# Upravljanje memorijom

Memorija je, kao i procesor, fundamentalni deo modernih računarskih sistema. Memorija se sastoji od niza memorijskih reči, a svaka ima jedinstvenu adresu. Prilikom izvršavanja prcesa, procesor na osnovu programskog brojača (Program counter, PC) čita instrukciju iz memorije (fetch). Pročitane instrukcije u toku izvršenja dodatno mogu zahtevati čitanje operanada ili upisivanje podataka na druge memorijske lokacije.

## *Uvodne napomene*

Razni operativni sistemi koriste različite metode upravljanja radnom memorijom. Ove metode mogu biti krajnje jednostavne, ali i veoma složene, poput straničenja (paging) i segmentacije, a svaka ima svoje prednosti i mane. Izbor metode za upravljanje memorijom u velikoj meri zavisi i od hardverske podrške, tj od procesorske arhitekture. Na nivou sloja upravljanja memorijom treba postići sledeće ciljeve:

- Alokacija memorije, tj dodela memorije procesima
- Razdvajanja fizičkog i logičkog adresnog prostora programa i vezivanje adresa (prevođenje relativnih relokatibilnih adresa u fiksne)
- Logička orginazacija memorije, što znači razdvajanje neizmenljivih segmenata (modula i procedura) od segmenata s promenljivim sadržajem tj podacima.
  Razdvanjanjem segmenata program se postiže:
  1. nezavisan rad na segmentima koji čine program
  2. zaštita segmenata sa malim premašenjem, u vidu listi za kontrolu pristupa segmentima (u višeprocesnom okruženju, procesi ne smeju menjati vrednost memorijskih lokacija koje pripadaju drugim procesima bez dozvole)
  3. deljenje segmenata između većeg broja procesa (u slučaju da proces ima dozvolu da izmeni vrednost memorijskih lokacija koje koristi drugi proces, operativni sistem mora da obezbedi kontrolisan pristup deljenom memorijskom području)
- Relokacija, koja obuhvata sažimanje tj defregmentaciju radne memorije (vezivanje određenog broja diskontinualnih memorijskih blokova u jedan kontinualni nefragmentisan prostor) i swap (suspendovanje procesa njegovim smeštanjem na disk)
- Podrška za dinamičko punjenje memorije programom i dinamičko vezivanje

## *Dodeljivanje memorije*

U radnu memoriju se, pored korisničkih procesa, smešta i rezidentni deo operativnog sistema (npr jezgro). Da bi u višeprocesnom okruženju obezbedio stabilan i pouzdan rad sistema, neophodno je što efikasnije dodeliti različite delove memorije. Mamorija se deli na najmanje dve particije od kojih je jedna (najčešće niži deo) namenjena rezidentnom delu operativnog sistema (kernel space) a druga particija (viši delovi) – korisničkim procesima (user space). Obično se u najnižem delu memorije nalazi tabela prekidnih rutina.

U korisničkom adresnom prostoru nalazi se više procesa koji formiraju red čekanja na procesor. Takođe, postoje i procesi koji nastoje da uđu red čekanja, obično sa diska. Da bi proces ušao u red čekanja na procesor, neophodno je da najpre dobije potrebnu memoriju. Glavni problem pri upravljanju memorijom jeste dodela slobodne memorije procesima koji se u ulaznom redu (alokacija memorije).

Tehnike za dodelu memorije procesima grubo se mogu podeliti na dve vrste:

- ***Kontinualna alokacija (contiguous allocation)*** – i logički i fizički adrsni prostor procesa sastoje se od kontinualnih niza memorijskih reči, pri čemu memorijske particije koje se dodeljuju prcesima po veličini mogu biti jednake ili razčličite
- ***Diskontinualna alokacija (discontiguous allocation)*** – fizički adresni prostor procesa nije realizovan kao kontinualan niz memorijskih adresa; diskontinualna alokacija obuhvata metode ***straničenja***, ***segmentacije i straničenja sa segmentacijom.***

***Vezivanje adresa***

Po pravilu, program se nalazi na disku kao binarna izvršna datoteka (*binary executable*). Program sa diska se mora mora učitati u memoriju, unutar adresnog prostora novostvorenog procesa. Zavisno od metode upravljanja memorijom koja se koristi, proces se u toku izvršavanja može više puta pomerati na relaciji disk-memorija. ***Kolekcija procesa na disku, koja čeka povratak u memoriju i nastavak izvršenja, naziva se ulazni red (input queue).***

U sistemima sa omogućenim multiprogramiranjem, veći broj procesa deli radnu memoriju računara. Programer ne može unapred znati koji će procesi biti zastupljeni u memoriji prilikom izvršenja programa u takvom okruženju, niti koje će memorijske lokacije biti slobodne. On ne može unapred odrediti fiksne memorijske lokacije za smeštaj programa i zato koristi relativne ili simboličke (zavisno od programskog jezika i konkretne upotrebe). Dužnost operativnog sistema je da prevede relativne tj relokatibilne adrese u fiksne prilikom učitavanja programa u memoriju.

Pre samog izvršavanja, korisnički program obično prolazi kroz više faza, tokom kojih se memorijske adrese predstavljaju na različite načine. Prevodilac (compiler) prevodi izvorni kôd programa i vezuje (bind) simboličke adrese iz izvornog koda za relativne tj relokatibilne aderese, koje punilac (loader) pretvara u apsolutne prilikom učitavanja programa u memoriju.

