

## VIRTUELNA MEMORIJA

=====

(ovaj fajl sadrzi beleške sa predavanja, za više detalja pogledati Dandamudi 735-760, Tanenbaum 428-450, Hamacher, 305-310).

### \*) MOTIVACIJA ZA UPOTREBU VIRTUELNE MEMORIJE

Prve generacije elektronskih računara su memoriju adresirale direktno, tj. adresa koju program koristi kao operand u instrukciji se zaista postavlja na adresnu magistralu i koristi se kao fizička adresa u memoriji (slično je i sa vrednošću PC registra koja se direktno koristi kao fizička adresa instrukcije). Ovaj pristup je bio prihvatljiv kada su računari izvršavali samo jedan program koji je koristio celu memoriju. Sa pojavom prvih operativnih sistema, omogućeno je da se više programa istovremeno nalaze u memoriji i izvršavaju se na procesoru (pomocu deljenja vremena). Na ovaj način omogućeno je da se procesor efikasnije koristi (jer dok jedan program čeka na ulazno-izlaznu operaciju, drugi program može da koristi procesor). Međutim, ovo je dovelo do novih problema kada je u pitanju kontrola upotrebe memorije:

- ) Veći broj programa zahtevaju više memorije. Ovo znači da programi ne mogu čeli da se učitaju u memoriju, jer nema dovoljno mesta. Zbog toga su programi dobijali malu zonu memorije u koju su morali da se uklape. Ukoliko je program prevazilazi veličinu tog prostora, tada je programer sam morao da svoj program podeli na više delova koji se mogu nezavisno učitavati u memoriju (tzv. overlay tehnika). Ovo je bilo mukotržno za programere i podložno greškama. Dakle, javila se potreba da se ovaj postupak automatizuje od strane operativnog sistema, a da bude potpuno nevidljiv za programera koji ne bi trebalo time da se bavi.
- ) Veći broj programa u memoriji može dovesti do konflikta, jer jedan program može (slučajno ili namerno) da pristupi adresama u memoriji koje su dodeljene drugom programu. S obzirom da se u istoj memoriji nalazi i operativni sistem, pristup adresama koje pripadaju operativnom sistemu može potencijalno predstavljati i funkcionalni, ali i bezbednosni problem.
- ) S obzirom da programi u svojim instrukcijama koriste fizičke adrese podataka i drugih instrukcija, ukoliko su te adrese izražene kao apsolutne, tada se program uvek mora učitavati počev od iste adrese u memoriji. Ovo znači da prilikom učitavanja više programa u memoriju oni moraju biti napisani tako da se adrese koje koriste ne preklapaju; u suprotnom, neće moći da koegzistiraju u memoriji. Postoje načini da se ovaj problem prevaziđe, ali nisu tako jednostavni.
- ) S obzirom da programi po pravilu zauzimaju blokove susednih adresa u memoriji (instrukcije su uvek na susednim adresama, a najčešće i podaci), prilikom njihovog učitavanja moramo ih smestiti u neprekidne segmente u memoriji. Kako se koji program učitava, tako mu se dodeljuje sledeći blok susednih adresa u memoriji. Kada se neki program završi, tada se on briše iz memorije, a odgovarajući blok memorije ostaje upraznjen. Na ovaj način vremenom dolazi do FRAGMENTACIJE: može se, na primer, dogoditi da ukupno gledano imamo dovoljno slobodnog prostora za učitavanje nekog programa, ali ga nemamo "u komadu".

Svi ovi problemi se prilično elegantno i efikasno resavaju koriscenjem tehnike virtuelne memorije zasnovane na stranicenju (engl. paging).

#### **\*) PRINCIP RADA VIRTUELNE MEMORIJE ZASNOVANE NA STRANICENJU**

Implementacija virtuelne memorije zahteva kako hardversku, tako i softversku podrsku (od strane operativnog sistema). Virtuelna memorija podrazumeva da svaki program ima sopstveni virtuelni adresni prostor fiksirane velicine. Velicina virtuelnog adresnog prostora ne mora da odgovara velicini fizicke memorije (najcesce ne odgovara). Svaki program misli da je sam na tom racunaru i da je sva njemu vidljiva virtuelna memorija njegova. U instrukcijama, prilikom adresiranja, program koristi virtuelne adrese. Takodje, PC registar sadrzi virtuelnu adresu tekuce instrukcije koja se izvsava. Program uopste nije svestan postojanja fizicke memorije i fizickih adresa.

