Operativni sistemi skripta

**NITI – Prva prezentacija**

**Proces** je program u izvršavanju. Procesor izvršava instrukcije procesa, isprepletano sa izvršavanjem drugih procesa. Na procesor „dolaze“ procesi po redosljedu onako kako ih operativni sistem rasporedi. Proces sadrži virtuelni adresni proctor u koji je smještena slika procesa, takođe može mu se dodjeliti vlasništvo nad memorijom, nekim ulazno/izlaznim uređajem ili datotekom.

Stanja procesa:

Izvršavanje - proces čije instrukcije procesor trenutno izvršava

Spreman - proces koji je spreman za izvršavanje, ali trenutno procesor ne izvršava njegove instrukcije

Blokiran (U čekanju) - proces koji se ne može izvršavati dok se ne pojavi neki događaj (npr. završetak U/I operacije)

Novi - proces koji je upravo stvoren, ali ga OS još nije prihvatio u red spremnih procesa

Izlaz - proces koji je OS uklonio iz reda spremnih procesa zato što je završio rad

Sposobnost OSa da podrži više konkurentnih putanja izvršavanja jednog procesa se naziva **višenitna obrada**. Nit je jedan tok izvršavanja u okviru procesa.

Odnos procesa i niti:

|  |  |
| --- | --- |
| Jedan proces – jedna nit   * OS podržava samo jedan proces koji sadrži samo jednu nit | Jedan proces – više niti   * Java VM ( postoji više niti u okviru jednog procesa) |
| Više procesa – jedna nit po procesu   * UNIX | Više procesa – više niti po procesu   * Windows, Linux, Solaris,... |

Proces u višenitnoj obradi sadrži virtuelni adresni proctor u kome je smještena slika procesa. Pristup procesoru je zaštićen, kao i fajlovima i ulazno/izlaznim uređajima.

Jedna nit u višenitnoj obradi posjeduje:

* stanje izvršavanja (izvršavanje, spremna, ...)
* kontekst niti - sačuvano stanje procesorskih registara dok se ne izvršava
* stek izvršavanja niti
* pristup memoriji i resursima svog procesa deljeno sa drugim nitima istog procesa

Svaka nit unutar procesa ima svoj upravljački blok i stek, ali sve niti dijele isti adresni proctor i imaju pristup istim podacima (ono što koristi proces).

Ovo sun neke od koristi višenitne obrade:

* Pozadinski posao - Dugačke ili blokirajuće pozadinske operacije se mogu izvršavati u posebnoj niti, npr. upis u log fajl, takođe interakcija sa korisnikom je moguća i dok druga nit nije završila operaciju
* Asinhrona obrada - Naredba predaje posao drugoj niti. Nit koja je predala posao može da nastavi izvršavanje drugih naredbi dok se prethodna naredba još nije izvršila
* Brzina izvršavanja - Dok je jedna nit procesa blokirana, druge mogu da se izvršavaju
* Deljenje resursa unutar aplikacije - Niti istog procesa dele istu memoriju. Brža i jednostavnija komunikacija između niti nego između procesa
* Ekonomičnost - Brže je napraviti novu nit nego poseban proces. Brže je komutiranje između dve niti istog procesa nego između dva procesa.
* Iskorišćenje višeprocesorske arhitekture - Mogućnost ubrzanja aplikacije ukoliko se niti izvršavaju na različitim procesorima
* Modularna struktura programa - Organizacija koda u više niti razdvaja logičke delove programa, a ujedno to predstavlja jednostavniju implementaciju i održavanje.

Ukoliko je riječ o raspoređivanju niti OS ukoliko podržava niti, raspoređivanje se vrši na nivou niti. Većina informacija o stanju izvršavanja se čuva na nivou niti. Neke od akcija utiču na stanje svih niti (na njihovo ponašanje). Ukoliko je riječ o prekidanju porocesa, tada se prekidaju sve njegove niti, to je primjer jedne akcije koja utiče na sve niti unutar procesa.

Stanja niti se ne razlikuju u suštini od stanja samog procesa. Niti mogu biti u stanju izvršavanja , stanju spreman, kao i u stanju blokiran.

Akcije kojima se menja stanje niti:

* Kreiranje - Kreira se upravljački blok niti i stek i nit se postavlja u stanje spreman.
* Blokiranje - Kada nit mora da čeka na događaj. Sačuva se sadržaj procesorskih registara i procesor se prebacuje na neku drugu nit iz skupa spremnih niti.
* Deblokiranje - Kada se događaj koji je nit čekala desi.
* Završavanje – Nit je završila svoj posao, koji joj je dodjeljen.

Primjer: Slanje zahtjeva na udaljeni server.

|  |  |
| --- | --- |
| Varijanta sa jednom niti:   * Upućuje se zahtjev i proces čeka na odgovor (stanje blokiran procesa, samim tim i niti). Potom se upućuje drugi zahtjev i process opet čeka na odgovor. | Varijanta sa dvije niti:   * U odvojene dvije niti se salju zahtjevi ka udaljenom serveru. Svaka nit posebno. Blokiranje jedne niti ne blokira drugu nit kao što je slučaj kod varijante sa jednom niti. |

Sve niti jednog proceas dijele isti adresni proctor te je potrebno izvršiti sinhronizaciju pristupa ovim dijeljenjim resursima kako ne bi došlo do trke do podataka, odnosno neispravnog rada programa. Resursi takođe mogu da dospiju u nekonzistentno stanje. Ukoliko bi došlo do nesinhronizovanog pristupa podacima, rezultat rada takvog programa bi bio krajnje nedeterminitički.

Tipovi niti:

**Kernel** je jezgro OS-a, odnosno softver koje komunicira sa hardverom i ostatkom softvera unutar OS-a.

* **Niti nivoa korisnika** – Sav posao upravljanja nitima vrši aplikacija. Kernel nije svijestan postojanja niti. Niti se kreiraju i uništavaju u okviru aplikacije korišćenjem biblioteke niti (sadrži kod za pravljenje i uništavanje, prosljeđivanje poruka između niti, raspoređivanje niti, ka oi čuvanje i oporavljanje sadržaja niti). Sve aktivnosti se u odvijaju u korisničkom prostoru unutar jednog procesa. Aplikacija brine o nitima, odnosno korisnik posredstvom biblioteke za rad sa nitima.

Kernel raspoređuje procese i dodeljuje im stanje. Niti imaju svoje stanje koje je „logička“ kategorija i nije direktno povezana sa stanjem procesa. Stanje niti označava status niti unutar procesa. Primjer: Stanje izvršavanja neke niti u okviru procesa ne mora nužno da znači da tu nit procesor trenutno izvršava nego da proces kada dođe na procesor, nit koja je u stanju izvršavanje će se izvršavati.

Prednosti - Komutacija niti ne zahteva prelazak u režim kernela. Aplikacija može da implementira svoj algoritam raspoređivanja niti. ULT se mogu izvršavati na svakom operativnom sistemu.

Mane - Kada se proces blokira, sve niti istog procesa postaju blokirane. Ne može se iskoristiti multiprocesiranje, jer su sve niti jedan proces pa se izvršavaju na jednom procesoru.

* **Niti nivoa kernela** – Aplikacija inicira kreiranje niti, potom svo upravljanje nitima vrši kernel. Kernel održava podatke za process u cjelini, ali i za svaku pojedinačnu nit procesa. Kerneč vrši raspoređivanje na bazi niti.

Prednosti - Kernel može istovremeno da rasporedi više niti istog procesa na više procesora. Ako je jedna nit u procesu blokirana, kernel može da rasporedi drugu nit istog procesa

Mane - Prebacivanje sa jedne na drugu nit istog procesa zahteva prelazak u režim kernela i zato je komutacija KLT za red veličine sporija od komutacije ULT.

Postoji takozvani kombinovani pristup gdje se pravljenje niti i najveći dio upravljanja i raspoređivanja vrši u korisničkom prostoru. Više ULT se može preslikati u jednak ili manji broj KLT.

**Konkurentnost – Međusobna isključivost i sinhornizacija – Druga prezentacija**

OS može pri upravljanju procesima da omogući:

* Multiprogramiranje - Više procesa unutar jednoprocesorskog sistema.
* Multiprocesiranje - Više procesa unutar multiprocesora.
* Distribuiranu obradu - Više procesa na više distribuiranih računara

Konkurentnost je važan aspekt pri svakom upravljanju višestrukim procesima. Odnosi se na pitanje međusobne interakcije i zavisnoti procesa. Konkurentnost se javlja na mjestima gdje se vrši preplitanje različitih procesa (nepredvidivo), javlja se u višetrukim aplikacijama kod multiprogramiranih sistema gdje različiti procesi pristupaju istim resursima. Konkurentnost se javlja i unutar samog OS-a, jer su funkcije OS-a implementirane kao više procesa/niti.

Neki od problema koje konkurentnost donosi:

* Deljenje globalnih resursa - Štetno preplitanje (Dva procesa vrše upis/čitanje nad istom promjenljivom.
* Komplikovanija dodela resursa - Resurs može biti dodeljen procesu koji ga ne koristi.
* Detekcija grešaka - Teže je utvrditi zašto se program neočekivano ponaša. Rezultati nisu deterministički i teže je reprodukovati neočekivano ponašanje.

Prethodno nabrojani problemi se rješavaju uz pomoć **međusobne isključivosti** (mogućnost procesa da obavi akciju bez negativnog uticaja drugih procesa) i **sinhronizacije** (usklađivanje ponašanja procesa aktivnošću drugih procesa.

Ukoliko procesi nisu svesni drugih procesa, odnosno ukoliko su to nezavisni procesi koji nisu predviđeni da rade zajedno, te se dešava nadmetanje za resurse. A ukoliko su procesi su svesni drugih procesa, odnosno ukoliko su to procesi projektovani da zajednički obave posao, dešava se eksplicitna sinhronizacija aktivnosti.

Za ispravan rad programa potrebno je da u jednom trenutku samo jedan proces pristupa resursu – ovakav resurs nazivamo **kritični resurs**, a ovakav način pristupanja resursima naziva se uzajamna isključivost procesa.

**Kritična sekcija** je dio programa unutar kojeg se pristupa kritičnom resursu. Za ispravan rad, kod u kritičnoj sekciji procesi moraju izvršavati sekvencijalno (jedan po jedan). U jednom trenutku samo jedan proces bi smio biti u kritičnoj sekciji.

Procesi koju se nalaza van kritične sekcije ukoliko postoji proces koji je u njoj:

* Prva varijanta – busy waiting – proces koji ne može da uđe u kritičnu sekciju ostaje aktivan neprekidno provjeravajući da li može da uđe u nju.
* Druga varijanta – blokiranje procesa – proces koji ne može da uđe odlazi u stanje blokiran i čeka da ga sistem obavijesti kada može da uđe u kritičnu sekciju.

Propusnica –

Svaki proces pre ulaska u kritičnu sekciju zatraži zaključavanje (prije toga potrebno je provjeriti da li je propusnica slobodna, ako jeste zauzmi, ako nije idi u stanje blokiran) nekog **deljenog** objekta. Proces koji prvi zatraži zaključavanje, uspeva da zaključa objekat i ulazi u kritičnu sekciju. Ovo zaključavanje je implementirano uz oslonac na hardversku podršku za uzajamnu isključivost. Svaki naredni proces ne uspeva da zaključa objekat i prelazi u stanje blokiran. Proces pri izlasku iz kritične sekcije otključava objekat. Jedan od procesa koji čekaju, uspeva da zaključa objekat i ulazi u kritičnu sekciju. Da bi ovo radilo, svi procesi moraju da zaključavaju **isti** objekat (ne sme svaki proces raditi sa svojom lokalnom kopijom).

Hardverska podrška leži u sljedećoj činjenici da postoje specijalne hardverske instrukcije koje obavljaju više operacija nedjeljivo. Uz pomoć ovih instrukcija se zauzima propusnica. Proces u jednom nedjeljivom koraku provjerava da li može da zauzme propusnicu i ako može zauzima je.

Klasa mutex predstavlja propusnicu u C++ jeziku, ali se koristi unique\_lock šablon klase kojem se proslijedi tip mutex. Konstruktor unique\_lock-a pravi objekat i zaključava prosljeđeni mutex. Prilikom izlaska iz kritične sekcije objekat se briše, i automatski se mutex otključava. Mutex nije moguće kopirati kako ne bi došlo do situacije da svaki proces zaključava svoju kopiju, što ne donosi sinhronizaciju među procesima.

**Konkurentnost – međusobna isključivost i sinhronizacija – Treća prezentacija**

**Uzajamno blokiranje** – Deadlock, svi procesi međusobno čekaju da drugi procesi oslobode resurse.

**Livelock** – situacija u kojoj dva ili više procesa nisu blokirani, ali mijenjaju stanja tako da se međusobno onemogućavaju da napreduju. Nijedan proces nije blokiran ali nijedan proces ne napreduje u vršenju korisnog rada.

