nedeljainformatike v8.0

Virtuelna memorija (kako se računar nikada ne pretrpa)

Elena Galjak

Matematička gimnazija

16. 05. 2023.

Agenda

1. Malo o osnovnim stvarima
2. Organizacija memorije
3. Optimizacija prostora
4. Algoritmi za poboljšanje performansi

Malo o osnovnim stvarima

- Šta je **pokazivač**?
- Šta je **memorija**?
- Šta je **operativna memorija**?
- Šta znači da je **računar 64-bitni**?
- Šta je **proces**?

Agenda

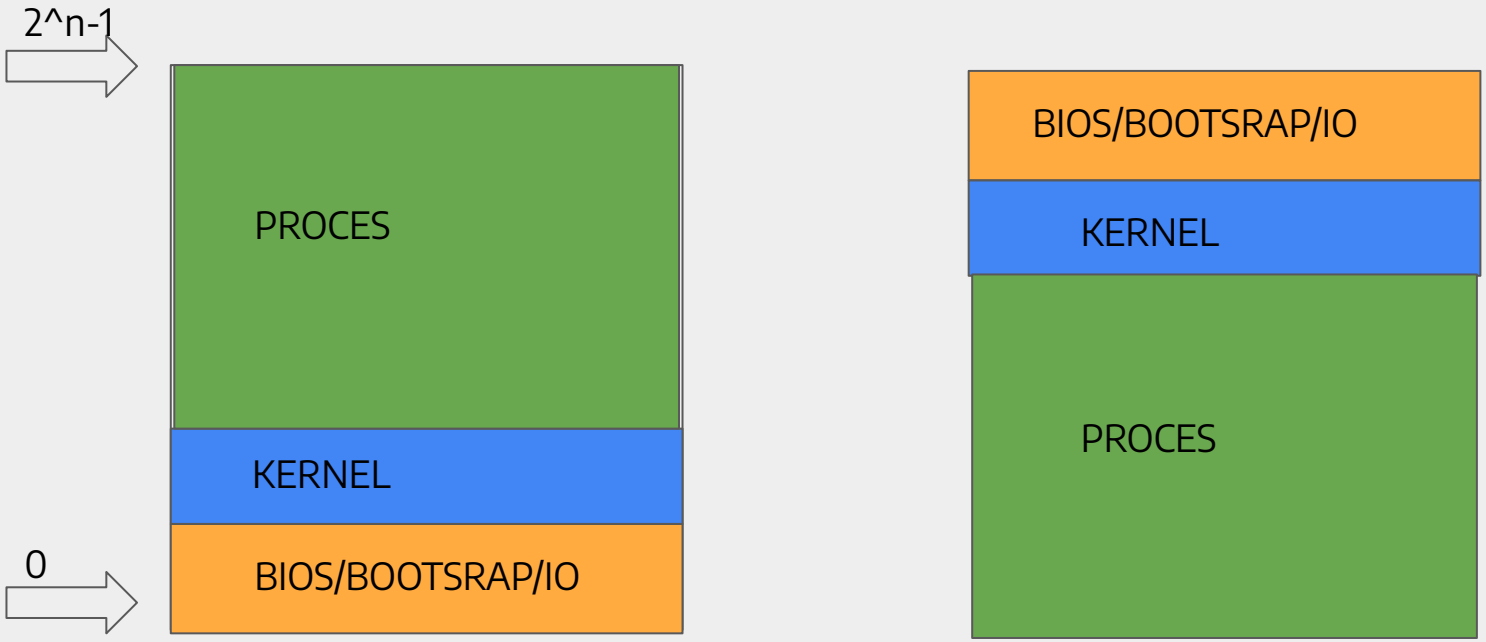
1. Malo o osnovnim stvarima
2. Organizacija memorije
3. Optimizacija prostora
4. Algoritmi za poboljšanje performansi

Organizacija memorije

Ako pogledamo današnje računare, može nam se učiniti kao da oni mogu da izvršavaju na stotine programa u isto vreme. Kao da rade potpuno paralelno. Mi, kao korisnici, nemamo osećaj da procesor šalta s jednog na drugi proces, a opet, znamo da je procesor atomična struktura – on moze izvršavati samo jednu instrukciju u trenutku. To je omogućeno konceptom koji se zove **paralelizacija.**

