Projektovanje elektronskih sistema

Predavanje 5
Operativni sistemi

Doc.dr Borisav Jovanović

preuzeto iz predavanja prof. Milunke Damnjanovic i prof. Miluna Jevtica

Sadržaj:

- Operativni sistemi.
- Osnovna podela softvera i zadatak operativnog sistema.
- Slojevi računarskog sistema.
- Programi i procesi.
- Zavisni i nezavisni procesi.
- Sinhronizacija procesa.
- Osnovna stanja procesa.
- Prelazi između stanja procesa. PCB – Process Control Block.
- Osnovne operacije OS sa procesima.
- Podela OS prema broju procesa. Heap. Stack.
- Konkuretno i sekvencijalno programiranje.

Sledeća>

UVOD

Osnovna podela softvera i zadatak operativnog sistema

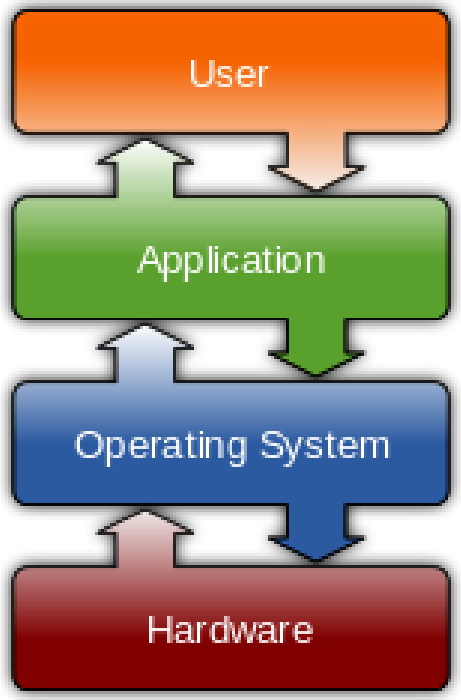
Softver se može podeliti u dve grupe:

- 1. sistemski programi – upravljaju računarom
- 2. korisnički (aplikacioni) programi – rešavaju probleme korisnika

Operativni sistem je fundamentalni deo sistemskih programa, čiji je zadatak upravljanje resursima računara i koji obezbeđuje osnovu za pisanje aplikacionih programa.

- Kako OS obezbeđuje osnovu za pisanje aplikacionih programa?

- Računar je kompleksni sistem sastavljen od raznih delova, kao što su: procesor, memorija, diskovi, tastatura, miš, štampač, skener itd.
- Pisanje programa na način da se ti delovi računara programiraju direktno je vrlo težak posao. Zato je došlo do ideje da se stavi jedan sloj između aplikacionih programa i hardvera.
- Uloga tog sloja je da obezbedi neki interfejs (virtuelnu mašinu) ostalim programima radi lakšeg i bezbednijeg pristupa hardveru.
- Taj sloj je upravo OS.



Koji su slojevi računarskog sistema?

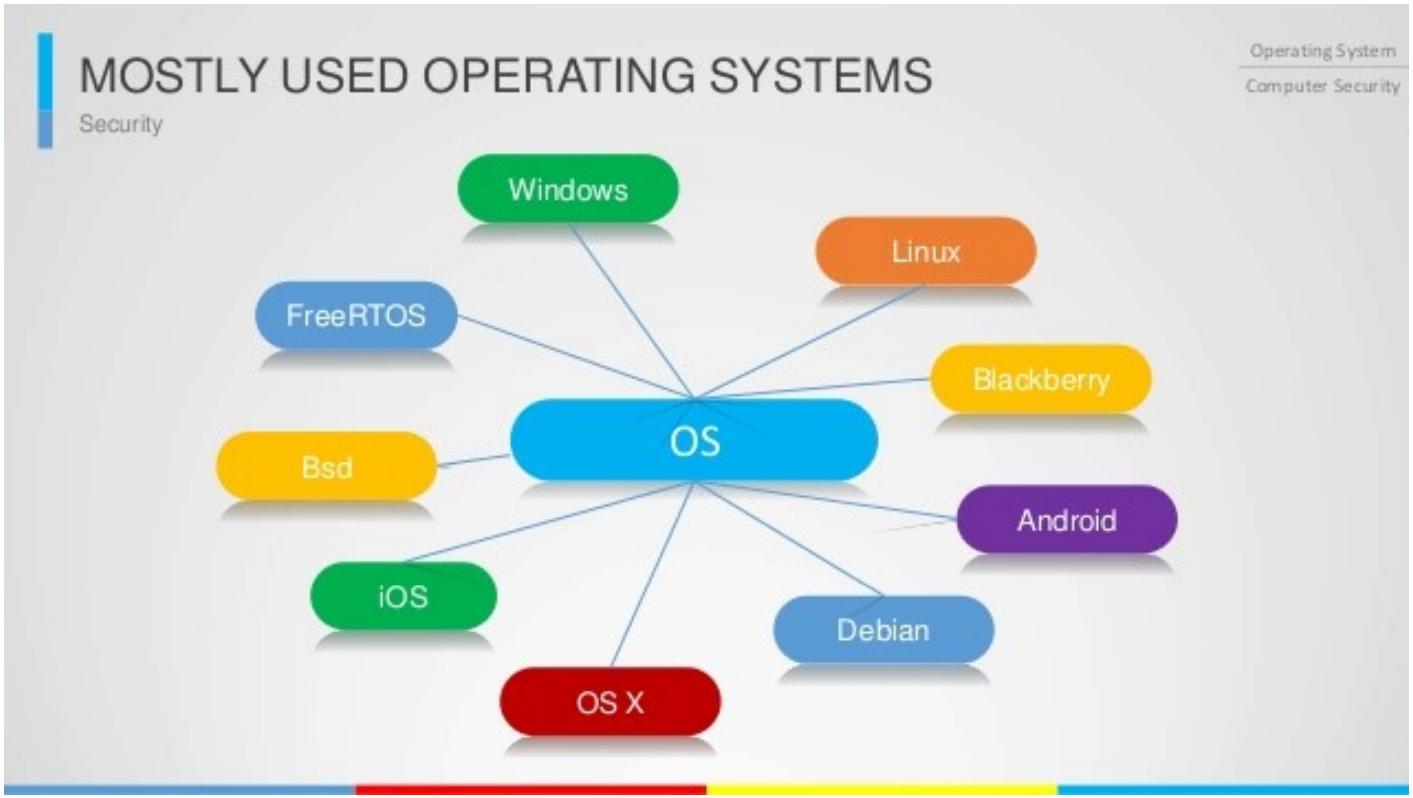
<table>
<tr><td>office</td><td>baze podataka…</td><td>igre…</td><td>Korisnički programi</td></tr>
<tr><td>kompajleri,interpreteri</td><td>editori</td><td>linkeri</td><td rowspan="2">Sistemski programi</td></tr>
<tr><td colspan="3">operativni sistem</td></tr>
<tr><td colspan="3">mašinski jezik</td><td rowspan="3">HARDVER</td></tr>
<tr><td colspan="3">mikro programi</td></tr>
<tr><td colspan="3">fizički uređaji</td></tr>
</table>

- Na najnižem nivou imamo fizičke uređaje – delove računara.
- Iznad tog sloja su mikro programi – direktno kontrolišu fizičke uređaje i obezbeđuju interfejs prema sledećem nivou. Predstavljaju elementarne korake od kojih se sastoje instrukcije mašinskog jezika.
- Mašinski jezik je skup instrukcija koje procesor direktno razume (izvršava ih pomoć u svojih mikroprograma).
- Glavna funkcija operativnog sistema je sakrivanje detalja ovih nižih nivoa od ostalih programa i pružanje niza jednostavnijih instrukcija za pristup hardveru.

- Šta je operativni sistem?

- **OS kao proširena (extended) ili virtuelna (virtual) mašina.**
- Arhitektura (niz instrukcija, organizacija memorije, IO, itd.) računara na nivou mašinskog jezika je primitivna i nije pogodna za programiranje.
- Zadatak OS-a (kao proširene ili virtuelne mašine) je upravo upravljanje delovima računara radi umesto nas i da nam pruža neke funkcije višeg nivoa apstrakcije radi lakog pristupa hardveru.
- **Radi kao upravljač resursima računara**
- OS ima zadatak, da vodi računa u resursima računara – da zadovolji potrebe programa, da prati koji program koristi koje resurse, itd.

Osnovni pojmovi, koncepti OS-a

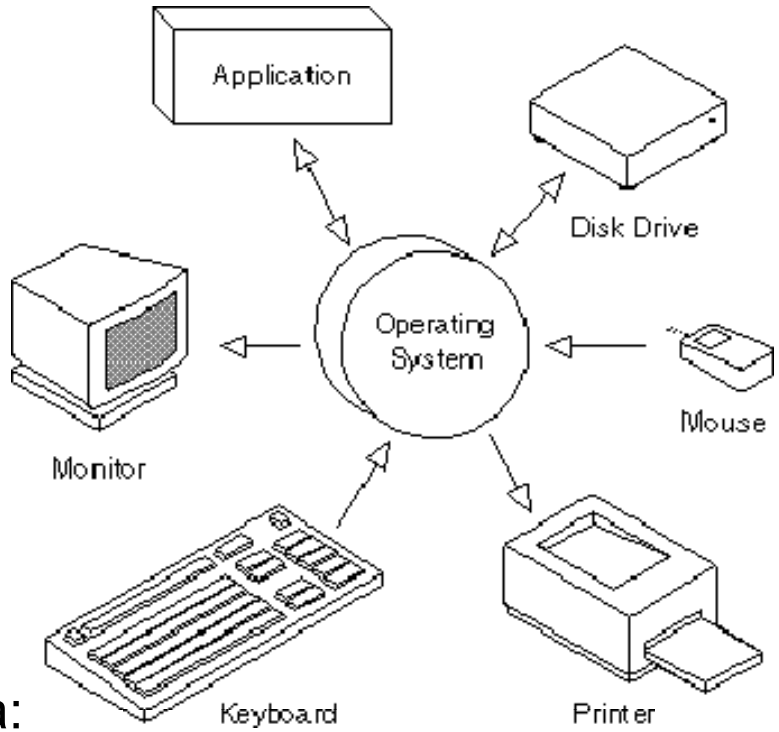


- Imamo hardver, operativni sistem i korisničke programe. Videli smo da je jedan od zadataka OS-a da sakrije hardver od aplikacionih programa, odnosno da obezbedi lakši pristup hardveru. To se ostvaruje preko niza proširenih instrukcija, koji se zovu sistemski pozivi (*System calls*).

