

8 VIRTUALNA MEMORIJA

U poglavlju 8 opisani su različite strategije upravljanja memorijom koje se susreću ili su se susretale u računarskim sustavima. Sve navedene strategije imaju jedinstven cilj: imati u memoriji što veći broj procesa kako bi se realizirao što veći stupanj višeprociranja. Također, imaju jedinstven zahtjev da cijeli program koji se izvodi mora biti u radnoj memoriji.

Virtualna memorija je strategija dodjele memoriju koja dozvoljava da samo dio programa koji se izvodi bude u radnoj memoriji. Temeljna prednost ovakvog pristupa je da program može biti i veći od radne memorije. Tako korisnički program može poprimiti proizvoljnu veličinu, a sustav za upravljanje memorijom preslikava logički prostor korisnika u ograničeni prostor u radnoj memoriji. Ovakav sustav za upravljanje memorijom nije jednostavno realizirati. Loša implementacija ovakvog sustava može značajno smanjiti performanse cjelovitog računarskog sustava. U ovom poglavlju razmatrati će se realizacija straničenja na zahtjev (*demand paging*), njena složenost i cijena.

8.1 Uvodna razmatranja

Algoritmi opisani u prethodnom poglavlju potrebni su iz razloga što naredbe koje se izvode moraju biti u glavnoj memoriji. Najjednostavnije je da se cijeli program upiše u glavnu memoriju. Prebacivanje (*overlay*) i dinamičko punjenje rješavaju posljednje ograničenje, ali zahtijevaju dodatne mjere od strane programera.

Analize programa ukazuju da obično nije potreban cijeli program da bi se potrebna obrada izvela. Tako npr.:

- Programi sadrže procedure za obradu slučajnih ili namjernih pogrešaka. Budući da se takvi slučajevi relativno rijetko dešavaju, program izvede potrebnu obradu bez poziva spomenutih procedura.
- Program za polja, liste, tablice, i slične statičke strukture obično rezervira više memorije nego je stvarno potrebno.
- Pojedine opcije programa relativno se rijetko koriste. Tako npr. pravnici kad pišu u Word tekst procesoru vjerojatno nikad neće koristiti Equation editor.

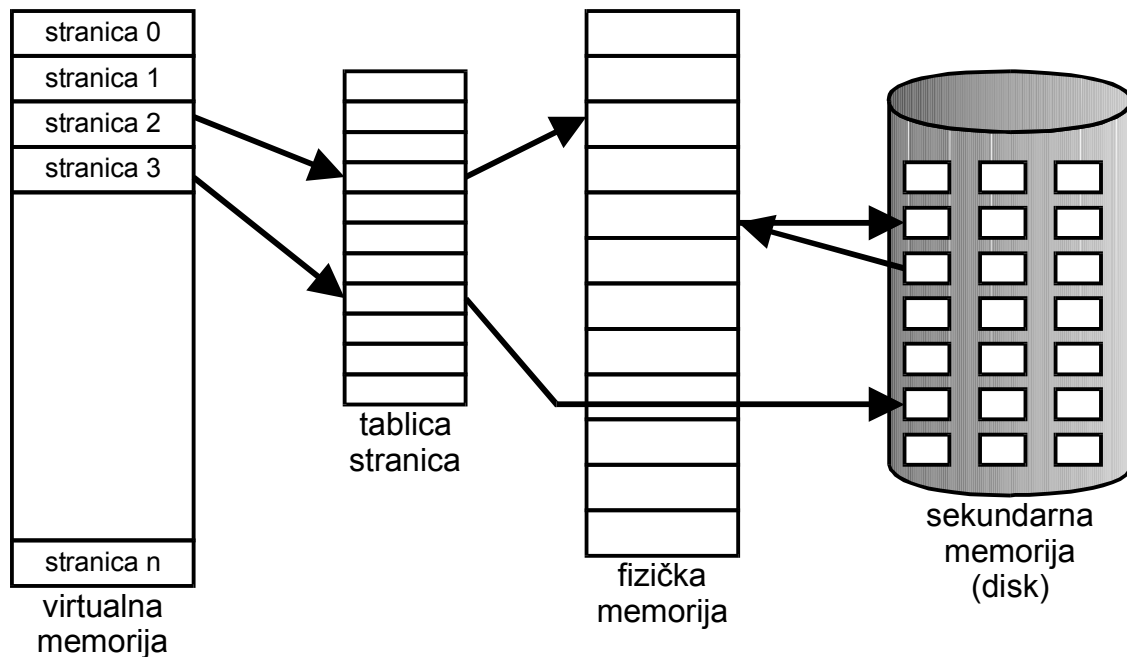
Čak i u slučajevima koji ne spadaju u navedene kategorije, činjenica je da cijeli program nije istovremeno potreban. Tako svojstvo da samo dio programa koji se izvodi se nalazi u memoriji ima niz prednosti:

- Veličina programa nije ograničena veličinom radne memorije. Programer može napisati programa kao da sustav raspolaže s neograničenom memorijom, odnosno programer raspolaže s neograničenim virtualnim logičkim adresnim prostorom. Ovo uveliko pomaže kod pisanja programa.
- Korisnički program može se izvoditi sa znatno manjom dodijeljenom fizičkom memorijom, što omogućava veći stupanja višeprogramskog rada. Time se povećava iskoristivost kao i propusna moć sustava.
- Manje U/I operacija potrebno je za prebacivanje korisničkih programa iz i u memoriju. Tako se korisnički programi brže izvode.