Monoprocesni sistem

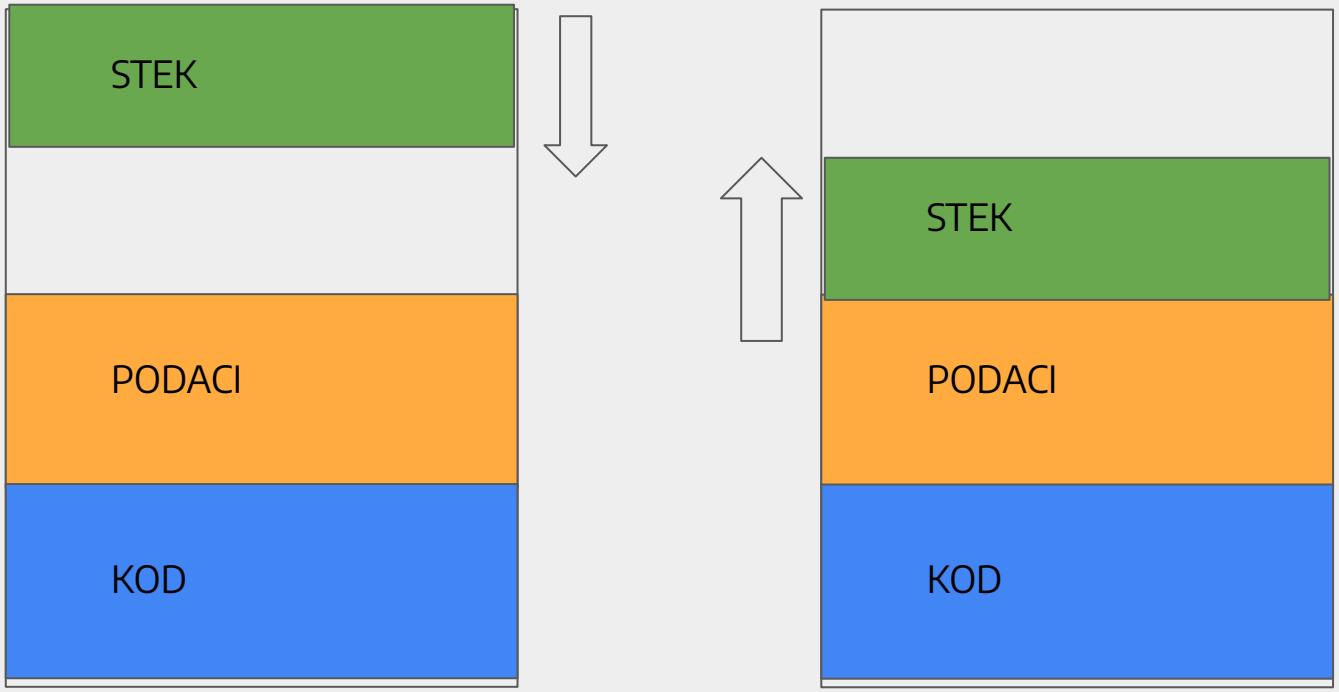
- Zamislimo da imamo samo jedan proces u memoriji.
- Budući da nema nijednog drugog ⇒ Smeštamo ga u celu operativnu memoriju
- Adresa od koje počinje naš proces je uvek **fiksna**, jer je ROM deo i kernel deo uvek iste veličine.



Monoprocesni sistem

- Unutar svog raspoloživog prostora, proces može na proizvoljan način da organizuje logičke segmente

- Deo za kod, deo za dinamički alocirane podatke, deo za statički alocirane podatke, stek...



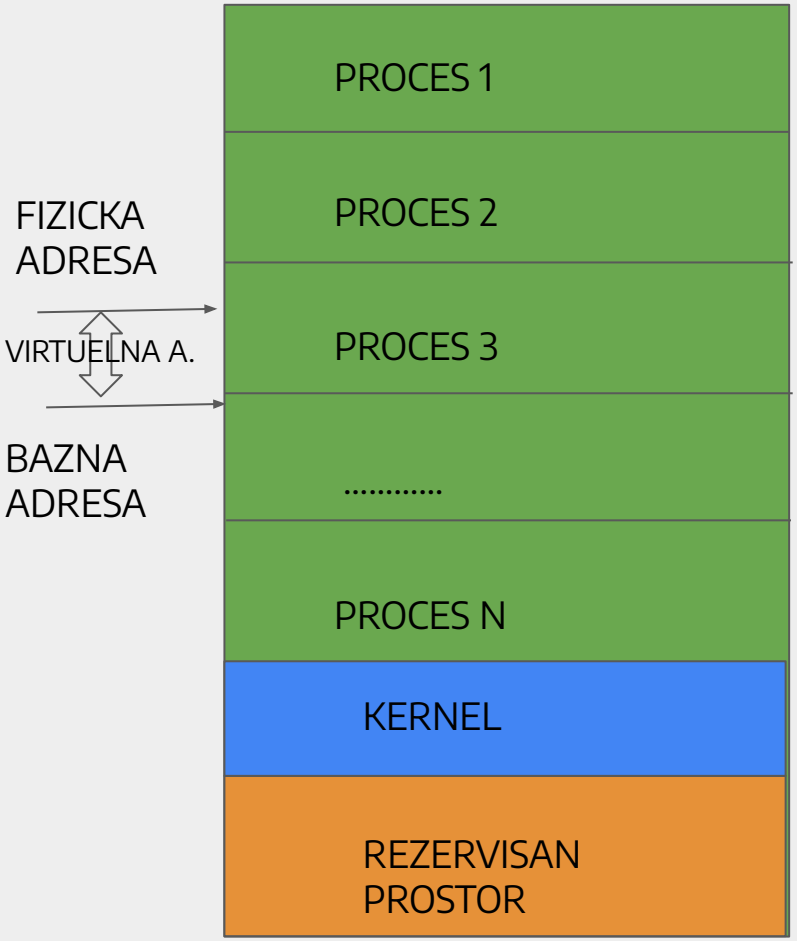
Monoprocesni => multiprocesni

- Primer je bio jednostavan, ali praktično neupotrebljiv.
- U multiprocesnom sistemu, mi imamo N procesa koji se „uporedo“ izvršavaju.
- **Kako ćemo organizovati njihove podele u memoriji?**

Particionisanje

- Intuitivno, mogli bismo operativnu memoriju (tj. njen RAM) da podelimo na N jednakih delova
- Svaki proces dobija jedan chunk memorije - n-ti deo.
- Sada **adresa od koje počinje prostor** dodeljen procesu više **nije poznata unapred**, za vreme prevođenja, pa adresa kojom se adresira fizička memorija nije ista kao adresa koju generiše proces, tj. instrukcije .
- Fizička adresa je sada jednaka **logičkoj + baznoj**
- Skup adresa koje instrukcije mogu da generišu zato čini logički ili **virtuelni adresni prostor,** koji počinje od adrese 0.

