1 - Pregled OS

OPERATIVNI SISTEM je program koji:

1. Upravlja izvršavanjem aplikacionih programa
2. Služi kao interfejs između programa i hardvera računara

CILJEVI OS

OPERATIVNI SISTEM treba da obezbedi sledeće:

1. **POGODNOST** – *da računar bude korisniku lakši za korišćenje*
2. **EFIKASNOST** – *da se resursi računarskog sistema koriste na efikasan način*
3. **PROŠIRIVOST** – *mogućnost izvršavanja novih korisničkih aplikacija tj. Dodavanje novih funkcionalnosti sistema nezavisno od ugrađenih osnovnih servisa*

OS KAO INTERFEJS IZMEĐU KORISNIKA I RAČUNARA

Krajnji korisnik vidi računarski sistem kao skup aplikacija. Aplikacija može biti implementirana kao skup mašinskih instrukcija koji je u potpunosti odgovoran za upravljanje hardverom računara.

Kako je to vrlo složeno za implementaciju, obezbeđuje se skup sistemskih programa koji upravljaju hardverom. U tom slučaju, aplikacija se obraća sistemskom programu da bi obavila određene funkcije. Najvažniji sistemski program je **OPERATIVNI SISTEM**.

Postoje 2 osnovna načina interakcije korisnika sa OS:

- Komandna linija (CLI)
- Grafički interfejs (GUI)

KOMANDNA LINIJA KAO VEZA SA OS

Interakcija se vrši unosom tekstualnih komandi. Interpreter komandi je ugrađen u kernel ili je poseban program u okviru OS (*shell*).

Postoje 2 načina implementacije komandi:

1. Interpreter komandi sadrži u sebi kod za izvršenje komande
2. Komande se izvršavaju pozivom sistemskih programa (*Češći slučaj. Komandna linija ne razume komande već služi samo za identifikaciju fajla koji će biti učitan i izvršen.* **Primer:** ls folder startuje program ls I prosleđuje folder kao parametar)

GUI KAO VEZA SA OS

Korisnik korišćenjem miša ili prsta manipuliše grafičkim elementima koji predstavljaju programe, foldere, fajlove i sistemske funkcije.

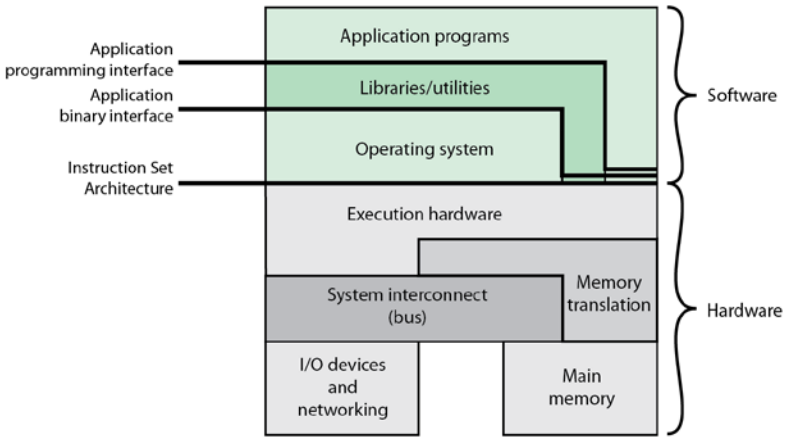


Figure 2.1 Computer Hardware and Software Infrastructure

Infrastruktura hardvera I softvera

PROGRAMSKI INTERFEJSI U OS

1. **API – Interfejs aplikativnog programiranja**
   Specifikacija servisa koje aplikacija može da poziva
   Primer: *Funkcije u sistemskoj biblioteci*
2. **ABI – Binarni interfejs aplikacije**
   Sličan API-ju, ali nižeg nivoa – predstavlja vezu sa OS jer definiše format komunikacije sa OS
   (Sistemski pozivi, načini poziva funkcija, predstavljanja tipova podataka...)
3. **ISA – Arhitektura skupa instrukcija**
   Repertoar instrukcija mašinskog jezika koje hardver podržava

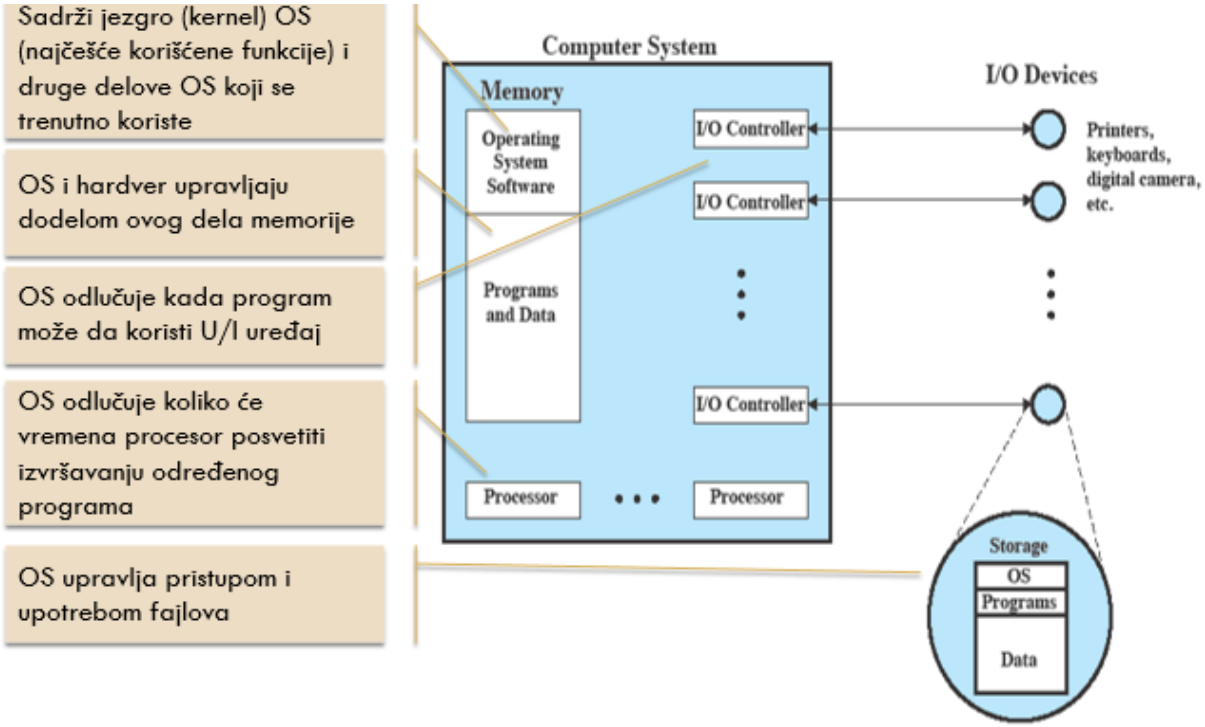
STANDARDNE USLUGE OS

1. **IZVRŠAVANJE PROGRAMA** – OS upravlja resursima koje program koristi (učitavanje instrukcija i podataka u glavnu memoriju, inicijalizacija U/I uređaja...)
2. **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.
3. **PRISTUP FAJLOVIMA** – OS vodi računa o internoj strukturi uređaja za skladištenje podataka, načinu skladištenja podataka, i obezbeđuje mehanizme kontrole pristupa fajlovima.
4. **DELJENI PRISTUP SISTEMU** – OS obezbeđuje zaštitu resursa od neovlašćenog pristupa i rešava konflikte u takmičenju za resurse
5. **UPRAVLJANJE GREŠKAMA** – OS obezbeđuje reakciju na hardverske i softverske greške
6. **NADZOR SISTEMA** – OS prikuplja statistiku upotrebe resursa i nadgleda performanse

OS KAO UPRAVLJAČ RESURSA

Računar je skup resursa za prenos, skladištenje i obradu podataka. OS je odgovoran i za upravljanje ovim resursima. OS funkcioniše na isti način kao i ostali softver – on je program ili skup programa koje izvršava procesor. OS se često odriče upravljanja i zavisi od procesora koji će mu dozvoliti da ponovo preuzme upravljanje.

Razlika u odnosu na drugi softver je **NAMENA** – OS sadrži instrukcije čijim se izvršavanjem upravlja resursima sistema (*raspoređivanje, zauzimanje memorije itd.*)



ISTORIJA RAZVOJA OS

1. **SERIJSKA OBRADA**

   U početku razvoja računari nisu imali OS već su korisnici direktno pristupali hardveru. Samo jedan korisnik je u jednom trenutku imao pristup računaru.

   Problemi:

   - Korisnici moraju da rezervišu računare unapred što otežava organizaciju
   - Značajan deo vremena se trošio na postavljanje programa da se izvrši
   - Direktno se upravlja hardverskim resursima, što nije optimalno
   - Nema istovremenog izvršavanja više programa

2. JEDNOSTAVNI SISTEMI PAKETNE OBRADE

Korak ka efikasnijem korišćenju računara. Korisnik više ne pristupa računaru direktno već predaje poslove operatoru koji ih kao skup programa postavlja na računar. Pripremljen skup programa izvršava poseban softver – **MONITOR**.

MONITOR je program sa posebnom namenom da upravlja izvršenjem drugih programa. On izvršava jedan po jedan program, a svaki program je kreiran tako da se vrati na monitor kada se završi. Nakon toga, monitor automatski učitava naredni program.

Funkcije monitora:

- Kontroliše sekvencu programa
- Učitava korisnički program I predaje mu kontrolu *(procesor prelazi da izvršava instrukcije u delu memorije gde je smešten korisnički program*)
- Korisnički program vraća kontrolu monitoru (*procesor narednu instrukciju izvršava iz programa monitora*)
- Poseban deo memorije je namenjen smeštanju programa monitora

Funkcije procesora:

- Procesor izvršava instrukcije iz dela memorije u kojoj je smešten monitor
- Kada monitor učita korisnički program, procesor će izvršiti instrukciju grananja koja određuje da procesor nastavi izvršavanje instrukcija sa početka korisničkog programa
- Procesor izvršava instrukcije u korisničkom programu dok ne završi sve ili dok se ne pojavi greška

Hardverska podrška za monitor:

- ZAŠTITA MEMORIJE – korisnički program ne sme da pristupa memoriji u kojoj je monitor. Hardver procesora mora da detektuje I spreči takve instrukcije.
- VREMENSKI BROJAČ – Zaštita da jedan program ne koristi predugo, nakon isteka brojača program se zaustavlja I kontrola se predaje monitoru
- PRIVILEGOVANE INSTRUKCIJE – Deo instrukcija može da izvrši samo monitor
- PREKIDI – Daju više fleksibilnosti jer omogućuju da se upravljanje predaje I oduzima programima

Režimi izvršavanja

Kako bi se sproveli koncepti zaštite memorije I privilegovanih instrukcija, u OS su uvedena 2 režima izvršavanja:

KORISNIČKI REŽIM – u ovom režimu se izvršavaju korisnički programi, nije moguće pristupiti zaštićenoj memoriji niti izvršiti privilegovane instrukcije

REŽIM KERNELA – monitor se izvršava u ovom režimu, može da pristupa zaštićenoj memoriji I da izvršava privilegovane instrukcije

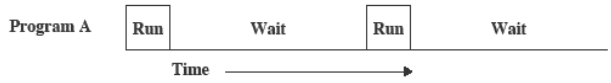
3. MULTIPROGRAMIRANI SISTEMI PAKETNE OBRADE

Ideja je da se programi više ne delegiraju monitoru sekvencijalno. Istovremeno je više programa postavljeno za izvršavanje ali i dalje u jednom trenutku procesor može da izvršava jedan program.

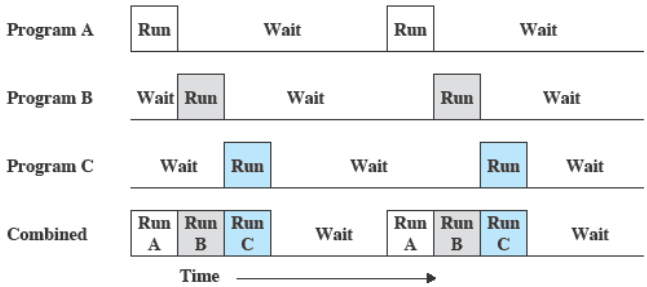
Kada je procesor posvećen samo jednom programu, najveći deo vremena je besposlen. Dok traje izvršavanje U/I operacije, procesor je slobodan. U/I uređaji su puno sporiji od procesora.

Primer: Ako čitanje i pisanje reda u fajlu traje po 15 mikrosekundi, a izvršavanje 100 instrukcija traje 1 mikrosekundu, imamo iskorišćenje procesora od 3.2%.

JEDNOPROGRAMIRANJE - Procesor izvršava instrukcije dok ne dođe do U/I instrukcije, a onda mora da sačeka završetak U/I operacije da bi nastavio rad.



MULTIPROGRAMIRANJE - Višeprocesna obrada podataka. Pored OS, u memoriju se smešta više korisničkih programa. Kada jedan program čeka na završetak U/I operacije, procesor može da pređe da izvršava drugi program. Iskorišćenost resursa se uveliko poboljšava na ovaj način.



HARDVERSKA PODRŠKA ZA MULTIPROGRAMIRANJE

- U/I Prekidi
- DMA - Direktan pristup memoriji

Omogućavaju da procesor izda naredbu za U/I operaciju i nastavi da izvršava drugi posao dok U/I operaciju vrši kontroler uređaja. Po završetku U/I operacije, kontroler uređaja postavlja prekid i kontrola se predaje programu za obradu prekida u OS. OS će predati kontrolu drugom poslu.

OS PODRŠKA ZA MULTIPROGRAMIRANJE

Da bi se realizovalo multiprogramiranje, OS treba da omogući:

UPRAVLJANJE MEMORIJOM - Poslovi koji se izvršavaju trebaju biti u memoriji istovremeno

RASPOREĐIVANJE POSLOVA - Ako je više poslova spremno za izvršavanje, potreban je algoritam raspoređivanja koji odlučuje koji posao se sledeći izvršava i koliko dugo

4. **SISTEMI SA DELJENJEM VREMENA**

Razvijeni '60ih godina za podršku višekorisnčkom pristupu računarskim resursima.

DELJENJE VREMENA - Procesor se deli između više korisnika. Korisnici istovremeno pristupaju sistemu preko terminala, a OS prepliće izvršavanje korisničkih programa. Pošto je vreme ljudske reakcije relativno sporo, vreme odziva je blisko onom u jednokorisnčkom režimu.

	Multiprogramirana paketna obrada	Deljenje vremena
Šta obezbeđuje	Multiprogramiranje	Višekorisnički rad
Glavni cilj	Što veća iskorišćenost procesora	Smanjenje vremena odgovora
Izvor naredbi za OS	Naredbe jezika za upravljanje poslovima dobijaju se uz posao	Naredbe se unose na terminalu

REALIZACIJA SISTEMA SA DELJENJEM VREMENA

- **CTSS** - Učitava i izvršava jedan po jedan korisnički program
- **Time slice** - Sistemski generator takta pravi prekid na svake 0.2 sekunde, a pri svakom prekidu OS preuzima upravljanje i može da dodeli procesor drugom korisniku

5. **SAVREMENI OPERATIVNI SISTEMI**

Deljenje vremena i multiprogramiranje su uveli standardne zahteve za dizajn savremenih OS. Multiprogramiranje je uvelo zahtev za deljenje resursa, konkurentni pristup, takmičenje za resurse, zaštita od štetnog preplitanja. Deljenje vremena je uvelo zahtev višekorisničkog pristupa resursima, zaštita i raspoređivanje. Današnji OS su dizajnirani da ispune sve ove zahteve.

Savremeni OS: GNU/Linux, Windows, Mac OS X, BSD Unix, Solaris, Android, iOS

IMPLEMENTACIJA OS

Prvi operativni sistemi pisani su u asembleru. Danas je većina pisana u jezicima visokog nivoa, mada različiti delovi mogu biti u jezicima različitog nivoa apstrakcije (*niži nivoi kernela u jezicima nižeg nivoa, sistemski programi u jezicima višeg nivoa*). Na primer, MS-DOS je pisan u Intel 8088 asembleru, Linux najvećim delom u C (*niži nivoi u asembleru, visi u C++, Lisp, Perl i Python*) dok je Windows implementiran u C-u ali se oslanja na koncepte objektno-orijentisanog dizajna.

Osnovna dostignuća na koja se oslanjaju savremeni OS:

- **PROCESI**
- **UPRAVLJANJE MEMORIJOM**
- **ZAŠTITA INFORMACIJA I BEZBEDNOST**
- **RASPOREĐIVANJE I UPRAVLJANJE RESURSIMA**

PROCESI

POJAM PROCESA

- Program u izvršavanju
- Primerak programa koji se izvršava na računaru
- Entitet koji se može dodeliti I izvršavati na procesoru
- Jedinica aktivnosti koju karakterišu naredbe za izvršavanje, tekuće stanje I dodeljeni skup sistemskih resursa

KOMPONENTE PROCESA

- **IZVRŠNI PROGRAM** - *Instrukcije koje procesor treba da izvrši*
- **PODACI** - *Pridruženi podaci koji su potrebni programu pri izvršavanju*
- **KONTEKST IZVRŠENJA - STANJE PROCESA**
  - *Kompletno stanje procesa u bilo kom trenutku*
  - *Interni podaci pomoću kojih OS može da upravlja procesom*
  - *Sadrži prioritet procesa, da li proces čeka na U/I, sačuvan sadržaj registara procesa (programski brojač, registri podataka) da bi proces mogao da nastavi izvršavanje nakon prekida*

UPRAVLJANJE PROCESOM

Lista procesa sadrži jedan zapis za svaki proces koji sadrži pokazivač na lokaciju bloka memorije koji sadrži taj proces

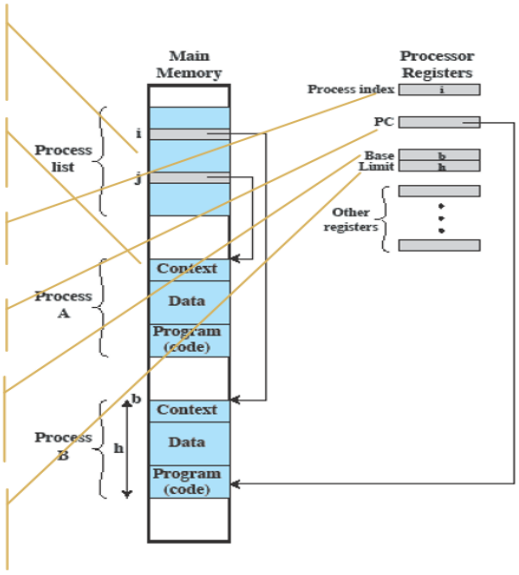
Kontekst procesa može biti smešten sa samim procesom ili odvojeno

Proces koji se trenutno izvršava na procesoru

Naredna instrukcija u aktivnom procesu

Početna adresa bloka u kojem je proces. Služi za relativno referenciranje programskog brojača i svih podataka

Dužina bloka u kojem je proces. Služi za zaštitu pristupa memoriji. Ne smeju se referencirati lokacije izvan granice



VIŠENITNA OBRADA

Proces je podeljen na niti koje se izvršavaju konkurentno

NIT je izvršna jedinica posla. Sadrži kontekst i sopstvenu oblast podataka. Izvršava se sekvencijalno i može se prekinuti.

PROCES je skup jedne ili više niti i dodeljenog sistemskog resursa. Kreiranjem niti programer ima veću kontrolu nad modularnošću aplikacije i vremenskim usklađivanjem događaja u njoj.

UPRAVLJANJE MEMORIJOM

ODGOVORNOST OS PRI UPRAVLJANJU MEMORIJOM

1. **IZOLACIJA PROCESA** - *Proces ne sme da pristupi delu memorije rezervisanom za drugi proces*
2. **AUTOMATSKO DODELJIVANJE I UPARIVANJE** - *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. Implementacija toga je **FAJL SISTEM**.*

VIRTUELNO ADRESIRANJE

Mehanizam koji omogućuje da programi vrše adresiranje sa logičke tačke gledišta. Program ne mora da vodi računa o količini fizički raspoložive glavne memorije, kao ni o stvarnoj adresi u glavnoj memoriji u koju je smešten (stvarna adresa se određuje u trenutku izvršavanja a ne u trenutku pisanja i prevođenja programa).

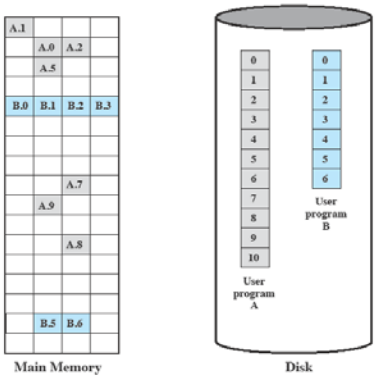
STRANIČENJE

Proces je podeljen u određeni broj blokova fiksne dužine koji se nazivaju stranice. Program adresira reč pomoću virtuelne adrese. Virtuelna adresa se sastoji od broja stranice i pomeraja unutar nje. Stranica može biti smeštena bilo gde u glavnoj memoriji.

Sistem straničenja pruža dinamičko mapiranje virtuelne adrese koja se koristi u programu ***NA REALNU ADRESU*** (realna adresa = fizička adresa u glavnoj memoriji)

VIRTUELNA MEMORIJA

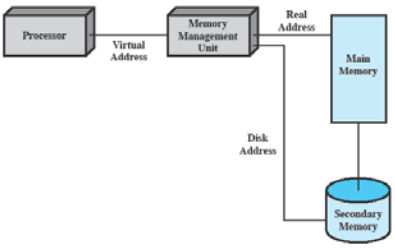
Ne moraju sve stranice procesa da budu sve vreme u glavnoj memoriji. Sve stranice se nalaze na disku, a samo neke u glavnoj memoriji.



Ukoliko se adresira stranica koja nije u glavno j memoriji, hardver za upravljanje memorijom (*MMU*) to otkriva i učitava u memoriju posebnu stranicu.

ADRESIRANJE VIRTUELNE MEMORIJE

Ako referencirana stranica nije u glavnoj memoriji, prebacuje se sa diska u glavnu memoriju.



Ako nema dovoljno mesta, neka od stranica u glavnoj memoriji mora da se prebaci na disk.

Hardver obezbeđuje mapiranje adresa a OS određuje politiku dodeljivanja memorije.(koje i koliko stranica će se naći u memoriji, koji se algoritam koristi za zamenu stranica u glavnoj memoriji, a koji algoritam za prebacivanje stranice iz glavne memorije na disk...)

ZAŠTITA INFORMACIJA I BEZBEDNOST

- **RASPOLOŽIVOST** - Zaštita sistema od prekida rada
- **POVERLJIVOST** - Korisnici ne smeju da čitaju podatke za koje nemaju pravo pristupa
- **INTEGRITET PODATAKA** - Zaštita podataka od nedozvoljene izmene
- **AUTENTIČNOST** - Provera identiteta korisnika i validnost poruka i podataka

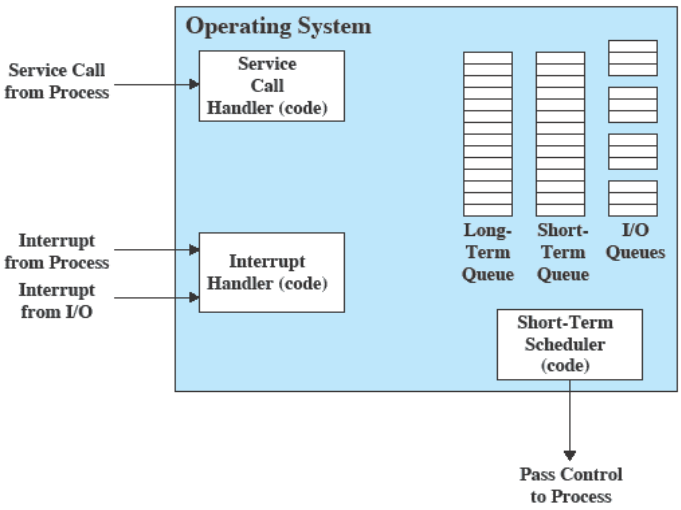
RASPOREĐIVANJE I UPRAVLJANJE RESURSIMA

OS upravlja resursima i raspoređuje njihovu upotrebu prema procesima. Pri tome, OS mora da obezbedi:

1. **NEPRISTRASNOST** - Poslovi sličnog tipa koji se takmiče za jedan resurs, treba da dobiju isti pristup tom resursu
2. **RAZLIČITOST ODZIVA** - Namerna pristrasnost dodelom resursa određenom procesu, što će omogućiti da se bolje ispuni ukupan skup zahteva
3. **EFIKASNOST** - OS pokušava da uveća propusnu moć, a smanji vreme odziva. Pošto su to suprotstavljeni kriterijumi, OS mora da nađe balans.

ELEMENTI OS ZA RASPOREĐIVANJE

- U kratkoročnom redu su procesi spremni za izvršavanje (kružno dodeljivanje ili prema prioritetu).
- 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 njegovu upotrebu.
- OS opslužuje prekide i sistemske pozive.
- Po opsluživanju prekida ili siste mskog poziva, kratkoročni raspoređivač određuje koji će se proces izvršiti kao naredni.



SIMETRIČNA VIŠEPROCESNA OBRADA – SMP

Bavi se raspoređivanjem procesa na hardveru sa simetričnim multiprocesorima. Procesori dele istu glavnu memoriju i U/I uređaje. Svi procesori 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 su:

- **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 sa različitim cenama i performansama na osnovu broja procesora.*

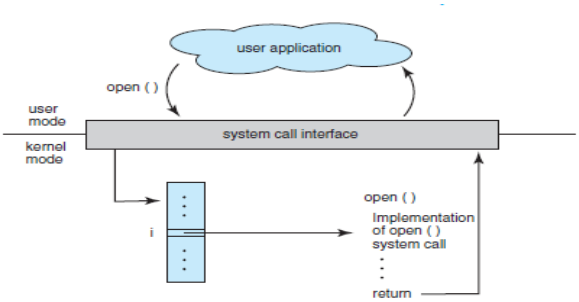
DIZAJN OS ZA SMP

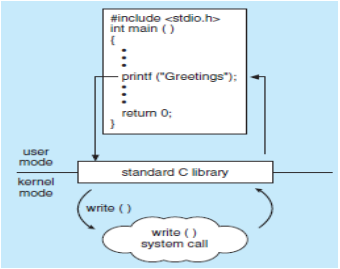
1. **Simultani konkurentni procesi ili niti** - Strukture kernela treba da podrže mogućnost konkurentnog pristupa od strane više procesa koji se istovremeno izvršavaju na različitim procesorima
2. **Raspoređivanje** - Svaki procesor može da sprovodi raspoređivanje, što komplikuje politiku raspoređivanja
3. **Sinhronizacija** - Međusobna isključivost i signaliziranje više aktivnih procesa istovremeno koji imaju pristup deljenim adresnim prostorima i U/I resursima
4. **Upravljanje memorijom** - Potrebno je iskoristiti paralelizam hardvera za bolju performansu, koordinirati procesore pri straničenju kako bi bili konzistentni kada više procesora deli istu stranicu
5. **Pouzdanost i otpornost na greške** - OS treba da reaguje na gubitak procesora smanjenjem performanse i ponovnim raspoređivanjem

KOMUNIKACIJA KORISNIČKE APLIKACIJE SA OS

SISTEMSKI POZIVI

OS pruža skup servisa koji se mogu pozvati iz sloja korisničkih aplikacija i koji obavljaju određenu funkcionalnost OS. Mogu se pozvati direktno ili indirektno korišćenjem sistemske biblioteke.





- **DIREKTAN POZIV** - Korisnička aplikacija zahteva uslugu OS slanjem sistemskog poziva a OS u režimu kernela izvršava traženi zadatak i vraća odgovor aplikaciji
- **INDIREKTAN POZIV** - SISTEMSKA BIBLIOTEKA sadrži programski kod za upućivanje sistemskih poziva. Služi kao srednji sloj između korisničke aplikacije i OS (omotač oko sistemskih poziva). Korisnička aplikacija ne mora da vodi računa o specifikaciji sistemskog poziva.

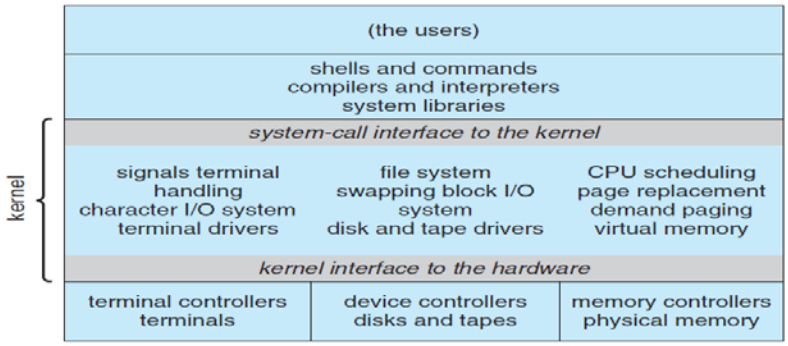
TIPOVI SISTEMSKIH POZIVA

- **UPRAVLJANJE PROCESIMA** - *kreiraj, učitaj, izvrši, završi, prekini, zauzmi memoriju, oslobodi memoriju, čekaj određeno vreme, čekaj na događaj, signaliziraj događaj, preuzmi i postavi atribute procesa*
- **UPRAVLJANJE FAJLOVIMA** - *kreiraj, otvori, zatvori, obriši, pročitaj, upiši, pozicioniraj se, preuzmi i postavi atribute fajla*
- **UPRAVLJANJE UREĐAJIMA** - *zatraži uređaj, otpusti uređaj, pročitaj, upiši, logički dodaj i izbaci uređaj iz sistema, preuzmi i postavi atribute uređaja*
- **EVIDENCIJA INFORMACIJA** - *preuzmi i postavi sistemsko vreme i datum, uzmi i postavi podatke iz sistema*
- **KOMUNIKACIJA** - *kreiraj i obriši komunikacionu vezu, pošalji poruku, primi poruku*

STRUKTURA KERNELA OS

MONOLITNO JEZGRO

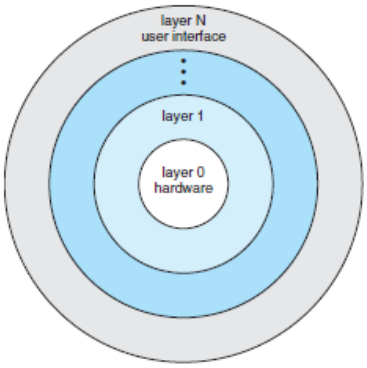
U jednom velikom jezgru ugrađene sve funkcionalnosti OS (raspoređivanje, fajl sistem, upravljanje memorijom...). Primer strukture UNIX sistema.



SLOJEVITA STRUKTURA

Funkcionalnosti su podeljene u hijerarhijski organizovane slojeve. Svaki sloj poziva operacije koje pruža niži nivo i pruža operacije koje poziva viši nivo

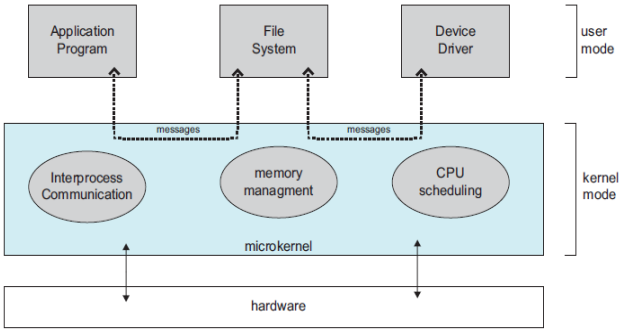
PROBLEM: Određeni delovi komuniciraju dvosmerno pa nije moguće odrediti koji deo treba da bude na višem hijerarhijskom nivou. Takođe je sporo jer se prolazi kroz niz sistemskih poziva kroz slojeve.



MIKROKERNEL

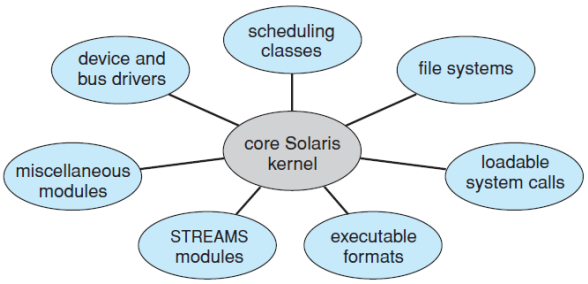
Jezgro sadrži samo nekoliko osnovnih funkcija a ostali servisi se obezbeđuju preko procesa koji rade u korisničkom režimu. Pojednostavljuje implementaciju, pruža fleksibilnost, i dobro je prilagođeno za distribuirana okruženja

PROBLEM: Slabe performanse zbog toga što servisi iz korisničkog moda moraju preko sistemskih poziva da obavljaju operacije OS.



MODULARNA STRUKTURA

Funkcionalnosti podeljene u module koji se dinamički učitavaju. Kernel uvek sadrži ključne funkcije a ostale se učitavaju dinamički u toku izvršavanja. Sistem je podeljen u nezavisne celine, kao kod slojevite strukture, ali svaka celina može da komunicira sa svakom. Modularno struktuirani sistemi - *Linux, Windows, MacOS X, Solaris*



HIBRIDNA STRUKTURA

Većina sistema u sebi kombinuje različite strukture.

Primer: Linux i Solaris su ujedno i monolitni i modularni. Windows je najvećim delom monolitan, ali ima deo funkcija koje se izvršavaju u korisničkom režimu kao kod mikrokernela.

POKRETANJE OPERATIVNOG SISTEMA

Kako pri pokretanju računara hardver zna gde je kod operativnog sistema da bi krenuo da ga izvršava?

BOOTSTRAP LOADER

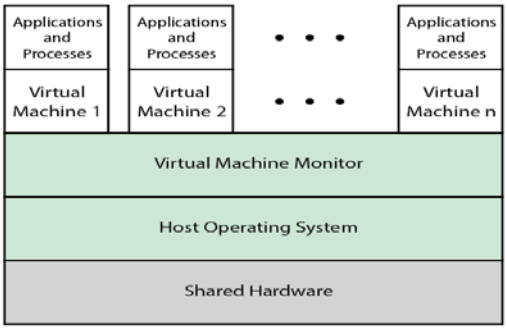
Program koji se prvi pokreće i koji je odgovoran za učitavanje OS u memoriju i pokretanje. Učitavanje je realizovano u 2 nivoa: Jednostavni bootloader zapisan u ROM memoriji koji se prvi pokreće (BIOS, UEFI). Ovaj loader je odgovoran da se sa predefinisane lokacije na disku učita kompleksniji loader koji učitava OS.

VIRTUELIZACIJA

Omogućuje da se na jednom računaru istovremeno izvršava više OS ili više instanci jednog OS. Na taj način moguće je na istoj platformi izvršavati više aplikacija koje rade na različitim OS. **HOST** operativni sistem može da podrži više virtuelnih mašina. Svaka mašina ima karakteristike posebnog OS i u nekim verzijama virtualizacije čak i posebne hardverske platforme.

KONCEPT VIRTUELNE MAŠINE

***Host* OS** se izvršava na deljenom hardveru. Svaka virtuelna mašina izvršava svoj OS a na ***Host* OS** postoji VIRTUAL MACHINE MONITOR – **VMM**, koji obezbeđuje vezu ovih virtuelnih OS sa hardverom (preko host OS).



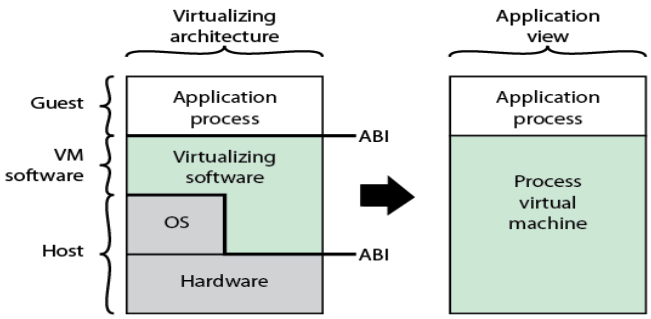
POJAM MAŠINE

- **Iz perspektive aplikacije**, mašinu definišu:
  - svojstva jezika visokog nivoa i sistemski bibliotečki pozivi
  - API definiše mašinu onako kako je vidi aplikacija
- **Iz perspektive procesa**, koji izvršava prevedeni aplikativni program, mašina je:
  - virtuelni memorijski prostor dodeljen procesu
  - procesorski registri koje može da koristi
  - skup mašinskih instrukcija korisničkog nivoa koje može da izvršava
  - skup sistemskih poziva OS koje može da poziva za U/I
  - ABI definiše mašinu onako kako je vidi proces
- **Iz perspektive OS**, mašina je:
  - fajl sistem i U/I resursi kojima procesi pristupaju
  - stvarna memorija i U/I resursi koje sistem adresira za procese
  - ISA predstavlja interfejs između OS i mašine

PROCESNA VIRTUELNA MAŠINA

Virtuelna platforma za izvršavanje jednog procesa. Stvara se kada i proces, i ukida kada se on završi. Kompajliranje palikacije stvara virtuelni binarni kod koji se izvršava u VM, tako da može da se izvrši na svakoj platformi za koju postoji VM.

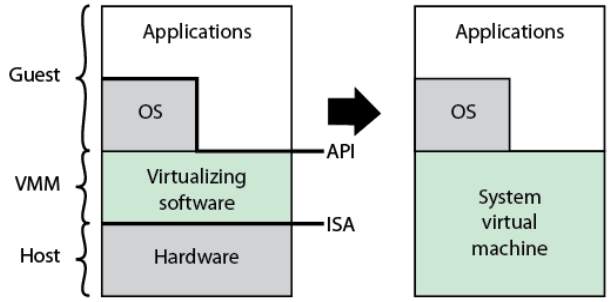
Primer: Java VM, Microsoft .NET Framework



SISTEMSKA VIRTUELNA MAŠINA

Jedna hardverska platforma može da podrži više izolovanih gostujućih OS. Virtuelizujući softver je host određenom broju gostujućih OS.

Primer: VMware, VirtualBox



2 - Procesi

POJAM PROCESA

OS upravlja izvršavanjem aplikacija. Resursi treba da budu dostupni za više aplikacija. Procesor deli više aplikacija. Procesor i U/I uređaji treba da budu korišćeni efikasno. Pristup u savremenim OS koji ovo obezbeđuje zasniva se na modelu u kojem izvršavanje aplikacije odgovara postojanju jednog ili više procesa.

DEFINICIJE PROCESA

- Program u izvršavanju
- Primerak programa koji se izvršava na računaru
- Entitet koji se može dodeliti I izvršavati na procesoru
- Jedinica aktivnosti koju karakterišu naredbe za izvršavanje, tekuće stanje I dodeljeni skup sistemskih resursa

ZAHTEVI OS PRI UPRAVLJANJU PROCESIMA

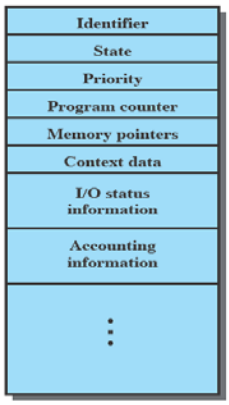
1. Preplitanje izvršavanja više procesa
2. Dodela resursa procesima i zaštita dodeljenih resursa od nekontrolisanog pristupa drugih procesa
3. Deljenje i razmena informacija između procesa
4. Sinhronizacija između procesa

ELEMENTI PROCESA

- **PROGRAMSKI KOD** - govori šta proces treba da uradi, može biti deljen između više instanci istog procesa.
- **SKUP PODATAKA NAD KOJIMA SE POSAO IZVRŠAVA**
- **ATRIBUTI KOJI OPISUJU STANJE PROCESA** - dodatne informacije koje OS skladišti da bi mogao da upravlja procesom
    - IDENTIFIKATOR
    - STANJE
    - PRIORITET
    - PROGRAMSKI BROJAČ (*adresa sledeće instrukcije u kodu*)
    - POKAZIVAČI NA MEMORIJSKE BLOKOVE (*na kod i podatke vezane za proces*)
    - KONTEKSTNI PODACI (*podaci iz registara procesora vezani za trenutno stanje procesa*)
    - U/I STATUSNE INFORMACIJE (*lista fajlova koje proces koristi, neobrađeni U/I zahtevi, dodeljeni U/I uređaji itd.*)
    - RAČUNOVODSTVENE INFORMACIJE (*korišćeno procesorsko vreme, vremenska ograničenja itd.)*

UPRAVLJAČKI BLOK PROCESA (UBP)

OS je zadužen za kreiranje i upravljanje UBP koji sadrži atribute procesa



UBP omogućuje multiprogramiranje:

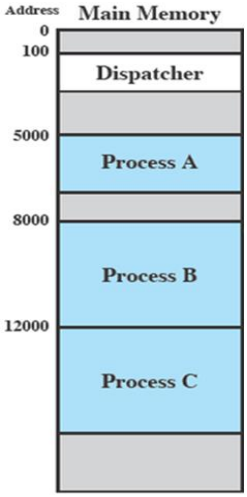
- sadrži informacije koje omogućuju da se proces prekine i kasnije nastavi izvršavanje kao da nije bilo prekida
- kada se proces prekine, snimaju se vrednosti procesorskih registara u UBP
- kada proces nastavlja izvršavanje, snimljene vrednosti se učitavaju iz UBP u procesorske registre

PRAĆENJE IZVRŠAVANJA PROCESA

- **TRAG PROCESA** - Niz instrukcija koje proces izvršava, čime je predstavljeno ponašanje procesa. Ponašanje se može predstaviti isprepletanim prikazom tragova različitih procesa
- **RASPOREĐIVAČ** (*dispatcher*) - program koji prebacuje procesor sa jednog procesa na drugi

PRIMER

Sva tri procesa su kompletno u glavnoj memoriji. Programski kod raspoređivača je takođe u memoriji.