Npr, prevodilac vezuje promenljivu **count** za lokaciju na adresi 14 u odnosu na početak programskog modula, koju punilac pretvara u apsolutnu adresu 74704. Vezivanje adresa se može shvatiti kao transformacija, tj prevođenje adrese sa „jezika“ jednog adresnog prostora u drugi.

Povezivanje instrukacija i podataka sa memorijskim adresama obavlja se u sledećim fazama:

- ***Vreme prevođenja (compile time)***

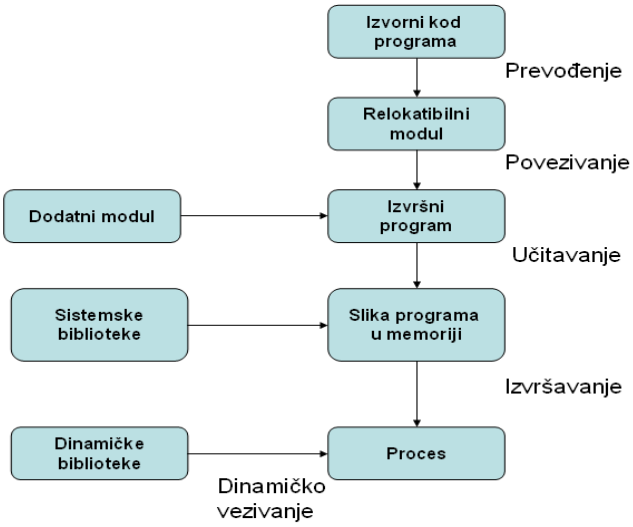
Kako nije poznato gde će proces koji će izvršavati generisani kôd biti smešten u memoriji, prevodilac generiše relativne, a ne apslolutne adrese. Program se kasnije može smestiti bilo gde u memoriji.

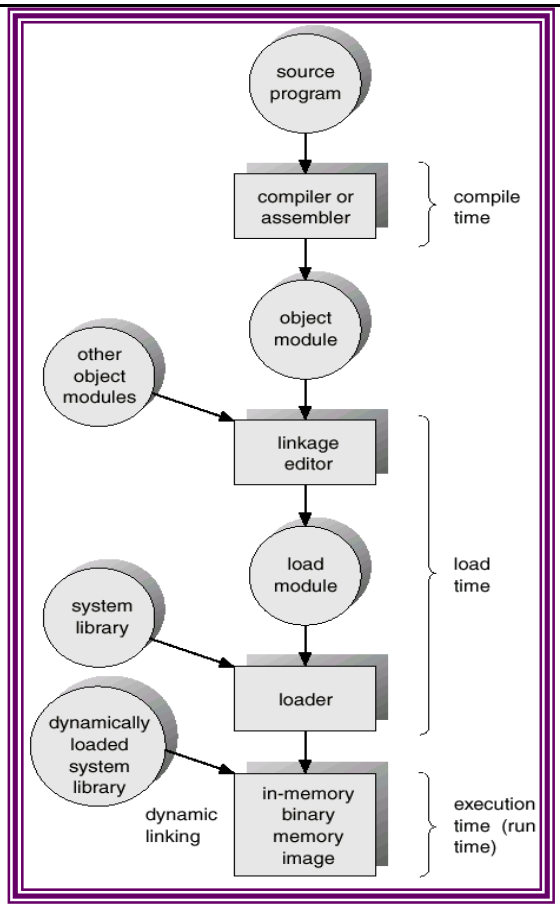
- ***Vreme učitavanja u memoriju (load time)***

U fazi učitavanja povezivač (linker) i punilac (loader) na bazi relokatibilnog koda generišu apsloutne adrese i pune memoriju programom. Povezivač može povezati korisnički program sa drugim relokatibilnim modulima (object modules).

- ***Vreme izvršavanja (execution time)***

Za vreme izvršavanja, proces se može pomerati s jednog segmenta na drugi (uključujući i disk, ukoliko se koristi i virtuelna memorija)



