S A D R Ž A J

6.1- Pojam viruelne memorije

- ***Virtuelna memorija*** je strategija dodele memorije koja omogućava da se samo deo programa koji se izvršava nalazi u radnoj memoriji.
- Osnovna prednost ovakvog pristupa je da program može biti i veći od radne memorije jer svaki proces ima iluziju posedovanja cele memorije
- Tako korisnički program može imati proizvoljnu veličinu, a sistem za upravljanje memorijom preslikava logički prostor korisnika u ograničeni prostor u radnoj memoriji.
- Korisnik ima utisak da je radna memorija neograničena
- Osnovna ideja virtuelne memorije- da ukupna veličina programa, steka i podataka može prekoračiti veličinu raspoložive fizičke memorije
- Loša implementacija ovakvog sistema može značajno smanjiti performanse celog računarskog sistema.
- Osnovna tehnika kojom OS upravlja virtuelnom memorijom je **alokacija stranica** na koje su programi i podaci podeljeni
- Ono što ne može da smesti u radnu memoriju, OS smešta u sekundarnu

6.1- Zašto se uvodi virtuelna memorija

- Analize ukazuju da obično nije potrebno da ceo kod programa bude istovremeno prisutan u radnoj memoriji da bi se on izvršio:

1. Programi sadrže procedure za obradu slučajnih ili namernih grešaka.
2. Programi za polja, liste, tablice i slične statičke strukture obično rezervišu više memorije nego što je stvarno potrebno.
3. Pojedine opcije programa relativno se retko koriste. (primer kod *Microsoft Office Word*-a korišćenje *Equation* editora).

- Osnovne prednosti kada se samo deo programa nalazi u radnoj memoriji su:

1. Veličina programa nije ograničena veličinom radne memorije
2. Korisnički program može se izvršiti sa znatno manjom radnom memorijom čime se povećava iskoristivost kao i propusna moć sistema - paralelno izvršavanje većeg broja programa.
3. Manje U/I operacija je potrebno za prebacivanje korisničkih programa iz i u radnu memoriju. Tako se korisnički programi brže izvršavaju.

6.1- Realizacija virtuelne memorije

1. **Stranična VM** - podela VM na delove fiksne dužine - **stranice**
2. **Segmentna VM** - podela VM na delove varijabilne dužine - **segmente**
3. **Segmentno-stranična** VM - podela VM na delove varijabilne dužine koji se sastoje od delova fiksne dužine

Prilikom implementacije VM moraju se rešiti sledeći problemi:

1. **Mehanizam mapiranja adresa** - ***način na koji se logičke adrese prevode u fizičke adrese***.
   - Prevođenje se vrši jednom ili dva puta prilikom izvršavanja svake instrukcije.
   - OS čuva tabele koje definišu način prevođenja adresa.
2. **Strategija pozicioniranja** - ***Ggde u fizičkoj memoriji smestiti delove virtuelne memorije*** ?
   - Kod segmente organizacije se koristi jedan od algoritama iz sistema sa varijabilnim particijama *(first-fit, best-fit, next-fit, worst-fit).*
   - Kod stranične organizacije je pozicioniranje uprošćeno jer se koristi prva slobodna fizička stranica.

6.1- Realizacija virtuelne memorije

3. **Strategija zamene** - ***Šta uraditi kada je potrebno u fizičku memoriju dovesti segment iz VM a nema dovoljno mesta***?
- Kod sistema bez VM, kompletna memorija jednog procesa je smeštena na disk u *swap datoteku*.
- Kod VM, može se smestiti samo jedan deo, stranica ili segment (koriste se algoritmi **FIFO** i **L**ast **R**ecently **U**sed).

4. **Kontrola zauzeća** - ***Koliki deo VM čuvati u fizičkoj memoriji***?
- Ako se čuva preveliki deo, dolazi do dugih *swapping-a* jer memorija različitih procesa se stalno premešta između swap i fizičke memorije.
- Ako se čuva premali deo, dolazi do čestog *swapping-a* jer se delovi procesa stalno dovlače iz *swap* datoteke u fizičku memoriju.
- Dovlačenje po potrebi (po zahtevu) *demand paging / segmentation*.
- Kada stranica nije u memoriji generiše se stranični izuzetak *(page/segment fault trap)*

5. **Deljenje memorije** - Pojedini procesi mogu deliti iste stranice ili segmente u fizičkoj memoriji. Češće se sreće kod segmentne organizacije VM jer je podela na segmente logička a na stranice fizička

