Ema Djedović

Prirodno-matematički fakultet Sarajevo

Odsjek za matematičke i kompjuterske nauke

OPERATIVNI SISTEMI

SKRIPTA

2022/23

UVOD

Ovi materijali su nastali kroz predavanja prof. dr. Samira Ribića iz predmeta Operativni sistemi na Elektrotehničkom fakultetu Univerziteta u Sarajevu. Sadržaj obuhvata ključna pitanja za usvajanje gradiva, a koja su podijeljena na četiri poglavlja:

Iz predgovora knjige “Operativni sistemi”:

“Operativni sistemi su već decenijama stvar svakodnevnice. Od nekadašnjih mainframe računara, preko personalnih računara, konzola za igru, medicinskih uređaja do pametnih telefona, svaki računarski bazirani uređaj ima neku vrstu pogonskog softvera koji omogućava izvršavanje ostalih programa. Svi programi u sistemu koriste usluge operativnog sistema, pa bi se bez pretjerivanja operativni sistem mogao smatrati najvažnijim dijelom računarskog softvera. Principi rada operativnih sistema se stoga izučavaju na gotovo svakom studiju računarstva u svijetu.”

Veliko hvala prof. dr. Samiru Ribiću,
Vaša studentica.

UPRAVLJANJE PROCESIMA, RASPOREĐIVANJE I MEĐUPROCESNA KOMUNIKACIJA

Šta je to proces, razlika programa i procesa?

Program je pasivni entitet a proces je aktivni entitet koji uključuje i stanje memorije i procesora u toku izvršenja. Proces predstavlja izvršenje programa nad datim skupom podataka.

Koji su fundamentalni memorijski dijelovi procesa?

U **internoj (operativnoj, glavnoj) memoriji** su samo oni dijelovi procesa koji su u datom trenutku potrebni - samo te dijelove procesor može izvršavati. Ostatak čeka na **disku (u eksternoj memoriji)** jer je interna memorija relativno malog kapaciteta. Dešava se recimo da je jedan proces sam veći od cijele interne memorije, onda (kod multiprogramiranja) da nemamo dovoljno memorije da bi sve spremne procese učitali. Sam proces nema pojma da je podijeljen na ova dva memorijska dijela. Posljedica ove diobe je neznatno usporavanje programa.

Šta se čuva u kontrolnom bloku procesa?

1. stanje (new, ready, running, waiting - blocked, terminated)
2. PID (jedinstveni identifikator procesa)
3. pokazivač na parent process
4. pokazivač na child process (ako postoji)
5. prioritet (odnosi se na raspoređivanje)
6. procesor na kome radi (u slučaju da imamo više procesora)
7. pokazivač na memoriju
8. informacije o memoriji
9. područje registara
10. lista otvorenih datoteka
11. status zauzetih IO resursa

Koji događaji pokreću proces?

Inicijalizacija sistema, inicijalizacija batch poslova, korisnički zahtjev i sistemski poziv iz nekog drugog procesa.

Koji događaji završavaju proces?

1. normalni izlaz (dobrovoljno)
2. izlaz sa greškom (dobrovoljno)
3. izlaz sa fatalnom greškom (nije dobrovoljno)
4. ubijen - killed, od strane drugog procesa (nije dobrovoljno)

U kojim stanjima može biti proces?

new, running, waiting (blocked), ready, terminated

Koji događaji prebacuju proces s izvršenja u neke od redova čekanja?

- vrijeme za proces je isteklo (kvantum)
- u stanju waiting (blocked) jer je pokrenuo drugi proces čije rezultate čeka
- u stanju waiting (blocked) jer čeka neki drugi događaj
- hardverski prekid, npr. pritisak tastera na tastaturi, nastavlja se izvršavanje u jezgru

Šta radi dispečer?

Dispečer djeluje nakon raspoređivača i radi nekoliko stvari. Obavlja izmjenu konteksta, prepušta procesor onom procesu koga je raspoređivač odabrao kao idućeg, zadužen je za prelazak u korisnički režim rada te skok na odgovarajuću lokaciju u korisničkom programu.

Šta se dešava pri izmjeni konteksta?

Sačuva se stanje procesa ili niti koja se do tada izvršavala i učitava se sačuvano stanje od niti koja se iduća treba izvršavati. U biti imamo pomjeranje pokazivača s prethodnog na idući skup registara.

Kako je postignuto na i386 da jednom instrukcijom skoka možemo promijeniti kontekst procesa?

Svi registri našeg procesa se snimaju na lokaciju na koju pokazuje skriveni task segment registar (task registar) a u njega samog će se upisati adresa novog procesa (adresa skoka).

Kojom Windows funkcijom se kreira proces, koliko ima parametara i koji su najvažniji?

CreateProcess, 10 složenih parametara (većini se može dodijeliti prazna podrazumijevana vrijednost). Neki parametri su:

IpApplicationName - ime programa
IpCommandLine, IpProcessInformation
IpEnvironment - konfiguracijske vrijednosti sistema
IpStartupInfo - veličina i položaj prozora, boja (za tekstualne aplikacije) itd.

Kojom POSIX funkcijom se kreira proces i kako ona radi?

fork() - Kreira se jedan child proces koji dobiva (plitke) kopije koda i podataka.

Kada nastaje Zombie proces?

Ovdje roditelj proces ne čeka svoje dijete wait() pozivom već nastavlja dalje s radom. Dijete proces se ne može obrisati (izgubio bi se izlazni kod), podaci o tom procesu se još uvijek čuvaju ali on sam više nije u memoriji - dijete postaje zombie proces. Njih preuzima init proces i briše ih.

Šta su programske niti i koje su im prednosti nad procesima?

Programska nit ili LWP (lightweight proces, thread) je osnovna jedinica korištenja CPU.

Prednost niti je što je potrebno manje vremena za prebacivanje jer dijele isti adresni prostor - lakše komuniciraju. Partnerske niti međusobno dijele sekciju koda i sekciju podataka (i OS resurse) a svaka nit ima svoj stek. Također mogu biti u stanju new, running, waiting, ready, terminated.

Niti pojednostavljuju neke programe, npr. radimo više stvari odjednom preko različitih niti, sve one imaju pristup istom dokumentu. Jedan proces može u sebi imati više niti ali može biti i jednonitni.

Kako je realizovano raspoređivanje niti u korisničkom prostoru?

Ovdje jezgro ne zna ništa o postojanju niti - što se njega tiče svaki proces je jednonitni. Ovaj tip niti se može implementirati i na OS koji uopšte ne podržava niti. Mnogi OS su još uvijek takvi. Imamo doduše neke probleme, npr. niti ne mogu praviti sistemske pozive a često su one najpotrebnije u aplikacijama gdje nam trebaju blokirajući sistemski pozivi kao što je višenitni Web server.

U procesu se nalaze tabela niti i biblioteka niti, a u jezgru tabela procesa i raspoređivač procesa. S obzirom da jezgro uopšte ne prepoznaje niti, njihovo raspoređivanje obavlja biblioteka niti.

Kako je realizovano raspoređivanje niti u jezgru?

U procesu se ne nalazi ništa, sve je u jezgru: tabela procesa, raspoređivač procesa i niti, tabela niti. Omogućeni su pozivi u jezgro. Samo raspoređivanje je sporije jer se obavljaju pozivi u jezgro.

Postoji i hibridna implementacija uz preslikavanje skupova niti iz user space-a u kernel space (više u jednu, jedna u jednu ili više u više - u isti broj niti u jezgru ili manje).

Kako se u Windows-u kreiraju niti?

CreateThread() - 6 parametara (koliko stek memorije zahtijeva, funkcija koja će se izvršavati kada se nit pokrene, parametar koji se prosljeđuje funkciji, pointer na mjesto gdje će se smjestiti TID…)

Kako se POSIX API-jem kreiraju niti?

pthread_create - parametri slični kao u Windowsu? pointer na TID, funkcija, parametar za funkciju?

Objasnite kriterije algoritama za raspoređivanje: poštenost, skalabilnost, transparentnost.

poštenost - svaki proces treba dobiti približno jednako CPU vrijeme za izvršavanje (i prije svega *šansu* za izvršavanje, ne želimo izgladnjavanje procesa)
skalabilnost - prilagođenost algoritma različitim stupnjevima opterećenja, npr. želimo da raspoređivač može podnijeti povećanje broja procesa
transparentnost - za korisnika ne treba biti razlike između načina raspoređivanja na jednom ili na drugom procesoru (višeprocesorski OS), na lokalnom ili udaljenom računaru (distribuirani sistemi)

Šta su algoritmi sa istiskivanjem, a šta bez istiskivanja?

Bez istiskivanja znači da kad izaberemo proces za izvršavanje ne može ga ništa blokirati osim sam sebe niti ga išta može izbaciti sem da on dobrovoljno oslobodi procesor. Ovakvi algoritmi su jednostavniji ali se može desiti da, recimo, jedan program zaglavi u petlji pa sruši cijeli sistem.

Mogućnost istiskivanja daje za pravo raspoređivaču da skloni trenutni proces sa procesora ukoliko smatra da je neophodno (tipa istekao kvantum).

Objasnite algoritam raspoređivanja procesa FCFS.

First Come First Served, najjednostavniji algoritam, nepreemptivni (bez istiskivanja). Procesor uzima procese za izvršenje onim redoslijedom kako pristižu.

Šta je efekt konvoja i kada nastaje?

Kod FCFS algoritma, efekt konvoja je fenomen koji se dešava kada cijeli OS uspori radi jednog procesa kojem je potrebno puno vremena da se završi. Ako najduži proces prvi ugrabi CPU onda svi procesi iza njega moraju čekati (po prirodi FCFS algoritma).

Objasnite algoritam raspoređivanja procesa SJF.

Shortest Job First, nepreemptivni, izvršavaju se procesi po principu - prvi ide onaj kraći. Mana je što je potrebno unaprijed znati vrijeme izvršavanja svakog procesa. Također, nisu svi procesi odmah raspoloživi pa se ne mogu realno usporediti.

Objasnite algoritam raspoređivanja procesa SRTN.

Shortest Remaining Time Next, preemptivna verzija SJF (u igru ulazi istiskivanje). I ovdje vrijeme izvršavanje mora biti poznato unaprijed, ali u CPU se ubacuje onaj proces čije je preostalo vrijeme izvršavanja najkraće.

Objasnite algoritam raspoređivanja procesa HRRN.

Highest Response Ratio Next (najbolji odnos odziva) - radi kao SRTN s tim da se izabire proces s najvećim količnikom vremena provedenog u čekanju i *preostalog vremena izvršenja.* Znači prednost se daje procesima koji već dugo čekaju a brzo bi završili.

Objasnite algoritam raspoređivanja procesa Round Robin.

Svakom procesu se pridružuje kvantum, vremenski interval, unutar kojeg se proces smije izvršavati. Nakon što mu kvantum istekne, ili ukoliko se proces završi, isti se skida s CPU i dovodi se novi.

Objasnite algoritam raspoređivanja procesa Kooperativno raspoređivanje.

Kao Round Robin, s tim da radimo na OS koji nisu dizajnirani za istiskivanje procesa usred izvršavanja. Istiskivanje je moguće jedino preko sistemskih poziva koje prave procesi, a koji će obaviti izmjenu konteksta (kroz dio koda). Mana je ta što se programi moraju pisati strogo uzimajući u obzir ovakvu politiku raspoređivanja.

Koja je uloga kredita kod prioritetnog raspoređivanja sa vremenskim prilagođavanjem prioriteta?

Znači, imamo unaprijed određene prioritete za svaki proces i na osnovu njih se svakom procesu dodjeljuje kredit. Da se visoko prioritetni procesi ne bi izvršavali previše dugo, raspoređivač **smanjuje kredit tekućeg procesa koji se izvršava na svakom prekidu sata**.

Prva varijanta algoritma je da se proces izvršava dok mu kredit ne dosegne 0, a druga je da se proces izvršava dok mu kredit ne padne ispod kredita nekog drugog procesa. U prvoj varijanti se kredit ponaša kao kvantum, a u drugoj kao prioritet. Kada svi krediti dođu do 0 iznova se računaju.

Objasnite algoritam raspoređivanja procesa Round Robin sa prioritetima.

Procesi su grupirani u klase, svakoj klasi je dat prioritet. Za raspoređivanje klasa se koriste ti prioriteti a unutar svake klase se koristi Round Robin i kvantumi. Klasni prioriteti se moraju dodjeljivati dinamički inače procesi unutar niže klase nikad neće doći na red.

Objasnite algoritam raspoređivanja Garantovano raspoređivanje.

Modifikovani Round Robin. Ako u sistemu imamo n procesa, svaki proces bi mogao dobiti 1/n CPU vremena (kvantum?). Ovo je *relativna količina vremena* za sve procese i ona je izračunata u *trenutku obećanja* (od ovog trenutka se mjeri proteklo vrijeme). Da bi izabrali proces koji će sljedeći biti dodijeljen CPU za svaki proces se računa vrijeme n*(t provedeno u izvršavanju)/(t prošlo nakon *trenutka obećanja*). Što je ova vrijednost manja to je obećanje lošije ispunjeno. Proces s najlošije ispunjenim obećanjem prvi dobija CPU.

Objasnite algoritam raspoređivanja Ručno raspoređivanje.

Ovo možemo raditi kad imamo mali broj procesa i kad je poznato njihovo ponašanje (npr. vrijeme odziva). Pravimo statičke tabele koje kažu u koje vrijeme se pokreću koji procesi. U sistemima u realnom vremenu imamo periodične događaje i neperiodične (nepredvidljive).

Objasnite algoritam raspoređivanja Lutrijsko raspoređivanje.

Procesima se daju tiketi. Ako proces ima n1 tiketa od ukupno njih n, tada je vjerovatnoća da dobije CPU jednaka n1/n. Tiketi se biraju ("izvlače") slučajno - u igri je vjerovatnoća.

Objasnite algoritam raspoređivanja najraniji rok prvo.

CPU dobijaju procesi kojima je deadline najbliži, odnosni čiji rok završetka dolazi najranije.

Objasnite algoritam raspoređivanja monotonom stopom.

Prioritetniji su procesi čiji je period kraći (**češće se pojavljuju)**. Npr. senzor šalje podatke o poziciji svakih 10 ms a podatke o temperaturi svakih 100 ms. Prednost obrade imaju podaci o poziciji.

Objasnite algoritme raspoređivanja Particionirano i Globalno raspoređivanje.

particionirano - jedan procesor = jedan red čekanja (sa svojim algoritmom raspoređivanja), proces je, sa svim svojim nitima, trajno dodijeljen jednom CPU (od početka do kraja)

globalno - zajednički red čekanja za sve procesore, izbjegavamo situaciju da puno procesa čeka na jednom CPU a da je drugi prazan (proces za svoga života može biti izvršavan na više procesa)

Koji algoritmi se koriste u Podjeli otorećenja?

U zajedničkom redu čekanja uvezane su niti koje pripadaju različitim procesima i redom se dodjeljuju nekom od procesora kako koji postaje slobodan.

FCFS - proces sve niti ubacuje u red čekanja, one se dodjeljuju procesorima kako se koji oslobodi
Prvo najmanji broj niti - kako bi manji procesi što prije završili
Prvo najmanji broj niti s istiskivanjem - isto kao prethodno samo se procesi mogu prekidati.

Objasnite algoritam raspoređivanja družbeno raspoređivanje.

gang scheduling - U isto vrijeme raspoređujemo **sve niti jednog procesa**. Manja je izmjena konteksta, bolja saradnja između niti, veći stepen paralelizma unutar pojedinačnog procesa.

Koje uslove treba zadovoljavati dobro rješenje kritične sekcije?

1) U kritičnoj sekciji se smije izvršavati najviše jedan proces/nit.
2) Uzajamno isključivanje se treba osigurati da radi i na jednoprocesorskim i na višeprocesorskim sistemima i ne smije ovisiti o vremenu boravka u kritičnoj sekciji.
3) Proces ne smije beskonačno dugo čekati da uđe u kritičnu sekciju.
4) Proces ne smije beskonačno dugo boraviti u kritičnoj sekciji.
5) Proces van kritične sekcije ne smije blokirati druge procese.

Koji su nedostaci rješenja kritične sekcije dijeljenom varijablom?

Dijeljena varijabla u početku ima vrijednost 0 (kritična sekcija je slobodna) a kad proces uđe u kritičnu sekciju prima vrijednost 1. Problem je kada jedan proces pročita vrijednost varijable 0, uđe i ne stigne dovoljno brzo promijeniti varijablu na 1 - drugi proces je već uskočio u isto područje. Nije zadovoljen uslov uzajamnog isključenja (nezgodan trenutak).

Koji su nedostaci rješenja kritične sekcije zabranom prekida?

Ovdje se radi o zabrani prekida sata realnog vremena, ali korisnički programi u većini OS nemaju pravo da isključuju ove prekide. Također, u višeprocesorskim sistemima zabrana prekida odnosi samo na jedan procesor (nije zadovoljen uslov uzajamnog isključenja za višeprocesorske sisteme).

Šta je to instrukcija TSL i kako se njom rješava kritična sekcija?

Test and Set Lock instrukcija - imaju je mnogi računari a posebno oni **višeprocesorski**

TSL RX, LOCK - sadržaj memorijske riječi LOCK se smješta u registar RX, pa se vrijednost LOCK postavlja na različito od 0. Ove operacije su nedjeljive (atomske). Procesor dok izvršava ovu instrukciju zaključava memorijsku sabirnicu, za čega drugi procesori nemaju pristup memoriji.

Koje operacije imaju semafori sa zaposlenim čekanjem?

wait (p, down operacija) - poziva se prije ulaska u kritičnu sekciju kako bi se provjerilo je li moguć ulazak u istu. Ukoliko je semafor 0 (zauzeto) onda proces čeka (blokiran) u petlji dok se ne poveća barem na 1 - tada ulazi u kritičnu sekciju i umanjuje vrijednost semafora za jedan (down).

signal (v, up operacija) - poziva se pri izlasku - povećava vrijednost semafor varijable za 1.

Šta su to semafori sa blokiranjem i deblokiranjem?

0	kritična sekcija je nedostupna ali nijedan proces nije blokiran
+x	u kritičnu sekciju može ući još x procesa, semafor je otvoren za još x down operacija
-x	blokirano je x procesa, toliko njih je u waiting stanju za ulazak u kritičnu sekciju

Kako se kreiraju, inkrementiraju i oslobađaju semafori u Windows-u?

kreiranje - **CreateSemaphore**
inkrementiranje - **ReleaseSemaphone** (izlazak procesa)
oslobađanje - možda uništavanje? to bi bilo **CloseHandle**

Kako se kreiraju, inkrementiraju i oslobađaju semafori u Posix API?

kreiranje - **sem_open**,
inkrementiranje - **sem_post**
oslobađanje - **sem_destroy** za neimenovane semafore, **sem_close** ili **sem_unlink** za imenovane

Šta su to muteksi?

Mutexi su pojednostavljeni semafori koji dopuštaju samo vrijednosti 0 i 1 (*mutual exclusion*).

Šta su to monitori?

Monitor je skup procedura, varijabli i struktura podataka grupisanih u jedan modul (funkcionalnu jedinicu). Procesi mogu pozivati **procedure iz monitora** kad god žele ali ne mogu (direktno) pristupiti internim strukturama podataka iz **procedura van monitora**.

Šta su to barijere?

Barijere su mehanizam za sinhronizaciju (grupe) procesa u aplikacijama gdje imamo prelazak iz jedne faze u drugu. Npr. jedan proces želi preći u drugu fazu ali ostali još nisu spremni, zadržavamo ga pomoću barijere. U iduću fazu će se preći tek kad svi procesi budu spremni - barijera se podiže.

Problem proizvođača i potrošača.

Ovo je problem ograničenog bafera kojeg dijele dva procesa: proizvođač - ubacuje informacije u bafer, i potrošač - uzima (koristi) informacije iz bafera. Ukoliko je bafer pun proizvođač blokira jer nema mjesta da stavi išta, odblokirat će kad potrošač ukloni jedan element. Ukoliko je bafer prazan a potrošač želi uzeti element tada on blokira, sve dok proizvođač ne stavi element u bafer.

Problem 5 filozofa.

Ovdje imamo ograničen broj resursa i procese koji se natječu za pristup istim. Ovaj problem modeliramo filozofima (procesi) i viljuškama (resursi).

Pet filozofa sjedi za okruglim stolom, ispred svakog tanjir sa špagetama, ali između svakog tanjira samo jedna viljuška. Da bi filozof jeo potrebne su mu dvije viljuške (ograničeni resursi). Svaki filozof je u jednom od tri moguća stanja: jede (koristi obje viljuške), razmišlja, gladan je (pokušava doći do obje viljuške).

Prvo rješenje je da proces iz stanja *razmišlja* prelazi u stanje *gladan* na način da uzima prvo lijevu viljušku ukoliko je slobodna, onda desnu ukoliko je slobodna, onda *jede.* Ukoliko se desi da je lijeva viljuška slobodna a onda desna nije, proces spušta i tu lijevu jer nema koristi samo od jedne (a drugome će možda koristiti). Nastaje problem ukoliko svi procesi krenu u isto vrijeme. Možemo ovo korigirati uvođenjem semafora *limit* ograničiti broj filozofa koji mogu jesti na 4 filozofa.

Drugo rješenje je uvođenje konobara, odnosno mutex-a čija će vrijednost 1 značiti "slobodne su ti obje viljuške - možeš jesti".

Treće rješenje je da filozof ne uzima prvo lijevu pa desnu viljušku nego prvo viljušku s najmanjim rednim brojem, zatim onu s najvećim: filozof 4 prvo uzima viljušku 0, pa onda viljušku 4…

Problem čitača i pisaca.

Problem koji služi za modeliranje baza podataka. Imamo procese koji čitaju bazu i oni koji je ažuriraju (pišu). Nije problem ukoliko imamo više čitača istovremeno u bazi ali ukoliko imamo pisca tada on mora biti jedini proces prisutan u bazi.

Rješenje je da prvi čitač koji uđe u bazu odradi **down operaciju** na semaforu (baza aktivna) i poveća **brojač prisutnih** za 1. Svaki idući čitač treba samo povećati za 1 taj brojač. Pisci ne smiju ulaziti. Kada čitači izlaze smanjuju brojač, a zadnji čitač vrši i **operaciju up** (označava bazu kao praznu). Ako je do tada blokirao neki pisac on tada može pristupiti (opet radi down operaciju?).

Prednost i mana realizacije međuprocesne komunikacije dijeljenim datotekama.

Svaki OS koji podržava rad s datotekama podržava i međuprocesnu komunikaciju kroz datoteke. Na disku se kreira jedna zajednička datoteka kojoj imaju pristup različiti procesi. Ukoliko imamo mrežne diskove tada komunikaciju možemo vršiti i između udaljenih računara.