**Gladovanje** – situacija u kojoj proces ostaje vječno spreman, te nikad ne dobija procesor od raspoređivača.

Zahtjevi za međusobnu isključivost:

Međusobna isključivost se mora sprovesti ako u jednom trenutku samo jedan proces sme da koristi resurs. Ne sme se desiti uzajamno blokiranje ili gladovanje, takođe proces ne sme beskonačno dugo da čeka na ulazak u kritičnu sekciju. U kritičnu sekciju se ulazi bez odlaganja ukoliko nijedan drugi proces nije u sekciji i proces mora izaći iz kritične sekcije u konačnom vremenu.

Koncepti višeg nivoa za sinhronizaciju:

* Semafori – predstavlja djeljeni brojač koji ima metodu koja umanji brojač za 1 (sačekaj da brojač bude veći ili jednak nuli) i metodu koja uveća brojač za 1. Obe ove operacije su nedjeljive. Pozitivan brojač označava broj procesa koji mogu izvršiti operaciju a da ne budu blokirani, dok negativan brojač označava broj blokiranih procesa koji čekaju u redu. Ovakav semafor naziva se opšti semafor, dok postoji i binarni koji je u suštini kao mutex. Postoji jak semafor kojeg odlikuje činjenica da proces koji je otišao u čekanje, prvi biva signaliziran, takozvani FIFO red ujedno garantuje da nema gladovanja. Slab semafor je onaj kod kog nije određeno koji od blokiranih procesa će postati spreman.

Neki od problema semafora:

* + Algoritmi često zahtjevaju upotrebu više smemafora
  + Teško je pronaći greške programa ukoliko se koriste semafori
  + Semafori se koriste istovremeno za obezbjeđivanje međusobne isključivosti i sinhronizacije što bi se trebalo odvojeno tretirati
* Uslovne promenljive – predstavlja uslov na koji nit može da čeka da se ispuni ili za koje nit može da obavjesti druge niti da je ispunjen. Operacije koje donose uslobne promjenljive su operacija čekanja (**wait(unique\_lock<mutex> l);**), operacija signaliziranja jednoj niti da je uslov ispunjen(**notify\_one();**) i operacija signaliziranja više niti da je uslov ispunjen (**notify\_all();**).
* Monitori – predstavlja softverski šablon koji enkapsulira podatke, propusnicu, operacije nad podacima i uslovne promjenljive za sinhronizaciju. Procesi monitor „vide“ kao black-box, pristupaju podacima putem metoda koje se jednim dijelom izvršavaju sekvencijalno jer je potrebno zauzeti propusnicu.

Postoje dvije vrste monitora u zavisnosti od operacije koja signalizira jednoj niti da je uslov ispunjen:

* + Hoare monitori – operacija signal odmah aktivira signaliziranu nit
  + Mesa monitori – operacija signal samo prevodi signaliziranu nit u stanje spremna, nit koja je signalizirala nastavlja da se izršava. Nit koja je signalizirana čeka propusnicu i prije rada provjerava uslov, da nije došlo do promjene dok je čekala na uslov.

**Konkurentnost – Uzajamno blokiranje – Četvrta prezentacija**

Uzajamno blokiranje predstavlja trajno blokiranje skupa procesa koji se nadmeću za isti skup resursa. Svaki proces je blokiran čekajući na događaj koji može aktivirati neki od već blokiranih procesa. Uzajamno blokiranje se može dogoditi u sistemu u kojem procesi koriste resurse na sledeći način -> Zahtev ( proces pre upotrebe zahteva resurs i ako resurs nije u tom trenutku dostupan, proces odlazi u čekanje) -> Upotreba ( u ovoj fazi proces može da pristupa resursu i da ga koristi) -> Otpuštanje (proces otpušta resurs nakon korišćenja).

Uslovi za uzajamno blokiranje:

1. Uzajamno isključivanje - Samo jedan proces u datom trenutku može koristiti resurs
2. Držanje i čekanje - Proces može držati resurs dok čeka dodeljivanje ostalih resursa
3. Bez prekida - Resurs se ne može nasilno oduzeti procesu koji ga drži
4. Kružno čekanje - Potencijalni rezultat prva tri uslova. Postoji zatvoren krug procesa takav da svaki proces drži bar jedan resurs koji je potreban sledećem procesu u krugu.

Uz prva tri uslova postoji mogućnost da se uzajamno blokiranje desi, dok sa svim uslovima postojanje uzajamnog blokiranja je zagarantovano.

Rješavanje problema uzajamnog blokiranja:

* Sprječavanje – primjena mehanizama koji eliminišu usloev koji dovode do uzajamnog blokiranja
* Izbjegavanje – dinamički se vrši raspodjela resursa tako da ne dođe do uzajamnog blokiranja
* Otkrivanje – kada se desi uzajamno blokiranje preduzimaju se akcije za oporavak

Sprječavanje:

Nije moguće onemogućiti uslov uzajamnog isključivanja jer u određenim scenarijima pristupa djeljenim resursima procesi moraju pristupati jedan po jedan.

Sprječavanje uslova držanja i čekanja se može sprječiti ako process istovremeno zatraži sve potrebne resurse, ili ostaje blokiran dok svi resursi ne budu slobodni međutim to je neefikasna varijanta (proces čeka, a ima neke koji su slobodni, ili process drži resurse koje ne koristi).

Odbacivanje uslova “bez prekidanja” vrši se naredbom procesu da otpusti resurse ukoliko mu je odbijen zahtjeb za zauzimanje nekog resursa. Druga varijanta je da ako process zatraži resurs koji drži drugi blokiran proces, OS prekida drugi process, koji mora da otpusti sve svoje resurse. Primjenljivo na resursima čije se stanje može lako sačuvati i kasnije oporaviti – npr procesorski registri.

Sprječavanje kružnog čekanja je ako se uradi linearno slaganje tipova resursa, gdje se svakom resursu dodjeli indeks i ako je proces zauzeo resurs Ri  onda može da zauzme resurs R j samo ako je j > i. Međutim ovo može biti neefikasno jer procesu se možda zabranjuje zauzimanje resursa bespotrebno.

Sprječavanje je veoma restriktivno i smanjena je propusna moć sistema kao i oslabljena iskoriščenost resursa.

Izbjegavanje:

Dinamička raspodjela resursa kako ne bi došlo do uzajamnog blokiranja. Prave se razumni izbori tako da se nikad ne stigne u tačku uzajamnog blokiranja, a i omogućuje se više konkurentnosti od sprečavanja. Dinamički se donosi odluka da li će tekući zahtev dodele resursa dovesti do uzajamnog blokiranja. Postoje varijante kod kojih se proces i ne pokreće ukoiko njegovi zahtjevi mogu dovesti do uzajamne blokade kao i ne obrađivanje zahtjeva za resursom ako dodjela resursa može dovesti do blokiranja. \*\* Pogledati predavanje za objašnjenje Bankarevog algoritma koji utvrđuje da li će sistem biti bezbjedan (da li će se procesi izvršiti bez uzajamnog blokiranja) tokom rada programa.

Prednosti – Manje restriktivno od sprječavanja. Nije neophodno da prekinuti i vratiti unazad procese kao kod otkrivanja uzajamnog blokiranja.

Mane -Maksimalni zahtevi za resursima svakog procesa moraju se unapred definisati. Proces koji se razmatra mora biti nezavisan (redosled izvršavanja nije ograničen sinhronizacijom). Mora postojati fiksan broj resursa za dodelu i proces ne može otpustiti resurs.

Otkrivanje:

Kada se desi uzajamno blokiranje preduzimaju se akcije za oporavak. Ne graničava se pristup resursima. Povremeno se vrši provjera da li je došlo do kružnog čekanja, u slučaju da jeste vrši se oporavak sistema. \*\* Algoritam za otkrivanje uzajamnog blokiranja na prezentaciji.

Strategije oporavka:

1. Prekini sve uzajamno blokirane procese - Najjednostavnije rešenje, međutim velika cijena ovakvog pristupa jer se poništava sve što je proces do tada uradio.
2. Vrati sve uzajamno blokirane procese na neku raniju kontrolnu tačku i ponovo ih pokreni - Može opet doći do uzajamnog blokiranja. Ali zbog nedeterminantnosti konkurentne obrade, u ponovnom izvršavanju možda ne bude blokiranja
3. Redom prekidaj sve uzajamno blokirane procese dok više ne bude blokiranja - Redosled kojim će procesi biti prekidani treba da bude na osnovu nekog kriterijuma
   * 1. Prioritet
     2. Utrošak procesorskog vremena do sada
     3. Predviđeno preostalo vreme rada
     4. Tip resursa koje je proces koristio
     5. Količina resursa potrebna za završetak rada
4. Redom oduzimaj resurse od procesa i dodeljuj ih drugim procesima dok više ne bude blokiranja – Moraju se izabrati resursi i procesi tako da se minimizuju troškovi. Proces kojem je oduzet resurs mora se vratiti na neko prethodno stanje u kojem je bio pre nabavke resursa. Sprečiti gladovanje procesa – ne smiju se resursi oduzimati od istog procesa uvijek.

Ne isplati se angažovati resurse za bilo koji način izbjegavanja uzajamnog blokiranja. OS je implementiran sa namjerom da ne ulazi u deadlock, a korisnički programi su odgovornost programera. Windows i GNU/Linux su implementirani na ovaj način.

**Operativni sistem – Osnovni koncepti – Peta prezentacija**

Operativni sistem je program koji upravlja izvršavanjem aplikacionih programa, te služi kao interfejs između programa i hardvera računara.

Operativni sistem treba da obezbjedi:

Pogodnost - Da računar bude korisniku pogodniji za korišćenje.

Efikasnost - Da se resursi računarskog sistema koriste na efikasan način.

Mogućnost izvršavanja novih korisničkih aplikacija - Da je omogućeno dodavanje novih funkcionalnosti sistema nezavisno od ugrađenih osnovnih servisa.

Krajnji korisnik vidi računarski sistem kao skup aplikacija. Obezbjeđen je skup sistemskih programa koji upravlja hardverom. Aplikacije koje vidi korisnik obraćaju se sistemskim programima da obavi određene funkcije. Najvažniji sistemski program je operativni sistem.

Programski interfejsi u računarskom sistemu:

Interfejs aplikativnog programiranja (API) - specifikacija servisa koje aplikacija može da poziva (npr. funkcije u sistemskoj biblioteci)

Binarni interfejs aplikacije (ABI) - kao API, ali nižeg nivoa – veza sa operativnim sistemom koja definiše format komunikacije sa OS (sistemski pozivi, predstavljanja tipova podataka, način poziva funkcija, ...)

Arhitektura skupa instrukcija (ISA) - repertoar instrukcija mašinskog jezika koje hardver podržava.

Dva su osnovna načina interakcije korisnika sa OS-om, a to su **komandna linija** (interakcija se vrši unosom tekstualnih komandi + interpreter komandi je ugrađen u kernel ili je to poseban program u okviru OS-a (shell), komande se izvršavaju pozivom sistemskih programa)i **grafički korisnički interfejs** (gdje korisnik korišćenjem miša ili prsta manipuliše grafičkim elementima koji predstavljaju programe, fajlove,...).

Neke od standardnih usluga OS-a:

Izvršavanje programa - OS upravlja resursima koje program koristi (učitavanje instrukcija i podataka u glavnu memoriju, inicijalizacija U/I uređaja, ...)

Pristup U/I uređajima - OS pruža interfejs za pristup U/I uređajima. Programer zahteva U/I operaciju kroz interfejs, a OS je zadužen za dalju komunikaciju sa U/I uređajem.

Pristup fajlovima - OS vodi računa o internoj strukturi uređaja za skladištenje i načinu skladištenja i obezbjeđuje mehanizme kontrole pristupa fajlovima.

Deljeni pristup sistemu - OS obezbeđuje zaštitu resursa od neovlašćenog pristupa i rešava konflikte u takmičenju za resurse

Upravljanje greškama - OS obezbeđuje reakciju na hardverske i softverske greške (prekid programa, ponavljanje operacije, prijava greške)

Nadzor sistema - OS prikuplja statistiku upotrebe resursa i nadgleda performanse.

Računar je skup resursa za prenos, skladištenje i obradu podataka. OS je odgovoran za upravljanje ovim resursima. OS je program ili skup programa koje izvršava procesor što ga ne razlikuje od ostalih programa u računaru.

Razlika u odnosu na drugi softver je namjena - Softver OS sadrži instrukcije čijim izvršavanjem se upravlja resursima sistema (raspoređivanje aplikacija, zauzimanje memorije, ...)

Istorija razvoja operativnih sistema:

1. Serijska obrada – računari nisu imali OS, jedan korisnik u jednom trenutku ima pristup računaru. Problemi: neefikasnost, direktno se upravlja hardverom, nema istovremenog izvršavanja više programa.
2. Jednostavni sistemi paketne obrade – korisnik ne pristupa računaru direktno (hardveru). Postoji operator koji skup programa prikupi pa postavi na računar. Pripremljen skup programa izvršava poseban softver koji se naziva **monitor** – je program sa posebnom namjenom da upravlja izvršavanjem drugih programa, izvršavajući jedan po jedan program, nakon izvršenog programa, program se vraća na monitor a monitor prelazi na naredni program. Zaštita memorije je izvršena tako da korisnički program ne smije da pristupa memoriji u kojoj je monitor. Hardver procesora mora da detektuje i spreči takve instrukcije. Vremenski brojač omogućava zaštitu da jedan program ne koristi procesor predugo, te nakon isteka brojača, program se zaustavlja i kontrola predaje monitoru. Privilegovane instrukcije jesu dio instrukcija može da izvrši samo monitor. Prekidi daju više fleksibilnosti jer omogućuju da se upravljanje predaje i oduzima programima. Postoje dva režima korisnički (restriktivniji) režim i režim kernela kome je dozvoljeno da pristupa zaštićenoj memoriji i da izvršava prvilegovane instrukcije.
3. Multiprogramirani sistemi paketne obrade – istovremeno je više programa postavljeno za izvršavanje ali i dalje u jednom trenutku procesor može da izvršava samo jedan program. Mana je besposleno čekanje procesora na U/I zahtjev npr.