Neki delovi programa (instrukcija i podataka) ce zaista biti prisutni u fizickoj memoriji. Drugi delovi ce biti prisutni na disku, ali ne i u operativnoj memoriji. Kada program zahteva pristup nekoj virtuelnoj adresi, operativni sistem ce proveriti da li se odgovarajuci podatak nalazi na nekoj fizickoj adresi. Ako je odgovor potvrdan, izvsice se prevodjenje iz virtuelne u fizicku adresu (ovo se odvija hardverski, u procesoru, ali uz pomoc struktura podataka koje inicijalizuje i odrzava operativni sistem). Nakon toga ce se dobijena fizicka adresa postaviti na adresnu magistralu i pristupice se trazenom podatku. Ukoliko se ispostavi da trazeni podatak nije u fizickoj memoriji, tada se on dohvata sa diska, smesta se u operativnu memoriju, azuriraju se strukture podataka operativnog sistema koje mapiraju fizicke i virtuelne adrese i zatim se odgovarajuca fizicka adresa postavlja na adresnu magistralu da bi se dohvatio podatak. Ceo ovaj proces je potpuno nevidljiv za program.

Posledice ovakve organizacije su:

- ) Program moze imati znatno vecu virtuelnu memoriju na raspolaganju nego sto ima fizicke memorije. Npr. na 32-bitnim Intel-ovim procesorima velicina virtuelnog adresnog prostora svakog programa bila je  $2^{32}$ , tj. 4GB. U vreme kada su ovi procesori bili aktuelni, vecina korisnika nije imalo 4GB RAM-a instalirano na racunaru. Uz pomoc diska, bilo je moguće da samo neki delovi programa budu zaista u memoriji, cime je veliki broj programa mogao da koegzistira u memoriji (naravno, veliki broj programa je dovodio do prevelikog koriscenja diska, sto je svakako znacajno usporavalo rad racunara). Pritom, programeri nisu morali da se bave organizacijom memorije -- za programera postoji samo njegov privatni virtuelni adresni prostor velicine 4GB.
- ) Koriscenje odvojenih virtuelnih adresnih prostora je omogucilo izolaciju programa i sprecavanje da jedan program pristupi podacima drugog (ili operativnog sistema). Ovo je doprinelo bezbednosti i smanjilo je broj fatalnih gresaka.
- ) Program se uvek moze ucitavati na isto mesto u svom virtuelnom adresnom prostoru, s obzirom da je on uvek jedini program u tom prostoru. Otuda nije problem koristiti apsolutne (virtuelne) adrese. Problemi sa apsolutnim adresama nastaju samo ako vise programa koriste iste delove koda (npr. deljene biblioteke), kada

se isti kod mapira u virtuelne adresne prostore vise programa. Zbog toga je za pisanje deljenih biblioteka neophodno voditi racuna o tome da kod bude nezavisan od pozicije (tzv. position-independent code (PIC)). U moderno vreme, o tome obicno brinu kompajleri.

- ) Funkcija prevodjenja virtuelnih u fizicke adrese ne mora biti (i obicno nije) linearna. Ovo znaci da susedne lokacije u virtuelnoj memoriju ne moraju biti susedne i u fizickoj. Ovo eliminise problem fragmentacije, jer program u fizickoj memoriji ne mora zauzimati neprekidan blok (iako u virtuelnoj zauzima susedne adrese).

#### \*) NACIN ORGANIZACIJE VIRTUELNE MEMORIJE

Najcesci nacin realizacije virtuelne memorije je tehnika STRANICENJA (engl. paging). Virtuelni adresni prostor podeljen je na STRANICE (engl. pages) jednake velicine. Na primer, na 32-bitnim Intel-ovim procesorima stranice su bile velicine 4KB ( $2^{12}$  bajtova). S obzirom da je virtuelni adresni prostor bio 32-bitni (4GB), to znaci da se on sastojao iz  $2^{20}$  stranica.

