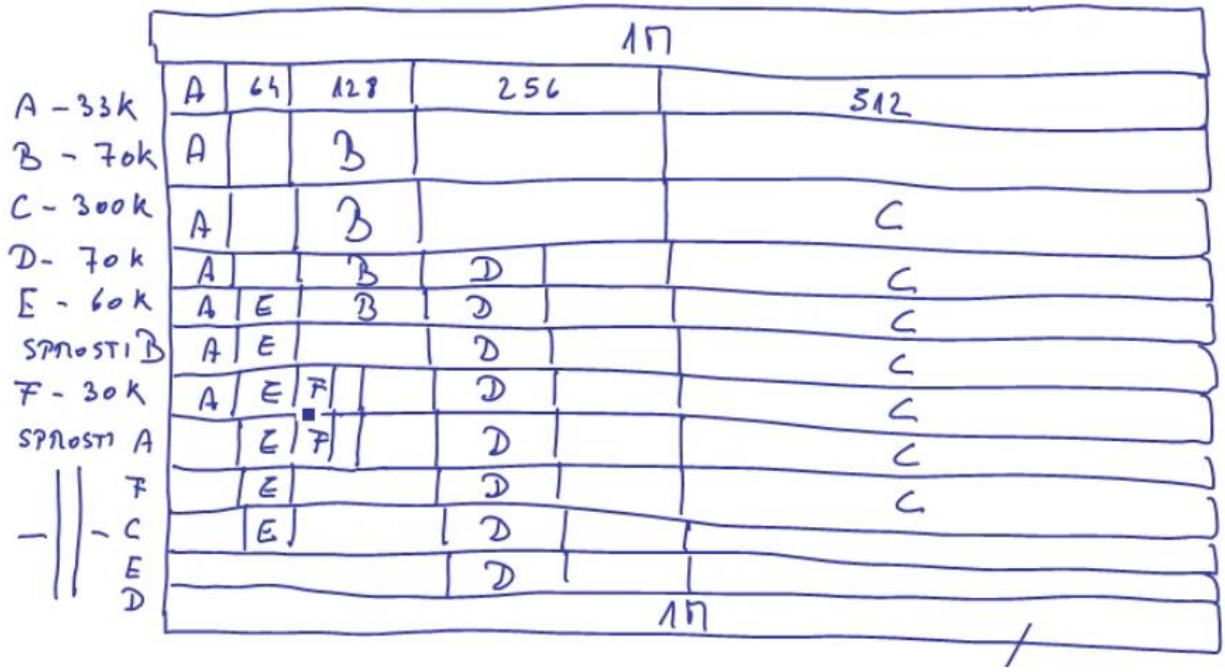
Figure 7.6 Example of Buddy System

Pogleda svojo velikost, ki se mora zapisti v pomnilnik. Npr velika je 100kb. Po formuli: ali je s med 512 in 1024? Ne. ali je med 256 in 512? Ne. ali je med 128 in 256 ne. ali je med 64 in 128? Da. Ter se zapiše v ta blok. Sepravi samo deli z 2 ter gleda če paše v prostor.

Naloga iz izpita:

Skicirajte rezervalni prostor po sistemu prijateljev.

$$2^{v-1} < size \leq 2^{v}$$



Relokacija

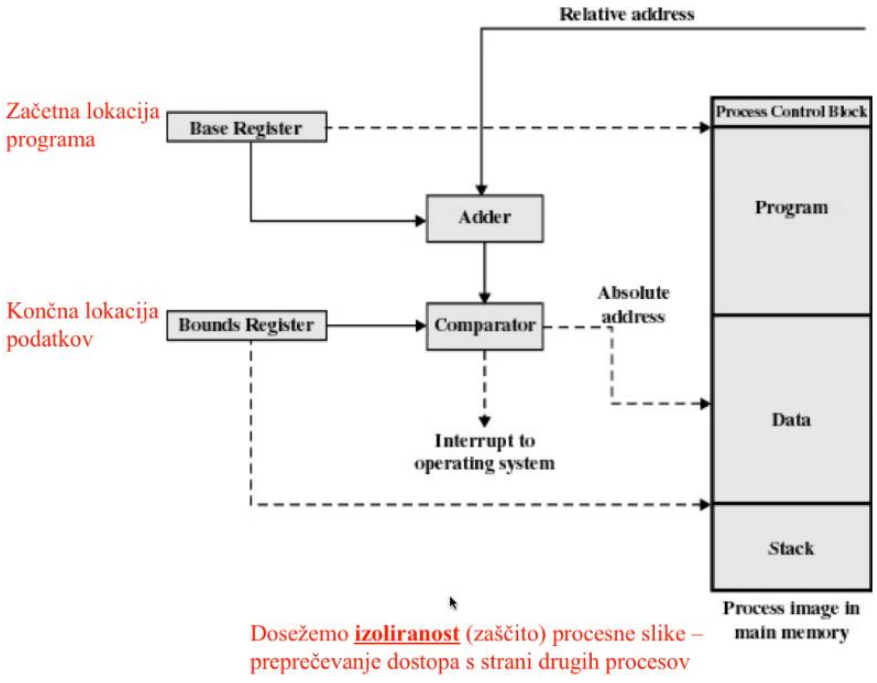


Figure 7.8 Hardware Support for Relocation

Tehnike (sistemskega) upravljana s pomnilnikom, torej pretoka

- Razdeljevanje/particioniranje
- (enostavno) ostranjevanje
- (Enostavna) segmentacija
- (Modreni) proces je razdeljen na dele in ni nujno cel v pomnilniku

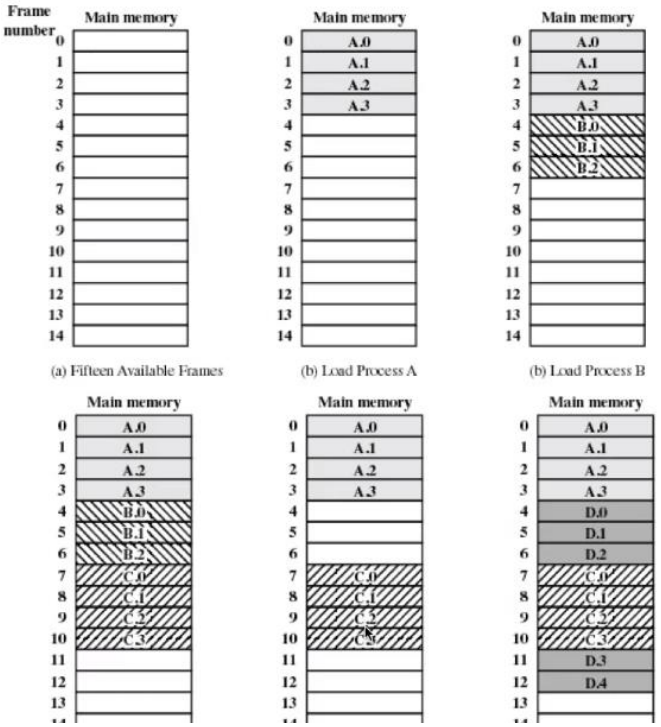


Figure 1 Enostavno ostranjevanje. Zapiše ukaz, ko ga ne rabi več ga zbriše. Pride nutr nov

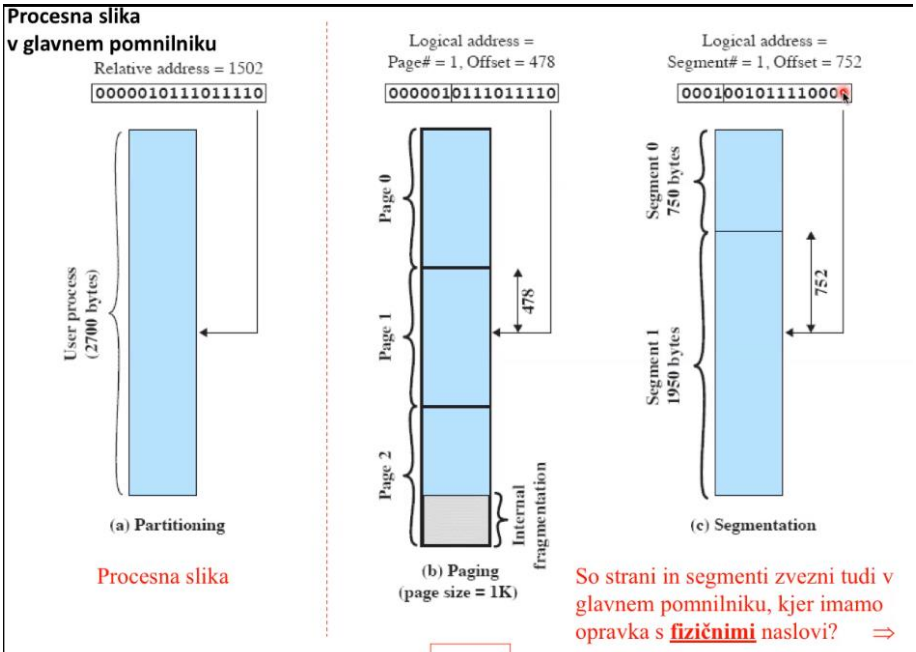


Figure 7.11 Logical Addresses

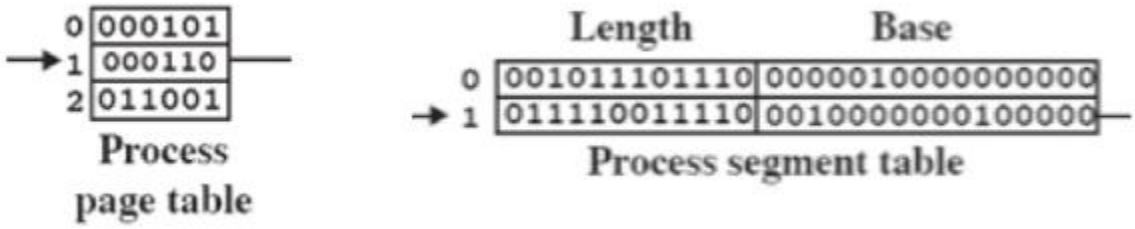
Napake pomnilnika

- Napaka strani
- Smetenje
- Princip lokalnosti

Navidezni pomnilnik

Kje mora biti podprt koncept navideznega pomnlinika?

Spomnimo se tabelo strani in tabelo segmentov:



V strojni opremi

V os.

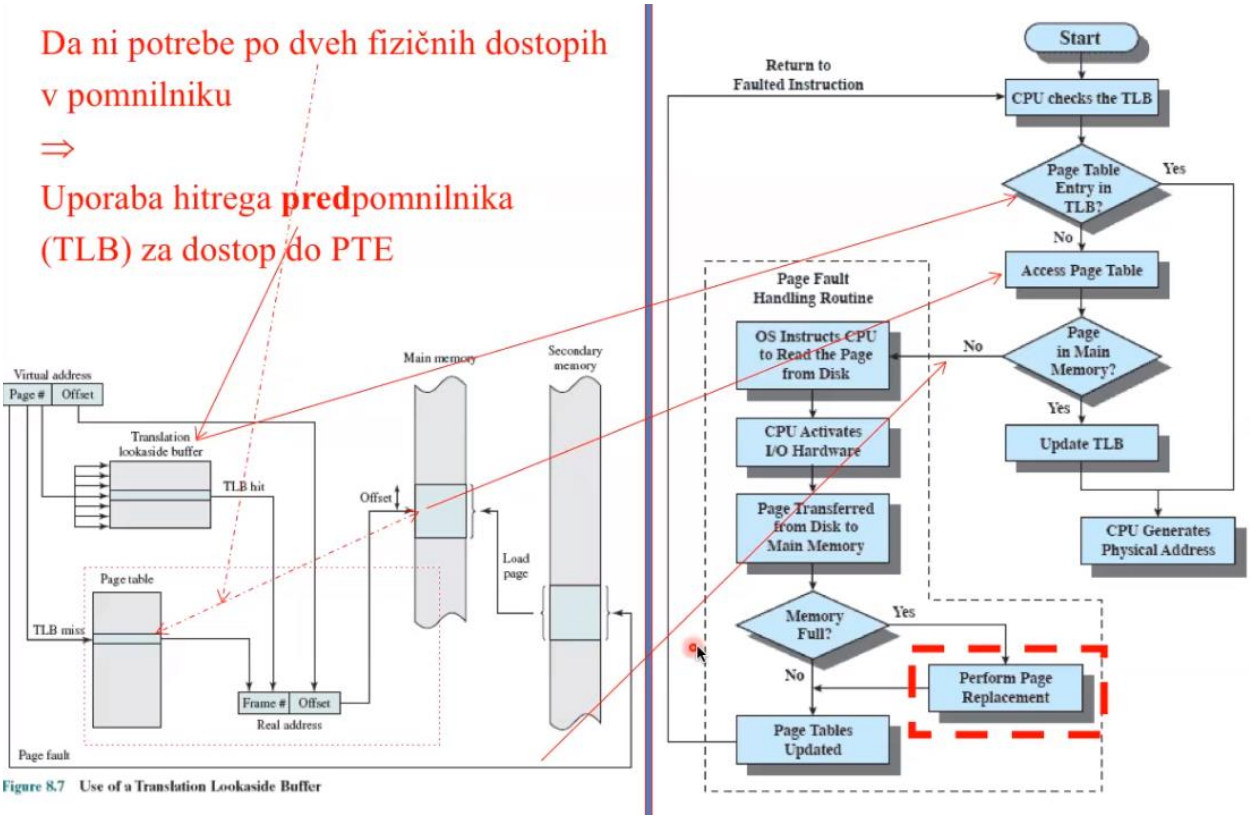


Figure 8.7 Use of a Translation Lookaside Buffer

Kaj pa je s podporo v operacijskem sistemu?

- **Kdaj** naj se stran prenese v glavi pomnilnik?
- **Kdaj** naj se spremenjena stran prenese v pomožni pomnilnik?

