Sistemska programska oprema

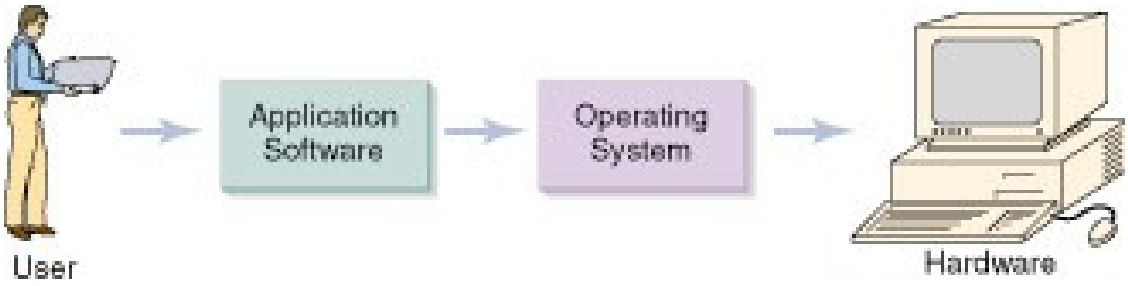
S.Divjak – Delovno gradivo, januar 2002

Vsebina:

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

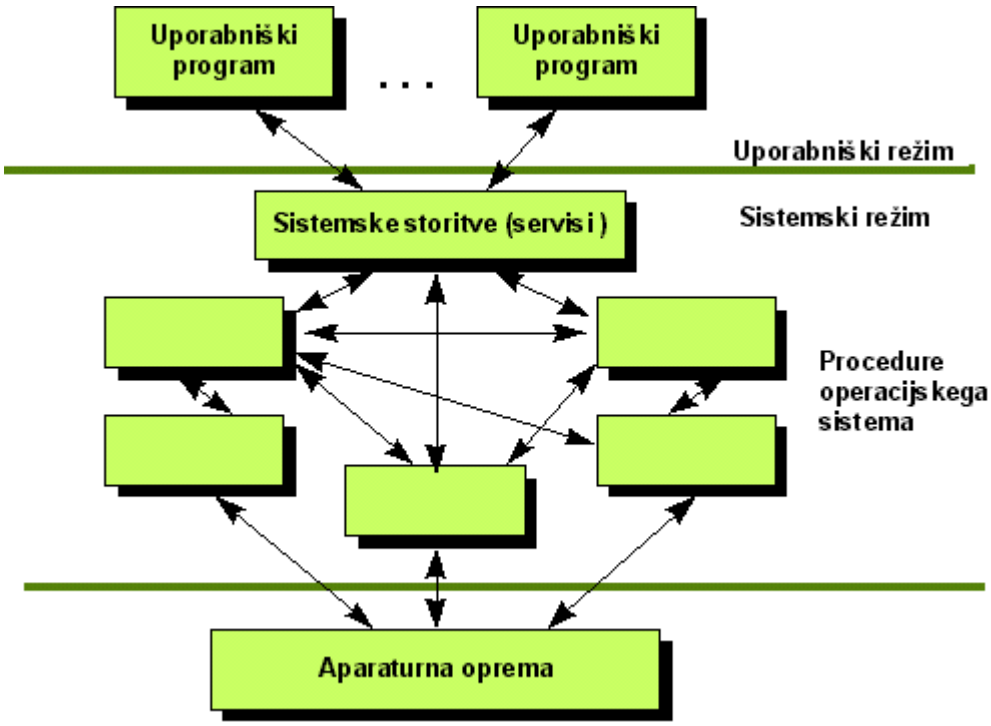
Operacijski sistem zagotavlja programom okolje in storitve, potrebne za njihovo izvajanje. Sestavlja ga več sistemskih komponent:

- 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.

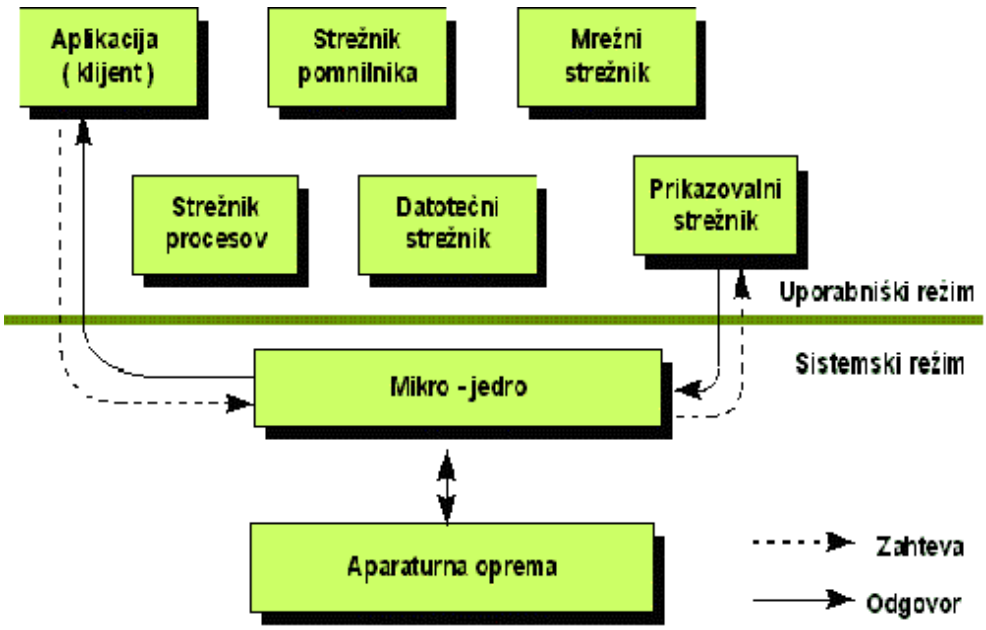
Praviloma pozna računalnik dva režima izvajanja nekega procesa: uporabniški režim (user mode) in sistemski režim (kernel mode, system mode). Normalno so naši programi v uporabniškem režimu. Šele ko to zahtevamo s primernim sistemskim klicem, preide proces v sistemski režim. V takem stanju izvedejo rutine 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.

