DEO 1: ORGANIZACIJA I ARHITEKTURA RAČUNARA

1. ORGANIZACIJA I ARHITEKTURA RAČUNARA

1.1. Uvod

Prvi potpuno funkcionalan elektronski digitalni računar - **ENIAC** - napravljen je 1946. godine. Računarski sistemi se sastoje od softvera i hardvera. **Računari** su mašine za obradu podataka, koje na osnovu skupa ulaznih generišu skup izlaznih podataka.

Osnovna podela (klasifikacija) softvera je na aplikativni i sistemski. **Aplikativni softver** se projektuje za rešavanje konkretnih problema, npr: plaćanje preko Interneta, rezervacija avionskih karata, upravljanje bankomatima, itd. **Sistemski softver** predstavlja programsko okruženje koje omogućava programerima da brže i jednostavnije kreiraju aplikacije prema zahtevima korisnika, i to su npr: operativni sistemi, programski prevodioci, linkeri i interpreteri.

Arhitektura računara je naučna oblast koja daje odgovor na pitanje: „Kako projektovati delove računarskih sistema koji su vidljivi programeru?", tj. AR prikazuje računarski sistem iz ugla programera. Interfejs je spoljašnji pogled na ono što sistem radi, a implementacija je unutrašnji.

Organizacija računara prikazuje strukturne veze unutar računara koje nisu vidljive od strane programera, npr: interfejsi ka periferijskim uređajima, učestanost takta i tehnologija koja se koristi za memoriju, tok informacija između komponenata, itd.

1.2. Von Neumann-ova arhitektura

Osnovni hardverski elementi računara su **elektronska kola.** Skup operacija i redosled njihovog izvršavanja predstavljaju **program** koji računar treba da izvrši. Računari sa uskladištenim programom se često nazivaju **računarima opšte namene**.

Osnovne komponente von Neumann-ove mašine su: /slika 1.1, str. 6 i slika 1.2, str. 7/

- **memorija** (skladišti programe i podatke);
- **upravljačka jedinica** (uzima jednu po jednu instrukciju iz memorije i svaku interpretira, odnosno odredi koju operaciju treba izvršiti, a zatim pošalje odgovarajući upravljački signal aritmetičko-logičkoj jedinici);
- **aritmetičko-logička jedinica** (izvršava operaciju specificiranu instrukcijom i rezultat se upisuje u memoriju);
- **ulazno-izlazne (U/I) jedinice** (obezbeđuju razmenu poruka i podataka između računara i okruženja).

Još jedna bitna komponenta je **akumulator** - pripada skupu uređaja koji se nazivaju registri; on privremeno skladišti vrednost jednog operanda, a po izvršenju operacije skladišti njen rezultat.

U savremenim računarima upravljačka, aritmetičko-logička jedinica i registri se obično nalaze na jednom čipu i nazivaju se **centralni procesor**.

Reč je uređen skup bitova u kome se skladište podaci ili instrukcije.

Upravljačka jedinica u trenutku izvršenja jedne instrukcije mora da čuva adresu sledeće instrukcije u programu, a tu funkciju obavlja registar posebne namene kojie se naziva **brojač instrukcija (BN)**. Proces izvršavanja instrukcija se odvija u dva koraka i naziva se **ciklus instrukcije**. Prvi korak je uzimanje instrukcije (fetch cycle), a drugi izvršavanje instrukcije (execute cycle).

Harvard arhitektura predstavlja varijaciju von Neumann-ove arhitekture - ima posebnu memoriju za instrukcije i posebnu memoriju za podatke. Sadrži i posebne putanje za instrukcije i podatke nezavisno od toga da li se koriste posebne memorije. /slika 1.3, str. 8/

***Karakteristike von Neumann-ove arhitekture*:**

1. sekvencijalno izvršavanje instrukcija, tj. odsustvo bilo kakvog paralelizma ili preklapanja instrukcija;
2. postojanje samo jedne putanje između operativne memorije i upravljačke jedinice centralnog procesora. To direktno utiče na performanse sistema i naziva se **von Neumann-ovo usko grlo**.

1.3. Višenivoska mašina /slika 1.4, str. 9/

Skup tipova podataka, operacija i osobina zajednički se zove **arhitektura datog nivoa apstrakcije**.

Savremeni računari kao višenivoske mašine: /slika 1.5, str. 10/

- **nivo hardvera** (fizički delovi računara);
- **mikroprogramski nivo** (direktno upravlja fizičkim uređajima i obezbeđuje interfejs prema konvencionalnom mašinskom nivou);
- **konvencionalni mašinski nivo** (primenjuju se instrukcije koje procesor razume i koje izvršava pomoću svojih mikro programa);
- **nivo operativnog sistema** (sakriva detalje nižih nivoa i obezbeđuje jednostavne instrukcije za pristup hardveru);
- **nivo asemblera**;
- **nivo viših programskih jezika**.

Kompajler (compiler) je program za prevođenje.

Princip ekvivalencije hardvera i softvera - sve što se može uraditi pomoću softvera može se takođe uraditi i pomoću hardvera i sve što se može uraditi pomoću hardvera, može se uraditi i pomoću softvera. Hardverske implementacije su skoro uvek brže od softverskih. Izbor između hardverskog i softverskog rešenja zavisi od faktora kao što su cena, brzina, pouzdanost i frekvencija očekivanih promena.

1.4. Flynn-ova klasifikacija računara

Michael Flynn je 1966. godine polazeći od toga da računari koriste podatke i instrukcije klasifikovao sve računarske sisteme u četiri grupe. Osnova klasifikacije su tok instrukcija i tok podataka. Flynn je definisao **tok instrukcija** kao niz instrukcija koje izvršava data mašina i **tok podataka** kao niz podataka koje koristi tok instrukcija. Četiri grupe su:

1. **SISD** (Single Instuction stream, Single Data stream), /slika 1.6, str. 13/
2. **SIMD** (Single Instruction stream, Multiple Data streams), /slika 1.7. str. 14/
3. **MIMD** (Multiple Instruction streams, Multiple Data streams), /slika 1.9, str. 15/
4. **MISD** (Multiple Instruction streams, Single Data stream). /slika 1.11, str. 17/

1.5. Istorija razvoja računarskih sistema

Najznačajnije implementacione tehnologije u dosadašnjem razvoju računarskih sistema:

- tehnologija integrisanih kola,
- tehnologija magnetnih diskova,
- tehnologija DRAM (Dynamic Random Access Memory) memorija,
- tehnologija mreža.

Jedinica mere ***MIPS*** (Millions of Instructions Per Second) označava izvršavanje milion mašinskih instrukcija u sekundi. Jedinica mere ***MFLOPS*** - megaflops (Millions of FLoating point Operations Per Second) označava milion operacija u pokretnom zarezu izvršenih u jednoj sekundi. Danas se koriste i jedinice **GFLOPS** - gigaflops i **TFLOPS** - teraflops.

Brzina rada računara se može meriti i ***vremenom koje je potrebno za izvršavanje jednog ciklusa instrukcije*** - kod najsporijih računara to vreme je reda veličine milisekunde (10^{-3}s), a kod najbržih je reda veličine pikosekunde (10^{-12}s).

Brzina takta poslednje generacije procesora se meri u ***GHz***, tj. u milijardama ciklusa u sekundi.

Kompatibilnost „naniže" podrazumeva da procesor može da izvršava prevedeni kod za bilo koji prethodni model procesora, tj. viši model u istoj seriji procesora može da izvršava sve sistemske i korisničke programe koji se mogu izvršavati na prethodnim modelima.

Murov zakon (Moor's Law) - činjenica da se performanse računarskih sistema poboljšavaju geometrijski, a ne linearno. *Objašnjenje:* Poboljšanje performansi iz jedne godine u drugu ima relativno konstantan odnos, a ne konstantnu apsolutnu vrednost.

2. STRUKTURA RAČUNARA OPŠTE NAMENE

2.1. Struktura računara

Procesor je sa ostalim komponentama povezan pomoću sistemske magistrale. ***Sistemska magistrala*** je skup paralelnih žica za prenos podataka, adresa i upravljačkih signala, tj. sastoji se od:

- **magistrale podataka** (preko nje se prenose podaci);
- **magistrale upravljačkih signala** (koristi se za upravljanje funkcijama magistrale i omogućava korisnicima magistrale da signaliziraju kada su podaci raspoloživi);
- **adresne magistrale** (vrši se adresiranje gde se prenose podaci i odakle).

/slika 2.1, str. 28/

Širina magistrale se pokazuje u bitima - što je veća širina, više se podataka može preneti. **Brzina magistrale** pokazuje koliko bita se može preneti kroz svaku žicu u sekundi. **Širina propusnog opsega** pokazuje koliko se ukupno podataka može preneti preko magistrale u datoj jedinici vremena. Magistrale mogu biti **eksterne** (povezuju procesor sa primarnom memorijom i U/I uređajima) i **interne** (unutar procesora prenose informacije). Kod računarskih sistema postoji ***memorijska magistrala*** koja se koristi za komunikaciju između procesora i primarne memorije i ***U/I magistrala*** koja se koristi za komunikaciju između procesora i U/I uređaja.

Komponente računara /slika 2.2, str. 29/

2.2. Centralni procesor

Centralni procesor (CPU - Central Processing Unit) je obavezna komponenta svakog računarskog sistema. On obavlja aritmetičke, logičke i druge operacije nad podacima koji se nalaze u primarnoj memoriji i upravlja radom računarskog sistema.

Komunikacija između procesora i glavne memorije /slika 2.3, str. 30/

2.2.1. Organizacija centralnog procesora

Centralni procesor se sastoji od sledećih ***osnovnih komponenata***:

1. **upravljačke jedinice** (odgovorna je za pristupanje instrukcijama u glavnoj memoriji, njihovo tumačenje i generisanje signala za njihovo izvršavanje. Takođe, upravlja izvršenjem programa u celini);
2. **aritmetičko-logičke jedinice - ALU** (na osnovu signala upravljačke jedinice izvršava potrebne operacije nad operandima instrukcije i generiše rezultat);
3. **registara.**

Zadatak ***brojača naredbi (BN)*** je da u svakom trenutku izvršenja programa sadrži adresu sledeće instrukcije programa. U ***registru naredbi (RN)*** se vrši dekodiranje instrukcije i generišu se odgovarajući upravljački signali.

Čitanje instrukcije iz memorije /slika 2.5, str. 32/ - brojač naredbi sadrži adresu sledeće instrukcije koju treba učitati iz memorije. Ta adresa se upisuje u MAR (memorijski adresni registar), a zatim na adresnu magistralu. Nakon toga, upravljačka jedinica zahteva čitanje iz memorije. Pročitana vrednost se upisuje na magistralu podataka, kopira u PRM (privatni registar memorije) i prenosi u registar naredbi RN. Brojač naredbi se povećava za jedan (tj. za dužinu instrukcije) i na taj način priprema za čitanje sledeće instrukcije.

2.2.2. CISC i RISC arhitektura

Arhitektura procesora je određena sa:
- skupom registara,
- skupom tipova podataka koje procesor podržava,
- formatom instrukcija,
- skupom instrukcija,
- načinima adresiranja,
- mehanizmom prekida.

CISC mašine, tj. mašine sa složenim skupom instrukcija imaju veći broj formata instrukcija, instrukcije različitih dužina, veći izbor načina adresiranja i veću složenost hardvera.

Osnovne **prednosti RISC** arhitekture: 1. postoji više prostora za registre procesora, 2. više prostora za keš memoriju, 3. korišćenje pipelining-a je jednostavnije. Glavni **nedostaci RISC** arhitekture: 1. programi imaju veću veličinu, 2. potrebno je više memorije za izvršavanje programa.
Poređenje CISC i RISC arhitekture /tabela 2.1, str. 35/

2.3. Memorija

U memoriju se smeštaju instrukcije programa i podaci nad kojima se izvršavaju instrukcije. ***Karakteristike memorije***, pre svega brzina pristupa i kapacitet igraju bitnu ulogu u brzini izvršavanja programa i efikasnosti računarskog sistema u celini.

2.3.1. Primarna (glavna) memorija

Sastoji se od niza memorijskih ćelija, tj. **memorijskih lokacija**. Svaka memorijska lokacija ima svoju adresu, pri čemu je adresa njen redni broj u memoriji. Nad memorijskim lokacijama se obavljaju ***dve operacije***: 1. operacija čitanja i 2. operacija pisanja (upisivanja u lokaciju).

2.3.2. Skrivena memorija

U poređenju sa operativnom, skrivena memorija je znatno brža, ali usled visoke cene izrade znatno je manjeg kapaciteta. Postavlja se između centralnog procesora i operativne memorije, što omogućava procesoru da joj direktno pristupi.

2.3.3. Asocijativna memorija

To je specijalna, mala keš memorija sa brzim pretraživanjem koja se koristi za memorisanje tabela stranica (page tables). Kada se tabele stranica nalaze u operativnoj memoriji to je sporo rešenje. Korišćenje asocijativne memorije može da ubrza pristup, tj. da smanji efektivno vreme pristupa.

2.4. Izvršavanje instrukcija i programa

Za svaki procesor je karakterističan ***„uzmi-analiziraj-izvrši" ciklus.*** Najčešće se realizuje u sledećih šest koraka (faza):
1. procesor iz primarne memorije uzima instrukciju koju treba da izvrši na osnovu adrese u BN,
2. dekodira se operacioni kod instrukcije,
3. procesor uzima iz memorije operand ili operande ako je potrebno,
4. izvršava se instrukcija,
5. rezultat se upisuje ili u registar ili u operativnu memoriju,
6. prelazak na korak 1.