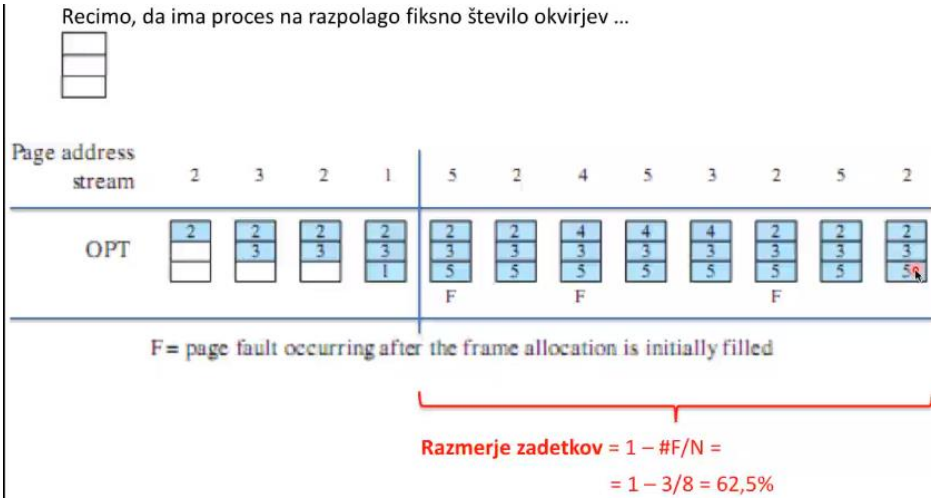
- **Katero** stran zamenjati , ko je pom. že poln?
    - Tisto, ki ima najmanjšo verjenost, da bo uporabljena v prihodnosti. (zamenjevalni algoritmi*

Zamenjalni algoritmi

Faza zajemanje je dokler se vse celnice ne napolnijo, sepravi do črte.

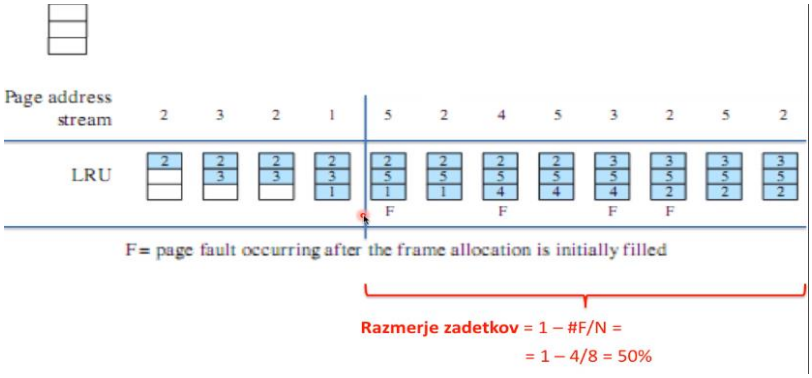
Optimalni (OPT) ->

- Recimo, da ima proces na razpolago fiksno št. Okvirjev..
- Imamo 3 bloke procesa. In tok strani (page address) to so zahteve in jih filamo v celice... Ko je poln potegnemo navpično črto, zato ker je poln in moramo iskati zamenjavo (F napaka strani). En bo šou vn in drug notr.
- **Razmerje zadetkov** je zgornja meja performance našega sistema. Vse F-je delimo z številom zamenjav.



Najdlje neuporabljen (LRU)

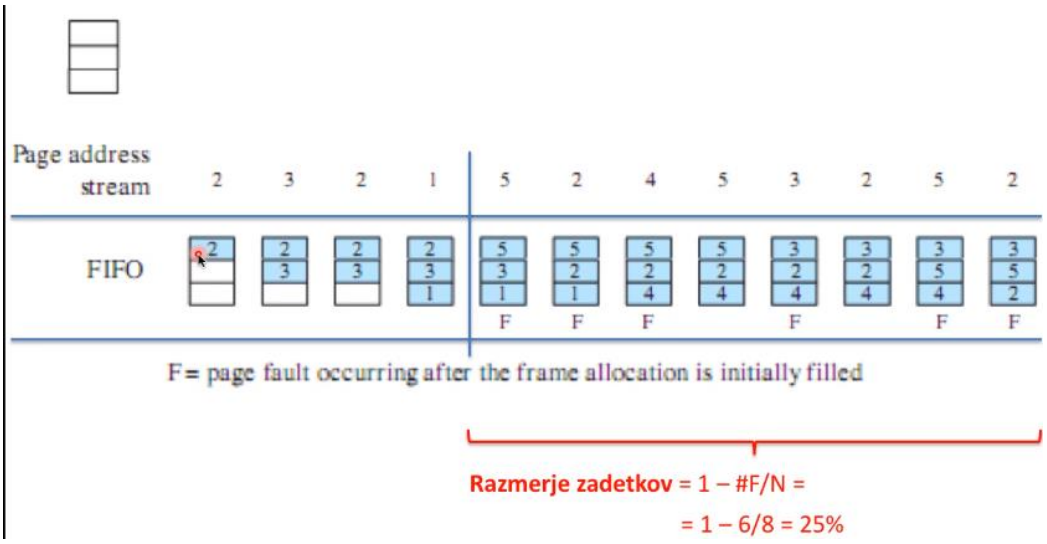
- Primer za črto: Ker je bila 3 najdalje neuporabljena, ga bomo zamenjali s 5.
- 2-ka je že v celici in se nič ne spremeni.
- Najdlje neuporabljena je sedaj 1-ka ter jo vržemo vn in zamenjamo s 4-ko



FIFO (First in First out)

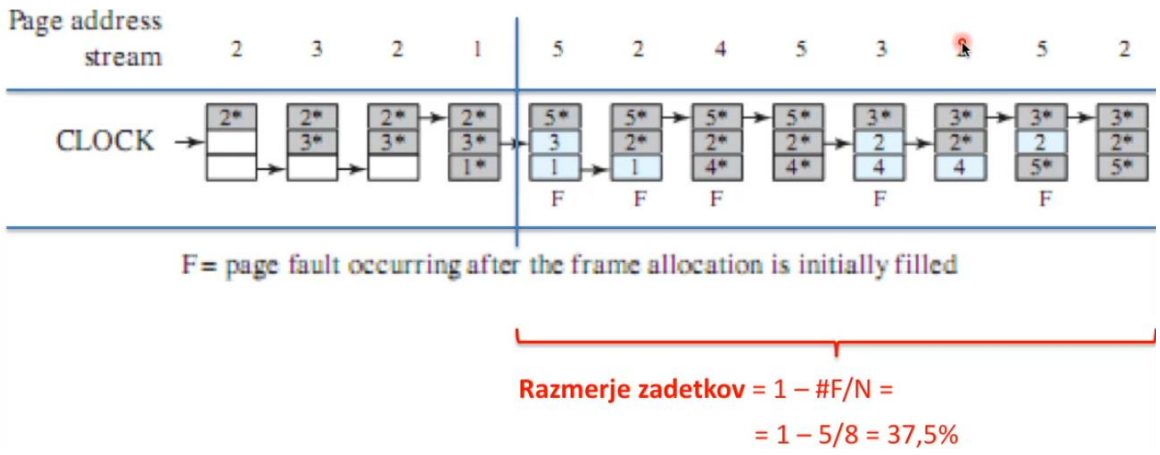
Pišem menjave od črte dalje:

- Tista ko je najdlje v pomnilnem prostoru. 2ka pri črti. Zato zamenjamo s 5ko.
- Katera ima najdaljšo sled? 3ka in jo vržemo vn ter zamenjamo z 2ko
- 1ko z 4ko
- 5ka je notr
- 5ka je najdlje notr in jo zamenjamo s 3ko



Princip ure (CP)

- Ko uporabimo neko celico gre used bit na 1 (ko je zraven zvezdica), če ne je na 0
- Tam kjer kaže puščica se bo nafilalo
- Vedno kažemo na modre celice
- Ko so vse celice sive, kaže na tisto, ki je najdlje neuprorabljena
- Če je uporabimo ji oddamo used bit (*)



Vse skupi:

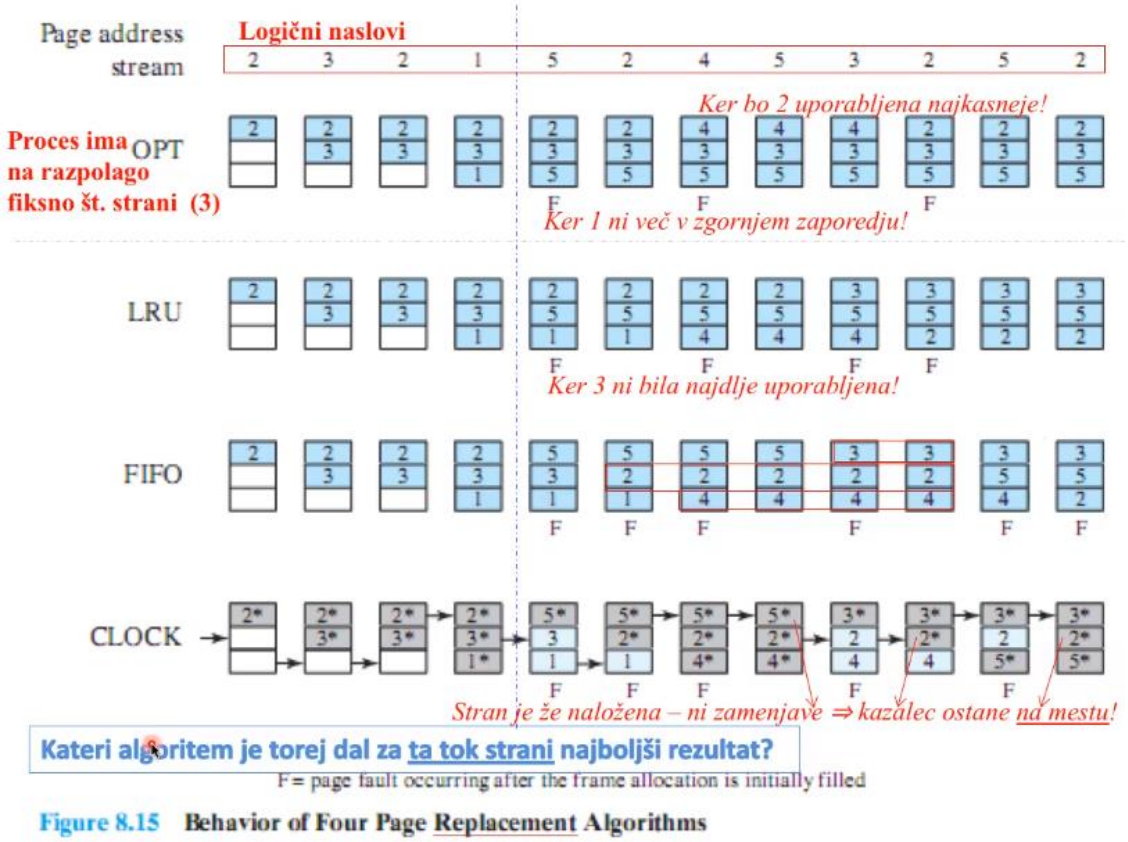


Figure 8.15 Behavior of Four Page Replacement Algorithms

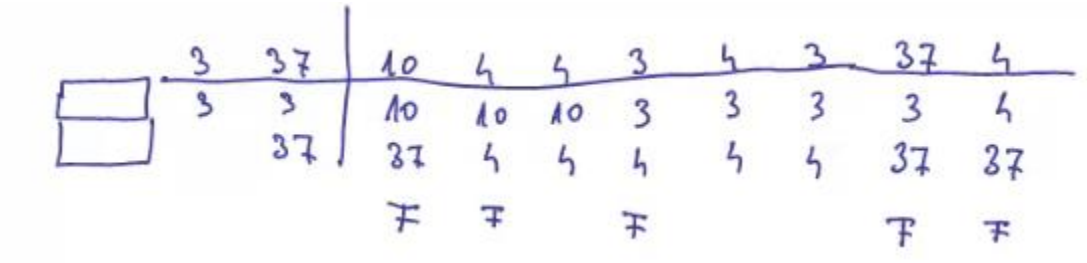
Najboljši rezulrat za tok strani je bil LRU.

Kako velika je bila v našem primeru rezidenčna množica strani procesa?

- Nespremevljiva velikost – 3
- Spremenljiva velikost
    - Sepremevljiva velikost, globalna zamenjava
    - Sepremevljiva velikost, lokalna zamenjava

1. Imamo dani pomnilnik z 2ma okvirjema.
Za oba primera izračunate razmerje zadetkov.

Fifo:

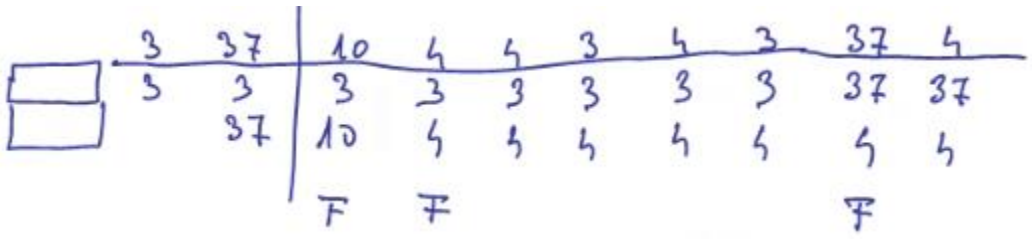


Razmerje zadetkov: $1 - \frac{5}{8} = \frac{3}{8} = 37{,}5\%$

OPT:

Pogledamo kiro bomo zahteval nasledno.