Koja je razlika razlika između programa i procesa?

- **Program** je niz instrukcija koji ostvaruje neki algoritam.
- Procesi predstavljaju jedan od najvažnijih koncepata operativnih sistema.
- Procesi su osnovni elementi koji se koriste pri gradnji operativnog sistema. Operativni sistem se sastoji od skupa asinhronih procesa koji moraju međusobrno da komuniciraju i sinhronizuju svije aktivnosti.
- **Proces** je program u statusu izvršavanja, zajedno sa svim resursima koji su potrebni za rad programa.
- Pojednostavljeno: program je samo fajl na hard disku. Tek kada se taj fajl učita u memoriju i počinje da se izvršava dobijemo proces.

Tokom napredovanja izvršavanja procesa unutar računara u svakom trenutku se mogu uočiti promene sadržaja registara procesora kao i odgovajućih memorijskih lokacija.

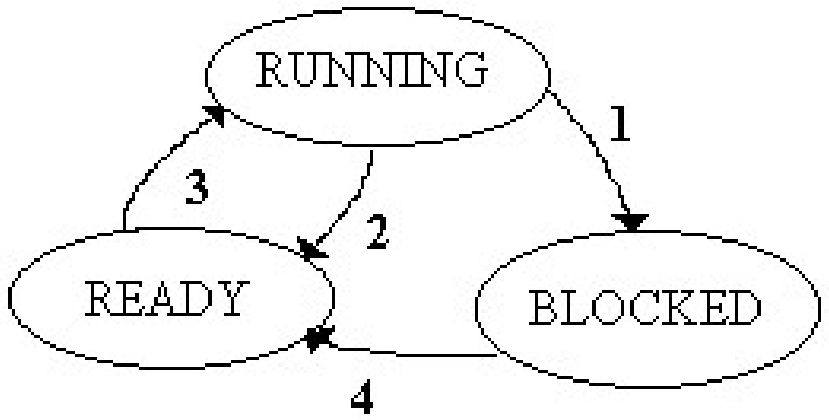


Koja su osnovna stanja procesa?

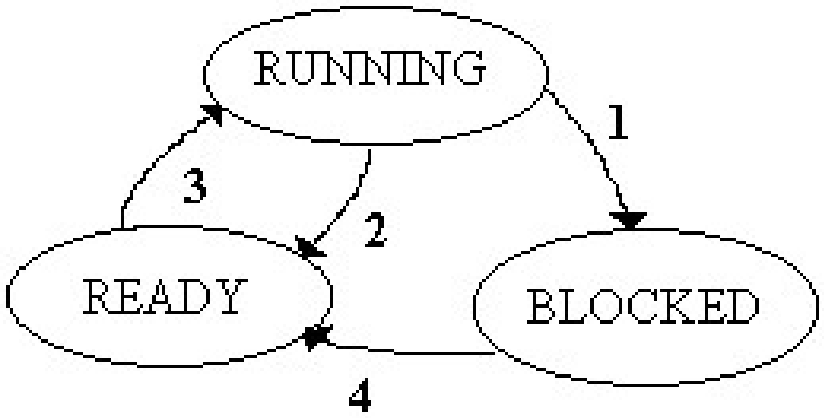
Procesi se nalaze u jednom od sledećih stanja:

- **proces se izvršava** (RUNNING) - procesor upravo izvršava k**o**d ovog procesa
- **proces je spreman**, ali se ne izvršava (READY) - proces je dobio sve potrebne resurse, spreman je za izvršavanje, ali čeka procesor
- **proces je blokiran**, čeka na nešto (npr. čeka štampač a da završi sa štampanjem – BLOCKED) - za dalji rad procesa potrebni su neki resursi, koji trenutno nisu na raspolaganju, čeka IO operaciju, rezultat nekog drugog procesa itd.

Koji su osnovni prelazi između stanja procesa?

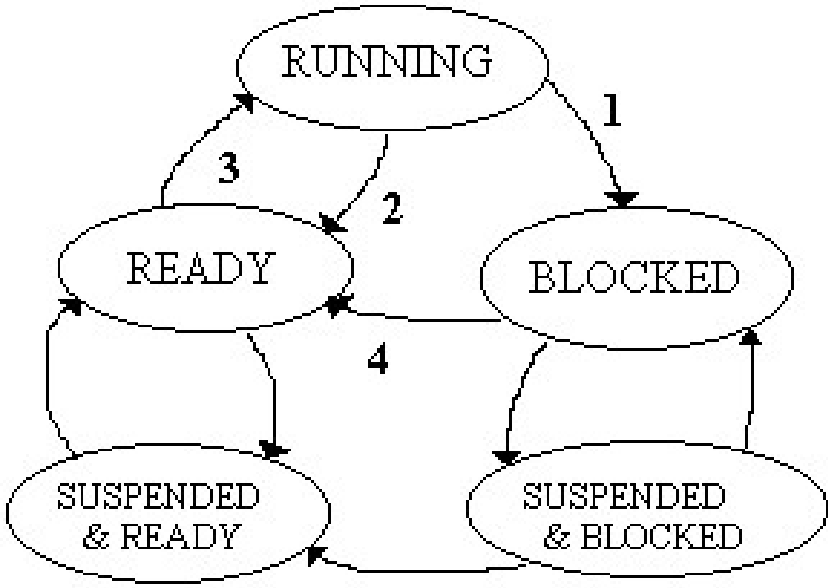


- Imamo četiri prelaska između različitih stanja:
- **1. proces prelazi iz stanja IZVRŠAVANJA u stanje BLOKIRAN,** kada su mu za dalje izvršavanje potrebni neki resursi, koji trenutno nisu dostupni. Ovu promenu stanja vrši sam proces: predaje zahtev za neki resurs, pa čeka pojavljivanje tog resursa. Npr.: pošalje zahtev skeneru da skenira neku sliku i čeka rezultat skeniranja.
- **2. proces prelazi iz stanja IZVRŠAVANJA u stanje SPREMAN,** ako mu istekne dodeljeno procesorsko vreme (time-sharing) – tada proces prelazi u listu procesa koji čekaju na procesor.



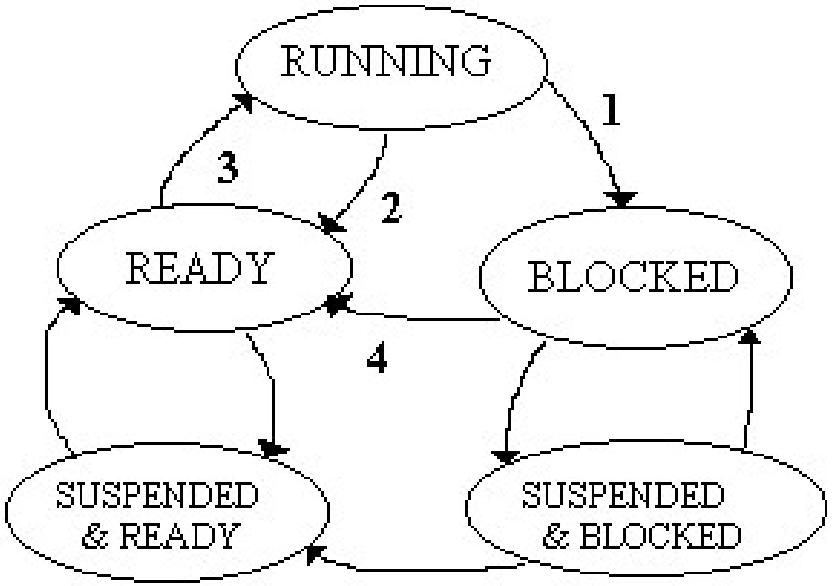
- 3. **proces prelazi iz stanja SPREMAN u stanje IZVRŠAVANJA**, kada se procesor oslobodi i može da izvršava k**o**d posmatranog procesa (izabere se iz liste čekanja po nekom kriterijumu i izvršava se)
- 4. **proces prelazi iz stanja BLOKIRAN u stanje SPREMAN**, kada dođe do potrebnih resursa i spreman je za dalji rad, ali procesor trenutno nije slobodan, pa prelazi u listu čekanja (npr. skener je završio skeniranje, i sad proces može nastaviti sa radom (spreman je), ali procesor je trenutno zauzet izvršavanjem nekog drugog procesa, pa mora da čeka u red…)

Zašto se procesi suspenduju?



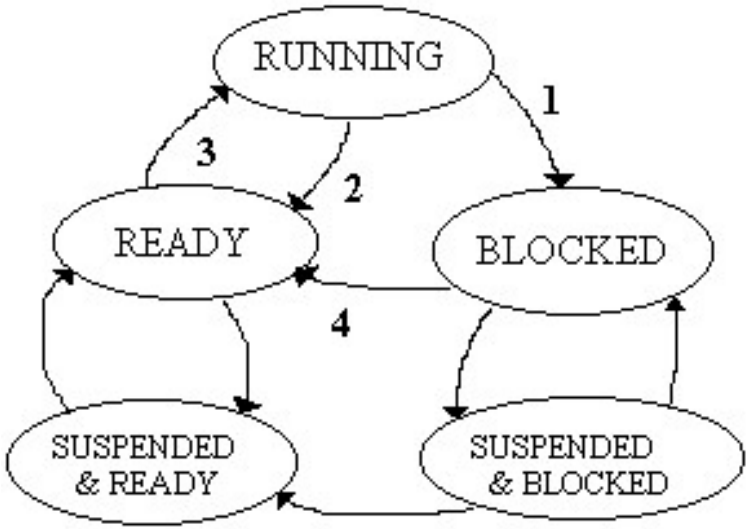
- Kod nekih operativnih sistemima procesi mogu biti i suspendovani (suspended). Na taj način dobijamo još dva stanja:
- proces je suspendovan i spreman (ako je došlo do suspendovanja u stanju spreman)
- proces je suspendovan i blokiran (ako je došlo do suspendovanja u stanju blokiran)

Proces koji je suspendovan, prestaje da se takmiči za resurse, oslobađaju se resursi koje je zaouzeo, ali ostaje i dalje proces.



