7 UPRAVLJNJE MEMORIJOM

7.1 UVOD

Memorija je jedna od temeljnih komponenata računarskog sustava. Ona je veliko polje okteta ili riječi koji imaju svaki svoju adresu. Procesor dohvata naredbe iz memorije u ovisnosti o vrijednosti upisanoj u programsko brojilo. Naredbe mogu rezultirati dodatnim pristupima memoriji kako bi se dohvatilo operande, odnosno pohranilo rezultate operacija.

7.1.1 Generiranje adrese

Obično su programi zapisani na disku (ili nekoj drugoj neizbrisivoj memoriji) u binarnom obliku (*executable*). Program se s diska upisuje u memoriju kako bi postao proces koje se može izvoditi. U zavisnosti o sustavu za upravljanje memorijom proces se može i tijekom obrade prebacivati iz radne memorije na disk i obratno. Skup procesa na disku koji čekaju da budu upisani u memoriju naziva se ulazni red (*input queue*). Normalna procedura se sastoji u odabiru jednog od procesa iz ulaznog reda i upisa tog procesa u radnu memoriju. Tijekom izvođenja proces dohvata iz radne memorije naredbe i podatke te u nju upisuje rezultate obrade. Po završetku obrade memorijski prostor koji je proces koristio proglašava se slobodnim.

Većina računarskih sustava dozvoljava da se proces nalazi bilo gdje u radnoj memoriji. Tako iako je početak adresnog prostora računala obično adresa 00000H, prva naredba korisničkog procesa ne mora se nalaziti na toj adresi. U većini slučajeva korisnički program prolazi kroz više faza pripreme prije njegovog izvođenja (slika 8.1). Adrese se različito prikazuju u svakom od navedenih koraka. Adrese u izvornom programu obično su simbolički prikazane kao npr. za simbolički programski odsječak:

```
start.
        mov    al,$00H
        mov    bl,$0AH
petlja.
        ...
```

adrese su *start* i *petlja*. Program za tumačenje simboličkog jezika (*assembler*) ove adrese vezuje (*bind*) s relativnim (*relocatable*) adresama (u odnosu na početak programa) Tako adresa *start* dobiva vrijednost 0000H, a adresa *petlja* 0004H (uz pretpostavku da su naredbe duljine 2 okteta). Nakon punjenja programa u memoriju programu se dodjeljuje stvarna, apsolutna adresa pa tako adresa pa tako npr. adresa *start* dobiva vrijednost 10000H, a adresa *petlja* 10004H.

Praktički dodjeljivanje (vezivanje) adresa naredbama i podacima moguće je realizirati u bilo kojoj fazi pripreme programa za izvođenje:

- **Vrijeme prevođenja**: Ukoliko je za vrijeme prevođenja poznata adresa lokacije od koje će se upisivati program, tada se već za vrijeme prevođenja generiraju apsolutne adrese.
- **Vrijeme punjenje:** Kod suvremenijih sustava za vrijeme prevođenja obično nije poznato gdje će se upisati program. Tada program prevodilac generira relativne adrese, a apsolutne se generiraju u trenutku upisivanja (punjenja) programa u radnu memoriju. Ovakav pristup je fleksibilniji od prethodnog iz razloga što kod promjene položaja programa u memoriji (npr. prijenos programa s jednog računala na drugo) potrebno je samo ponovo upisati program u memoriju, a program punjač automatski mijenja apsolutne adrese. Program nije potrebno ponovo prevoditi.

- **Vrijeme izvođenja:** U ovom slučaju moguće je da program mijenja svoje adrese i tijekom izvođenja, odnosno prebacuje se iz jednog u drugo memorijsko područje. Posebna sklopovska podrška potrebna je za realizaciju ovakvog načina rada.

Slika 8.1.: *Priprema korisničkog programa za izvođenje.*

7.1.2 Dinamičko punjenje

Bolje korištenje memorijskog prostora postiže se dinamičkim punjenjem (*dinamic loading*). Kod ovakvog načina punjenja programa u memoriju potprogrami ili rutine smještene su na disku, a upisuju se u memoriju tek nakon što ih glavni program pozove. Kod poziva potprograma, prvo se provjerava da li je potprogram u radnoj memoriji, a ako nije poziva se program za dinamičko punjenje i povezivanje (*relocatable linkage editor*) koji upisuje potprogram u memoriju te upisuje početnu adresu potprograma u tablicu adresa programa. Potom se izvođenje prosljeđuje na pozvani potprogram.

Potprogrami ili procedure koje se nikad ne koristi ovim pristupom nikad neće biti upisane u memoriju što je prednost dinamičkog punjenja. Ovaj pristup posebice je učinkovit kada veliki dio programskog paketa je posvećen procedurama za otkrivanje i otklanjanje pogrešaka, a koje se rijetko događaju. Tako je programski kod koji se koristi relativno mali iako je cijeli programski paket dosta velik.

Dinamičko punjenje ne postavlja posebne zahtjeve na sklopovlje računala kao i posebnu dodatnu podršku operacijskog sustava. Programer jedino mora poznavati princip rada kako bi svoj program prilagodio ovom sustavi i iskoristio sve njegove prednosti.

7.1.3 Dinamičko povezivanje

Na slici 8.1 prikazana je i biblioteka s programima koji se dinamički mogu povezivati s programom koji se izvodi (*dynamically linked libraries*). Većina operacijskih sustava podržava samo statičko povezivanje (*static linking*) kod kojih se sistemske biblioteke tretiraju identično ostalim objektnim modulima koji se povezuju u jedinstven program koji program punjač upisuje u radnu memoriju. Koncept dinamičkog povezivanja sličan je konceptu dinamičkog punjenja. Umjesto da se potprogrami ili procedure upisuju u memoriju prema zahtjevu, isti se prema zahtjevu povezuju s programom koji se izvodi. Ovo je posebice pogodno sa sistemskim bibliotekama. Prednost ovog pristupa u poredbi s dinamičkim punjenjem je u boljoj iskoristivosti radne memorije i diska jer nije potrebno da svaki program ima sve procedure, odnosno da više programa upisanih u radnu memoriju, a koji koriste istu proceduru, imaju njenu kopiju. Kod dinamičkog povezivanja koristi mali segment koda (opušak engl. *stub*) koji pokazuje na sistemsku proceduru koju program koristi. Ova procedura može biti stalno upisana u memoriji (*resident*) ili ju je potrebno upisati s diska.

Izvođenjem ovog koda prvo se provjerava da li je procedura već prisutna u memoriji ili ju je potrebno s diska upisati u memoriju. Nakon što je pozvana procedura pronađena ili upisana u memoriju, "stub" se zamjenjuje s adresom početka pozvane procedure te se procedura izvodi. Sljedeći put kada se ista procedura pozove ona se automatski izvodi bez izvođenja "stub"-a, odnosno bez dodatnog vremena potrebnog za dinamičko povezivanje. Prema opisanom postupku, bez obzira koliko programa koriste istu proceduru iz sistemske biblioteke postoji u memoriji samo jedna kopija.

Prednost ovakvog pristupa je i u jednostavnijoj nadogradnji sustava. Naime, dodavanjem novih verzija sistemskih biblioteka nije programe potrebno ponovo povezivati budući se isto izvodi dinamički.

Kao i kod dinamičkog punjenja, dinamičko povezivanje zahtijeva uslugu operacijskog sustava. Pod pretpostavkom da sustav izvodi istovremeno, konkurentno, više programa oni moraju međusobno biti zaštićeni od namjernih ili nenamjernih pogrešaka. Jedan od načina zaštite je da se jednom programu ne dozvoljava pristup memorijskim lokacijama drugog. Ali ukoliko dva programa koriste istu proceduru iz sistemske biblioteke tada

operacijski sustav dozvoljava različitim programima pristup istim memorijskim lokacijama.

7.1.4 Prekrivanje (*overlay*)

U razmatranjima koja su se provodila u prethodnim poglavljima pretpostavljeno je da se cijeli program koji se izvodi nalazi u radnoj memoriji. Time je maksimalna dozvoljena veličina programa određena veličinom radne memorije što predstavlja ozbiljno ograničenje. Moguće rješenje problema je da se u radnoj memoriji drži samo dio programa i podataka koji su trenutno potrebni. Kada su potrebni sljedeći podaci kao i program tada se iz memorije izbacuje prethodni dio programa i podataka, a novi se s diska upisuju. Ova tehnika naziva se prekrivanje (*overlay*).