Za črto: 37ko zamenjamo s 10 ker bomo 3ko prej uporabili in zato 3ko pustimo4ko zamenjamo z 10 ker 10ke sploh ne bomo več rabili



Razmerje zadetkov: $1 - \frac{3}{8} = \frac{5}{8} = 62{,}5\%$

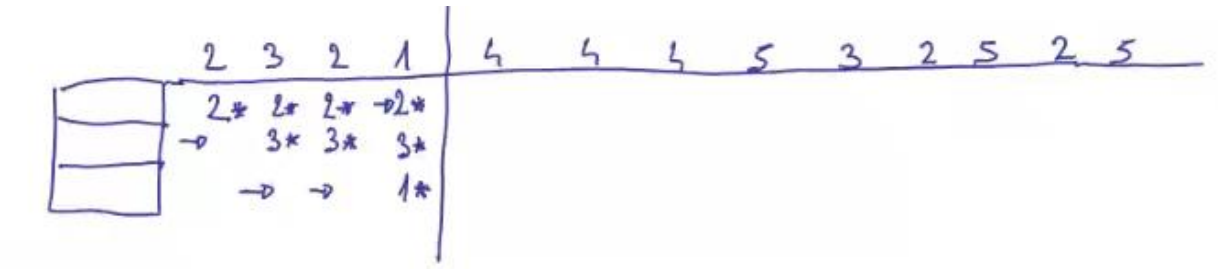
Primeri s kolokvija: polnenje pomnilnika (Fifo, LRU, CP)

2. Imamo glavni pomnilnik, ki ima 3 okvirje za proces. Pri izvjanju programa imamo naslednje zahtve:

Skicirajte kako bo izgledala zamenjava strani naslednjima algoritmoma. Kateri je najbolj učinkovit?

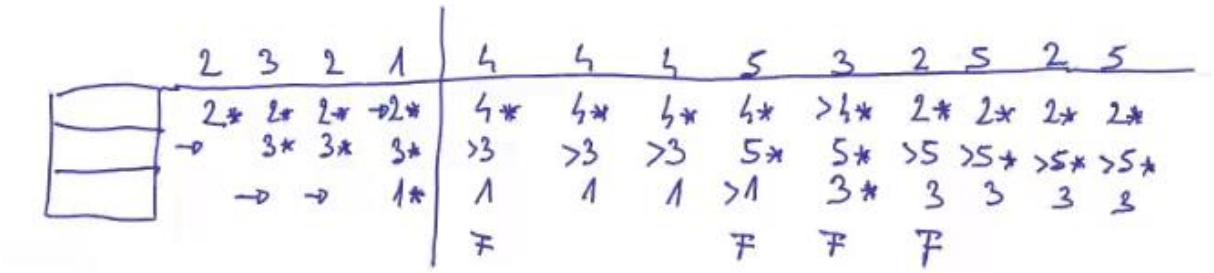
Princiom ure CP:

Fetching stage:



Prvo celico, ko najdem ki ime used bit na 0, ga vržem vn in ga dam na 1.

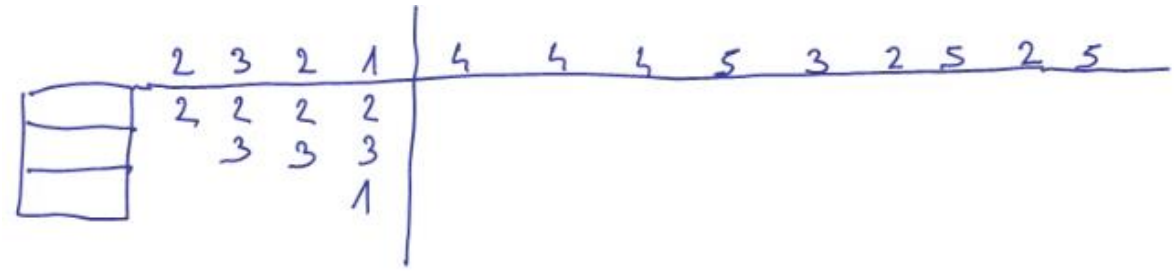
Za črto grem čez vse celice in jim dam used bit na 0 (odstranim zvezdico)
2ko zamenjam s 4ko, ji dodam used bit in dam kazalec na naslednjega, ki ga nima (3ka)
2 stolpec samo prepišem, ker je 4ka že notr in used bit že ima na 1
3 stolpec tudi prepišem
4 stolpec: 3ko vržem vn in dam notr 5ko
5 stoplec: 1ko zamenjam s 3ko
6 stoplec: vsem poberem bite. 4ko zamenjam z 2ko ker kazalec kaže na njo
7 stoplec: 5ki dodamo used bit
8 stoplec: prepisemo
9 stoplec: tudi prepisemo



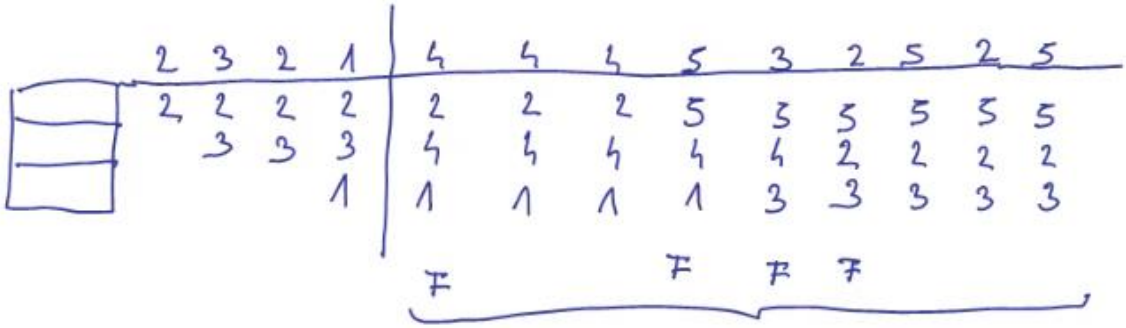
Razmerje zadetkov: 1- – $\frac{\#F}{N}$

$1 - \frac{4}{9} = \frac{5}{9} = 55{,}5\%$

LRU:



4 stoplec: najdlje ni bila v uporabi 2ka, zato jo vržem vn in napišem 5ko
5 stoplec: 1 gre ven, ker je najdlje nismo uporabljali, zato gre ven in notr 3ka
6 stoplec: 4ka vn 2ka not



Razmerje zadetkov: $1 - \frac{4}{9} = \frac{5}{9} = 55{,}5\%$

Odgovor: Oba algoritma sta enako učinkovita.

Enoprocesorsko razporejanje

Zakaj razporjejamo

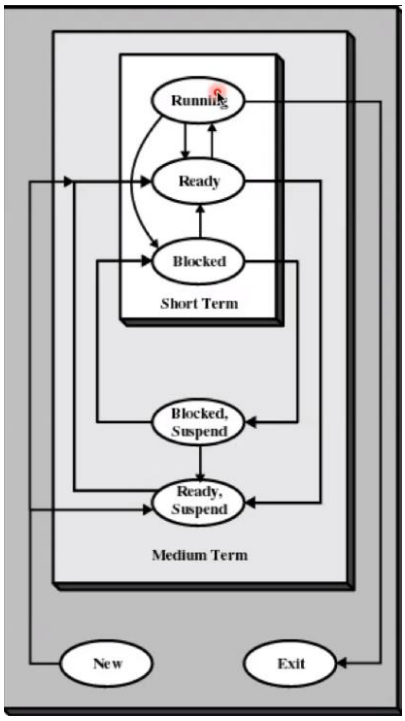
- Da minimiziramo odzivni čas
- Da maksimiziramo pretočnost
- Da maksimalno izrabimo procesor

Kakšne razporejevalnike poznamo?

- Dolgoročne (ustvarjanje procesov)
- Srednjeročne (odstranjevanje procesov iz glavnega pomnilnika)
- **Kratkoročne** (izvajanje samih procesov) iz stanja pripravljenih procesov izberemo enega in ga poženemo
- V/I (dodeljevdanje)

Kako izberemo pravilni proces za izvajanje?

Predstavitev z gnezdenjem funkciji razporejanja

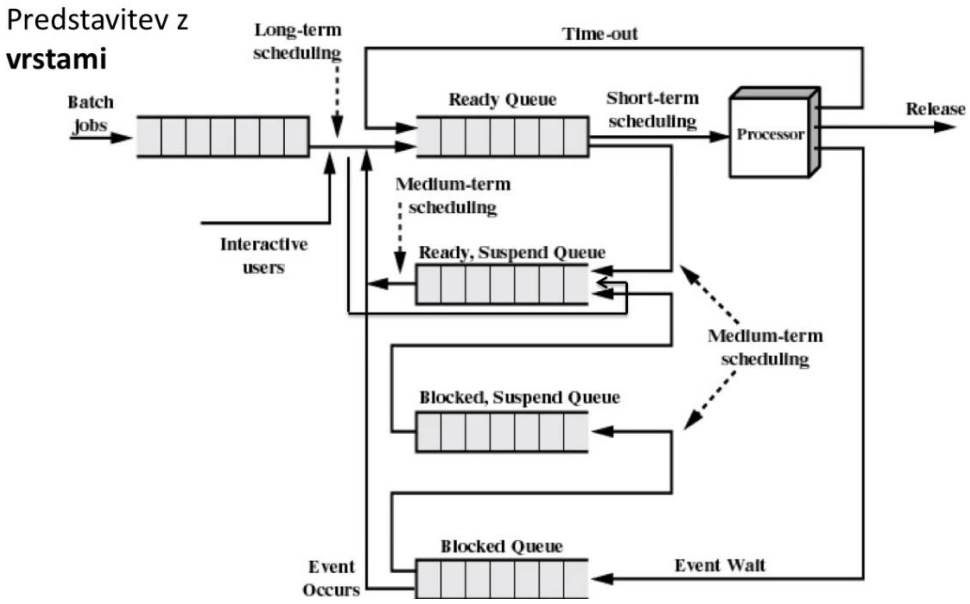


Predstavitev z **vrstami**

Long term – ustvarja procese

Srednjeročni – prenaša strani iz sekundarnega pormnilnika v glavni pomnilnik

Kratkoročni -upravljanje z vrstami. Kdo pride na vrsto



Kriteriji za oceno učinkovitosti postopka razporejanja

- Uporabniško usemrjeni
- Sistemsko usmerijeni

- Kvanititativni (rezultat je vendno neka številka in jo daje okolju za primerijavo)
- Kvalitativni (napovedljivost) gre za idejo zagotavljanja enakih kareteristik v sistemu

---

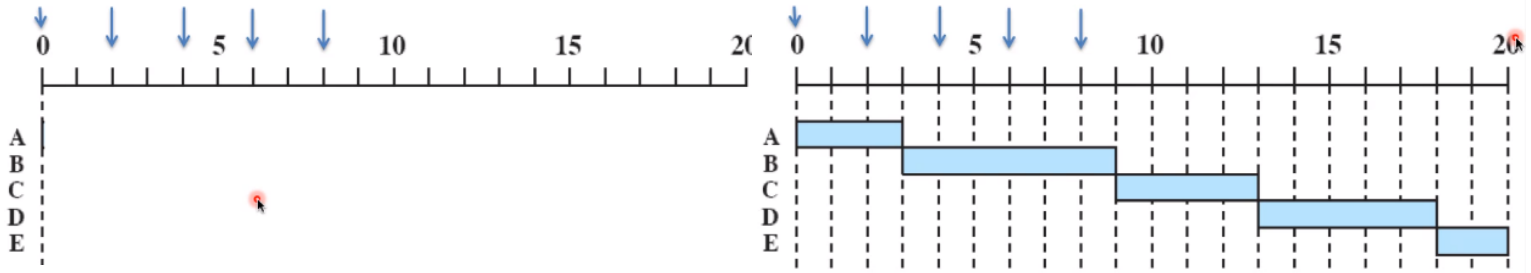
- **Preklopen** način izvajanja
- **nepreklopen** način izvajanja (ga ne moramo prekiniti, se vedno izvede do konca)

Algoritmi razporejanja

1. **prioritetno razporjanje**

2. **Najprej najstarejši** – (FIFO-FCFS) frist-come-frist-served (FCFS) (nepreklopen, ga

Proces	Čas dospetja	Čas izvajanja
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

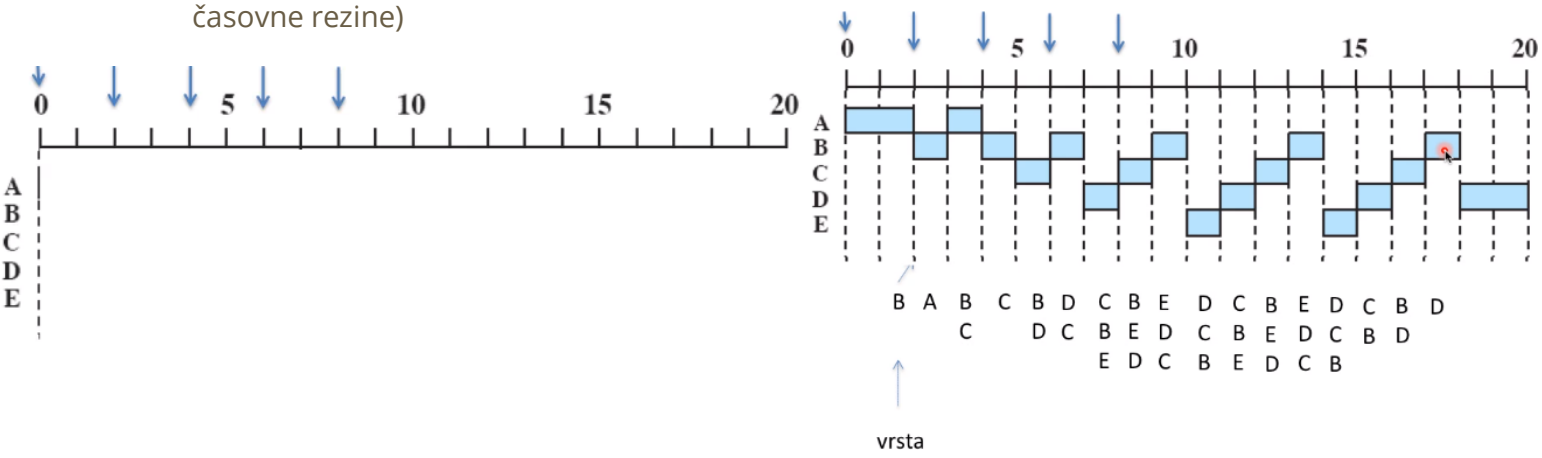


nemoremo ustaviti)