Zaključaj je da izvođenje programa koji nisu cijeli upisani u memoriji ima brojne prednosti i za korisnika i za računalni sustav.

Virtualna memorija je razdvajanje logičkog adresnog prostora koji vidi korisnik od fizičkog adresnog prostora u kojem se program izvodi. Ovo razdvajanje omogućava

programeru da raspolaže s neograničenim logičkim prostorom iako se program stvarno izvodi u relativno malom fizičkom adresnom prostoru. Virtualna memorija olakšava posao programeru, ne samo što raspolaže s neograničenim logičkim prostorom, nego što ne treba da vodi računa o strukturi programa koju je zahtijevao sustav s prebacivanjem ili dinamičkim punjenjem. Princip virtualne memorije prikazan je slikom 9.1.



Slika 9.1. Prikaz virtualne memorije koja je veća od fizičke memorije.

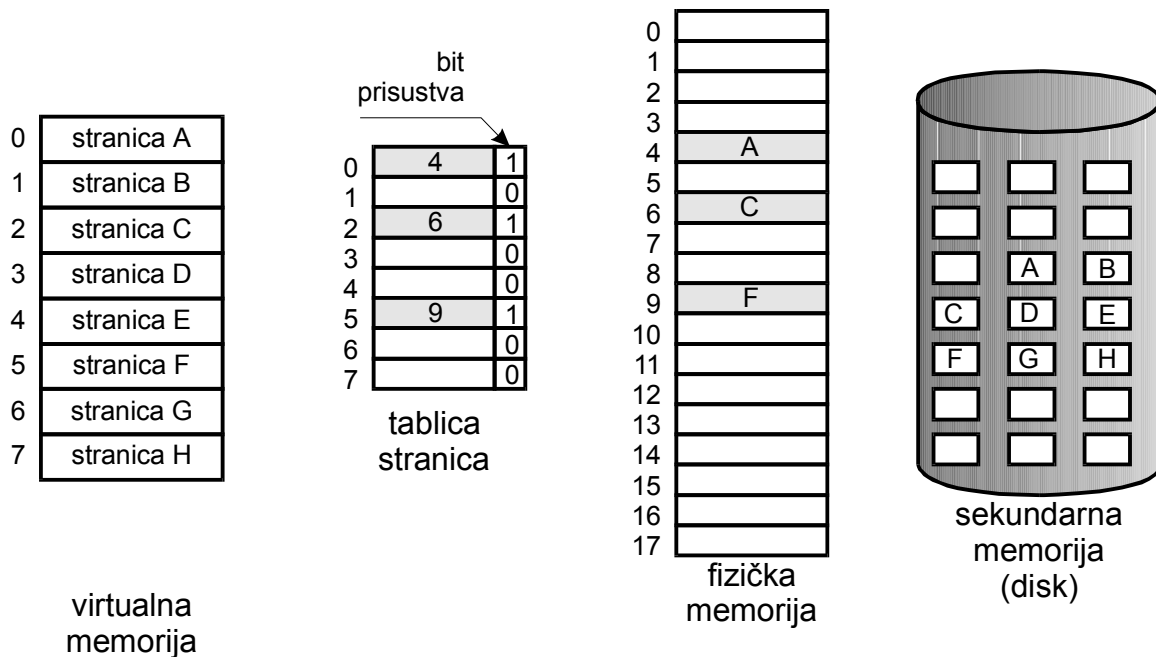
Virtualna memorija obično se realizira kao *straničenje na zahtjev (demand paging)*. Moguće ju je primijeniti u sustavima koji koriste podjelu memorije na segmente. Nekoliko sustava riješilo je virtualnu memoriju pomoću segmenata, gdje su segmenti podijeljeni na stranice. Tako korisnik vidi program podijeljen na segmente, a operacijski sustav dijeli segmente na stranice. Tako je IBM OS/2 koristio koncept segmentacije na zahtjev. Važno je napomenuti da algoritmi zamjene segmenata su znatno složeniji od algoritama zamjene stranica budući su segmenti promjenjive veličine, a stranice fiksne.

8.2 Straničenje na zahtjev (*demand paging*)

Straničenje na zahtjev koristi koncept sličan prebacivanju. Proces je pohranjen na sekundarnoj memoriji, obično disku. Kada se namjerava izvesti proces upisuje se samo jedan njegov dio u radnu memoriju. Kod upisivanja procesa u memoriju uobičajeno se koristi tzv. lijeni prebacivač (*lazy swapper*) koji upisuje stranicu u memoriju tek kada je ona potrebna. Termin *swapper* može se smatrati netočnim jer se on obično odnosi na prebacivanje cijelog procesa. Ispravnije je korištenje termina *pegger* koji se odnosi na prebacivanje stranice. Proces se sada može promatrati kao niz stranica koji se prema potrebi upisuju u memoriju.

Ovakav pristup dodjeli memorije zahtijeva i određenu sklopovsku podršku. Prvenstveno potrebno je razlučiti koje su stranice upisane u memoriju, a koje se samo nalaze na

disku. Ovaj problem rješava se proširenjem tablice stranica bitom prisutnosti koji daje informaciju da li se stranica nalazi u radnoj memoriji ili ne, slika 9.2.