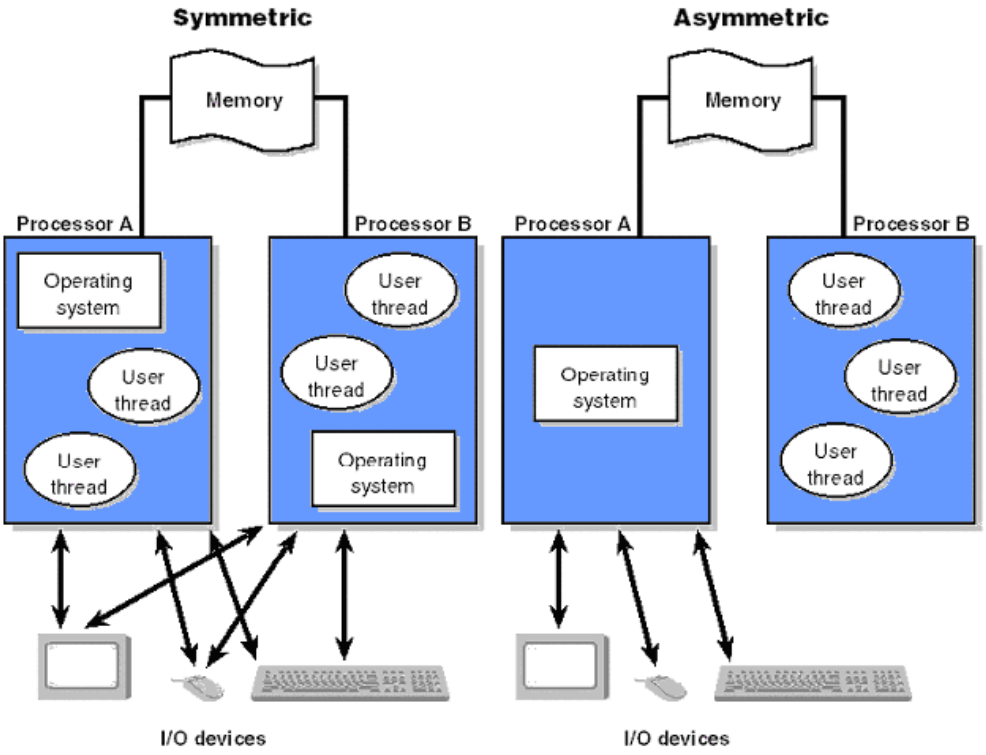
Do prekinitve programa lahko pride tudi zaradi aparaturne prekinitve (zahtevek z neke periferne naprave). Tudi v tem primeru pride do preklopa iz uporabniškega režima v sistemski (če se v tem še ne nahajamo) in do izvedbe ustrezne prekinitvene rutine.

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.



Command interpreter

When a user interacts with an Operating System they always do so through the intermediary of a command interpreter. This responds to user input in the following ways

- It starts applications
- It stops applications
- It allows the user to switch control between applications
- It may allow control over communication between an application and other applications or the user.

The command interpreter may be character-based, as in the MSDOS `COMMAND.COM` or the Unix shells.

The interpreter may be simple, or can have the power of a full programming language. It may be imperative (as in the Unix shells), use message passing (as in AppleScript) or use visual programming metaphors such as drag-and-drop for object embedding (as in Microsoft's OLE).

It is important to distinguish between the command interpreter and the underlying Operating System. The command interpreter may only use a subset of the capabilities offered by the Operating System; it may offer them them in a clumsy or sophisticated way; it may require complex skills or be intended for novices

Zagon in zaustavitev operacijskega sistema

To boot (as a verb; also "to boot up") a computer is to load an operating system into the computer's main memory or RAM (random access memory). Once the operating system is loaded (and, for example, on a PC, you see the initial Windows or Mac desktop screen), it's ready for users to run application programs. Sometimes you'll see an instruction to "reboot" the operating system. This simply means to reload the operating system (the most familiar way to do this on PCs is pressing the Ctrl, Alt, and Delete keys at the same time).

On larger computers (including mainframes), the equivalent term for "boot" is "Initial Program Load (IPL)" and for "reboot" is "re-IPL." Boot is also used as a noun for the act of booting, as in "a system boot." The term apparently derives from "bootstrap" which is a tool for getting your heel into a leather boot so that you can get the whole thing on. There is also an expression, "pulling yourself up by your own bootstraps," meaning to leverage yourself to success from a small beginning. The booting of an operating system works by loading a very small program into the computer and then giving that program control so that it in turn loads the entire operating system.

Booting or loading an operating system is different than installing it, which is generally an initial one-time activity. (Those who buy a computer with an operating system already installed don't have to worry about that.) When you install the operating system, you may be asked to identify certain options or configuration choices. At the end of installation, your operating system is on your hard disk ready to be booted (loaded) into random access memory, the computer storage that is closer to the microprocessor and faster to work with than the hard disk. Typically, when an operating system is installed, it is set up so that when you turn the computer on, the system is automatically booted as well. If you run out of storage (memory) or the operating system or an application program encounters an error, you may get an error message or your screen may "freeze" (you can't do anything). In these events, you may have to reboot the operating system.

How Booting Works

Note: This procedure may differ slightly for different operating systems.

When you turn on your computer, chances are that the operating system has been set up to boot (load into RAM) automatically in this sequence:

As soon as the computer is turned on, the Basic Input-Output System (BIOS) on your system's read-only memory (ROM) chip is "woken up" and takes charge. BIOS is already loaded because it's built-in to the ROM chip and, unlike RAM, ROM contents don't get erased when the computer is turned off.

BIOS first does a "power-on self test" (POST) to make sure all the computer's components are operational. Then the BIOS's boot program looks for the special boot programs that will actually load the operating system onto the hard disk.

First, it looks on drive A (unless you've set it up some other way or there is no diskette drive) at a specific place where operating system boot files are located. (If the operating system is MS-DOS, for example, it will find two files named IO.SYS and MSDOS.SYS.) If there is a diskette in drive A but it's not a system disk, BIOS will send you a message that drive A doesn't contain a system disk. If there is no diskette in drive A (which is the most common case), BIOS looks for the system files at a specific place on your hard drive.

Having identified the drive where boot files are located, BIOS next looks at the first *sector* (a 512-byte area) and copies information from it into specific locations in RAM. This information is known as the *boot record* or Master Boot Record.

It then loads the boot record into a specific place (hexadecimal address 7C00) in RAM.

The boot record contains a program that BIOS now branches to, giving the boot record control of the computer.

The boot record loads the initial system file (for example, for DOS systems, IO.SYS) into RAM from the diskette or hard disk.

The initial file (for example, IO.SYS, which includes a program called SYSINIT) then loads the rest of the operating system into RAM. (At this point, the boot record is no longer needed and can be overlaid by other data.)

The initial file (for example, SYSINIT) loads a system file (for example, MSDOS.SYS) that knows how to work with the BIOS.

One of the first operating system files that is loaded is a system configuration file (for DOS, it's called CONFIG.SYS). Information in the configuration file tells the loading program which specific operating system files need to be loaded (for example, specific device drivers.

Another special file that is loaded is one that tells which specific applications or commands the user wants to have included or performed as part of the boot process. In DOS, this file is named AUTOEXEC.BAT. In Windows, it's called WIN.INI.