2.5. Pomoćne memorije

2.5.1. Magnetni disk

Za ***upisivanje*** i čitanje podataka koriste se upisno-čitajuće glave diska (disk heads). Svaka površina ploče ima svoju upisno-čitajuću glavu, čime se omogućava da pristup podacima na različitim površinama može biti paralelan. Podaci na magnetnom disku se zapisuju u obliku koncentričnih krugova koji se zovu **staze** (tracks). Širina staze je jednaka širini upisno-čitajuće glave.
Struktura magnetnog diska /slika 2.10, str. 45/
Organizacija diska na staze i sektore na jednoj površini magnetne ploče /slika 2.11, str. 45/

Vreme pristupa disku ima tri komponente:
1. vreme traženja (vreme potrebno za pokretanje glave diska do adresiranog cilindra);
2. vreme usled rotacije diska (dodatno vreme potrebno da disk rotira do adresiranog sektora);
3. vreme prenosa podataka sa diska u operativnu memoriju (direktno zavisi od količine podataka koja se prenosi).

2.5.2. RAID tehnologija

RAID (Redundant Array of Independent Disks) tehnologija je razvijena u cilju poboljšanja performansi i pouzdanosti memorisanja podataka na magnetnim diskovima. Podaci se, umesto na samo jedan, upisuju na više magnetnih diskova, pa se tako može povećati brzina, pouzdanost i kapacitet memorisanja podataka.

RAID 0 je najjednostavniji nivo RAID tehnologije. Podaci se distribuiraju na sve diskove sekvencijalno (u delovima). Može se smanjiti vreme pristupa podacima i ostvaruje se najniža moguća cena primene RAID tehnologije. Međutim, ne koriste se tehnike redundantnosti, pa ovaj nivo ne obezbeđuje zaštitu od otkaza pojedinih diskova, a otkazom bilo kog diska u nizu diskova gube se svi podaci.

RAID 1 - za svaki disk postoji identična kopija diska (mirroring). Podaci se mogu čitati ili sa originalnog diska ili sa kopije. Upisivanje podataka se mora uraditi na oba diska. Otkazom bilo kog diska podaci ostaju sačuvani na kopiji i oporavak sistema je jednostavan. Za implementaciju RAID 1 nivoa potreban je dvostruki broj magnetnih diskova.

RAID 2 - svi diskovi se koriste za svaki pristup. Postoje diskovi za podatke i diskovi za korekciju grešaka. Kodovi za korekciju grešaka izračunati na osnovu podataka na svim diskovima za podatke se upisjuju na dodatne diskove. Svi diskovi su međusobno sinhronizovani za bilo koju operaciju upisa ili čitanja podataka. Greške na nivou bita se detektuju i koriguju trenutno. Glavni razlozi zašto se RAID 2 danas ne koristi su složen i skup RAID kontroler i minimalni broj diskova koji su potrebni.

RAID 3 nivo koristi sekvencijalnu distribuciju podataka na nivou bajta čime se ostvaruje poboljšanje performansi. Takođe postoji jedan redundantan disk - za kontrolu parnosti, čime se povećava otpornost sistema na moguće otkaze. Potrebna su najmanje 3 magnetna diska: 2 za podatke i 1 za kontrolu parnosti. Iako su performanse dobre dodatni disk za parnost malo usporava upis podataka. Za slučaj većeg broja aplikacija koje se izvršavaju istovremeno disk parnosti može postati „usko grlo" sistema.

RAID 4 je veoma sličan RAID 3 nivou. Jedina razlika je da se podaci sekvencijalno distribuiraju na nivou bloka, a ne na nivou bajta. To je prednost, jer veličina bloka može da se podešava prema potrebama aplikacije. Kod RAID 4 nivoa biti parnosti se skladište na redundantnom disku. Jedan logički upis podataka zahteva dva fizička upisa. Disk parnosti može postati „usko grlo" sistema pri istovremenom radu više aplikacija.

RAID 5 nivo sekvencijalno distribuira podatke na nivou bloka i informacija o parnosti se upisuje na sve diskove na kojima su podaci (round-robin šema), tj. nije potreban poseban disk za parnost. Nedostatak je da je oporavak sistema nešto složeniji zbog distribuirane prirode informacija o parnosti. Veličina bloka podataka se može menjati prema potrebama aplikacije. RAID 5 je jedan od najpopularnijih nivoa RAID tehnologije koji se danas koriste, jer se za mnoge aplikacije ostvaruje najbolja kombinacija performansi, redundantnosti i efikasnosti memorije.

RAID 6 - povećana je složenost obrade za kontrolu parnosti, jer se dve šeme za kontrolu parnosti koriste istovremeno. RAID 6 nivo omogućava rekonstrukciju podataka i kada dva diska istovremeno otkažu.

2.6. U/I procesori i uređaji

Postoje **dva osnovna tipa U/I uređaja**: 1. multipleksni (vrše razmenu podataka sa nekoliko periferijskih jedinica koje rade paralelno) i 2. selektorski (vrše razmenu podataka samo sa jednom periferijskom jedinicom).

Tri osnovna načina za komunikaciju centralnog procesora sa U/I uređajima:

1. Procesor može periodično da proverava status uređaja (polling). Na taj način se ostvaruje ciklus čekanja (busy-wait cycle) U/I aktivnosti na datom uređaju.

2. Korišćenje sistema prekida - signal prekida prihvata funkcija za upravljanje prekidima. Prekidi se mogu maskirati kada je potrebno da se prekid odloži ili ignoriše.

3. Korišćenje direktnog pristupa memoriji (DMA) - nije potreban programiran U/I, ali je neophodan DMA kontroler. Podaci se direktno prenose između U/I uređaja i memorije, bez potrebe da procesor učestvuje u prenosu. Na kraju prenosa podataka DMA kontroler prekidom obaveštava procesor da je prenos podataka završen.

2.7. Sistem prekida

Prekidi se koriste za komunikaciju periferijskih uređaja sa centralnim procesorom. Računarski sistemi bez sistema prekida ne mogu da funkcionišu efikasno. Procesor je mnogo brži od rada periferijskih uređaja, tako da procesor bez sistema prekida mora da čeka završavanje U/I operacije, što je veoma neefikasno.

Postoje ***tri vrste prekida***:

1. softverski, **2.** hardverski, **3.** izuzeci.

2.7.1. Softverski prekidi

Nastaju na osnovu programskih instrukcija. Omogućavaju pristup „ugrađenom" kodu unutar BIOS-a, operativnog sistema ili periferijskih uređaja. To su različiti funkcijski pozivi koji se iniciraju korišćenjem različitih instrukcija za pozive i različite konvencije. Softverski prekidi se iniciraju instrukcijom INT.

2.7.2. Hardverski prekidi

Nastaju na osnovu signala prekida koje generišu U/I uređaju. Predstavljaju obaveštenje procesoru da je nastala neka hardverska situacija u kojoj je potrebna dodela procesora. Signale prekida koriste U/I uređaji kako bi obavestili procesor da su završili U/I operaciju. Nastaju asinhrono, tj. u bilo koje vreme izvršavanja nekog drugog

programa, pa su i pozivi za izvršavanje prekidnih programa za hardverske prekide asinhroni. Hardverski prekidi se dele u **dve grupe**: 1. maskirajući i 2. nemaskirajući. ***Maskirajućim prekidima*** se može dozvoliti ili zabraniti rad. Za maskirajuće prekide dozvola rada se vrši softverski postavljanjem odgovarajućeg bita u statusnoj reči.

2.7.3. Izuzeci

Izuzeci su sinhroni događaji koji se pojavljuju kao rezultat nastanka neke greške u toku izvršavanja instrukcija. ***Sistemski pozivi*** omogućavaju korisničkim procesima da zahtevaju servise operativnog sistema.

2.7.4. Prioritet prekida

U višekorisničkom okruženju se može desiti da više programa ili više uređaja generiše ***više prekida u isto vreme***. Takva situacija se može rešiti ili zabranom više prekida u isto vreme ili uvođenjem prioriteta prekida. **Zabrana više prekida** u isto vreme se može realizovati tako što procesor ne obrađuje novi prekid dok ne završi sa obradom tekućeg. Kod **sistema prekida sa prioritetom** niži vektori prekida imaju viši prioritet i niži prioritet ne može da prekive viši, ali viši prioritet može da prekine niži.

Vreme reakcije predstavlja vreme koje protekne od trenutka pojave signala prekida do trenutka prelaska na prekidni program. **Dubina prekida** je maksimalan broj programa koji mogu prekidati jedan drugog. **Zasićenje sistema prekida** nastaje ako je vreme reakcije sistema prekida veliko tako da u slučaju pojave novog signala prekida od istog izvora prekida dolazi do gubitka ovog novog signala prekida.

2.8. Preklapanje i paralelno izvršavanje instrukcija

2.8.1. Paralelizam

Paralelizam je izvršavanje dve ili više aktivnosti u isto vreme. Može se ostvariti: 1. na nivou instrukcije i 2. na nivou procesora.

2.8.1.1. Paralelizam na nivou instrukcije

Cilj je da se na datom računarskom sistemu izvrši što više instrukcija u sekundi. ***Jedan*** pristup je uzimanje instrukcija unapred. ***Drugi*** je korišćenje tehnike protočne obrade (pipeline processing). ***Treći*** pristup je korišćenje superskalarnih arhitektura.

Koncept instrukcijskog toka ili tehnika protočne obrade (***pipelining-a***) deli izvršavanje instrukcije u više delova, pri čemu za izvršavanje svakog tog dela postoji namenski deo hardvera i što je najbitnije svi ti delovi se mogu izvršavati paralelno.

Problemi koji mogu da nastanu kod pipelining-a su:

1. **strukturni hazardi** (prouzrokovani su nedostatkom hardvera);
2. **kontrolni hazardi** (prouzrokovani su instrukcijama grananja ili strategijama pomerenog grananja);
3. **hazardi nad podacima** (prouzrokovani su zavisnošću podataka od rezultata izvršavanja prethodne instrukcije)

Ovi hazardi se mogu izbeći restrukturiranjem koda tako da se eliminišu zavisnosti ili nemogućnosti izvršavanja takvih instrukcija sve dok se ne završi prethodna instrukcija.

Uvođenjem više pipelining-a istovremeno nastaju ***superskalarne arhitekture računara***. One nisu savršeno rešenje, jer drugi ili treći pipelining mogu da izvršavaju samo ograničen skup instrukcija koji je nezavisan od instrukcija koje izvršava prvi pipelining.

Primer superskalarne arhitekture sa 3 pipelining-a /slika 2.16, str. 56/
Superskalarni procesor sa 6 funkcionalnih jedinica /slika 2.17, str. 57/

2.8.1.2. Paralelizam na nivou procesora

Višeprocesorski sistem sa deljenom memorijom (GMSV - Global Memory, Shared Variable) /slika 2.19, str. 59/

Može se desiti da ***više procesora pristupa globalnoj memoriji***. Tada se za rešavanje mogućeg nastanka kolizije koriste sledeća tri modela:

1. **EREW** (Exclusive Read Exclusive Write) - ne dozvoljava nastajanje konflikta niti za čitanje niti za pisanje. Ako prilikom izvršavanja programa dodje do konflikta, stanje programa je tada nedefinisano.
2. **CREW** (Concurrent Read Exclusive Write) - dozvoljava konkurentno čitanje za više procesora. Međutim, ako dođe do konkurentnog pisanja, stanje programa tada je nedefinisano.
3. **CRCW** (Concurrent Read Concurrent Write) - dozvoljava istovremeni pristup više procesora i za operaciju čitanja i za operaciju pisanja. U slučaju da ***više procesora pokušava da istovremeno upiše neku vrednost*** u istu memorijsku lokaciju, treba definisati koji će od tih procesora stvarno izvršiti upis. Moguća su tri slučaja:
   - svi procesori moraju da upišu istu vrednost,
   - bilo koji procesor može da izvrši operaciju pisanja i nema ograničenja za vrednost koja se upisuje,
   - svaki procesor ima unapred definisan prioritet i operaciju izvršava procesor sa najvišim prioritetom.

Višeprocesorski sistem sa distribuiranom deljenom memorijom (DMSV - Distributed Memory Shared Variable) /slika 2.20, str. 60/
Višeračunarski sistem sa distribuiranom nezavisnom memorijom (DMMP - Distributed Memory Message Passing) /slika 2.21, str. 60/

3. KONVENCIONALNI MAŠINSKI NIVO

3.1. Format instrukcija

Program možemo posmatrati kao niz instrukcija, a za svaku je neophodno definisati format. ***Format instrukcije*** je definisan ukupnom dužinom instrukcije izraženom u bitovima, brojem, dužinom i značenjem svakog njenog polja. Svaka instrukcija ima **dva dela**: 1. operacioni deo (kod operacije) i 2. adresni deo (pomoću koga se pristupa podacima).

Postoje sledeći formati instrukcija:

1. troadresne instrukcije (imaju tri adresna dela),
2. dvoadresne instrukcije,
3. jednoadresne instrukcije i
4. nulaadresne (primena instrukcija bez adresnog dela omogućena je pomoću **stek mašine** (niz memorijskih lokacija)).

Dva osnovna ***nedostatka formata instrukcije sa 3 memorijske adrese*** su: velika dužina instrukcije i tri memorijska pristupa.

3.1.1. Kriterijumi pri projektovanju formata instrukcija

Kriterijum za dužinu instrukcije: potrebno je da instrukcija bude što kraća jer tada zauzima manje memorije i prenos instrukcije iz memorije u procesor se izvršava brže.

Kriterijum za dužinu koda operacije: za skup od 2^n instrukcija dužina polja za kod operacije u okviru instrukcije ne može biti manja od n bitova.

Kriterijum za dužinu adresnog dela instrukcije: potrebno je da adresa bude što duža, što je u suprotnosti sa prvim zahtevom.

3.2. Tipovi instrukcija

Osnovne operacije savremenih računara mogu se podeliti u šest kategorija: 1. aritmetičke i logičke, 2. operacije pomeranja i rotacije bitova, 3. memorijske operacije, 4. operacije poređenja, 5. upravljačke operacije (grananja), 6. ostale operacije.

3.2.1. Aritmetičke i logičke operacije

Aritmetičke operacije omogućavaju osnovna izračunavanja (sabiranje, oduzimanje, množenje, deljenje, inkrementiranje i dekrementiranje). **Logičke operacije** - and, or, xor...