**Multiprogramiranje –** Pored OS-a, u memoriju se smješta više korisničkih programa. Dok jedan program čeka na U/I zahtjev, drugi spremni programi se mogu izvršavati na procesoru. Hardverska podrška za multiprogramiranje su U/I prekidi, DMA, mogućnost da procesor nastavi sa radom dok neki program čeka na U/I kontroler. Nakon što kontroler završi postavlja prekid i onda OS odlučuje o daljoj sudbini tog programa.

Da bi se ralizovalo multiprogramiranje OS mora da obezbjedi upravljanje memorijom (oni programi koji se izvršavaju prisutni su u memoriji) i raspoređivanje poslova (ako imamo više spremnih treba nam algoritam da odredimo prvog za izvršavanje).

1. Sistemi sa djeljenjem vremena – podrazumjevaju sisteme gdje se procesor dijeli na više korisnika. Svi koriste OS preko terminala, te potom OS upravlja korisničkim programima relativno dobro. CTSS (*Compatible Time Sharing System*) – svakih 0.2 sekunde generator takta izdaje prekid te onda OS preuzima upravljanje i može da dodjeli procesor drugom korisniku.

Zahtjevi za savremeni OS:

Multiprogramiranje uvelo zahtev deljenja resursa i konkurentni pristup, takmičenje za resurse zaštita od štetnog preplitanja,...

Deljenje vremena uvelo zahtev višekorisničkog pristupa resursima, zaštitu i raspoređivanje.

Osnovna dostignuća na koja se oslanjaju današnji OS:

* Procesi – programi u izvršavanju, odnosno proces je jedinica aktivnosti koju karakterišu naredbe za izvršavanje, tekuće stanje i dodjeljeni skup sistemskih resursa. Proces se sastoji iz tri dijela:

1. Izvršni program – instrukcije koje procesor izvršava.
2. Podaci – pridruženi podaci samom programu
3. Kontekst izvršenja procesa – interni podaci pomoću kojih OS može da upravlja procesorom, onda kompletno stanje procesa u bilo kom trenutku, te kontekst sadrži prioritet procesa ili informaciju da li proces čeka na U/I operaciju.

Proces je podjeljen na niti (izvršna jedinica posla) koje se izvršavaju konkurentno. Proces predstavlja skup jedne ili više niti i dodjeljenog sistemskog resursa.

* Upravljanje memorijom – neke od odgovornosti OS-a pri upravljanju memorijom:

1. Izolacija procesa - Proces ne sme da pristupi delu memorije rezervisanom za drugi proces.
2. Automatsko dodeljivanje i upravljanje - Korisnička aplikacija ne mora direktno da vodi računa o upravljanju memorijom.
3. Zaštita i kontrola pristupa - Delovima memorije mogu na različite načine da pristupe različiti korisnici.
4. Dugotrajna memorija - OS mora da omogući programima da skladište informacije na duže vreme, po isključenju računara. Fajl sistem je implementacija dugotrajne memorije

**Virtuelno adresiranje** – mehanizam koji omogućuje da programi vrše adresiranje sa logičke tačke gledišta. Program ne vodi računa o količini memorije ili o stvarnoj adresi unutar glavne memorije (stvarna adresa se određuje dinamički prilikom prevođenja). Proces je podjeljen u odeđeni broj blokova fiksne dužine – **stranice**. Program adresira riječ pomoću virtuelne adrese koja se sastoji od broja stranice i pomjeranja unutar stranice. Straničenje pruža dinamičko mapiranje logičke na realnu adresu.

Virtuelna memorija donosi pogodnost da ne moraju sve stranice procesa da budu cijelo vrijeme u glavnoj memoriji. Sve stranice se nalaze na disku a samo neke u memoriji. Ako se referencira stranica procesa koja nije u memoriji onda se sa diska prebacuje u memoriju, ako nema dovoljno mjesta onda neka od postojećih stranica koja je u memoriji se izbacuje da napravi mjesto. OS određuje politiku dodjeljivanja memorije (koliko stranica se nalazi u virtuelnoj memoriji, algoritam za zamjenu stranica , algoritam prebacivanja stranice sa diska u memoriju,...).

* Zaštita informacija i bezbjednost:
  + Raspoloživost - zaštita sistema od prekida rada
  + Poverljivost - korisnici ne smiju da čitaju podatke za koje nemaju pravo pristupa.
  + Integritet podataka - zaštita podataka od nedozvoljene izmene.
  + Autentičnost - provera identiteta korisnika i validnost poruka ili podataka.
* Raspoređivanje i upravljanje resursima – OS upravlja resursima (glavna memorija, procesor, U/I uređaj) i raspoređuje njihovu upotrebu prema procesima, pri čemu mora obezbjediti:

1. Nepristrasnost – Poslovi sličnog tipa koji se takmiče za jedan resurs treba da dobiju približno isti i fer pristup tom resursu.
2. Različitost odziva – Namjerna pristrasnost dodjelom resursa određenom procesu, što će omogućiti da se bolje ispuni ukupan skup zahtjeva
3. Efikasnost – OS pokušava da poveća propusnu moć sistema, a smanji vrijeme odziva.

Elementi OS-a za raspoređivanje:

* Kratkoročni red - U kratkoročnom redu su procesi spremni za izvršavanje, vrši se kružno dodeljivanje prema prioritetu.
* Dugoročni red - U dugoročnom redu su novi poslovi koji čekaju da se ubace u kratkoročni red.
* Za svaki U/I uređaj formira se red procesa koji čekaju na upotrebu uređaja.
* Operativni sistem opslužuje prekide i sistemske pozive
* Po opsluživanju prekida ili sistemskog poziva, kratkoročni raspoređivač određuje proces koji će naredni da se izvrši.

Simetrična multiprocesna obrada (SMP) se bavi raspoređivanjem procesa na hardveru na simetričnim multiprocesorima, ovi procesori dijele istu glavnu memoriju i svi oni mogu da obavljaju istu funkciju. Više procesa može paralelno da se izvršava. OS vodi računa o raspoređivanju niti ili procesa na pojedinačne procesore i sinhronizaciji između njih.

Prednosti višeprocesne obrade:

* Performanse - Ako se delovi posla mogu odraditi paralelno, sistem sa više procesora će postići bolje performanse.
* Raspoloživost - Otkaz jednog procesora neće zaustaviti sistem. Pošto svi procesori mogu da obavljaju iste funkcije, sistem će nastaviti da radi sa smanjenim performansama.
* Postepeno poboljšanje sistema - Moguće je poboljšati performanse dodavanjem procesora.
* Skaliranje - Proizvođači mogu ponuditi opseg proizvoda različite cijene i performansi na osnovu broja procesora

Komunikacija korisničke aplikacije sa OS-om:

* Sistemski pozivi – skup servisa koji se mogu pozvati iz sloja korisničkih aplikacija i koji obavljaju određenu funkcionalnost, mogu se pozvati direktno ili posredstvom sistemske biblioteke. OS kada mu stigne zahtjev u vidu sistemskog poziva u režimu kernela izvršava traženi zadatak i vraća odgovor aplikaciji.

**Sistemska biblioteka** sadrži programski kod za upućivanje sistemskih poziva. Služi koa srednji sloj između aplikacije i OS-a.

Neki tipovi sistemski poziva:

* Upravljanje procesima – Završi (prekini), učitaj (izvrši), kreiraj, čekaj određeno vrijeme, čekaj na događaj,...
* Upravljanje datotekama – Kreiraj i obriši, otvori i zatvori, pročitaj upiši i pozicioniraj se, preuzmi i postavi atribute datoteke,...
* Upravljanje uređajima – Zatraži i otpusti uređaj, pročitaj i upiši, logički dodaj i izbaci uređaj iz sistema,...
* Evidencija infromacija – Preuzmi i postavi sistemsko vrijeme i datum, podatke sistema,...
* Komunikacija – Kreiraj i obriši komunikacionu vezu, pošalji i primi poruku,...

Struktura kernela OS-a sadrži **monolitno jezgro** u kome se nalaze svev funkcionalnosti OS-a (raspoređivanje, datotečki sistem, upravljanje memorijom,...), To je slojevita struktura gdje su funkcionalnosti podjeljene u hijerarhijski organizovane slojeve. Svaki sloj poziva operaije koje pruža niži nivo i pruža operacije koje mogu biti pozvane od strane višeg nivoa. Postoje dijelovi koji komuniciraju dvosmjerno pa je onda netrivijalno postaviti hijerarhiju slojeva OS-a.

**Mikrokernel** – jezgro koje sadrži samo nekoliko osnovnih funkcija, dok se ostali servisi obezbjeđuju preko procesa koji rade u korisničkom režimu.

Pored slojevite hijerarhijske strukture, OS može biti **modularan**, gdje su funkcionalnosti podjeljene u module. Kernel sadrži ključne funkcije, a ostale se učitavaju kad zatrebaju. Kod modularne strukture kao i kod slojevite hijerarhijske strukture, imamo nezavisne cjeline koje komuniciraju jedne s drugima.

Pokretanje OS-a vrši *Bootstrap loader* – program koji se prvi pokreće i koji je odgovoran za učitavanje OS-a u memoriju i pokretanje. Prvi korak je učitavanje bootstrap loadera odnosno BIOS-a iz ROM-e, drugi korak je učitavanje kompleksnijeg loader-a sa diska.

Virtuelizacija (virtuelne mašine) omogućava da se na jednom računaru istovremeno izvršava više OS ili više istanci jednog OS-a. Neke od VM-a su Java VM, Microstft .NET framework,...

**Procesi – Šesta prezentacija**

Proces je program u izvršavanju, odnosno entitet koji se može dodjeliti i izvršavati u procesoru. To je jedinica aktivnosti koju karakterišu naredbe za izvršavanje, tekuće stanje i dodjeljeni skup sistemskih resursa.

Neki od zahtjeva koje OS treba da ispuni prilikom upravljanja procesima:

* Mora da obezbjedi efektivno preplitanje izvršavanja procesa
* Dodjelu resursa procesima i zaštita dodjeljenih resursa od nekontrolisanog prisupa drugih procesa
* Djeljenje i razmjena infromacija između procesa
* Sinhronizacija između procesa

Elementi jednog procesa:

* **Programski kod** – to je suština jednog procesa odnosno programski kod kazuje šta taj proces treda da uradi, a jedan programski kod mogu dijeliti više različitih istanci istog procesa.
* **Skup podataka nad kojima se proces izvršava**
* **Atributi koji opisuju stanje procesa** – Informacije o procesu koje OS skladišti kako bi lakše upravljao procesima

Atributi procesa:

* + Identifikator (jedinstevno obilježje svakog procesa) – neki pozitivan cijeli broj uz pomoć kojeg upravljačke strukture OS mogu da referenciraju taj proces
  + Stanje (Opisuje stanje u kom se proces nalazi – aktivan, blokiran/čeka,...)
  + Prioritet (Procesi imaju prioritete jedni u odnosu na druge, tako OS može da odredi ko će prije da se izvrši u zavisnoti od njegovog prioriteta odnosno od bitnosti tog procesa)
  + Programski brojač (Adresa sljedeće instrukcije u programskom kodu)
  + Pokazivači na memorijske blokove (Pokazuju na kod i podatke vezane za proces)
  + Kontekstni podaci (Podaci iz registra procesora vezani za trenutno stanje procesa)
  + U/I statusne informacije (Neobrađe U/I zahtjevi, dodjeljeni U/I uređaji, lista datoteka koje proces koristi, ...)
  + Računovodstvene informacije (Korišćeno procesorsko vrijeme, neka vremenska ograničenja)

Upravljački blok procesa (UBP) sadrži atribute procesa za čije je kreiranje i upravljanje njime zadužen sam OS. UBP omogućuje multiprogramiranje jer sadrži informacije koje omogućuju da se proces prekine i kasnije nastavi izvršavanje bez posljedica kao da nije ni došlo do prekida. Kada se proces prekine, snimaju se vrijednosti procesorskih registara u UBP i kada proces dođe u momenat da se nastavi izvršavanje njegovog prog. koda snimljene vrijednosti procesorskih registara učitavaju se iz UBP u procesorske registre.