3. **Kriterji konstantne časovne rezine** (RR) Round-Robin

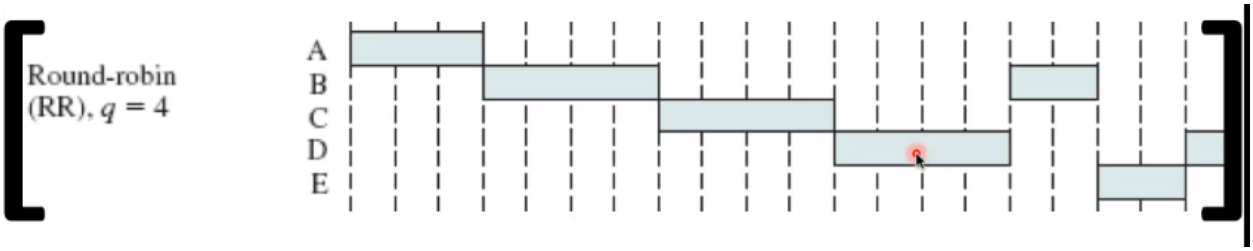
Proces	Čas dospetja	Čas izvajanja
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

q = 1 kvantum izvajanja (na kolk se delijo časovne rezine)

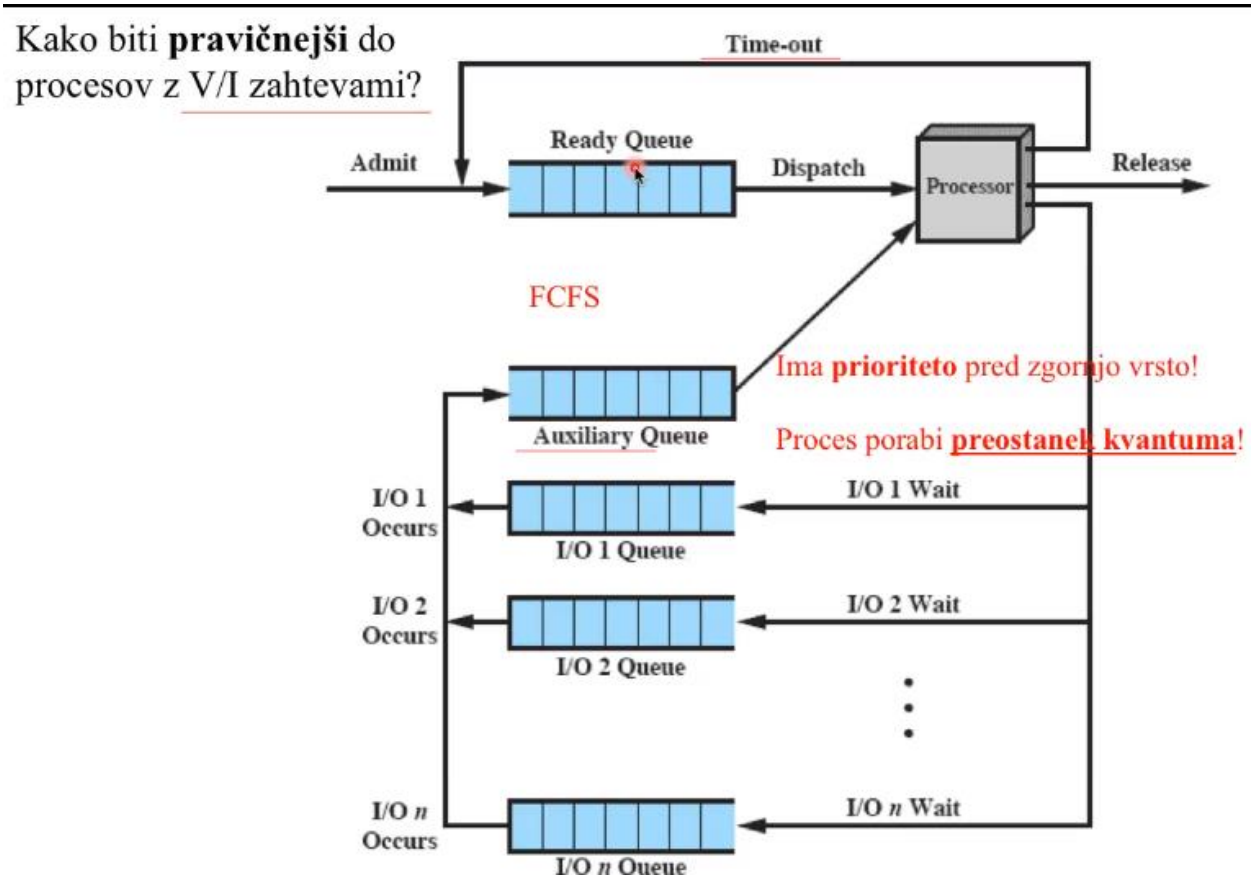


Tist ko je prvi v vrsti se začne izvajat. Novi procesi vedno pridejo na konec vrste in se za vsak interval q pomikajo višje. Če se proces še ni izvršiv do konca, se ta prestavi na konec vrste in je v obtoku dokler se ne izvede do konca. Procesi prideko na vrsto po dodeljenem času dospetja

Če bi imeli q nastavljen na 4:



Kako biti pravičnejši do procesov z V/I zahtevami?

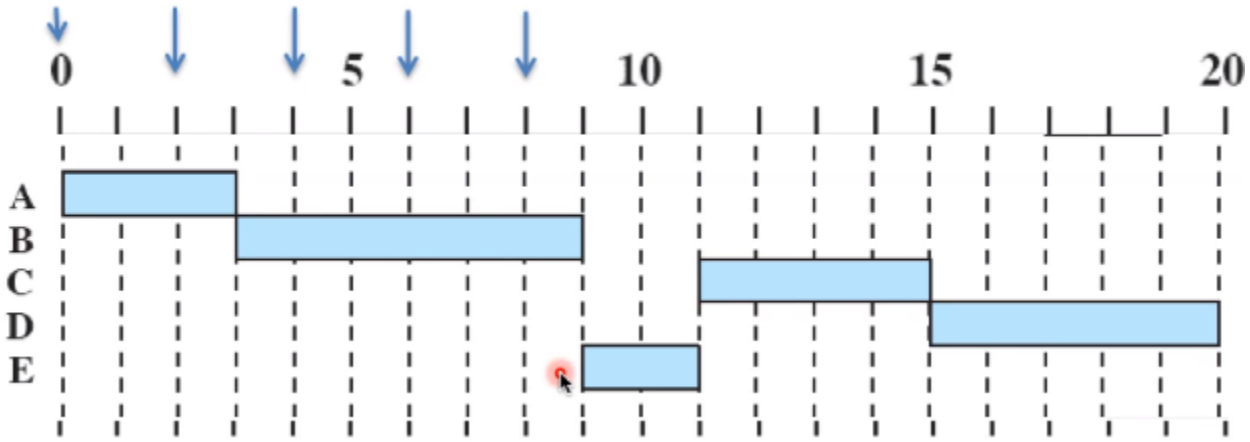


4. **Najprej najkrajši proces** Shortest process Next (SPN) nepreklopen

Proces	Čas dospetja	Čas izvajanja
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

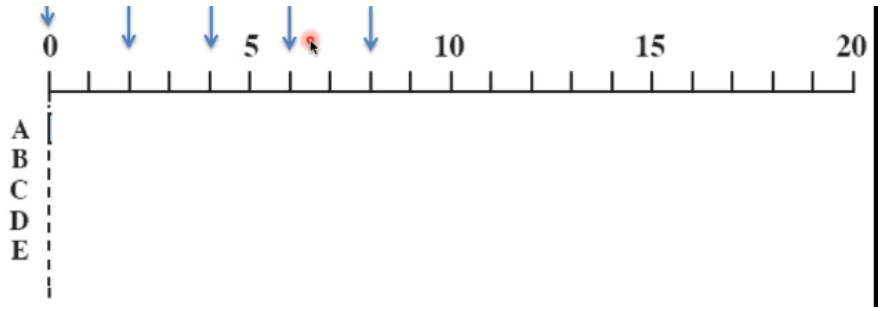
Če ni nobenga v vrsti, se izvede samo tisti ki ga imamo.V časovni rezini 9 imamo v vrsti C,D,E in pogledamo njihov čas izvajanja – E je najkrajši zato se izvede. Sedaj imamo v vrsti C,D. C je krajši in se izvede.

Če imamo v vrsti 2 procesa z enako časovno rezino, potem vzamemo najsnovnejši princip, seprafi FCFS.



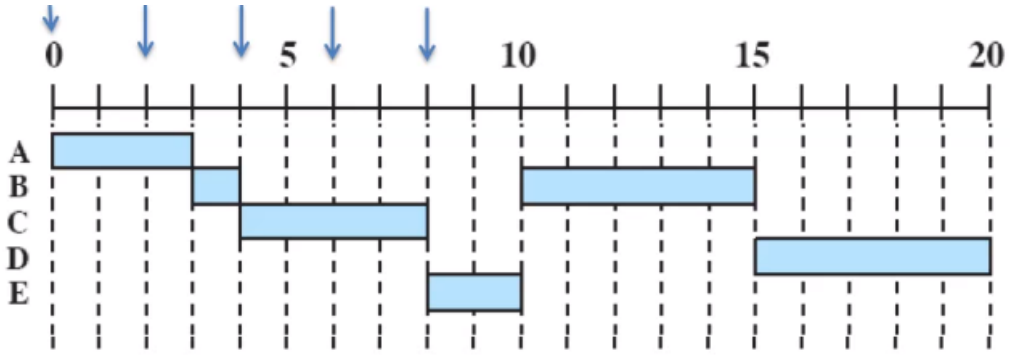
5. **Najprej tisti z najkrajšim preostankom časa** (SRT) – je preklopen

Proces	Čas dospetja	Čas izvajanja
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



A se prvo izvede do konca, ker ko B pride na vrsto se mora A izvajati še 1 časovno enoto B pa 6. Zato A ostane.
V čas. Enoti 4 pride v igro C. B ima še 5 časovnih enota za izvajanje C pa 4. Zato se zamenjata.
Vmes še prideta D, E. Vendar ima C vseeno najmanj časa za izvajanje.
Sedaj imamo v vrsti B, D, E. Ker je E najkrajši se izvede. Sedaj sta v vrsti B, D z enako dolžino in vzamemo proces FCFS zato se izvede B ter nato D.

6.



Najprej tisti z največjim odzivnim razmerjem (HRRN) (nepreklopen, ga nemoremo ustaviti)

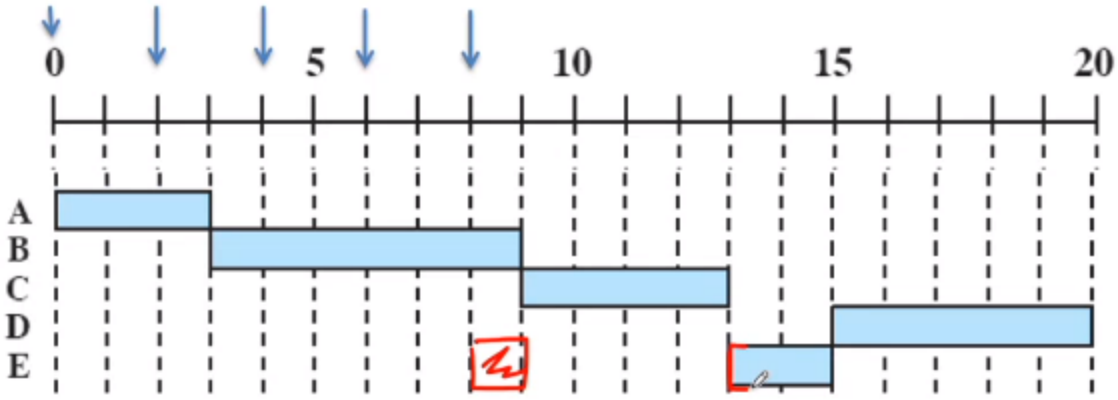
Proces	Čas dospetja	Čas izvajanja
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

MAKS($\frac{čas\ čakanja + pričakovan\ čas\ izvajanja}{pričakovan\ čas\ izvajanja}$)

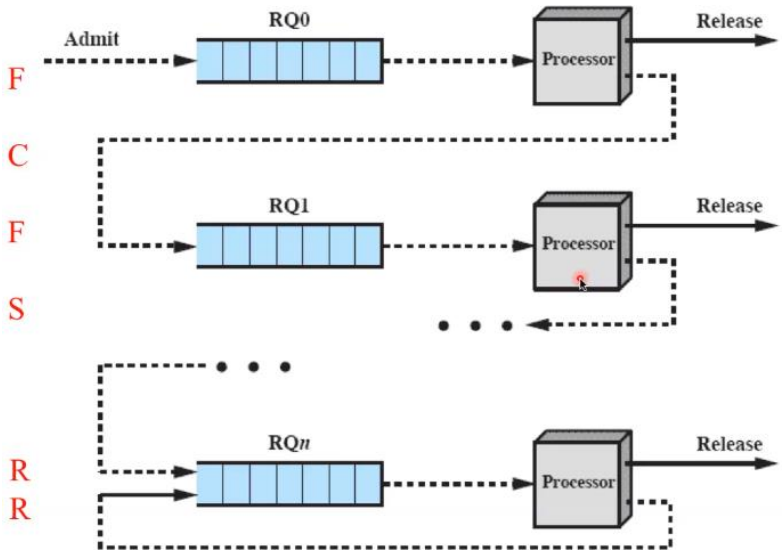
Pri 9 čas. Enoti za E je čas čakanja 1 + pričakovan čas izvajanja je 2, zato je $\frac{3}{2}$

C=$\frac{9}{4}$ D=$\frac{8}{5}$

Pri 15: D = $\frac{7+5}{5}$ E = $\frac{5+2}{2}$



7. **Razporejanje s provratnim odgovorom** (feedback) - preklopen



Časovno odvisen (ne statičen) diagram (procesor je le en)

Kako se borimo proti stradanju?