6.2- Demand Paging - DP

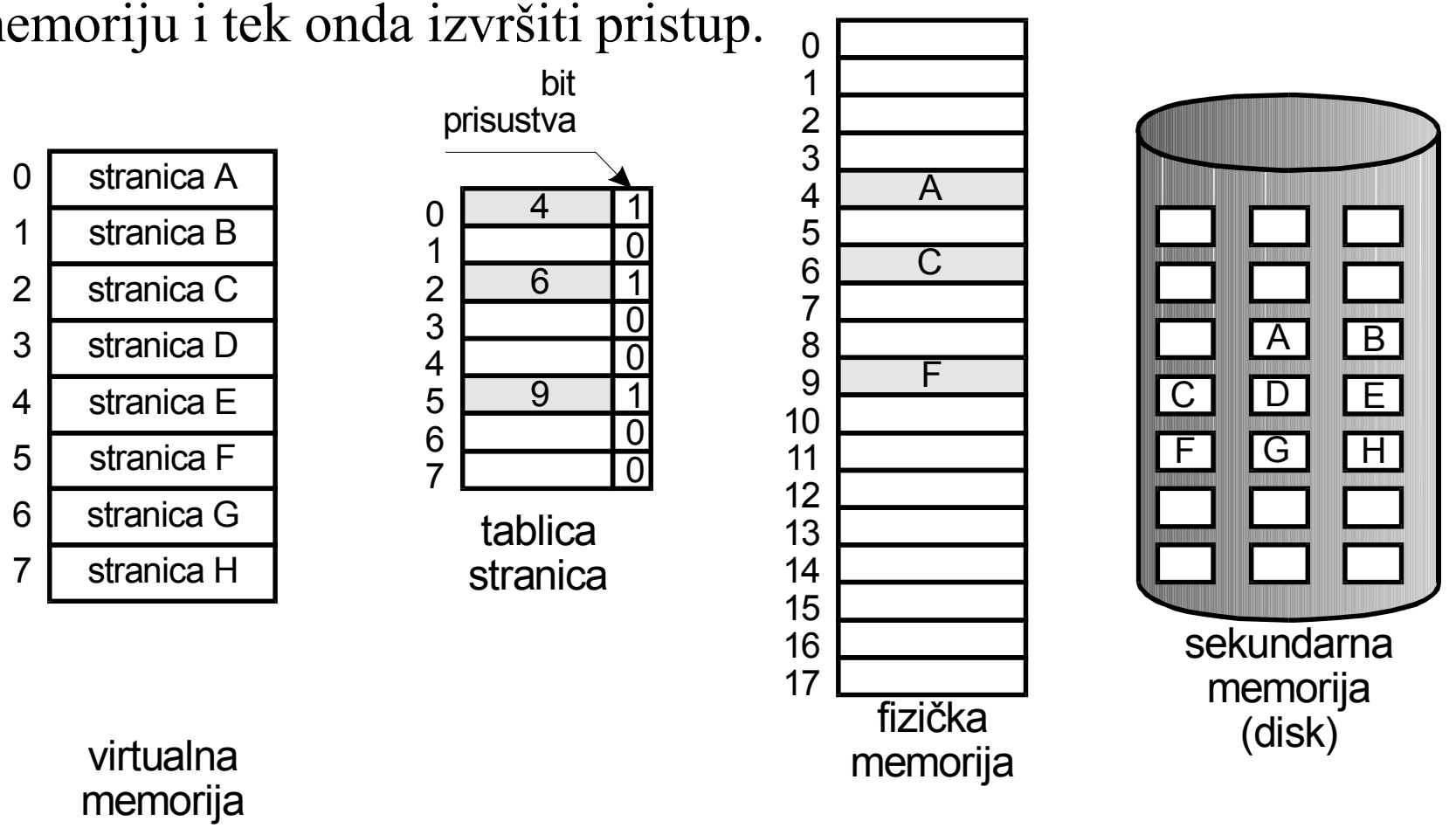
- Virtuelna memorija deli logički i fizički adresni prostor u stranice-*page*
- Sve stranice obuhvataju isti mem.prostor,jednak nekom stepenu broja 2
- Logički adresni prostor se sastoji od logičkih stranica, koje se nalaze u sekundarnoj memoriji, a fizički adresni prostor sačinjavaju fizičke stranice, koje su u radnoj memoriji.
- Pošto stranica predstavlja jedinicu prenosa na relaciji sekundarna-radna memorija, prirodno je da ona obuhvata ceo broj blokova sek.memorije
- Podela logičkog i fizičkog adresnog prostora u stranice znači da se adresa svake lokacije sastoji od adrese stranice i adrese pomeraja-*offset*
- Adresa pomeraja (*offset*) u stranici je ista kod logičke i fizičke adrese.
- Pri tome, adresu pomeraja u stranici određuje **n** manje značajnih bitova i u logičkoj i u fizičkoj adresi, ako stranica obuhvata $\mathbf{2^n}$ bajta.
- Preostali, značajniji bitovi logičke, odnosno fizičke adrese određuju adresu logičke, odnosno fizičke stranice.
- Pošto je logički adresni prostor veći od fizičkog, on sadrži više stranica