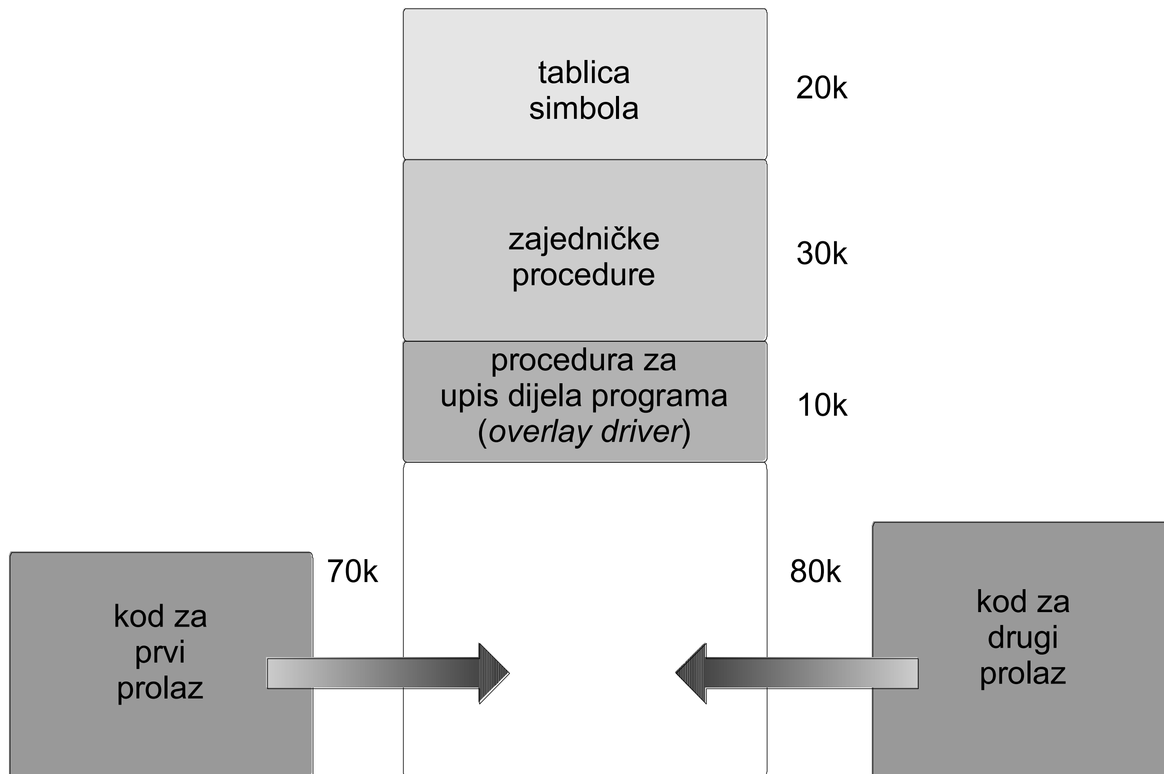
Kao primjer može se uzeti program prevodilac simboličkog programa (*assembler*). Ovaj program prevodi pomoću dva prolaza kroz izvorni kod. U prvom prolazu stvara se tablica simbola s odgovarajućim vrijednostima, dok se u drugom prolazu generira izvedbeni kod. Ovaj program može se podijeliti na dio koji generira tablicu simbola, dio koji generira izvedbeni kod, zajedničke procedure i tablicu simbola. Neka pojedini dijelovi programa imaju sljedeće memorijske zahtjeve:

prvi prolaz 70k
drugi prolaz 80k
tablica simbola 20k
zajedničke procedure 30k

Ukoliko se program cjelovito izvodi potrebno je minimalno 200k glavne memorije. Ukoliko sustav raspolaže sa samo 150k radne memorije ovaj program nije moguće izvesti. Analizom programa može se primijetiti da nije potrebno istovremeno u memoriji držati kod za prvi i drugi prolaz. Tako je praktično program moguće podijeliti na dva podskupa (*overlay*), prvi koji ima kod za prvi prolaz, zajedničke rutine i tablicu simbola i drugi dio koji sadrži kod za drugi prolaz, zajedničke rutine i tablicu simbola. Prvi dio zahtijeva 120k radne memorije, a drugi 130k. Ovim pristupom moguće je program izvoditi sa samo 150k radne memorije. Naravno izvođenje programa biti će nešto sporije jer je potrebno dodatno upisivanje dijela programa u radnu memoriju. Slika 8.2 prikazuje ovakav pristup izvođenju programa.

Kao i dinamičko punjenje, ovakav pristup ne zahtijeva posebnu podršku od strane operacijskog sustava. Njegova implementacija moguća je primjenom sustava datoteka. Dio programa koji je potrebno naknadno izvesti upisuje se u memoriju preko datoteke koja više nije potrebna i izvođenje se nastavlja od njene početne adrese kao što je prikazano slikom 8.2. Naravno, zasebni algoritmi povezivanja (*linking*) i dodjeljivanja apsolutnih adresa (*relocation*) potrebni su prilikom stvaranja modula.

Ovakav sustav s druge strane postavlja dodatne zahtjeve na programera. On mora znati kako ovaj sustav funkcionira kako bi stvorio zasebne module.



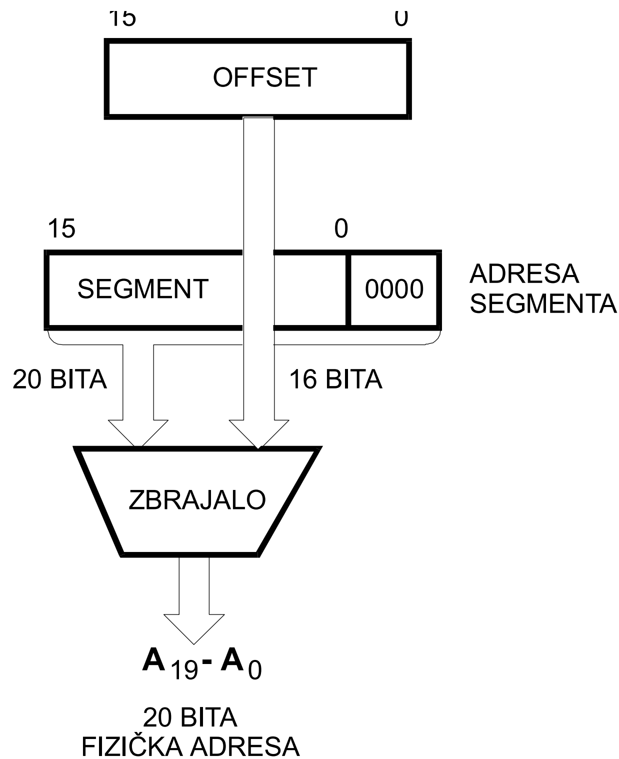
Slika 8.2: *Program prevodilac simboličkog jezika realiziran u dva modula (overlay).*

7.2 Logički i fizički adresni prostor

Adrese koje se koriste u izvornom programu su simboličke. Nakon prevođenja i povezivanja program sadrži logičke adrese. Adresa u memoriji, odnosno adresa koju vidi sklop za upravljanje memorijom (*Memory Management Unit* ili *MMU*) nazivaju se fizičke adrese.

Za vrijeme prevođenja i povezivanja logičke i fizičke adrese su jednake. Tijekom izvođenja, transformacija adresa rezultira u različitim fizičkim i logičkim adresama. U tom slučaju logičke adrese nazivaju se virtualne adrese. Sukladno navedenim definicijama, adresni prostor koji koristi program prije punjenja u memoriju naziva se logički adresni prostor koji za vrijeme izvođenja (nakon punjenja u memoriju) prelazi u fizički adresni prostor. Tako za vrijeme izvođenja, logički i fizički adresni prostori se razlikuju.

Funkciju preslikavanja iz logičkog u fizički adresni prostor realizira zasebno sklopovlje nazvano sklop za upravljanje memorijom (*memory management unit*). Naravno, postoje brojna rješenja ove funkcije koja će biti obrazložena u sljedećim razmatranjima. Kao primjer na slici 8.3 prikazan je sustav za generiranje fizičkih adresa temeljom logičkih adresa kod procesora Intel 80x86. Logičke adrese programa počinju od adrese 0000H do FFFFH. One se preslikavaju linearno dodavanjem logičkoj adresi sadržaja baznog spremnika pomaknutog za četiri mjesta u lijevo (pomnožen s 16). Ovim pristupom logički adresni prostor od 64k koji vidi korisnik, a koji počinje na adresi 0000H, preslikava se u fizički adresni prostor unutar 1M, a koji počinje na adresi koju određuje bazni spremnik.

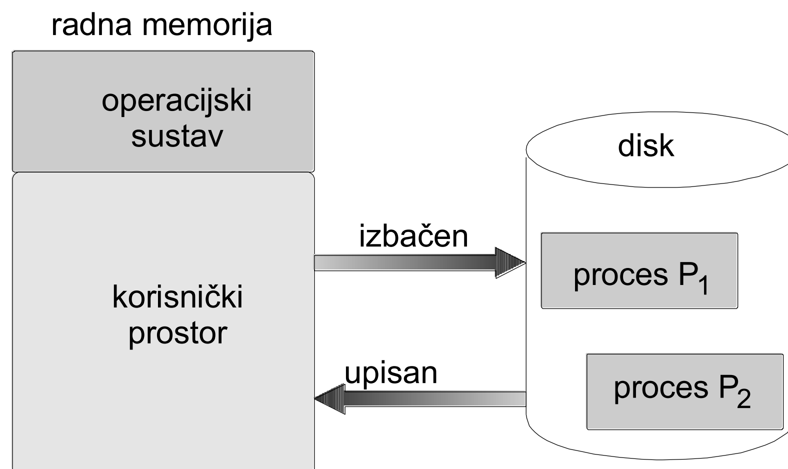


Slika 8.3: *Određivanje apsolutne adrese kod procesora I80x86.*

Koncept preslikavanja logičkog u fizički adresni prostor temelj je sljedećih razmatranja i značajan je za učinkovito korištenje računarskog sustava.