After all operating system files have been loaded, the operating system is given control of the computer and performs requested initial commands and then waits for the first interactive user input.

The boot procedure in UNIX is as follows:

- The memory runs a self test
- Probes the bus for a bootable device

- Reads the boot device for a boot program
- Boot program reads the the kernel and passes it control of the system
- The kernel identifies and configures the devices
- Initializes the system and starts the system processes
- Runs the appropriate start-up scripts depending on the mode

The shutdown procedure in UNIX is as follows:

- The users on the machine are notified of the shutdown
- All running processes are given a signal telling them to stop
- All the sub systems go down
- Remaining users are logged off
- The system is either halted, rebooted, or booted into single user mode

The boot procedure in Windows NT is as follows:

- NTLDR (NT Loader) looks for boot.ini, ntdetect.com, bootsect.dos, and ntbootdd.sys.
- NTDETECT figures out what hardware you have
- Loads the kernel --> The kernel initialization phase, services phase, subsystem start phase.

The shutdown procedure for Windows NT is as follows:

- Select Start --> Shutdown and you will be prompted to select on of the following
- Shutdown the computer
- Restart the computer
- Close all programs and log on as a different user

Delo z operacijskim sistemom UNIX

V tem poglavju bomo spoznali koncept dela v večuporabniškem okolju. Pri tem se bomo naslonili na popularni operacijski system UNIX.

Delo na večuporabniških sistemih, kot je UNIX, predvideva, da mora imeti posameznik dovoljenje za delo na sistemu. Sistemski operater mora novega uporabnika vpisati v seznam uporabnikov. Skupaj izbereta **uporabniško ime**. To je beseda, po kateri sistem "spozna" uporabnika. Za zaščito svojih podatkov pa dobi novi uporabnik tudi ustrezno geslo, ki ga kasneje lahko sam spreminja. Ob vpisu imena novega uporabnika se v datotečnem sistemu odpre nov, uporabniku namenjen direktorij (njegov **domači** direktorij). Določi se tudi tip lupine, ki jo bo uporabnik uporabljal za svoj pogovor z računalnikom. V nadaljevanju bomo predvideli, da je uporabniku na voljo lupina ***ksh***. Zaenkrat predpostavimo, da je sistemski operater že vključil računalnik in pognal operacijski sistem. Delo posameznega uporabnika se začne z vklopom terminala, na katerem se pojavi napis

```
login:
```

Uporabnik vpiše svoje (uporabniško) ime, zatem pa na vprašanje

```
password:
```

še veljavno geslo. Če je bilo ime in geslo pravilno, izbere UNIX domači (HOME) direktorij uporabnika in prepusti nadaljnjo komunikacijo z uporabnikom takoimenovani ***lupini*** (shell). Delo normalno poteka interaktivno. Lupina izpiše ***najavko*** (prompt), na katero mora uporabnik odgovoriti z ukazno vrstico v skladu z dogovorjeno slovnico. Najavka je običajno znak ***$*** ali ***%*** .

Ukazno vrstico normalno sestavljajo ukaz in določeno število argumentov, ki so med seboj ločeni s presledki.

Delo z računalnikom zaključimo z vtipkanjem ukaza

```
$ exit
```

Ko ukazno vrstico zaključimo s tipko ***ENTER*** (ali z *RETURN*), bo lupina poklicala ustrezni uslužnostni program, ki bo izpeljal želeno akcijo.

Z ukazi lahko rokujemo z datotekami in direktoriji, kličemo lastne programe in podobno.

Primer:

```
$ lp pismo
```

Pri tem je ***lp*** ukaz, ***pismo*** pa je argument, ki je v našem primeru ime neke datoteke. Po vtipkanju *ENTER* bo prišlo do izpisa vsebine datoteke *pismo* na tiskalniku.

Če ukazno vrstico zaključimo z znakom ***&*** pred tipko *ENTER*, pomeni to zahtevek, da lupina sicer sproži izvajanje podanega ukaza (oziroma njemu ustreznega procesa), vendar se tudi

takoj spet oglasi z najavko in čaka na nov ukaz. Prvi ukaz bo torej izvajan paralelno, pravzaprav v ***ozadju*** (background) naše interakcije z lupino.

Primer:

```
$ lp pismo &
1234
$
```

Vidimo, da računalnik v odgovor na našo ukazno vrstico najprej napiše neko številko, nato pa že omenjeno najavko ***$*** . Številka je v bistvu oznaka programa (oziroma pravilneje ***procesa***), ki teče v ozadju. Tej oznaki strokovno pravimo ***PID*** (process identification number). Pri vrsti ukazov lahko navedemo med argumenti tudi takoimenovana ***opcijska stikala*** (option switches). Ta se pri sistemu UNIX normalno začenjajo s pomišljajem -, kateremu sledi črka.

Primer:

```
$ lp -m pismo
```

Tu stikalo -***m*** pomeni, da zahtevamo (od poštnega servisa *mail*) obvestilo, ko bo izpis zaključen. Nabor in pomen posameznih opcijskih stikal je odvisen od posameznega ukaza.

Včasih moramo vtipkati ukaz, ki je daljši od ene vrstice na zaslonu. V tem primeru jo moramo zaključiti z znakom ***C** (in nato *ENTER*), kar lupini pove, da mora pred klicem ustreznega programa prebrati še naslednjo vrstico.

Po želji lahko v isti vrstici vpišemo tudi več ukazov. Ti morajo tedaj biti ločeni s podpičji.

Primer:

```
$ lp pismo; ls *
```

V isti vrstici smo zahtevali izpis pisma s tiskalnikom in nato izpis imen vseh datotek v tekočem direktoriju.

Pogosto se zgodi, da želimo nek program predčasno prekiniti. To dosežemo s prekinitveno tipko, ki je običajno kombinacija ***CTRL/C***, lahko pa je v ta namen uporabljena tudi tipka ***delete*** ali ***break*** ali ***rubout***. Proces, ki teče v ozadju, pa prekinemo z ukazom ***kill***, pri čemer moramo navesti že omenjeno oznako *PID* danega procesa.

Primer:

```
$ kill 1234
```

Spoznali smo, da je lupina UNIX interpreter ukazov. Splošna oblika ukaza lupini je

```
ukaz A1 A2 .. An
```

Pri tem je *ukaz* ime danega ukaza, *A1,A2,..An* pa so imena argumentov, ki so med seboj ločeni z enim ali več presledki. Argumenti so največkrat imena datotek ali direktorijev ali opcijska stikala.