Prednost je što se komunikacija može obavljati **i kada proces primaoc nije aktivan**. Mana je **problem konkurentnosti** koji može nastati ukoliko više procesa treba upisivati u isti datotečni slog - treba koristiti mutex ili semafor.

Šta su to cijevi i koje vrste cijevi postoje?

Cijev je jednosmjeran komunikacioni kanal između dva procesa u UNIX sistemima. Jedan kraj je za pisanje (prvi proces stavlja informacije), drugi kraj je za čitanje (drugi proces čita s tog kraja).

(UNIX) **neimenovana cijev** - razmjena poruka između dva child procesa s istim roditeljem, kreira se sistemskim pozivom *pipe()* - vraća 0 za uspješno kreiranje, 1 za grešku
(UNIX) **imenovana cijev** - u svim drugim slučajevima, pogodnije su kada imamo više vidova komunikacije između procesa

Cijevi u Windowsima se koriste principijelno isto kao u UNIX-u, s tim da se koriste druge funkcije (CreatePipe umjesto pipe() i tako).

Objasnite komunikaciju procesa porukama sa i bez mailboxa.

BEZ MAILBOXA

Svaki proces ima svoj red poruka koje su njemu namijenjene. Dvije funkcije: *send(odredište, poruka)* i *receive(izvor, poruka).* Može se dodati i *tryreceive(izvor, poruka)* koja završava s kodom greške ukoliko nema poruka (nakon *receive* proces blokira sve dok ne dobije poruku).

MAILBOX

Jedan zajednički red poruka - mailbox. U pozivima *send, receive, tryreceive* nemamo odredište i izvor nego navodimo mailbox u koji šaljemo, a odredište i izvor se onda čitaju iz tijela poruke.

Pri slanju i prijemu može doći do gubitka poruke, gubitka potvrde koja se vraća od primaoca ka pošiljaocu, greške (imamo kontrolne sume i provjere pri prijemu), uvode se retransmisije onda…

Šta su Clipboard i DDE u Windows?

clipboard (*odlagalište*) - zajedničko područje u memoriji gdje se smještaju podaci dostupni svim aplikacijama (iz jedne aplikacije ubacujemo podatke preko *Copy ili CTRL+C,* iz druge aplikacije preuzimamo preko *Paste ili CTRL+V*).

DDE (*Dynamic Data Exchange)* je protokol - mehanizam međuprocesne komunikacije koristeći sistem Windows poruka koje se šalju između aplikacija

Šta su to signali u Unix sistemima?

Signal u Unixu je **vrsta prekida** i šalje se programima npr. kada se pokuša pristupiti nedozvoljenom segmentu. Uobičajene akcije koje se vrše nakon prijema signala se mogu predefinirati, ali tipično se program prekine, snima se slika memorije u trenutku završavanja i poziva se **obrađivač signala**. Kada obrađivač završi, OS preuzima kontrolu i vraća stanje prekinutog procesa. Imamo *alarm* kao posebnu vrstu signala.

RESURSI I ZASTOJI, SIGURNOST, UPRAVLJANJE MEMORIJOM

Koji su koraci u upotrebi resursa u operativnim sistemima?

1. **zahtjev za resursom** - postiže se odgovarajućim sistemskim pozivom. Ukoliko je resurs u tom trenutku zauzet, proces može čekati zaposleno ili nezaposleno (uspavano).
2. **korištenje resursa** - samo nakon što je zahtjev odobren/uspješan
3. **oslobađanje resursa** - mora se pozvati ako nam resurs više ne treba (možda drugima treba)

Šta je to potpuni zastoj?

Potpuni zastoj je stanje jednog skupa procesa gdje svaki proces u tom skupu čeka na neki događaj koji može proizvesti jedino neki drugi proces iz tog istog skupa.

Uslovi nastajanja potpunog zastoja.

(Coffman) Svi uslovi moraju biti zadovoljeni:

1. uslov **uzajamnog isključivanja**… resurs ili dostupan ili dodijeljen tačno jednom procesu (npr. resurs ne može biti nedodijeljen i nedostupan)
2. uslov **neoduzivosti resursa**… resurs može osloboditi jedino proces koji ga koristi trenutno
3. uslov da **proces koji zauzima neki resurs mora tražiti novi resurs**
4. uslov **kružnog čekanja**… lanac, ciklus

Kako se modelira potpuni zastoj grafom dodjele resursa?

Krugovi su procesi, kvadrati su resursi. Strelica kvadrat->krug znači da je resurs dodijeljen procesu. Strelica krug->kvadrat znači da proces zahtijeva resurs (čeka na njega). Zatvorena kontura (ciklus) nagovještava mogućnost potpunog zastoja, ali moraju biti ispunjena i ostala 3 Coffmanova uslova. Za više primjeraka jednog resursa u njegovom kvadratu crtamo po jednu tačku za svaki primjerak.

Nojev algoritam.

“Zavući glavu u pijesak i praviti se da problem ne postoji.” Matematičari se ne slažu, inženjeri prakticiraju. Možda potpuni zastoj nije uvijek toliki problem? U sistemima realnog vremena jeste.

Algoritam za detekciju potpunog zastoja.

Imamo nekoliko dobrih algoritama za detekciju (ali ne i oporavak):

1) **detekcija mjerenjem vremena**

Uočavamo kada su neki procesi predugo u stanju waiting. Npr. Windows API funkcija *WaitForSingleObject()* kao parametar ima maksimalan broj milisekundi koliko proces može čekati blokiran, a kad to istekne funkcija vraća WAIT_TIMEOUT. Problem je što možemo ovakve poruke dobivati i kad se jednostavno neki proces malo oduži, a i tad ne znamo koji je to tačno proces - zastoj može biti bilo gdje.

2) **detekcija grafom alokacije (dodjele) resura**

Pojava zatvorenih kontura je dobar indikator da imamo potpuni zastoj. Ovo je lagano na papiru ali treba to implementirati. Algoritam kreće od jednog čvora i posmatra ga kao korijen podstabla s čvorovima koji izlaze iz njeg - ide dubinska pretraga. Označavamo sve čvorove koje smo posjetili. Ukoliko naiđemo na čvor koji smo već posjetili to znači da imamo konturu (potencijalan zastoj).

3) **detekcija matričnom metodom**

Matricu pravimo kada ima više kopija jednog resursa ili kad jedan resurs može koristiti istovremeno više procesa.

matrica alokacije A - resursi trenutno u upotrebi
matrica novih zahtjeva N - još potrebno dodijeliti da bi procesi završiti s radom
vektor resursa R - broj preostalih (raspolagajućih) primjeraka svakog resursa

i-ti red == proces broj i, i-ti proces po redu
N3 ≤ R - svaki element reda 3 u matrici N je manji od elementa vektora R na istoj poziciji
označen proces == nije uzrok potpunog zastoja

Sistem nije u potpunom zastoju ako su svi procesi označeni. Algoritam je sljedeći:

Ako je red A1 jednak nula-vektoru onda proces P1 označavamo. Sada gledamo proces P2 gdje A2 nije nula-vektor. Ukoliko je recimo N2 ≤ R, tada se zahtjevi procesa 2 mogu ispuniti (proces je moguće uspješno završiti). Pretpostavimo da se završio i ide osloboditi resurse koje je zauzimao - na vektor R se nadodaju vrijednosti reda A2. Proces se označi (gotov je).

Načini oporavka od potpunog zastoja.

oduzimamo jedan resurs - Rijetko izvodivo, pouzdano samo kada nam je poznat cijeli tok izvođenja programa (npr. batch sistemi).

prekidamo jedan proces - Trebamo dovesti administratora da prekine proces. Njegovi resursi se oslobađaju i dodjeljuju drugim procesima koji su čekali na njih. Koji proces prekinuti? Tu odluku

može donijeti i sam OS na osnovu prioriteta, proteklog vremena, tipu zauzetog resursa itd. Nekada je neophodno prekinuti i više od jednog procesa.

restart sistema ili podsistema - preporučuje se u složenijim slučajevima (npr. Java virtualna mašina ako višenitne aplikacije uđu u potpun zastoj)

Kada prekidamo proces ili restartujemo sistem dešava se gubitak odrađenog posla. Da bismo to ublažili možemo snimati kontrolne tačke (stanje cijelog sistema). Snimanje cijele memorije je sporo pa se kontrolne tačke uglavnom ne koriste u OS, ali se koriste u bazama podataka.

Pojam sigurnog stanja.

Sigurno stanje je redoslijed dodjele (alokacije) resursa koji neće dovesti do potpunog zastoja.

Bankarov algoritam za izbjegavanje potpunog zastoja za više resursa.

Nije isto što i sprječavanje koje je sigurnije. Slično je matričnom metodu za detekciju zastoja (matrica A, matrica N, vektor R). Konstruišemo algoritam za dodjelu resursa koji neće dovesti do potpunog zastoja. Ista pravila važe kao i ranije, to da ne ponavljamo.

Na početku su svi procesi neoznačeni, a sistem je u sigurnom stanju ako su na kraju svi procesi označeni (možemo zaključiti da postoji bar jedan redoslijed izvršavanja procesa i dodjele resursa koji ne vodi potpunom zastoju).

Algoritam je u praksi beskoristan jer broj procesa mora bit konstantan, a to rijetko imamo. Također, kako mi da znamo koliko će procesu trebat resursa i kojih?

- nađi neoznačen proces Pi za kojeg je red $Ni \leq R$ (sve mu zahtjeve možemo ispuniti)
- ako takav Pi postoji onda saberi red Ai sa R i rezultat smjesti u R
- označi Pi i kreni ponovo s idućim neoznačenim Pj
- kraj kada nema više procesa koji zadovoljavaju prvi uslov (neki možda ostaju neoznačeni pa opet imamo potpuni zastoj - izbjegavali smo ga ali ga **nismo uspjeli izbjeći**)

Kako se mogu spriječiti potpuni zastoji izbjegavanjem uzajamnog isključivanja?

Zastoj možemo spriječiti ukoliko onemogućimo jedan od 4 Coffmanova uslova.

Onemogućimo uslov uzajamnog isključivanja **specijaliziranim procesima** koji zauzmu jedan resurs i samo oni koriste taj resurs i ne traže uopšte nijedne druge resurse. Takvi procesi neće biti uzrok potpunog zastoja. (Samo njih da koristimo?)

Kako se mogu sprječiti potpuni zastoji izbjegavanjem prisvajanja i čekanja?

Moramo spriječiti da proces čeka neke resurse dok jedne već drže zauzetim.

Jedno rješenje je da navedemo listu resursa koji program koristi i da **na početku odmah pokuša zauzeti sve njih**. Ako bar jedan resurs nije slobodan onda se proces neće pokrenuti (Nešto slično kao s filozofima i viljuškama?). Opet moramo unaprijed znati koji će se resursi koristiti. Čak i ako znamo, druga mana je što će vjerovatno **mnogo resursa biti u stanju mirovanja**.

Kako se mogu sprječiti potpuni zastoji izbjegavanjem kružnog čekanja?

Možemo uvesti da **svaki resurs ima svoj redni broj i resursi se dodjeljuju tim redom**. Moramo biti sigurni u konstantan broj mali resursa (Jer pravimo statičke tabele?), odnosno da se broj resursa neće mijenjati (onda bismo morali obnavljati redoslijed).

Sprječavanje zastoja tako što ćemo neoduzive resurse proglasiti oduzivim (preostali Coffmanov uslov) - nije moguće. Oni nisu bez razloga neoduzivi (CD, pisač, štampač). Nijedan od ovih rješenja nije savršen ali u praksi je problem potpunog zastoja minimiziran.

Šta su to primarna, sekundarna i tercijarna memorija?

Primarnoj memoriji se pristupa direktno preko adresiranja memorijskih lokacija, bez IO kontrolera. U ovaj tip memorije spadaju **interna memorija (glavna, operativna), keš memorija i procesorski registri**. Glavna memorija može biti RAM i ROM. O organizaciji RAM-a brine OS, dok za ROM nekad jest a nekad nije zadužen OS. Procesor izvršava samo procese koji su u internoj memoriji, stoga se njom pretežno i bavimo.

Sekundarna memorija zahtijeva upotrebu IO kontrolera, i to može i bez ljudske ili mašinske intervencije, ali je pristup znanto sporiji nego pristup primarnoj memoriji.

Tercijarna memorija se koristi za čuvanje rijetko korištenih podataka (npr. starih video zapisa) i nije stalno spojena na računare. Optički diskovi, optičke trake, magnetne trake, floppy diskovi…

Koje su četiri tipične konfiguracije memorije u jednoprocesnim sistemima?

1. **samo korisnički program (proces) cijeli u memoriji**

mainframe računari 50-ih, prvi kućni računari u 70-im, prve konzole za igru… Program dobiva punu kontrolu nad računarom. Oslanja se isključivo na sebe, ne trebaju mu OS ni drajveri.

Kasnije se uvodi OS s kojim korisnički program dijeli memoriju. OS sadrži i sve neophodne drajvere, a učitava se kada se računar uključi.

2. **korisnički program u RAM-u, OS u RAM-u**

Model je pun grešaka bio upotrebljivan tokom rane faze razvoja OS. Problem nastaje ako nestane električne energije: nema ROM memorije - dobar dio OS se mora unositi bit po bit u mašinskom (ne čak ni asemblerskom) jeziku.

3. **korisnički program u RAM-u, OS u ROM-u**

Kućni računari u 80-im (napredak u odnosu na 70-te), današnji džepni računari. OS je u nižim adresama memorije u ROM-u, a korisnički program na višim adresama. OS je otporan na nestanak električne energije. Nema fleksibilnosti: greške u OS se ne mogu popraviti niti se sistem može nadograditi bez servisa matične ploče.

4. **BIOS, korisnički program u RAM-u, OS u RAM-u**

BIOS je na najvišim adresama (tip ROM memorije, Basic Input Output System) i sadrži boot program koji učitava OS te osnovni skup funkcija za rad sa uređajima. Sada imamo fleksiblnost izmjene operativnog sistema, ali ga malo sporije startamo.

Šta je to swapping?

Swapping je tehnika koja snima na disk sadržaj memorije **cijelog procesa** i učitava ga nazad kad je to potrebno. U budućnosti nastavljamo izvršavati proces gdje smo stali, a ne ispočetka.

Koji su pristupi za određivanje listi čekanja kod multiprogramiranja s fiksnim particijama?

Liste/redovi čekanja se mogu organizirati na način da **svaka particija ima svoju listu čekanja** ili da postoji **jedinstvena lista čekanja** za sve particije. Kako su u pitanju fiksne particije, njihov broj, veličinu i lokacije na kojima svaka počinje unaprijed znamo.

Jedina prednost fiksnih particija je jednostavna implementacija. U memoriji se istovremeno može nalaziti okoliko procesa koliko ima particija; procesi koji su veći od najveće particije se ne mogu nikako pokretati; često se dešava da je kraći proces smješten u veću particiju i onda ostane neiskorištene memorije u toj particiji…

Kako je realizovana statička relokacija?

Imamo logičke (virtualne) i fizičke adrese jednog procesa. Skup svih logičkih adresa unutar jednog procesa naziva se **logički (virtualni) adresni prostor.** Skup svih fizičkih adresa naziva se samo **adresni prostor.** Pod relokacijom se općenito podrazumijeva mapiranje logičkih u fizičke adrese.

Neka je program dizajniran da se izvršava u particiji koja počinje od adrese 1000, ali kad je došlo vrijeme da se iz reda čekanja prebaci u izvršavanje ispostavi se da je ta particija zauzeta, a slobodna particija (u koju se može ubaciti) počinje od adrese 3000. **Treba izmijeniti neka mjesta u programu prije nego se ubaci u particiju 3000 za koju nije bio spreman**.

Tabelu tih mjesta koji se moraju promijeniti ima program punilac - **loader.** Izvršava se sada **statička relokacija** (samo jednom, isključivo pri pokretanju programa) - sva ta "za particiju 1000" mjesta sabiraju se sa razlikom *3000-1000*. Npr. na jednom mjestu u programu se pristupa lokaciji 1230 - sad se pristupa lokaciji 3230. Moramo imati dovoljno susjednih fizičkih adresa.

Koja je uloga relokacionog i limit registra kod dinamičke relokacije?

Mapiranje se obavlja dinamički pomoću **MMU** (*Memory Management Unit*, jedinica za upravljanje memorijom) i to tijekom cijelog izvršavanja programa.

Relokacioni registar određuje početnu fizičku adresu procesa. Uz njega se može dodati i **limit registar** koji predstavlja najveću dopuštenu fizičku adresu kojoj se može pristupiti.

Relokacija s relokacionim registrom.

Ovo je najjednostavniji način da mapiramo logičku adresu u fizičku. Svaka logička adresa koju zahtjeva proces sabira se sa vrijednošću relokacionog registra (početna fizička adresa).

Dodamo li i limit registar onda ograničavamo memoriju kojoj proces može pristupiti. Provjeravamo fizičku adresu koju smo dobili računom i fizičku adresu limit registra koja predstavlja gornju granicu - nećemo "ulijetati" u područje memorije koje možda koristi neki drugi proces.

U čemu je razlika alokacije memorije s promjenjivim particijama u odnosu na fiksne?

Kod promjenljivih particija ne zna se unaprijed ni njihov broj, ni veličina, ni lokacija. Ti parametri se određuju dinamički - particije se kreiraju i oslobađaju po potrebi. Memorija je bolje iskorištena ali je teža implementacija. Pored evidencije o zauzetosti particija trebamo čuvati i informacije gdje particije počinju i završavaju.

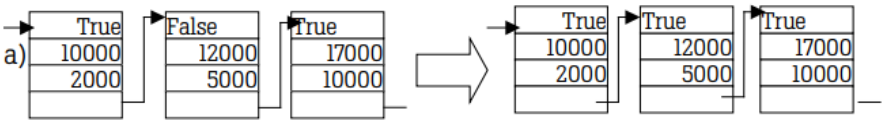
Koje slučajeve razlikujemo kod zauzimanja i oslobađanja memorije s promjenjivim particijama ako se korite ulančane liste?

Jedan element oveulančane liste je jedna particija s atributima:

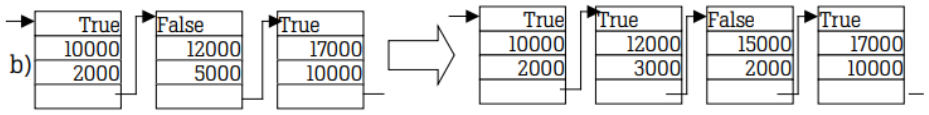
- zauzeta (1 True ili 0 False)
- početna adresa
- veličina
- sljedeća particija (pokazivač)

Imamo 6 slučajeva: dva slučaja zauzimanja particija i četiri slučaja oslobađanja. Kod zauzimanja je bitno jesu li particije **iste veličine ili ne**, a kod oslobađanja je bitna zauzetost susjednih particija:

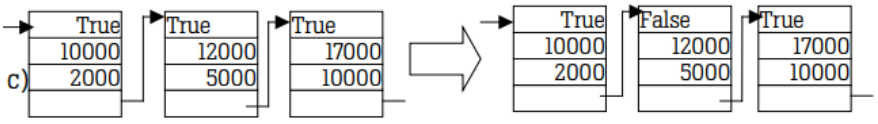
1. veličina particije koja nam treba je **jednaka particiji koja je slobodna** - dovoljno promijeniti polje te slobodne particije "zauzeta" sa False na True



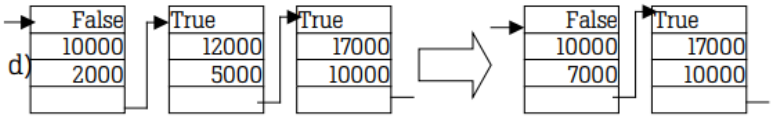
2. veličina particije koja nam treba je **manja od particije koja je slobodna -** polje "zauzeta" se postavlja na True, dodijeljena particija se skrati a preostali slobodni dio se pretvori u novi element (nezauzeta particija veličine onog viška što nam je ostao)



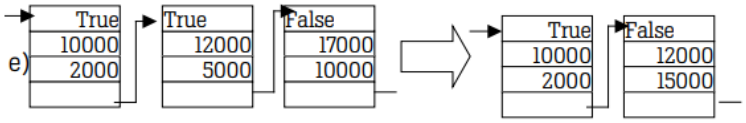
3. ne treba nam više particija koja se nalazi **između dvije zauzete** - samo promijenimo status "zauzeta" na False (*zauzeta, slobodna, zauzeta*)



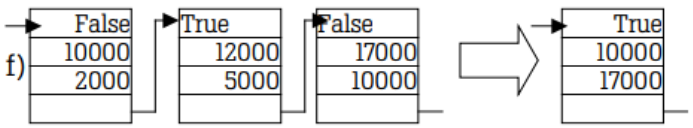
4. ne treba nam više particija koja se nalazi **nakon slobodne a prije zauzete** - ta naša se spaja s ovom slobodnom u jednu slobodnu



5. ne treba nam više particija koja se nalazi **nakon zauzete a prije slobodne -** opet spajamo našu particiju ali ovaj put sa sljedećom



6. ne treba nam više particija koja se nalazi **između dvije slobodne** - sad u biti imamo 3 slobodne particije zaredom, možemo ih sve spojiti u jednu veličine zbira sve tri



Šta su kontrolni blokovi memorije?

Podaci o particijama se čuvaju na istom mjestu gdje i one same: ispred svake particije se nalazi **zaglavlje + Memory Control Block.**

Kontrolni blok je fiksne dužine, sadrži stanje zauzetosti particije i njenu dužinu (slično kao i oni čvorovi od ranije). Umjesto početne adrese particije, čuva se početna adresa kontrolnog bloka i na to se nadodaje njegova veličina da se dobije početna adresa particije. Sljedeći blok se nalazi odmah iza prethodnog (ne treba nam onaj pokazivač više).

I ovdje imamo 6 slučajeva ažuriranja, kao kod ulančane liste, s tim da je oslobađanje particije malo jednostavnije - lakše brišemo elemente. Kontrolne blokove ne možemo koristiti ako imamo isprekidan adresni prostor, npr. 32K RAM-a pa 4K za IO uređaje pa opet 8K RAM-a itd.

Šta su to bitmape za evidentiranje zauzetih blokova?

Bitmapa je niz bitova gdje jedan bit predstavlja jedan komad memorije (0-slobodan, 1-zauzet ili obrnuto, kako se dogovorimo). Skup svih komada zajedno (iste su veličine) čini našu memoriju (bitmapa onda predstavlja cjelokupno stanje zauzetosti memorije).

Jedna particija može zauzimati **više tih komada memorije** (veličine particija nisu iste) i slobodna particija je predstavljena kontinualnim nizom 0 (onoliko koliko ista ima komada memorije). Kada se particija alocira onda se taj niz 0 pretvara u niz 1. Analogno se pri oslobađanju 0 vraćaju u 1.

Mana ovog pristupa je sporo pretraživanje, najsporije zapravo od sva tri pristupa (prvi su ulančane liste, drugi su kontrolni blokovi), ali je ažuriranje najjednostavije (zauzimanje i oslobađanje).

