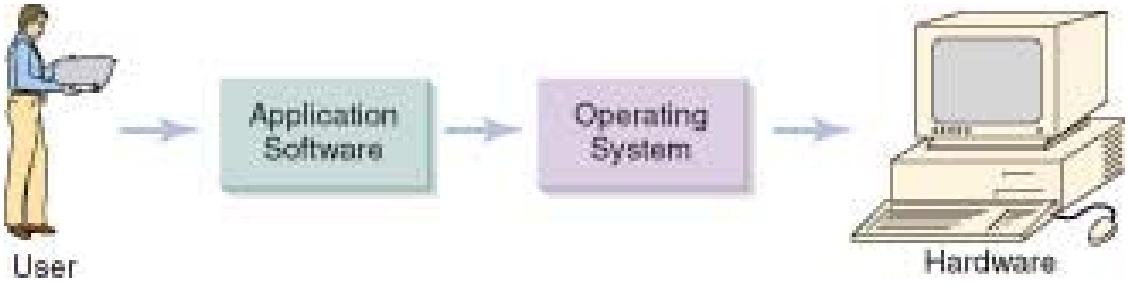
Operacijski sistemi

O operacijskih sistemih

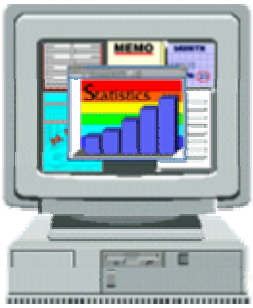
Definicija

Operacijski sistem je program ali bolje množica programov, ki računalniku ponujajo kontrolne in podporne funkcije in mu tako omogočajo nalaganje, nadzor in komunikacijo z aplikacijsko programsko opremo.

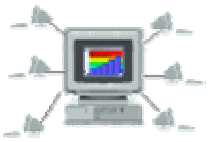


Vrste operacijskih sistemov

Starejši operacijski sistemi so bili pogosto **enoopravilni**. Omogočali so sočasno izvajanje le enega programa (bolje procesa). Če smo hoteli izvajati več programov, smo morali med njimi ročno preklapljati.



Večopravilni (multitasking) operacijski sistemi dovoljujejo (navidezno) sočasno izvajanje več programov. V resnici računalnik izmenično dodeljuje posameznim programom (bolje procesom) časovne rezine. Največ časovnih rezin običajno dobiva program (proces), ki deluje v ospredju (foreground). Manj rezin dobivajo programi, ki delujejo v ozadju (background). Najmanj časa je posvečeno programom, ki trenutno (še) ne delajo nič).



Mnogouporabniški sistemi dovoljujejo uporabo istega računalnika, včasih celo istega programa, večim uporabnikom. Taki sistemi pogosto delujejo po principu delitve časa med uporabniki (**time sharing**)

Večprocesorski sistemi temeljijo na uporabi več procesnih enot (CPE). Te so dodeljevane posameznim programskim procesom. Hitrost izvajanja takih proccesov je(lahko) večja (ni pa nujno).

Nekateri sistemi lahko dopuščajo (sočasno) uporabo **več operacijskih sistemov** z uporabo koncepta navideznih strojev (**virtual machines**).

Posebej omenimo sistemsko programsko opremo omrežnih računalnikov. Osnovna zamisel omrežnih računalnikov je, da imajo sami po sebi le minimalno programsko opremo in v bistvu tako operacijski system kot uslužnostne in aplikativne programme naložijo iz nekega centralnega računalnika. Omrežni računalniki torej sami po sebi ne morejo zadovoljivo delovati. Zato pa je njihova fleksibilnost precej večja.

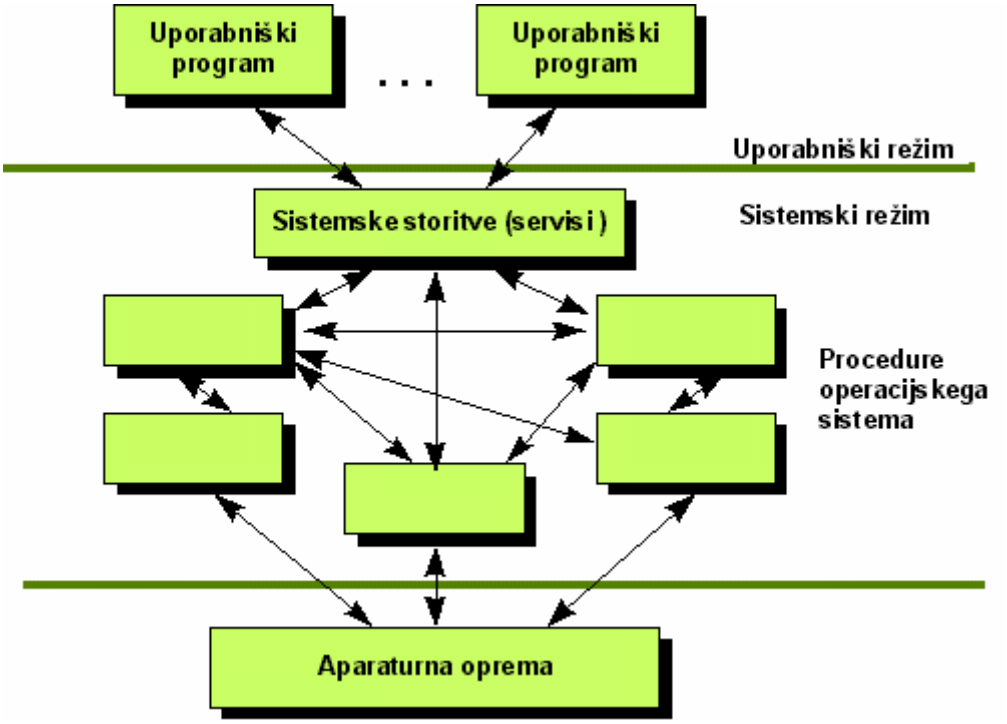
Struktura operacijskega sistema

Operacijski sistem zagotavlja programom okolje in storitve, potrebne za njihovo izvajanje. Sestavlja ga več sistemskih komponent:

- Upravljanje procesov,
- Upravljanje s primarnim pomnilnikom,
- Upravljanje s sekundarnim pomnilnikom,
- Upravljanje z vhodno-izhodnimi napravami,
- Upravljanje z datotečnim sistemom,
- Sistem zaščite (dostop programov, procesov, uporabnikov, ugotavljanje napak),
- Sodelovanje v mreži (porazdeljeni sistemi),
- Interpretiranje ukazov.

Vmesnik med procesi in operacijskim sistemom predstavljajo sistemski klici. Z njimi zahtevajo (uporabniški) programi neko storitev operacijskega sistema. Po drugi strani pa lahko pride do prekinitev izvajanja uporabniških programov tudi s strani operacijskega sistema.

Splošno sliko strukture operacijskega sistema prikazuje spodnja slika:



Operacijski sistem interaktira neposredno z aparaturno opremo, programom pa zagotavlja storitve tako, da so le-ti izolirani od aparaturne opreme. Na celoten sistem lahko gledamo kot na množico plasti. Aplikacijski programi predstavljajo zunanje plasti, sam operacijski sistem pa predstavlja sistemsko jedro (kernel). Aplikacijski programi lahko zahtevajo od jedra izvedbo posameznih storitev s takoimenovanimi sistemskimi klici.

Praviloma pozna računalnik dva režima izvajanja nekega procesa: uporabniški režim (user mode) in sistemski režim (kernel mode, system mode). Normalno so naši programi v uporabniškem režimu. Šele ko to zahtevamo s primernim sistemskim klicem, preide proces v sistemski režim. V takem stanju izvedejo rutine v jedru zahtevano storitev in nato vrnejo klicočemu programu kodo s statusom izvedbe (error code).Proces se povrne v uporabniški režim.

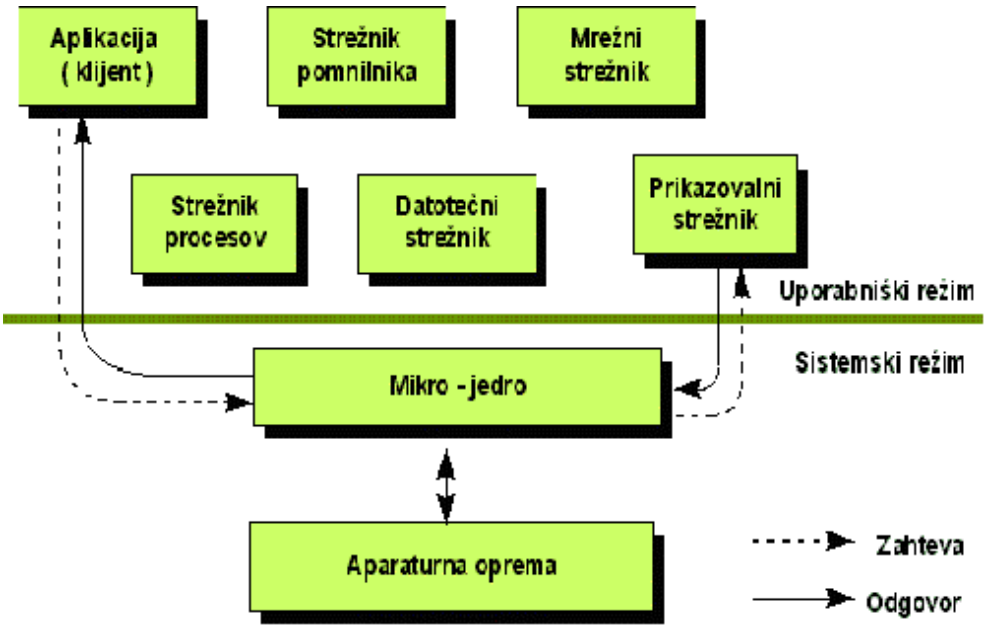
Proces lahko preide iz uporabniškega v sistemski nivo tudi zaradi izjemnega dogodka (exception), kakršnega predstavlja na primer poskus izvedbe nelegalne instrukcije. Tako v primeru nepredvidenega dogodka kot tudi v primeru namernega sistemskega klica pride do takoimenovane programske prekinitve (ali pasti), ki povzroci preklop v sistemski režim in klic ustrezne servisne rutine jedra.

Do prekinitve programa lahko pride tudi zaradi aparaturne prekinitve (zahtevek z neke periferne naprave). Tudi v tem primeru pride do preklopa iz uporabniškega režima v sistemski (če se v tem še ne nahajamo) in do izvedbe ustrezne prekinitvene rutine.

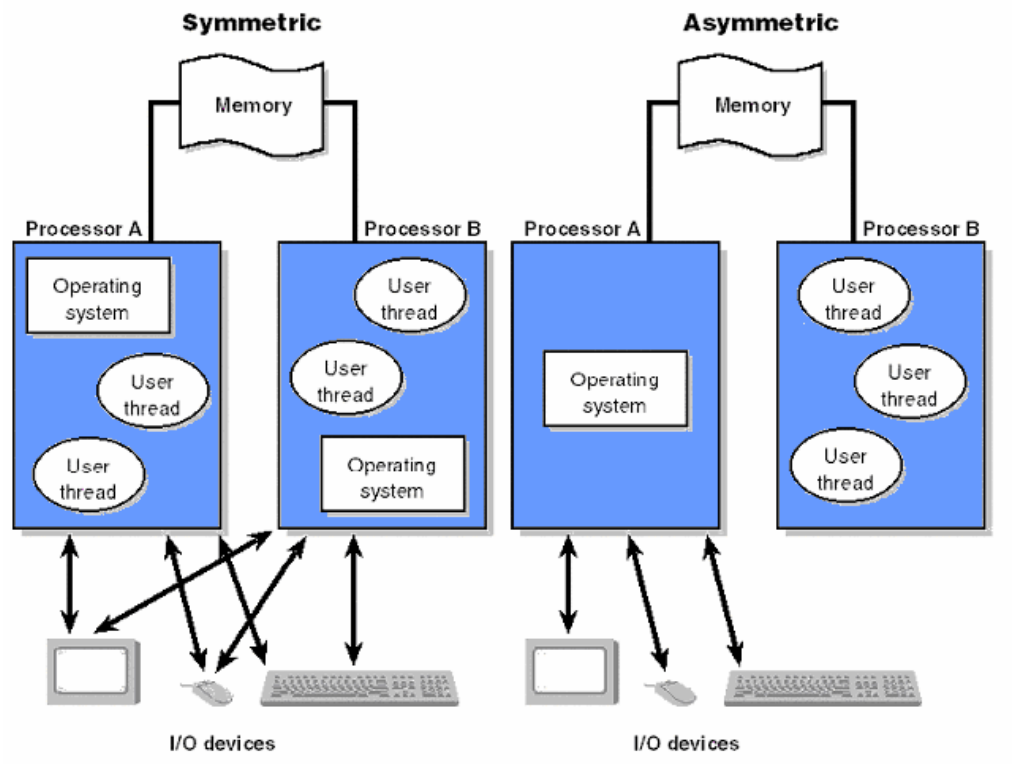
Ko je proces v sistemskem režimu, lahko pride do preklopa na drug proces. Pri takem preklopu se mora seveda ohraniti stanje (prekinjenega) procesa tako, da se bo kasneje lahko nemoteno nadaljeval. Pravimo, da ima proces nek kontekst in da pride do preklopa konteksta (context switch).

V sodobnejših operacijskih sistemih zasledimo organiziranost operacijskega sistema po konceptih klijent- strežnik (client-server). Tu so komponente operacijskega sistema majhne in samostojne. Vsak strežnik teče kot poseben proces v uporabniškem režimu. Jedro skrbi le za

komunikacijo med takimi strežniki (message passing). Govorimo o mikro-jedru. Tak sistem je bolj varen pred izpadi in bolj fleksibilen. Njegov koncept prikazuje spodnja slika:



V obdobju večprocesorskih sistemov ločimo med simetričnimi (SMP) in asimetričnimi modeli. V prvem primeru nimamo nekega "glavnega" procesorja, pri asimetričnih modelih pa operacijski sistem uporablja en procesor, aplikacije pa druge. Primer simetričnega operacijskega sistema je Windows 2000.



Ukazni interpreter

Ko uporabnik sodeluje z operacijskim sistemo, to opravlja preko vmesnika imenovanega ukazni interpreter. Ta odgovarja na uporabnikove zahteve na naslednje načine:

- Zažene aplikacijo.
- Ustavi aplikacijo.
- Omogoči uporabniku preklop med aplikacijami.
- Lahko omogoči nadzor nad komunikacijo med aplikacijo in ostalimi aplikacijami ali uporabniki.

Ukazni interpreter je lahko zasnovan kot konzola v tekstovnem načinu delovanja, ali pa v obliki grafičnega vmesnika za zajem ukazov.

Potrebno je razlikovati med interpretejem in operacijskim sistemom. Interpreter ponavadi uporablja samo majhno podmnožico ukazov, ki jih ponuja OS.

Zagon operacijskega sistema

Zagnati računalnik pomeni, naložiti operacijski sistem v delovni pomnilnik računalnika. Ko je OS enkrat naložen, je pripravljen za zagon uporabniških programov. Če morate kdaj slučajno ponovno zagnati računalnik to preprosto pomeni, da je potrebno ponovno naložiti OS.

Zagon in nalaganje OS ni enak postopek kot je namestitev OS. Ko nameščamo OS, moramo prehodit določene namestitvene izbire, v povezavi z konfiguracijo računalnika. Ko z nameščanjem zaključimo, se OS nahaja na trdem disku in je pripravljen za nalaganje v delovni pomnilnik. Tipično je OS nameščen tako, da se ob vklopu avtomatsko naloži in zažene. Če med delom pride do problemov zaradi katerih računalnik obstoji, to rešujemo s ponovnim nalaganjem operacijskega sistema v delovni pomnilnik.

Kako nalaganje deluje

Nalaganje se razlikuje od sistema do sistema, generalna ideja pa je povsod podobna.

Takoj ko vklopite računalnik Basic Input-Output System BIOS ki se nahaja v ROM pomnilniku prevzame nadzor. Prva stvar ki jo naredi je Power-On Self Test POST, kjer preveri, ali vse komponente računalnika delujejo. Zatem poišče posebni zagonski program, ki bo dejansko naložil OS. Prvo pogleda v pogon A: (če seveda ni drugače nastavljeno) na posebno mesto, kjer se nahaja zagonski program. Če gre za MS-DOS bo tam našel datoteki IO.SYS in MSDOS.SYS. Če je v disketniku disketa, ki ne vsebije OS, bo to javil. Če diskete ni (ponavadi je tako), bo z iskanjem nadaljeval na točno določenem mestu na disku C:. Poiskal bo prvi sektor (512 bytov) in ga prekopiral na točno določeno mesto v RAMu (naslov 7C00$). Ta sektor se imenuje Master Boot Record MBR.

Boot record vsebuje programček, ki sedaj dobi nadzor nad sistemom in poskrbi za nalaganje osnovnega sistema v pomnilnik RAM. Za DOS je to IO.SYS. Ta program pa potem poskrbi za nalaganje celotnega OS v pomnilnik.

Ena prvih datotek, ki se bo naložila je sistemska konfiguracijska datoteka (za DOS CONFIG.SYS), ki pove katere specifične datoteke se morajo naložit polek splošnih (gonilniki).

Naslednja specifična datoteka, ki se bo naložila je datoteka, ki pove, katere aplikacije ali ukaze želi uporabnik vključit v proces nalaganja (DOS – AUTOEXEC.BAT WIN – WIN.INI)

Ko se je OS uspešno naložil, se mu preda nadzor nad računalniškim sistemom. OS izvede zahtevana začetna opravila, nato pa čaka na ukaze s strani uporabnika.

Pojem procesa

Programski proces je pravzaprav odvijanje nekega programa skupaj z okoljem, ki je za to odvijanje potrebno.

V to okolje sodijo predvsem potrebne programske strukture, kot je njegova koda, podatkovne strukture in sklad. V to okolje sodijo tudi vsi ostala potrebnasredstva (resources). Tako potrebujemo za omenjene programske strukture zadostno količino pomnilnika, dostop do perifernih naprav itd.

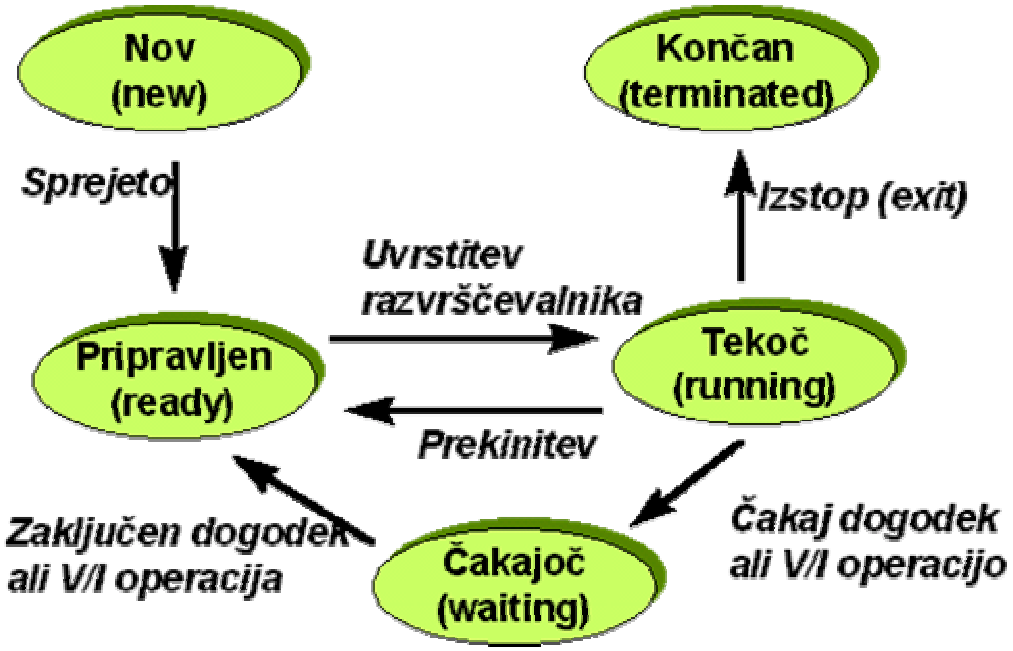
Procesi so paralelni, če obstojajo istočasno. Paralelni procesi so lahko drug od drugega neodvisni, lahko pa so asinhroni, kar pomeni, da zahtevajo občasno sinhronizacijo in sodelovanje. Interakcija asinhronih programov je včasih zelo kompleksna.

