Upravljanje memorijom (Virtualna memorija: segmentacija i straničenje)

Hajmiler, Ivan

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I

INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Preddiplomski stručni studij Elektrotehnike, smjer Informatika

UPRAVLJANJE MEMORIJOM (VIRTUALNA MEMORIJA: SEGMENTACIJA I STRANIČENJE)

Završni rad

Ivan Hajmiler

Osijek, 2020.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1S: Obrazac za imenovanje Povjerenstva za završni ispit na preddiplomskom stručnom studiju

Osijek, 18.09.2020.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za završni ispit na preddiplomskom stručnom studiju

Ime i prezime studenta:	Ivan Hajmiler
Studij, smjer:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI4544, 26.09.2019.
OIB studenta:	91713605269
Mentor:	Goran Bokun
Sumentor:	-
Sumentor iz tvrtke:	-
Predsjednik Povjerenstva:	Prof. dr. sc. Goran Martinović
Član Povjerenstva 1:	Goran Bokun
Član Povjerenstva 2:	Izv. prof. dr. sc. Ivica Lukić
Naslov završnog rada:	Upravljanje memorijom (Virtualna memorija: segmentacija i straničenje)
Znanstvena grana rada:	**Procesno računarstvo (zn. polje računarstvo)**
Zadatak završnog rada	Ukratko objasniti upravljanje memorijom. Dati pregled virtualne memorije općenito. Opisati straničenje kao metodu implementacije virtualne memorije. Opisati segmentaciju kao metodu virtualne memorije. Dati usporedbu te dvije metode.
Prijedlog ocjene pismenog dijela ispita (završnog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3. razina
Datum prijedloga ocjene mentora:	18.09.2020.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 05.10.2020.

Ime i prezime studenta:	Ivan Hajmiler
Studij:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI4544, 26.09.2019.
Turnitin podudaranje [%]:	2

Ovom izjavom izjavljujem da je rad pod nazivom **Upravljanje memorijom (Virtualna memorija: segmentacija i straničenje)**

izrađen pod vodstvom mentora Goran Bokun

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.

Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

U ovom završnom radu detaljno će se obraditi straničenje i segmentacija kao praktična ostvarenja mehanizma upravljanja radnom memorijom kojeg zovemo virtualna memorija.

U uvodnom dijelu rada bit će objašnjene teme koje su bitne za razumijevanje ove problematike, a to se odnosi na organizaciju memorije, adresiranje memorije, memorijsku hijerarhiju te vezu memorije i procesora. Također, kratko će biti objašnjen pojam upravljanja memorijom, na koji se nadovezuje tema virtualne memorije, koja će biti detaljno obrađena u središnjem dijelu ovoga rada.

Objasnit će se preklopni način upravljanja memorijom kao preteča virtualne memorije, a onda pojam virtualne memorije te zašto je ona nastala. Nakon toga detaljno će biti obrađene straničenje i segmentacija kao metode implementacije virtualne memorije. Nadalje, bit će uspoređene ove dvije metode te će biti objašnjeno koje su prednosti i nedostaci pojedine od ovih metoda. Dobit će se uvid u to kako programer može utjecati na segmentiranu memoriju. Također, bit će objašnjeno kako se ove metode mogu kombinirati kako bi se smanjili nedostatci u odnosu kada su korištene zasebno. S obzirom da se tehnologija stalno razvija, pojavljuje se potreba za prilagodbom i nadogradnjom ovih metoda. Navedene promjene i njihova svrha bit će objašnjene na primjeru Windows 10 operacijskog sustava koji radi na 64-bitnoj računalnoj arhitekturi.

Kroz ovaj rad dobit će se uvid u funkcioniranje i primjenu virtualne memorije koja unatoč razvoju sve većih kapaciteta radne memorije i dalje ostaje važan dio svakog računalnog sustava budući se i veličina programa povećava, a nerijetko se u radnoj memoriji nalazi i veliki broj pokrenutih programa.

2. ORGANIZACIJA MEMORIJE I ADRESIRANJE

2.1. Program i virtualne adrese

Osnovna funkcija računala je izvršavati zadatke koji su mu dani kroz odgovarajući program, a program je skup instrukcija čiji je smisao riješiti neki računalni problem. Program najčešće piše programer u nekom od dostupnih programskih jezika, npr. u C programskom jeziku. Ovako napisani program zove se izvorni kod (engl. *source code*) te kažemo da je ovakav kod čovjeku razumljiv, ali ne i računalu. Kako bi računalo razumjelo napisani kod potrebno ga je prevesti u jezik koji je njemu razumljiv, a taj jezik zove se strojni kod i sastoji se od niza 0 i 1. Zadatak prevođenja izvršava poseban program koji se zove prevoditelj (engl. *compiler*). Nakon prevođenja dobiveni kod potrebno je povezati u jednu funkcionalnu cjelinu, a ovaj zadatak obavlja program koji se zove povezivač (engl. *linker*). Ako su svi koraci uspješno obavljeni, kao rezultat ovih aktivnosti dobiva se izvršna datoteka koju je moguće pokrenuti na računalu.[1]

Svaki program ima svoj adresni prostor, a zovemo ga virtualni adresni prostor te je sastavljen od niza virtualnih adresa. Ove adrese zovemo virtualnima jer je moguće, prilikom izvođenja programa, da traženi sadržaj kojeg virtualne adrese referenciraju još nije učitan u fizičku memoriju. Nakon prevođenja virtualne adrese koje referenciraju pojedine podatke i metode postaju dijelom instrukcija. Dakle, procesor obrađuje virtualne adrese, a poseban dio hardvera ih onda pretvara u fizičke adrese, odnosno stvarne lokacije u memoriji, iz kojih se dohvaćaju određeni podatci ili se u njih spremaju.[2] Korištenje virtualnih adresa stvorilo je preduvjet za razvoj mehanizma upravljanja memorijom koji se zove Virtualna memorija, a što će biti detaljno objašnjeno dalje u radu. Međutim, prije toga još će ponešto biti rečeno o temama koje su bitne za bolje razumijevanje ove problematike.

2.2. Primarna memorija i fizičke adrese

Općenito memorija je dio računala gdje se pohranjuju programi te podatci koje ti programi referenciraju. Primarna memorija je ona koja je izravno dostupna procesoru (npr. radna memorija) stoga se u nju učitava program kojeg procesor obrađuje. Suprotan pojam je sekundarna memorija (npr. tvrdi disk) te nije izravno dostupna procesoru. Za primarnu memoriju još se koriste izrazi glavna memorija i fizička memorija.

Osnovna jedinica memorije je bit, što se odnosi na binarnu znamenku, koja može imati vrijednost 0 ili 1. Ovakva konstrukcija memorije usko je vezana uz računalno sklopovlje, odnosno elektroničke sklopove koji informaciju zapisuju kao različite razine napona. Visoka razina napona odnosi se na znamenku 1, a niska razina napona na znamenku 0.

Više bitova čini memorijsku stanicu. Svaka stanica sastoji se od jednakog broja bitova. Prema tome računalnu memoriju možemo zamisliti kao jednodimenzionalni niz koji se sastoji od uzastopnih (susjednih) memorijskih stanica koje se koriste za pohranjivanje i dohvaćanje podataka. Svaka od ovih stanica ima svoju jedinstvenu adresu, odnosno broj koji jednoznačno određuje gdje se određena stanica u memoriji nalazi, a na koju se program onda može referencirati. Ovakva memorijska adresa zove se fizička adresa. Računala koja koriste binarni brojevni sustav, kao što je prethodno objašnjeno, adresu zapisuju u binarnom brojevnom sustavu. Valja naglasiti da je stanica najmanji dio memorije koji se može adresirati, pa tako i najmanji dio informacije kojem se može pristupiti odnosno kojeg se može unositi u memoriju ili iz nje dohvaćati.

U današnje vrijeme, većina proizvođača računala standardizirala je veličinu memorijske stanice na 8 bita. Ovakva stanica zove se bajt (engl. *byte*), a memoriju zovemo bajt adresibilnom (engl. *byte adressable*), svaka adresa referencira jedan bajt. Međutim, bajti se mogu grupirati u riječi (engl. *words*), pa se tako primjerice 32-bitna riječ sastoji od 4 bajta, a 64-bitna riječ od 8 bajta. Današnja računala uglavnom rade sa riječima, odnosno procesor dohvaća, obrađuje i sprema riječi u memoriji. Korištenje 32-bitnih riječi zahtijeva 32-bitnu računalnu arhitekturu te instrukcije za manipulaciju 32-bitnih riječi, a korištenje 64-bitnih riječi zahtijeva 64-bitnu računalnu arhitekturu te instrukcije za manipulaciju 64-bitnih riječi. Međutim, neovisno o veličini registra ili količini bajta koju sadrži, memorija je i dalje bajt adresibilna. Ako adresa ima n bita, najveći broj stanica koje može adresirati je 2^n, pri čemu adrese idu od 0 do 2^n-1. Prema tome 32-bitni registar može koristiti 2^{32} različitih kombinacija 0 i 1, odnosno 2^{32} različitih adresa. Ako je memorija bajt adresibilna, u ovom slučaju to znači da se može adresirati 2^{32}, odnosno 4 294 967 296 bajta, što je 4 GB radne memorije.

Iz navedenog se može vidjeti kako će veći kapacitet bajt adresibilne radne memorije, primjerice 16 GB, što je slučaj u velikom broju modernih računala, zahtijevati veći kapacitet registra kako bi se njezin adresni prostor mogao u cijelosti referencirati, odnosno kako bi se dostupna radna memorija u potpunosti mogla iskoristiti. Zbog navedenih razloga, u današnje vrijeme uglavnom se koristi 64-bitna računalna arhitektura koja može adresirati puno veći kapacitet radne memorije.[2]

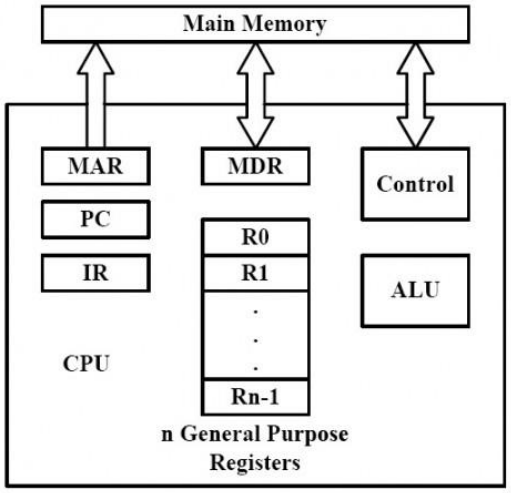
2.3. Procesor i veza s memorijom

Kada korisnik pokrene program na računalu on se mora učitati u radnu memoriju kako bi procesor mogao dohvatiti, obrađivati i izvršavati instrukcije i podatke koji su u njemu definirani, ovakav program u izvršavanju sada zovemo proces.

Procesor je elektronički sklop koji izvršava instrukcije koje su mu dane kroz program. Sastoji se od nekoliko osnovnih komponenti, a to su: aritmetičko-logička jedinica, kontrolna jedinica i registri. Za potrebe ovog rada valja spomenuti još jednu komponentu koja se zove jedinica za upravljanje memorijom (engl. *Memory Managemet Unit*), koja može, ali i ne mora biti dio procesora, međutim najčešće jest njegov dio, a obavlja vrlo važan zadatak prevođenja virtualnih adresa koje su dio programa u izvođenju, odnosno procesa, u fizičke adrese koje su razumljive memoriji, a što će biti detaljno obrađeno dalje kroz rad. Aritmetičko logička jedinica je ta koja zapravo izvršava instrukcije pohranjene u programu. Kontrolna jedinica je odgovorna za pravovremeno dohvaćanje instrukcija iz memorije, njihovo dekodiranje te pronalazak i dohvaćanje odgovarajućih operanda koje onda prosljeđuje aritmetičko-logičkoj jedinici na obradu. Registri su mjesta privremene pohrane podataka, dio su arhitekture procesora, imaju mali kapacitet, ali pružaju najbrži pristup podatcima, odnosno čitanje i pisanje podataka. Više je registara potrebno kako bi se omogućile aktivnosti procesora. Neki od ovih registara su:

- Memorijski adresni registar (engl. *Memory address register* - MAR) i memorijski podatkovni registar (engl. *Memory data register* - MDR), koji služe za prijenos podataka između glavne memorije i procesora. MAR pohranjuje fizičke adrese, a MDR podatke.
- Instrukcijski registar (engl. *Instruction register* - IR), koji pohranjuje instrukcije koje se trenutno izvode.
- Programsko brojilo (engl. *Program counter* - PC), koje pokazuje na iduću instrukciju koju treba dohvatiti iz memorije.
- Registri za opću upotrebu(engl. *General purpose registers*), koji mogu pohranjivati adrese i podatke.
- Unutarnji registri (engl. *Internal register*) vezani uz aritmetičko-logičku jedinicu, koji mogu biti ulazni i izlazni.

Kako bi se lakše razumjela ova problematika, pogledati sliku 2.1. koja prikazuje vezu između procesora i memorije.



Slika 2.1. *Veza memorije i procesora*

U idućem primjeru na slikovit način sumiran je i povezan prethodno napisani sadržaj. Naprimjer, zamislimo da procesor treba izvršiti operaciju zbrajanja i rezultat pospremiti u varijablu:

$$C= A+B$$

Neovisno koji programski jezik koristili za pisanje programa iz ovog primjera, tijekom prevođenja instrukcije će biti prevedene u strojni kod. Ovaj kod će sadržavati sve informacije koje su potrebne za izvršenje danog zadataka, a to je kod za instrukciju zbrajanja, adresu prvog operanda, adresu drugog operanda i adresu varijable C u koju će se spremiti rezultat zbrajanja.