3.2.2. Operacije pomeranja i rotacije bitova

Pripadaju **unarnim operacijama** - operacije koje se izvršavaju nad jednim operandom i daju jedan rezultat. Koriste se kod ispitivanja vrednosti pojedinih (ili svih) bitova u jednoj reči, a u kombinaciji sa binarnim operacijama, i za izvlačenje (maskiranje) ili ubacivanje (pakovanje) niza bitova u jednoj reči.

3.2.3. Memorijske operacije

Omogućavaju prenos podataka između procesora i memorije. Sadrže: adresu izvora, adresu odredišta i dužinu podatka koji se prenosi.

3.2.4. Operacije poređenja

Porede dve ili više zadatih vrednosti. Na osnovu rezultata se vrši postavljanje odgovarajućih flegova, a u zavisnosti od postavljenih vrednosti flegova vrši se dalje izvršavanje programa.

3.2.5. Upravljačke operacije

Utiču na tok izvršavanja programa tako što menjaju sadržaj registra BN. Mogu se podeliti u dve kategorije: bezuslovno grananje i uslovno grananje.

3.2.6. Ostale operacije

Operacija **AND** se često koristi za izvlačenje bitova iz jedne reči /primer 1, str. 73/. Funkcije **AND** i **OR** se koriste za ubacivanje (pakovanje) bitova /primer 2, str. 73/.

3.3. Adresiranje

Načini adresiranja određuju adresu objekta kome se pristupa. Postoje sledeći ***načini adresiranja***: trenutno, direktno memorijsko, direktno registarsko, indirektno memorijsko, indirektno registarsko, bazno registarsko i indeksno adresiranje.

3.3.1. Trenutno adresiranje

Kod ovog adresiranja, operand se nalazi u samoj instrukciji. Trenutno adresiranje se koristi za rad sa konstantama. /slika 3.9, str. 74/

3.3.2. Direktno memorijsko adresiranje

Koristi se kada se pristupa operandu koji se nalazi u memoriji. Tada se u adresnom delu instrukcije nalazi memorijska adresa operanda. /slika 3.10, str. 74/ Koristi se za pristup statičkim podacima.

3.3.3. Direktno registarsko adresiranje

Koristi se kada se pristupa operandu koji se nalazi u datom registru. Tada se u adresnom delu instrukcije nalazi adresa registra u kome se nalazi operand. /slika 3.11, str. 75/

3.3.4. Indirektno memorijsko adresiranje

Kod ovog adresiranja u adresnom delu instrukcije se nalazi adresa memorijske lokacije koja sadrži memorijsku adresu operanda. Koristi se za rad sa pokazivačima. /slika 3.12, str. 76/

3.3.5. Indirektno registarsko adresiranje

U adresnom delu instrukcije se nalazi adresa registra koji sadrži memorijsku adresu operanda. Koristi se za pristup podacima korišćenjem pokazivača ili izračunate adrese. /slika 3.13, str. 77/

3.3.6. Bazno registarsko adresiranje

Kod ovog adresiranja memorijska adresa operanda se dobija kao zbir vrednosti koja se nalazi u registru čija je adresa data u prvom delu adrese instrukcije i pomeraja čija je vrednost data u drugom delu iste adrese instrukcije. /slika 3.14, str. 78/ Koristi se za pristup lokalnim promenljivama. Pomoću baznog registarskog adresiranja može se simulirati registarsko indirektno i direktno memorijsko adresiranje.

3.3.7. Indeksno adresiranje

Koristi se za adresiranje nizova. /slika 3.15, str. 79/

3.4. Predstavljanje podataka

Sistemi koji preslikavaju električni signal u dve vrednosti su poznati kao **binarni sistemi**. Dodeljena vrednost zavisi od nivoa napona tog signala - niski naponi se interpretiraju kao 0, a visoki kao 1.

3.5.1. Upravljanje tokom izvršavanja instrukcija unutar modula

Naredba ***bezuslovnog skoka*** u svom adresnom delu sadrži adresu instrukcije na koju se vrši prelazak (skok). Simbolički operacioni kod ove naredbe na konvencionalnom mašinskom nivou razlikuje se od računara do računara. Oblik GOTO je uobičajen u literaturi, a izvršenje naredbe GOTO se svodi na upisivanje njenog adresnog dela u BN.

Upravljanje iterativnom programskom strukturom na konvencijalnom mašinskom nivou se realizuje pomoću instrukcija poređenja i uslovnog grananja.

3.5.2. Upravljanje tokom izvršavanja modula unutar programa

Realizacija poziva podprograma i povratka u pozivajući modul može se realizovati na tri načina: pomoću unapred određene lokacije (ili fiksne reči), pomoću prve reči pozvanog podprograma i pomoću steka.

U slučaju ***rekurzivnih modula*** pri prvom rekurzivnom pozivu došlo bi do gubitka adrese povratka u pozivaoca. Rešenje problema direktne i indirektne rekurzije može se realizovati čuvanjem adrese povratka na steku.

Korutine se koriste u projektovanju i implementaciji programa prevodioca (compiler-a) i u simulaciji uporednih procesa.

3.6. Prevođenje, povezivanje i punjenje programa

Program koji vrši prevođenje korisničkog programa iz simboličkog u binarni oblik naziva se **asembler.**

Elementi koji su potrebni da se nađu u memoriji kod realizacije asemblera (pored korisničkog programa):
- **tabela instrukcija** - sadrži simboličke i binarne operacione kodove svih instrukcija koje procesor može da izvrši,
- **jedna memorijska lokacija** koja treba da igra ulogu posebnog brojača koji se naziva brojač lokacija (BL),
- **dovoljno slobodne memorije**, tako da asembler može u toku rada da formira posebnu tabelu koja se naziva tabela simbola.

Princip rada asemblera na primeru asemblera sa dva prolaza. Na početku rada asemblera BL se postavlja na nulu. Sledeći korak je da se inicijalizuje tabela simbola, koja je u tom trenutku još uvek prazna. Asembler onda uzima prvu instrukciju simboličkog programa i na osnovu tabele instrukcija pronalazi odgovarajući binarni opkod, a zatim uzima simboličku adresu i proverava da li se ta adresa već nalazi u tabeli simbola. Kako to nije slučaj, simboličku adresu upisuje u posmatranu tabelu, a zatim instrukciju prepisuje u memoriju u binarno-simboličkom obliku. Kako jedna instrukcija zauzima jednu memorijsku lokaciju, prva instrukcija zauzeće lokaciju sa adresom 0 i zbog toga se sadržaj BL-a povećava za 1. Postupak se ponavlja za sve instrukcije simboličkog programa, čime se završava tzv. prvi prolaz asemblera kroz program. Rezultat prvog prolaza je program u binarno-simboličkom obliku. U drugom prolazu asembler prolazi kroz binarno-simbolički oblik programa i u svakoj instrukciji simboličku adresu zamenjuje odgovarajućom memorijskom adresom na osnovu tabele simbola.

Compiler (program prevodilac) služi za prevođenje višeg programskog jezika na asemblerski jezik. ***Linker*** (program za povezivanje) služi da se glavni program i svi podprogrami povežu u jednu celinu. Linker na

svom ulazu dobija programske module u binarno-simboličkom obliku, a na izlazu daje povezanu programsku celinu u binarnom obliku. ***Loader*** (program za punjenje) se aktivira svaki put kada korisnik sistema zahteva izvršavanje prevedenog i povezanog programa.

Povezani oblik programa naziva se ***rekolatibilni binarni kod***, zbog toga što je u takvom obliku da se može uneti (locirati) u bilo koji slobodan deo memorije. Prilikom više izvršavanja programa on se može svaki put uneti u neki drugi deo memorije i svaki put se može uspešno izvršiti.

4. MIKROPROGRAMSKI NIVO

Mikroinstrukcija je skup mikrooperacija koje se izvršavaju u isto vreme. Skup mikroinstrukcija na osnovu koga je napravljen potpun skup instrukcija zove se ***mikroprogram*** ili ***mikrokod***. ***Mikroprogramiranjem*** se projektovanje pojednostavljuje, lakše se vrši modifikacija, ali je to sporije rešenje. Izuzetno puno se koristilo na mnogim prvim procesorima jer je tada postojao opšti trend da skupovi instrukcija računara budu sve složeniji.

4.1. Komponente procesora i mikroarhitektura

Skup instrukcija na konvencionalnom mašinskom nivou ***može biti*** implementiran pomoću različitih mikroarhitektura. *Obrazloženje* - str. 102.

Prikaz ciklusa uzimanja i izvršavanja instrukcija na mikroprogramskom nivou /slika 4.1, str. 103/

Prikaz ciklusa uzimanja i izvršavanja instrukcija sa prekidima /slika 4.2, str. 103/

Detaljna blok šema Intel-ovog 8086 procesora /slika 4.3, str. 105/

Mikroarhitektura se može podeliti u dve sekcije: sekciju podataka (datapath) i upravljačku sekciju. ***Sekciju podataka*** čine registri i ALU, a ***upravljačku sekciju*** upravljačka jedinica.

Mikroprogram se često zove i ***firmver (firmware)*** jer povezuje hardver i softver, odnosno smanjuje razliku između hardvera i softvera. ***Osnovna namena firmvera*** je interpretiranje skupa instrukcija koji je vidljiv krajnjem korisniku. ***Firmver se skladišti*** u upravljačku memoriju. Za programiranje firvera koristi se ***mikro-asembler*** koji ***nije vidljiv*** krajnjem korisniku sistema. Bilo koja ***promena u skupu instrukcija*** na konvencionalnom mašinskom nivou prouzrokuje i promenu u firmveru, dok ***promena softvera na aplikativnom nivou*** ne utiče na firmver.

HDL (Hardware Description Language) je poseban jezik za opis hardvera pomoću koga se upravlja složenošću projektovanja kod implementacije hardverskih komponenti.

4.2. Hipotetička stek mašina

Stek je struktura podataka tipa LIFO (Last In, First Out). Sastoji se od skupa memorijskih lokacija u koje se upisuju vrednosti podataka. **Dve osnovne operacije** za rad sa stekom: PUSH (dodaje, tj. upisuje zadatu vrednost na vrh steka) i POP (čita, tj. sklanja vrednost sa vrha steka.

Hipotetička stek mašina je mašina sa apstraktnom strukturom podataka tipa LIFO, pri čemu program ne pristupa registarskoj datoteci, već isključivo steku. Podrazumeva se da program može da upiše na stek proizvoljnu količinu podataka, odnosno da nema ograničenja za veličinu steka. U praksi, stek mašina se realizuje korišćenjem bafera u memoriji koji ima konačnu veličinu. Ako količina podataka koju treba upisati na stek prevazilazi veličinu memorijskog prostora koji je dodeljen steku, pojavljuje se greška za prekoračenje steka.

Realizacija steka u memoriji /slika 4.4, str. 107/

Radi ***ubrzanja pristupa steku*** koristi se registarska datoteka unutar procesora gde se memoriše prvih N vrednosti sa steka, gde je N veličina registarske datoteke. Kada se registarska datoteka napuni, njen sadržaj se kopira na stek. Kada registarska datoteka postane prazna tada se puni vrednostima sa steka.

Upravljačka memorija je ROM (Read Only Memory) memorija (64x27 bita).

4.4. Poboljšanje performansi

Poboljšanje performansi može se ostvariti poboljšanjem implementacije i poboljšanjem arhitekture sistema.

DEO 2: OPERATIVNI SISTEMI

5. OPERATIVNI SISTEMI

5.1. Uvod

5.1.1. Šta je operativni sistem?

Operativni sistem (OS) je osnova svakog savremenog računarskog sistema i njegova osnovna namena je da obezbedi takvo okruženje u kome će korisnik izvršavati programe na efikasan i pogodan način. To je sistemski softver koji upravlja dodelom i efikasnim korišćenjem resursa datog računarskog sistema potrebnih za rešavanje datog problema. ***Poznati operativni sistemi*** su: UNIX, Linux, BeOS, QNX, iRMX, Mac OS, DOS, Microsoft Windows 95/98/XP/NT/2000/2003/Vista, Solaris, Symbian OS, Palm OS, itd.

Operativni sistem je ***reaktivni program*** i ponovo dobija kontrolu kada korisnička aplikacija izvrši sistemski poziv ili ako se pojavi hardverski prekid, tj. OS je stalno u stanju čekanja da se desi jedan od dva tipa događaja: prekid ili sistemski poziv.

Primer kreiranja više virtuelnih mašina nad istim hardverom /slika 5.3, str. 120/

Postoji više mogućih ***interfejsa između krajnjeg korisnika i OS-a***. Interfejs može biti GKI (grafički korisnički interfejs), bilo koja aplikacija ili interpreter komandi. OS može imati više od jednog interfejsa i kod komercijalnih OS-a uvek postoji GKI i interpreter komandi. /slika 5.4, str. 121/

Iz ***stanja praznog hoda*** OS se budi:
- prekidom dobijenim od nekog hardverskog uređaja,
- pojavom izuzetka od nekog korisničkog programa,
- sistemskim pozivom iz nekog korisničkog programa.

Deo OS-a koji sadrži najčešće korišćene funkcije naziva se **jezgro (kernel)** ili nukleus OS-a.

OS ima **dva načina rada:** korisnički (ograničeni način izvršavanja predviđen za izvršavanje aplikacija) i supervisor način rada (pristup bez ograničenja predviđen za sam OS).

5.1.2. Osnovni ciljevi savremenih operativnih sistema

Ciljevi OS-a su određeni relativnom cenom hardvera i rada ljudi. Danas je glavni cilj omogućiti jednostavno i efikasno korišćenje računara.

Osnovni ciljevi svakog savremenog OS-a su:
- da olakša izvršavanje aplikativnih programa i rešavanje korisničkih problema,
- da učini računarski sistem jednostavnim i podesnim za korišćenje,
- da obezbedi deljenje računarskih resursa,
- da omogući korišćenje računarskog hardvera na efikasan način.