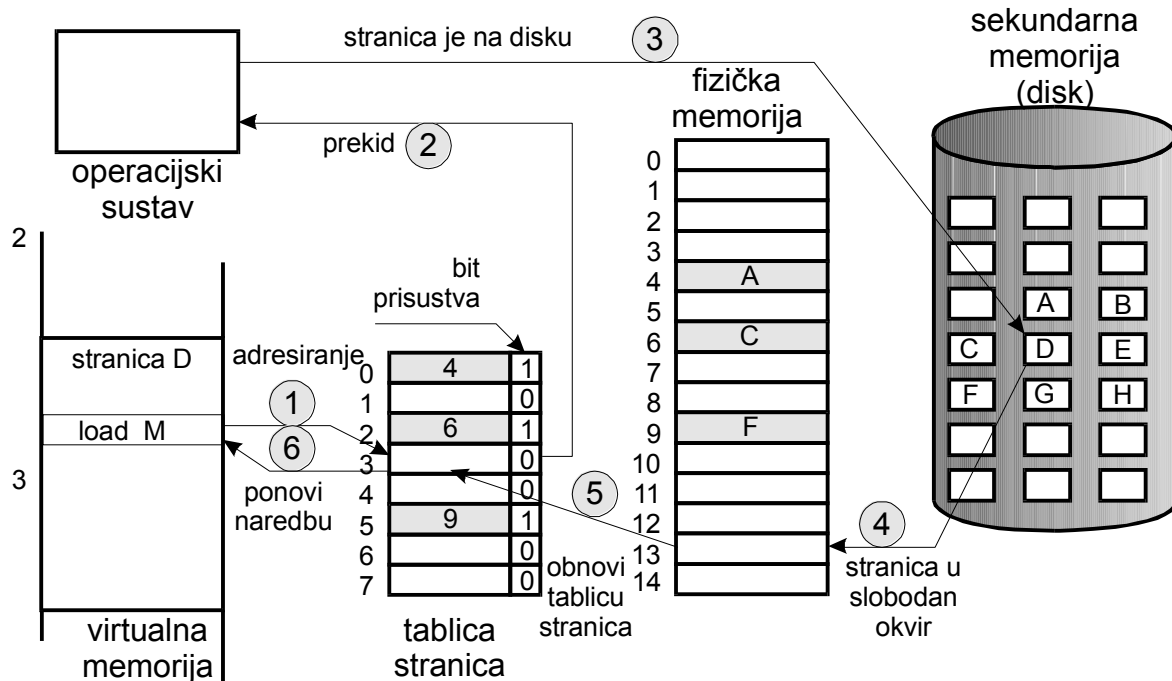


Slika 9.2. Tablica stranica kad sve stranice nisu u radnoj memoriji.

Kada se pristupa pojedinoj stranici, operacijski sustav preko tablice stranica ispituje da li je adresirana stranica u memoriji ili ne. Ako je stranica u upisana u fizičku memoriju, bit prisustva je postavljen u jedinicu, tada se izračunava fizička adresa naredbe ili podatka i pristupa mu se. Ali ako adresirana stranica nije u memoriji tada ju je potrebno upisati u memoriju i tek onda izvršiti pristup. Procedura pristupa stranici može se opisati na sljedeći način:

- ❑ sklopovlje prvo provjerava bit prisustva adresirane stranice kako bi se odredilo da li je stranica u memoriji ili ne.
- ❑ Ukoliko stranica nije u memoriji (došlo je do tzv. promašaja) generira se prekid koji dojavljuje operacijskom sustavu da treba pronaći stranicu na sekundarnoj memoriji i prebaciti 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 disk.
- ❑ Operacijski sustav pronalazi slobodan okvir u radnoj memoriji (operacijski sustav vodi listu slobodnih okvira).
- ❑ Prebacuje se tražena stranica u odabrani okvir.
- ❑ Osvježava se tablica stranica procesa na način da se stranici pridružuje dodijeljeni okvir. Ovim je praktički proces pripravan da nastavi s izvođenjem.
- ❑ Prekinuta naredba se ponovo izvodi a stranici se pristupa kao da je ona oduvijek bila u memoriji.

Opisani proces prikazan je slikom 9.3.



Slika 9.3. Koraci u slučaju promašaja stranice.

U cijelom ovom procesu važno je primijetiti da se nakon prekida sačuva stanje procesa (programsko brojilo i ostali spremnici opće namjene) te da se nakon unosa stranice u radnu memoriju te osvježavanja tablice stanja proces može nesmetano nastaviti. Jedina razlika je u tome što je sada adresirana stranica u radnoj memoriji.

Teoretski postoji vjerojatnost da tijekom izvođenja jedne naredbe dođe do višestrukih promašaja. Naime, program se može nalaziti na jednoj stranici, a podaci na drugoj. Ovakve situacije, koje na sreću nisu učestale, dovele bi do značajnog pada performansi sustava.