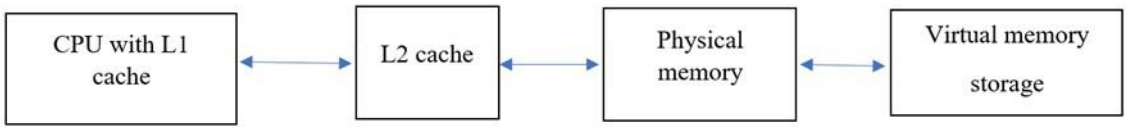
Kada korisnik pokrene program, on će se učitati u radnu memoriju i tek tada će biti dostupan procesoru za izvršavanje. Kontrolna jedinica će dohvatiti instrukciju, dekodirati ju te shvatiti da se radi o operaciji zbrajanja. Instrukcija koja se trenutno izvodi bit će pohranjena u instrukcijskom registru. S obzirom da ova instrukcija ima tri parametra, dohvatit će prva dva te ih smjestiti u registre za opću upotrebu, a iz njih u unutarnje ulazne registre aritmetičko-logičke jedinice ili konkretnije u registre njezine komponente koja se zove zbrajalo (engl, *adder*). Međutim, prije toga će morati pronaći fizičke adrese na kojima se nalaze operandi, jer adrese koje su pohranjene u programu su virtualne adrese, a radna memorija razumije samo fizičke adrese. Zadatak prevođenja virtualnih u fizičke adrese odrađuje poseban hardver koji je već spomenut prethodno u radu, a zove se jedinica za upravljanje memorijom. Kada je adresa prevedena i ako je sadržaj na toj adresi učitan u radnu memoriju, adresa će biti učitana u memorijski adresni registar, a podatke koji se nalaze na toj adresi će kontrolna jedinica procesora učitati u memorijski podatkovni registar odakle će se prebaciti u druge registre kako je rečeno.

Ako podatak na nekoj adresi nije učitan, hardver će izdati prekid koji će operacijski sustav registrirati, obraditi i učitati traženi sadržaj u radnu memoriju. Ova problematika, a radi se o upravljanju radnom memorijom, detaljno će biti obrađena kroz rad. Kada su operandi učitani u unutarnje registre zbrajala, kontrolna jedinica će izdati signal da se može izvršiti instrukcija zbrajanja. Kada zbrajalo završi, kontrolna jedinica će prebaciti rezultat koji se nalazi u unutarnjem izlaznom registru u memorijski podatkovni registar, odakle će se rezultat učitati u radnu memoriju, ako je podatak potreban za daljnju obradu prebacit će se u neki od registara za generalnu upotrebu, odakle će se dalje prosljeđivati prema zadanim instrukcijama. Programsko brojilo pokazuje na iduću instrukciju koju treba izvršiti, npr. ako dodamo operaciju množenja u naš program i ponovno ga pokrenemo programsko brojilo može pokazivati na tu instrukciju, dok se izvršava instrukcija zbrajanja.[3]

Pojedine komponente procesora i memorije povezuju specijalni vodiči koji se zovu sabirnice. Ovaj pojam također obuhvaća i komunikacijske protokole koji se koriste tijekom razmjene informacija. U prethodno objašnjenom slučaju razmjenjuju se adrese, podatci i kontrolni signali pa prema tome postoji adresna, podatkovna i kontrolna sabirnica. Moguće je imati i jednu sabirnicu koja prenosi sve vrste signala, što je bio slučaj kod nekih starijih računala, ali podjela na više sabirnica ubrzava prijenos podataka i pojednostavljuje mehanizme korištenja sabirnice stoga se ovakav pristup koristi u modernim računalima. Pojedine sabirnice moraju biti odgovarajuće široke kako bi na efikasan način prenosile informacije te kako ne bi postale usko grlo sustava. Adresna sabirnica bi trebala biti dovoljno široka kako bi mogla prenositi adresu zadnje stanice u memoriji. Podatkovna sabirnica bi trebala biti u mogućnosti prenositi barem veličinu memorijske stanice. Ako je premala morat će podatke prenositi u više ciklusa, što značajno smanjuje performanse sustava.[2]

2.4. Memorijska hijerarhija

Kako je procesor brži od memorije često dolazi do situacije u kojoj procesor mora čekati na podatke koji su mu potrebni da uspješno izvrši dani mu zadatak. Idealno rješenje za ovaj problem bilo bi imati veliku količinu trajne, jeftine i vrlo brze memorije (po mogućnosti brzine iste kao procesor), ali trenutno nije razvijena tehnologija koja bi zadovoljila sve ove zahtjeve. Međutim, kako bi se umanjila razlika u brzini između procesora i memorije, osmišljen je računalni sustav koji koristi više tipova memorije. Svaka od njih pruža neke prednosti te se međusobno nadopunjuju i tvore hijerarhijski memorijski sustav računala (slika 2.2.).



Slika 2.2. *Opća memorijska hijerarhija*

Što je memorija bliža procesoru ona je u pravilu manjeg kapaciteta, brža, ali i skuplja, a što je dalje od njega to je većeg kapaciteta, sporija i jeftinija. Ovdje će biti pojašnjene vrste memorija koje su bitne za razradu, a ostale će biti izostavljene.

Na vrhu memorijske hijerarhije nalaze se registri kojima se može pristupiti na razini brzine procesora. Oni su privremena spremišta podataka malog kapaciteta te su dio arhitekture procesora. Kod x86 arhitekture veličine su 32 bita, a u modernim osobnim računalima najčešće su veličine 64 bita, odnosno jedan registar može pohraniti 64 bita.

Iduća u hijerarhiji dolazi priručna memorija (engl. *cache*). Računala mogu imati više priručnih memorija koje služe za pohranjivanje često korištenih podataka i instrukcija što značajno povećava brzinu pristupa podatcima. Veličina ove memorije uobičajeno se kreće od 16 KB do nekoliko megabajta. Ovakva vrsta memorije često se koristi i u drugim dijelovima računalnog sustava pa tako postoji npr. priručni međuspremnik za prevođenje adresa (engl. *translation lookaside buffer*) koji je dio procesorske jedinice za upravljanje memorijom te ubrzava proces prevođenja memorijskih adresa tako što pohranjuje česte prijevode. O njemu će se detaljnije govoriti dalje u radu s obzirom na njegovu važnost u procesu upravljanja memorijom.

Radna memorija se naslanja na priručnu memoriju te je iduća u hijerarhiji. Radnu memoriju obično zovemo RAM (skraćeno od *Random Access Memory*). Ova je memorija zbog svojih karakteristika, odnosno omjera brzine, veličine i cijene, zauzela središnju i najbitniju ulogu u memorijskoj hijerarhiji, u nju se učitava program kojeg procesor obrađuje. U današnje vrijeme obično je veličine od nekoliko gigabajta do 16 GB, 32 GB pa i više. Sadržaj ove memorije kao i priručne memorije koja je prethodno spomenuta je nestalan, odnosno briše se kada nema struje. Stoga sve što se želi trajno pohraniti treba prebaciti na neki memorijski medij za postojanu pohranu podataka, npr. tvrdi disk.

Na kraju memorijske hijerarhije nalazi se tvrdi disk koji je daleko najsporiji od svih dosad spomenutih memorija, ali njegova prednost je veliki kapacitet koji se u današnjim računalima, u

nekim slučajevima može izražavati i u terabajtima te trajna pohrana podataka neovisna o izvoru struje.

Također jedan njegov dio koji se zove *swap* disk, *swap file* ili *page file* koristi se kao proširenje prethodno spomenute radne memorije. Na taj način povećava se kapacitet radne memorije, odnosno dio programa koji se izvršava može se ovdje učitati ako u cijelosti ne stane u radnu memoriju te se prema potrebi može učitavati u radnu memoriju, što je omogućilo pokretanje programa koji su veći od dostupne radne (fizičke) memorije. Ovakav mehanizam upravljanja memorijom nazivamo Virtualna memorija.[1]

2.5. Upravljanje memorijom

Upravljanje memorijom jedna je od najvažnijih odgovornosti računalnog sustava. Kao što je bilo rečeno, idealno bi bilo imati jako brzu memoriju (brzine procesora) koja je postojana i ima veliki kapacitet. Međutim, to još nije praktično izvedivo pa je osmišljena memorijska hijerarhija koja je prethodno objašnjena. Ovakva raspodjela memorije zahtijeva brižno upravljanje, a to je zadatak koji obavlja operacijski sustav, konkretnije njegov dio koji se naziva upravitelj memorijom. Operacijski sustav usko surađuje s hardverom kako bi na efikasan način upravljao memorijom. Mora voditi računa da procesi koji su pokrenuti na računalu dobiju odgovarajuću količinu memorije kada im je to potrebno. Također, kada procesi završe s radom treba osloboditi memoriju kako bi se drugi procesi mogli izvršavati. Ovakav način rada, kada postoji više procesa koji se izvršavaju na računalu, naziva se više-programski način rada.

S vremenom razvijene su radne memorije velikog kapaciteta, ali se značajno povećala i veličina programa. Kako bi se mogli pokretati programi koji su veći od dostupne radne memorije, čije se zauzeće značajno povećava kod više-programskog rada, osmišljen je mehanizam upravljanja memorijom koji se naziva Virtualna memorija.[1] Ovo je upravo i tema ovoga rada stoga će nadalje detaljno biti obrađena. Krenut će se nešto općenitije pa će se prvo pobliže objasniti pojam virtualne memorije, a onda će se nastaviti s metodama implementacije virtualne memorije koje će na koncu biti i uspoređene.

3. VIRTUALNA MEMORIJA

U početcima računarstva, memorije su bile malog kapaciteta i vrlo skupe. Programeri su provodili puno vremena pokušavajući smanjiti programe kako bi stali u jako ograničeni kapacitet fizičke memorije. Nadalje, često puta bilo je potrebno koristiti algoritam koji je radio puno sporije, nego neki drugi bolji algoritam. Tome je bilo tako jer je program s boljim algoritmom bio prevelik za postojeću fizičku memoriju.[2]

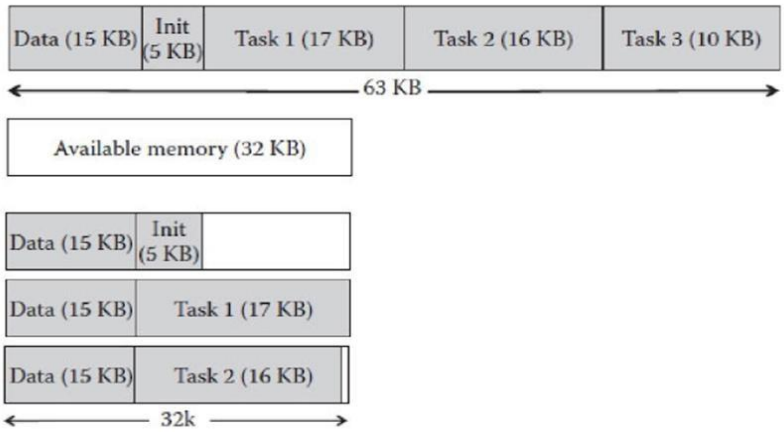
Rješenje za ovaj problem bilo je podijeliti program na dijelove koji ne moraju svi u svakom trenutku biti učitani u fizičku memoriju. Dijelovi programa koji se trenutno koriste, odnosno koje procesor obrađuje, učitaju se u fizičku memoriju, a ostali dijelovi se pohrane u sekundarnoj memoriji, kao što je tvrdi disk. Program se obično dijelio na jedan temeljni dio koji je bio učitan u fizičku memoriju tijekom cijelog trajanja izvođenja programa i dijelove koji su se naizmjence učitavali u preostali dio fizičke memorije. Ovakve dijelove programa zovemo preklopi (engl. *overlays*), a način na koji se koriste nazivamo preklopnim načinom upotrebe fizičke memorije. [4]

Npr., pretpostavimo da je za pokretanje neke aplikacije u nekom računalnom sustavu potrebno 63 KB memorije (pogledati gornji dio slike 3.1.). Međutim, zbog nekih ograničenja, ovo računalo pruža samo 32 KB dostupne memorije. Kako bi se riješio raskorak između fizičkih ograničenja i memorijskih zahtijeva programa, korišteni su preklopi. Sustav radi s osnovnim tablicama i varijablama što zahtijeva 15 KB memorije. Ovaj dio treba biti dostupan svim ostalim dijelovima stoga će biti učitan tijekom cijelog trajanja izvođenja programa. Sam program je podijeljen na četiri neovisna dijela:

- Prvi dio koji zahtijeva 5 KB memorije za pokretanje sustava i zbog toga se koristi samo u početku, prilikom pokretanja
- Drugi dio koji zahtijeva 17 KB prostora
- Treći dio koji zahtijeva 16 KB prostora
- I četvrti dio koji zahtijeva 10 KB prostora

Dijelovi 2, 3 i 4 rade neovisno, ali trebaju pristup osnovnom dijelu. Drugim riječima, ovo znači da dijelovi ne pozivaju jedni druge i ne pristupaju podatcima pohranjenim u drugom dijelu. Kako bi se smanjila količina potrebne memorije, dostupna 32 KB memorije su podijeljena na dva odjeljka. Prvi je statički odjeljak od 15 KB koji je namijenjen za zajedničke podatke dok će se drugi odjeljak od 17 KB koristiti za kod. Drugi odjeljak mora biti dovoljno velik kako bi mogao primiti najveći

raspoloživi preklopni dio koda (drugi dio u ovom primjeru). Prilikom inicijalizacijske faze, prvi dio programa je u učitan u drugi odjeljak dostupne memorije. Kada je inicijalizacija završena, u isti odjeljak će biti učitani drugi dijelovi koda što će ovisiti o ponašanju programa.[3]