**Trag procesa –** predstavlja niz instrukcija koje proces izvršava, čime je predstavljeno ponašanje procesa. Ponašanje procesora se može predstaviti isprepletenim prikazom tragova različitih procesa. **Raspoređivač (dispatcher) je program koji prebacuje procesor sa jednog procesa na drugi proces.**

Stanja procesa –

* Izvršavanje procesa je stanje u kom se proces nalazi ukoliko procesor izvršava trenutno instrukcije programskog koda tog procesa.
* Spreman je stanje u kojem se proces nalazi ukoliko je spremam ua izvršavanje ali procesor ne izvršava trenutno njegove instrukcije.
* Blokiran je stanje u kome se proces nalazi ukoliko čeka na neki događaj, najčešće U/I uperacija
* Novi je stanje preocesa neposredno nakon stvaranja, dok ga OS ne prihvati u red spremnih procesa
* Izlaz je stanje u kome se proces nalazi nakon što ga je OS izbacio iz reda spremnih procesa jer je završio svoj kod

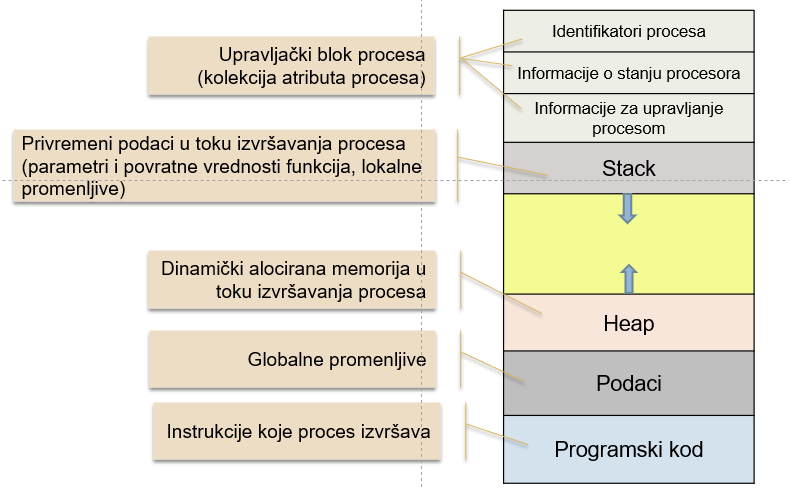
Pri dodavanju novog procesa OS kreira strukture podataka za upravljanje procesom i učitava u RAM proces ili njegov neki dio (misli se na kod procesa).

OS je zadužen da na znak ( korisnička akcija, greška u programu, instrukcija roditeljskog procesa za završetak procesa potomka ili završavanje roditeljskog procesa) završi proces i oslobodi resurse koje je proces koristio.

OS prebacuje procese iz stanja u stanje na osnovu logičkih karakteristika kao npr:

* IZVRŠAVANJE -> SPREMAN
  + Kada procesu istekne maks dozvoljeno vrijeme neprekinutog procesorskog izvršavanja (quantum)
  + Kada se pojavi proces višeg prioriteta
  + Kada proces dobrovoljno prepusti procesor.

Da bi OS znao kako da upravlja procesima on mora znati lokaciju procesa i njegove atribute.



Jedan proces je podjeljen u stranice, koje se nalaze u sekundarnoj memoriji (na disku). Jedan dio stranica se nalazi u glavnoj memoriji, a stranice ne moraju uzimati fizički susjedne lokacije u memoriji. Infromacije o sadržaju registara procesora čuvaju vrijednosti registara nakon prekida procesa ( registri vidljivi korisniku, statusni i upravljački registri, stack pointer, ...)

Infromacije za upravljanje procesom olakšavaju posao OS-u a sadrže stanje procesa, prioritet, zavisno od algoritma raspoređivanja posjeduju i neke dodatne informacije, događaj na koij čeka proces. Takođe je važa struktura koja povezuje sve procese, primjer je red čekanja istog prioriteta. Pored navedenog bitna je i međuprocesna komunikacija, privilegije procesa (kojim dijelovima memorije proces može da pristupa i koje instrukcije ima pravo da izvršava), upravljanje memorijom kao i vlasništvo nad resursima.

\*OS se izvršava u režimu kernela u kome ima potpunu kontrolu nad procesorom i svim njegovim instrukcijama, registrima i memorijom, u ovom režimu moguće je izvršavati privilegovane instrukcije dok se u korisničkom režimu izvršavaju korisnički programi sa neprivilegovanim instrukcijama.

BIT REŽIMA – postavlja se npr ukoliko se desi neki sistemski poziv, a znači prelaz iz korisničkog u režim kernela.

Stvaranje procesa:

1. Novom procesu se dodeljuje jedinstveni identifikator - u tabelu procesa dodaje se nova stavka.
2. Dodeljuje se prostor za sve elemente slike procesa – Ovaj prostor može da bude i virtuelan, a ne fizički.
3. Inicijalizuje se upravljački blok procesa, programski brojač se postavlja na prvu instrukciju, a ostale vrednosti na *default*
4. Ažuriraju se strukture sa vezama procesa – npr. proces se uvezuje u listu procesa u odgovarajućem stanju
5. Kreiraju se ili ažuriraju druge strukture podataka – npr. kreira se fajl sa podacima o aktivnosti procesa

Komutiranje procesa je termin koji označava promjenu procesa na procesoru. Prekida se proces koji se trenutno izvršavana na procesoru, OS postavlja stanje IZVRŠAVANJE drugom procesu i predaje se procesor tom drugom procesu. (Procesor kreće da izvršava instrukcije drugog procesa)

Kada se prekid desi vrši se komutiranje procesa na procesoru. Prekid može biti:

* Prekd generatora takta – ako je isteklo maksimalno dozvoljeno korišćenje procesora, proces se prekida i prebacuje u stanje Spreman
* U/I prekid – OS prebacuje procese blokirane zbog čekanja na događaj u stanje Spreman
* Greška memorije – proces referencira stranicu koja nije u glavnoj memoriji, proces prepušta procesor nekom drugom procesu a OS traži stranicu u spoljnoj memoriji i prebacuje je u glavnu memoriju.
* Greška u izvršavanju procesa
* Poziv operacije OS iz procesa (sistemski poziv) – npr. proces zahtjeva pristup nekoj datoteci i OS preuzima kontrolu da bi izvršio U/I akciju.

Promjena stanja procesa:

1. Čuvanje sadržaja procesora (programski brojač, drugi registri)
2. Ažuriranje upravljačkog bloka procesa (promena stanja na Spreman ili Blokiran i ažuriranje drugih polja
3. Prebacivanje upravljačkog bloka procesa u odgovarajući red spremnih ili blokiranih
4. Izbor narednog procesa za izvršavanje
5. Ažuriranje upravljačkog bloka izabranog procesa, promena stanja na Izvršavanje
6. Ažuriranje strukture podataka za upravljanje memorijom – moguće izmene stranica u glavnoj memoriji
7. Postavljanje sadržaja registara procesora u skladu sa sadržajem upravljačkog bloka procesa izabranog za izvršavanje (učitavanje vrijednosti registara novog procesa u registre procesora)

Kooperacija i komunikacija između procesa:

Procesi koji se konkurentno izvršavaju su:

* Nezavisni – Ako na njihovo izvršavanje ne utiču drugi procesi, proces koji ne dijeli podatke sa drugim procesima je nazacvisan.
* Međusobno zavisni ako utiču jedni drugima na izvršavanje (Svaki proces koji dijeli podatke sa drugim procesima je zavisan od njih)

Neki od razloga za međuprocesnu komunikaciju su djeljenje infromacija jer različitim procesima trebaju isti podaci, primjer je datoteka kojoj pristupaju. Potom ubrzanje izračunavanja je pojam koji zahtjeva da više različitih procesa vrše izvračunavanja nad istim podacima kako bi na multiprocesorskoj platformi ubrzali rad. I jedan od posljednjih je modularnost a to podrazumjeva da prilikom implementacije postoje logički tokovi (niti npr.) izvršavanja koji pristupaju istim podacima.

Dva osnovna načina komunikacije jesu **djeljena memorija** (brža jer ne zahtjeva sistemske pozie za pristup podacima) i **razmjena poruka** (Pristupačnije za distribuirane sisteme u kojima uglavnom nema djeljene memorije)**.**

**Djeljena memorija** predstavlja jedan region unutar memorije koji proces koji ga kreira, kreira ga u svom adresnom prostoru a drugi procesi zakače taj region u svoj adresni prostor. Format podataka, sinhronizacija i isključivost su odgovornost samih procesa.

**Razmjena poruka** se vrši kroz slanje i primanje poruka putem funkcija. Postoje više načina razmjene poruka:

* Blokirana (blokirano stanje – nakon slanja poruke pošiljala je blokiran dok primalac ne primi poruku, blokirani prijem – nakon poziva prijema poruke primalac ostaje blokiran dok poruka ne bude raspoloživa)

ili neblokirana komunikacija (neblokirano stanje – pošiljalac nastavlja sa radom nakon slanja poruke, neblokirani prijem – nakon poziva prijema poruke, primalac dobija poruku ako je raspoloživa ili NULL)

* Direktno (Proces mora eksplicitno navesti identifikator drugog procesa sa kojim komunicira)

ili indirektno adresiranje (Procesi ne navode direktnu adresu procesa kome šalju nego šalju nekoj djeljenoj strukturi u koju neki upisuju poruke a neki čitaju poruke. Mogu biti sve kombinacije \*:\* gdje \* predstavlja 1 ili N pošiljalaca/primalaca.)

* Baferovani (Poruke stoje u privremenom redu koji može primati jednu poruku, ograničen broj poruka ili neogrančen) ili nebaferovani zapis

PIPES – Poseban tip komunikacije između dva procesa po principu proizvođač (upisuje u jedan kraj) – potrošač (čita podatke sa drugog kraja). Mogu biti i dvosmjerni pipes (full duplex ukoliko u istom trenutku mogu podaci putovati u oba smjera ili half duplex ukoliko u jednom smjeru mogu podaci da putuju u jednom trenutku).

OS kao proces! OS je implementiran kao skup procesa, funkcije njegovog jezgra su organizovane u procese i oni se izvršavaju u režimu kernela. Dakle, kada je potrebno izvršiti neki sistemski poziv proces se samo prebaci u režim kernela i izvršava kod OS iz djeljenog adresnog prostora. Za komutiranje samih procesa postoji dio OS koij se izvršava izvan korisničkih procesa.

**Sedma Prezentacija – UPRAVLJANJE MEMORIJOM**

Da bi se podržalo mulitprogramiranje OS mora da čuva više procesa u memoriji. Radna memorija je veliki linearni niz bajtova gdje svaki bajt ima svoju adresu. Procesor instrukciju preuzima iz memorije na osnovu vrijednosti programskog brojača instrukcija, potom se instrukcija dekodira. Nakon izvršenja instrukcije, a i u toku nje za rad dobavljanja operanada mogući su pristupi nekim memorijskim lokacijama.

Računarski hardver i OS moraju podržati RELOKACIJU, DJELJENJE I ZAŠTITU.

Relokacija je pojam koji se odnosi na izmjenu pozicije procesa u radnoj memoriji, jer se može desiti da proces bude zamjenjen iz radne memorije na disk i ponovo vraćen (vjerovatno ne na poziciju na kojoj je bio).

OS i sistem za upravljanje memorijom treba da omogući kontrolisani pristup djeljenim područjima unutar memorije bez ogrožavanja zaštite.

Zaštita memorije se sprovodi u toku izvršavanja jer proces nema fiksno određenu lokaciju u kojoj se nalazi zbog relokacije. Isto tako proces ne smije da pristupa lokacijama u kojima se nalaze drugi procesi.

Kako bi se memorija zaštitila svaki proces mora imati odvojen memorijski prostor. To se omogućava uz dva posebna procesorska registra BASE – početna adresa procesa i LIMIT – posljednja adresa kojoj proces pristupa.

Adresa unutar programa:

* U trenutku kompajliranja mora se znati gdje će program biti u memoriji i u samom programu se upišu stvarne adrese kojima će pristupati. (Ako je podržano multiprogramiranje nije moguće znati adresu u trenutku kompajliranja.
* U trenutku učitavanja – stvarne adrese se upisuju tek u trenutku učitavanja programa u memoriju, međutim ako program može biti pomjeran u toku izvršavanja po memoriji onda ovaj metod nije upotrebljiv pa se pribjegava LOGIČKIM ADRESAMA
* U toku izvršavanja – ovo je jedina varijanta koja podržava relokaciju (Ako se u toku izvršavanja programa adresa početka procesa promjeni, tek u trenutku izvršavanja se može odrediti stvarna adresa kojoj se pristupa – Savremeni OS koristi metodu prevođenja adresa u stvarne tek u trenutku izvršavanja.

**Logička adresa** je adresa koja je upisana u programu. To je relativna adresa u odnosu na početak bloka. U toku izvršavanja se ove logičke adrese pretvaraju u fizičke adrese u memoriji.

**Fizička adresa** je adresa kojoj se pristupa. Ona je apsolutna adresa u memoriji.

Načini za upravljanje memorijom:

1. Alociranje susjednih mem. lokacija
   * Djeljenje memorije na particije
   * Partnerski sistem – Cjelokupna raspoloživa memorija se tretira kao jedan blok veličine 2U, ako se zahtjeva blok veličine s, 2U-1 < s <= 2U onda se alocira čitav početni blok. U suprotnom blok se dijeli na dva jednaka dijela (partnera). Ponavljamo postupak dok se ne napravi blok koji je manji ili jedna s, i onda mu se dodjeli taj blok memorije.