Pri obravnavi paralelnih procesov zasledimo vrsto zanimivih pojavov in pred programerjem takih (programskih) procesov se pojavijo dodatni problemi, na kakršne ni naletel niti pri zelo kompliciranih klasičnih programskih sistemih.

Tako si lahko posamezni programi med seboj konkurirajo za sredstva (resources), ki so v njihovem "delovnem" okolju na voljo. Pravimo, da so taki procesi konkurenčni. Preprost zgled za tako konkurenčnost so na primer sočasne zahteve po dostopu do diska v mnogouporabniškem računalniškem sistemu.

Pogosto se pojavi potreba po medsebojni časovni koordinaciji posameznih paralelnih procesov. In ne nazadnje, Posamezni procesi se morajo pogosto med seboj pogovarjati. Govorimo o medprocesni koordinaciji in komunikaciji.

Prav zaradi konkurenčnosti procesov in tudi potreb po nihovi medsebojni sinhronizaciji moramo uvesti pojem stanja procesa. Zaenkrat omenimo le, da je proces, ki mora čakati na nek drug proces, na sprostitev nekega sredstva (resource) ali na nek dogodek, v blokiranem stanju. Poenostavljeno naj velja, da je proces, ki ni blokiran, tekoč. V resnici so pri gradnji posameznih operacijskih sistemov uvedli še vec procesnih stanj. Splošen koncept prehajanja med procesnimi stanji prikazuje naslednja slika.

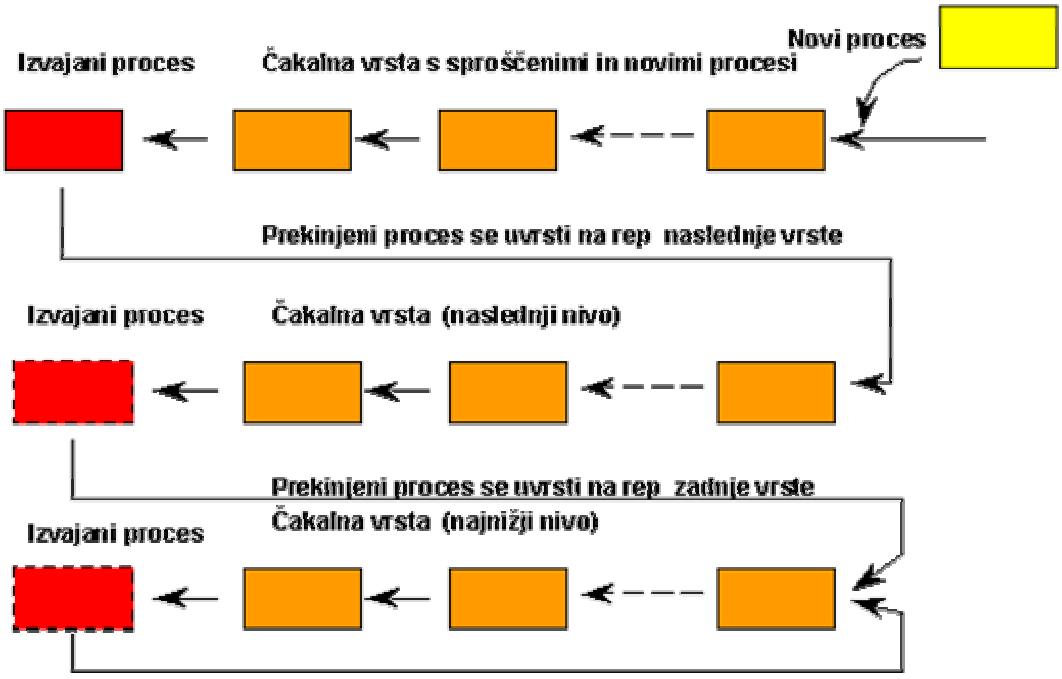


Razvrščanje procesov

Sodobni operacijski procesi so tipično večprocesni oziroma celo večnitni. Zato mora posebni modul, ki mu pravimo razvrščevalnik (scheduler) poskrbeti, da izmenično vsakemu od teh procesov dodeljuje krajšo časovno rezino. Tako imamo vtis, kot da vsi procesi potekajo sočasno.

Eden od algoritmov, ki ga uporabljajo razvrščevalniki, je tako imenovani **Round-Robin,** ki ga prikazuje spodnja slika.

Bolj kompleksni algoritmi razvrščanja skušajo zagotoviti ugodno prepustnost za procese, ki jim zadoščajo kratke časovne rezine. V to skupino sodijo krajši programi in interaktivni programi, ki večinoma čakajo na konec neke vhodno-izhodne operacije. Numerično intenzivnim procesom pa prepuščajo čas, ko računalnik ni zaposlen. Priper takega algoritma je večnivojski algoritem razvrščanja, ki ga prikazuje spodnja slika.



Pri tem algoritmu dodeli razvrščevalnik računalniški čas procesu, ki je prvi na čakalni listi na najvišjem nivoju. Če tak proces ne konča svoje naloge v času njegove časovne rezine, ga razvrščevalnik prestavi na rep čakalne vrste nižjega nivoja. Procesom, ki čakajo na čakalnih vrstah nižjih nivojev, razvrščevalnik dodeli časovno rezino le, če so čakalne vrste višjih nivojev prazne. Vrsta na najnižjem nivoju sama vase predstavlja spet klasični Round Robin algoritem, v njej pa končajo numerično intenzivni procesi.

Medprocesna komunikacija

Medprocesna komunikacija (IPC InterProcess Communication) je način, kako lahko dva ali več procesov medsebojno izmenjuje podatke. Protokol zagotavlja da lahko katerikoli par procesov pošilja ali sprejema sporočila, ukaze in odzive. Pri tem se lahko programa izvajata na istem ali različnem sistemu.

Najsplošnejši protokoli, ki se uporabljajo v ta namen so:

- Imenske cevi (Named pipes)

- Klici oddaljeni procedur (RPC)
- Vtiči (Sockets)

Imenske cevi

To je osnovni IPC protokol, ki je zelo podoben pisanju v datoteko. Razlika je samo v tem, da se piše v datoteko imenska cev, do katere lahko dostopajo različni procesi istočasno

Klici oddaljeni procedur

Protokol izhaja iz UNIXa in omogoča komunikacijo z procedurami na oddaljenih računalnikih. Potrebujemo posebno RPC knjižnico, ki prestreže lokalne klice, poišče oddaljeno proceduro, vzpostavi povezavo in izvrši komunikacijo.

Vtiči

Ideja vtičev je, da dva procesa, ki želita komunicirat vzpostavita par vtičev in na ta način kreirata komunikacijski kanal med njima.

Ta mehanizem je najmočnejši znotraj IPC protokolov, saj omogoča povezavo procesov, ki tečejo na istem računalniku ali pa na različnih sistemih. Uporaba vtičev pri komunikaciji ponavadi sledi modelu klijent/strežnik.

Niti

Zamislimo si, da mora biti na primer nek datotečni strežni proces blokiran zaradi čakanja na disk. Če bi tak strežni proces imel več niti, bi med spanjem ene niti lahko potekala druga. Propustnost sistema bi bila večja. Tega ne moremo doseči z dvema (ali več) strežnima procesoma, ker si morata deliti isti medpomnilnik (v istem naslovnem prostoru). Uvedli so torej nov mehanizem.

Imejmo računalnik z nekaj procesi. Vsak proces si lasti programski števec in delovne registre, ima svoj sklad, ima svoj naslovni prostor. Taki procesi bi lahko medsebojno komunicirali le s pomočjo primitivnih medprocesnih mehanizmov. Uvedimo pojem "peresno lahkih" procesov, ki jim pravimo niti. Vsaka ima še vedno "svoje" delovne registre in svoj sklad, kar zadošča za informacijo o statusu takega mini-procesa. Uporabljajo pa take niti isti naslovni prostor in s tem tudi iste globalne spremenljivke. Medsebojne zaščite med nitmi ni, saj praviloma pripadajo vse take niti istemu "uporabniku" oziroma procesu. Niti si zato delijo tudi vse odprte datoteke, signale, semaforje in celo otroke, ki jih je generiral proces (lahko pa imajo tudi svoje otroke). Lastno vsaki niti je tudi njeno stanje, ki je lahko tekoče, blokirano, pripravljeno ali zaključeno (running, blocked, ready, terminated).

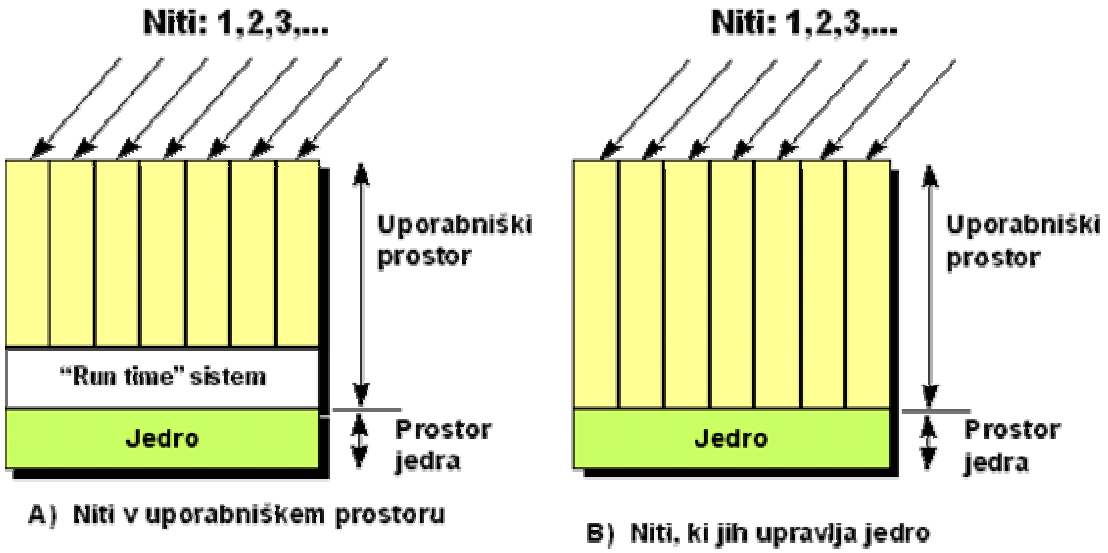
Niti tako omogočajo kombinacijo paralelnosti v sicer sekvenčnem izvajanju (in blokiranju) procesov v sistemskih klicih.

Implementacija niti

Uporabniku morajo biti na voljo primerni primitivi (na primer funkcijski klici) za delo z nitmi. Niti lahko realiziramo statično ali dinamično. V prvem primeru je število niti v nekem procesu določeno v fazi prevajanja. Vsaki niti je določen ločen sklad. Tak pristop je preprost a nefleksibilen. V drugem primeru niti nastajajo in se ukinjajo med izvajanjem programa. S primernim klicem definiramo tako glavni program niti (kazalec na primerno proceduro), kot tudi prostor za sklad, njeno prioriteto razvrščanja itd.

Ker niti delujejo na istih (skupnih) podatkih, se poraja problem kritičnih sekcij. Uporabimo lahko tehniko mutex ali pa sinhronizacijo s pomočjo pogojnih spremenljivk. Muteksi so pravzaprav binarni semaforji z dvema stanjema ((ne)zaklenjen). Pogojne spremenljivke pa so podobne pogojnim spremenljivkam za sinhronizacijo monitorjev. Ključna razlika med obema principoma je, da so muteksi bolj primerni za kratkotrajno zaklepanje, pogojne spremenljivke pa za dolgotrajna čakanja na sprostitev vira.

Primerni paket za delo z nitmi (thread package) je lahko implementiran v uporabniškem ali v sistemskem prostoru. Oboje ponazoruje naslednja slika:



V prvem primeru jedro nič ne ve o nitih, še vedno upravlja enostavne, eno-nitne procese. Tak pristop je primeren predvsem za implementacijo na sistemih, ki nitkanja ne poznajo (primer starejše verzije UNIX). Niti tečejo nad posebnim modulom "run time", ki je pravzaprav zbirka procedur za rokovanje z nitmi. Preklop med nitmi je že na uporabniškem nivoju, kar je vsaj za razred hitrejše, kot če bi bil potreben prehod na nivo jedra. Dodatna prednost tega koncepta je, da ima vsak proces svoj algoritem razvrščanja procedur.

Po drugem principu upravlja z nitmi jedro. Zato ne potrebujemo modula "run-time". Tabele za delo z nitmi se nahajajo v jedru. Preklop med nitmi je precej bolj zahteven, saj blokiranje neke niti dosežemo le z bolj zahtevnim sistemskim klicem. Zato pa lahko jedro izbere za

nadaljevanje poljubno nit (ne nujno od istega procesa). Implementacija paketov za delo z nitmi na uporabniškem nivoju je sicer bolj učinkovita, ima pa druge probleme. Nit v uporabniškem prostoru ne bi smela uporabljati sistemskih klicev, saj to povzroči zaustavitev vseh niti danega procesa. Druga slabost tega pristopa je tudi, da je delo drugih niti (v istem procesu) pogojeno s "prostovoljnim" preklopom trenutno izbrane niti (podobno kot pri korutinah). če niti krmili jedro, je enakopravnost vseh niti zagotovljena z razvrščevalnikom.

Uvod v porazdeljene sisteme

Porazdeljeni sistemi

S pojavom predvsem lokalnih računalniških mrež so se pojavile tendence po vzpostavitvi računalniškega sistema, ki bi lahko izkoriščal večje število CPE, ki jih povezujejo hitra omrežja. Programska oprema takega, decentraliziranega sistema pa je nujno drugačna. Lahko predstavlja mehko povezavo med sistemi (loosely coupled software), ki dopušča praktično avtonomno delovanje posameznih računalnikov, povezanih v mrežo. Na drugem ekstremu pa imamo tesno sklopljene multiprocesorske sisteme.

Naslednja stopnja v razvoju porazdeljenih operacijskih sistemov je bil koncept, ki naj uporabniku (še vedno enake aparaturne infrastrukture) ustvarja iluzijo, da dela pravzaprav za enim sistemom (virtual uniprocesor). To je zahtevalo predvsem predelavo sistemskih klicev. Jedra posameznih računalnikov morajo sodelovati, pri čemer mora vsako jedro še vedno skrbeti za lokalne vire (upravljanje s pomnilnikom ipd).

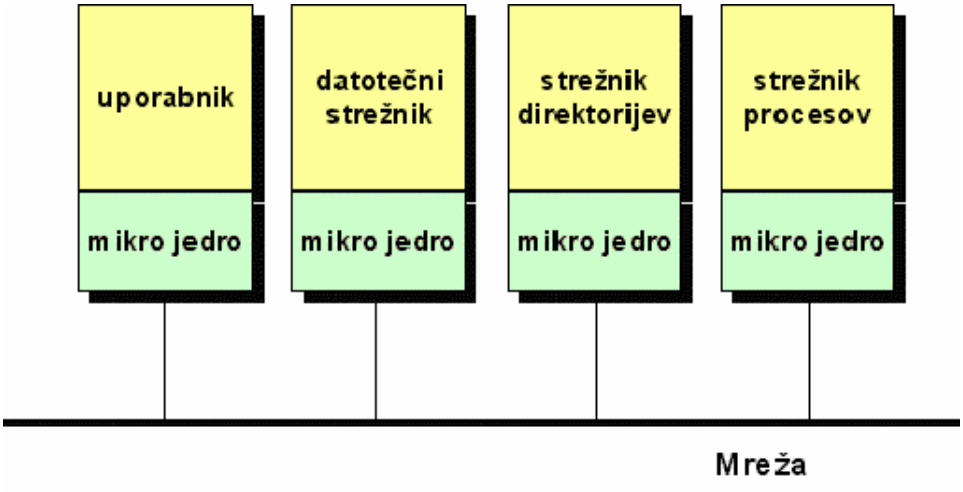
Pri načrtovanju porazdeljenih sistemov je bilo potrebno najti odgovore na naslednja vprašanja:

- Transparentnost: Uporabnik ne ve, kje so locirani viri, ne ve, koliko kopij obstaja, Več uporabnikov morda uporablja iste vire, Aktivnosti se lahko dogajajo paralelno.
- Fleksibilnost: Preprosto dodajanje in spreminjanje porazdeljenega sistema.
- Zanesljivost
- Učinkovitost

Uvedli so pojem mikro-jedra (microkernel), ki je bolj fleksibilno, saj ne dela skoraj nič;. Prepuščeni so mu le nekateri najbolj osnovni servisi:

- Medprocesna komunikacija,
- Delno opravljanje s pomnilnikom,
- Delno upravljanje s procesi,
- Nizkonivojske vhodno-izhodne operacije

Ostale storitve, kot je na primer upravljanje z datoteč;nim sistemom, so prepuščene ustreznim strežnikom, kot to ponazoruje naslednja slika:



Splošno vodilo pri gradnji porazdeljenih operacijskih sistemov je tudi uporaba porazdeljenih podatkovnih struktur (na primer raznih tabel). Tudi algoritmi morajo biti decentralizirani. To pa pomeni, da:

- Noben računalnik nima popolne informacije o stanju celotnega sistema,
- Računalniki sprejemajo odločitve na osnovi lokalnih informacij,
- Izpad enega stroja ne blokira algoritma
- Ni globalne "ure" (takta) in s tem sinhronizacije.