6.2- Učitavanje stranica prema potrebi

- Proces je smešten na sekundarnoj memoriji, obično na disku.
- Proces se može posmatrati kao niz stranica koji se prema potrebi upisuju u OM.
- Kada se želi izvršiti proces, upisuje se samo jedan njegov deo (jedna ili više stranica) u OM.
- Kod upisivanja procesa u OM koristi se lenji prebacivač (***lazy swapper***) koji upisuje stranicu u memoriju tek kada je ona potrebna.
- Ovakva realizacija zahteva i određenu hardversku podršku: hardver za straničenje i hardver za razmenjivanje stranica
- Za realizaciju VM obavezne su tabela stranica i sekundarna memorija
- U tabeli stranica eksplicitno se zahteva prisustvo bita validnosti
- Kada se pristupa pojedinoj stranici, OS preko tablice stranica (bit validnosti) ispituje da li je adresirana stranica u memoriji ili ne.
- Vrednost bita **v** (*valid*) ili **1** ukazuje da se logička stranica nalazi u OM
- Vrednost bita **i** (*invalid*) ili **0** ukazuje da se logička stranica ne nalazi u OM ili da stranica ne pripada adresnom prostoru sekundarne memorije

6.2- Učitavanje stranica prema potrebi

- Ako je stranica u upisana u fizičku memoriju, bit validnosti je postavljen na 1, tada se izračunava fizička adresa naredbe ili podatka i pristupa mu se.
- Ako adresirana stranica nije u memoriji tada je potrebno nju upisati u memoriju i tek onda izvršiti pristup.