**Prednosti:**

Zauzima se blok koji najbolje odgovara veličini procesa i jednostavno se sažimaju susjedni slobodni blokovi

**Nedostaci:**

Interna fragmentacija(podjeljenost) memorije.

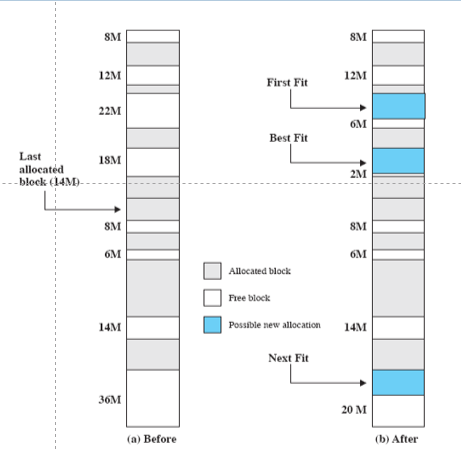
1. Straničenje
2. Segmentacija

**Djeljenje memorije na particije** je klasičan način upravljanaj memorijom koji je korišćen prije pojave virtuelne mašine, gdje se kompletan proces smješta u uzastopne mem. lokacije. Potrebno je podjeliti memoriju na particije jednake fiksne veličine. Svaki proces koji je manji ili jednak od te zadate veličine može se smjestiti u bilo koju raspoloživu particiju. Kod particija jednake veličine postoji problem ukoliko program ne može da stane u particiju, a problem koji se stvara jeste interna fragmentacija i to što program iako je manji od veličine particije zauzme cijelu particiju.

Particije nejedanke veličine umanjuju probleme sa particijama jednakih veličina ali ne rješavaju ih potpuno. Za razliku od particija nejednake veličine ove sa jednakim veličinama se zauzimaju nezavisno od rednog broja particije, sve su jednake pa nije ni bitno u kojoj se program nalazi.

Smještanje u particije nejednakih veličina se odvija tako što svaka particija ima red čekanja i proces se smješta u najmanju particiju u koju može da stane. Neefikasan način jer proces čeka, a mogao bi da se smjesti u neku drugu particiju npr., dok je varijanta sa jednim redom čekanja dosta efikasnija (Proces se smješta u najmanju raspoloživu particiju).

Nedostaci fiksnog djeljenja memorije jesu da sa ograničenim brojem particija u memoriji ograničavamo i broj aktivnih procesa u datom momentu. A veliki broj malih procesa neće efikasno korstiti memoriju.

Dinamičko djeljenje memorije na particije jeste varijanta u kojoj su particije promjenljive veličine a i ima ih promjenljiv broj. Proces dobija tačno onoliko memorije koliko mu i treba. Problem kod dinamičkog djeljenja jeste eksterna fragmentacija memorije (mnogo manjih rupa koje uzrokuju problem da proces ne može da se smjesti jer ima dovoljno memorije ali ona nije kontinualna). Rješenje ovog problema je sažimanje memorije, tako da se slobodna memorija kontinualizije.

Kod smještanja procesa u memoriju koja je dinamički podjeljena u particije postoje tri opcije:

* + Prvi odgovoarajući blok
  + Najbolji odgovarajući blok
  + Sljedeći odgovarajući blok

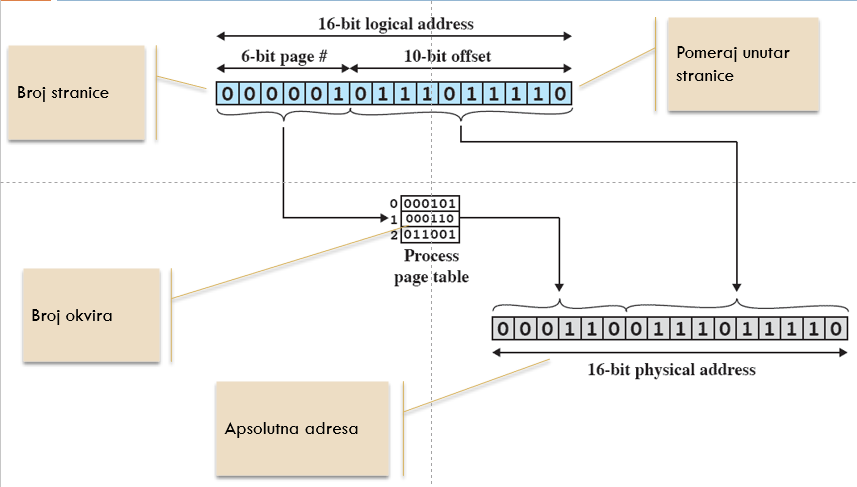
Problem kod djeljenja memorije je interna (Kad se alocira memorija za proces i on ne popuni cijelu particiju, dio koji je neiskorišćen se smatra internom fragmentacijom) i eksterna fragmentacija (Kad se alocira memorija procesu i on se oslobodi, neiskorišćena memorija između procesa se naziva eksternom fragmentacijom memorije). Ovaj metod se ne koristi u savremenim OS.

**Straničenje**

Memorija se deli na male delove jednake veličine, kao i proces. Delovi memorije se zovu **okviri.** Delovi procesa se zovu **stranice.** Veličina okvira i stranice je jednaka.Stranice procesa se mogu dodeljivati okvirima, **koji ne moraju biti susedni.** U varijanti upravljanja memorijom koja se vodi straničenjem nema eksterne fragmentacije dok je interna fragmentacija prisutna samo u posljednoj stranici procesa.

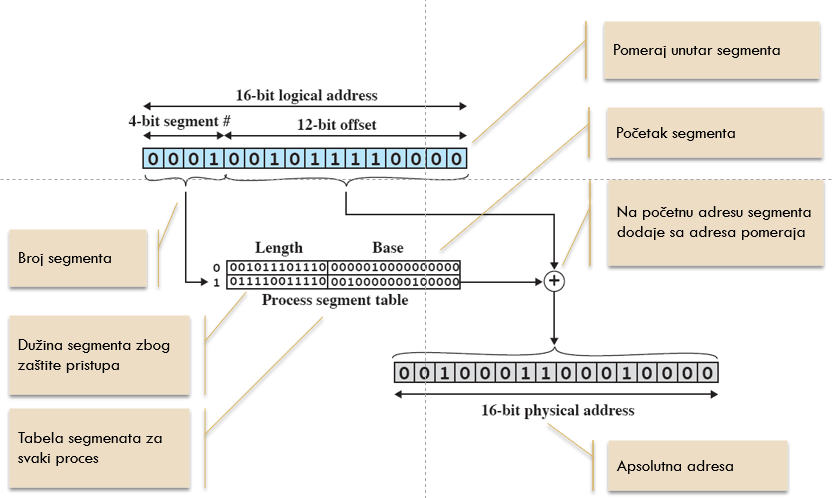
OS za svaki proces održava tabelu stranica, gdje za svaku stranicu tabela sadrži broj okvira u koji je smještena. Na osnovu relativnog pomjeraja i na osnovu broja stranice formira se apsolutna adresa na osnovu koje procesor pristupa stranici.

Relativna adresa označava pomjeraj u odnosu na početak programa procesa. Dok je logička adresa broj stranice i pomjeraj unutar nje. Ako su stranice veličine stepena dvojke, onda su relativna i logička adresa identične.



**Segmentacija memorije**

Segmentacija je način upravljanja memorijom koji se bazira na posmatranju memorije iz ugla programera (Programer vidi program kao skup segmenata, koji mogu biti različite dužine i postoji ograničenje u vidu maksimalne dužine segmenta). Kompajler generiše odvojene segmente za različite dijelove koda, u kojima nema interne fragmentacije. Adresa se sastoji iz broja segmetnta u kome se nalazi i pomjeraj unutar segmenta.



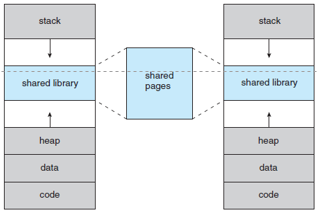
**Osma prezentacija – Virutelna memorija**

Ukoliko su memorijske reference logičke adrese koje se dinamički prevode u fizičke i ako se proces može izdjeliti na manje djelove koji ne moraju zauzimati uzastopne memorijske lokacije u memoriji tada nije neophodno da svi djelovi proces budu istovremeno u glavnoj memoriji u toku izvršavanja. U tom slučaju se svi djelovi procesa ne koriste uvijek i jednako često i iz tog razloga nema potrebe da djelovi koji nisu u upotrebi zauzimaju memoriju.

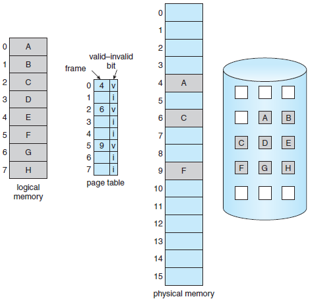
**Virtuelna memorija** – je dio diska namjenjen za smještanje djelova onih procesa koji se trenutno izvršavaju, koji omogućuje efikasno multiprogramiranje.

Prisustvom koncepta virutelne memorije veći broj procesa može da se održava u glavnoj memoriji. (Od svakog procesa se učitava samo dio, pa više različitih procesa može da bude učitano. Sa većim brojem procesa veća je šansa da će u svakom trenutku bar neki od njih biti u stanju *Spreman*)

Sam proces može biti veći od ukupne glavne memorije.

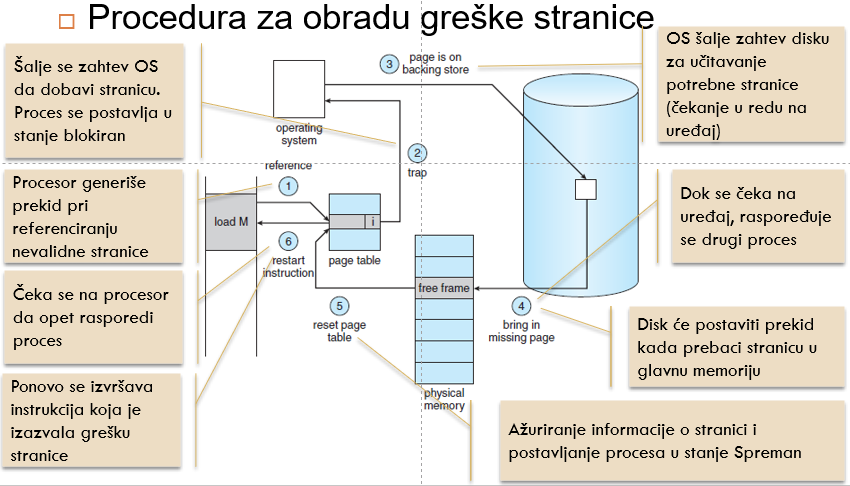
Heap i stack memorija dodjeljena procesu se dinamički mjenja(prostor za ovu memoriju je dio virutelnog adresnog prostora).

Dio virt. memorije može biti deljen od strane više procesa (Ista stranica se mapira u virtuelni prostor više procesa). Primenjuje se za sistemske biblioteke koje djeli više procesa, komunikaciju između procesa putem djeljene memorije, djeljenje istog koda od strane više instanci istog procesa.



**Rezidentni skup** predstavljaju djelovi procesa koji su u glavnoj memoriji u određenom trenutku (svaka stranica nosi informaciju o tome da li je unutar memorije).

Ako se referencira stranica koja nije u glavnoj memoriji to predstavlja ***Page fault***, hardver utvrđuje da je za tu stranicu postavljen invalid bit i potom OS šalje zahtjev da se dobavi potrebna stranica. Ukoliko se referencirana stranica nalazi u memoriju, pristupa se traženoj lokaciji i nastavlja se sa izvršavanjem.



Ako potrebni dio procesa nije u glavnoj memoriji mora se dobaviti sa diska. Ukoliko nema mjesta u glavnoj memoriji za novi dio, dio nekog drugog procesa se mora izbaciti iz glavne memorije.

Prosječno efektivno vrijeme pristupa (mat – vrijeme pristupa memoriji, pft – vrijeme obrade greške stranice, 0 <= p <= 1)

Tri grupe operacija pri obradi greške stranice:

* obrada prekida za grešku stranice
* dobavljanje stranice sa diska – vremenski zahtjevno
* restartovanje instrukcije

**Vrijeme pristupa je direktno proporcionalno učestalosti greške (p).**

**Brbljanje** predstavlja kada sistem provodi najveći deo vremena razmenjujući delove između stvarne i virtuelne memorije umesto u izvršavanju instrukcija.

Izbegavanje brbljanja se vrši tako što OS izbacuje stranicu za koju utvrdi da je najmanja verovatnoća da će biti uskoro referencirana, za koju se predviđanje vrši na osnovu nedavne istorije.