- Večanje št. Časovnih rezin q, recimo 2^i za RQ*i* izvajanja

Proces	Čas dospetja	Čas izvajanja
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

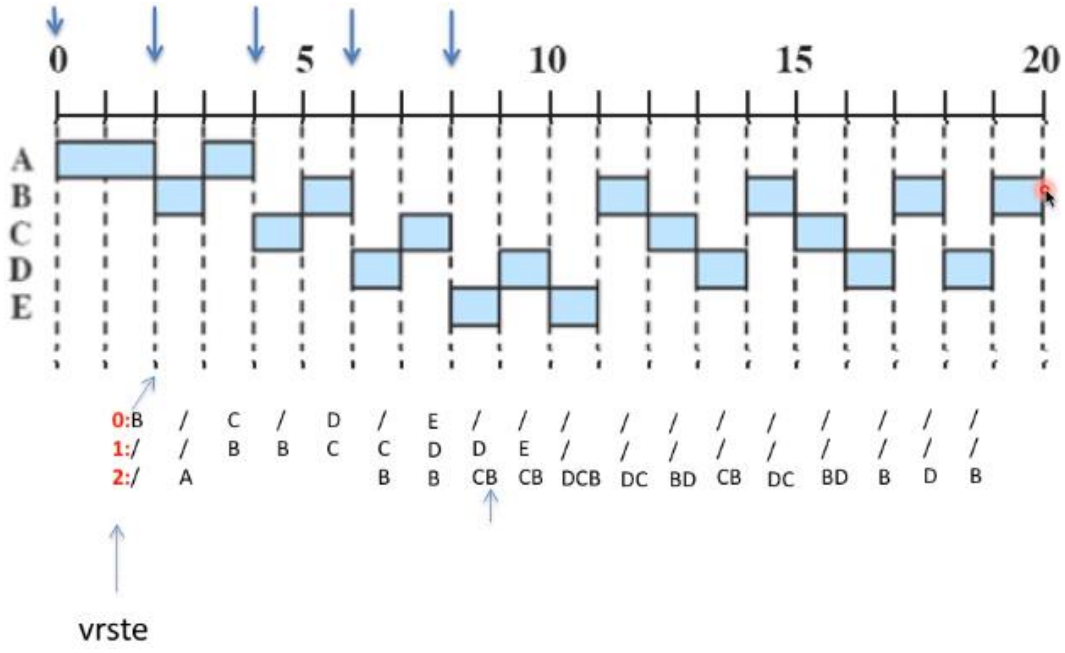
Q = 1

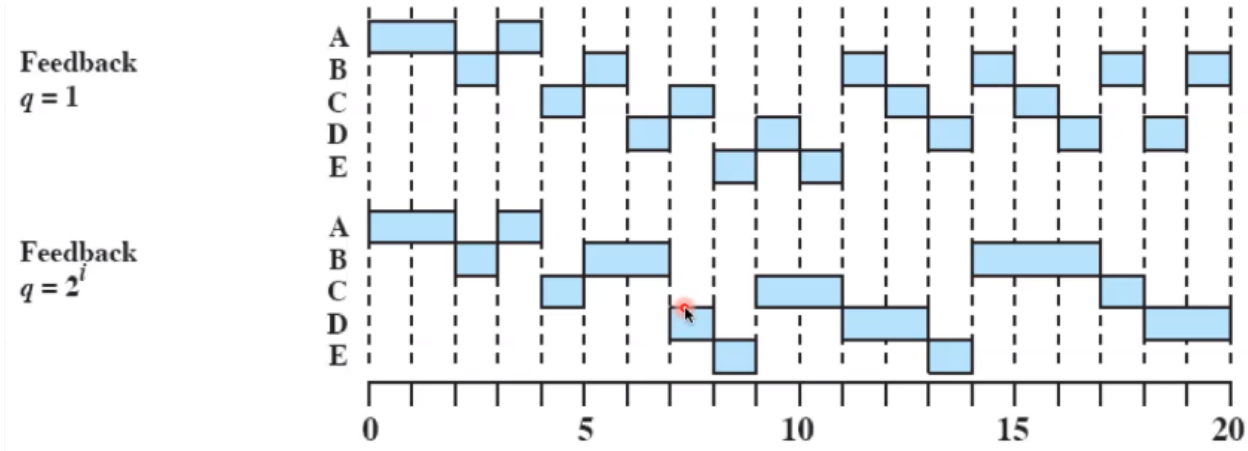
V vsaki časovni se bo označil proces, ki je na procesorju. Zapisali bomo vrste. Vzamemo najvišji proces v najvišji vrsti (tisti ki je prvi pršu). Če se ne izvede gre na konec vrste.

Ves čas se sprašujemo ali bomo zamenjali proces.
2. čas. Enota – B se še ni nikoli izvajal (0) zato gre B v izvajanje. A gre nazaj v vrsto. Izvedel se je 2x, zato ga damo v vrsto z (2). Ker je v vrsti samo A se izvede, čeprav je na dnu vrste.

V vrsto nastopi C, ker se še ni izvedel je vrsti z indeksom 0 ter se izvede pred B. V vrsti je sedaj samo B in se izvede.

Na 6 časovni enoti se pridruži D ter se izvede, saj je na najvišjem mestu.

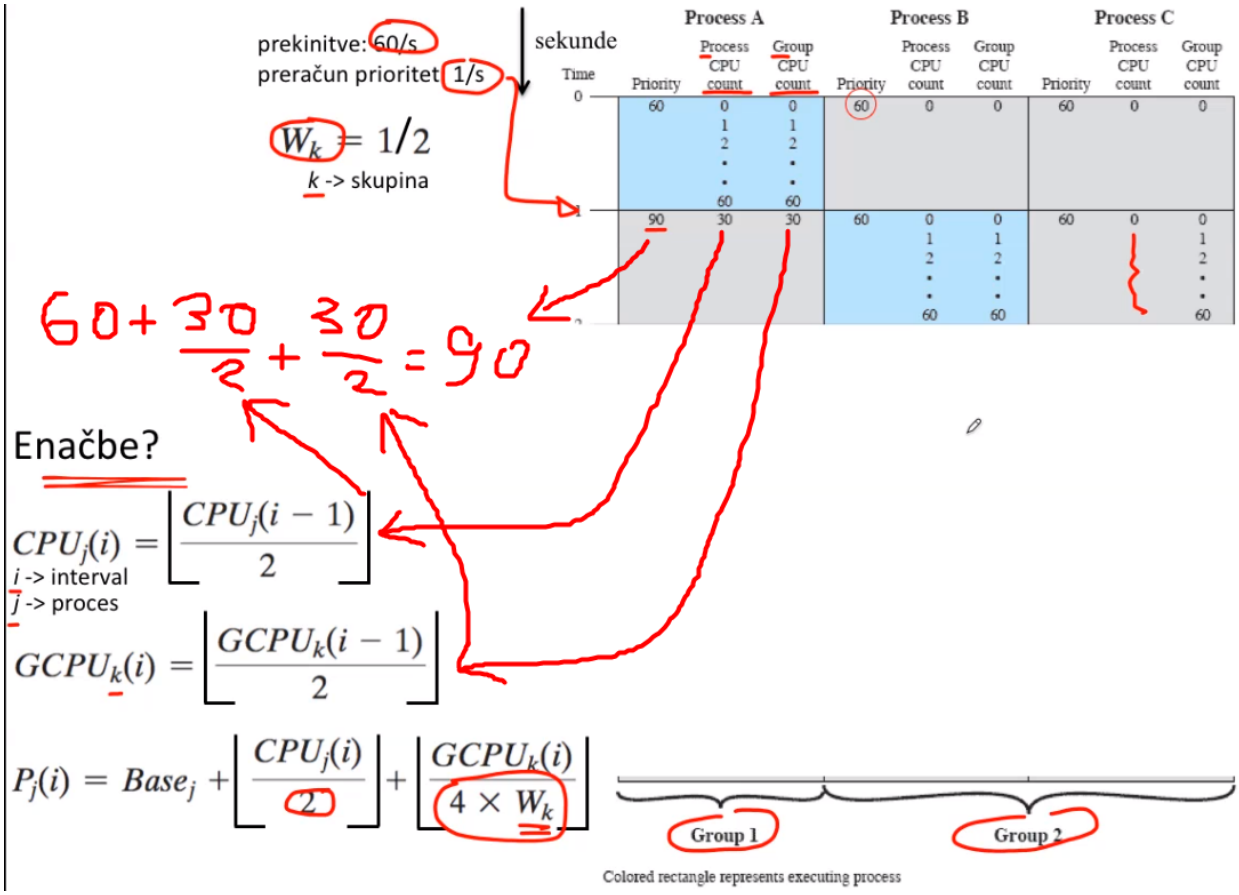




Q^i -> 2^i:

RQ0: ena časovna rezina (2^0) -> proces v vrsti 1 se bo izvajal 1 čas. enoto
RQ1: dve časovna rezina (2^1) -> proces v vrsti 2 se bo izvajal 2 čas. enoti
RQ2: štiri časovne rezine (2^2) -> proces v vrsti 3 se bo izvajal 4 čas. enote (glej primer zgoraj)

8. **Pravično razporejanje (FSS)**



Bazna prioriteta je podana na začatku naloge ter vedno izviramo iz nje. (60)

prekinitve: 60/s
preračun prioritet: 1/s

$W_k = 1/2$

k -> skupina

sekunde

Time	Process A Priority	Process A Process CPU count	Process A Group CPU count	Process B Priority	Process B Process CPU count	Process B Group CPU count	Process C Priority	Process C Process CPU count	Process C Group CPU count
0	60	0 … 60	0 … 60	60	0	0	60	0	0
1	90	30	30	60	0 … 60	0 … 60	60	0	0 … 60
2	74	15 … 75	15 … 75	90	30	30	75	0	30
3	96	37	37	74	15	15	67	0	15

Enačbe?

$$CPU_j(i) = \left\lfloor \frac{CPU_j(i-1)}{2} \right\rfloor$$

i -> interval
j -> proces

$$GCPU_k(i) = \left\lfloor \frac{GCPU_k(i-1)}{2} \right\rfloor$$

$$P_j(i) = Base_j + \left\lfloor \frac{CPU_j(i)}{2} \right\rfloor + \left\lfloor \frac{GCPU_k(i)}{4 \times W_k} \right\rfloor$$

$60 + \frac{15}{4 \cdot 0{,}5} = 74$

$W_k = \frac{1}{2}$

Group 1 Group 2

prekinitve: 60/s
preračun prioritet: 1/s

$W_k = 1/2$

k -> skupina

sekunde

Time	Process A Priority	Process A Process CPU count	Process A Group CPU count	Process B Priority	Process B Process CPU count	Process B Group CPU count	Process C Priority	Process C Process CPU count	Process C Group CPU count
0	60	0 … 60	0 … 60	60	0	0	60	0	0
1	90	30	30	60	0 … 60	0 … 60	60	0	0 … 60
2	74	15 … 75	15 … 75	90	30	30	75	0	30
3	96	37	37	74	15	15 … 75	67	0 … 60	15 … 75
4	78	18 … 78	18 … 78	81	7	37	93	30	37
5	98	39	39	70	3	18	76	15	18

Enačbe?

$$CPU_j(i) = \left\lfloor \frac{CPU_j(i-1)}{2} \right\rfloor$$

i -> interval
j -> proces

$$GCPU_k(i) = \left\lfloor \frac{GCPU_k(i-1)}{2} \right\rfloor$$

$$P_j(i) = Base_j + \left\lfloor \frac{CPU_j(i)}{2} \right\rfloor + \left\lfloor \frac{GCPU_k(i)}{4 \times W_k} \right\rfloor$$

$81 = 60 + 3 + 18$

Group 1 Group 2

Colored rectangle represents executing process

ocena

Primerjava učinkovitosti:
T_s – čas izvajanja
T_r – čas čakanja+izvajanja= **obračalni čas**
T_r/T_s – **normaliziran obračalni čas**
(r – *residence*, s – *service*)

Table 9.5 A Comparison of Scheduling Policies

Process	A	B	C	D	E	T_w
Arrival Time	0	2	4	6	8	
Service Time (T_s)	3	6	4	5	2	Mean
FCFS						
Finish Time	3	9	13	18	20	
Turnaround Time (T_r)	3	7	9	12	12	8.60
T_r/T_s	1.00	1.17	2.25	2.40	6.00	2.56
RR $q = 1$						
Finish Time	4	18	17	20	15	
Turnaround Time (T_r)	4	16	13	14	7	10.80
T_r/T_s	1.33	2.67	3.25	2.80	3.50	2.71
RR $q = 4$						
Finish Time	3	17	11	20	19	
Turnaround Time (T_r)	3	15	7	14	11	10.00
T_r/T_s	1.00	2.5	1.75	2.80	5.50	2.71
SPN						
Finish Time	3	9	15	20	11	
Turnaround Time (T_r)	3	7	11	14	3	7.60
T_r/T_s	1.00	1.17	2.75	2.80	1.50	1.84
SRT						
Finish Time	3	15	8	20	10	
Turnaround Time (T_r)	3	13	4	14	2	7.20
T_r/T_s	1.00	2.17	1.00	2.80	1.00	1.59
HRRN						
Finish Time	3	9	13	20	15	
Turnaround Time (T_r)	3	7	9	14	7	8.00
T_r/T_s	1.00	1.17	2.25	2.80	3.5	2.14
FB $q = 1$						
Finish Time	4	20	16	19	11	
Turnaround Time (T_r)	4	18	12	13	3	10.00
T_r/T_s	1.33	3.00	3.00	2.60	1.5	2.29
FB $q = 2^i$						
Finish Time	4	17	18	20	14	
Turnaround Time (T_r)	4	15	14	14	6	10.60
T_r/T_s	1.33	2.50	3.50	2.80	3.00	2.63

Obračalni čas = čas čakanja + čas izvajanja procesa

Normaliziran čas = čas čakanja + izvajanja / čas izvajanja

Večprocesorsko razporejanje & razporejanje v realnem času

Algoritmi razporejanja in pogostost sinhronizacije med procsei v sistemu

Granularnost?