Poznavanje rada OS-a pravi razliku između nekoga ko se profesionalno bavi računarskim sistemima i nekoga ko je samo krajnji korisnik. Razumevanje rada OS-a omogućava uspešniju primenu računarskih sistema i predstavlja važan preduslov za poboljšanje performansi širokog opsega aplikacija kod kojih se u toku životnog ciklusa može pojaviti problem performansi.

5.1.3. Ključni aspekti savremenih operativnih sistema

Ključni aspekti svakog OS-a su:
1. apstrakcije resursa i 2. deljenje resursa.

Primer apstrakcije hardvera računarskih sistema /tabela 5.1, str. 123/

Primer apstrakcije ostalih resursa računarskih sistema koji nisu hardver /tabela 5.2, str. 123/

Deljenje resursa u računarima može biti na osnovu prostornog multipleksiranja i na osnovu vremenskog multipleksiranja. Kod **prostornog** resurs se deli na dva ili više različitih delova, i svaki pojedinačni deo se dodeljuje jednom procesu (npr: zgrada sa apartmanima, vozila gradskog saobraćaja, itd.; primer u računaru - glavna memorija i disk). Kod **vremenskog** resurs se dodeljuje jednom procesu u kratkom vremenskom periodu (npr: taxi vozila, pojedinačna parking mesta, itd.; primer u računaru - centralni procesor).

Najvažniji ***osnovni zadaci OS-a*** su:
- omogućiti da sistem radi,
- omogućiti da sistem korektno radi,
- omogućiti da sistem efikasno radi.

5.1.4. Osnovne karakteristike savremenih operativnih sistema

Savremeni OS imaju sledeće **karakteristike**:
- obezbeđuju jednostavno korišćenje računara (korišćenjem apstrakcije),
- obezbeđuju standardan interfejs (portabilnost),
- obezbeđuju efikasno deljenje resursa,
- značajno troše resurse.

Savremeni OS obezbeđuju sledeće ***servise***:
1. izvršavanje programa,
2. U/I operacije,
3. komunikacije,

4. upravljanje sistemom datoteka,
5. detekcija grešaka.

5.2. Uloga i životni ciklus operativnog sistema

Osnovne funkcije savremenih OS-a su:
- upravljanje procesima,
- upravljanje memorijom,
- upravljanje uređajima,
- upravljanje podacima,
- zaštita,
- komunikacija sa drugim računarima u mreži,
- upravljanje greškama i oporavak sistema.

Svi savremeni procesori (CPUs) imaju bar **dva različita načina (režima) rada**:
1. korisnički i 2. sistemski (kernel) ili privilegovani (supervisor) način rada.

Životni ciklus sistema /slika 5.7, str. 127/

Program koji vrši punjenje drugih programa u memoriju i prenosi kontrolu na te izvršne programe zove se **loader.** Loader je takođe program koji se nalazi u memoriji. Program koji puni „prvi program" tj. kernel u memoriju zove se **bootstrap loader** ili **bootloader**. Bootstrap loader zahteva podršku firmvera i obično ima dva dela: primarni i sekundarni. Firmver prvo napuni primarni bootstrap loader u memoriju, zatim primarni bootstrap loader napuni sekundarni, a nakon toga bootstrap loader prenosi kernel sa diska u memoriju.

Punjenje bootloader-a i kernel-a u operativnu memoriju /slika 5.8, str. 127/

5.3. Vrste operativnih sistema

Po broju procesora operativni sistemi mogu biti:
1. jednoprocesorski i 2. višeprocesorski.

Po broju podržanih procesa i procesora:
1. **monoprogramski** - najstariji operativni sistemi. Oni mogu da izvršavaju samo jedan posao u datom vremenskom intervalu;
2. **višeprogramski** - u datom vremenskom intervalu mogu da izvršavaju više poslova. Kod njih postoji preklapanje zahteva za procesorom i U/I uređajima od strane više programa ili korisnika. Na taj način povećavaju iskorišćenje procesora, jer se teži tome da procesor uvek ima šta da radi;
3. **višeprocesorski** - upravljaju radom računarskog sistema koji ima više procesora, pri čemu svi procesori koriste deljenu memoriju. Mogu biti simetrični i asimetrični. Mogu da izvršavaju programe brže i da povećaju pouzdanost sistema. Složeniji su i hardverski i softverski od jednoprocesorskih sistema;
4. **distribuirani** - projektovani su tako da u datom vremenskom intervalu mogu da izvršavaju više poslova na više procesora koji u opštem slučaju ne koriste deljenu memoriju.

U pogledu ***tipa obrade, načina interakcije korisnika sa računarom i vremena odziva*** na zahtev korisnika razlikujemo:
- **paketnu (batch) obradu** - poslovi se izvršavaju onim redosledom kojim pristižu u red poslova spremnih za izvršavanje. Nedostatak je to što neki kratki poslovi mogu dugo da čekaju završetak prethodno unetih poslova;
- **obradu sa deljenjem procesorskog vremena (time-sharing)** - svakom procesu (korisniku) se u unapred definisanom kvantumu vremena dodeljuje procesor, a nakon isteka vremena OS oduzima procesor od datog procesa i dodeljuje ga sledećem procesu (korisniku);
- **obradu u realnom vremenu (real-time)** - potrebna je kod svih sistema gde se vreme odgovora zahteva u strogo definisanim granicama. Ovde nije dozvoljeno prekoračenje maksimalno dozvoljenog vremena odgovora.

Danas razlikujemo sledeće ***tipove operativnih sistema***:
1. batch sisteme - paketna obrada,
2. interaktivne sisteme (jednokorisnički i višekorisnički),
3. sisteme opšte namene (time-sharing, interaktivni, batch obrada),
4. mrežne operativne sisteme, npr. Windows, UNIX,
5. distribuirane sisteme - grupa računara koja radi kao jedan računar,
6. specijalno projektovane operativne sisteme - sistemi za rad u realnom vremenu. Postoje ***dve grupe sistema za rad u realnom vremenu*** - 1. hard OS (ako se data akcija mora uraditi u datom trenutku) i 2. soft OS (povremeni izostanak aktivnosti koja se mora uraditi u datom trenutku je prihvatljiv).

5.4. Projektovanje operativnih sistema

OS koji ima ***monolitnu strukturu*** napisan je kao skup procedura pri čemu svaka od njih može pozvati bilo koju drugu proceduru iz tog skupa kad god je to potrebno. Svaka procedura u sustemu ima dobro definisan interfejs u obliku parametara i rezultata i svaka je vidljiva svakoj drugoj proceduri. /slika 5.9, str. 131/

Alternativa strukturi monolitnog OS-a je arhitektura sa **mikro jezgrom (microkernel)**. Kod arhitektura sa mikro jezgrom samo najvažnije funkcije OS-a se nalaze u jezgru.

Osnovne ***prednosti OS-a sa mikro jezgrom*** su:
- dodavanje novog servisa ne zahteva modifikovanje jezgra OS-a,
- sistem je bezbedniji,

- omogućava uniformni interfejs, proširivost, fleksibilnost i portabilnost,
- predstavlja podršku distribuiranim sistemima,
- predstavlja podršku objektno orijentisanim operativnim sistemima,
- omogućava jednostavnije projektovanje jezgra i pouzdaniji OS.

Savremeni OS se može dekomponovati u module koji obezbeđuju sledeće funkcije:
- upravljanje procesima,
- upravljanje memorijom,
- upravljanje ulazno/izlaznim uređajima,
- upravljanje datotekama,
- upravljanje mrežom.

Prikaz modularnog operativnog sistema /slika 5.12, str. 134/

Prednosti modularnih operativnih sistema su:
1. lakše je modifikovati sistem i ispravljati greške,
2. informacije se čuvaju samo gde je to potrebno,
3. informacijama se pristupa samo unutar definisane i ograničene oblasti.

5.5. Istorija razvoja operativnih sistema

Razlika između mrežnih OS-a i konvencionalnih OS-a koji se izvršavaju nad jednim procesorom - npr. dodatak kontrolera za mrežni interfejs, kao i programa za daljinsko prijavljivanje i daljinski pristup datotekama.

Razlika između konvencionalnih operativnih sistema koji se izvršavaju nad jednim procesorom i distiruiranih je značajna - npr. mogućnost paralelizacije izvršavanja aplikacija je u korist distribuiranih OS-a. Jedan od pravaca daljeg razvoja distribuiranih OS-a je impementacija poboljšanih, kao i novih tehnika i mehanizama zaštite.

6. UPRAVLJANJE PROCESIMA

6.1. Pojam programa, procesa i niti

Program je statički entitet koji se sastoji od programskih instrukcija koje kada se izvršavaju nad nekim skupom podataka definišu proces. ***Proces*** je dinamički entitet koji izvršava program nad datim skupom podataka koristeći resurse sistema. Proces je operacija procesora nad programom kao argumentom. Proces je osnovna jedinica izvršavanja u operativnom sistemu. To je sekvencijalni program koji se izvršava i koji menja stanje sistema. ***Komponente procesa*** su:
- kod programa koji se izvršava,
- podaci koje program koristi,
- resursi potrebni programu i
- status procesa.

Proces nastaje na osnovu zahteva za izvršavanjem datog programa. ***Zahtev može biti od***: korisnika, operativnog sistema ili nekog drugog već aktivnog procesa.

Minimalni potrebni resursi za izvršavanje nekog programa su:
- operativna memorija predviđena za kod programa i podatke koje program koristi,
- skup registara procesora predviđenih za izvršavanje programa.

Najvažnije aktivnosti operativnog sistema u vezi upravljanja procesima su:
1. kreiranje i brisanje i korisničkih i sistemskih procesa,
2. suspenzija i nastavljanje procesa,
3. priprema mehanizama za sinhronizaciju procesa,
4. priprema mehanizama za komunikaciju procesa,
5. priprema mehanizama za upravljanje potpunim zastojem.

Tabela u kojoj se nalaze podaci o svim procesima zove se ***tabela procesa***. Za svaki aktivan proces sistem mora da vodi određene podatke. Skup takvih podataka zove se ***kontrolni blok procesa (KBP)*** ili vektor stanja ili deskriptor procesa. ***KBP*** je struktura podataka koja za dati proces ***sadrži***: ime procesa, identitet vlasnika, prioritet, PSW, oblast programa, oblast podataka, vrednosti registara, logičko stanje, itd.

KBP se ***formira*** u toku kreiranja procesa, a ***uništava*** se po završetku procesa.

Thread ili ***nit*** je entitet koji se izvršava koristeći program i druge resurse od pridruženog procesa. Svaka nit je uvek pridružena nekom procesu. Jednom procesu se može pridružiti više niti.

U datom trenutku samo jedan proces može da se izvršava iako više procesa može biti spremno za izvršavanje. Dispečer je važan deo OS-a koji vodi računa o raspoređivanju procesa. Postoji više algoritama na osnovu kojih se vrši raspoređivanje procesa.

6.1.1. Dijagram stanja procesa

Moguća stanja procesa su:
- NOV - proces je kreiran;
- IZVRŠAVA SE - instrukcije datog programa se izvršavaju;
- ČEKA - proces čeka da se neki događaj dogodi;
- SPREMAN - proces čeka da bude dodeljen procesoru;

- ZAVRŠEN - proces je završio izvršavanje.

Prelasci između mogućih stanja procesa. Nakon kreiranja procesa on se nalazi u stanju NOV. Kada se procesu dodele potrebni resursi, osim procesora, prelazi u stanje SPREMAN. Ako nema procesa koji je spreman za izvršavanje procesor prelazi u tzv. prazan hod. Ako ima bar jedan proces koji je u stanju SPREMAN tada proces prelazi u stanje IZVRŠAVA SE. Po isteku dodeljenog kvantuma vremena proces prelazi iz stanja IZVRŠAVA SE u stanje SPREMAN. Ako u stanju IZVRŠAVA SE proces mora da sačeka neki događaj, tada prelazi u stanje ČEKA. Po završetku određene aktivnosti proces iz stanja ČEKA prelazi u stanje SPREMAN. Proces prelazi iz stanja SPREMAN u stanje IZVRŠAVA SE kada procesor postane raspoloživ. Ako se u toku izvršavanja procesa završi izvršavanje svih programskih instrukcija, tada proces prelazi u stanje ZAVRŠEN.

Za proces se kaže i da je blokiran kada se nalazi u stanju ČEKA. ***Razlozi blokiranja procesa*** su: čekanje na dobijanje nekog uređaja, upravljanje straničnim prekidom, čekanje ulaznih podataka koje unosi korisnik preko terminala, čekanje na dozvolu ulaska u kritičnu sekciju, čekanje na unapred definisano vreme izvršavanja dela koda, itd.

Dijagram stanja procesa /slika 6.1, str. 143/

6.1.1.1. Kreiranje procesa

Procesi mogu kreirati nove procese. Proces koji kreira novi proces zove se **proces roditelj**, a kreirani proces zove se **proces dete.** Deljenje resursa između procesa roditelj i procesa dete može biti tako da:
- roditelj i dete dele sve resurse,
- dete deli podskup roditeljskih resursa,
- roditelj i dete nemaju zajedničke resurse.

6.1.1.2. Makroinstrukcija WAIT

Makroinstrukcijom WAIT proces obaveštava OS da mora da sačeka neki događaj. Rezultat izvršavanja ove makroinstrukcije je prelazak procesa iz stanja IZVRŠAVA SE u stanje ČEKA.

6.1.1.3. Makroinstrukcija SIGNAL

Makroinstrukcijom SIGNAL operativni sistem obaveštava procese o pojavi događaja (na koji može da čeka više procesa). Ova informacija se upisuje u blok procesa. Rezultat izvršavanja je prelazak procesa iz stanja ČEKA u stanje SPREMAN.

Kod realnih OS-a proces može biti i ***suspendovan.*** Prilikom suspendovanja procesa oslobađaju se svi resursi koje je proces do tada zauzeo. Uvođenjem ovih procesa uvode se još dva stanja: SUSPENDOVAN I SPREMAN i SUSPENDOVAN I ČEKA. Iz ovih stanja proces se vraća u stanja SPREMAN i ČEKA samo na zahtev korisnika. ***Proces može biti suspendovan*** na zahtev korisnika, OS-a radi izbegavanja potpunog zastoja (deadlock) ili ako se prekorači maksimalni dozvoljeni broj procesa u stanju SPREMAN.