Princip lokalnosti:

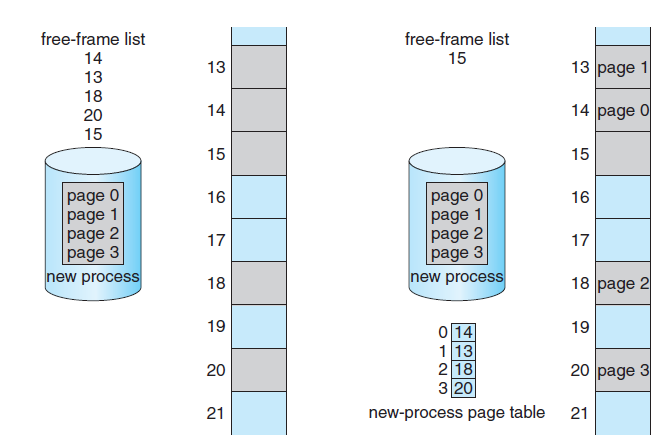
Program i reference teđe da se grupišu unutar procesa. Tokom dužeg perioda mjenjaju se djelovi procesa koji se koriste, međutim u kratkom intervalu procesor uglavnom radi sa malim i ograničenim skupom adresa te je potrebno samo nekoliko djelova procesa u kratkom vremenskom periodu.

Zašto postoji lokalnost u programima?

* Izvršavanje programa je skoro uvek sekvencijalno (Naredna adresa je najčešće ona koja sledi trenutnoj, gdje su izuzetak grananja i pozivi funkcija)
* U kratkom periodu program lokalizovan na određenu funkciju (U funkciji se pristupa ograničenom skupu njenih parametara, lokalnih promenljivih i podskupa globalnih promenljvih, rijetko u programu postoji dugačak uzastopan niz poziva funkcija)
* iterativnih delova koda (Petlje se uglavnom sastoje od malog broja instrukcija koje se ponavljaju mnogo puta, te je obrada ograničena na mali susedni deo adresa)
* Često se podaci čuvaju u strukturi koja čuva podatke u uzastopnim lokacijama (Rad sa nizovima)

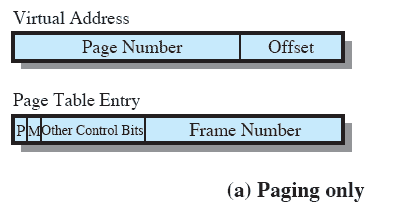
Zbog principa lokalnosti virtuelna memorija je efikasna i moguće je napraviti pretpostavku koji dijelovi procesa će biti potrebni uskoro.

Podrška od strane hardvera za rad virt. memorije se ogleda u tome da hardver procesora mora da podrži adresiranje stranica/segmenata.

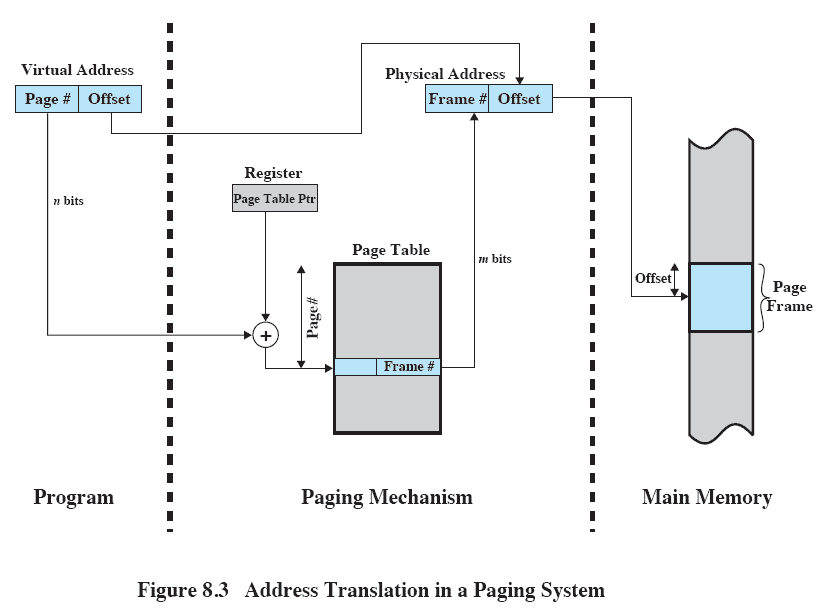
Podrška od strane OS se ogleda u tome da mora pomjerati stranice/segmente između glavne i sekundarne memorije.

Proces je podjeljen na stranice koje su smještene djelom na disku dijelom u glavnoj memoriji.

**Tabela stranica** je evidenciaj u kojim okvirima su stranice smještene. Svaki proces ima svoju tabelu stranica.



Potrebno je znati broj okvira ako je stranica u glavnoj memoriji. P – da li je stranica prisutna u memoriji, M – da li je stranica mjenjana otkako je učitana u glavnu memoriju, ukoliko jeste prije izbacivanja potrebno je upisati izmjenu na disk.



Pri izvršavanju procesa, najčešće se početna adresa tabele stranica drži u registru (u upravljačkom bloku procesa, a pri komutaciji se ubacuje u procesorski registar).

Broj stranice iz virtuelne adrese se koristi za indeksiranje tabele stranica i preuzimanje broja okvira. Broj okvira se kombinuje sa brojem pomeraja za dobijanje fizičke adrese intrukcije.

Ako se koristi virtuelna memorija, pri svakom referenciranju podataka trebaju dva pristupa memoriji:

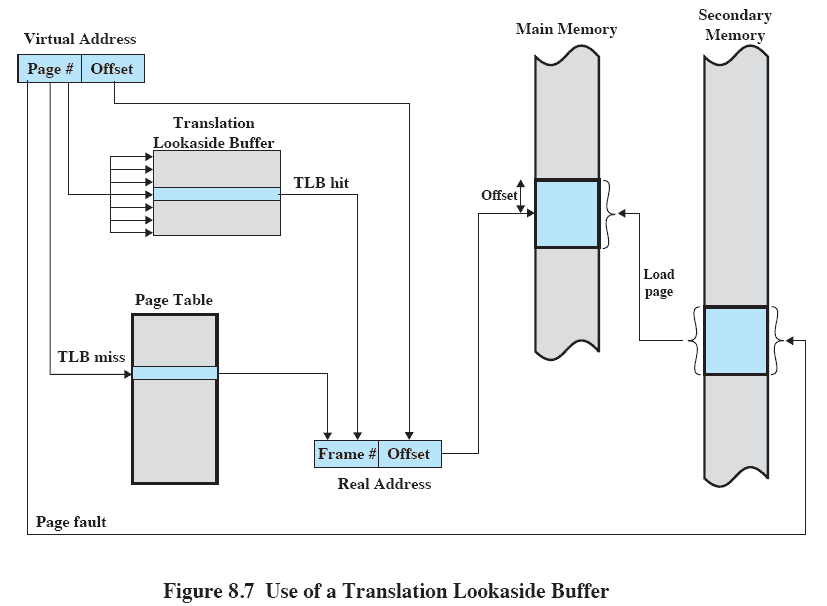
* Jedan da se preuzme stavka iz tabele stranica
* Jedan da se pročita vrednost sa izračunate fizičke adrese

Da bi se prevazišao ovaj problem koristi se specijalni brzi keš za stavke tabele stranica koji se zove **Translation Lookaside Buffer** (TLB) - Sadrži stavke tabele stranica koje su najskorije korišćene

U savremenim procesorima najčešće postoji više nivoa TLB koji su hijerarhijski organizovani po veličini i brzini. (Svaki sledeći nivo veći, ali sporiji)

Obično postoje odvojeni TLB za adrese koje se odnose na instrukcije i one koje se odnose na podatke.

Referenciranje uz TLB

Pri referenciranju na osnovu virtuelne adrese, prvo se provjerava TLB, ako se stavka za referenciranu stranicu nalazi u TLB preuzima se broj okvira i formira se stvarna adresa. A ukoliko se stavka ne nalazi u TLB, pristupa se stavci u tabeli stranica. Tamo se provjerava da li je stranica u glavnoj memoriji (ukoliko nije Page fault, OS prebacuje stranicu iz spoljne u radnu memoriju i ažurira tabelu stranica). TLB se ažurira tako što uključuje referenciranu stavku tabele stranica.

Naravno, TLB sadrži samo neke stavke tabele stranica (nemoguće je smjestiti svaku stavku u TLB.) Jedna stavka TLBa sadrži broj stranice, i kompletnu stavku tabele stranica.

Procesor može istovremeno da provjerava više stavki TLB istovremeno – **asocijativno preslikavanje.** Veličina pretrage može u zavisnosti od procesora varirati između 1 i veličine TLB.

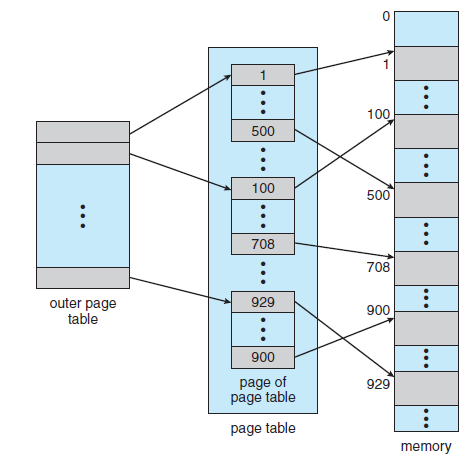
Stavke u TLB baferu se mogu odnositi na sve procese u sistemu i da sadrže podatke o stranicama aktivnog procesa.

Ako se stavke odnose na različite procese u svakoj stavci je potrebna informacija na koji se proces odnosi. Pri referenciranju, pronalaženje je uspešno samo ako se nađe stavka koja se odnosi na referenciranu stranicu konkretnog procesa

Ako TLB ne podržava stavke koje se odnose na različite procese pri komutaciji procesa mora se TLB isprazniti.

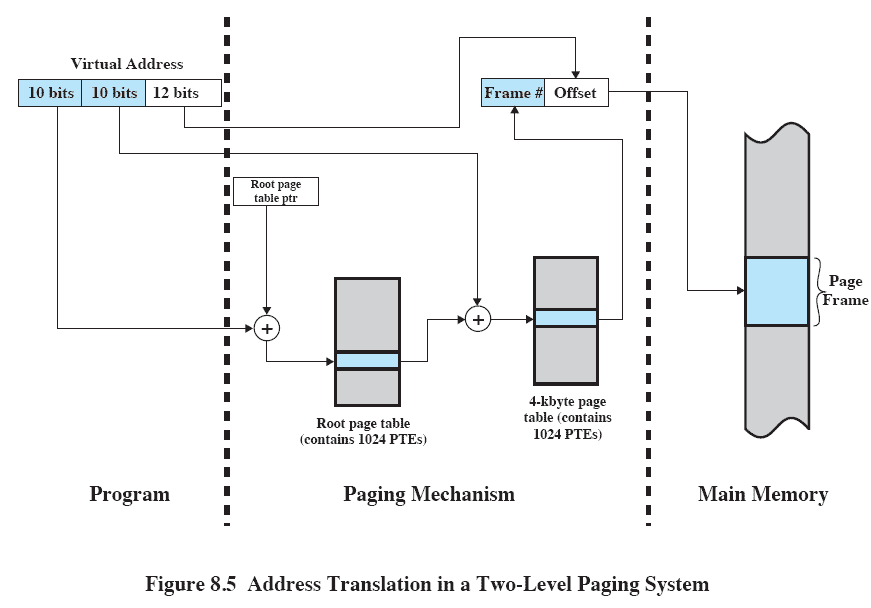
Tabela stranica može da bude prevelika da bi se čuvala u glavnoj memoriji.

Jedna varijanta rešenja je smeštanje tabele stranica u virtuelnu memoriju gdje se tabela stranica se djeli u stranice.

Pri izvršavanju procesa deo tabele stranica mora da bude u glavnoj memoriji, potrebno je pristupiti stavci koja se odnosi na stranicu koja se trenutno referencira.

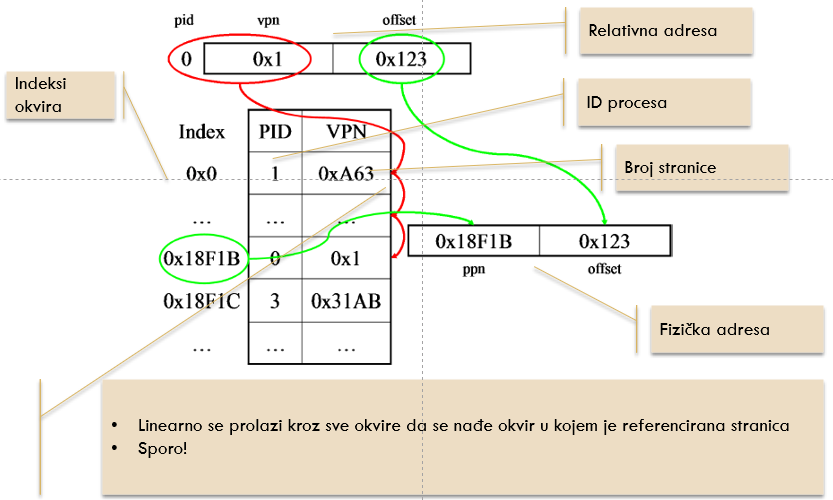
Druga varijanta za prevelike tabele stranica je da postoji mala osnovna tabela stranica čije

stavke pokazuju na stranicu u drugoj tabeli stranica. U toj drugoj tabeli stranica su podaci o stranicama procesa. Može se realizovati u N nivoa, ali najčešće tri.