5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

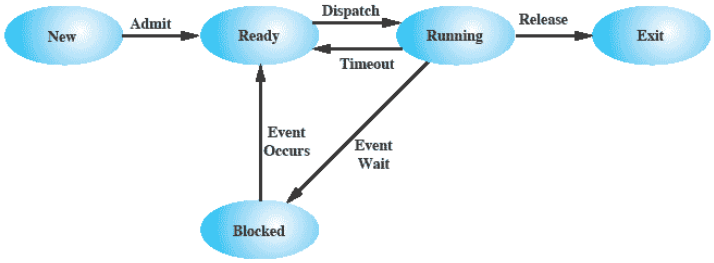
Proces A | Proces B | Proces C

tri procesa - trag

Proces A	1	5000	13	8000	23	12000	35	5006	47	12006
	2	5001	14	8001	24	12001	36	5007	48	12007
	3	5002	15	8002	25	12002	37	5008	49	12008
	4	5003	16	8003	26	12003	38	5009	50	12009
	5	5004	17	100	27	12004	39	5010	51	12010
Prekid	6	5005	18	101	28	12005	40	5011	52	12011
	7	100	19	102	29	100	41	100		
Raspoređivač	8	101	20	103	30	101	42	101		
	9	102	21	104	31	102	43	102		
	10	103	22	105	32	103	44	103		
	11	104			33	104	45	104		
	12	105			34	105	46	105		

STANJA PROCESA

1. **IZVRŠAVANJE** - proces čije instrukcije procesor trenutno izvršava
2. **SPREMAN** - proces koji je spreman za izvršavanje, ali trenutno procesor ne izvršava njegove instrukcije
3. **BLOKIRAN (*U ČEKANJU*)** - proces koji se ne može izvršavati dok se ne pojavi neki događaj, npr. završetak U/I operacije



4. **NOVI** - proces koji je upravo stvoren, ali ga OS nije još prihvatio u red spremnih procesa
5. **IZLAZ** - proces koji je OS uklonio iz reda spremnih procesa zato što je završio rad

PRELAZI STANJA

- **NULL -> NOVI** - *kada se kreira novi proces*
- **NOVI -> SPREMAN** - *prebacuje se kada je OS spreman da prihvati novi proces, obično zbog performansi postoji ograničenje broja aktivnih procesa*
- **SPREMAN -> IZVRŠAVANJE** - *raspoređivač prebacuje jedan od spremnih procesa u stanje izvršavanja*
- **IZVRŠAVANJE -> IZLAZ** - *kada proces da znak da je završio izvršavanje ili ako prekine sa radom*
- **IZVRŠAVANJE -> SPREMAN** - *kada procesu istekne maksimalno dozvoljeno vreme neprekidnog izvršavanja (quantum), kada se pojavi spreman proces višeg prioriteta ili kada proces dobrovoljno prepusti procesor*
- **IZVRŠAVANJE -> BLOKIRAN** - *kada proces zahteva operaciju čiji završetak mora da sačeka, npr. ako proces zahteva U/I operaciju ili čeka podatke od drugog procesa*
- **BLOKIRAN -> SPREMAN** – *kada se pojavi događaj zbog kojeg je proces bio blokiran*
- **SPREMAN -> IZLAZ** i **BLOKIRAN -> IZLAZ** - *proces potomak može biti prekinut od roditelja dok je u stanju SPREMAN ili BLOKIRAN*

STVARANJE PROCESA

Pri dodavanju novog procesa, OS kreira strukture podataka za upravljanje procesom i učitava u radnu memoriju proces ili deo procesa. Novi proces se kreira od strane OS pri startovanju programa ili pri izvršavanju procesa kada proces (*parent*) može od OS da zatraži kreiranje drugog procesa (*child*).

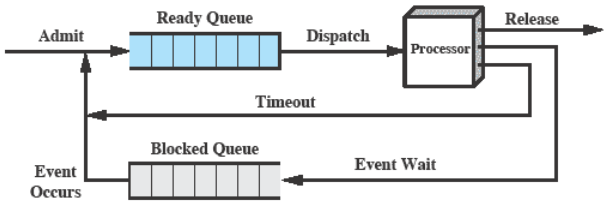
ZAVRŠAVANJE PROCESA

Mora postojati mehanizam da OS završi proces na osnovu indikatora dobijenog od procesa. OS je zadužen da oslobodi resurse koje je proces koristio. Indikator za izvršavanje može da bude:

- korisnička akcija (*exit*)
- greška u programu
- instrukcija roditeljskog procesa za završetak procesa potomka
- završavanje roditeljskog procesa
-

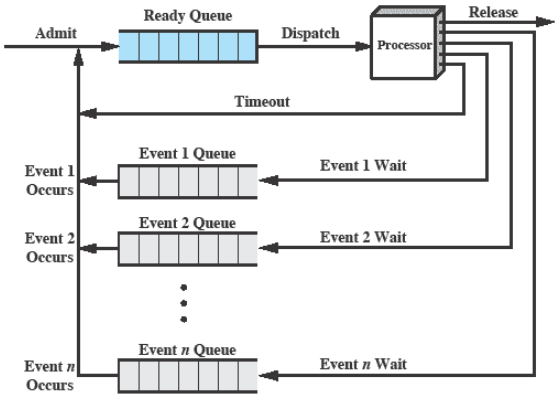
RASPOREĐIVANJE SA DVA REDA

1. **RED SPREMNIH PROCESA** - Iz ovog reda se bira jedan za izvršavanje
2. **RED BLOKIRANIH PROCESA** - Kada se pojavi događaj, proces prelazi u red spremnih a OS mora da proveri ceo red



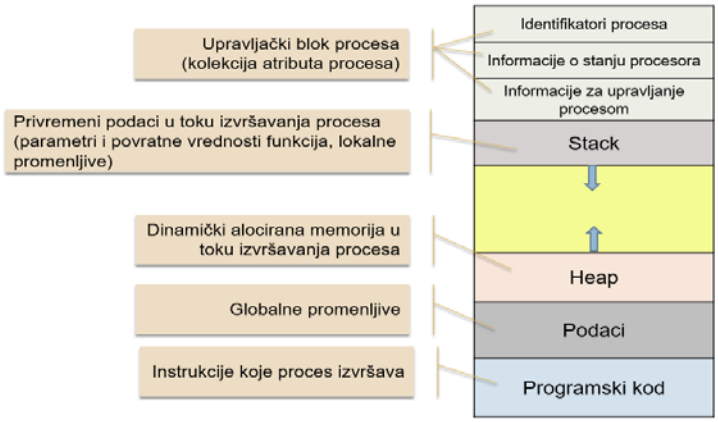
RASPOREĐIVANJE SA VIŠE REDOVA ZA BLOKIRANE PROCESE

Za svaki događaj postoji odvojen red blokiranih procesa. Kada se pojavi događaj, svi procesi iz tog reda se prebacuju u red spremnih.



UPRAVLJAČKE STRUKTURE PROCESA

Da bi upravljao procesima OS mora da zna lokaciju procesa i atribute procesa (identifikator, stanje ...)



Fizička reprezentacija procesa (slika procesa)

LOKACIJA SLIKE PROCESA

Proces je podeljen u stranice. Sve stranice se nalaze u sekundarnoj memoriji. U svakom trenutku jedan deo stranica je u glavnoj memoriji. Stranice ne moraju biti smeštene u fizički susednim blokovima memorije.

IDENTIFIKATORI PROCESA

- **JEDINSTVENI IDENTIFIKATOR PROCESA** je obično pozitivan ceo broj. Upravljačke strukture OS mogu da referenciraju proces preko tog identifikatora.
- **IDENTIFIKATOR RODITELJSKOG PROCESA** je identifikator procesa koji je stvorio proces
- **KORISNIČKI IDENTIFIKATOR** je identifikator korisnika koji je odgovoran za proces

INFORMACIJE O STANJU PROCESORA

Predstavljaju sadržaj registara procesora. Kada se proces prekine, potrebno je sačuvati trentno stanje procesorskih registara. Registri su vidljivi korisniku i mogu se referencirati iz mašinskog koda programa.

UPRAVLJAČKI I STATUSNI REGISTRI su programski brojač, uslovni kodovi (*rezultat poslednje operacije*) i informacije o statusu. **POKAZIVAČ NA STEK** pokazuje na vrh steka procesora.

INFORMACIJE ZA UPRAVLJANJE PROCESOM

1. **INFORMACIJE ZA RASPOREĐIVANJE –** stanje procesa, prioritet, događaj na koji proces čeka, dodatne informacije (npr. Vreme čekanja – zavisi od algoritma za raspoređivanje)
2. **STRUKTUIRANJE PODATAKA –** struktura koja međusobno povezuje procese, npr. red čekanja procesa istog prioriteta
3. **MEĐUPROCESNA KOMUNIKACIJA -** signali i poruke vezani sa komunikacijom između procesa
4. **PRIVILEGIJE PROCESA -** kojim delovima memorije proces može da pristupa i koje instrukcije ima pravo da izvršava
5. **UPRAVLJANJE MEMORIJOM -** pokazivači na tabelu stranica za virtuelnu memoriju
6. **VLASNIŠTVO NAD RESURSIMA –** koje resurse proces kontroliše

UPRAVLJANJE PROCESOM

REŽIMI IZVRŠENJA PROCESA

- **KORISNIČKI REŽIM** - manje privilegija, za korisničke programe
- **REŽIM KERNELA** - moguće je izvršavati i privilegovane instrukcije, potpuna kontrola nad procesorom i svim njegovim instrukcijama, registrima i memorijom. OS se izvršava u režimu kernela.

Promena režima izvršavanja se vrši promenom bita u statusnom registru procesora koji ukazuje na režim. Taj bit se menja kada se desi određeni događaj. Sistemski poziv prevodi sistem iz korisničkog režima u režim kernela.

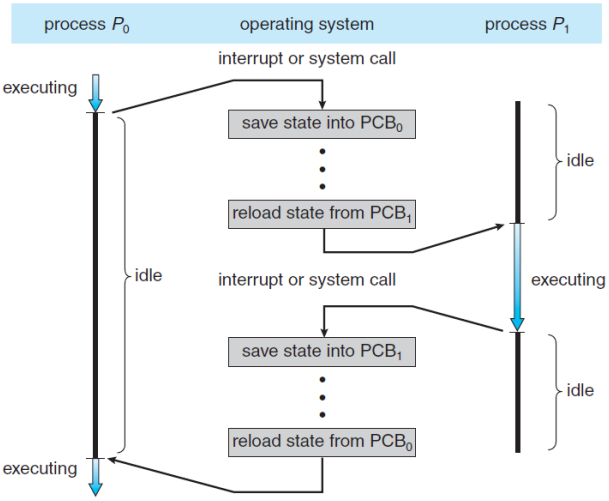
STVARANJE PROCESA

1. Novom procesu se dodeljuje jedinstveni identifikator a u tabelu procesa se dodaje nova stavka
2. Dodeljuje se prostor za proces za sve elemente slike procesa. Ovaj prostor može da bude i virtuelan, ne samo fizički
3. Inicijalizuje se upravljački blok procesa (*UBP*). 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

Promena trenutno aktivnog procesa koji se izvršava na procesoru.

1. Prekida se proces koji se izvršava
2. OS drugom procesu dodeljuje stanje **IZVRŠAVANJE**
3. Predaje se kontrola drugom procesu



Koji događaj pokreće komutiranje procesa?

1. **KADA SE DESI PREKID**
   - ***Prekid generatora takta*** (*ako je isteklo maksimalno dozvoljeno korišćenje procesora (time slice) proces se prekida i prebacuje u stanje SPREMAN)*
   - ***U/I prekid*** *(OS prebacuje procese blokirane zbog tog U/I događaja u stanje SPREMAN)*
   - ***Greška memorije*** *(proces referencira stranicu koja nije u glavnoj memoriji, OS preuzima kontrolu da bi prebacio traženu stranicu iz spoljne u glavnu memoriju)*

2. **KADA SE DESI GREŠKA U IZVRŠAVANJU**
   - *OS preuzima kontrolu da bi reagovao na grešku*

3. **KADA IMAMO POZIV OPERACIJE OS IZ PROCESA** ***(****SISTEMSKI POZIV****):***
   - *npr. Proces zahteva pristup falju i OS preuzima kontrolu da bi izvršio U/I akciju*

   Ne podrazumeva nužno prelazak na izvršavanje drugog procesa. Ako operacija ne zahteva blokiranje procesa, proces će nastaviti da izvršava kod OS u režimu kernela. Moraju se samo sačuvati stanja procesorskih registara.

Koje strukture podataka OS ažurira pri komutaciji procesa?

1. **ČUVANJE SADRŽAJA PROCESORA** (*programski brojač i drugi registri*)
2. **AŽURIRANJE UBP** (*promena stanja na SPREMAN ili BLOKIRAN, ažuriranje drugih polja*)
3. **PREBACIVANJE UBP U ODGOVARAJUĆI RED** (*red spremnih ili blokiranih*)
4. **IZBOR NAREDNOG PROCESA ZA IZVRŠAVANJE**
5. **AŽURIRANJE UBP 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 UBP IZABRANOG ZA IZVRŠAVANJE**

MEĐUPROCESNA KOMUNIKACIJA

KOOPERACIJA IZMEĐU PROCESA

Procesi koji se konkurentno izvršavaju mogu biti nezavisni ili međusobno zavisni.

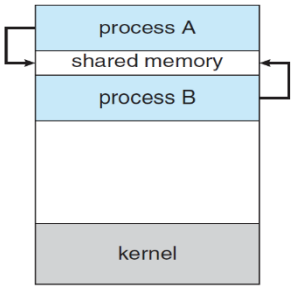
1. **NEZAVISNI** - ako na njihovo izvršavanje ne utiču drugi procesi. Proces koji ne deli podatke sa drugim procesima je nezavisan.
2. **MEĐUSOBNO ZAVISNI** - ako utiču jedni drugima na izvršavanje. Svaki proces koji deli podatke sa drugim procesima je zavisan od njih. Međusobno zavisni procesi moraju da sarađuju putem međuprocesne komunikacije.

RAZLOZI ZA MEĐUPROCESNU KOMUNIKACIJU

1. **DELJENJE INFORMACIJA** - različitim procesima trebaju isti podaci, npr. fajl kojem pristupaju različiti procesi
2. **UBRZANJE IZRAČUNAVANJA** - nad istim podacima izračunavanja mogu da vrše različiti procesi kako bi na multiprocesorskoj arhitekturi dobili ubrzanje
3. **MODULARNOST** - pri implementaciji se mogu napraviti logički tokovi izvršavanja koji pristupaju istim podacima

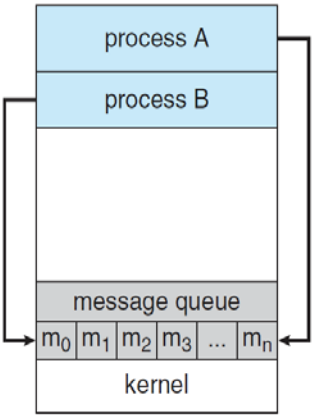
NAČINI MEĐUPROCESNE KOMUNIKACIJE

1. DELJENA MEMORIJA



Procesi uspostavljaju region deljene memorije, najčešće u adresnom prostoru procesa koji kreira region. Drugi proces zakači region u svoj adresni prostor. Procesi komuniciraju kroz upis i čitanje podataka u ovom regionu. Format podataka, isključivost i sinhronizacija su odgovornost procesa.

2. RAZMENA PORUKA



Procesi ne moraju eksplicitno da kreiraju region deljene memorije, već komuniciraju pozivima operacija *send(msg)* i *receive(msg)*

POREĐENJE TIPOVA MEĐUPROCESNE KOMUNIKACIJE

Oba tipa komunikacije se koriste u savremenim OS

DELJENA MEMORIJA - brža jer ne zahteva sistemske pozive za pristup podacima

RAZMENA PORUKA - pogodnija za distribuirane sisteme u kojima uglavnom nema deljene memorije

TIPOVI RAZMENE PORUKA

- ***BLOKIRANA KOMUNIKACIJA***

Blokirano slanje - nakon slanja poruke pošiljalac ostaje blokiran dok primalac ne primi poruku

Blokirani prijem - nakon poziva prijema poruke, primalac ostaje blokiran dok poruka ne bude raspoloživa

- ***NEBLOKIRANA KOMUNIKACIJA***

Neblokirano slanje - nakon slanja poruke pošiljalac nastavlja sa izvršavanjem

Neblokirani prijem - nakon poziva prijema poruke, primalac dobija poruku ako je raspoloživa ili NULL

- ***DIREKTNO ADRESIRANJE***

  Proces mora eksplicitno da navede identifikator drugog procesa sa kojim komunicira. Komunikacioni link povezuje tačno 2 procesa. Operacija receive može biti realizovana i tako da se ne navodi od kojeg pošiljaoca se prima poruka tako što se u parametar id upiše id procesa čija je poruka pročitana.

- ***INDIREKTNO ADRESIRANJE***

  Poruke se ne šalju direktno od pošiljaoca do primaoca već se šalju deljenim strukturama podataka - POŠTANSKI SANDUČIĆI (MAILBOX). Procesi upisuju podatke u MAILBOX a drugi procesi čitaju podatke iz njega. Oblici indirektnog adresiranja mogu biti: 1-1, 1-N, N-1 I N-N.

  VLASNIŠTVO NAD SANDUČIĆEM

  1. *AKO JE VLASNIK PROCES* - sandučić nestaje kada se proces završi a procesi koji komuniciraju sa ovim sandučićem moraju dobiti obaveštenje da on više ne postoji
  2. *AKO JE VLASNIK OS* - sandučić postoji nezavisno od procesa, OS pruža mehanizam da se kreira ili obriše sandučić i pošalje ili primi poruka

  BAFEROVANJE PORUKA

  Poruke u sandučiću stoje u privremenom redu.
  Red može biti takav da ima NULTI KAPACITET (1 poruka), OGRANIČEN KAPACITET (n poruka) ili NEOGRANIČEN KAPACITET.

- ***BAFEROVANI ZAPIS***
- ***NEBAFEROVANI ZAPIS***

PIPES

Poseban tip međuprocesne komunikacije koji OS standardno pružaju. Omogućuju komunikaciju 2 procesa po principu PROIZVOĐAČ - POTROŠAČ. Proizvođač upisuje u jedan kraj a potrošač čita podatke sa drugog kraja.

TIPOVI PIPES

Da li je komunikacija jednosmerna ili dvosmerna? Ako je dvosmerna, da li istovremeno podaci mogu da putuju u oba smera (full duplex) ili u jednom trenutku samo u jednom smeru (half duplex)?

Da li treba neka relacija da postoji između procesa (parent - child)?

Da li je komunikacija samo između procesa na istom računaru ili može i preko mreže?

Primer pipes u Linux-u

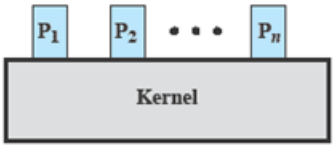
IZVRŠAVANJE OS

OS je skup programa koje izvršava procesor, radi kao i svaki drugi softver. Da li je OS proces i ko i kako njime upravlja? Postoje 3 pristupa u dizajnu:

1. **JEZGRO OS KOJE NIJE PROCES**

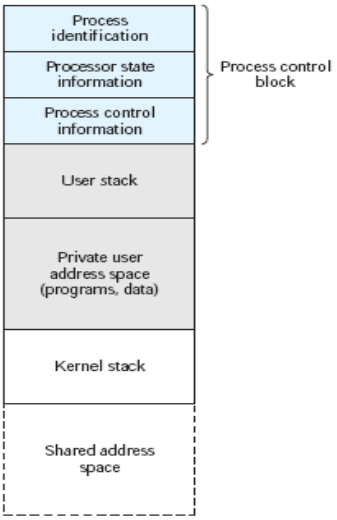
   Kod starijih OS, OS se izvršava izvan bilo kog procesa, on je zaseban entitet.

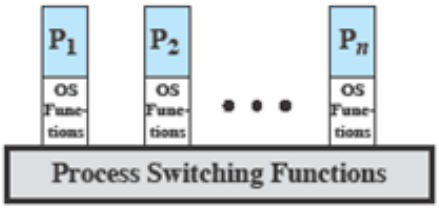
   Kod OS izvršava se u privilegovanom režimu a kao procesi se tretiraju samo korisnički procesi.



2. **IZVRŠAVANJE UNUTAR KORISNIČKIH PROCESA**

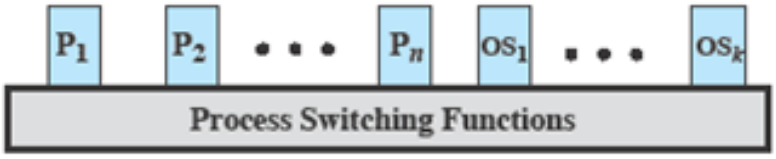
   Softver OS se izvršava unutar korisničkih procesa. Kod i podaci OS dele svi korisnički procesi. Kada je potrebno izvršiti sistemski poziv, proces se samo prebaci u režim kernela i izvršava kod OS iz deljenog adresnog prostora. Za komutiranje procesa postoji deo OS koji se izvršava izvan korisničkih procesa.





3. **OS ZASNOVAN NA PROCESIMA**

   OS je implementiran kao skup procesa. Funkcije jezgra organizovane su u procese, a ovi procesi se izvršavaju u režimu kernela.

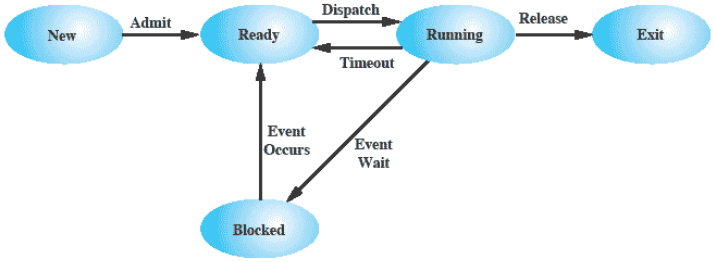


3 - Niti

PROCESI

- Program u izvršavanju
- Proces kao jedinica izvršavanja I raspoređivanja
  - Procesor izvršava instrukcije procesa isprepletano sa izvršavanjem drugih procesa
  - OS raspoređuje procese
- Proces kao jedinica kojoj se dodeljuje resurs
  - Proces sadrži virtuelni adresni prostor u koji je smeštena slika procesa
  - Procesu se može dodeliti vlasništvo nad memorijom, U/I resursom ili fajlom

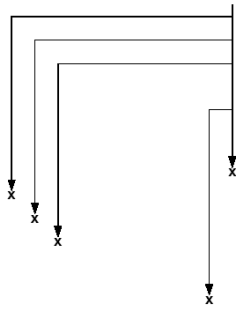
STANJA PROCESA



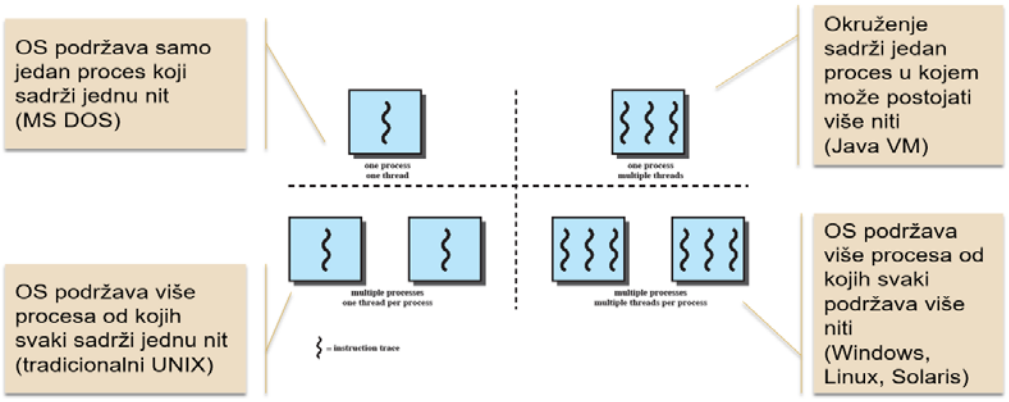
1. **IZVRŠAVANJE** - proces čije instrukcije procesor trenutno izvršava
2. **SPREMAN** - proces koji je spreman za izvršavanje, ali trenutno procesor ne izvršava njegove instrukcije
3. **BLOKIRAN (*U ČEKANJU*)** - proces koji se ne može izvršavati dok se ne pojavi neki događaj, npr. završetak U/I operacije
4. **NOVI** - proces koji je upravo stvoren, ali ga OS nije još prihvatio u red spremnih procesa
5. **IZLAZ** - proces koji je OS uklonio iz reda spremnih procesa zato što je završio rad

VIŠENITNA OBRADA

Sposobnost operativnog sistema da podrži više konkurentnih putanja izvršavanja jednog procesa.
Nit je jedan tok izvršavanja u okviru procesa. Treba razlikovati višenitnu obradu i paralelizam!



ODNOS PROCESA I NITI



PROCES U VIŠENITNOJ OBRADI

- **JEDINICA DODELE RESURSA** - Virtuelni adresni prostor u kome je smeštena slika procesa
- **JEDINICA ZAŠTITE RESURSA** - Zaštićen pristup procesoru, drugim procesima (međuprocesna komunikacija), fajlovima i U/I resursima (uređaji, kanali itd.)

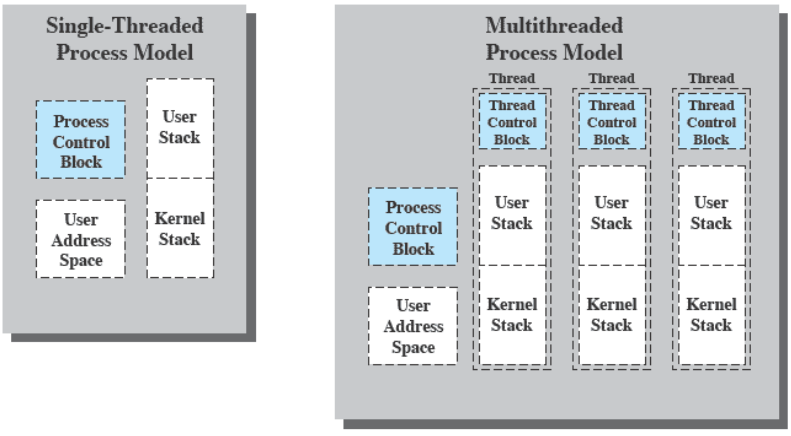
NIT U VIŠENITNOJ OBRADI

Jedan proces može da sadrži više niti. Svaka nit ima:

1. **STANJE IZVRŠAVANJA** (izvršavanje, spremna...)
2. **KONTEKST NITI** (sačuvano stanje procesorskih registara dok se ne izvršava)
3. **STEK IZVRŠAVANJA NITI**
4. **PRISTUP MEMORIJI I RESURSIMA SVOG PROCESA** (deljeno sa drugim nitima istog procesa)

MODEL PROCESA SA VIŠENITNOM OBRADOM

Svaka nit ima svoj UBP i stek. Sve niti dele adresni prostor i imaju pristup istim podacima.



KORISTI OD VIŠENITNE OBRADE

1. **POZADINSKI POSAO** - Dugačke ili blokirajuće pozadinske operacije se mogu izvršavati u posebnoj niti, (npr. upis u LOG fajl). Interakcija sa korisnikom je moguća i dok druga nit nije završila operaciju.
2. **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.
3. **BRZINA IZVRŠAVANJA** - Dok je jedna nit procesa blokirana, druge mogu da se izvršavaju
4. **DELJENJE RESURSA UNUTAR APLIKACIJE -** Niti istog procesa dele istu memoriju. Komunikacija između niti je jednostavnija i brža nego između procesa.
5. **EKONOMIČNOST** - Brže je napraviti novu nit nego poseban proces. Takođe, brže je komutiranje između dve niti istog procesa nego između 2 procesa.
6. **ISKORIŠĆENJE VIŠEPROCESORSKE ARHITEKTURE** - Mogućnost ubrzanja aplikacije ukoliko se niti izvršavaju na različitim procesorima.
7. **MODULARNA STRUKTURA PROGRAMA** - Organizacija koda u više niti razdvaja logičke delove programa što čini program jednostavnijim za implementaciju i održavanje.

RASPOREĐIVANJE NITI

Ako OS podržava niti, raspoređivanje se vrši na nivou niti. Većina informacija o stanju izvršavanja čuva se na nivou niti.

Neke akcije utiču na sve niti. OS tada mora raspoređivanje vršiti na nivou procesa (npr. prekidanjem procesa prekidaju se sve njegove niti)

STANJA NITI

Kao i kod procesa: **IZVRŠAVANJE**, **SPREMAN** i **BLOKIRAN**

Akcije kojima se menja stanje niti:

1. ***KREIRANJE*** - kreira se upravljački blok niti i stek a zatim se nit postavlja u stanje SPREMAN
2. ***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
3. ***DEBLOKIRANJE*** - kada se događaj koji je nit čekala desi
4. ***ZAVRŠAVANJE***

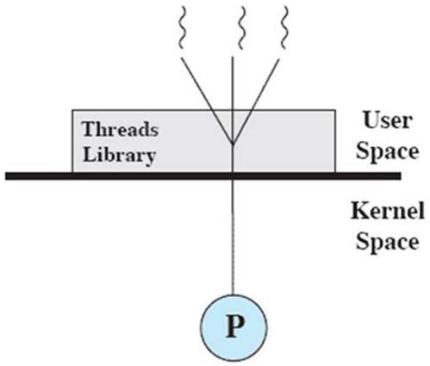
SINHRONIZACIJA NITI

Sve niti jednog procesa dele isti adresni prostor. Potrebno je sinhronizovati pristup ovim deljenim resursima. Bez sinhronizacije može doći do neispravnog rada programa i do nekonzistentnog stanja resursa.

TIPOVI NITI

NITI NIVOA KORISNIKA – ULT (User level threads)

Sav posao upravljanja nitima vrši aplikacija. Kernel nije svestan postojanja niti.



Niti se kreiraju u okviru aplikacije korišćenjem biblioteke niti.

Biblioteka sadrži kod za:

- pravljenje i uništavanje niti
- prosleđivanje poruka i podataka između niti
- raspoređivanje niti
- čuvanje i oporavljanje sadržaja niti

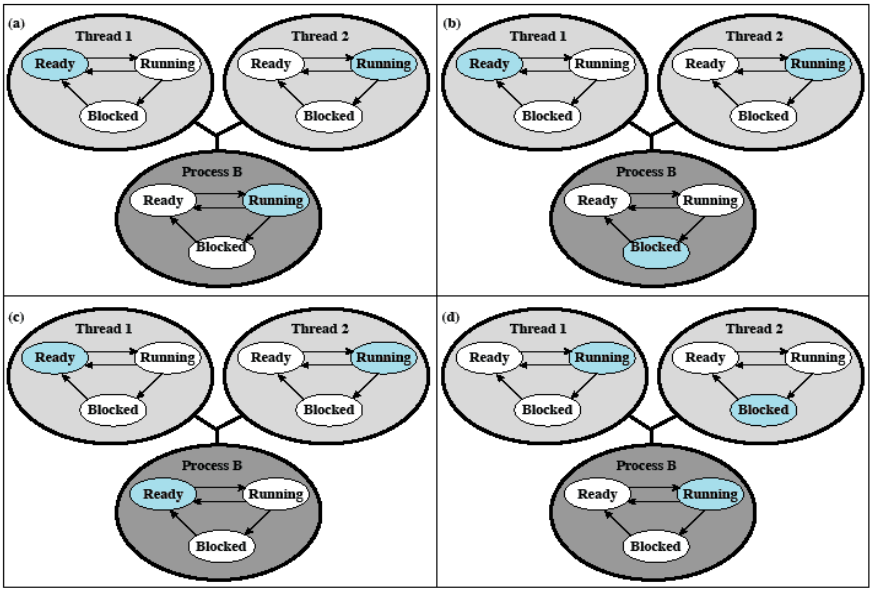
Sve aktivnosti se odvijaju u korisničkom prostoru ***UNUTAR JEDNOG PROCESA***.

RASPOREĐIVANJE ULT

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.

Npr. stanje niti IZVRŠAVANJE

- znači da je nit aktivna u okviru procesa
- ne mora nužno da znači da se nit trenutno izvršava na procesoru
- kada procesor krene da izvršava proces, izvršavaće se nit u stanju IZVRŠAVANJE iz tog procesa



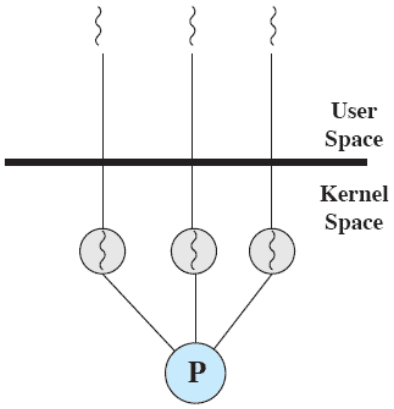
PREDNOSTI ULT

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

MANE ULT

1. Kada se proces blokira sve niti istog procesa postaju blokirane
2. Ne može se iskoristiti za multiprocesiranje jer su sve niti jedan proces pa se izvršavaju na jednom procesoru

NITI NIVOA KERNELA – KLT (Kernel level threads)



Aplikacija inicira kreiranje niti. Svo upravljanje nitima obavlja kernel. Kernel održava podatke za proces u celini kao i za svaku pojedinačnu nit. Kernel vrši raspoređivaje na bazi niti.

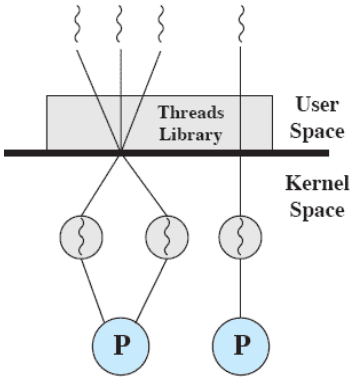
PREDNOSTI KLT

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

MANE KLT

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

KOMBINOVANI PRISTUPI

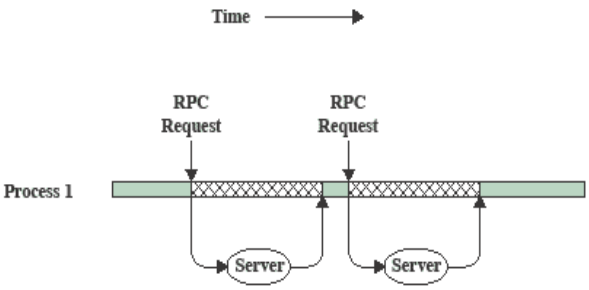


Pravljenje niti i najveći deo upravljanja vrši se u korisničkom prostoru. Višestruke ULT se mogu preslikati u neki (manji ili jednak) broj KLT.

Primer poboljšanja performansi upotrebom višenitne obrade

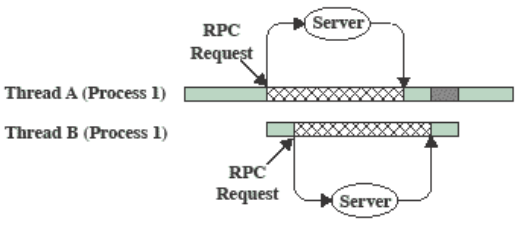
Program upućuje dva zahteva udaljenom serveru da bi izračunao rezultat.

Varijanta sa jednom niti



Varijanta sa dve niti

**blokiranje jedne niti ne blokira drugu nit*



NITI U C++11 STANDARDU

- Predstavljene objektom klase thread iz zaglavlja *<thread>*
- Instanciranje objekta kreira novu nit koja postaje spremna.
- Parametar konstruktora klase thread je funkcija koju nit izvršava (funkcija se zove telo niti)
- Ulazna tačka programa je i dalje *main*(). Završetak je kada se završi *main*()
- Metoda join() klase thread
    - stvaralac čeka da stvorena nit završi rad, čime se spajaju dva toka izvršavanja
    - metoda je ***BLOKIRAJUĆA*** - stvaralac će biti blokiran dok stvorena nit ne završi rad
- Metoda detach() klase thread
    - razdvaja stvaraoca od stvorene niti
    - tada je dozvoljeno da stvaralac završi rad bez obzira na rad stvorene niti
- Parametri tela niti se prenose kao parametri konstruktora klase *thread* nakon naziva funkcije. Potrebno je prenositi objekte po referenci u funkciju ***EKSPLICITNO*** funkcijom *ref*(objekat)
- Povratna vrednost funkcije se zanemaruje

4 - Konkurentnost

UPRAVLJANJE PROCESIMA

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

Konkurentnost se javlja svuda gde se vrši preplitanje različitih procesa. Važan je aspekt pri svakom upravljanju višestrukim procesima. Pitanja međusobne interakcije i zavisnosti procesa. Kompletan scenario i vremenski trenutak preplitanja je u pravilu nepredvidiv.

Javlja se kod:

- **Višestruke aplikacije** (kod multiprogramiranih sistema različiti procesi pristupaju istim resursima)
- **Struktuirane korisničke aplikacije** (jedna aplikacija može da sadrži više konkurentnih procesa/niti)
- **OS** (funkcije OS su implementirane kao više procesa/niti)

PROBLEMI

1. **DELJENJE GLOBALNIH RESURSA** (ŠTETNO PREPLITANJE) - Dva procesa vrše upis/čitanje nad istom globalnom promenljivom. Neodređen redosled izvršavanja rezultata i operacija.

   Primer: Dva procesa pozivaju metodu *echo*, x je deljeni resurs. Neispravan rad ako se desi preplitanje nakon naredbe u liniji 3. Problem se može rešiti ako se odredi da samo jedan proces u datom trenutku može izvršavati kod metode.

   ```
   1. int x;
   2. void echo() {
   3.    cin >> x;
   4.    int y = x;
   5.    cout << y;
   6. }
   ```

2. **KOMPLIKOVANIJA DODELA RESURSA** - Resurs može biti dodeljen procesu koji ga ne koristi. Procesi mogu biti međusobno blokirani zbog čekanja na resurse.
3. **DETEKCIJA GREŠAKA** - Teže je utvrditi zašto se program neočekivano ponaša. Rezultati nisu deterministički i teže je reprodukovati neočekivano ponašanje

REŠENJA

Problemi se rešavaju tako što se odredi da samo jedan proces u datom trenutku može izvršavati kod metode ili koristiti određenu promenljivu.

- **MEĐUSOBNA ISKLJUČIVOST**- Mogućnost procesa da obavi akciju bez neg. uticaja drugih procesa

1. **SINHRONIZACIJA** - Usklađivanje ponašanja procesa sa aktivnošću drugih procesa

DATA RACE vs DETERMINIZAM

DATA RACE - Rezultat programa zavisi od scenarija preplitanja i redosleda izvršavanja instrukcija između procesa. Ponovna izvršavanja istog koda ne daju isti rezultat.

DETERMINISTIČKO PONAŠANJE - Rad procesa mora biti nezavisan od brzine izvršavaja relativne u odnosu na brzinu ostalih tekućih procesa.

UZAJAMNO DELOVANJE PROCESA

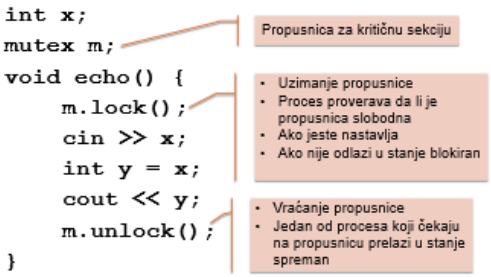
1. **PROCESI SU SVESNI DRUGIH PROCESA**
   - **EKSPLICITNA SINHRONIZACIJA AKTIVNOSTI** - Procesi su projektovani da zajednički obave posao
2. **PROCESI NISU SVESNI DRUGIH PROCESA -** Do nadmetanja za resurse dolazi kada nezavisni procesi koji nisu predviđeni da rade zajedno ipak pristupaju istim resursima.
   - **UZAJAMNA ISKLJUČIVOST PROCESA** - više procesa treba da koristi isti resurs. Za ispravan rad potrebno je da u jednom trenutku samo jedan proces pristupa resursu. Takav resurs nazivamo ***KRITIČNI RESURS***.
   - **KRITIČNA SEKCIJA** - deo programa u kojem se pristupa kritičnom resursu. Za ispravan rad, kod u kritičnoj sekciji procesi moraju da izvršavaju sekvencijalno (jedan po jedan). U jednom trentku samo jedan proces sme da bude u kritičnoj sekciji.

     Ako je jedan proces ušao u kritičnu sekciju, šta se dešava sa drugim procesima koji pokušavaju da uđu?

     - **BUSY WAITING** - proces koji ne može da uđe u kritičnu sekciju ostaje aktivan neprekidno proveravajući da li može da uđe. Proces troši procesorsko vreme i kada nema uslova da radi. (*while*)

```
int x;
bool is_lock_free = true;
void echo() {
     while (!is_lock_free) {
     }
     is_lock_free = false;
     cin >> x;
     int y = x;
     cout << y;
     is_lock_free = true;
}
```

     - **BLOKIRANJE PROCESA** - proces koji ne može da uđe u kritičnu sekciju odlazi u stanje blokiran. Sistem ga obaveštava kada može da uđe u kritičnu sekciju. (*mutex*)



PROPUSNICA (LOCK)

2. Svaki proces pre ulaska u kritičnu sekciju zatraži zaključavanje **DELJENOG OBJEKTA**
3. Proces koji prvi zatraži zaključavanje, uspeva da zaključava objekat i ulazi u kritičnu sekciju
4. Zaključavanje je implementirano uz oslonac na hardversku podršku za uzajamnu isključivost
5. Svaki naredni proces ne uspeva da zaključa objekat i prelazi u stanje blokiran
6. Proces pri izlasku iz kritične sekcije otključava objekat
7. Jedan od procesa koji čekaju, uspeva da zaključa objekat i ulazi u kritičnu sekciju
8. Da bi ovo radilo, svi procesi moraju da zaključavaju **ISTI** objekat (ne sme svaki proces raditi sa svojom lokalnom kopijom)

IMPLEMENTACIJA

Pri uzimanju propusnice potrebno je proveriti da li je propusnica slobodna i ako jeste, zauzeti propusnicu, a ako nije, postaviti proces u stanje blokiran. Dva procesa mogu da utvrde da je propusnica slobodna i da oba uđu u kritičnu sekciju. Zato se zauzimanje propusnice ne implementira na ovaj način, već se oslanja na hardversku podršku za uzajamnu isključivost.

```
1. if (is_lock_free)
2.     is_lock_free = false;
```

HARDVERSKA PODRŠKA ZA UZAJAMNU ISKLJUČIVOST

ONEMOGUĆAVANJE PREKIDA - Proces se izvršava do prekida. Ako onemogućimo prekide pri zauzimanju propusnice, ne može doći do preplitanja. **MANE**: Neefikasno jer je procesor ograničen da prepliće programe. Ne radi na multiprocesorskoj arhitekturi (procesi se izvršavaju na različitim procesorima, onemogućavanje prekida na jednom procesoru ne sprečava drugi proces da pristupi resursu).