Slika 3.1. *Primjer preklopnog načina upotrebe radne memorije*

Preklopni način upotrebe fizičke memorije omogućio je pokretanje programa čiji je virtualni adresni prostor veći od dostupne fizičke memorije. Međutim, ovaj pristup zahtijevao je iscrpnu pripremu programa. Iako je posao zamjene preklopnih dijelova bio automatiziran, posao podjele programa na dijelove i implementaciju mehanizama koji bi odlučivali o redoslijedu učitavanja pojedinih dijelova programa u fizičku memoriju morao je odraditi programer. Dakle, unutar programa moralo se voditi računa o korištenju pojedinih dijelova programa. Ovaj posao oduzimao je programeru puno vremena te je postajao znatno složeniji u više-programskom radu jer su se u fizičkoj memoriji mogli naći preklopni dijelovi više programa. Stoga nije prošlo puno vremena dok se netko nije dosjetio načina kako cijeli posao prepustiti računalu, odnosno automatizirati ovaj posao.[4]

Mehanizam ili metoda koja je razvijena u tu svrhu naziva se VIRTUALNA MEMORIJA. Ovaj mehanizam utemeljen je na ideji preklopnih dijelova, ali uključuje značajno poboljšanje. Kod virtualne memorije program je podijeljen na dijelove fiksne duljine kojima automatski upravljaju hardver i operacijski sustav.[3] Dakle, cijeli posao podjele programa na dijelove i upravljanja memorijom prepušten je računalu. U današnje vrijeme razvijeni su puno veći kapaciteti fizičke memorije, nego što je to bio slučaj u prošlosti, ali i programi postaju sve veći i traže sve više fizičke memorije. Osim toga, u fizičkoj memoriji često se nalazi veći broj programa koji su pokrenuti na računalu, a zajedno nerijetko nadilaze kapacitet fizičke memorije. Tako da virtualna memorija ostaje bitan dio svakog modernog računala.[1]

Mehanizam se zove virtualna memorija jer se programima na raspolaganje daje prividna memorija koja zapravo ne postoji kao fizička memorija. Kao što je već rečeno, fizička memorija je jednodimenzionalni niz koji se sastoji od uzastopnih (susjednih) stanica. Fizička adresa jednoznačno određuje mjesto gdje se određena stanica nalazi u fizičkoj memoriji. Skup ovih adresa koje se direktno odnose na fizičku memoriju, naziva se fizički adresni prostor. S druge strane, svaki program ima svoj virtualni adresni prostor koji se sastoji od niza međusobno poredanih virtualnih adresa. Ove adrese koriste se kao reference na stvarne lokacije u fizičkoj memoriji. Nazivaju se virtualnima jer je moguće, prilikom izvođenja programa, da traženi sadržaj kojeg virtualne adrese referenciraju još nije učitan u fizičku memoriju. Na ovaj način program dobiva iluziju da ima na raspolaganju „neograničenu“ količinu fizičke memorije. Nadalje, stanice u fizičkoj memoriji koje virtualne adrese referenciraju mogu se nalaziti na različitim lokacijama i ne moraju biti uzastopno poredane. Dakle, podjela na virtualni i fizički adresni prostor omogućila je pokretanje programa koji su veći od dostupne fizičke memorije i razvoj virtualne memorije kao mehanizma upravljanja memorijom.

Kako fizička memorija ne razumije virtualne adrese one moraju biti prevedene u fizičke adrese. Za vrijeme izvođenja programa, odnosno kada procesor obrađuje program, posebni dio hardvera prevodi virtualne adrese u fizičke adrese. Kao dio prevođenja, hardver također provjerava je li sadržaj na traženoj adresi učitan u fizičku memoriju. U slučaju da je, pristup će biti izveden. Ako sadržaj na adresi nije učitan u memoriju, hardver će izdati prekid (engl. *interrupt*) kojeg će dohvatiti i obraditi operacijski sustav. Nadalje, odgovornost je operacijskog sustava da učita traženi sadržaj. Za vrijeme učitavanja program je zaustavljen stoga drugi programi mogu koristiti procesor. Samo nakon što je sadržaj na adresi učitan, prekinuti program može nastaviti sa izvršavanjem. U narednoj razradi detaljno će se objasniti ova problematika. Dvije su osnovne metode implementacije virtualne memorije. To su straničenje i segmentacija koje će nadalje biti pojašnjene.[3]

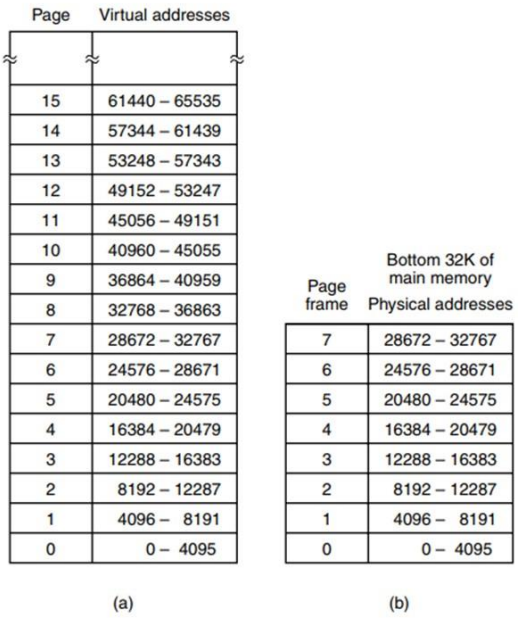
3.1. Straničenje

U računalnim operacijskim sustavima, straničenje (engl. *paging*) je metoda implementacije virtualne memorije gdje računalo pohranjuje i dohvaća podatke u sekundarnoj memoriji za korištenje u glavnoj (radnoj) memoriji. Drugim riječima, ovo je metoda automatizacije preklopnog načina upotrebe radne memorije, onoga koji je prethodno objašnjen. Kao što je već rečeno, osnovna svrha ovog mehanizma je omogućiti pokretanje programa koji su veći od dostupne fizičke

memorije, ali s bitnom razlikom da je cijeli posao provedbe ovog mehanizma prepušten operacijskom sustavu i radi se s dijelovima programa fiksne veličine.[2]

3.1.1. Podjela virtualnog adresnog prostora na stranice i fizičke memorije na okvire

Kod straničenja, cijeli skup virtualnih adresa, one koje su izlaz iz CPU-a (skraćeno od *Central Processing Unit*), podijeljen je na dijelove nazvane stranice (engl. *page*). Svaka stranica je jednake veličine i svaka logička adresa nalazi se na točno jednoj stranici. Analogno, fizička memorija podijeljena je na okvire (engl. *frames*). Skup svih okvira nekog procesa zovemo njegovim radnim skupom (engl. *working set*). U jedan okvir fizičke radne memorije može se smjestiti jedna stranica logičkog adresnog prostora. Svaki okvir veličine je kao jedna stranica. Veličinu stranice zapravo definira veličina okvira, a ona obično iznosi od 1 KB do 8 KB, a najčešće je 4 KB za 32 bitni računalni sustav. Slika 3.2. pokazuje jednu takvu konfiguraciju za relativno jednostavan CPU, koji ima logički adresni prostor veličine 64 KB. U ovom sustavu nalazi se 32 KB fizičke memorije i veličina stranice je 4 KB. Može se vidjeti kako je logički adresni prostor podijeljen na 16 stranica, a fizička memorija na 8 okvira.[5]



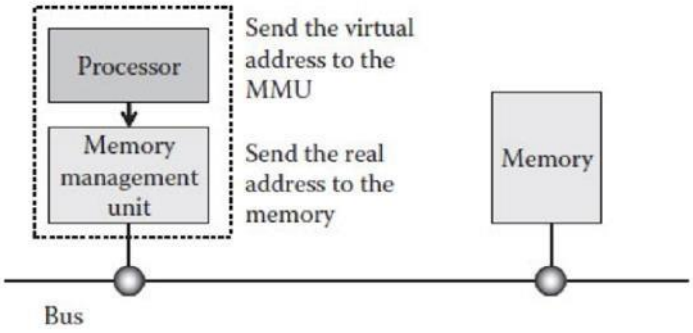
Slika 3.2. *Podjela logičkog adresnog prostora na stranice i fizičke memorije na okvire:*

a) logički adresni prostor i b) fizička memorija

Straničenje pomiče stranice iz *swap* diska (koriste se još i nazivi *page disk i page file*) u okvire fizičke memorije kako bi procesor mogao pristupiti podatcima. *Swap* disk je memorijski prostor na tvrdom disku koji služi za privremenu pohranu podataka koji se izmjenjuju između diska i radne memorije. Bilo koja stranica može zauzeti bilo koji okvir.[6]

3.1.2. Jedinica za upravljanje memorijom

Kako memorija razumije samo fizičke adrese, a ne virtualne, one moraju biti predane prilikom dohvaćanja podataka. Svako računalo s virtualnom memorijom ima uređaj koji prevodi virtualne memorijske adrese u fizičke memorijske adrese. Ovaj uređaj zove se jedinica za upravljanje memorijom (engl. *Memory Management Unit* - MMU). Obično je dio procesora. MMU je odgovoran za upravljanje komunikacijom između procesora i memorije kao i za prevođenje virtualnih adresa koje su pohranjene u instrukcijama u stvarne adrese u memoriji (slika 3.3.).[2]



Slika 3.3. *Prevođenje memorijskih adresa*

3.1.3. Virtualne adrese

Program kojeg procesor izvršava generira virtualne adrese, odnosno one su izlaz iz procesora. Virtualna adresa možda referencira lokaciju u fizičkoj memoriji na kojoj se nalazi varijabla koja je potrebna za izvršenje instrukcije, ili iduća instrukcija koju treba izvršiti. Primjerice u 32-bitnom računalnom sustavu referenca na varijablu koja se nalazi u korisničkom programu bit će u obliku 32-bitne virtualne adrese. Ako se radi sa stranicama od 4 KB, donjih 12 bita predstavljaju adresu

unutar stranice, a gornjih 20 bita predstavljaju broj stranice.[2] Broj bita koji je potreban za prikaz svih stranica može se dobiti tako što se virtualni adresni prostor podijeli sa veličinom stranice. U ovom primjeru to je 2^{32} bajta virtualnog adresnog prostora podijeljeno na 2^{12} bajta (veličina stranice), što daje 2^{20} bajta, a to znači da je potrebno 20 bita za prikaz svih stranica u ovom sustavu. Stranica veličine 4 KB može pohraniti 2^{12} adresa, što znači da je potrebno 12 bita kako bi se mogla prikazati svaka od adresa.[7] Kontrolna jedinica procesora (engl. *Control Unit* - CU), koja je odgovorna za dohvaćanje iduće instrukcije kao i za dohvaćanje traženih operanda, šalje virtualnu adresu u MMU. Ovaj koristi tablicu stranica (engl. *page table*) kako bi preko reference virtualne stranice odredio adresu fizičkog okvira u kojem se nalazi traženi podatak.[2]

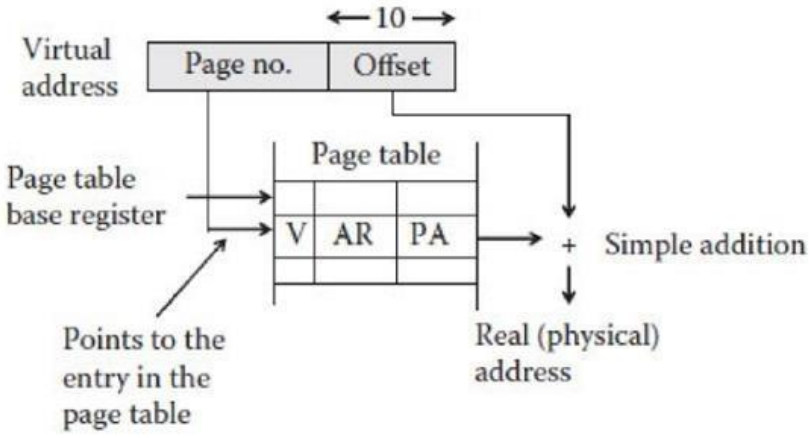
3.1.4. Tablica stranica

Tablica stranica za svaku virtualnu adresu unutar stranice sadrži fizičku adresu (engl. *physical adress* - PA), ali dodatno, svaki ulaz u tablicu stranica (engl. *page table entry*) sadrži informacije o pristupnim pravima (zaštitni bitovi) kao i bit validnosti (engl. *validity bit V*). Bit validnosti još zovemo bit prisutnosti (engl. *presence bit* ili *present/absent bit*). Zaštitni bitovi govore što je moguće raditi sa stranicom, a što nije. U najjednostavnijem obliku, ovo polje sadrži jedan bit, 0 što znači da je moguće čitanje/pisanje, a 1 samo čitanje. Nešto naprednije je korištenje 3 bita, po jedan za svako pravo (čitanje, pisanje i izvršavanje stranice). Valja spomenuti još i modificirajuće i referencirajuće bitove koji također mogu biti dio ulaza u tablicu stranica. Kada se u stranicu piše, hardver automatski postavlja modificirajući bit. Ovaj bit još nazivamo i prljavi bit (engl. *dirty bit*) jer govori o stanju stranice. Kada se stranica uklanja iz fizičke memorije i ako je stanje stranice promijenjeno ona mora biti ponovno učitana na *swap* disk. Ako stranica nije modificirana može ju se jednostavno odbaciti budući je njezina kopija na *swap* disku aktualna.[5] Referencirajući bit se postavlja kada se neka stranica referencira te ima bitnu ulogu kada operacijski sustav odlučuje koju će stranicu izbaciti iz okvira kako bi se oslobodio prostor za stranicu koju treba učitati. Primjerice, stranica koja se malo koristi dobar je kandidat da ju se izbaci iz okvira fizičke memorije.[1]

Svaki program i zapravo svaki proces koji je pokrenut u sustavu ima vlastitu tablicu stranica. Ona se nalazi u memoriji i poseban registar kojeg zovemo bazni registar tablice stranica (engl. *page table base register*) pokazuje na njezin početak. Kada se doda virtualni broj stranice sadržaju ovog

registra, stvara se pokazivač koji se koristi kako bi se za određenu virtualnu adresu pronašla odgovarajuća fizička adresa (PA), a na kojoj se onda nalazi traženi resurs.[3]