Objasnite algoritme first fit, best fit, next fit, worst fit.

Ovo su algoritmi za izbor slobodne (promjenljive) particije koja će se dodijeliti procesu.

first fit (prvo uklapanje) - idemo od početka i tražimo prvu dovoljno veliku particiju, nastaje puno šupljina (formiraju se nezauzete particije kada manji proces dobije veću particiju), brz algoritam

best fit (najbolje uklapanje) - prolazimo kroz cijelu listu da bi odabrali particiju koja je najmanja a dovoljno velika da zadovolji naše potrebe, dobra iskorištenost memorije, spor algoritam

next fit (sljedeće uklapanje) - kao *first fit* ali ne kreće svaki put od početka nego odakle je stao

worst fit (najgore uklapanje) - prolazi kroz cijelu listu i odabire najveću moguću šupljinu, od nastalih šupljina možda formiramo nezauzetu particiju koja se još jednom može iskoristiti, ipak najlošiji algoritam

Šta je to buddy sistem?

Ovo je neka kombinacija fiksnih i promjenljivih particija - alokacija particija stepena broja 2.

Npr. zahtijeva se količina memorije od 300KB - možemo dodijeliti particiju od 512KB. Problem je što je možda takvih konkretnih particija nestalo, možda smo ih već razdijelili. Uzmemo veću particiju - neku koju imamo, npr. 4MB. Nju prepolovimo, pa opet, pa opet, dobijemo 2MB, 1MB, 512KB, 512KB - jednu od 512KB dodijelimo a druga ostane slobodna. Ako se onda zatraži 1MB za neki novi proces, odmah imamo da mu dodijelimo tačno 1MB. Ako dođe proces sa zahtjevom od 200KB onda polovimo onih 512KB kojih nam je ostalo i dodijelimo tu jednu 256KB particiju.

Kako se alocira memorija u Windows API a kako u Posix API?

Windows API ima više funkcija (sistemskih poziva) za alokaciju memorije:

VirtualAlloc - alocira dio virtualne memorije, *VirtualFree*
HeapAlloc - alocira dio memorije unutar procesa, *HeapFree*

Pomoću ovih funkcija se realiziraju (implementiraju) poznate C funkcije *malloc* i *free*. Možemo koristiti *HeapReAlloc* da promijenimo veličinu alocirane memorije za proces i *HeapSize* da dobijemo veličinu alociranog memorijskog bloka.

Posix API ne pruža mnogo sistemskih poziva za upravljanjem memorijom generalno. Aplikacije opet trebaju koristiti C funkcije *malloc, free, calloc* i slično. Imamo dvije funkcije koje povećavaju ili smanjuju memorijski opseg na raspolaganju:

brk - "pomiče" kraj područja memorije za proces na datu lokaciju
sbrk - uvećava područje memorije za dati broj bajta

Šta su dinamički dijeljene biblioteke?

Ovo su .dll fajlovi (User32.dll) - *Dynamic-link Library*

Nema logike u svaki program ugrađivati recimo funkciju *printf* (za 10 programa to bi oduzelo po 6K za svaki, znači 60K memorije). Dovoljno je imati jednu kopiju ove funkcije u memoriji i da je ona dostupna svim procesima.

Dinamički dijeljene biblioteke su memorijske cjeline koje grupišu više potprograma koji se mogu pozivati iz različitih procesa (samo treba uključiti - vezati te biblioteke?). Štedimo prostor na disku.

U čemu je razlika između implicitno i eksplicitno vezanih dinamički dijeljenih biblioteka?

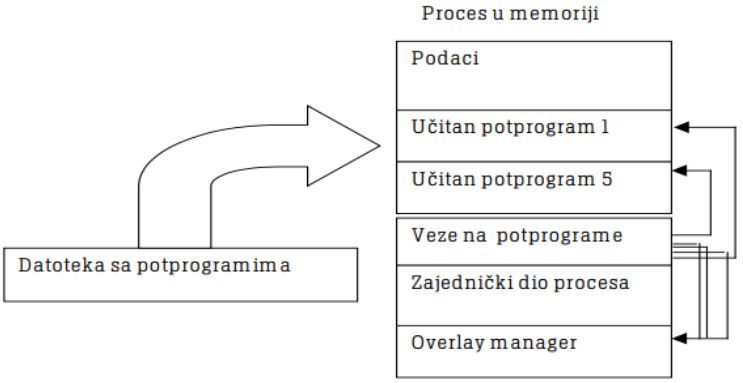
Program se povezuje s dinamički dijeljenim bibliotekama implicitno ili eksplicitno.

1. implicitno: Unutar programa imamo spisak dinamičkih biblioteka i njihovih potprograma koji se mogu koristiti. Pri startu programa ***loader* (punilac) poveže** sve te dinamičke biblioteke. Programer je dužan samo koristiti odgovarajuće datoteke sa zaglavljima i pravilno konfiguriran kompajler.

2. eksplicitno: Ovo je način povezivanja s dinamički dijeljenim bibliotekama gdje je nužno pozivati određene funkcije: *LoadLibrary, FreeLibrary, GetModuleFileName, GetProcAddress.* Koristimo ovo ako unaprijed ne znamo imamo li uopšte biblioteku u sistemu koja nama treba. Zgodno je i za nadograđivanje programa u budućnosti bez da znamo ime biblioteke ili funkcije.

Kako je organizovan program ako se koristi overlay?

U situaciji smo da su nam particije limitirane na, recimo, 32 KB a naš program traži sveukupno 80KB. Rješenje je **overlay organizacija:**

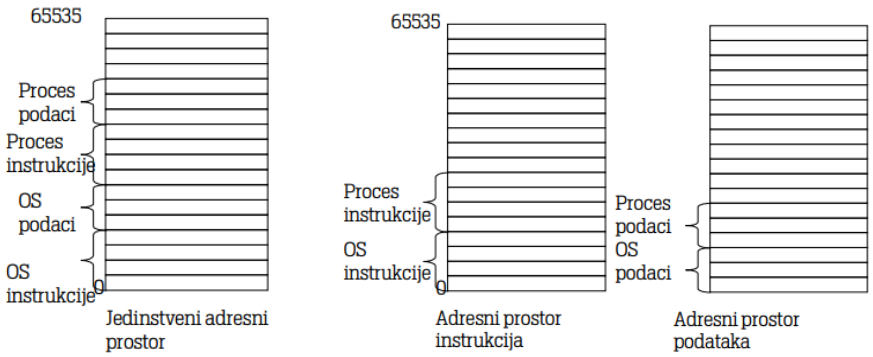


Podijeliti naš program u potprograme koji će se svi snimiti u jednu **datoteku s potprogramima.** Iz te datoteke se učitavaju potprogrami u memoriju. U proces u memoriji se ugradi i ***overlay manager***.

Overlay manager iz datoteke učitava potprogram u memoriju onda kada je potreban. Ako se nema dovoljno prostora u memoriji onda **overlay manager** mora izbaciti neki ranije učitani potprogram.

Šta je Harvard arhitektura?

Znamo za klasičnu von Neumannovu arhitekturu - jedinstveni adresni prostor. Neki sistemi razlikuju **prostor instrukcija i prostor podataka.**



Ova razdvojena je Harvard arhitektura. Instrukcije i podaci su znači odvojeni. Instrukcije možemo poistovijetiti s terminom "program". Kod kalkulatora je ovo ovako.

Kada se koriste memorijske banke?

Banka je fiksna memorijska particija koja se ubacuje u adresni prostor. Npr. dobili smo slabiji procesor u cilju kompatibilnosti s ranijim modelima, on zauzima samo 64K i nema MMU, a naš napredniji model ima 128K memorije. Da iskoristimo ovu memoriju koristimo memorijske banke.

Šta radi memory management unit?

MMU pretvara virtualnu adresu u fizičku (dinamička relokacija?). Virtualna memorija dopušta se dio procesa čuva u primarnoj (specifično, RAM) memoriji a drugi dio u sekundarnoj memoriji (disk), a sam proces ne zna ništa o tome.

Ove pretvorba može biti linearna (jednostavna, brza) ali zahtjeva da kontinualno područje virtualne memorije odgovara kontinualnom području fizičke memorije, tabelarna (fleksibilnost) ali zauzima previše prostora i sporije je nego da radimo linearnu pretvorbu, a kompromis se postiže kombinacijom: relativno male tabele pretvorbe + iskoristiti kontinualna područja memorije.

Kako se računa fizička adresa sa segmentacijom s više relokacionih registara?

Linearna pretvorba? Za svaki segment imamo po jedan relokacioni registar (sadrže početne fizičke adrese). Svaki segmentni (relokacioni) registar se prvo množi sa 16 (jer su logičke adrese predstavljene sa 16 bita) pa se tek nadodaje logička adresa koju mapiramo - dobijamo fizičku.

Kako se računa fizička adresa sa tabelom segmenata?

Elementi ove tabele su deskriptori**.** Svaki je predstavnik jednog segmenta, sadrži sve njegove podatke. Za dati segment **pogleda se njegov deskriptor, izvuče se početna fizička adresa i ona se sabere s logičkom adresom unutar segmenta**. Pri svakom pristupu memoriji moralo bi se pročitati gdje segment počinje, a to je sporo. Nemamo kao ranije kontinualnu memoriju da ne moramo brinuti o tim pozicijama.

Šta je deskriptorski keš?

Deskriptorski keš je memorija koja se ugrađuje u procesor, a **ubrzava pristup memoriji kod tabele segmenata** - pamti početne fizičke adrese zadnje pristupanih segmenata.

Šta je to tabela stranica?

Virtualan adresni prostor je podijeljen na (jednake) stranice. Stranice se preslikavaju u **okvire** u fizičkom adresnom prostoru Taj proces se naziva straničenje - paging.

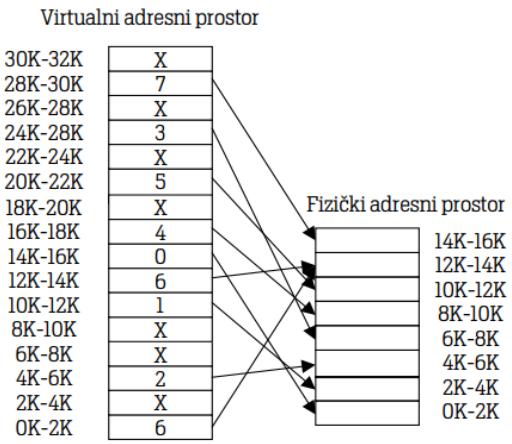


Tabela stranica služi da preslikava virtualnu stranicu u fizički okvir. Broj stranice je indeks u tabeli stranica. Za svaku stranicu imamo uređen par **(broj stranice, pozicija unutar nje)**.

Čemu služe bit prisutnosti, modifikacije, referenciranja i zaštite?

bit prisutnosti (*present bit*) - je li virtualna stranica u fizičkoj memoriji (1 za jeste, 0 za nije)
bit modifikacije M - 1 ukoliko se pisalo unutar stranice, 0 inače
bit referenciranja (pristupa) R - 1 ukoliko se pristupalo podacima iz stranice, 0 inače
biti zaštite - npr. podacima stranice se može samo čitati, podacima smije pristupiti samo jezgro itd.

Čemu služi TLB?

TLB (*Translation lookaside buffer*) je memorija u kojoj se čuvaju **kopije najčešće korištenih elemenata tabele stranica i njihovi okviri.** To se radi da bi se izbjeglo višestruko pristupanje memoriji za istu instrukciju. Ukoliko se virtualna stranica ne nalazi u TLB onda se normalno pristupa tabeli stranica. Noviji procesori imaju primarne i sekundarne TLB registre.

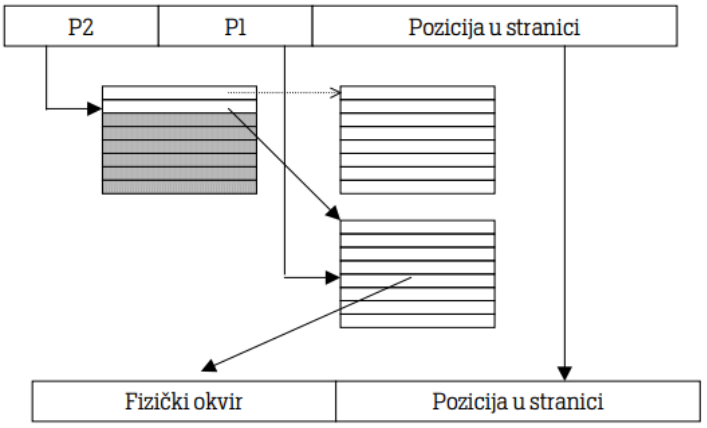
Kada se koristi hijerarhijsko straničenje?

Do sad smo imali straničenje na jednom nivou (broj stranice, pozicija), a sada možemo imati hijerarhiju: dva, tri, četiri nivoa... To koristimo kada želimo na praktičan način **smanjiti veličinu tabele stranica.** Sa straničenjem na jednom nivou, tablica je imala odgovor na pristup bilo kojoj memorijskog adresi, što je puno, a većina stranica nije u memoriji. Umjesto jedne tablice sa svim mogućim stranicama, sada imamo više tabela (uvodimo njihove redne brojeve), **neke sadrže sve stranice prazne, stoga im ne pristupamo.**

Straničenje na **dva nivoa** podrazumijeva logičku adresu u obliku **(redni broj tabele, broj stranice u tabeli, pozicija unutar stranice)**.

Na slici je P2 broj tablice, P1 broj stranice u tablici. Sve tablice od treće pa nadalje su prazne (sivo). Nakon što nas P2 odvede do stranice, P1 nas pozicionira na pravo mjesto unutar nje.

Samo tablica najvišeg nivoa mora stalno biti u memoriji (u našem slučaju P2). Ove nižeg nivoa se mogu ubacivati u memoriju po potrebi.



U čemu je razlika između metode jednake raspodjele i proporcionalne raspodjele?

Dodjela fizičkih okvira procesu: koje okvire dodijeliti i koliko njih? Dvije metode:

1. **jednaka raspodjela**

Imamo recimo 1000 okvira i 5 procesa, svakom procesu se dodjeljuje 1000/5=200 okvira. Ako je sistemska memorija 128M, a izvršavaju se 4 procesa, svakom će biti dodijeljeno 128/4=32M. Mana je što ne traže svi procesi istu količinu memorije - jedan proces će imati dosta neiskorištene memorije a drugi će generirati mnogo grešaka stranica.

2. **proporcionalna raspodjela -** oni procesi koji su prijavili veći adresni prostor će i dobiti više fizičkih okvira

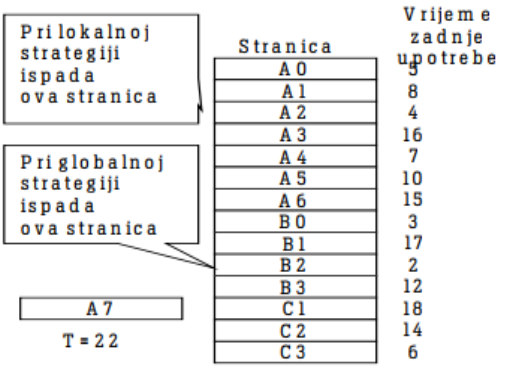
Imamo i druge metode, tipa prioritetne metode (MS Windows daje više fizičkih okvira onom procesu s kojim korisnik trenutno radi, *front-end* proces). Također, broj dodijeljenih okvira ne smije pasti ispod minimalne vrijednosti, koja dovodi do čestih grešaka stranica.

U čemu je razlika između lokalne i globalne strategija straničenja?

Pitanje je: "Treba li algoritam zamjene stranice uzeti u obzir samo njegove okvire ili pretražiti čitavu memoriju?" Ovisi koja je strategija straničenja korištena.

Broj dodijeljenih okvira može ostati isti tokom izvršavanja procesa ili se može mijenjati. U slučaju da ostaje isti, govorimo o **lokalnoj strateriji straničenja.**

Ako se broj okvira može mijenjati, govorimo o **globalnoj strategiji straničenja.** Ovdje ako proces generiše grešku stranice ista se ne traži samo iz skupa okvira njemu dodijeljenih nego iz skupa svih okvira - može se izbaciti neka stranica iz drugog procesa u korist ovog prvog. Bolja je iskorištenost memorije ali je pretraživanje zahtjevnije..



Slika 65 Lokalna i globalna strategija straničenja

U čemu je razlika između učitavanja po potrebi i učitavanja s predviđanjem?

Kada će se stranica učitati u memoriju? Samo onda kada je MMU traži - ***demand paging*, po potrebi.** Program se sporo pokreće jer u ranoj fazi program zahtijeva više memorijskih lokacija ali kasnije štedimo radnu memoriju. Na brzini možemo dobiti ako koristimo ***anticipatory paging* - predviđamo** koje će stranice biti potrebne i ubacimo ih odmah u memoriju (obično kada procesor ne izvršava ništa). Ovo nam nije problem ako imamo dovoljno interne memorije, a to uglavnom i jeste slučaj. Smanjujemo kasniji broj grešaka stranica.

Algoritam određivanja zamjene stranica: Slučajni izbor.

Treba nam stranica koja nije u memoriji - greška stranice. Neku od stranica treba izbaciti kako bi se u njen okvir upisala ona stranica koja nam treba. Nije svejedno koju stranicu izbaciti. Taktika je **izbaciti bilo koju stranicu.** Prednost je brzina jer ne vršimo nikakvo pretraživanje.

Algoritam određivanja zamjene stranica: FIFO.

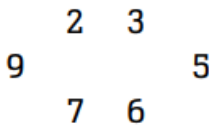
First In First Out - **izbaciti najstariju stranicu** (onu s početka liste).

Nije nam potreba evidencija o korištenju stranica pa se može koristiti i u jednostavnijim sistemima. Pretpostavka kojom se ovdje vodimo jeste da su stranice koje su davno učitane odradile svoje i više nam nisu potrebne. To i ne mora biti tačno, npr. stranica koja pripada glavnoj petlji nekog programa.

Algoritam određivanja zamjene stranica: LDF.

Longest Distance First - gleda se udaljenost između stranica, s tim da se stranice poredaju redom u krug (kao kad računamo po nekom modulu). Npr. na ovoj slici udaljenost između stranica 6 i 9 je 2.

Izbacuje se najudaljenija stranica od tražene. Ako su dvije stranice jednako udaljene izbacuje se ona s manjim rednim brojem. (Ovdje bi se izbacila stranica 5 ako se zatraži 9?) Loše su performanse ako nakon stranice k neće biti potrebna stranica k+1 već neka s drugog kraja memorije.



Algoritam određivanja zamjene stranica: NRU.

Not Recently Used - imamo evidenciju kada se stranicama pristupalo i da li su mijenjane u međuvremenu (preko dva statusna bita R i M, *referenced* i *modified*).

Oba bita su u početku 0 za sve stranice u tabeli a onda se ažuriraju. Na svaki prekid sata (periodično) R bit se briše ali M bit se zadržava zauvijek. Imamo 4 klase i kad se pojavi greška stranice ovaj NRU algoritam **briše onu stranicu koja pripada najnižoj klasi (0, 0).**

0 R=0 | M=0 ... niti referencirana niti mijenjana
1 R=0 | M=1 ... mijenjana ali joj se nije pristupalo nakon tog
2 R=1 | M=0 ... nije mijenjana ali joj se nedavno pristupalo
3 R=1 | M=1 ... mijenjana i ponovo joj se pristupalo (definitivno ostaviti u memoriji)

Algoritam određivanja zamjene stranica: Optimalni algoritam.

Izbacuje se stranica za koju se smatra da će **najkasnije biti potrebna u budućnosti** (ili još bolje da uopće neće biti potrebna).

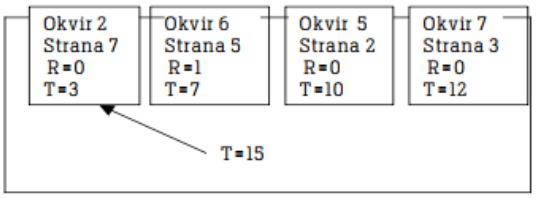
Ovaj algoritam rezultuje najmanjih brojem grešaka stranice ali ga nije moguće realizirati (ne možemo tako predvidjeti ponašanje programa). Optimalnom algoritmu se jedino možemo približiti **praćenjem prošlosti** i dosadašnjem pristupanju stranicama (LRU).

Algoritam određivanja zamjene stranica: Druga šansa.

Ovaj algoritam rješava problem FIFO (ona stranica koja se ubacuje u memoriju samo na početku glavne petlje, izuzetno nam je bitna pa ne bi trebala biti izbačena). Pored starosti stranice uzimamo u obzir i **bit pristupa/referenciranja R.**

Ako FIFO pođe izbaciti najstariju stranicu, Second Chance će ipak malo pričekati i prvo pogledati R bit. Ako je R=0 izbacit će je, a ako je R=1 znači da je nedavno korištena pa se ostavlja u memoriji ali se bit R postavlja na 0 i vrijeme dolaska stranice se postavlja na trenutno. Ako se stranica tako ipak zadrži u memoriji, potraga za idućom se nastavlja od stranice nakon nje (po vremenu dolaska) - opet se gleda R bit, možda se i njoj da druga šansa itd.

Algoritam određivanja zamjene stranica: Satni (Clock page) algoritam.



Isto kao Second Chance ali se stranice čuvaju **u kružnoj listi**. Imamo jedan pokazivač koji pokazuje na najstariju učitanu, njezin R bit se provjerava i ista priča kao i ranije. Novina je u tome što se stranice ne pomijeraju već pokazivači pa dobijamo na brzini i performansama.

Algoritam određivanja zamjene stranica: LRU (Least Recently Used).

Inverzija optimalnog algoritma (aproksimiramo ga) - umjesto odbacivanja stranice koja će najkasnije u budućnosti biti korištena, mi odbacujemo stranicu koja se **najranije u prošlosti** koristila (ako gledamo linearno to je skroz skroz lijevo). I ovaj je teško ostvarljiv jer zahtijeva posebnu hardversku podršku - R i M bitovi nam nisu dovoljni (kao kod NRU): treba nam i vrijeme zadnjeg pristupa a ne samo činjenica je li stranici pristupano.

Proteklo vrijeme možemo evidentirati brojačem (ovo je ta dodatna hardverska podrška?) koji se **uvećava nakon svake procesorske instrukcije.** Kada se desi greška stranice pregledaju se svi brojači (i izabire se onaj s najvećom vrijednošću).

Algoritam određivanja zamjene stranica: LFU (Least Frequently Used).

Svakoj stranici se opet dodjeljuje brojač (i postavlja se na 0) na koji se nadodaju R bitovi - brojač se **uvećava svaki put kad se stranici pristupa** (NE nakon svake procesorske instrukcije). Izbacuje se ona stranica **s** najmanjom vrijednošću brojača. Ovaj algoritam se negdje u literaturi zove i *NFU - Not Frequently Used.* Znači, radi na principu akumuliranog pamćenja pristupa stranica.