SPECIJALNE MAŠINSKE INSTRUKCIJE - Hardver obezbeđuje instrukcije koje obavljaju više operacija atomski. Ove instrukcije se mogu iskoristiti za zauzimanje propusnice. Proces u jednom nedeljivom koraku proverava da li može da zauzme propusnicu i ako može, postavlja indikator da je zauzeo

INSTRUKCIJA Compare&Swap - Instrukcija nedeljivo izvršava skup operacija. Proverava se trenutna vrednost i postavlja se nova vrednost ako trenutna vrednost ispunjava uslov.

```
int compare_and_swap(int* word, int testval, int newval)
{
      int oldval = *word;
      if (oldval == testval)
            *word = newval;
      return oldval;
}
```

Ilustracija upotrebe i ponasanja u jeziku viseg nivoa. Zaključava se objekat pri ulasku u kritičnu sekciju.

```
if (compare_and_swap(locked_flag, 0, 1) == 0) {
     locked = true;
} else {
     //proces ide na cekanje
}
```

INSTRUKCIJA Exchange - Ilustracija upotrebe i ponasanja u jeziku viseg nivoa. Zaključava se objekat pri ulasku u kritičnu sekciju.

```
void exchange(int* register, int* memory) {
     int temp = *memory;
     *memory = *register;
     *register = temp;
}
```

```
bool locked = false; //globalna
```

```
bool keyi = true; //lokalna za svaki proces
exchange(keyi, locked);
if (keyi == true) {
     //proces ide na cekanje
}
```

KRITIČNA SEKCIJA U C++11 STANDARDU

Kreiranje kritične sekcije vrši se objektima klase *mutex*. Objekat klase *mutex* je taj deljeni objekat kojeg procesi zaključavaju. Metode klase mutex su:

9. lock() - zaključava mutex objekat ako je otključan, u suprotnom proces odlazi u čekanje
10. unlock() - otključava mutex objekat
11. try_lock() - pokušava da zaključa mutex, ako ne uspe vraća false, proces ne odlazi u čekanje

Pri korišćenju klase *mutex* postoji opasnost da propusnica ostane zaključana.
Klasa *unique_lock* koristi se umesto *mutex* klase :

12. Upravlja zaključavanjem mutexa
13. U konstruktoru se kao parametar prosleđuje mutex koji je potrebno zaključati
14. U destruktoru vrši automatsko otključavanje mutexa

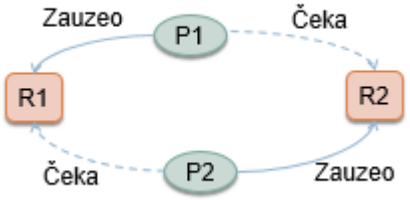
Korišćenjem objekta klase unique_lock propusnica se oslobađa automatski kada objekat prestane da postoji.

KOPIRANJE MUTEX OBJEKTA

Objekat klase *mutex* nije moguće kopirati po vrednosti jer bi se desilo da niti zaključavaju različite kopije propusnice. Realizovano je eksplicitnom naredbom kompajleru da ne generiše konstruktor kopije u klasi *mutex*.

```
class mutex  {
   . . .
   mutex(const mutex&) = delete;
   . . .
}
```

NADMETANJE PROCESA ZA RESURSE



1. **UZAJAMNO BLOKIRANJE (**DEADLOCK) - Situacija u kojoj svi procesi međusobno čekaju da drugi procesi oslobode resurse koji im trebaju.

   Primer: *P1 i P2 koriste resurse R1 i R2. I P1 I P2 trebaju oba resursa da bi završili posao. Scenario – P1 zauzme R1, P2 zauzme R2 I oba procesa čekaju da onaj drugi proces oslobodi resurs.*

2. **LIVELOCK -** Situacija u kojoj dva ili više procesa nisu blokirani ali menjaju stanja tako da se međusobno onemogućavaju da napreduju. Nijedan proces nije blokiran, ali ni ne napreduje u vršenju korisnog rada.

3. **GLADOVANJE** - Situacija u kojoj proces, iako je spreman, nikad ne dobija procesor od raspoređivača.

   Primer: *Procesi P1, P2 i P3 pristupaju resursu R u kritičnoj sekciji. Na početku su sva tri procesa u redu spremnih procesa. U kritičnu sekciju ulazi P1. Nakon izlaska P1 iz kritične sekcije, iz reda spremnih procesa bira se P3. Nakon izlaska P3 iz kritične sekcije, iz reda spremnih procesa bira se P1 i tako u krug. P2 nikad ne dobija procesor, iako je spreman*

ZAHTEVI ZA MEĐUSOBNU ISKLJUČIVOST

1. Mora se sprovesti ako u jednom trenutku samo jedan proces sme da koristi resurs
2. Ne sme se desiti uzajamno blokiranje ili gladovanje tj. proces ne sme beskonačno dugo da čeka na ulazak u kritičnu sekciju
3. U kritičnu sekciju se ulazi bez odlaganja ukoliko nijedan drugi proces nije u njoj
4. Ne mogu se praviti pretpostavke o relativnim brzinama procesa
5. Proces mora izaći iz kritične sekcije u konačnom vemenu

SINHRONIZACIJA

Uvođenje kritične sekcije može da obezbedi sekvencijalan pristup resursima. Za sinhronizaciju rada procesa putem signalizacije koriste se mehanizmi za sinhronizaciju.

KONCEPTI VIŠEG NIVOA ZA SINHRONIZACIJU

1. **SEMAFORI [Dijkstra]** - Deljeni brojač koji ima operacije za umanjivanje i uvećanje, i za čekanje. Implementira se zajedno sa redom. Ako je brojač veći od nule, označava broj procesa koji mogu izvršiti operaciju bez blokiranja, a ukoliko je manji od nule označava broj blokiranih procesa koji čekaju u redu.

   VARIJANTE PREMA TIPU BROJAČA

   - *BROJAČKI* – opisana varijanta u kojoj brojač dobija proizvoljnu celobrojnu vrednost
   - *BINARNI* – specijalna varijanta u kojoj brojač može imati vrednost 0 ili 1 (ponaša se kao mutex)

   VARIJANTE PREMA NAČINU RASPOREĐIVANJA

   - *JAK SEMAFOR* (FIFO red, nema gladovanja procesa)
   - *SLAB SEMAFOR* (nije određeno koji od blokiranih procesa će postati spreman)

   PROBLEMI:

   - Algoritmi često zahtevaju upotrebu više semafora
   - Pozivi metoda nisu upareni, ako redosled metoda nije odgovarajući, program ima neispravan rad. Te greške je teško pronaći.
   - Semafori se koriste istovremeno za obezbeđivanje međusobne isključivosti i sinhronizacije, što bi trebalo da bude različito

2. **USLOVNE PROMENLJIVE –** Uslovna promenljiva koja predstavlja uslov na koji nit može da čeka da se ispuni i za koji nit može da obavesti druge niti da je ispunjen. Veoma koristan mehanizam za sinhronizaciju putem signalizacije između niti.

   Operacije:

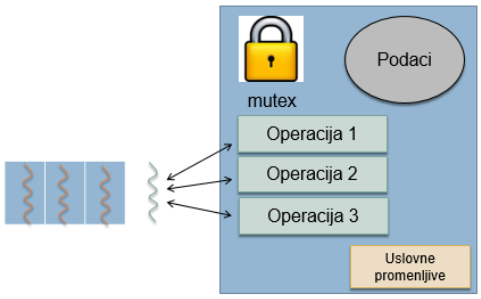
   - wait (nit se blokira dok se uslov ne ispuni)
   - signal (obaveštenje JEDNOJ NITI koja čeka, da je uslov ispunjen)
   - broadcast (obaveštenje SVIM NITIMA koje čekaju, da je uslov ispunjen)

   U C++11 su uslovne promenljive predstavljene klasom *condition_variable*. Metode se moraju pozivati unutar kritične sekcije zaštićene *unique_lock* objektom. Metode:

   - *wait(unique_lock<mutex> l)* - Nit se blokira dok neka druga nit nad ovim *condition_variable* objektom ne pozove *notify*() ili *notify_all*(). Otključava propusnicu pre blokiranja, a čeka na propusncu da bi izašla iz wait. Nakon izlaska iz wait, mora se ponovo proveriti uslov jer je uslov možda promenjen dok je nit dobila propusnicu. Zato wait **UVEK IDE** unutar while petlje.
     LAŽNO BUĐENJE (SPURIOUS WAKEUP) - Nit može izaći iz čekanja čak i ako nije notificirana!

   - notify_one() - U stanje spreman prevodi JEDNU od niti koje su pozvale wait nad ovim CV objektom.
   - notify_all() - U stanje spreman prevodi SVE niti koje su pozvale wait nad ovim CV objektom.

3. **MONITORI -** Softverski patern koji enkapsulira PODATKE, PROPUSNICU, OPERACIJE NAD PODACIMA I USLOVNE PROMENLJIVE ZA SINHRONIZACIJU



   Procesi vide monitor kao BLACK BOX, pristupaju podacima putem metoda, a metode se jednim delom izvršavaju sekvencijalno jer je potrebno zauzeti propusnicu. Metode implementiraju internu logiku sinhronizacije - mogu blokirati nit, a niti ne moraju da vode računa o sinhronizaciji jer monitor garantuje ispravan rad metoda.

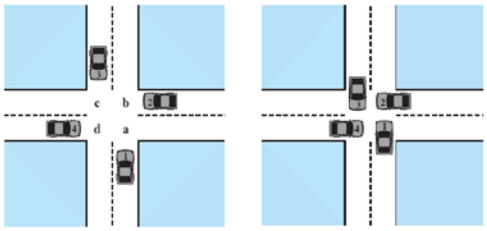
   2 TIPA MONITORA PO SEMANTICI SIGNALIZIRANJA:

   HOARE (1974) - Operacija signal ODMAH aktivira nit, a nit koja je izvršila signalizaciju se blokira

   MESA (1980) - Operacija signal samo prevodi nit u stanje SPREMNA, nit koja je izvršila signalizaciju nastavlja da se izvršava, a signalizirana nit mora da sačeka propusnicu i da ponovo proveri uslov kad se aktivira jer je moglo doći do promene uslova tokom čekanja na propusnicu (C++11 metoda *notify*() radi na ovaj način)

5 - Uzajamno blokiranje

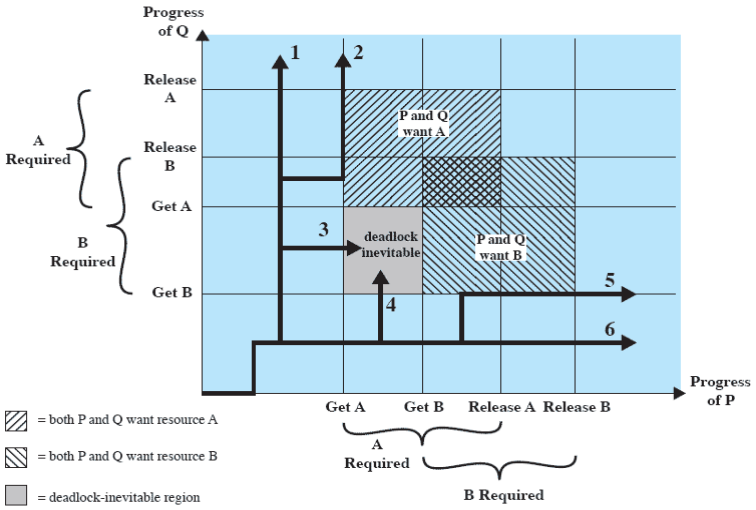
Trajno blokiranje skupa procesa koji se nadmeću za isti skup resursa. Svaki proces u skupu je blokiran čekajući na događaj koji može aktivirati neki blokirani proces iz skupa.



PUTANJE NAPRETKA PROCESA

Horizontalna linija - izvršava se P, a Q čeka

Vertikalna linija – izvršava se Q, a P čeka



MODEL SISTEMA

Uzajamno blokiranje se može dogoditi u sistemu u kojem procesi koriste resurse na sledeći način:

- **ZAHTEV** - Proces pre upotrebe zahteva resurs, a ako on nije 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

UZAJAMNO ISKLJUČIVANJE - Samo jedan proces u datom trenutku može koristiti resurs

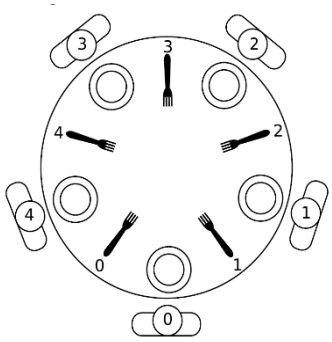
DRŽANJE I ČEKANJE - Proces može držati resurs dok čeka dodeljivanje ostalih resursa

BEZ PREKIDA - Resurs se ne može nasilno oduzeti procesu koji ga drži

KRUŽNO ČEKANJE - Potencijalni rezultat prva 3 uslova - postoji zatvoren krug procesa takav da svaki proces drži bar jedan resurs koji je potreban sledećem procesu u krugu

Mogućnost uzajamnog blokiranja	**Postojanje uzajamnog blokiranja**
1. Uzajamno isključivanje	1. Uzajamno isključivanje
2. Držanje i čekanje	2. Držanje i čekanje
3. Bez prekida	3. Bez prekida
	4. Kružno čekanje

REŠAVANJE PROBLEMA UZAJAMNOG BLOKIRANJA



SPREČAVANJE

Primenjuju se mehanizmi koji eliminišu uslove koji dovode do uzajamnog blokiranja. Sistem se projektuje tako da je isključena mogućnost uzajamnog blokiranja tako što se sprečava pojava jednog od 4 uslova za uzajamno blokiranje.

1. Nije moguće onemogućiti uslov uzajamnog isključivanja. U određenim scenarijima pristupa deljenim resursima procesi moraju pristupati jedan po jedan.

2. **Sprečavanje uslova držanja i čekanja** - može se sprečiti ako proces istovremeno zatraži sve potrebne resurse. Proces ostaje blokiran dok svi traženi resursi ne budu raspoloživi. Ovaj način je neefikasan iz više razloga:

   - proces čeka sve resurse čak i kad može da radi sa onima koje može da zauzme
   - resursi dodeljeni procesu su zauzeti za druge procese, čak i ako ih proces ne koristi
   - može doći do gladovanja procesa ako je uvek bar jedan potreban resurs alociran nekom drugom procesu

3. **Odbacivanje uslova BEZ PREKIDANJA** - Naredba procesu da otpusti svoje resurse ukoliko mu je odbijen zahtev za zauzimanje nekog resursa. Druga varijanta je da ako proces zatraži resurs koji drži drugi blokirani proces, OS prekida drugi proces koji mora da otpusti sve svoje resurse. Primenljivo je samo kod resursa čije stanje se može lako sačuvati i kasnije oporaviti npr. procesorski registri.

4. **Sprečavanje kružnog čekanja** - Ako se uradi linearno slaganje tipova resursa i svakom resursu se dodeli indeks, ako je proces zauzeo resurs R_i onda može zauzeti R_j SAMO AKO JE $j > i$. U ovoj varijanti ne može doći do kružnog čekanja ali može biti neefikasno jer se procesu može nepotrebno zabraniti zauzimanje resursa. Ako programer ne poštuje predviđeni redosled, opet može doći do uzajamnog blokiranja.

ZAKLJUČAK: SPREČAVANJE je suviše restriktivno - imamo slabu iskorišćenost resursa i smanjenu propusnu moć sistema. Pristup koji je manje restriktivan i dozvoljava više konkurentnosti je izbegavanje uzajamnog blokiranja

IZBEGAVANJE

Manje restriktivan pristup koji dozvoljava više konkurentnosti od sprečavanja. Dinamički se vrši raspodela resursa tako da ne dođe do uzajamnog blokiranja. Dozvoljava se postojanje 3 neophodna uslova i prave se razumni izbori tako da se nikad ne stigne u tačku uzajamnog blokiranja. Dinamički se donosi odluka da li će tekući zahtev dodele resursa dovesti do uzajamnog blokiranja.

VARIJANTE:

- Ne pokretati proces ukoliko njegovi zahtevi mogu dovesti do blokiranja
- Ne obrađivati zahtev sa resursom ako dodela resursa može dovesti do blokiranja.

STANJE SISTEMA

Ako imamo sistem sa n procesa i m tipova resursa:

Vektor R = [R_1, R_2, …, R_m] - Ukupna količina svakog resursa u sistemu

Vektor V = [R_1, R_2, …, R_m] - Ukupna količina svakog resursa u sistemu koja nije dodeljena nijednom procesu (raspoloživo)

Matrica zahteva gde svaki element C_{ij} označava zahtev procesa i za resursom j. Mora biti unapred određeno za svaki proces.

$$C = \begin{bmatrix} C_{11} & C_{12} & \dots & C_{1m} \\ C_{21} & C_{22} & \dots & C_{2m} \\ \dots & \dots & \dots & \dots \\ C_{n1} & C_{n2} & \dots & C_{nm} \end{bmatrix}$$

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4

Matrica dodele gde svaki element A_{ij} označava trenutnu dodeljenost resursa j procesu i.

$$A = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1m} \\ A_{21} & A_{22} & \dots & A_{2m} \\ \dots & \dots & \dots & \dots \\ A_{n1} & A_{n2} & \dots & A_{nm} \end{bmatrix}$$

	R1	R2	R3
P1	1	0	0
P2	6	1	2
P3	2	1	1

Za promenljive važi:

- Svi resursi su raspoloživi ili dodeljeni
- Proces ne može zahtevati više od ukupne količine resursa
- Nijednom procesu nije dodeljeno više resursa nego što je zahtevao.

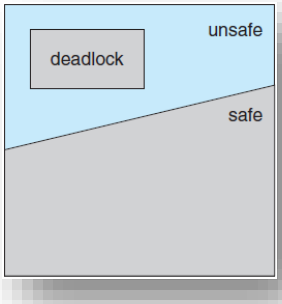
ODBIJANJE INICIJALIZACIJE PROCESA

Proces neće biti pokrenut ako njegovi zahtevi za resursima mogu dovesti do uzajamnog blokiranja. Proces će biti pokrenut ako se mogu zadovoljiti maksimalni zahtevi tekućih procesa uvećanih za zahteve novog procesa.

Ovo je suviše restriktivno jer se pretpostavlja da će svi procesi istovremeno koristiti resurse.

ODBIJANJE DODELE RESURSA

Zasniva se na odluci o dodeli resursa na osnovu stanja sistema.



BEZBEDNO STANJE - Stanje je bezbedno ako sistem može da ispuni zahteve za resursima svakog procesa u nekom redosledu koji ne dovodi do uzajamnog blokiranja. Znači da postoji sekvenca u kojoj procesima treba dodeljivati resurse tako da će svaki proces čekati na resurse konačno vreme.

U bezbednom stanju nema uzajamnog blokiranja. Kada stanje nije bezbedno može doći do uzajamnog blokiranja.

UTVRĐIVANJE BEZBEDNOG STANJA - Stanje je bezbedno ako postoji sekvenca procesa P1,P2, ..., Pn takva da za svaki Pi, resursi koje Pi zahteva mogu biti zadovoljeni sa trenutno raspoloživim resursima plus resursima koje koriste procesi Pj pri čemu je j < i.

ALGORITAM IZBEGAVANJA UZAJAMNOG BLOKIRANJA

Utvrđuje da li postoji sekvenca u kojoj procesi zauzimaju i oslobađaju resurse tako da zahtevi svih procesa budu zadovoljeni. Cilj je da sistem uvek bude u bezbednom stanju. Proces napravi zahtev za resursom, simulira se da je zahtev odobren, proveri se da li je novo stanje u koje bi se prešlo odobravanjem zahteva bezbedno. Ako je bezbedno odobrava se zahtev, ako nije, proces se blokira.

ALGORITAM UTVRĐIVANJA BEZBEDNOG STANJA - BANKAREV ALGORITAM

1. Definišemo vektor W koji predstavlja simulaciju stanja raspoloživih resursa u nekom budućem trenutku. Inicijalno jednak vektoru raspoloživih resursa V. [$W = V$]
2. Pronađemo proces čiji se zahtevi trenutno mogu ispuniti tj. proces za koji važi [$C_{ij} - A_{ij} \leq W_j \; za \; svako \; j = 1, \ \ldots, m$]. Ako nema takvog procesa algoritam se prekida.
3. Simuliramo da je proces i zauzeo resurse, završio rad i oslobodio ih. [$W_j = W_j + A_{ij}$]
4. Vratimo se na korak 2 i ako na ovaj način uspemo da prođemo kroz sve procese i za sve simuliramo dodelu resursa, onda je stanje bezbedno.

PREDNOSTI IZBEGAVANJA UZAJAMNOG BLOKIRANJA

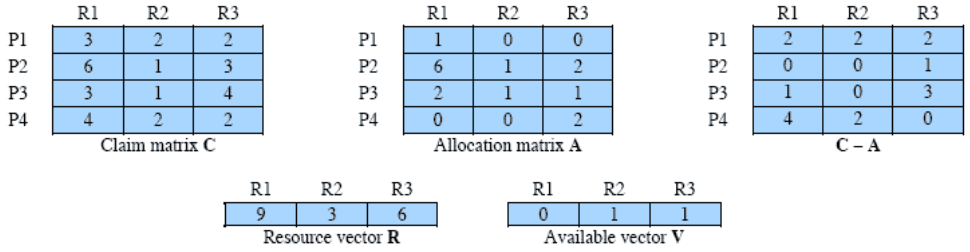
- Manje restriktivno od sprečavanja
- Nije neophodno prekinuti i vratiti unazad procese kao kod otkrivanja uzajamnog blokiranja

MANE IZBEGAVANJA UZAJAMNOG BLOKIRANJA

- 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
- Proces ne može da otpusti resurs

Primer utvrđivanja bezbednog stanja

Inicijalno stanje. Proces P2 može da zadovolji svoje zahteve za resursima



	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	1	0	0
P2	6	1	2
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	2	2	2
P2	0	0	1
P3	1	0	3
P4	4	2	0

C – A

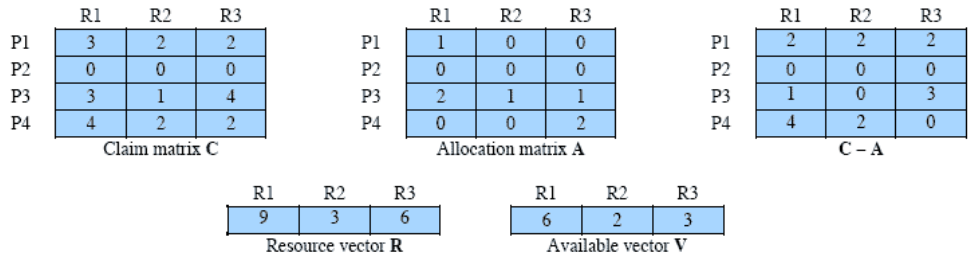
R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
0	1	1

Available vector **V**

Proces P2 se izvršio do kraja. Resursi koje je koristio sada su rapoloživi. Svaki od preostalih procesa ima uslove da se izvrši.



	R1	R2	R3
P1	3	2	2
P2	0	0	0
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	1	0	0
P2	0	0	0
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	2	2	2
P2	0	0	0
P3	1	0	3
P4	4	2	0

C – A

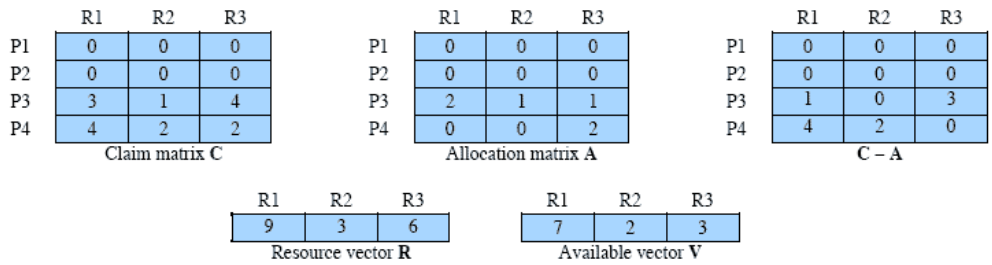
R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
6	2	3

Available vector **V**

Proces P1 se izvršio do kraja. Resursi koje je koristio sada su rapoloživi.



	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	1	0	3
P4	4	2	0

C – A

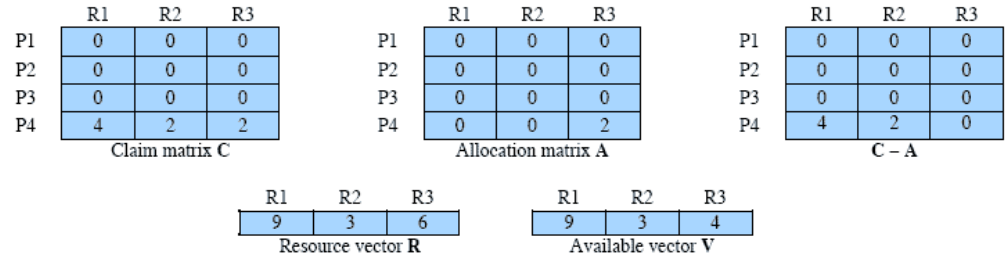
R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
7	2	3

Available vector **V**

Proces P3 se izvršio do kraja. Resursi koje je koristio sada su rapoloživi. Znači da je inicijalno stanje bilo bezbedno.



	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	4	2	0

C – A

R1	R2	R3
9	3	6

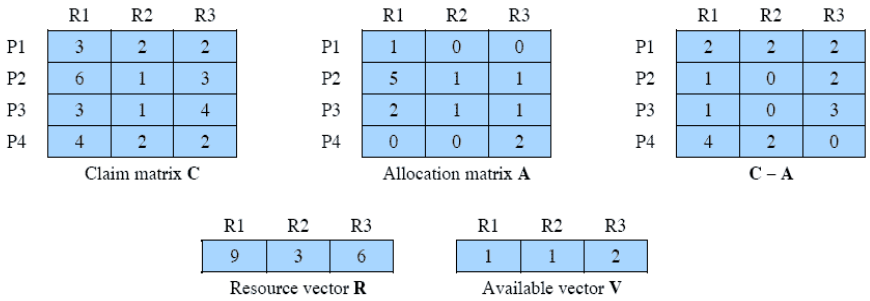
Resource vector **R**

R1	R2	R3
9	3	4

Available vector **V**

Primer utvrđivanja nebezbednog stanja

Inicijalno stanje. Proces P1 napravi zahtev za još jednom jedinicom resursa R1 i R3. Proverićemo u kakvo bi stanje došli ako odobrimo ovaj zahtev.



	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	1	0	0
P2	5	1	1
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	2	2	2
P2	1	0	2
P3	1	0	3
P4	4	2	0

C – A

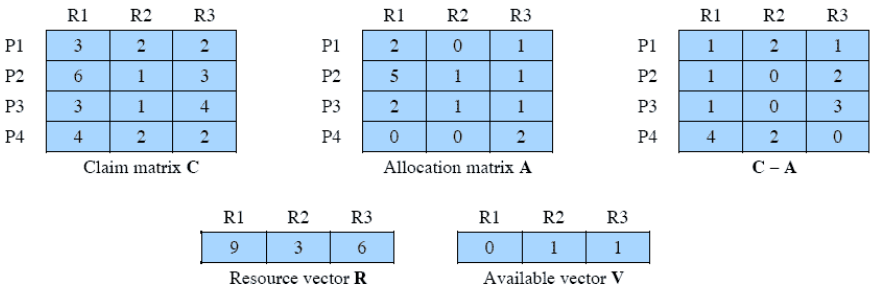
R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
1	1	2

Available vector **V**

Stanje u koje bi došli odobravanjem zahteva procesu P1 je nebezbedno. Nijedan od procesa ne bi mogao da ispuni svoje zahteve. Zato procesu P1 ovaj zahtev neće biti odobren.



	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	2	0	1
P2	5	1	1
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	1	2	1
P2	1	0	2
P3	1	0	3
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
0	1	1

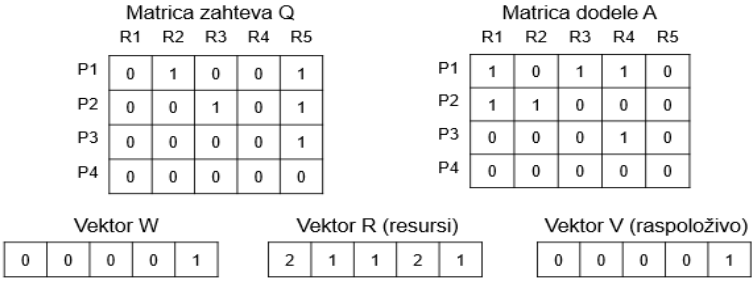
Available vector **V**

OTKRIVANJE

Kada se desi uzajamno blokiranje, preduzimaju se akcije za oporavak. Ne ograničava se pristup resursima već se procesima dodeljuju zahtevani resursi. Povremeno se vrši provera da li je došlo do kružnog čekanja i ako jeste vrši se oporavak sistema.

ALGORITAM OTKRIVANJA UZAJAMNOG BLOKIRANJA

Cilj je da označimo procese koji nisu uzajamno blokirani, a oni koji ostanu su uzajamno blokirani.



Matrica zahteva Q

	R1	R2	R3	R4	R5
P1	0	1	0	0	1
P2	0	0	1	0	1
P3	0	0	0	0	1
P4	0	0	0	0	0

Matrica dodele A

	R1	R2	R3	R4	R5
P1	1	0	1	1	0
P2	1	1	0	0	0
P3	0	0	0	1	0
P4	0	0	0	0	0

Vektor W

0	0	0	0	1

Vektor R (resursi)

2	1	1	2	1

Vektor V (raspoloživo)

0	0	0	0	1

1. Inicijalizujemo vektor W da bude jednak V
2. Označimo sve procese koji u matrici dodela imaju sve nule (oni nemaju dodeljene nikakve resurse pa nisu uzajamno blokirani) *(Označavamo proces P4)*
3. Pronađemo neoznačen proces čiji su zahtevi manji ili jednaki od W, a ako nma takvog procesa, prekidamo algoritam *(P3 ispunjava uslov)*
4. Označimo pronađeni proces i dodamo njegov red iz matrice dodele u vektor W (simuliramo da se proces izvršio i oslobodio resurse) *(Označavamo P3)*
5. Vratimo se na 3. korak (Nema više procesa koji ispunjavaju uslov. Neoznačeni su ostali P1 i P2, što znači da su uzajamno blokirani)

STRATEGIJE OPORAVKA OD UZAJAMNOG BLOKIRANJA

Prekini sve uzajamno blokirane procese - Najjednostavnije rešenje, ali velika cena ovakvog pristupa jer se poništava sve što je proces do tada uradio

Vrati sve uzajamno blokirane procese na neku raniju kontrolnu tačku i ponovo ih pokreni - Može doći do uzajamnog blokiranja, ali zbog nedeterminantnosti konkurentne obrade, u ponovnom izvršavanju možda ne bude blokiranja

Redom prekidaj sve uzajamno blokirane procese dok više ne bude blokiranja - Redosled prekidanja treba da bude na osnovu kriterijuma :

- *prioritet*
- *utrošak procesorskog vremena do sada*
- *predviđeno preostalo vreme rada*
- *tip resursa koje je proces koristio*
- *količina resursa potrebna za završetak rada*

Redom oduzimaj resurse od procesa i dodeljuj ih drugim procesima dok više ne bude blokiranja - Izabrati resurse i procese tako da se minimizuju troškovi. Proces kojem je oduzet resurs mora se vratiti na neko prethodno stanje u kojem je bio pre nabavke resursa. Potrebno je sprečiti gladovanje procesa - ne smeju se resursi uvek oduzimati istom procesu.

STRATEGIJE ZA UZAJAMNO BLOKIRANJE U SAVREMENIM OS

Tipično ne pružaju podršku za zaštitu već se ignoriše mogućnost pojave uzajamnog blokiranja. Ne isplati se angažovati resurse za sprečavanje/izbegavanje/otkrivanje uzajamnog blokiranja. OS se implementira sa namerom da ne ulazi u deadlock, a korisnički programi su odgovornost programera. Na ovom principu funkcionišu Windows i Linux.

6 – Upravljanje memorijom

Da bi se podržalo multiprogramiranje, OS mora da čuva više procesa u glavnoj memoriji. Memorija mora da se deli između više procesa. Radna memorija je veliki linearni niz bajtova, gde svaki bajt ima svoju adresu.

IZVRŠAVANJE INSTRUKCIJE - Procesor preuzima instrukcije iz memorije na osnovu vrednosti brojača instrukcija. Instrukcija se dekodira i može da uzrokuje dobavljanje operanada iz memorije. Nakon izvršenja instrukcije nad operandima, može biti potrebno smestiti rezultat u memoriju.

POTREBA ZA UPRAVLJANJEM MEMORIJOM

Lokacija podataka i instrukcija procesa stvorili su potrebu za upravljanjem memorijom. Glavna memorija i registri procesora su jedina memorijska skladišta kojima procesor može direktno da pristupa. U trenutku izvršavanja instrukcije, instrukcija i podaci koje koristi moraju biti u jednom od ova dva skladišta (ili u procesorskoj keš memoriji).

RADNA vs MASOVNA MEMORIJA

Upravljanje memorijom podrazumeva prebacivanje blokova podataka između masovne i glavne memorije. Cilj je optimizovati trenutke i količinu podataka koji se razmenjuju, jer su memorijski U/I uređaji spori a kapacitet radne memorije je značajno ograničen.

Radna memorija	**Masovna memorija**
Manjeg kapaciteta	Većeg kapaciteta
Brža	Sporija
Skuplja	Jeftinija
Nije trajna	Trajna

ZAHTEVI ZA UPRAVLJANJE MEMORIJOM

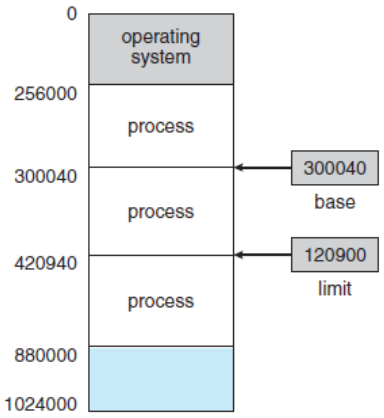
Računarski hardver i OS moraju da podrže:

1. **RELOKACIJU** – U toku izvršavanja proces može da menja lokaciju u glavnoj memoriji. Ne može se unapred znati u kojem delu memorije će se proces nalaziti. Procesi (ili delovi procesa) se po potrebi ubacuju/izbacuu iz glavne memorije. Proces može biti zamenjen na disk (swapping) i biti vraćen na drugu lokaciju u glavnoj memoriji (relociranje).
2. **DELJENJE MEMORIJE** – Više procesa treba da pristupa istim delovima memorije (npr. Različite instance istog programa treba da dele programski kod). Sistem za upravljanje memorijom treba da omogući kontrolisani pristup deljenim područjima u memoriji bez ugrožavanja zaštite.
3. **ZAŠTITA** – Proces ne sme da pristupa bez dozvole lokacijama u koje su smešteni drugi procesi. Zbog relokacije, lokacija procesa nije unapred određena i nepromenjiva. Zato je nemoguće zaštitu sprovesti u vreme prevođenja (compile time) već zaštita mora biti sprovedena u toku izvršavanja (runtime). Hardver procesora obezbeđuje zaštitu.

HARDVERSKA PODRŠKA ZA RELOKACIJU I ZAŠTITU

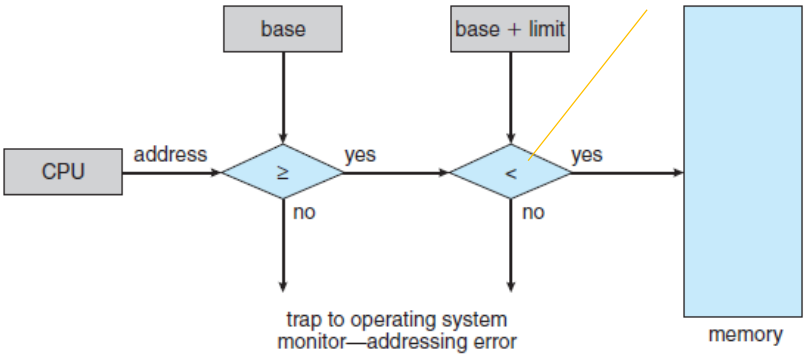
Svaki proces mora da ima odvojen memorijski prostor. Potrebno je definisati skup adresa kojima proces može da pristupa. Za to služe 2 procesorska registra:

- ***base*** – početna adresa na koju je proces smešten
- ***limit*** – poslednja adresa (počevši od 0) kojoj proces pristupa



Adresa početka bloka u kojem je proces trenutno

Dobija se memorijska lokacija kojoj je potrebno pristupiti

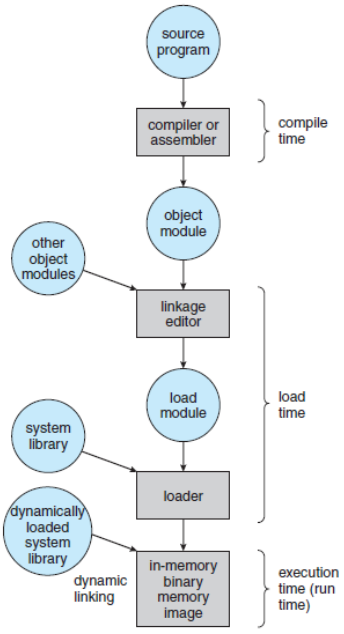


Provera zaštite – proces sme da referencira samo dozvoljene lokacije

KREIRANJE I IZVRŠAVANJE PROGRAMA

Program je binarni fajl na disku. Da bi se izvršio, ubacuje se u glavnu memoriju. U toku izvršavanja referencira stvarne adrese u memoriji. Izvorni kod ne koristi direktno adrese, nego promenljive.

Kada se definišu stvarne adrese?

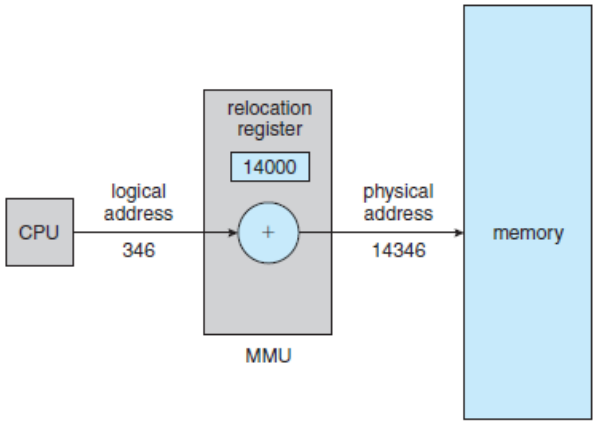


DEFINISANJE ADRESA U PROGRAMU

- **U TRENUTKU KOMPAJLIRANJA** – Mora se u trenutku kompajliranja znati gde će program biti smešten u memoriji. U samom programu se upišu stvarne memorijske adrese kojima će pristupati. Ako treba podržati multiprogramiranje, nije moguće u trenutku kompajliranja znati lokaciju programa.
- **U TRENUTKU UČITAVANJA** – Stvarne adrese se upisuju tek u trenutku učitavanja programa u memoriju. Nije upotrebljivo ako u toku izvršavanja program može da bude pomeran na druge lokacije u memoriji.
- **U TRENUTKU IZVRŠAVANJA** – Ovo je jedina varijanta koja podržava relokaciju. Ako nije unapred definisana lokacija na koju će program biti učitan i ako u toku izvršavanja programa ta lokacija može da se menja, tek u trenutku izvršavanja svake instrukcije se može odrediti stvarna adresa kojoj je potrebno pristupiti. Potrebno je da hardver procesora podrži utvrđivanje adrese u trenutku izvršavanja. Savremeni OS koriste ovu metodu.

LOGIČKE I FIZIČKE ADRESE

Adresa upisana u programu je logička (relativna) adresa. Stvarna adresa kojoj se pristupa je fizička (apsolutna adresa). Logičke adrese su definisane relativno u odnosu na blok u koji je proces trenutno smešten. U toku izvršavanja, ove relativne adrese se prevode u fizičke adrese u memoriji. Hardver procesora (jedinica za upravljanje memorijom) prevodi relativne u fizičke adrese.



NAČINI UPRAVLJANJA MEMORIJOM

1. **Alociranje susednih memorijskih lokacija**
   a. Deljenje memorije na particije
   b. Partnerski sistem
2. **Straničenje**
3. **Segmentacija**

DELJENJE MEMORIJE NA PARTICIJE

Klasičan način upravljanja memorijom, korišćen pre pojave virtuelne memorije. Kompletan proces smešta u uzastopne memorijske lokacije. Danas se slabo koristi ali je dobar uvod za razumevanje koncepta virtuelne memorije.

MANE: Problem interne I eksterne fragmentacije. Na primer, kod dinamičkog deljenja na particije i korišćenja algoritma 'prvi odgovarajući' za N alociranih blokova još N/2 blokova će biti izgubljeno zbog eksterne fragmentacije. Ne koristi se u savremenim OS.

FIKSNO DELJENJE NA PARTICIJE JEDNAKE VELIČINE

Deljenje glavne memorije u delove (particije) tj. blokove fiksne veličine. Svaki proces čija veličina je manja ili jednaka veličini particije može da se smesti u bilo koju raspoloživu particiju. OS može da zameni proces u particiji – ako treba ubaciti novi proces u glavnu memoriju a sve particije su zauzete.

PROBLEMI:

- Program može da bude prevelik za particiju.
- Bez obzira na veličinu programa, zauzima se cela particija. To dovodi do neefikasnog korišćenja glavne memorije **– INTERNA FRAGMENTACIJA**.

FIKSNO DELJENJE NA PARTICIJE NEJEDNAKE VELIČINE

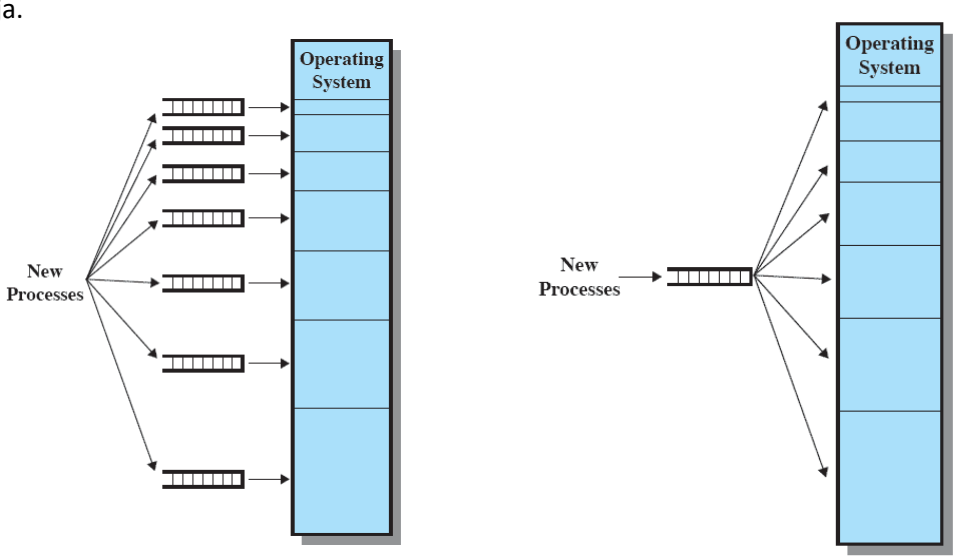
Umanjuje pomenute probleme ali ih ne rešava potpuno. Ako pogledamo sliku sa desne strane videćemo da programi manji od 16M mogu da se smeste u memoriju. Manji programi mogu da se smeste u manje particije čime se smanjuje interna fragmentacija.