Fizicka memorija je takodje podeljena na OKVIRE (engl. page frames) iste velicine (4KB kod Intel-a). Intel-ova arhitektura je u vreme Pentium procesora imala fizicki adresni prostor koji je takodje bio velicine 4GB, ali je vec kod PentiumPro procesora taj fizicki adresni prostor uvecan na 64GB (36-bitne fizicke adrese). Ovaj rezim se obicno naziva PAE (physical address extension) i ukljucuje se posebnim bitom u kontrolnom registru procesora. Mi cemo se u nastavku fokusirati na osnovni slucaj kada je fizicki adresni prostor takodje 32-bitni.

Virtuelna (32bitna) adresa se sada moze logicki podeliti na sledeci nacin:

BROJ STRANICE	POMERAJ
-----	-----
20 bita	12 bita

Ovu adresu je potrebno prevesti u fizicku adresu (32-bitnu u nasem primeru):

BROJ OKVIRA	POMERAJ
-----	-----
20 bita	12 bita

Ovo se postize pomocu TABELE STRANICA (Page Table). Ova tabela se sastoji iz STAVKI (Page Table Entry (PTE)). Za svaku stranicu postoji jedna stavka (ukupno  $2^{20}$  stavki u nasem primeru). Otuda se broj stranice koristi kao indeks u ovoj tabeli. Tabela stranice se takodje nalazi u fizickoj memoriji, a njena adresa se cuva u posebnom registru procesora (koji se inicijalizuje od strane operativnog sistema i deo je podataka o programu koji se izvorsava; svaki program koji se izvorsava ima svoju tabelu stranica koja se cuva u memoriji, a prilikom zamene programa koji se izvorsava menja se i vrednost ovog registra, tako sto se u njega ucitava adresa tabele stranica programa koji treba da nastavi sa izvorsavanjem).

Svaka PTE stavka sadrži broj okvira fizicke memorije u kom se nalazi stranica, ukoliko je stranica zaista u fizickoj memoriji. U suprotnom, PTE stavka sadrži informaciju da stranica nije u memoriji. Pored toga, PTE stavka sadrži i dodatne bitove koji mogu blize odredjivati način koriscenja ove stranice (npr. da li je stranica samo za citanje, ako npr. sadrži instrukcije, ili se u nju može i pisati, da li je koriscena od kako je učitana u fizicku memoriju, da li je modifikovana, i sl.). Prilikom pristupa virtuelnoj adresi, procesor automatski pronalazi stavku u tabeli stranica. Ukoliko je stranica prisutna u RAM-u, tada se broj okvira i pomeraj kombinuju u fizicku adresu i salju se na adresnu magistralu. U suprotnom, dolazi do izuzetka koji se naziva GRESKA STRANICE (engl. page fault). Ovaj izuzetak automatski pokreće izvršavanje funkcije za obradu prekida operativnog sistema koji odlazi na disk, učitava odgovarajuću stranicu sa diska u RAM, azurira tabelu stranica i vraća kontrolu programu, pri čemu se instrukcija koja je izazvala izuzetak ponovo pokreće. Ovog puta, stranica je u memoriji.

Stranice programa koje sadrže instrukcije se svakako nalaze i na disku (kao deo odgovarajućeg izvršnog fajla), odakle se po potrebi mogu učitavati. Stranice koje odgovaraju drugim zonama virtuelne memorije (stek, heap) koje dinamički rastu se inicijalno alociraju u okvirima u RAM-u, po potrebi. Međutim, kada se RAM popuni (ne zaboravimo da u njega smestamo stranice svih programa koji rade na računaru, uključujući i sam operativni sistem), više nije moguće alocirati nove okvire za stranice. Tada je neophodno izvršiti ZAMENU STRANICA. Potrebno je neku od stranica izbaciti iz okvira u fizickoj memoriji, sacuvati je na disku, a okvir osloboditi za drugu stranicu kojoj želimo da pristupimo. Kada nam stranica koju smo zamenili u nekom trenutku ponovo zatreba, ponovo ćemo je učitati sa diska u RAM (ne obavezno u isti okvir). Iz svega ovoga sledi da će mnoge stranice istovremeno postojati i na disku i u RAM-u. Ako postoji u RAM-u, pristupamo joj iz RAM-a, a ako ne postoji, učitavamo je sa diska u RAM, pa joj odatle pristupamo. Ovo dosta liči na ono što smo imali kod keš memorije -- tamo smo imali kopije istog keš bloka i u kešu i u RAM-u. Otuda se i ovde postavljaju ista pitanja kao i kod keša: koja politika se koristi prilikom zamene stranica, kao i koja politika se koristi prilikom promene stranica?