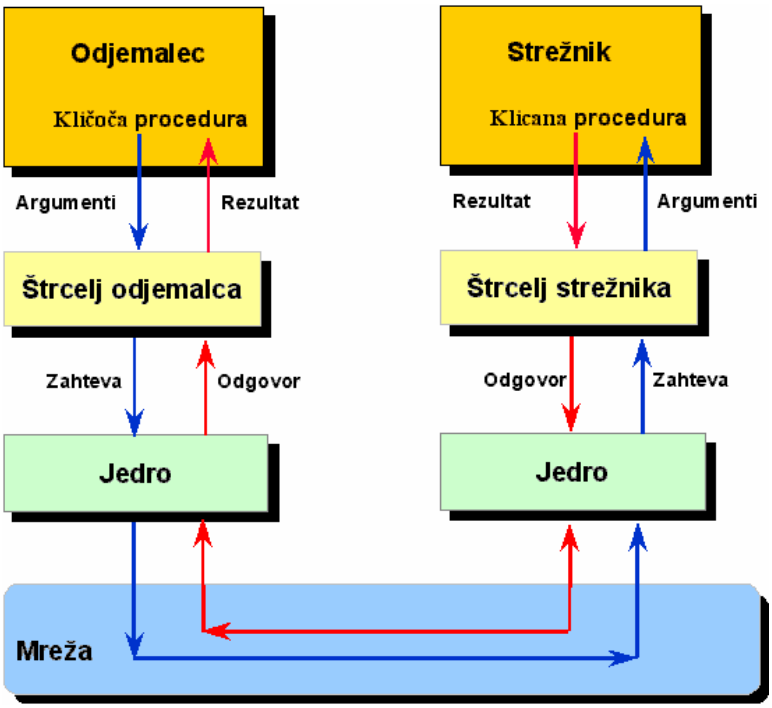
Komunikacija v porazdeljenih sistemih

Klic oddaljenih procedur

Pri uvedbi pojma klica oddaljenih procedur (**RPC**, remote procedure call) je potrebno predvsem razumeti delovanje klasičnih sistemskih klicev. Za primer vzemimo sistemski klic ***read( )***, ki ima naslednjo splošno obliko:

```
count = read (fd, buffer, numBytes);
```

Kot je znano, pri takem klicu naloži kličoča funkcija parametre na sklad, vključno s povratnim naslovom (return address). Če so parametri kazalci, pomeni, da bo klicana funkcija spreminjala podatke naslovnem prostoru kličoče funkcije (oziroma programskega procesa). Pri povratku iz klicane funkcije se kličoča funkcija lahko nadaljuje zaradi restavracije povratnega naslova, ki je bil na skladu. Vse se dogaja v istem naslovnem prostoru danega procesa. To pa ne more veljati v primeru, ko so klicane funkcije locirane na drugem računalniku. Želimo pa, da so tudi taki klici čimbolj podobni klasičnim. Uporabimo lahko idejo "klica oddaljene procedure" (RPC), ki je ponazorjena s spodnjo sliko:



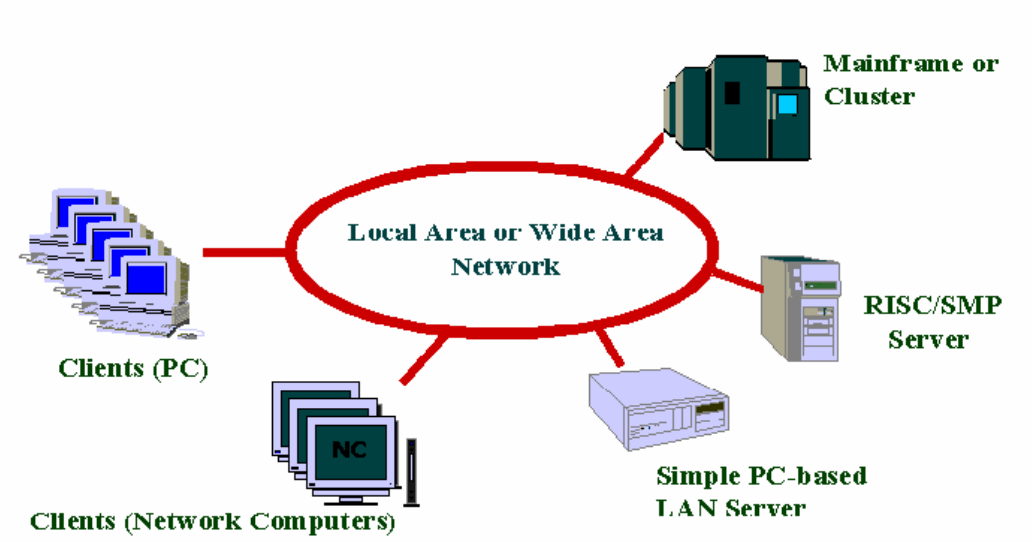
Vzemimo za primer klic oddaljene funkcije *read( )*. Pri centraliziranih sistemih bi klic potekal v okviru istega procesa in s tem istega naslovnega prostora. Pri decentraliziranem sistemu potrebujemo na oddaljenem računalniku dodaten proces, strežnik. Oddaljena procedura *read( )* bo delovala v naslovnem prostoru strežnika. V naslovnem prostoru kličočega procesa (odjemalca) moramo imeti na voljo malo drugačno proceduro, ki ji rečemo štrcelj odjemalca (client stub), ki igra vlogo nekakąnega vmesnika. Analogno proceduro, štrcelj strežnika (server stub) imamo na strani strežnika. Oddaljen klic procedure poteka nato na naslednji način:

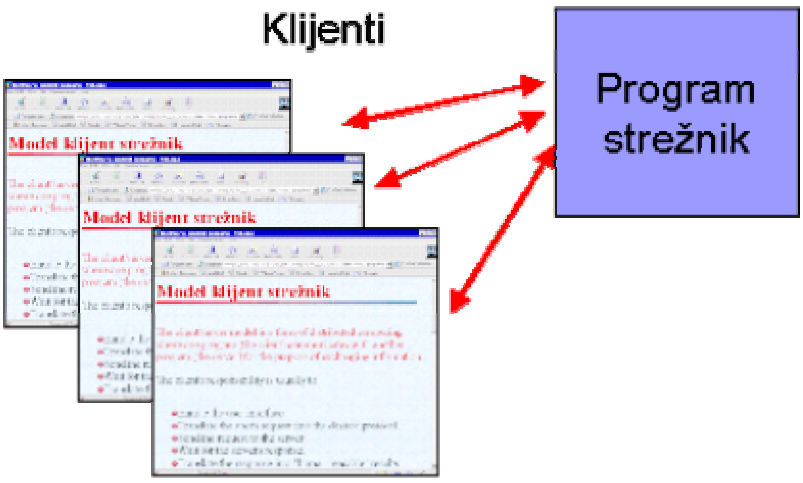
- Proces-odjemalec pokliče svoj štrcelj na normalni način (prenos parametrov in povratnega naslova na lastni sklad).
- Štrcelj odjemalca formira obvestilo (message) tako, da vanj pakira parametre in s primernim sistemskim klicem pokliče jedro.
- Jedro pošlje obvestilo oddaljenemu jedru.
- Oddaljeno jedro posreduje obvestilo štrclju strežnika.
- Štrcelj strežnika razpakira parametre v sprejetem obvestilu in kliče strežnik.
- Strežnik opravi zahtevano nalogo in vrne rezultat svojemu štrclju.
- Štrcelj strežnika zapakira rezultate v obvestilo in s sistemskim klicom pade v (svoje, oddaljeno) jedro.
- Oddaljeno jedro posreduje obvestilo z rezultati jedru odjemalca.
- Jedro odjemalca pošlje obvestilo štrclju odjemalca.
- Štrcelj odjemalca razpakira rezultat in ga vrne odjemalcu.

Pakiranju parametrov v obvestilo pravijo po angleško marshalling (urejanje, pakiranje), obratni operaciji pa unmarshalling (razpakiranje).

Koncept klijent - strežnik

Klijent – strežnik je princip porazdeljenega procesiranja, kjer so podatki shranjeni centralno na nekem strežniku, uporabniki pa do njih dostopajo preko različnih klijentov.





Model klijent-strežnik je model porazdeljenega procesiranja kjer program klijent komunicira s programom strežnik iz razloga izmenjave informacij.

Naloge klijenta so:

- Izvedba uporabniškega vmesnika.
- Prevod želja uporabnika v ustrezen protokol.
- Poslati zahtevo strežniku.
- Počakat na odziv strežnika.
- Pretvorba dobljeni rezultatov v format, ki je razumljiv uporabniku.

Funkcije, ki jih zagotavlja strežnik:

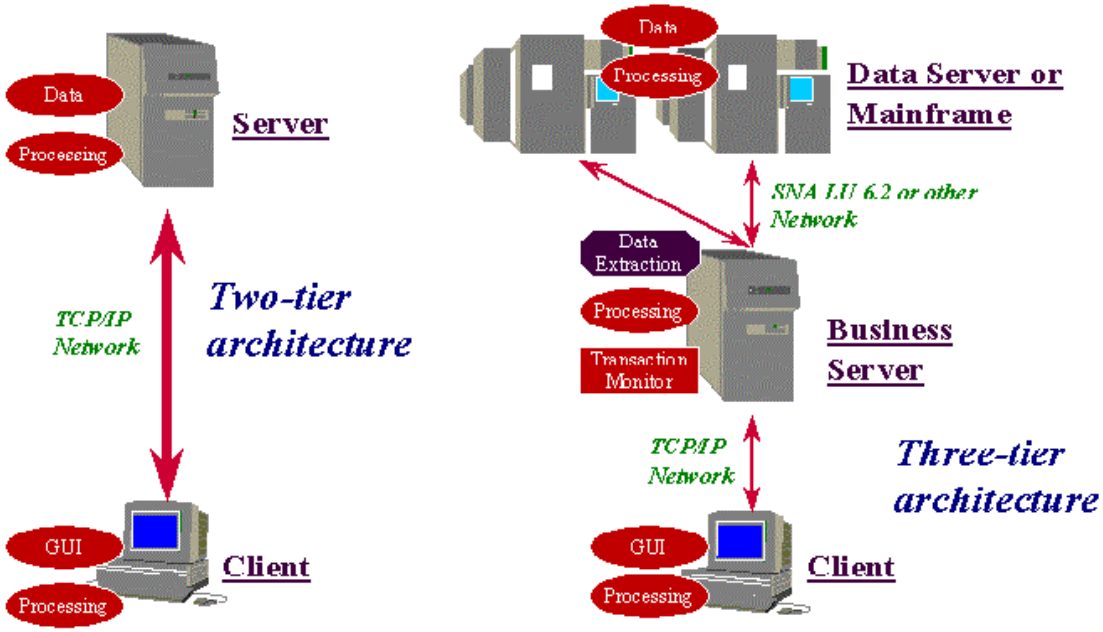
- Čakanje na klijetovo zahtevo (poizvedbo).
- Izvršitev zahtevane poizvedbe.
- Vrnitev rezultatov klijentu.

Potek tipične iterakcije med klijentom in strežnikom:

- Uporabnik s pomočjo aplikacije klijenta ustvari poizvedbo.
- Klijent se priključi na strežnik.
- Pošlje poizvedbo strežniku.
- Stežik analizira poizvedbo.
- Strežnik pridobi rezultate poizvedbe.
- Dobljene rezultate pošlje nazaj klijentu.
- Klijent dobljene rezultate prezentira uporabniku.
- Če je potrebno se zadeva ponovi.

Najvidnejša prednost takšne arhitekture je prilagodljivost uporabniškega vmesnika. Omogoča namreč kreiranje vmesnikov, ki so neodvisni od strežnika, ki vsebuje podatke. Na strani klijentov lahko imamo različne računalnike, ki jih združimo s strežnikom in nanje namestimo zmogljivejše informacije. Ker je uporabniški vmesnik domena klijentov je s tem strežnik razbremenjen in se lahko bolj temeljito posveti analizi poizved in izvrševanju le teh. Tako lahko kot strežnik uporabimo manj zmogliv računalnik kot bi ga sicer.
Na kratko povedano, arhitektura klijent-strežnik predstavlja mehanizem za povezavo različnih računalnikov v skupino, ki opravlja isto opravilo.

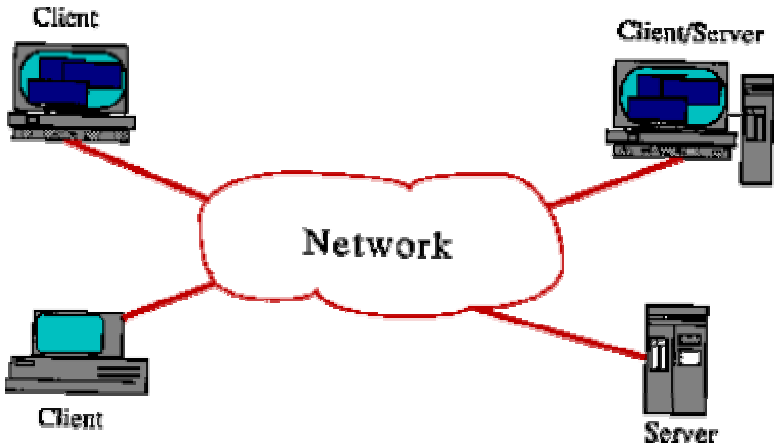
Arhitekture klijent strežnik



Heterogeni sistemi

V heterogenih sistemih je lahko posamezno omrežno vozlišče (računalnik) karakterizirano kot klijent, strežnik ali pa klijent/strežnik.

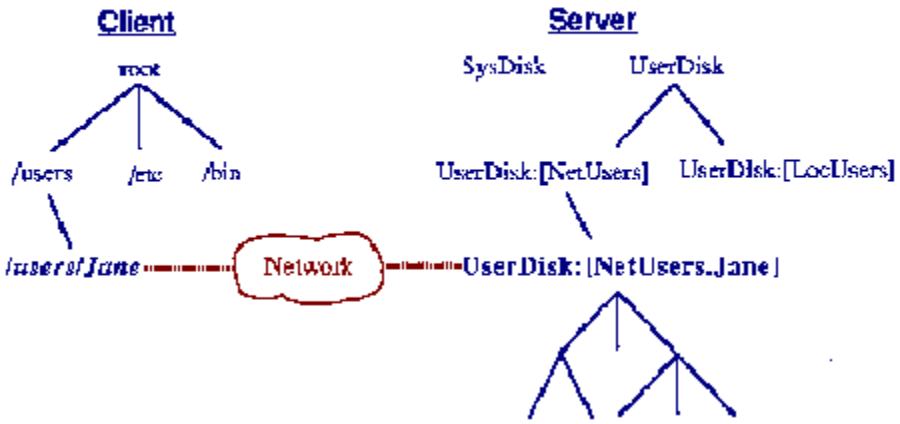
Vozlišče klijent ostalim vozliščem omrežju ne nudi nobenih storitev. Sposoben je samo koristiti storitve, ki jih ponujajo vozlišča. Storitve ponujajo vozlišča, ki jih imenujemo strežniki. Pogosto pa je vozlišče strežnik lahko tudi klijent. Takšnim vozliščem pravimo klijent/strežnik.



Na slikci sta PC in delovna postaja klijenta, ki ne ponujata nobenih storitev ostalim vozliščem, lahko pa koristita storitve, ki jih ponujajo ostala vozlišča. Strežnik samo nudi svoje storitve, klijent/strežnik pa se obnaša kot klijent in strežnik istočasno.

V osnovi so se v vozliščih nahajali veliki mainframe ali mini računalniki. Uporabniki so dostopali do omrežja preko terminalov, ki so bili priklopljeni na določeno vozlišče. Storitve, ki jih je omrežje ponujalo so bile FTP, email, virtual terminal. Vsako vozlišče je bilo za te tri storitve klijent/strežnik.

Z razvojem PC računalnikov se je spremenila tudi struktura omrežij. Sedaj so bili v vozliščih PC računalniki, ki so bili ali klijenti ali strežniki. Ni se več potrebno prijavljati v vozlišča, ker se aplikacije poganjajo lokano na PC računalnikih. Prenos datotek je naprimer rešen z kopiranjem datotek iz strežnika in na njega nazaj. Pošta se pošilja preko poštnega strežnika.



Slika: Dostop klijenta do pomnilnega prostora na strežniku.

Razlika med velikimi računalniškimi orežji in omrežji sestavljenimi iz osebnih računalnikov pa je počasi začela izginjat. V heterogenih računalniških okoljih se storitve velikih računalniških mrež in mrež osebnih računalnikov zbližujejo in združujejo v tri osnovne grupe storitev ki so dostopne klijentom vseh velikosti, realizirane pa so na strežnikih ali klijentih/stežnikih vseh velikosti.

Te tri osnovne skupine storitev so:

- Porazdeljen datotečni sitem,
- porazdeljeno procesiranje,
- prenašanje sporočil.

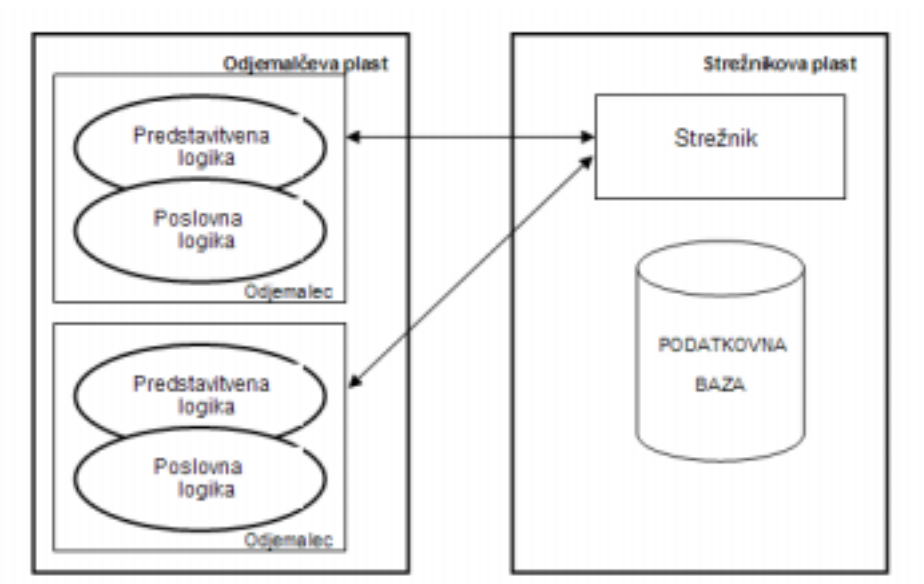
Dvoslojna arhitektura odjemalec / strežnik

Aplikacija je s časom napredovala od enostavnega programa do prvega informacijskega sistema (t.j. sistem programov, ki jih povezuje enoten vir podatkov). Večna želja programerjev pa je bila ponovna uporaba že napisane in stestirane kode. Dele kode z neko zaokroženo funkcionalnostjo se je začelo zbirati v ločenih zbirkah – knjižnicah. Knjižnice je bilo možno uvoziti v kodo in s tem izkoristiti njihovo funkcionalnost. Vendar je bil s tem pospešen samo razvoj aplikacij, programi sami pa ob tem niso bili nič manj razbremenjeni, poleg tega pa je bilo povzročeno veliko tratenje virov. Tako je prišlo v razvoju programiranja do novega izziva: logika, skupna vsem programom, naj se izvaja na enem mestu, programi pa naj do nje dostopajo preko omrežnih povezav. S tem program razpade na dve plasti – plast odjemalca in plast strežnika.

Funkcionalnost aplikacije lahko na logičnem nivoju razdelimo na :

- Predstavitvene storitve: te skrbijo za prikaz uporabniškega vmesnika in za posredovanje vpisanih podatkov.
- Poslovne storitve: skrbijo za apliciranje poslovnih pravil in logike, kamor prištevamo skrb za pravilni vrstni red proženja, usmerjanja, vrednotenja in procesiranja podatkov v skladu s poslovnimi zahtevami.
- Podatkovne storitve: zagotavljajo neposredni dostop do podatkov in integriteto podatkov vezano na sistem za upravljanje s podatkovno bazo.