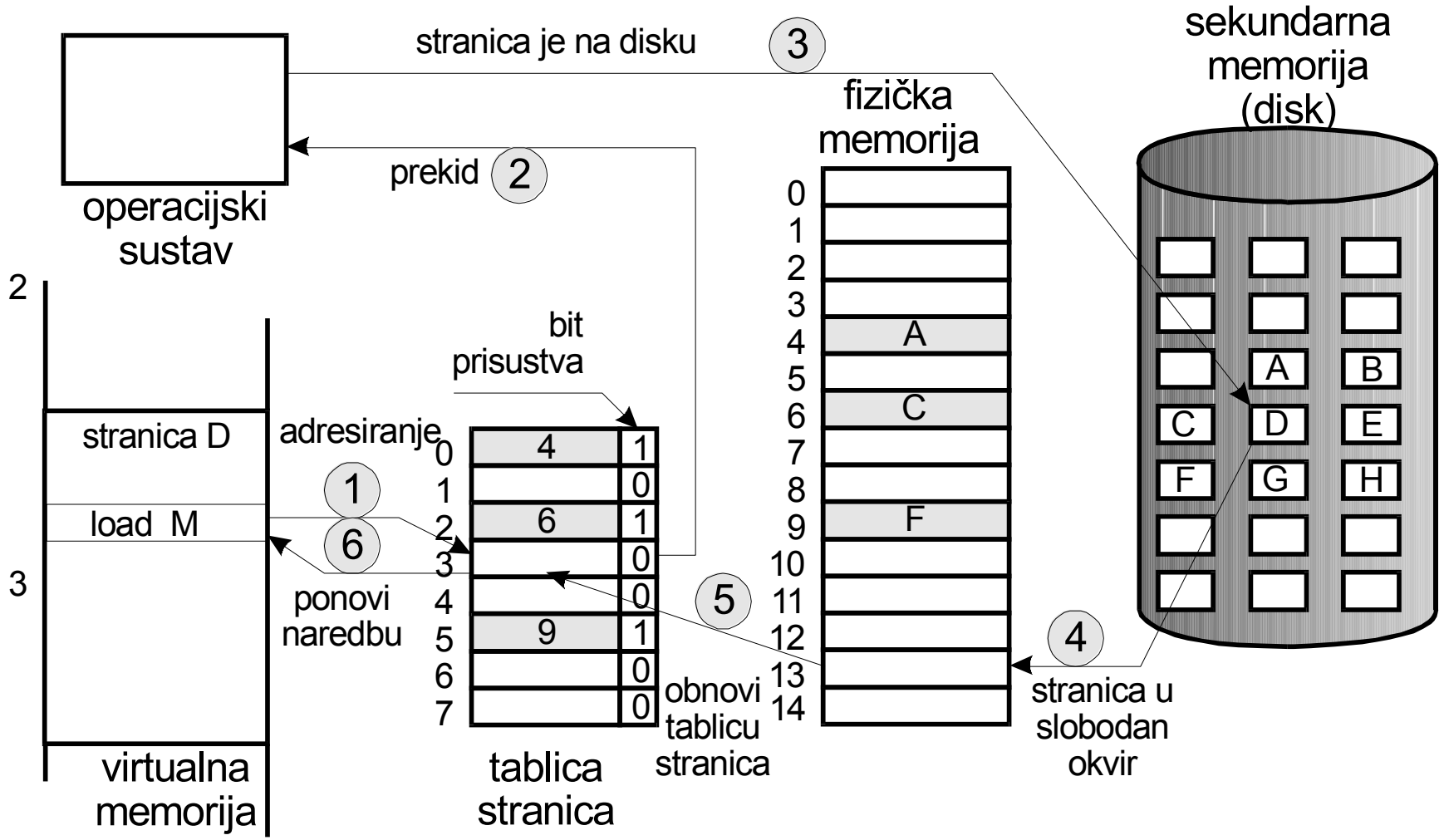


6.2- Učitavanje stranica prema potrebi

Procedura pristupa stranici može se opisati na sledeći način:

1. Prvo se proverava bit validnosti adresirane stranice kako bi se odredilo da li je stranica u memoriji ili ne.
2. Ukoliko stranica nije u memoriji (došlo je do tzv. ***promašaja***) generiše se prekid koji javlja OS da treba da pronađe stranicu u sekundarnoj memoriji i prebaci je u radnu memoriju. Obično promašaj rezultira prekidom prava korištenja procesora, te se proces prebacuje u red čekanja na U/I uređaj, u ovom slučaju hard disk.
3. OS pronalazi slobodan okvir u radnoj memoriji (OS vodi listu slobodnih okvira).
4. Prebacuje se tražena stranica u odabrani okvir.
5. Osvežava se tablica stranica procesa na način da se stranici pridružuje dodeljeni okvir. Ovim je praktično proces spreman da nastavi sa izvršavanjem.
6. Prekinuta naredba se ponovo izvodi a stranici se pristupa kao da je ona oduvek bila u memoriji.

6.2- Učitavanje stranica prema potrebi



6.2- Problemi kod DP

- Glavni problem pri korišćenju DP tehnike su procesorske instrukcije za pomeranje ili prebacivanje blokova podataka (stranica)
- Ove instrukcije ne smeju da se izvršavaju ponovno od početka nakon PF (*Page Fault*) prekida

  Šta se dešava ako OS utvrdi da u memoriji nema slobodnih okvira ?
- Veoma česti slučaj kod višeprocesnog (*multy tasking*) rada
- Jedini način je da se neka od stranica koje se nalaze u OM izbace iz nje
- Izbor stranice koja će se izbaciti, tzv. ***žrtvu***, posebno je osetljivo pitanje i izbor algoritma zamene može značajno uticati na performanse sistema.
- Informaciju o tome da li se nešto menjalo u sadržaju stranice moguće je dobiti uvođenjem zasebnog bita, bita izmene (*modify bit*, *dirty bit*).
- Algoritam za zamenu stranica prvo proverava bitove izmene stranica koje bi se mogle izbaciti i bira one koje nisu menjane, odnosno one kojima je bit izmene nula.

6.3-Alternativne tehnike učitavanja stranica

1. **Tehnika CoW** (*Copy on Write*) - zasniva se na principu da se stranice roditeljskog procesa inicijalno ne kopiraju niti se nove stranice dodeljuju procesu detetu već oni dele sve stranice. Samo u slučaju da neki proces treba da modifikuje stranicu tada se vrši kopiranje te stranice i ona mu se dodeljuje.

2. **Straničenje unapred -** potrebno je poznavati listu stranica koje će proces zahtevati (suspednovan proces) koje se odmah sve učitavaju kada se želi da se proces aktivira (radni set-*working set* je ranije upamćen kada je proces suspendovan)

3. **Memorijski mapirane datoteke -** deo virtuelnog prostora se dodeljuje datotekama tj. stranice se posmatraju kao deo neke datoteke. Inicijalno se stranicama nekog procesa pristupa preko DP tehnike ali se naredni delovi (stranice) učitavaju kao deo datoteke pa se ne koristi sistemski poziv *read*().

6.4 - Algoritmi zamene

- U praksi se susreću brojne strategije zamene stranica koje imaju jedinstven cilj, zadržati što manji broj promašaja uz prihvatljivu hardversku i programsku podršku.