Prilikom zamene stranica koriste se slični algoritmi kao kod keš memorija. Najčešće se koristi neka varijanta LRU strategije. Iako se ovde algoritam zamene implementira softverski, ipak nije moguće implementirati punu LRU strategiju, s obzirom na veliki broj okvira. Zbog toga se i ovde primenjuje neka varijanta pseudo-LRU strategije: procesor najčešće održava informaciju o tome koje su stranice korišćene, a koje ne (postoji poseban bit u PTE stavkama koji tome služi). Stranice kod kojih je ovaj bit isključen se mogu smatrati stranicama koje se dugo ne koriste i neka od njih se može izbaciti. Operativni sistem s vremena na vreme resetuje ove bitove.

Kada su u pitanju politike pisanja, za razliku od keša, ovde nema smisla koristiti politiku sa propustanjem (write-through), s obzirom da je razlika u brzini između RAM memorije i hard diska još drasticnija nego između keša i RAM-a: upis na disk pri svakoj promeni stranice značio bi veliki gubitak na efikasnosti. Zbog toga se kod virtuelne memorije po pravilu koristi politika pisanja sa prepisivanjem (write-back): uz svaku stranicu se u PTE stavkama čuva

i "prljavi bit" (dirty bit) koji procesor automatski azurira prilikom promene sadržaja stranice u RAM-u. Prilikom zamene stranice, ako je bilo promena, stranica će biti sacuvana na disku, a u suprotnom neće, jer je kopija koju već imamo na disku azurna.

#### \*) STRANICENJE NA VISE NIVOVA

Glavni problem sa gore opisanom organizacijom virtuelne memorije je u tome što tabele stranica mogu biti prilično velike. U gornjem primeru, imamo  $2^{20}$  stavki u tabeli stranica. Svaka stavka je na Intel-ovoj arhitekturi velicine 4 bajta (32 bita). Ovo znači da imamo ukupno  $2^{22} = 4\text{MB}$  prostora koji zauzima sama tabela stranica. Ova tabela mora zauzimati neprekidan prostor u fizickoj memoriji (to je ukupno  $2^{10}=1024$  uzastopnih okvira u RAM-u). Tabela stranica za program koji se trenutno izvršava mora biti u celini prisutna u fizickoj memoriji. Tabele stranica ostalih programa se možda i mogu privremeno izmestiti na disk. Međutim, nakon što pokušamo ponovo da ih učitamo, imaćemo problem fragmentacije (jer im je potrebno pronaći slobodnih uzastopnih 1024 okvira u RAM-u).

Ovaj problem se rešava stranicenjem na više nivoa. Naime, umesto da insistiramo da tabela stranica mora da zauzima 1024 uzastopnih okvira u RAM-u, možemo dozvoliti da se tih 1024 stranica koje sadrže stavke tabele stranica i same rasprse po memoriji u proizvoljne okvire. Da bismo imali informaciju o tome gde se koja stranica nalazi, uvodi se dodatna tabela koja se zove DIREKTORIJUM TABELE STRANICA (engl. Page Directory). Ovaj direktorijum sadrži stavke (page-directory entry (PDE)), koje su takodje na Intel-ovoj arhitekturi velike 32 bita (4 bajta). Ovakvih stavki ima 1024 (za svaku stranicu tabele stranica po jedna), a svaka stavka sadrži broj fizickog okvira u kome se nalazi odgovarajuća stranica tabele stranica. Ukupna velicina direktorijuma tabele stranica je  $1024 \times 4\text{B} = 4\text{KB}$ , što staje u jedan okvir! Sada će poseban registar u procesoru sadržati redni broj fizickog okvira koji sadrži direktorijum tabele stranica. Prilikom prevodjenja, najvisih 10 bitova virtuelne adrese se koristi kao indeks u direktorijumu, iz koga se određuje broj okvira u kome se nalazi odgovarajuća stranica tabele stranica. Zatim se sledećih 10 bitova virtuelne adrese koriste kao indeks u toj stranici da bi se odredio broj okvira same stranice koju tražimo:

BROJ DIREKTORIJUMA	BROJ STRANICE	POMERAJ
10 bita	10 bita	12 bita

Prednost ovog pristupa je to što se sada samo direktorijum stranica mora nalaziti u fizickoj memoriji, dok se stranice iz kojih se sastoji tabela stranica mogu nalaziti i na disku i po potrebi učitavati. Takodje, rešen je problem fragmentacije, jer se ne moraju nalaziti u susednim okvirima u fizickoj memoriji. Nedostatak je u izvesnom gubitku efikasnosti, jer sada imamo jedan nivo indirekcije više prilikom prevodjenja adresa. Ipak, ovaj gubitak nije drastican u praksi, zahvaljujući koriscenju TLB keša (videti dole).

#### \*) KORISCENJE VECIH STRANICA

Noviji Intelovi procesori podržavaju i veće velicine stranica, npr.

od 2MB ili 4MB. U slučaju da koristimo stranice od 4MB, tada će struktura virtuelne adrese biti sledeća:

	BROJ STRANICE		POMERAJ	
-----		-----		
	10 bita		22 bita	

Dakle, nizih 22 bita predstavljaju pomeraj u okviru 4MB velike stranice, dok visih 10 bita određuju broj stranice (sada je ceo virtuelni 32-bitni prostor sastavljen iz svega 1024 stranice od po 4MB). Tabela stranica sadrži ukupno 1024 stavke od po 32 bita, što nam daje tabelu velicine 4KB. Dakle, imamo veoma male tabele stranica, jer imamo mali broj velikih stranica. Ovo je drugi način kako se može rešiti problem sa velikim tabelama stranica: samo treba koristiti veće stranice, pa će tabele stranica biti manje.

Jos jedna dobra strana velikih stranica je u tome što efikasnije koriste disk: s obzirom na način rada diska, citanje stranice od 4MB sa diska ne traje mnogo duže od citanja stranice od 4KB, a s obzirom na drastično manji broj stranica citanje i pisanje na disk biće mnogo manje frekventno.

Medjutim, postoje i loše strane upotrebe velikih stranica. Prvi problem je velika interna fragmentacija. Naime, kako program zauzima određeni broj uzastopnih stranica virtuelnog adresnog prostora, poslednja "naceta" stranica će po pravilu biti polu-prazna. Sto su stranice veće, to je taj baceni prostor u operativnoj memoriji veći. Drugi problem je manja preciznost: kada nam je potrebna neka adresa, mi ćemo sa njom učitati i celu njenu okolinu velicine 4MB, što često uključuje i mnogo toga što nam ne treba niti će nam biti potrebno u narednom periodu.

#### \*) KORISCENJE VEĆEG FIZICKOG ADRESNOG PROSTORA

Počet od PentiumPro procesora, Intel je omogućio veći fizički adresni prostor od 64GB (36-bitne fizičke adrese), dok je virtuelni adresni prostor ostao 32-bitni. Sada su brojevi okvira 24-bitni, pa stavke u tabeli stranica uz sve ostale bitove koje moraju da sadrže ne mogu da stanu u 32 bita. Zbog toga se koriste 64-bitne stavke, što tablice čini duplo većim. Otuda se u slučaju 4KB velikih tablica koristi čak tri nivoa stranicenja: najpre se na prvom nivou bira jedan od četiri POKAZIVACA DIREKTORIJUMA STRANICA: oni su smesteni u jednoj TABELI POKAZIVACA koja sadrži samo četiri stavke, a indeksira se pomoću dva najviša bita virtuelne adrese. Poseban registar u procesoru čuva adresu ove tabele u fizičkoj memoriji. Stavke u ovoj tabeli ukazuju na jedan od četiri direktorijuma stranica. U svakom direktorijumu stranica se narednih 9 bitova virtuelne adrese koriste kao indeksi za jednu od 512 stavki (svaka stavka je 8 bajtova velika, što direktorijum čini 4KB velikim, što staje u jednu stranicu). Stavka iz direktorijuma nam daje fizičku adresu stranice tabele stranica koju indeksiramo pomoću sledećih 9 bitova virtuelne adrese. Stavka u ovoj tabeli nam daje 24-bitni redni broj okvira u kome se nalazi tražena stranica.