Na fizičnem nivoju je bila ta razdelitev izvedena v dveh plasteh – odjemalčevi plasti in strežnikovi plast, kot je prikazano na sliki.



Slika: Aplikacija odjemalec strežnik

Namizni računalniki so postajali vedno bolj zmogljivi in razvijalci so nanje selili del kode iz velikih strežnikov. Ta trend se je nadaljeval tako daleč, da so veliki računalniki skrbeli le še za upravljanje podatkov in so se na njih izvajali sistemi za upravljanje s podatki (DBMS). Namizni računalniki pa so izvajali funkcije uporabniškega vmesnika in večino poslovne logike, velike sisteme v ozadju pa so izrabljali le za shranjevanje podatkov. S tem pa je prihajalo do vedno večjih problemov, kot na primer:

- razvijali so se sistemi, kjer je bil uporabniški vmesnik neločljivo povezan s poslovno logiko, oziroma lahko bi rekli, da je bila poslovna logika integrirana v uporabniški vmesnik. Spremembe v poslovni logiki ali uporabniškem vmesniku so bile v tem primeru zelo naporne (spremembo je bilo potrebno napraviti na vseh mestih v kodi)

- z večanjem števila odjemalcev je bilo potrebno instalirati nove verzije programov na vse računalnike, kar je bilo povezano s časom in velikimi stroški (posebej, ker nismo več omejeni na odjemalce, ki bi bili fizično blizu)
- ob prenosu večje količine podatkov (npr. večjih appletov) postanejo komunikacijske poti ozko grlo

Prednost tega načina pa je bila robustna uporaba strežniškega okolja z dobrimi servisi, prav tako pa je bilo enostavno tudi upravljanje sistema varnosti.

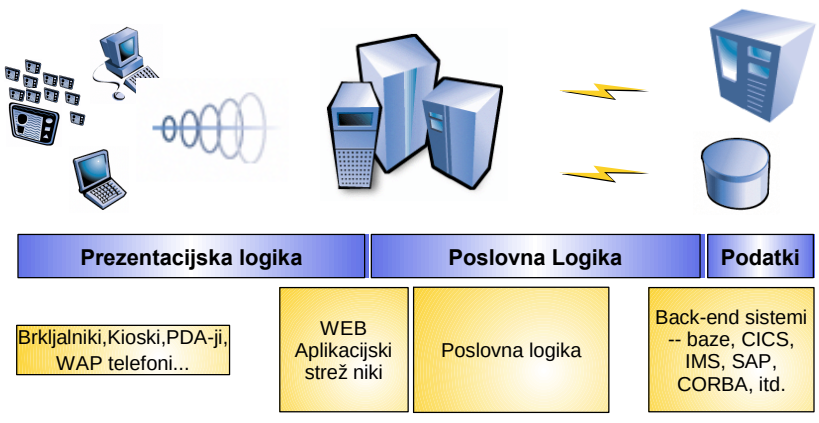
V dobi nove ekonomije bo potrebna popolna preobrazba programske opreme, kajti programska oprema bo postala storitev in ne več izdelek. Poleg tega so zaprti cikli, v katerih nastajajo nove različice programov, prepočasni za svet nove ekonomije.

Pri razvoju programske opreme, ki je delovala po načelu odjemalca in strežnika, je bilo značilno, da je programiranje nove različice programa vzelo leto ali dve časa, medtem pa so bili uporabniki prisiljeni uporabljati trenutno verzijo. Zdaj se razvojni cikel spreminja v nenehen proces, v katerem so nadgradnje vsakdanji, a za uporabnika povsem nemoteč pojav, programska oprema pa je razslojena v zamenljive enote in porazdeljena na številne naprave.

Takšna programska oprema ne more temeljiti niti na podatkovnem niti na spletnem strežniku, seveda pa tudi ne sme biti odvisna od posameznega brskalnika, saj je pred nami kup novih odjemalcev, ki bodo imeli do elektronskih storitev dostop na povsem nove načine. Da bo takšna programska oprema učinkovito delovala, je potrebna trdoživa infrastruktura z visoko zmogljivim in poljubno razširljivim programskim strežnikom.

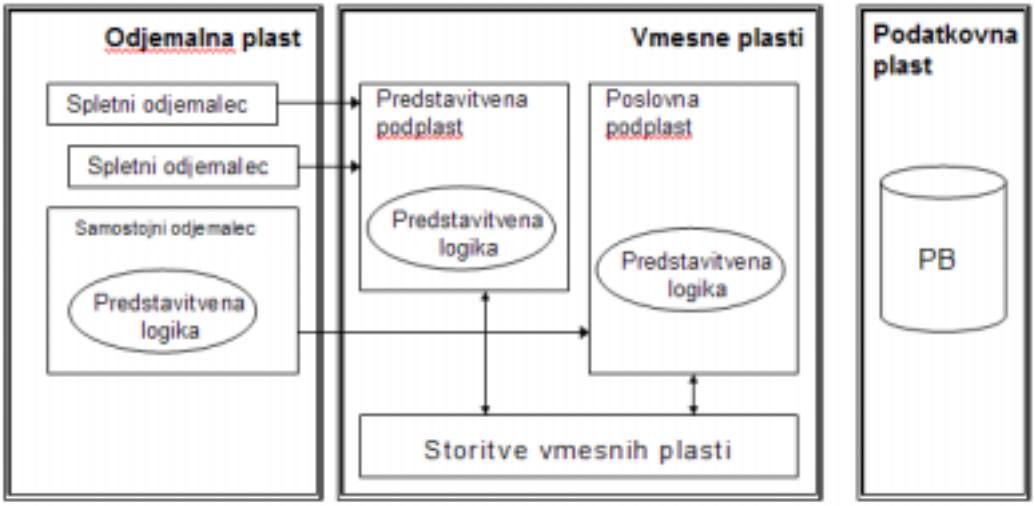
Troslojna / večslojna arhitektura

Zaradi razširitve poslovanja na Internet postaja dvoplasten model tipa odjemalec – strežnik vedno manj primeren. Kompleksnost razpošiljanja informacijskih storitev do vsakega uporabnika in administrativni problemi, ki jih povzroča nameščanje ter vzdrževanje poslovne logike na vsakem uporabnikovem računalniku, kažejo na neprimernost tega modela. Primernejši je večplastni model aplikacij, kjer se ločuje prezentacijska in poslovna logika ter podatkovna plast



Slika: Moderna tri plastna arhitektura

Med implementacijami večplastnega modela je najpogostejši štiri-plastni model, kjer se vmesna plast razdeli na predstavitveno in poslovno podplast. Vsaka povezava med plastmi je v bistvu povezava odjemalec – strežnik.



Slika: Štiri plastna arhitektura

Predstavitvena podplast je namenjena abstrakciji odjemalca – deluje kot posrednik med plastjo odjemalca in poslovno podplastjo tako da prilagaja vsebino podatkov različnim tipom odjemalcev.

Med odjemalca, ki skrbi za prikaz podatkov, in uporabniški vmesnik ter tradicionalni strežnik, ki skrbi za zbirke podatkov ali transakcijske sisteme, se tako vrine **aplikacijski strežnik**, ki prevzame nase glavnino bremena pri preračunavanju poslovnih operacij. V tem času spada tržišče aplikacijskih strežnikov med najhitreje rastoče segmente trga programske opreme.

Vsa poslovna pravila so realizirana na aplikacijskem strežniku. Čeprav različni aplikacijski strežniki nudijo različne možnosti, je vsem skupno to, da so vsi viri dostopni kot objekti, torej močan poudarek je na povezavi z objektnimi modeli EJB in Corba ter DCOM.

Storitve aplikacijskega strežnika lahko razdelimo na osnovne in razširjene. Med osnovne storitve tako prištevamo nadzor nad pravilnostjo delovanja aplikacije ob primeru sočasnega dostopa večjega števila uporabnikov, dostop do DBMS sistemov, upravljanje z vsebino procesov, upravljanje z dinamičnimi spletnimi stranmi; med razširjene pa upravljanje z predpomnilniškim sistemom, razporejanje bremena, nadzor nad razpoložljivostjo sistema itd. Z razpoložljivostjo označujemo zmožnost aplikacije za toleriranje napak, ki se zgodijo pri virih strežnika. Le te so lahko programske ali strojne narave. S stališča aplikacije je najpomembnejši mehanizem v okviru te točke transakcijsko izvajanje. S stališča odjemalca je priporočena uporaba načela »ponovi ob neuspehu« in zagotavljanja alternativ v primeru napake.

V sklopu aplikacijskega strežnika se nahaja tudi **spletni strežnik**, ki skrbi za upravljanje z dinamičnimi spletnimi stranmi. Na začetku razvoja je bil spletni strežnik le preprost program,

katerega naloga je bila le pošiljanje zahtevanih statičnih spletnih vsebin. Kasneje so se pojavili spletni strežniki z mehanizmi za komunikacijo z drugimi programi na strežniku in pojavila se je možnost izdelave dinamičnih spletnih strani, kjer je vsebina odvisna od vnesenih parametrov (tehnologija CGI). Slabost te rešitve pa je bila v tem, da vsaka zahteva po dinamični vsebini zajema kreiranje novega procesa, v katerega je bila naložena skripta, oziroma program, kar pa je povzročalo nepotrebno tratenje sistemskih virov. Rešitev je torej procesiranje zahteve na samem strežniku. Ta sedaj pridobi nadzor nad izvajajočimi procesi in lahko učinkovito razpolaga s sistemskimi viri. Tehnologije, ki pa to omogočajo so Active Server Pages (ASP), servleti in strani JSP.

Upravljanje s primarnim pomnilnikom

Asociacija naslovov

Programi so normalno shranjeni na nekem disku v binarni izvršljivi obliki. Pred izvedbo ustreznega procesa morajo biti (vsaj deloma) naloženi v pomnilnik računalnika. Skupina procesov, ki čaka na prepis v pomnilnik, tvori nekakšno čakalno vrsto.

Večina sistemov omogoča, da procesi leže kjerkoli v fizičnem pomnilniku. Čeprav se naslovni prostor računalnikovega pomnilnika začenja z naslovom 00000, je normalno prvi naslov uporabniškega procesa na drugače alociran. Preden postane nek program izvršljiv, mora preiti več korakov. Naslovi v programu so pri tem različno (v začetku lahko povsem simbolično) predstavljeni. Ti naslovi se nato transformirajo običajno v relativne naslove (relocatable addresses). Tako kodiran program je premestljiv glede na fizični pomnilnik. Program z absolutnimi naslovi ni tako fleksibilen.

Tipično lahko pride do asociacije (binding) naslovov programskih instrukcij in podatkov s pomnilniškimi naslovi v enem od naslednjih korakov:

Čas prevajanja (compile time): Absolutno programsko kodo tvorimo le, če vnaprej vemo, kje v pomnilniku bo moral ležati program oziroma njegovi podatki. Če želimo kasneje spremeniti lokacijo programa, ga moramo ponovno prevesti.

Čas nalaganja (load time): Prevajalnik je moral tvoriti relativno (premestljivo) programsko kodo. Če se spremeni začetni naslov, moramo program ponovno naložiti.

Čas izvajanja (execution time): Proces lahko premikamo iz enega v drug pomniloniški segment celo med izvajanjem. Asociacijo naslovov v času izvajanja programa omogoča aparaturna oprema.

Naslovu, ki ga glede na programski proces pripravi CPE računalnika, pravimo logični naslov. Naslov, kot ga vidi pomnilnik, pa je fizični naslov. Pri asociaciji naslovov v času prevajanja in času nalaganja so logični naslovi enaki fizičnim, pri asociaciji v času izvajanja pa se ti naslovi običajno razlikujejo. Logičnemu naslovu tedaj pravimo tudi virtualni naslov.

Dinamično nalaganje in povezovanje

Dinamično nalaganje programov zagotavlja boljše izkoriščanje pomnilnika. Neko rutino naložimo v pomnilnik šele, ko jo potrebujemo. Prednost takega koncepta je, da programske module, ki jih ne potrebujemo, niti ne naložimo v pomnilnik.

Koncept dinamičnega povezovanja (dynamic linking) je podoben diamičnemu nalaganju. Namesto, da prestavimo v čas izvajanja nalaganje, prestavimo v ta čas kar povezovanje (na primer knjižnjic). Zato binarna (izvršljiva) kopija programa ne potrebuje že navezanih sistemskih rutin. Binarni zapisi programov na disku so zato krajši. Namesto potrebnih sistemskih rutin imajo taki zapisi le ustrezne štrclje (stubs). Tak štrcelj je kratek košček programske kode, s katero proces locira zahtevano rutino v pomnilniku (če se ta že nahaja v pomnilniku), po potrebi pa zahteva njeno nalaganje s knjižnjice na disku. Ta koncept tudi omogoča, da vsi procesi uporabljajo isto kopijo sistemske rutine v pomnilniku.

Dinamično povezovanje (sistemskih) knjižnjic ima še eno prednost.Ko dobimo novo verzijo takih knjižnjic, ni nujna ponovna predelava (povezovanje) uporabniških programov, saj bodo avtomatsko uporabljali novo verzijo. Da pa ne pride do nekompatibilne uporabe novih verzij zagotavlja informacija o zahtevani verziji, ki se nahaja tako v štrclju kot v knjižnjici rutin. Starejši programi bodo pač še vedno uporabljali starejše verzije knjižnjic. Takemu sistemu pravimo sistem skupnih knjižnjic (shared libraries).

Virtualni pomnilnik

Koncept virtualnega pomnilnika temelji na dejstvu, da normalno ne potrebujemo celotnega programa v primarnem pomnilniku. Virtualni pomnilnik tako temelji na abstrakciji uporabniškega logičnega pomnilniškega prostora (ki je lahko velik) od fizičnega pomnilnika (ki je normalno precej manjši).. Programiranje je zato bolj preprosto, programi so lahko precej bolj obsežni.

Ne bomo se spoščali v same tehnike ostranjevanja pomnilnika. Navedimo le algoritme, ki jih lahko uporabljamo pri odločitvah o zamenjavi strani:

FIFO algoritem: Je najbolj preprost. Ko potrebujemo novo stran, izberemo v ta namen tisto ki je "najstarejša". Ta algoritem ni nujno najbolj optimalen. Tako lahko res zamenjamo stran, v kateri je tekel nek inicializacijski (in ne več potreben) modul. Lahko pa je v njej neka pogosto uporabljana podatkovna struktura.

Optimalni algoritem: Naj bi zamenjal tisto stran, ki povzroča najmanj izpadov strani (page faults). V praksi pa ga ne moremo implementirati, saj ne moremo vnaprej poznati, katere strani bomo potrebovali.

LRU algoritem: Je približek optimalnega algoritma. Za vsako stran vodimo evidenco, kdaj je bila nazadnje uporabljena. Ko potrebujemo novo stran, jo zamenjamo s tisto, ki je že najdlje nismo uporabili (Last Recently Used). Pogosto si pri tem pomagamo s takoimenovanim referenčnim bitom (za vsako stran), ki se setira ob vsakem naslavljanju te strani, Sistem pa ga periodično resetira. Poznamo več izpeljank tega algoritma.

Števni algoritmi: Temeljijo na tem, da za vsako stran štejemo število referenc te strani (v danem časovnem intervalu). Imamo lahko več strategij. Tako imamo lahko za kriterij, da zamenjamo tisto stran, ki je bila najmanj pogosto uporabljena (LFU, Last Frequently Used). Argument za tako izbiro je, da so najbolj aktivne strani tiste z največ referencami. Inverzen je algoritem MFU (Most Frequently Used), ki pravi, da take strani ne smemo (takoj) zamenjati, saj morda še nima dovolj referenc, ker je bila vstavljena mred kratkim.

Sekundarni pomnilnik

Disk je najbolj pogost sekundarni pomnilni medij. Kljub njegovi velikosti je optimizacija njegovega prostora zelo pomembna. Diskovne bloke lahko alociramo datotekam na tri različne načine:

- Zaporedno alociranje
- Povezano alociranje
- Indeksirano alociranje

Zaporedno alociranje(Contiguous Allocation): Pri tej metodi pomnimo podatke datoteke v linearno zaporedje blokov na disku. Datotečni direktorij mora za vsako datoteko vsebovati ime datoteke, številko začetnega bloka in dolžino datoteke.

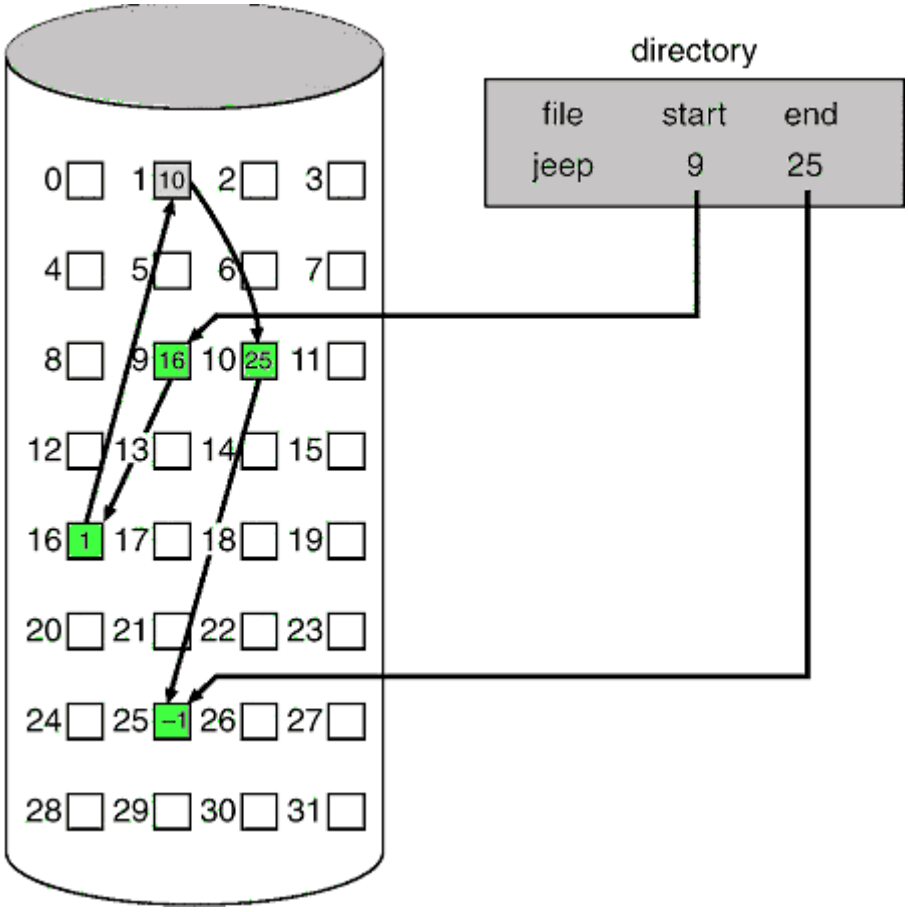
Prednosti:

- Enostavna implementacija
- Tako linearen kot sekvenčni dostop do podatkov sta enako preprosta.

Slabosti:

- Širjenje datotek ni enostavno.
- Tvegamo zunanjo fragmentacijo.
- Problem fragmentacije lahko rešujemo z zgoščevanjem, ki pa je časovno potratno.
- Jedro bi moralo alocirati in rezervirati zaporedne lokacije na disku že ob prvi tvorbi datoteke.