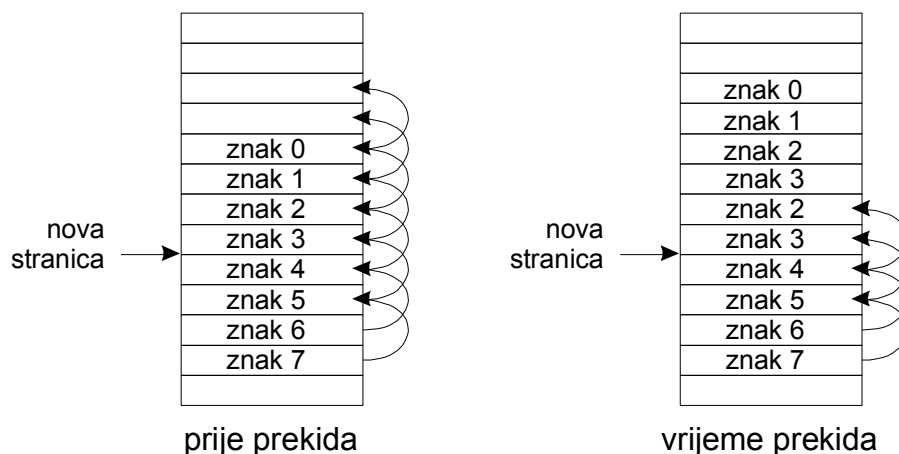
Kao i kod dodjele memorije po stranicama, virtualna memorija zahtjeva značajnu sklopovsku podršku kako bi se održale dobre performanse sustava. Uz sklopovlje potrebna je i dodatna programska podrška. Također, za implementaciju ovakvog sustava dodjele memorije potrebne su i određene preinake arhitekture računala. Temeljni problem je potreba za ponavljanjem naredbe ukoliko dođe do promašaja. Do promašaja može doći u bilo kojoj fazi izvođenja naredbe. Ukoliko do promašaja dođe kod dohvata naredbe jednostavno se nakon unosa stranice u memoriju ponavlja njen dohvat te nastavlja s njenim izvođenjem. Ali ukoliko dođe do promašaja tijekom dohvata operanda tada je potrebno ponovo dohvatiti naredbu, dekodirati je te dohvatiti operand koji je sada u radnoj memoriji.

Neka se izvodi naredba koja zbraja dvije memorijske varijable A i B i rezultat pohranjuje na treću memorijsku lokaciju C. Naredba se izvodi kroz sljedeće korake:

1. dohvati i dekodiraj naredbu,
2. dohvati operand A,
3. dohvati operand B
4. zbroji A i B,
5. pohrani rezultat u C.

Neka se rezultat C nalazi na lokaciji koja pripada stranici koja trenutno nije u memoriji. Promašaj tada rezultira procedurom unosa stranice u memoriju, te ponavljanja koraka 1-5. Program normalno nastavlja s izvođenjem, a ovakve situacije ako nisu učestale neće značajno utjecati na cjelokupne performanse sustava.

Problem može nastupiti ukoliko procesor raspolaže s naredbama koje mijenjaju istovremeno sadržaje više memorijskih lokacija. Tako je npr. IBM 370, a i Intel 80x86 ima naredbe za prebacivanje niza karaktera s jedne na drugu lokaciju. Poseban problem nastaje ukoliko izvorni i odredišni niz dijele neke lokacije, slika 9.4. Ako je promašaj nastupio nakon što je izmijenjen dio izvornog niza tada nije moguće ponoviti započetu a prekinutu naredbu.



Slika 9.4. Prebacivanje niza znakova s jedne na drugu memorijsku lokaciju.

Navedeni problem moguće je riješiti na dva načina. Prvi je da se prije prebacivanja ispituju početne i krajnje adrese oba niza te da se prema potrebi odmah unesu potrebne stranice u memoriju. Drugi pristup bila bi primjena privremenih spremnika u koju bi se pohranile vrijednosti izmijenjenih lokacija. U slučaju obnovili bi se sadržaji izmijenjenih lokacija i ponovila bi se naredba.

Sličan problem susreće se kod procesora koji koriste posebne modove adresiranja kao npr. autoinkrement i autodekrement (npr. PDP 11 ili Motorola 68000). Ovi adresni modovi koriste sadržaj spremnika kao pokazivač na podatak kojem se pristupa, pri čemu se automatski sadržaj spremnika inkrementira ili dekrementira. Autodekrement umanjuje vrijednost pokazivača (sadržaj spremnika) prije pristupa operandu, dok autoinkrement povećava vrijednost pokazivača (sadržaj spremnika) nakon pristupa operandu. Tako naredba

```
MOV (R2)+, -(R3)
```

kopira sadržaj memorijske lokacije na koju pokazuje spremnik R2 na memorijsku lokaciju na koju pokazuje spremnik R3, s time da se sadržaj spremnika R2 povećava za 2 nakon upisa vrijednosti u memoriju, a sadržaj spremnika R3 umanjuje se za 2 prije dohvata operanda. Ukoliko operand na koji pokazuje umanjena vrijednost spremnika R3 nije u memoriji (promašaj) tada nastupa problem pri ponovljenom izvođenju naredbe. Moguće rješenje je uvođenje zasebnog privremenog spremnika koji pamti prethodni

sadržaj spremnika tako da je pri ponavljanju naredbe moguće obnoviti prvobitnu vrijednost spremnika.

Svakako navedeni problemi postavljaju posebne zahtjeve na arhitekturu procesora koji koristi straničenje na zahtjev. Sklopovi koji upravljaju memorijom nalaze se između procesora i memorije i moraju biti potpuno transparentni korisniku. Zato ovakav sustav za upravljanje memorijom nije moguće realizirati kod svih procesora.