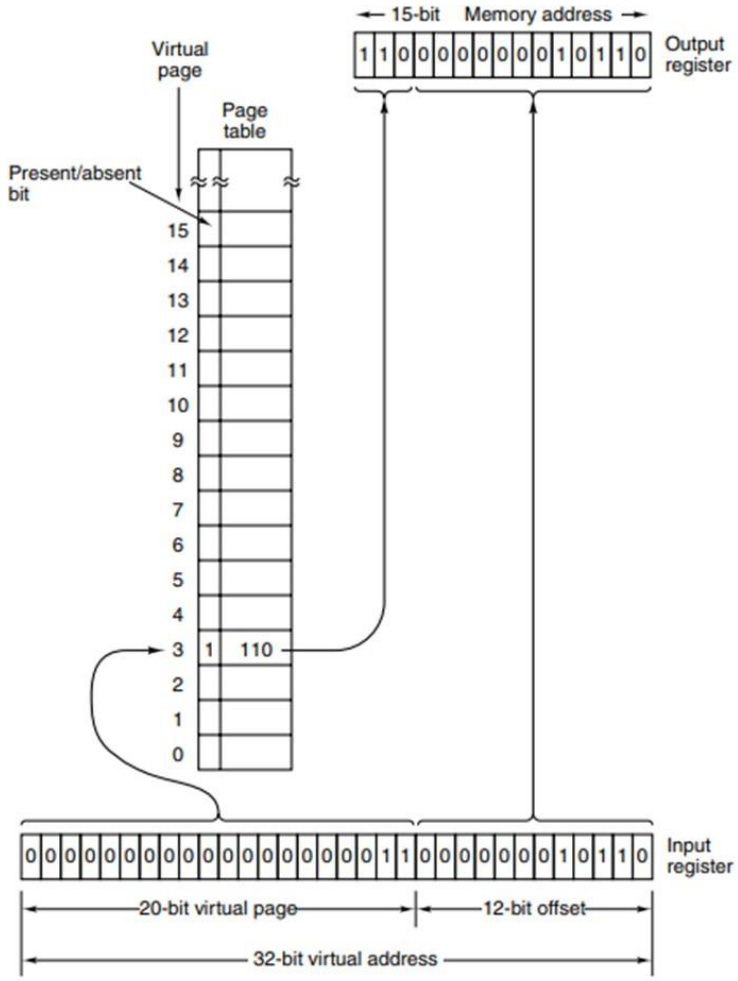
Pogledati sliku 4.3.



Slika 3.4. *Ulaz u tablicu stranica*

3.1.5. Prevođenje virtualnih u fizičke adrese

Kako bi se lakše razumjelo kako radi MMU, odnosno kako se virtualne adrese prevode u fizičke adrese, pogledati primjer na slici 3.5. U ovom primjeru MMU prevodi 32-bitne virtualne adrese u 15-bitne fizičke adrese stoga treba 32-bitni ulazni registar i 15-bitni izlazni registar. Kada MMU primi 32-bitnu virtualnu adresu, on ju podijeli u 20-bitni broj stranice i 12-bitni odmak (engl. *offset*) unutar stranice (zato što su stranice u primjeru 4 KB). Broj stranice se koristi kao indeks, odnosno kao referenca za pristup stranici unutar tablice stranica. Na slici broj stranice je 3 stoga je taj broj izabran za ulaz u tablicu stranica. S obzirom da sve stranice ne mogu u isto vrijeme biti u glavnoj memoriji jer je više stranica nego što ima okvira, prva stvar koju MMU čini je provjera je li referencirana stranica trenutačno u glavnoj memoriji. Tu informaciju dobiva analizirajući bit prisutnosti u tablici stranica. U ovom primjeru bit je 1 što znači da je stranica trenutačno u fizičkoj memoriji. Sljedeći korak uzeti je vrijednost iz tablice stranica i kopirati ju u gornjih 3 bita 15-bitnog izlaznog registra. Tri bita su potrebna jer imamo osam okvira u fizičkoj memoriji koje treba predstaviti. Paralelno s ovom operacijom, donjih 12-bita virtualne adrese su kopirani u donjih 12 bita izlaznog registra, kao što je prikazano. Ovakva 15-bitna adresa sada je poslana u memoriju.[2]



Slika 3.5. *Stvaranje fizičke adrese iz virtualne adrese*

3.1.6. Proces prevođenja adresa na razini računalne arhitekture

Operacijski sustav odgovoran je za upravljanje informacijama koje su vezane uz pojedini proces. Ove informacije nalaze se u upravljačkom bloku procesa (engl. *process control block*), unutar jezgre operacijskog sustava (engl. *kernel*). Radi se o identifikacijskim podatcima procesa, podatcima o stanju procesa i upravljačkim podatcima procesa. Identifikacijski podatci procesa sadrže mapiranje između identifikatora procesa (engl. *process ID*) i bazne adrese tablice stranica (početna adresa tablice stranica u fizičkoj memoriji). Svaki procesor ima bazni registar tablice stranica koji je dostupan jedino operacijskom sustavu. Ovaj ga koristi kako bi pohranio trenutnu

baznu adresu tablice stranica. Prilikom promjene procesa, operacijski sustav jednostavno promjeni baznu adresu tablice stranica u baznom registru tablice stranica. Na ovaj način svaki proces zna koja je njegova tablica stranica. Operacijski sustav je taj koji kreira tablicu stranica kada se program pokrene na računalu.

Procesor posjeduje programsko brojilo koje virtualnu adresu pohranjuje u dva dijela. Gornji bitovi sadrže broj stranice, a donji bitovi sadrže odmak unutar stranice.

MMU prvo pročita baznu adresu tablice stranica iz baznog registra tablice stranica kako bi pristupio tablici stranica. Onda, pročita broj stranice koja je dio virtualne adrese pohranjene u programskom brojilu kako bi pronašao odgovarajući ulaz u tablicu stranica, a u kojem se nalazi broj fizičkog okvira. MMU generira fizičku adresu spajanjem broja fizičkog okvira s odmakom koji je pohranjen u programskom brojilu (drugi dio virtualne adrese). Nakon toga, novonastala fizička adresa se pohranjuje u memorijski adresni registar. Procesor pročita tu adresu kako bi dohvatio instrukciju iz fizičke memorije, koju onda izvršava.[7]

3.1.7. Međuspremnik za prevođenje adresa

Proces prevođenja memorijskih adresa zna biti dugotrajan stoga kako bi se ubrzao osmišljen je priručni međuspremnik za prevođenje adresa (*translation lookaside buffer* - TLB) koji je dio procesorske jedinice za upravljanje memorijom (MMU). Predstavlja priručnu memoriju koja pohranjuje prethodne prijevode virtualnih u fizičke adrese te na taj način smanjuje vrijeme pristupa nekoj korisničkoj memorijskoj lokaciji. Puno je brže dohvatiti fizičku adresu iz TLB-a, nego direktno iz radne memorije. Kada procesor generira virtualnu adresu, MMU ju paralelno proslijedi u tablicu stranica i u TLB. Ako je prijevod pohranjen u TLB-u, on će poslati odgovarajuću adresu u priručnu i radnu memoriju. Ako prijevoda nema, onda će tražiti u tablici stranica, kada generira fizičku adresu, osvježit će stanje TLB-a i sada će se prijevod nalaziti u TLB-u. Kod idućeg pristupa toj adresi, puno će se brže odvijati pristup memoriji jer se neće morati upućivati zahtjev na tablicu stranica u radnoj memoriji.[5] Kada je stranica pronađena u TLB-u, to nazivamo TLB Hit, a ako nije pronađena, kažemo da je to TLB Miss.[7]

3.1.8. Straničenje na zahtjev

Kao što se moglo vidjeti u prethodnom primjeru, bit prisutnosti bio je 1, što je značilo da je stranica prisutna u fizičkoj memoriji pa ju je kontrolna jedinica odmah prebacila u procesor na obradu. Međutim ako tražena stranica nije u memoriji i mora biti učitana iz diska, odnosno bit prisutnosti je 0, MMU izdaje „*Page Fault*“ prekid (engl. *interrupt*) kojeg operacijski sustav registrira, obrađuje te prebacuje stranicu u slobodni okvir. Nadalje, istovremeno upisuje njezinu novu fizičku adresu u tablicu stranica. Također, ako je potrebno pritom se uklanja neku od drugih stranica koje su već učitane u okvire i najmanje se koriste kako bi se otvorio prostor (okvir) za novu stranicu. Ovakav način straničenja dobio je naziv straničenje na zahtjev (engl. *demand paging*). Kontrolu cijelog procesa uključujući i prekida provodi operacijski sustav.[2] Tijekom „*page fault*“ prekida, operacijski sustav mora:

- Pronaći okvir u memoriji gdje će ova stranica biti učitana. To može učiniti alocirajući novi okvir iz tzv. bazena praznih okvira ili ispražnjavajući stranice koje pripadaju drugim aktivnim procesima, uključujući i onoga koji je zatražio stranicu.
- Pronaći stranicu, koja je obično dio izvršne datoteke (engl. *execution file*) programa u izvršavanju, ali operacijski sustav mora otkriti njezinu lokaciju.
- Izdati instrukcije da se učita stranica.
- Za vrijeme dok se stranica učitava, aktivni proces je stavljen na čekanje te operacijski sustav dopušta drugom procesu da koristi procesor.
- Kada stranica završi s učitavanjem, operacijski sustav će proces staviti ponovno u red za izvršavanje pa će se pokrenuti kada dođe njegov red.[3]

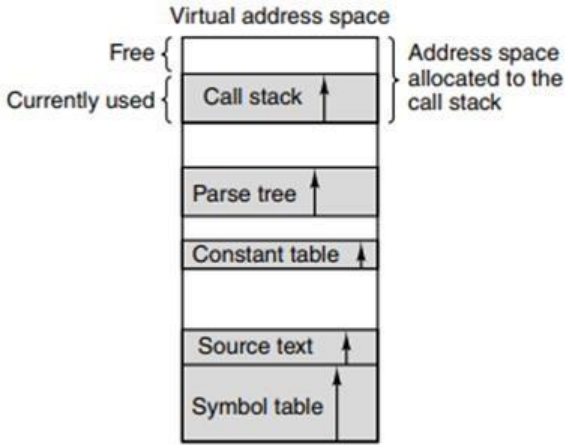
Valja još spomenuti neke od teorijskih strategija zamjene stranica na kojima se bazira praktično ostvarenje postupka odabira stranica koje će biti izbačene iz svog okvira kada to bude potrebno, a to su:

- LRU strategija, koja podrazumijeva izbacivanje stranice koja se u prošlosti najmanje koristila (LRU dolazi od engl. *least recently used*).
- FIFO strategija, koja podrazumijeva izbacivanje stranice koja je najviše vremena provela u radnoj memoriji (FIFO dolazi od engl. *first-in, first-out*).
- Optimalna strategija (OPT), koja podrazumijeva izbacivanje stranice za koju se smatra da se ubuduće neće koristiti.[4]

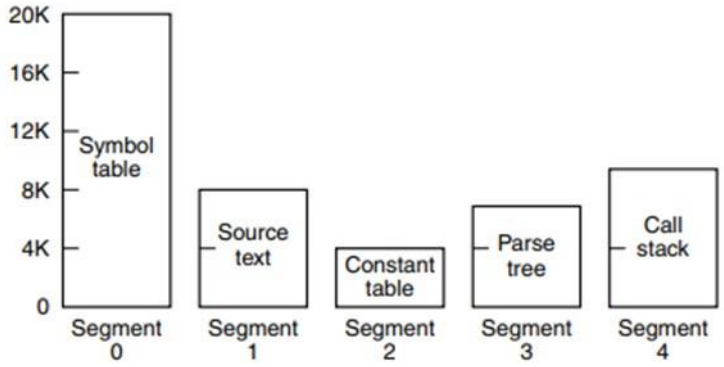
Kod jako ograničenog broja okvira u odnosu na broj stranica i čestog premještanja stranica iz radne memorije na *swap* disk i obrnuto može doći do pojave koju zovemo *trashing*. U ovom slučaju značajno se narušavaju performanse sustava jer procesor puno vremena provodi čekajući na učitavanje potrebnih stranica u memoriju. Također, može se dogoditi da neki zadatci praktično nikad ne dobiju resurse kako bi se izvršili. U ovom slučaju korisnik mora prekinuti neku od aplikacija koje se izvršavaju ako želi da u nekom razumnom vremenu računalo izvrši određene zadatke. Jedino rješenje za ovaj problem je povećati kapacitet radne memorije.[6]

3.2. Segmentacija

Segmentacija (engl. *segmentation*) je metoda implementacije virtualne memorije. Može biti korištena sama za sebe, ali i u kombinaciji sa straničenjem. U svojoj čistoj formi (engl. *pure segmentation*) virtualni adresni prostor programa podijeljen je na mnoštvo neovisnih adresnih prostora koji se nazivaju segmenti (engl. *segments*). Uočavamo osnovnu razliku u odnosu na straničenje gdje je virtualni adresni prostor jednodimenzionalni niz koji ide od nulte adrese do nekog maksimuma, pri čemu adrese slijede jedna za drugom.[2] Za primjer, pogledati sliku 3.6. koja prikazuje jednodimenzionalni adresni prostor i sliku 3.7. koja pokazuje segmentiranu memoriju.