- Proces koji je u stanju **suspendovan i blokiran** prelazi u stanje **suspendovan i spreman**, ako postaje spreman, tj. ako može da nastavi sa radom (npr. proces pošalje zahtev skeneru da skenira sliku, čeka da skener završi sa radom, pa se blokira, u međuvremenu se suspenduje, pa postaje suspendovan i blokiran, kada skener završi skeniranje, proces prelazi iz stanja suspendovan i blokiran u stanje suspendovan i spreman.)
- Iz stanja **suspendovan i blokiran** u stanje **blokiran** i iz stanja **suspendovan i spreman** u stanje **spreman** procesi mogu preći samo eksplicitno, tj. zahtevom korisnika

- Iz stanja spreman u stanje suspendovan i spreman proces prelazi iz nekog od sledećih razloga:



- **prevelik broj spremnih procesa –** procesi se suspenduju kao zaštita od preopterećivanja sistema

- **explicitno suspendiovanje procesa od strane korisnika** (npr. da bi korisnik mogao proveriti neke međurezultate izvršavanja procesa – i nakon toga mogao nastaviti rad bez ponovnog pokretanja celog programa.)
- **izbegavanje zaglavljivanja (dead lock) –** do zaglavljivanja se dolazi kada dva (ili više) procesa blokiraju jedan drugi u izvršavanju (npr. procesu P1 treba resurs A koji je kod procesa P2, a procesu P2 treba resurs B koji drži P1 – ovi procesi su se zaglavili, jer nijedan od njih ne može nastaviti sa radom – u ovom slučaju jedan od procesa se suspenduje, pa drugi može da odradi svoj zadatak, pa kada se resursi oslobode i prvi će moći da završi svoj rad).

Kod takmičenja više procesa za korišćenje ograničenog skupa računarskih resursa moraju se poštovati sledeća pravila:

- U nekom trenutku samo jedan resurs sme da koristi samo jedan proces
- Kada se istovremeno pojave više zahteva za korišćenjem istog resursa, procesor dodeljuje resurs samo jednom procesu i to tokom ograničenog vremenskog intervala
- Kada proces dobije resurs, mora ga predati u konačnom vremenu.
- Dok čeka dodelu resursa, proces ne troši vreme procesora

12. Šta je PCB – Process Control Block?

- Posmatrajmo sledeću situaciju: imamo okruženje sa dva procesa.
- Proces P1 se izvršava dok ne dođe do blokiranja zbog čekanja na neki događaj (npr. skeniranje).
- Tada krenemo sa izvršavanjem procesa P2 koji se nakon nekog vremena isto postaje blokiran.
- U međuvremenu se desi događaj na koji je čekao proces P1 i sada možemo nastaviti sa izvršavanjem.
- Da bismo znali, gde treba nastaviti, potrebno je pamtiti neke informacije o procesu.
- Upravo tome služi PCB tj. **Process Control Block**.

- Svakom procesu se dodeljuje jedinstveni PCB koji sadrži informacije koje su potrebne za nastavak izvršavanja tog procesa. Te informacije uključuju:
  - jedinstveni identifikator procesa (pid – process ID)
  - stanje procesa I prioritet procesa (iz liste čekanja biramo najpre procese sa većim prioritetima)
  - adresa memorije gde se nalazi proces I adrese zauzetih resursa
  - sadržaj registara procesora, itd.
- PCB-ovi svih procesa u memoriji smeštaju se u neki niz ili povezanu listu

Operacije sa procesima

- Operativni sistem treba da obezbedi sledeće operacije sa procesima:
  - kreiranje novog procesa (kreiranjem PCB-a)
  - uništavanje procesa (brisanjem PCB-a iz liste)
  - menjanje stanja procesa
  - menjanje prioriteta procesa
  - izbor procesa za izvršavanje (dodela procesora nekom procesu **– context switch**)
  - sinhronizacija procesa
  - komunikacija između procesa ...

Odnos procesa

- Sami procesi mogu kreirati nove procese. U tom slučaju proces koji kreira novi proces zove se roditelj a novi proces dete, pa je odnos procesa hijerarhijski (u obliku stabla).
- Između roditelja i deteta možemo imati dve vrste veze:
- 1. proces-roditelj kreira novi proces i čeka dok proces-dete završi sa radom
- 2. proces-roditelj kreira novi proces i nastavljaju sa radom oba procesa (rade paralelno)

Kako se dele OS kada se razmatra broj procesa?

- **mono tasking** (jednoprocesni, monoprogramiranje): u memoriji istovremeno imamo samo jedan program, tj. “ istovremeno” se izvršava samo jedan proces (npr. DOS)

- **multi tasking** (višeprocesni, multiprogramiranje) : u memoriji istovremeno imamo I više programa, tj. “ istovremeno” se izvršavaju i više procesa (npr. Windows,Linux)

U odnosu na broj korisnika, OS mogu biti:

- **monouser (jednokorisnički):** npr. DOS (jednokorisnič ki, jednoprocesni), Windows 98 (jednokorisnički, višeprocesni)

- **multiuser (višekorisnički):** npr. UNIX (višekorisnič ki, višeprocesni)

Šta je heap?

- **HEAP** (hrpa) je deo memorije koji je rezervisan za dinamičke promenljive, tj. za promenljive koje se stvaraju u toku izvršavanja programa.
- Pristup dinamički kreiranim promenljivama se ostvaruje pomoću pokazivača koji sadrže početnu adresu zauzete memorije.
- Ako smo zauzeli neku količinu memorije i nemamo više ni jednog pokazivača na tu memoriju, taj deo HEAP-a je izgubljen (nemamo više pristupa, ne znamo adresu zauzete memorije, ne možemo je osloboditi) – to se zove smeće u memoriji (*garbage*) i može dovesti do ozbiljnih grešaka u radu programa.

- Ovaj problem se kod nekih OS-a rešava pomoć u sakupljača smeća (garbage collector) koji vode računa o zauzetim delovima memorije: npr. broje pokazivače na zauzetu memoriju i ako taj brojač postaje nula, oslobađaju memoriju.
- Postoje i programski jezici koji imaju ugrađen sakupljač otpadaka kao što je Java.

Šta je *stack*?

- *STACK* (stek) je deo memorije koji je rezervisan za čuvanje lokalnih promenljivih, parametara procedura, povratnih adresa itd.

Razlika između konkuretnog i sekvencijalnog programiranja

- Kod SEKVENCIJALNOG programiranja, - progam se sastoji od niza naredbi koji se izvršavaju u tačno određenom redosledu od strane jednog procesora. Znači, sekvencijalni program predstavlja tačno jedan proces koji se izvršava od početka do kraja na jednom procesoru.
- KONKURENTNO programiranje uzima u obzir mogućnost postojanja i više procesora, pa se konkurentni program sastoji od više nezavisnih procesa ili zadataka (*task), koji se mogu* ***istovremeno*** izvršavati.
- Stil konkurentnog programiranja koristimo kada se zadatak može razbiti na više međusobno relativno nezavisnih delova – procesa, koji se mogu istovremeno izvršavati.
- Konkurentni programi se mogu izvršavati i na računaru sa jednim procesorom i na računaru sa više procesora.

Kako se procesi OS dele prema težini?

Podela procesa na teške i lake vrši se na osnovu toga, kako koriste delove memorije :

- **svaki teški proces** ima sopstveni memorijski prostor za kod, globalne promenljive, stek i heap koju ne deli ni sa kim, pristup tim delovima ima samo dati proces.
- **laki procesi (niti, threads)** mogu deliti memorijski prostor za kod, globalne promenljive i heap. Stek se ne može deliti jer ne možemo unapred znati koji proces koristi stek: proces A stavi nešto na stek i nastavlja rad, dolazi proces B i on isto stavi nešto na stek, A je završio svoj rad i sad su mu potrebni podaci sa steka, a tamo su podaci procesa B… Znači laki procesi imaju sopstveni stek a mogu deliti kod, globalne promenljive i heap.

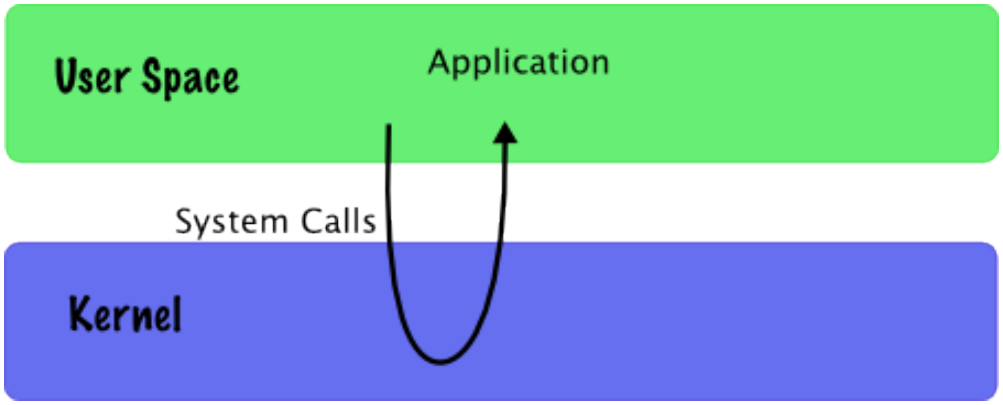
Sadržaj:

- Jezgro operativnog sistema.
- Pristup jezgru OS.
- Procedure jezgra, sistemski pozivi.
- Upravljanje procesorom.
- Upravljanje U/I uređajima.
- Vrste prekida.
- Koncept virtualnih mašina.
- Upravljanje memorijom.
- Vrste memorija.
- Tehnike za proširenje interne memorije računara.
- Koncept virtualne memorije.
- Segmentacija procesa.
- Paging tehnika.

Šta je fajl sistem (file system)?