8.3 Performanse sustava sa straničenjem na zahtjev

Straničenje na zahtjev ima značajan utjecaj na performanse računarskog sustava. Bolji uvid na utjecaj ovakve strategije dodjele memorije moguće je procijeniti pomoću efektivnog vremena pristupa memoriji (*effective access time*). Ukoliko se stranica nalazi u memoriji efektivno vrijeme pristupa jednako je vremenu pristupa memoriji ma . Ali ukoliko stranica nije u memoriji potrebno je obraditi prekid, unijeti stranicu u memoriju i ponoviti naredbu što predstavlja znatno duže vrijeme pristupa tf . Neka je vjerojatnost promašaja p ($0 \leq p \leq 1$). Efektivno vrijeme pristupa iznosi:

$$eat = (1 - p) \cdot ma + p \cdot tf.$$

Vrijeme pristupa memoriji kod današnjih procesora je reda ns, npr. 10ns. Za procjenu vremena pristupa memoriji u slučaju promašaja potrebno je sagledati sve korake koje je potrebno napraviti a to su:

1. postaviti zahtjev za prekid procesoru,
2. sačuvati stanje prekinutog procesa,
3. odrediti da se prekid odnosi na promašaj stranice,
4. provjeriti da li je dozvoljen procesu pristup toj stranici i pronaći stranicu na disku,
5. postaviti zahtjev za prijenos stranice u slobodan okvir što zahtijeva:
 - a) čekanje u redu dok se prihvati zahtjev za U/I prijenosom,
 - b) čekanje na pronalaženje traženog sektora na disku (vrijeme traženja ili latentnost),
 - c) prijenos stranice s diska u slobodan okvir,
6. za vrijeme čekanja na obradu postavljenog U/I zahtjeva dodijeliti procesor nekom drugom procesu,
7. po završetku prijena s diska dojaviti operacijskom sustavu da je prijenos obavljen (postavlja se novi zahtjev za prekidom),
8. sačuvati stanje prekinutog procesa,
9. odrediti o kojoj vrsti prekida se radi (U/I prijenos s diska završio),
10. unijeti promjene u tablicu stranica procesa koji je ima promašaj,
11. čekaj dok proces dobije ponovo pravo na korištenje procesora,
12. obnovi stanje procesa i ponovi prekinutu naredbu.

Naravno, ne moraju se dogoditi svi koraci u svakom slučaju promašaja. Npr. korak broj 6 podrazumijeva da se procesor dodjeljuje drugom procesu kako bi se održala što veća zaposlenost procesora, ali ona rezultira u dodatnom vremenu potrebnom za obradu promašaja.

U svakom slučaju za procjenu vremena obrade promašaja moraju se uzeti u obzir sljedeće tri radnje:

1. obrada prekida zbog promašaja,
2. unos stranice u memoriju,
3. nastavak prekinutog procesa.

Prvi i teći zadatak mogu se pažljivim kodiranjem sistemskih procedura ograničiti na nekoliko stotina naredbi, što se može procijeniti na vrijeme od nekoliko mikrosekunda. Drugi zadatak vremenski je znatno zahtjevniji, jer je potrebno pronaći stranicu na disku, postaviti glavu diska na traženi sektor te prebaciti stranicu u memoriju. Ovo vrijeme kod današnjih diskova može se procijeniti na deset do dvadeset milisekunda. Prema tome kod promašaja praktički najveći dio vremena troši se na unos stranice u memoriju. Kod višeprocenog rada situacija je nepovoljnija jer ovom vremenu potrebno je dodati prosječno čekanje u različitim redovima.

Za ilustraciju uzeti će se vrijeme pristupa memoriji 10ns, a vrijeme obrade promašaja 25ms. Efektivno vrijeme promašaja iznosi:

$$e_{\text{at}} = (1 - p) \cdot 10 + p \cdot 25\,000\,000 = 10 + 24\,999\,990 \cdot p$$

Neka je zahtjev da efektivno vrijeme pristupa može biti samo 10% veće od vremena pristupa memoriji dobiva se dozvoljena vjerojatnost promašaja $p < 4 \cdot 10^{-8}$, odnosno da je dozvoljen samo jedan promašaj na 25 milijuna pristupa. Ovaj grubi proračun govori da je važno održati vjerojatnost promašaja jako malom kao se ne bi značajno degradirale performanse cjelovitog sustava.