Slika 3.6. *U jednodimenzionalnom adresnom prostoru s rastućim tablicama, jedna tablice može ući u prostor one druge*



Slika 3.7. *Segmentirana memorija dopušta svakoj tablici da raste ili se smanjuje neovisno o drugim tablicama*

3.2.1. Segmentirana memorija

Segmentirana memorija je dvodimenzionalna memorija. Svaki segment sastoji se od linearne sekvence (niza) adresa, od 0 do maksimuma. Različiti segmenti obično imaju različitu duljinu. Nadalje, duljina pojedinih segmenta može se mijenjati tijekom izvršavanja. Općenito, nekoliko je osnovnih tipova segmenata:

- Segment koda (engl. *code segment*), koji se koristi za programske instrukcije. Drugi procesi ne mogu pristupiti (čitati ili pisati) sadržaju ovog segmenta. Samo operacijski sustav i MMU mogu pristupiti njegovom sadržaju.
- Podatkovni segment (engl. *data segment*), koji se koristi za pohranu programskih podataka. Svi podatci u podatkovnom segmentu mogu biti čitani i upisivani.
- Stogovni segment (engl. *stack segment*), koji predstavlja dinamičku memoriju koja se koristi prilikom izvršavanja programa. Dakle, služi za privremenu pohranu podataka neophodnih za izvođenje nekog programa.
- Segment hrpe (engl. *heap segment*), sadrži globalne podatke koji su dostupni programu tijekom izvršavanja.

Kako svaki segment sadrži odvojeni adresni prostor, različiti segmenti mogu rasti ili se smanjivati neovisno, tako da ne utječu jedan na drugoga.[3,7]

Segment je logički entitet u programu, kojeg je programer svjestan te može utjecati na njega.[1] Primjerice, prilikom pisanja programa, programer može znati da se globalne varijable zapisuju u

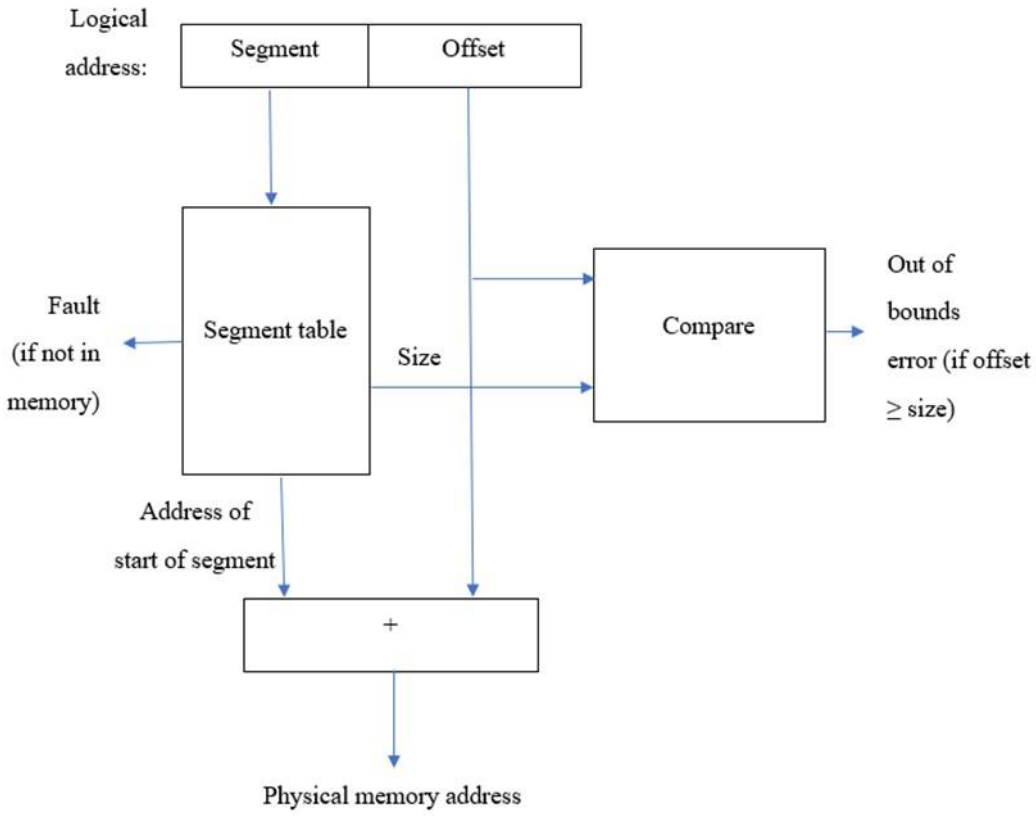
podatkovni segment i da je njegova veličina definirana tijekom prevođenja koda, odnosno ne mijenja se tijekom izvođenja programa. Može također znati da je stog dio memorije koji se koristi za privremenu pohranu podataka tijekom izvođenja programa, npr. lokalnih varijabli unutar funkcije, koje se brišu čim funkcija završi s radom. Prostor na stogu dinamički se alocira i briše tijekom izvođenja programa, a taj zadatak prepušten je sustavu. Dakle, programer se ne mora baviti brisanjem ovog sadržaja. Iako stogovni segment varira u veličini tijekom izvođenja programa, njegov kapacitet zadan je na razini sustava pa ga valja razumno koristiti. Ako se želi alocirati veća količina globalno dostupne memorije, to se može učiniti dinamičkom alokacijom na hrpi, koja nema zadano ograničenje, osim fizičkog kapaciteta radne memorije. Unatoč tome, ipak valja voditi računa da se alocirana memorija, koja se ne koristi ili više nije potrebna obriše, kako bi bilo dovoljno memorije za druge procese koji se izvršavaju. Naprimjer, u C programskom jeziku za dinamičku alokaciju memorije mogu se koristiti funkcije *malloc* i *calloc*. Nadalje, funkciju *realloc* može se koristiti za promjenu veličine alocirane memorije tijekom izvođenja programa, a funkcija *free* može se koristiti za oslobađanje zauzetog prostora. Budući se ovakva memorija dinamički zauzima i briše tijekom izvršavanja programa, veličina hrpe može varirati tijekom izvođenja programa.[8]

3.2.2. Segmentacija u čistom obliku

Kako bi se dohvatio sadržaj pohranjen u segmentima mora se odrediti fizička adresa segmentirane memorije. Prvo program mora dostaviti virtualnu adresu koja se sastoji od dva dijela, a to su: broj segmenta i odmak (engl. *offset*) unutar segmenta. Broj segmenta služi kao ulaz u tablicu segmenata (engl. *segment table*) koja, ako se segment nalazi u memoriji, kao izlaz daje početnu adresu segmenta i njegovu veličinu. S obzirom da segmenti variraju u veličini ove informacije su jako bitne. Ako segment nije u fizičkoj memoriji, generirat će se segmentni prekid (engl. *segment fault*). Operacijski sustav će ga registrirati i učitati traženi segment u memoriju. Kada je segment u memoriji proces prevođenja može se nastaviti. Idući zadatak je usporediti odmak s veličinom segmenta. Ako je odmak veći ili jednak veličini segmenta, što će reći da lokacija nije dio segmenta, generira se pogreška. Ako odmak ima odgovarajuću vrijednost, dodan je na početnu adresu segmenta kako bi se generirala točna fizička memorijska adresa. Kao i kod straničenja, segmentirani MMU također može imati priručni međuspremnik za prevođenje adresa koji se nalazi

na procesorskom čipu (engl. *translation lookaside buffer* - TLB) u svrhu bržeg generiranja početnih adresa segmenata i njihove veličine.[5]

Na slici 3.8. može se vidjeti prevođenje logičke adrese u fizičku adresu koristeći segmentaciju.

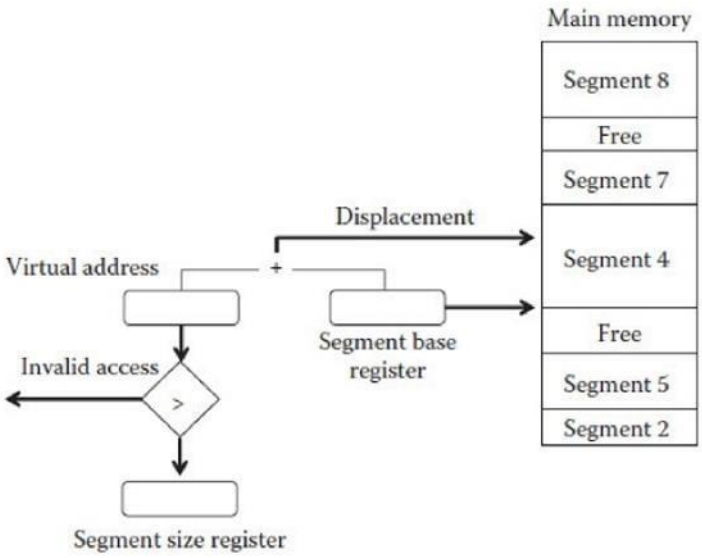


Slika 3.8. *Prevođenje logičke adrese u fizičku koristeći segmentaciju*

Slika 3.8. također dočarava jedan nedostatak segmentacije kao metode implementacije virtualne memorije.

Kod straničenja, broj stranice je poslan u tablicu stranica (i TLB) kako bi se proizveo broj okvira. Ova je vrijednost onda spojena s odmakom (engl. *offset*) kako bi se proizvela fizička adresa. U segmentaciji, početna adresa koju generira tablica segmenata ili TLB je zbrojena s odmakom. Radi se o procesu koji je dugotrajniji nego spajanje, što je jedan nedostatak segmentacije. Nadalje, MMU mora ovo učiniti za svaki pojedini pristup memoriji. Tome je tako jer se broj segmenta zapravo nalazi u baznom registru (engl. *base register*) odvojen od logičke adrese.

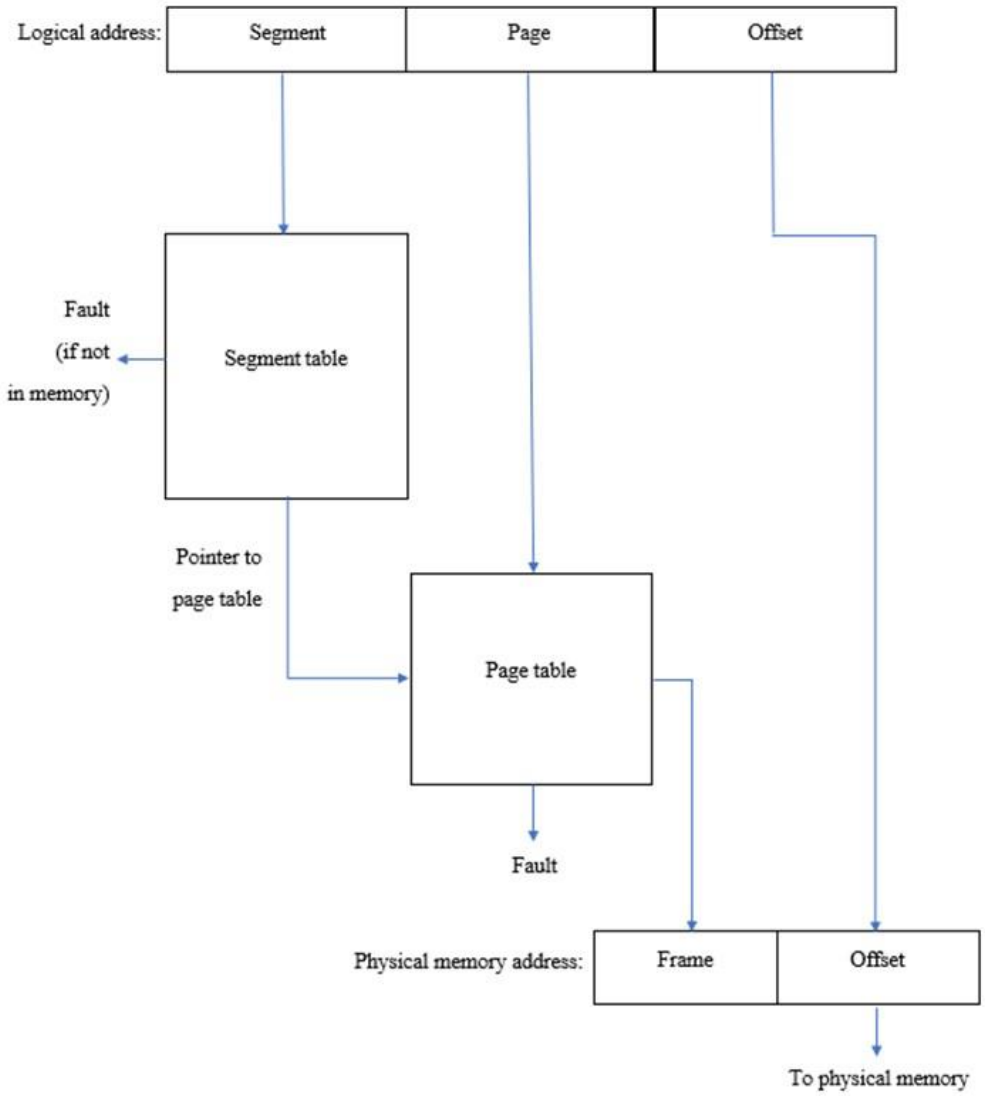
Logička adresa sadrži odmak i veličinu pojedinog segmenta što je određeno proširenje na dosadašnje objašnjenje.[5] Pogledati sliku 3.9.



Slika 3.9. *Segmentirana memorija i proces prevođenja adresa*

3.2.3. Kombinacija segmentacije i straničenja

Prethodno je bila riječ o segmentaciji u njenom čistom obliku, međutim moguće je kombinirati segmentaciju i straničenje. To se čini tako što se kreiraju segmenti iz stranica, odnosno segmenti se podijele na stranice, što je bolje nego jedan kontinuirani blok memorije, o čemu će nešto više biti rečeno u idućem poglavlju. U ovom slučaju, logička adresa podijeljena je na tri dijela: broj segmenta, broj stranice, i odmak. Broj segmenta ulaz je u tablicu segmenata, kao i kod čiste segmentacije. Međutim, umjesto početne adrese, tablica segmenata na izlazu daje pokazivač na tablicu stranica. Ako se segment ne nalazi u fizičkoj memoriji, generira se prekid. Tablica stranica onda daje odgovarajući broj okvira u fizičkoj memoriji ili generira prekid ako stranica nije u memoriji. Kada su svi prethodni koraci uspješno izvršeni, broj okvira se spaja s odmakom kako bi se proizvela fizička memorijska adresa.[5] Pogledati sliku 3.11.