Particionisanje



- Preslikavanje virtuelne adrese u fizičku se radi pri svakom adresiranju memorije tokom izvršavanja instrukcije, i za adresiranje instrukcije i za adresiranje podataka.
- Zato mora da postoji podrška hardvera u vidu posebnog dela procesora koji se bavi preslikavanjem logičke u fizičku adresu. Takav deo procesora naziva se jedinica za upravljanje memorijom (engl. memory management unit, **MMU**).

Kontinualna alokacija

- Koji su potencijalni problemi ovakvog smestanja procesa u memoriju?
  - Interna fragmentacija
  - Ponavljanje podatka
  - Ograničenost

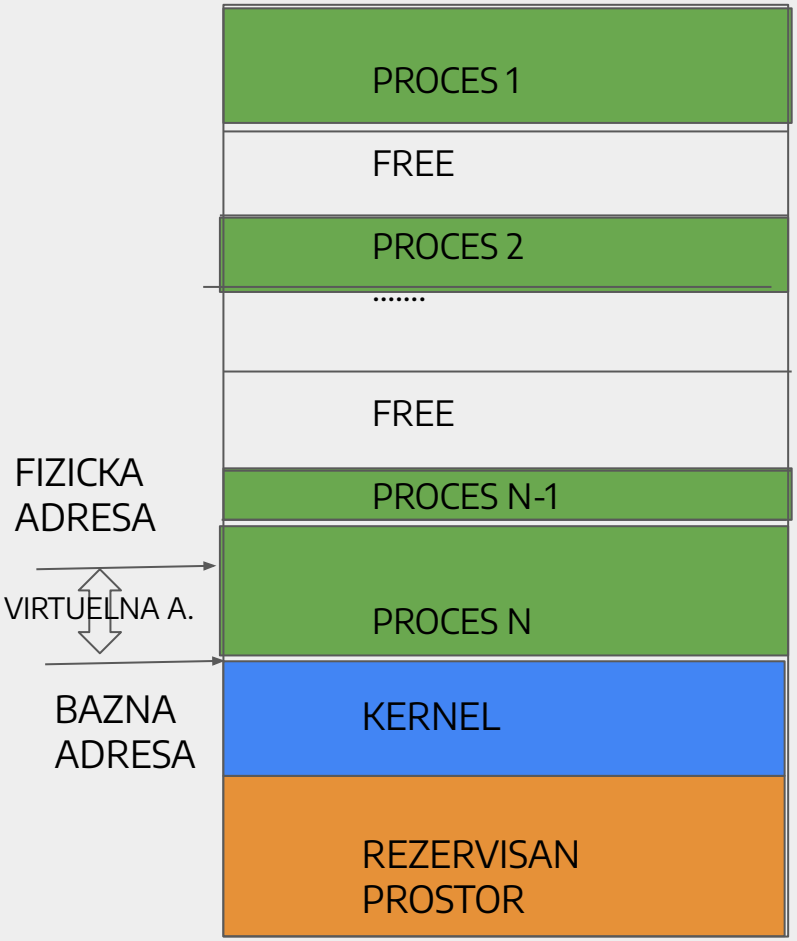
- Bolje rešenje - da **smeštamo procese** u memoriju **kontinualno**.

Kontinualna alokacija

- Smeštanje se zakomplikovalo i sada treba voditi evidenciju o tome gde imamo mesta i koliko imamo mesta za smeštanje novih procesa.

- Takođe, kada su slobodni fragmenti različitih veličina (što je gotovo uvek), pri upisu novog procesa u memoriju moramo izabrati fragment dovoljne veličine, ali koji?
  - Da li onaj koji najviše odgovara po veličini?
  - Ili prvi?

Kontinualna alokacija



- Koje su mane kontinualne alokacije?
  - Eksterni fragmetni
  - Potencijalno 0 uštede - interna fragmentacija

Segmentna alokacija

- Kako ćemo da rešimo problem eksterne fragmentacije?
- Batalimo koncept kontinualnog smeštanja!
- =====> Segmentna organizacija