- Programi na višim nivoima ne treba da vode računa o tome, kako su u stvari fajlovi smešteni na disku, o tome će voditi računa OS. Za rad sa fajlovima, OS treba da obezbedi operacije, kao što su:
  - otvaranje fajlova
  - zatvaranje fajlova
  - promena pozicije fajl-pokazivača (FilePos)
  - čitanje iz fajlova
  - pisanje u fajlove
  - kreiranje novih fajlova
  - brisanje postojećih fajlova
  - reimenovanje,kopiranje, itd.
- OS podržavaju i koncept direktorijuma, kao način grupisanja fajlova – pa obezbeđuju i operacije za rad sa njima.
- Elementi direktorijuma mogu biti fajlovi i drugi direktorijumi, tako dolazimo do fajl sistema u obliku stabla.

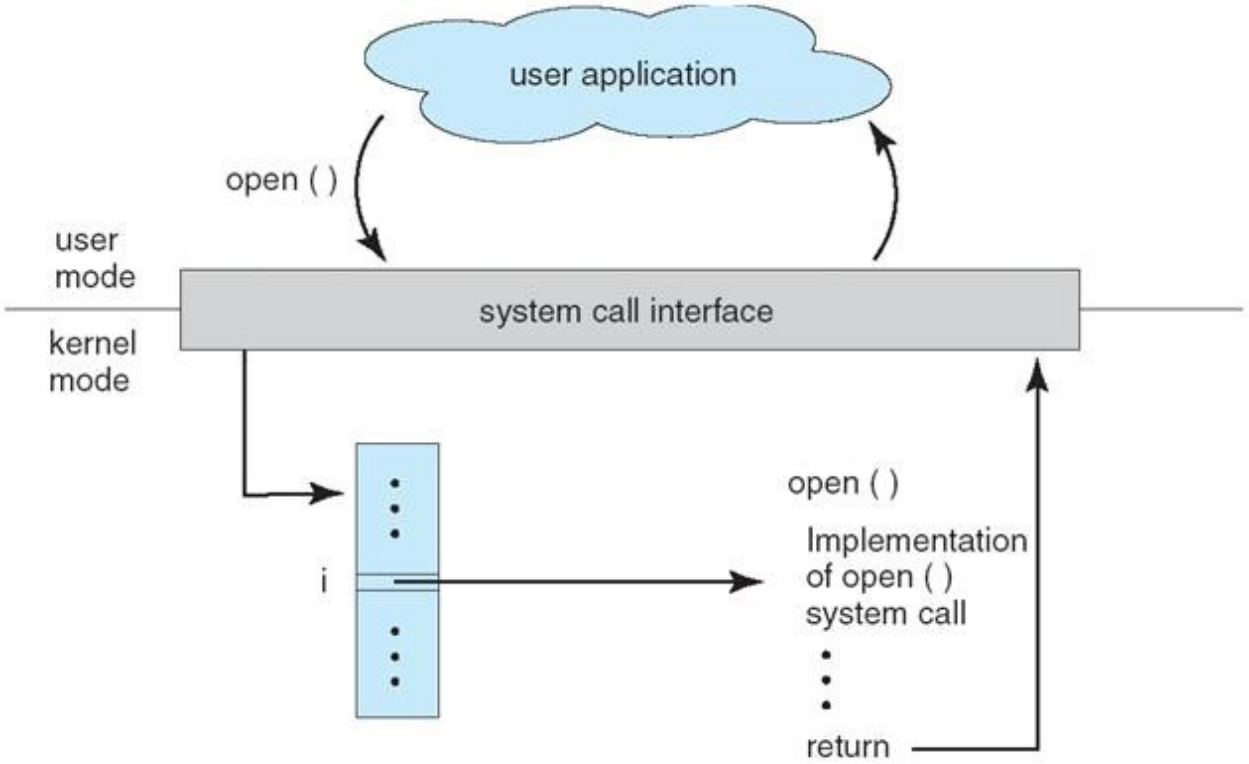
Sistemski pozivi (system calls)



- Aplikacioni programi komuniciraju sa OS-om pomoću **sistemskih poziva**, tj. preko operacija definisanih od strane OS-a.
- Sistemski pozivi se realizuju pomoću sistema prekida: korisnički program postavlja parametre sistemskog poziva na određene memorijske lokacije ili registre procesora, inicira prekid, jezgro OS preuzima kontrolu, uzima parametre, izvrši tražene radnje, rezultat stavi u određene memorijske lokacije ili u registre i vraća kontrolu programu.

Uobičajeni sistemski pozivi su:

- **Kontrola procesa**. Kraj. Prekid. Napuni. Izvedi. Kreiraj proces. Dovrši proces. Definiši atribute procesa. Špročitaj atribute procesa.Čekaj na istek vremena. Čekaj na pojavu događaja.
- **Rad sa datotekama.**Kreiraj datoteku. Izbriši datoteku. Otvori datotreku. Zatvori datoteku. Pročitaj. Upiši. Postavi pokazivač. Definiši svojstva datoteke. Pročitaj svojstva datoteke.
- **Upravljanje uređajima.** Zatraži uređaj. Oslobodi uređaj. Pročitaj. Upiši. Postavi pokazivač. Definiši svojstva. Pročitaj svojstva.



- Sistemske pozive često podržava i hardver, tj. procesor, na taj način što razlikuje dva režima rada: **korisnički režim (user mode)** i **sistemski režim (kernel mode, system mode, supervisor mode**) rada.
- Korisnički programi mogu raditi isključivo u korisničkom režimu rada procesora, sistemski režim rada je predviđen za OS. Ako korisnički program pokuša izvršiti neku operaciju koja je dozvoljena samo u sistemskom režimu rada, kontrola se predaje OS-u.
- Prilikom sistemskih poziva procesor prelazi iz korisničkog režima rada u sistemski, OS obradi poziv, pa se procesor vraća u korisnički režim rada.

Koje strategije za upravljanje uređajima postoje?

- Imamo glavni procesor, koji izvršava programe i imamo IO uređaje (disk, štampač, skener,itd.). Svaki IO uređaj ima svoj kontroler koji sadrži neki slabiji procesor koji je jedino zadužen za upravljanje tim uređajem i za komunikaciju sa glavnim procesorom.
- Postoje dve strategije za upravljanje uređajima:
- **1. polling**: u određenim vremenskim intervalima glavni procesor prekida svoj rad i proveri da li neki kontroler ima neku poruku za njega, ako ima, obradi poruku i nastavlja sa radom.
- **2. prekidi (interrupt):** glavni procesor radi svoj posao, i uređaji rade svoj posao. Ako uređaj završi svoj posao ili dođe do neke greške, uređaj o tome obaveštava glavni procesor zahtevom za prekid (*interrupt request*). Kada procesor dobije zahtev za prekid, prekida svoj rad, zapamti gde je stao, obradi prekid, pa nastavlja sa radom tamo gde je bio prekinut (ako prekid ne predstavlja neku fatalnu grešku, takvu da se rad ne može nastaviti).

Opisati algoritam prekida.

- Obrada prekida se izvršava po sledećem algoritmu:
  - procesor radi svoj zadatak
  - stiže zahtev za prekid
  - sačuva se stanje trenutnog procesa
  - onemogućavaju se dalji prekidi
  - u tabeli prekida (*interrupt table* – koja čuva adrese procedura za obradu svih mogućih prekida) traži se adresa procedure za obradu prekida
  - izvršava se procedura za obradu prekida
  - omogućavaju se dalji prekidi
  - nastavlja se rad na prekinutom procesu

Koje sve vrste prekida postoje?

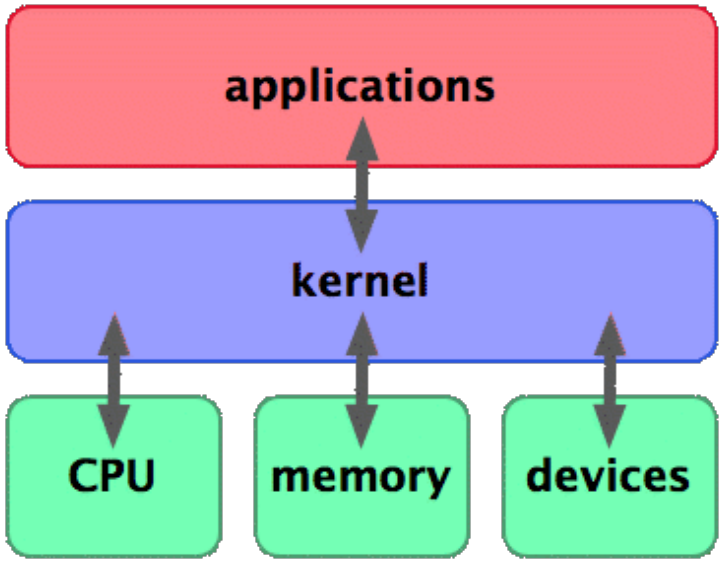
- Vrste prekida:
- 1. **Hardverski prekidi** – generišu se od strane hardvera, mogu biti:
  - prekidi koji se **mogu maskirati** – ove prekide procesor može ignorisati ako je dobio takvu naredbu (npr. pritisnuta je tipka na tastaturi)
  - prekidi koji se **ne mogu maskirati** – prekidi čija obrada ne može biti odložena – to su neke ozbiljne greške hardvera (npr. greška memorije)
- 2. **Softverski prekidi** – prekidi generisani od strane programa
- 3. **Sistemski pozivi** (jesu softverski prekidi)
- 4. **Izuzeci (exceptions)** – generišu se od strane procesora, npr. ako se pokuša deliti sa nulom

- Operativni sistemi se globalno dele na **tzv. superstrukturu** koja obuhvata grupu više aktivnosti i **jezgro (kernel)**, čiji je zadatak da obezbedi uslove u kojima se konkurentne aktivnosti mogu uporedo izvoditi.
- Superstruktura se nadalje deli na niz podsistema od kojih svaki ima poseban zadatak. Sistemski programi se mogu podeliti na sledeće grupe:
- Za manipulisanje datotekama. Ovi programi kreiraju, brišu kopiraju, preimenuju, štampaju, obavljaju,...
- Za davanje informacija o stanju OS. U toku izvođenja programa mogu da se pojave zahtevi o tačnom vremenu, količinom slobodnog prostora u radnoj memoriji i hard disku, trenutnom broju logovanih korisnika sistema.
- Za podršku izvođenja viših programskih jezika. Programi - prevodioci, asembleri, interpreteri uglavnom su standardni deo operativnih sistema. Tu još padaju tekst editori, grafički paketi, baze podataka.
- Najznačajniji sistemski program je komandni interpreter. Komadni interpreter je takođe proces koji za zvaku unesenu komandu (copy, rm, mkdir) aktivira novi odgovarajući proces.
- Komandni interpreter zatim čeka da se kreirani proces završi.