6.1.2. Adresni prostor procesa

Adresni prostor procesa /slika 6.2, str. 146/ Svaki proces u toku izvršavanja zahteva memoriju i zato se svakom procesu pridružuje memorijski adresni prostor. Memorija je potrebna radi skladištenja programskih instrukcija, podataka, dinamičkih struktura podataka i sadržaja koji se čuva na steku. U delu memorije *Tekst* nalaze se instrukcije programa i taj deo je fiksne veličine. Sadržaj ovog dela generisan je od strane programskog prevodioca i može se deliti između više procesa koji izvršavaju isti program. U delu memorije *Podaci* nalaze se predefinisane strukture podataka koje su inicijalizovane pre početka izvršavanja programa. Sadržaj ovog dela kreiran je od strane programskog prevodioca. Deo *Heap* se koristi za dinamičku dodelu memorije. Ovaj deo je promenljive veličine i memorija se dinamički dodeljuje na osnovu zahteva. U delu *Stek* se nalaze izvršni okviri funkcija koje su pozvane od strane datog programa. Veličina ovog dela je promenljiva; u toku izvršavanja programa *Stek* memorija se povećava prema potrebi. Povećanje i smanjenje je automatski.

6.1.3. Dijagram stanja niti

Multithreading (rad sa više niti) se odnosi na mogućnost OS-a da podrži izvršavanje više niti u okviru jednog procesa. ***Moguća stanja niti*** su ista kao kod dijagrama stanja procesa. U okviru jednog procesa može biti jedna ili više niti pri čemu svaka nit ima:
- stanje izvršavanja,
- sačuvan kontekst niti kada se ne izvršava.

Dijagram stanja niti /slika 6.4, str. 148/

Resursi za kreiranje niti. Za kreiranje niti potrebno je manje resursa nego za kreiranje procesa. Kreiranje procesa zahteva dodelu kontrolnog bloka procesa (KBP), tj. prilično velike strukture podataka. Kreiranje bilo korisničke bilo sistemske niti podrazumeva dodelu male strukture podataka u kojoj se čuvaju sadržaji skupa registara, sadržaj steka i prioritet.

Primeri aplikacija sa više niti:
- Web server - svaki zahtev servisira u posebnoj niti;
- interaktivni program sa grafičkim korisničkim interfejsom (debugger) - jedna nit se koristi za nadgledanje korisničkih ulaznih podataka, druga predstavlja aplikaciju koja se izvršava, a treća može da nadgleda performanse;

- aplikacija za množenje matrica - različiti delovi matrice mogu da se koriste paralalno pomoću više niti.

6.1.4. Raspoređivanje procesa (process scheduling)

Postoji više različitih redova čekanja. Npr. red čekanja poslova (red čekanja svih procesa u sistemu), red čekanja spremnih poslova (red čekanja svih procesa u glavnoj memoriji koji su spremni i čekaju na izvršavanje), svakom zajedničkom resursu se dodeljuje red čekanja (npr. red čekanja za neki U/I uređaj).

Blok šema rada operativnog sistema sa redovima čekanja /slika 6.5, str. 149/

Prelasci procesa između različitih redova čekanja. Procesi migriraju između različitih redova čekanja. Prvo se svaki proces nalazi u redu čekanja poslova. Nakon prebacivanja u glavnu memoriju i dodele potrebnih resursa proces prelazi u red čekanja spremnih poslova. Kada proces dobije procesor mogući su sledeći slučajevi:
- proces se izvršava sve dok ne istekne dodeljeno vreme,
- postoji zahtev za izvršavanje U/I operacije i proces se prebacuje u U/I red čekanja,
- nastaje prekid i proces prelazi u stanje spreman.

U toku svog životnog ciklusa proces može da kreira nov podproces/podprocese.

6.1.5. Strukture podataka za predstavljanje poslova, procesa i niti

Operativni sistem koristi **strukture podataka** za predstavljanje različitih objekata sa kojima radi.

Struktura podataka koja predstavlja proces u obliku povezane liste /slika 6.6, str. 150/
Struktura podataka koja predstavlja posao (task), tj. proces /slika 6.7, str. 150/
Struktura podataka koja predstavlja nit /slika 6.8, str. 151/
Struktura podataka koja predstavlja nepostojano okruženje niti /slika 6.9, str. 151/

6.2. Konkurentni procesi

Na sistemu sa jednim procesorom moguće je samo kvaziparalelno izvršavanje procesa. U bilo kom trenutku izvršava se samo jedan proces. /slika 6.10, str. 152/

Primer paralelnih procesa /slika 6.11, str. 152/

6.2.1. Kooperacija

Jedna nit može da zahteva servis neke druge niti. Tada ta nit mora da sačeka da se pozvani servis završi. Potreban je mehanizam sinhronizacije jedne niti sa drugom niti ili sa hardverom. To se može uraditi pomoću signala koji ima značenje da je pozvani deo posla završen.

6.2.2. Takmičenje (konkurentnost)

Konkurentne niti su niti koje se izvršavaju u isto vreme. Mogu da se takmiče za ekskluzivno korišćenje resursa; mogu da postavljaju istovremene zahteve za istim resursom ili istim servisom.

Kritična sekcija je segment koda čije instrukcije mogu da utiču na druge niti. Kada jedna nit izvršava kritičnu sekciju nijedna druga nit ne sme da izvršava tu istu kritičnu sekciju.

6.3 Problem upravljanja konkurentnim procesima

Da bi dva procesa, S i R, ispravno funkcionisali, u situaciji kada njihove relativne brzine nisu unapred poznate, neophodno je da budu ispunjeni sledeći uslovi:
- pošiljalac S ne sme da šalje poruku kada je bafer pun jer bi se ona izgubila,
- primalac R ne sme da uzima poruku kada je bafer prazan jer bi to mogao protumačiti kao signal da treba da prestane sa radom,
- procesi ne smeju istovremeno pristupiti baferu,
- vrednost promenljive N mora, u trenutku pristupa baferu bilo kog procesa, biti jednaka stvarnom broju poruka u baferu.

Osnovni zahtevi koje operativni sistem mora da obezbedi u cilju uspešnog upravljanja konkurentnim procesima:
zahtev 1: uporedni procesi sa kritičnim sekcijama, jedan u odnosu na drugog, moraju se međusobno isključivati tako da se istovremeno ne nalaze u svojim kritičnim sekcijama;
zahtev 2: proces koji se zaustavio van svoje kritične sekcije ne sme onemogućiti dalje odvijanje drugih nezavisnih procesa;
zahtev 3: procesi ne smeju beskonačno dugo čekati na resurse ili signale.

Osnovna četiri problema u upravljanju procesima:
1. ***Međusobno isključivanje procesa u odnosu na kritične sekcije*** znači da se u bilo kom vremenskom trenutku samo jedan proces može naći u svojoj kritičnoj sekciji;
2. ***Sinhronizacija procesa*** znači da određen proces ne može ići dalje od unapred definisane tačke bez eksplicitnog signala koji on sam ne može da generiše;
3. Dva procesa su ***blokirana*** ako nijedan od njih ne može da nastavi sa radom dok drugi ne nastavi sa radom. Sistem je blokiran ako su svi procesi blokirani;
4. U računarskom sistemu neophodno je realizovati mehanizam za komunikaciju između procesa na takav način da im omogući da međusobno sarađuju i uspešno razmenjuju poruke.

6.4. Semafori i primena semafora

E. Dijkstra je dao ***rešenje problema međusobnog isključenja i sinhronizacije procesa*** tako što je uveo nov tip promenljive koji se naziva **semafor** i čiji je skup dozvoljenih vrednosti skup negativnih celih brojeva. Uveo je dve dozvoljene operacije nad ovim promenljivama koje je nazvao P i V operacijama, pri čemu je:

P(s)= s: = s - 1 ako je s>0
čekaj u suprotnom

V(s) = s: = s + 1,

gde je *s* promenljiva tipa semafor.

„Busy wait" situacija. Proces koji pokušava da izvrši P operaciju u situaciji kada je vrednost promenljive *s* jednaka nuli, troši vreme centralnog procesora, pri čemu ne može da napreduje dalje, a istovremeno onemogućava ostale procese da se izvršavaju.

U toku daljeg razvoja softvera za rad sa konkurentnim nitima uvedeni su semafori. Oni mogu da se koriste za:
- upravljanje međusobnim isključivanjem konkurentnih niti,
- sinhronizaciju niti,
- dodelu resursa.

U okviru operativnih sistema implementacija semafora uključuje interakciju sa **planerom**:
- za blokiranje niti,
- za deblokiranje niti,
- za upravljanje redovima čekanja niti u stanju čekanja.

7. UPRAVLJANJE MEMORIJOM

Upravljanje memorijom je jedna od osnovnih funkcija operativnih sistema. Za izvršavanje svakog procesa je potrebna operativna memorija. **Glavni cilj kod upravljanja memorijom** je da se kombinovanjem velike spore memorije sa malom brzom memorijom ostvari efekat velike brze memorije. Za upravljanje memorijom bitni su programski prevodilac, operativni sistem i hardver.

7.1. Problem upravljanja memorijom

Najvažnije aktivnosti OS-a u delu za upravljanje memorijom su:
1. vođenje evidencije o tome koji se delovi memorije trenutno koriste i ko ih koristi,
2. donošenje odluke o učitavanju procesa u memoriju,
3. dodela i oslobađanje memorijskog prostora po potrebi.

Komponente upravljanja memorijom:
- upravljanje unošenjem (fetch policy),
- upravljanje smeštanjem (placement policy),
- upravljanje zamenom (replacement policy).

Dodela memorije može biti **kontinualna** (ako se za dati proces koriste susedne memorijske lokacije u okviru datog dela memorije) i **diskontinualna** (ako se koriste delovi memorije kod kojih postoji diskontinuitet u pogledu susednosti lokacija).

Upravljanje memorijom može biti **statičko** (ceo program se unosi u memoriju pre izvršavanja programa) i **dinamičko** (kada se veličina memorije određuje na osnovu veličine programa u trenutku unošenja programa u memoriju ili kada se delovi programa mogu unositi u memoriju u toku izvršavanja programa). U opštem slučaju postoje sledeći ***načini upravljanja memorijom***:
- pomoću statičkih particija;
- pomoću dinamičkih particija;
- pomoću statičkih stranica;
- pomoću statičkih segmenata;
- pomoću dinamičkih stranica;
- pomoću dinamičkih segmenata;
- pomoću dinamičkih stranica i segmenata. /slika 7.1, str. 173/

7.1.1. Višenivoski sistem

Memorijski sistem savremenih računarskih sistema je višenivoski sistem. /slika 7.2, str. 174/

Memorijski sistem kod savremenih računarskih sistema /slika 7.3, str. 175/ Povećavanjem brzine pristupa smanjuje se jedinična cena memorije i obrnuto. Program za upravljanje memorijom (**memory manager**) izvršava dve osnovne operacije: dodelu kontinualnog memorijskog prostora i oslobađanje zauzete memorije.

7.1.2. Vezivanje adresa

Vezivanje adresa je proces pridruživanja fizičkih memorijskih adresa adresama podataka i adresama programskih instrukcija. Može biti statičko i dinamičko. Kod ***statičkog*** vezivanja nove lokacije se određuju pre izvršavanja programa, a kod ***dinamičkog*** nove lokacije se određuju za vreme izvršavanja programa. Kod dinamičkog vezivanja adresa ***apsolutne adrese generiše hardver***.

Upravljanje memorijom obuhvata sledeće ***funkcije preslikavanja***:
1. preslikavanje imena (odnosi se na preslikavanje simboličkih adresa);

2. preslikavanje adresa (odnosi se na preslikavanje programskih adresa u stvarne, fizičke memorijske adrese);
3. preslikavanje sadržaja (odnosi se na preslikavanje memorijskih adresa u vrednosti koje one sadrže).

7.1.3. Virtuelne i fizičke adrese

Razlikujemo logički i fizički memorijski prostor. Adresa koju generiše procesor se zove **logička** adresa, a adresa kojom se puni memorijski adresni registar (MAR) je **fizička** adresa. ***Fizička adresa*** je adresa operativne memorije. ***Virtuelna adresa*** je adresa u programu i nju generiše procesor.

7.1.4. Fragmentacija

Fragmentacija se odnosi na neiskorišćenu memoriju koju sistem za upravljanje memorijom ne može da dodeli procesima. Postoje dva tipa fragmentacije: interna i eksterna. ***Interna fragmentacija*** je deo memorije unutar regiona ili stranice koja je dodeljena datom procesu i ne koristi se od strane tog procesa. Prouzrokovana je različitom veličinom dodeljene memorije i programa koji je učitan u taj deo memorije. Ne postoji kod upravljanja memorijom pomoću dinamičkih particija, kod statičkih segmenata i kod dinamičkih segmenata. ***Eksterna fragmentacija*** je neiskorišćena memorija između particija ili segmenata. Ova memorija nije kontinualna, već se sastoji iz više manjih delova. Ne postoji kod upravljanja memorijom pomoću statičkih i dinamičkih stranica.

7.1.5. Relociranje

Relociranjem programa se ostvaruje efikasno i fleksibilno korišćenje memorijskih resursa.

7.2. Rad sa particijama

Memorija se deli na kontinualne delove koji se zovu **particije.** Razlikujemo statičke i dinamičke particije.

7.2.1. Statičke particije

Upravljanje memorijom pomoću statičkih particija je jedan od najjednostavnijih načina upravljanja. Memorija se deli na particije fiksne veličine koje su fiksirane u vreme inicijalizacije sistema i ne mogu se menjati u vreme izvršavanja programa.

7.2.2. Dinamičke particije

Kod upravljanja memorijom pomoću dinamičkih particija u operativnu memoriju se može učitati onoliko programa koliko ima slobodne memorije. Particije nisu fiksne veličine; veličina particije odgovara veličini programa koji je učitan. Kod dinamičkih particija se mora definisati poseban algoritam za upravljanje smeštanjem.