Če je sinhronizacijski intervač večji kot 20 ukazov

Kaj pa če imamo opravka s pogosto odvisno vzporednostjo (<20 ukazov)?

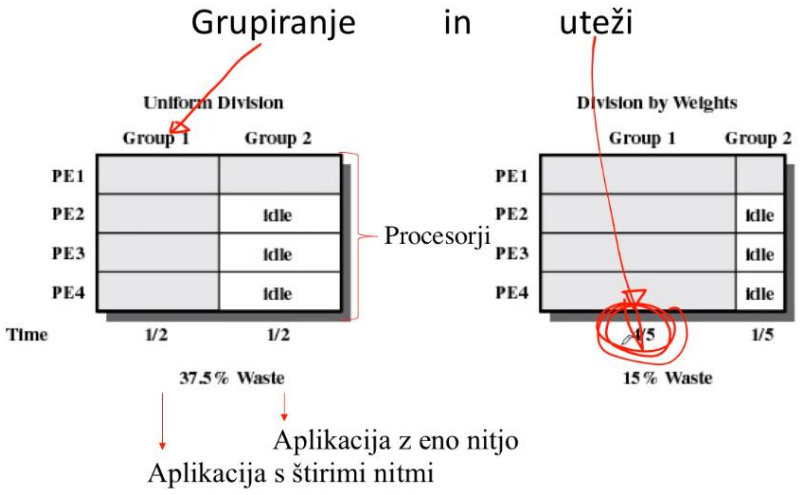
Razporejanje procesov procesorjem

- Permamentna dodelitev?
- Globlana vrsta?
- Raziskave so pokozala, da je izbira algoritma **manjn pomembna**

Razporejanje nitk na procesorje

- Ali dramatično izboljšajo učinkovitost?
  - Almadahov zakon.
- **Delitev bremena** (load sharing)
- Skupinsko razporejnaje

Grupiranje in uteži:



- Enkratko razporejanje (extremno skupinsko razporejanje)
- Dinamično razporejanje (so bol experimentalini sistemi)

Sistemi, kir razporejajo v relanem času

- Od česa je dovisna pravilnost delovanja OS
  - Logični rezultat procesiranja + čas v katerem je rezultat na voljo
- Primer:
  - Komunikacija Dsikord, Skype
- Trda/mehka realno-časovnaa zahteva
- Periodična/aperiodična relano-časovnna zahteva

Posebnosti realno-časovnih sistemov (OS)

- Determinističnost
- Odzivnost (kdaj ustreže zahtevi)
- Uporabniški nadzor
- Zanseljivost
  - Ob napaki jo sistem skuša odpraviti

Pristopi k razporejanju v realnem času

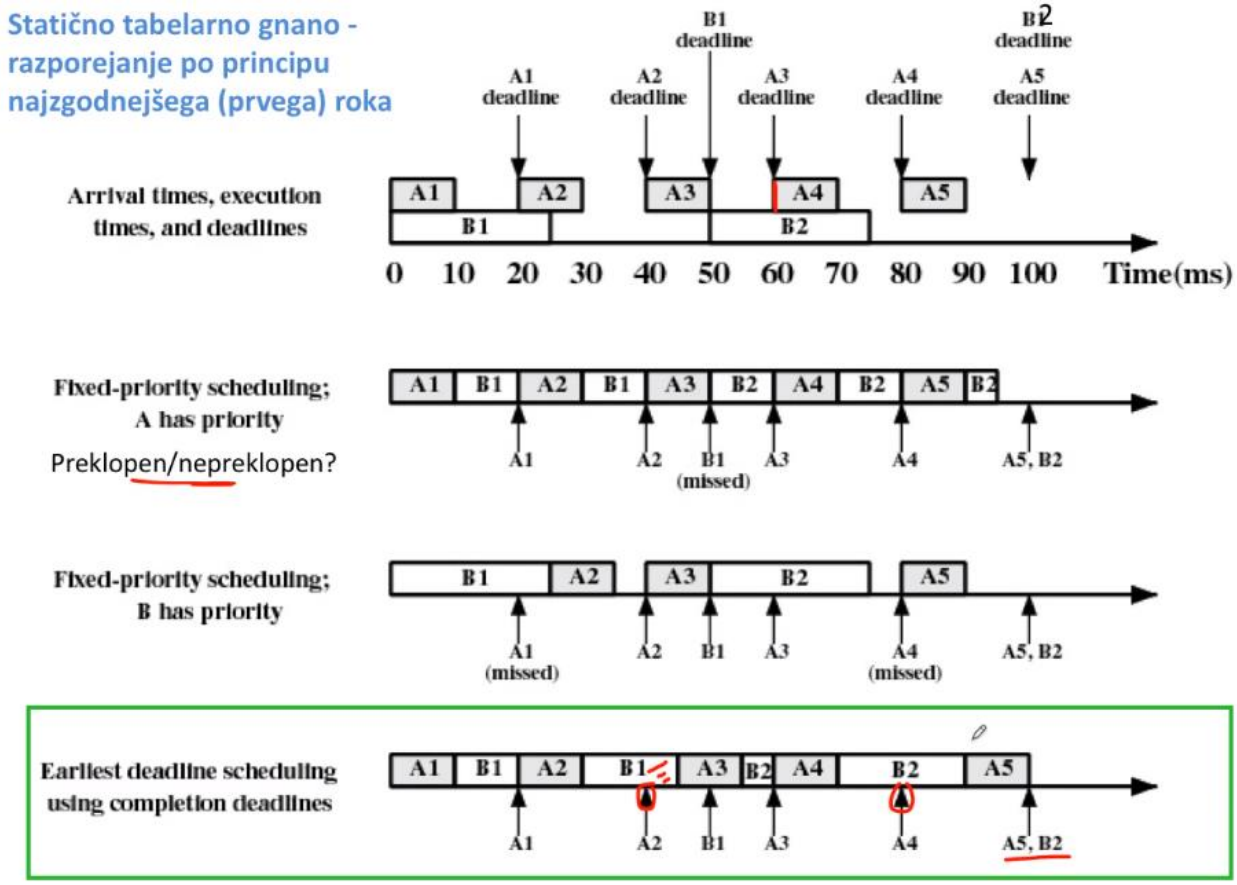
- **Statično tabelarno gnano (razporejanje po principu najzgodnejšega prvega roka)**

Odčitaj in interpretiraj podatke periodično
iz dveh senzorjev: A in B ⇒

Table 10.2 Execution Profile of Two Periodic Tasks

Process	Arrival Time	Execution Time	Ending Deadline
A(1)	0	10	20
A(2)	20	10	40
A(3)	40	10	60
A(4)	60	10	80
A(5)	80	10	100
. . .	. . .	. . .	. . .
B(1)	0	25	50
B(2)	50	25	100
. . .	. . .	. . .	. . .

(roki zaključka)



Razporejanje po principu najzgodnejšega (prvega) rok u neizsiljenim časom nedeljavnosti procesorja

Slika....

- **Statično prioritetno gnano prekopno**

1. Monotono razporejanje (RMS)

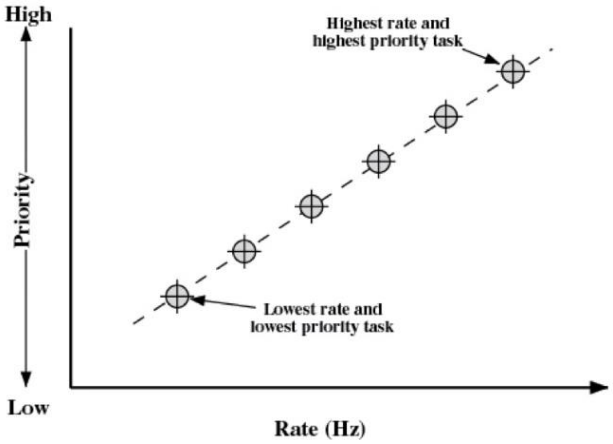
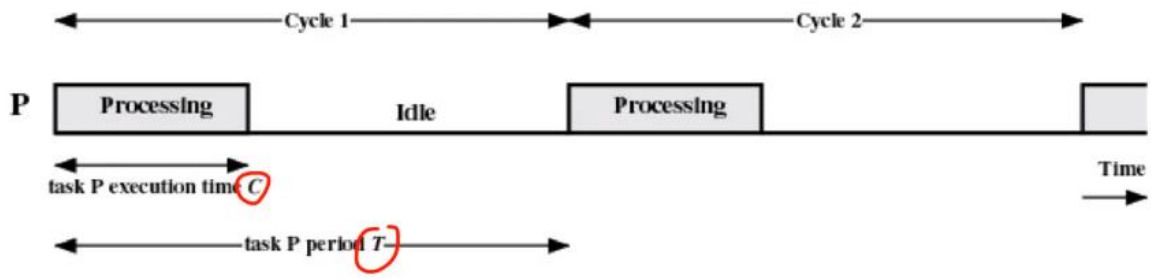


Figure 2 Monotono razporejanje

Ali ulovimo vse roke?

Pomembni parametri periodičnih zahtev? Ali ulovimo roke?



$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \cdots + \frac{C_n}{T_n} \leq 1$$

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \cdots + \frac{C_n}{T_n} \leq n(2^{1/n} - 1)$$

Task P_1: $C_1 = 20; T_1 = 100; U_1 = 0.2$
Task P_2: $C_2 = 40; T_2 = 150; U_2 = 0.267$
Task P_3: $C_3 = 100; T_3 = 350; U_3 = 0.286$

$0.2 + 0.267 + 0.286 = 0.753$

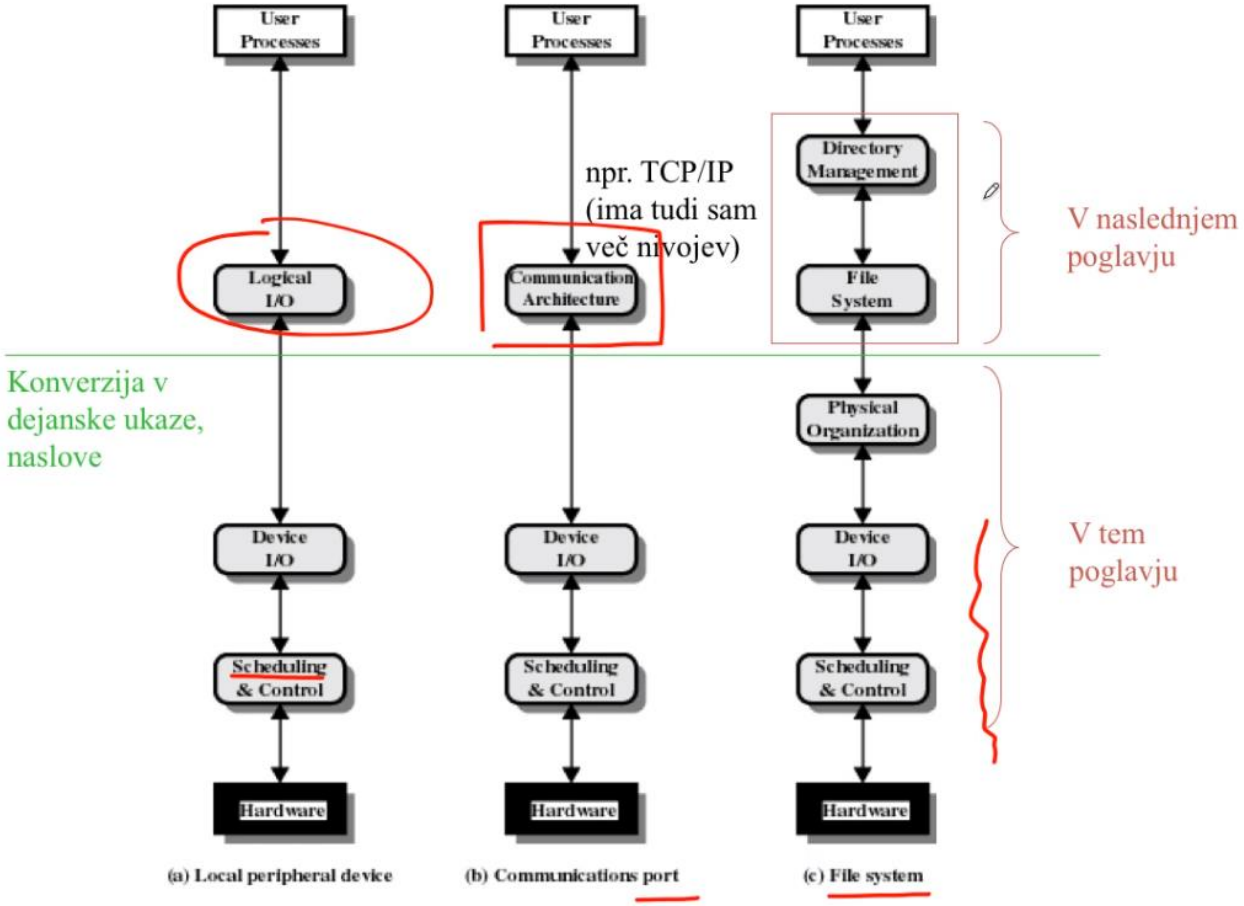
$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} \leq 3(2^{1/3} - 1) = 0.779$

Da. $0.753 < 0.779$

- **Dinamično plansko osnovano**
- **Dinamično celostno**

Upravljanje V/I in detalneje diska

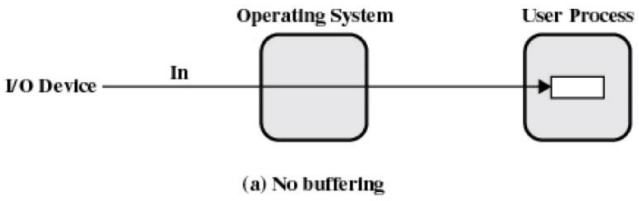
Organizacija V/I:

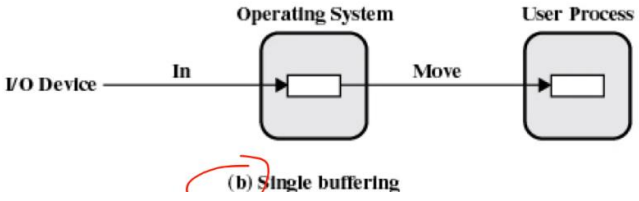


Medpomnilnik V/I

Zakaj potrebujemo medpomnlinik?