Šta je jezgro operativnog sistema?

- Sve operacije definisane na procesima kontrolisane su od dela operativnog sistema koji se naziva jezgrom (*core, nucleus, kernel*).
- Jezgro predstavlja samo manji deo operativnog sistema, ali je i najčešće u upotrebi. Stoga, jezgro se stalno nalazi u radnoj memoriji, dok se ostali delovi operativnog sistema mogu po potrebi prebacivati sa spoljašnje memorije.
- Jezgro pruža osnovnu kontrolu nad svim delovima računara:
  - upravlja pristupom RAM memoriji tokom izvršavanja programa,
  - određuje koji će program pristupiti kojim hardverskim resursima,
  - postavlja modove rada CPU za optimalan rad u svakom trenutku,
  - i organizuje trajno skladištenje podataka preko sistema datoteka

- **Jezgro OS-a (kernel, core)** obavlja najbitnije operacije:
  - upravljanje prekidima
  - kreiranje i uništavanje procesa
  - odabiranje procesa iz liste spremnih procesa (context switch)
  - suspenzija i nastavljanje procesa
  - sinhronizacija procesa
  - komunikacija između procesa
  - manipulacije sa *PCB-om*
  - podrška za ulaz/izlaz (IO)
  - upravljanje memorijom
  - upravljanje fajl sistemom, itd.



Koji slojevi OS postoje?

5.	komandni interpreter
4.	korisnički programi
3.	ulazne,izlazne operacije
2.	procesi
1.	upravljanje memorijom
0.	procesor i multiprogramming

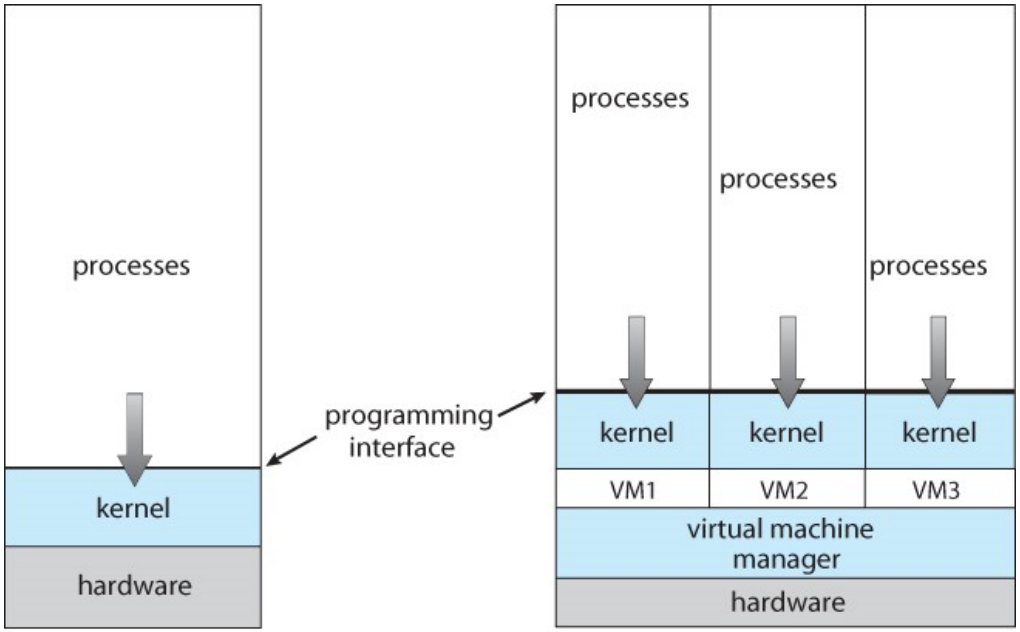
- Nulti sloj se bavi upravljanjem procesora, obezbeđuje prebacivanje između različitih procesa.
- Prvi sloj upravlja memorijom: zauzima potrebnu memoriju za procese.
- Slojevi iznad prvog sloja **ne treba da brinu** o memorijskim potrebama, to će umesto njih uraditi prvi sloj.
- Drugi sloj upravlja komunikacijom između različitih procesa i komandnog interpretatora.

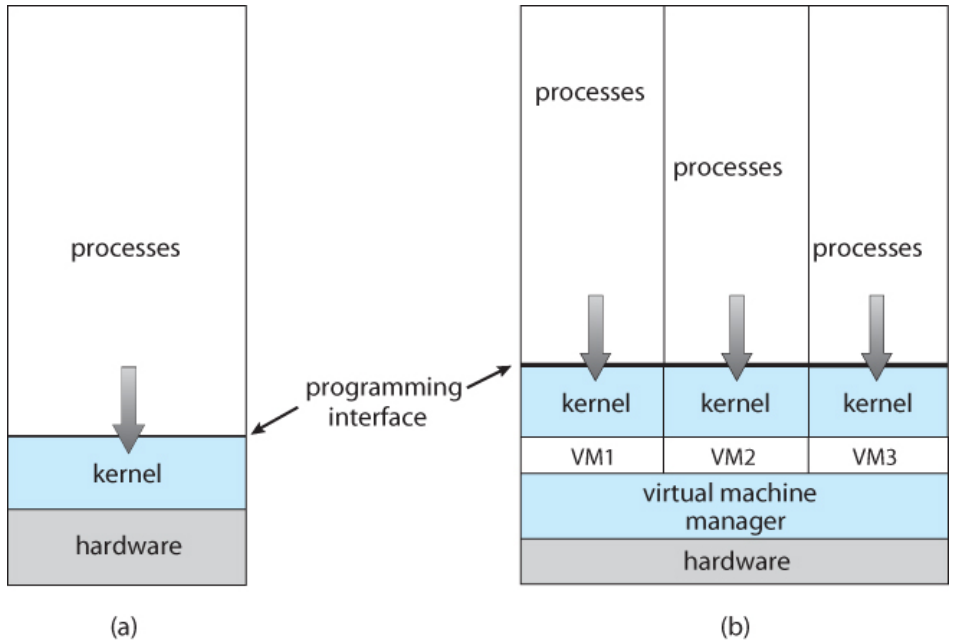
5.	komandni interpreter
4.	korisnički programi
3.	ulazne,izlazne operacije
2.	procesi
1.	upravljanje memorijom
0.	procesor i multiprogramming

- Treći sloj obavlja ulazno/izlazne operacije.
- Slojevi od 0 do 3 predstavljaju jezgro OS-a i rade u sistemskom režimu rada.
- Na četvrtom nivou imamo korisničke programe – oni ne treba da se brinu oko procesa, memorije, komandnog interpretera, IO operacija, sve to obavljaju slojevi ispod.

Šta su virtuelne mašine?

- Koncept **virtualnih mašina** obezbeđuje kao da postoji više **nezavisnih kopija hardvera**, na kojima su instalirani i istovremeno rade različiti OS, iako u stavrnosti, svi ti operativni sistemi rade na istom hardveru.
- Svaki OS smatra da ima pristupa i da upravlja radom CPU-a, RAM-a, I/O uređaja, hard diskova, itd.





- Na najnižem nivou postoji hardver, iznad hardvera je sistem koji se zove monitor virtuelnih mašina (*virtual machine monitor*) i koji obezbeđuje niz virtuelnih mašina - to nisu proširene mašine kao operativni sistemi koji pružaju niz operacija za pristup hardveru, već tačne kopije hardvera ispod monitora virtuelnih mašina.
- Zatim se na te virtuelne mašine mogu biti instalirani operativni sistemi – koji mogu biti i različiti.
- Sistemske pozive korisničkih programa primaju operativni sistemi, a hardverske operacije koje šalju ti OS-ovi prema svojim virtuelnim mašinama hvata monitor virtuelnih mašina i realizuje ih u skladu sa hardverom ispod sebe.

Koje sve vrste memorija postoje, gledano sa strane OS?

- Možemo razlikovati 3 vrste memorija :
  - 1. eksternu memoriju (najvećeg kapaciteta, najsporija, najjeftinija)
  - 2. internu memoriju (manjeg kapaciteta, brža, skuplja)
  - 3. keš (cache) memoriju (najmanjeg kapaciteta, najbrža, najskuplja)

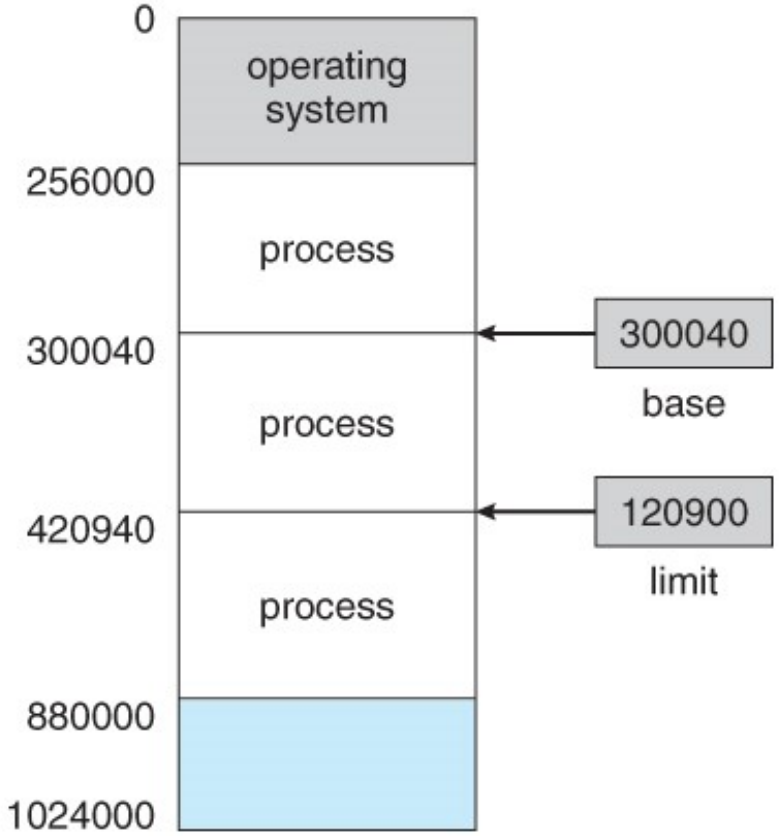
- Nas će najviše interesovati **interna memorija** (ili **operativna memorija**) i njen odnos sa eksternom memorijom jer procesor može izvršavati samo procese koji su u internoj memoriji.