Dešava se, doduše, da se neke stranice tipa intenzivno koriste samo na početku (instrukcije za inicijalizaciju programa recimo) i onda tu "naberu" svoje brojače, a kasnije nam više ne trebaju. Termin za ovako stečene brojeve je "stara slava". To je motiv za uvođenje *MFU* algoritma (*Most Frequently Used*) koji se vodi logikom da su se stranice s najmanjim brojačima tek nedavno počele koristiti pa da su više potrebne. Nije vrijedno doduše, kao *Worst Fit* i slični algoritmi, nanose više štete nego koristi.

Algoritam određivanja zamjene stranica: Algoritam starenja.

Izbacuju se ponovo stranice s najmanjim brojem pristupa **bez problema "stare slave"**. Ideja je da se brojač pomjera udesno i uzima u obzir samo npr. zadnjih 8 prekida sata, tako da će se sve stranice zadnji put korištene prije tih 8 prekida tretirati ravnopravno, odnosno kandidati za brisanje će biti i stranica zadnji put korištena prije 10 prekida sata i ona zadnji put korištena prije 10000 prekida.

Šta je to radni skup stranica?

Working set ili radni skup stranica je skup svih stranica koje proces u određenom trenutku koristi. U početku je on prazan i sistem generiše mnogo grešaka stranica dok se sve neophodne stranice za rad ne ubace u radnu memoriju. Nakon toga, broj grešaka stranica opada. Radni skup ne bi trebao biti ni premal ni prevelik, a njegova veličina prirodno varira tokom izvršavanja procesa.

Šta je to Beladejeva anomalija?

To je fenomen gdje više fizičkih okvira može proizvesti i veći broj grešaka stranice nego da imamo manje okvira. Belady je 1969. otkrio da FIFO s 4 okvira ponekad proizvodi više grešaka stranice nego s 3 okvira. Ovo se ne javlja s optimalnim algoritmom niti njegovom LRU aproksimacijom.

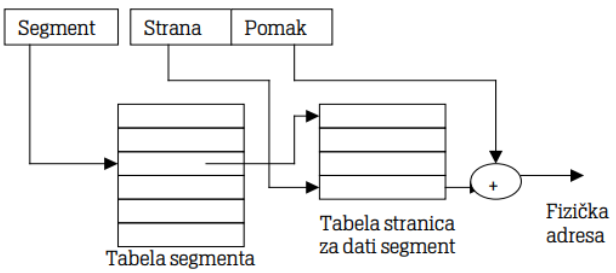
Kako izgleda segmentacija sa straničenjem?

Segmentacija i straničenje su dva slična metoda organizacije memorije. Oba imaju prednosti i mane (vezano za organizaciju, zaštitu aplikacija, fleksibilnost implementacije, tipu fragmentacije...).

Prva razlika je u tome što aplikativni programer nije svjestan straničenja a segmentacije jeste i u svojim programima adresu uvijek navodi kao uređen par (segment, pozicija). **Druga razlika** je što su segmenti različite veličine, a sve stranice su iste. **Treća razlika** je što kod segmentacije imamo više adresnih prostora a straničenje ima samo jedan.

Neki računarski sistemi imaju hardver koji omogućava izbor između ova dva metoda ili možemo birati njihovu **kombinaciju - segmentaciju sa straničenjem:**

Po planu programera, adresni prostor je razbijen u niz segmenata. Svaki segment je dalje razbijen u niz stranica (stranice su fiksne veličine, jednake su veličini fizičkog okvira). Ako se desi da je segment manji od stranice onda on zauzima cijelu jednu stranicu. Odnosno, imamo tabelu segmenata pa tabelu stranica za dati segment.



U uređenom paru (segment, pozicija) *segment* nas vodi do specifičnog segmenta unutar *tabele segmenata,* te on onda otvori *tabelu stranica* za taj segment. Unutar *pozicije* (ovog sad programer nije svjestan) imamo situaciju *stranica+pomak,* gdje nas *stranica* pozicionira na specifičnu stranicu unutar *tabele stranica*. Odatle nastavljamo onoliko koliko nam *pomak* kaže.

Koja je uloga LDTR i GDTR registara na i386?

Na one tabele segmenata (sa ili bez straničenja) mogu pokazivati dva tipa registara: LDT registar (***Local Descriptor Table***) ili GDT registar (***Global Descriptor Table***). LDTR pokazuje na tabelu segmenata za proces, a GDTR na tabelu segmenata jezgre OS.

Kako izgleda deskriptor na i386?

Deskriptor je element tabele segmenata, odnosno **segment + početna adresa, veličina, je li u memoriji ili nije, prva pristupa, tip**… Zahtjeva 8 B memorije. Postoje 3 tipa inače: sistemski, izvršni i deskriptor za podatke. Kompletan format deskriptora se može predstaviti tabelarno.

Kako se linearna adresa transformiše u fizičku na i386?

Linearna, ne logička! Krećemo od segmenata.

Ako imamo samo segmentaciju bez straničenja onda se linearna adresa interpretira kao fizička.

Ukoliko uz segmentaciju imamo i straničenje onda se linearna adresa interpretira kao virtualna (logička) i transformira se u fizičku uz **straničenje na dva nivoa** (tabela segmenata s tabelama stranica, priča od ranije). Do početne fizičke adrese dolazimo kada se pozicioniramo unutar tabele stranica odabranog segmenta. *Pomak* nas dovede do fizičke adrese koja nam je potrebna.

Da bi to ubrzali možemo koristiti *deskriptorske keševe* (pamte sadržaje deskriptora za trenutne segmente) i *TLB registre* (pamte nekoliko zadnjih mapiranja stranica-okvir).

Koje prijetnje postoje za operativne sisteme?

- **prirodne katastrofe:** požari, potresi, poplave
- **hardverske greške:** nestanak električne energije, kvarovi diskova, pregorijevanje

Na ovo se možemo samo pripremiti ali ne možemo spriječiti. Treba raditi redovan backup na udaljenom serveru, ne na vlastitom računaru, možda čak i ne u istoj zgradi, na internetu recimo.

- **softverske greške:** programerske greške u kodu, često su neminovne, prekoračenje buffera.
- **zlonamjerni softver** (više o ovome u idućem pitanju)
- **ljudski faktor** (namjerno ili nenamjerno)**:** pogreško otkucana komanda koja briše drugu datoteku umjesto one koja je trebala biti obrisana, otkrivanje lozinke, postavljanje privatnih podataka na javni server, svjesno uništavanje podataka, špijunaža, obaranje sistema…

Koje su vrste zlonamjernog softvera?

Programi napisani ciljno da uzrokuju štetu: **virusi, crvi, trojanac, logička bomba** i **spyware.**

Virus se zalijepi za drugi program i pokreće se kada se i program pokreće.

Crv je neovisan program što znači da mu nije potreban drugi program na koje će se "nakačiti", bitno mu je samo da vlastite kopije doguraju do drugih računara.

Trojanac izgleda kao sasvim normalan program (npr. igrica) kojeg skidamo s interneta, a čiji štetni efekti prolaze tiho i nečujno.

Logička bomba se pokreće iz nekog programa (kao tajna opcija) pod određenim uslovima, npr. program za knjigovodstvo ukoliko vide da na autorov žiro-račun nisu uplaćeni neki novci.

Spyware špijunira, odnosno šalje informacije o korisničkom računaru, obično osobama koje bi to mogle iskoristiti u loše svrhe. Npr. software instaliran kao dodatak web browser-u koji koristi informacije o najčešće posjećenim stranicama.

Šta je matrica kontrole pristupa i koji su načini pogleda na nju?

Access Control Matrix **(ACM)** za redove ima korisnike, za kolone resurse (obično datoteke), a elementi matrice su prava koja dobivaju svi procesi pokrenuti od strane tih korisnika. Većina korisnika nema pristup većini datoteka, tako da je to u praksi vrlo rijetka (gotovo prazna) matrica koja zauzima previše prostora. Dva su načina pogleda na ovu matricu:

1. Umjesto matrice, možemo **svakoj datoteci pridružiti jednu uređenu listu** koja sadrži sve korisnike i njihova prava nad tom datotekom (ACL - *Access Control List*). Možemo imati i grupe korisnika. U biti, radimo evidenciju po kolonama. Npr. za jednu datoteku imamo ovakvu listu pristupa:

   ```
   duje: RW; fikret: RW; grafika: RX
   ```

   R - read, W - write, X - execute, *duje* i *fikret* su korisnici, *grafika* je grupa

2. Drugi pristup pogleda je preko redova. Sada se **svakom korisniku pridružuje lista** prava pristupa nad datotekama. To su **liste mogućnosti** ili **C-liste** (*capabilities*). Za jednog korisnika možemo imati ovakvu situaciju:

   ```
   Dat3:R; Dat4:RX; Dat5:R; Dat8:R; Dat9:RX; Port1:R
   ```

Kako se postiže izolacija između procesa?

Izolacija između procesa se postiže različitim hardverskim i softverskim tehnikama. Na taj način štitimo procese jedne od drugih. Jedna tehnika je odvojiti adresni prostor procesa P1 od adresnog prostora P2. Sprječavamo da P1 upisuje ili čita podatke koji pripadaju procesu P2, i obratno.

Šta je tehnika pješčanika?

Sandboxing izolira nepouzdan softver od ostatka sistema, npr. ako je softver netestiran ili iz nepovjerljivog izvora - stavimo ga “u pješčanik" i tu ima kontrolirani skup resursa koji može koristiti. Taj pješčanik (*sandbox*) je obično **odvojeni direktorij ili memorijsko područje**. Ako taj softver sadrži virus on neće moći izaći iz pješčanika i neće načiniti štetu ostatku sistema.

Drugi način jeste da softveru dopustimo samo minimalan broj sistemskih poziva.

U koje grupe možemo podijeliti kriptografske algoritme?

1) tajni kriptografski algoritmi: sigurni smo sve dok se ne otkrije algoritam, nije preporučljivo

2) algoritmi zasnovani na ključu: algoritam nije tajan, sigurnost leži u ključu, dalje se dijele se na *simetrične* (jedan ključ i za šifriranje i za dešifriranje), *asimetrične* (dva ključa, javni za šifriranje i tajni za dešifriranje) i *hibridne*.

3) algoritmi za jednosmjerno šifriranje: hash funkcije u biti, uopšte nam nije od koristi dekripcija (npr. kod čuvanja lozinki, provjere bilo kakve autentičnosti - dovoljno je uporediti hash vrijednosti pa makar takve nečitljive)

Koje tri klase korisnika i tri vrste prava postoje u Unix sistemima?

Tri klase korisnika: vlasnik, članovi grupe, ostali korisnici.
Tri klase prava: čitanje R, pisanje W, izvršavanje X.

Za svaku datoteku čitamo redom: prva 3 bita su prava vlasnika, iduća 3 bita su prava grupe, zadnja 3 bita su prava ostalih korisnika. Slijede primjeri.

Binarno	Oktalno	Simbolički	Prava s pristupom datoteka
111000000	700	`rwx------`	Samo vlasnik može čitati pisati i izvršavati
110110110	666	`rw-rw-rw-`	Svi mogu čitati i pisati
000000000	000	`---------`	Niko nema pristupa
110100100	644	`rw-r--r--`	Vlasnik može čitati i pisati, ostali samo čitati
100100000	440	`r--r-----`	Vlasnik i grupa imaju pravo čitanja

Slika 178 Neka prava nad Unix datotekama

Šta su SID i pristupni token u Windows sistemima?

SID je sigurnosni identifikator svih subjekata u sistemu: korisnika, grupa, kompjutera… Podsistem **Security Account Manager**, čuva listu svih korisnika. Kada se korisnik prijavi, pokupe se svi SID-ovi tog korisnika i SID-ovi svih grupa kojima on pripada, plus još neki podaci, i to se sve upakuje u **pristupni token procesa** (za sve procese koje pokreće ovaj korisnik?). Ako proces kreira nove procese ti pristupni tokeni se nasljeđuju, ali niti unutar tog procesa mogu imati i sasvim druge pristupne tokene.

Koja je razlika u načinu prijave na radnu grupu, domenu i aktivni direktorij u Windows sistemima?

Imamo tri organizacije računarskih mreža: radne grupe, NT domena, aktivni direktorij.

1) Ako imamo radne grupe onda se **provjera prijave vrši na toj lokalnoj mašini** i tako za svaku mašinu (računar) - **korisnično ime i password**, eventualno se nekad traže i dodatne informacije. Korisnik se mora prijavljivati na svaki računar zasebno. Održavanje je otežano jer u slučaju promjene lozinke korisnik to mora ažurirati na svakom računaru posebno.

2) Ukoliko se prijavljujemo na računar koji je unutar NT domene onda se **provjera vrši na kontroleru domene** (računar preko kojeg se prijavljuje cijela ta domena). Osim korisničkog imena i passworda evidentira se i **grupa kojoj korisnik pripada**. Održavanje je lakše jer se ažuriranje passworda ili dodavanje korisnika ne mora raditi na svakom računaru.

3) **Aktivni direktorij** sadrži traži i **kontaktne informacije, organizacione podatke i certifikate**. Provjeru prijave vrši **kontroler aktivnog direktorija**. Informacije se čuvaju kroz hijerarhijski prostor imena u odvojenim serverima za pojedine dijelove mreže (nije linearna struktura kao kod kontrolera domene). Serveri i dalje mogu surađivati po potrebi.

radna grupa: korisničko ime, password (lokalna mašina)

NT domena: korisničko ime, password, grupa (kontroler domene - linearna struktura)

aktivni direktorij: korisničko ime, password, grupa, kontakt info, organizacioni podaci, certifikati (kontroler aktivnog direktorija - hijerarhijska struktura)

PERIFERIJSKI UREĐAJI I DATOTEČNI SISTEMI

Klasifikujte uređaje po namjeni, smjeru i količini podataka.

Po namjeni, uređaji se dijele na:

a) uređaji za dugotrajno smještanje podataka (diskovi, trake, diskete, CD)
b) uređaji za prijenos podataka (mrežna kartica, ruter, modem)
c) uređaji koji pružaju interface ka korisniku (tastatura, ekran, terminal, miš, štampač)
d) ostali uređaji (sat)

Prema smjeru prijenosa, uređaji se dijele na:

a) ulazne (miš, tastatura, skener)
b) izlazne (printer, monitor)
c) ulazno-izlazne (mrežna kartica, modem, disk)

Prema količini prenesenih podataka, uređaji se dijele na:

a) znakovne (jedinica pristupa znak, bajt)
b) blokovske (jedinica pristupa blok, 512 ili 1024 bajta)
c) mrežne (jedinica pristupa paket, fiksirano ali podesivo)
d) ostale uređaje (uređaj sata opet)

Koje vrste registara U/I uređaja postoje?

- **kontrolni registri:** u njih se upisuju podaci, npr. *zvučna kartica* upisuje jačinu zvuka
- **statusni registri:** opisuju status IO uređaja, isključivo se čitaju
- **ulazni registri podataka:** isključivo se čitaju (podaci koji stižu s nekog uređaja)
- **izlazni registri podataka:** u njih se isključivo samo upisuje, prosljeđuju podatke nekom uređaju (npr. upisujemo tekst koji ide na štampanje štampaču)

Koja je uloga kontrolera prekida?

IO komponenta (na matičnoj ploči) se sastoji od **IO kontrolera, kontrolera prekida i DMA kontrolera.**

Kada CPU želi da komunicira s uređajem, on daje instrukciju IO kontroleru i nastavlja s drugim operacijama. Uloga kontrolera prekida je da pošalje signal prema CPU (preko prekidne linije, *interrupt request line, IRQ*) **kada je uređaj spreman.** CPU nema potrebu da provjerava uređaje ima li tko šta za slanje. Kada dobije signal prekida, CPU spašava svoje trenutno stanje i prelazi na komunikaciju s uređajem. Ukoliko se više prekida desi istovremeno, kontroler prekida donosi odluku čiji prekid obraditi na osnovu **prioriteta prekida.**

Šta je to DMA kontroler?

Motiv za uvođenje ovog dodatnog uređaja (u sklopu IO komponente) je neefikasnost kod prenosa velike količine podataka između uređaja i memorije kada to ide preko procesora. DMA kontroler koristi sistemsku sabirnicu da prenose podatke bez angažovanja CPU (CPU samo mora poslati nešto malo instrukcija DMA kontroleru i tjt).

U kojim režimima radi DMA kontroler?

CPU i DMA ne mogu koristiti sistemsku sabirnicu u isto vrijeme - moraju naći način da je dijele.

burst režim - DMA kad dobije sabirnicu blokira procesor i odjednom prenosi čitav blok (512 ili 1024) podataka, najbrži prenos ali zadržava procesor

režim krađe ciklusa - malo manja brzina jer DMA prenosi bajt po bajt podatke, odnosno blokira procesor prije prenosa svakog bajta i onda ga odblokira

transparentni režim - najsporiji prenos ali nikad ne zaustavlja procesor nego prenosi podatke isključivo kada procesor ne pristupa sabirnici, kompleksna realizacija u praksi

Razlika između memorijski mapiranog U/I i U/I s odvojenim adresnim prostorom.

S memorijski mapiranim UI, CPU vidi **UI kontroler** kao običnu lokaciju u memoriji - CPU komunicira s UI kontrolerom na način da piše ili čita podatke iz ove lokacije. Ekran je praktično ovako programirati. Inače je ovaj tip uređaja lako realizirati.

S UI mapiranim pristupom, UI kontroler ne dijeli adresni prostor s memorijom, nego se koriste specijalne instrukcije za komunikaciju CPU s kontrolerom. Ovo daje više adresnog prostora i za memoriju i za UI, ali zahtijeva **dodatnu liniju procesora prema sabirnici** kako bi se znalo da li se pristupa memoriji ili UI kontroleru. Gubimo i malo neke fleksibilnosti u komunikaciji.

Kako se rješava problem u obrađivačima interapta: da bude što brži i uradi što više?

Obrađivač prekida, u kratici **ISR** (od *Interrupt Service Routine),* je potprogram koji jezgro izvršava svaki put kada se desi neki prekid.

Spomenuti konflikt se rješava razdvajanjem obrađivača na **gornju polovinu** i **donju polovinu**. U toku gornje polovine obavljaju se **vremenski kritični poslovi** (npr. potvrda prijema), nakon čega jezgro ili korisnički program mogu nastaviti od mjesta gdje su prekinuti. Donja polovina se može izvršiti i nekad kasnije, kad je pogodno.

Koja je glavna razlika u funkcijama drajvera blokovskih i znakovnih uređaja?

Glavna razlika je u operacijama čitanja i pisanja. Kod znakovnih uređaja čitanje i pisanje se izvršavaju u toku sistemskih poziva read i write. Kod blokovnih uređaja zahtjevi za čitanje i pisanje se stavljaju u red čekanja za asinhroni pristup.

Koja su tri osnovna pristupa u realizaciji čitanja i pisanja kod drajvera znakovnih uređaja i po čemu su karakteristični?

Znači radimo sa znakovnim uređajima, i na tri načina možemo realizirati operacije čitanja i pisanja: programirani UI, UI vođen prekidima, DMA bazirana komunikacija. Uglavnom, imamo komunikaciju između CPU i periferijskog uređaja, sad je pitanje na koji način.

Programirani ulaz/izlaz je najjednostavnija direktna metoda za komunikaciju između CPU i uređaja - CPU je za sve odgovoran. Ovo može bit neefikasno jer bi procesor, umjesto što čeka na UI kontroler, mogao izvršavati druge procese. Nema posrednika, nema prekida. CPU periodično "proziva" uređaje da provjeri jesu li spremni za slanje.

UI vođen prekidima ne zahtjeva procesorsko prozivanje da li je uređaj spreman za slanje, nego uređaj to sam javlja (kontroler prekida?). Imamo sinhroni i asinhroni dio slanja podataka. I kod **DMA bazirane komunikacije** također imamo sinhrone i asinhrone dijelove drajvera. To ide preko onog DMA kontrolera o kojem je već bilo riječi.

Šta je to bafer?

Bafer je dio memorije koji služi za čuvanje privremenih podataka prilikom prenosa podataka između dva uređaja ili između uređaja i aplikacije. Funkcionira na principu proizvođač-potrošač. Potpuno je moguće imati i komunikaciju bez bafera, ali i sa jednim ili dva bafera. Bafer može biti pozicioniran i u korisničkom prostoru ali i u jezgru.

Šta je to spooler?

Spooler je pozadinski proces (najčešće vezan za štampače i crtače) koji omogućava istovremeni pristup nedjeljivim uređajima. Proces postavlja zahtjev u **bafer za štampanje**, nastavlja s drugim aktivnostima - spooler onda gleda redoslijed dolaska tih podataka u bafer i šalje ih redom štampaču.

Kako se imenuju uređaji u korisničkom prostoru u Linuxu?

Simbolička imena svih uređaja su smještena u direktoriju **/dev** (od *device*).

U tabeli su neki blokovski i znakovni uređaji sa svojim standardnim imenima. Ta simbolička imena se kreiraju alatom ***mknod*** koji poziva istoimeni *mknod sistemski poziv* (preko sistemskih poziva uređajima znači dajemo imena).

mknod /dev/mouse0 kreira simboličko ime *mouse0* (vjerovatno za uređaj miš)

Blok Uređaj	Namjena	Znak uređaj	Namjena
/dev/fd0	Prvi floppy disk.	/dev/dsp	Digitalni zvučni izlaz
/dev/hda	Prvi IDE disk	/dev/fb0	Pristup video memoriji
/dev/ht0	IDE uređaj trake	/dev/tty1	Prva konzola
/dev/cdrom	Glavni CDROM uređaj	/dev/ttyS0	Prvi serijski port
/dev/loop0	Datoteka koja simulira disk	/dev/lp0	Prvi paralelni štampač
/dev/md0	RAID struktura više diskova	/dev/mixer	Mikser zvučne kartice
/dev/pda	IDE disk, paralelni port	/dev/zero	Stalno šalje vrijednost 0
/dev/pcd0	CDROM, paralelni port	/dev/psaux	PS/2 miš
/dev/sda	Prvi SCSI disk	/dev/random	Šalje slučajne brojeve

Dodamo tip, recimo **c za "znakovni"**, **oznaku drajvera 13** i **dodatnu informaciju 32** (nebitno šta znači), pa onda uređaj imenujemo na sljedeći način: `mknod /dev/mouse0 c 13 32`

Kako se imenuju uređaji u korisničkom prostoru u Windowsu?

Kod **Windows NT** ObjectManager imenuje uređaje interno, što korisniku nije vidljivo, recimo prvi CD ROM uređaj je \Device\CdRom0.