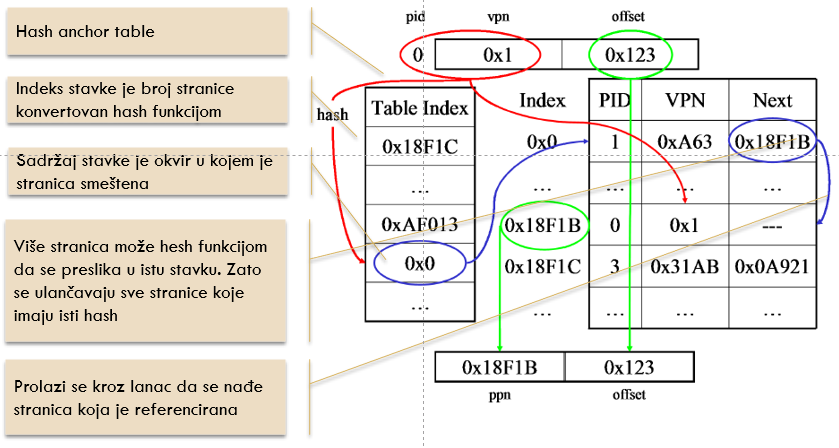


U okviru druge tabele se nalazi 210 sa po 210 stavki.

Loša strana klasične tabele stranica je da je njena veličina proporcionalna veličini adresnog prostora. Jedna od alternativa jeste *invertovana tabela stranica* gdje imamo jednu stavku za jedan okvir (značajno manje stavki od klasične tabele stranica). Svaka stavka specificira koja stranica kojeg procesa je trenutno smještena u okvir.



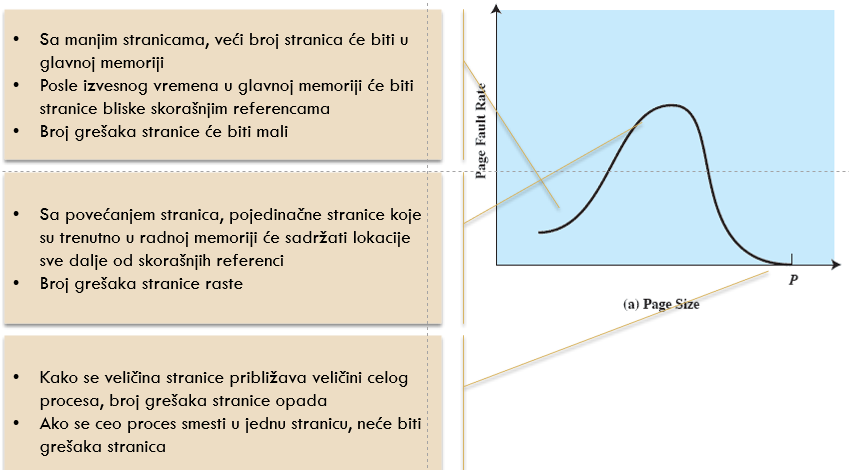
Linearna invertovana tabela stranica



Heširana invertovana tabela stranica

**Zaštita memorije** se implementira dodatnim bitovima u stavci tabele stranica u zavisnosti da li je stranica samo za čitanje, za čitanje i upis ili samo za izršvanje. Zabrana pristupa stranicama koje ne pripadaju procesu je implicitno ugrađena u logiku rada virtuelne memorije. Kako proces vidi samo svoj virtuelni adresni prostoe, proces nema mehanizam da pristupi nečemu što nije deo njegovog adresnog prostora jer referencira virtuelne adrese, a hardver procesora ih mapira na fizičke adrese u radnoj memoriji.

Straničenje omogućuje deljenje zajedničkog koda koji se smješta samo jednom u glavnu memoriju. Različiti procesi u svojoj tabeli stranica referenciraju isti okvir sa deljenim kodom, ali je sekcija za podatke svakom procesu različita.

Veličina stranice:

Sa manjim stranicama biće manje interne fragmentacije, ali sa manjim stranicama, ima više stranica po procesu. (To znači veće tabele stranica)

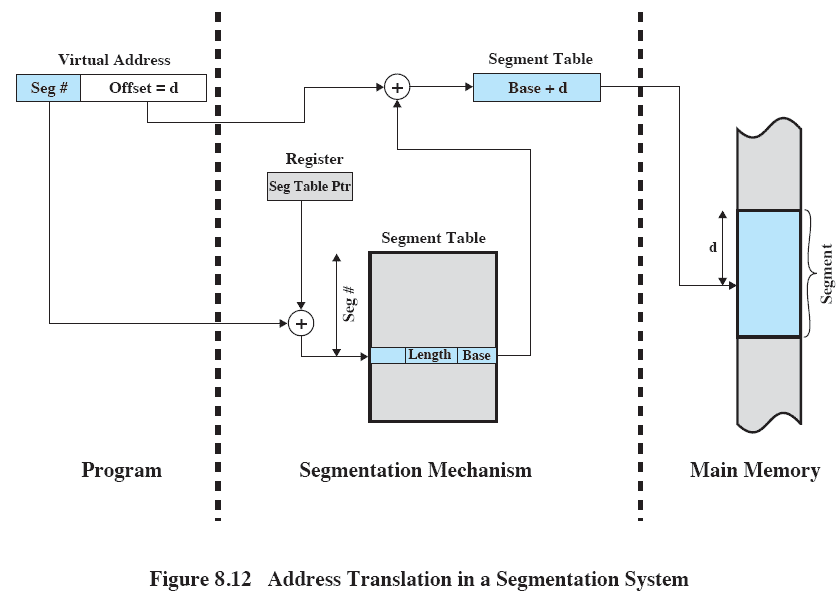
Veća tabela stranica znači da zauzima više prostora u radnoj memoriji ili da je dio nje u virtuelnoj memoriji pa su česte greške stranica za pristup tabeli stranica.

Sekundarna memorija efikasnije prenosi veće blokove podataka pa to daje prednost većim stranicama.

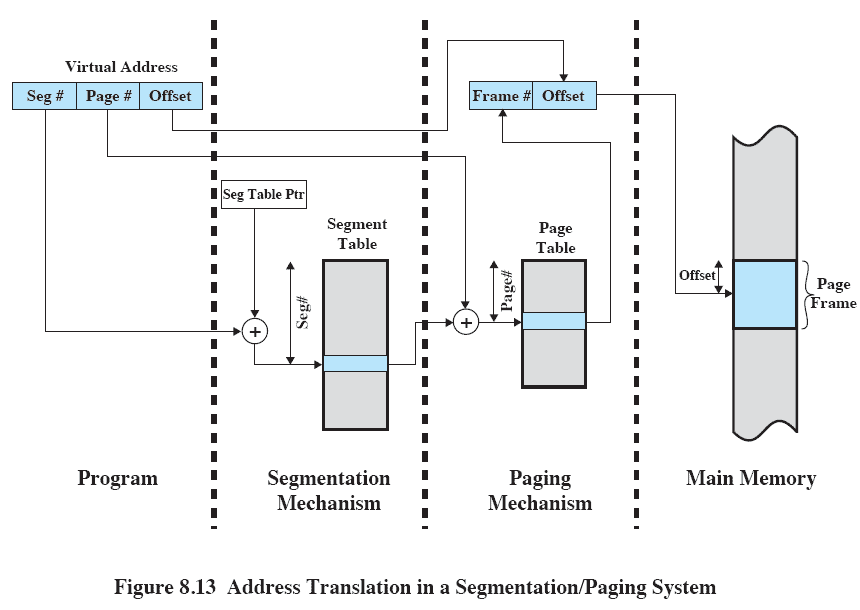
Što se više stranica se ubaci u glavnu memoriju, time je manji broj grešaka stranica. Ali više stranica jednog procesa u memoriji, veća učestalost grešaka stranica za druge procese.

Segmentacija

Proces je podeljen na segmente koji su smešteni na disku. Jedan dio je u glavnoj memoriji.

Svaki segment ima stavku tabele stranica:

* Adresa početka segmenta
* Dužina segmenta
* P – da li je segment u glavnoj memoriji
* M – da li je segment menjan otkako je učitan u glavnu memoriju. Ako jeste, potrebno je pre izbacivanja upisati izmenu na disk
* Segmentacija
  + Vidljiva za programera
  + Nema unutrašnje fragmentacije
  + Podržava modularnost, deljenje i zaštitu
* Straničenje
  + Transparentno za programera
  + Odstranjuje spoljašnju fragmentaciju
  + Olakšava rukovanje memorijom jer su delovi fiksne veličine

Ukoliko imamo kombinovani pristup straničenja i segmentacije. Adresni prostor se dijeli u određen broj segmenata, te se svaki segment dijeli na stranice. Adresa sadrži broj segmenta i broj stranice unutar segmenta. Za svaki proces potrebna je tabela segmenata, a za svaki segment potrebna je tabela stranica.

**Deveta prezentacija – Virutelna memorija 2**

Hardver treba da podrži tehnike virtuelne memorije i da podržava straničenje, segmentaciju ili oboje.

OS je odgovoran za algoritme koji se primjenjuju pri upravljanju memorijom.

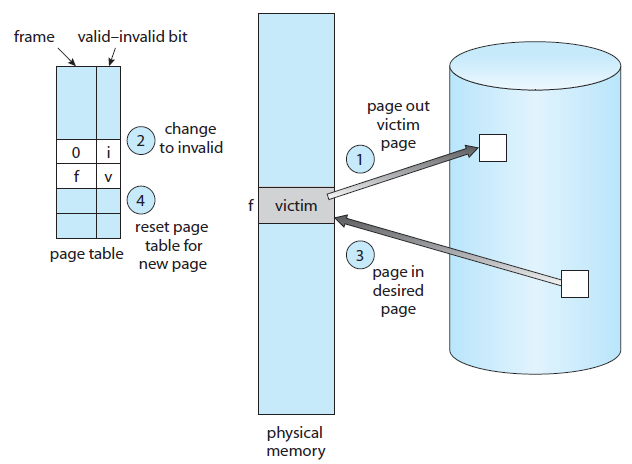
Najvažnije je pitanje performanse:

* + potrebno je minimizovati učestalost grešaka stranica
  + OS donosi odluke kada, kako i koju stranicu zamjeniti
  + dok se zamjenjuje stranica, OS raspoređuje drugi proces

Politika donošenja:

Dvije osnovne varijante:

* + Straničenje po zahtjevu – stranica se donosi u glavnu memoriju samo kada se napravi referenca na lokaciju u toj stranici (Mnogo grešaka stranica u početku rada programa, kasnije broj grešaka stranice opada)
  + Predstraničenje – donose se unaprijed stranice koje nisu zahtjevane. Efikasnije je donijeti više stranica odjednom ako se na disku nalaze u fizički susjednim lokacijama, ali može se nepotrebno donijeti stranica koja se neće koristiti.

Politika smještanja – određuje gdje u glavnoj memoriji proces treba da bude smješten, ali ukoliko se koristi straničenje ovo pitanje je suvišno jer hardver radi jednako brzo bez obzira u kom okviru u stvarnoj memoriji se stranica nalazi.

Politika zamjene – kada su svi okviri u glavnoj memoriji zauzeti potrebno je odrediti koja stranica će ustupiti mjesto dolazećoj stranici. Cilj je naravno da se minimizuje broj grešaka stranice tako što će biti zamjenjena ona koja u skorije vrijeme neće biti referencirana.

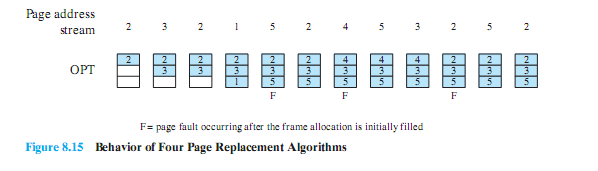
Ukoliko se uvede još jedan bit koji označava da li je stranica izmjenjena u periodu dok je bila u glavnoj memoriji, tada se ako je stranica bila mjenjana onda čuva na disk, u suprotnom nema potrebe.

Postoji i koncept *zaključavanja okvira* koji se realizuje uvođenjem bita zaključanosti, koji ima za svrhu da drži u sebi kernel OSa, glavne upravljačke strukture, U/I bafere itd. Međutim treba biti oprezam sa ovim, jer uvođenjem zaključavanja smanjuje se broj raspoloživih okvira glavne memorije.

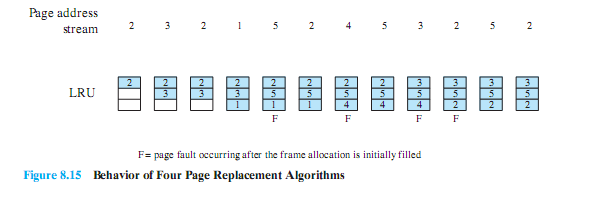
*Program prilikom izvršavanja redom referencira određene lokacije u memoriji. Politike zamjene se porede kroz broj grešaka stranice na određenom nizu referenci.*

*Primjer niza referenci: 2 3 2 1 5 2 4 5 3 2 5 2*

Politike za izbor stranice za zamjenu:

* + Optimalna – bira se ona stranica za koju je vrijeme do sledeće reference najduže. Ova politika nije moguća za implementaciju jer zahtejva da OS poznaje buduće događaje. Obično služi kao reper za ostale stranice.
  + Najmanje skoro korišćena (LRU)– bira se ona stranica koja najduže nije bila korišćena pod pretpostavkom da ni uskoro neće biti korišćena. Po principu lokalnosti trebalo bi da bude zamjenjena stranica za koju je najmanje vjerovatno da će biti referencirana u bliskoj budućnosti. Dvije varijante:
    - brojač – procesor evidentira broj otkucaja, a u tabeli stranica se doda informacija koja označava trenutak posljednjeg referenciranja. Prilikom svakog referenciranja ažurira se ta info. sadašnjim trenutkom. Zamjenjuje se ona sa najmanjim trenutkom u vremenu.
    - stek – čuva se stek referenciranih stranica, kada se stranica referencira stavlja se na vrh. Ako je već bila referencirana, izbacuje se sa tog mjesta i prebacuje na početak potiskujući ostale.