- Kako je operativna memorija relativno malog kapaciteta, dolazimo do problema da nemamo dovoljno memorije da učitamo sve READY procese.

Koje zadatke izvršava upravljač memorije?

- Deo operativnog sistema koji upravlja memorijom zove se **upravljač memorije** (**memory manager**).
- Njegov zadatak je da vodi računa o tome,
  - koji delovi memorije su zauzeti,
  - da zauzme potrebnu količinu memorije za nove procese, odnosno da oslobodi memoriju zauzetu od strane nekog procesa,
  - da upravlja prebacivanjem procesa iz interne u eksternu memoriju i obrnuto, pri tome mora voditi računa da procesi ne štete jedni drugima, a ni operativnom sistemu.

- Korisnički procesi moraju se ograničiti da koriste memoriju koja pripada samo tom procesu.
- To se obično radi korišćenjem ***base*** **i** ***limit*** registara.
- Svaki pristup memoriji, koji napravi korisnički proces, proverava se preko ***base*** **i** ***limit*** registara, ako se pristupa nekoj lokaciji van opsega, prijavljuje se fatalna greška.



Koje se tehnike koriste za proširenje interne memorije?

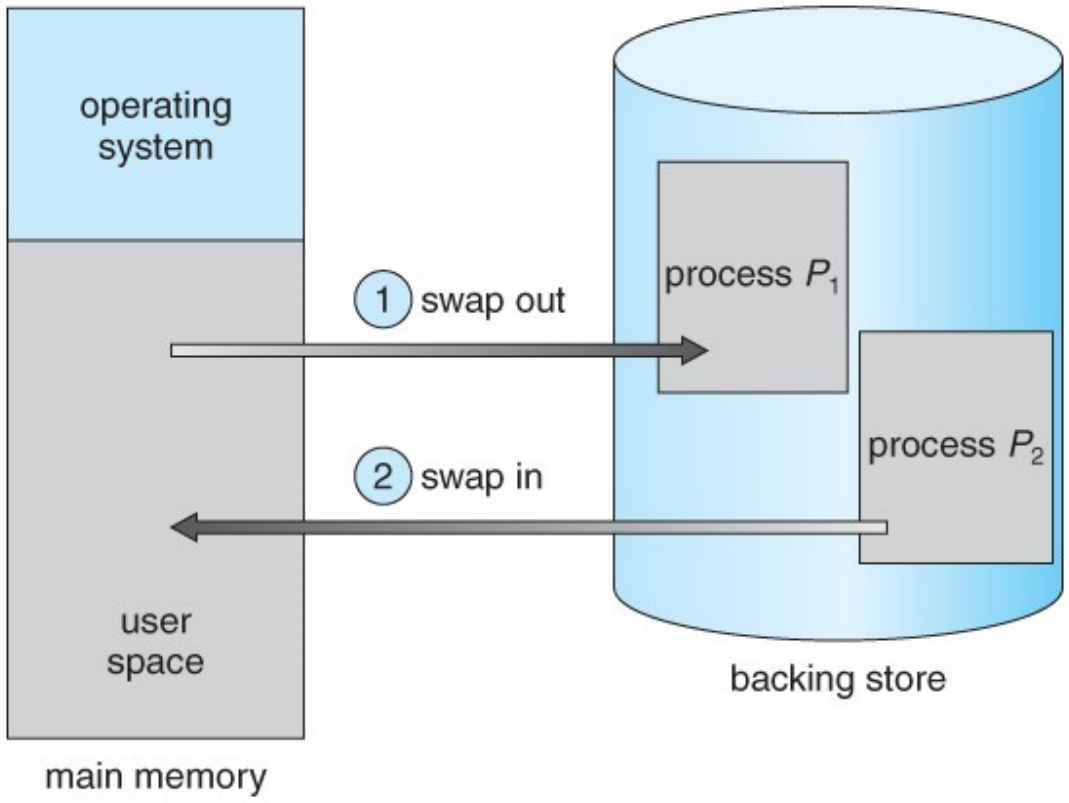
- U slučaju kada nemamo dovoljno interne memorije za nove procese, koriste se sledeće tehnike:

- 1. SWAPPING (prebacivanje procesa) – ako nemamo dovoljno mesta u operativnoj memoriji za smeštanje *Ready* procesa, neki se prebacuju na disk. Kada je to potrebno, procesi se iz interne memorije prebacuju na disk, odnosno procesi sa diska se prebacuju u internu memoriju.

- 2. Tehnika *Overlay*

- 3. PAGING (straničenje) – delove procesa držimo na disku, učitavaju se po potrebi.

- Proces da bi se izvršavao potrebno je da se nalazi u internoj memoriji.
- Ako nema dovoljno prostora da se svi procesi izvršavaju u isto vreme, tada se procesi koji trenutno ne koriste CPU, prebacuju (svapuju) na lokalni hard disk.
- Svapovanje je spor proces.



Objasniti tehniku Overlay.

- U slučaju, da je program veći od operativne memorije, kao rešenje možemo koristiti tehniku koja se zove overlay.
- Program delimo na nezavisne delove. Ti delovi će formirati pakete koji se učitavaju po potrebi za vreme izvršavanja, sa diska u memoriju.
- Za pakete *overlay*-a rezervišemo poseban deo memorije. Inicijalno se zauzima onoliko memorije koliko je potrebno za najveći paket (pored mesta za one delove programa koji stalno moraju biti u memoriji).
- Zatim, u toku rada programa, kada pozovemo neku proceduru iz nekog paketa koji se nalazi na disku, novi paket biće prebačen sa diska u internu memoriju umesto trenutnog paketa.

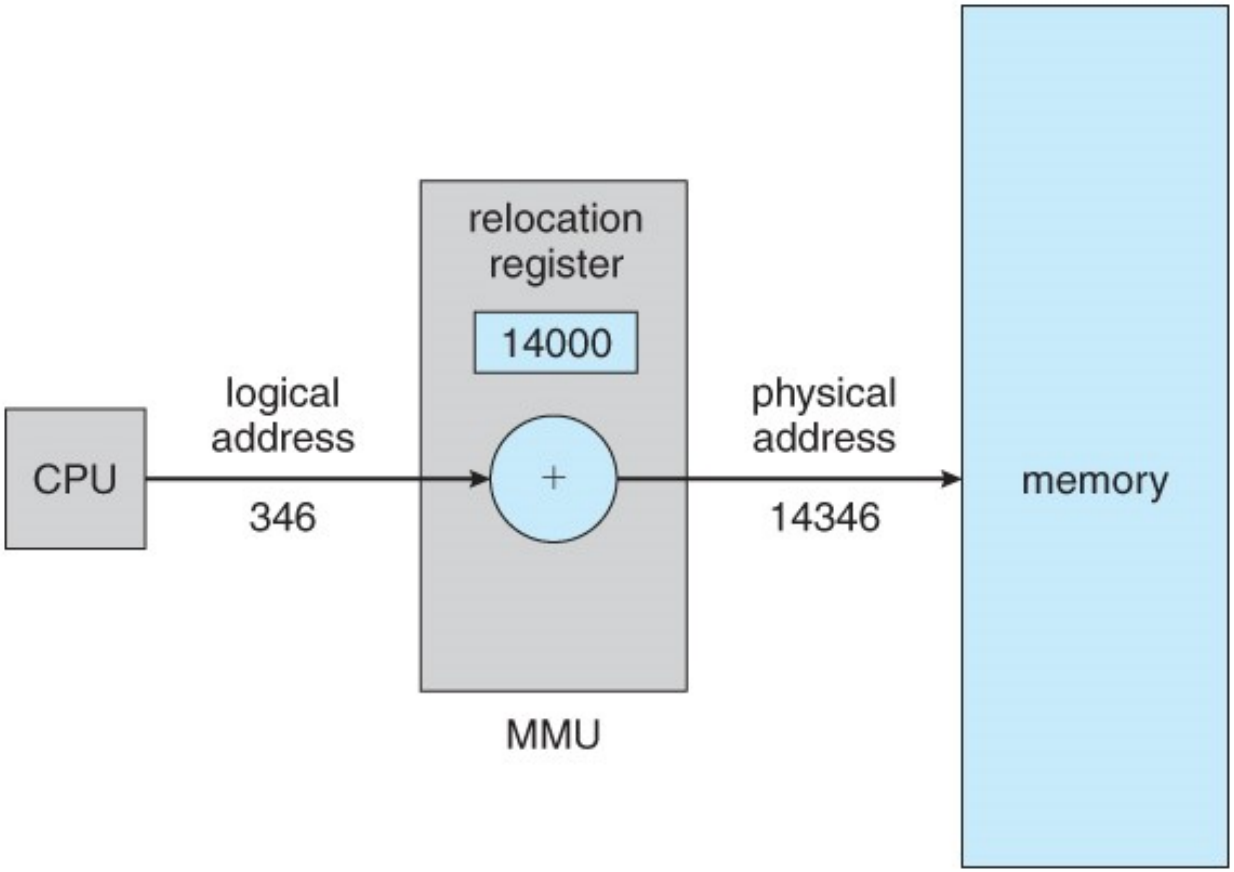
- Razbijanje programa na manje, međusobno nezavisne delove i prebacivanje raznih *overlay* paketa sa diska u memoriju su zadaci OS-a.
- Osnovna ideja je da pustimo lepo programera neka napiše program veličine koje želi, i onda će OS voditi računa o tome da u memoriji budu samo oni delovi procesa (i kod i podaci) koji su u datom trenutku potrebni, a ostatak neka čeka na disku.
- Znači jedan deo procesa je u internoj memoriji a drugi deo je u eksternoj memoriji, a sam proces nema pojma o tome – njemu se čini da ima dovoljno operativne memorije i da se čitav proces nalazi u internoj.
- Drugačije rečeno: ako nema dovoljno memorije, uzećemo malo od diska – procesi neće znati za to, a možda i korisnik ukoliko ne primeti usporavanje rada programa.