Zamena se može vršiti:

- **globalno**: kandidat za zamenu se traži među svim fizičkim stranicama bez obzira kom procesu pripadaju
- **lokalno**: kandidat za zamenu se traži među svim fizičkim stranicama tekućeg procesa

Optimalni algoritam zamene: zamenjuje se stranica koja u budućnosti najduže neće biti korišćena. Ovo je idealan algoritam ali neostvariv jer u realnim sistemima ne možemo predvideti koja stranica najduže neće biti korišćena-potrebno je poznavati unapred sve procese koji će se izvršavati

Slučajni algoritam zamene: stranica koja se zamenjuje slučajno se bira. Loše rešenje jer se obično koriste adrese koje su vremenski ili prostorno povezane.

6.4 – Algoritmi zamene

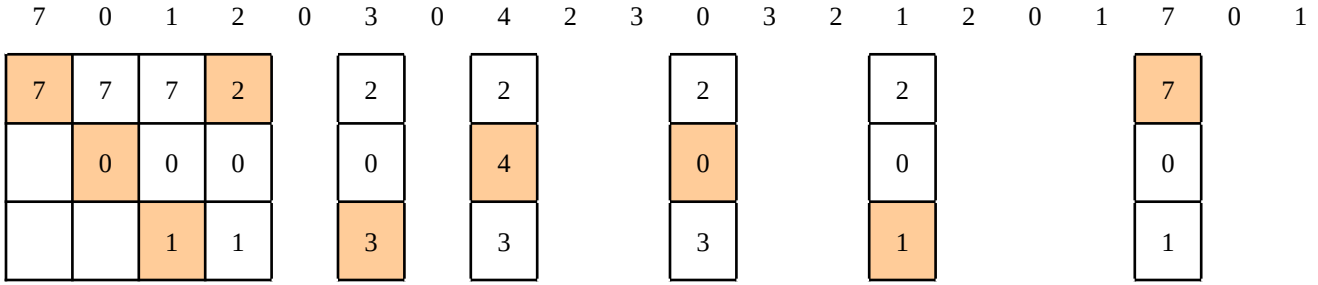
- Osnovni cilj je postizanje minimuma grešaka kod izbora stranica
- Rangiranje algoritama vrši se na osnovu niza memorijskih referenci i izračunavanja broja grešaka stranica na tom nizu referenci

1. Optimalni algoritam zamene stranica
2. Prva došla, prva odlazi (FIFO – *First In First Out*)
3. Algoritam druge prilike po principu sata (*Second Chance Clock*)
4. Ne skoro korišćena stranica (*Not Recently Used*)
5. Najmanje skoro korišćena stranica (LRU – *Last Recently Used*)
   - Ne često korišćena (NFU – *Not Frequently Used*)
   - Starenje (*Aging*)
6. Radni skup (*Working Set*)
7. Radni skup po principu sata (*WSClock*)

6.4 - Optimalni algoritam

- Optimalno bi bilo da se zameni ona stranica koja će se u budućnosti najkasnije koristiti, tako da se greška što je moguće kasnije odloži.
- Primenom ovog algoritma smanjen je broj promašaja na minimum.
- Najbolji algoritam, ali ga je praktično nemoguće implementirati, jer u trenutku pojave greške stranica, OS ne zna kada će se koja stranica u memoriji referencirati.
- U trenutku nastanka greške u memoriji su smeštene stranice različitih procesa od kojih se svaka može referencirati u budućnosti
- Koristi se za evaluaciju algoritama zamene stranica pa se primenjuje na osnovu prvog izvršenja programa i beleženja referenci u toku izvršenja

Primer: niz referenci **7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1** izvršava se u memoriji koja sadrži tri stranična okvira