	DIR_PTR		DIR		BR STRANICE		POMERAJ	
-----		-----		-----		-----		

2 bita      9 bita      9 bita      12 bita

U slučaju koriscenja vecih stranica (4MB), broj okvira je  $2^{14}$ , pa je 14 bitova dovoljno za odredjivanje rednog broja okvira. Otuda stavke ponovo mogu da stanu u 32 bita, pa imamo samo jednu tabelu stranica sa 1024 stavki od po 4 bajta (tabela velicine 4KB). Tabela se indeksira pomocu visih 10 bitova virtuelne adrese.

BROJ STRANICE	POMERAJ
10 bita	22 bita

#### \*) STRANICENJE KOD X86-64 PROCESORA

Noviji 64-bitni Intel/AMD procesori teorijski koriste 64-bitni virtuelni adresni prostor. Medjutim, aktuelne implementacije visih 16 bitova virtuelne adrese uvek oznaceno prosiruje, tj. efektivno ih ne koristi. Otuda je stvarna velicina virtuelnog adresnog prostora 48-bitna. Fizicki adresni prostor varira od 36-bitnog do 40-bitnog, pa i vise. Pritom, koriste se sledece dve varijante organizacije virtuelne memorije:

1) 4KB stranice: prevodjenje adrese se vrsi na cetiri nivoa (4 x 9 visih bitova virtuelne adrese, ne uzimajuci najvisih 16 koji se ne koriste). Svaka od stavki u svim tabelama i direktorijumima je 64-bitna, a sve tabele i direktorijumi sadrze  $2^9=512$  stavki, pa staju u po jednu stranicu. [Direktorijum na najvisem nivou se oznacava sa PML4 (skraceno od Page-Map-Level 4, sto oznacava cetvri nivo mapiranja; u pitanju je Intel-ova terminologija i nije toliko bitna).

PML4	DIR PTR	DIR	BR STRANICE	POMERAJ
9 bita	9 bita	9 bita	9 bita	12 bita

2) 2MB stranice: prevodjenje adrese se vrsi na tri nivoa (3 x 9 visih bitova virtuelne adrese ne uzimajuci u obzir visih 16 bitova koji se ne koriste). Nizih 21 bit se koristi kao pomeraj u okviru 2MB velike stranice. Stavke su velike 64-bita, a svaka tabela ili direktorijum ima po 512 ovakvih stavki, pa staje u 4KB. Otuda stavke u direktorijumima sadrze redne brojeve okvira od po 4KB (tj. 28-bitne brojeve), dok stavke u samim tabelama stranica sadrze 19-bitne brojeve 2MB-nih okvira (do 40 bita su fizicke adrese).

DIR_PTR	DIR	BR STRANICE	POMERAJ
9 bita	9 bita	9 bita	21 bita

#### \*) TLB (Translation-Lookaside Buffer)

Kako se tabele i direktorijumi stranica nalaze u memoriji, da bismo izvršili prevodjenje virtuelnih u fizicke adrese, neophodno je da odemo u memoriju po odgovarajuće stavke iz tabela. Kako je pristup memoriji spor, ovo bi bilo nedopustivo neefikasno. Zbog toga je neophodno koristiti keširanje. TLB je specijalni keš u procesoru koji se koristi u ove svrhe. U pitanju je veoma mali i veoma brz keš, obično implementiran kao razdvojen keš (poseban za instrukcije, a poseban za podatke). U slučaju manjih TLB-ova koristi se potpuno asocijativno mapiranje, dok u slučaju većih koristi se skup-asocijativno mapiranje. TLB dohvata i čuva u sebi stavke tabela i direktorijuma stranica. Stranice koje su nedavno korišćene se sada mogu razrešiti direktno u TLB-u, pa se fizicka adresa može izračunati na osnovu virtuelne bez odlaska u memoriju. U slučaju da TLB ne sadrži traženu stavku, odlazi se u memoriju i stavka se prebacuje u TLB.