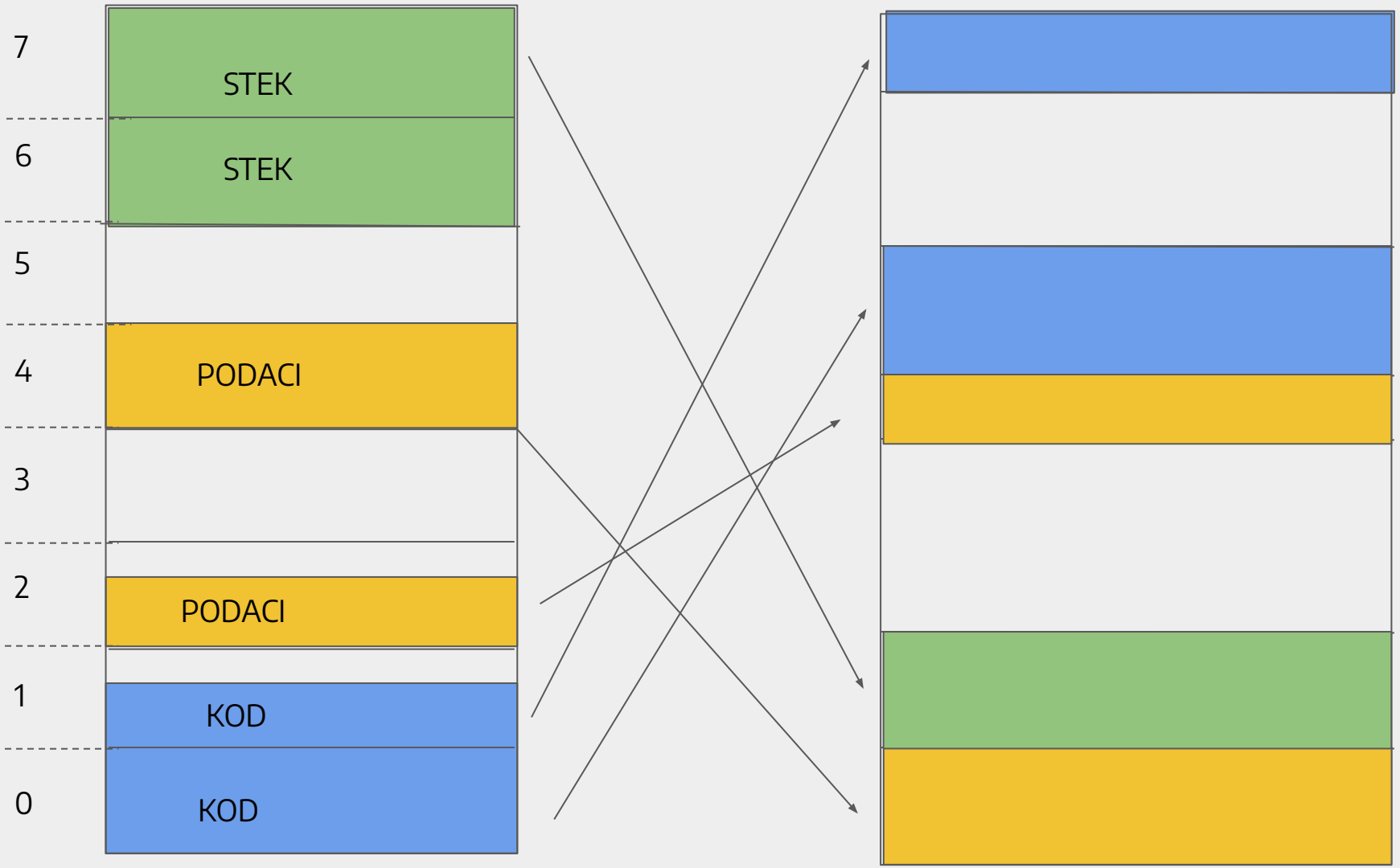
Segmentna alokacija

- Ideja je sledeća : slično kao sa početka, podelićemo proces na segmente, gde svaki segment predstavlja neku logičku celinu.
- Umesto da proces smeštamo kontinualno, sada segmente smeštamo kontinualno, a proces rasparčano.
- Svaki segment ima svoju baznu adresu i granicu.
- Proces mapiranja je složeniji, mora da postoji tabela za svaki proces koja nam govori na kojoj adresi u memoriji počinje koji segment => SMT (segment map table).

Segmentna alokacija

- Sada već naziremo punu moć virtuelne memorije. Proces uopšte ne mora da bude smešten u operativnu memoriju celim svojim logičkim adresnim prostorom, već samo onim delom koji se zapravo koristi.
- Stoga, on može svoj adresni prostor da gleda kao da je veličine cele operativne memorije, pa čak i veći! Za programera je to potpuno transparentno.
- Koje su mane segmentne alokacije?