Povezano alociranje (Linked Allocation): Pri tej metodi so podatkovni bloki datoteke razpršeni po disku. Direktorij datoteke vsebuje ime datoteke in številko začetnega bloka. Vsak podatkovni blok pa porabi nekaj bajtov za kazalec na naslednji blok. Kazalec v zadnjem bloku ima posebno vrednost EOF (End Of File).



Prednosti:

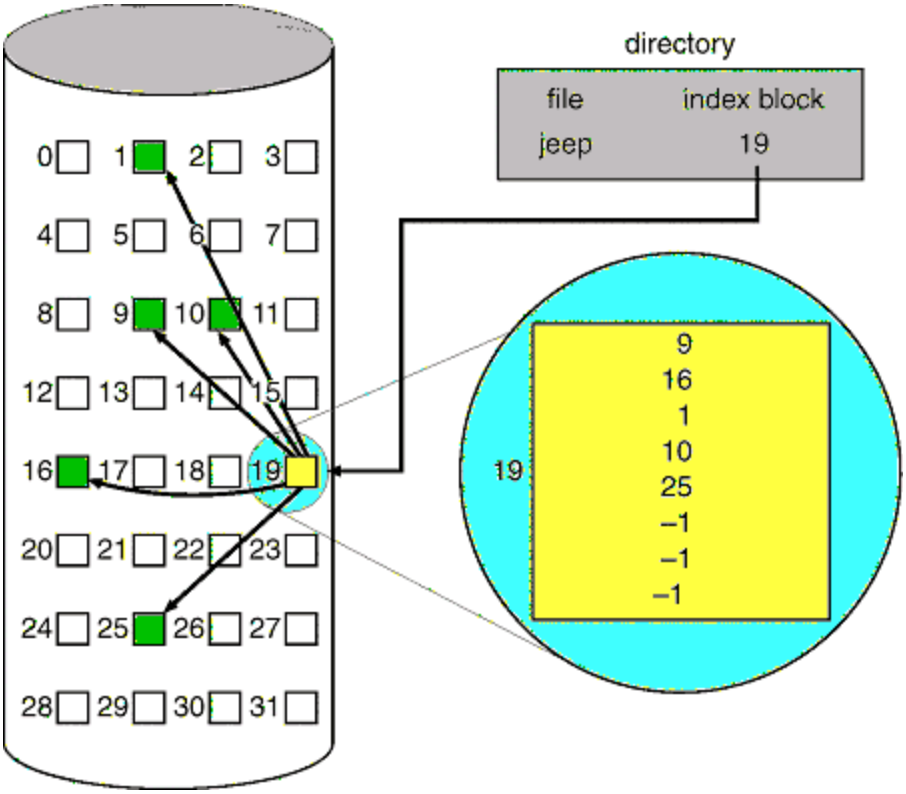
- Ni zunanje fragmentacije. Uporabimo lahko katerikoli prosti blok.
- Datoteko lahko širimo tudi kasneje in ni potrebno deklarirati velikosti datoteke že ob njeni tvorbi.
- Zgoščevanje diska ni potrebno.

Slabosti:

- Direkten dostop je zelo neučinkovit.
- Kazalci porabijo nekaj prostora v podatkovnih blokih.

Razpršenost kazalcev po disku pomeni določeno nezanesljivost.

Indeksirano alociranje (Indexed Allocation): Pri tej metodi alociramo za vsako tvorjeno datoteko indeksni blok. Ta blok vsebuje vse kazalce, ki kažejo na podatkovne bloke. Direktorij vsebuje ime datoteke in številko indeksnega bloka.



Prednosti:

- Z združevanjem vseh kazalcev na enem mestu se zanesljivost poveča.
- Direktni dostop do podatkov je učinkovit. i-ti kazalec v indeksni tabeli kazalcev že kaže na i-ti blok datoteke.

Slabosti:

- Problem s porabo prostora na disku s kazalci je še večji, saj za vsako datoteko porabimo cel blok, čeprav bi potrebovali le nekaj kazalcev

Uvod v datotečne sisteme

Atributi datotek

Datoteke so abstraktni podatkovni tipi, pravzaprav logične enote za pomnenje podatkov določenega tipa. Vsako datoteko opredeljujejo atributi, med katerimi tipično zasledimo naslednje:

- Ime
- Tip
- Lokacija (kazalec na napravo in na lokacijo datoteke na tej napravi)
- Velikost

- Zaščita
- Čas (tvorbe, zadnje spremembe ali zadnje uporabe)
- Lastništvo

Operacije nad datotekami

Operacijski sistem normalno nudi na voljo sistemske klice za tvorbo, zapisovanje in branje, premeščanje po datoteki in brisanje datotek.

Tvorba datotek: Sistem mora najti prostor za novo datoteko in uvesti novo datoteko v ustrezno kazalo (direktorij). V kazalu pomnimo prej navedene atribute datoteke.

Zapisovanje: Sistem mora locirati datoteko . Pomniti mora kazalec na mesto bodočega zapisa v datoteko in ta kazalec po vsakem zapisu ustrezno povečati.

Branje: Je podobno zapisovanju. Pri branju in zapisovanju običajno uporabljamo isti kazalec, ki se pri vsaki operaciji ustrezno poveča.

Premeščanje: Temu ustreza datotečna operacija seek. Sam dostop do datoteke je enak kot pri zapisovanju ali branju, vendar tu s primernim sistemskim klicem le premestimo kazalec v datoteko in ne izvedemo nobene vhodno - izhodne operacije.

Odpiranje in zapiranje datotek: Da ne bi pri vsaki operaciji potrebovali zamudnega iskanja oziroma lociranja datoteke, moramo načeloma vsako datoteko najprej s primernim klicem odpreti. V jeziku C imamo v ta namen klic open ( ) . Tak klic tipično vrne kazalec na rubriko v tabeli odprtih datotek. V nadaljnjih datotečnih operacijah nato uporabljamo ta kazalec, kar je precej hitreje kot ponovno iskanje datoteke preko informacij v direktoriju. Ko datoteke ne potrebujemo več, jo z ustreznim sistemskim klicem (na primer close( )) zapremo. Pri sodobnih operacijskih sistemih Lahko odpre isto datoteko tudi več procesov hkrati. Tipično ima vsak proces svojo tabelo odprtih datotek. V taki tabeli pomnimo tiste podatke o (odprti) datoteki, ki so specifični za dani proces (na primer kazalec na trenutno pozicijo v datoteki). Poleg tega imamo običajno še eno, skupno tabelo vseh odprtih datotek. Tu so vsi ostali podatki o datoteki in še števec, ki pove, kolikokrat je datoteka trenutno odprta. Ta števec se poveča pri vsakem klicu open( ) in zmanjša pri vsakem klicu close( ).

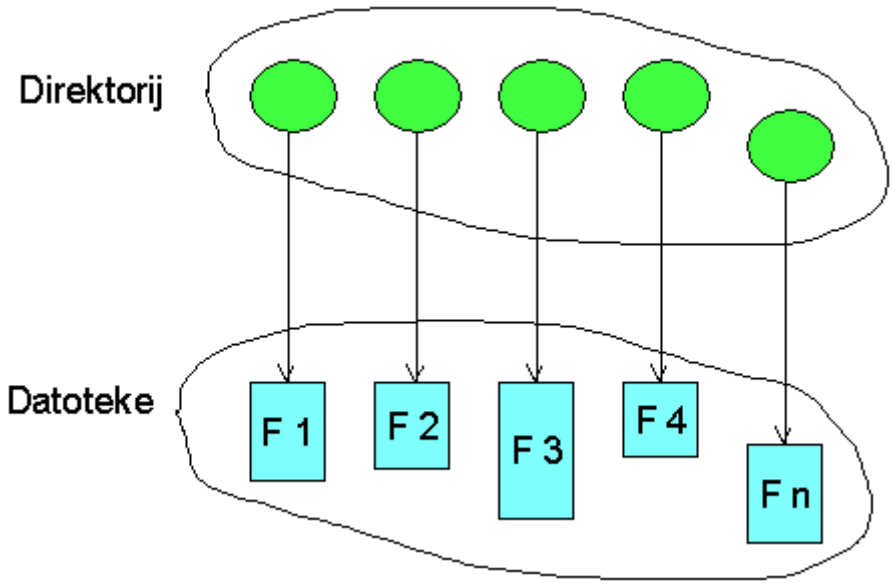
Brisanje datoteke: Sistem mora preko informacij v direktoriju poiskati datoteko, sprostiti njene podatkovne bloke in končno sprostiti tudi rubriko njeno rubriko v direktoriju.

Tipi datotek

Eno od temeljnih vprašanj je, ali naj operacijski sistem razpozna vrsto datoteke in tako ustrezno ukrepa. Pogosto je tip datoteke razpoznaven iz njenega imena, lahko pa je tip skrit v kakšnem drugem atributu, ki se avtomatsko nastavi ob tvorbi datoteke. Tak atribut lahko uporabimo tudi za implicitno ponovno proženje programa (na primer urejevalnika ali programskega generatorja), ki je tako datoteko pripravil.

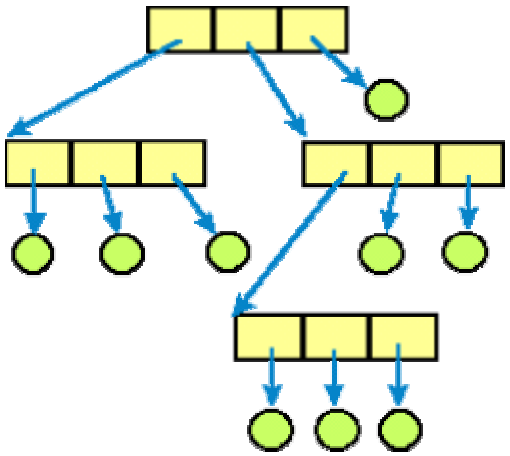
Vrste direktorijev

Datoteke na diskih morajo biti primerno organizirane. V nekaterih primerih je disk razdeljen v več logičnih enot (ponekod na primer particij, ponekod na primer obsegov (volumes)). Takim logičnim enotam lahko pravimo tudi virtualni diski. Vsak tak virtualen disk mora pomniti informacijo o datotekah, ki so tu spravljene. Tipično so podatki o datotekah podani v obliki simboličnih tabel oziroma direktorijev.

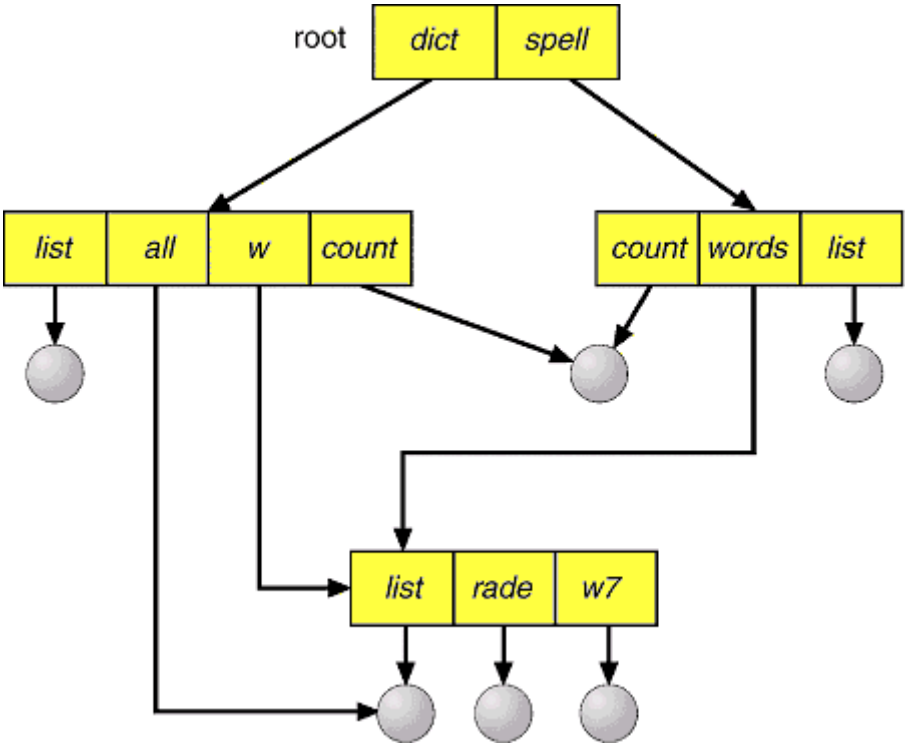


Pri organiziranju takih tabel je potrebno upoštevati operacije, ki pridejo v poštev. Te so:

- Iskanje datoteke: Datoteke imajo pridruženo eno ali več simboličnih imen. Iz teh pa moramo priti do podatkovnih blokov.
- Tvorba datotek: Dodajanje novih datotek v nek direktorij.
- Brisanje datoteke iz direktorija
- Preimenovanje datoteke. V nekaterih primerih pomeni to tudi premestitev imena datoteke iz enega direktorija v drugega.
- Prehajanje po datotečnem sistemu: Imeti moramo dostop do vseh direktorijev in datotek nekega datotečnega sistema.



Zaradi velikega števila datotek mora datotečni sistem zagotavljati njihovo preglednost.
Pogosto imamo zato direktorije organizirane drevesno, kot ilustrira prejšnja slika:



Sodobni datotečni sistemi dopuščajo, da ima ista datoteka po več imen in da so ta v različnih direktorijih. To precej lajša souporabo datotek in direktorijev Imamo bolj splošno organizacijo direktorijev v obliki acikličnega grafa.

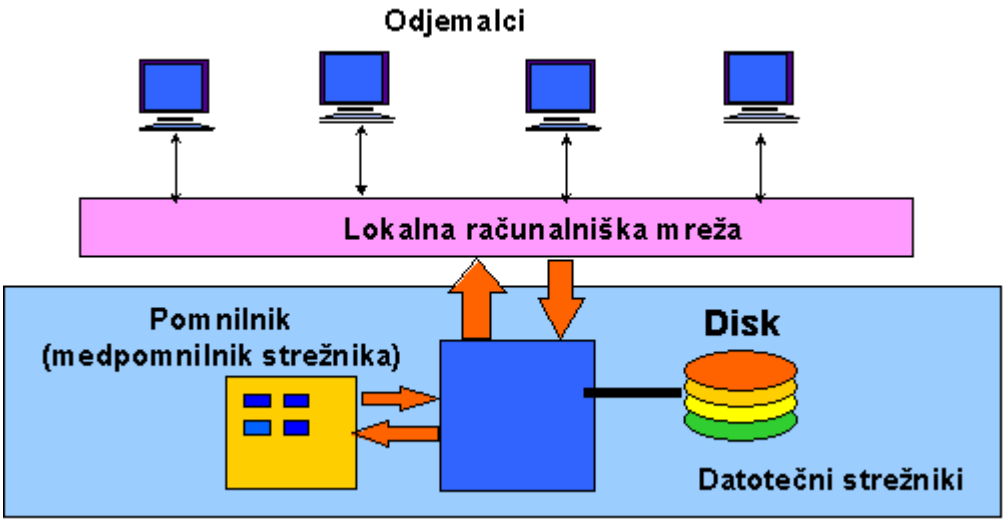
Pri nekaterih operacijskih sistemih (UNIX) dosežemo to tako, da dodajamo na isto datoteko nove simbolične vezi (symbolic links). Pri prehajanju po drevesu direktorijev operacijski sistem simbolične linke ignorira, saj bi lahko prišlo do cikliranja.

Ker je dostop do posameznih datotek tako možen preko več imen, mora biti zaradi konsistence datotečnega sistema brisanje datotek drugače urejeno. Datoteko lahko v resnici zbrišemo šele, ko brišemo zadnjo referenco nanjo. V ta namen imamo na primer pri sistemu UNIX za vsako datoteko števec povezav nanjo.

Porazdeljeni datotečni sistemi

Porazdeljeni datotečni sistem (DFS, distributed file system) je datotečni sistem, pri katerem so njegovi odjemalci, strežniki in pomnilne naprave razpršene po računalnikih porazdeljenega sistema .

Sistem ima lahko tudi več datotečnih strežnikov, ki so lahko med seboj lahko tudi različni.
Vse to pa naj bi bilo za uporabnika nevidno oziroma nepomembno.



Cilj porazdeljenega datotečnega sistema:

- Uporabniki fizično ločenih računalnikov naj souporabljajo podatke in pomnilne naprave
- Uporabniki naj imajo dostop do oddaljenih datotek tudi iz naprav brez lastnih diskov

Sistem naj bo neodvisen od tipa računalnikov

Lastnosti porazdeljenega datotečnega sistema

Transparentnost:

Transparentnost lokacije: Ime datoteke naj ne ne bo odvisno od lokacije fizičnega pomnilnika. Odjemalci naj uporabljajo za dostop do lokalnih in oddaljenih datotek enake operacije.
Neodvisnost od lokacije: Ime datoteke naj se ne spreminja, če spremenimo lokacijo fizičnega pomnilnika. Govorimo o transparentnosti migracije in o dinamičnem preslikavanju imen datotek. Večina porazdeljenih datotečnih sistemov (vključno z NFS) uporablja statično transparentnost lokacije.

Mobilnost uporabnikov

Uporabnik ni vezan na določen računalnik temveč se lahko najavi na kateremkoli sistemu in uporablja svoje datoteke.

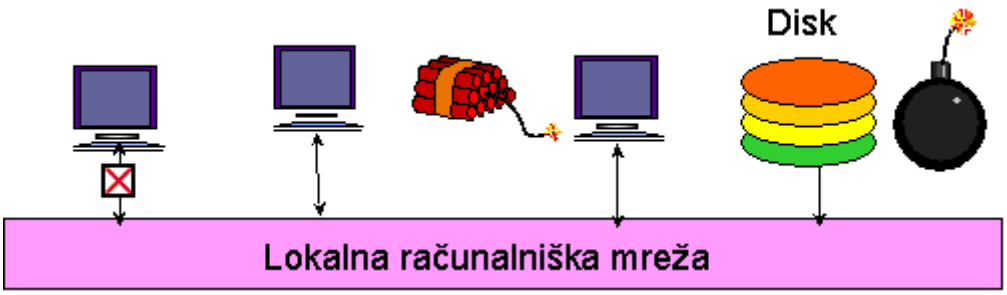
Zmogljivost

Upoštevati moramo:

čas CPE + čas dostopa do diska + čas porabljen z mrežo

Zmogljivost porazdeljenega datotečnega sistema mora biti primerljiva navadnemu datotečnemu sistemu kljub temu, da mora reševati dodatne probleme, kot so: zaklepanje in sinhronizacija ter imenovanje (naming service).

Toleranca izpadov



- Izpadi komunikacije
- Izpadi računalnikov
- Okvare pomnilnih naprav

Strategije razvrščanja dostopov do diska

Propustnost sistema poveča tudi primerna strategija razvrščanja zahtevkov dostopov do diska.

Najbolj preprosta strategija je tipa FCFS (first come first served). Po tej se zahtevki za pomik glave diska obravnavajo v enakem vrstnem redu, v katerem tudi prihajajo. Nedvomno je tak algoritem tudi enako pravičen za vse zahtevke, zanesljivo pa s stališča propustnosti ni optimalen.