7.3 Prebacivanje (*swapping*)

Proces se može izvoditi samo ako se nalazi u radnoj memoriji. U više programskoj sredini koja ima tendenciju da učinkovito iskoristi resurse računala, više procesa moguće je konkurentno izvoditi tako da se proces koji se izvodi upiše u memoriju, a kada se prekida s njegovim izvođenjem (temeljom različitih događaja koji rezultiraju gubitkom prava korištenja procesora) on se zamijeni s drugim procesom s diska. Npr. ukoliko jednom procesu koji se izvodi istekne dodijeljeni kvant vremena, proces se prebacuje na disk. Time se oslobađa radna memorija, a u memoriju se upisuje jedan od procesa iz reda pripravnih (ovisno o algoritmu dodjele procesora), slika 8.4.



Slika 8.4: *Dodjela memorije prebacivanjem (swapping).*

Ovakav pristup rješavanja dodjele memorije u više procesnom sustavu jednostavan je za implementaciju uz nedostatak što je vrijeme prebacivanja (*context-switch*) relativno veliko. Kao primjer može se uzeti zamjena dvaju programa iste veličine od 100k. Neka sustav koristi disk prosječnog vremena pristupa od 15ms te brzine prijenosa 10M/s. Ukupno vrijeme zamjene iznosi:

$$T_{zamjene} = T_{pristua_{P1}} + T_{prijenosa_{P1}} + T_{pristua_{P2}} + T_{prijenosa_{P2}} = 15 + \frac{100}{10} + 15 + \frac{100}{10} = 50\text{ms}.$$

Zamjena procesa zamišljena je s ciljem povećanja iskoristivosti računalnog sustava. Ali ukoliko je vrijeme zamjene reda veličine vremena obrade pojedinog procesa, što vrijedi za ovaj primjer, tada je ovakav pristup neprihvatljiv.

Dodatan problem je što proces da bi bio izbačen iz radne memorije mora biti potpuno neaktivan. Naime, ukoliko je proces zahtijevao U/I operaciju, a zatim je izbačen iz memorije, podaci koji se prosljeđuju ili dolaze s U/I uređaja prepisani su odnosno pripadaju drugom procesu. Zato je potrebno da proces obavi U/I operaciju a tek onda da bude zamijenjen. Time je dodatno narušen koncept višeprocesnog rada.

Kao posljedica navedenih nedostataka ovakav pristup praktički se ne koristi u današnjim računalnim sustavima. Ali on je poslužio kao polazište novijim, sofisticiranijim algoritmima za dodjelu i upravljanje radnom memorijom.

7.4 Kontinuirana dodjela memorije, dodjela po particijama

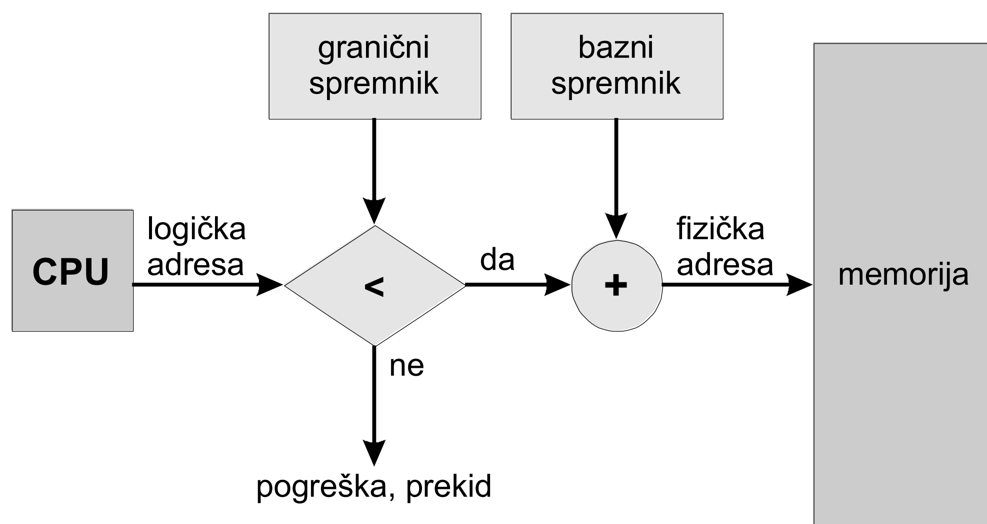
U glavnoj memoriji nalazi se operacijski sustav i korisnički program. Tako kod najjednostavnijih sustava memorija se dijeli na dva dijela, jedan koji koristi operacijski sustav, a drugi koji koristi korisnički program ili programe, slika 8.5. Operacijski sustav može se nalaziti ili na memorijskim lokacijama s najnižim ili najvišim adresama što ovisi o sklopovskoj realizaciji sustava. Kod nekih operacijskih sustava, kao što je npr. MS-DOS dio operacijskog sustava postavljen je na najviše memorijske adrese, a drugi dio na najniže. Ovo je opet rezultat sklopovskih rješenja samog računala. Dio operacijskog sustava upisan je u neizbrisivoj memoriji (ROMu), a ostatak u RAMu. Ovakvo rješenje osigurava veću fleksibilnost računarskog sustava na način da se osnovni operacijski sustav nalazi u ROMu, a u RAMu sve što je podložno promjenama sustava (pogonski programi za tipkovnicu, miša, disketnu jedinicu, magnetski disk, optički disk, itd.).

Slika 8.4: *Podjela memorije.*

7.4.1 Sustav s jednom particijom

Neka se operacijski sustav nalazi na nižim adresama, a više adrese koristi korisnički program. U memoriji može se nalaziti samo jedan korisnički program.

Logičke adrese korisničkog programa potrebno je linearno pomaknuti kako bi se stvorile fizičke adrese. Kako je već opisano ovo je moguće realizirati u različitim fazama pripreme programa za izvođenje. Najjednostavnije je fizičke adrese generirati tijekom povezivanja programa. Ukoliko dođe do promjena u veličini operacijskog sustava tada se program mora ponovo povezivati što je neprikladno. Bolje rješenje je fizičke adrese generirati za vrijeme punjenja programa u memoriju. Program punjač zna početnu adresu programa i preslikava logičke adrese u fizičke. Kako operacijski sustavi mogu dinamički tijekom rada mijenjati svoju veličinu (npr. kao posljedica međuspremnika U/I operacija) tada ni ovaj pristup nije prihvatljiv. Poboljšanje se postiže dodatnim sklopovljem koje je prikazano na slici 8.5.



Slika 8.6: *Sklopovska podrška generiranju fizičke adrese linearnim pomakom logičke adrese.*

Sklop za upravljanje memorijom prvo provjerava da li je logička adresa unutar dozvoljene granice, a zatim se logička adresa pomiče za iznos baznog spremnika. Ukoliko logička adresa nije unutar dozvoljene granice generira se procesoru prekid.

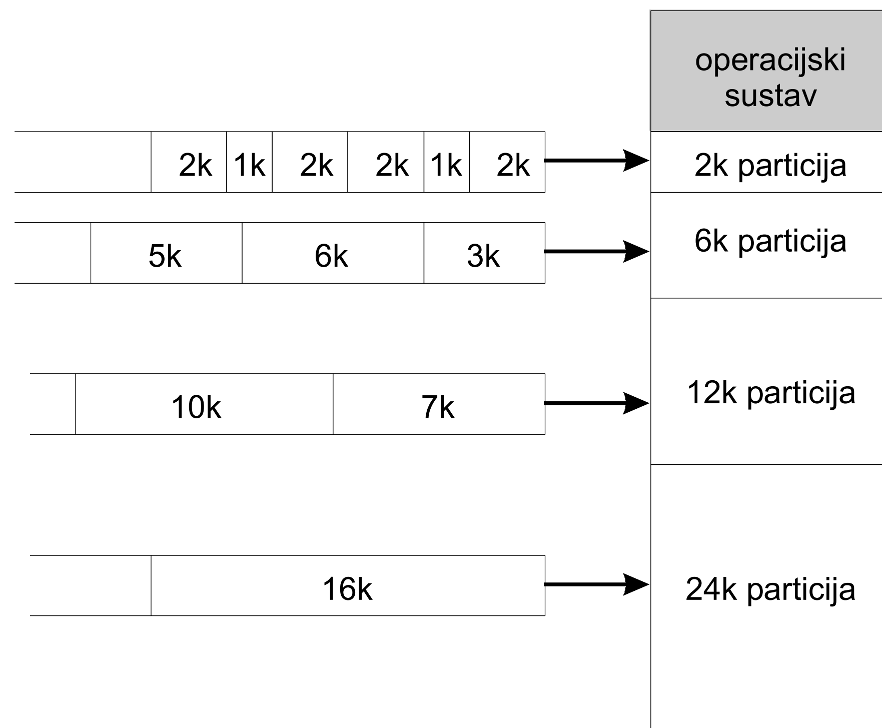
Ovakav sklop za upravljanje memorijom omogućava da se mijenja veličina operacijskog sustava. Naime, ukoliko se uspostavi da raspoloživa memorija je nedostatna korisničkom programu, moguće je iz memorije izbaciti dio operacijskog sustava koji je nepotreban i time osloboditi dodatni memorijski prostor.