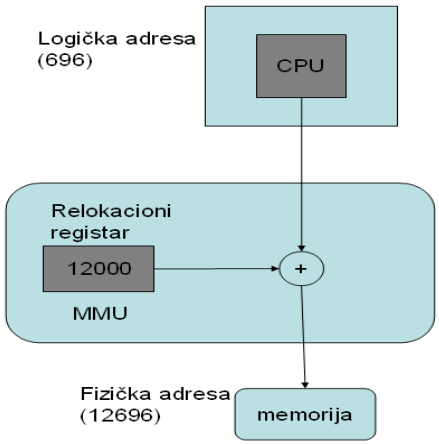


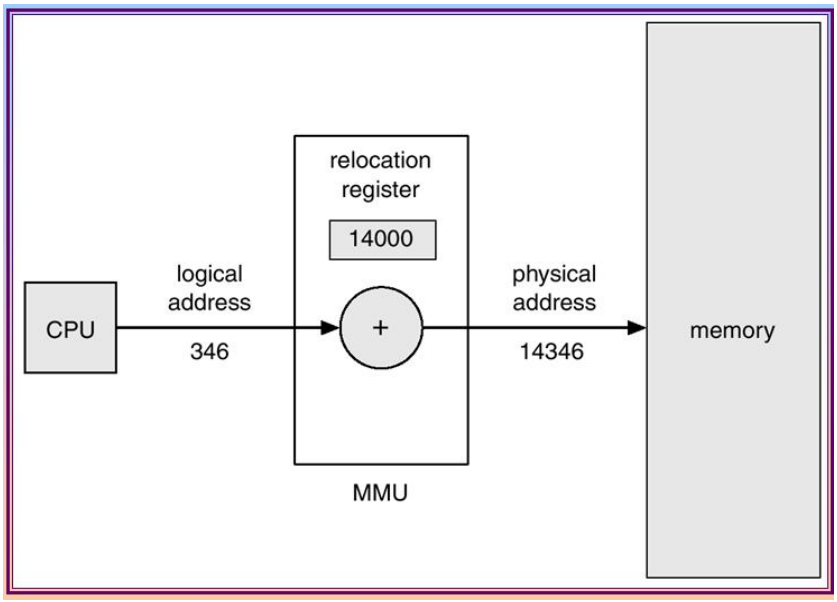

*Faze koje prethode izvršavanju programa*

***Logički i fizički adresni prostor***

***Adresa koju generiše procesorska instrukcija je logička, a adresa same memorijske jedinice je fizička. Logičke i fizičke adrese su potpuno iste u fazi prevođenja i u fazi učitavanja programa, ali se razlikuju u fazi izvršavanja (logičke adrese se u fazi izvršavanja nazivaju i virtuelnim adresama). Skup svih logičkih adresa koje generiše program naziva se logički ili virtuelni adresni prostor, a skup svih fizičkih adresa koje njima adgovaraju naziva se fizički adreni prostor.***

Mapiranje (preslikavanje) virtuelnog adresnog prostora u fizički obavlja hardverski uređaj koji se naziva MMU (Memory-Management Unit) – jedinica za upravljanje memorijom.



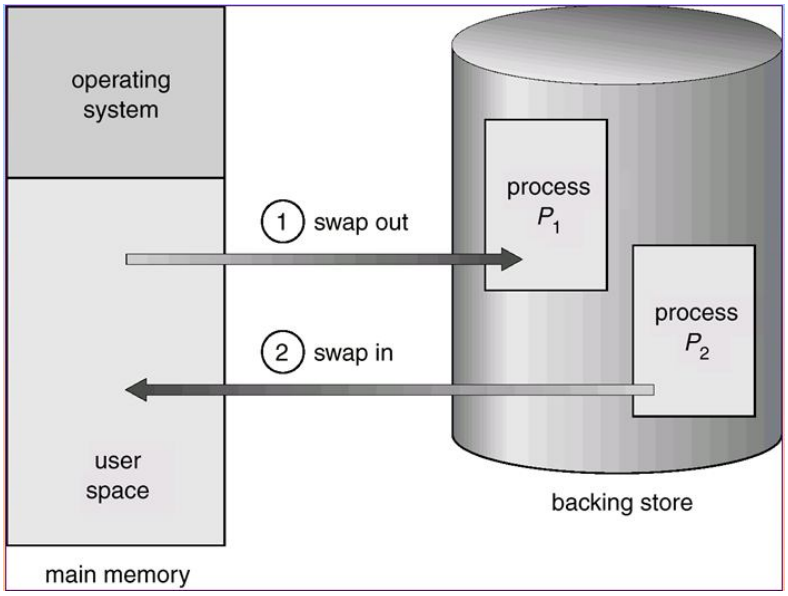



*Mapiranje pomoću relokacionog registra*

Relokacioni registar definiše adresu fizičkog početka programa. Svaka logička adresa koju generiše progrma se sabira s vrednošću relokacionog registra i tako se dobija fizička adresa. Korisnički program uvek počinje od nulte adrese i ne treba voditi računa o svom fizičkom prostoru, osim o gornjoj granici programa (max). Logički adresni prostor koji se nalazi u opsegu [0,max] mapira se u opseg [R+0, R+max], gde je R vrednost relokacionog registra, tj fizička adresa početka programa.

***Razmena (swap)***

Prilikom izvršavanja, proces se mora nalaziti u radnoj memoriji. Postoje situacije kada se proces može privremeno prebaciti iz memorije na disk, kako bi se oslobodila memorija. Oslobođenja memorija puni se drugim procesom.




*Razmena*

Razmena (swap) koristi se u prioritetnim šemama za raspoređivanje procesa gde se procesi visokog prioriteta čuvaju u memoriji, dok se svi procesi niskog prioriteta upisuju na disk i čekaju da se oslobodi memorija. Ova varijanta razmenjivanja naziva se ***roll out, roll in***. Proces koji se razmenjuje mora biti potpuno oslobođen aktivnosti – ne sme da radi, niti da čeka kraj neke ulazno-izlazne operacije (stanja WAIT i READY).