FIKSNO DELJENJE - ALGORITAM SMEŠTANJA

Ako su particije jednake veličine, nije bitno koja će particija biti zauzeta. Zauzima se prva raspoloživa particija.

Ako su particije nejednake veličine, procesu se dodeljuje najmanja particija u koju može da stane. Za svaku particiju postoji red čekanja. Ovo je neefikasno jer proces može da čeka I kada postoji slobodna particija u koju može da se smesti.

Efikasnija varijanta je da postoji jedan red čekanja za sve particije I da se uvek bira najmanja raspoloživa particija.



NEDOSTACI FIKSNOG DELJENJA NA PARTICIJE

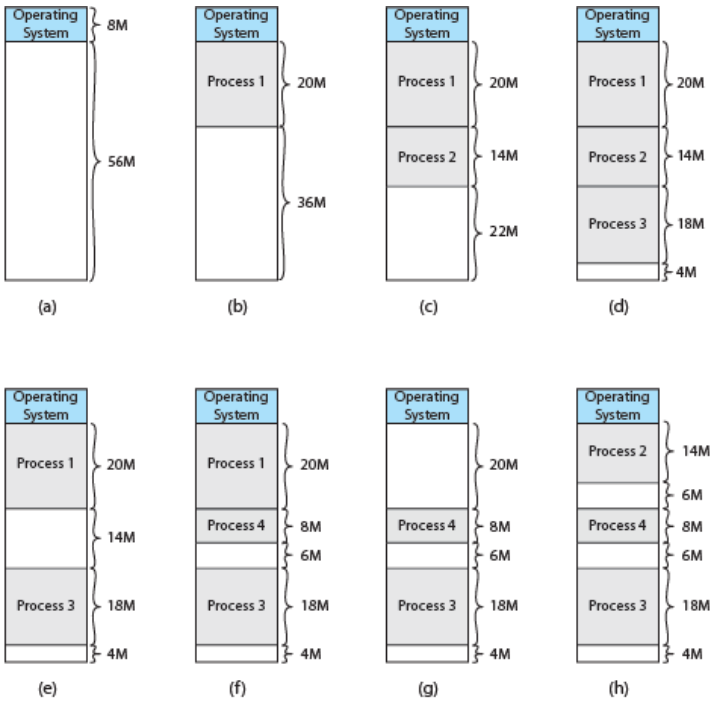
- **Ograničen broj aktivnih procesa** – predefinisanim brojem particija ograničeno je koliko procesa može biti u radnoj memoriji.
- **Veliki broj malih procesa neće efikasno koristiti memoriju** – značajni delovi particija ostaće prazni

DINAMIČKO DELJENJE NA PARTICIJE

Promenljiv broj particija, particije su promenljive veličine a procesu se dodeljuje tačno onoliko memorije koliko zahteva.

EKSTERNA FRAGMENTACIJA – Mnogo malih rupa u memoriji. Dovoljno slobodne memorije ali nije u povezanim lokacijama. Proces ne može da se smesti iako ukupno ima dovoljno memorije za smeštanje.

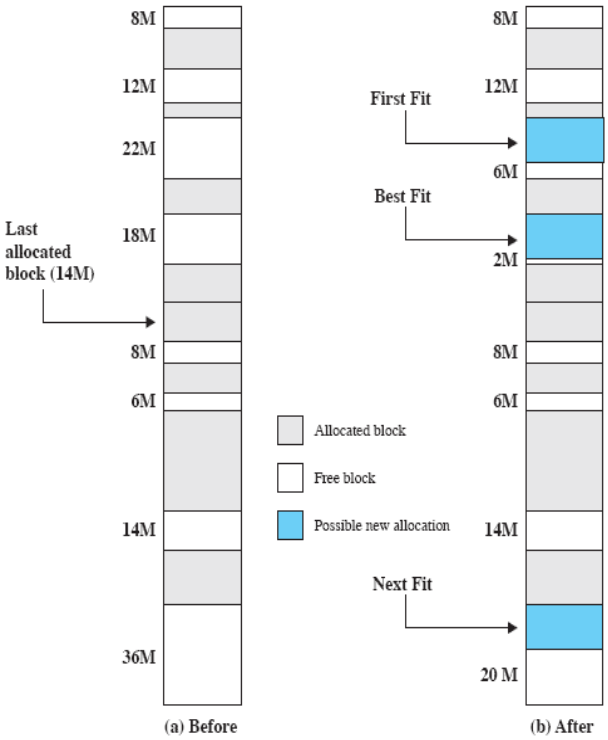
Ovo se može rešiti sažimanjem – s vremena na vreme se pomeraju procesi tako da budu u susednim lokacijama. Sva slobodna memorija se tada nalazi na kraju u jednom bloku. Problem je što to traje dugo I troši procesorsko vreme.



ALGORITAM SMEŠTANJA

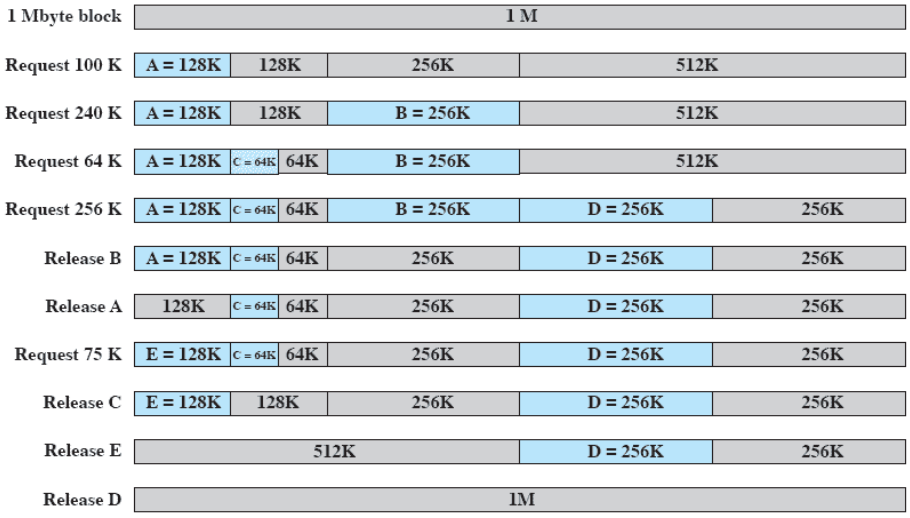
OS mora da odluči u koji blok da smesti proces.

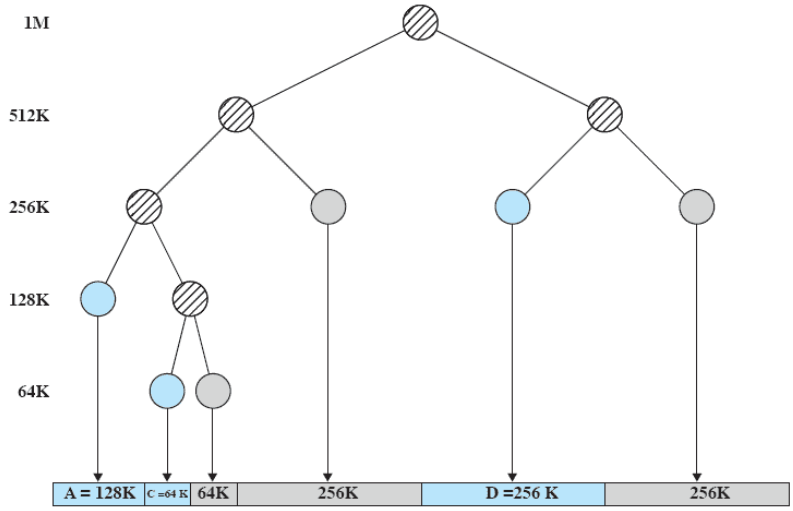
- **NAJBOLJI ODGOVARAJUĆI** – Bira se slobodan blok koji je po veličini najbliži zahtevu. Najmanji mogući deo bloka ostaje neiskorišćen. Sažimanje mora češće da se radi nego kod ostalih algoritama jer ostaje puno malih blokova. Ovaj algoritam ima najlošije performanse.
- **PRVI ODGOVARAJUĆI** – Prolazi se kroz memoriju od početka. Pronalazi se prvi slobodan blok dovoljno velik da se proces smesti. Može da rezultira u mnogo malih slobodnih particija u početnom delu memorije, koje se uvek moraju iznova pretraživati. Svakako, ovo je najjednostavniji i najbrži algoritam.
- **SLEDEĆI ODGOVARAJUĆI** – Prolazi se kroz memoriju počevši od lokacije na kojoj je izvršeno poslednje ubacivanje. Često procese smešta na kraj memorije gde je najveći slobodan blok. Ovaj blok će biti izdeljen na male fragmente. Potrebno je sažimanje da bi se opet dobio veliki slobodan blok na kraju memorije.



PARTNERSKI SISTEM

Celokupna raspoloživa memorija se tretira kao jedan blok veličine 2^U. Ako se zahteva blok veličine s, gde je $2^{U-1} < s <= 2^U$, ceo blok se alocira. U suprotnom, blok se deli na dva jednaka bloka (partnera). Postupak se rekurzivno ponavlja dok god se ne stvori najmanji blok koji je veći ili jednak *s*.





PREDNOSTI:

- Zauzima se blok koji najbolje odgovara veličini procesa
- Jednostavno se sažimaju susedni slobodni blokovi

NEDOSTACI:

- Interna fragmentacija.

STRANIČENJE

U savremenim OS se kao osnovni način upravljanja memorijom koriste straničenje I segmentacija.

Memorija se deli na male delove jednake veličine. Proces se takođe deli na male delove jednake veličine. Delovi memorije se zovu **OKVIRI** a delovi procesa **STRANICE**. Veličina okvira I stranice je jednaka. Stranice procesa se mogu dodeljivati okvirima, koji ne moraju biti susedni.

Eksterne fragmentacije nema dok interne fragmentacije ima samo u poslednjoj stranici procesa.

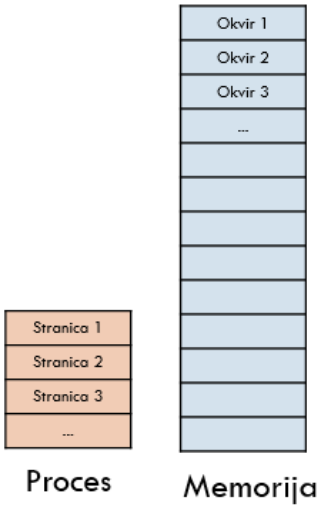
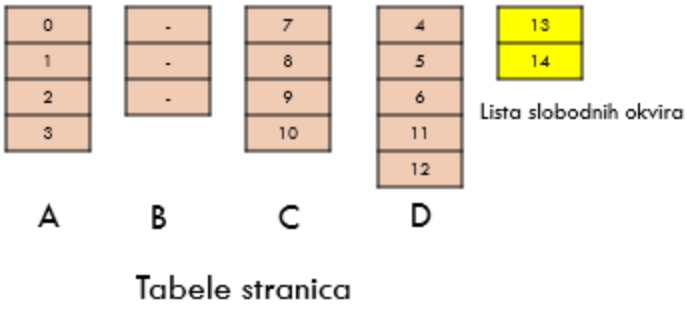
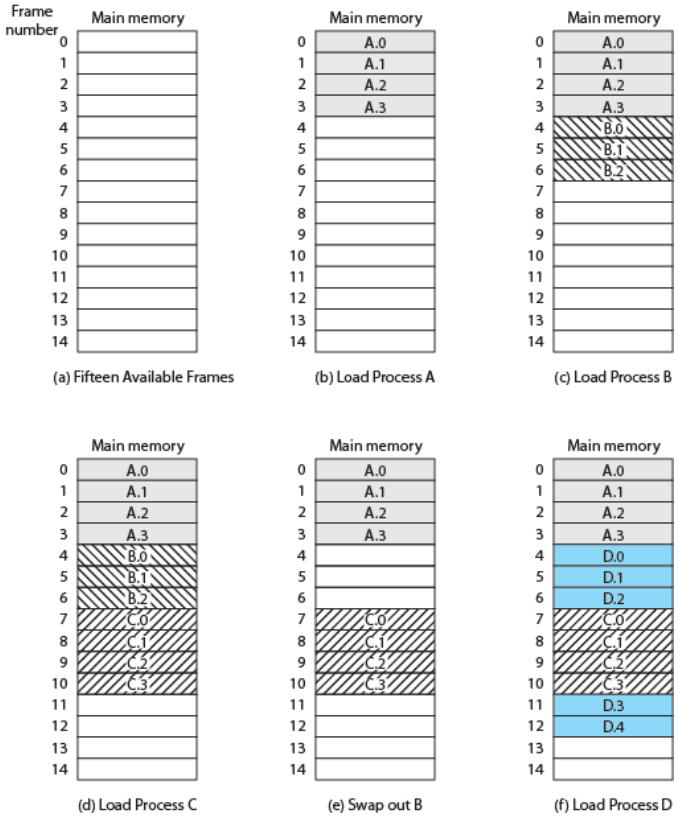


TABELA STRANICA

OS za svaki proces održava tabelu stranica. Za svaku stranicu tabela sadrži broj okvira u koji je smeštena. Procesor mora da zna da pristupi tabeli stranica procesa. Na osnovu broja stranice i relativnog pomeraja u okviru stranice, formira se apsolutna adresa.



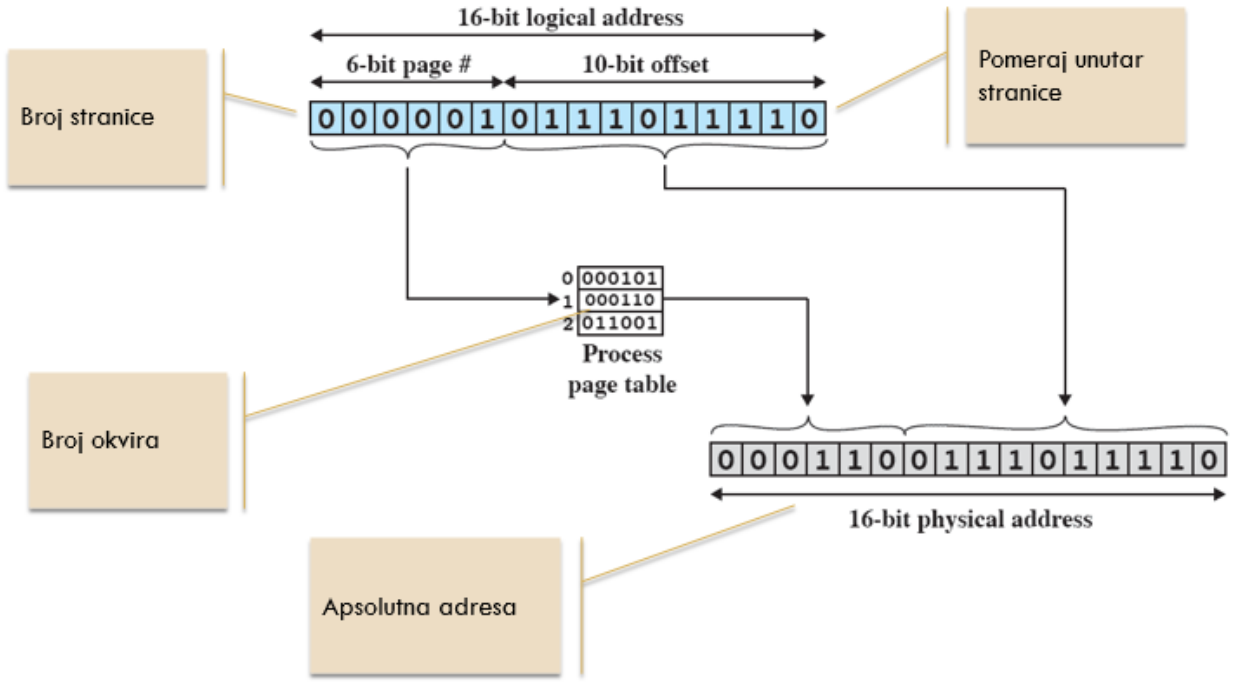


LOGIČKE ADRESE

RELATIVNA ADRESA – Pomeraj u odnosu na početak.

LOGIČKA ADRESA – Broj stranice i pomeraj unutar nje

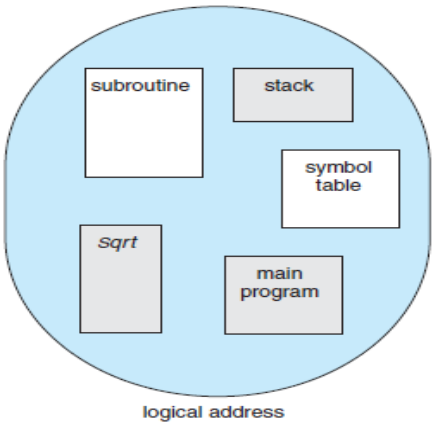
Ako se odredi da je veličina stranice stepen broja 2, tada su relativna i logička adresa iste.



Prevođenje logičke adrese u fizičku kod straničenja

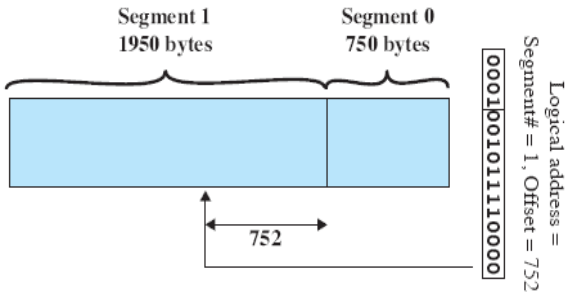
SEGMENTACIJA

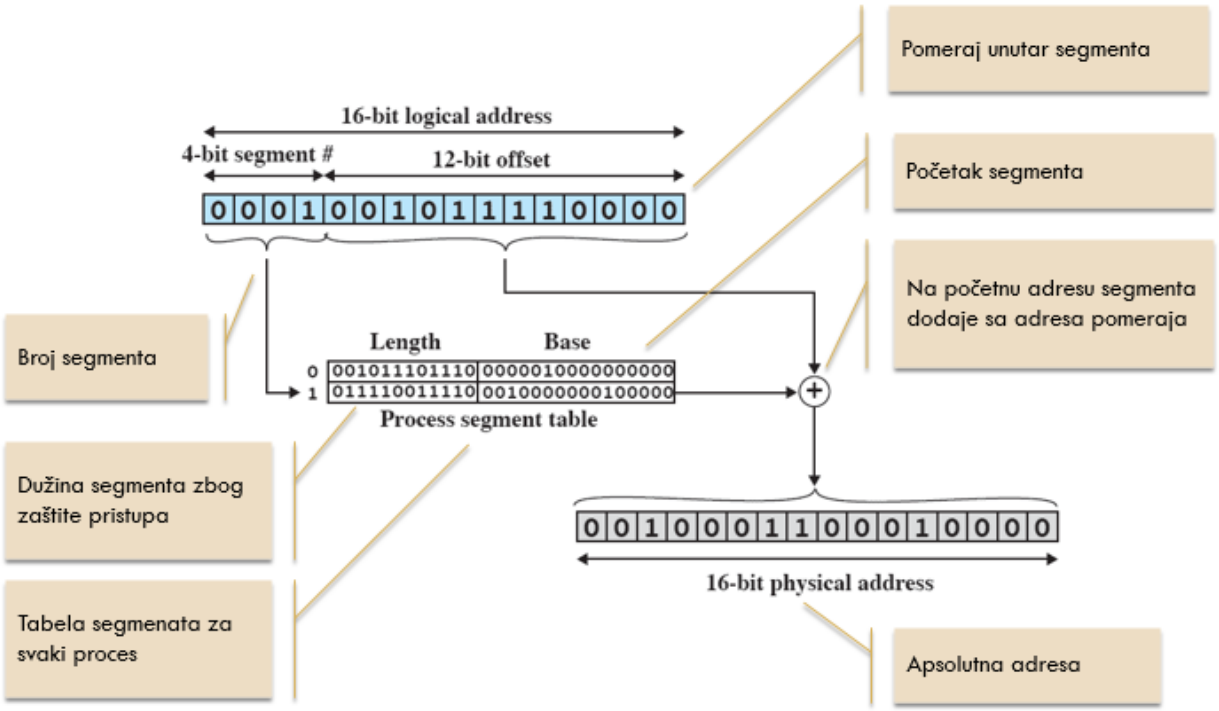
Programer ne vidi program kao niz adresiranja linearno organizovanih memorijskih lokacija. On vidi program kao skup segmenata, gde su segmenti različite dužine i gde postoje ograničenja za maksimalnu veličinu.



Segmentacija je način upravljanja memorijom koji se bazira na posmatranju memorije iz ugla programera. Logički adresni prostor se deli u segmente promenljive veličine. Kompajler generiše odvojene segmente za različite delove koda. Ovo je slično dinamičkom deljenju na particije, ali nema interne fragmentacije.

Adresa se sastoji iz 2 dela: broj segmenta i pomeraj unutar segmenta





Prevođenje logičke adrese u fizičku kod segmentacije

7 – Virtuelna memorija

KARAKTERISTIKE UPRAVLJANJA MEMORIJOM

Memorijske reference su logičke adrese koje se dinamički prevode u fizičke adrese u toku izvršavanja. Proces može da bude ubacivan i izbacivan u/iz glavne memorije u toku izvršavanja. Svaki put može biti smešten u različit deo memorije.

Proces se može izdeliti na manje delove koji ne moraju zauzimati uzastopne lokacije u memoriji.

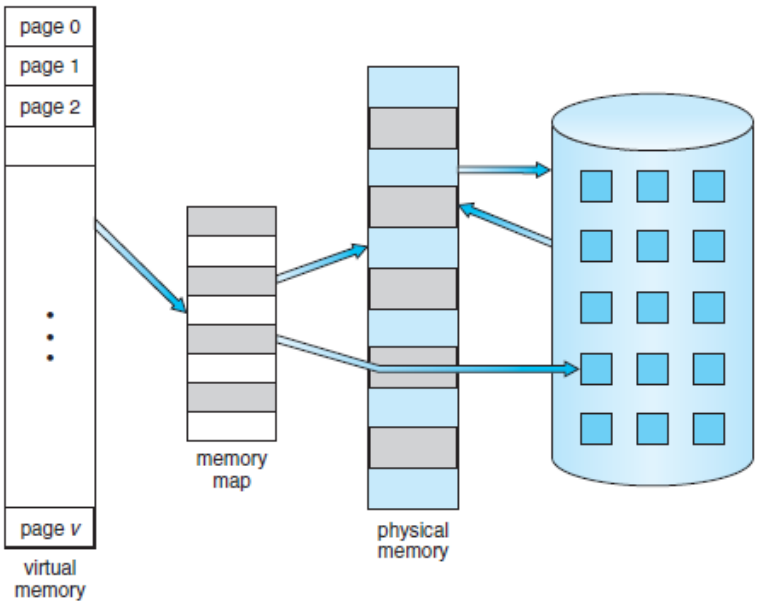
Ako su **obe** karakteristike prisutne:

- Tada nije neophodno da svi delovi procesa istovremeno budu u glavnoj memoriji u toku izvršavanja
- Ne koriste se svi delovi procesa uvek I jednako često
- Nema razloga da delovi koji nisu u upotrebi zauzimaju memoriju

VIRTUELNA MEMORIJA

Youtube link

Deo diska (može biti fajl ili posebna particija) namenjen za smeštanje delova procesa koji se trenutno izvršavaju. Omogućuje nam efikasno multiprogramiranje, a sa stanovišta korisnika umanjuje ograničenja glavne memorije.



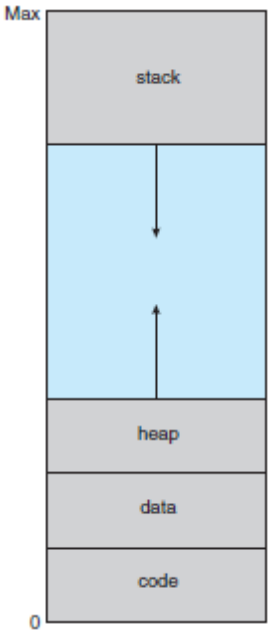
Prednosti nove strategije upravljanja memorijom

1. **Veći broj procesa može da se održava u glavnoj memoriji** – od svakog procesa se učitava samo deo, pa više različitih procesa može da bude učitano. Takođe, sa većim brojem procesa veća je I šansa da će u svakom trenutku bar neki od njih biti u stanju SPREMAN.

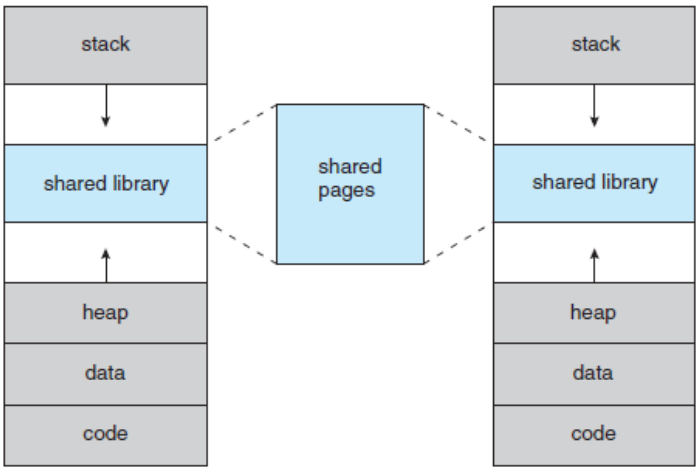
2. **Proces može biti veći od ukupne glavne memorije**

3. Sa stanovišta korisnika, nova strategija upravljanja memorijom je transparentna – korisnik vidi virtuelni adresni prostor sa kontinualnim logičkim adresama koje kreću od 0.

4. ***HEAP*** I ***STACK*** memorija dodeljena procesu dinamički se menja – prostor za ovu memoriju je deo virtuelnog adresnog prostora, I zahteva fizičke stranice u memoriji samo ako je popunjen.



5. **Deo memorije može biti deljen od strane više procesa** – ista tranica se mapira u virtuelni prostor više procesa. Primenjuje se za sistemske biblioteke koje deli više procesa, komunikaciju između procesa putem deljene memorije I deljenje istog koda od strane više instanci istog procesa.

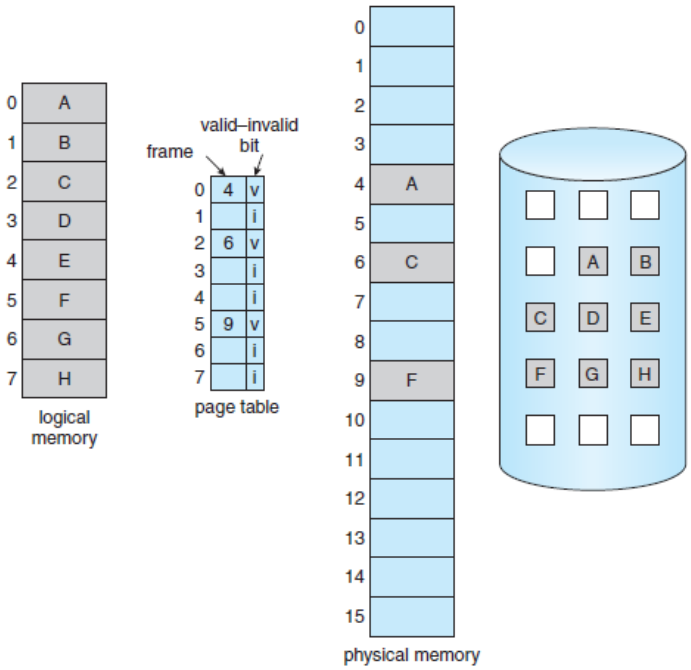


IZVRŠAVANJE PROCESA

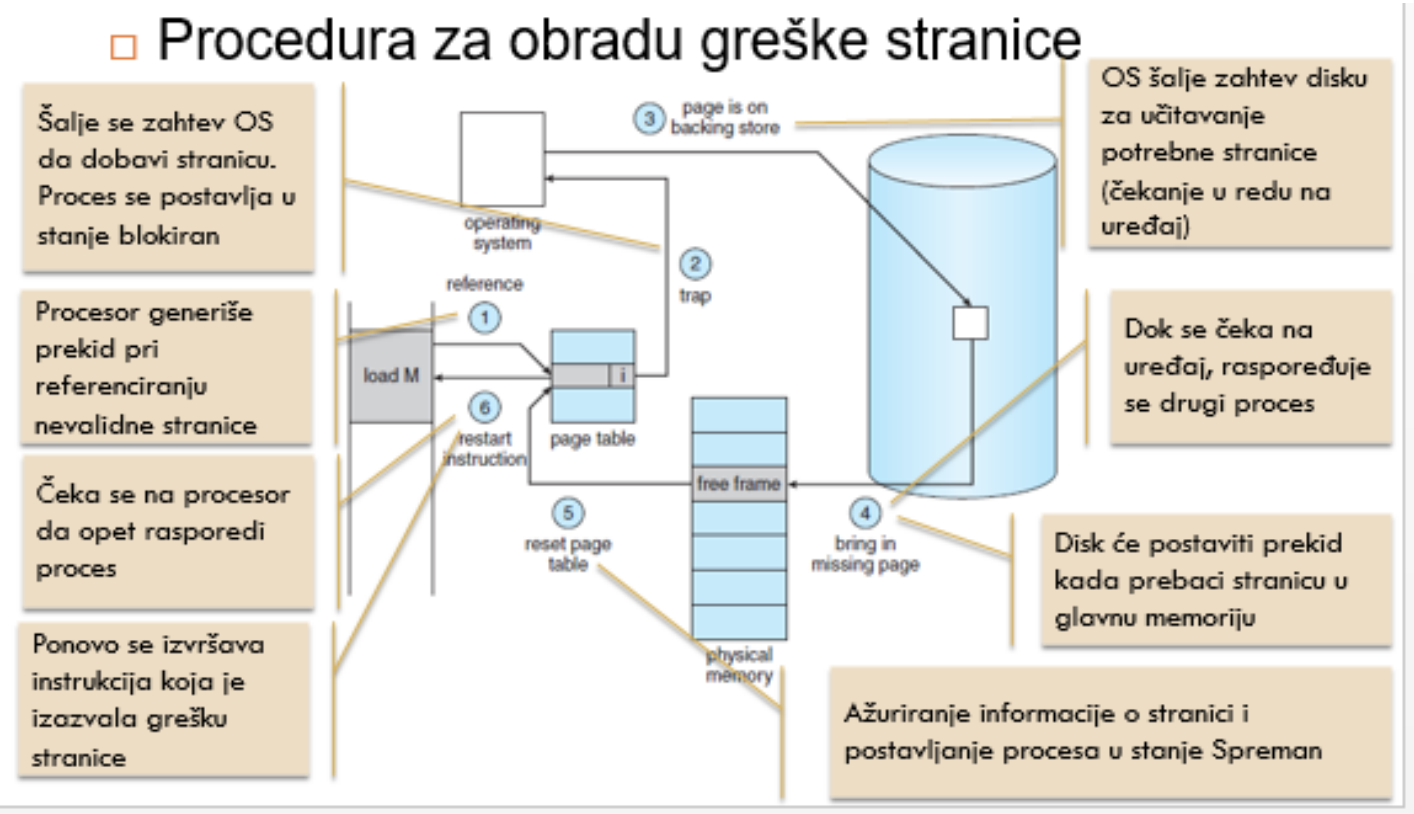
REZIDENTNI SKUP – delovi procesa koji su u glavnoj memoriji u određenom trenutku. Postoji indikacija za svaku stranicu da li je u memoriji.

Ako se pri izvršavanju procesa referencira stranica koja je u glavnoj memoriji, pristupa se traženoj lokaciji I nastavlja izvršavanje.

Ako se referencira stranica koja nije u glavnoj memoriji dolazi do greške stranice (*eng. Page fault*). Hardver utvrđuje da je za tu stranicu postavljen *invalid* bit. Šalje se zahtev operativnom sistemu da dobavi potrebnu stranicu.



Procedura za obradu greške stranice



EFIKASNOST VIRTUELNE MEMORIJE

- Ako potrebni deo procesa nije u glavnoj memoriji, mora se dobaviti sa diska.
- Ako nema mesta u glavnoj memoriji za novi deo, deo nekog procesa se mora izbaciti iz glavne memorije

Efikasnost zavisi od verovatnoće pojave greške stranice. Postoje 3 grupe operacija pri obradi greške stranice:

1. Obrada prekida za grešku stranice
2. Dobavljanje stranice sa diska (***vremenski zahtevna***)
3. Restartovanje instrukcije

Vreme pristupa direktno je proporcionalno ućestalosti greške stranice. Ako svako hiljadito referenciranje izaziva grešku, dobija se usporenje od 40 puta u odnosu na vreme potrebno za pristup samo radnoj memoriji. Kako bi imali dobre performanse, da bi usporenje bilo manje od 10%, moramo imati manje od jednog na 400.000 referenciranja koje bi izazvalo grešku stranice.

BRBLJANJE (*Thrashing*) – situacija kada sistem provodi najveći deo vremena razmenjujući delove između stvarne I virtuelne memorije umesto u izvršavanju instrukcija. Izbegava se tako što OS izbacuje stranicu za koju utvrdi da je najmanja verovatnoća da će biti uskoro referencirana. Predviđanje se vrši na osnovu nedavne istorije.

PRINCIP LOKALNOSTI

Program I reference podataka unutar procesa teže da se grupišu. Tokom dužeg perioda menjaju se delovi procesa koji se koriste. U kratkom intervalu procesor uglavnom radi sa malim I ograničenim skupom adresa – samo nekoliko delova procesa je potrebno u kratkom vremenskom periodu.

Razlozi za postojanje lokalnosti u programima:

1. **Izvršavanje programa je skoro uvek sekvencijalno** – naredna adresa je najčešće ona koja sledi trenutnoj, izuzetak su grananja I pozivi funkcija
2. **U kratkom periodu program je lokalizovan na određenu funkciju** – u funkciji se pristupa ograničenom skupu njenih parametara, lokalnih promenljivih I podskupa globalnih promenljivih. Retko u programu postoji dugačak uzastopan niz poziva funkcija. Uglavnom je program ograničen na uzak nivo dubine poziva funkcija.
3. **Zbog iterativnih delova koda** – petlje se uglavnom sastoje od malog broja instrukcija koje se ponavljaju mnogo puta. Obrada je ograničena na mali susedni deo adresa.
4. **Često se podaci čuvaju u strukturi koja čuva podatke o uzastopnim lokacijama** – podaci kojima se pristupa su blisko locirani, npr. Rad sa nizovima.

Zato što postoji lokalnost, moguće je napraviti pretpostavku koji delovi procesa će biti potrebni uskoro I tako izbeći ‘brbljanje’. Iz tog razloga, zbog principa lokalnosti, virtuelna memorija je efikasna.

PODRŠKA ZA VIRTUELNU MEMORIJU

HARDVERSKA – Hardver procesora mora da podrži adresiranje stranica/segmenata

SOFTVERSKA – OS mora da pomera stranice/segmente između glavne I sekundarne memorije

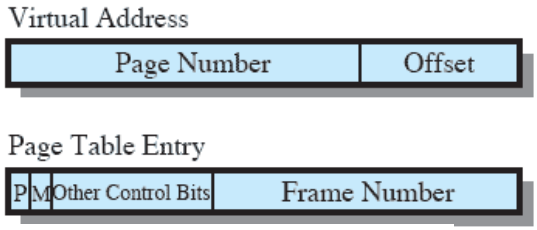
STRANIČENJE

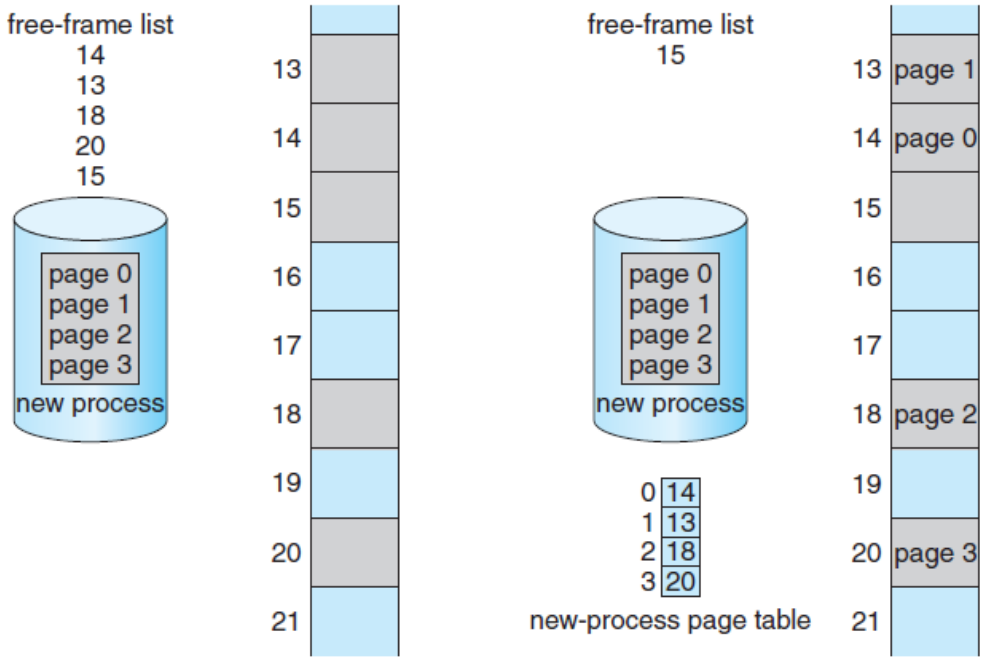
Proces je podeljen na stranice. Sve stranice su smeštene na disku, dok je jedan deo stranica smešten u glavnoj memoriji.

TABELA STRANICA – Evidencija u kojim okvirima su stranice smeštene. Svaki proces ima svoju tabelu.

STAVKA TABELE STRANICA – sadrži podatke potrebne za evidenciju stranice

- Broj okvira ako je stranica u glavnoj memoriji
- P – da li je stranica prisutna u memoriji
- M – da li je stranica menjana otkako je učitana u glavnu memoriju. Ako jeste, potrebno je pre izbacivanja upisati izmenu na disk.

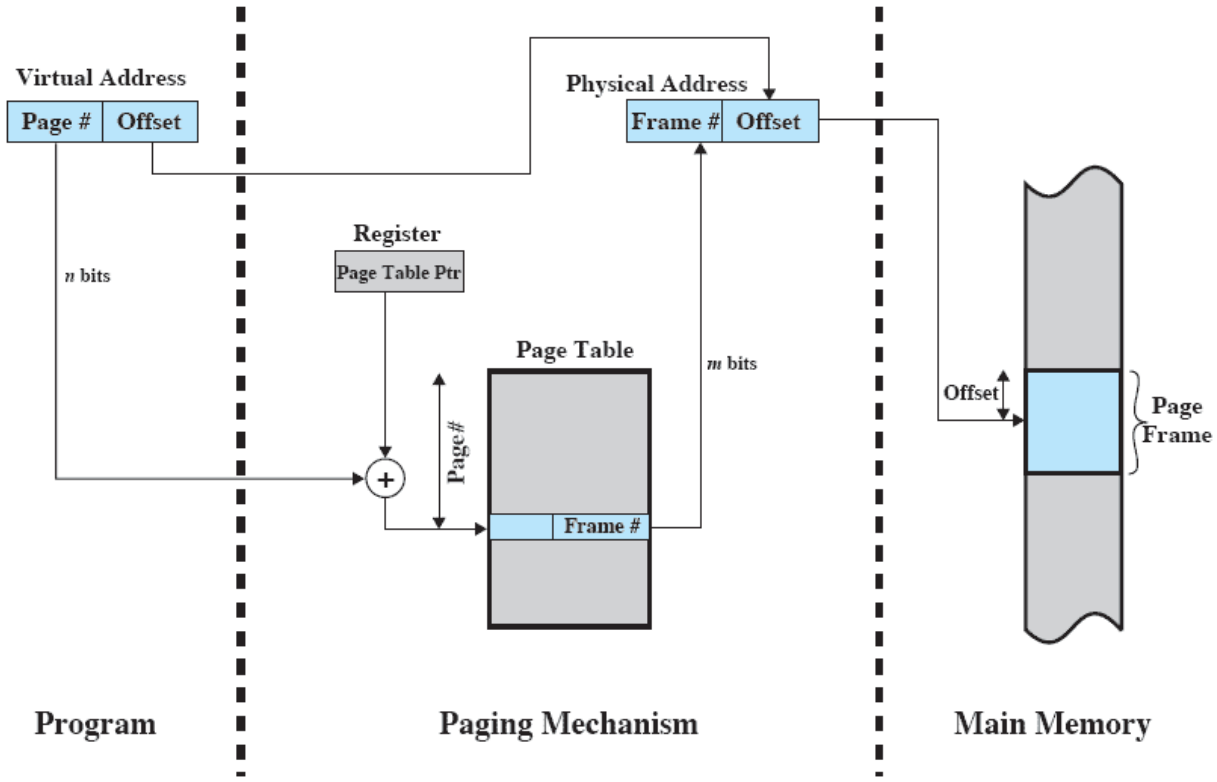




STRUKTURA TABELE STRANICA

- Pri izvršavanju procesa, najčešće se početna adresa tabele stranica drži u registru. Ova adresa se čuva 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

ADRESIRANJE SA VIRTUELNOM MEMORIJOM



TLB BAFER

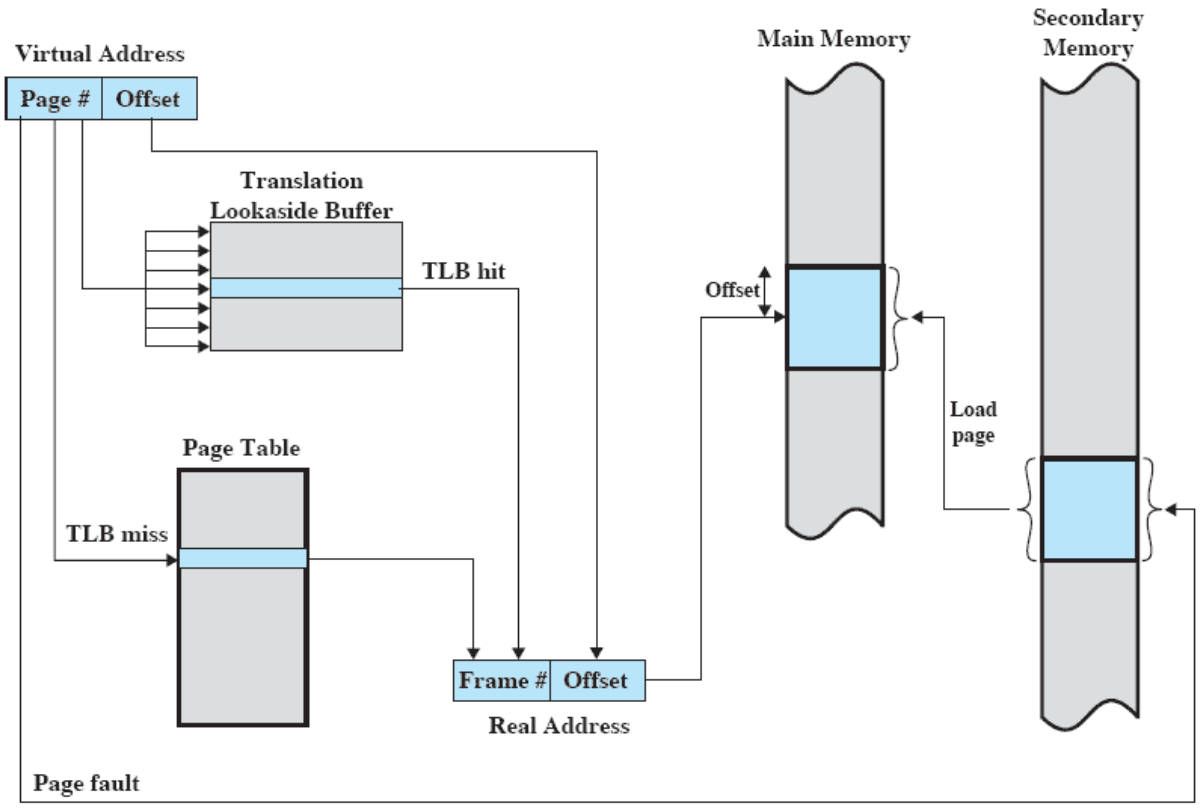
Ako se koristi virtuelna memorija, pri svakom referenciranju podataka potrebna su 2 pristupa memoriji: jedan da se preuzme stavka iz tabele stranica I drugi da se pročita vrednost sa izračunate fizičke adrese.