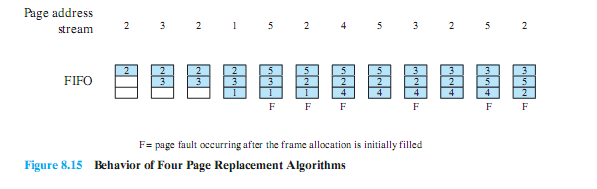
Ni jedna od ovih implementacija nije jednostavna, te se ova politika najrjeđe koristi.



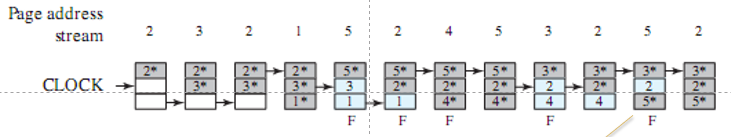
Postoje aproksimacije LRU politike. Baziraju se na tome da hardver za svaku stranicu ažurira bit upotrebe. Kada se stranica referencira ili donese u memoriju bit se ažurira i postavlja na 1. Ovim dobijamo info. koje su stranice referencirane, ali ne i u kojem redoslijedu.

Politika dodatnih bitova upotrebe –

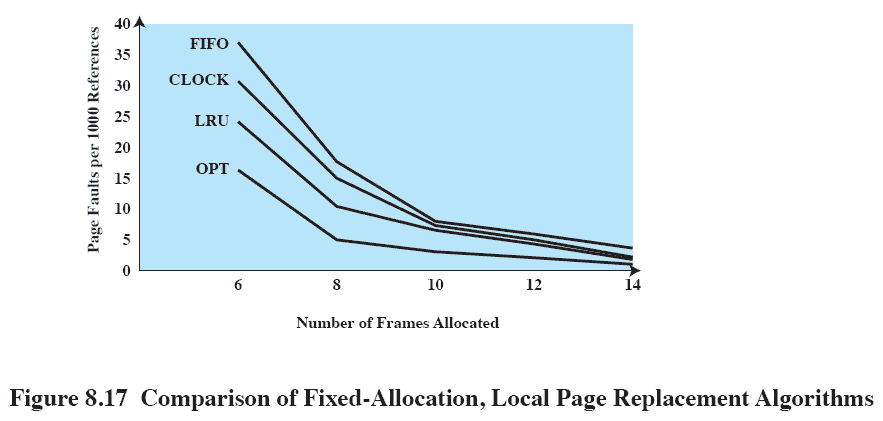
OS evidentira 8-bitnu reč za svaku stranicu. U regularnim intervalima OS prebacuje bit upotrebe na najvišu poziciju u reči, trenutne bite pomera u desno, dok bit na najmanjoj poziciji ispada iz reči. Ova 8-bitna reč time sadrži istoriju referenciranja. Ako 8-bitnu reč posmatramo kao dekadni broj stranica sa većom vrednošću reči je skorije referencirana. (11000100 je skorije referencirana od 01110111) Stranica sa najmanjom vrednošću će biti zamenjena.

* + Prva unutra, prva napolje – tretira okvire kao kružni bafer. Postoji pokazivač na narednu stranicu koju treba ukloniti (na početku on pokazuje na prvu stranicu), i ovaj pokazivač kruži redom kroz okvire. Biće zamjenjena ona stranica koja je najduže u memoriji (greška ukoliko posmatramo da stranice koje se često referenciraju će stalno biti izbacivane i ponovo ubacivane u glavnu memoriju)

Za razliku od “najmanje skoro korišćena” politike ova ne prepoznaje stranice koje su češće referencirane od ostalih.

* + Časovnik – samo jedan bit upotrebe za svaku stranicu. Kada je potrebno pronaći stranicu za zamjenu OS prolazi kružno kroz okvire i kad god naiđe na okvir sa bitom upotrebe 1, postavi ga na 0. Biće zamjenjena stranica za koju je bit upotrebe 0. Naredno pretraživanje će krenuti od okvira koji se nalazi poslije okvira koji je prethodno bio zamjenjen. 

Algoritam časovnika može koristiti i bit zamjene tako da prednost za zamjenu daje stranicama koje nisu mjenjane, time se štedi vrijeme jer ne treba upisivati izmjenu u sekundarnu memoriju prije zamjene.



Baferovanje stranica – dio glavne memorije se odvoji da radi kao keš stranica. Zamjenjena stranica se ubacuje u taj keš (Ako nije mjenjana dodaje se u listu slobodnih, ako je mjenjana dodaje se u listu mjenjanih stranica). Kada se ponovo referencira postoji šansa da se stranica nalazi u baferu, pa je vraćanje u upotrebu brzo i jednostavno. Izmjenjene stranice se mogu upisivati u grupama što je takođe brže.

**Rezidentni skup** – za svaki proces OS odlučuje koliko okvira da mu dodjeli za smještanje stranica.

* + Što je manje memorije dodjeljeno jednom procesu, to više proces može istovremeno da bude u memoriji
  + Što je manji broj okvira dodjeljen procesu, biće više grešaka stranice
  + Zbog lokalnosti, povećavanje broja dodjeljenih okvira preko određene granice nema zapažen efekat.
  + Neophodno je da se procesu dodjeli bar onoliko okvira koliko svaka pojedinačna instrukcija može da referencira stranica (Arhitektura skupa instrukcija diktira minimalan broj dodjeljenih okvira procesu).
  + Teži se da bude što manje grešaka stranice!

Politike veličine rezidentnog skupa:

* Fiksno dodjeljivanje :
  + Jednako dodjeljivanje – Svaki proces dobija n-ti dio od ukupnog broja okvira, međutim to je neracionalno kada su procesi različitih veličina (Isto tako proces može dobiti i više okvira nego što stranica posjeduje)
  + Proporcionalno dodjeljivanje – Proces dobija broj okvira proporcionalno broju svojih stranica u odnosu na broj stranica svih drugih procesa.
* Promjenljivo dodjeljivanje – broj okvira dodjeljenih procesu se mjenja u toku životnog vijeka procesa. U odnosu na broj grešaka stranica procesu se dodjeljuje više/manje okvira.

Opseg zamjene stranice u slučaju greške stranice:

* *Lokalni opseg zamjene* – bira se stranica za zamjenu iz procesa koji je napravio grešku
* *Globalni opseg zamjene* – bira se stranica od svih stranica koje se nalaze u nezaključanim okvirima. (savremeni OS)

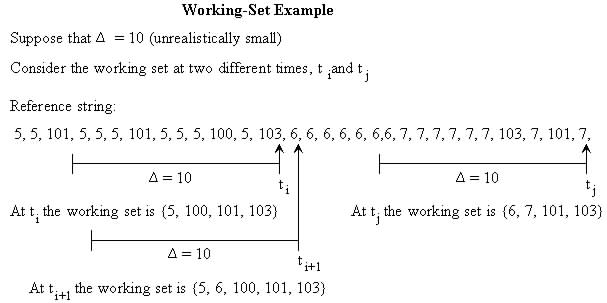
Fiksno dodjeljivanje uz lokalni opseg zamjene je nefleksibilno rješenje (malo okvira – puno grešaka i obrnuto).

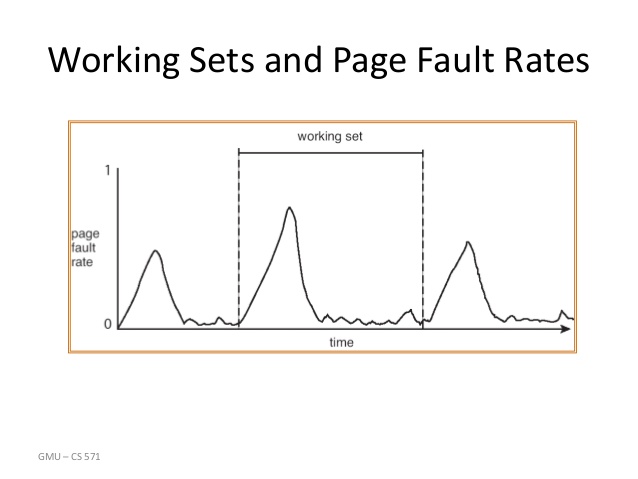
Fiksno dodjeljivanje uz globalni opseg zamjene nema smisla!

Promjenljivo dodjeljivanje uz globalni opseg zamjene odlikuje to što proces dobija slobodan okvir kada se desi greška stranice. Ako nema okvira, zamjenjije se neka stranica (drugog procesa čime se smanjuje rezidentni skup tog procesa, smanjuje u performanse).

Promjenljivo dodjeljivanje uz lokalni opseg zamjene odlikuje to što postoji povremena analiza broja okvira dodjeljenih procesu pa se vrši smanjenje/povećanje broja radi poboljšanja performansi.

**Radni skup** - se zasniva na principu lokalnosti. u trenutku t, *skup stranica koje su bile referencirane* u posljednjih x jedinica vremena. Vrijeme se mjeri referenciranjem stranica, jedno referenciranje je jedna jedinica vremena.



Radni skup zbog principa lokalnosti ima relativno stabilne periode i periode kada se drastično mjenja zbog prelaska na novu lokalnost.

Strategija radnog skupa svakog procesa nalaže da se periodično uklanjaju one stranice iz memorije koje nisu u radnom skupu. (Nepraktično i zahtjevno je mjerenje radnog skupa).

PFF – Page fault frequency – nalaže da umjesto nadgledanja radnog skupa nadgledamo ušestalost grešaka stranice. Učestalost grešaka opada povećanjem rezidentnog skupa. Ako je učestalost manja od nekog praga, smanjujemo rezidentni skup i obratno.

Algoritam učestalosti greške:

* Svakoj stranici se pridružuje bit upotrebe
* Brojač referenci koji broji koliko je proteklo referenci od poslednje greške stranice
* Kada se desi greška stranice, poredi se brojač sa pragom *F*
  + Ako je vreme od poslednje greške stranice manje od F, rezidentni skup se proširuje novom stranicom
  + Ako je vreme veće od F, odbacuju se stranice sa bitom upotrebe 0 i tako smanjuje rezidentni skup
  + Pri svakoj greški stranice, resetuje se na 0 bit upotrebe svim stranicama u rezidentnom skupu

Prilikom prelaza na novu lokalnost algoritam ne radi baš najbolje (Povećava se radni skup a da bi se stranica izbacila iz radnog skupa treba proći F jedinica virtuelnog vremena od kad je posljednji put bila referencirana.)

Radni skup sa promjenljivim intervalom uzorkovanja zahtjeva da se u izvršavanju procesa mjeri virutelno vrijeme. Ako je prošao maksimalni period od posljednjeg uzorkovanja suspenduje se proces i skeniraju se bitovi upotrebe. Ako u toku intervala broj grešaka stranice pređe određeni prag i ako je prošao minimalan period uzorkovanja ponovo se vrši uzorkovanje. Ako nije prošao, čeka se da istekne minimalni period i tada se vrši uzorkovanje. Ova varijanta bolje radi na prelazima, jer stranice stare lokalnosti će brže ispadati iz rezidentnog skupa nego kod PFF algoritma.

Politika čišćenja odlučuje kada se izmjenjena stranica upisuje nazad u sekundarnu memoriju. Stranica se upisuje kada je izabrana za zamjenu. Postoji koncept predčišćenja, gdje se stranice upisuju u paketima unaprijed, međutim ako dođe ponovo do izmjene trošili smo bespotrebno resurse. Za politiku čišćenja najbolji pristup je *baferovanje strnica****.*** Lista izmjenjenih se periodično upisuju na disk. Iz liste neizmjenjenih stranice se vraćaju u okvire dodjeljene procesu ako se referenciraju ili izbacuju iz glavne memorije ako nema više mjesta za njih u listi.

Suspenzija procesa se dešava ukoliko je previše procesa u memoriji te neki od njih mora biti suspendovan. Izbor procesa za suspenziju može biti:

* + Proces najnižeg prioriteta
  + Proces koji je izazvao grešku (svakako nema randi skup u memoriji, svakako bi bio blokiran dok čeka na zamjenu stranice)
  + Proces koji je najduže bio aktivan – za ovaj proces je najvjerovatnije da neće imati svoj radni skup u memoriji
  + Proces sa najmanjim rezidentnim skupom – jer je najmanji napor da učita svoj radni skup ponovo u memoriju
  + Najveći proces – dobijamo najviše slobodnih okvira
  + Proces koji je najmanje hitno da bude završen.