Slika 3.11. *Prevođenje logičkih adresa u fizičke koristeći segmentaciju sa straničenjem*

3.2.4. Segmentacija na razini računalne arhitekture

Virtualni adresni prostor procesa definiran je unutar tablice segmenata. Ona se nalazi pohranjena u baznom registru tablice segmenata (engl. *segment table base register*) koji se nalazi unutar procesora. Tablica segmenata sadrži skup brojeva segmenata, što uključuje informacije koje su potrebne za prevođenje virtualnih adresa u odgovarajuće fizičke adrese, a to su baza segmenta (engl. *segment base*), ograničenje (engl. *limit*) i zaštitne informacije pojedinog segmenta.

Baza segmenta je početna adresa segmenta u fizičkoj memoriji i procesor ju koristi kao referencu koja pokazuje na odgovarajući segment. Ograničenje definira veličinu segmenta koji je

promjenjiv, a određuje ga program prilikom prevođenja i izvršavanja. Dakle, tablica segmenata mapira virtualnu adresu u kontinuirani dio fizičke memorije, a za to koristi baznu adresu i ograničenje.

Može se uočiti kako bazni registar tablice segmenata sadrži samu virtualnu memoriju, odnosno tablicu segmenata, što je razlika u odnosu na bazni registar tablice stranica koji sadrži adresu virtualne memorije, odnosno tablice stranica.

Programsko brojilo generira virtualnu adresu s gornjim bitovima koji sadrže broj segmenta i donjim bitovima koji sadrže odmak unutar segmenta. Broj segmenta predstavlja ulaz u tablicu segmenata, a gdje će pronaći informacije koje su potrebne za generiranje odgovarajuće fizičke adrese. Odmak se koristi kako bi se provjerilo da fizička memorija koja se zauzima za određeni segment ne prolazi definirano ograničenje te kao dio fizičke adrese, prilikom njezina generiranja. Dakle, MMU pročita broj segmenta unutar programskog brojila, pronalazi odgovarajući ulaz u tablicu segmenata te dobiva informaciju o baznom segmentu (baznoj adresi). Zatim se unutar programskog brojila bazna adresa spaja s odmakom kako bi se dobila fizička adresa, a koja se onda pohranjuje u memorijski adresni registar (MAR). Procesor dohvaća instrukciju na toj memorijskoj adresi te ju izvršava.

Međutim, segmentacija u svojem čistom obliku, koja je prethodno objašnjena, uzrokuje vanjsku fragmentaciju pri čemu dolazi do curenja memorije. To se događa jer slobodni memorijski prostori koji ostaju između segmenata nerijetko nisu dovoljno veliki kako bi se u njih mogao učitati novi segment.

Kako bi se riješio ovaj problem, koristi se kombinacija segmentacije i straničenja na zahtjev. U ovom slučaju svaki segment sastavljen je od niza stranica koje će se prema potrebi učitavati i uklanjati iz segmentirane memorije, što rješava problem vanjske fragmentacije, ali ponovno uvodi problem unutarnje fragmentacije. Problem je kod segmentacije što segmenti zauzimaju fiksne lokacije u memoriji te su relativno veliki, dok su stranice u pravilu manje veličine i mogu se učitavati na različite lokacije u memoriji, odnosno mogu popunjavati slobodne memorijske prostore koji su razbacani u fizičkoj memoriji. O ovoj problematici reći će se nešto više u zadnjem poglavlju. Svaki proces ima svoju tablicu segmenata, a svaka tablica segmenata ima odgovarajuću tablicu stranica. Tablica segmenata ima isti sadržaj kao kod čiste segmentacije, ali u ovom slučaju bazna adresa referencira baznu adresu tablice stranica u fizičkoj memoriji, a ograničenje se odnosi na broj virtualnih stranica za pojedini segment. Ulaz u tablicu stranica sadrži informaciju o odgovarajućim okvirima fizičke memorije. Zaštita čitanja/pisanja može biti pohranjena u tablici

segmenata kao i u tablici stranica ovisno je li zaštita potrebna na razini čitavog segmenta ili na razini stranice. Tablica segmenata nalazi se u baznom registru tablice segmenata unutar procesora, a tablica stranica nalazi se u fizičkoj memoriji. S obzirom da segmenti mogu imati različitu veličinu, imaju i različit broj stranica.

Operacijski sustav za proces koji je pokrenut na računalu unosi njegovu tablicu segmenata u bazni registar, a ovaj sadrži bazne adrese tablice stranica. Programsko brojilo generira virtualnu adresu. Gornji bitovi ove adrese predstavljaju broj segmenta, slijede bitovi koji predstavljaju broj stranice i na kraju donji bitovi koji predstavljaju odmak unutar stranice.

MMU pročita broj segmenta u programskom brojilu te na temelju njega pronalazi odgovarajući ulaz u tablicu segmenata. U tablici segmenata pronalazi baznu adresu odgovarajuće tablice stranica u fizičkoj memoriji. Onda, broj stranice iz programskog brojila koristi kako bi pristupio odgovarajućem ulazu u tablici stranica, a tamo će pronaći broj fizičkog okvira. Ovaj broj će potom spojiti s odmakom iz programskog brojila kako bi generirao fizičku adresu okvira. Ovu adresu će pohraniti u memorijski adresni registar kako bi procesor mogao dohvatiti instrukcije u memoriji te ih izvršiti.[7]

3.3. Virtualna memorija u Windows 10 operacijskom sustavu sa 64-bitnom računalnom arhitekturom

Računalni sustavi koji rade sa 64 bita mogu teoretski pokriti fizički adresni prostor od 2^{64} bajta. Ovakvi sustavi mogu adresirati puno više fizičke memorije nego što je danas dostupno pa su smanjeni i rade sa 48 bita odnosno mogu pokriti fizički adresni prostor od 2^{48} bajta, što je 256 TB. Dakle, u ovom slučaju ulaz u tablicu stranica predstavljen je sa 48 bita, što je 6 bajta. Neka je veličina stranice 4 KB. Veličinu tablice stranica dobiva se tako što se adresni prostor tablice stranica podijeli s veličinom stranice i onda pomnoži s ulazom u tablicu stranica. U ovom slučaju to je:

Veličina tablice stranica= $2^{48} / 2^{12} * 6$ bajta= 384 GB

Ovo je veličina tablice jednog procesa koja je neovisna o stvarnoj veličini procesa.

Tablica stranica bila je nešto praktičnije veličine kod 32-bitnog sustava, ali pretpostavlja ogromnu vrijednost za 64-bitni sustav, a to narušava sami smisao virtualne memorije. Posebni mehanizmi kao što su više-razinska tablica stranica i invertirana tablica stranica osmišljeni su kako bi smanjili

tablicu stranica u 32-bitnom sustavu zbog malog kapaciteta radne memorije. U 64-bitnom sustavu ovi mehanizmi postali su ključni za smanjivanje ogromne veličine tablice stranica, iako se kapacitet radne memorije značajno povećao.

Ako se u gornjem primjeru koristi odmak od 21 bita, dobiva se veličina stranice od 2^{21} bajta što je jednako 2 MB.

U ovom slučaju veličina tablice stranica iznosit će:

Veličina tablice stranica= $2^{48} / 2^{21}$ * 6 bajta= 768 MB

Veličina ove tablice već je puno prihvatljivija. Međutim, pojavit će se velika unutarnja fragmentacija. Što je stranica veća, bit će više neiskorištenog prostora u najgornjoj stranici segmenta. U sustavima bez segmentacije unutarnja fragmentacija je još veća. Otkriveno je da veličina stranice u rasponu od 4 do 8 KB pruža najbolji omjer između veličine tablice stranica i unutarnje fragmentacije. Ako pretpostavimo da je u sustavu pokrenuto 100 procesa, a svaki ima 5 segmenata i 50% najgornjeg okvira segmentirane memorije je neiskorišteno, onda će približna unutarnja fragmentacija za veličinu stranice od 4 KB biti:

Unutarnja fragmentacija= (0.5*4KB)*5*100= 1MB,

a za veličinu stranice od 2 MB bit će:

Unutarnja fragmentacija= (0.5*2MB)*5*100= 500MB.

Veličina stranice postavljena je unutar hardvera procesora, a ovisi o arhitekturi sustava i veličini instalirane radne memorije.

Tablica stranica postaje ogromna u 64-bitnom sustavu pa adresiranje kombinacijom segmentacije i straničenja nije praktično u sustavima koji imaju više od 32 bita. Stoga, 64-bitni procesor pruža segmentaciju u 32-bitnom modu, ali ne i u 64 bitnom modu.[7]

3.3.1. Više-razinske tablice stranica

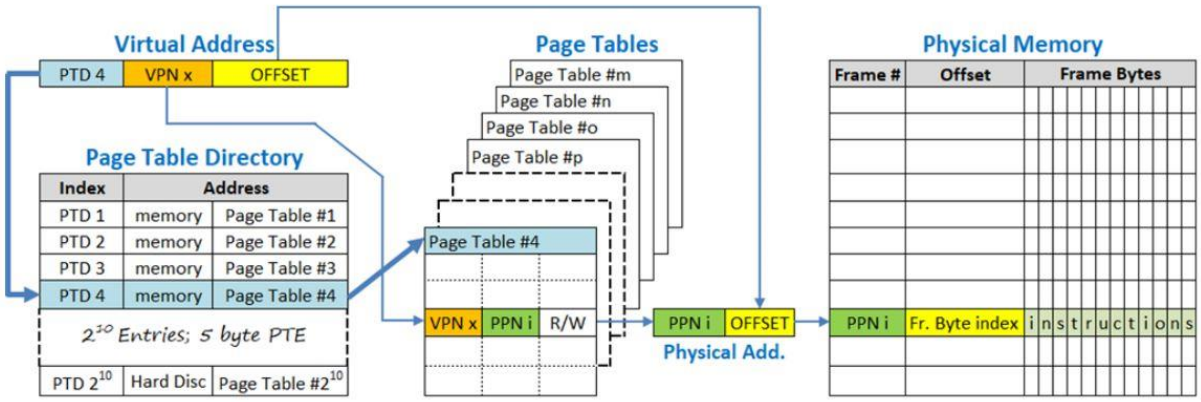
Svrha je više-razinske tablice razdvojiti linearnu tablicu stranica u više tablica stranica koje imaju istu veličinu stranica, ali manji broj stranica po tablici pa će na taj način veličina tablice stranica biti manja. Zbroj veličina svih manjih tablica stranica odgovara veličini jedne velike tablice stranica iz koje su ove nastale. Međutim, svaka od ovih manjih tablica stranica predstavlja proces

pa operacijski sustav može selektivno učitavati tablice stranica u memoriju, ovisno o dostupnosti memorije, te izbaciti tablicu stranica s tvrdog diska kada proces postane neaktivan ili ugašen.

Više-razinske tablice stranica su raspoređene u hijerarhijsku skupinu. Svaki ulaz u tablicu stranica identificira drugu tablicu stranica niže u hijerarhiji. Na ovaj način stvara se struktura stabla tablice stranica. Zadnja razina u hijerarhiji sadrži tablice stranica procesa, dok one više u hijerarhiji pomažu identificirati tablicu procesa na kraju stabla.

Povećavanjem broja razina povećavamo broj tablica na kraju stabla, ali one imaju manju veličinu. Na ovaj način, operacijskom je sustavu lakše alocirati tablice stranica u fizičku memoriju. Međutim, veći broj razina također povećava broj indeksiranja, koji je potreban da bi se identificirala tablica stranica procesa, što postaje manje efikasno. Međutim, tu uskače TLB koji ubrzava indeksiranje tako što posprema česte prijevode adresa tablica.

Razmotrimo primjer s 2-razinskom tablicom stranica u 64-bitnom sustavu s veličinom stranice od 4 KB. Pogledati sliku 3.12.



Slika 3.12. *Prevođenje virtualnih adresa u fizičke koristeći više-razinsku tablicu stranica*

Adresni prostor podijeljen je u 3 polja: pokazivač u direktorij tablice stranica (engl. *Page Table Directory* -PTD) koji sadrži bazne adrese tablica stranica procesa (gornji bitovi), broj stranice koji pokazuje na odgovarajući ulaz u pojedinoj tablici stranica (srednji bitovi) i odmak unutar stranice (donji bitovi).

Kao u 32-bitnoj arhitekturi, neka svaki proces ima tablicu stranica s 20-bitnim brojem stranice i 12-bitnim odmakom i dodatno imamo 10-bitni direktorij tablice stranica koji će predstaviti 2^{10} tablica procesa, što je jednako 1024 tablica procesa.

Polje virtualnih adresa u direktoriju tablice stranica, polje virtualnih adresa u tablici stranica i odmak zajedno sudjeluju u generiranju cjelovite virtualne adrese u programskom brojilu.

MMU koristi gornjih 10 bitova virtualne adrese kao pokazivač u direktorij tablice stranica i određuje baznu adresu tablice stranica. Onda koristi idućih 20 bita virtualne adrese kao pokazivač u tablicu stranica gdje pronalazi adresu okvira u fizičkoj memoriji. Najnižih 12 bita odmaka u programskom brojilu koristi se kao pokazivač u fizički okvir kako bi se pronašla lokacija instrukcije.

Direktorij tablice stranica cijelo vrijeme mora biti prisutan u fizičkoj memoriji jer je on početna točka za sve ostale tablice koje će se prema potrebi onda učitavati u fizičku memoriju i iz nje uklanjati.