Uređaji čija su imena dostupna korisniku su napravljena da pored **internog imena** imaju i **globalno ime (alternativno definirano)** i ona počinju besmislenim imenom direktorija **\\.** (moramo imati neki direktorij da bismo imali datoteku). pa opet \ pa ime datoteke:

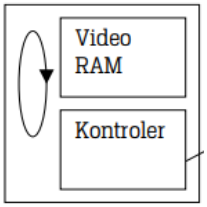
Globalno Ime	Uređaj
\\.\PhysicalDrive0	Prvi fizički disk
\\.\COM1	Prvi serijski port
\\.\LPT1	Prvi paralelni port
\\.\DISPLAY1	Prvi ekranski uređaj.

Kako se konvertuje scan kod u ASCII?

Tasteri na tastaturi se označavaju *scan kodovima*. Tastatura može generirati prekid koji se dešava kada je taster pritisnut ili otpušten - tada se očitava koji je taster pritisnut i ide u ulazni bafer. Iz ulaznog bafera **drajver tastature** očitava *scan kodove* i treba iste konvertovati u razumljive znakove (ASCII). Npr. pritisne se taster A, u ulazni buffer ide njegov *scan kod* 30, onda drajver odredi je li to a, A, CTRL-A, ALT-A i slično.

Kako promjena podatka u video memoriji utiče na sliku na ekranu?

Područja na ekranu su povezana s odgovarajućim područjima u memoriji, što znači da se promjena u memoriji reflektuje na promjenu na ekranu.



Grafički adapter (kartica) u sebi sadrži video RAM memoriju i video kontroler. Video RAM je dio adresnog prostora računara - komunikacija između video RAM-a i CPU se odvija kao i komunikacija između CPU i ostatka memorije. **Ovdje je pohranjena slika ekrana** (u vidu znakova ili bitmape). Video kontroler očitava iz video RAM-a znakove/bitove i **generira video koji vidimo na ekranu**.

Šta su to ANSI sekvence u ispisu na terminal?

ANSI sekvence su dodatne mogućnosti koje podržavaju neki drajveri za tekstualni ekran (nisu sastavni dio ASCII znakova). **ANSI *escape* sekvence** imaju posebno ponašanje nakon ESC znaka, i neke mogu imati parametre. Prije svega su namijenjene za precizno podešavanje kursora na ekranu:

- pomjeri se x linija gore ili dolje
- pomjeri se x znakova lijevo ili desno
- obriši liniju od kursora do kraja
- obriši ekran od kursora do kraja
- režima **bold** slova

Šta je to paleta?

Sada je video RAM predstavljen preko **bitmape** gdje jedan bit ili više bitova predstavlja jedan *pixel* na ekranu. Ukoliko imamo mapiranje *jedan bit == jedan pixel* onda imamo samo dvije boje na ekranu (nebitno koje, žuta i zelena, najčešće crna i bijela). Općenito, ako odvojimo *n* bita za svaki *pixel* onda možemo imati ukupno 2^n različitih boja.

Paleta je skup boja koji imamo na raspolaganju, i ovisi od ovog broja n bita koje dodijelimo za svaki *pixel*. Npr. ako svaki pixel predstavimo sa 3 bajta (24 bita) onda možemo imati 2^{24} različitih boja (kvantitet palete).

Šta radi funkcija WriteConsole u Windows?

tekstualne aplikacije == konzolne aplikacije

WriteConsole (+parametri) je konzolna funkcija ispisuje tekst na konzolni ekran. Unutar parametara navodi se tekst koji želimo ispisati, broj karaktera koji taj tekst sadrži itd.

Koja su četiri osnovna dijela Windows API grafičkog programa?

1. registracija prozorske klase: svaki prozor mora imati povezani objekt klase WNDCLASS sa 10 atributa koji definiraju svojstva prozora - najvažniji atribut je pokazivač na funkciju koja upravlja proslijeđenim porukama (iz petlje)

2. kreiranje i prikaz prozora: *CreateWindow, ShowWindow*, *UpdateWindow*

3. petlja koja prosljeđuje poruke: dobavlja poruke, vrši konverziju, prosljeđuje dalje da Windows pozove proceduru za obradu (onaj spomenuti atribut)

4. prozorska procedura koja obrađuje poruke: malo je komplicirano kako zapravo radi

Čemu služi terminal i šta je njegova moderna verzija?

Terminal je u izvornom značenju računarski sistem koji **služi za komunikaciju s centralnim kompjuterom** (a taj ima zadatak izvršavanja OS i korisničkog programa). Terminal to posljednje ne radi - on ima samo svoj mikroprocesor, sposobnost prikaza teksta koji mu se pošalje (bez grafike), ima naravno u tu svrhu i monitor i tastaturu...

Ranije su se koristili RS-232 terminali a danas se i osobna računala mogu koristiti kao terminali - imamo mrežne kartice i protokole koji nam omogućavaju pristup udaljenom računaru (prvo **telnet**, onda kriptirana verzija **ssh**).

Štaviše, danas se termin *terminal* uopšte ne veže za poseban računarski sistem, već više kao interfejs preko kojeg pristupamo drugom računaru. Pojavljuju se i koncepti **tankih klijenata i zero klijenata.** Podizanje OS računara **tankog klijenta** se može obaviti preko mreže recimo, dok **zero klijent** računar uopšte nema OS nego koristi neki specijalizirani procesor.

Koje su vrste udarnih a koje beskontaktnih štampača?

Udarni printer je najstarija tehnologija a koja se još uvijek proizvodi (u neke specijalizirane svrhe, tu su funkcionalni a najjeftiniji su). Imamo **udarni matrični, udarni lepezni, udarni linijski**. Ovo su one stare stare mašine što bukvalno UDARAJU.

Kod beskontaktnih štampača u dodir s papirom ne dolaze njegovi metalni dijelovi nego samo tinta. Puno su dakle tiši i štampa je boljeg kvaliteta. Imamo vrsta: **termalni, *inkjet*, laserski, štampači termalnog voska, štampači sa sublimacijom boje, štampači čvrste tinte.**

Termalni štampači koriste specijalan papir osjetljiv na temperaturu ili električni naboj. Odštampani tekst vremenom nestaje sam od sebe (hladi se). To su oni štampači za štampanje računa u prodavnici.

Inkjet štampač koristi boju na bazi vode koja se brzo suši i niz malih mlaznica rasprkava tintu na papir. Mogu imati i samo jednu mlaznicu za crno-bijeli ispis ili 4 mlaznice za štampanje u boji (to se naziva CMYK kombinacija za cijan, magenta, žuta i crna). Izgledaju isto kao laserski.

Laserski štampači su oni što većina kući ima - iz ladice za papir vuku jedan po jedan papir kroz valjak, površina ispisnog bubnja se isprazni i na papiru ostanu tačkice u vidu našeg teksta (papir i bubanj tu imaju različita naelektrisanja). Možemo koristiti samo crni toner ali imamo i laserske štampače u boji koji koriste opet CMYK toner kombinaciju.

Za malo kvalitetnije boje se koriste štampači termalnog voska, čvrste boje i sa sublimacijom boje.

Navedite primjere jezika za opis stranica.

Page Descriptor Language (PDL) je specijalizirani jezik za formatiranje dokumenata u svrhu komunikacije računara sa štampačem. Svaki proizvođač ima svoj jezik za formatiranje dokumenta koji radi na njihovom štampaču i nema garancije da će taj jezik biti kompatibilan s drugim štampačem. Rade na istom principu kao i obični programski jezici.

PDL Epson štampača (doba udarnih matričnih štampača) pod nazivom ESC/P
Adobe-ov PDL PostScript
Printer Command Language (PCL)

Na koji način Windows priprema stranicu za štampu?

Sistemski poziv ***StartDoc*** za početak, onda pozivi za crtanje ***TextOut*** i ***BitBit*** (komande se privremeno čuvaju). Nakon poziva ***EndDoc,*** sačuvane komande se konvertuju u odgovarajući PDL (jezik štampača) i tako dobiveni podaci se šalju ***spooler-u*** (spoolsv.exe) koji to šalje štampaču.

Šta je CUPS?

Unix isprva nije imao pozadinski proces *spooler* nego su se podaci direktno slali na štampač. Tek se kasnije uveo printer *spooler* ali prilagođen samo linijskim (udarnim) štampačima. Onda se uveo **CUPS (*Common Unix Printing System*)**, kompletan sistem za upravljanje štampačima - nešto kao *spooler* ali i više.

- poznaje novi *Internet Printing Protocol (IPP)*
- može se konfigurisati komandnim, grafičkim ili web interfejsom
- po defaultu koristi PostScript jezik
- ima programe koji različite tekstualne i grafičke formate mogu prevesti u PostScript
- također može PostScript prevesti u format koji odgovara štampaču

Koji principi dodirnih ekrana postoje?

rezistivni - dva sloja provodnog materijala (jedan ima vertikalne a drugi horizontalne linije, zajedno određuju poziciju na ekranu koja je dodirnuta)

kapacitivni - iz sva četiri ugla ide vrlo mala struja, kada se ekran dodirne lokacija se odredi na osnovu jačine elektriciteta koji dolazi iz tih uglova

infracrveni - vertikalne i horizonalne infracrvene zrake koje korisnik prekida dodirom

SAW (*Sound Acoustic Wave*) - ekran emituje visokofrekventne zvučne talase (vertikalne i horizontalne), korisnik te talase prekida dodirom

Koji tri vrste naredbi poznaju interpreteri komandi?

komandni interpreteri - programi za izvršavanje komandi i programa u skriptnim jezicima.
skriptni jezik - jezik namijenjen za pokretanje drugih aplikacija i manipulaciju datotekama

komande su oblika *naredba + parametri*

Naredbe mogu biti **interne, uključene** (dio OS) **i vanjske** (nisu dio OS).

Koja je razlika između Unix komandi cp, mv i ln?

cp služi da **kopira** datoteku u drugu datoteku ili direktorij (izvor, odredište)
mv služi da **premjesti** datoteku u drugu datoteku ili direktorij (izvor, odredište)
ln služi za kreiranje **simboličke veze**, plitke kopije, shortcut

cp datoteka1.txt /tmp
cp datoteka1.txt datoteka2.txt
ln datoteka1.txt datoteka2.txt - promjena sadržaja prve datoteke će se odraziti i na drugu

Čemu služe Window manager, Display manager i Session manager?

Display manager se aktivira odmah u startu kako bi se koristici mogli prijaviti na računarski sistem u grafičkom režimu rada. Znači, kada se on aktivira korisnici kucaju korisničko ime i password.

Window manager je pozadinski program koji crta ukrase oko prozora, omogućava njihovo pomjeranje, širenje, maksimiziranje, Start dugme itd.

Session manager čuva stanje aplikacije - položaje i sadržaj svih otvorenih prozora. Omogućava da se odjavimo i ponovnom prijavom pristupimo istom sadržaju kojeg smo ostavili prije odjave.

Display manager i session manager komuniciraju posebnim protokolom. Ako se sva tri menadžera pišu sa zajedničkim ciljem da čine jedinstvenu cjelinu onda dobivamo **desktop okruženje**.

Koji su slojevi ISO/OSI mrežnog modela?

1. fizički sloj: definiše hardver, komunikacione kanale
2. sloj veze podataka: okviri, kontrolna suma, kontrola pristupa mediju, dva susjedna čvora
3. mrežni sloj: paketi, komunikacija računar-računar, tabele rutiranja
4. transportni sloj: segmenti, mehanizmi potvrde, retransmisija, protokoli UDP, TCP
5. sesijski sloj: uspostavljanje i prekidanje sesije
6. sloj prezentacije: formatira podatke da odgovaraju aplikaciji, (de)kompresija, kriptografija
7. aplikativni sloj: email, web server, telnet, ssh

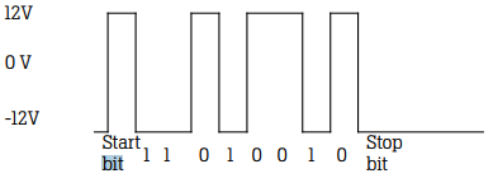
TCP/IP ima samo četiri sloja (fizički i sloj veze su jedan sloj, sloja sesije i prezentacije nema).

Kako se bit predstavlja na RS232?

RS232 je jedan od najstarijih standarda (protokola) za serijsku komunikaciju. Signali se šalju bit po bit kroz jednu liniju (žicu).

1 se predstavlja bilo kojim naponom između -15V i -3V.
0 se predstavlja bilo kojim naponom između +3V i +15V.

+12V je startni bit (0)

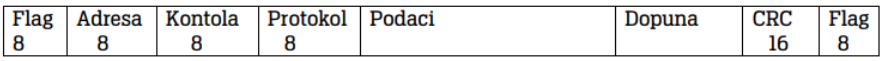


Kako se podaci pakuju u SLIP a kako u PPP protokolu?

Nalazimo se u sloju veze (radimo s okvirima). SLIP protokol šalje okvir u formatu: **podaci+END znak (kraja prenosa).** Nemamo kontrolu grešaka.

Slika 106 SLIP okvir

U PPP okviru prije podataka imamo **adresu, kontrolu i polje za protokol mrežnog sloja** (iznad, obično IP). Početak i kraj okvira su **flegovi**, a prije zadnjeg flega imamo još i **dopunu+CRC** (kontrolna suma, *Cyclic Redundancy Check*).



Slika 107 PPP okvir

Šta je MAC adresa?

MAC (*Media Access Control*) je **jedinstvena adresa od 48 bita** koju ima svaka mrežna kartica. Tih 48 bita se predstavlja u heksadecimalnom obliku (4 bita za svaki hex znak, znači 12 hex znakova). Prva polovica je za proizvođača a druga za serijski broj kartice, znači PP-PP-PP-SS-SS-SS.

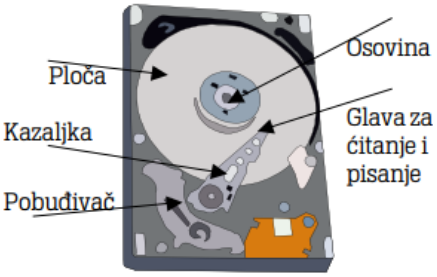
Šta je IP adresa?

IP (*Internet Protocol*) adresa je 32-bitna adresa (IPv4) koja se predstavlja kao 4 broja odvojena tačkom, od 0-255 (8 bita za svaki) i radi na mrežnom sloju. Unutar te adrese imamo nekoliko fiksnih bita koji su isti kod svih računara unutar iste mreže (ako je fiksih n bita onda subnet maska ima n jedinica), a ostali bitovi su jedinstveni za svaki računar u mreži. Može se još naširoko o ovome... IPv6 je 128-bitni protokol.

Kada se koristi TCP a kada UDP protokol?

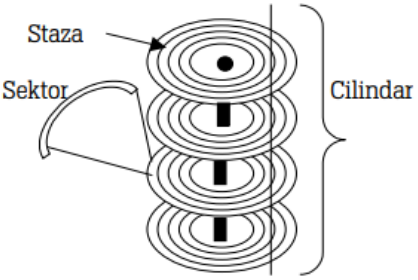
Oba su protokoli transportnog sloja, TCP je konekcioni, pouzdan, s potvrdom, a UDP je nepouzdan, beskonekcioni, bez potvrde. UDP se koristi kad nam ne treba potvrda prijema, možda nije bitno ukoliko bude grešaka i neki paketi ne stignu, bitnija nam je brzina (multimedija npr.). TCP se koristi svaki put kad je od važnosti očuvanje podataka - kombinuje se recimo s IP.

Kako je organizovan hard disk?



Hard disk se sastoji od magnetnih ploča koje u centru imaju zajedničku osovinu, glave za čitanje/pisanje (koja miruje, a ploča rotira ispod glave), pobuđivača i pobuđivačke kazaljke.

Disk se vjerovatno poistovjećuje s pločom? Svugdje gdje kaže disk misli ploča, a cijeli hard disk je uređaj?



Na ploči imamo **staze** (koncentrični krugovi) i svaka je podijeljena na **sektore** i svi sektori su međusobno razdvojeni međuprostorom (i susjedne staze su odvojene međuprostorom, radi smanjenja interferencije). Staze se broje počevši od vanjske strane ploča (staza 0) pa prema centru.

Kako na jednom hard disku imamo više ploča oko iste osovine, ako gledamo sve staze koje su jednako udaljene od osovine na svim pločama onda definišemo jedan **cilindar** (doslovno šuplji valjak).

Šta je zonski zapis?

... zone recording

Staze bliže centru imaju manju dužinu, pa nam njih stanje manje sektora (to su odsječci staze) nego na vanjske staze. Uzimamo u obzir da je veličina sektora ista. To je motiv da se uvede zonski zapis - **cilindri se grupišu u zone** tako da svi cilindri u istoj zoni imaju **jednak broj sektora**.

Šta je zapis u formi crijepa?

... shingled magnetic recoring

Tehnika kod koje se upis u jednu stazu **preklapa dijelom i na susjedne staze**. Pri ažuriranju jedne staze se moraju pročitati i obje susjedne, onda se piše u sve tri, i to usporava čitav proces upisa. S modernim diskovima doduše ne primijetimo ovo usporavanje jer imamo dovoljno RAM-a.

Koja je formula prosječnog vremena pristupa diska?

Parametri koji nam prvo trebaju su:

search time Ts (prosječno vrijeme traženja)
r - rotaciona brzina
b - broj bajta za prijenos
N - broj bajta na stazi

access time Ta = Ts + 1/2r + b/rN (prosječna brzina pristupa) $Ta = Ts + \frac{1}{2r} + \frac{b}{rN}$

Što je veća rotaciona brzina manje je vrijeme pristupa.
Što je veći broj bajtova za prenijeti više nam i vremena treba.

Šta je formatiranje na niskom nivou?

Ovo je prvi korak u pripremi diska za rad - **upisuju se oznake za granice staza i sektora**. Poslije slijedi kreiranje particija i formatiranje particija. Ovaj proces je na vrlo starim diskovima radio sam kupac ali se danas taj proces radi isključivo fabrički.

Slika 122 Sektor formatiran na niskom nivou sa redundantnim podacima

Znači, pripreme se mjesta za sektore, staze i međuprostor.

Koja je uloga Master boot record?

Particije se ponašaju kao zasebni diskovi i više njih se kreira onda kada želimo više operativnih sistema, ali možemo imati (i često imamo) samo jednu.

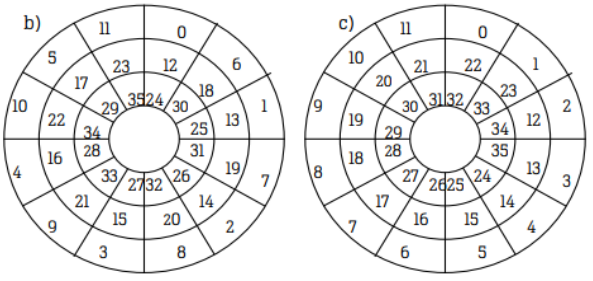
Master Boot Record je sektor koji se nalazi na samom početku diska (površina 0, staza 0, sektor 0) i **sadrži listu svih particija.** Nakon pokretanja računara kratki program unutar MBR pročita tu tabelu particija, nađe se **aktivna particija** i pročita se njen prvi sektor (**startni sektor)** gdje se nalazi program čijim pokretanjem započinje bootstrap rutina i **učitava se OS u memoriju**.

Kada se koriste preplitanje i zakretanje?

Logički u fizički sektor možemo transformisati **prostom transformacijom, preplitanjem ili zaktretanjem**. Ovo je ono gdje logičke sektore u krugu numerišemo po nekom pravilu (ali uvijek krećemo brojati od vanjskih staza ka unutrašnjim, i krećemo od 0).

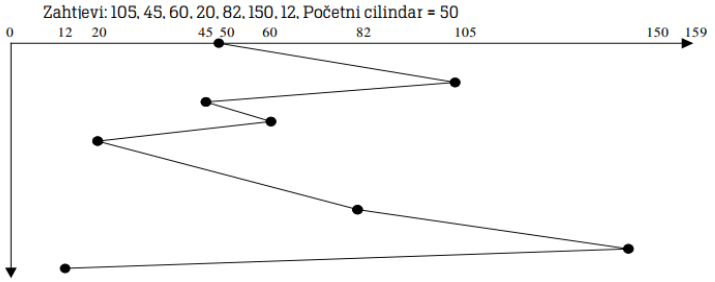
Preplitanje i zakretanje koristimo kad **na sporijim računarima prijenos prethodnog sektora u bafere nije dovoljno brzo odrađen - glava diska je već došla na naredni sektor.** Taj naredni sektor mora čekati cijelu rotaciju diska da bi se mogao pročitati.

Kod preplitanja **preskače se po jedan ili dva sektora**. Najbolje se vidi sa slike b). Kod zakretanja se enumeracija vrši klasično kao kod proste transformacije, s tim da kad prelazimo u donju stazu nastavljamo brojanje **od sektora "koji bi nas zadesio"** usljed rotacije diska, odnosno iznad kojeg sektora bi se nalazila glava (slika c).



Algoritam za raspoređivanje diska FCFS.

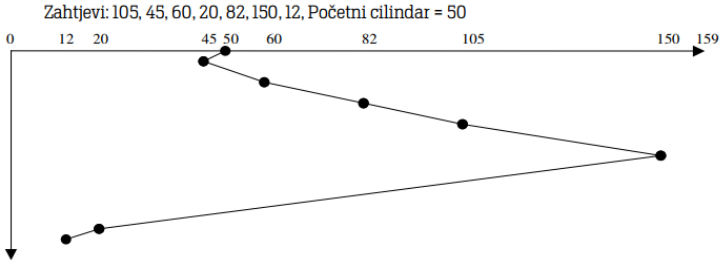
First Come, First Served - Od inicijalne pozicije opslužujemo zahtjeve za čitanjem cilindara onim redom kojim pristižu. Slabije performanse, ne odgovara nam da glava prelazi velik put. Performanse mjerimo po broju pređenih cilindara.



Slika 124 Algoritam FCFS

Algoritam SSTF.

Shortest Seek Time First - od pristiglih zahtjeva za cilindrima bira se onaj cilindar **najbliži trenutnoj poziciji glave.** Bolje performanse od FCFS. Problem je *izgladnjavanje* za udaljene cilindre.

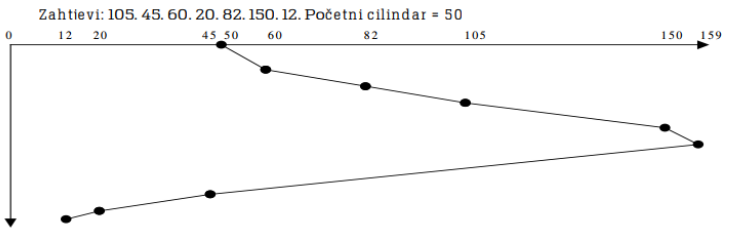


Slika 125 Algoritam SSTF

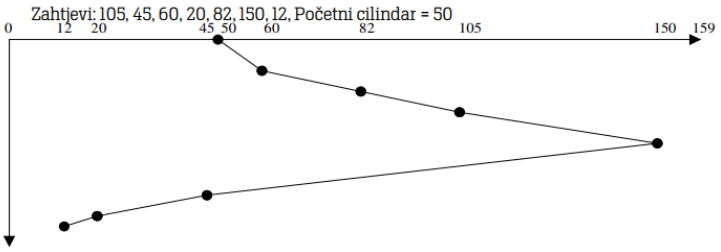
Algoritmi SCAN, LOOK i S-LOOK.