Izvedba procedure lupine je tesno navezana na tri datoteke, ki jih lupina odpre ali tvori. Te so: ***standardni vhod*** (*standard input*), ***standardni izhod*** (*standard output*) in ***izhod za obvestila o napakah*** (*standard error output*). Vsaki od teh datotek je prirejena številčna oznaka, ki ji pravimo ***datotečna številka*** (*file descriptor*). Vrednosti teh oznak so 0, 1 oziroma 2. Datoteka s številko 0 je vhodna, datoteki 1 in 2 pa sta izhodni. Med izvajanjem programov se vhodno-izhodne operacije usmerijo na standardno vhodno oziroma izhodno datoteko, v primeru, ko eksplicitno ne navajamo imena neke datoteke. Na datoteko za izpis napak pa se zapisujejo obvestila sistema UNIX o napakah med izvajanjem programa.

Standardni vhod, izhod in izpis napak so normalno prirejeni tipkovnici oziroma zaslonu uporabnikovega terminala. (UNIX gleda na periferne naprave kot na datoteke (posebne vrste) v svojem datotečnem sistemu. Normalno (če ni drugače definirano) velja:

- standardni vhod .....tipkovnica
- standardni izhod .....zaslon

- zapis napak.....zaslon

Značilnost lupine je prav v tem, da lahko katerikoli standardno datoteko nadomesti s kakršnokoli drugo datoteko. Tako lahko določimo, da naj klicani program bere vhodne podatke z datoteke namesto s tastature. To dosežemo s klicem naslednje oblike:

```
ukaz A1 A2 ..An <podatki
```

Pri tem je *podatki* ime vhodne datoteke. Analogno lahko zahtevamo, da naj program zapisuje rezultate v neko datoteko namesto na zaslon. To dosežemo s programskim klicem naslednje oblike:

```
ukaz A1 A2 .. An > izhod
```

Pri tem je *izhod* ime datoteke, v katero se bodo zapisovali izpisi, ki bi jih sicer dobili na zaslonu.

Če želimo preusmeriti standardni izhod za izpis obvestil o napakah, si moramo pomagati s številko ustrezne izhodne datoteke (ta pa je enaka 2). Klic oblike

```
ukaz A1 A2 .. An 2> napake
```

bi povzročil izpis morebitnih obvestil o napakah v datotekol *napake*.

Ključna posledica teh lastnosti je možnost veriženja več programskih procesov tako, da rezultate enega avtomatsko uporabimo kot vhodne podatke drugemu. Pri tem lahko potekajo taki procesi sočasno oziroma paralelno. To je pomembna značilnost sistema UNIX in izhaja iz njegove splošne filozofije. V skladu z le-to naj bi namesto splošnih in zato kompleksnih programov uporabljali raje množico preprostih in hkrati učinkovitih funkcij, ki bi jih med seboj poljubno povezovali.

Kot je običaj pri interaktivnih operacijskih sistemih, lahko pogosto uporabljano zaporedje ukazov vpišemo v tekstovno datoteko, iz katere lahko kasneje lupina razbira in izvaja tako programirane akcije. Taki datoteki recimo ***ukazna datoteka*** (*shell script*).

Omenimo naj še, da lupina lahko uporablja tako numerične kot tekstovne spremenljivke, pogojne skoke in druge konstrukte, ki jih uporabljamo predvsem v sklopu ukaznih datotek.

Lupine za delo z OS, njihovo programiranje

Lupina je interpreter ukaznih vrstic. Ko jih razpozna, jih izvede. V bistvu pomeni pisanje za lupino kar neko vrsto programiranja.

Tvorba ukazne datoteke

Recimo, da moramo pogosto tipkati naslednji ukaz:

```
find . -name file -print
```

Izmislimo si lahko bolj preprost ukaz, na primer naslednje oblike

```
sfind file
```

napišimo zato takoimenovano ukazno datoteko za lupino (shell script)

```
% cd ~/bin
% emacs sfind
% page sfind
find . -name $1 -print
% chmod a+x sfind
% rehash
% cd /usr/local/bin
% sfind tcsh
./shells/tcsh
```

Morda to ni didaktično najbolj čist primer, vendar mu dodajmo nekaj komentarjev :

```
#Shell scripts are simple text files created with an editor.
#Shell scripts are marked as executeable
```

```
%chmod a+x sfind
#Should be located in your search path and ~/bin should be in your search path.
#You likely need to rehash if you're a Csh (tcsh) user (but not again when you login).
#Arguments are passed from the command line and referenced. For example, as $1.
```

#!/bin/sh

Vse ukazne datoteke za lupino Bourne moramo začenjati z naslednjo vrstico

```
#!/bin/sh
```

Za znakoma "#!" je ime programa, ki mora interpretirati vsebino udatoteke. Ta program je seveda lahko naša lupina. To vrstico včasih izpuščamo, vendar je bolje, da jo navedemo, saj poznamo raztlične vrste ukaznih interpreterjev: Csh, Ksh, Bash itd.

Komentarji

Komentarje začenja znak #. Komentar lahko začnemo kjerkoli v posamezni vrtstici in velja do konca vrstice.

Pot iskanja (Search Path)

Datoteke lahko vsebujejo specifikacijo poti iskanja programov:

```
PATH=/usr/ucb:/usr/bin:/bin; export PATH
```

Lupina Bourne ne izvozi okoljskih spremenljivk k svojim procesom - otrokom, v kolikor tega eksplicitno ne zahtevamo z ukazom ***export.***

Preverjanje argumentov

Dobro zasnovana ukazna datoteka vsebuje verifikacijo morebitno posredovanih argumentov .

```
if [ $# -ne 3 ]; then
    echo 1>&2 Usage: $0 19 Oct 91
    exit 127
fi
```

V tem primeru preverjamo, ali smo ukazni datoteki posredovali natančno tri argumente. V primeru, da ni tako, pride do izstopa z izbrano kodo.

Status Exit

Vsi uslužnostni programi (utilities) UNIX vračajo izstopni status.

```
# is the year out of range for me?
if [ $year -lt 1901  -o  $year -gt 2099 ]; then
    echo 1>&2 Year \"$year\" out of range
    exit 127
fi
etc...
# All done, exit ok
exit 0
```

Po dogovoru pomeni izstopni status, različen od 0, da je nekaj narobe in pove kodo napake. To kodo lahko uporabimo v pogojnih stavkih jezika komandne lupine.

Primer takega pogojnega stavka:

```
if command; then
    command
fi
```

Ali še bolj konkretno:

```
if tty -s; then
    echo Enter text end with \^D
fi
```

Stdin, Stdout, Stderr

Standard input, output, in error so datotečni opisniki (file descriptors) 0, 1, and 2 in pomenijo standardni vhod in izhod oziroma izhod za beleženje napak.