Segmentna alokacija



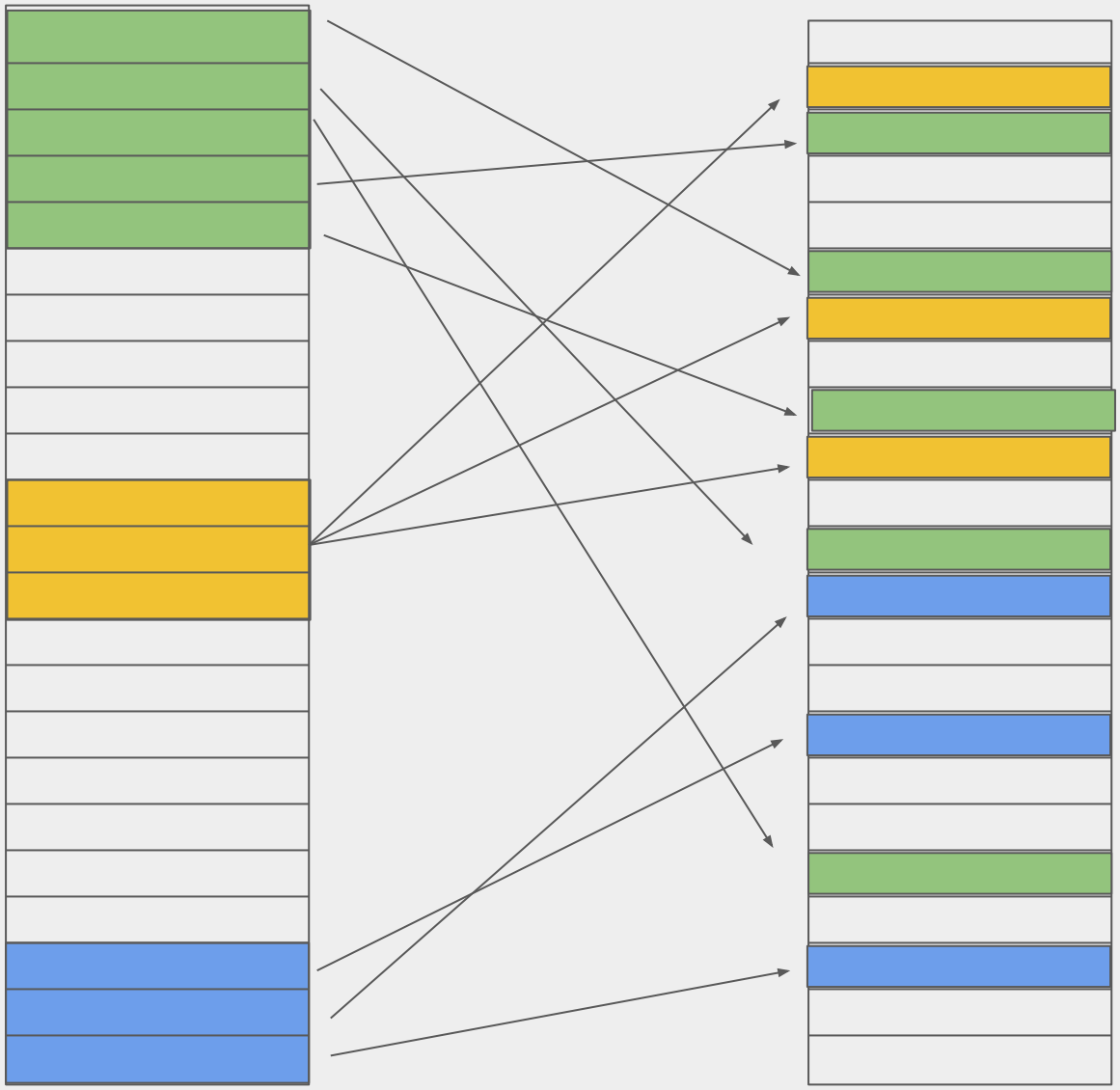
Stranična alokacija

- Prema zaključku iz prethodnog odeljka, ideja o kontinualnoj alokaciji treba da se napusti.
- Stoga dolazimo do konačne ideje koja unapređuje segmentnu a to je stranična organizacija.
- Za pojam logičkog segmenta i segmentnu organizaciju virtuelnog adresnog prostora mogu da znaju samo prevodilac/asembler i operativni sistem, dok koncept segmenta uopšte ne mora da bude podržan i prepoznat od strane hardvera za preslikavanje adresa (MMU-a).

Stranična alokacija

- Kod stranične organizacije, virtuelni adresni prostor procesa podeljen je na stranice iste veličine (uvek stepen dvojke).
- Operativna memorija je podeljena u okvire tj frame-ove, koji su iste veličine kao i stranice
- Svaka stranica se može smestiti u bilo koji okvir
- Slično kao kod segmentne, mora postojati tabela koja nam govori gde se koja stranica slika - PAGE MAP TABLE (PMT)
- Potencijalni problemi?

Stranična alokacija



Agenda

1. Malo o osnovnim stvarima
2. Organizacija memorije
3. Optimizacija prostora
4. Algoritmi za poboljšanje performansi

Optimizacija prostora

- Ne moramo da držimo u memoriji više puta nešto što nam se ponavlja u više procesa. Dovoljno je samo da u PMT tabeli upišemo istu adresu.
- Copy on write
- Dinamičko učitavanje
  - Odgovornost programera
- Preklopi
  - Odgovornost programera
- Zamena procesa
- Alokacija stranica na zahtev
- Zamena stranica