Algoritam prvog uklapanja - first fit - kod njega OS vodi tabelu slobodnog prostora uređenu po rastućim adresama slobodnih delova memorije. Algoritam se svodi na pronalaženje prvog slobodnog prostora, računajući od početka memorije, koji je dovoljno veliki da prihvati dati program.

Algoritam najboljeg uklapanja - best fit - kod njega se tabela slobodnog prostora uređuje po rastućoj veličini slobodnog prostora. Svodi se na nalaženje najmanjeg dovoljno velikog slobodnog memorijskog prostora u koji može da se smesti novi program.

Postoje i drugi algoritmi smeštanja (***algoritam najgoreg uklapanja - worst fit***, itd).

7.3. Statički segmenti i statičke stranice

Upravljanje memorijom pomoću segmenata i stranica karakteriše mogućnost diskontinualne dodele memorije. Adresni deo naredbe kod upravljanja memorijom pomoću stranica sastoji se iz dva dela, uređenog para (p, d).

7.4. Virtuelna memorija

Osnovna **prednost** korišćenja virtuelne memorije je mogućnost izvršavanja programa koji zahteva memorijski prostor veći od fizičke memorije raspoložive na datom računarskom sistemu. **Osnovne vrste organizacije virtuelne memorije**: segmentna, stranična i segmentno-stranična.

7.4.2. Organizacija sa dinamičkim stranicama

Tipična ***tabela stranica osim adrese okvira ima sledeće dodatne podatke***:
- bit koji pokazuje da li se stranica nalazi u operativnoj memoriji,
- bit koji pokazuje da li je stranica u operativnoj memoriji modifikovana ili ne,
- bit koji pokazuje da li je stranica bila korišćena skoro ili ne,
- dozvola pristupa koja označava da li je stranica „read-only" ili je „read-write",
- nekoliko bita namenjenih za stvarno adresiranje stranice u operativnoj memoriji.

U specijalnoj keš (cache) memoriji se čuva informacija o skoro korišćenim preslikavanjima stranica u okvire. Ta keš memorija se zove ***TLB (Translation Lookaside Buffer) bafer*** i deo je jedinice za upravljanje memorijom (MMU). TLB bafer je nezavisan od keš memorije za podatke i keš memorije za instrukcije. Korišćenjem TLB bafera pristup stranicama za koje postoje podaci u TLB baferu je trenutan. Za stranice za koje ne postoje podaci u TLB baferu potrebna su dva memorijska pristupa: za čitanje tabele stranica i za pristup stvarnim podacima).

Za svaki pristup podacima su potrebna ***dva pristupa fizičkoj memoriji***. Prvi memorijski pristup je pristup tabeli stranica, a drugi stvarnim podacima. Tabela stranica može biti veoma velika, pa ako se npr. 20 bitova koristi za adresiranje stranice tada u adresnom prostoru ima 2^{20} = 1 048 576 stranica.

7.4.2.1. Stranični prekidi

Kada se u toku izvršavanja programa traži pristup adresi koja pripada strani koja nije u memoriji dolazi do prekida programa. Ova vrsta prekida naziva se ***stranični prekid*** (page fault). Postupak u izvršavanju jedne instrukcije može se prikazati na sledeći način:
1. utvrditi da li je adresa kojoj se pristupa u memoriji; ako jeste, nastaviti sa izvršavanjem instrukcije, u suprotnom - korak 2;
2. prekinuti izvršavanje programa;
3. naći slobodan okvir u memoriji;
4. pronaći na disku stranu kojoj se pristupa i upisati je u slobodan okvir;
5. ažurirati tabelu strana;
6. inicirati izvršavanje instrukcije koja je izazvala prekid.

7.4.2.2. Algoritmi za zamenu stranica

Belady-eva anomalija - kada sa povećanjem broja okvira raste i broj straničnih prekida (kod FIFO algoritma).

7.4.2.3. Trashing

Trashing je stanje u kome su stranični prekidi česti i gde nijedan proces ne uspeva da napreduje sa izvršavanjem. ***Pojava Belady-eve anomalije*** ne znači automatski i pojavu trashing-a. Belady-eva anomalija se može pojaviti u sistemu koji ima samo jedan proces, a takav sistem ne ispunjava zahteve za nastanak trashing-a.

7.4.2.4. Moguća stanja stranica

Stranice mogu biti u jednom od tri stanja:
1. preslikane u okvir fizičke memorije;
2. inicijalno locirane na disku ili vraćene na disk;
3. ne koriste se.

7.4.3. Dinamička segmentno-stranična organizacija

Segmentiranje i straničenje se često kombinuju kako bi se međusobno poboljšali. Segmentirano straničenje je korisno kada tabela stranica postane veoma velika. Straničenje segmenata donosi poboljšanja kada postoje veoma veliki segmenti koji zahtevaju veliko vreme za dodelu memorijskog prostora.

7.6. Zaštita memorije

Može se implementirati hardverski i softverski. Hardversko rešenje je preslikavanje adresa (address translation). Hardverska zaštita memorije se može ostvariti korišćenjem dva registra: **bazni** (base) i **granični** (limit) registar. U ***baznom registru*** se nalazi najmanja adresa fizičke memorije gde je dati program smešten, a u ***graničnom*** veličina opsega memorije unutar koga program može pristupati. /slika 7.20, str. 201/
Najpre se proverava da li je generisana adresa veća ili jednaka vrednosti koja se nalazi u baznom registru. Ako je ovaj uslov ispunjen proverava se da li je generisana adresa manja od zbira vrednosti u baznom i graničnom registru. Ako je i ovaj uslov ispunjen dozvoljava se pristup fizičkoj memoriji. Ovakvom hardverskom zaštitom eliminisana je mogućnost da neki korisnički program nakon adresiranja pristupi delu memorije koji je izvan njegovog dozvoljenog adresnog prostora. Ako je generisana adresa van dozvoljenog adresnog prostora tada će doći do sistemske greške, tzv. fatalne greške.

Jedino OS može da napuni bazni i granični registar korišćenjem specijalne privilegovane instrukcije.

8. DODELA PROCESORA

8.1. Uloga planera i dispečera

Operativni sistem upravlja svim procesima koji se izvršavaju na datom sistemu i svakom procesu dodeljuje kvantum vremena za korišćenje procesora. Delovi operativnog sistema koji upravljaju dodelom procesora se zovu **planer** i **dispečer**.

8.1.1. Dugoročni i kratkoročni planer

Određivanje načina na koji se procesorsko vreme dodeljuje procesima se zove **planiranje**. Program koji vrši dodelu procesorskog vremena se zove planer. Kod operativnih sistema postoje ***dve vrste planera***:
1. ***dugoročni planer*** (selektuje proces iz skupa procesa koji su poslati na izvršene i prebacuje ga u red procesa spremnih za izvršenje);
2. ***kratkoročni planer*** (selektuje proces iz reda procesa spremnih za izvršavanje i dodeljuje mu centralni procesor).

Učestanost pozivanja kratkoračnog i dugoročnog planera se bitno razlikuje. Kratkoročni planer se poziva veoma često. To vreme se meri milisekundama. Dugoročni se ne poziva tako često, već su u pitanju minuti ili sekunde.

Kada procesor dodeli drugom procesu sistem mora da sačuva stanje prethodnog procesa. To se zove ***čuvanje konteksta.***

8.1.2. Dispečer

Dispečer je softverski modul koji dodeljuje procesor procesu koji je izabran od strane kratkoročnog planera.
Funkcije dispečera su:
- promena konteksta;
- prelazak u korisnički način rada,
- skok na odgovarajuću lokaciju unutar korisničkog programa radi ponovnog startovanja tog programa.

Vreme koje je potrebno da dispečer zaustavi jedan proces i pokrene drugi se zove ***dispečersko kašnjenje (dispatch latency).***

8.2. Algoritmi dodele procesora

Kriterijumi koji se koriste kod algoritama planiranja su:
- iskorišćenje centralnog procesora,
- propusnost sistema,
- vreme procesa provedeno u sistemu,
- vreme čekanja,
- vreme odziva.

U sistemu u kome postoji n procesa koji se izvršavaju na jednom centralnom procesoru broj mogućih načina za dodelu procesora je: n! = n * (n-1) * (n-2) * ... * 2*1.

Algoritmi za dodelu procesora mogu biti:

1. sa prekidanjem (preemptive) – omogućavaju da proces može biti prekinut u toku izvršavanja i da se procesor dodeli nekom drugom procesu.

2. bez prekidanja (non-preemptive) – onemogućavaju prekidanje procesa koji se trenutno izvršava.

Tri okruženja koja se razlikuju po načinu optimizacije dodele procesora:
1. paketna obrada; 2. interaktivan rad; 3. rad u realnom vremenu.

Najpoznatiji i najznačajniji algoritmi dodele procesora su:
1. FCFS, 2. SJF, 3. SRTF, 4. HPF, 5. RR, 6. MLS.

FCFS (First Come First Served) – Prvi došao, prvi opslužen. To je najjednostavniji algoritam. Implementacija se svodi na FIFO red za čekanje. Procesor se dodeljuje procesu koji je prvi u redu čekanja procesa spremnih za izvršavanje. Svi ostali procesi koji se kasnije pojave se stavljaju na kraj reda. Kada se blokira proces koji se trenutno izvršava, prvi sledeći proces iz reda čekanja počinje sa izvršavanjem. Kada blokirani proces postane spreman, on se stavlja na kraj reda. Glavna prednost ovog algoritma je jednostavnost.

SJF (Shortest Job First) – Najkraći posao prvi. Postoje dve klase procesa CPU bound i I/O bound. CPU bound intenzivno koriste centralni procesor, a I/O bounds intenzivno koriste ulazno-izlazne aktivnosti. SJF algoritam minimizira prosečno vreme provedeno u sistemu. SJF algoritam se još zove i SPTF algoritam (Shortest Processing Time First). ***Optimalan je*** samo kada su svi poslovi raspoloživi istovremeno. ***Nedostatak*** je da u opštem slučaju algoritam ne može biti implementiran. Moguće je i „gladovanje" (starvation), tj. da neki proces ne može da dobije procesor jer se stalno pojavljuju procesi kojima je za završetak rada potrebno manje procesorskog vremena. SJF algoritam se samo u kombinaciji sa drugim algoritmima koristi u operativnim sistemima.

SRTF (Shortest Remaining Time First) – Prvi je posao kome je preostalo najmanje vremena do završetka. Planer iz reda čekanja uvek bira posao koji zahteva najmanje procesorskog vremena. Kod ovog algoritma moguće je prekidanje procesa koji se trenutno izvršava. Ako se u trenutku izvršavanja proesa pojavi proces koji zahteva manje procesorskog vremena, izvršavanje datog procesa se prekida i procesor se dodeljuje tom drugom procesu.

HPF (Highest Priority First) – Planiranje po prioritetu. Svaki proces ima odgovarajući prioritet. Naredni proces za izvršavanje je onaj koji ima najveći prioritet. Ako su procesi sa istim prioritetom tada se primenjuje FCFS algoritam. Kod Linux OS-a opseg prioriteta je od 0 do 99. Kod UNIX OS-a sistemskim pozivom nice() se može podešavati prioritet u opsegu -20 do +20. Prioritet procesa može biti postavljen od strane korisnika, OS-a ili kombinacijom korisnika i OS-a.
Ako je procesu visokog prioriteta potreban pristup zaštićenom resursu koji trenutno poseduje proces niskog prioriteta, tada je proces visokog prioriteta blokiran od strane procesa niskog prioriteta sve dok proces niskog prioriteta ne završi pristup resursu i oslobodi resurs. Na taj način proces niskog prioriteta ima u stvari „viši" prioritet, jer forsira proces višeg prioriteta da čeka. To se zove ***inverzija prioriteta.***

RR (Round-Robin). RR algoritam je parametrizovan i kao parametar koristi interval zauzeća procesora, tj. vremenski interval ili vremenski kvantum (time-slice). Za RR algoritam svi procesi su jednake važnosti. FCFS je specijalni slučaj RR algoritma kod koga je vremenski kvantum beskonačan. Primenom RR algoritma jedan proces dobija procesorsko vreme u trajanju jednog vremenskog intervala, a zatim se procesor dodeljuje narednom procesu. Bitna karakteristika RR algoritma je trajanje vremenskog kvantuma. Ako se izabere previše mala vrednost ima previše context switching-a i na taj način se smanjuje efikasnost centralnog procesora. Ako se izabere previše velika vrednost posledica toga može biti slabo vreme odziva kod kratkih interaktivnih upita. Zato se često za trajanje vremenskog kvantuma kao kompromis uzima vrednost između 20 i 50 milisekundi.

MLS (Multilevel Strategy) – Planiranje sa redovima čekanja u više nivoa. U sistemu postoji više grupa procesa, npr: interaktivni procesi, sistemski i procesi koji rade u paketnom režimu rada. Za svaku grupu formira se poseban red čekanja. Planiranje se zatim vrši po prioritetu redova ili po procentualnoj raspodeli procesorskog vremena između redova. Vremenski kvantum može imati različite vrednosti na različitim nivoima višenivoskog sistema sa redovima čekanja.

Hijerarhijski prikaz algoritama po kriterijumu „is-a special case of" /slika 8.4, str. 214/

9. POTPUNI ZASTOJ

9.1. Potrebni uslovi za nastanak potpunog zastoja

Potpuni zastoj (deadlock) – situacija kada proces trajno ostaje u stanju čekanja, jer u isto vreme postoje drugi procesi koji su takođe u stanju čekanja, a zauzeli su resurse koje zahteva dati proces i te resurse drže zauzetim. U takvoj situaciji dati proces nikada ne dobija zahtevane resurse.