Princip rada sličan kako se liftom "opslužuju" spratovi u zgradama - ***elevator algoritam***.

Nalazimo se na nekom početnom cilindru i idemo u jednom pravcu (desno ili lijevo, tu informaciju dobijemo). SCAN algoritam ide **do kraja** i usput opslužuje zahtjeve, dok LOOK ne ide do kraja već **do zadnjeg zahtjeva u tom smjeru.** Kada nemaju dalje onda se oba algoritma krenu vraćati nazad i opslužuju zahtjeve koji su preostali. Pri povratku, oba se zaustavljaju kada opsluže sve zahtjeve (to je rijetko cilindar 0, obično se zaustave ranije tipa na cilindru 12). Ova taktika rješava problem *izgladnjavanja.*



Slika 126 Elevator ili SCAN algoritam

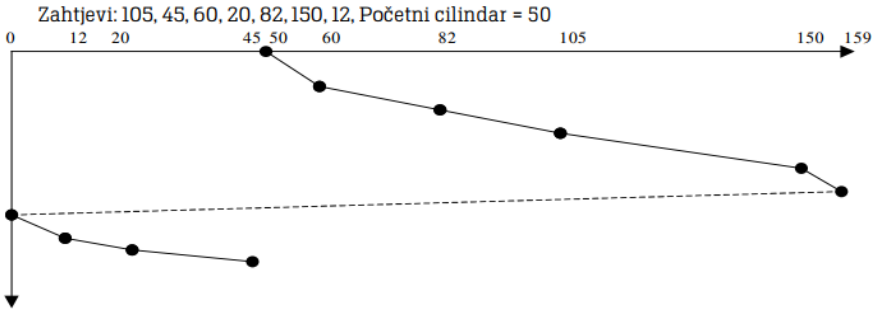


Slika 127 Algoritam LOOK

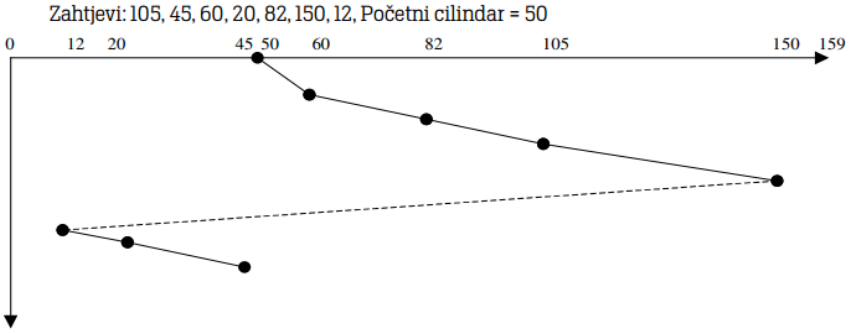
S-LOOK je varijanta gdje se smjer kretanja određuje na osnovu pozicije **najbližeg cilindra** i onda nastavlja u tom kretanju. Obični SCAN i LOOK samo nastavljaju u onom pravcu u kojem su se kretali pri obradi ranijih zahtjeva.

Algoritam C-SCAN i C-LOOK.

Ovo je modifikacija algoritama SCAN i LOOK koja **smanjuje favoriziranje centralnih cilindara**. Zahtjevi za čitanjem cilindara se opslužuju **samo u jednom smjeru.**



Slika 128 Algoritam C-SCAN



Slika 129 Algoritam C-LOOK

C-SCAN radi isto kao SCAN samo što kad dođe do posljednjeg cilindra **brzo vrati glavu na cilindar 0** bez da opslužuje ikakve zahtjeve. Onda nastavlja opsluživati u istom smjeru.

C-SCAN ide od kraja do kraja a C-LOOK se zaustavlja na zadnjem zahtjevu i vraća se unazad ne do 0 već do prvog zahtjeva.

Koji je razlog keširanja diskova?

Razlog je brži pristup blokovima. Ako imamo dovoljno RAM-a možemo u kešu čuvati blokove memorije kojima smo nedavno pristupali, i prilikom obrade zahtjeva provjeriti nalaze li se ti blokovi u RAM kešu. Ukoliko da, **pristupamo tom bloku brže nego da pristupamo disku**. Ukoliko se blok ne nalazi u kešu onda ga učitavamo u keš.

Za pretragu keša često koristimo **hash tabele**. Pošto nerijetko dva bloka imaju istu *hash vrijednost,* svi blokovi s istim *hashom* se smještaju u kratku povezanu listu i onda se ta lista pretražuje (a prvi element te liste smo našli preko *hash vrijednosti*).

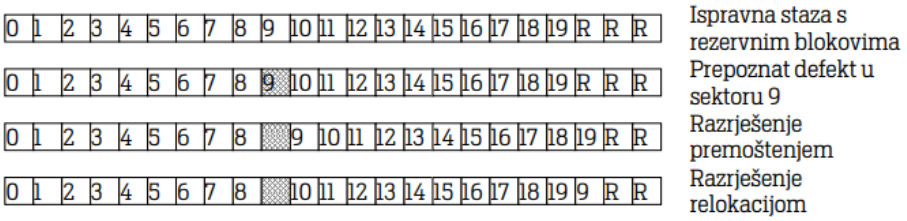
Koji su načini zamjene lošeg sektora rezervnim?

Nekoliko sektora **na kraju svake staze** se ostavi za rezervu. Defektni sektor nastaje kada vrijednost koju smo upisali nije ona koju smo naknadno pročitali (sektor se proglašava neispravnim). Nakon detekcije imamo dvije opcije:

1. razrješenje premošćavanjem: pomjerimo sve sektore u stazi (krećući od defektnog sektora) za jedan sektor naprijed

2. razrješenje relokacijom: taj defekt prebacujemo u jedan rezervni sektor na kraju

Praznine može zamijeniti disk kontroler ili OS. Disk kontroler mijenja preambule sektora na stazi. OS pravi drugačije tablice.



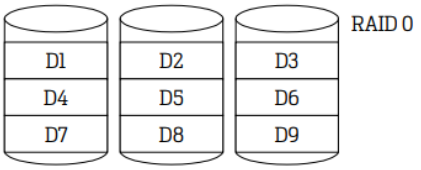
Slika 131 Rezervni sektori

RAID 0.

*Redundant Array of **Independent** Disks,* 7 različitih RAID konfiguracija (0-6)
svaki RAID znači ima više fizičkih diskova, logički je to još uvijek jedan disk.

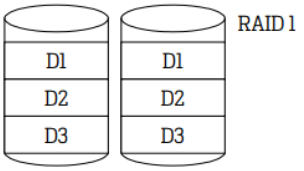
Svi RAID nivoi osim RAID 0 imaju dodatne (redudantne) informacije koje služe kao backup u slučaju pada nekog od diskova. RAID 0 se koristi kada samo **želimo povećati kapacitet i dobiti na brzini**. Npr. umjesto jednog diska od 32 TB kojeg nemamo možemo nabaviti 4 diska od 8 TB i povezati ih u jedan logički disk.

Odsječci se redom postavljaju na svaki od fizičkih diskova. Otkaz bilo kog diska znači gubitak svih podataka. Svaki fizički disk može imati svoj red čekanja na zahtjeve. Ti zahtjevi se mogu obrađivati nezavisno, pa se postižu bolje performanse nego samo s jednim diskom većeg kapaciteta (32 TB).



RAID 1.

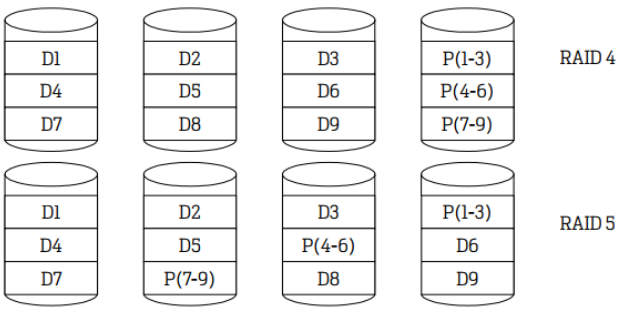
Mirroring - svi diskovi imaju identičan sadržaj. Zahtjev se može proslijediti bilo kojem disku. U sistemu sa n diskova, dopušteno je da ispadne njih n-1. Loša je iskorištenost memorije: npr. za 4 diska stvarna iskorištenost memorije je samo 25% jer onih 75% ode na kopije.



RAID 4 i 5.

RAID 4 raspoređuje podatke po diskovima kao RAID 0, samo što od n diskova odvoji jedan koji će biti paritetni, odnosno u koji će smještati rezultat XOR operacije kao na slici.

RAID 5 ne čuva paritetne informacije na jednom disku nego se koristi neka šema, tipično ona cirkularna (pogledati sliku). Ravnopravno se tretiraju svi fizički diskovi.



Kako su organizirani SSD diskovi?

Magnetni diskovi kod klasičnih hard diskova se zamjenjuju poluprovodničkim, tako da dobijamo **čisto elektronske diskove**. Nemamo mehaničkih dijelova - nemamo ni vibracije, buke, smanjena je potrošnja energije, zagrijavanje itd. Sad su svi oni algoritmi za optimizaciju kretanja glave nepotrebni (SCAN, LOOK i ostali) jer je vrijeme pristupa jednako. Za zapis informacija se koriste **NAND flash i DRAM memorija.** Podaci se pohranjuju u **NAND flash memorijske ćelije** čiji se sadržaj mora prvo izbrisati jer se u te ćelije može direktno pisati samo kad su one prazne.

Kako je organizovana staza na CD-ovima?

CD (Compact Disk) je vrsta optičkog medija, inicijalno dizajniran kao alternativa gramofonskim pločama za čuvanje digitalizirane muzike (kao što je DVD za digitalizirane filmove). Na CD-ovima uopće **nemamo staze nego samo sektore**. Sve je kao jedna staza, **jedan zvrk.** Sektori su veći nego ranije, umjesto 512 bajta sad je jedan sektor 2352 bajta.

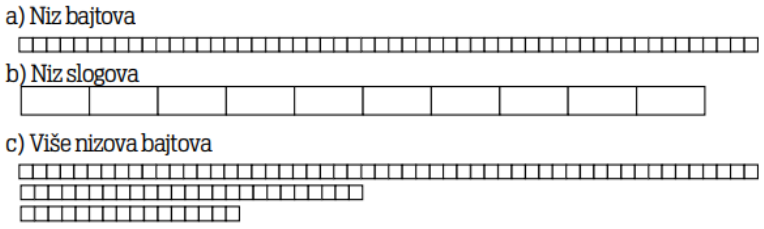
Šta su to datotečni atributi?

Datotečni sistem je skup datoteka, a datoteka je imenovana kolekcija podataka (međusobno vezanih) u najprostijem smislu. Datotečni sistem opisuje datoteku preko datotečnih atributa, sadržaja datoteke i podataka o njoj.

U datotečne atribute spadaju sve dodatne informacije o datoteci koje OS čuva u tablicama:

ime, ekstenzija, verzija, kreator, vlasnik, prava pristupa, tip binarna ili ascii ili neka druga, kriptirana ili ne, kompresovana ili ne, vrijeme kreiranja, vrijeme pristupa, vrijeme izmjene…

Koji su načini realizacije fizičkog sadržaja datoteke?



Slika 144 Fizičke organizacije datoteka

a) su uglavnom datoteke opće namjene, nema neke strukture

b) najmanja jedinica podataka je veća od jednog bajta, svaki slog ima svoju internu strukturu

c) nezavisni nizovi bajtova, NTFS

U čemu je razlika između ASCII i binarnih datoteka?

Ovo su dvije osnovne grupe datoteka.

ASCII datoteke se sastoje od **linija teksta** na kraju kojih se nalazi **bajt CR (*Carriage Return*)**. Prilikom prikaza na ekranu se razumiju kao čitljivi tekst.

Binarne datoteke na ekranu tvore nečitljiv tekst. Nije namijenjen da ga čita čovjek nego aplikacija i OS. Najvažnija grupa binarnih datoteka su **izvršne .exe datoteke**. Pisane su u mašinskom kodu kojeg procesor može direktno izvršavati.

Koji su načini pristupa logičkim slogovima datoteka?

Neke datoteke je moguće čitati samo sekvencijalno (slogovi se čitaju fiksnim redoslijedom od početka), a kod nekih se slogovima može pristupiti u bilo kojem redoslijedom - datoteke s direktnim pristupom. Znači, logičkim slogovima se može pristupati **sekvencijalno ili direktno.**

Sekvencijalan pristup je tipičan za ASCII datoteke - ne možemo čitati na preskoke slogove kako nam dođe. Kod datoteka s direktnim pristupom se uvijek navodi **ključ** kao dodatni podatak i onda se formulom izračuna pozicija unutar datoteke. Sasvim je logično imati direktan pristup recimo u bazama podataka. Multimedijalne datoteke kombinuju sekvencijalni i direktni pristup.

Razlozi za mapiranje datoteke u memoriji.

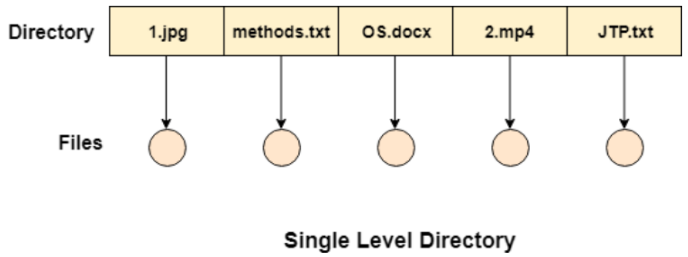
Memorijski mapirane datoteke povezuju dijelove memorije sa dijelovima datoteke. Upis podataka u memorijsku lokaciju se uz pozive *map* i *unmap* reflektira na datoteku koja je mapirana. Razlog za ovo je što nam je **pristup podacima u memoriji jednostavniji nego pristup datotekama**. Pristup podacima datoteke se može poistovijetiti s prostim pisanjem ili čitanjem u memorijsku lokaciju.

Jednonivovski, dvonivovski i hijerarhijski direktorij.

Direktorij, folder i mapa su jedno te isto. Služe za organizaciju datoteke ali zapravo su i sami specijalizirane datoteke.

jednonivovski direktorij

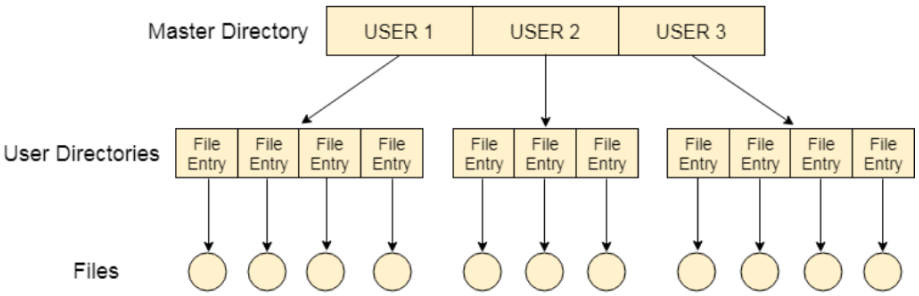
Jedan direktorij koji sadrži sve datoteke u datotečnom sistemu (skup svih datoteka). Bilo od kod ranijih personalnih računara **s jednim korisnikom**. Snalaženje u više hiljada datoteka na istom mjestu je otežano.



Single Level Directory

dvonivovski direktorij

Rješavamo situaciju s više korisnika: možemo svakom dodijeliti njegov **korisnički direktorij** a u korijenskom direktoriju čuvati evidenciju



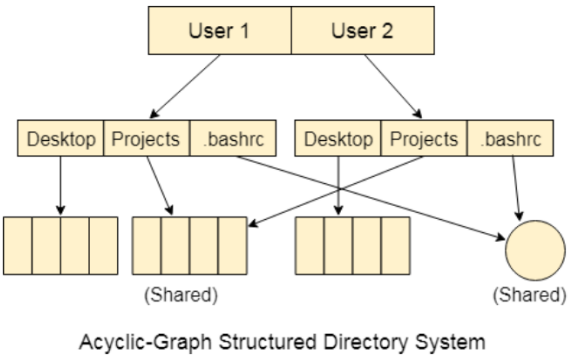
Two Level Directory

hijerarhijski direktorij

Jedan direktorij za jednog korisnika nije dovoljan - sada za svakog korisnika imamo strukturu direktorija **poput stabla**, svako može imati direktorije unutar direktorija unutar direktorija

Šta se postiže organizacijom direktorija u formi acikličnog grafa?

Ova organizacija dopušta da imamo jednu datoteku istovremeno u više direktorija, odnosno dijeljenje datoteka između direktorija (*Shared* na ovoj slici).



Acyclic-Graph Structured Directory System

Unix pozivi creat, open, read, write.

creat - otvori i eventualno kreiraj datoteku (ne mora postojati)

open - otvori datoteku (mora postojati)

read - čitanje datoteke

write - pisanje u datoteku

creat kao parametre ima: ime datoteke, prava pristupa.
open kao parametre ima: ime datoteke, način pristupa (čitanje, pisanje, čitanje i pisanje).
read i write kao parametre imaju: datotečni deskriptor, adresa, broj bajta koji se prebacuje.

Kako je organizovana tipična particija?

Hard disk je znači podijeljen na particije i MBR (Master Boot Record sektor) sadrži tabelu tih particija. Na te particije treba rasporediti podatke koje čuva datotečni sistem. Struktura svake od tih particija se razlikuje u nečemu, ali može se govoriti o nekoj generalnoj strukturi:

- **boot područja:** na početku particije, u njima se nalazi kratki program koji učitava jezgro OS

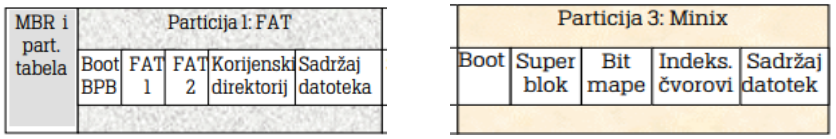
- područje opisa particija: u FAT datotečnom sistemu ovo je dio Boot područja (BPB), na Unix datotečnim sistemima (npr. Minix) ovo područje se zove superblok

- područje sadržaja datoteka: ovdje se nalaze svi podaci, kod većine su i direktoriji tu

- područja za praćenje alociranog prostora: u FAT datotečnim sistemima tu ulogu imaju tabele FAT1 i njena kopija FAT2, Unix sistemi (npr. Minix) koriste indeksne čvorove

- područja za praćenje slobodnog prostora: kod FAT datotečnih sistema ovo područje uopšte ne postoji (jer se ti podaci efikasno dobivaju iz FAT tabela), dok Unix datotečni sistemi koriste bitmape (to su one nule i jedinice)

- područja direktorija: ukoliko sadržaj direktorija nije opisan u *području sadržaja* onda je opisan ovdje, npr. u FAT svi direktoriji su smješteni tu ali **korijenski direktorij nije** - on je opisan ovdje.



Kontinualna alokacija datoteka.

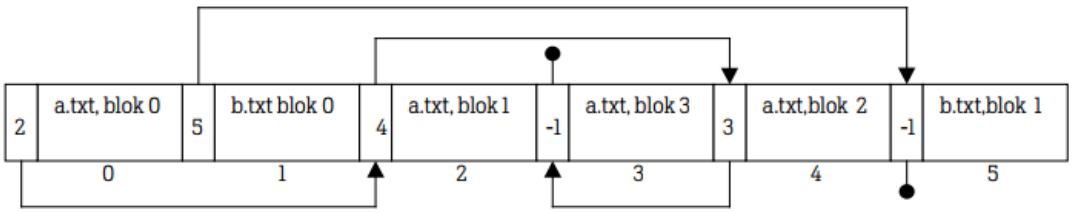
Najjednostavniji način alokacije jeste da se svaki sljedeći logički blok mapira na susjedni fizički blok - **kontinualna alokacija**. Svaka datoteka na disku tada zauzima niz susjednih (fizičkih) blokova. Vodimo evidenciju samo adrese prvog bloka datoteke + ukupan broj blokova u datoteci.

Ovo nije praktično ako podatke treba često mijenjati. Odlično je za CD i DVD uređaje čiji se sadržaj ne mijenja nakon snimanja.

Alokacija datoteka ulančanim listama s pokazivačima u blokovima.

alokacija ulančanim listama == **spregnuta alokacija**

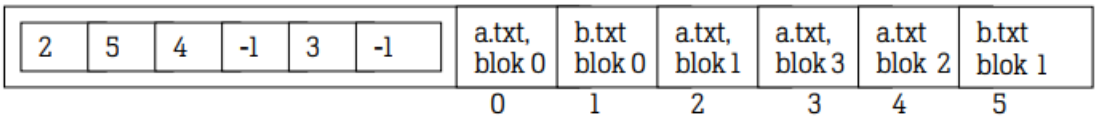
Još uvijek nemamo nikakve potrebe za tablicama lokacija pri alokaciji datoteka. Imamo ulančanu listu blokova datoteka koji na svom početku ili kraju imaju pokazivače na blok od kojeg se datoteka nastavlja. Npr. nakon bloka 0 dolazi blok 2, nakon bloka 2 dolazi blok 4... 0-2-4-3 (krenuli od 0)



Za kućni računar ***Commodore 64 (C64)*** operativni sistem KERNAL je koristio ovakvu alokaciju.

Alokacija datoteka ulančanim listama sa pokazivačima u tabeli.

Mana kod prethodnog pristupa je taj što je vrlo teško pristupiti nekom proizvoljnom bloku - morali bismo ići redom čitati i blokove prije dok ne dođemo do njeg (tipičan slučaj kod liste).



Slika 161 Alokacija ulančanom listom sa pokazivačem lanca u tabeli

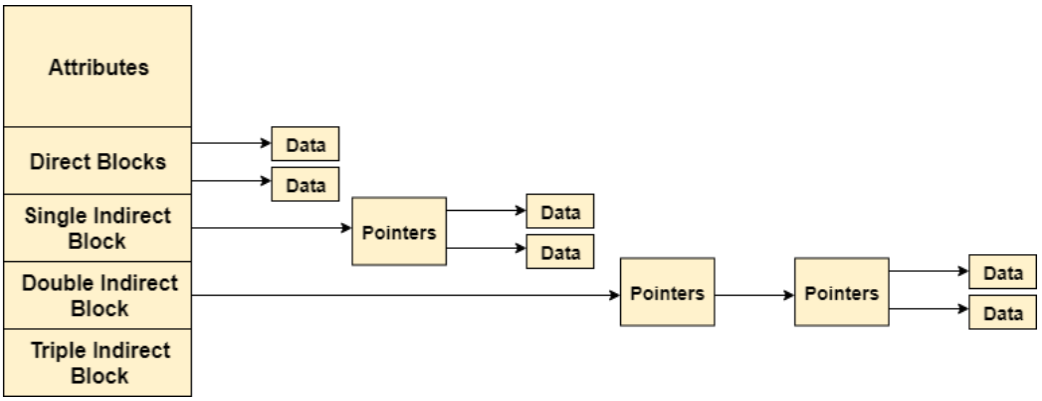
Rješenje je da pokazivače maknemo odatle i stavimo ih u zasebnu tabelu - **FAT tabelu (*File allocation table*).** Cijela tabela mora biti u memoriji cijelo vrijeme ako želimo postići direktni pristup, ali u praksi se ipak (zbog prevelike veličine ove tabele) u RAM učitavaju samo dijelovi tabele.