Objasniti koncept virtulne memorije.

- Tehnika Paging (straničenje) se koristi kao tehnika za realizaciju **virtuelne memorije**. Kada nemamo dovoljno interne memorije da bismo učitali ceo proces, koristimo virtuelnu memoriju – koja je veća od fizičke interne memorije.
- Proces nema pojma o tome da se izvršava u virtuelnoj memoriji, ubeđen je da je u internoj memoriji – i ponaša se u skladu sa time: pristupa memorijskim lokacijama koje fizički možda ne postoje.
- Zato je za realizaciju virtuelne memorije potrebna pomoć od strane hardvera. Memorijska adresa generisana od strane programa zove se **virtualna adresa** iz virtualnog adresnog prostora.
- Kod računara bez virtuelne memorije, adresa generisana od strane procesa ide direktno od procesora ka memoriji.

- Adresa koja je generisana od strane CPU je logička (ili virtuelna) adresa, dok, adresa koju memorijski hardver vidi je fizička adresa**.** Mapiranje adresa se obavlja u ***Memory-management unit, MMU***

- Ako koristimo virtuelnu memoriju, virtuelna memorija ide najpre do dela procesora koji se zove **jedinica za upravljanje memorijom** (**MMU – Memory Managment Unit**), koja virtuelnu adresu pretvara u fizičku memorijsku adresu.



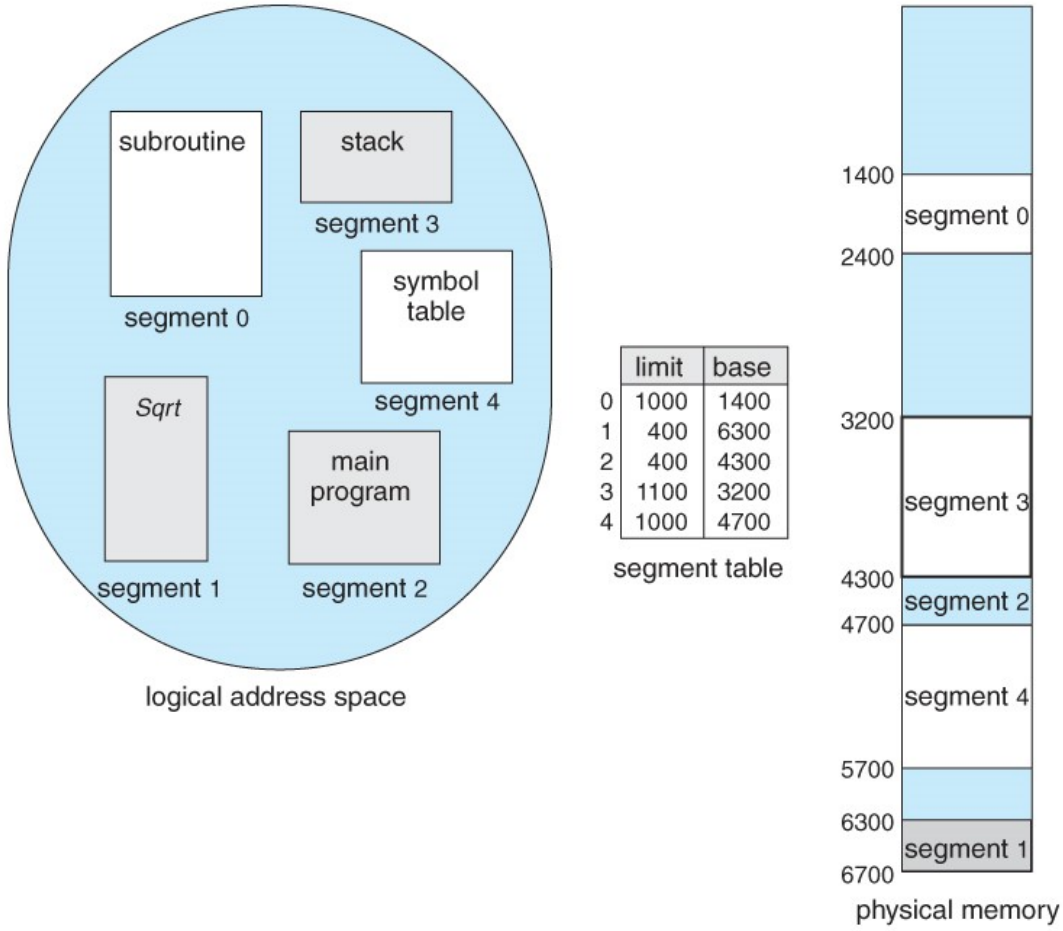
Šta je to segmentacija procesa?

- Svakom procesu se dodeljuje **više linearnih adresnih prostora** (koji počinju od nule i idu do nekog maksimuma).
- Ti prostori ili delovi procesa se nazivaju segmentima. Svaki segment predstavlja neku linearnu memoriju.
- Segmenti predstavljaju funkcionalne delove programa kao što su: kod programa, stek, *heap*, ali segment može da sadrži i samo jednu proceduru, niz ili nešto drugo.
- Veličina svakog segmenta može se menjati za vreme izvršavanja od nule do nekog maksimuma. Segmenti mogu biti različitih veličina. Pošto svaki segment ima svoj adresni prostor (koji je linearan), različiti segmenti nemaju uticaja jedan na drugi.
- Sa uvođenjem segmentacije, adresiranje zahteva dve adrese: jednu **segmentnu adresu** koja identifikuje segment i jednu **ofset adresu** koja daje poziciju lokacije unutar tog segmenta. Znači adrese su oblika (**segment, ofset**).

- Većina korisnika (programera) ne razmišlja o svojim programima kao da su realizovani u kontinualnom linearnom memorijskom prostoru. Recimo, razmišlja o memoriji kao o skupu segmenata koji se koriste na primer, za realizaciju programskog koda, podataka, steka, heap-a, itd.

- Segmentacija memorije omogućava upravo ovo.

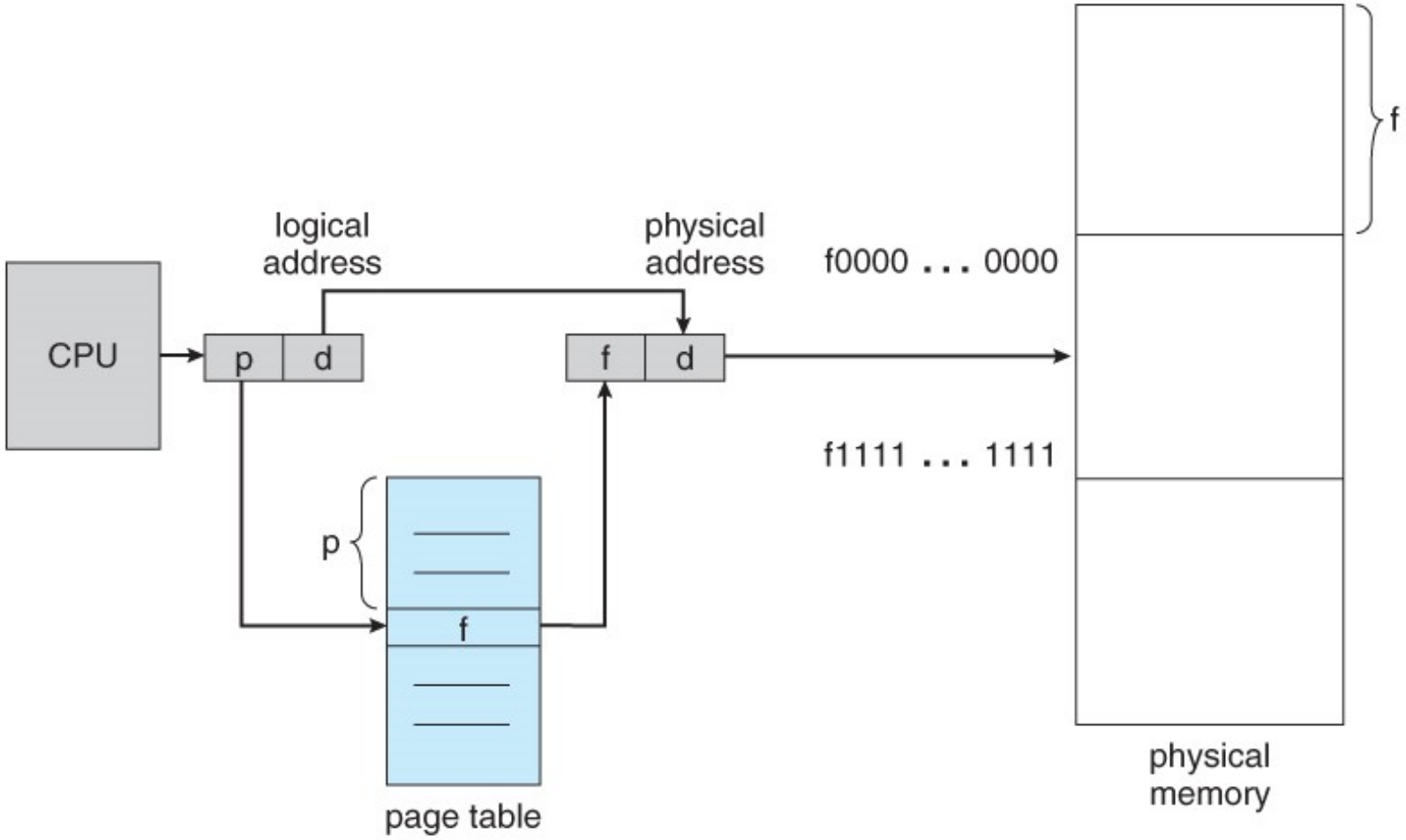
- Adresa se sastoji od **adrese segmenta** (preslikava se u baznu adresu) i **ofseta** (koja broji poziciju od početka tog segmenta).



Objasniti tehniku Paging (straničenje).