8.4 Problem zamjene stranica

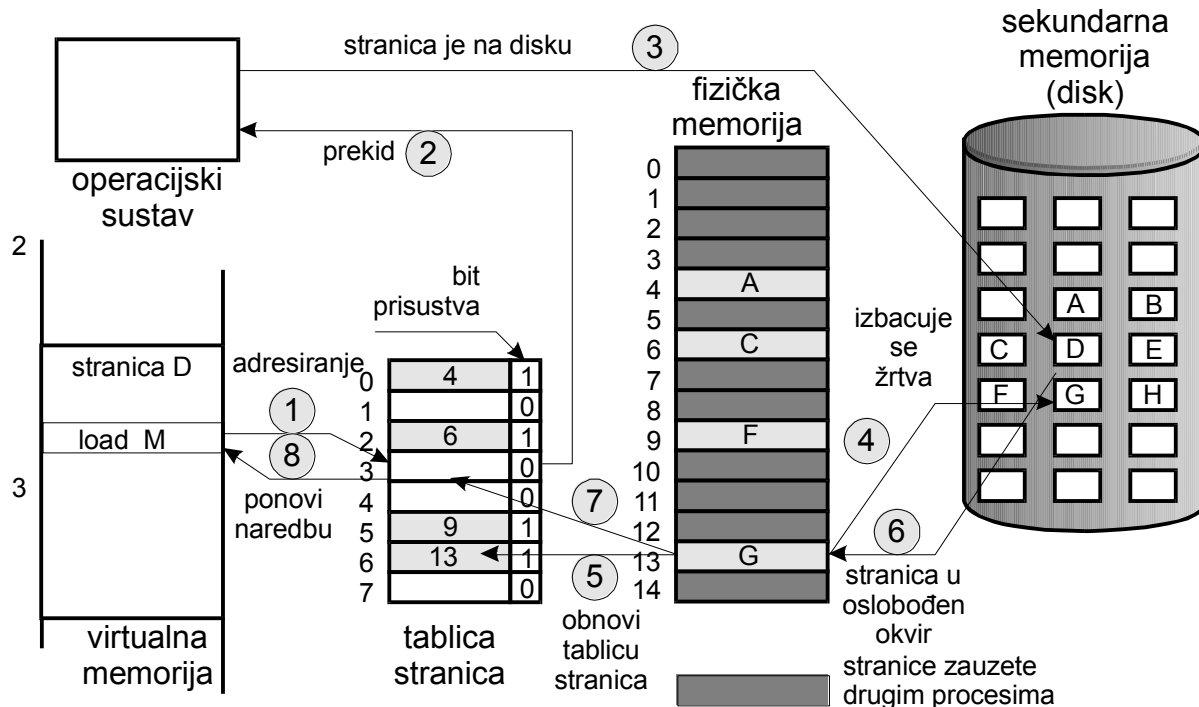
U do sada provedenim razmatranjima pretpostavilo se da u slučaju promašaja operacijski sustav uvijek ima u memoriji slobodan okvir u koji će upisati traženu stranicu. Naravno u praksi situacija je sasvim drugačija. Tijekom rada, a posebice u višeprocenom sustavu, memorija se sasvim popuni te je u slučaju promašaja potrebno osloboditi okvir za novu stranicu izbacivanjem neke od stranica koje su već u memoriji. Kako će biti naknadno opisano, odabir stranice koju će se izbaciti, tzv. žrtvu (*victim*) posebno je osjetljivo pitanje i strategija zamjene stranica može značajno utjecati na performanse sustava.

Prvo, načelno se može proširiti procedura obrade promašaja na sljedeći način (slika 9.5):

1. pronalaženje tražene stranice na disku,
2. odabir okvira u koji će se stranica upisati:
 - a) ako postoji slobodan okvir njega se koristi,
 - b) ako nema slobodnih okvira poseban algoritam odabire žrtvu i prebacuje prema potrebi odabranu stranicu na disk, ažurira tablicu stranica i oslobađa okvir,
3. upisuje traženu stranicu u oslobođeni okvir i ažurira tablicu stranica,
4. ponavlja prekinutu naredbu.

Izbacivanje stranice iz okvira zahtijeva dodatni prijenos na disk što dvostruko produžava vrijeme obrade promašaja, a time i efektivno vrijeme pristupa memoriji.

Pogodnost je što nije uvijek nužno žrtvu prebaciti natrag na disk. Ukoliko sadržaj stranice nije mijenjan, npr. odnosi se na program ili ulazne podatke, tada je sadržaj stranice na disku i u memoriji isti te nije potrebno vraćati stranicu na disk. Dovoljno je samo sadržaj okvira prepisati sadržajem nove stranice. Informaciju o tome da li se što mijenjalo u sadržaju stranice moguće je dobiti uvođenjem zasebnog bita, bita izmjene (*modify bit, dirty bit*), koji se postavlja u jedinicu ukoliko je išta mijenjano u sadržaju stranice. Algoritam za zamjenu stranica prvo provjerava bitove izmjene stranica koje bi se mogle izbaciti i odabire one koje nisu mijenjane, odnosno one kojima je bit izmjene nula.

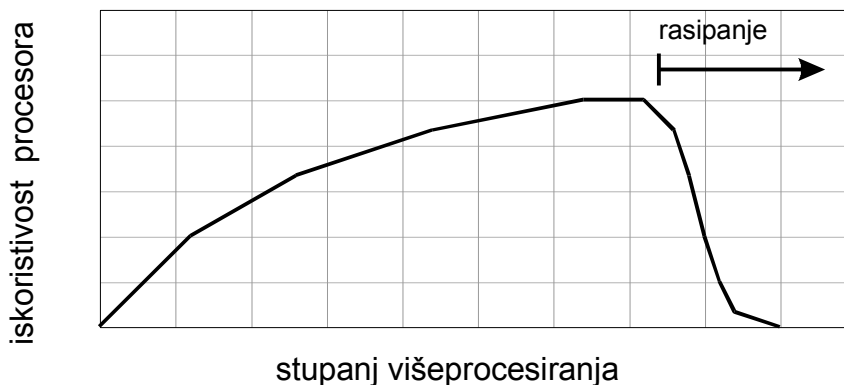


Slika 9.5. Koraci u slučaju promašaja stranice kada nema slobodnih okvira.

Sljedeći problem koji je potrebno riješiti je izbor algoritma odabira stranice žrtve koju će se izbaciti iz memorije.