Primeri:

```
# is the year out of range for me?
if [ $year -lt 1901  -o  $year -gt 2099 ]; then
    echo 1>&2 Year \"$year\" out of my range
    exit 127
fi
etc...
# ok, you have the number of days since Jan 1, ...
case `expr $days % 7` in
0)
    echo Mon;;
1)
    echo Tue;;

etc...

# give the fellow a chance to quit
if tty -s ; then
```

```
    echo This will remove all files in $* since ...
    echo $n Ok to procede? $c;      read ans
    case "$ans" in
        n*|N*)
echo File purge abandoned;
exit 0   ;;
    esac
    RM="rm -rfi"
else
    RM="rm -rf"
fi
```

Seveda je lahko standardni vhod preusmerjen v cevovod (pipe) ali v neko datoteko.

Jezikovni konstrukti lupine

Stavek for

Tako tvorimo ponavljanje zaporedja stavkov. Pri tem prihaja do zamenjave vrednosti spremenljivke, navedene v stavku for:

```
for variable in word ...
do
    command
done
```

Na primer:

```
for i in `cat $LOGS`
do
        mv $i $i.$TODAY
        cp /dev/null $i
        chmod 664 $i
done
```

Lahko bi imeli tudi zaporedje stavkov naslednje oblike:

```
for variable in word ...; do command; done
```

Case

Tu pride do programske izbire glede na ujemanje vzorca. V splošnem velja naslednja sintaksa:

```
case word in
[ pattern [ | pattern ... ] )
    command ;; ] ...
esac
```

In konkreten primer:

```
case "$year" in

[0-9][0-9])
        year=19${year}
        years=`expr $year - 1901`
        ;;
[0-9][0-9][0-9][0-9])
        years=`expr $year - 1901`
        ;;
*)
        echo 1>&2 Year \"$year\" out of range ...
        exit 127
        ;;
esac
```

Pogojno izvajanje

Preverimo izstopni status ukaza (programa) in izvedemo ustrezno razvejitev

```
if command
then
    command
[ else
    command ]
fi
```

Na primer:

```
if [ $# -ne 3 ]; then
        echo 1>&2 Usage: $0 19 Oct 91
        exit 127
fi
```

Ali pa kar v eni vrstici:

```
if command; then command; [ else command; ] fi
```

Iteracija While/Until

Ponavljamo ukaz (command) dokler ta vrača izstopni status 0.

```
{while | until} command
```

```
do
    command
done
```

Na primer:

```
# for each argument mentioned, purge that directory
while [ $# -ge 1 ]; do
        _purge $1
        shift
done
```

Ali zapisano v eni vrstici :

```
while command; do command; done
```

Spremenljivke

Imena spremenljivke+ so zaporedja črk, številk in podčrtajev. Začnejo s črko ali podčrtajem. Če hočemo dobiti vrednost take spremenljivke, moramo pred njeno ime dodati znak $.

Posebnost so pozicijske spremenljivke (kot na primer $1) ki so v bistvu posredujejo vrednost argumenta, navedenega v klicu ukazne datoteke.

Prirejanje vrednosti spremenljivkam

Za prirejanje vrednosti spremenljivkam uporabljamo stavek oblike

```
variable=value
```

Na primer:

```
PATH=/usr/ucb:/usr/bin:/bin; export PATH
```

ali

```
TODAY=`(set \`date\`; echo $1)`
```

Izvažanje vrednosti spremenljivk

Kot žeomenjeno, nekatere lupine ne posredujejo (izvažajo) vrednosti spremenljivk svojim otrokom (proženim programom), razen, če to eksplicitno zahtevamo s stavkom ***export***:

```
# We MUST have a DISPLAY environment variable
if [ "$DISPLAY" = "" ]; then
              if tty -s ; then
              echo "DISPLAY (`hostname`:0.0)? \c";
              read DISPLAY
            fi
if [ "$DISPLAY" = "" ]; then
            DISPLAY=`hostname`:0.0
fi
                  export DISPLAY
fi
```

Naslavljanje spremenljivk

Če nas zanima vrednost neke spremenljivke, uporabimo obliko ***$variable*** (ali včasih ${variable}).

```
# Most user's have a /bin of their own
if [ "$USER" != "root" ]; then
      PATH=$HOME/bin:$PATH
else
      PATH=/etc:/usr/etc:$PATH
fi
```

Zavite oklepqaje uporabljamo pri konkatenaciji (veriženju).

```
$p_01
```

Tako dobimo vrednost spremenljivke "p_01".

```
${p}_01
```

Tako pa vrednost spremenljivke "p" , ki ji na koncu pripnemo "_01".

Pogojno naslavljanje spremenljivk

```
${variable-word}
```

Tako povemo, da naj lupina uporabi spremenljivko variable, če je ta setirana, sicer pa naj uporabi word.

```
POSTSCRIPT=${POSTSCRIPT-PostScript};
export POSTSCRIPT
```

${variable:-word}

Lupina naj uporabi spremenljivko variable, če je setirana in ni nič, sicer pa naj uporabi word.

${variable:?word}

Lupina naj uporabi spremenljivko variable, če je setirana in ni nič, sicer pa naj izpiši word in izstopi.

Argumenti

Argumenti v ukazni vrstici lupine, ki kliče ukazno datoteko, so pozicijske spremenljivke

$0, $1, ...

Pri tem je $0 sam ukaz, ostalo pa so pravzaprav argumenti. Število argumentov pove spremenljivka

$#

Na voljo imamo tudi vse argumente v istem nizu, ločene s presledki

$*, $@

Pa še nekaj ukazov s tem v zvezi:

shift

Ta ukaz pomakne vse pozicijske spremenljivke za eno mesto v levo in zmanjša število argumentov za 1. Uporabimo ga zato pri pregledovanju oziroma razpoznavanju zapisa v ukazni vrstici:

```
# parse argument list

while [ $# -ge 1 ]; do
        case $1 in
    process arguments...
        esac
        shift
done
```

Tako pa lahko priredimo pozicijskim spremenljivkam vrednosti v seznamu argumentov:

```
set arg arg ...
```

Primer uporabe:

```
# figure out what day it is
  TODAY=`(set \`date\`; echo $1)`
  cd $SPOOL
  for i in `cat $LOGS`
  do
        mv $i $i.$TODAY
        cp /dev/null $i
        chmod 664 $i
  done
```

Posebne spremenljivke

$$

Ta spremenljivka vsebuje PID tekočega procesa. Uporabljamo jo na primer pri tvorbi začasnih datotek:.

```
tmp=/tmp/cal0$$
trap "rm -f $tmp /tmp/cal1$$ /tmp/cal2$$"
trap exit 1 2 13 15
/usr/lib/calprog >$tmp
```

$?