Za ***nastanak*** potpunog zastoja potrebno je i dovoljno da istovremeno budu ispunjeni uslovi:
- međusobno isključenje (znači da je svaki resurs ili trenutno dodeljen tačno jednom procesu ili je raspoloživ);
- posedovanje i čekanje (znači da proces koji poseduje bar jedan dodeljeni resurs može u bilo kom trenutku da zahteva druge resurse koje poseduju drugi procesi);
- kružno čekanje (postojanje zatvorenog kruga procesa, tako da svaki proces u tom krugu čeka na dodelu resursa koji je već dodeljen sledećem procesu);
- nema prekidanja (nije dozvoljeno da OS prekine posedovanje resursa od strane datog procesa).

Da bi nastao potpuni zastoj moraju ***istovremeno*** biti ispunjena sva 4 uslova. Ukoliko bilo koji nije ispunjen, tada u sistemu nema potpunog zastoja.

9.2. Modeliranje potpunog zastoja

Za modeliranje sistema kod kojih su ispunjeni potrebni uslovi za nastanak potpunog zastoja koriste se usmereni grafovi. Takvi grafovi se zovu **grafovi dodele resursa** i imaju ***dva tipa čvorova***: procese koji se prikazuju kao krugovi i resurse koji se prikazuju kao kvadrati.

Primer sistema sa potpunim zastojem /slika 9.3, str. 228/

Primer sistema sa kružnim čekanjem kod koga ne mora da dođe do potpunog zastoja /slika 9.5, str. 230/

U opštem slučaju se može reći da:
1. ako u grafu ne postoji kružni tok, nijedan proces nije u zastoju, dok u suprotnom zastoj može da postoji,
2. ako u grafu postoji kružni tok i pri tome postoji samo po jedan primerak svakog resursa, zastoj postoji.

9.3. Strategije za rešavanje problema potpunog zastoja

9.3.1. Sprečavanje potpunog zastoja

Da bi se sprečio nastanak potpunog zastoja dovoljno je obezbediti da bar jedan od potrebnih uslova nije ispunjen. ***Strategije sprečavanja potpunog zastoja*** podrazumevaju da se najpre napravi model stanja sistema, a zatim da se izabere strategija koja neće omogućiti da sistem uđe u stanje potpunog zastoja.

9.3.2. Izbegavanje potpunog zastoja

Za ***izbegavanje*** potpunog zastoja potrebno je da sistem ima neku informaciju unapred pomoću koje se potpuni zastoj može izbeći. Najjednostavniji model izbegavanja potpunog zastoja se zasniva na uvođenju pravila da svaki proces deklariše za svaki tip resursa maksimalni broj resursa koji mu mogu biti potrebni u toku izvršavanja.

Stanje dodele resursa je definisano:
- brojem raspoloživih resursa,
- brojem dodeljenih resursa,
- maksimalnim zahtevima procesa.

Potpuni zastoj se može izbeći omogućavanjem prekidanja. Strategija izbegavanja potpunog zastoja zasniva se na činjenici da se skup svih stanja sistema može podeliti u tri podskupa. Moguća stanja sistema su: bezbedno, nebezbedno i stanje zastoja.

Bankarov algoritam - za slučaj kada u računarskom sistemu postoji više primeraka istog resursa. Po ovom algoritmu banka nikad ne ulaže raspoloživu količinu novca na takav način da ne može da zadovolji svoje komitente. Procesi su obavezni da prilikom aktiviranja deklarišu tip i maksimalan broj resursa koji će zahtevati. Pri svakom zahtevu za dodelu resursa u toku izvršavanja procesa, sistem, koristeći Bankarov algoritam, proverava da li će posle dodele ostati u bezbednom stanju. Ukoliko je to istina, procesu se dodeljuje resurs, a u suprotnom se zahtev odbija i proces se smešta u red čekanja na resurs.

9.3.3. Otkrivanje potpunog zastoja i oporavak sistema

U sistemima koji mogu biti kritični po život, npr. upravljački sistem u avionu, bezbednost uvek ima prednost u odnosu na jednostavnost sistema.

Ako sistem omogućava nastanak zastoja, potrebno je da se zastoj otkrije i da se uradi oporavak sistema. Najjednostavniji pristup je da kada se potpuni zastoj pojavi da se OS pretvara kao da problem ne postoji i da ostavi problem krajnjem korisniku da ga on reši. Bolji pristup je automatizacija otkrivanja potpunog zastoja korišćenjem posebnog procesa koji nadgleda stanje sistema. Taj namenski proces skenira interne tabele u kojima

je zabeleženo stanje svakog aktivnog procesa. Moguće je pročitati informacije o resursima koje svaki proces poseduje, da li ima čekanja, šta se čeka, itd. Ako se otkrije da postoji kružni lanac procesa koji čekaju, tada postoji potpuni zastoj.

10. UPRAVLJANJE PODACIMA

10.1. Logički i fizički sistem za upravljanje podacima

Najvažnije aktivnosti OS-a u delu za upravljanje podacima na sekundarnoj memoriji su:

1. planiranje dodele sekundarne memorije,
2. dodela slobodne sekundarne memorije,
3. upravljanje slobodnim nemorijskim prostorom na sekundarnoj memoriji.

Sistem za upravljanje podacima ima sledeće ***funkcije***:

- identifikovanje i lociranje izabrane datoteke,
- korišćenje direktorijuma za opisivanje lokacije svih datoteka i njihovih atributa,
- opis kontrole pristupa korisnika u deljenom sistemu,
- rad sa blokovima radi pristupa datotekama,
- dodela slobodnih blokova datotekama,
- upravljanje slobodnim memorijskim prostorom.

Komponente sistema za upravljanje podacima /slika 10.1, str. 240/

10.1.1. Datoteke

Datoteka je imenovani i postojan skup podataka koji je memorisan na nekom memorijskom medijumu, odnosno na nekom periferijskom uređaju. Datoteka je logička jedinica memorije koja predstavlja apstrakciju fizičkih karakteristika datog memorijskog uređaja.
Bitne **karakteristike** sistema datoteka su: struktura, imenovanje, zaštita, fizička organizacija datoteka i način korišćenja. Na jednom disku se može nalaziti više različitih sistema datoteka.
Svaka datoteka ima svoje **atribute**, od kojih su najvažniji: ime, veličina (broj bajtova), datum i vreme poslednjeg ažuriranja, vlasnik, dozvole, lokacija podataka, itd.

Najčešće komande sistema za upravljanje podacima su:

- kreiranje datoteke,
- čitanje i pisanje unutar datoteke,
- pozicioniranje unutar datoteke radi operacije čitanja ili pisanja,
- postavljanje i korišćenje mehanizma zaštite,
- promena vlasništva nad datotekom,
- listanje datoteka u datom direktorijumu,
- brisanje datoteke.

Deskriptor datoteke (file descriptor) na jednistven način identifikuje tu datoteku za sve naredne operacije. Korišćenje celobrojnih nenegativnih vrednosti je jednostavnije od korišćenja imena datoteka. Svaki proces kod UNIX OS-a na početku svog životnog ciklusa ima 3 otvorene datoteke: standardni ulaz (deskriptor 0), standardni izlaz (deksriptor 1) i standardni error (deskriptor 2).

Datoteke ili delovi datoteka mogu biti ***uništeni*** na više načina: hardverske greške, otkazi napajanja, otkazi glava diska, prašina, velike temperaturne promene, velike promene vlažnosti, softverske greške, vandalizam drugih korisnika, prisustvo jakih magnetnih polja, itd.

10.1.2. Direktorijumi

Direktorijum ili ***katalog*** je struktura podataka koja sadrži listu datoteka i poddirektorijuma. Direktorijum omogućava automatsko vođenje evidencije o datotekama i preslikavanje između imena datoteka i samih datoteka. I sam direktorijum je datoteka – posebna datoteka koja sadrži jednu ili više datoteka.

10.1.3. Sistem direktorijuma

Linearni prostor imena /slika 10.2, str. 244/
Hijerarhijski prostor imena /slika 10.3, str. 245/
Aciklični graf /slika 10.4, str. 246/

Operacije nad direktorijumima su: kreiranje, promena imena, brisanje, pretraživanje, ažuriranje, listanje sadržaja, itd.

10.2. Organizacija sistema datoteka

Svaki disk se može podeliti na jednu ili više particija, pri čemu na particijama mogu biti nezavisni sistemi datoteka. Particije mogu biti: primarne, logičke i dodatne.

Primer strukture sistema datoteka /slika 10.5, str. 247/

10.2.1. Organizacija datoteka

Organizacija datoteka označava logičku strukturu slogova datoteke na osnovu načina na koji im se pristupa.
Kriterijumi koji se koriste kod izbora organizacije datoteke su:

- brz pristup podacima,

- jednostavnost ažuriranja,
- jednostavnost održavanja,
- pouzdanost.

Najčešće se koriste sledeće ***organizacije datoteka***:

- serijska, - sekvencijalna, - spregnuta, - rasuta ili direktna,
- indeks-sekvencijalna, - indeksna sa B-stablima, - sa više ključeva, itd.

10.2.2. Primeri sistema datoteka

Najčešće korišćeni sistemi datoteka su: FAT16, FAT32, NTFS, ext2, ext3 i HPFS. FAT16, FAT32 i NTFS su sistemi koji se koriste u Windows operativnim sistemima.

Strukture podataka koje se koriste kod tipičnog sistema datoteka /slika 10.6, str. 251/

Strukture podataka za rad sa datotekama kod UNIX OS-a /slika 10.7, str. 252/

10.2.3. Metode pristupa

Načini pristupa uskladištenim podacima se zovu **metode pristupa.** Najjednostavniji je **sekvencijalni pristup.** Koristi se npr. kod editora i programskih prevodilaca. Drugi metod je **direktan** ili **relativni pristup**. Koristi se npr. kod baza podataka. Postoje i druge metode, kao što je npr. **indeksirani pristup**, tj. pristup zasnovan na primeni indeksa. Kod ovog pristupa za datu datoteku se kreira indeks koji sadrži pokazivače na blokove datoteke. Kod takvih metoda pristupa najpre se pretražuje indeks, a zatim se na osnovu pokazivača direktno pristupa željenom slogu. Pri radu sa velikim datotekama sama indeksna datoteka može biti suviše velika zasmeštanje unutar operativne memorije. Jedino moguće rešenje je da se kreira indeks za indeksnu datoteku. Tada prvi nivo indeksa pokazuje na drugi nivo indeksa, a drugi na stvarne podatke.

Keš memorija omogućava komponentama računara da efikasnije komuniciraju privremenim premeštanjem podataka sa sporijeg uređaja na brži uređaj (keš memorija). Keš memorija je skuplja od uređaja sa kojih se podaci privremeno prebacuju, tako da povećanjem veličine keš memorije cena sistema raste.

10.2.4. Metode dodele prostora na disku

Tri glavne metode za dodelu prostora na disku su: dodela susednih memorijskih lokacija, dodela povezanih blokova fiksne veličine i korišćenje šema sa indeksima.

Metoda dodele susednih memorijskih lokacija. Za skladištenje date datoteke koristi susedne blokove diska. Dodela susednih blokova je jednostavna. Susednost blokova poboljšava performanse. Moguć je i sekvencijalan i direktan pristup podacima. Nedostatak ove metode jeste eksterna fragmentacija. Takođe, nije uvek lako pronaći najbolje mesto za skladištenje nove datoteke. Problem može biti i kada datoteka postaje veća, jer može biti neophodno da se premesti cela datoteka.

Metoda dodele povezanih blokova fiksne veličine. Omogućava skladištenje svih datoteka tako što se koriste blokovi fiksne veličine. Susedni blokovi se povezuju u povezanu listu. Osnovna prednost je da nema eksterne fragmentacije. Upravljanje memorijom je jednostavno jer su svi blokovi iste veličine. Blokovi mogu biti „razbacani" bilo gde na disku. Nedostatak može biti degradacija performansi kod direktnog pristupa, jer je potrebno slediti pokazivače od jednog bloka diska do sledećeg.

Metode koje koriste šeme sa indeksima. One svakoj datoteci pružaju tabelu indeksa. Svaki indeks u tabeli indeksa pokazuje na blokove diska koji sadrže stvarne podatke date datoteke. Slog za svaku datoteku unutar direktorijuma sadrži broj bloka indeksa i ime datoteke. Ovaj način obezbeđuje brz direktan pristup podacima.

10.3. Udaljeni sistemi datoteka

Udaljeni sistemi datoteka (remote file systems) omogućavaju datom računaru da „priključi" (mount), tj. učini dostupnim jedan ili više sistema datoteka sa jednog ili više udaljenih računara. To je klijent-server model. Računar na kome se nalaze datoteke je **server**, a računar sa koga se pristupa udaljenim datotekama je **klijent**. Server može imati više klijenata i klijent može pristupati datotekama koje se nalaze na više servera.

LDAP (Lightweight Directory-Access Protocol) protokol, poznat kao X.509 standard je implementiran u više OS-a. To je podskup hijerarhijskog protokola X.500 za distribuirane direktorijume. LDAP je jednostavan protokol za pretraživanje podataka.

10.4. Algoritmi za dodelu diska

Algoritmi koji se koriste za dodelu diska su: FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK.

FCFS (First Come First Served). Najjednostavniji algoritam za dodelu diska. Izvršava zahteve za dodelu diska onim redosledom kojim su zahtevi nastali i ne postoji mogućnost promene redosleda ni mogućnost „gladovanja", jer se svaki zahtev servisira. Posebno je pogodan za jednokorisničke sisteme.

SSTF (Shortest Seek Time First). U datom trenutku vrši pokretanje glave diska do najbliže staze diska uzevši u obzir zadate U/I zahteve. Osnovna ideja je da se minimizira vreme pozicioniranja u odnosu na trenutnu poziciju glave diska. SSTF algoritam smanjuje ukupno vreme pozicioniranja u poređenju sa FCFS algoritmom. Nedostatak je mogućnost „gladovanja", jer se mogu stalno pojavljivati zahtevi koji u odnosu na dati zahtev imaju manje vreme pozicioniranja. SSTF nije optimalan algoritam.

SCAN algoritam. Razrešava problem „gladovanja" koji se može javiti kod SSTF algoritma. Problem razrešava tako što nakon pokretanja glave diska dalje pokretanje se nastavlja u istom smeru, tj. najpre se servisiraju zahtevi počev od najdalje spoljašnje staze na krajnjjoj unutrašnjoj, a zatim od najbliže unutrašnje ka najdaljoj spoljašnjoj. Glava diska stalno skenira disk sa jednog kraja na drugi.