Da bi se prevazišao ovaj problem, koristi se specijalni brzi keš za stavke tabele stranica po nazivu ***Translation Lookaside Buffer*** **– TLB**. On sadrži stavke tabele stranica koje su najskorije korišćene. Obično postoje odvojeni TLB za adrese koje se odnose na instrukcije I one koje se odnose na podatke. U savremenim procesorima najčešće postoji više nivoa TLB koji su hijerarhijski organizovani po veličini I brzini (svaki sledeći nivo veći ali sporiji).

REFERENCIRANJE UZ KORIŠĆENJE TLB

Pri referenciranju na osnovu virtuelne adrese procesor najpre proverava TLB.

- Ako se stavka za referenciranu stranicu nalazi u TLB, preuzima se broj okvira I formira stvarna adresa
- Ako se stavka za referenciranu stranicu ne nalazi u TLB, pristupa se stavci u tabeli stranica



ASOCIJATIVNO PRESLIKAVANJE

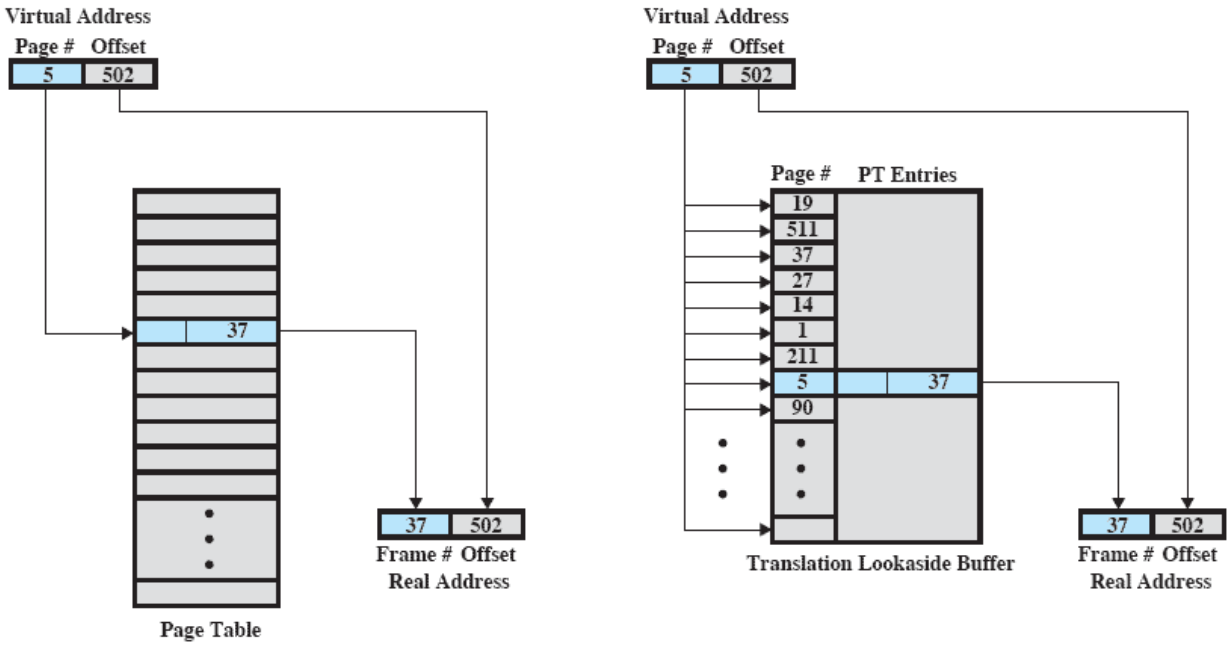
TLB sadrži samo neke od stavki tabele stranica. Ne može se stavka jednostavno pronaći indeksiranjem na osnovu broja stranice. Stavka TLB mora da sadrži broj stranice I kompletnu stavku tabele stranica.

ASOCIJATIVNO PRESLIKAVANJE - Procesor može istovremeno da ispita više stavki TLB da pronađe odgovarajuću stranicu.

DIREKTNO PRESLIKAVANJE

ASOCIJATIVNO PRESLIKAVANJE

N mogućih lokacija u kojima može biti tražena stavka.



STAVKE TLB BAFERA

Stavke u TLB baferu mogu da se odnose 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, potrebno je isprazniti TLB pri komutaciji procesa.

PROCENAT POGODAKA

Deo ukupnih referenciranja stranica se pronađe u TLB baferu. Cilj je ostvariti što veći procenat pogodaka jer tada nije potrebno pristupati tabeli stranica. Time se ubrzava pristup referenciranoj adresi.

VREME PRISTUPA MEMORIJI – Efektivno vreme pristupa memoriji tj. ukupno vreme potrebno za čitanje/upis podatka u memoriju. Vreme uključuje I pristup tabeli stranica I traženoj memorijskoj lokaciji.

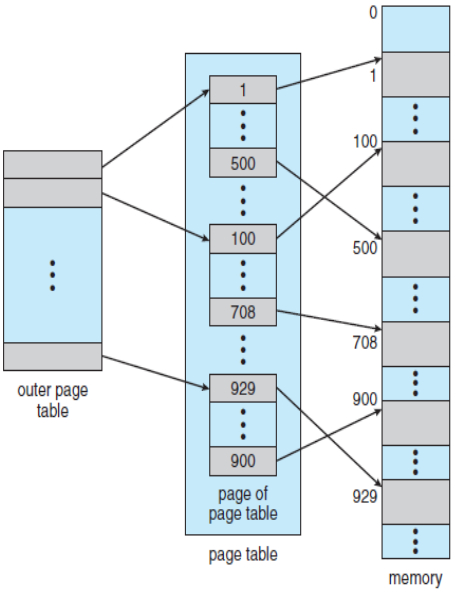
TABELA STRANICA

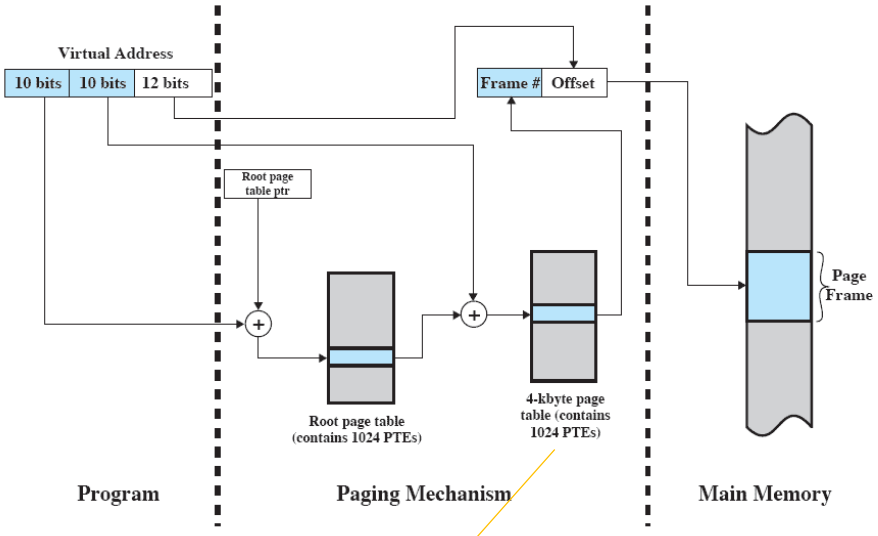
Tabela stranica može da bude prevelika da bi se čuvala u glavnoj memoriji. Jedna varijanta rešenja je smeštanje tabele stranica u virtuelnu memoriju (deljenje tabele stranica na stranice). Pri izvršavanju procesa deo tabele stranica mora da bude u glavnoj memoriji. Potrebno je pristupiti stavci koja se odnosi na stranicu koja se trenutno referencira.

HIJERARHIJSKA TABELA STRANICA

Druga varijanta rešenja za prevelike tabele stranica je mala osnovna tabela stranica. Stavke te tabele stranica pokazuju na stranicu u drugoj tabeli stranica. U toj drugoj tabeli stranica su podaci o stranicama procesa. Može se realizovati u N nivoa – tabele stranica na svim nivoima se čuvaju u radnoj memoriji, najčešće ima 3 nivoa.

OS ne kreira sve delove tabela stranica na nižim nivoima. Ne kreiraju se stavke tabele stranica za stranice iz heap memorije koje se ne koriste. Na ovaj način se troši manje radne memorije u odnosu na klasičnu tabelu stranica u jednom nivou.

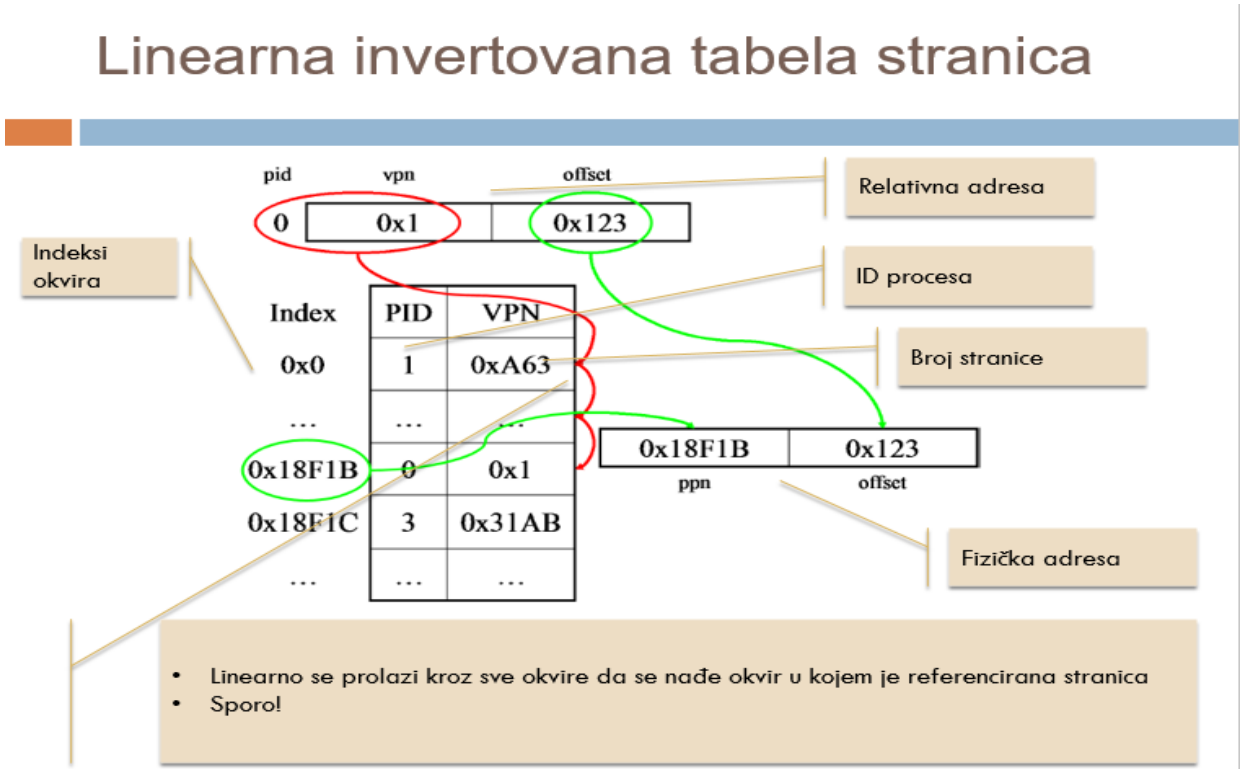


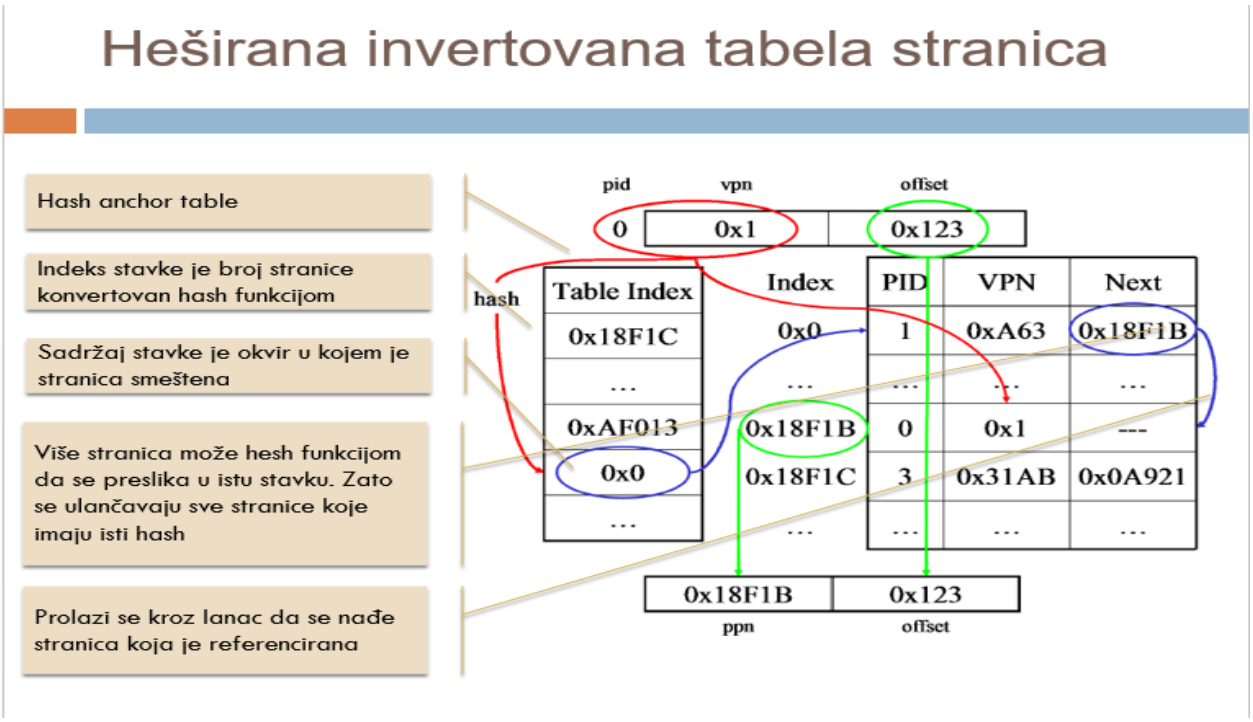


2^{10} stranica sa po 2^{10} stavki

VELIČINA TABELE STRANICA

Loša strana klasične tabele stranica je da je njena veličina proporcionalna veličini virtuelnog adresnog prostora. Alternativa je invertovana tabela stranica koja sadrži po jednu stavku za svaki okvir. Ona sadrži značajno manje stavki od klasične tabele stranica. Stavka specificira koja stranica kog procesa je trenutno smeštena u okvir.





ZAŠTITA MEMORIJE

Zaštita memorije se implementira dodatim bitovima u stavci tabele stranica. Po jedan bit služi za svaki od tri tipa zaštite:

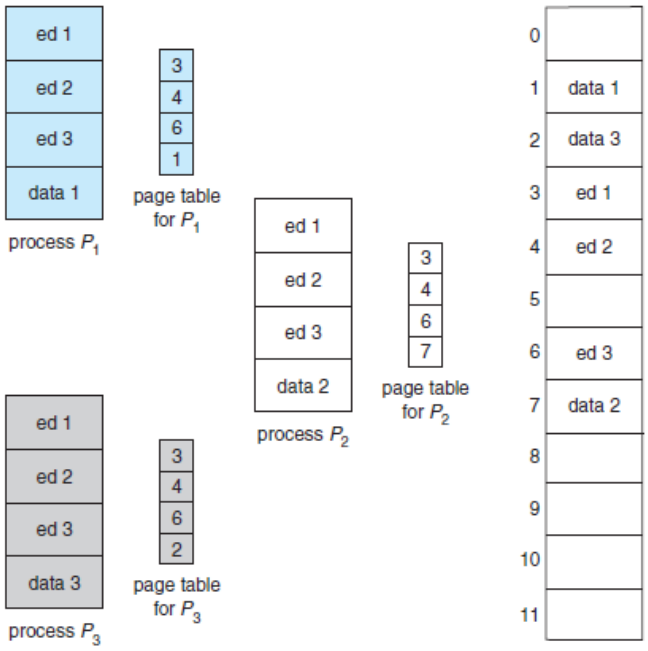
1. Mem. Lok. Samo za čitanje
2. Mem. Lok. Samo za izvršavanje
3. Mem. Lok. Za čitanje I upis

Zaštita memorije je zabrana pristupa stranicama koje ne pripadaju procesu. Implicitno je ugrađena u logiku rada virtuelne memorije. Proces vidi samo svoj virtuelni adresni prostor I 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)

DELJENJE STRANICE

Straničenje omogućuje deljenje zajedničkog koda. Zajednički kod se smešta samo jednom u glavnu memoriju a različiti procesi u svojoj tabeli stranica referenciraju isti okvir sa deljenim kodom.

Primer: 3 instance istog editora – samo je sekcija za podatke svakog procesa različita



VELIČINA STRANICE

Sa manjim stranicama biće manje interne fragmentacije, ali ih onda ima više po procesu, što znači da će tabele stranica biti veće.

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

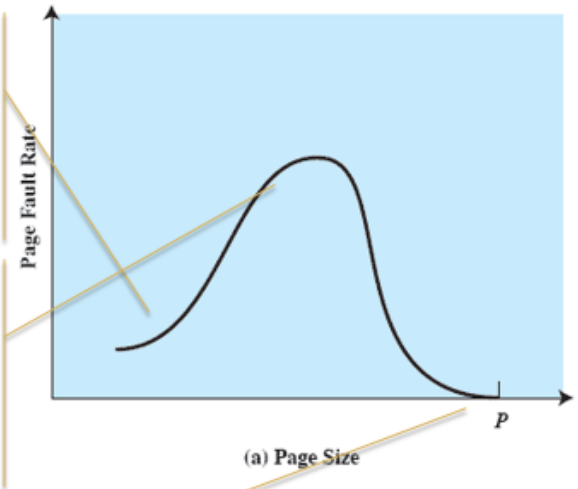
Sekundarna memorija efikasnije prenosi veće blokove podataka, što daje prednost većim stranicama.

Uticaj veličine stranice na učestalost grešaka stranice

- Sa manjim stranicama, veći broj stranica će biti u glavnoj memoriji
- Posle izvesnog vremena u glavnoj memoriji će biti stranice bliske skorašnjim referencama
- Broj grešaka stranice će biti mali

- Sa povećanjem stranica, pojedinačne stranice koje su trenutno u radnoj memoriji će sadržati lokacije sve dalje od skorašnjih referenci
- Broj grešaka stranice raste

- Kako se veličina stranice približava veličini celog procesa, broj grešaka stranice opada
- Ako se ceo proces smesti u jednu stranicu, neće biti grešaka stranica



Uticaj broja dodeljenih okvira na učestalost grešaka stranice

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

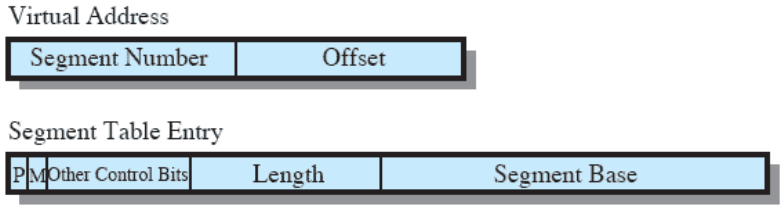
SEGMENTACIJA

Programer vidi program kao skup segmenata sa određenom funkcionalnošću.

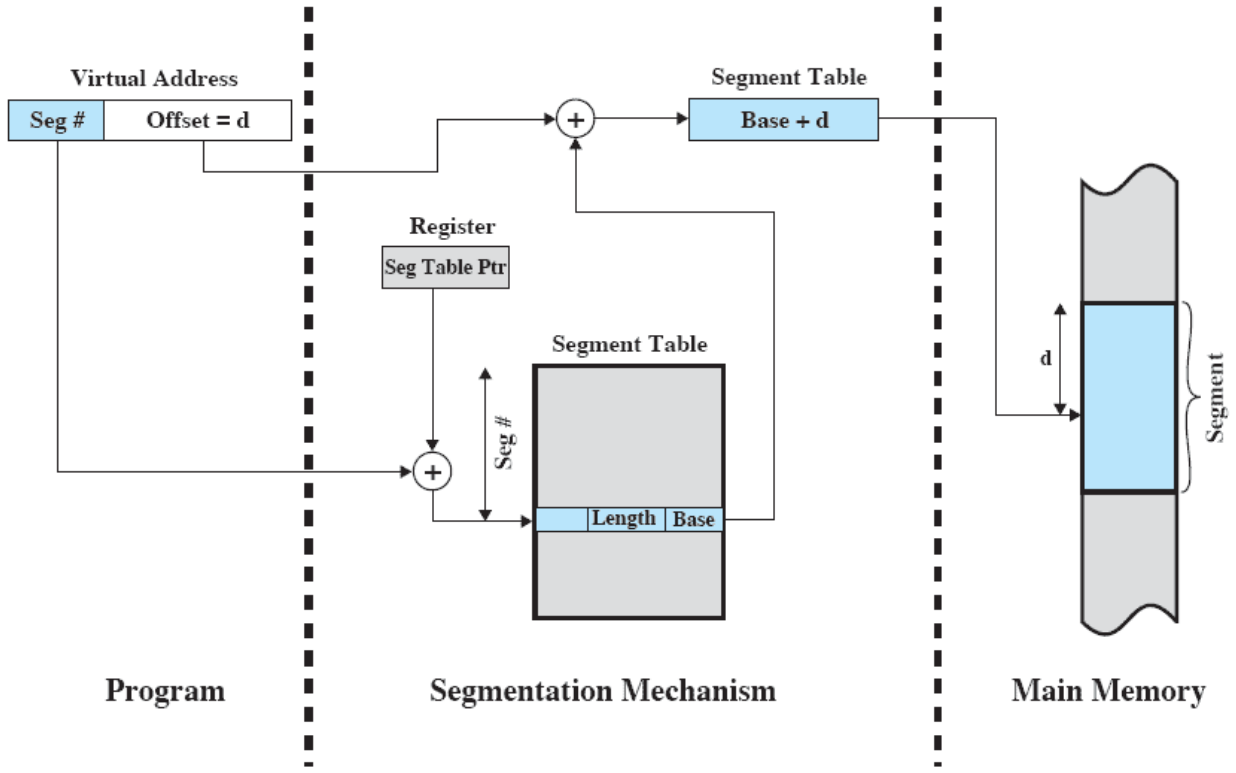
SEGMENTACIJA je način upravljanja memorijom koji podržava ovakav korisnikov pogled na memoriju. Omogućuje programeru da vidi memoriju kao da se sastoji od više adresnih prostora ili segmenata koji mogu biti nejednake dinamičke veličine.

Proces je podeljen na segmente – svi segmenti su smešteni na disku, dok je deo segmenata smešten u glavnoj memoriji

Svaki segment ima tabelu stranica – ona sadrži adresu početka I dužinu segmenta, flag da li je segment u glavnoj memoriji I flag da li je segment menjan otkako je učitan u glavnu memoriju. Ukoliko jeste potrebno je pre izbacivanja upisati izmenu na disk.



PREVOĐENJE ADRESE KOD SEGMENTACIJE



DOBRE STRANE STRANIČENJA I SEGMENTACIJE

SEGMENTACIJA:

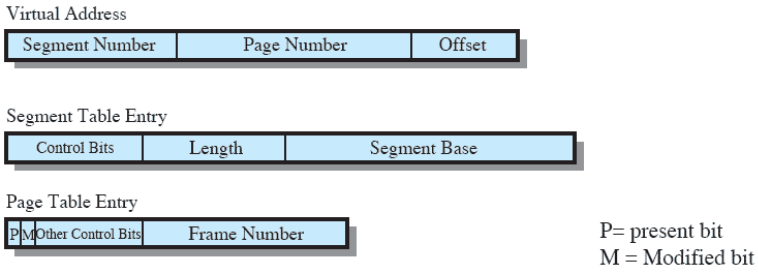
1. Vidljiva za programera
2. Nema unutrašnje fragmentacije
3. Podržava modularnost, deljenje I zaštitu

STRANIČENJE:

1. Transparentno za programera
2. Odstranjuje spoljašnju fragmentaciju
3. Olakšava rukovanje memorijom jer su delovi fiksne veličine

KOMBINOVANO STRANIČENJE I SEGMENTACIJA

Adresni prostor se deli u izvestan broj segmenata a svaki segment se deli u stranice. Adresa sadrži broj segmenata I broj stranice unutar segmenta. Za svaki proces je potrebna tabela segmenata a za svaki segment tabela stranica.



PREVOĐENJE ADRESE KOD KOMBINOVANOG STRANIČENJA I SEGMENTACIJE

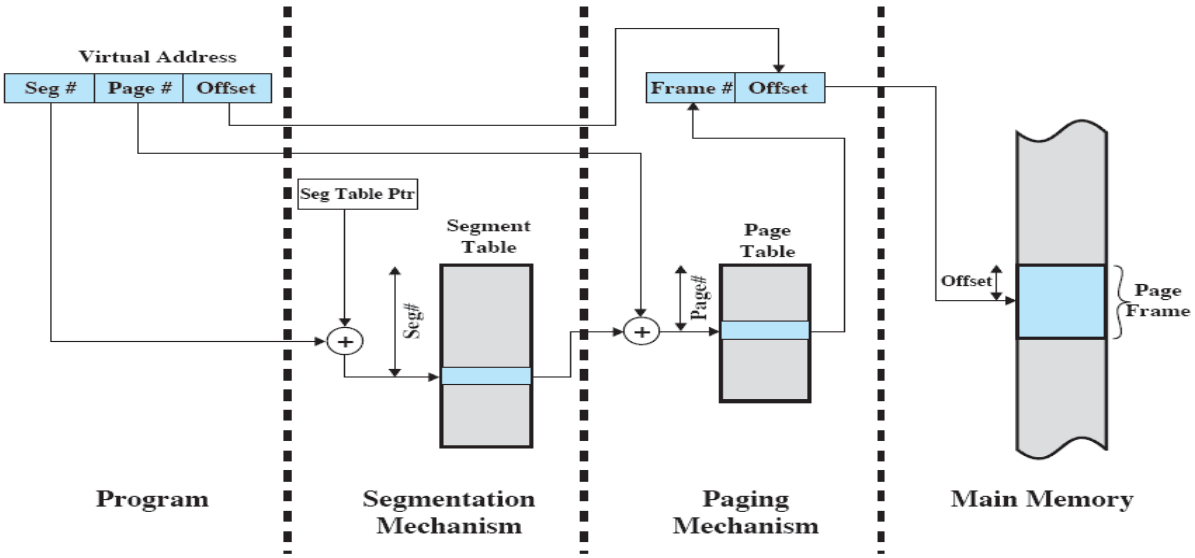


Figure 8.13 Address Translation in a Segmentation/Paging System

ASPEKTI UPRAVLJANJA MEMORIJOM

HARDVERSKI ASPEKT – odgovornost hardvera procesora

SOFTVERSKI ASPEKT – odgovornost softvera operativnog sistema

OS PODRŠKA ZA UPRAVLJANJE MEMORIJOM

Preduslov je da hardver koristi tehnike virtuelne memorije I podržava straničenje ili segmentaciju. Softver OS je odgovoran za algoritme koji se primenjuju pri upravljanju memorijom.

Najvažnije je pitanje performanse:

- Minimizovati učestalost grešaka stranica
- Donošenje odluke, kada kako I koju stranicu treba zameniti
- Raspoređivanje drugog procesa dok se zamenjuje stranica

POLITIKA DONOŠENJA – Odlučuje kada stranica treba da se donese u glavnu memoriju. Dve osnovne varijante:

- **STRANIČENJE PO ZAHTEVU**
  1. Stranica se donosi u glavnu memoriju samo kada se napravi referenca na lokaciju u toj stranici
  2. Na početku rada dolazi do mnogo grešaka stranica, ali kasnije taj broj opada

- **PREDSTRANIČENJE**
  1. Unapred se donose stranice koje nisu zahtevane
  2. Efikasnije je doneti više stranica odjednom ako se na disku nalaze u susednim lokacijama.
  3. Problem je što može nepotrebno doneti stranicu koja se neće koristiti

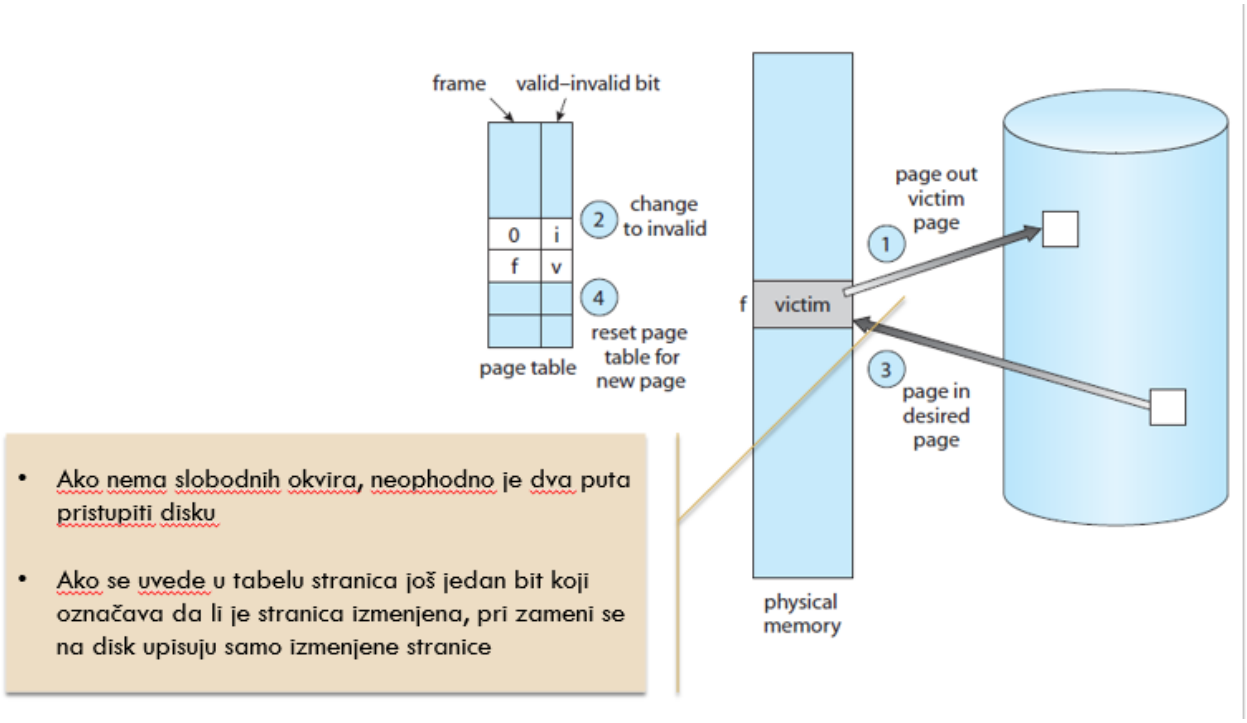
POLITIKA SMEŠTANJA – Određuje gde u glavnoj memoriji proces treba da bude smešten. Ako sistem koristi straničenje, nevažno je. Hardver za prevođenje adresa radi jednako bez obzira u kojem je konkretno okviru stranica smeštena.

POLITIKA ZAMENE – Kada su svi okviri u glavnoj memoriji zauzeti potrebno je da se donese nova stranica. Politika zamene određuje koja stranica iz glavne memorije treba da bude zamenjena.

Cilj je minimizovati broj grešaka stranice tako što se zameni ona stranica za koju je najmanje verovatno da će uskoro biti referencirana. To se vrši uz pomoć principa lokalnosti.

Većina politika pokušava da predvidi buduće ponašanje na osnovu ponašanja u prošlosti.

ZAMENA STRANICA



ZAKLJUČAVANJE OKVIRA – Ako je okvir zaključan, stranica smeštena u njemu ne može da se zameni. U zaključanim okvirima čuvaju se kernel OS, glavne upravljačke strukture I U/I baferi.

Mogu se zaključati I tek donešene stranice da ne bi bile zamenjene pre korišćenja zato što drugi proces treba slobodan okvir.

Zaključavanje se postiže pridruživanjem bita zaključavanja svakom okviru. Mora se oprezno koristiti jer se zaključavanjem okvira smanjuje broj raspoloživih okvira za smeštanje stranica.

POLITIKE ZA IZBOR STRANICE ZA ZAMENU

1. **OPTIMALNA**
2. **NAJMANJE SKORO KORIŠĆENA (***LRU – Last recently used***)**
3. **PRVA UNUTRA, PRVA NAPOLJE (***FIFO***)**
4. **ČASOVNIK**

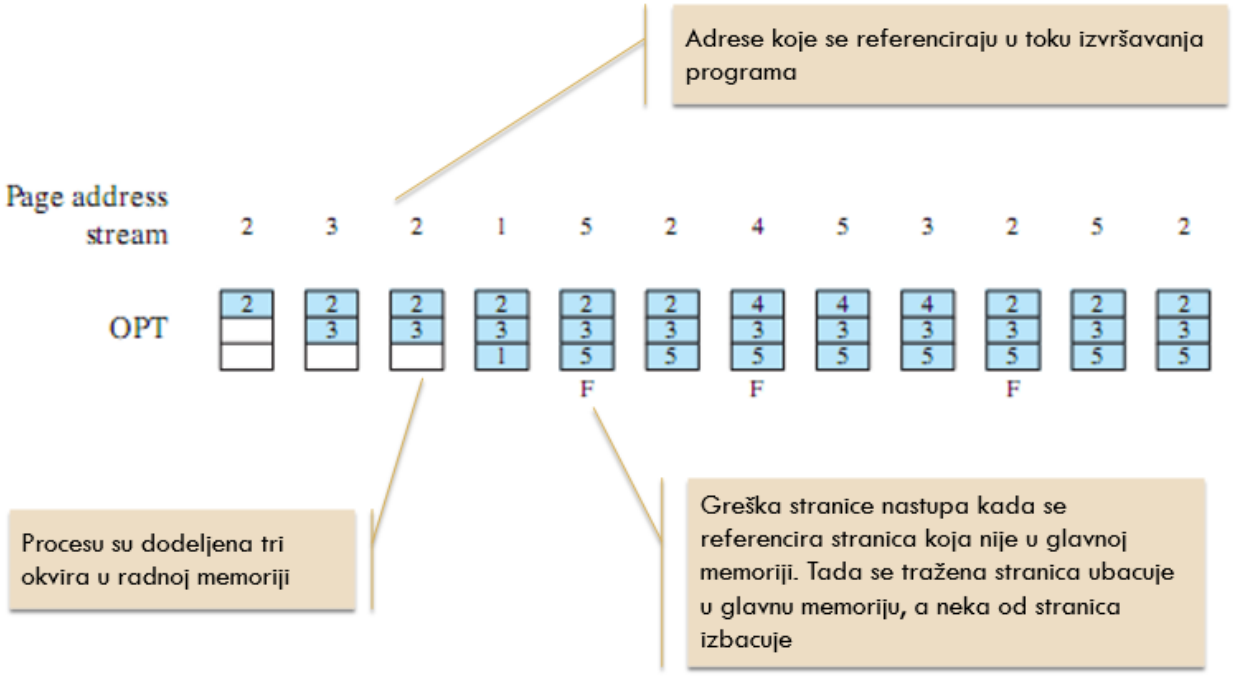
NIZ REFERENCI

Program pri izvršavanju redom referencira određene lokacije u memoriji. Politike zamene se porede kroz broj grešaka stranice na određenom nizu referenci. Kod referenci nas interesuju samo adrese stranica (pomeraj unutar stranice sada nije važan). Uzastopne reference na istu stranicu se mogu posmatrati kao jedna referenca. ***Primer:*** 2 3 2 1 5 2 4 5 3 2 5 2

OPTIMALNA POLITIKA

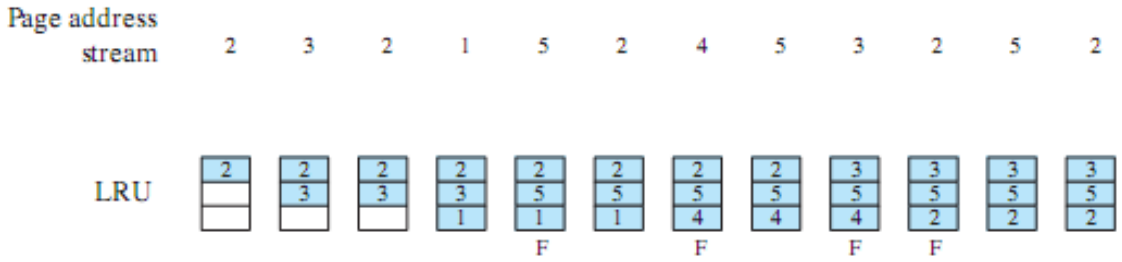
Bira se ona stranica za koju je vreme do sledeće reference najduže.

Ova politika ne može da se implementira jer zahteva od OS savršeno poznavanje budućih događaja. Služi kao standard u odnosu na koji se procenjuju algoritmi koje je moguće implementirati.



NAJMANJE SKORO KORIŠĆENA

Ako sistem ne poznaje buduće događaje, ima evidenciju prošlih. Zamenjuje stranicu koja najduže nije bila korišćena pod pretpostavkom da ni uskoro neće biti.

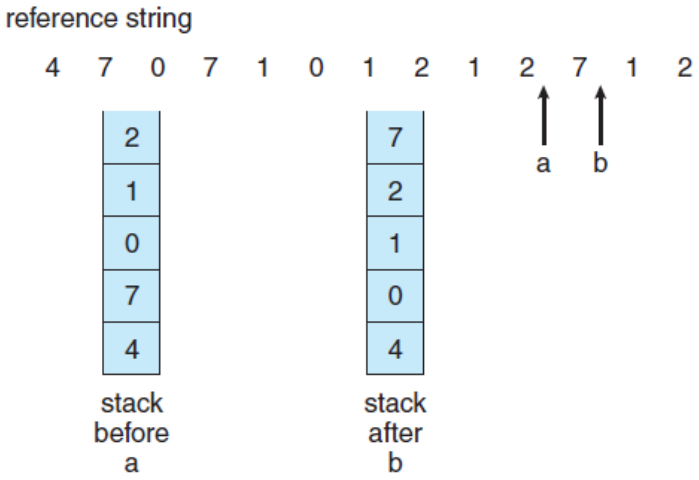


Po principu lokalnosti, trebalo bi da bude zamenjena stranica za koju je najmanje verovatno da će biti referencirana u bliskoj budućnosti. Radi skoro jednako dobro kao I optimalna politika, ali je teška za implementaciju.

Dve varijante implementacije:

- BROJAČ – Procesor evidentira broj otkucaja. U tabeli stranica se uz svaku stranicu doda polje koje označava trenutak poslednjeg referenciranja. Pri svakom referenciranju se upiše trenutak referenciranja. Zamenjuje se stranica sa najmanjim trenutkom vremena.

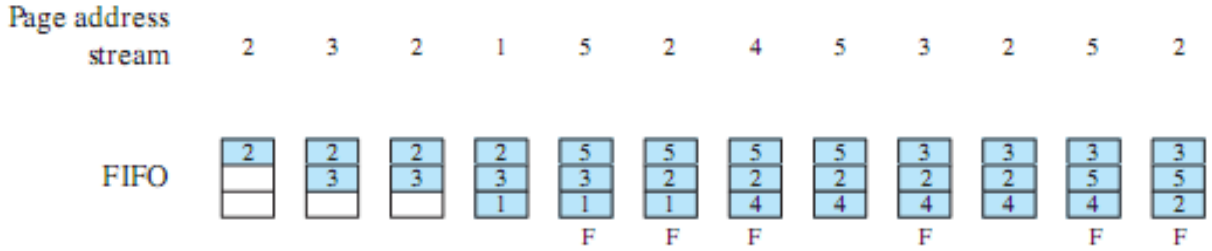
- STEK – čuva se stek referenciranih stranica. Kada se stranica referencira, stavlja se na vrh steka. Ako je već bila referencirana izbacuje se iz tog dela steka. Ovakav stek se može implementirati kao dvostruko spregnuta lista.