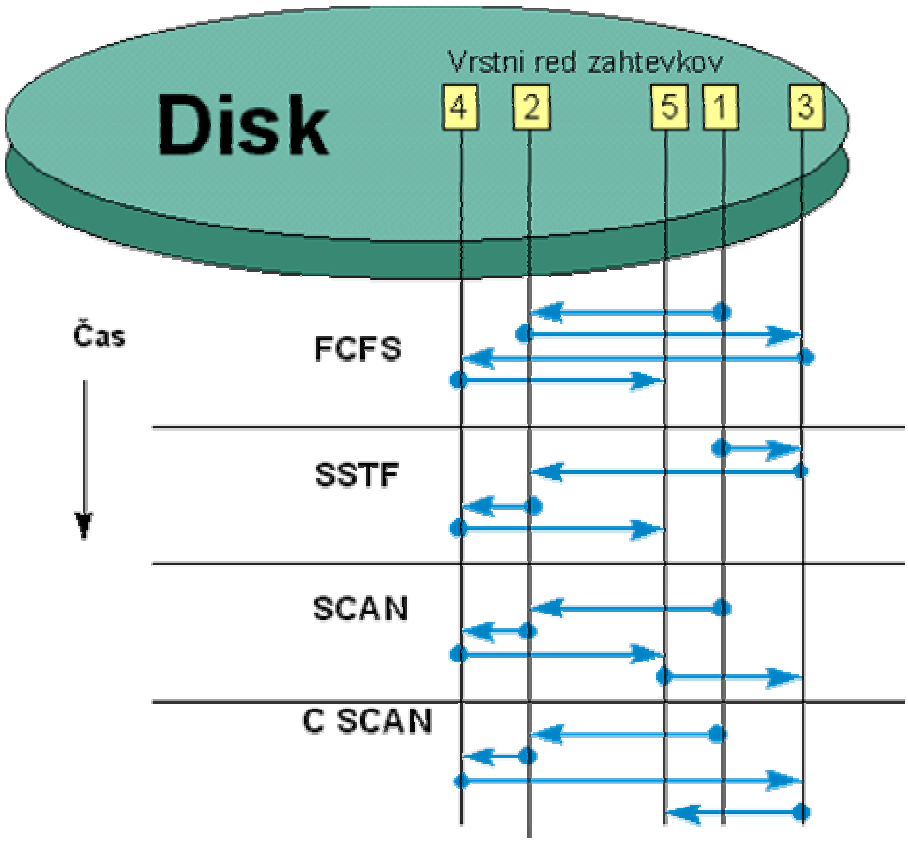
Druga strategija bi lahko kot kriterij za izbiro naslednjega zahtevka imela najkrajši pomik glave In s tem povezan čas pomika). Taki strategiji pravimo **SSTF** (Shortest Seek Time First). Tak algoritem daje žal prednost notranjim sledem diska, zunanje pa pridejo manjkrat na vrsto. Obstaja celo nevarnost stradanja.

Tretji princip je znan pod imenom SCAN. Glava se premika v dani (preferenčni) smeri in izpolnjuje zahtevke. Tudi tu pridejo zunanje sledi manjkrat na vrsto, kot notranje. Je pa varianca izpolnjevanja zahtevkov bolj stabilna.

Alternativa strategiji SCAN je strategija N-SCAN (Next scan). Pri tej obravnava sistem le tiste zahtevke v preferenčni smeri, ki jih `e ima na seznamu, novinci pa morajo počakati naslednji obhod.

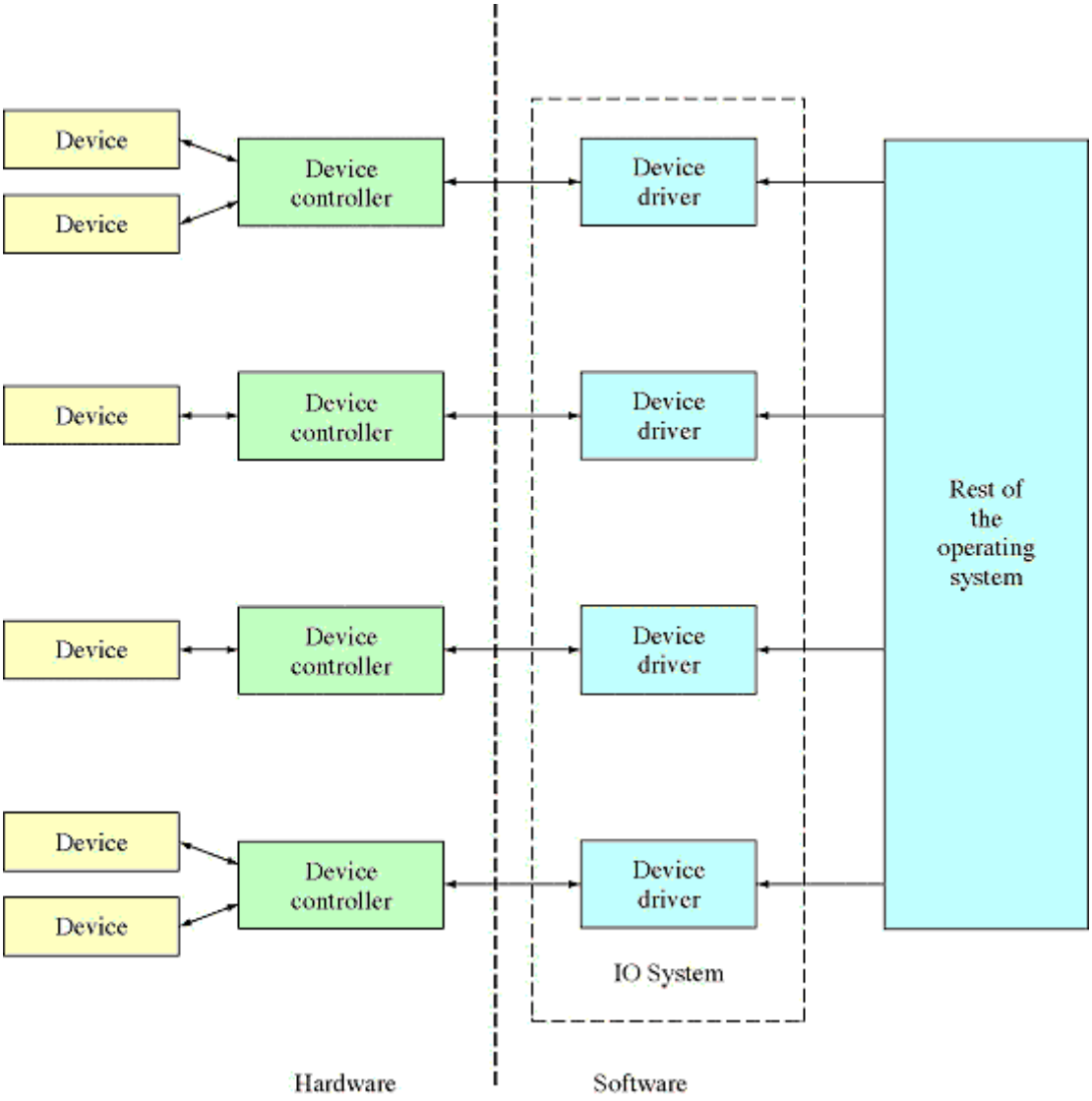
Končno imamo tudi strategijo C-SCAN (Circular scan). Pri tem se glava po vsakem obhodu takoj vrne za začetek, ne da bi pri tem pomiku obravnavala zahtevkov. Algoritem je zato do vseh zahtevkov enako pravičen.

Navedene strategije ponazoruje tudi spodnja slika:



Vhodno - izhodni podsistem

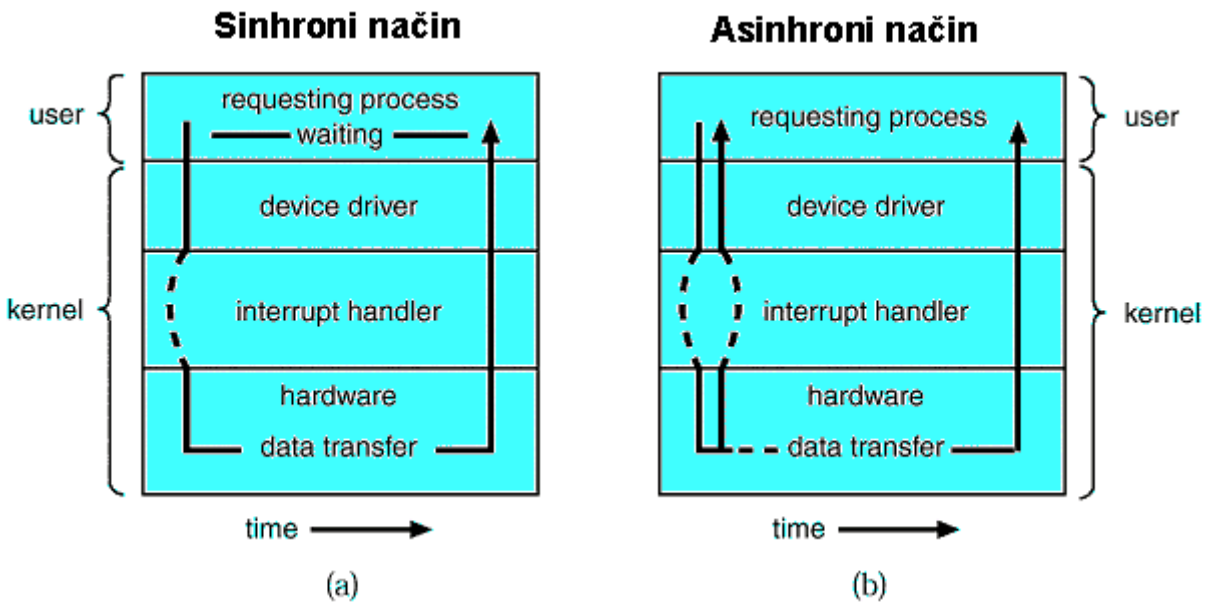
V sklopu računalniškega sistema je več *krmilnikov perifernih naprav* (device controllers). Dovolj tipično so ti krmilniki priključeni na skupno vodilo, ki omogoča dostop do pomnilnika. Na posamezen krmilnik je lahko povezanih tudi več naprav. Primer takega krmilnika je SCSI krmilnik. (Small Computer System Interface). Naloga krmilnika je premik podatkov med periferno napravo in lokalnim pomnilnikom.



Za start posamezne vhodno-izhodne operacije mora CPE vpisati ustrezne kode v registre v perifernega krmilnika. Krmilnik nato izvede ustrezno operacijo (običajno prenos) in nato obvesti centralno procesno enoto, da je operacija zaključena. Temu dogodku pravimo *prekinitveni zahtevek.*

Tako situacijo normalno posredno sproži **programski proces, ki zahteva prenos podatkov**. Po takem zahtevku lahko program čaka, da se vhodno-izhodna operacija zaključi (sinhrona vhodno-izhodna operacija), ali pa program teče dalje hkrati s potekom vhodno-izhodne operacije. V tem primeru potrebujemo še primeren sistemski klic, ki omogoči procesu, da kasneje počaka na zaključek sprožene vhodno-izhodne operacije. Sistem mora imeti možnost istočasnega sledenja večjemu številu vhodno-izhodnih zahtevkov. Zato mora imeti tabelo z opisom stanja posameznih naprav (periferna statusna tabela, device status table). V taki tabeli so normalni podani: tip posamezne naprave, njeni naslovi, stanje).

Ko pride do **prekinitvenega zahtevka**, operacijski sistem ugotovi, katera naprava ga je povzročila, naslovi ustrezni element periferne statusne tabele in ustrezno popravi njegove podatke (status, števce podatkov ipd).



Ko je **vhodno-izhodna operacija zaključena**, pogleda sistem, kateri proces je čakal na njen zaključek, in ga sprosti za nadaljevanje. Tak način vhodno-izhodnih operacij je asinhron

Običajno so vse vhodno-izhodne inštrukcije zaščitene in normalnemu uporabniku prepovedane. Računalnik jih izvaja le, če je v takoimenovanem sistemskem stanju (kernel mode, system mode).

S stališča operacijskega sistema mora biti zagotovljena čim večja neodvisnost od posebnosti posameznih perifernih naprav oziroma njihovih krmilnikov. Zato vhodno-izhodni podsistem običajno sestavljajo naslednji podsklopi:

- Ustrezno *vmesno polje* za pomnenje podatkov,
- *Splošen vmesnik* do gonilnikov perifernih naprav,
- *Gonilniki* (drivers) za posamezne vrste perifernih naprav.

Vsaki periferni napravi lahko ustreza poseben gonilnik, če pa je več naprav istega tipa, lahko imamo en gonilnik za vse take naprave (na primer diske). Sistem lahko podpira tudi takoimenovane *programske naprave* (software devices), katerim ni prirejena nobena fizična periferija. Tako bi lahko na primer obravnavali pomnilniške lokacije, ki so sicer procesu nedostopne.

Gonilniki naprav

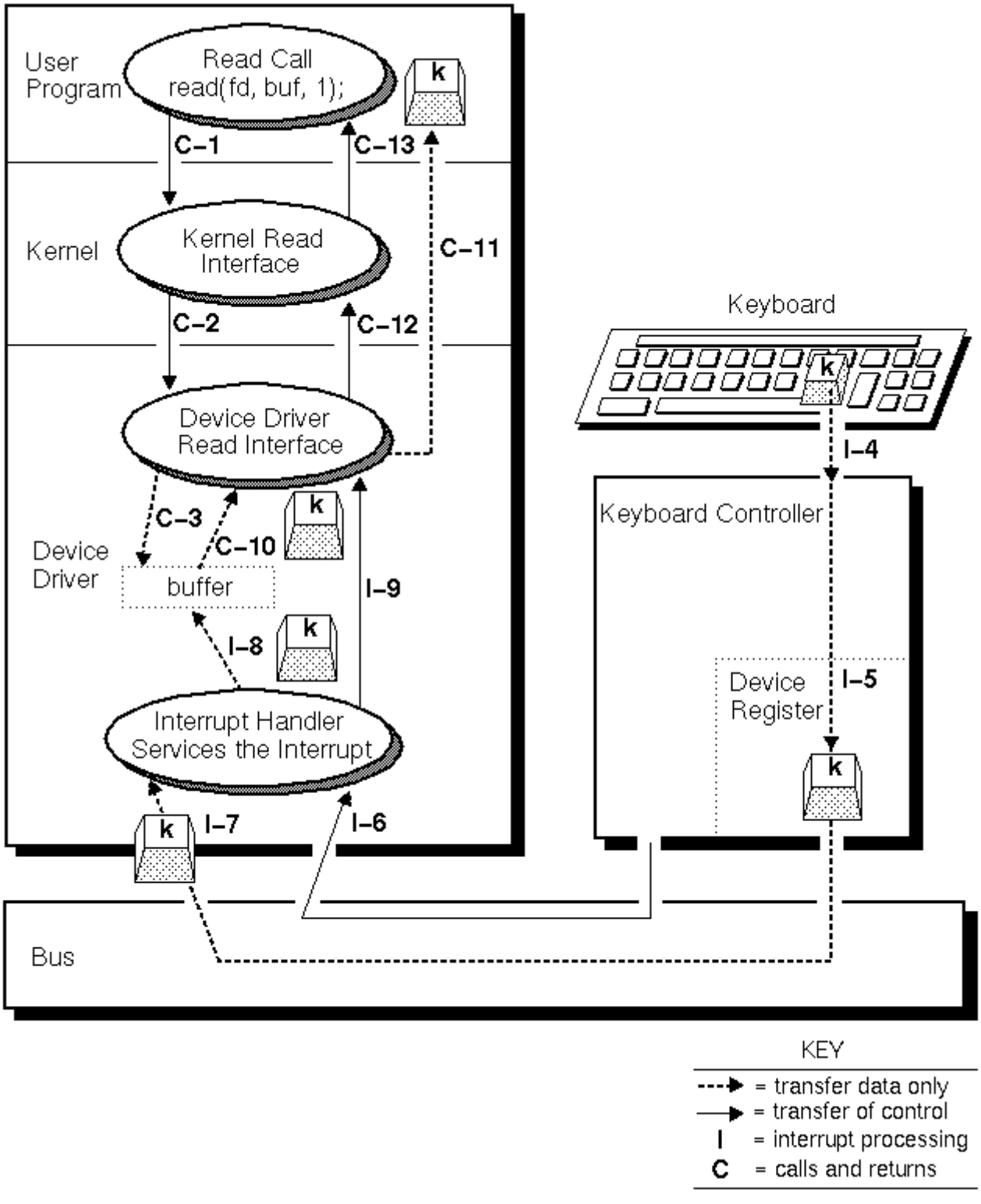
Namen gonilnika naprave je rokovanje z zahtevki, ki jih v zvezi z napravo danega tipa posreduje jedro operacijskega sistema.Vmesnik za jedro mora biti dobro definiran in konsistenten. Izoliranje programske kode, specifične za dano napravo od jedra poenostavlja dodajanje novih naprav.

Primer branja enega znaka

Primer predstavlja branje enega znaka s terminala.

Slika podaja zaporedje dogodkov od klica v uporabnikovem programu naprej .

- Gonilniku je posredovan zahtevek za branje (C-1 do C-3).
- Periferija zajame znak (I-4 do I-5).
- Pride do prekinitve(I-6).
- Rokovalnik prekinitev (interrupt handler) servisira prekinitev (I-7 do I-9).
- Uporabniški program dobi vrnjen prebrani znak (C-10 do C-13).



Bralni zahtevek gonilniku

Uporabniški program pokliče sistemski klic **`read`** (C-1).V sistemskem klicu pride do prenosa treh argumentov: opisnika datoteke(`fd`), kazalec na mesto,kjer pomnimo znak (`buf`) in celoštevilčno spremenljivko (vrednost 1), ki pove vmesniku `read` , koliko bajtov želimo prebrati. Proces je blokiran v gonilniku, ker je medpomnilnik, kjer naj bi bil znak, prazen.

Sprejem znaka

Kasneje vtipkamo na primer znak "k" (I-4). Koda znaka je pomnjena v podatkovnem registru naprave (I-5).

Generiranje prekinitve

Ob vtipkanju znaka se spremeni status krmilnika naprave, ki pošlje procesorju (CPU) prekinitveni zahtevek. Ta sproži rokovalnik prekinitev (I-6). Trenutno tekoči proces je pred tem prekinjen in njegovo stanje shranjeno, da se bo lahko kasneje nadaljeval.

Rokovalnik prekinitev servisira prekinitev

Rokovalnik prekinitev (včasih mu pravimo tudi prekinitvena servisna rutina) pomakne znak v lokacijo, ki jo poznajo tudi ostali moduli gonilnika. Zatem obudi proces, ki je moral zaradi čakanja na znak zaspati. Končno izstopi in pri tem restavrira proces, ki je bil prekinjen ob servisiranju prekinitve.

Vračamo znak

Ko kasneje razvrščevalnik procesov zbujenemu procesu dovoli nadaljevanje, je brani znak že v ustreznem medpomnilniku. Proces ga premesti v uporabniški naslovni prostor (C-11). Vmesnik `read` gonilnika naprave vrne nadzor jedru, ta pa uporabniškemu programus (C-13).

Povzetek

Zgled poenostavljeno predstavlja branje znaka. Opazimo, kako uporabniški program zahteva servis jedra. Vidimo še, da je obravnava prekinitvenih zahtevkov asinhrona glede na druge aktivnosti gonilnika.