7.4.2 Sustav s više particija

Kod sustava s jednom particijom, ukoliko se implementira višeprocesni rad, potrebno je prilikom promjene konteksta izbacivati prekinuti proces na disk i upisati drugi u radnu memoriju (*swapping*). Kako je već objašnjeno ovakav pristup zahtijeva neprihvatljivo veliko vrijeme izmjene konteksta. Kako su u međuvremenu značajno povećani kapaciteti glavne memorije nametnulo se rješenje podjele memorije na više dijelova, particija. U svakoj particiji smješta se po jedan korisnički proces. Stupanj višeprogramskog rada određen je brojem particija. Korisnik kada starta svoj program, njega operacijski sustav postavlja na disk kao proces koji je prihvaćen na obradu ali nije još dobio pravo da se natječe za korištenje procesora (izvođenje). Kada se oslobodi u

sustavu jedna memorijska particija, program se upisuje u nju i postaje proces koji može dobiti pravo korištenja procesora. Po završetku obrade proces se izbacuje iz memorije i slobodna particija dodjeljuje se sljedećem procesu koji čeka na disku.

Broj particija može biti konstantan ili promjenjiv. Svaka particija sadrži jedan jedinstveni proces. U prvobitnim verzijama koristio se konstantan broj particija, a svaka particija imala je svoj vlastiti red čekanja na disku (*long term scheduler*), slika 8.7. Procesi su se pripremali za određenu particiju u ovisnosti o njegovim memorijskim zahtjevima. Zahtjev za memorijom je određivao korisnik ili se proračunavalo automatski. Za napomenuti je da su se u to vrijeme koristili primarno statički programski jezici koji nisu mogli zahtijevati dodatnu memoriju tijekom izvođenja (FORTRAN).



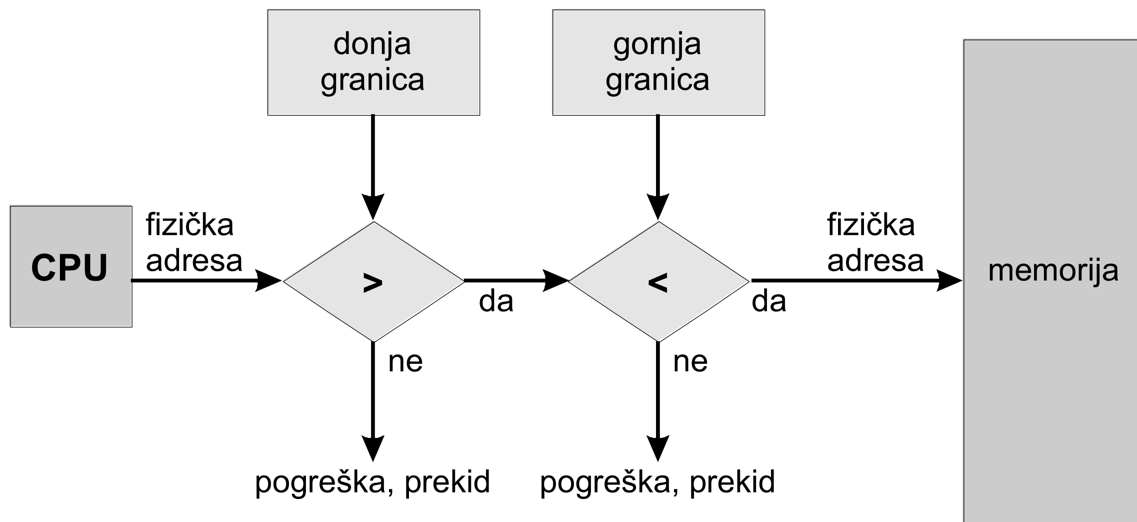
Slika 8.7.: *Memorijski sustav s 4 particije fiksne veličine.*

Procesi pripremljeni za određenu particiju nisu se mogli izvoditi u drugim particijama, jer je program punjač određivao fizičke adrese. Zato se ovakav način dodjele memorije naziva statička dodjela memorije.

Procesi upisani u memoriju dijelili su resurse sustava i izvodili su se paralelno. Koji od procesa je trenutno aktivan određivao je sustav za upravljanje radom procesora, opisan u poglavlju 5. Izbacivanje procesa iz jedne particije na disk i upis novog procesa na njegovo mjesto, odvijalo se paralelno radu procesora. Procesor bi obrađivao proces iz druge particije dok bi operaciju zamjene obavljao U/I procesor.

Ovakav način dodjele memorije koristio se kod IBM 360 računala i poznat je pod nazivom OS/MFT (*Multiprogramming with Fixed number of Tasks*) sa zasebnim redom čekanja za svaku pojedinu particiju. Ovakav pristup dodjeli memorije nije više u uporabi, a primarno se koristio kod operacijskih sustava sa skupnom obradom.

Ovakav sustav zahtijevao je i dodatnu zaštitu kao se korisnici ne bi međusobno ugrožavali i da ne bi ugrožavali operacijski sustav. Uvedenena je kontrola donje i gornje granice memorijskog prostora koje smije koristiti svaki pojedini proces. Ove granice određene su particijom koje proces koristi.

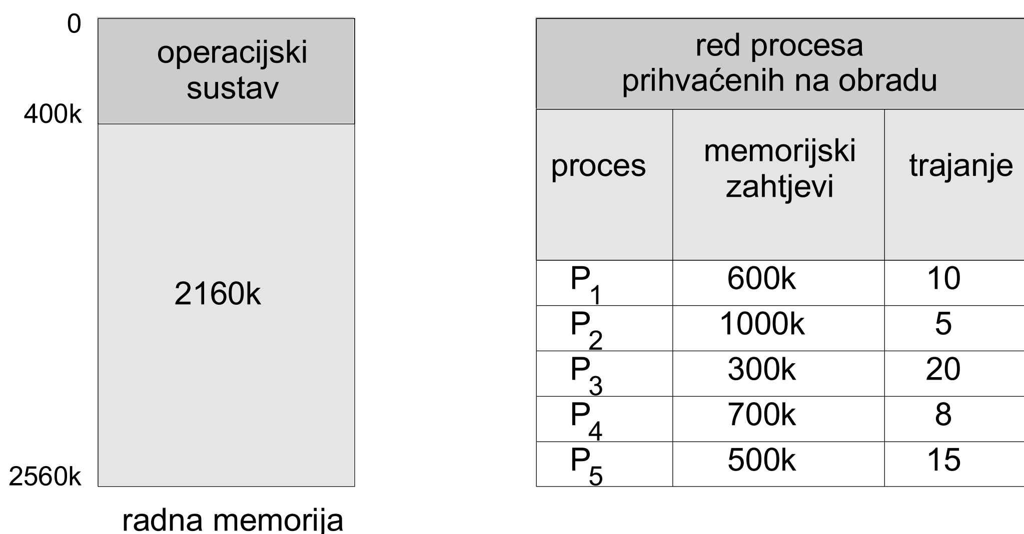


Slika 8.8.: *Provjera granica fizičke adrese.*

Nedostatak ovog rješenja je što bi u praksi znale neke particije biti prazne dok bi za druge postojali poveći redovi čekanja. Ovo je naročito bio slučaj kada bi operater pogrešno procijenio veličine particija. Također ukoliko je program bio toliki da nije mogao stati u ni jednu particiju (sustav ima za taj program dovoljno radne memorije) moralo se podnositi poseban zahtjev operateru kako bi se mogao izvesti navedeni program.

Bolje rješenje je sustav za upravljanje memorijom s promjenjivim brojem particija. Operacijski sustav vodi evidenciju o zauzetoj i slobodnoj memoriji. Kod ovog pristupa početno je cijeli korisnički memorijski prostor tretiran kao jedna particija. Prvi proces koji se prihvati na obradu dobiva na korištenje onoliko memorijskog prostora koji mu je potreban, koji je smješten odmah iznad operacijskog sustava, te taj prostor operacijski sustav bilježi kao zauzet. Prihvaćanjem sljedećeg procesa operacijski sustav ispituje da li je raspoloživi slobodni prostor dovoljan za ovaj proces i ako je njemu dodjeljuje prostor odmah iza prvog procesa. Ovaj postupak se nastavlja dok se mogu zadovoljiti zahtjevi dolazećih procesa. Kada u međuvremenu neki od procesa završi s obradom, oslobađa se memorija koju je on zauzimao i ona se može dodijeliti drugom procesu.