Dakle, u slučaju više-razinskih tablica jedna virtualna adresa je pokazivač na drugu virtualnu adresu, dok se ne dođe do fizičke adrese na zadnjoj razini.[7]

Windows 10 operacijski sustav koji radi na 64-bitnoj arhitekturi u praksi koristi 4-razinsku tablicu stranica kako bi na efikasan način raspolagao virtualnom memorijom. Kao što je rečeno, 64-bitni adresni prostor iz praktičnih razloga smanjen je na 48 bita. Uglavnom se koristi veličina stranice od 4 KB (2^{12} bajta). Prema tome potrebno je 12 bita za prikaz svih adresa unutar stranice. Ako oduzmemo 12 bita od 48 bita ostaje nam 36 bita za prikaz 4 tablice stranica. Ovih 36 bita ravnopravno je raspodijeljeno pa dobivamo po 9 bita za indeksiranje pojedine tablice. Ovo znači da se može pokriti 2^9 (512) ulaza za svaku tablicu. Svaki ulaz u tablicu veličine je 8 bajta (jer radimo sa 64-bitnom arhitekturom).

Na temelju rečenog može se zaključiti da se virtualna adresa u ovom slučaju sastoji od 12 bita odmaka koji se u programskom brojilu nalaze na najnižim bitovima, a onda idu bitovi koji redom predstavljaju tablice P1, P2, P3 i P4 koja je predstavljena najgornjim bitovima. Kako bi se prevela virtualna adresa, procesor čita početnu adresu P4 tablice koja se nalazi u tzv. CR3 registru što je zapravo bazni registar tablice stranica. Zatim koristi indeksiranje kako bi prošao kroz hijerarhiju tablica. Virtualna adresa daje informaciju o odmacima, a ulazi pokazuju na početnu adresu iduće tablice. Ulaz u zadnjoj tablici daje početnu adresu stranice i u kombinaciji sa 12 bita odmaka

dobiva se konkretna fizička adresa. Na ovaj način, radi se s više manjih tablica, umjesto s jednom velikom koja bi zauzimala ogroman prostor u radnoj memoriji.[9,10]

Međutim, u memoriji se može naći jako puno procesa istovremeno, što opet dovodi do problema zauzimanja velikog kapaciteta radne memorije na tablice stranica jer svaki proces zahtijeva svoju tablicu. Ovaj problem postaje posebice izražen na poslužiteljskim računalima koja obično rade s ogromnim brojem procesa. U tu svrhu osmišljena je invertirana tablica stranica. Ova tablica sadrži jedan ulaz u tablici stranica za svaki okvir u radnoj memoriji. Prema tome broj ulaza u tablici stranica invertirane tablice stranica smanjuje se na broj okvira fizičke memorije te se jedna tablica stranica koristi kako bi se prikazale informacije o tablicama više procesa. Na ovaj način, uklanja se potreba za učitavanjem pojedine tablice stranica svih procesa te se učitava jedna stranica koja pohranjuje informacije o svim procesima. Ova tablica zove se invertirana jer se za indeksiranje koristi broj okvira umjesto virtualnog broja stranice. Ova tema bit će detaljno objašnjena u nastavku.[11]

3.3.2. Invertirana tablica stranica

Invertirana tablica stranica je strategija čiji je smisao smanjiti veličinu tablice stranica tako što se definira tablica stranica gdje se adresni prostor odnosi na fizički adresni prostor radne memorije umjesto na virtualni adresni prostor.

Ovo ima smisla na poslužiteljskim sustavima koji imaju veliku količinu instalirane radne memorije kako bi se zadovoljile potrebe svih procesa. Dok je normalna tablica stranica indeksirana koristeći broj stranice kako bi se pronašao ulaz u tablicu stranica koja onda sadrži podatke o broju okvira, invertirana tablica stranica je indeksirana koristeći broj okvira kako bi se pronašao ulaz u tablicu stranica koji sadrži broj stranice te na ovaj način ograničava virtualni adresni prostor koji referencira fizički adresni prostor.

U normalnoj implementaciji virtualne memorije, svaki proces ima svoju tablicu stranica gdje svaka tablica stranica ima svoj skup virtualnih adresa, pa je virtualni adresni prostor promjenjiv ovisno o broju pokrenutih procesa. Kod invertirane tablice stranica, virtualni adresni prostor je fiksiran jer broj okvira indeksira pojedinu tablicu stranica koju dijele svi pokrenuti procesi čiji su brojevi stranica nasumično raspoređeni u tablici. Alociranje procesa u kontinuirane stranice dovest će do unutarnje fragmentacije same tablice stranica.

S obzirom da skup virtualnih adresa mora biti jednak za sve procese koji su fiksirani u programskom brojilu, identifikator procesa (PID) se traži u svakom ulazu u tablicu stranica invertirane tablice stranica kako bi se identificirao broj stranice koji odgovara određenom procesu jer procesi više ne mogu biti razlikovani pojedinom baznom adresom tablice stranica. PID je veličine 2 bajta i dio je kontrolne informacije procesa u ulazu u tablicu stranica.

Dijeljeni procesi nisu mapirani unutar skupa virtualnih adresa za pojedini proces, ali su izvršavani poziv-povrat izrazom prema dijeljenom procesu koji postoji u invertiranoj tablici stranica.

Invertirana tablica stranica, odnosno polje, prvo se stvara s brojem okvira kao indeksom, koji je onda modificiran sa PID-om i brojem stranice, i konačno okviri radne memorije su kreirani u fizičkoj memoriji u indeksiranom nizu brojeva okvira unutar invertirane tablice.

Izračunajmo veličinu invertirane tablice stranica za 64-bitni sustav sa ulazom u tablicu stranica koji se sastoji od 40 adresnih bitova (5 bajta) + 2 bajta za ID procesa + 1 bajt za kontrolne informacije, za veličinu stranice od 4 KB i 32 GB radne memorije.

Adresni prostor= 32 GB radne memorije= 2^{35} bajta

Ulaz u tablicu stranica= 5+2+1 bajta= 8 bajta

Veličina stranice = 2^{odmak} =2^{12} bajta=4 KB

Veličina tablice stranica= (adresni prostor/veličina stranice)*ulaz u tablicu stranica= (2^{35}/2^{12})*8 bajta

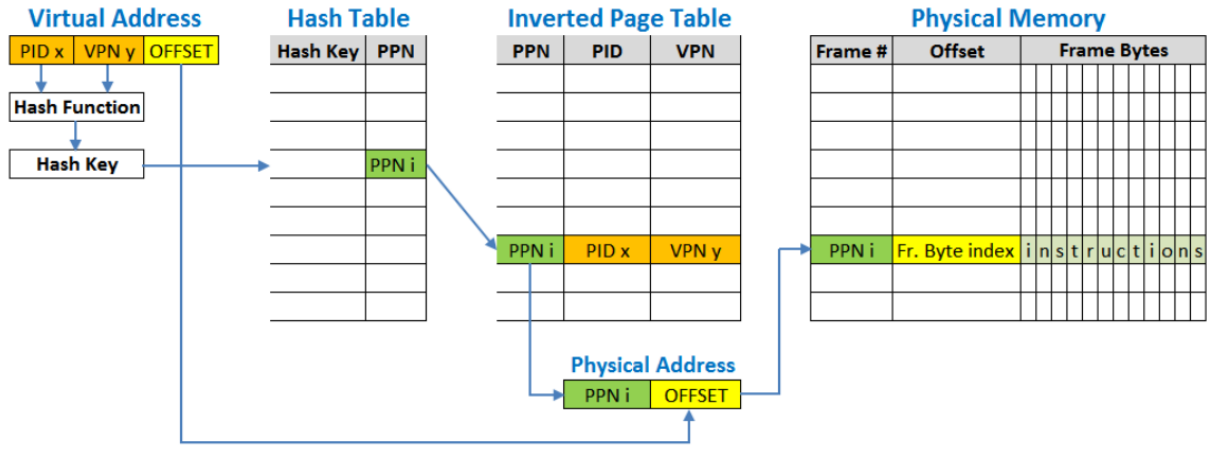
Veličina tablice stranica= 2^{23}*8= 64 MB što je samo 0.2 posto od 32 GB radne memorije

Invertirana tablica stranica je polje čiji indeksi odgovaraju adresama okvira u fizičkoj memoriji, a ulazi u tablicu stranica su reverzno mapiranje u virtualnu adresu stranice procesa koji posjeduje okvir. Ovo ne mijenja princip iza operacija virtualne memorije, gdje virtualna adresa služi kao memorijski pokazivač u instrukcijama operacijskog sustava kako bi se pristupilo fizičkoj memoriji.

Dakle, umjesto indeksiranja u normalnoj tablici stranica kroz broj stranice generiran u programskom brojilu, broj stranice i njegov odgovarajući PID moraju se pronaći u ulazu invertirane tablice stranica kako bi se dobio broj okvira. Vrijeme provedeno tražeći PID i broj stranice prilično je veliko i značajno utječe na performanse sustava.

Funkcija traženja eliminirana je koristeći *hash* tablicu što je niz čiji indeks je formiran tako što se PID i broj stranice pretvaraju u *hash* ključ, korištenjem *hash* funkcije, dok ulazi u polje odgovaraju broju okvira. Stoga, *hash* tablica je u biti normalna tablica stranica izvedena iz invertirane tablice stranica. Vrijeme i prostor koji se potroše tijekom kreiranja dodatne *hash* tablice značajno nadilaze gubitak performansi u odnosu na izravno korištenje invertirane tablice stranica.

Zadatak se izvršava kada operacijski sustav promjeni PID u procesorskom registru, što resetira broj stranice u programskom brojilu. Procesor koristi *hash* funkciju kako bi generirao *hash* ključ iz PID-a i broja stranice koji će služiti kao indeks u *hash* tablicu . Broj okvira pohranjen na ovom indeksu *hash* tablice koristi se kako bi se dalje indeksiralo u invertiranu tablicu, odnosno u njezin ulaz. Zatim, broj okvira se koristi za generiranje fizičke adrese spajanjem s odmakom u programskom brojilu. Ova fizička adresa onda je korištena za dohvaćanje procesnog koda i podataka za izvršavanje. Pogledati sliku 3.13.



Slika 3.13. *Prevođenje virtualnih adresa u fizičke koristeći invertiranu tablicu stranica*

Ako se ne pronađe poklapanje između procesnih podataka i ulaza u tablicu stranica, koristi se tehnika rezolucije *hash* sudara kako bi se odredio broj fizičkog okvira, tako što se povezuju *hash* ulazi koji pokazuju na druge ulaze u *hash* tablici. Čisto traženje ulaza u tablicu stranica u invertiranoj tablici stranica može se alternativno nametnuti.

Hash tablica eliminira potrebu pretrage tablice, ali pod cijenu dvostrukog indeksiranja, jednoga u *hash* tablici i drugoga u tablici stranica. Korištenje TLB-a je potrebno kako bi se poboljšale performanse invertirane tablice stranica tako što se TLB koristi kao prva razina mapiranja virtualne

u fizičku adresu. Prilikom TLB promašaja, *hash* stranica se koristi kao mapiranje sekundarne razine (mapa sekundarne razine), i na koncu *page handler* operacijskog sustava je pokrenut kako bi riješio *page fault*.

Sve ovo napravljeno je kako bi se osiguralo da se potroši najmanja količina fizičke memorije na tablicu stranica koja ima upravo zadatak upravljanja ograničenim kapacitetom fizičke memorije.[7]

4. USPOREDBA STRANIČENJA I SEGMENTACIJE

U prethodnom tekstu objašnjene su metode straničenja i segmentacije. Osim očitih razlika te nekih prednosti i nedostataka koje se moglo uočiti u dosadašnjoj razradi postoje i neke druge koje će sada biti obrađene. Naposljetku, sve će biti sumirano kako bi se dobio kompaktan uvid u osnovne sličnosti i razlike obje metode.

Segmentirana memorija ima drugih prednosti osim do sada spomenutog pojednostavljenja upravljanja strukturama podataka koje se povećavaju ili smanjuju. Ako svaka procedura zauzme odvojeni segment, s adresom nula kao svojom početnom adresom, povezivanje procedura koje su zasebno prevedene (engl. *compiled*) je značajno pojednostavljeno. Nakon što su sve procedure koje su dio programa prevedene i povezane, poziv procedure u segmentu X koristit će dvodijelnu adresu (X, 0) kako bi se adresirala riječ 0 (ulazna točka). Ako je procedura u segmentu X zatim promijenjena i ponovno prevedena, druge procedure ne moraju biti promijenjene jer početne adrese nisu promijenjene. U slučaju jednodimenzionalne memorije, procedure su smještene tijesno jedna uz drugu, bez adresnog prostora između njih. Posljedično, promjena veličine jedne procedure može utjecati na početnu adresu druge, neovisne procedure. Ova situacija zahtijeva promjenu svih procedura koje pozivaju bilo koju od pomaknutih procedura kako bi se pripojile njihove nove početne adrese. Ako program sadrži stotine procedura, ovaj proces može se značajno zakomplicirati.

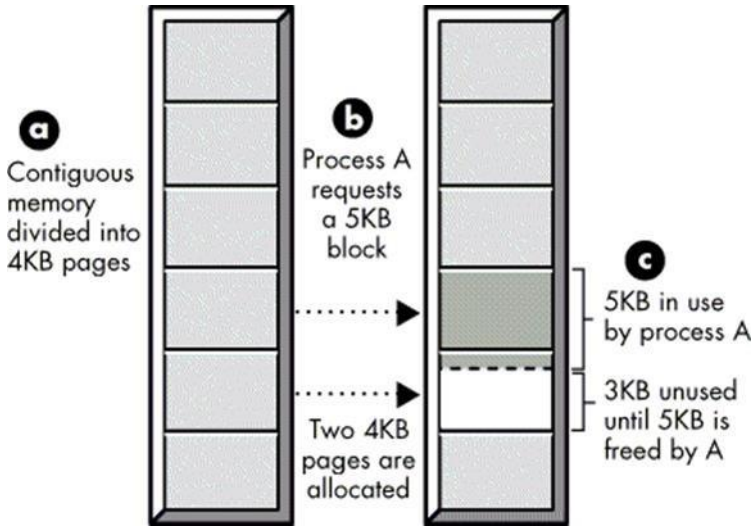
Segmentacija također olakšava dijeljenje procedura ili podataka između različitih procesa. Klasični primjer je dijeljena biblioteka (engl. *shared library*). U segmentiranim sustavima, grafička biblioteka može biti stavljena u segment i podijeljena između više procesa, što isključuje potrebu da bude u adresnom prostoru svakog procesa. Iako je moguće imati dijeljene biblioteke u čistim straničnim sustavima stvar se značajno komplicira. Zapravo ovi sustavi to rade simulirajući segmentaciju.