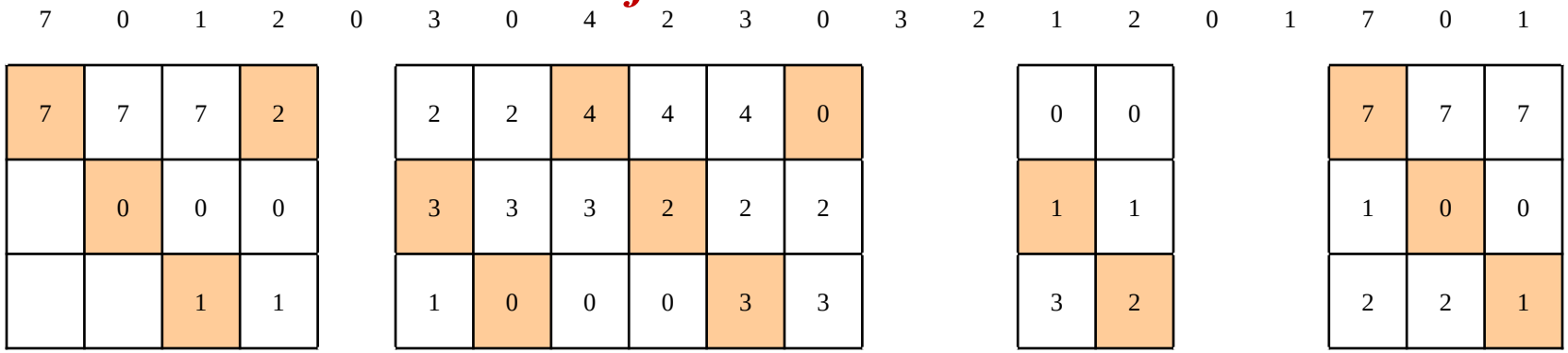


Algoritam generiše 9 promašaja stranica

6.4 - FIFO (First In/First Out)

- Najjednostavniji algoritam za implementaciju je zameniti stranicu koja je od stranica u memoriji prva unesena (*First In First Out*).
- Stranice se stavljaju u red kako se unose u memoriju i kada nema slobodnih okvira izbacuje se ona koja je prva u redu (najstarija) a nova se dodaje na kraj liste (najmlađa).
- Analiza pokazuje da je učestala zamena stranica kojoj se sledećoj pristupa, znatno povećava broj promašaja, a time i efikasnost sistema.
- FIFO algoritam se retko koristi u osnovnom obliku
- Tako je za prethodni primer **generisalo 15 promašaja stranica**.

Zamenjuje se stranica koja se najduže nalazi u memoriji. Loše rešenje jer se može desiti da se upravo stranica koja je najduže u memoriji i najčešće koristi



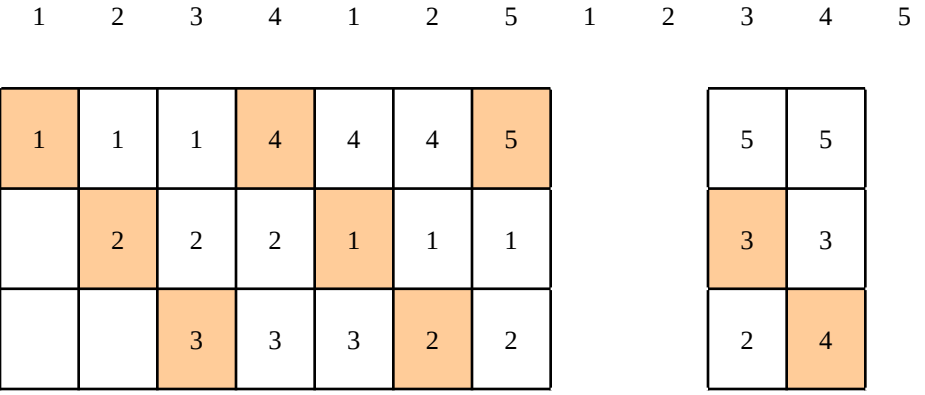
6.4 - FIFO (First In/First Out)

- Uz spomenuti ozbiljan nedostatak ovog algoritma, koji se ipak odlikuje jednostavnošću implementacije uočena je i sledeća anomalija koja se može ilustrirati na sledećim referentnim nizom stranica:
- Analiza je pokazala da je za tri dodeljena okvira usledilo devet promašaja.

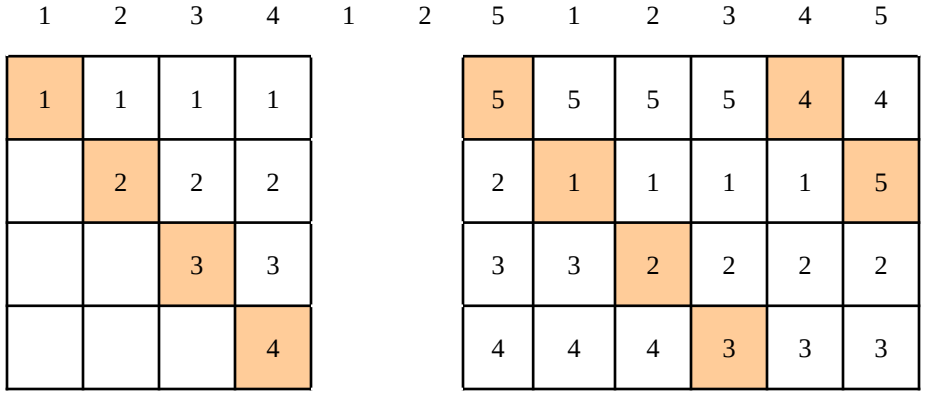
Neka se broj okvira poveća na četiri:

- Za očekivati je bilo da se broj promašaja smanji, ali broj promašaja se povećao i sada je to 10 promašaja.
- Ovo se naziva ***Beladyeva*** anomalija, a posledica je zamene stranice koja će biti adresirana u sledećem pristupu memoriji.