PROBLEM: Ne može se koristiti u realnim sistemima jer bilo koja implementacija usporava sistem. Ažuriranje trenutaka referenciranja ili steka mora biti obavljeno pri svakom referenciranju, što uvodi usporenje od preko 10 puta.

PRVA UNUTRA, PRVA NAPOLJE – FIFO

Tretira okvire kao kružni bafer, najjednostavnija politika za implementaciju. Postoji pokazivač na narednu stranicu koju treba ukloniti. Pokazivač kruži redom kroz okvire. Biće zamenjena stranica koja je najduže u memoriji, što je često pogrešno. Stranice koje se često referenciraju će stalno biti izbacivane I ponovo ubacivane.



**više grešaka od LRU, ne prepoznaje stranice koje su češće referencirane od ostalih*

APROKSIMACIJE POLITIKE LRU

Postoje politike koje pokušavaju da dobiju rezultate kao LRU a da pri tome izbegnu usporenje koje klasična implementacija ove politike donosi.

Baziraju se na tome da hardver za svaku stranicu ažurira bit upotrebe:

- Kada se stranica donese u memoriju ili referencira, bit se postavlja na 1
- OS može da resetuje bit upotrebe za određenu stranicu
- Dobijamo informaciju koje su stranice referencirane, ali ne I u kojem redosledu

POLITIKA DODATNIH BITOVA UPOTREBE

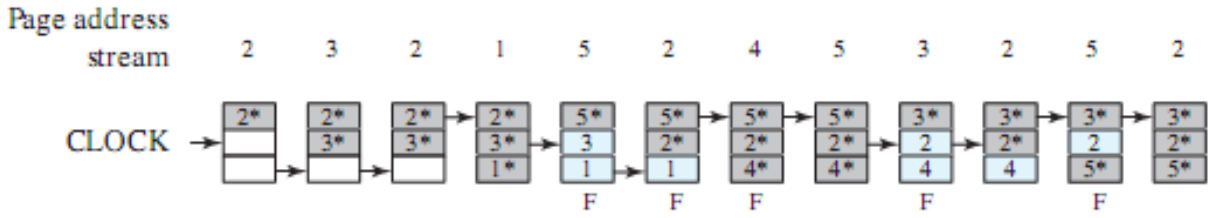
OS evidentira 8-bitnu reč za svaku stranicu. U regularnim intervalima, OS prebacuje bit upotrebe na najvišu poziciju u reči, trenutne bite pomera na desno, a bit na najmanjoj poziciji ispada iz reči.

8-bitna reč time sadrži istoriju referenciranja. Ako je posmatramo kao dekadni broj, stranica sa većom vrednošću reči je skorije referencirana, a stranica sa najmanjom vrednošću će biti zamenjena.

POLITIKA ČASOVNIKA

Samo jedan bit upotrebe za svaku stranicu. Kada je potrebno pronaći stranicu za zamenu, OS prolazi kružno kroz okvire, I kad god naiđe na okvir sa bitom upotrebe 1, postavi ga na 0. Biće zamenjena stranica za koju je bit upotrebe 0.

Pošto ide u krug, u najgorem slučaju će u drugom prolasku pronaći okvir čiji je bit u prvom prolasku postavljen na 0. Naredno pretraživanje će krenuti od okvira koji se nalazi posle okvira koji je zamenjen.



- FIFO bi zamenio 2, umesto 4
- Pošto je 2 skoro referencirana, algoritam časovnika je ne zamenjuje

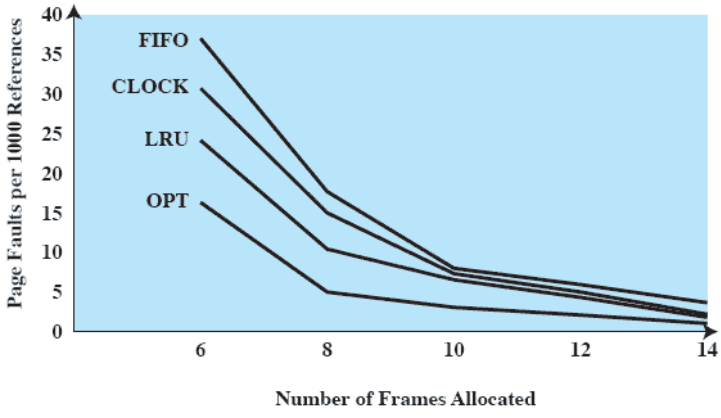
UNAPREĐENJE POLITIKE ČASOVNIKA

Povećava se broj bitova koje algoritam koristi. Svaka stranica ima bit promene koji ukazuje na to da li je sadržaj menjan otkako je donešena u glavnu memoriju. Ako je menjana, pre zamenjivanja se izmena mora upisati u sekundarnu memoriju.

Algoritam časovnika može da koristi I ovaj bit – prednost za zamenu daje stranicama koje nisu menjane čime se štedi vreme jer ne treba upisivati izmenu u sekundarnu memoriju pre zamene.

POREĐENJE ALGORITAMA

Zbog lokalnosti, nakon određene granice, dalje povećanje broja okvira dodeljenih procesu ima manji uticaj



BAFEROVANJE STRANICA

Deo glavne memorije se odvoji da radi kao keš stranica.
Zamenjena stranica se ubacuje u taj keš:

- Ako stranica nije menjana, dodaje se u listu slobodnih stranica
- Ako je stranica menjana dodaje se u listu menjanih stranica

Kada se ponovo referencira stranica, postoji šansa da se ona nalazi u baferu, pa je vraćanje u upotrebu brzo I jednostavno. Promenjene stranice se mogu upisivati na disk u grupama, što je brže.

UPRAVLJANJE REZIDENTNIM SKUPOM

OS za svaki proces treba da odluči koliko okvira da mu dodeli za smeštanje stranica. Što je manje memorije dodeljeno jednom procesu, to više procesa može istovremeno da bude u memoriji. Što je manje okvira dodeljeno procesu, biće više grešaka stranice. Zbog principa lokalnosti, povećavanje broja dodeljenih okvira preko određene granice neće imati zapažen efekat na broj grešaka stranice.

MINIMALNA VELIČINA REZIDENTNOG SKUPA

Kada se desi greška stranice, instrukcija se restartuje. Neophodno je da se procesu dodeli bar onoliko okvira koliko svaka pojedinačna instrukcija može da referencira. Arhitektura skupa instrukcija definiše minimalan broj okvira za proces.

Drugi parametar za određivanje minimalne veličine jeste broj grešaka stranica.

POLITIKE VELIČINE REZIDENTNOG SKUPA

1. **FIKSNO DODELJIVANJE** – Proces dobija fiksan broj okvira u glavnoj memoriji u koji će se smeštati stranice procesa

    - **JEDNAKO DODELJIVANJE** – Svaki proces dobija n-ti deo ukupnog broja okvira. Ovakav pristup je neracionalan kada su procesi različite veličine. Proces može da dobije više okvira nego što sadrži stranica.
    - **PROPORCIONALNO DODELJIVANJE** – Proces dobija broj okvira proporcionalno broju svojih stranica u odnosu na broj stranica svih drugih procesa.

2. **PROMENLJIVO DODELJIVANJE** – Broj okvira dodeljenih procesu se menja u toku životnog veka procesa. Procesima sa velikim brojem grešaka stranca se dodeli više okvira stranica, a sa malim brojem grešaka stranica se smanjuje broj dodeljenih okvira.

OPSEG ZAMENE

Određuje koje stranice dolaze u obzir za zamenu kada se desi greška stranice

- **LOKALNI OPSEG ZAMENE** – bira se stranica među stranicama procesa koji je napravio grešku stranice
- **GLOBALNI OPSEG ZAMENE** – kandidati za zamenu su sve stranice u nezaključanim okvirima, bez obzira kojem procesu pripadaju

FIKSNO DODELJIVANJE I LOKALNI OPSEG – unapred se odredi broj okvira koji su dodeljeni procesu. Ako se dodeli premalo okvira biće puno grešaka stranice, ako se dodeli previše okvira, biće premalo procesa u glavnoj memoriji I često nijedan od procesa neće biti spreman. Ovo je nefleksibilno rešenje.

PROMENLJIVO DODELJIVANJE I LOKALNI OPSEG – novom procesu se dodeli određeni broj okvira, kada se desi greška stranice, zamenjuje se jedna od stranica tog procesa. S vremena na vreme se analizira broj okvira dodeljenih procesu I vrši se smanjenje ili povećanje tog broja da se poboljšaju performanse.

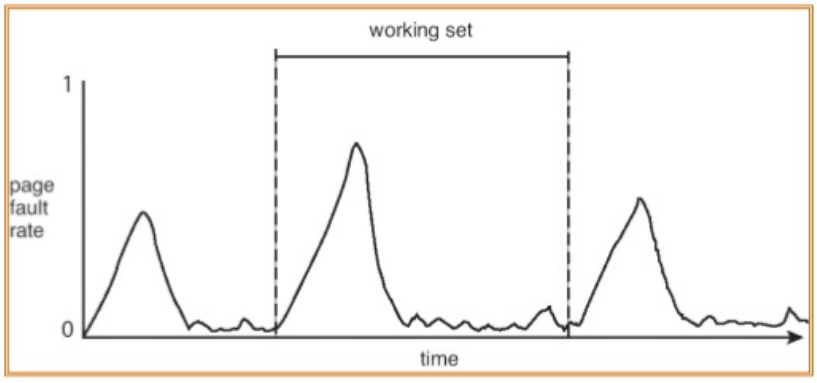
PROMENLJIVO DODELJIVANJE I GLOBALNI OPSEG – jednostavno za implementaciju, OS održava listu slobodnih okvira. Procesu se dodeljuje slobodan okvir kada se desi greška stranice. Ako nema slobodnih okvira zamenjuje se neka stranica – izbor se pravi među svim nezaključanim okvirima u memoriji, tom procesu se smanjuje rezidentni skup što mu može smanjiti performanse.

RADNI SKUP

Strategija radnog skupa se zasniva na principu lokalnosti. Koristi se za određivanje odgovarajuće veličine rezidentnog skupa.

Radni skup W(t,d) – u trenutku vremena t, skup stranica koje su bile referencirane u poslednjih d jedinica vremena, gde se vreme meri referenciranjem stranica (1 referenciranje je 1 jed. Vremena)

Radni skup procesa zbog principa lokalnosti ima relativno stabile periode ali I periode kada se drastično menja zbog prelaska na **novu lokalnost.**



STRATEGIJA RADNOG SKUPA je nadgledati radni skup svakog procesa I periodično uklanjati iz memorije one stranice koje nisu u radnom skupu.

PROBLEM: merenje radnog skupa je nepraktično I zahtevno sa stanovišta performansi

ALGORITAM UČESTALOSTI GREŠKE STRANICE

PFF – Page fault frequency

Umesto nadgledanja radnog skupa, nadgleda se učestalost grešaka stranica. Učestalost grešaka stranice opada povećanjem rezidentnog skupa procesa. Ako je učestalost manja od nekog praga, smanjujemo rezidentni skup procesu, a ako je veća povećavamo rezidentni skup.

Svakoj stranici se pridružuje bit upotrebe. Brojač referenci broji koliko je proteklo referenci od poslednje greške stranice. Kada se desi greška stranice, brojač se poredi sa pragom i:

- 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

Ne radi dobro u toku prelaznih perioda kada se prelazi u novu lokalnost:

- Pri prelasku u novu lokalnost veliki broj grešaka stranica će povećavati rezidentni skup ne izbacujući odmah stranice stare lokalnosti
- Kasnije će se rezidentni skup opet smanjivati, jer se stranica stare lokalnosti neće referencirati
- Da bi se stranica izbacila iz radnog skupa potrebno je da prođe F jedinica virtuelnog vremena od kad je poslednji put referencirana

RADNI SKUP SA PROMENLJIVIM INTERVALOM UZORKOVANJA

VSWS – Variable interval sampled working set

Definiše se interval uzorkovanja. Na početku intervala svim stranicama u rezidentnom skupu se resetuje bit upotrebe. U toku intervala svaka stranica koja prouzrokuje grešku stranice se dodaje u rezidentni skup (u toku intervala on može samo da raste). Na kraju intervala se izbacuju stranice koje imaju bit upotrebe 0 (na kraju intervala se rezidentni skup smanjuje).

U toku izvšravanja procesa meri se virtuelno vreme.

- Ako je prošao maksimalni period od poslednjeg uzorkovanja, suspenduje se proces I skeniraju bitovi upotrebe
- Ako u toku intervala broj grešaka strance pređe određeni prag, čeka se da istekne minimalni period uzorkovanja I onda se vrši uzorkovanje

VSWS algoritam se bolje ponaša u prelazima između lokalnosti. Učestalost uzorkovanja zavisi od učestalosti grešaka stranica. Stranice stare lokalnosti će brže ispadati iz rezidentnog skupa nego kod PFF algoritma.

POLITIKA ČIŠĆENJA

Odlučuje kada se izmenjena stranica upisuje nazad u sekundarnu memoriju.

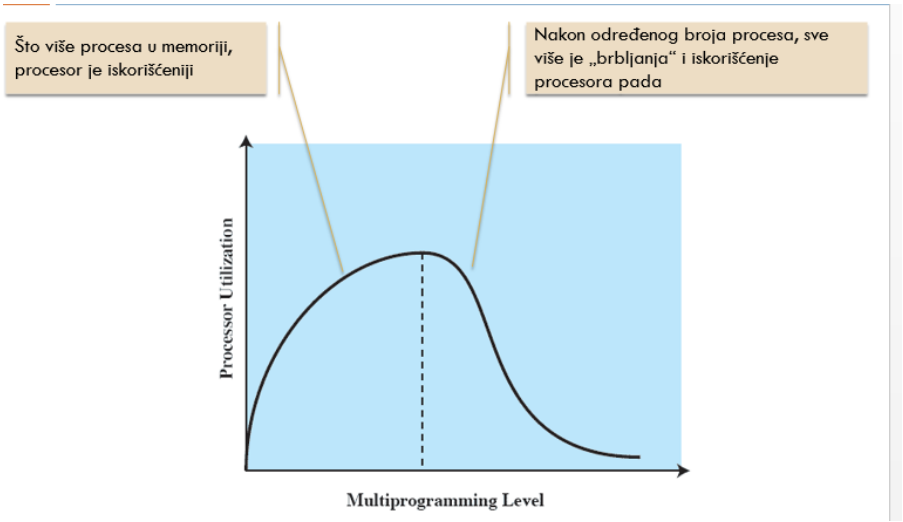
ČIŠĆENJE PO ZAHTEVU – stranica se upisuje kada je izabrana za zamenu

PREDČIŠĆENJE – stranice se upisuju unapred u paketima pre nego što je traženo da se zamene. Ako se opet stranica izmen, onda je prethodno upisivanje na disk bilo bespotrebno trošenje resursa.

Najbolji pristup je **BAFEROVANJE STRANICA** – Zamenjene stranice se nalaze u dve liste, u jednoj izmenjene a u drugoj neizmenjene. Stranice iz liste izmenjenih se periodično upisuju na disk. Stranice iz liste neizmenjenih se vraćaju u okvire dodeljene procesu ako se referenciraju, ili se izbacuju iz glavne memorije ako nema više mesta za njih u listi.

Bavi se nivoom multiprogramiranja. Ako je u memoriji:

- ***PREMALO PROCESA*** – često će svi procesi biti blokirani pa će procesor biti besposlen
- ***PREVIŠE PROCESA*** – veličina rezidentnog skupa svakog procesa je mala, javlja se puno grešaka stranice I 'brbljanje' da bi se dobavile stranice sa diska I upisivale nazad na disk. Neki od procesa mora biti suspendovan, za šta postoje različite varijante izbora procesa koji će biti suspendovan



SUSPENZIJA PROCESA

1. **PROCES NAJNIŽEG PRIORITETA**
2. **PROCES KOJI JE IZAZVAO GREŠKU** – nema radni skup u memoriji pa bi svakako bio blokiran čekajući zamenu stranice
3. **PROCES KOJI JE NAJDUŽE NEAKTIVAN** – najverovatnije da neće imati svoj radni skup u memoriji
4. **PROCES SA NAJMANJIM REZIDENTNIM SKUPOM** – najmanji napor da se ponovo učita
5. **NAJVEĆI PROCES** – dobija se najviše slobodnih okvira
6. **NAJMANJE HITAN PROCES**

8 – Jednoprocesorsko raspoređivanje

Cilj procesorskog raspoređivanja je da se procesoru dodeljuju procesi za izvršavanje tako da se ispune ciljevi sistema kao što su vreme odziva, propusna moć I efikasnost procesora.

CILJEVI RASPOREĐIVANJA

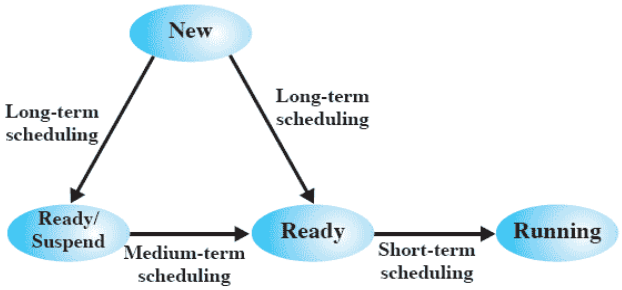
Raspoređivanje treba da:

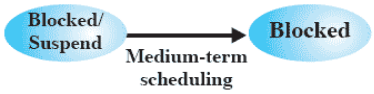
1. obezbedi fer deljenje vremena između procesa
2. spreči gladovanje procesa
3. obezbedi efikasno korišćenje procesora
4. uspostavi privilegije (prioritet) među procesima
5. pomenute poslove obavlja uz što manje dodatnog opterećenja

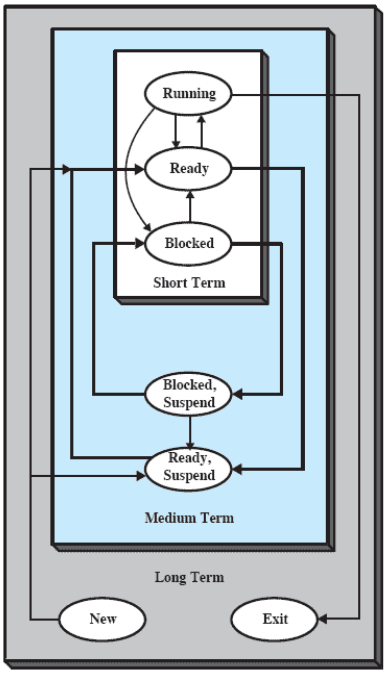
TIPOVI PROCESORSKOG RASPOREĐIVANJA

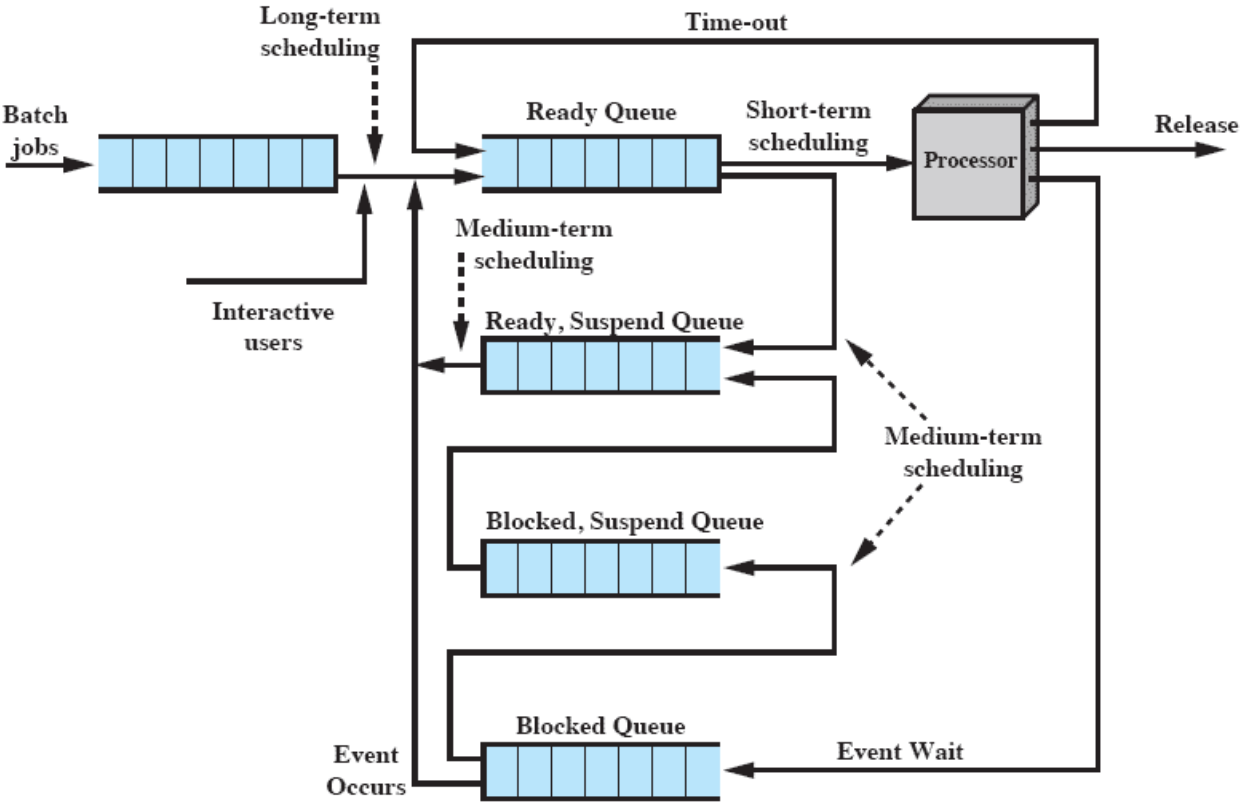
1. **DUGOROČNO RASPOREĐIVANJE**
   - Stvaranje novog procesa
   - Odluka da se proces doda u skup procesa koji konkurišu za izvršavanje
   - Upravljanje stepenom multiprogramiranja – što više procesa manji procenat vremena u kojem proces može da se izvršava
2. **SREDNJEROČNO RASPOREĐIVANJE**
   - Deo funkcije razmene
   - Odluka da se proces doda broju procesa koji su delimično ili potpuno u glavnoj memoriji
   - Određuje kada proces treba da bude zamenjen (suspendovan) na disk
   - Raspoređivanje se vrši tako da se zadovolji ciljni stepen multiprogramiranja
3. **KRATKOROČNO RASPOREĐIVANJE**
   - Naziva se I dispečer
   - Najčešće se izvršava
   - Instrukcije kojeg procesa će procesor da izvršava
   - Aktivira se kada se desi događaj (prekidi generatora takta, U/I prekidi, pozivi OS, signaliziranja)

RASPOREĐIVANJE U ŽIVOTNOM CIKLUSU PROCESA









Raspoređivanje se svodi na upravljanje redovima čekanja

DIMENZIJE KRITERIJUMA

stanovište korisnika vs stanovište sistema

KORISNIČKI ORIJENTISANI KRITERIJUMI – Ponašanje programa kako ga vidi korisnik (*npr. Vreme odziva*). Vreme proteklo od podnošenja zahteva do odbijanja odziva kao izlaza. Cilj je politika raspoređivanja koja obezbeđuje dobro vreme odziva.

SISTEMSKI ORIJENTISANI KRITERIJUMI – Usresređeni su na delotvorno I efikasno iskorišćenje procesora (*npr. Propusna moć*). Broj obrađenih procesa u jedinici vremena. Cilj je politika raspoređivanja koja će uvećati propusnu moć.

Razlikujemo 2 vrste kriterijuma:

- Koji se odnose na performanse – obično su kvantitativni I merljivi (*vreme odziva, propusna moć*)
- Koji nisu povezani sa performansama – kvalitativni I teže merljivi (*predvidljivost*)

KRITERIJUMI KRATKOROČNOG RASPOREĐIVANJA

Glavni cilj je da se procesorsko vreme dodeljuje na način da optimizuje različite aspekte ponašanja sistema. Ustanovljava se skup kriterijuma na osnovu kojih se ocenjuju različite politike raspoređivanja.

Korisnički orijentisani, u vezi sa performansama:

1. **VREME PROLASKA ZADATKA** – vreme od podnošenja zahteva do završetka procesa, uključuje I vreme koje proces provede čekajući na resurse
2. **VREME ČEKANJA** – vreme koje proces proveo u redovima čekanja
3. **VREME ODZIVA** – vreme od podnošenja zahteva do početka primanja odziva
4. **ROKOVI** – politika raspoređivanja treba da obezbedi da se proces završi u zadatom roku (ako takav rok postoji). Drugi ciljevi se podređuju ispunjenju roka.

Korisnički orijentisani, ostali kriterijumi

5. **PREDVIDLJIVOST** – posao treba da se završi sa približno istim vremeno I troškovima bez obzira na opterećenje sistema.

Sistemski orijentisani, u vezi sa performansama:

6. **PROPUSNA MOĆ** – broj završenih procesa u jedinici vremena, cilj je uvećati propusnu moć
7. **ISKORIŠĆENJE PROCESORA** – procenat vremena u kojem je procesor zauzet, cilj je da procesor bude što više zauzet

Sistemski orijentisani, ostali:

8. **PRAVIČNOST** – procesi treba ravnopravno da dele procesorsko vreme I ne smeju da gladuju
9. **PRIMENA PRIORITETA** – ako su procesima dodeljeni različiti prioriteti, politika raspoređivanja treba da daje prednost procesima sa višim prioritetom
10. **URAVNOTEŽENJE RESURSA** – raspoređivanje treba da optimizuje upotrebu resursa (*npr. Prednost se može dati procesima koji manje koriste preopterećene resurse*)

REŽIM IZBORA PROCESA ZA IZVRŠAVANJE – određuje trenutke u vremenu kada se vrši izbor narednog procesa kojem će biti dodeljen procesor:

- **BEZ PREKIDANJA** – proces se izvršava dok god se sam ne završi ili se sam blokira da sačeka U/I ili uslugu OS
- **SA PREKIDANJEM** – OS može da prekine proces koji se izvršava I postavi ga u stanje spreman. Na osnovu prekida generatora takta, OS može da prekine proces I ako mu je isteklo maksimalno vreme neprekinutog izvršavanja ili ako raspoređivač želi da da prednost drugom procesu.

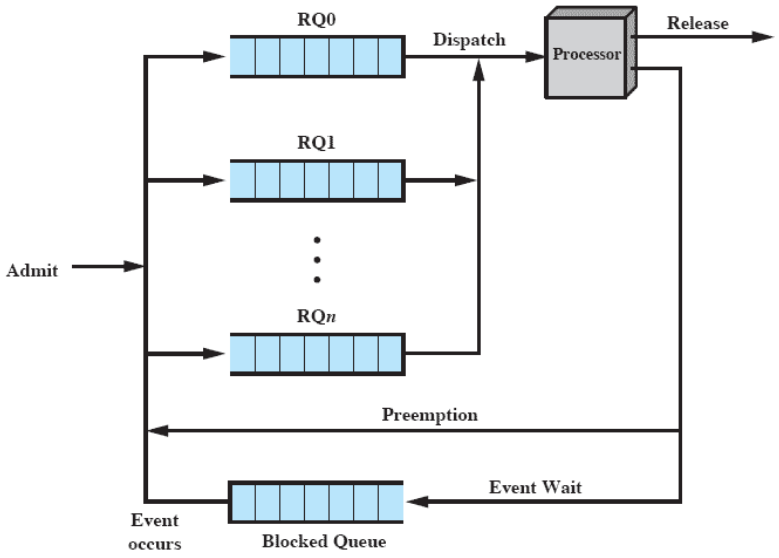
PRIORITETI

Procesima se dodele prioriteti takvi da proces višeg prioriteta ima prednost. Obično je brojna vrednost.

Može biti definisan:

INTERNO – prioritet proističe I izračunava se iz podataka procesa (na osnovu veličine memorijskih zahteva, količine utrošenog procesorskog vremena itd.)

EKSTERNO – prioritet je definisan u procesu izvan OS (npr. Na osnovu važnosti I tipa procesa)



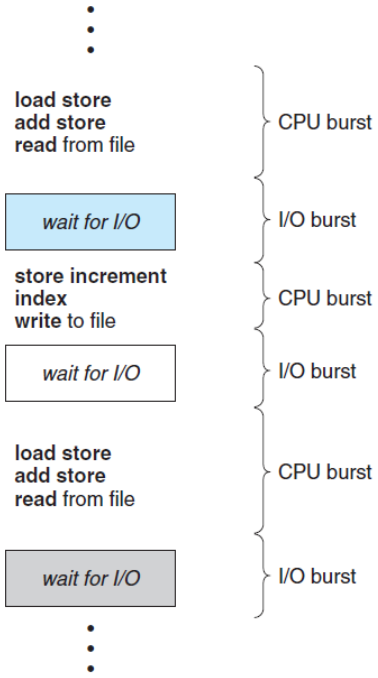
Raspoređivač bira uvek proces sa višim prioritetom kao sledeći za izvršavanje. Postoje različiti redovi čekanja za svaki nivo prioriteta.

GLADOVANJE – Procesi sa nižim prioritetom ne dobijaju procesor ako uvek ima onih sa višim. Jedno rešenje je promena prioriteta zavisno od starosti I istorije izvršavanja.

CIKLUSI IZVRŠAVANJA PROCESA

CPU-Burst – vreme proteklo od trenutka dobijanja procesora do gubljenja procesora. Završava se blokiranjem procesa ili vraćanjem u red spremnih zbog isteka određenog vremena. Proces u toku životnog ciklusa obično više puta dobija I gubi procesor, tako da se CPU-burst menja u toku izvršavanja.

Procesi sa intenzivnom potrebom procesora imaju dugačak, dok procesi sa intenzivnim U/I kratak CPU-burst.



POLITIKE RASPOREĐIVANJA PROCESA

1. **Po redosledu dolaska**
2. **Kružno dodeljivanje**
3. **Najkraći proces sledeći**
4. **Najkraće preostalo vreme**
5. **Sledeći sa najvećim odnosom odziva**
6. **Povratna sprega**

PARAMETRI ZA TESTIRANJE POLITIKA

Posmatramo jedan CPU-burst za svaki proces.

- Trenutak dolaska – trenutak u kom proces postaje spreman
- Trenutak završetka – trenutak kada je izvršavanje instrukcija kompletirano
- Vreme usluge – vreme potrebno procesoru da izvrši instrukcije procesa
- Vreme prolaska – vreme koje proces provodi u sistemu *(vreme usluge + vreme čekanja)*
- Normalizovano vreme prolaska – relativno kašnjenje procesa

PREMA REDOSLEDU DOLASKA

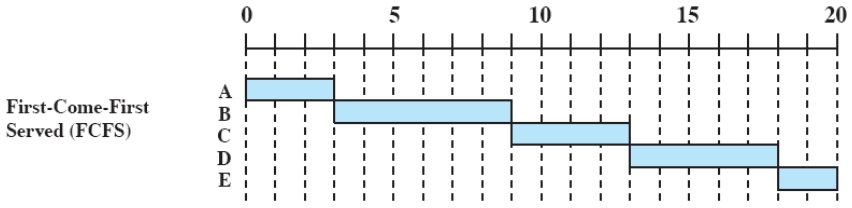
FCFS – first come, first served

Politika bez prekidanja. Proces se stavlja u red spremnih procesa. Kada tekući proces prestane da se izvršava bira se proces koji je najduže bio u redu čekanja.

Kratki procesi moraju dugo da čekaju da se izvrše. Kada dočekaju izvršiće se za kratko vreme. Loš je odnos vremena prolaska I vremena usluge.

Bolje prolaze procesi koji intenzivno koriste procesor. Procesi sa dosta U/I kratko koriste procesor, a nakon U/I ponovo moraju dugo da čekaju procesor.

FCFS nije dobra politika sama po sebi ali je u kombinaciji sa prioritetima efektivna.



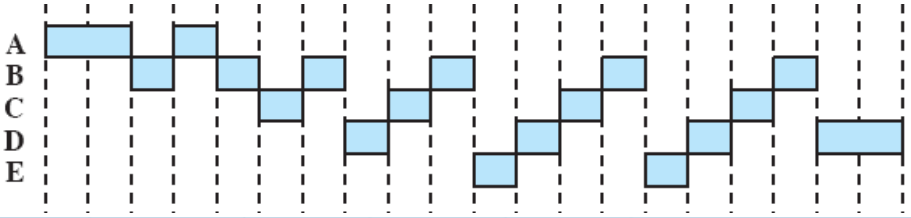
Proces	A	B	C	D	E	Prosek
Vreme završetka	3	9	13	18	20	
Vreme prolaska zadatka (T_r)	3	7	9	12	12	8.60
T_r / T_s	1.00	1.17	2.25	2.40	6.00	2.56

KRUŽNO DODELJIVANJE

RR – round robin

Politika sa prekidanjem. Deljenje na vremenske isečke (*time slicing*). Svaki proces dobija isečak vremena pre nego što bude prekinut. Generator takta periodično izaziva prekid. Ako je istekao vremenski isečak trenutno izvršavani proces se stavlja u red čekanja a sledeći spremni proces se bira na osnovu FCFS.

Round-Robin (RR), $q = 1$

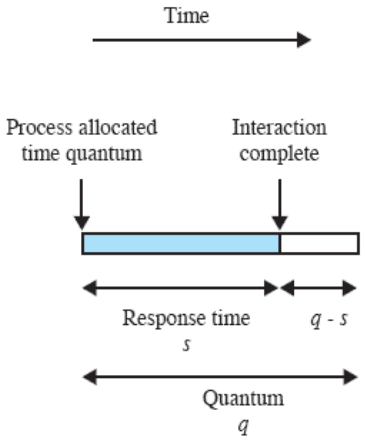


Proces	A	B	C	D	E	Prosek
Vreme završetka	4	18	17	20	15	
Vreme prolaska zadatka (T_r)	4	16	13	14	7	10.80
T_r / T_s	1.33	2.67	3.25	2.80	3.50	2.71

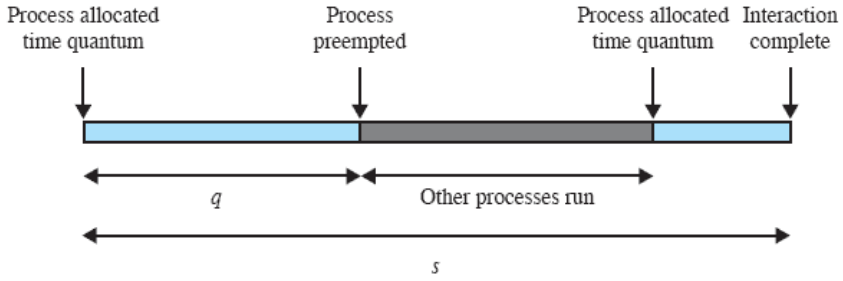
Bolje prolaze procesi sa intenzivnom upotrebom procesa jer iskoriste ceo vremenski isečak I odmah se postave u red spremnih procesa. Procesi sa dosta U/I kratko koriste isečak, blokiraju se zbog U/I i kada se završi U/I dodaju se u red spremnih.

Kako izabrati dužinu vremenskog isečka? Ako je isečak suviše mali, dolazi do čestih prekidanja I komutacije što uvodi usporenje. Ako je isečak suviše veliki, RR se degeneriše u FCFS. Isečak bi trebalo da bude neznatno veći od prosečnog vremena za interakciju sa procesorom (10-100ms). Na taj način, proces uglavnom obavi posao ili se blokira zbog U/I pre isteka vremenskog isečka. Procena je da isečak treba da bude toliki da 80% CPU-bursts bude kraće od isečka.

Ako je isečak veći od vremena za interakciju sa procesorom



Ako je isečak manji od vremena za interakciju sa procesorom

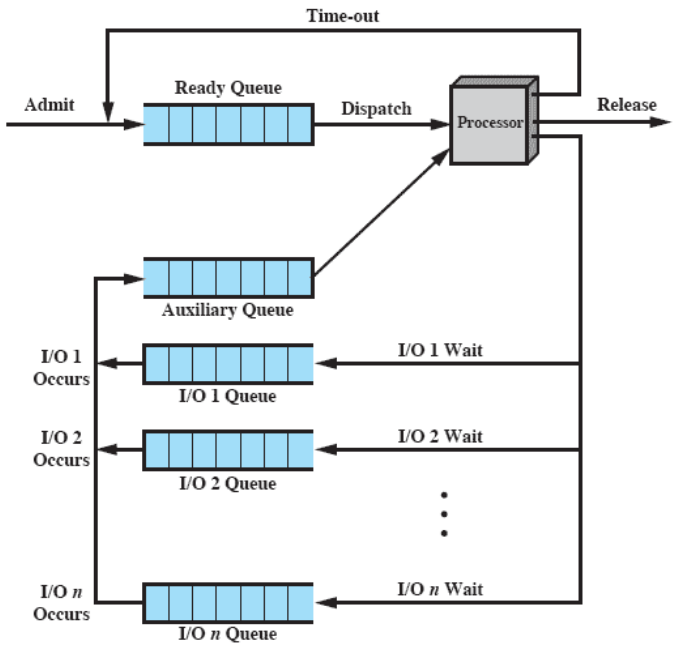


VIRTUELNO KRUŽNO DODELJIVANJE

VRR – *virtual round robin*

Rešava nepravedan tretman procesa sa dosta U/I kod klasičnog RR algoritma.

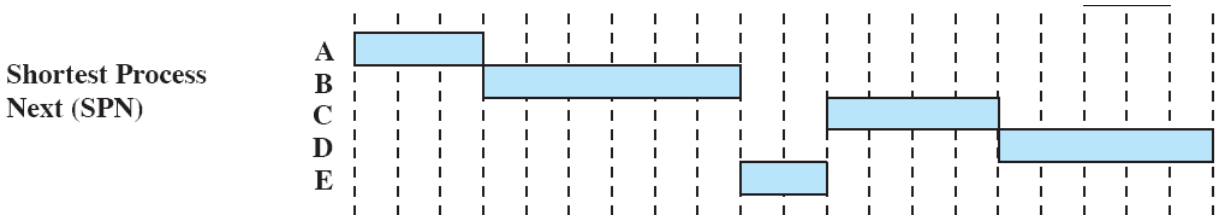
Nakon završetka U/I operacije, proces se ubacuje u poseban FCFS red čekanja, dispečer daje prednost procesima iz ovog reda u odnosu na glavni red spremnih procesa, a procesi Iz pomoćnog reda dobijaju isečak vremena umanjen za vreme koliko su se prethodni put izvršavali kad su uzeti iz reda spremnih procesa.



NAJKRAĆI PROCES SLEDEĆI

SPN – *shortest process next*

Politika bez prekidanja. Bira se proces sa najkraćim očekivanim CPU-burst. Kratak proces skače na čelo reda ispred drugih procesa.



Proces	A	B	C	D	E	Prosek
Vreme završetka	3	9	15	20	11	
Vreme prolaska zadatka (T_r)	3	7	11	14	3	7.60
T_r / T_s	1	1.17	2.75	2.80	1.50	1.84

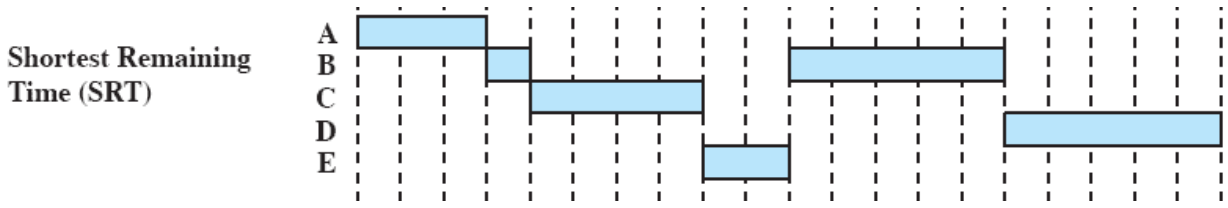
Smanjuje se predvidljivost vremena prolaska, posebno za duže procese. S druge strane, dužim procesima preti gladovanje, jer kraći procesi stalno uskaču ispred njih u redu čekanja. Nije poželjna za sisteme sa deljenjem vremena jer ne podržava prekide.

**Potrebno je proceniti koliki će sledeći CPU-burst biti za proces – jednostavno ili eksponencijalno usrednjavanje*

NAJKRAĆE PREOSTALO VREME

SRT – *shortest remaining time*

Politika sa prekidanjem. Prekidna varijanta SPN – kada se pojavi novi proces u redu spremnih procesa, dobiće procesor umesto tekućeg procesa ako ima kraći očekivani CPU-burst.



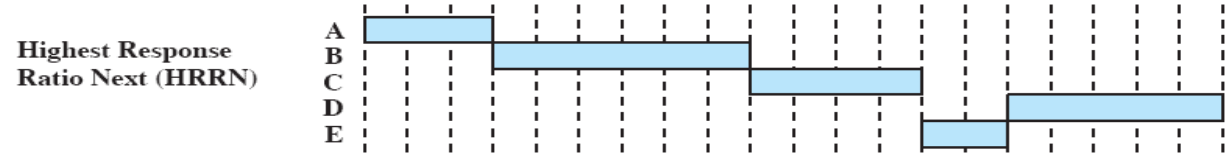
Proces	A	B	C	D	E	Prosek
Vreme završetka	3	15	8	20	10	
Vreme prolaska zadatka (T_r)	3	13	4	14	2	7.20
T_r / T_s	1	2.17	1	2.80	1	1.59

Prednost je što nema dodatnih prekida – nema potrebe za prekidima u svakom taktu jer je već najkraći proces dobio procesor. Prekid nastaje samo pojavom novog procesa. Samo novi proces u redu može da ima kraći CPU-burst. Problem je dodatna režija za evidenciju I predviđanje CPU-burst. U suštini radi veoma dobro ako se dobro predvidi CPU-burst.

SLEDEĆI SA NAJVEĆIM ODNOSOM ODZIVA

HRRN – *highest response ratio next*

Politika bez prekidanja. Za svaki proces se računa proporcija ***R = (w + s) / s*** gde je ***w*** vreme koje je proces proveo u čekanju na procesor do sada a ***s*** očekivano vreme usluživanja. Kada se tekući proces završi ili blokira, bira se proces sa najvećom vrednošću **R**.