Kako svaki segment formira logički entitet kojeg je programer svjestan, kao što je procedura, polje ili stog, različiti segmenti mogu imati različite vrste zaštita. Proceduralni segment može biti određen samo za izvršavanje, zabranjujući pokušaje čitanja ili pisanja u njega. Polje realnih brojeva može biti određeno za čitanje/pisanje, ali ne izvršavanje. Takva zaštita korisna je prilikom dohvaćanja programskih pogrešaka (engl. *errors*). Nadalje, ova zaštita ima smisla kod segmentirane memorije, ali ne i kod jednodimenzionalne stranične memorije jer je korisnik svjestan što je u pojedinom segmentu. Npr. segment normalno neće sadržavati proceduru i stog,

nego jedno ili drugo. Kako svaki segment sadrži samo jednu vrstu objekta, segment može imati prikladnu zaštitu za taj specifični tip objekta.[1]

Iako je već spomenuto u razradi segmentacije, nije na odmet ponoviti u kontekstu usporedbe kako je proces prevođenja virtualne adrese u fizičku značajno vremenski dugotrajniji kod segmentacije, nego kod straničenja, zbog već opisanih razloga, što je veliki nedostatak segmentacije.

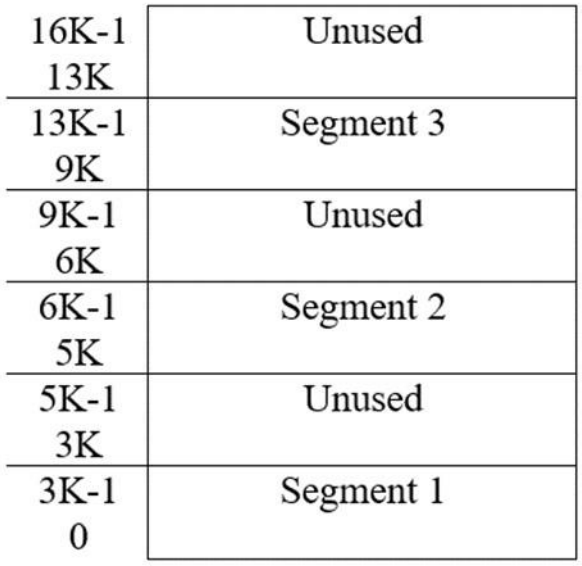
S obzirom da segmenti mogu imati različite veličine, ova metoda ima i svoje prednosti i nedostatke u usporedbi sa straničenjem. Uzmimo primjer straničenja za relativno jednostavni CPU. Kod implementacije memorije straničenjem, svaka stranica veličine je 4 KB. Program veličine 5 KB bi zahtijevao od MMU-a da alocira dvije stranice u memoriji iako bi druga stranica koristila samo jednu od njezinih 4 KB lokacija, odnosno 3 KB u drugoj stranici bilo bi neiskorišteno. Ovo zovemo unutarnjom fragmentacijom (engl. *internal fragmentation*). Pogledati sliku 4.1.



Slika 4.1. *Unutarnja fragmentacija kao posljedica straničenja*

Ako se koristi segmentacija, segment veličine točno 5 KB može biti alociran i na taj način bi se izbjegao ovaj problem. Segmentacija međutim ima problem koji se zove vanjska fragmentacija. Uzmimo u obzir situaciju prikazanu na slici 4.2. Vidimo tri segmenta koji su smješteni u memoriji i 8 KB praznog prostora. Međutim, prazan prostor je podijeljen tako da segment veći od 3 KB ne može biti učitan u memoriju bez pomicanja ili uklanjanja jednog od trenutačno učitanih segmenata.

Iako ovo može biti učinjeno, značajno smanjuje performanse sustava jer se previše vremena gubi na premještanje i učitavanje segmenata.



Slika 4.2. *Vanjska fragmentacija u fizičkoj memoriji kao posljedica segmentacije*

Problem vanjske fragmentacije kod čiste segmentacije može se riješiti koristeći kombinaciju segmentacije i straničenja, ali i to ima svoje prednosti i nedostatke. U ovom slučaj, alokacija segmenata fizičkoj memoriji je jednostavnija jer više nije potrebno pronaći jedan kontinuirani blok memorije koji bi bio dovoljno velik da drži cijeli segment. Stranice koje čine segment mogu biti smještene bilo gdje u fizičkoj memoriji. „Cijena“ za ovu jednostavniju alokaciju stranica i uklanjanje vanjske fragmentacije je ponovna pojava unutarnje fragmentacije. Međutim, više nije potrebno eksplicitno pohraniti veličinu segmenta u tablici segmenata. Bitovi validnosti (engl. *validity bits*) iz tablice stranica pružaju istu informaciju ukazujući koje (i stoga koliko) stranice sadrže odgovarajuće podatke. Konačno, kada je eksplicitno generiran broj okvira, više nije potrebno dodavati odmak na generiranu vrijednost. Umjesto dodavanja korišteno je spajanje koje je značajno brže. Na ovaj način iskorištene su neke prednosti obje metode.[5] Pogledati tablicu 4.1.

Tablica 4.1. *Usporedba segmentacije i straničenja*

	Segmentacija	Straničenje
Koliko linearnog adresnog prostora ima pojedina metoda?	Više	1
Dozvoljava li pokretanje programa čiji je virtualni adresni prostor veći od dostupne fizičke memorije?	Da	Da
Je li potrebno da programer bude svjestan da se ova metoda koristi?	Da	Ne
Postoji li mogućnost da se podatci i procedure odvoje i zasebno zaštite?	Da	Ne
Mogu li se adresni prostori čija veličina raste jednostavno i efikasno upravljati?	Da	Ne
Je li moguće dijeljenje podataka između procesa?	Da	Da (ali je zahtjevnije)
Koja metoda podržava jednostavniji i brži proces prevođenja?	Složeniji i sporiji	Jednostavniji i brži
Uzrokuje li vanjsku fragmentaciju?	Da	Ne
Uzrokuje li unutarnju fragmentaciju?	Ne	Da

5. ZAKLJUČAK

U ovom radu objašnjeno je kako virtualna memorija koristi pomoćni spremnik (poseban dio tvrdog diska koji zovemo *swap* disk ili *page* file) da bi proširila kapacitet radne memorije te na taj način omogućila pokretanje programa koji su mnogo veći od dostupne fizičke memorije. Ova opcija dosta je jeftinija nego dodavanje fizičke memorije u sustav i u većini slučajeva ne utječe značajno na performanse sustava.

Objašnjene su straničenje i segmentacija kao metode implementacije virtualne memorije. Straničenje dijeli virtualni adresni prostor na fiksne i jednake stranice koje učitava u analogne okvire fizičke memorije po principu jedna stranica-jedan okvir. Jedinica za upravljanje memorijom (MMU) koristi tablice stranica i priručni međuspremnik za prevođenje adresa (TLB) kako bi otkrila koji okvir odgovara kojoj stranici. Također, vidljivo je kako korištenje TLB-a ubrzava proces prevođenja. Nadalje, stranice se prema potrebi učitavaju u radnu memoriju i iz nje uklanjaju, a ovaj proces zovemo straničenje na zahtjev. Nakon toga, obrađena je segmentacija koja dijeli program na odvojene segmente različite veličine što značajno olakšava rad sa strukturama podataka koje rastu ili se smanjuju tijekom izvođenja programa. Objašnjeno je kako programer može utjecati na segmentiranu memoriju. Može se uočiti kako ova metoda smanjuje unutarnju fragmentaciju koja je prisutna kod straničenja, ali uzrokuje vanjsku fragmentaciju. Također, može se vidjeti kako je moguće kombinirati ove dvije metode u sustavu koji koristi segmente promjenjive veličine napravljene korištenjem promjenjivog broja stranica fiksne veličine kako bi se smanjio problem vanjske fragmentacije kod čiste segmentacije. Nadalje, pojašnjeno je korištenje više-razinske i invertirane tablice stranica kada se radi sa velikim adresnim prostor, na primjeru Windows 10 operacijskog sustava koji radi na 64-bitnoj arhitekturi. Vidljivo je kako njihova primjena smanjuje ogromnu tablicu stranica i omogućuje efikasno korištenje virtualne memorije. Na kraju je napravljena usporedba metoda implementacije virtualne memorije te se može uočiti kako svaka metoda ima neke svoje prednosti i nedostatke i kako neki od nedostataka mogu biti riješeni ili umanjeni koristeći kombinaciju ove dvije metode.

Iako se u današnje vrijeme razvijaju radne memorije velikog kapaciteta, veličina programa također postaje sve veća. Nerijetko je pokrenut veliki broj programa u sustavu koji također zauzimaju puno radne memorije. Stoga, virtualna memorija ostaje jako bitan dio memorijske hijerarhije i zapravo cijelog računalnog sustava. Međutim, virtualna memorija ima svoja ograničenja pa ju treba razumno koristiti kako ne bi došlo do pojave koju zovemo *trashing*.

LITERATURA

[1] Tanenbaum, Andrew. Modern Operating Systems. Treće izdanje. Amsterdam: Pearson, 2007.

[2] Austin, Todd; Tanenbaum, Andrew. Structured Computer Organization. Šesto izdanje. Pearson, 2013.

[3] Yadin, Aharon. Computer Systems Architecture. Prvo izdanje. Izrael: Chapman & Hall/CRC, 2016.

[4] Budin, Leo; Golub, Marin; Jakobović, Domagoj; Jelenković, Leonardo. Operacijski sustavi. Prvo izdanje. Zagreb: Element, 2010.

[5] Carpinelli, John. Computer Systems Organization and Architecture. Prvo izdanje. Pearson, 2002.

[6] Williams, Rob. Computer Systems Architecture: A Networking Approach. Drugo izdanje. Pearson, 2006.

[7] Physical and Virtual Memory in Windows 10. Dostupno na: https://answers.microsoft.com/en-us/windows/forum/windows_10-performance/physical-and-virtual-memory-in-windows-10/e36fb5bc-9ac8-49af-951c-e7d39b979938

[8] Memory Segments. Dostupno na: http://www.cs.uwm.edu/classes/cs315/Bacon/Lecture/HTML/ch10s04.html

[9] Turning the Pages: Introduction to Memory Paging on Windows 10 x64. Dostupno na: https://connormcgarr.github.io/paging/

[10] Page Tables. Dostupno na: https://os.phil-opp.com/page-tables/#mapping-page-tables

[11] Inverted Page Table in Operating System. Dostupno na: https://www.geeksforgeeks.org/inverted-page-table-in-operating-system/

SAŽETAK

Virtualna memorija je mehanizam upravljanja radnom memorijom. Omogućuje pokretanje programa koji su veći od dostupne fizičke memorije. U tu svrhu koristi pomoćni spremnik koji se nalazi na tvrdom disku, a naziva se *swap* disk ili *page file*.

Straničenje i segmentacija metode su implementacije virtualne memorije. Segmentacija program dijeli na mnoštvo neovisnih adresnih prostora koji se nazivaju segmenti. S druge strane, straničenje dijeli program na fiksne i jednake stranice koje učitava u odgovarajuće okvire radne memorije. Osnovna svrha segmentacije olakšati je rad sa strukturama podataka koje rastu ili se smanjuju tijekom izvođenja programa. S druge strane, svrha straničenja je da na jednostavan i brz način omogući pokretanje programa koji su veći od dostupne fizičke memorije. Segmentacija uzrokuje vanjsku fragmentaciju, a straničenje unutarnju. Moguće je kombinirati ove dvije metode kako bi se umanjili ovi problemi.

Napretkom tehnologije pojavljuje se potreba za nadogradnjom postojećih metoda pa se primjerice uvode više-razinske i invertirane tablice stranica. Njihova svrha je povećati efikasnost sustava koji rade sa velikim adresnim prostorom.

Ključne riječi: virtualni adresni prostor, fizički adresni prostor, virtualna memorija, straničenje, segmentacija

ABSTRACT

Virtual memory is a memory management mechanism. It allows running programs that are larger than available physical memory. For this purpose, it uses an auxiliary container located on the hard disk, which is called a swap disk or page file.

Paging and segmentation are methods that implement virtual memory. Segmentation divides program into a multitude of independent address spaces that are called segments. On the other hand, paging technique divides program into fixed and equal pages that it loads into the appropriate working memory frames. The main purpose of segmentation is to facilitate work with data structures that increase or decrease during program execution. On the other hand, the purpose of paging is to allow running of programs that are larger than the available physical memory in a simple and fast way. Segmentation causes external fragmentation, and paging causes internal. It is possible to combine these two methods to minimize these problems.

With the advancement of technology, there is a need to upgrade existing methods, so for example, multi-level and inverted page tables are being introduced. Their purpose is to increase the efficiency of systems that work with a large address space.

Keywords: virtual address space, physical address space, virtual memory, paging, segmentation

ŽIVOTOPIS

Ivan Hajmiler rođen je u mjestu Chur (Švicarska) 29.10.1991. godine. Pohađao je osnovnu školu Dore Pejačević u Našicama. Nakon toga u razdoblju od 2006. do 2010. godine stječe srednjoškolsko obrazovanje, prirodoslovno-matematička gimnazija, u Srednjoj školi Isidora Kršnjavog u Našicama. U razdoblju od 2010. do 2016. godine pohađa Medicinski fakultet u Osijeku. Od 2016. godine pohađa Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, stručni studij Elektrotehnike, smjer Informatika.