Tehnika razmene zahteva postojanje 3 komponente:

- Prostor na disku (***swap space***) na koji će se smeštati uspavani procesi
- Mehanizam ***swap-out*** koji prebacuje proces iz memorije na disk
- Mehanizam ***swap-in*** koji vraća uspavani proces sa diska u memoriju

Najveći deo vremena u ciklusima razmene otpada na prenos podataka između memorije i diska. Trajanje jedne razmene zavisi od količine podataka za prenos, karakteristike diskova i pratećeg hardvera. Kako je to vreme ogromno u odnosu na vreme izvršavanja memorijskih ciklusa, ne preporučuje se često korišćenje tehnike razmenjivanja.

Swap postoji na svim modernim operativnim sistemima i to u različitim modifikovanim varijantama. Retko se razmenjuju celi procesi – uglavnom se razmenjuju manji delovi memorije (npr stranice). Tehnika razmenjivanja je tesno povezana s tehinkom virtuelne memorije.

## PROGRAMSKE TEHNIKE UPRAVLJANJA MEMORIJOM

Operativni sistem je dužan da obezbedi odgovarajuću podršku za rad sa radnom memorijom. Međutim, prilikom projektovanja programa koji zbog veličine ne mogu stati celi u memoriju, programeri koriste odgovarajuće tehnike kojima se postiže prividno povećanje i bolje iskoriščenje memorijskog prostora.

### *Dinamičko učitavanje programa u memoriju*

U dosadašnjim diskusijama podrazumevalo se da se ceo program smešta u memoriju, pri čemu veličinu procesa ograničava količina fizičke memorije. Memorija se može efikasnije iskoristiti ukoliko se primene tehnike dinamičkog punjenja programa. Suština dinamičkog punjenja (dynamic loading) odnosi se na smeštaj samo potrebnih delova programa u memoriju, pri čemu se odgovarajuće rutine učitavaju u memoriju samo kad ih program pozove. Da bi to bilo moguće, sve rutine programa čuvaju se na disku u relokatibilnom formatu.

Kada se rutina pozove iz programa, proverava se da li je ona već u memoriji i – ako nije – poziva se punilac da je učita u memoriju.

Prednosti dinamičkog punjenja su u tome što rutine koje nisu trenutno potrebne ne zauzimaju mesto u memoriji, što je zgodno za velike programe. Takođe, dinamičko punjenje ne zahteva specijalnu podršku od operativnog sistema – programer mora sam projektovati svije programe tako da koriste principe dinamičkog punjenja. Operativni sistem može pomoći programeru tako što obezbeđuje biblioteku za dinamičko punjenje.

### *Dinamičko povezivanje*

Prilikom statičkog povezivanja, sistemske biblioteke se tretiraju kai i svi drugi objektni moduli koji se kombonuju s korisničkim programom u jedinstvenu izvršnu verziju i njom se puni memorija. Neki operativni sistemi podržavaju jedino koncept statičkog povezivanja. Koncept dinamičkog povezivanja (*dynamic linking*) sličan je konceptu dinamičkog punjenja. Kao što se punjenje modula odlaže (memorija se puni modulima po potrebi), isto se postupa i sa povezivanjem (*link*) u vreme izvršenja – rutina iz sistemeske biblioteke puni se po potrebi u vreme izvršavanja.

Pri statičkom povezivanju, svaki poziv sistemske biblioteke zahteva da se kôd kompletne rutine biblioteke kopira u kôd programa i to za svaki poziv pojedinačno, čime raste program i na disku a i u memoriji kada dođe vreme izvršavanja. Pri dinamičkom povezivanju naparavi se vezivna funkcija (*stub*), kao parče koda za svaki poziv sistemske biblioteke. Vezivna funkcija pokazuje kako da se locira odgovarajuća sistemska rutina i kako da se napuni, ako već nije u memoriji. Preko vezivne funkcije, rutina koja se jedanput dovede u memoriju može se koristiti više puta. Dakle, za svaki poziv iste funkcije u programu imamo samo jedno memorijsko punjenje i zauzeće. Dinamičko punjenje omogućava laku izmenu sistemskih biblioteka, a da stari programi ne moraju da se prevode ponovo (*shared library*).

Za razliku od dinamičkog punjenja, dinamičko povezivanje traži podršku na nivou operativnog sistema – istu sistemsku rutinu koja je u memoriji može koristiti više procesa.

### *Tehnika preklapanja*

Da bi se omogućilo izvršenje procesa koji je veći od same fizičke memorije, koristi se tehnika preklapanja (*overlay*). Preklapanje omogućava da se u memoriji čuvaju samo oni delovi programa koji su potrebni u tom trenutku. Kada drugi delovi programa budu potrebni, oni će se učitati u memoriju umesto delova koji više nisu potrebni.