Agenda

1. Malo o osnovnim stvarima
2. Organizacija memorije
3. Optimizacija prostora
4. Algoritmi za poboljšanje performansi

Algoritmi za poboljšanje performansi

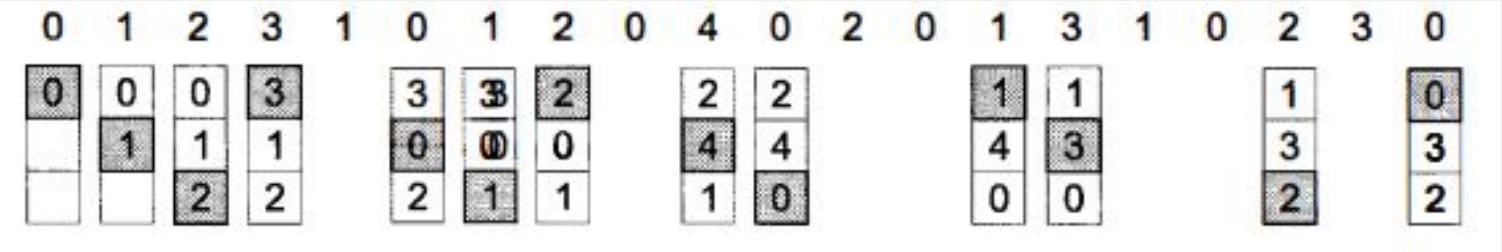
- Mehanizam virtuelne memorije sa zamenom stranica predstavlja vrlo moćan mehanizam savremenih računara koji se vrlo često koristi.
- Izbor stranice koja će se izbaciti ne predstavlja semantički problem za proces - on toga nije ni svestan
- Ali može predstavljati problem za performanse sistema. Svako izbacivanje je jedan režijski trošak.
  - Šta ako izbacimo neku stranicu koja će se uskoro ponovo adresirati?
- Kako da pretpostavimo to?

Algoritmi za poboljšanje performansi

- Da bismo mogli da pratimo performanse algoritama, moramo imati neki kriterijum i način na koji to radimo.
- Prikupljaćemo informacije o tome koje su stranice bile adresirane tokom nekog protoka vremena i informacije o broju page-faultova koje dobijemo.
- Manji broj straničnih grešaka znači bolji algoritam.
- Intuitivno nam se nameće i da veći broj okvira automatski znači manje page fault-ova.
  - NETAČNO! (videćemo na primeru)
- Performanse mogu i da zavise od načina na koji smo strukturirali podatke u našem programu

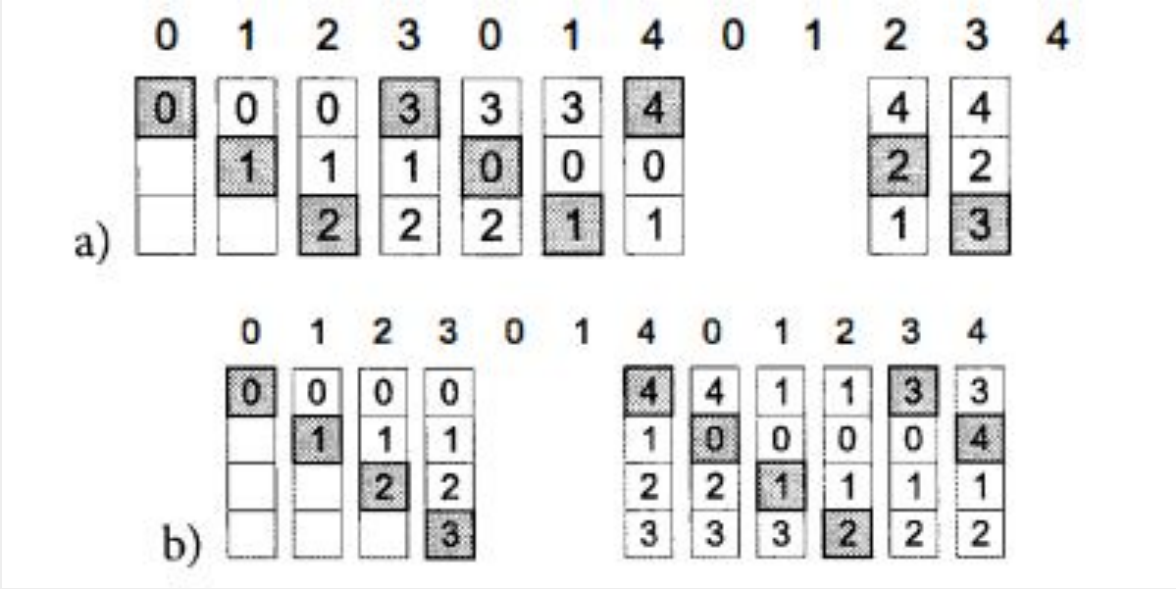
Osnovna ideja - FIFO

- Osnovni algoritam zamene stranica, koji bi verovatno svakom pao na pamet, jeste da se izbaci ona stranica koja je najranije učitana.
- FIFO princip
- Jednostavan za implementaciju
- Koliko straničnih grešaka se generiše?