- Virtuelni adresni prostor se deli na blokove iste veličine (**virtuelne stranice *- virtual pages*).**
- Isto tako, interna memorija se deli na blokove (**okvire za stranice - *page frames*). Pritom,** virtuelne stranice i okviri za stranice su iste veličine.
- Računanje fizičke adrese na osnovu virtuelne adrese se radi na osnovu **Tabele stranica (Page table)**
- **Tabele stranica preslikava virtuelnu stranicu u okvir stranice.**
- Ova tabela se dodeljuje svakom procesu i menja se u toku izvršavanja procesa: ako proces želi pristupiti nekoj adresi koja se nalazi na virtuelnoj stranici, koja nije učitana u internu memoriju, moramo izbaciti neki okvir za stranice (*page frame*) iz operativne memorije (ako je cela operativna memorija već zauzeta) na eksternu memoriju, a iz eksterne memorije moramo dovući traženu stranicu na mesto izbačenog okvira za stranice.

- Osnovna ideja iza tehnike PAGING jeste podela fizičke memorije u **okvire za stranice** – (*page frames*) koji su jednake veličine i programske virtuelne memorije u blokove koje se nazivaju **virtuelne stranice**.
- Svaka virtualna stranica (iz bilo kog procesa) može se postaviti u bilo koji slobodan okvir za stranice.

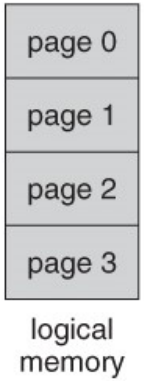


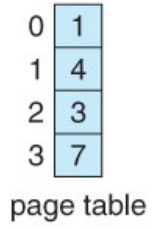
Logička (virtualna) adresa sastoji se od dva dela:

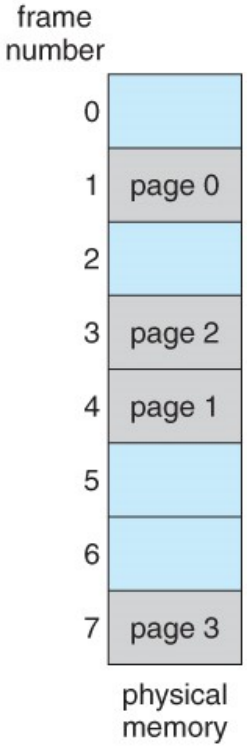
- **adrese virtuelne stranice** (***page number***) i
- **ofseta** (***page offset*** – pozicija lokacije od početka stranice).

Tabela stranica (***Page table***) preslikava ***Page number*** u ***Frame number***, da bi se dobila **fizička adresa**, koja se sastoji iz dva dela:

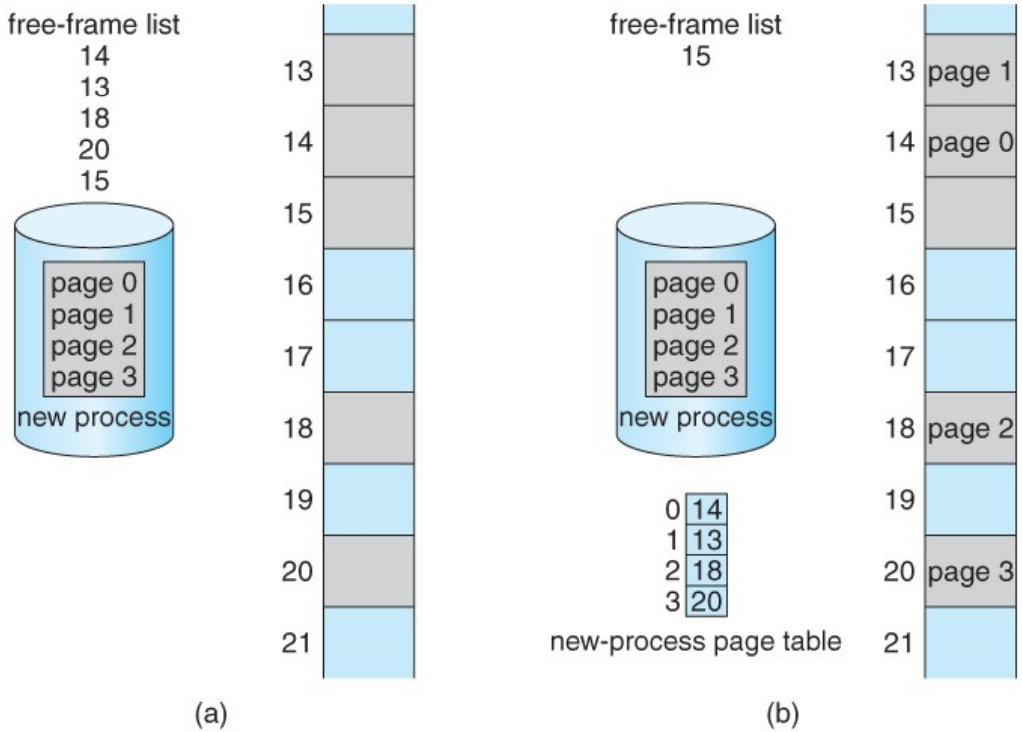
- **adrese okvira (*frame number*) i**
- **ofseta** (od početka unutar tog okvira)





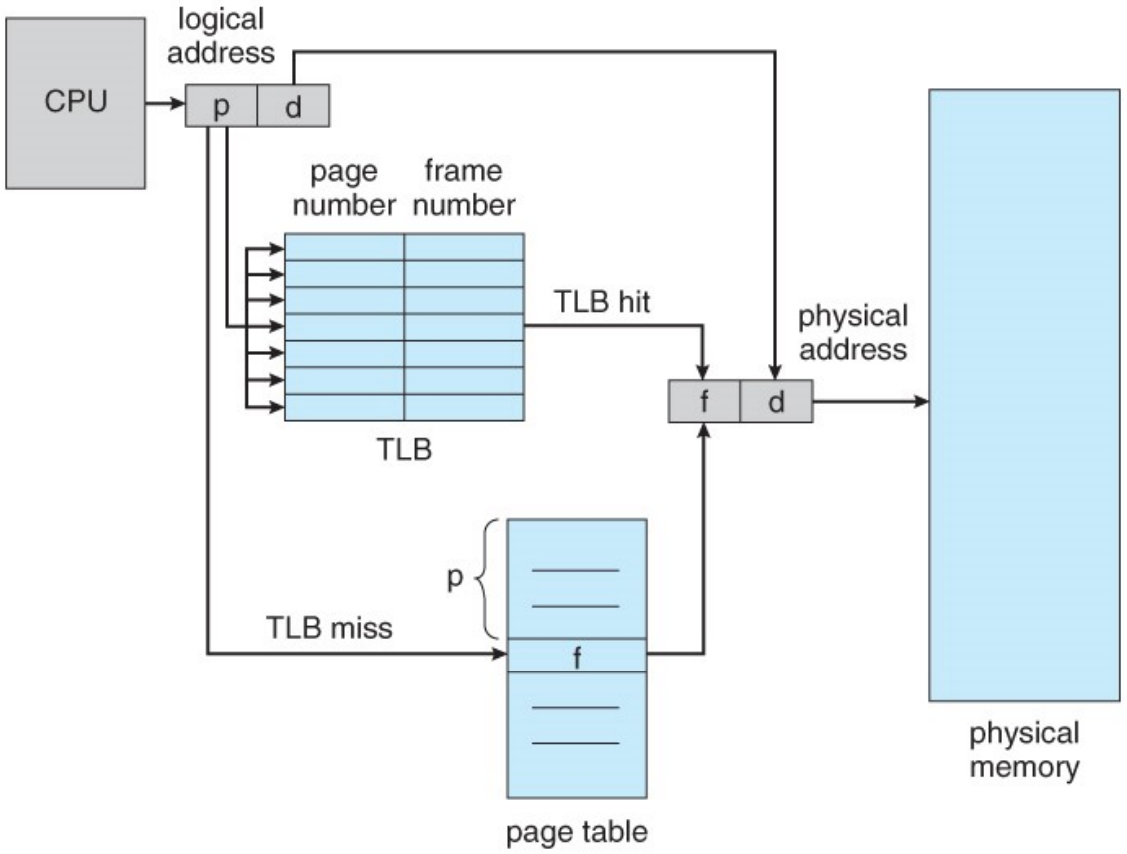


- Kada neki proces zahteva memoriju (na primer, kada se njegov kod loaduje sa diska), rezervišu se slobodni okviri za stranice iz **liste slobodnih okvira** (***free frame list***) i ubacuju u Tabelu stranica (***Page table***) tog procesa.
- Procesima je zabranjen pristup memoriji koja je rezervisana za neki drugi proces, jer se svi memorijski zahtevi procesa proveravaju kroz **Tabelu stranica** (***page table***) tog procesa. Ne postoji način da se za neki proces generiše adresa koja je deo memorijskog prostora nekog drugog procesa.



- Operativni sistem prati stanje ***Tabele stranica*** za svaki pojedinačni proces i ažurira sadržaj***Tabele*** kada se proces seli u ili van interne memorije.
- Tako OS koristi uvek ispravnu *Tabelu stranica* kada se izvršava sistemski poziv (*system call*) nekog procesa.

- Kada se proces se prebacuje u ili izvan interne memorije, njegova Tabela stranica mora da bude takođe pronađena u internoj memoriji.
- Potrebno da postoji poseban hardver za operaciju pronalaženja Tabele stranica kako da se to obavilo što je moguće brže, i da bi na kraju, prebacivanje procesa sa hard diska u internu memoriju trajalo što kraće.
- Rešenje ovog problema je da se koristi posebna brza memorija koja se zove ***Translation look-aside buffer***, **TLB**, koja sadrži listu adresa okvira za stranice (*Frame number*) za odgovarajuće adrese virtualnih stranica (*Page number*).
- Adresa virtuelne stranice (*page number*) se najpre traži u brzoj TLB memoriji. Paralelno se pretražuje čitava tabela, da bi se brzo odredila adresa okvira za stranice (*frame number*) nekog procesa.



- Adresa virtualne stranice (*Page number*) se prvo pronalazi u TLB-u, ali ako adresa nije tamo, (*TLB miss*), ona se traži u internoj memoriji, u kojoj se nalazi Tabela stranice procesa (*Page table*). Nakon pronalaska okvira (*Frame number*) sadržaj TLB memorije se updejtuje pronađenim parom (*Page number*, *Frame number*).

- Procenat uspeha nalaženja adrese virtuelne stranice u tabeli TLB predstavlja ***hit ratio***.