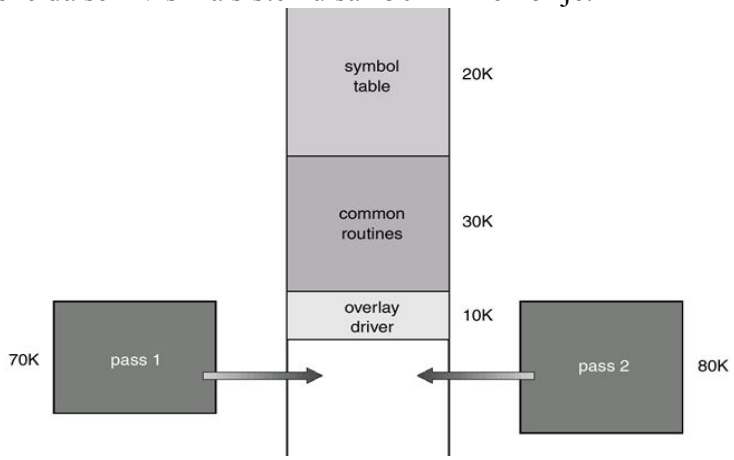
Programer može podeliti sam izvorni kôd programa na dva dela. Pretpostavimo da je veličina programskih komponenata sledeća:

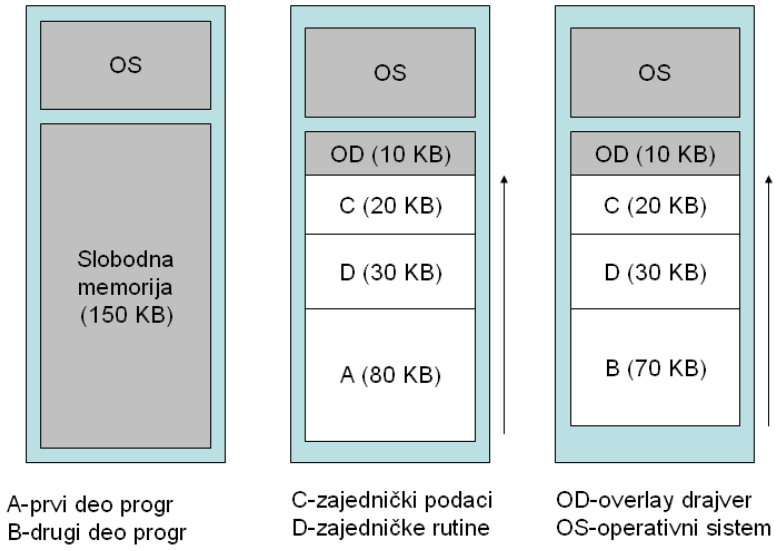
- Prvi deo (80 KB)
- Drugi deo (70 KB)
- Zajednički podaci (20 KB)
- Zajedničke rutine (30 KB)
- Drajver koji upravlja tehnikom preklapanja (10 KB)

Za izvršenje programa potrebni je najmanje 200 KB slobodne memorije (ne računajući rezidentne delove OS), što znači da program ne može da se izvšava na hipotetičkom sistemu od 150 KB radne memorije. Međutim, prilikom izvršavanja prvog dela programa, kôd drugog dela ne mora da biti učitan u memoriju. Takođe, drugi deo programa se može izvršavati i ukoliko prvi deo nije u memoriji. Zato se definišu 2 komponente preklapanja:

- *Overlay A*, koji se sastoji od zajedničkih podataka i rutina, i koda za prvi deo programa, što iznosi 130 KB
- *Overlay B*, koji se sastoji od zajedničkih podataka i rutina, i koda za drugi deo programa, što iznosi 120 KB

Programu se se mora dodati drajver koji upravlja tehnikom preklapanja. U prvoj fazi imamo *overlay* drajver i *overlay A* što zauzima 140 KB i može se izvršiti u 150 KB. Kada *overlay A* završi svoje, drajver će učitati u memoriju *overlay B* preko njega. U drugoj fazi, u memorji su prisutni *overlay* drajver i *overlay B*, što zauzima 130KB i takođe može da se izvrši na sistemu sa 150KB memorije.






*Tehnika preklapanja*

Tehnika preklapanja usporava rad samog programa, jer se delovi programa učitavaju u memoriju u više iteracija, ali se njome postiže ono što drugačije ne bi bilo moguće. Tehnika preklapanja može biti korisna na sistemima sa ograničenim resursima.

*Overlay* drajveri su uključeni u neke programske jezike (Borland Pascal i Clipper), tako da primena tehnike preklapanja ne zahteva posebnu podršku operativnog sistema. Konkretnu strukturu *overlay* delova definiše programer jer najbolje poznaje svoj program

## KONTINUALNO DODELJIVANJE MEMORIJE

Ukoliko se koristi kontinualno dodeljivanje memorije, i logički i fizički adresni prostor procesa se sastoje od kontinualnog niza memorijskih adresa. Prosto rečeno, svaki proces dobija jedan kontinualni deo memorije. Metode kontinualnog dodeljivanja memorije su:

- Multiprogramiranje sa fiksnim praticijama
- Multiprogramiranje sa particijama promenljive veličine

### *Multiprogramiranje sa fiksnim particijama*

Jedan od najstarijih i najprostijih metoda dodeljivanja memorije jeste podela cele fizičke memorije na više delova fiksne veličine, pri čemu se u jednom delu može naći samo jedan proces. U ovakvoj organizaciji, stepen multiprogramiranja je jednak broju memorijskih particija. Ova metoda je korišćena u sistemu IBM OS/360. Cela memorija se izdeli na više delova. Svi procesi se stavljaju u red čekanja (*input queue*) koji može biti jedinstven za ceo sistem ili poseban za svaku particiju.