Tri okvira

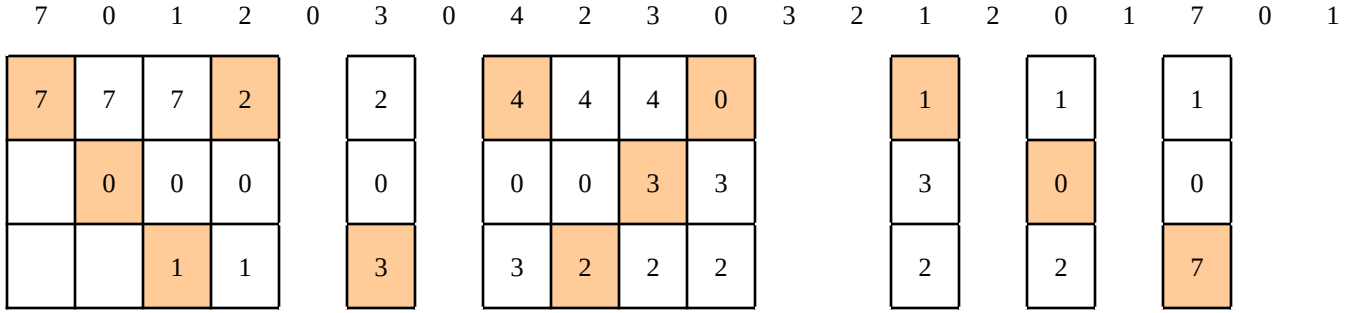


Četiri okvira



6.4-Algoritam druge prilike po principu sata

- Jednostavna modifikacija FIFO algoritma koja pored starosti stranica uzima u obzir i vrednost R bita - postavlja se na 1 kada je stranica referencirana, a resetuje se na 0 svaki put sa prekidom tajmera ~ 20ms.
  - Ako je R=0, stranica je stara i nije skoro referencirana pa se zamenjuje novom stranicom
  - Ako je R=1, bit R se postavlja na 0 i stranica se postavlja na kraj liste stranica (daje joj se druga prilika pre izbacivanja) i ispitivanje se nastavlja
- Ukoliko su R bitovi svih stranica postavljeni na 1 - FIFO algoritam
- Implementacija se svodi da se sve stranice povežu i vidu ciklične liste, pri čemu se kazaljka na satu koristi kao pokazivač na najstariju stranicu koja se ispituje kao kandidat za zamenu



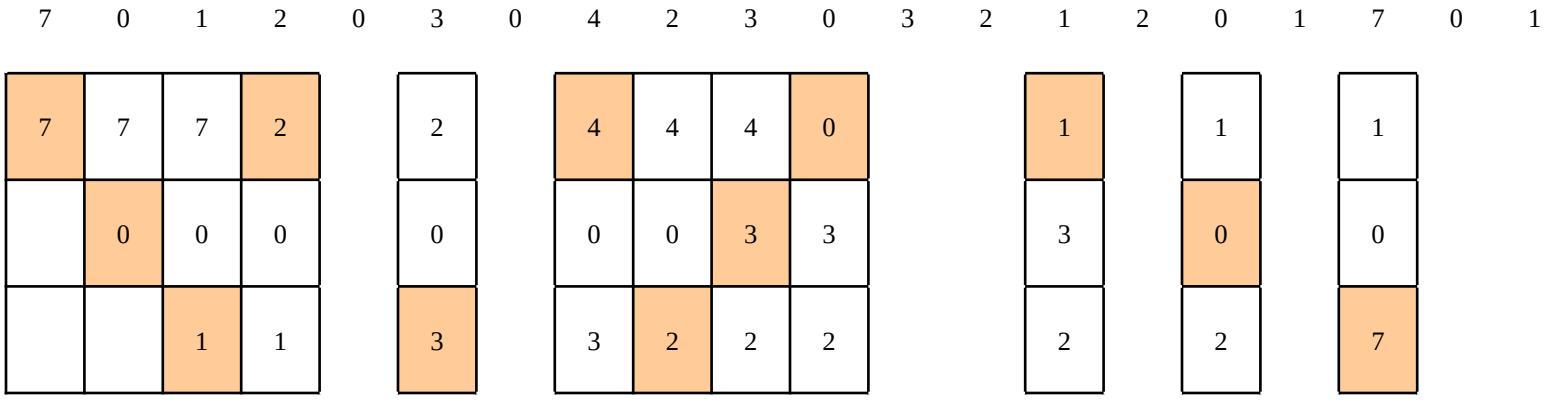
Algoritam generiše 12 promašaja stranica