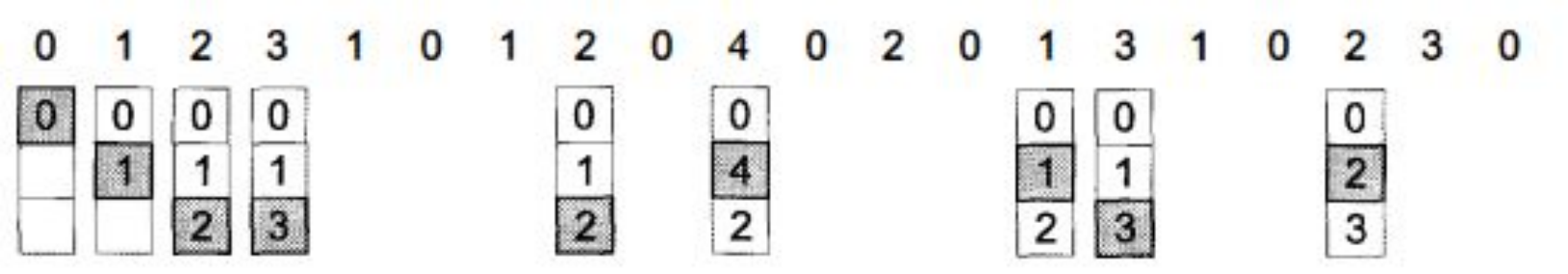
Osnovna ideja - FIFO

- Problem - možemo izbaciti stranicu koja će se uskoro ponovo adresirati
- Još jedan problem - primer



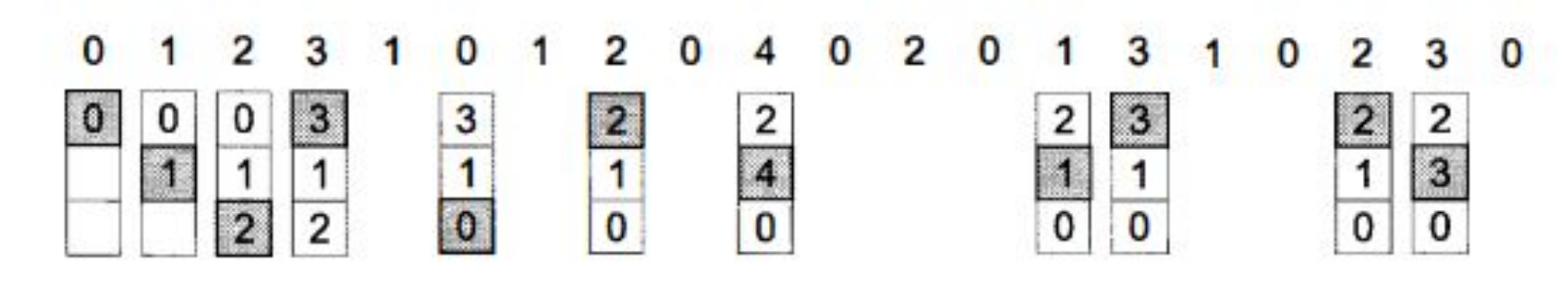
Osnovna ideja - FIFO

- Problem u osnovnoj ideji nas navodi na bolje rešenje
  - Optimalni algoritam zamene - stranica koja će poslednja biti adresirana ponovo u budućem adresiranju
- Teorijski optimalno, ali praktično neizvodljivo.
- Dobra osnova za dalje
  - Ponašanje nekog sistema se ne može unapred znati, ali se može predvideti



LRU

- LRU - least recently used, biramo stranicu koja je najranije korišćena
- LRU se generalno smatra dobrim algoritmom, jer ima dobar učinak koji se u praksi približava učinku optimalnog algoritma
- Implementacija nije toliko jednostavna
  - , neophodna je neposredna podrška hardvera, jer jedino hardver (MMU) zna za svako adresiranje neke stranice



LRU

- Jedna moguća hardverska podrška je:
- Svakoj stranici pridruži se jedna vrednost vremenske marke koja se pamti pored svake stranice u PMT-u koju koristi MMU.
- U MMU postoji neki registar koji služi kao časovnik tj. svako adresiranje bilo čega ga povećava za 1 i dobija funkciju brojača.
- Kada bira stranicu za zamenu, operativni sistem treba da pronađe stranicu sa najmanjom vrednošću ove vremenske marke, što može da bude zahtevna operacija.
  - Koja složenost?

LRU

- Druga moguća harvderska podrška - Hardver čuva dvostruko ulančanu listu stranica po hronološkom poretku, i kada se neka stranica adresira ide na početak.
- Biranje stranica za zamenu je u O(1)
- Hardverski komplikovano
- Vremenski zahtevno

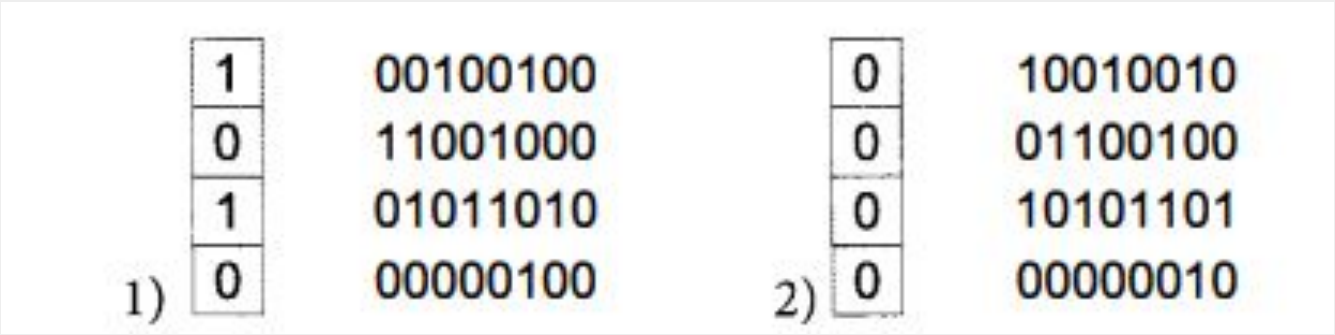
Aproksimacija LRU

- Umesto da hardverski implementiramo, bolje da aproksimiramo.
- Skup stranica u memoriji je prilično velik, pa nije ni bitno da li će se za zamenu izabrati baš ona stranica koja je najranije korišćena; dovoljno je izabrati neku koja je dovoljno davno korišćena.