Ovo je standardni datotečni sistem na USB medijima, floppy diskovima, podržan je i u svim verzijama Windowsa.

Datotečni sistem s indeksnim čvorovima.

Pozicije blokova datoteka čuvamo u pokazivačima koji se nalaze u indeksnom čvoru svake datoteke, kako bi imali brži pristup. Znači, jedan i-node za jednu datoteku i svaki sadrži **pokazivače na blokove svoje datoteke.** Ovo je indeksna alokacija na jednom nivou.

Super bi bilo da svaki indeksni čvor ima tačno onoliko pokazivača koliko njegova datoteka ima blokova. S premalo pokazivača nije moguće zapisati velike datoteke, a sa previše će indeksni čvorovi nekad biti veći od sadržaja svoje datoteke, što je neracionalno.



Kod indeksne alokacije s više nivoa uvodimo i **indirektne pokazivače** koji mogu pokazivati na blokove pokazivača koji pokazuju na blokove datoteka. Ukoliko nastavimo dalje stvaramo sve više slojeva… Ovakva je situacija kod svih OS iz porodice Unix sistema.

Ekstentna alokacija blokova na disku.

Kontinualna alokacija nam u nekim slučajevima više godi od indeksne jer je kod velikih datoteka **direktni pristup brži ukoliko imamo kontinualnu alokaciju.** Sada pravimo kombinaciju: zadržavamo indeksne čvorove na jednom nivou + njegovi pokazivači koji sada ne pokazuju na blokove datoteke nego na **kontinualne nizove blokova promjenljive veličine (ekstenti).**

jedna datoteka --> jedan indeksni čvor
indeksni čvor == pokazivači na ekstente (kontinualne nizove blokova)

Svaki ekstent sadrži, kao kod kontinualne alokacije, početni blok + ukupan broj blokova.

Navesti po primjer alokacija kontinualne, spregnute, ekstentne, indeksne na jednom nivou, indeksne na više nivoa.

Kontinualna alokacija se koristi npr. na CD i DVD uređajima (tj. koristi ih njihov operativni sistem). Nakon snimanja se podaci ne mijenjaju, proširivanje datoteke nije moguće.

Spregnuta alokacija s pokazivačima u blokovima: OS KERNAL na Commodore 64 (C64)

Spregnuta alokacija s pokazivačima u tabeli: floppy disk, USB, MSDOS…

Indeksna na jednom nivou: CP/M operativni sistem (Control Program/Monitor ili Control Program for Microcomputers)

Indeksna na više nivoa: svi Unix operativni sistemi (Oracle Solaris, Darwin i drugi)

Ekstentna: NTFS (*New Technology File System*) koji je standardni Microsoftov datotečni sistem

Kako Windows 98 FAT pamti duga imena datoteka?

Ranije smo imali 8 znakova za ime + 3 znaka za ekstenziju. Sada se kreiraju dva elementa. Element prvog tipa, osnovni element, je još uvijek formata 8+3 i predstavlja "skriveno kratko ime". Drugi element je element drugog tipa, sadrži dugo ime datoteke. Kada se kreira datoteka s dugim imenom, tehnikom VFAT (Virtual File Allocation Table) se generira kratko ime 8+3 i to ime se **čuva zajedno s dugim imenom**. Ovo omogućava kompatibilnost s ranijim verzijama. Kada se pristupa datoteci s dugim imenom, pristupa se i elementu dugog imena i elementu kratkog imena.

Koja je uloga atributa u NTFS?

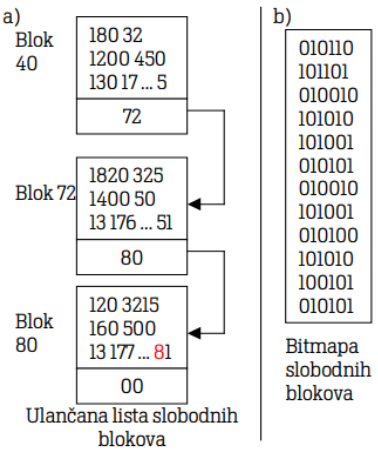
U NTFS datotečnom sistemu (standardni Microsoftov) je apsolutno sve datoteka i svaka datoteka je **skup atributa** čija je uloga da **formiraju datoteku sa svim njezinim osobinama**. Svaki atribut je niz bajta. Iako aplikacije i dalje NTFS datoteke vide kao niz bajta, kao što su bile Unix datoteke, one su ustvari skupovi više nizove bajtova (ovo smo imali ranije na onoj slici).

Koji su načini evidencije slobodnih blokova na disku?

1. ulančane liste blokova - često se koriste upravo slobodni blokovi da čuvaju evidenciju, oni sadrže redne brojeve slobodnih blokova + pokazivač na sljedeći takav blok

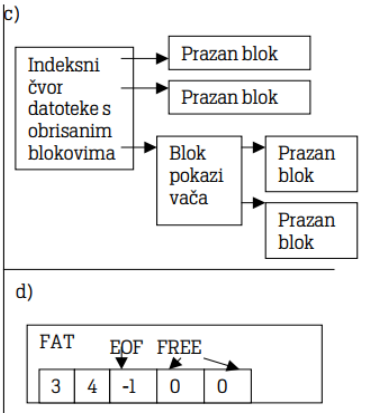
2. bit mapa: za n blokova nam treba mapa sa n bita, 1 je slobodan blok, 0 je zauzet ili obrnuto, zahtjeva vrlo malo prostora

3. preko indeksnih čvorova koji pokazuju na slobodne blokove: može jedan nivo a može i više



4. korištenje podataka iz druge strukture: npr. FAT tablica se može koristiti i za informaciju da li je neki blok slobodan ili neispravan

5. rezervisati slobodni kontinualni prostor za konkretnu datoteku, čak i ako je ona kraća od tog prostora: ukoliko joj zatrebaju novi blokovi prvo se gleda ima li ovdje prostora



Kako se provjerava konzistentnost datotečnog sistema?

Greške datotečnog sistema imaju najgore posljedice od svih (npr. oštećenje podataka koje čuvamo godinama). Najgore što se može desiti radi grešaka u upravljaju procesima ili memorijom je ponovno pokretanje OS i gubitka podataka od nekoliko sati posla (barem nije nekoliko godina, to se ne može baš ispraviti).

Sistem je **konzistentan** kada **sve tabele imaju ispravne vrijednosti.** Može se recimo desiti da se isti slobodan blok dodijeli dvjema različitim datotekama i tako dođe do gubitka podataka.

Konzistentnost datotečnog sistema se provjerava alatima: **chkdsk** ili **ScanDisk** u Windowsu, **fsck** u Unixu (**File System Consistency checK).** Oni se mogu pokrenuti kad god je potrebno.

- pripreme se u memoriji tabela zauzetih i tabela slobodnih blokova
- pregledaju se obje strukture i usporede se
- ako su svi blokovi **ili zauzeti ili slobodni** (ne oba istovremeno) onda imamo **konzistentnost**
- ako je bar jedan blok istovremeno i zauzet i slobodan ili da nije nijedno ili da je zauzet u dvije datoteke, onda je datotečni sistem **nekonzistentan**
- alat treba da sredi potrebne tabele

Navedite neke tehnike povećanja performansi datotečnih sistema.

Keširanje datotečnog sistema.

Čitanje unaprijed - slično straničenju s predviđanjem, blokovi datoteka se ubacuju i prije nego su potrebni (npr. ako datotečni sistem zaključi da se datoteka čita sekvencijalno).

Treba nastojati da što više logičkih blokova bude fizički susjedno analizom bit mape ili liste slobodnih blokova.

Defragmentacijom možemo postići da logičke blokove premjestimo da ih što više bude na susjednim fizičkim lokacijama. Ovaj postupak često dugo traje ali kasnije baš puno poboljšava performanse sistema.

Također, ključno je **izabrati pravi datotečni sistem** u skladu s primjenom. Imali smo gore u jednom od prethodnih pitanja primjere za različite tipove alokacije datoteka.

KLASIFIKACIJE OPERATIVNIH SISTEMA, ARHITEKTURA WINDOWS, LINUX I ANDROID

Šta ulazi u operativni sistem po tradicionalnom i savremenom shvatanju?

Po modernom shvatanju, u OS ulazi sve što je njegov proizvođač uključio, tako bi se moglo reći da je i Internet Explorer sastavni dio Windowsa. Generalno znači sve korisničke aplikacije koje "dođu uz operativni sistem", ugrađene u njeg.

Nešto ispravniji bi bio tradicionalni pogled na OS kao sistemski softver koji upravlja hardverskim i softverskim resursima i pruža zajedničke usluge računarskim programima. Može se čak reći i da je operativni sistem samo skup sistemskih poziva i apstrakcija hardvera, ali ovo bismo inače zvali samo jezgrom. Ako jezgro posmatramo samo kao dio OS, onda možemo posmatrati OS kao cjelinu od 3 komponente: jezgro OS, školjka (tekstualna ili grafička) i skup osnovnih sistemskih alata.

Po tradicionalnom smislu bi onda bilo da u OS ulazi minimum potreban za pokretanje aplikativnih programa, dok su u modernom smislu oni dio OS.

Opišite batch sisteme.

Batch sistemi su najranija vrsta OS iz 1960-ih i 1970-ih (sistemi s grupnom obradom). Davno su iščezli iz upotrebe. Korisnik pripremi poslove (jobove) za obradu u vidu kartica i to prvo snimi na traku (preko čitača kartica), to postavlja na sistem i pokrene izvršavanje. Jedan job sadrži program + podatke. Ovaj batch OS redom učitava poslove iz reda čekanja. Kada završi s radom, rezultati se šalju na izlazni uređaj kao što je linijski štampač. U toku izvršavanja korisnik ne komunicira s OS.

Opišite multiprogramirane batch sisteme.

Sada imamo situaciju gdje, dok jedan posao tipa čeka za završetak UI operacije, procesor može preći na izvršavanje drugog posla. U običnim batch sistemima se idući posao ne učitava dokle trenutni u potpunosti ne bude gotov (procesor dugo vremena nije radio ništa). Korisnici još uvijek nemaju mogućnost komunikacije s poslom dok se on izvršava.

Opišite time sharing sisteme.

Daljnjim razvojem multiprogramiranih batch sistema nastali su time-sharing sistemi (u dijeljenom vremenu) kao **prvi tip interaktivnih sistema (od ukupno tri)**. Interaktivni sistemi uvode pojam **procesa** umjesto posla. Proces je program u vrijeme izvršavanja.

Time sharing sistemi omogućavaju također više programa istovremeno u memoriji, kao i multiprogramirani batch sistemi, ali razlika je u tome što se **svaki program u memoriji izvršava u malim intervalima (ne proizvoljno dugo kao ranije)**, tako da programi dijele procesorsko vrijeme. Procesor toliko često vrši prebacivanje programa da korisnik to i ne primijeti.

Klasifikacija operativnih sistema prema broju korisnika, procesa i načinu obrade.

Po broju korisnika: jednokorisnički OS i višekorisnički OS.
Po broju procesa: jednoprocesni OS i višeprocesni OS.
Po načinu obrade: batch, interaktivni, hibridni, sistemi u realnom vremenu, distribuirani sistemi.

Kod sistema u realnom vremenu vrijeme odziva aplikacije je ključno. Kod distribuiranih sistema obrada se dijeli između različitih računara. Hibridni su kombinacija batch i interaktivnih sistema. Kod batch sistema obrada se obavlja sekvencijalno bez komunikacije između korisnika i posla.

Klasifikacija operativnih sistema prema strukturi jezgra.

Monolitni sistemi, sistemi s mikrojezgrom, sistemi s hibridnim jezgrom.

“Imagine you have a big toy box with lots of toys. In a monolithic kernel, all the toys are inside the box, and you can easily reach any toy you want to play with. But if something goes wrong with one toy and it breaks, it can make all the other toys stop working too.

Now, in a microkernel, the toys are divided into different boxes. Each box has a special toy that helps the other toys talk to each other. If one toy breaks, it doesn't affect the other toys because they are in separate boxes. This way, even if one toy stops working, you can still play with the other toys.

So, a monolithic kernel is like having all the toys in one box, while a microkernel is like having different boxes for different toys to make sure they don't all stop working if one of them has a problem.”

Šta je to sistemski poziv?

Sistemski poziv je definirana ulazna tačka u jezgru koja obezbjeđuje usluge korisničkim procesima. Skup funkcija koje pozivaju sistemske pozive imaju imena upravo tih sistemskih poziva pa ih često poistovjećujemo. Ispravnije je, doduše, definirati sistemske pozive ne kao funkcije nego kao **zahtjeve računarskog programa upućene jezgru OS**. To je način da program (proces, izvršava se) komunicira s operativnim sistemom i to **preko softverskih prekida.**

Šta je to API poziv?

API pozivi su skup funkcija koje se koriste kao sistemski pozivi, imaju svoje parametre, dobijemo ih u sklopu neke biblioteke, što znači da **ne ovise od operativnog sistema kojeg koristimo**. Kod običnih sistemskih poziva nismo imali ovu kompatibilnost, bili smo vezani za specifični primjerak OS na kojem radimo.

Koje vrste prekida postoje?

Softverski prekid se pokreće kada korisnički program izvrši specijalnu instrukciju. Izvršavanje se nastavlja od memorijske lokacije koja je data kao parametar te instrukcije (broj prekida) i kontrolu nad tim programom preuzima jezgro.

Hardverski prekid je npr. pritisak tastera na tastaturi, nevezani su za proces koji se izvršava.

Izuzeci su prekidi do kojih dolazi ukoliko se recimo pokuša pristupiti nedozvoljenom području memorije, ukoliko je memorijska lokacija nepostojeća, ukoliko se dijeli s nulom, pokuša izvršiti nepostojeća instrukcija itd. Mogu se definisati postupci koji slijede.

Šta je Monolitni kernel?

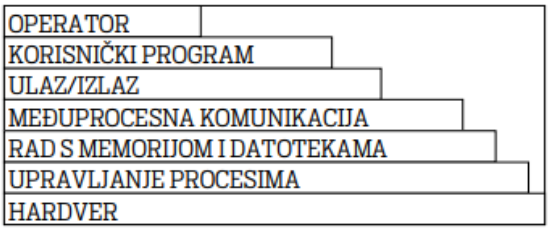
Monolitno jezgro je **jedinstvena struktura** gdje su sve funkcionalnosti OS smještene u **isti adresni prostor**. Ovaj pristup omogućava efikasnu komunikaciju potprograma, ali greška u jednom dijelu jezgra može srušiti cijeli sistem. Razumijevanje rada ovog jezgra je teško jer svi potprogrami ovise jedni od drugih i to je teško pratiti. Nije moguće proširivanje niti dodavanje novih funkcionalnosti.

Šta je Mikrokernel?

Mikrokernel je baziran na **modularnom pristupu** - jezgro se razdijeli na više modula (funkcionalnih jedinica, podsistema). Manje je podložan krahiranju jer greške u jednom podsistemu nemaju utjecaja na sve ostale. Podsistemi znači nisu svi u istom adresnom prostoru, u odvojenim su područjima memorije. Mikrokernel je puno manji od monolitnog kernela, zadržava samo osnovne OS funkcionalnosti, ali je **puno sporiji radi otežane komunikacije (razmjena poruka).**

Šta je Slojeviti operativni sistem?

Slojeviti OS je OS uređen po slojevima postavljenim logički jedan iznad drugog. Ti slojevi se mogu poredati na razne načine, a jedan je predstavljen na slici. Potprogrami koji pripadaju jednom sloju smiju pozivati samo potprograme iz svog sloja ili sloja niže. Mrežne mogućnosti je teško uklopiti u ove slojeve.



ARHITEKTURA THE SISTEMA

Slika 4: Slojeviti sistem

Šta je operativni sistem zasnovan na virtuelnim mašinama?

Virtualne mašine su potpuna kopija hardvera, odnosno savršena simulacija pravog hardvera, te kao takva može izvršiti bilo koji OS koji bi radio nad samim hardverom. **Na istom računaru možemo postići da imamo više OS istovremeno na način da su oni zasnovani na virtuelnim mašinama.**

Ovo je recimo korisno kada nam trebaju posebni serveri za elektronsku poštu, baze podataka i slično i bilo bi dobro da je svaki na zasebnom računaru (ako padne sistem za elektronsku poštu da nam ne padnu ovamo i baze). Umjesto da trošimo novac i prostor za dodatni hardver (računare) možemo na jedan računar postaviti dva ili više virtualizirana operativna sistema.

Kako je organizovana memorija u MSDOS?

MSDOS cijeli memorijski prostor vidi kao područje od 1 MB (to je kraj dostupnog područja, heksadecimalna adresa FFFFF).

Na adresama nižim od 0 se nalaze **interrupt vektori**, nakon njih prostor za **jezgro**, pa prostor za **korisnički program do 640 KB** - kraj konvencionalnog memorijskog područja.

Nakon tog je prostor za video memoriju, ROM video kartice, prostor za proširenje RAM-a i **na vrhu memorije je ROM BIOS** - kraj dostupnog područja, 1 MB.

Adresa	Područje
00000	Interrupt vektori
00400	ROM BIOS područje
00500	DOS parametarsko područje
00700	IO.SYS (u PC DOS IBMBIO.COM)
Promjenjivo	MSDOS.SYS (u PC DOS IBMDOS.COM)
Promjenjivo	Prostor koji je rezervirao CONFIG.SYS za drajvere i buffere
Promjenjivo	Rezidentni dio COMMAND.COM
Promjenjivo	Pokazivač na varijable okruženja
Promjenjivo	Blok varijabli okruženja rezidentnog programa
Promjenjivo	Rezidentni program
Promjenjivo	Blok varijabli okruženja trenutnog programa
Promjenjivo	Trenutni program
Promjenjivo	Izbrisivi dio COMMAND.COM
8FFFF	Kraj konvencionalnog memorijskog područja (640 K)
A0000	Video memorija
C0000	ROM video kartica
E0000	Memorijska proširenja EMS (proširena memorija)
F0000	ROM BIOS
FFFFF	Kraj 8086 dostupnog područja
100000	Visoka memorija od DOS 5.0, tu se prebaci MSDOS.SYS
110000	Produžena memorija preko ekstendera VCPI ili DPMI
FFFFFF	Krajnja adresa dostupna preko ekstendera (32 M)

Slika 180 MS DOS, memorijska mapa

Koja je uloga tri najvažnije datoteke u MSDOS?

IO.SYS - sadrži osnovne drajvere periferijskih uređaja, učitava u memoriju MSDOS.SYS

MSDOS.SYS - jezgro, sadrži implementaciju sistemskih poziva kroz softverske zamke između INT 20h i INT 21h

COMMAND.COM - komandni interpreter, od korisnika očekuje unos komande s parametrima

Koje su osnovne razvojne linije Windows operativnih sistema?

1) 16-bitne verzije (MSDOS + grafički interfejs)
2) verzije 9x
3) Windows NT verzije
4) verzije CE

Koji režimi rada su u 16-bitnim Windows-ima i koja im je glavna mana?

Imamo tri režima rada: **realni režim** (Windows je samo GUI za MSDOS, bez ikakve zaštite), **standardni režim** (imamo segmentaciju sa zaštitom), **proširени režim** (koristi se segmentacija sa straničenjem, teoretsko adresiranje do 4 GB a u praksi do 512 MB).

Mana je što se u sva tri režima imamo kooperativno raspoređivanje (to je ono Round Robin sa istiskivanjem preko sistemskih poziva?). Krah jednog programa znači krah cijelog sistema.

Koja je uloga KERNEL, USER i GDI u Windows-ima?

kernel.exe - koordinira UI zahtjeve prema hardveru, upravljanje memorijom i pokretanje programa
user.exe - obrađuje UI zahtjeve, poruke, tastaturu, miš, zvuk, korisnički interfejs
gdi.exe - *Graphic Driver Interface*, odgovoran za crtanje i štampanje

kernel32.dll, gdi32.dll i user32.dll su njihove 32-bitne verzije, datoteke-biblioteke baznog Windows API. To su klijentski DLL-ovi (*Dynamic Link Library*).

Kako Windows 9x obrađuje 16 bitne a kako 32 bitne drajvere?

U Windows 9x spadaju Windows 95, Windows 98, Windows 98SE i Windows ME. Te verzije se odlikuju 32-bitnim GUI s velikom kompatibilnošću sa starijim aplikacijama. Znači, dizajnirani su da obrađuju 32-bitne aplikacije i drajvere, ali također uvode i **VMM (*Virtual Machine Manager*)** koji daje propusnicu i 16-bitnim drajverima.

32-bitni drajveri su, logično, efektivniji u ovom slučaju, imaju direktan pristup sistemskim resursima, dok 16-bitni drajveri zahtjevaju dodatni softverski modul ***Virtual Device Drivers*** koji uspostavlja komunikaciju između 16-bitnih drajvera i 32-bitnog OS i pristup sistemskim resursima.

Šta je to Registry?

Registry je centralizirana konfiguracijska baza za softverske i hardverske postavke.

Nabrojte verzije Windows sa NT jezgrom.

Ima ih puno, cijela tabela je tu od 20-30 Windows verzija. **Windows NT 3.1, 3.5 i 3.51**. Onda **Windows NT 4.0, Windows 2000 i Windows XP**, **Windows Vista, Windows 7, 8, 10.**

Čemu služi HAL.DLL?

Hardware Abstraction Layer je još jedan **sloj između jezgra OS i hardvera**, interfejs prema samom procesoru, spona softvera i hardvera. HAL.DLL je datoteka (biblioteka) u koju se pišu dijelovi koda **specifični za pojedini procesor**. Potrebno je praviti i jednoprocesorske kao i višeprocesorske verzije HAL.DLL biblioteke (debug verziju isto tako).

Koje su dvije osnovne uloge NT kernela?

1) **raspoređivanje niti i procesa**: NT raspoređivač se zove **dispečer** (i nalazi se znači u jezgru), implementira 32 nivoa prioriteta, radi preemptivno tj. s izbacivanjem i koristi kvantume. Za svaki nivo prioriteta postoji po jedan red čekanja.

prioritet 0 - za sistemsku nit
prioriteti 1-15 - za programe
prioriteti 16-31 - pristupaju im samo administratori

2) **upravljanje prekidima**: Prekidi su definirani osobinama procesora, svaki procesor ima svoju tabelu prekidnih vektora (IDT, *Interrupt descriptor table*).