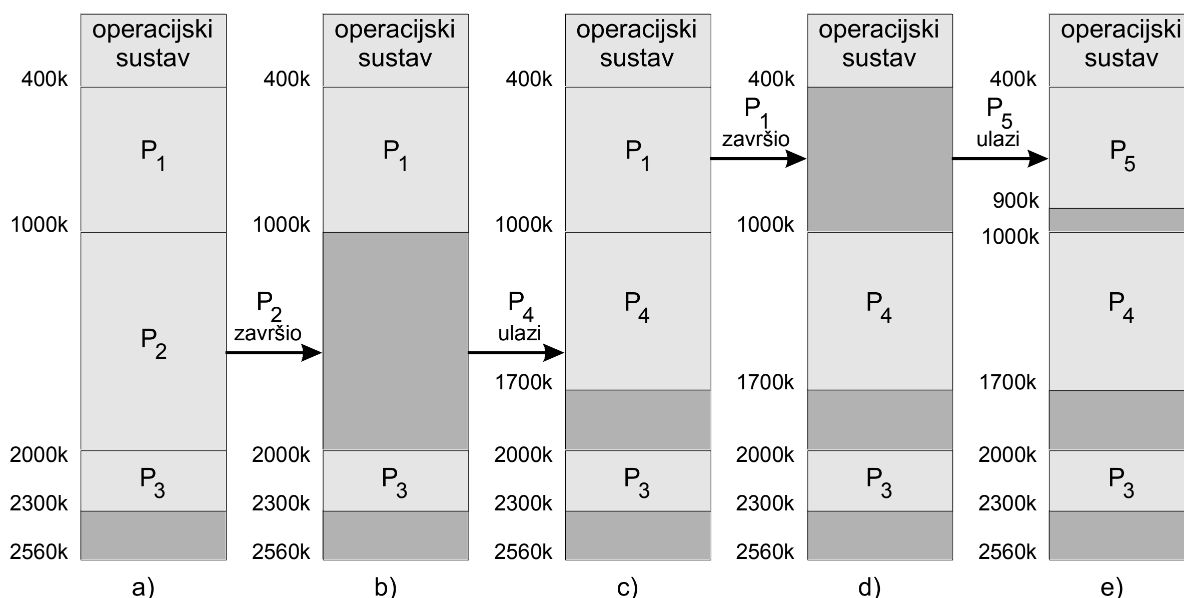
Princip rada , te problemi koji se susreću kod ovakvog sustava za upravljanje memorijom pojasniti će se na sljedećem primjeru. Neka sustav ima na raspolaganju 2560k memorije od koje operacijski sustav koristi 400k. Korisniku je znači na raspolaganju ostalo 2160k memorije. Neka sustav prihvati na obradu procese P_1 – P_5, trajanja upisanog u tablici na slici 8.9. i neka se za dodjelu procesora koristi Roun-Robin algoritam s vremenskim kvantom 1. Sustav za prihvat poslova na obradu u memoriju će prvo upisati proces P_1 i započeti s njegovim izvođenjem. U tom trenutku zauzeto je 1000k memorije, a slobodno 1560k. Paralelno izvođenju procesa P_1 upisati će se u slobodnu memoriju i proces P_2 čiji zahtjevi na memoriju su manji od slobodnih kapaciteta. Sada je zauzeto 2000k memorije, a slobodno 560k. Slobodan prostor dovoljan je da se upiše proces P_3 nakon čega ostaje slobodno svega 260k memorije što nije dovoljno da se upišu sljedeći procesi, slika 8.10.a).



red procesa prihvaćenih na obradu		
proces	memorijski zahtjevi	trajanje
P_1	600k	10
P_2	1000k	5
P_3	300k	20
P_4	700k	8
P_5	500k	15

Slika 8.9: *Primjer za sustav za upravljanje memorijom s promjenjivim brojem particija.*

Procesi upisani u memoriju, P_1, P_2 i P_3 izvoditi će se naizmjenično, a prvi će završiti proces P_2 nakon 14 vremenskih jedinica i osloboditi će zauzetu memoriju, slika 8.10.b). Na slobodno mjesto upisati će se proces P_4, slika 8.10.c). Nakon 28 vremenskih jedinica završiti će proces P_1 i osloboditi zauzetu memoriju, slika 8.10.d). Na slobodno mjesto upisati će se proces P_5, slika 8.10.e).



Slika 8.10: *Dodjela memorije i prihvaćanje procesa na obradu.*

Pomoću ovog primjera moguće je ilustrirati određena svojstva opisanog sustava za dodjelu memorije. Proces prihvaćen na obradu upisuje se u memoriju kada je u memoriji slobodan segment (particija) veći od memorijskih zahtjeva samog programa. Programi upisani u memoriju konkurentno se izvode. Kada program završi s izvođenjem oslobađa zauzetu memoriju.

U ovakvom sustavu dodjele memorije privilegirani su manji programi. Naime, s vremenom u memoriji se nalazi veći broj manjih programa koji su neravnomjerno raspoređeni, a također i veći broj slobodnih mjesta manjeg kapaciteta, tzv. šupljina, koji su također razbacani po memoriji. Veći programi se ne mogu upisati u memoriju jer ne

postoji kontinuirani slobodni prostor potrebne veličine iako je ukupna slobodna memorija dovoljna. Ovaj problem naziva se fragmentacija memorije. Uz nju je povezan i sljedeći problem. Neka postoji više (m) slobodnih segmenata u koji se može upisati sljedeći proces. Postavlja se pitanje koji od njih odabrati. Moguća su sljedeća rješenja:

- Prvi koji zadovoljava (*first-fit*): Procesu se dodjeljuje prvi segment koji zadovoljava postavljene memorijske zahtjeve. Obično pretraživanje počinje od početka liste slobodnih segmenata ili se nastavlja od mjesta gdje je prethodno ispitivanje zaustavljeno.
- Najbolje poklapanje (*best-fit*): Procesu se dodjeljuje onaj segment koji nabolje odgovara njegovim memorijskim zahtjevima. Iako na prvi pogled ovim pristupom se najbolje iskorištava slobodna memorija, rezultat je stvaranje malih segmenata, šupljina, koji su posljedica razlike veličine segmenta i programa.
- Najlošije poklapanje (*worst-fit*): Procesu se dodjeljuje najveći slobodni segment. Ovaj algoritam ima za cilj stvaranje što većih šupljina, suprotno prethodnom algoritmu.

Provedene simulacije pokazale su da su prva dva algoritma bolja od posljednjeg u smislu bolje iskoristivosti memorije kao i prosječnog vremena izvođenja procesa.

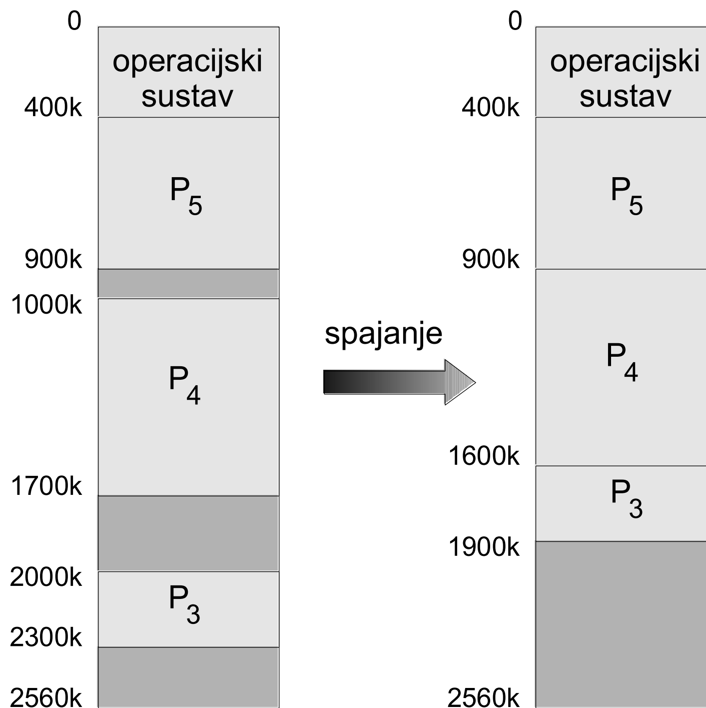
7.4.3 Vanjska i unutarnja fragmentacija

Kao što je već opisano kako se procesi upisuju te izbacuju iz memorije slobodna memorija razbija se u manje segmente. Tako s vremenom iako je ukupna veličina slobodne memorije dovoljna da prihvati nove procese to nije moguće iz razloga što ne postoje dovoljno velika kontinuirana područja. To se naziva vanjska fragmentacija memorije. Prema primjeru na slici 8.10.a) ukupna vanjska fragmentacija iznosi 260k i odnosi se na samo jednu šupljinu. Na slici 8.10.c) ukupna vanjska fragmentacija iznosi 560k (300k+260k) i ovaj prostor bio bi dostatan za upis procesa P_5 veličine od 500k da se radi o kontinuiranom segmentu.