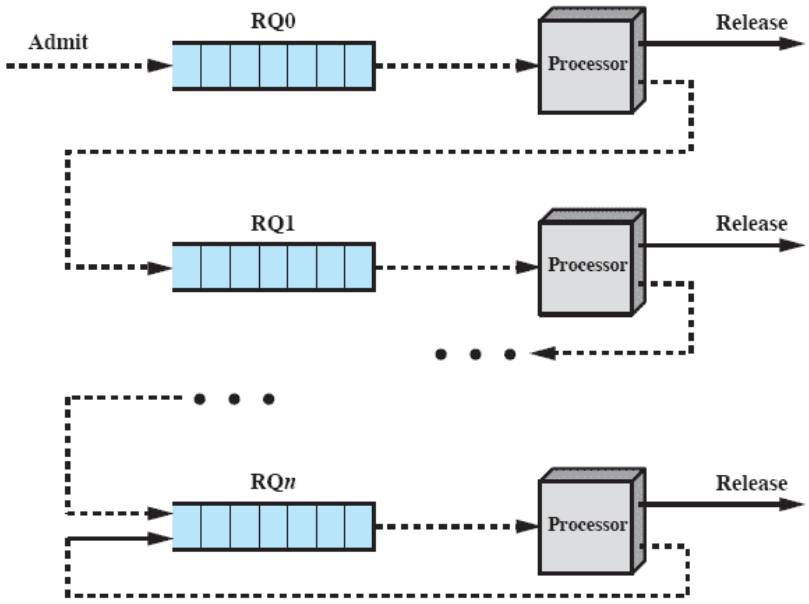


Proces	A	B	C	D	E	Prosek
Vreme završetka	3	9	13	20	15	
Vreme prolaska zadatka (T_r)	3	7	9	14	7	8.00
T_r / T_s	1	1.17	2.25	2.80	3.5	2.14

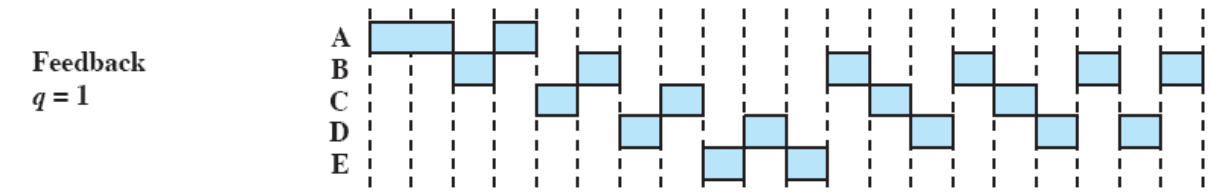
Pruža dobar tretman kako za kraće tako I za procese koji dugo čekaju, tako da će se približno ravnopravno tretirati duži I kraći poslovi.

POVRATNA SPREGA

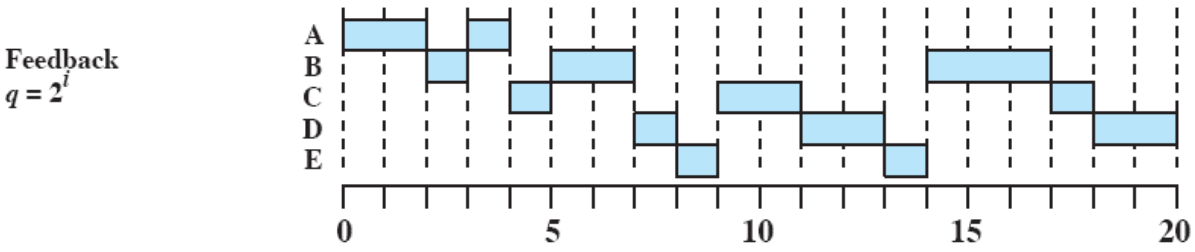
Prekidna politika. Svaki put kada se proces prekine stavlja se u FCFS red čekanja nižeg prioriteta. Procesor brže dobijaju noviji procesi, a brže prolaze kraći procesi.



Kada je kvantum svima isti, drugi procesi mogu da gladuju



Kada se kvantum menja dinamički (obrnuto proporcionalan prioritetu), procesi iz manje prioritetnih redova imaju šanse da duže koriste procesor kada ga dobiju.



RASPOREĐIVANJE SA FER DEOBOM

FSS – *fair share scheduler*

Procesi se klasifikuju u grupe a raspoređivanje se vrši uzimajući u obzir grupu kojoj proces pripada. Prioritet procesa se izračunava na osnovu osnovnog prioriteta, dotadašnjeg vremena korišćenja procesora od strane procesa I od strane grupe kojoj taj proces pripada.

LINUX RASPOREĐIVAČ

Completely fair scheduler

Zasniva se na ideji da se procesima dodeljuje procenat procesorskog vremena u zavisnosti od trenutnog broja procesa u sistemu I prioriteta procesa. Vremenski isečak se dodeljuje dinamički da bi se postigla planirana proporcija u korišćenju ukupnog procesorskog vremena.

U sistemu sa N procesa istog prioriteta, svaki proces bi dobio 1/N procesorskog vremena. Prioriteti služe kao težinski faktor pri raspodeli procesorskog vremena.

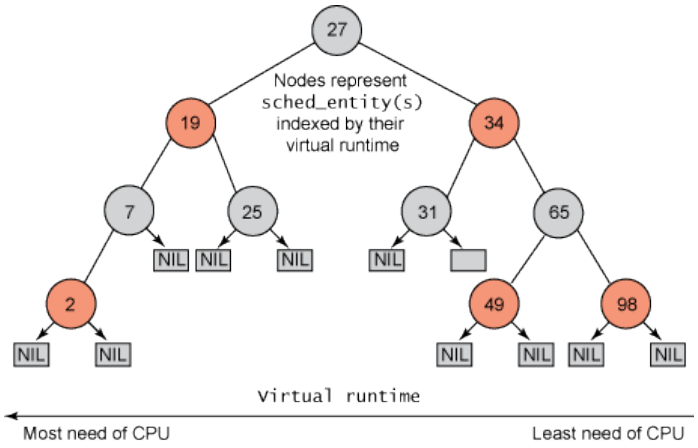
Maksimalno kašnjenje – konfigurabilna vrednost koja određuje dužinu vremenskog intervala u kojem svaki proces mora dobiti proces bar jednom.

Primer: Maksimalno dozvoljeno kašnjenje je 10 ms, dva procesa u sistemu sa odnosom prioriteta 4:1. Prvi proces dobija isečak 8 ms, a drugi 2 ms .

Sa prevelikim brojem procesa u sistemu, isečak bi postao veoma mali a troškovi komutacije preveliki.

Minimalna granularnost – vrednost koja određuje minmalnu dužinu isečka bez obzira na broj procesa. Ovim se za veliki broj procesa gubi ravnopravnost po cenu upotrebljivosti sistema.

Način odabira sledećeg procesa se ne zasniva na organizovanju procesa u redove, već se procesima dodeljuje brojna vrednost u vidu virtuelnog vremena izvršavanja. Virtuelna vremena izvršavanja se organizuju u binarno stablo pretrage, a za sledeći proces se bira onj koji ima najmanje virtuelno vreme izvršavanja (krajnji levi proces u stablu).



9 – Multiprocesorsko raspoređivanje

KLASIFIKACIJA MULTIPROCESORSKIH SISTEMA

- LABAVO SPREGNUTI PROCESORI – zbirka relativno autonomnih procesora gde svaki ima svoju memoriju I U/I kanale
- FUNKCIONALNO SPECIJALIZOVANI PROCESORI – asimetrični multiprocesor, jedan glavni procesor opšte namene. Glavni procesor rukuje skupom procesora od kojih svaki pruža specijalizovanu uslugu.
- **ČVRSTO SPREGNUTI MULTIPROCESOR** – simetrični multiprocesor. Skup ravnopravnih procesora koji dele glavnu memoriju. Pod integrisanim su upravljanjem operativnog sistema.

TRI VAŽNA PITANJA RASPOREĐIVANJA ZA MULTIPROCESOR:

1. Dodeljivanje procesa procesorima
2. Upotreba multiprogramiranja na pojedinačnom procesoru – da li da više procesa konkuriše za izvršavanje na procesoru
3. Stvarno raspoređivanje procesa – način izbora procesa koji će sledeći da se izvršava na procesoru

Izbor zavisi od broja procesora I stepena interakcije između procesa.

DODELJIVANJE PROCESA PROCESORIMA

- **STATIČKO DODELJIVANJE** – Proces se predefinisano dodeli određenom procesoru. Imamo manje režije oko raspoređivanja. Takođe je moguće grupno raspoređivanje tj. moguće je rasporediti grupu procesa na isti namenski procesor. Mana je nedostatak balansa - jedan procesor može biti besposlen dok je drugi preopterećen.
- **DINAMIČKO DODELJIVANJE** – Proces se u različitim trenucima može izvršavati na različitim procesorima. Jedno rešenje je zajednički red čekanja gde svi procesi ulaze I raspoređuju se na bilo kakav raspoloživi procesor. Drugo rešenje je odvojen red čekanja za svaki procesor pri čemu proces može da menja redove (često rešenje u savremenim OS).

Kod oba pristupa potreban je softver koji raspoređuje procese na procesore. Varijante su:

1. **GLAVNI-POMOĆNI** (master – slave)
2. **JEDNAKA RAVNOPRAVNOST** (peer)
3. Pristupi koji kombinuju oba

MASTER – SLAVE MULTIPROCESORSKA ARHITEKTURA

Ključne funkcije kernela se uvek izvršavaju na jednom procesoru (glavni procesor). Pomoćni procesori izvršavaju korisničke programe, dok je glavni procesor odgovoran za raspoređivanje. Pomoćni procesor šalje zahtev glavnom procesoru kada mu je potrebna neka usluga (npr. U/I).

NEDOSTACI: Otkaz glavnog procesora obara ceo sistem. Glavni procesor može postati usko grlo sistema.

JEDNAKA RAVNOPRAVNOST

Kernel može da se izvršava na svakom procesoru, I svaki procesor može da vrši raspoređivanje. Zahteva složeniji OS jer se mora voditi računa da dva procesora ne odaberu isti proces.

SKLONOST PROCESA ZA PROCESOR

Pri izvršavanju procesa, podaci se upisuju u keš procesora. Cilj je da naredno raspoređivanje procesa može da iskoristi ranije keširane podatke. Zato se nastoji da proces ne menja procesor na kojem se prethodno izvršavao.

RASPODELA OPTEREĆENJA

load balancing

Važno je rasporediti procese na procesore tako da se poveća iskorišćenost svakog procesora.

Raspoređivanje opterećenja ravnopravno na procesore u SMP sistemu. Potrebno samo u sistemima gde svaki procesor ima poseban red spremnih procesa. Negativno utiče na sklonost procesa za procesor.

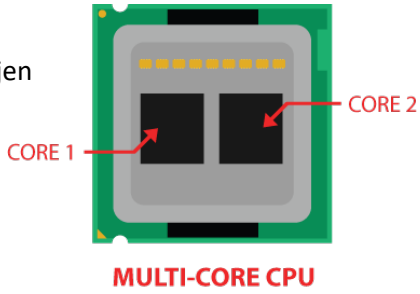
REALIZACIJA RASPODELE OPTEREĆENJA:

- **PUSH** (postavljanje procesa) – poseban deo OS periodično proverava opterećenje na procesorima I prebacuje proces iz reda jednog procesora u red drugog, manje opterećenog procesora.
- **PULL** (preuzimanje procesa) – nezaposleni procesor povlači kod sebe neki proces iz reda opterećenog procesora.

Push I Pull strategija se međusobno ne isključuju – Linux izvršava svakih 200ms kod koji radi raspodelu opterećenja (*push*), isti kod se izvršava I ako nakon izvršavanja procesa red spremnih procesa na tom procesoru ostane prazan (*pull*).

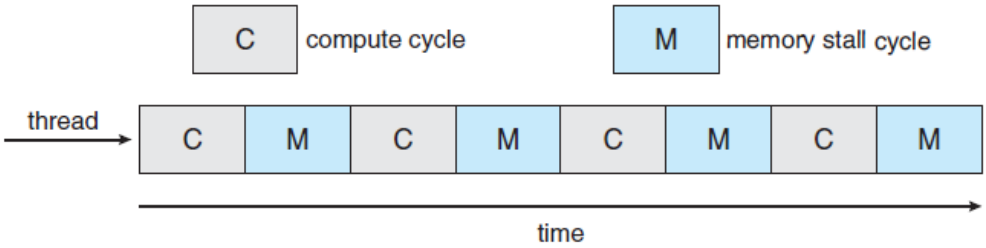
PROCESOR SA VIŠE JEZGARA

Više procesorskih jezgara na istom čipu. Svako jezgro je zapravo odvojen procesor.

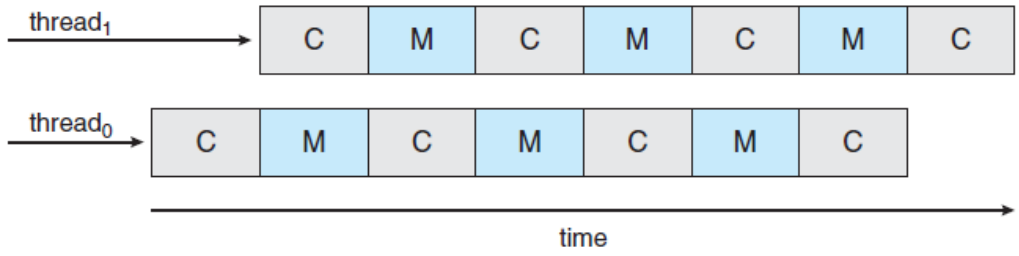


HARDVERSKE NITI PROCESORSKOG JEZGRA

USPORENJE MEMORIJE – procesor pri izvršavanju instrukcije značajan deo vremena provodi čekajući na dostupnost podataka. Ako ne pronađe podatak u kešu, mora da pristupa nižim nivoima keša ili radnoj memoriji.



Dok podatak ne bude učitan, procesor može da izvršava druge instrukcije. Kreiraju se posebne hardverske niti na jednom jezgru procesora – dok jedna čeka na memoriju, druga se izvršava. Svaka hardverska nit se ponaša kao logički procesor. Sistem sa dva jezgra sa po dve hardverske niti OS vidi kao 4 procesora.



OS vrši raspoređivanje softverskih niti procesa na hardverske niti jezgra.

RASPOREĐIVANJE SOFTVERSKIH NITI

Aplikacija se može organizovati kao skup niti. Niti sarađuju I izvršavaju se konkurentno u istom adresnom prostoru. Značajno poboljšanje performansi dobija se u multiprocesorskom sistemu u odnosu na jednoprocesorski sistem jer se niti istovremeno izvršavaju na različitim procesorima.

4 PRISTUPA ZA MULTIPROCESORSKO RASPOREĐIVANJE NITI:

1. **DELJENJE OPTEREĆENJA**

   Niti se ne dodeljuju posebnom procesoru već se opterećenje ravnomerno raspodeljuje po procesorima preko centralizovanog raspoređivača ili se rutina raspoređivanja izvrši na procesoru kada postane raspoloživ za preuzimanje nove niti. Red čekanja se može organizovati po nekom od algoritama korišćenih kod jednoprocesorskog raspoređivanja.

   NEDOSTACI: Malo je verovatno da će prekinuta nit nastaviti izvršavanje na istom procesoru, pa rad sa keš memorijom postaje manje efikasan. Takođe, malo je verovatno da će sve niti jedne aplikacije istovremeno dobiti pristup procesorima (nedovoljno iskorišćene prednosti paralelizma)

2. **GRUPNO RASPOREĐIVANJE -** Skup povezanih niti se raspoređuje za izvršavanje u isto vreme na skup procesora. Može da poboljša performanse jer su komutacije procesora ređe – nit se brže sinhronizuje sa drugom niti ukoliko se obe izvršavaju. Takođe, nije potrebno da se vrši komunikacija da bi druga nit dobila procesor.
3. **DODELJIVANJE NAMENSKOG PROCESORA -** Ekstremni oblik grupnog raspoređivanja. U startu se za svaku nit odredi procesor na kojem se nit izvršava od početka do kraja. Nema multiprogramiranja – neki od procesa mogu biti besposleni.

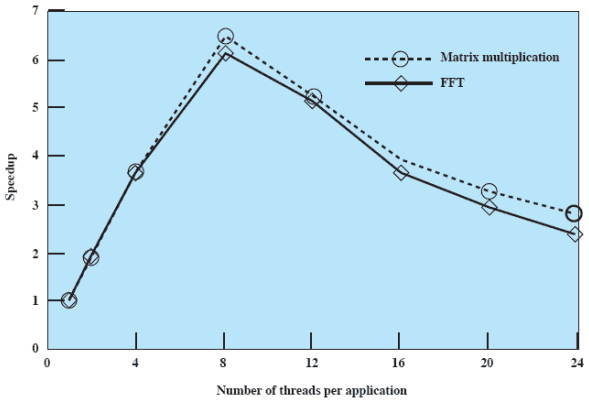
   Može drastično da ubrza performanse u veoma paralelnom sistemu gde je važnija efektivnost od iskorišćenja procesora. Potpuno odsustvo komutacije procesa dodatno ubrzava program.

4. **DINAMIČKO RASPOREĐIVANJE -** Broj niti u procesu može da se promeni u toku izvršavanja. Aplikacija dinamički menja broj niti zavisno od broja procesora koji su joj trenutno dodeljeni. Nije zaživelo u stvarnim sistemima.

Ubrzanje aplikacije zavisno od broja niti:

16 procesora – 2 aplikacije (množenje matrica I Furijeove transformacije)

Najbolje performanse kada je u obe aplikacije po 8 niti jer svaka nit tada ima svoj procesor. Povećanje broja niti usporava programe zbog komutacije.



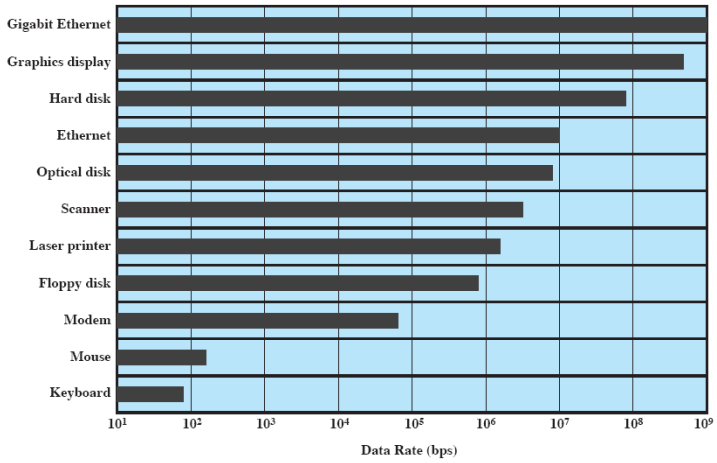
10 – U/I Upravljanje i raspoređivanje diska

Kategorije U/I uređaja:

- Za komunikaciju sa korisnikom računara – *tastatura, miš, štampač …*
- Za komunikaciju sa hardverom računara – *diskovi, senzori, kontroleri …*
- Za komunikaciju sa udaljenim uređajem – *modemi, mrežne kartice*

RAZLIKE IZMEĐU U/I UREĐAJA

☐ **Brzina prenosa podataka**



☐ **Uloga u sistemu** – utiče na podršku OS za uređaj

☐ **Složenost upravljanja** – neki uređaju zahtevaju jednostavno upravljanje, a neki složeno (*disk*). Od složenosti upravljanja zavisi složenost U/I modula OS za taj uređaj.

☐ **Jedinica prenosa podataka** – orijentisani na blokove *(jedinica prenosa je blok*) ili orijentisani na tokove (*podaci se prenose jedan po jedan kao tok bajtova*)

☐ **Predstavljanje podataka** – način kodovanja podataka (*format, parnost...*)

☐ **Upravljanje greškama** – različiti načini reakcije na grešku i skupovi kodova grešaka

☐ **Način pristupa –** sekvencijalni (*uređaj šalje/prima podatke u fiksnom redosledu – ruter, štampač*) ili proizvoljan (*moguće je uputiti zahtev uređaju da se pozicionira na bilo koju lokaciju na kojoj se nalaze podaci - disk*)

☐ **Podrška za upis i čitanje –** Upis i čitanje, samo upis ili samo čitanje

ZAKLJUČAK: Potreban je jednoobrazan mehanizam za upravljanje raznolikim skupom U/I uređaja

U/I HARDVER

Uređaj razmenjuje podatke sa računarom putem nekog prenosnog medijuma (*kabl, vazduh*). Uređaj je povezan sa računarom kroz PORT (*kada jedan uređaj koristi komunikacioni kanal*) ili MAGISTRALU (*kada više uređaja deli isti skup kanala*).

KONTROLER – hardver koji upravlja portom, magistralom ili uređajem

HOST ADAPTER – kontroler u računaru zadužen za komunikaciju sa uređajem

DEVICE CONTROLLER – kontroler u uređaju namenjen za upravljanje uređajem

KOMUNIKACIJA PROCESORA SA U/I HARDVEROM

PRVA VARIJANTA – Procesor izvršava specijalnu U/I instrukciju koja direktno upisuje bitove u registre konrolera (ili ih čita iz registra)

DRUGA VARIJANTA – Memmory mapped I/O – proces upisuje podatke u radnu memoriju, a lokacije u radnoj memoriji su namapirane na registre kontrolera.

Ove dve varijante se mogu kombinovati – za jednostavnije upravljanje direktni pristup, a za veću količinu podataka druga varijanta.

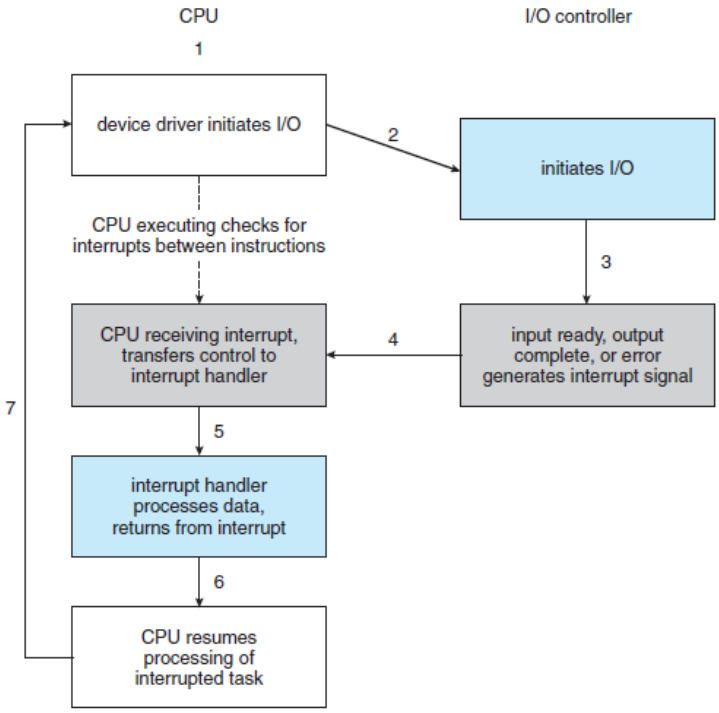
REGISTRI U/I KONTROLERA

- **DATA-IN** – iz njega procesor preuzima podatke
- **DATA-OUT** – u njega procesor upisuje podatke za U/I uređaj
- **STATUS** – informacija o statusu operacije za procesor (da li je završena, kod greške …)
- **UPRAVLJAČKI REGISTAR** – u njega procesor upisuje komandu za U/I uređaj

ORGANIZACIJA U/I FUNKCIJE

PROGRAMIRANI U/I – Procesor na zahtev procesa izdaje U/I komandu. Dok U/I modul obavlja funkciju, proces se nalazi u 'busy waiting' dok se U/I operacija završi. Proces stalno proverava da li je operacija završena.

U/I POKRETAN PREKIDOM – Procesor na zahtev procesa izdaje U/I komandu. Ako U/I instrukcija nije blokirajuća, procesor nastavlja da izvršava instrukcije procesa. Ako je U/I funkcija blokirajuća, proces se stavlja u stanje blokiran I raspoređuje se drugi proces. U/I uređaj postavlja prekid kada završi operaciju. Rutina za obradu prekida (ISR) se izvršava kao reakcija na završenu U/I operaciju.



ISKLJUČIVANJE PREKIDA

Prekidi se isključuju dok traje neka kritična operacija koja ne sme biti prekinuta, najčešće dok se obrađuje prethodni prekid. Procesor ima bit koji označava da li su prekidi uključeni. Vrednost ovog bita može se postaviti privilegovanom instrukcijom. Bit se odnosi samo na standardne prekide.

Specijalni prekidi se ne mogu isključiti – ovi prekidi se aktiviraju u slučaju fatalnih grešaka sistema.

U/I POKRETAN PREKIDOM

Kada uređaj postavi prekid, potrebno je izvršiti rutinu za obradu prekida. U drajveru uređaja nalazi se kod za obradu prekida. Svaki drajver registruje svoju rutinu.

VEKTOR PREKIDA – niz koji sadrži adrese rutina za obradu prekida. Koristi se kako bi procesor znao koji uređaj je postavio prekid. Pri signalizaciji prekida uređaj postavlja broj (adresu) koji ukazuje na tip prekida. Na osnovu adrese koju prekid postavi indeksira se element vektora prekida.

Obično vektor prekida ima manje elemenata nego što ima uređaja – tada element vektora sadrži adresu liste obrađivača prekida. Skeniraju se svi elementi liste da se nađe odgovarajući obrađivač. OS pri inicijalizaciji skenira uređaje I postavlja adrese obrađivača prekida u vektor prekida.

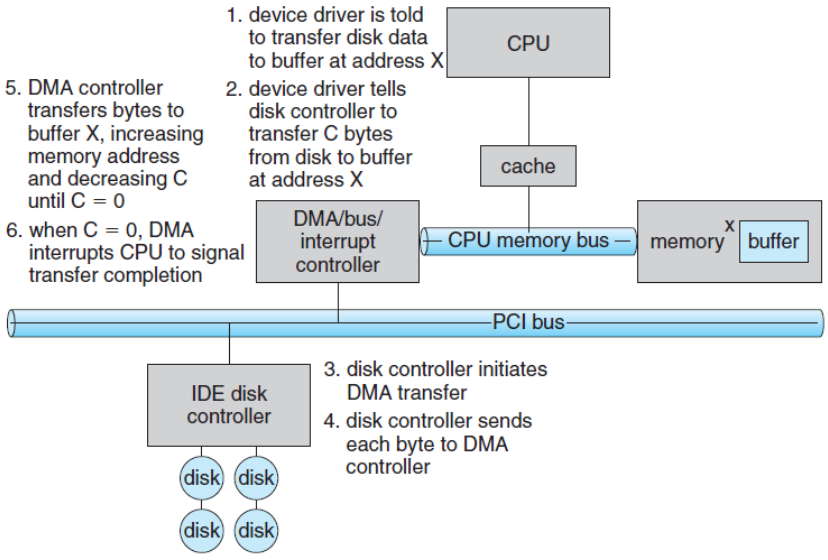
NIVOI PRIORITETA PREKIDA – Može da postoji I više nivoa prioriteta prekida. Obrada prekida nižeg prioriteta može biti prekinuta zbog obrade prekida višeg prioriteta.

DIREKTAN PRISTUP MEMORIJI – DMA

Procesor može da delegira izvršavanje U/I operacija specijalizovanom procesoru – DMA kontroleru.

DMA kontroler upravlja razmenom podataka između glavne memorije I U/I modula. Procesor šalje DMA modulu zahtev za prenost bloka podataka adresa sa koje se podaci prenose, adresa na koju se podaci prenose I broj bajtova za prenos. DMA modul postavlja prekid kada završi prenos.

Koraci u DMA prenosu:



RAZVOJNE FAZE U/I FUNKCIJE

1. **Procesor direktno upravlja U/I uređajem**
2. **Dodaje se kontroler (U/I modul)** – koristi se programirani U/I bez prekida
3. **Ista konfiguracija kao 2. ali sa prekidima** – rezultat je efikasniji sistem jer proces ne mora stalno da proverava završetak U/I operacije
4. **U/I modul može direktno da upravlja memorijom preko DMA** – procesor može da obrađuje druge zadatke dok DMA prenosi ceo blok podataka
5. **U/I modul postaje poseban procesor** – čitav niz U/I aktivnosti obavlja U/I procesor bez uplitanja glavnog procesora
6. **U/I modul dobija i sopstven memoriju** – kao odvojeni računar koji samostalno može da upravlja U/I uređajem uz minimalno angažovanje procesora.

CILJEVI PROJEKTOVANJA U/I PODSISTEMA

- EFIKASNOST

Većina U/I uređaja je izuzetno spora u odnosu na glavnu memoriju i procesor. Multiprogramiranje dozvoljava da se drugi proces izvršava dok jedan proces čeka na U/I.

Razmenjivanje – dobavljaju se novi procesi sa diska da procesor ne bi bio besposlen, najznačajnije je kako je realizovan U/I diska.

- OPŠTOST

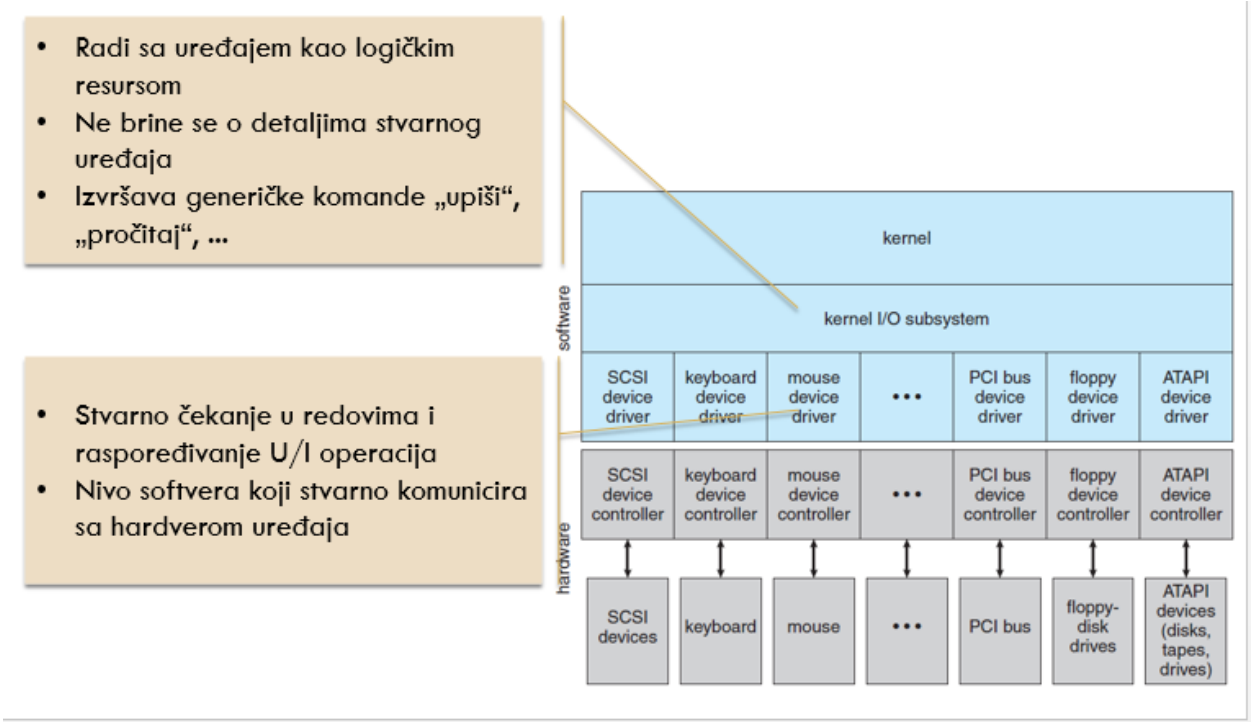
Radi jednostavnosti i obrade grešaka poželjno je upravljati svim U/I uređajima na jedinstven način. Sistem apstrahuje razlike između uređaja tako što identifikuje zajedničke osobine, zato OS koristi generičke funkcije za rad sa U/I. Rutine nižeg nivoa obavljaju operacije nad konkretnim uređajem – drajveri realizovani specifično za svaki uređaj implementiraju generičke funkcije za konkretan uređaj.

HIJERARHIJSKA STRUKTURA

U/I funkcije se organizuju hijerarhijski. Rutine nižeg nivoa sakrivaju detalje uređaja.

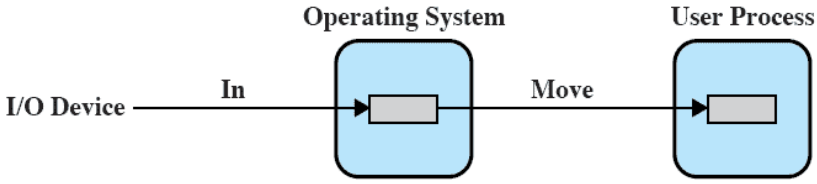
Sistem se organizuje po slojevima. Svaki sloj se oslanja na sledeći sloj da bi izvodio jednostavnije funkcije, a obezbeđuje servise sloju iznad sebe. Promene u jednom sloju ne bi trebalo da zahtevaju izmene u drugim slojevima, pa zato ne treba menjati kod OS da bi se podržao novi uređaj već je dovoljno dodati drajver za uređaj. Drajver se implementira specifično za svaki OS.

Organizacija slojeva u U/I podsistemu



U/I BAFEROVANJE

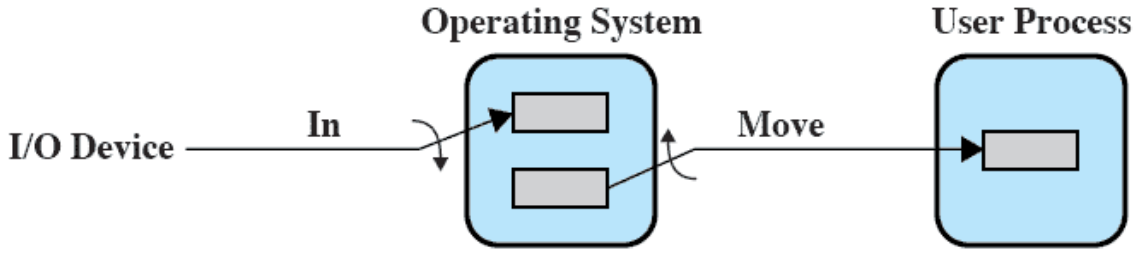
Bafer je prostor u memoriji u koji se skladište podaci koji se prenose od i ka U/I uređaju.



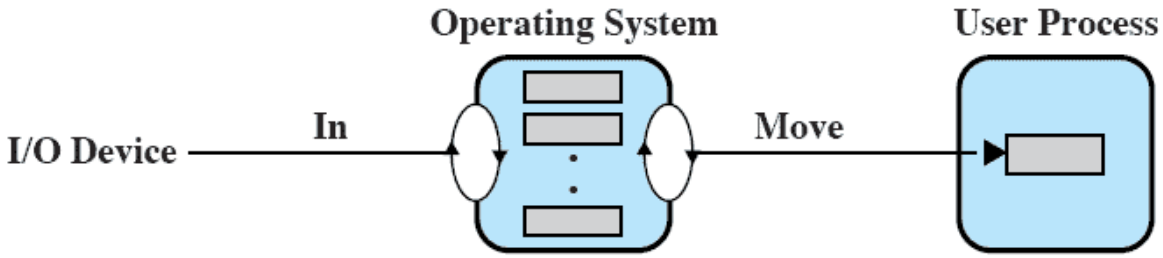
Proces ne mora da čeka na spore U/I operacije. Pri U/I upisu, proces postavi sve podatke odjednom u bafer a kontroler ih šalje ka uređaju, a pri čitanju se najpre svi podaci prihvate u bafer nakon čeka im proces pristupa.

Pošto U/I operacija koristi podatke iz bafera, pri upisu podataka nije problem ako proces promeni podatke nakon izdavanja U/I operacije. Stranica procesa ne mora u toku U/I operacije ni da bude u radnoj memoriji – može biti zamenjena na disk.

DVOSTRUKI BAFER – uvode se dva bafera. Proces učitava podatke iz jednog, a za to vreme OS učitava sledeći blok u drugi bafer. Zatim proces pređe na drugi bafer a OS se vrati na prvi i tako redom.

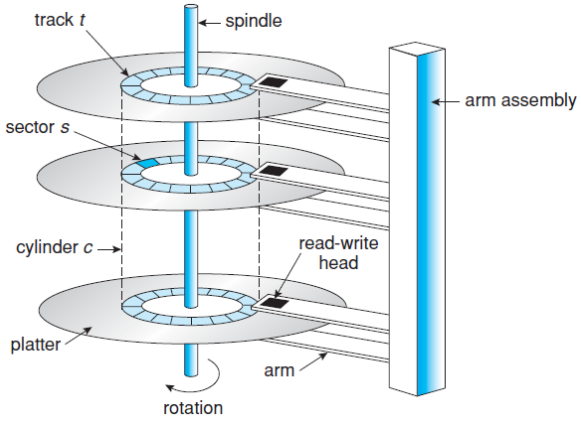


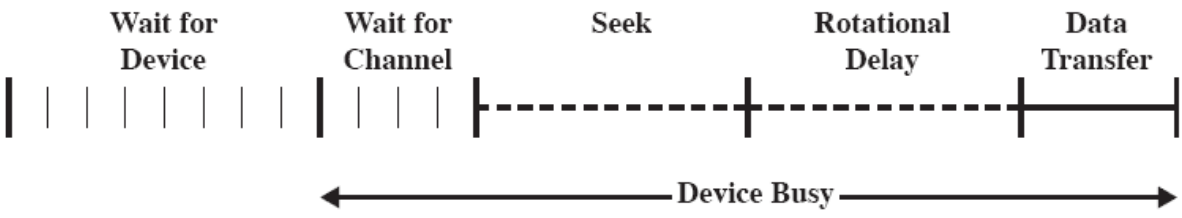
KRUŽNI BAFER – dalje proširenje ideje dvostrukog bafera. Uvodi se N bafera, redom U/I uređaj puni/prazni bafere dok OS čita/upisuje iz njih. Nakonrada sa poslednjim baferom, vraća se na prvi.



RASPOREĐIVANJE DISKA

Performanse diska su izuzetno značajne za performanse celog sistema jer je disk puno sporiji od radne memorije. Često se pristupa disku zbog pristupa fajlova ili stranica procesa u virtuelnoj memoriji.





PARAMETRI PERFORMANSE DISKA – više operacija mora da se izvede za svaki zahtev prenosa podataka od/ka disku

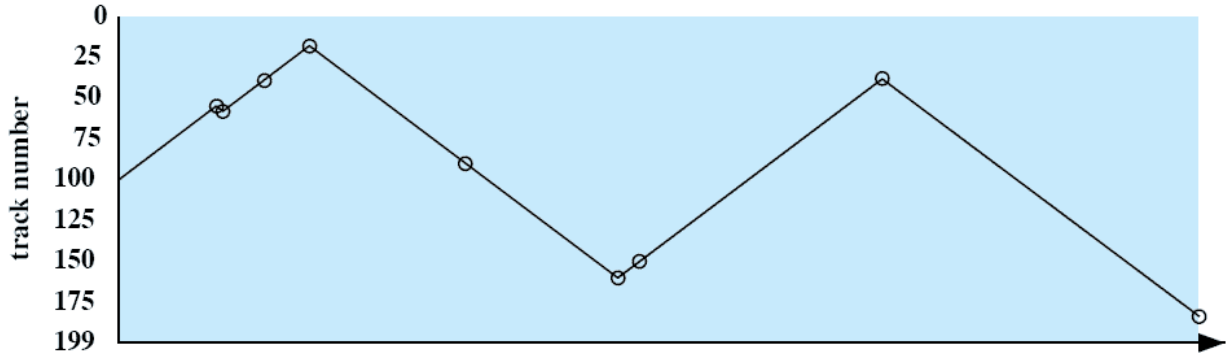
- **ČEKANJE NA UREĐAJ** – čekanje da uređaj postane raspoloživ procesu
- **ČEKANJE NA U/I KANAL** – čekanje da kanal postane raspoloživ ako se kanal deli sa drugim U/I uređajima
- **VREME POZICIONIRANJA** – vreme potrebno da se glava diska pomeri na posebnu stazu
- **ROTACIONO KAŠNJENJE** – vreme potrebno da disk rotira tako da traženi sektor bude ispod glave
- **VREME PRENOSA** – vreme potrebno da se podaci pročitaju/upišu tako što se sektor kreće ispod glave rotiranjem diska

POLITIKE RASPOREĐIVANJA DISKA

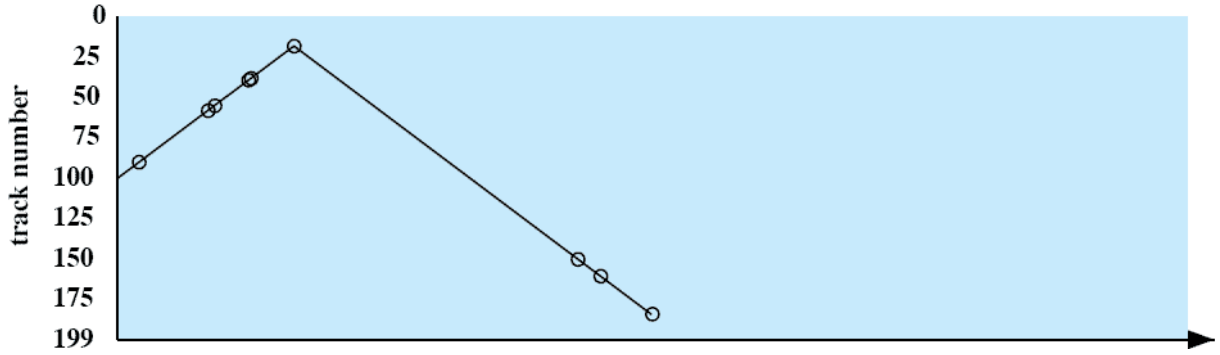
OS može da utiče na vreme pozicioniranja organizacijom zahteva

Primer: Disk ima 200 staza a glava je pozicionirana na stazi 100. Zahteva se pristup stazama u redosledu 55, 58, 39**,** 18, 90, 160, 150, 38, 184

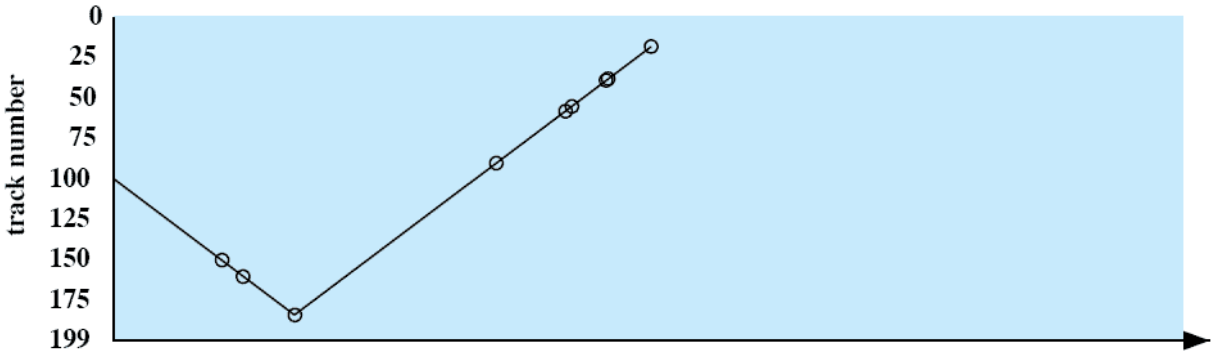
1. **FIFO** – stavke iz reda čekanja obrađuju se redom. Fer prema svim procesima ali loše performanse jer se zahtevi ne organizuju



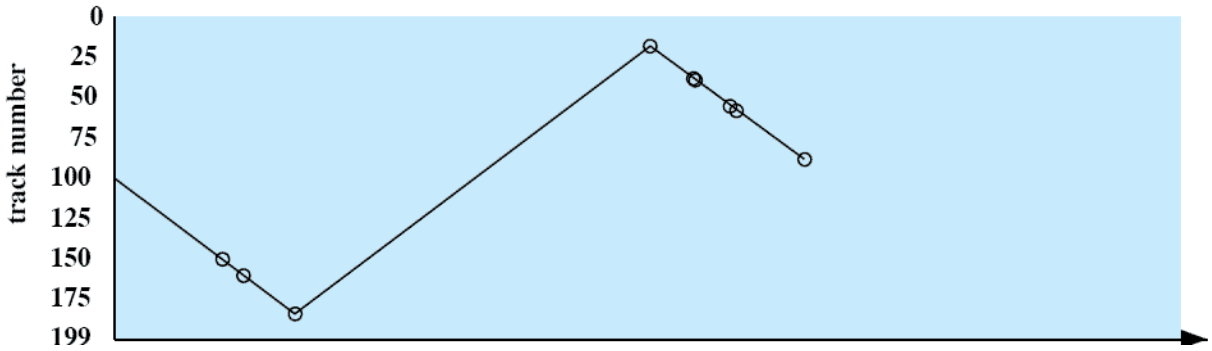
2. **PRIORITET** – određenoj vrsti poslova daje se veći prioritet pa oni imaju prednost
3. **SSTF** – bira se zahtev za koji je potrebno najmanje pomeranje ručice diska, minimizira se vreme pozicioniranja



4. **SCAN** – prioritet i SSTF mogu da dovedu do gladovanja zahteva. SCAN politika pomera ručicu samo u jednom smeru i zadovoljava redom sve zahteve sve dok ne dođe do poslednje staze, a tada menja smer i postupak se ponavlja



5. **C-SCAN** – SCAN dovodi do dugog čekanja za staze na periferiji. C-SCAN ograničava skeniranje samo na jedan smer a kada se dođe do kraja, ručica se vrati na početak i onda opet skenira staze



FORMATIRANJE DISKA

FIZIČKO FORMATIRANJE – Deljenje diska u sektore (zaglavlje, podaci, checksum)

PARTICIONISANJE – obavlja ga OS deljenjem diska na delove (particije). Svaka particija je logički disk.