Zaščita in varnost OS

Zaščita je mehanizem nadzora dostopa programskih procesov ali uporabnikov do resursov računalniškega sistema. Varnost takega sistema je mera zaupanja, da bo ohranjena integriteta sistema in njegovi podatki.

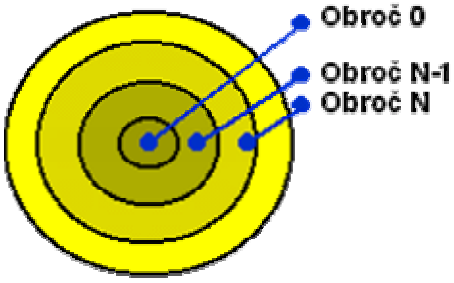
Domena zaščite:

Računalniški sistem je nabor procesov in objektov. Objekti so lahko aparaturni (CPE, pomnilniški segmenti, periferne naprave) ali programski (datoteke, programi, semaforji itd.).

Proces deluje v domeni zaščite (protection domain), ki definira množico objektov in operacij nad njimi, ki so procesu dovoljene. Domena zaščite je lahko uporabnik, nek proces ali celo procedura. Posamezne domene si lahko pravice do dostopa (access rights) tudi delijo.

Asociacija med procesom in domeno je lahko statična ali dinamična. V slednjem primeru lahko proces prehaja iz ene domene zaščite v drugo.

Organizacija zaščite je lahko hierarhična (obročna) (tako je bil organiziran zaščitni sistem pri MULTICS). Kontroliran prehod med domenami nastopi pri prečkanju med obroči.



Bolj učinkovit model zaščite ponuja matrika dostopnosti (Access matrix).

Objekti / Domene	O1	O2	O3	O4		D1	D2	D3	D4
D1	branje						preklop		
D2								preklop	preklop
D3		branje	izv.						
D4	Branje pisanje		branje pisanje			preklop			

Komentar: Proces, ki se izvaja v domeni D3, ima pravico do branja objekta O2 in izvajanja objekta O3.

Vrste v matriki predstavljajo domene zaščite, kolone pa so posamezni objekti. Pri tvorbi novega objekta enostavno dodamo matriki dostopnosti novo kolono.

Matriki dostopnosti so dodane še kolone, ki določajo dopustne preklope med posameznimi domenami.

Komentar: Proces, ki se izvaja v domeni D2, lahko preide v domeno D3 ali D4.

Najbolj preprosta implementacija matrike dostopnosti je z globalno tabelo, ki jo predstavlja urejena množica trojčkov <domena, objekt, pravice> . V tej tabeli iščemo trojček <Di,Oj,Rk>. Če ga najdemo, je operacija dovoljena, sicer pa ne. Slabost take implementacije je v velikosti (razpršenosti) tabele.

Drug način implementacije obravnava vsako kolono matrike kot seznam dostopnosti za en objekt. Prazne elemente lahko izpuščamo. Za vsak objekt dobimo tako urejene pare <domena, pravice>.

Tretja različica predvideva, da namesto kolon (objektov) gledamo vrstice (domene). Tako dobimo za vsako domeno seznam zmožnosti (capability list).Operacijo na določenem objektu lahko izvedemo v dani domeni, če se v seznamu parov <objekt, pravice> za to domeno nahaja tudi tak par. Tako organizacijo zaščite zasledimo na primer pri sistemu MACH.

Tudi tak seznam je sam po sebi zaščiten objekt.

Varnost sistemov

Medtem, ko je zaščita sistema interni problem operacijskega sistema, vključuje varnost tudi okolje, v katerem deluje. Zagotoviti moramo fizično zaščito sistema ter zaščito pred neavtoriziranimi dostopi, namernimi uničenji podatkov, naključnim vnašanjem nekonsistenc itd..

Naloge in postopki sistemskega administratorja

Pregled nalog

Neglede na to, na kakšnem sistemu sistemski administrator dela, lahko naštejemo nekaj rutinskih opravil, ki čakajo sistemskega administratorja.

- Dodajanje/brisanje uporabnikov.
- Dodajanje novih strežnikov na mrežo.
- Arhiviranje podatkov.
- Nameščanje popravkov (patch) obstoječega softwarea.

- Nameščanje novega softwarea.
- Nadziranje učinkovitosti sistema.
- Nadzor tiskanja.
- Iskanje varnostnih lukenj v sistemu.
- Pisanje script za avtomatizacijo dela.
- Zagotavljane delovanja sistema (da lahko šef prebere svoj email).

Najpomembnejša naloga pa je seveda, da v primeru problemov čim prej vzpostavi sistem v delujoče stanje.

Podpora uporabnikom

1. Pomemben aspect dela sistemskega administratorja je podpora uporabnikom, vendar se mora administrator zavedat, da mu je primarna naloga zagotavljanje delovanja celotnega sistema, ne po posameznega programa na enem računalniku.
2. Zbiranje informacij o problemih uporabnikov, da bi se problemi lahko odpravili na sistemski ravni.

Vzdrževanje strojne oprame

- Zajema vzdrževanje računalnikov, terminalov, tiskalnikov, modemov in drugiv V/I naprav.

Vzdrževanje programske opreme

- Vzdrževanje operacijskega sitema, programov, servisov.
- Nameščanje in nadgradnja operacijskih sistemov.
- Vzdrževanje uporabniških računov, datotek, dostopa do tiskalnika.

Vzdrževanje računalnikov in omrežja

- Vzdrževanje fizičnega omrežja
- Konfiguriranje strežnikov

Vzdrževanje varnostne politike

- Nadzorovanje mreže, servisov in delovnih postaj z namenom ugotavljanja morebitnih zlorab (sumljivi dostopi).
- Nameščanje varnostnih popravkov.
- Nameščanje varnostnih dodatkov v omrežje za preprečevanje neavtoriziranega dostopa.
- Nameščanje protivirusne programske opreme.

Vzdrževanje informacijskih servisov

- E-mail
- News
- Ftp
- World-wide web

Usposabljanje uporabnikov

- Reševanje problemov povezanih z uporabo sistema.
- Priprava in izvajanje seminarjev o uporabi nanovo nameščene programske opreme.
- Priprava brošur z navodili o uporabi nanovo nameščene programske opreme oz. nadgradenj.

Vzdrževanje sistemske dokumentacije in knjižnic

- Vzdrževanje lokane dokumentacije
- Hranjeneje dokumentacije o programski in strojni opremi, ki je nameščena v sistemu in morebitnih težavah.
- Hranjenje priročnikov

Razno

- Naročanje nove programske in strojne opreme.
- Izdelava varnostnih kopij.

Rezervne kopije

Izdelava varnostnih kopij in restavriranje sistema po napakah ni ravno najbolj priljubljeno opravilo sistemskih administratorjev. Uporabniki imajo to grdo navado, da redno brišejo pomembne datoteke, ki jih je potem potrebno restavrirat, pri velikem številu delovnih postaj pa tudi okvare diskov niso nekaj nemogočega. Tako da sta že ta dva razloga dovolj velik vzrok za izdelavo varnostnih kopij.

Obstajajo različni načini izdelav varnostnih kopij kakor tudi različna orodja, ki avtomatizirajo te postopke.

Avtomatozacija nalog

Z avtomatizacijo rutinskih postopkov se lahko prihrani veliko časa. Prav tako pa lahko poskrbimo, da bomo sistem ali omrežje obremenili takrat, ko bo to najmanj motilo normalno delo. Tipično naloga je npr. izdelava varnostnih kopij, ki jih ne bomo delalo med delovnim časom.

Nadzor sistemskih virov

Zmogljivosti računalniškega sistema je potrebno pravilno nadzirati, če želimo da nam sistem deluje vsaj približno tako, kot je sposoben. Ozka grla računalniškega sistema so naslednja:

- Izraba CPU
- Poraba pomnilnika
- Disk
- Omrežje

Operacijski sistemi sami imajo vgrajena orodja, ki omogočajo vpogled v delovanje prej omenjenih komponent in proženje opozoril ob dogodkih definiranih s strain administratorja.

Dodatek:

Spletne tehnologije

Technologije:

- HTML,
- Java,
- ActiveX,
- CGI Scripts,
- DB Access,
- TCP/IP

Prednosti:

- Platformna neodvisnost
- Hiter razvoj
- Manj programiranja
- Lahek dostop

Nov programski model, ki se uveljavlja v zadnjem času, je programiranje za svetovni splet, programiranje spletnih storitev. Vse značilne komunikacijske poti v poslovnem procesu, v organizacijah in med njimi ter s strankami, bodo temeljile na spletnih tehnologijah. Način, ki se je včasih uporabljal za gradnjo programske opreme, počasi izginja. To pa zato, ker ni več ogromnih namenskih programov, ki se ne morejo več zadosti hitro posodabljati.

Prvi val, ki je komuniciranje po internetu približal množici ljudi, so bile osnovne spletne tehnologije, kot sta protokol http in jezik HTML.

Drugi rod v razvoju programske opreme za splet je prinesel dinamične spletne strani, ki so zahtevale povezljivost z zbirkami podatkov.

Tretji rod programske opreme bo moral upoštevati spremenljivost povezovalnih zmogljivosti, mobilnost uporabnikov, predvsem pa bo moral zagotavljati zadostno varnost in kakovost storitve. Za doseganje tega pa potrebujemo sodobne standarde in nove programske tehnologije: komponentna tehnologija, XML, spletne storitve.

Objekti in komponente

Zastavlja se vprašanje načina organizacije programske opreme na vmesnem sloju, na sloju kjer želimo izvajati poslovno logiko. Želimo zaključene dele funkcionalnosti, ki nam bodo izvedli neko operacijo (npr. da preverijo kreditno kartico). To storimo s komponentami. Programske komponente so deli programske kode, ki navzven predstavljajo neko zaključeno celoto. Je paket enega ali več programov s katerimi upravljamo kot s celoto in do katerih dostopamo preko dokumentiranih vmesnikov dostopnih v realnem času [GartnerGroup]. Ideja je sicer že stara: v strukturnem pristopu se je funkcionalno sorodna koda združevala v funkcije in procedure, v modularnem pristopu pa v module. Vendar so bili, za razliko od komponent, takrat podatki in programska koda strogo ločeni. Pomanjkljivost takega ločevanja je bila v tem, da se koda, ki je bila napisana, da deluje z nekimi podatki v podatkovni bazi, ni dala

enostavno uporabiti nad drugimi podatki. Če smo kaj spremenili v podatkovni bazi, smo to morali narediti tudi v programski kodi, verjetno celo na več mestih.

Komponente lahko razumemo kot koščke ali pakete programske kode, za razliko od objektov, ki so način za implementacijo programske opreme. Komponente so neodvisne od programskega jezika, zato je lahko odjemalec napisan v drugem jeziku kot komponenta. Tudi upoštevanje načela ograjevanja je tu veliko bolj strogo kakor pri objektih. Zaradi neodvisnosti od jezika, komponentne tehnologije vpeljujejo nov jezik, v katerem definiramo vmesnik komponente. (IDL – Interface Definition Language).

Vmesnik komponente je sestavljen iz množice metod. Seznam metod pove, katere operacije komponenta nudi odjemalcem in vse, kar mora odjemalec vedeti o komponenti, je njeno ime. Proženje metode pa poteka s sporočilom v katerem odjemalec navede, katero metodo želi prožiti. Zaradi takega načina je nadgradnja komponent enostavna. Vse kar moramo zagotoviti je, da nova komponenta podpira enak vmesnik. Na ta način pa je rešena učinkovita porazdelitev komponent med računalniki, kar je bil bistven problem dvoslojne arhitekture.

Najpomembnejši komponentni modeli so:

- CORBA (Common Object Request Broker Architecture), je definiran s strani OMG (Object Management Group)
- COM/COM+ (Component Object Model); definicija Microsofta
- EJB (Enterprise Java Beans); definicija Sun-a

XML (Extensible Markup Language)

Zaradi velikih možnosti povezanosti med različnimi računalniškimi sistemi, ki jih je ustvaril internet, se je pričelo pojavljati veliko število problemov pri zagotavljanju protokolov in specifikacij, ki bi premostile (včasih res) ogromne razlike med sistemi z različnimi strojnimi in programskimi konfiguracijami.

Pri predstavitvi podatkov in standardizaciji izmenjave le teh, se je začel uveljavljati standard XML, ki sedaj spada med najpomembnejše tehnologije za standardizacijo, opis in prevedbo podatkov. XML je označevalni jezik podoben HTML, vendar je namenjen opisovanju in je usmerjen v to, kaj podatki vsebujejo, medtem ko je HTML namenjen prikazovanju podatkov in je usmerjen v to, kako so ti podatki videti.

XML lahko smatramo kot aparaturno in programsko neodvisen način prenašanja informacij med računalniki na svetovnem spletu. Za sam prikaz, oziroma formatiranje teh podatkov, pa je odgovoren HTML. Namenjen je za opis strukturnih dokumentov, ne glede na vsebino in omogoča prenosljivost podatkov, ne glede na izbrano okolje. Standardizacijo zapisa dokumenta se dosega na več načinov :

- z opisnimi oznakami podatkov (meta podatki)
- s standardizacijo opisnih oznak podatkov
- z ločevanjem vsebine od prikaza podatkov
- s prevedbo zapisa XML v poljubno obliko

Dokumenti XML se delijo na dve osnovni vrsti:

- podatkovno naravnani dokumenti XML (vsebujejo neke podatke, pri čemer je vsak podatek opisan z meta podatki)
- dokumentno naravnani dokumenti XML (vsebujejo neločljivo vsebino, ki je ni moč opisati z meta podatki)

Shema XML je tehnologija za preverjanje pravilnosti dokumentov XML in je zapisana v jeziku XML. Ko uporabnik izdela dokument XML, ga najprej obdela tolmač (parser), ki prepozna osnovne napake (nedovoljeni znaki, ipd.). Nato potrjevalec (validator) poišče semantične, oziroma vsebinske napake. Če je dokument ustrezen, se obdelava nadaljuje in podatki se npr. lahko začno uporabljati. V nasprotnem primeru program sporoči napako in obdelava se neuspešno konča.

Projekt **ebXML** želi na temeljih XML standardizirati izmenjavo poslovnih dokumentov. Predvideva se, da bo zajel in počasi nadomestil tudi standard EDI. Specifikacija poslovnega dokumenta XML je sestavljena iz opisa vsebine dokumenta XML, iz opisa struktura dokumenta XML in iz opisa relacijskega modela podatkov v dokumentu XML.

J2EE (Java 2 Enterprise Edition)

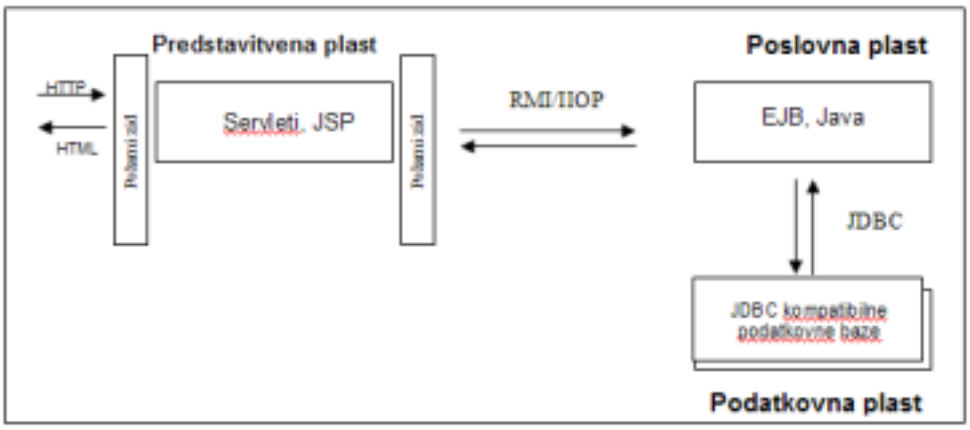
Java je bila definirana kot zbirka programskih orodij, strojne opreme in nove filozofije razvoja programske opreme. Bistvo Jave lahko strnemo v tri trditve:

- je programski jezik, ki je enostaven (posebej za nekoga , ki pozna C/C++), predmetno usmerjen, varen, robusten, porazdeljen, večopravilen, interpretiran, arhitekturno neodvisen in prenosljiv
- poleg jezika je pomen Jave tudi v tem, da nudi vsa potrebna razvojna orodja (prevajalnik, tolmač prevedene kode, razhroščevalnik). Vse to je bilo vključeno že v prvi paket JDK (Java Development Kit)
- je tudi strojna oprema, ki omogoča izvajanje javanskih programov. Strojna arhitektura je lahko izvedena v programski opremi – JVM (Java Virtual Machine), ki je prirejena različnim operacijskim sistemom, ali pa fizično na posebnem javanskem procesorju.

Platforma J2EE predstavlja specifikacijo za razvoj aplikacij za velika podjetja in je tehnologija, zgrajena na temeljih programskega jezika Java. Zaradi zmede pri uporabi izraza Java skozi vse prejšnje verzije, so se pri podjetju Sun odločili za spremembo poimenovanja in s tem odpravo vseh nejasnosti. Tako so ločili specifikacijo Java platforme od implementacije le-te. Specifikacija natančno določa lastnosti posamezne različice platforme (J2SE - Java 2 Standard Edition in J2EE), ki jim mora zadostiti implementacija različnih proizvajalcev.

Namen J2EE je torej ponuditi celovit način za izdelavo aplikacij, ki jih danes zahtevajo velika podjetja. Ponuja nam tehnologijo za podporo vsem plastem večplastnega modela.

J2EE torej podpira večplastni model porazdeljenih aplikacij, ki temelji na definicijah vsebnika in vsebniške komponente. Posamezni deli aplikacije se lahko torej izvajajo na različnih računalnikih. Kot splošen večplastni model tudi arhitektura J2EE definira odjemalčevo plast, vmesno plast (iz ene ali več podplasti) in informacijsko plast, ki skrbi za storitve obstoječega informacijskega sistema.