Kako Object Manager u Windows Executive vidi objekte a kako ih vide procesi?

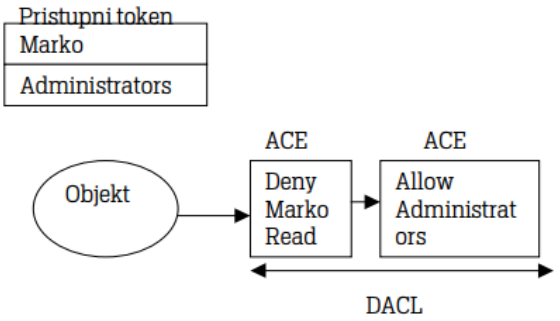
NT Executive upravlja resursima i sastoji se od puno podsistema (modula, funkcionalnih dijelova, jedinica), jedan od njih je Object Manager. Procesi (aplikativni programi) ne vide objekte preko imena nego **preko numeričkih identifikatora - handle.** Ti identifikatori su jedinstveni na nivou aplikacije, ali ne i između njih. Možemo imati dva objekta u dvije različite aplikacije s istim numeričkim identifikatorom. Object manager mora razlikovati ta dva objekta, stoga ih on identificira **preko imena** u NT prostoru za imena.

Na koji način Security Reference Monitor (SRM) određuje koji objekt se može dodijeliti?

Object Manager dakle poziva SRM da provjeri prava pristupa prije nego dopusti aplikaciji da otvori objekt ili ukoliko aplikacija najednom želi recimo pisati. Sigurnosti model koji SRM implementira je baziran na **sigurnosnim identifikatorima (SID) i diskrecionim listama za kontrolu pristupa (DACL).** SID-ovi se dodjeljuju korisnicima i grupama, aplikacijama se dodjeljuju pristupni tokeni sa SID-ovima vlasnika i njegovih grupa.

Svaki objekt kojeg Object Manager evidentira sadrži **pokazivač na DACL** a elementi te liste su **ACE-ovi** (elementi kontrole pristupa, *Access Control Elements*). Ti elementi za svakog korisnika i svaku grupu definišu prava pristupa.

Kada aplikacija zatraži da radi nad objektom, iz njenog pristupnog tokena se utvrde SID-ovi, onda se u objektu locira njegova lista za kontrolu pristupa s elementima kontrole pristupa. U tim elementima se traže SID-ovi koje imamo i zaključuje se kakva prava oni imaju nad objektom.



Koje načine alokacije memorije procesima koristi Virtual Memory Manager u Windows Executive?

Ovo nije onaj VMM koji smo imali ranije kod 16-bitnih drajvera koji su trebali raditi na 32-bitnim operativnim sistemima, to je bio *Virtual **Machine** Manager.*

Virtual Memory Manager transformiše virtualne adrese u fizičke i **kontroliše alokaciju fizičke memorije.** Fizičku memoriju za procese alocira na različite načine:

- **straničenje:** ubacivanje stranice u memoriju se obavlja na zahtjev (nakon greške stranice) uz nešto malo predviđanja
- **memorijski mapirane datoteke:** procesi znači tretiraju datoteke kao dio memorije i kada im se dodijeli njihova memorija isto kao da im se dodijelila ona sama (datoteka)
- **dinamička alokacija memorije** na heap-u

Čemu unutar Windows Executive služi I/O Manager?

Integriše dodatne drajver za uređaje. Drajver prevodi komande koje OS i aplikacije šalju uređaju. IO manager podržava asinhroni paketski prenos preko ***IRP - IO request packets.***

Čemu unutar Windows Executive služi Cache manager?

Održava NT **globalni keš za datoteke.** NT keš je znači datotečno orijentiran (za razliku od Windows 95 koji je blokovno orijentiran).

Čemu unutar Windows Executive služi Process Manager?

Definiše **objekte procesa i threadova.** Surađuje s kernelom: analizira kernelov objekt *Process* i dodaje mu PID identifikator, pristupni token (s onim SID-ovima), adresnu mapu i tabelu *handle*-ova (numerički identifikatori objekata unutar njega?). Sve to se nalazi u strukturi EPROCESS i to je ono što se o procesu vidi iz *Executive-a.*Unutar EPROCESS se nalazi struktura KPROCESS koja sadrži kernelov objekat *Process* i još neke informacije (o nitima, prioritetu, kvantumu itd.).

Čemu unutar Windows Executive služe PnP Manager i Power Manager?

PnP - Plug and Play Manager: od Windows 2000 omogućava prepoznavanje dodatih uređaja i drajvere za njih

Power Manager: od Windows 2000 upravlja promjenama statusa napajanja, s tim da drajveri moraju reagovati na informacije poput nivoa baterije, nivoa aktivnosti sistema, korisničke akcije kao što je pritisak na dugme napajanja, trebaju reagirati na zahtjeve za zatvaranje, spavanje, hibernaciju, postavke Control Panela poput gašenja na 10% baterije itd.

Čemu unutar Windows služi dispečer sistemskih servisa?

Prima sistemske pozive, uzima njihov redni broj, nađe u tabeli adresu gdje se nalazi potprogram i pozove poziv.

Šta su prirodni API i NT.DLL?

Prirodni API su nešto kao funkcije posrednice od aplikacije do Executive-a? Ima ih oko 2500 i smještene su u biblioteci NTDLL.DLL. Aplikacije koriste Win32 API-jevu funkciju recimo CreateProcess; ona poziva funkciju prirodnog API-ja **ntCreateProcess;** ona poziva funkciju iz Process Manager-a u Executive-u **psCreateProcess.** Imena prirodnog API-ja počinju sa nt.

Koji serveri okruženja su postojali ili postoje u Windows NT seriji?

Okruženje je posebni skup API funkcija i implementiran je kao klijent-server. Server okruženja je program koji se pokreće kao ispomoć. (API je način da različiti računarski programi komuniciraju.)

Do Windows 2000 imali smo okruženja Win32, POSIX i OS/2. Windows NT je bio prije 2000 verzije, čak i prije 9x, sad ne znam... Win32 okruženje je najvažnije i najviše se govori o njemu i njegovom serveru. Okruženja OS/2 i POSIC su od Windows XP ukinuta.

okruženje operativnog sistema Win32 je imalo server program CSRSS.EXE
okruženje operativnog sistema POSIX je imalo server program POSIX.EXE
okruženje operativnog sistema OS/2 je imalo server Presentation Manager (PM)

CSRSS.EXE *(Client/Server Runtime Subsystem)* koristi samo prirodni API. POSIX i OS/2 imaju minimalan skup funkcija, dok se Win32 stalno unapređuje i on je vremenom dominirao NT linijom.

Kako se pozivaju funkcije Win32 API na 32-bitnim platformama?

Poziv se obavlja kao poziv običnog potprograma, ne kao sistemski poziv. U 32-bitnim verzijama parametri se smještaju na stack prije poziva, a sama funkcija ih nakon završetka obriše sa stacka.

Zašto je uveden COM i koje su tehnike bazirane na njemu?

Komponenta je skup srodnih funkcija spojenih u zajedničku datoteku (DLL ili EXE). Funkcije te komponente se mogu **pozivati i ugrađivati u druge programe**. COM (*Component Object Model*) je **standard definisan za takve komponente.** To je tehnologija koja omogućava komponentama da komuniciraju jedne s drugima. Tehnike bazirane na COM-u su:

OLE (*Object Linking and Embedding*): olakšava aplikacijama da pozivaju druge aplikacije, npr. kada kreiramo tekstualni dokument i želimo u njeg "prilijepiti" rezultate nekih tablica iz druge aplikacije. Aplikacije koje koriste OLE kreiraju dokumente s podacima iz različitih aplikacija.

OLE automatizacija: pozivanje dijelova aplikacije **programski,** automatski, npr. iz eksternog programa se pozove Microsoft Word (znači bez korisničkog pokretanja Word-a ili diranja miša/tastature), kreira se dokument, obrađuje itd.

Distribuirani COM (DCOM): proširuje COM model tako da omogućava komunikacije komponenata preko mreže, kao da su sve lokalne.

ActiveX kontrole: npr. Adobe Flash Player, Windows Media Player je isto primjer ActiveX kontrole (možemo ugrađivati audio/video sadržaj unutar web stranica ili dekstop aplikacija)

Šta su Windows servisi?

Servisi su pozadinski programi koji nemaju interakciju s korisnikom, nego isključivo s datotekama, mrežom ili Registryjem.

- DNS Client (pretvara domenska imena u IP adrese)
- Event Log (čita i upisuje događaje u event log)
- Plug and Play (automatska detekcija i konfiguracija hardvera)
- Windows Print spooler (upravlja printerima i šalje datoteke na printanje)
- Remote Procedure Call (omogućava poziv potprograma na drugim računarima)
- Routing and Remote Acces (pretvara računar u mrežni ruter)
- Windows Error Reporting (javlja Microsoftu o krahovima aplikacija)
- Windows Firewall (blokira neodobrene konekcije prema računaru i od njega)
- Windows Update (ažurira OS)

Čemu služe NTVDM, WOW i WOW64?

U Windowsu NT, **NTVDM** je specijalni program koji se pokreće kada se pokrene program namijenjen za starije OS poput MSDOS i Windows 3.1. (NT Virtual nešto...) One 3 glavne datoteke su modifikovane: NTIO.SYS, NTDOS.SYS i COMMAND.COM. Ubačene su nelegalne instrukcije koje izazivaju skok u NTVDM od kojeg se nastavlja izvršavanje.

WOW (Windows on Windows) je podsistem koji preusmjerava stare 16-bitne pozive u nove 32-bitne ekvivalentne, stoga podržava otvaranje 16-bitnih Windows programa na 32-bitnoj Windows verziji. Isporučuju se 16-bitne verzije biblioteka i OLE objekata.

Ni jedan ni drugi ne postoje u 64-bitnoj verziji, samo 32-bitnoj, jer 64-bitni procesor nema podršku za 16-bitne programe. Međutim, možemo naći način da pokrenemo 32-bitne programe, i bilo bi dobro jer je njih daleko više nego 64-bitnih. To omogućava **podsistem WOW64**.

Šta je Windows runtime?

WinRT je objektno-orijentiran API implementiran u C++ a pušten od Windows 8 verzije. Motivacija je slab uspjeh Windows Phone OS u vremenu kada raste upotreba mobilnih uređaja. Veličina aplikacije se prilagođava uređaju: bilo mobilnom telefonu ili desktopu ili televizoru.

Šta je to Windows CE?

Windows CE je operativni sistem dijeljenog source (izvornog) koda, gdje proizvođači uređaja licenciraju izvorni kod, i optimiziran je za uređaje sa malom količinom memorije. Na bazi CE kernela su razvijeni i ostali OS za mobilne uređaje: Windows Phone, Mobile i Pocket PC.

Kako su nastali Unix, Minix i Linux?

U 1964. smo imali MULTICS kao operativni sistem opće namjene ali je imao prevelike hardverske zahtjeve. Jedna firma se povukla iz tog projekta i dvojica njezinih inženjera su 1969. kreira UNICS, kasnije **UNIX** operativni sistem, napisan u asemblerskom jeziku, za računare koje su koristili u svom laboratoriju. Kasnije je čitav sistem ponovo napisan u programskom jeziku C (a i njega je kreirao jedan od te dvojice inženjera) i mogao se koristiti i na drugim računarima. Tada su se za taj projekat zainteresirali univerziteti te druge firme, koje prave izmjene i kreiraju vlastite Unix-olike sisteme (Unix-like operating systems). Jedan od njih je **Minix** (kreator Andrew Tanenbaum), namijenjen prije svega u obrazovne svrhe, i iz tog razloga je koristio mikrokernel. U to vrijeme je Linux Torvalds razvio svoje **Linux jezgro**, slično Unix-ovom. Torvalds je znao za Minix, koristio ga, i uz Linux jezgro je bio temelj Linux operativnog sistema.

Šta je to POSIX i od kojih dijelova se sastoji?

Portable Operating System Interface je **standard** koji omogućava međusobnu kompatibilnost OS nastalih iz Unix-a. Jedan dio OS iz Unix porodice je certificiran prema POSIX standardu a drugi dio nije (tu spadaju i Android, Linux, Darwin), ali su ipak dovoljno POSIX kompatibilni za rad većine aplikacija.

Trenutna verzija je podijeljena u 4 dijela:

1. dio: osnovne definicije - termini, varijable, preporučena struktura direktorija, C biblioteke...
2. dio: sistemski interfejsi - opisuje funkcije za sistemske pozive, standardnu C biblioteku...
3. dio: školjka i alati - interne komande školjke i sintaksa alata
4. dio: razlozi - napomene o prethodna tri dijela, opisuje motivaciju za uvođenje ovog standarda

Objasnite pojam kernel modula u Linux sistemima?

Linux jezgro je u principu monolitno (cijelo jezgro je jedna datoteka). Ipak, moguće je kernel razdvojiti u više modula koji su u odvojenim datotekama i posebno se učitavaju, ali su u istom adresnom prostoru kao i ostatak jezgra. Motivacija za to nam je veliki potencijalni broj drajvera za periferijske uređaje. Taj cijeli sistem modula ima tri komponente:

1) sistem za upravljanje modulima: učitava ih u memoriju i izbacuje ih
2) sistem registracije drajvera: dopušta modulima da jave ostatku jezgra da postoji novi drajver
3) mehanizam razrješavanja konflikta: rezerviše hardverske resurse za jedan drajver i štiti ih od drugog drajvera

Šta je Linux distribucija?

Linux je proizvod otvorenog koda (nije komercijalni proizvod kojeg kontroliše i prodaje jedna kompanija). Prednosti su: cijena, sigurnost, pogodnost za istraživanje principa rada operativnih sistema (radi dostupnosti izvornog koda) itd.

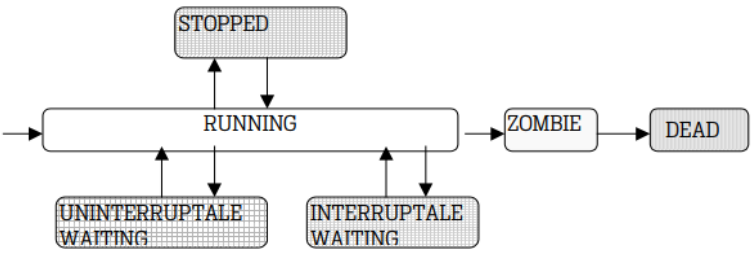
Linux distribuciju možemo definisati kao **OS utemeljen na Linux jezgri, uz instalacijski softver i dodatne alate.** Najpoznatije Linux distribucije su Ubuntu, RedHat, SUSE, Debian, Slackware, Gentoo, Mandriva… Međusobno se razlikuju po formatu datoteka u kojima isporučuju aplikacije.

Šta su deskriptori procesa u Linux-u?

Deskriptor procesa je način da se isti evidentiraju u jezgru: njegovo stanje, adresni prostor, otvorene datoteke, signali, resursi, pokazivači... Kada se proces ne izvršava, njegovo stanje se dijelom čuva u deskriptoru procesa a dijelom u području jezgra (predviđenom za stack). Kada se nastavi izvršavati ove (prethodno zapamćene) vrijednosti se vraćaju u procesor.

U kojim stanjima može biti proces u Linux-u?

running - trenutno se izvršava ili čeka u redu spremnih procesa
interruptible waiting - u stanju čekanja ali ga POSIX signal može reaktivirati
uninterruptible waiting - u stanju čekanja, signali ga ne mogu reaktivirati
stopped - zaustavljen signalom SIGSTOP
traced - u trenutku sesije sa *debuger*-om
zombie - završio je ali se njegov deskriptor još uvijek čuva9
dead - jezgro briše proces



Slika 195: Dijagram stanja procesa u Linux

Koje metode međuprocesne komunikacije postoje u Linux-u?

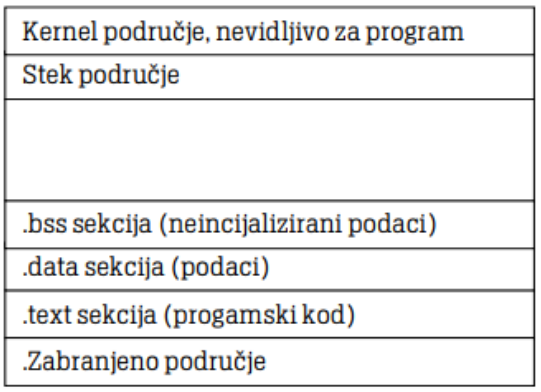
Prvo imamo klasične Unix komunikacijske mehanizme, signale i cijevi. **Signali** su najjednostavniji ali ih jezgro ne koristi, generišu se u slučaju izuzetaka. **Cijevi** su realizirane preko dvije datoteke koje dijele privremeni indeksni čvor: jedna datoteka je za čitanje a druga za pisanje.

Pored toga, imamo **semafore** za međuprocesnu komunikaciju i semafore za sinhronizaciju jezgra. Procesi mogu komunicirati i **redovima poruka** (poziv msgget i onda msgsnd za slanje ili msgrcv za čitanje poruke). Što se tiče **dijeljene memorije**, operativni sistem mapira jedan memorijski segment kojeg dijele procesi i onda oni u tu memoriju upisuju sadržaj ili iz njeg čitaju.

Kako je organizovana memorija procesa u Linux-u?

Na vrhu **kernel područje** (nevidljivo za program), na dnu **zabranjeno područje**, između **su stek** područje, **heap** područje, **.bss** sekcija, **.data** sekcija, **.text** sekcija.

Ovo prazno područje je heap memorija, odatle se uzima memorija kad je dinamički alociramo.



Slika 196 Memorijsko područje procesa

Koja je uloga biblioteka libc i libm u Linux-u?

Aplikacije se vezuju s dinamičkim bibliotekama. Gotovo svi programi u binarnom formatu se vezuju za **libc** - najvažniju biblioteku u Linux sistemima.

libc je biblioteka koja sadrži osnovne funkcije jezika C i sve funkcije koje direktno pozivaju sistemski poziv (*fork, open, read, write, brk, execv, exit...*)

libm sadrži matematičke funkcije iz standardne C biblioteke (*sin, cos, atan, sqrt...*)

Koja je uloga pozadinskih procesa crond, cupsd, sendmail i httpd u Posix sistemima?

Pozadinski procesi, **demoni**, obavljaju odgovarajuću sistemsku funkciju. To su obični procesi ali nemaju interakcije s korisnikom. Jedna grupa se pokreće pri inicijalizaciji sistema i aktivni su do gašenja, a druga grupa se pokreće kada su potrebni i rade samo dok su potrebni.

crond - za periodične zadatke (npr. preuzimanje nove verzije softvera svakog četvrtka)

cupsd - proces koji omogućava realizaciju printer spooler-a, po potrebi pokreće konverzione programe za štampu, može mu se pristupiti preko web interfejsa

sendmail - omogućava slanje elektronske pošte, prihvata poruke od korisnika i šalje ih prema drugim računarima na Internetu

httpd - Hypertext Transfer Protocol **daemon,** proces u korist web servera (računara za čuvanje web stranica, najčešće Apache)

Koja najpopularnija školjka u Linux i šta je Coreutils?

Najčešće se koristi **bash** školjka (**Bourne-Again shell**).

Coreutils je kolekcija najvažnijih komandi iz komandne linije koje se koriste i u Unix i u Linux (Unix-like) sistemima. Uključuje oko 100 komandi za rad s datotekama i direktorijima, prikaz datoteka i neke sistemske alate (chown, cp, dir, ln, ls, mkdir, mv, rm, rmdir...).

Na kojim operativnim sistemima su bazirana jezgra Androida, iOS, Windows Phone i Simbian?

Android je zasnovan na **Linux** jezgru, Apple iOS na **Unix** jezgru. Windows Phone do verzije 7 je bio baziran na **Windows CE** jezgru, poslije **Windows NT** jezgro. Symbian OS se dosta koristio u Nokia sistemima do 2012. godine. Godine 2008 je predstavljao 50% tržišta mobilnih operativnih sistema ali se do 2012. gotovo ugasio. Koristi **mikrojezgro**.

Koja je uloga procesa zygote u Androidu?

Mobitelima je potrebna velika brzina pokretanja procesa, a Java platforma nema tu osobinu. Zygot je pozadinski proces koji ubrzava otvaranje Android aplikacija na način da unaprijed učita resurse i kreira nove procese za svaku aplikaciju. Specifično, odgovoran je za podizanje i inicijalizaciju Dalvik VM do tačke gdje je spreman za pokretanje sistema ili aplikacijskog koda napisanog u Javi.

Šta je Dalvik?

Dalvik je virtualna mašina u Android operativnim sistemima za pokretanje aplikacija višeg nivoa napisanih u Javi. Otvorenog je koda, baziran na Linux kernelu, koristi arhitekturu baziranu na registrima (za razliku od Javine stack-bazirane arhitekture). Koristio se do Android 4.4 KitKat verzije, a od 5.0 aplikacije se i dalje isporučuju u Dalvik formatu ali se ne izvršavaju uz pomoć Dalvik virtualne mašine.

Koje vrste ulaznih tačaka postoje u Android aplikaciji?

Ulazne tačke su komponente ili metode koje služe kao "starting points" za izvršenje aplikacije, odnosno definišu šta se dešava kada korisnik pokrene aplikaciju. To su prve linije koda koje se krenu izvršavati kada pokrenemo aplikaciju. Shodno tome se definišu četiri tipa ulaznih tačaka: **aktivnost, prijemnik, servis i pružalac sadržaja.**

Aktivnost je ulazna tačka za interakciju s korisnikom, **jedan ekran s korisničkim interfejsom**. Npr. aplikacija za email može imati jednu *aktivnost* koja prikazuje listu nove pošte, drugu *aktivnost* za sastavljanje i treću *aktivnost* za čitanje elektronske pošte.

Prijemnik ima ulogu reagovanja na događaje izvana, npr. da prekine reproduciranje muzike kada registrira telefonski poziv, da nam blic ne radi kad baterija padne ispod 5% i slično.

Servis ili usluga je ulazna tačka koja održava rad aplikacije u pozadini. Ne pruža korisnički interfejs. Npr. usluga može reproducirati muziku u pozadini ali korisnik ne surađuje s njom. Nešto kao *aktivnost* ali bez otvorenog prozora. To recimo imamo svaki put kada radimo neku sinhronizaciju ili backup u pozadini.

Pružalac sadržaja upravlja zajedničkim skupom podataka koje aplikacija može spremiti u datotečnom sistemu (u bazi podataka, na webu ili bilo kojoj drugoj stalnoj lokaciji kojoj može pristupiti). Omogućava aplikaciji da dijeli podatke s drugim aplikacijama.

SADRŽAJ