Višestruki redovi čekanja običmo se formiraju za opsege veličina Q=1 KB, Q=2 KB, Q=4 KB. Ukoliko za proces koji je došao na red nema dovoljno memorije, uzima se sledeći manji proces iz liste. Kada postoji više redova čekanja, veći broj malih procesa može čekati u redu za male particije, dok su velike particije neiskorišćene. U tom slučaju ima dovoljno memorije,ali se ne koristi. Bolji je jedinstveni red čekanja, jer ako nema mesta u memoriji u redu čekanja za particiju koja odgovara veličini procesa, procesu se dodeljuje veća particija. Dva procesa ne mogu biti smeštena u jednoj particiji.

Multiprogramiranje sa fiksnim particijama korišćeno je u sistemima s grupnom obradom (*batch*). Ova metoda je nepogodna za interaktivne sisteme zbog postojanja većeg broja procesa promenljive veličine (obično malih) koji se pojavljuju po slučajnom rasporedu. Zbog toga se ova metoda više ne koristi.

### *Multiprogramiranje sa particijama promenljive dužine*

Umesto fiksnih particija, memorija se deli dinamički,a svaka šupljina (*hole*), tj slobodan kontinualni deo memorije, može se iskoristiti za smeštanje procesa – pod uslovom da je dovoljno velika. Šupljine dinamički nastaku i nestaju, zajedno sa procesima, mogu biti bilo gde u memoriji i imati bilo koju veličinu, što odgovara procesima na interaktivnim sistemima. Kada proces naiđe u sistem, traži se šupljina dovoljno velika za proces. Sav prostor koji proces ne zauzme od cele šupljine, predstavlja novu šupljinu u koju se može smestiti novi proces.

Alokacija memorije je dinamička – memorija se sastoji od procesa i šupljina, a OS dinamički vodi evidenciju o zauzetosti memorije na jedan od sledećih načina:

- Bit mape (*bit maps*)
- Povezane liste (*linked lists*)
- Sistem udruženih parova – drugova (*buddy system*)

## DISKONTINUALNO DODELJIVANJE MEMORIJE

Koriste se dve metode:

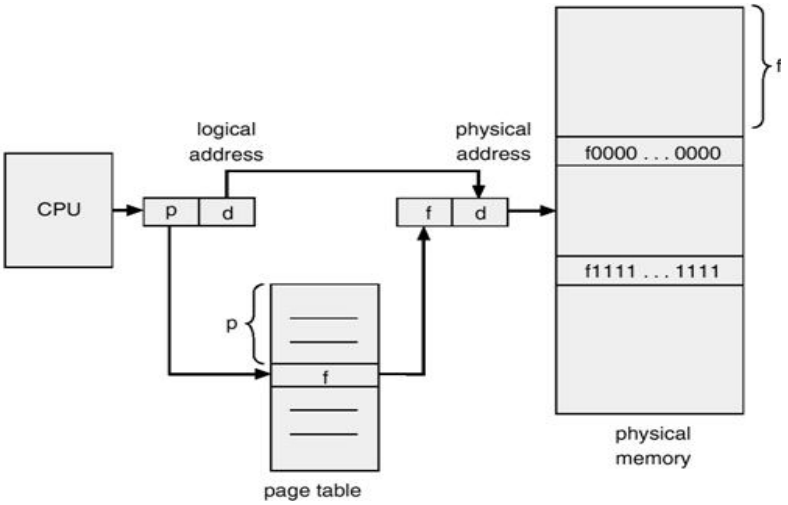
- **straničenje (*paging*)**
- **segmentacija (*segmentation*)**
- **straničenje sa segmentacijom**
- **segmentacija sa straničenjem**

### *Straničenje*

Straničenje je metoda sa hardverskom podrškom na nivou procesora koja se koristi u svim operativnim sistemima i na svim računarskim arhitekturama. Gotovo da ne postoji nijedan savremeni procesor koji hardverski ne podržava straničenje.

Fizička memorija, tj fizički adresni prostor, izdeli se na blokove fiksne veličine, koji se **nazivaju fizičke stranice ili okviri (*page frames*).** Logički adresni prostor takođe se izdeli na blokove istih veličina koji se nazivaju **logičke stranice (*pages*).** U daljem tekstu, pod terminom stranice podrazumevaćemo logički stranicu, a pod terminom okvir – fizičku. Veličine stranica su po pravilu stepen broja 2, najčešće u opsegu od 512 B do 8 KB, mada mogu biti i veće, 16 MB.

*Swap* prostor na disku takođe se deli na stranice koje po veličini odgovaraju memorijskim stranicama.




*Straničenje*

Metoda straničenja funkcioniše na sledeći način: svakoj logičkoj stranici odgovara jedna fizička, a korespodencija između njih se čuva u tabeli stranica (*page table*). To omogućava da se kontinualni logički prostor procesa razbaca svuda po memoriji. Svaka logička adresa koju generiše procesor deli se na dva dela:

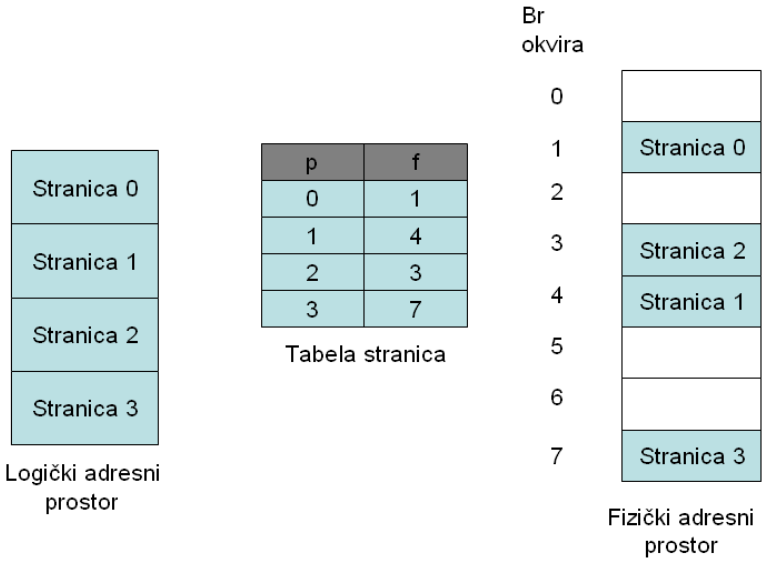
- broj stranice (p, *page number*) – koristi se kao indeks u tabeli stranica koja sadrži baznu adresu okvira. Bazna adresa predstvlja viši deo adrese
- pomeraj unutar stranice (d, *page offset*) – definiše položaj u odnosu na samu stranicu i – u kombinaciji sa baznom adresom (čisto sabiranje) – definiše punu fizičku adresu koja se šalje memorijskoj jedinici. Naglasimo da je osnovna adresabilna jedinica bajt i da je pomeraj isti i za logičku i za fizičku adresu

Ako je veličina logičkog adresnog prostora $2^m$, a veličina stranice $2^n$, tada viši deo adrese dužine m-n definiše broj stranice, dok najnižih n bitova adrese predstavljaju pomeraj unutar stranice.

Primer straničenja:
Zamislimo da imamo memoriju veličine 32 B. Definišemo 8 okvira veličine 4 B, što znači da je m=6, a n=2. Uzmimo korisnički proces koji zauzima 4 logičke stranice, sa logičkim adresama:

- stranica 0 (0-3)
- stranica 1 (4-7)
- stranica 2 (8-11)
- stranica 3 (12-15)

Logička adresa 0 ima broj logičke stranice 0. Pomoću broja logičke stranice ulazimo u tabelu stranica i nalazimo da je ona smeštena u okvir broj 1, i tako redom: logička stranica 1 smeštena je u okvir 4, logička stranica 2 u okvir 3 i logička stranica 3 u okvir 7.




*Primer straničenja*

Proces koji ulazi u stanje izvršavanja mora da dobije potrebnu memoriju, tako da se za dati proces preračunava koliko mu stranica memorije treba. Svaka stranica mora da se mapira u okvir. Ako proces zahteva n stranica, tada se alocira n okvira, koji se pune procesom, pri čemu se mapiranje stranica – okvir upisuje u tabelu stranica.

Svako straničenje predstavlja dinamičku relokaciju, a tabela stranica predstavlja relokacioni registar za svaki okvir fizičke memorije. Smanjenje stranica dovodi do povećanja njihove tabele, jer je svaka stranica opisana jednim zapisom u tabeli. Kao posledica, povećava se kašnjenje peilikom mapiranja ili pretraživanja. U slučaju korišćenja *swap* tehnike sa stranicama, poželjnije su veće stranice zato što je rad sa diskovima efikasniji kada su transferi veći.

Značajan aspekt straničenja je jasno razdvajanje korisničkog pogleda na memoriju i aktuelne fizičke memorije. Korisnik svoj deo memorije doživljava kao kontinualni prostor iako su stranice raznih procesa razbacane po memoriji. Mapiranje logičkog i fizičkog prostora zadatak je OS i korisnik ga ne vidi. Operativni sistem mora da prati koji su okviri slobodni, a koji su dodeljeni i to kom procesu. Sve se to čuva u jezgru, u tabeli okvira (*frame table*) u kojoj je svaki okvir opisan jednim zapisom. Na drugoj strani, operativni sistem mora da za svaki proces generiše tabelu stranica koja se odnosi samo na njegove stranice i koja definiše mapiranje za taj proces.

### *Segmentacija*

Prethodno razmatrani slučajevi odnose se na korisničke procese, tj programe, koji zahtevaju kontinualan memorijski prostor. Međutim, sama struktura programa nije kontinualna, jer programi se logički dele na više nezavisnih celina. Npr, sam kôd se sastoji od više celina kao što su tabele, polja, stek, promenljive, itd. Svaki od ovih logičkih segmenata dobija ime pomoću koga se referencira i može nezavisno da se učita u memoriji. Po pravilu, programer definiše logičke segmente u izvornom programu, a prevodilac na osnovu toga pravi memorijske segmente.

Segmentacija je metoda upravljanja memorijom koja podržava logički korisnički pogled na memoriju. Logički adresni prostor sastoji se od kolekcije segmenata, a svaki segment ima jedinstveno ime i dužinu. Logička adresa se sastoji od dva dela:

- imena segmenta (umesto imena segmenta obično se zadaje broj koji predstavlja identifikator segmenta)
- pomeraja unutar segmenta

Podsetimo se da je u straničenju logička adresa jednodimenzionalna, pri čemu razdvajanje delova adrese i transliranje obavlja hardver.