Fragmentacija može postati ozbiljan problem koji značajno umanjuje učinkovitost sustava. U najgorem slučaju moguće je da se između svih procesa nalazi šupljina, koja bi, kada bi se spojila u jedinstvenu cjelinu, bila dostatna za obradu nekoliko dodatnih procesa. Algoritam za odabir slobodne particije u koju će se smjestiti novi proces, prvi koji zadovoljava, najbolje poklapanje ili najgore poklapanje, značajnu utječe na stupanj fragmentacije. Statističke analize za algoritam “prvi koji zadovoljava” pokazuju da ukoliko je dodijeljeno N memoriskih blokova, prosječno je $0.5 \cdot N$ blokova neiskorišteno kao posljedica fragmentacije. Ova analiza govori da je praktički trećina memorije niskorištena. Ovo svojstvo poznato je pod nazivom 50% pravilo.

Kod dodjele memorije susreće se i sljedeći problem. Pretpostavimo da postoji slobodan memorijski blok veličine 18,464 okteta, a program zahtjeva 18,460 okteta. Tada je razumno dodijeliti programu cijeli blok, bez obzira što će 4 okteta ostati neiskorišteno. Ovo je bolje nego da ta četiri okteta tvore slobodan oktet za koji je potrebno voditi evidenciju u operacijskom sustavu. Tako dodijeljena memorija može biti veća od memorije koju zahtjeva proces. Razlika u zahtjevanoj i dodijeljenoj memoriji naziva se unutarnja fragmentacija, koja je praktički zanemariva u usporedbi s vanjskom.

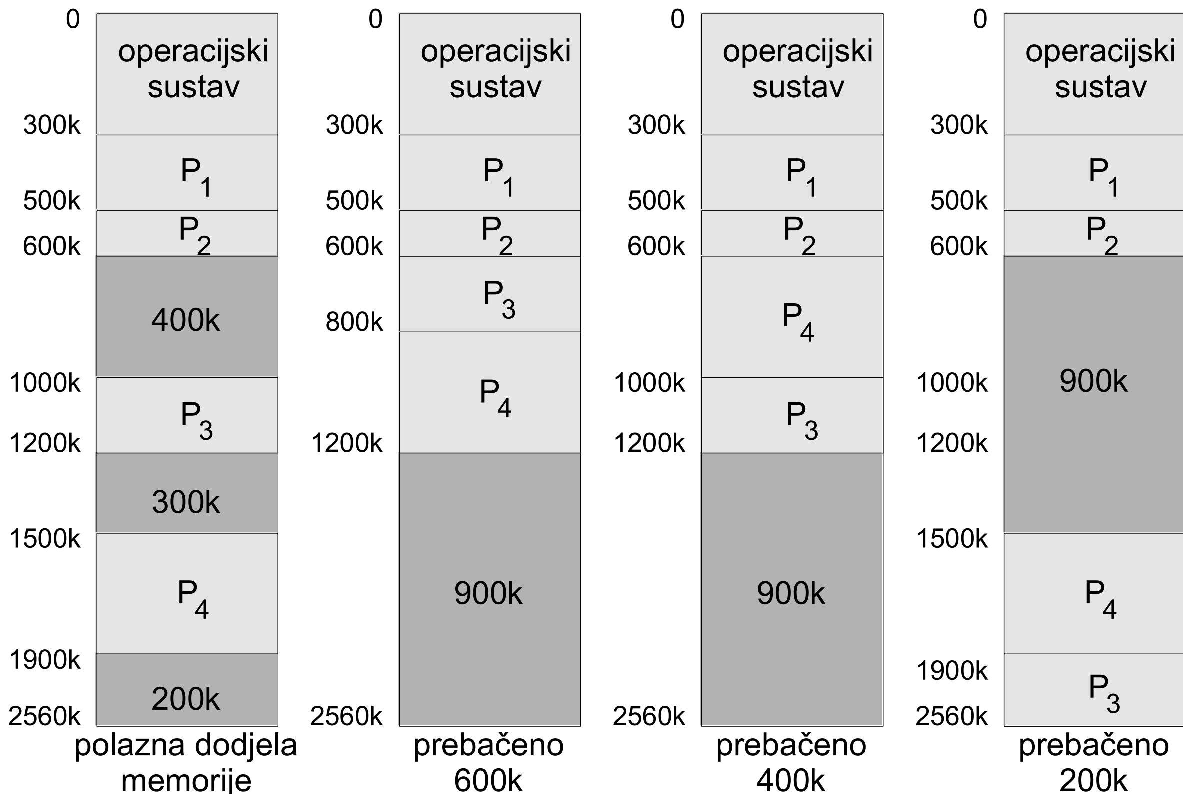
Problem vanjske fragmentacije rješava se premještanjem procesa i spajanjem slobodnih blokova u jedan kontinuirani blok (*compaction*). Tako u primjeru na slici 8.10.e) spajanje se može izvesti premještanjem procesa P_3 i P_4 kao što je prikazano slikom 8.11.



Slika 8.11: *Spajanje slobodnih memorijskih segmenata.*

Spajanje nije uvijek moguće provesti. Prema slici 8.11. može se primijetiti da su procesi P_4 i P_3 premješteni. Ovo nije moguće učiniti ukoliko su apsolutne, odnosno fizičke adrese ovih procesa generirane za vrijeme pripreme programa za izvođenje ili za vrijeme punjenja. Realizacija spajanja moguća je jedino ukoliko se fizičke adrese generiraju dinamički, odnosno za vrijeme izvođenja programa. Kada se fizičke adrese generiraju dinamički, potrebno je samo premjestiti procese i promijeniti sadržaje baznih spremnika.

Spajanje slobodnih memorijskih blokova premještanjem svih procesa prema jednom kraju memorije rezultira u nepotrebno velikom utrošku vremena. Zato je potrebno pažljivo odrediti postupak spajanja slobodnih blokova. Primjer je prikazan na slici 8.12. Ukoliko se svi procesi prema redoslijedu pomiču prema manjim memorijskim lokacijama potrebno je premjestiti 600k koda i podataka. U drugom slučaju može se zaključiti da je potrebno prebaciti samo proces P_4 da bi se dobila ista situacija. Tada je premješteno samo 400k koda. Detaljnijim ispitivanjem može se zaključiti da je dovoljno premjestiti proces P_3 na vrh memorijskog prostora. U posljednjem slučaju premješta se samo 200k koda i podataka. Primjer pokazuje složenost odabira optimalne strategije spajanja slobodnih memorijskih blokova.



Slika 8.12: *Primjeri različitih strategija spajanja slobodnih memorijskih blokova.*

7.5 Straničenje (*paging*)

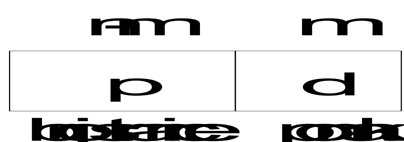
Moguće rješenje problema fragmentacije je da se dozvoli da proces ne mora biti upisan u kontinuirani memorijski blok. Program može biti raspršen proizvoljno po memoriji.

7.5.1 Princip dodjele memorije po stranicama

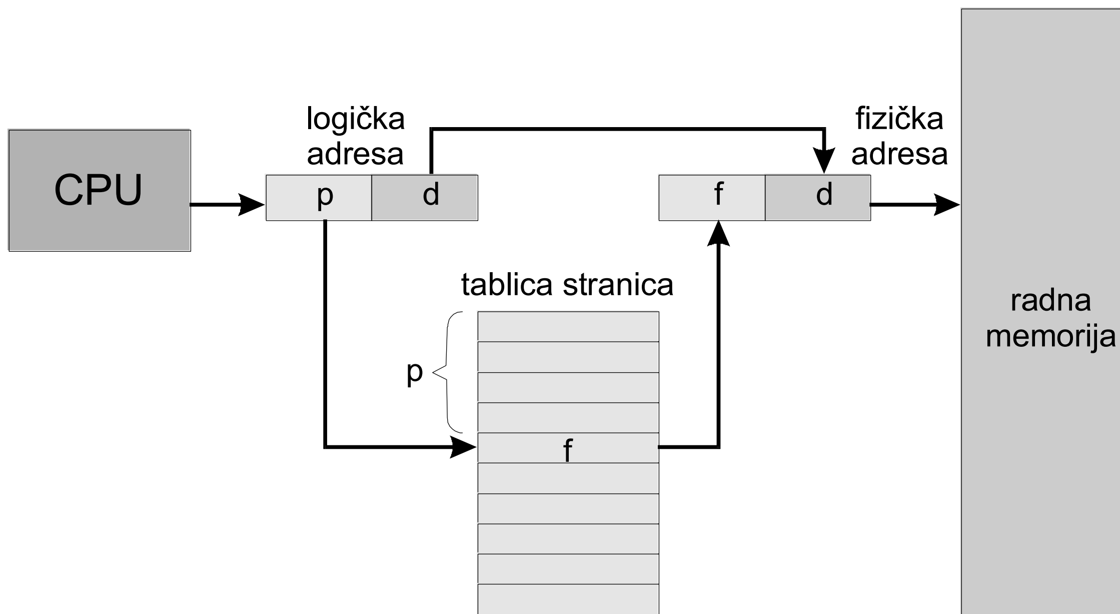
Radna memorija dijeli se na manje blokove fiksne veličine koji se nazivaju okviri (*frames*). Logički adresni prostor programa također se podjeli na blokove iste veličine koji se nazivaju stranice (*pages*). Kada se program upiše u memoriju stranice se upisuju u slobodne memorijske okvire. Radi jednostavnosti prebacivanja programa s diska u radnu memoriju i disk je podijeljen na okvire jednake veličine kao i okviri memorije. Tako se jedan okvir s diska upisuje u jedan okvir radne memorije.