6.4 - NRU (Non Recently Used)

- Ovde se dodaje još jedan bit M=1 kada je stranica modifikovana
- Kada se proces startuje oba bita (R i M) koja su pridružena njegovim stranicama se postavljaju na 0
- Periodično, na svaki prekid tajmera (10-100ms) R bit se resetuje na 0
- Prilikom greške stranica, OS pregleda bitove R i M svih stranica i na osnovu njih formira četri klase stranica:
  - **Klasa 0** – stranica nije referencirana ni modifikovana (R=0, M=0)
  - **Klasa 1** – stranica nije referencirana ali je modifikovana (R=0, M=1)
  - **Klasa 2** – stranica je referencirana ali nije modifikovana (R=1, M=0)
  - **Klasa 3** – stranica je referencirana i modifikovana (R=1, M=1)
- Algoritam zamenjuje stranicu iz neprazne klase sa najnižom oznakom
- Kod izbora stranice potrebno je obići nekoliko puta cikličnu listu dok se ne nađe stranica koja treba da bude zamenjena
- Koristi se kao proširenje algoritma druge prilike po principu sata

6.4 - LRU (*Last Recently Used*)

- Zasniva se na pretpostavci da će stranice koje su zadnje korišćene opet biti u budućnosti korišćene
- Prilikom greške stranica, predloženo je rješenje zamene stranice koja se u prošlosti najkasnije koristila (LRU – *Last Recently Used*).
- Primena ovog algoritma svodi se formiranje lančane liste svih stranica, pri čemu je najskorija korišćena stranica na početku liste, a najmanje skoro korišćena stranica na kraju
- Lančana lista se ažurira posle svakog memorijskog referenciranja
- Ovaj algoritam ne pati od ***Beladyeve*** anomalije - najčešće primjenjivan



Algoritam generiše 12 promašaja stranica

6.4 - LRU (Last Recently Used)

Hardverska realizacija

1 varijanta

- Uvodi se **64-bitni brojač** koji se inkrementira nakon svake instrukcije
- Zatim se njegova vrednost upisuje u polje pridruženo svakoj stranici koja se referencira.
- Menja se stranica čija vrednost polja je najmanja

2 varijanta

- Uvodi se matrica **n** x **n** bita gde **n** predstavlja broj straničnih okvira.
- Pri referenciranju neke stranice **k**, svi bitovi u vrsti **k** se postavljaju na 1 a svi bitovi kolone **k** se postavljaju na 0.
- Stranica koja ima najmanju vrednost u odgovarajućoj vrsti matrice je kandidat za zamenu

6.4 - LRU (Last Recently Used)

Softverska realizacija

1. NFU(Non Frequently Used)-LFU(*Last Frequently Used*)

- Svakoj stranici se pridruži brojač koji je inicijalno postavljen na 0.
- Pri svakom vremenskom prekidu OS pregleda sve stranice i vrednost R bita se dodaje brojaču.
- Stranica sa najmanjom vrednošću R bita se menja.

2. Starenje (*Aging*)

- Sadržaj brojača se pomera za jedan bit udesno a vrednost R bita se dodaje na krajnju levu poziciju (bit najviše težine).
- Najmanja vrednost brojača definiše najstariju stranicu koja se menja.

6.5 – Dopunska razmatranja

1. Veličina stranica - za dugačke stranice interna fragmentacija je izražen problem. Za kratke stranice - izražen problem veličine tabele stranica.
2. Brzina učitavnja stranica - mapiranje stranica mora biti veoma brzo.
3. Uticaj na strukturu programa – traži se da programer vodi računa o načinu programiranja kako bi se smanjio broj promašaja.

Primer: ako se pristupa matrici po kolonama (1024^2 promašaja) a ako joj se pristupa po vrstama onda imamo 1024 promašaja jer su matrice smeštene po principu jedna vrsta po stranici.

4. Zaključivanje stranica -nekada se zahteva da neke stranice zaključamo kako ih neki drugi proces ne bi izbacio i time izgubio te podatke.

Primer: U/I operacije kod kojih neka stranica može predstavljati bafer učitanih podataka. Ovo se može sprečiti zabranom baferovanja U/I operacija u korisničkoj memoriji, pa se to radi u U/I baferima jezgra. Ali to nije dobro jer se kasnije opet zahteva kopiranje iz bafera jezgra u korisničke bafere. Zaključana stranica ne dozvoljava da bude dodeljena nekom drugom procesu pa se i ne može izbaciti.

Hvala na pažnji !!!

Pitanja

? ? ?