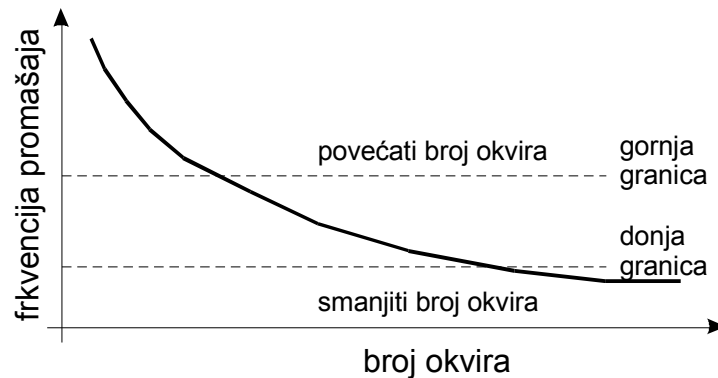
8.5 Rasipanje (*Trashing*)

Kako je već spomenuto, strategija globalne zamjene stranica može dovesti do situacije da pojedini procesi imaju na raspolaganju neprihvatljivo mali broj okvira što rezultira učestalim promašajima, a time i smanjenim performansama sustava. Ovakva situacija u kojoj veći broj procesa se dovede u situaciju da ima na raspolaganju mali broj okvira naziva se rasipanje (*trashing*). Zbog učestalih promašaja sustav veliki dio vremena provodi u izmjeni stranica, a manji je posvećen obradi. Slika 9.9 prikazuje ovisnost iskoristivosti procesora o stupnju višeproceniranja u sastavu koji koristi dodjelu stranica na zahtjev.



Slika 9.9. Ovisnost iskoristivosti procesora o stupnju višeproceniranja.

Očito je da operacijski sustav koji koristi straničenje na zahtjev, te globalnu strategiju zamjene mora imati mehanizam detekcije rasipanja. Najjednostavnije je detektirati povećanu frekvenciju promašaja, a problem otkloniti reduciranjem stupnja višeprociranja izbacivanjem nekih procesa iz obrade, slika 9.10.



Slika 9.10. Ovisnost frekvencije promašaja o broju okvira.

8.6 Dodatna razmatranja

Odabir algoritma i strategije zamjene stranica važna su pitanja koje je potrebno riješiti u sustavima koji koriste straničenje na zahtjev. Uz ova temeljna pitanja nameću se i dodatna rješenja koja mogu poboljšati rad sustava.

8.6.1 Predunos stranica u memoriju (*prepaging*)

Kod startanja procesa, odnosno njegovog prvog unosa u memoriju prema dosadašnjim strategijama broj promašaja je jako velik. Logično rješenje je prilikom prvog unosa procesa u memoriju unijeti veći broj stranica, a ne samo onu koju prvu proces zahtjeva.

Također, ukoliko je u nekom trenutku potrebno suspendirati proces iz radne memorije (npr. zbog rasipanja) zgodno je da operacijski sustav zapamti koje je stranice proces koristio u trenutku gubljenja prava natjecanja za dodjelu procesora kako bi se u trenutku kada se ponovo ostvare uvjeti za nastavak njegovog izvođenja to izvođenje učinkovito nastavilo. U trenutku ponovnog nastavka izvođenja unesu se sve stranice koje je proces koristio prije nego je suspendiran kako bi se izbjeglo nepotrebno prekidanje zbog promašaja.

Predunos ima prednost utoliko što u pojedinim fazama izvođenja procesa umanjuje broj promašaja.

8.6.2 Veličina stranica

Veličina stranica ili okvira je veličina na koju projektanti operacijskog sustava rijetko imaju utjecaj. Na nju je moguće jedinu utjecati za vrijeme projektiranja novih sustava. U tom slučaju potrebno je analizirati utjecaj veličine stranice na performanse sustava.

S gledišta tablice stranica bolja je veća veličina stranice jer je za isti proces potrebna manja tablica stranica. S druge strane manja veličina stranice rezultira boljom iskoristivosti memorije. Naime, kako je već opisano kod sustava koji koriste dodjelu memorije po stranicama postoji samo unutarnja segmentacija memorije koja se

prosječno može procijeniti na polovicu veličine stranice. Tako je prosječno pola veličine stranice po procesu neiskorištena memorija.

Sljedeći čimbenik je brzina unosa stranice u radnu sa sekundarne memorije. Ovo vrijeme sačinjava vrijeme postavljanja glave iznad traženog cilindra (*seek time*), vrijeme pronalaženja sektora, odnosno zakreta diska (*latency*), te vrijeme prebacivanja podatka (*transfer time*). Prva dva vremena neovisna su o veličini stranice i ovise samo o mehaničkim svojstvima diska. Tako npr. vrijeme postavljanja iznosi oko 8 ms, a vrijeme postavljanja oko 10ms. Vrijeme prijenosa ovisi o veličini stranice. Tako za brzinu prijenosa od 2 Okteta po sekundi je potrebno svega 0.5ms za prijenos stranice od 1024 okteta. Tako je za veće stranice potrebno veće vrijeme prijenosa. Površno gledajući lako se zaključi da je s aspekta brzine unosa stranice u memoriju povoljnije da je stranica manja. S druge strane mora se uzeti i vrijeme unosa preostalim stranica. Tada značajan utjecaj ima vrijeme postavljanja i pristupa. S tog stajališta može se zaključiti da je bolje što su stranice veće jer se u tom slučaju prenosi manji broj stranica. Tako npr. za unos 2k programa u memoriju kad su stranice svega 512 okteta potrebno je:

$$T_1 = 4 \cdot (8+10) + 1 = 73 \text{ms} ,$$

dok za 2k stranice:

$$T_2 = 1 \cdot (8+10) + 1 = 19 \text{ms} .$$

S druge strane, ukoliko je veličina stranice manja, u memorijski prostor dodijeljen programu moguće je upisati veći postotak programa koji se najviše koristi. Praktički, s manjim stranicama postiže se veća rezolucija pri izboru dijelova programa koji se više koriste.