Ta spremenljivka vsebuje izstopno kodo zadnjega ukaza.

$command

```
# Run target file if no errors and ...
if [ $? -eq 0 ]
then
 etc...
fi
```

Posebni znaki

Posebni znaki končujejo besede:

```
; & ( ) | ^ < > new-line presledek tab
```

Uporabljamo jih pri zaporedjih stavkov (oziroma ukazov), poslih v ozadju (background jobs) itd. Če hočemo tak znak navesti kot običajen znak, uporabimo vzvratno poševno črto (\) ali pa kot navednice ("" oziroma ").

Enojne navednice

Z enojnimi navednicami navajamo vse znake, vključno \ . Tako dobimo eno besedo.

```
grep :${gid}: /etc/group | awk -F: '{print $1}'
```

Dvojne navednice

V dvojnih navednicah imamo zamenjavo spremenljivk. .

```
if [ ! "${parent}" ]; then
  parent=${people}/${group}/${user}
fi
```

Vzvratne navednice

Z njimi zahtevamo izvedbo nekega ukaza in zamenjavo izhoda.

```
if [ "`echo -n`" = "-n" ]; then
 n=""
 c="\c"
else
 n="-n"
 c=""
fi
```

in še en primer:

```
TODAY=`(set \`date\`; echo $1)`
```

Funkcije

Funkcije so zelo uporabne, čeprav jih ne srečamo prav pogosto. Imajo naslednjo obliko:

```
name ( )
```

```
{
    commands
}
```

Primer:

```
# Purge a directory
_purge()
{
        # there had better be a directory

        if [ ! -d $1 ]; then
 echo $1: No such directory 1>&2
 return
        fi

    etc...
}
```

Znotraj take funkcije so pozicijski parametric $0, $1, itd. Argumenti funkcije in ne argumenti ukazne datoteke. Funkcija zaključujemo z return in ne z exit. Smisel funkcij je enkapsulacija kode.

Še nekaj trikov

Test

To je zelo uporaben ukaz. Njegova oblika je na primer taka:

```
if test expression; then
    etc...
```

kar lahko zapišemo tudi tako

```
if [ expression ]; then
    etc...
```

Nekaj uporabnih izrazov

```
test { -w, -r, -x, -s, ... } filename
```

kar pomeni, da se sprašujemo, ali je datoteka zapisljiva, berljiva, izvršljiva, prazna itd

```
test n1 { -eq, -ne, -gt, ... } n2
```

Preverjamo, ali je število n1 enako, neenako, večje kot,..

```
test s1 { =, != } s2
```

Ali sta niza enaka ali različna?

```
test cond1 { -o, -a } cond2
```

Binarna ali in in. Negacijo dobimo z znakom !.

Primer:

```
if [ $year -lt 1901  -o  $year -gt 2099 ]; then
    echo 1>&2 Year \"$year\" out of range
    exit 127
fi
```

Izpis najavke

Na sistemih BSD lahko sprožimo izpis najavke na naslednji način:

```
echo -n Ok to procede?;  read ans
```

Isto dosežemo na sistemu V tako:

```
echo Ok to procede? \c; read ans
```

(Interaktivno) branje

Vhod lahkopreusmerimo na naslednji način (lahko pa tudi kako drugače):

```
# take standard input (or a specified file) and do it.
if [ "$1" != "" ]; then
      cat $1 | do_file
else
      do_file
fi
```

Rokovanje z nizi

Nekaj uporabnih trikov

```
TIME=`date | cut -c12-19`
```

```
TIME=`date | sed 's/.* .* .* \(.*\) .* .*/\1/'`
TIME=`date | awk '{print $4}'`
TIME=`set \`date\`; echo $4`
TIME=`date | (read u v w x y z; echo $x)`
```

Testiranje ukaznih datotek

Testiranje ukaznih datotek omogoča uporaba različnih zaqstavic (flags) v ukazni vrstici:

```
sh -n command
```

Tako zahtevamo branje, ne pa izvajanja ukazne datoteke. Torej preverjamo le njeno sintakso.

```
sh -x command
```

Tako zahtevamo prikaz ukazov in argumentov med izvajanjem ukazne datoteke.

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.

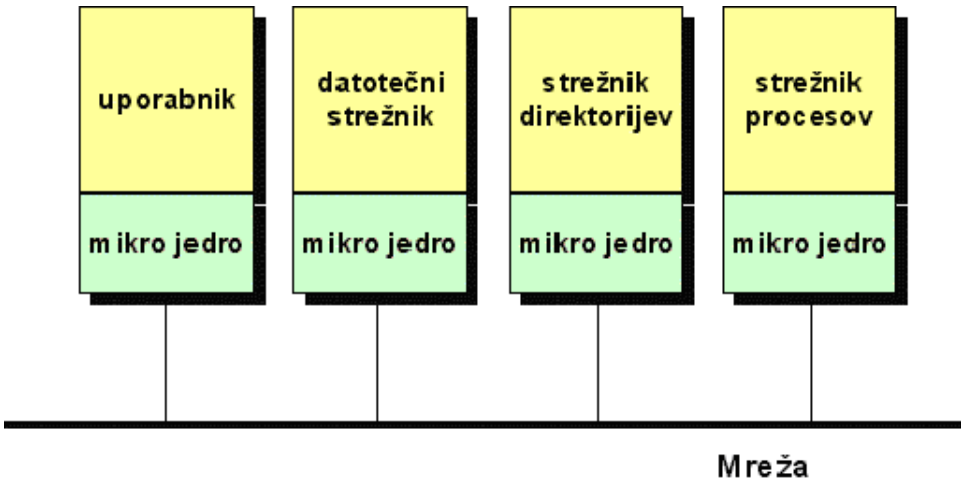
- Zanesljivost

- 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.