- Izogibanje problemom pri **odstranjevanju** in **hitrosti.**

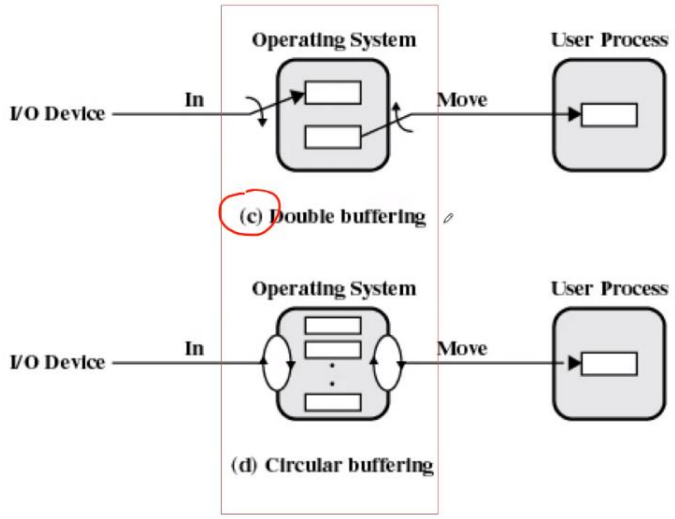




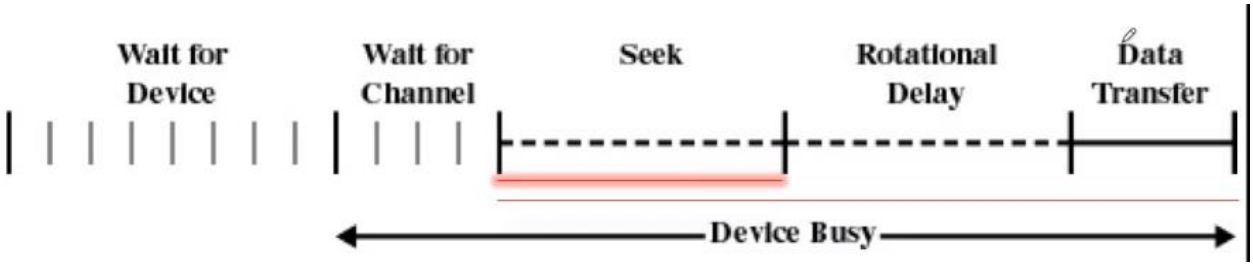
Kako povečuje hitrost?

- Medtem ko se en proces izvaja, drugega že nalaga v pripravo

Kako pospešimo komunikacijo?:



Parametri učinkovitosti diska:



Če zmanjšamo seek time, čas iskanja pohirimo disk oz sistem.

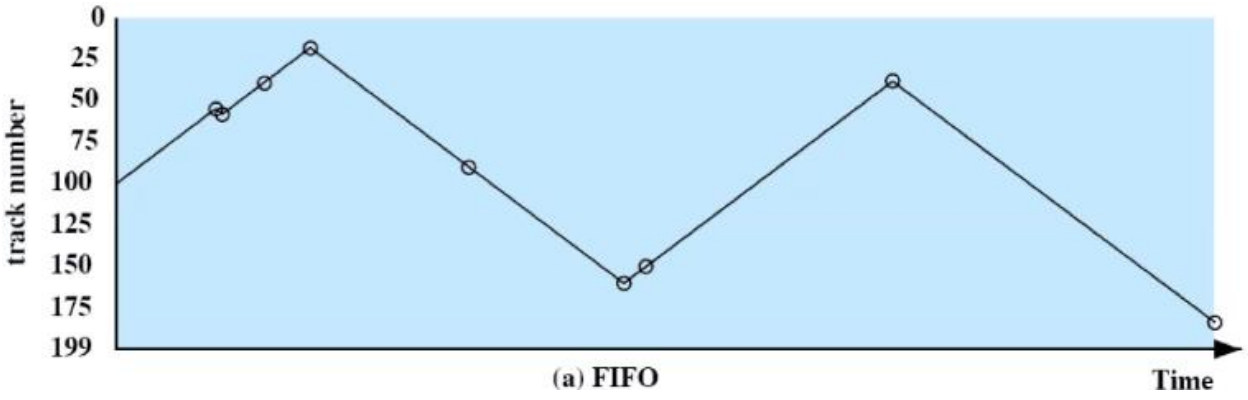
Načini razporejanja dostopa do diska

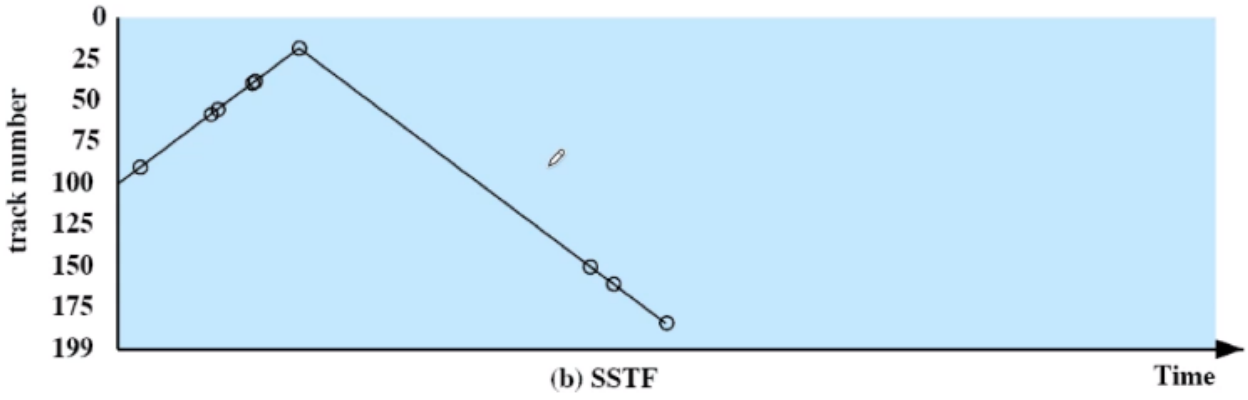
1. **Naključno** (kot benchmarking baseline)
2. **FIFO** (tko ko pridejo povrst jih daš notr)
3. **Prioritetno**
4. **LIFO**
5. **Najprej tista z najkrajšim časom premika roka** (SSTF) - če gremo navzdol (pogledamo celoten seznam in gremo do najnižje št. Potem se obrnemo in gremo do najvišje (39,38,18, 150, 160, 184)
6. **Skeniranje** (scan) – gre do najvišje, potem najde najvišjo od najnižjih in gre navzdol (150, 160, 184, 90, 58, 55, 39, 38...)

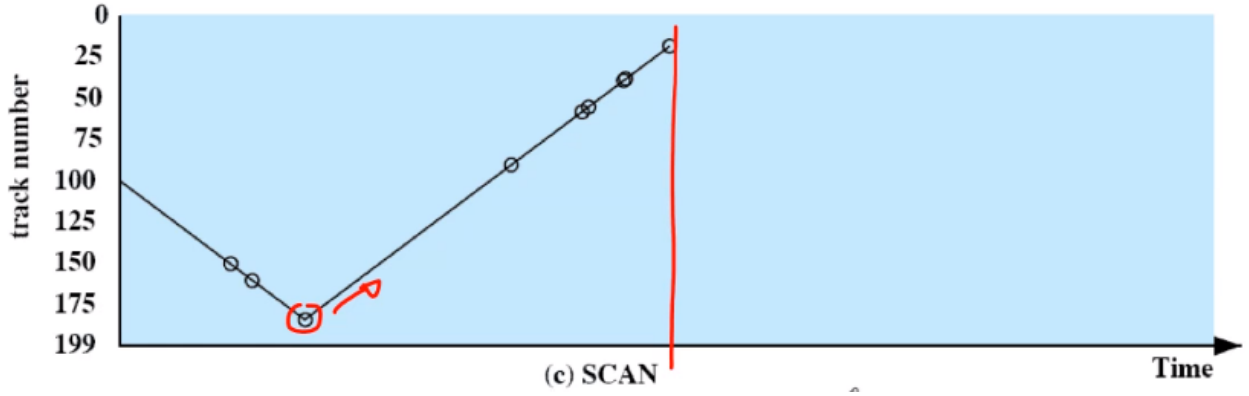
7. **Krožno skeniranje** (c-scan) skenranje samo v eno smer. Poskenira nek del kroga, ter premika naprej. Analogna ura. Zaasd (150,160, 184, 18, 38, 39, 55, 58, 90)
8. **N-koračno skeniranje** (N-step-scan) Vse zahteve razdelimo v vrste dolžine N. procesira vrsto za vrsto – razporedimo od najvišje do najmanjše, in vzamemo n št. Vn iz seznama. Ko najdemo največjo, se obrnemo in gremo potem v tisto smer
9. Pravično skeniranje (fscan)

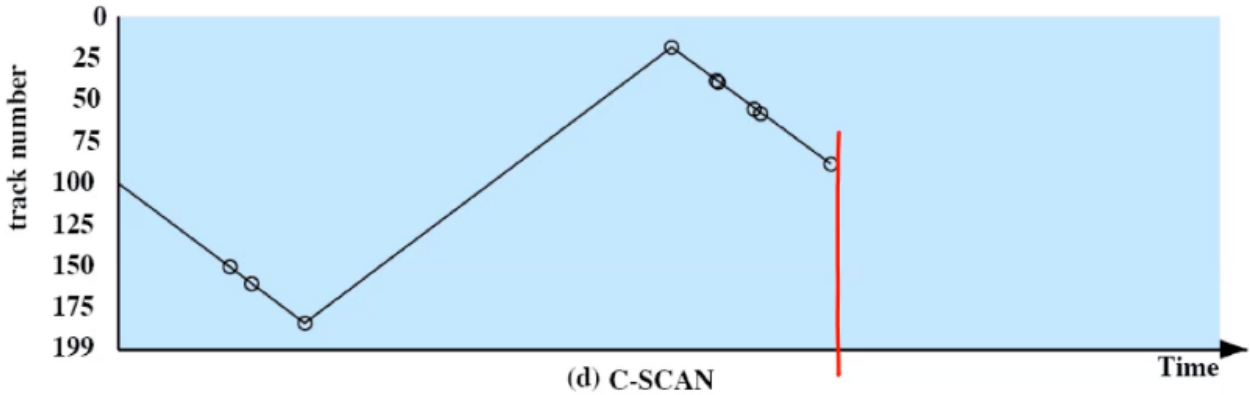
Primer:

- Disk ima 200 sledi
- Glava je na sledi 100
- Vrsta zahtev (dostopa do diska) je sestavljenja iz naključčnih zahtev
- Razporejevalnik diska dobi zahteve (po sledeh) v naslednjem vrstnem redu:
    - 55, 58, 39, 18 90, 160, 150, 38, 184
- Če algoritem dostopa ne predvideva drugače, glava potuje od 100 sledi proti koncu