Sklopovlje koje podržava ovakav pristup dodjeli memorije prikazano je na slici 8.13. Adresa koju generira procesor dijeli se na dva dijela: broj stranice () p i pomak unutar stranice (*page offset*) d. Broj stranice je indeks (pokazivač) na redak tablice stranica. U tablici stranica upisane su početne adrese okvira u kojima je smještena stranica. Kombinacija početne adrese okvira i pomaka određuje fizičku adresu memorijske lokacije kojoj se pristupa.

Veličine stranica variraju od sustava do sustava i obično su između 512 okteta i 8192 okteta i uvijek su potencija broja dva. Ovakvim pristupom jednostavno je podijeliti logičku adresu na broj stranice i pomak unutar stranice. Za n bitovnu logičku adresu

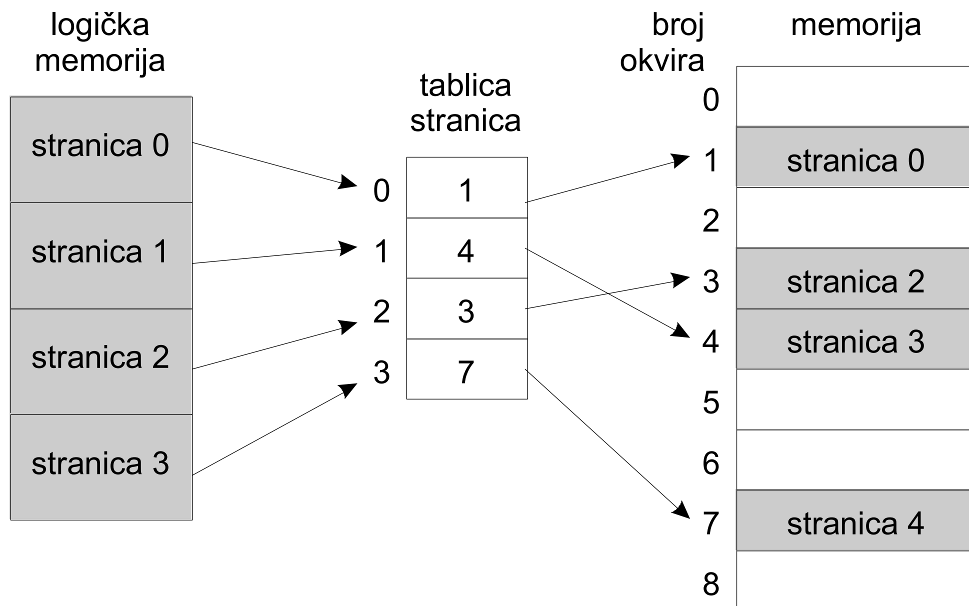


n-m bita većeg značenja određuje broj stranice, a m bita manjeg značenja pomak.



Slika 8.13: *Sklopovlje za dodjeljivanje memorije po stranicama.*

Model dodjele memorije po stranicama prikazan je slikom 8.14.



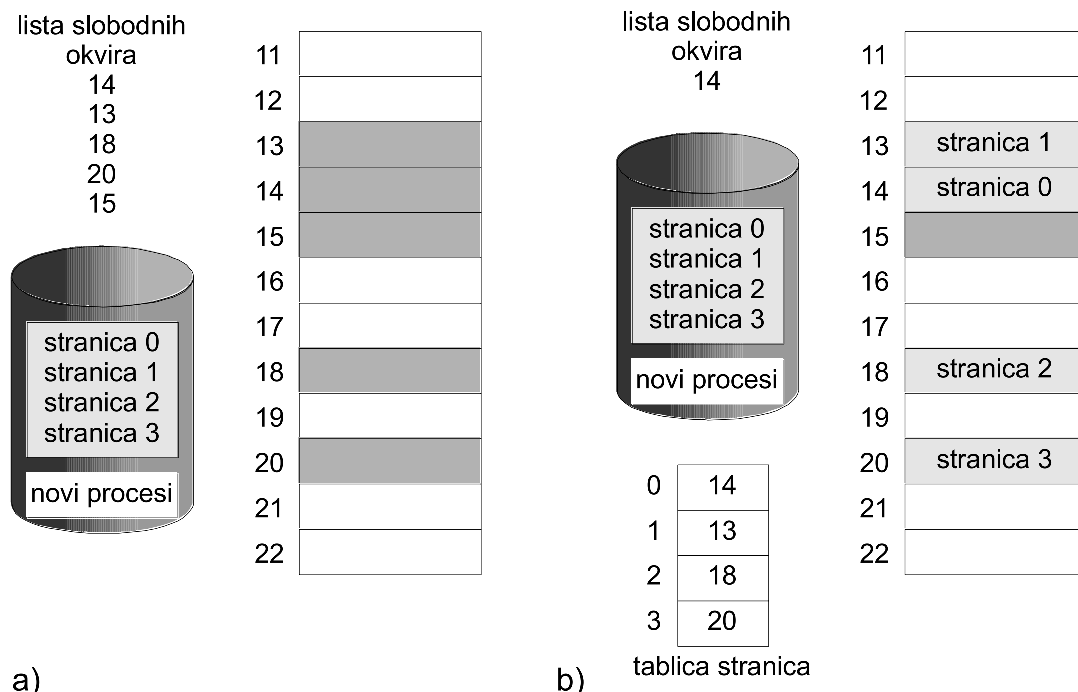
Slika 8.14: *Model dodjele memorije po stranicama.*

Dodjela memorije po stranicama oblik je dinamičke dodjele memorije. Sklopovlje vezuje svaku logičku adresu na određenu fizičku adresu. Tablica stranica je više baznih spremnika.

Primjenom dodjele memorije po stranicama nestaje efekt vanjske fragmentacije radne memorije. Svaki okvir može biti dodijeljen procesu koji ga treba. Naravno postoji problem unutarnje fragmentacije budući da se okviri dodjeljuju kao cjeline. Tako se dešava da je posljednji okvir nepopunjen. Npr. ukoliko su stranice veličine 2048 okteta, a proces zahtjeva 72,766 okteta, njemu će se dodijeliti 36 okvira od kojih će 35 biti

potpuno popunjeno, a u posljednjem će se nalaziti 1086 okteta. Rezultat je unutarnja fragmentacija posljednjeg okvira od 2048 – 1086 = 962 okteta. U najgorem slučaju unutarnja fragmentacija iznosi veličina okvira – 1 oktet, a može se očekivati prosječna u iznosu polovice veličine okvira. Sa stajališta unutarnje fragmentacije povoljnije je da su okviri manje veličine. Današnji sustavi koriste okvire veličine između 2k i 4k. Mora se napomenuti da je problem unutarnje fregmentacije sa stajališta iskoristivosti memorije praktički zanemariv.

Sustav za dodjelu memorije po stranicama djeluje na sljedeći način: Kad se program prihvati na izvođenje izračuna se potreban broj okvira i uspoređuje se s brojem slobodnih okvira u memoriji. Ukoliko je slobodan dovoljan broj okvira proces se upisuje u memoriju stranicu po stranicu. Istovremeno se za svaku stranicu u tablici stranica upisuje i broj okvira u koji je ona upisana. Ovaj proces prikazan je na slici 8.15.



Slika 8.15: *Slobodni okviri; a) prije dodjele; b) poslije dodjele memorije procesu.*

Kod sustava za dodjelu memorije po stranicama važno je razlučiti između korisničkog viđenja memorije koju program koristi i stvarne fizičke memorije koja se koristi tijekom njegovog izvođenja. Korisnik vidi memoriju koju zahtijeva program kao kontinuirani prostor, dok je stvarni program kao i podaci razbacani po fizičkoj memoriji. Razliku između korisničkog viđenja memorije i stvarnog korištenja memorije tvori sklopovlje za preslikavanje adresa. Preslikavanje logičkog u fizički adresni prostor transparentno je (skriveno) za korisnika. Također, važno je primijetiti da korisnički program nije u mogućnosti pristupati memorijskim lokacijama izvan svoje tablica stranica. Time je i realizirana zaštita među programima.

Upravljanje memorijom je u nadležnosti operacijskog sustava. On mora voditi evidenciju o stanju memorije: koji su okviri zauzeti, koji proces koristi pojedine okvire koji su okviri slobodni. Ovi podatci obično se pohranjuje u tablici nazvanoj tablica okvira (*frame table*). Ova tablica ima redak za svaki okvir koji pokazuje da li je okvir slobodan ili ne i ako nije koji ga proces ili procesi koriste.