Danas veličine stranica variraju između 512 okteta i 4k okteta. Tako npr. Intel 80386 ima veličinu stranice od 4k okteta, Motorola 68030 dozvoljava promjenjivu veličinu stranice od 256 okteta do 32 okteta. Povijesno, kako su se razvijali procesori te je memorija postajala jeftinija, veće brzine i kapaciteta, uz činjenicu da se brzine pristupa disku ne povećavaju proporcionalno njima, osjeća se trend prema većim stranicama.

8.6.3 Struktura programa

Straničenje na zahtjev trebalo bi biti transparentno za korisnika. U većini slučajeva korisniku je svejedno kako piše program i kako se program upisuje u memoriju. Ipak postoje slučajevi u kojima poznavanje mehanizma straničenja na zahtjev može pomoći u pisanju programa koji će se brže izvoditi. Kao primjer može se navesti inicijalizacija cjelobrojnog polja veličine 256 x 256. Svaki cjelobrojni broj prikazan je s četiri okteta. Veličina stranice je 1k oktet. Tipičan kod je:

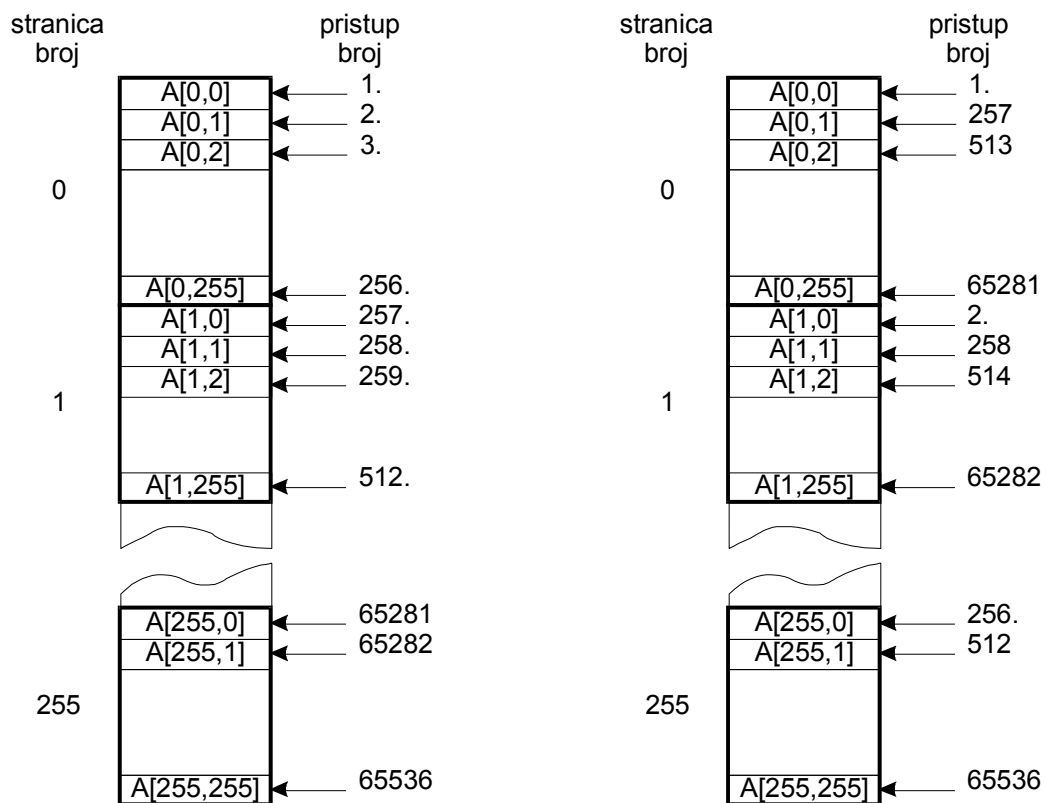
```
long int   A[0..255,0..255];
int        i, j;
```

```
for (i=0; i<256; i++)
{ for (j=0; j<256; j++)
  A[i,j] := 0;};
```

ili

```
for (j=0; j<256; j++)
{ for (i=0; i<256; i++)
  A[i,j] := 0;};
```

Program prevodilac obično slaže polje po redcima tako da slijedom idu $A[0,0]$, $A[0,1]$, ..., $A[0,255]$, $A[1,0]$, ..., $A[255,255]$. Prema organizaciji stranica svaki redak stane u jednu stranicu. Neka operacijski sustav dodijeli procesu npr. 128 okvira. Tada prvi program ima svega 256 promašaja, dok drugi $256 \times 256 = 65536$ promašaja što rezultira znatno dužim izvođenjem. Razlog je opisan slikom 9.11.



Slika 9.11. Pristup stranicama dvodimenzionalnog polja.