Izračunamo AvrageSeekLength (povprečno dolžino iskanja) –

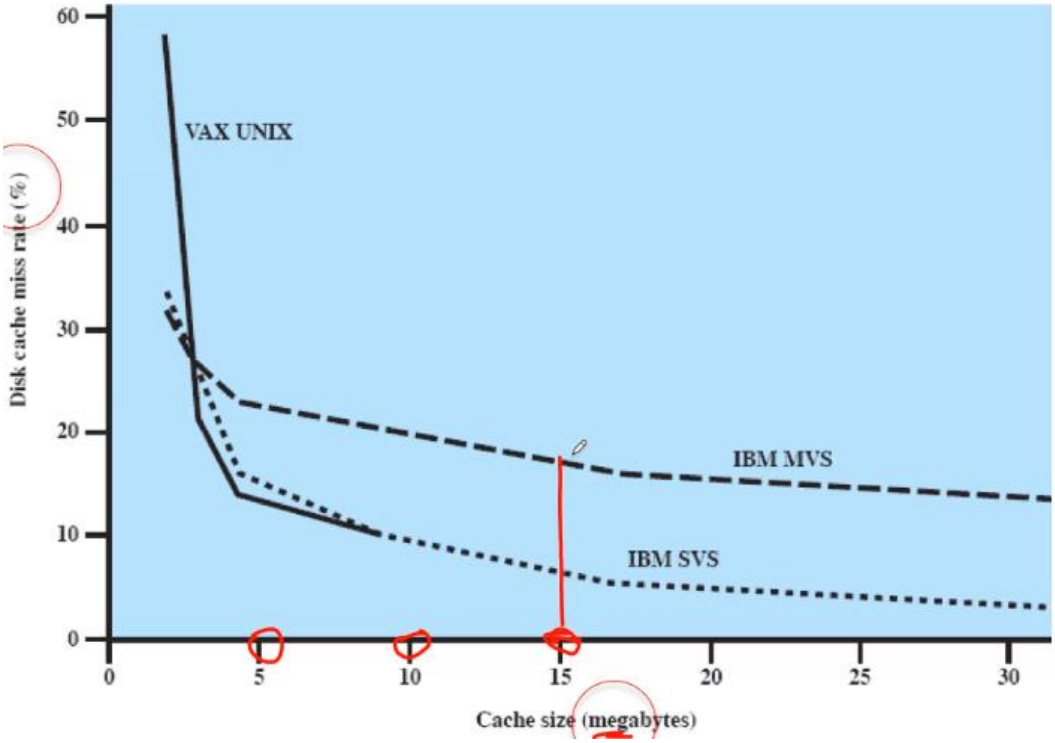
55, 58, 39, 18 90, 160, 150, 38, 184

(a) FIFO (starting at track 100)		(b) SSTF (starting at track 100)		(c) SCAN (starting at track 100, in the direction of increasing track number)		(d) C-SCAN (starting at track 100, in the direction of increasing track number)	
Next track accessed	Number of tracks traversed	Next track accessed	Number of tracks traversed	Next track accessed	Number of tracks traversed	Next track accessed	Number of tracks traversed
55	45	90	10	150	50	150	50
58	3	58	32	160	10	160	10
39	19	55	3	184	24	184	24
18	21	39	16	90	94	18	166
90	72	38	1	58	32	38	20
160	70	18	20	55	3	39	1
150	10	150	132	39	16	55	16
38	112	160	10	38	1	58	3
184	146	184	24	18	20	90	32
Average seek length	55.3	Average seek length	27.5	Average seek length	27.8	Average seek length	35.8

Kako lahko pospešimo dostop do diska?

- Več neodvisnih diskov (vzporedni dostop) RAID
- Predpomnilnik diska

Razmerje količine predpomnilnika in številom pogreškov (zgrešitev?):



Upravljanje z datotekami

Zakaj je datoteka središčni element aplikacije?

- Večina aplikaciji **dobi** vhodne podatke iz datoteke
  (očitna izjema je npr. Aplikacije, ki deluje v realnem času)
- Izhod se navadno **shranjuje** v datoteke – za nadaljno obelavo in arhiv
- **Upravljalec datotek** je del OS oziroma izkorišča del njegovih datotek

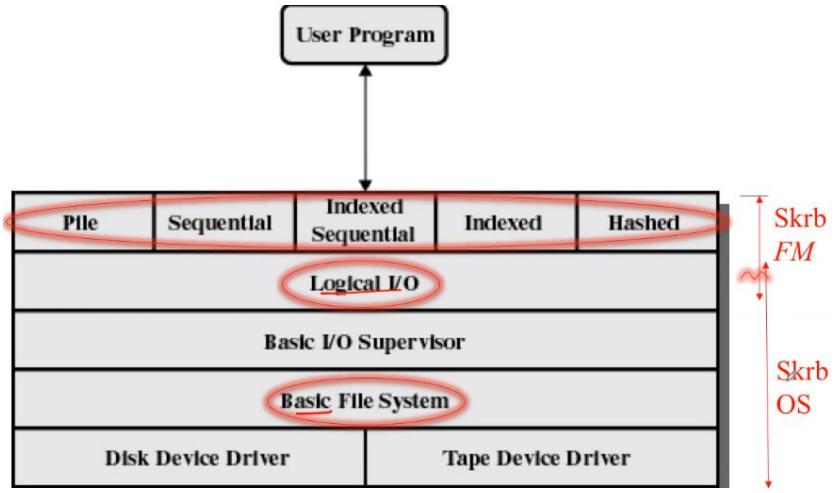
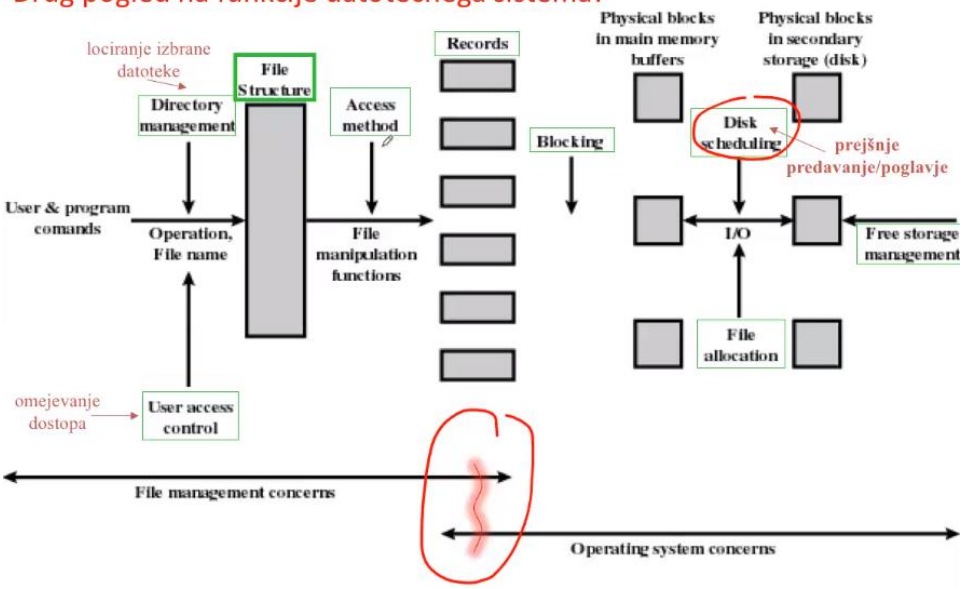


Figure 3 Softwerska arhitektura datotečnega sistema:

Metoda dostopa

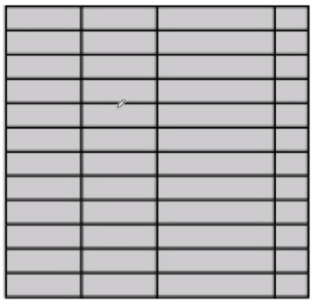
Drug pogled na funkcije datotečnega sistema!



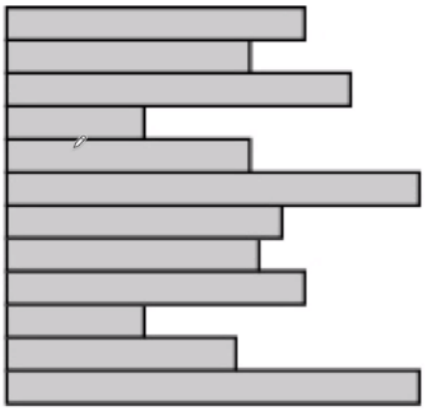
Logična organizacija datotek oz. strukturiranje zapisov

- Posledica metode dostopa

1. Nakopičena datoteka (File pile)
2. Zaporedna datoteka:



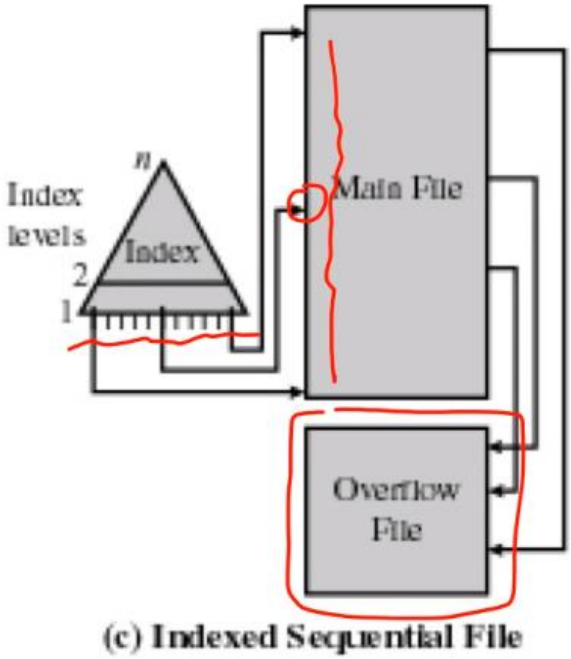
Fixed-length records
Fixed set of fields in fixed order
Sequential order based on key field



Variable-length records
Variable set of fields
Chronological order

Figure 4 Pile file

3. Indeksirana zaporedna datoteka



(c) Indexed Sequential File

Vsak zapis v glavni datoteki ima kazalec na datoteko prekoračitev oziroma **novih/dodanih zapisov** (*overflow file*)
=>
posodabljanje glavne datoteke in indeksa **v svežnju**

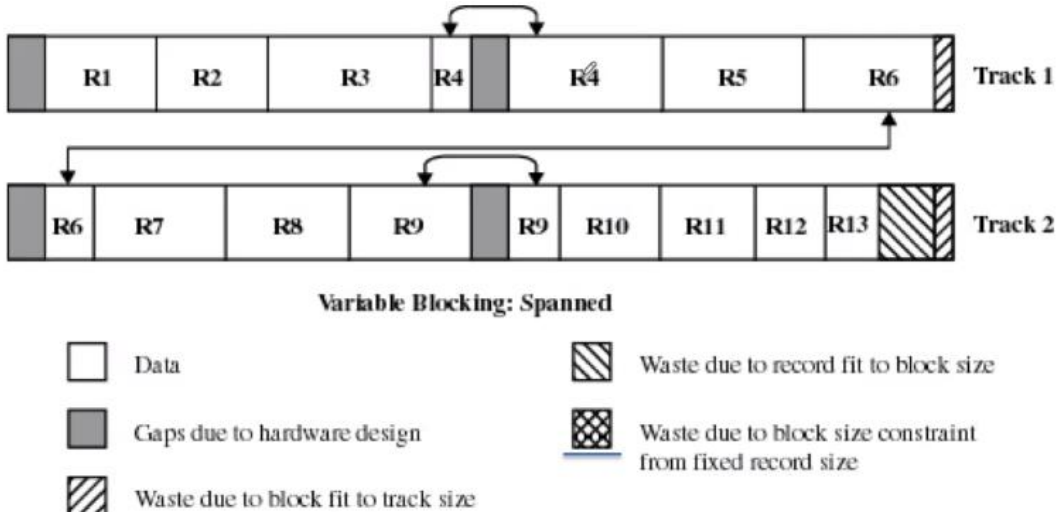
4. Indeksirana doatoteka
5. Neposredna datoteka (direct, hashed)
6. Usmerjevalna datoteka (file directory)

Fizična organizacija datotek

- Posledica strategije deliteva na bloke
  in

  Zaseganja prostora za doaoteke na pomožnem pomnilniku

- Blok lahko vsebuje več strani
  1. Bloki zapisov nespremeljivih velikosti
  2. Bloki zapisov spremeljivih velikosti:
     a. **Delitev zapisov**



Variable Blocking: Spanned

Data
Gaps due to hardware design
Waste due to block fit to track size
Waste due to record fit to block size
Waste due to block size constraint from fixed record size

3. Bloki zapisov

Variable Blocking: Unspanned

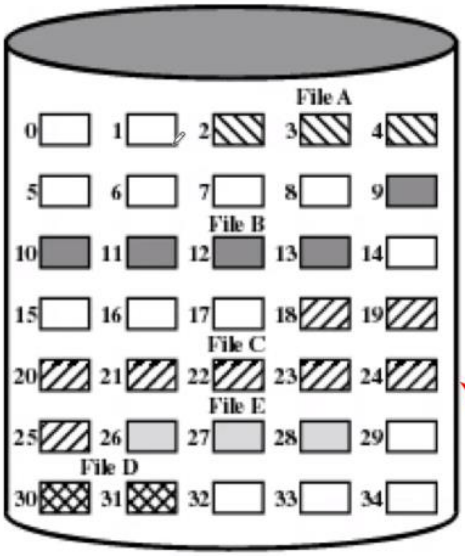
Data
Gaps due to hardware design
Waste due to block fit to track size
Waste due to record fit to block size
Waste due to block size constraint from fixed record size

Upravljanje pomožnega pomnilnika

- Zaseganje prostora za datoteke
- Vzdrževanje informacije o praznem prostoru
- **Tabela, ki zaseže dele (bloke) povezuje v datoteko (File Allocation Table, FAT; DAT)**

Metode zaseganja

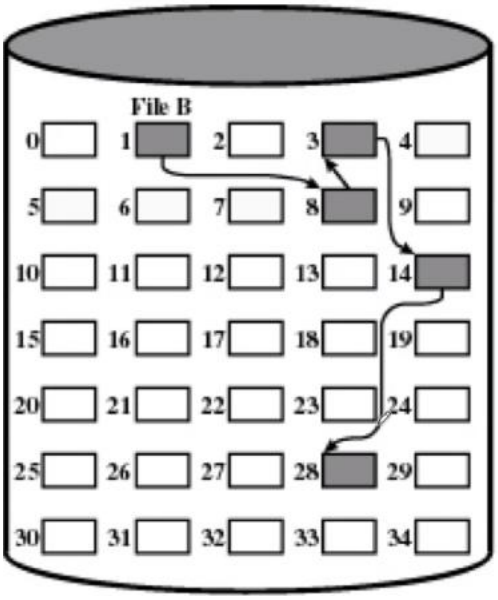
1. Zvezno zaseganje



File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

2. Verižno zaseganje



File Allocation Table

File Name	Start Block	Length
...	...	...
File B	1	5
...	...	...

3. Indeksirano zaseganje