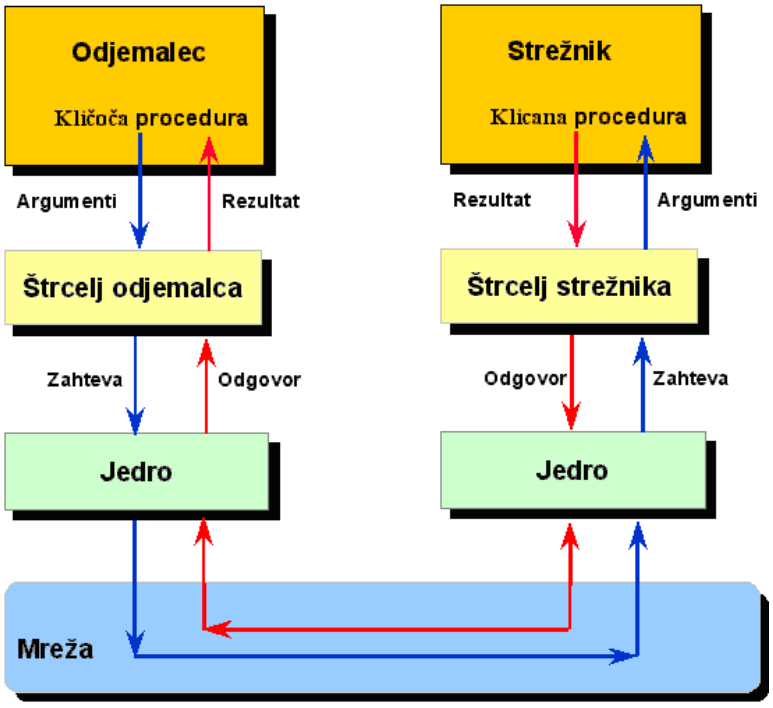
Komunikacija v porazdeljenih sistemih

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

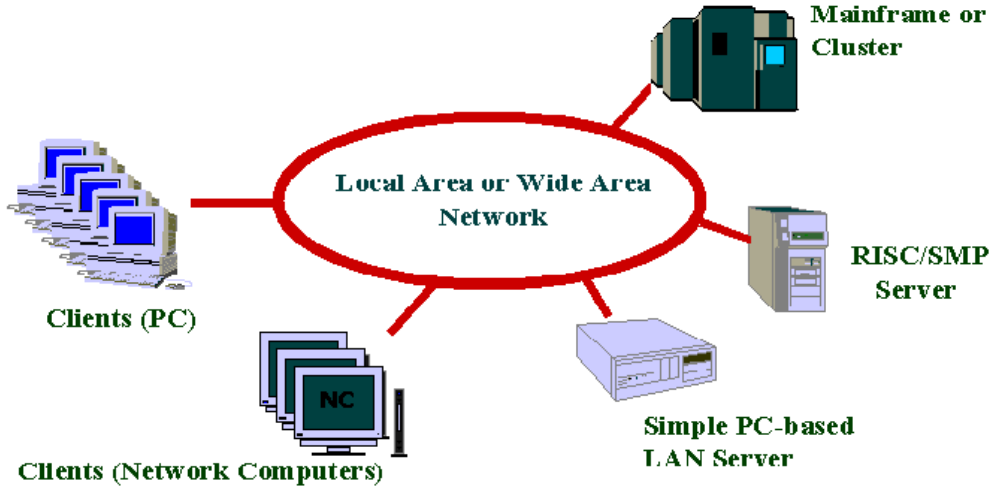
(client stub), ki igra vlogo nekakąnega vmesnika. Analogno proceduro, štrcelj strežnika (server stub) imamo na strani strežnika. Oddaljen klic procedure poteka nato na naslednji način:

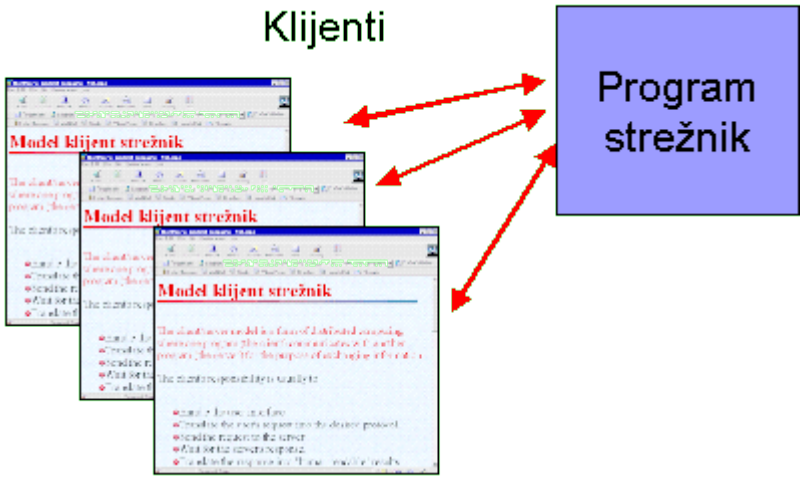
- Proces-odjemalec pokliče svoj štrcelj na normalni način (prenos parametrov in povratnega naslova na lastni sklad).
- Štrcelj odjemalca formira obvestilo (message) tako, da vanj pakira parametre in s primernim sistemskim klicem pokliče jedro.
- Jedro pošlje obvestilo oddaljenemu jedru.
- Oddaljeno jedro 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

Client/Server is a distributed computing paradigm, in which data storage is centralized on a server and user interaction is handled by different clients.





The client/server model is a form of distributed computing where one program (the client) communicates with another program (the server) for the purpose of exchanging information.

The client's responsibility is usually to:

- Handle the user interface.
- Translate the user's request into the desired protocol.
- Send the request to the server.
- Wait for the server's response.
- Translate the response into "human-readable" results.
  Present the results to the user.

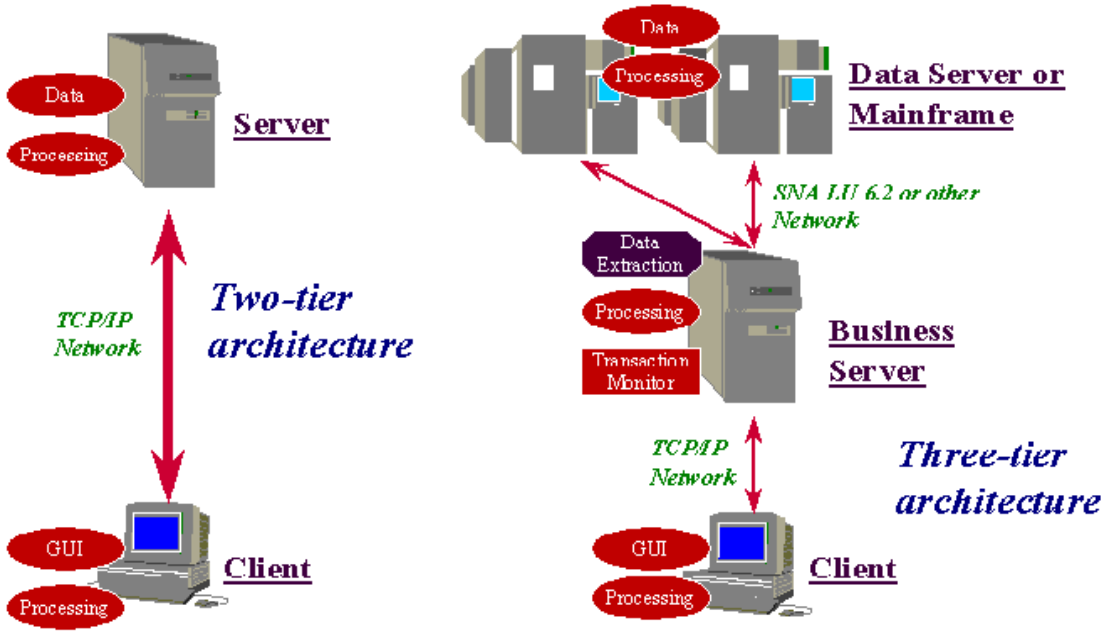
The server's functions include:

- Listen for a client's query.
- Process that query.
- Return the results back to the client.
- A typical client/server interaction goes like this:
- The user runs client software to create a query.
- The client connects to the server.
- The client sends the query to the server.
- The server analyzes the query.
- The server computes the results of the query.
- The server sends the results to the client.
- The client presents the results to the user.
- Repeat as necessary.