C-SCAN algoritam. Nastao je sa idejom da se otkloni nedostatak SCAN algoritma. On obezbeđuje uniformnija vremena odziva tako što se pokretanje glava diska uvek vrši u jednom smeru, odnosno servisiraju se svi zahtevi. Funkcioniše tako kao da su najudaljenija i najbliža staza od centra diska logički međusobno susedne.

LOOK algoritam. On modifikuje SCAN algoritam tako što se zaustavlja kretanje glave u istom smeru ako nema više U/I zahteva u tom smeru.

C-LOOK algoritam. Servisira zahteve pomeranjem glave diska od spoljašnjeg ka unutrašnjem cilindru dokle god ima zahteva, a zatim se vraća na spoljašni najudaljeniji cilindar za koji postoji zahtev.

12. ZAŠTITA

12.1. Problem zaštite sistema

Zaštita se primenjuje:

1. na nivou mreže,
2. na nivou operativnog sistema,
3. na nivou aplikacije,
4. na nivou baze podataka,
5. kao proceduralna zaštita.

OS može da obezbedi sledeće ***načine zaštite***:

- bez zaštite (kada se delovi koda sa kritičnim sekcijama izvršavaju u različito vreme),
- izolaciju (svaki proces se izvršava nezavisno od drugih procesa bez deljenja resursa i bez međusobne komunikacije),
- sve je deljivo ili nema deljenja resursa (vlasnik objekta deklariše objekat kao javni ili privatni),
- deljenje preko ograničenja pristupa (kada OS obezbeđuje da samo autorizovani korisnik može da pristupi datom objektu),
- deljenje preko dinamičkih sposobnosti (kada se koncept kontrole pristupa proširuje tako da omogući dinamičko kreiranje prava za deljenje objekata),
- ograničeno korišćenje objekata (kada se ograničava ne samo pristup datom objektu, već i operacije koje se mogu vršiti nad objektom).

12.2. Model zaštite sistema

Sistem zaštite se sastoji od skupa objekata, skupa subjekata i skupa pravila kojima se definiše politika zaštite. Politika zaštite određuje pravila zaštite. Dozvola za izvršavanje operacije nad datim objektom zove se ***pravo pristupa***. Svako polje u matrici definiše skup prava pristupa između odgovarajućeh subjekta i objekta. Mehanizmi zaštite određuju *kako* će nešto biti urađeno, a politika zaštite odlučuje *šta* će biti urađeno. Razdvajanje mehanizama i politike zaštite je važno za fleksibilnost sistema zaštite.

Opšti model zaštite /slika 12.1, str. 317/

Primer matrice pristupa /tabela 12.1, str. 318/

Domen zaštite se može realizovati na različite načine. Npr. svaki korisnik može biti domen i u tom slučaju promena domena nastaje prilikom promene korisnika (primenjivano kod UNIX OS-a). **Druga** mogućnost je da svaki proces može biti domen - tada sadržaj domena zavisi od identiteta procesa. **Treća** mogućnost je da svaka procedura može biti domen - promena domena nastaje prilikom poziva procedure.

Vezivanje procesa i domena zaštite može biti statičko i dinamičko. ***Statičko vezivanje*** je ukoliko je skup raspoloživih resursa fiksan, tj. nepromenljiv u toku životnog ciklusa procesa, a ***dinamičko vezivanje*** kada je skup raspoloživih resursa promenljiv u vremenu.

12.3. Zahtevi sistema zaštite savremenih sistema

Kod današnjih sistema postoje sledeći ***zahtevi sistema zaštite***:

1. međusobna autentikacija,
2. kontrola pristupa ili autorizacija,
3. zaštićena komunikacija,
4. neporicanje slanja, odnosno prijema podataka,
5. neponavljanje slanja,
6. nema odbijanja servisa.

12.4. Vrste napada

Kada je zaštita sistema narušena tada postoje sledeće mogućnosti:

- prekid toka informacija,
- modifikacija informacija,
- presretanje informacija,
- fabrikacija informacija.

Napade na operativne sisteme možemo podeliti u nekoliko grupa:

1. virusi, 2. crvi (worms), 3. Trojanski konji.

12.4.1. Virusi

Računarski virus je program koji se može sam reprodukovati tako što dodaje sopstveni kod nekom drugom programu. Rezultat izvršavanja virusa može biti ispisivanje neke poruke na ekranu, prikaz slike, modifikovanje ili brisanje neke datoteke, poziv nekog telefonskog broja, itd, tj. virus može uraditi sve što i računarski program može.

Ciljevi autora virusa su: brzo širenje virusa, da se virus teško detektuje i da odbrana od virusa bude što komplikovanija.

Najpoznatiji tipovi virusa su:

- parazitni (uvek se nalaze kao deo nekog izvršnog programa),
- stalno prisutni u operativnoj memoriji (obično su deo nekog sistemskog programa i inficiraju sve programe koji se izvršavaju),
- boot sektor (inficiraju glavni boot slog i šire se nakon podizanja sistema sa diska),
- stealth (posebno su projektovani da budu nevidljivi prilikom detekcije od strane antivirusnih softvera),
- polimorfni (virusi koji mutiraju prilikom svakog inficiranja, čime ostvaruju da detekcija na osnovu karakterističnog uzorka virusa nije moguća).

12.4.2. Crvi

Crv je računarski program koji kopira samog sebe sa jednog računara na drugi. Mogu se koristiti za prenošenje virusa ili za zamenu postojećih datoteka verzijama datoteka koje predstavljaju Trojanske konje. Obično se crvi prenose preko računarske mreže koristeći nedostatke operativnih sistema u pogledu zaštite. Brzo se repliciraju i troše puno memorije na „host" računarima. Za razliku od virusa, za aktiviranje crva nije potreban korisnik i oni skrivaju svoje širenje na druge računare.

12.4.3. Trojanski konji

Trojanski konj je program koji se najčešće prenosi na ciljni računar kao nevidljivi dodatak uz neki drugi program, a zatim dolazi to njegovog aktiviranja. Mogu se preneti kopiranjem programa, download-ovanjem sa Interneta, kao i otvaranjem attachment-a u e-mailu. Aktivnosti Trojanskih konja mogu biti veoma različite. Mogu da brišu podatke, pošalju svoju kopiju na sve računare sa liste e-mail adresa i da omoguće dodatne napade na dati računar. Obišno ostvaruju povezivanje sa nekim udaljenim računarom i prenose informacije sa računara na kome su instalirani. Najopasniji su oni koji se izvršavaju u okviru kernela.

12.4.4. Napadi u distribuiranim sistemima

Postoje ***4 kategorije napada*** kod distribuiranih sistema:

1. prekid normalnog toka informacija,
2. presretanje ili prisluškivanje informacija,
3. modifikacija informacija,
4. fabrikacija informacija.

12.5. Mehanizmi zaštite

Zaštita sistema može biti ***fizička*** kao svuda neophodan mehanizam zaštite, može biti dovoljna za zaštitu nekih sistema koji nisu povezani na druge računarske mreže. Podrazumeva se obezbeđenje zgrada, prostorija i na neki način kontrole identiteta korisnika sistema. Tradicionalan mehanizam zaštite je ***sistem lozinki*** kojim se verifikuje identitet korisnika i na taj način isključuje mogućnost rada neidentifikovanih korisnika.

U cilju bolje zaštite potrebno je da OS podrži jake lozinke, tj.:

- kontroliše da lozinke budu reči koje se ne mogu naći u rečniku,
- kontroliše da lozinke budu reči najmanje 6 karaktera dužine,
- kontroliše da lozinke budu sastavljene i od slova i od cifara,
- obezbedi da lozinke imaju period važenja,
- ograniči broj pokušaja prijavljivanja na sistem pogrešnom lozinkom.

Sistem zaštite koji se zasniva na lozinkama može se ***narušiti*** pogađanjem lozinki. To može biti ako napadač poznaje datog korisnika ili ima informacije o njemu. Drugi način je metodom „grube sile", gde se korišćenjem današnjih računara veoma brzo može pretražiti kompletan skup mogućih lozinki čija je maksimalna dužina unapred poznata. Sistem zaštite se može narušiti kao rezultat vizuelnog ili elektronskog monitoringa. Kriptovanje podataka koji sadrže lozinku rešava ovaj problem.

Mehanizam zaštite je i ***uvođenje prava pristupa*** jer u suprotnom svaki autorizovani korisnik bi mogao da pristupi svim podacima. Ovaj mehanizam ima ***slabosti*** jer prava pristupa ostaju nepromenjena u toku životnog ciklusa procesa. Ponekad je potrebno pristupiti datom resursu samo jednom, npr. u toku inicijalizacije. Međutim, prava pristupa ostaju i dalje nakon inicijalizacije iako nisu potrebna i to predstavlja potencijalni sigurnosni propust.

Mehanizam zaštite sa najfinijom granularnošću zaštite se zasniva na pravu pristupa samo uz treću aktivnost.

12.6. Principi projektovanja zaštite

To su:
1. princip zabrane,
2. princip kompletnosti,
3. princip najmanje privilegije,
4. princip otvorenosti,
5. princip bezbednog kernela,
6. princip jednostavnosti,
7. princip višeniviske zaštite.

Need-to-know princip zaštite - po njemu dati proces u bilo kom trenutku može da pristupi samo onim resursima za koje ima autorizovan pristup i koji su mu neophodni za završetak posla koji trenutno izvršava.

12.7. Zaštita u distribuiranim sistemima

Ti mehanizmi zaštite su: autentikacija, kriptografija i digitalni potpis.

12.7.1. Autentikacija

Sve metode autentikacije zasnivaju se na ***3 osnovna principa***:
1. nešto što korisnik zna
2. nešto što korisnik ima
3. nešto što korisnik jeste.

12.7.2. Kriptografija

Kriptografija je naučna disciplina koja se bavi izučavanjem matematičkih metoda i tehnika vezanih za različite aspekte informacione sigurnosti, kao što su tajnost ili poverljivost, integritet podataka, autentičnost entiteta, tj. izvora i odredišta podataka.

Prikaz komunikacije 2 korisnika preko nesigurnog komunikacionog kanala korišćenjem kriptografije /slika 12.6, str. 331/

U opštem slučaju ***postoje 2 grupe kriptografskih algoritama***: algoritmi sa simetričnim ključevima i algoritmi sa asimetričnim ključevima.

Blok šema sistema za rad sa simetričnim algoritmima /slika 12.7, str. 332/

Najpoznatiji *predstavnik simetričnih algoritama* danas je 3DES algoritam, koji je nastao iz DES algoritma koji je implemetiran još 1977. godine kao standard za siguran prenos podataka.

Najpoznatiji *predstavnih asimetričnih algoritama* je RSA. Rezultat je rada Ronalda L Rivest-a, Adi Shamir-a i Leonarda Adleman-a na prestižnom američkom univerzitetu MIT.

Blok šema sistema za rad sa asimetričnim algoritmima /slika 12.8, str. 334/

12.7.3. Digitalni potpis

Digitalni potpis garantuje autentičnost poslate poruke na mestu prijema i očuvanje integriteta, tj. da u toku prenosa nije došlo do promene sadržaja poruke. Takođe, garantuje nemogućnost poricanja slanja, odnosno prijema date poruke.

MD5 algoritam generiše 16-bajtni rezultat, a SHA generiše 20-bajtni rezultat.

PREDAVANJA

Tipovi izvršnih jedinica u IA-64.

I-jedinica (za celobrojnu aritmetiku, logičke instrukcije, instrukcije poređenja, ...)
M-jedinica (učitava i skladišti podatke između registra i memorije)
B-jedinica (instrukcije grananja)
F-jedinica (instrukcije za rad u pokretnom zarezu)

Spekulativno upravljanje kod IA-64.

Spekulativno upravljanje kod IA-64 omogućava procesoru da podatke iz memorije učita pre nego što programu budu potrebne da bi se izbeglo kašnjenje spore memorije.

Najvažniji flag-ovi:
1. carry flag (označava prenos/pozajmljivanje za MSB)
2. zero flag (postavlja se na 1 ako je rezultat 0)
3. sign flag (postavlja se na vrednost MSB koji označava bit znaka za označene cele brojeve)
4. overflow flag (postavlja se na 1 kada dođe do prekoračenja rezultata)

Magistrale koje se koriste kod PC računara: lokalna, cache, memorijska, pci, scsi, usb, ide, isa.
PCI magistrala služi za povezivanje sa brzim uređajima kao što su monitori, mrežne kartice, itd.
Cache magistrala - koristi se za komunikaciju procesora sa cache memorijom van chipa, tj. sa level 2 cache-om.
ISA magistrala - za povezivanje sporih uređaja (štampač, zvučna kartica, modem, itd.)

Razlika između spoljašnjih i unutrašnjih prekida. Unutrašnji prekidi su prekidi koje generiše procesor - kao rezultat nastanka greške prilikom izvršavanja tekuće instrukcije ili kao rezultat izvršavanja instrukcije prekida int. Spoljašnji prekidi su prekidi koje generišu kontroleri periferijskih uređaja kada su uređaji spremni za prenos podataka ili kontrolni uređaji koji kontrolišu ispravnost rada memorije, magistrale ili npr. napajanja.

Kontekst procesora. Procesor obuhvata: sadržaj brojača naredbi, sadržaj statusnog registra (PSW), sadržaj registara podataka, sadržaj adresnih registara, sadržaj indeksnih registara, sadržaj baznih registara, sadržaj registara opšte namene. Kontekst procesora se najčešće čuva na steku.

Koji se registar kod Intelovih Pentium procesora (IA-32) koristi kao:
- akumulator - EAX,
- I/O pokazivač - EDX,
- brojač naredbi - EIP,

Prednosti load/store arhitekture. Jednostavno enkodiranje instrukcija fiksne dužine (pojednostavljeno dekodiranje) i sličan broj ciklusa po instrukciji.