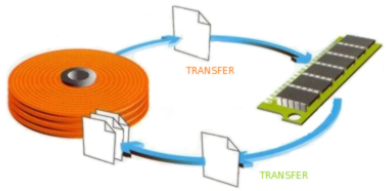
LOGIČKO FORMATIRANJE – inicijalizacija fajl sistema na disku od strane OS

RAW DISK – OS može da napravi particiju bez fajl sistema. Na ovoj particiji podacima se pristupa kao nzu blokova. To omogućava kontrolu niskog nivoa pri upravljanju podacima jer nema fajl sistema kao sloja između i može da bude efikasnije u specifičnim situacijama.

BOOT PARTICIJA – jedna particija mora da sadrži kod OS da bi se sistem mogao startovati. Računar iz ROM memorije pokreće jednostavan program koji identifikuje boot particiju. Startuje kod za učitavanje OS sa predefinisane lokacije (**MASTER BOOT RECORD** – prvi sektor na disku)

PROSTOR ZA RAZMENU – SWAP

Virtuelna memorija koristi disk kao dodatak radnoj memoriji.
Upravljanje ovim prostorom na disku veoma utiče na performanse.

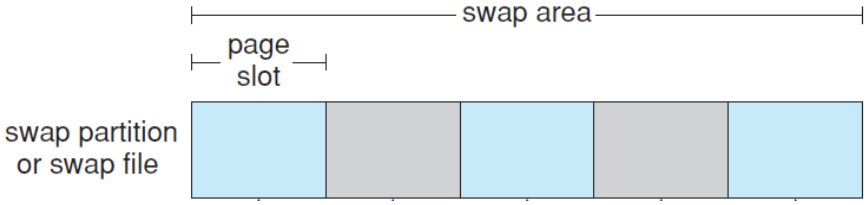


Prostor za razmenu (*swap space*) je prostor na disku namenjen za virtuelnu memoriju.

SADRŽAJ PROSTORA ZA RAZMENU

Izdeljen je na slotove za smeštanje stranica. Postoji više varijanti implementacije:

1. **Kompletna slika procesa** (kod I podaci)
2. **Samo podaci koji se menjaju u toku izvršavanja**
    a. Vrednosti iz stack I heap memorije
    b. Kod I statički podaci su svakako na disku
    c. Ubacuju se u radnu memoriju
    d. Pri izbacivanju nema potrebe upisivati ih I u swap



LOKACIJA PROSTORA ZA RAZMENU

1. **U FAJLU** – koriste se ugrađene funkcije za pristup fajlu. Jednostavno za upravljanje ali neefikasno.
2. **U POSEBNOJ RAW PARTICIJI** – ne koriste se ugrađene funkcije za rad sa fajlovima već specijalizovane funkcije za upravljanje swap prostorom. Efikasnije je ali nefleksibilno za promenu veličine swap prostora.

SSD DISKOVI

Dugoročna memorija koja se može koristiti kao sekundarno skladište umesto magnetnog diska. Pouzdaniji je od HDD jer nema mehaničkih delova. Brži je jer nema kašnjenja zbog rotacije I pozicioniranja. Manjeg je kapaciteta I skuplji je od HDD ali ima isto vreme pristupa za sve delove pa se najčešće koristi prosto FCFS raspoređivanje.

KORIŠĆENJE VIŠE DISKOVA

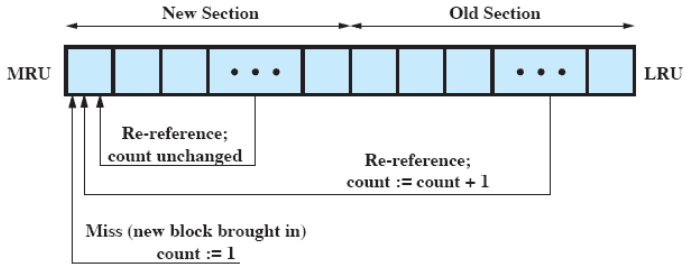
Performanse diska mogu se povećati raspoređivanjem operacija na više diskova paralelno. Podaci se mogu oporaviti pri otkazu diska ako se skladište dodatne informacije. Redundansa zbog poboljšanja pouzdanosti.

RAID – *Redundant array of independent disks* – skup odvojenih fizičkih diskova koje OS vidi kao jedan logički uređaj. Podaci su raspoređeni po svim diskovima a redundantni diskovi se koriste za skladištenje dodatnih informacija za obnovljivost podataka u slučaju otkaza diska.

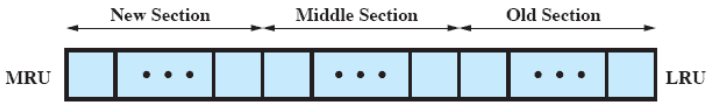
KEŠ DISKA

Bafer u glavnoj memoriji za sektore diska. Sadrži kopiju nekih od sektora diska. Kada se pojavi U/I zahtev za pristup sektoru, najpre se proverava da li je taj sektor u kešu. Postoje različite politike popunjavanja keša:

1. **Zamena najmanje skoro korišćenog** – novi blok se u keš ubacuje na mesto bloka od čijeg je referenciranja proteklo najviše vremena. Za keš postoji stek pokazivača na blokove. Najskorije referencirani blok je na vrhu steka. Kada se blok referencira ili dodaje u keš, postavlja se na vrh steka, a iz keša se izbacuje blok sa dna steka.
2. **Zamena najređe korišćenog** – zamenjuje se blok koji je referenciran najmanji broj puta. Dodeljuje se brojač svakom bloku I inkrementira se svaki put kada se pristupi bloku. Blok sa najmanjom vrednošću brojača se izbacuje iz keša kada je potrebno osloboditi mesto. (*Vrednost brojača može da zavara, ako se blok referencira u retkim trenucima, ali se tada zbog lokalnosti često pristupa bloku. Tada je vrednost brojača velika, iako se blok ne koristi često i verovatno mu se neće uskoro pristupati.*)
3. **Zamena na bazi učestalosti korišćenja** – inicijalna lošija varijanta sa dve sekcije. Kada se referencira blok iz stare sekcije povećava se brojač I prebacuje u novu sekciju. Kada se referencira blok iz nove sekcije, brojač se ne uvećava. Brojač ostaje isti ako se blok u kratkom periodu često referencira.



4. **Zamena na bazi učestalosti korišćenja** – poboljšana varijanta sa tri sekcije. Blokovima u novoj sekciji se brojač ne uvećava. Kandidat za zamenu su blokovi u staroj sekciji. Ovo ima najbolje performanse od svih politika. Ne izbacuju se blokovi koji se relativno često koriste I ne inkrementira se brojač pri čestim referenciranjima zbog lokalnosti.



11 – Upravljanje fajlovima

FAJLOVI – Resursi koji sadrže grupe podataka (*programski kod aplikacije, U/I podaci koji se koriste pri izvršavanju*)

FAJL SISTEM – Skup metoda I struktura podataka koje OS koristi za upravljanje fajlovima. Definiše kako su fajlovi na disku organizovani. (NTFS, FAT, ext2, ext3, ext4…)

POŽELJNE OSOBINE FAJLOVA

1. **Dugoročno postojanje** - Fajlovi se skladište u sekundarnoj memoriji i ne nestaju kada se aplikacija završi ili korisnik odjavi
2. **Deljivost između procesa -** Različiti procesi (ako imaju prava) mogu da koriste iste fajlove
3. **Struktura -** Interna struktura fajla je takva da aplikacijama bude pogodna za korišćenje. Međusobno se fajlovi organizuju u složene strukture

OPERACIJE NAD FAJLOVIMA: KREIRANJE, BRISANJE, OTVARANJE, ZATVARANJE, ČITANJE, PISANJE

ORGANIZACIJA PODATAKA

POLJE – osnovni element podataka, sadrži pojedinačnu vrednost. Definisan dužinom I vrstom podataka

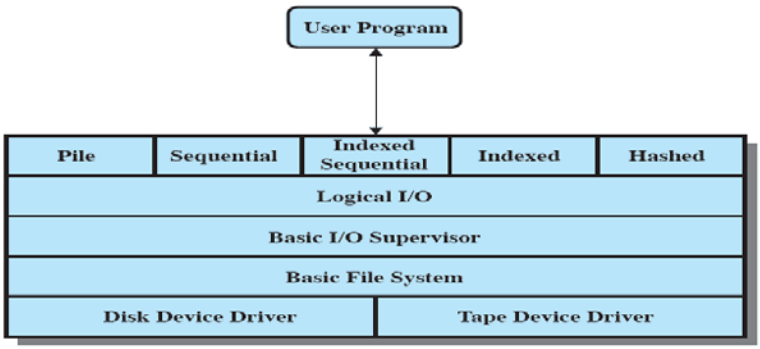
ZAPIS – skup logički povezanih polja. Aplikacija može sa zapisom da postupa kao sa jedinicom.

FAJL – skup logički povezanih zapisa, tretira se kao jedan entitet I adresira se po imenu. Kontrola pristupa je najčešće na nivou fajla

BAZA PODATAKA – struktuiran opis podataka aplikacije, skladišti se kao jedan ili više povezanih fajlova a upravljanje se vrši posebnim softverom koji nije deo OS – SUBP

SISTEM ZA UPRAVLJANJE FAJLOVIMA – obezbeđuje korisnicima I aplikacijama usluge vezan sa upotrebom fajlova. Jedini način da se pristupi fajlovima je kroz sistem za upravljanje fajlovima. Programer ne mora da razmišlja o načinu upravljanja fajlovima na niskom nivou. Ovim je upravljanje fajlovima uniformno za sve aplikacije

SOFTVERSKE KOMPONENTE ZA UPRAVLJANJE FAJLOVIMA



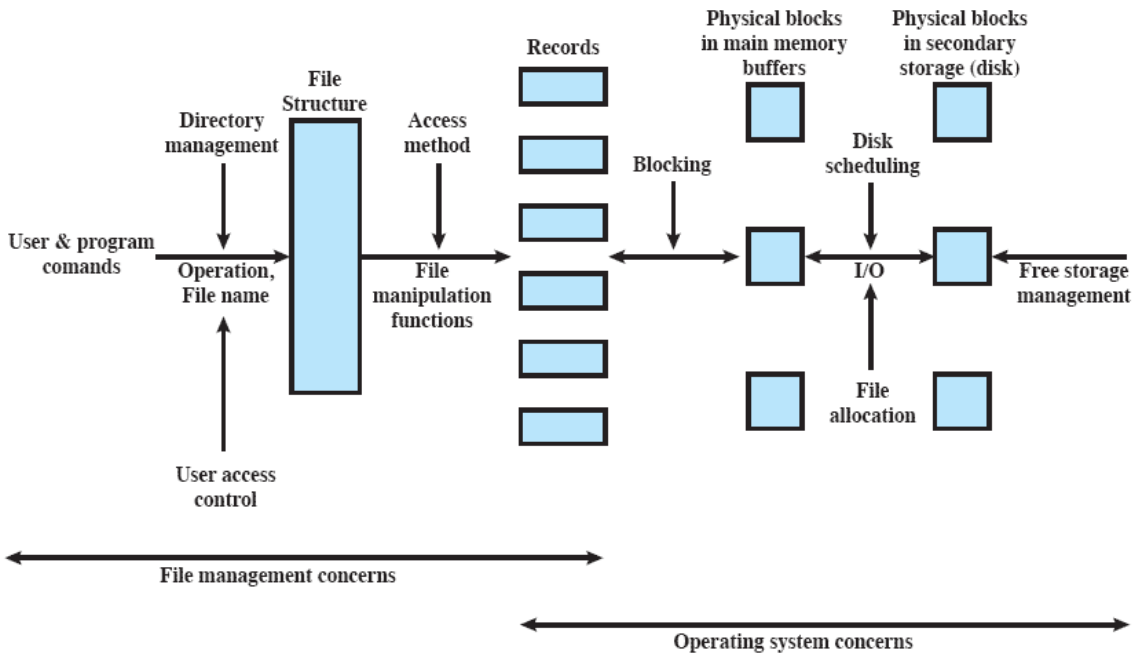
DRAJVERI UREĐAJA – Vrše direktno upravljanje uređajem na najnižem nivou. Započinju U/I operaciju na uređaju I obrađuju rezultat kada uređaj završi operaciju. Deo su OS.

OSNOVNI FAJL SISTEM – još uvek je to nivo fizičkog U/I. Bavi se blokovima podataka koji se razmenjuju sa uređajem (smeštanje blokova na uređaju, baferovanje blokova u glavnoj memoriji). Deo je os.

OSNOVNI U/I SUPERVIZOR – generalna organizacija U/I zahteva za sve uređaje (inicira I završava U/I zahteve, bira uređaj kojem se pristupa, vrši raspoređivanje zahteva za uređaje). Deo je OS.

LOGIČKI U/I – Čini ga sistem za upravljanje fajlovima kao poseban sistemski uslužni program. Apstrakcija upravljanja fizičkim blokovima u sekundarnom skladištu omogućuje korisnicima rad sa fajlovima. Pruža standardni interfejs između aplikacija preko fajl sistema do uređaja.

ELEMENTI UPRAVLJANJA FAJLOVIMA

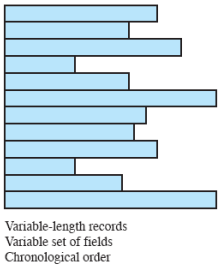


ORGANIZACIJA FAJLOVA – Logičko struktuiranje zapisa koje određuje način na koji se zapisima pristupa. Kriterijumi za izbor organizacije su:

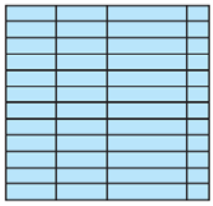
- Vreme pristupa
- Jednostavnost ažuriranja
- Ekonomičnost skladišta
- Jednostavnost održavanja
- Pouzdanost

TIPOVI ORGANIZACIJE FAJLOVA

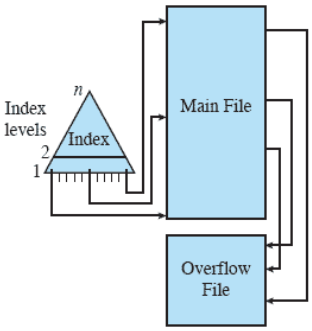
1. GOMILA – podaci se prikupljaju u redosledu u kojem dolaze, nema strukture, zapisi mogu da imaju različita polja I dužinu. Pretraga je komplikovana jer se moraju proći svi zapisi I sva polja.



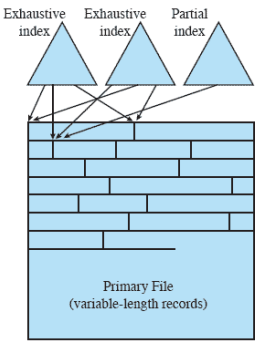
2. SEKVENCIJALNA DATOTEKA – fiksni format zapisa, svi zapisi iste dužine, ista polja u istom redosledu. Zapis ima polje koje je ključ, a zapisi su sortiran po vrednostima ključa. Pretraga je spora jer je sekvencijalnim prolaskom potrebno pronaći zapis sa odgovarajućim ključem. Ažuriranje je komplikovano jer su postojeći zapisi sortirani.



3. INDEKS SEKVENCIJALNA DATOTEKA – zapisi isti kao kod sekvencijalnog ali sadrži 2 proširenja. Indeks koji omogućuje brži pristup – putem ključa u indeksu se dolazi do odgovarajućeg dela glavnog fajla. Fajl prekoračenja – novi zapis ide u poseban fajl a u zapis koji prethodi novim zapisu se upiše pokazivač na novi zapis u fajlu prekoračenja. Periodično se vrši spajanje fajla prekoračenja sa glavnim fajlom.



4. INDEKSNA DATOTEKA – različiti indeksi za različita polja, omogućuju brzo pronalaženje ne samo zapisa nego I polja unutar zapisa. Zapisi mogu biti različitih dužina.



5. DIREKTNA ILI HEŠIRANA DATOTEKA – zapisi imaju ključ, a heš se izračunava na osnovu ključa. Pristupa se lokaciji na koju pokazuje izračunati heš. Na toj lokaciji može biti više zapisa koji imaju isti heš kod. Prolazi se sekvencijalno kroz sve takve zapise da bi se pronašao željeni zapis.

INFORMACIJE O FAJLU

UPRAVLJAČKI BLOK FAJLA

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

OSNOVNE INFORMACIJE:

IME FAJLA – jedinstveno unutar direktorijuma

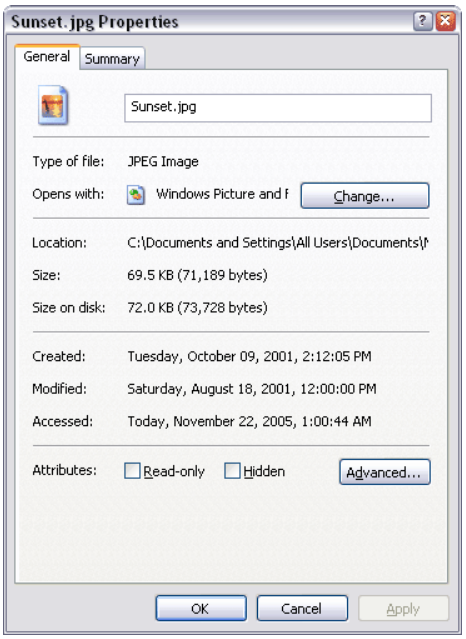
TIP FAJLA – korisniku predstavljeno ekstenzijom

ADRESNE INFORMACIJE:

VOLUMEN – uređaj na kojem je fajl skladišten

POČETNA ADRESA – adresa u sekundarnom skladištu u kojoj je smešten prvi blok fajla

VELIČINA – trenutna veličina fajla



UPRAVLJANJE PRISTUPOM:

VLASNIK – korisnik koji ima specijalna prava u upravljanju fajlom

PRAVA PRISTUPA – za različite korisnike definiše operacije koje mogu da se vrše nad fajlom

INFORMACIJE O UPOTREBI:

- Vreme kreiranja
- Korisnik koji je kreirao fajl
- Vreme poslednjeg čitanja
- Korisnik koji je poslednji izvršio čitanje
- Vreme poslednje modifikacije
- Korisnik koji je poslednji izvršio modifikaciju
- Informacije o trenutnoj upotrebi
    - Koji procesi koriste fajl
    - Da li je fajl zaključan
    - Da li je izmenjen u memoriji

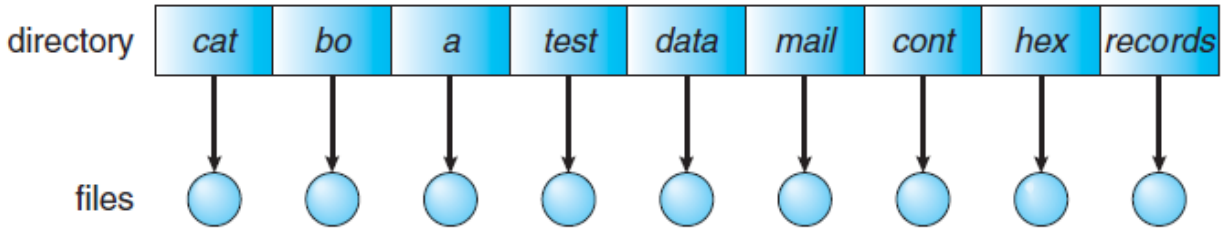
DIREKTORIJUMI FAJLOVA

Fajlovi se evidentiraju u direktorijumima. Direktorijum obezbeđuje mapiranje imena fajlova na fajl resurse na disku. I direktorijum je sam po sebi fajl.

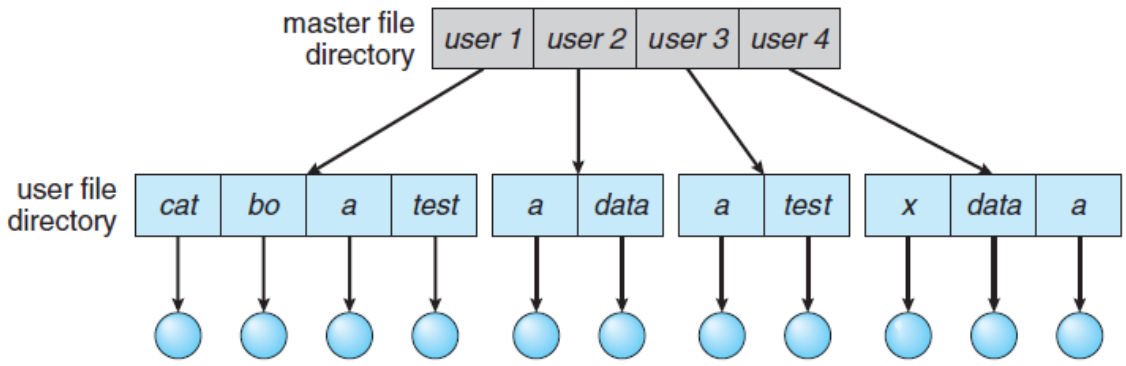
OPERACIJE NAD DIREKTORIJUMOM:

- Pretraživanje
  - Pronalaženje fajla u direktorijumu
- Kreiranje fajla
  - Potrebno je dodati novu stavku u direktorijum
- Brisanje fajla
  - Potrebno je obrisati stavku u direktorijumu
- Listanje direktorijuma
  - Spisak fajlova u direktorijumu sa dodatnim informacijama o fajlovima
- Ažuriranje direktorijuma
  - Promena atributa fajla koji se skladište u direktorijumu

STRUKTURA DIREKTORIJUMA – pravi se tako da efikasno podrži opisane operacije. Najjednostavnija varijanta – lista stavki gde svaka stavka opisuje jedan fajl. Lista se definiše u sekvencijalnom fajlu, a fajlovi nisu nikako organizovani.



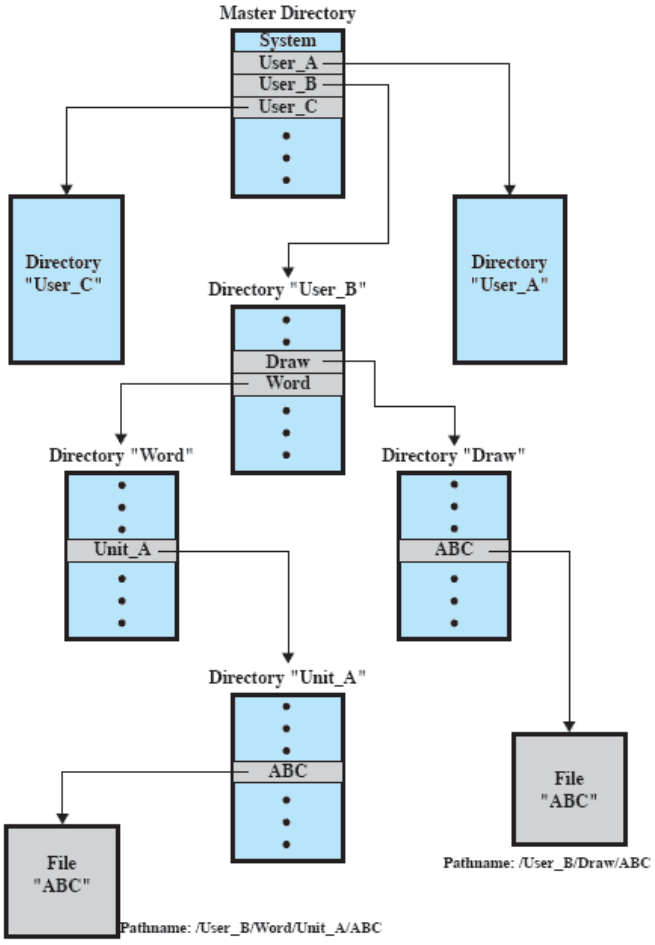
ORGANIZACIJA U 2 NIVOA – Pokušaj da se reše neki od problema sekvencijalne organizacije direktorijuma. Postoji jedan glavni direktorijum I jedan za svakog korisnika. Korisnički direktorijum je sekvencijalan spisak korisnikovih fajlova. Na nivou jednog korisnika fajlovi moraju imati jedinstveno ime.



HIJERARHIJSKA STRUKTURA – univerzalno prihvaćena varijanta, svaki direktorijum može da sadrži poddirektorijume I fajlove.

IMENOVANJE – korisnik treba da ima mogućnost referenciranja fajla po imenu. Fajlovi se referenciraju preko putanje, idući od korenskog direktorijuma do direktorijuma u kojem se nalazi traženi fajl. Ista imena fajlova su dozvoljena ako se putanje razlikuju.

RADNI DIREKTORIJUM – stalno navođenje pune putanje je naporno za korisnika. U svakom trenutku postoji radni direktorijum tj. direktorijum u kom je korisnik trenutno pozicioniran. Sve putanje do fajlova su relativne u odnosu na radni direktorijum, osim ako se eksplicitno navede puna putanja.



DELJENJE FAJLOVA – Standardni zahtev u multiprogramiranom sistemu je da iste fajlove koriste različiti procesi I korisnici. Potrebno je upravljati pravima pristpa fajlu I istovremenom pristupu fajlu.

ISTOVREMENI PRISTUP FAJLU – različiti procesi mogu istovremeno da pristupaju fajlu što je u principu dozvoljeno, ali je moguće zaključati fajl da bi se zabranila istovremena izmena.

PRAVA PRISTUPA FAJLU – pravo čitanja, pisanja I izvršavanja

ORGANIZACIJA KORISNIKA – Prava se mogu dodeliti specifičnom korisniku, korisničkoj grupi ili svima

UPRAVLJANJE SEKUNDARNIM SKLADIŠTEM

OS je odgovoran za dodeljivanje blokova diska fajlovima.

- Dva pitanja upravljanja
  - Prostor u sekundarnom skladištu treba dodeljivati fajlovima
  - Evidencija prostora raspoloživog za dodeljivanje

Kada se stvori novi fajl, da li se maksimalan prostor koji se traži dodeljuje odmah?

1. Prostor se dodeljuje fajlu u „delovima“
    a. Deo je susedni skup dodeljenih blokova
    b. Koja treba da bude veličina dela?
2. Kako voditi evidenciju o dodeljenim delovima?

DODELJIVANJE UNAPRED NASPRAM DINAMIČKOG DODELJIVANJA

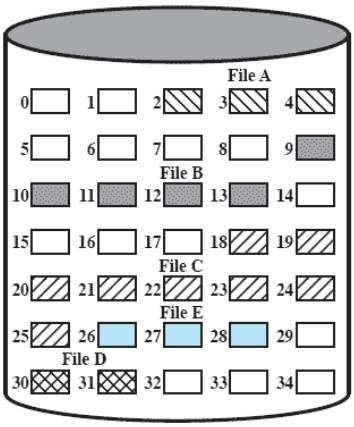
- DODELJIVANJE UNAPRED – Potrebno je u trenutku kreiranja znati maksimalni prostor koji će trebati fajlu. Teško je proceniti tu vrednost pouzdano.
- DINAMIČKO DODELJIVANJE – delovi se dodeljuju prema potrebi

VELIČINA DELA

1. **Deo je dovoljno velik da se u njega smesti ceo fajl** – podaci su smešteni u susednim lokacijama što povećava performanse pri pristupu. Teško je preraspoređivati prostor pri dodeljivanju novih fajlova. Ako su delovi promenljive veličine, nema interne fragmentacije
2. **Jedan deo ima veličinu jednog bloka** – komplikovanija evidencija dodeljivanja, jednostavnije preraspoređivanje prostora, rasipanje prostora minimalno

METODE DODELJIVANJA FAJLOVA

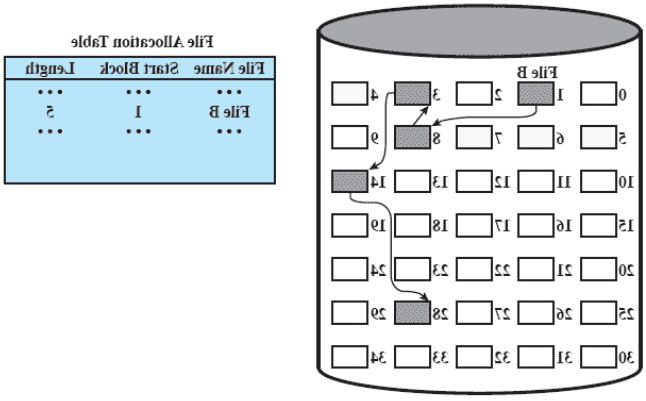
SUSEDNO DODELJIVANJE – Jedan skup susednih blokova dodeljuje se fajlu u trenutku stvaranja (dodeljivanje unapred, delovi promenljive veličine). Samo jedna stavka po fajlu u tabeli dodeljivanja – početni blok I dužina fajla. Jednostavno preuzimanje podataka I pristup bloku. Problem je eksterna fragmentacija – potrebno je sažimanje



File Allocation Table

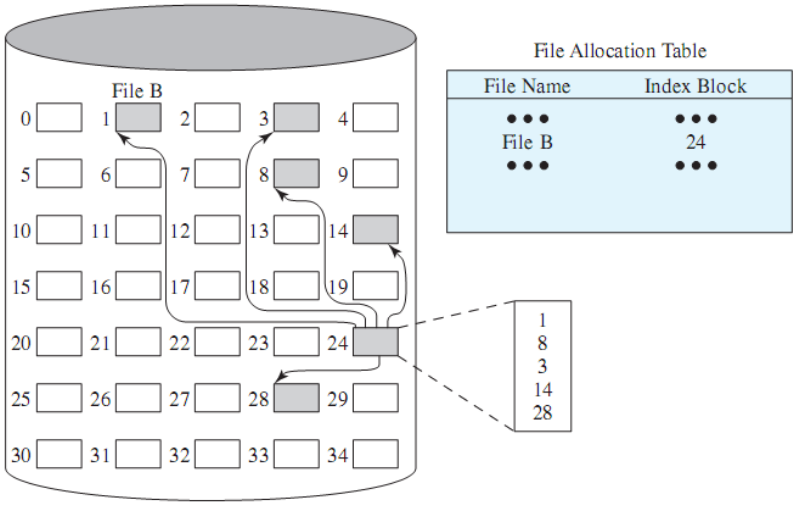
File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

ULANČANO DODELJIVANJE – dodeljivanje na bazi pojedinačnog bloa (delovi fiksne veličine, dodeljivanje po potrebi). Svaki blok sadrži pokazivač na sledeći blok u lancu. Samo jedna stavka po fajlu u tabeli dodeljivanja (početni blok I dužina fajla). Nema eksterne fragmentacije. Problem je što nije u saglasnosti sa principom lokalnosti – pristupa se različitim delovima diska. Takođe ne omogućuje direktan pristup bloku.

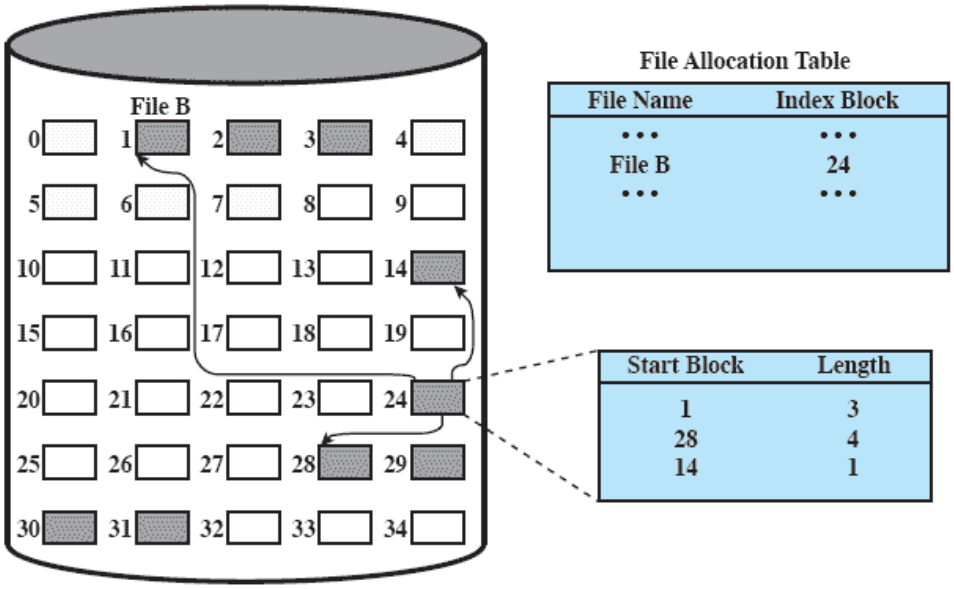


INDEKSNO DODELJIVANE – Dodeljeni delovi ne moraju biti susedni. Delovi ne pokazuju na sledeći deo. Tabela dodeljivanja za svaki fajl sadrži evidenciju dodeljenih delova. Evidencija se zove indeks delova. Indeks sadrži jednu stavku za svaki dodeljeni deo I čuva se u posebnom bloku. Stavka u tabeli dodeljivanja sadrži adresu bloka u kojem je indeks. Najpopularniji oblik dodeljivanja fajlova jer omogućava I sekvencijalan I direktan pristup.

PRVA VARIJANTA – dodeljivanje na nivou blokova fiksne veličine, nema spoljašnje fragmentacije



DRUGA VARIJANTA – dodeljivanje na nivou delova promenljive veličine, poboljšava lokalnost



UPRAVLJANJE SLOBODNIM PROSTOROM

Kao što se vrši upravljanje dodeljivanjem, mora se vršiti I upravljanje slobodnim prostorom. Sistem mora da zna koji je prostor slobodan kako bi ga dodelio fajlovima. Osim tabele dodeljivanja fajlova, potrebna je **TABELA DODELJIVANJA DISKA**. Postoji više varijanti za njenu realizaciju.

TABELE BITOVA – vektor koji sadrži jedan bit za svaki blok. Problemi su što je tabela prevelika za velike diskove I što je pronalaženje slobodnih blokova sporo

ULANĆANI SLOBODNI DELOVI – slobodni delovi se ulančavaju korišćenjem pokazivača na sledeći I dužine slobodnog dela. Evidentira se samo adresa I dužina prvog slobodnog dela. Nema potrebe za tabelom dodeljivanja diska, ali postoji problem eksterne fragmentacije.

INDEKSIRANJE – indeks slobodnih delova kao kod indeksnog dodeljivanja fajla. Evidentiraju se slobodni delovi promenljive veličine, po jedna stavka u tabeli za svaki slobodan deo. Efikasna je podrška za sve metode dodeljivanja diska.

LISTA SLOBODNIH BLOKOVA – svaki blok ima svoj broj, održava se lista brojeva slobodnih blokova. Lista je velika pa se skladišti na disku. Deo liste se skladišti u glavnoj memoriji zbog bržeg pristupa.

BEZBEDNOST FAJL SISTEMA

Svaki korisnik ima svoj identitet. OS sprovodi pravila korišćenja sistema tj. kontroliše pristup resursima sistema. Njemu je potreban skup pravila za sprovođenje kontrole pristupa.

MATRICA PRISTUPA – jedna varijanta evidencije prava pristupa. Za svakog subjekta definiše se skup operacija koje može da sprovodi nad određenim objektom

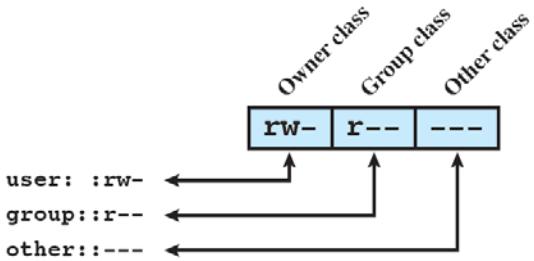
LISTE KONTROLE PRISTUPA - Dekompozicija matrice pristupa po kolonama. Za svaki fajl se navode korisnici i njihova prava pristupa.

LISTE SPOSOBNOSTI - Dekompozicija matrice pristupa po redovima. Za svakog korisnika se navode prava pristupa nad fajlovima

KONTROLA PRISTUPA FAJLOVIMA U UNIX BAZIRANIM SISTEMIMA

Prava pristupa se definišu za tri subjekta:

- Vlasnik fajla
- Korisnici koji pripadaju grupi kojoj fajl pripada
- Ostali korisnici



Moguća prava

- Čitanje
- Pisanje
- Izvršavanje

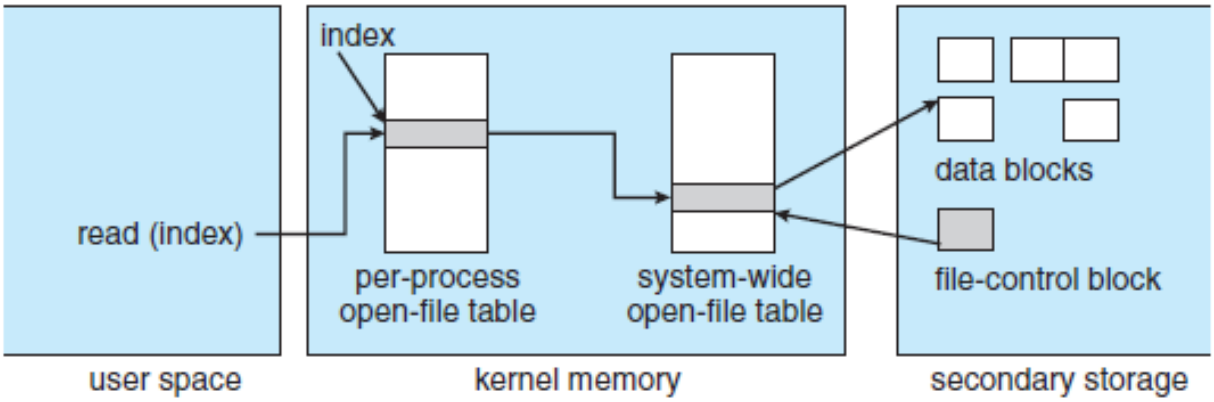
IMPLEMENTACIJA FAJL SISTEMA

STRUKTURE ZA UPRAVLJANJE FAJLOVIMA NA DISKU

1. **BOOT CONTROL BLOCK** – za svaki volumen po jedan. Sadrži informacije potrebne za startovanje OS. Najčešće su u prvom bloku.
2. **VOLUME CONTROL BLOCK** – za svaki volumen po jedan. Sadrži generalne informacije o volumenu.
3. **STRUKTURA DIREKTORIJUMA** – po jedna za svaki fajl sistem, definiše spisak I organizaciju fajlova
4. **UPRAVLJAČKI BLOK FAJLA** – po jedan za svaki fajl, sadrži informacije o fajlu

STRUKTURE ZA UPRAVLJANJE FAJLOVIMA U MEMORIJI - Informacije o svakom volumenu u fajl sistemu

1. **GLOBALNA TABELA OTVORENIH FAJLOVA** – jedna za ceo sistem, sadrži kopije upravljačkih blokova otvorenih fajlova
2. **LOKALNE TABELE OTVORENIH FAJLOVA** – po jedna za svaki proces, trenutna pozicija unutar fajla pri čitanju/pisanju kao I mod pristupa u kojem je fajl otvoren
3. **BAFERI** – sadrže blokove fajla kada se čitaju/upisuju na disk
4. – informacije o direktorijumima kojima je nedavno pristupano



Kako bi OS podržao različite fajl sisteme uvodi se APSTRAKCIJA. Umesto da OS implementira funkcije za svaki specifičan fajl sistem, on sadrži generičke funkcije za rad sa fajl sistemom. Konkretan fajl sistem specijalizuje generičku funkcionalnost za svoju strukturu I ponašanje. Nalik na polimorfizam u OOP.