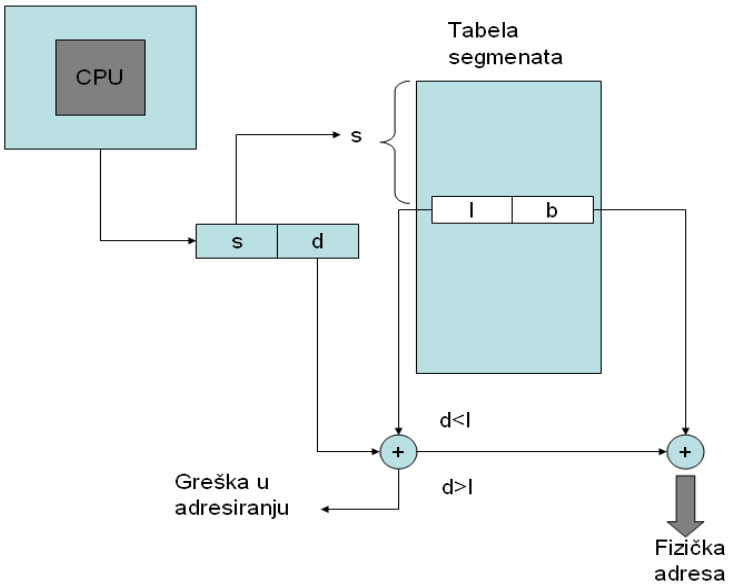
Pri segmentaciji, kao i pri multiprogramiranju s particijama promenljive dužine, javlja se eksterna fragmentacija. Slobodna memorija se ne može iskoristiti za smeštaj segmenata ukoliko ne postoji dovoljno velika šupljina, bez obzira na količinu slobodne memorije. Problem eksterne fragmentacije, koja zavisi od veličine segmenata i raspodele nailaska procesa, može se smanjiti sažimanjem memorije.

### *Hardverska podrška*

U metodi segmentacije, korisničke adrese su dvodimenzionalne, ali se moraju translirati u jednodimenzionalne fizičke adrese. Ovo mapiranje se obavlja preko tabele segmenata a ima podršku i u hardveru mikroprocesora. Svaki ulaz u tabeli segmenata opisuje tačno jedan segment, a sadrži dva parametra:

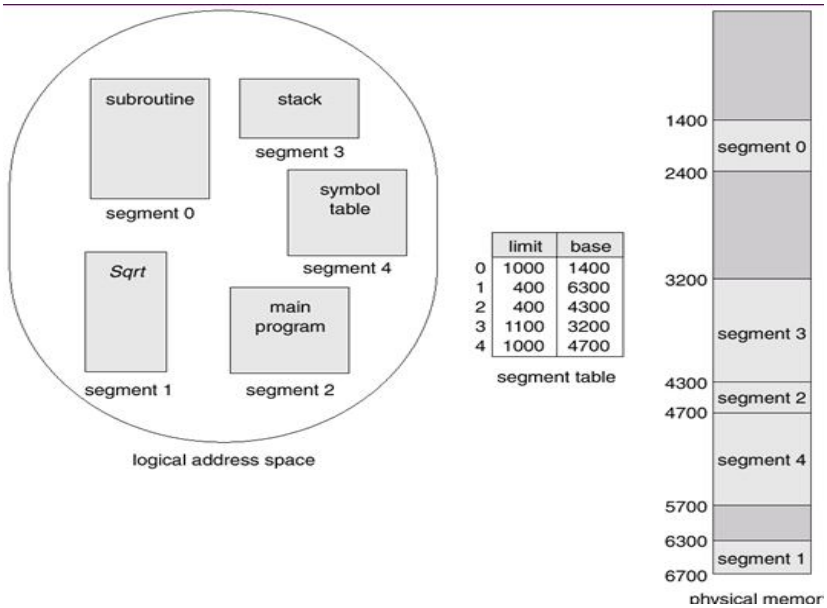
- baznu adresu segmenta, koja definiše početnu fizičku adresu segmenta u memoriji
- ograničenje segmenta, koje definiše dužinu segmenta

Broj segmenta s, koristi se kao ulaz u tabelu segmenata, iz koje se čitaju bazna adresa segmenta i ograničenje. Pomeraj d svakako mora biti manji od ograničenja segmenta.




*Segmentacija*

Na slici je prikazan slučaj segmentacije za 5 segmenata, čije su definicije date u tabeli segmenata.




*Primer segmentacije*

***Segmentacija sa straničenjem***

Najpopularnije serije procesora Intel (80x86 i Pentium) i Motorola (68000) imaju ugrađenu podršku i za segmentaciju i za straničenje, tako da omogućavaju primenu kombinovanih metoda diskontinualne alokacije pa se procesi mogu deliti na fizički diskontinualne logičke celine. Pri tome, straničenje poništava eksternu fragmentaciju segmenata. Procesor Intel 80386 koristi segmentacije sa straničenjem. Logička adresa se sastoji od identifikatora segmenta (*selector*) i pomeraja u okviru segmenta (*offest*). Iz tabele segmenata čita se adresa logičke stranice u katalogu stranica. Straničenje je realizovano u dva nivoa:

- spoljna tabela se zove katalog stranica (*page directory*)
- unutrašnja – tabela stranica