Aproksimacija LRU

- Hardverska podrška:
- Svakoj stranici pridružimo jedan bit referenciranosti koji MMU postavlja na 1 svaki put kada se data stranica adresira, bilo za čitanje ili za upis.
- Ovaj bit sastavni je deo deskriptora stranice u PMT-u
- Bit se stavlja na 0 kada se stranica prvi put učitava.
- Još bolja ideja je da imamo više bita referenciranosti

Aproksimacija LRU

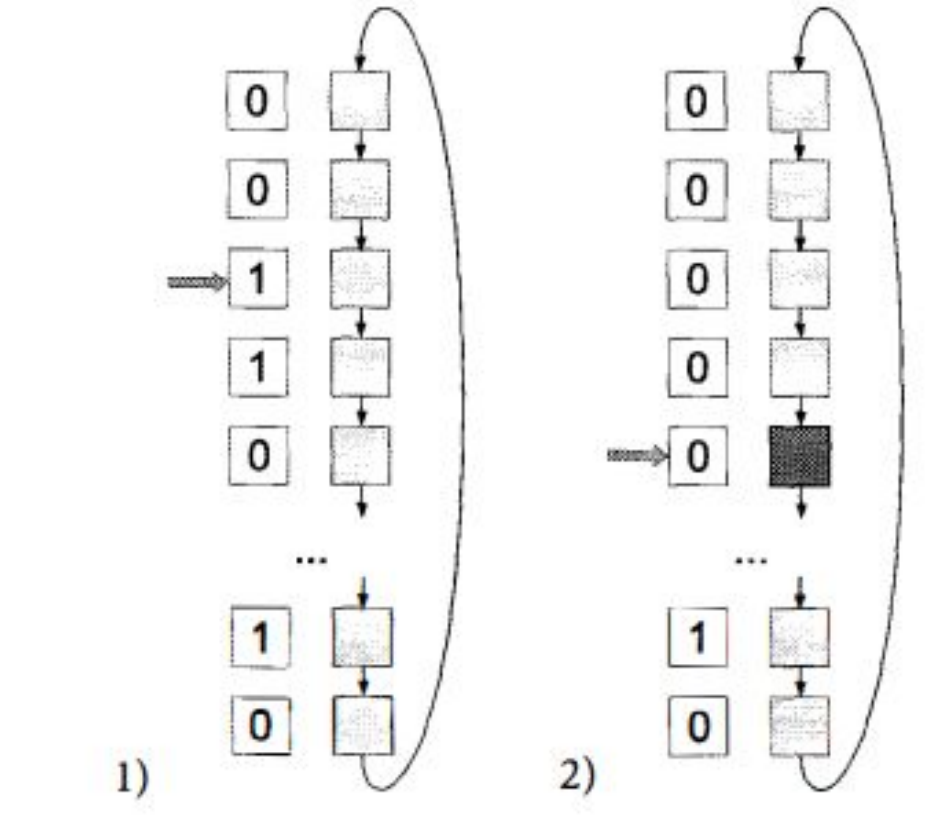


- Operativni sistem periodično obavlja sledeće:
  pomera trenutnu vrednost udesno za jedan bit, krajnji desni (najniži) bit odbacuje, u krajnji levi (najviši) bit upisuje vrednost bita referenciranosti te stranice, a potom bit referenciranosti te stranice obriše (postavi na 0).

Algoritam davanja nove šanse

- Stranice su ovde logički organizovane u kružnu listu, kao kod FIFO algoritma
- „Kazaljka" se kreće u krug po toj strukturi i ukazuje na stranicu koja je prvi sledeći kandidat za izbacivanje.
- Kada operativni sistem bira stranicu za izbacivanje, kandidat je ona stranica na koju trenutno ukazuje kazaljka
- Tada gledamo bit referenciranosti: ako je on postavljen na 1, što znači da je stranica bila korišćena, toj stranici se daje nova šansa tako što se njen bit referenciranosti obriše (postavi na 0)
- Kazaljka ide dalje i razmatra sledeću stranicu

Algoritam davanja nove šanse



Algoritam davanja nove šanse

- Postoji i naprednija varijanta - prošireni algoritam.
- Osim gledanja bita referenciranosti gledamo i bit zaprljanosti
- Bit zaprljanosti - da li je stranica bila promenjena od onda kada je dovučena u memoriju (npr. neka stranica koja čuva podatke)
- Zaprljane stranice moraju da se čuvaju pre nego što se izbace iz memorije, pa je to veći režijski trošak.
- Redosled:
  - 0,0
  - 0,1
  - 1,0
  - 1,1

Zaključak

- Postoji još pregršt načina na koje možemo optimizovati i poboljšati performanse računara
- Ovo su bili neki od osnovnih

Hvala na pažnji!

Pitanja?