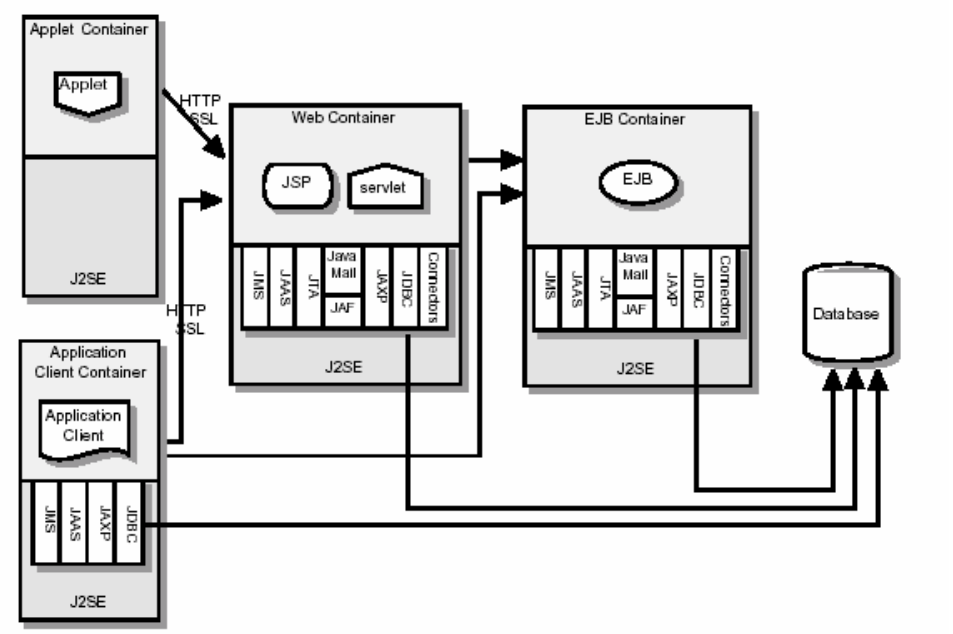
Slika večplastnega modela z J2EE:

Naloge posameznih plasti so razdeljene na sledeči način:

- odjemalna plast omogoča več vrst odjemalcev tako znotraj kot zunaj požarnega zidu
- vmesna plast odjemalcu nudi storitve preko spletnega vsebnika v spletni podplasti, poslovno logiko pa preko vsebnika EJB v podplasti EJB
- v informacijski plasti se zagotavlja dostop do obstoječega informacijskega sistema preko standardnih vmesnikov

Zgradba J2EE

Slika prikazuje logične povezave med posameznimi deli platforme, ne pomeni pa, da je takšna delitev tudi fizično potrebna in zahtevana. Praktično to pomeni, da lahko vsi zgoraj opisani deli obstajajo na enem samem računalniku.



Slika: Zgradba Java 2 Enterprise Edition

Jedro tehnologije arhitekture J2EE sta definiciji komponente in vsebnika. Poglejmo podrobneje iz katerih elementov je sestavljeno izvajalno okolje J2EE:

Aplikacijske komponente - J2EE programski model definira štiri tipe aplikacijskih komponent, ki jih implementacija specifikacije mora podpreti: aplikacijski odjemalci, apleti, servleti, JSP in EJB. Razdelimo jih lahko na tri kategorije in sicer:

- komponente, ki jih namestimo, upravljamo in izvajamo na strežniku (JSP, Servleti, EJB)
- komponente, ki so nameščene in upravljane na strežniku, vendar se izvajajo na odjemalcu (HTML strani, apleti)
- komponente, katerih nameščanje in upravljanje ni v celoti domena J2EE specifikacije (aplikacijski odjemalci).

Vsebniki - zagotavljajo izvajalsko okolje za aplikacijske komponente in predstavljajo enoten pogled na nižje ležeče knjižnice (API). Takšna arhitektura povezuje aplikacijske komponente in storitve, ki jih predpisuje specifikacija in proženje storitev napravi transparentno. Implementacija specifikacije mora zagotoviti vsebnike za vse tipe aplikacijskih komponent (odjemalski vsebnik, aplet vsebnik, spletni vsebnik, EJB vsebnik). Specifikacija zahteva, da vsebnik zagotovi izvajalsko okolje. Vsebnik apletov lahko uporablja tudi JavaPlug-in in z njim zagotovi potrebno izvajalsko okolje. Orodja vsebnika prav tako podpirajo in razumejo formate za pakiranje aplikacijskih komponent pred njihovo predajo (npr. JAR zbirke). Vsebnike izdela J2EE izdelovalec. Implementacija omenjenih vsebnikov je kombinacija obstoječih tehnologij za podporo transakcijskim storitvam v kombinaciji z Java 2 platformo. J2EE odjemalska osnova je tipično grajena na J2SE tehnologiji.

Standardni servisi – J2EE specifikacija določa tudi nabor standardnih storitev, ki jih mora vsak J2EE produkt podpirati in s tem tudi zagotoviti. Standardni servisi so: gonilniki za

upravljanje z viri, podatkovna baza, HTTP / HTTPS, JTA, RMI-IIOP, Java IDL, JDBC, JMS, JNDI, Java Mail, Java Help

Pravzaprav gre za vrsto knjižnic, ki omogočijo aplikacijskim komponentam dostop in uporabo teh storitev. Storitve je moč tudi razširiti, zato specifikacija podpira standarden način ravnanja z razširitvami.

Podatkovna baza – J2EE vključuje tudi podatkovno bazo, ki je poslovnim objektom dostopna preko JDBC knjižnice

Microsoftova arhitektura .NET

Sredi devetdesetih let je Microsoft popolnoma podcenil potencial interneta. Medtem je Sun v tem času predstavil platformo java in s tem ponudil način izdelave programske opreme, prirejen potrebam interneta. Microsoft je moral hitro reagirati na odziv trga s tem, da je predstavil tehnologijo ActiveX. Komponente ActiveX je uporabnik prenesel iz interneta v osebni računalnik in jih tam zaganjal. Ker pa so to bile v bistvu komponente COM, torej programi, ki so imeli dostop do vseh sistemskih virov, je bila s tem odprta varnostna luknja. Ni bilo namreč nobenega zagotovila, da komponenta, ki jo želimo namestiti na računalnik, ne vsebuje nevarne kode. S tem se je pokazala neustreznost arhitektur, ki so nastajale v času nepovezanosti osebnih računalnikov (kot na primer popoln nadzor nad sistemskimi viri).

Microsoftov odgovor na Sunovo specifikacijo J2EE je strategija .NET. Čeprav smo pri Microsoftu vajeni novih imen, pod katerimi se mnogokrat skrivajo stare, le dopolnjene tehnologije, je pri .NET to precej drugače. Naenkrat so vsi Microsoftovi izdelki začeli dobivati končnico .NET in nedvomno je .NET tista ključna beseda, ki jo bomo srečevali v prihodnosti.

Arhitekturo .NET lahko razdelimo na štiri področja:

- .NET infrastruktura in orodja
- .NET odjemalci
- .NET spletne storitve
- .NET uporabniške izkušnje in znanje

.NET infrastruktura in orodja

Infrastruktura omogoča razvijalcem razvoj, gradnjo in izvajanje aplikacij, programov, spletnih storitev. V to področje prištevamo:

- razvojno okolje Visual Studio .NET
- .NET strežnike
- ogrodje .NET (.NET Framework),

Visual Studio .NET

Pomemben del vsake podlage je seveda njeno okolje za razvoj programskih rešitev. Večkrat se je že potrdilo, da vsaka nova podlaga stoji in pade s podporo razvijalcev, ki ponujajo rešitve zanjo. Za podlago .NET bo na voljo (trenutno na voljo različica Beta 2) razvojno okolje Visual Studio .NET, ki je nova različica sicer že znanega izdelka, vendar tokrat prvič zares lahko govorimo o enotnem okolju za najrazličnejše programske jezike. Za ohranjanje združljivosti

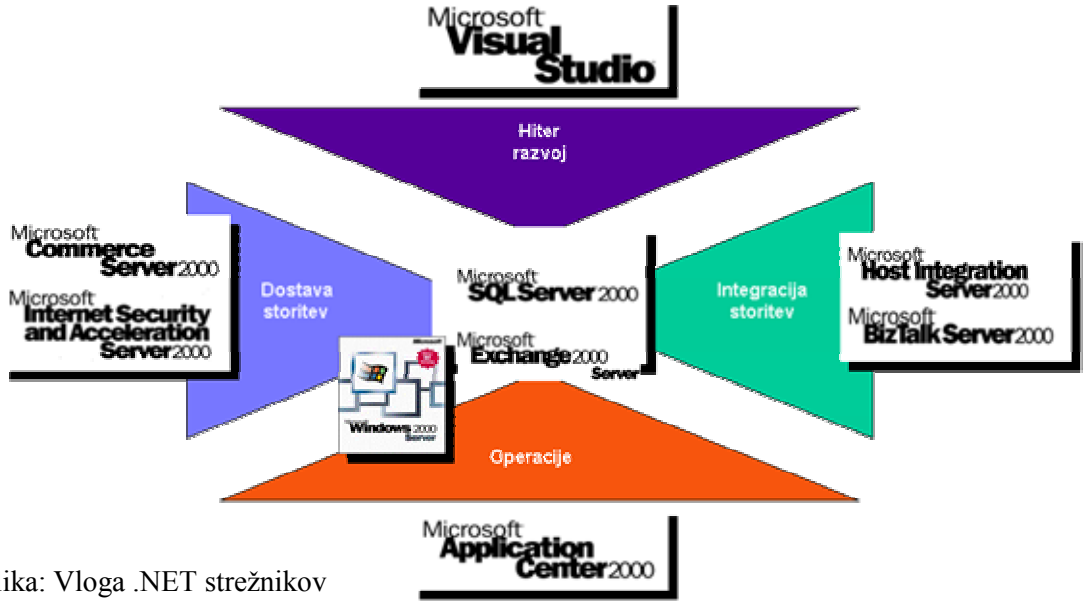
za nazaj, je na voljo Visual C++, s katerim lahko še vedno programiramo za podlago Win32. Ostale prvine pa so namenjene izključno podlagi .NET za ustvarjanje spletnih XML storitev in aplikacij v okolju Microsoft .NET Framework in z močno podporo protokola SOAP.

.NET strežniki (.NET Enterprise Server)

V skupino .NET strežnikov prištevamo:

- **Application Center 2000**: omogoča lažji nadzor in upravljanje spletnih aplikacij, npr. enostavnejše instaliranje nadgradenj neke aplikacije ter lažje upravljanje več strežnikov.
- **BizTalk Server 2000**: je celovita rešitev na trgu za medsebojno integracijo poslovnih aplikacij (EAI - Enterprise Application Integration) ter med podjetniško sodelovanje (B2B) in ponuja tehnologijo BizTalk Orchestration, ki omogoča izgradnjo dinamičnih poslovnih procesov na temeljih standarda XML. Orodje BizTalk Mapper omogoča pretvorbo iz ene sheme v drugo z generiranjem XSLT datotek. Za samo izmenjavo dokumentov preko standarda SOAP skrbi BizTalk FrameWork. Podprti so tudi protokoli EDI, HTTP, SMTP in drugi. Strežnik BizTalk omogoča združevanje različnih aplikacij in poslovnih procesov v celovito rešitev, kar je v tem trenutku največji izziv za računalniško industrijo. Enterprise različica je namenjena velikim organizacijam, trgovskim stičiščem in digitalnim tržnicam. Vgrajena je podpora za neomejeno število podjetniških aplikacij in trgovskih partnerjev, omogoča pa izdelavo gruč (clustering) in podporo za neomejeno število procesorskih enot. Standard različica je namenjena majhnim in srednje velikim podjetjem in vključuje podporo za integracijo petih podjetniških aplikacij ter povezave s petimi trgovskimi partnerji. Izdelek je namenjen uporabi v manj zahtevnih poslovnih okoljih, tako da ne vključuje podpore za gruče in več procesorjev.
- **SQL Server 2000**: je podatkovna baza, ki omogoča shranjevanje, obnovo in analizo strukturiranih podatkov
- **Commerce Server 2000**: ponuja uporabnikom izgradnjo rešitev elektronskega trgovanja po lastni meri. V paketu je združeno vse za začetek e-trgovanja, kar vključuje orodja za profiliranje kupcev ter nadzor nad izdelki in storitvami, orodja za obdelovanje transakcij ter izvajanje s pomočjo naprednih analiz oblikovanih tržnih prijemov. Paket CS 2000 zagotavlja vsestransko podporo odločanju s pomočjo podatkovnih skladišč in naprednih analiz, ki upoštevajo vse relevantne informacije. S pomočjo vgrajene tehnologije OLAP (On-line Analytical Processing) za poslovno poročanje in integracijo s podatkovnim strežnikom Microsoft SQL Server, je mogoče izdelati statične in dinamične analize spletnih trgovin. Integracija preko BizTalk strežnika pa nam omogoča, da podatke v XML obliki pošljemo v nadaljno obdelavo v naš poslovni sistem. Posebne tehnologije omogočajo tudi t.i. »cross-selling«, ki kupcem ponudi povezane izdelke, kar izboljša ponudbo in prilagodljivost spletnih trgovin.
- **Content Management Server 2001**: za upravljanje vsebin dinamičnih poslovnih spletnih strani in skrajšuje čas potreben za izdelavo spletnih in intranetnih strani. Strežnik ponuja podporo večjezičnim okoljem, prilagoditev vsebine različnim napravam in popolno integracijo s strežnikom MS Commerce Server 2000.
- **Exchange Server 2000**: omogoča sporočanje kadarkoli in kjerkoli in upravljanje s sporočili. Nudi storitve za upravljanje stikov in nalog, za vodenje debatnih skupin, itd.

- **Host Integration Server 2000**: omogoča integracijo novih aplikacij in rešitev z obstoječimi sistemi in dostop do njihovih podatkovnih baz
- **Internet Security and Acceleration Server 2000**: namenjen je nastavitvam predpomnilnika in požarnih zidov za bolj varno in hitro povezovanje v Internet. Z njim lahko tudi nadzorujemo razpoložljivost sistema in upravljamo z razporejanjem bremena prometa.
- **Mobile Information 2001 Server**: omogoča podporo aplikacijam za mobilne enote – GSM, dlančniki, itd. Učinkovito posreduje Microsoft .NET Enterprise aplikacije, podatke ter intranetne vsebine mobilnemu uporabniku in skupaj z orodji Outlook Mobile Access in Outlook Mobile Manager tvori zaključeno rešitev.
- **SharePoint Portal Server 2001**: za izgradnjo in upravljanje portalov ter intranet strani, s katerimi uredimo in lažje najdemo informacije razdeljene pa našem sistemu (datoteke iz datotečnega strežnika, HTML strani iz spletnega strežnika, elektronsko pošto iz sporočilnega strežnika)



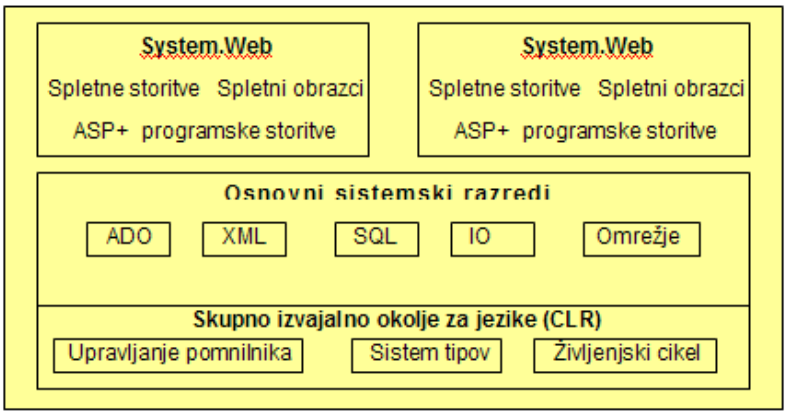
Slika: Vloga .NET strežnikov

Ogrodje .NET (.NET Framework)

Ogrodje .NET je največja prelomnica v zgodovini Microsofta in je najpomembnejši del celotne arhitekture .NET. Radikalno posega v samo jedro platforme Windows in določa arhitekturo, ki strogo temelji na konceptih predmetne tehnologije in komponentnega razvoja (temelji, ki jih je postavil COM+) in je okolje za razvoj, vzpostavitev in izvajanje spletnih in drugih aplikacij. Ogrodje .NET skrbi za nadzorovano izvajanje .NET aplikacij, nalaganje in izvajanje vmesne kode, za upravljanje in zaščito virov (procesor, pomnilnik, ...) in nudi podporo razvoju in razhroščevanju.

Osnova je skupno izvajalno okolje za programske jezike (CLR-Common Language Runtime), ki vpeljuje vmesni jezik (IL-Intermediate Language), ki izvorno kodo programa prevede v vmesno kodo in ne več v strojno kodo. S pomočjo prevajalnika JIT (Just-in-Time) se le ta prevede v strojno kodo. Tako se v vmesni jezik prevajajo vsi programski jeziki pisani za .NET. Omogočeno je dedovanje od predmeta pisanega v drugem programskem jeziku, saj edino vmesni jezik predstavlja omejitve, kaj je možno in kaj ne.

Velika prednost CLR in IL je tudi nadzorovano izvajanje programov (koncept peskovnika). V obstoječih Windowsih ni načina, kako preprečiti programom dostop do sistemskih sredstev, saj je vsa programska oprema prevedena v strojno kodo. Pri CLR pa je v vsakem trenutku možno omejiti dostop in s tem se seveda poveča varnost sistema.



Slika: Arhitektura ogrodja .NET

Širok je tudi nabor programskih storitev, ki nam jih ponuja ogrodje .NET. Razdelimo jih lahko na dva dela: na osnovne infrastrukturne storitve, ki vključujejo vhodno/izhodni (I/O) vmesnik, XML, podporo omrežjem in podporo dostopu do baz podatkov (ADO.NET). Nad njimi ležita podpora spletnim in Windows aplikacijam, v obliki WebForms in WinForms. V ta del so vključene tehnologije, kot npr. ASP+ (Active Server Pages). ASP+ je popolnoma integriran z CLR, torej nismo več omejeni na skriptne jezike za pisanje aktivnih spletnih strani, ampak lahko uporabimo katerikoli z .NET skladen programski jezik. Torej se ASP+ strani prevedejo v vmesni jezik, kar pomeni boljše zmogljivosti v primerjavi s sedaj interpretiranimi ASP stranmi.

Prav tako je spremenjen tudi način pakiranja in nameščanja programske opreme. Vpeljan je pojem zbirke (Assembly), ki je pravzaprav množica samo-opisnih datotek, ki vsebujejo informacije o načinu namestitve, različici, referencah, itd. S tem naj bi bilo konec registrov in deljenih DLL-jev. Namestitev in odstranitev programov pa bo postalo tako preprosto opravilo kot je bilo v času DOS operacijskega sistema.

Z ogrodjem .NET si je Microsoft sposodil nekatere ideje pri tekmecih, uvedel pa tudi mnogo izvirnih rešitev, katerih pravo vrednost bo pokazal čas. Čaka pa Microsoft še veliko dela, saj bo moral predelati skoraj vso programsko opremo.

.NET odjemalci

.NET odjemalci so osebni računalniki, prenosni računalniki, telefoni, dlančniki, igralne konzole in ostale pametne naprave. Kar naredi te naprave »pametne«, je zmožnost njihovega dostopa do XML storitev. Omogočajo nam, da dostopamo do podatkov ne glede na lokacijo, tip in število odjemalcev

.NET spletne storitve

Številni proizvajalci bodo lahko ponudili svoje spletne storitve. Microsoft v tem trenutku že ponuja osnovno spletno storitev Passport, ki omogoča enkratno, enotno prijavo in identifikacijo uporabnika. Kmalu se bodo tej storitvi pridružile še mnoge druge temeljne storitve, ki nastajajo pod imenom .NET My Services (spletni koledar, spletna e-pošta, myProfile, myAddress, myWallet,…)

.NET uporabniške izkušnje

Med .NET uporabniške izkušnje se prištevajo spletne storitve, ki omogočajo uporabniku dostop do informacij in novega znanja. Razdeljene bodo glede na potrebe na dve skupini: prva za poslovne uporabnike in druga za posameznike. Microsoft bo v tej skupini ponudil informacije iz spletnih strani MSN in Visual Studio.

Področje infrastrukture in orodij ter področje strežnikov vsebujeta konkretne Microsoftove izdelke, medtem ko pri ostalih področjih lahko pri izdelavi sodelujejo vsi uporabniki teh storitev.