Korisnički proces izvodi se u korisničkom adresnom prostoru. Ukoliko proces izvodi sistemski poziv (npr. U/I operaciju) i prosljeđuje sistemskom pozivu adresu kao

parametar, operacijski sustav se mora pobrinuti da se adresa ispravno preslika kako bi se proslijedila ispravna fizička adresa. Operacijski sustav održava kopiju tablice stranica na isti način kao što održava kopiju programskog brojila i sadržaje spremnika. To je razlog da ovakav sustav za dodjelu i upravljanje memorijom povećava vrijeme izmjene konteksta.

7.5.2 Struktura tablice stranica

Svaki operacijski sustav koristi vlastiti pristup za pohranu tablice stranica. Većina dodjeljuje svakom procesu vlastitu tablicu stranica. Pokazivač na nju sastavni je dio deskriptora procesa slično kao i sadržaj programskog brojila i spremnika. Kada se proces prebacuje iz stanja pasivan u stanje aktivan potrebno je upisati tablicu stranica procesa u sklopovlje jedinice za upravljanje memorijom. To je i razlog znatnog povećanja vremena izmjene konteksta.

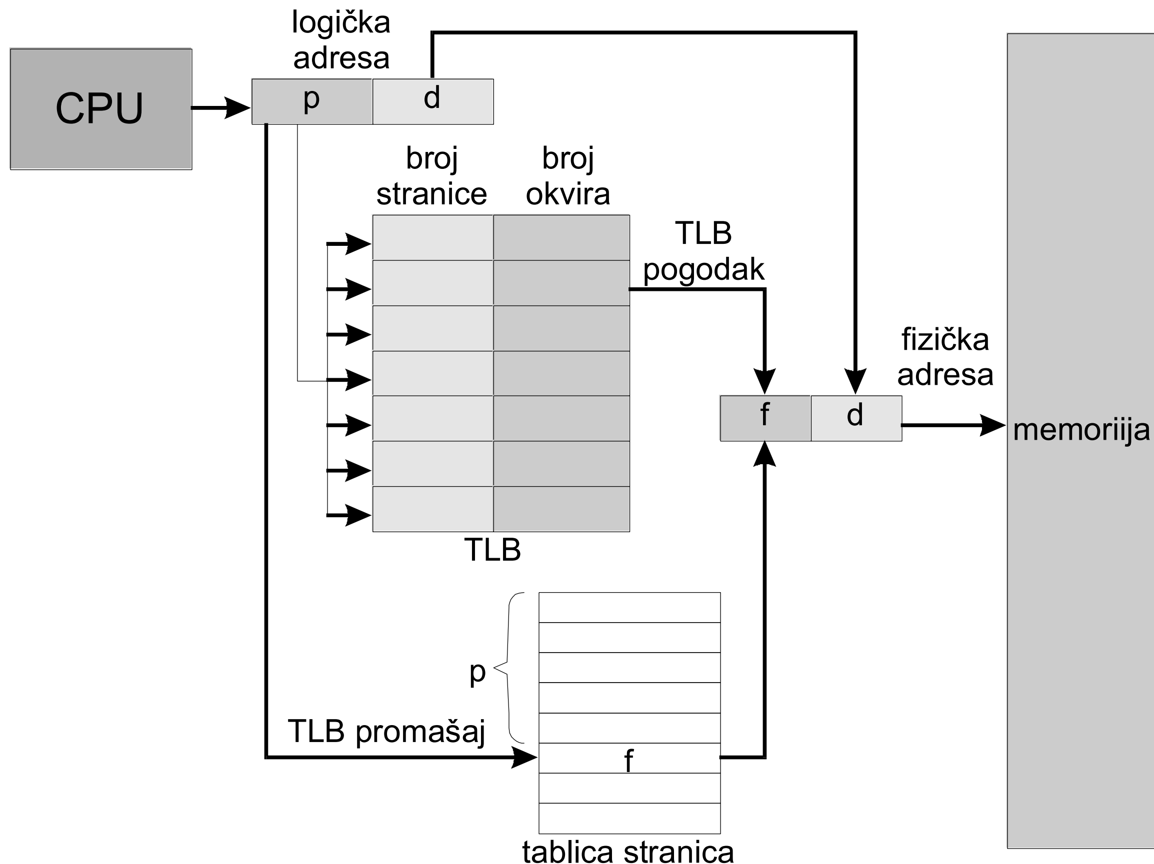
7.5.2.1 Sklopovska podrška sustavu za dodjelu memorije po stranicama

Tablicu stranica moguće je sklopovski ostvariti na više načina. Najjednostavnije je tablicu stranica implementirati kao skup posebnih spremnika. Ovi spremnici moraju biti brzi kako preslikavanje logičkih u fizičke adrese ne bi značajno utjecalo na brzinu rada sustava. Razlog je što svaki pristup memoriji mora proći kroz tu logiku. Prilikom izmjene konteksta u ove spremnike upisuje se tablica stranica procesa iz radne memorije. Naredbe za promjenu sadržaja ovih spremnika su privilegirane. Ovakav princip koristio je DEC PDP-11 čija adresa je imala 16 bita, a stranice su bile veličine 8k. Znači adresni prostor bio je podijeljen na 8 okvira, pa je PDP-11 imao 8 spremnika za tablicu stranica.

Ovakav pristup učinkovit je kada su tablice stranica relativno male, npr. 256 stranica. Ako je veličina okvira 4k to znači da je veličina programa ograničena na 1M. Za brojne aplikacije ovo je nedostatno, pa suvremeni sustavi dozvoljavaju velike tablice stranica, npr. 1M. Realizacija ovakvih tablica pomoću brzih spremnika nije ekonomski opravdana pa se tablica pohranjuje u glavnoj memoriji. Procesor ima spremnik koji sadrži pokazivač na početak tablice stranica (*page-table base register*). Prebacivanjem izvođenja s jednog procesa na drugi zahtjeva samo promjenu sadržaja ovog spremnika. Nedostatak ovog rješenja je u smanjenju brzine obrade, budući da su za pristup podatku u memoriju potrebna dva pristupa memoriji, jedan za dohvat baze okvira, a drugi za dohvat podatka.

Standardno rješenje ovog problema je korištenje asocijativnih spremnika (*translation loo-aside buffers* skr*. TLBs*) koji se sastoje od dva dijela, ključa i vrijednosti. Broj asocijativnih spremnika u zavisnosti o sustavu varira između 8 i 2048. U njih se upisuje dio tablice stranica, a cijela tablica stranica pohranjena je u memoriji. Ključ asocijativnog spremnika odgovara broju logičke stranice, a vrijednost je broj okvira u koji je ta stranica smještena. Procesor generira logičku adresu, a dio koji se odnosi na broj stranice uspoređuje se s ključem asocijativnih spremnika. Ukoliko je stranica pronađena, a to se naziva pogodak, vrijednost odabranog asocijativnog spremnika vodi se na sklop za proračun fizičke adrese. Cijeli ovaj postupak, prikazan na slici 8.16, praktički usporava pristup memoriji za samo 10-20%.

Ukoliko nije stranica upisana u asocijativnim spremnicima, došlo je do promašaja. Tada se pristupa tablici stranica u radnoj memoriji, a ukoliko asocijativni spremnici nisu popunjeni, na slobodno mjesto upisuje se broj stranice i broj okvira u koji je ta stranica pohranjena. Tako je ubrzan sljedeći pristupu toj stranici koji je vrlo vjerojatan s obzirom na običajan tijek izvođenja programa. U slučaju kada nema slobodnih mjesta u asocijativnim spremnicima operacijski sustav prepisuje jedan spremnik podacima nove stranice.



Slika 8.16: *Straničenje pomoću TLBa.*

Kod izmjene konteksta potrebno je izbrisati cijeli sadržaj TLBa što dodatno povećava vrijeme izmjene.

Prosječno povećanje vremena uz opisanu sklopovsku podršku za dodjelu memorije po stranicama može se procijeniti na sljedeći način: neka pristup memoriji iznosi 20ns, a vrijeme pretraživanja TLBa 4ns. Tada je vrijeme proračuna fizičke adrese i pristupa memoriji za slučaj kada nema promašaja iznosi 24ns. U slučaju promašaja potrebno je 4ns da se odredi da stranica nije u asocijativnim spremnicima, 20ns da se pristupi tablici stranica u glavnoj memoriji i dodatnih 20ns za pristup podatku, odnosno ukupno 44ns. Ako je očekivani broj promašaja 20% tada ovakav sustav upravljanja memorijom usporava rad:

$$t_{pristupa} = 0.80 \cdot 24 + 0.2 \cdot 44 = 28\text{ns} \quad \text{ili} \quad \frac{28 - 20}{20} = 40\% .$$

Usporenje za 40% relativno je veliko, ali ukoliko se vjerojatnost promašaja smanji na 2% dobiva se:

$$t_{pristupa} = 0.98 \cdot 24 + 0.02 \cdot 44 = 24.4\text{ns} \quad \text{ili} \quad \frac{24.4 - 20}{20} = 22\% ,$$

što je vrlo blizu maksimalno mogućoj brzini rada.

Svi moderni procesori imaju TLB. Tako Intel 486 ima TLB s 32 spremnika i proklamira vjerojatnost pogotka od 98%.