Flexible user interface development is the most obvious advantage of client/server computing. It is possible to create an interface that is independent of the server hosting the data. This allows information to be stored in a central server and disseminated to different types of remote computers. Since the user interface is the responsibility of the client, the server has more computing resources to spend on analyzing queries and disseminating information. This is another major advantage of client/server computing; it tends to use the strengths of divergent computing platforms to create more powerful applications. Although its computing and storage capabilities are dwarfed by those of the mainframe, there is no reason why a Macintosh could not be used as a server for less demanding applications.

In short, client/server computing provides a mechanism for disparate computers to cooperate on a single computing task.

Arhitekture klijent strežnik



Advantages of Client/Server System Architectures :

Client/server is an open system. The advantages of this environment include:

- **Interoperability** - key components (client/network/server) work together
- **Scalability** - any of the key elements may be replaced when the need to either grow or reduce processing for that element dictates, without major impact on the other elements.

- **Adaptability** - new technology (i.e., multi-media, broad band networks, distributed database, pen computing, etc.) may be incorporated into the system.

- **Affordability** - cost effectiveness is insured by using less expensive MIPs available on each platform.

- **Data Integrity** - entity, domain and referential integrity are maintained on the database server.

- **Accessibility** - data may be accessed from WANs and multiple client applications.

- **Performance** - performance may be optimized by hardware and process.

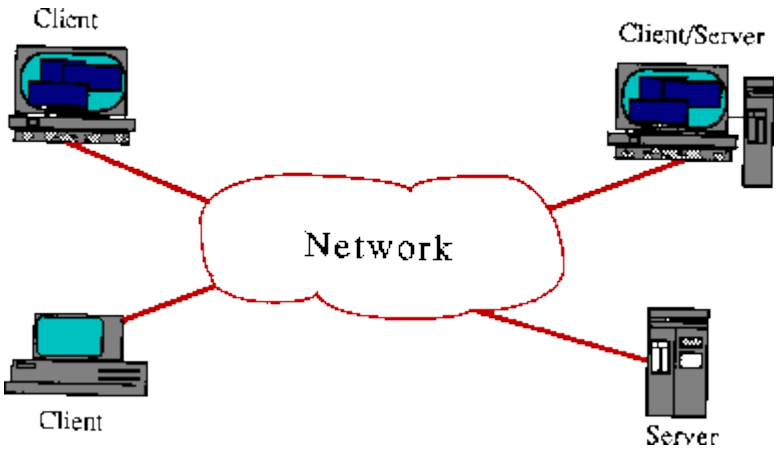
- **Security** - data security is centralized on the server.

- **Vendor independence**

Heterogeni sistemi

In a client/server model, a node in a heterogeneous computing environment (e.g., personal computer, workstation, minicomputer, mainframe) may be characterized either as a client, a server, or a client/server.

A client node provides no services to any other node. A client is only able to be the recipient of services provided by other nodes. A node that provides services is a server. Often a server can also be a client. Such a node is referred to as a client/server. The term client may also refer to a process which runs on a client node or to the person on whose behalf a client process is acting. Similarly, the term server may also refer to a process which runs on a server node. In this report, the term client refers to a client node and the term server refers to a server node.



In the figure, a personal computer and a workstation are clients. They provide no services to any other node, but they may be the recipient of a service, such as mounting a file system or directory, from either the server or client/server. The client/server may also be the recipient of services.

Originally, nodes of computer networks were exclusively mainframes and minicomputers. The user accessed the network only by means of a terminal connected to one of the nodes. The services provided by the network were usually login (sometimes called **virtual terminal** or *remote login*), file transfer, and mail. Each node of the network was a client/server for these three services. Users were provided with an interactive terminal session with any node on the network. They were able to transfer files between any two nodes. Finally, they were able to send mail to any node and receive mail from any node.

With the advent of the personal computer, networks were implemented whose nodes were exclusively small computers. Such a *personal computer* network provided **file service** and **mail**. Each node of a personal computer network was usually either a client or a server and typically, there were no client/server nodes. There was no login service provided since each personal computer ran applications locally. Although applications may be downloaded from a

server, their primary function was to provide file service for data (i.e., transparent access to data on mass storage). File transfer between clients was accomplished by copying files to and from a server. Additionally, servers sometimes acted as a post office for electronic mail. Clients accessed the mail server to send and receive mail.

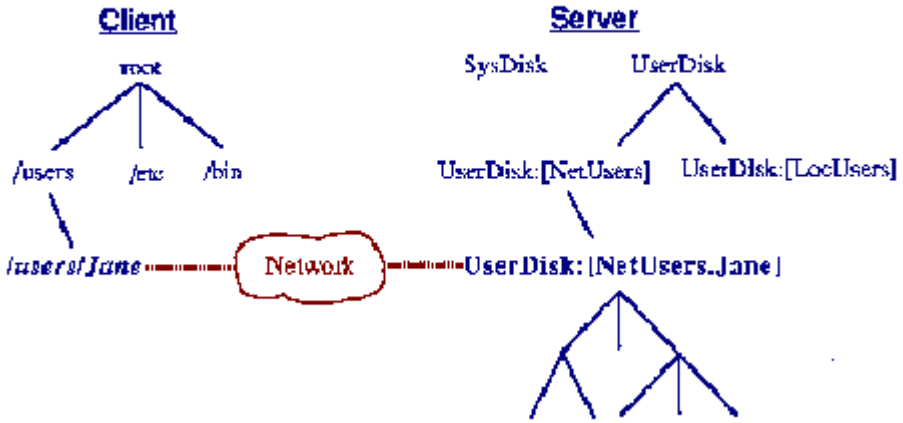


Figure: Client access to remote mass storage on server.

This distinction between large computer networks and personal computer networks has somewhat disappeared. In a heterogeneous computing environment, the services of the large computer network and the personal computer network are integrated into three basic types of services which are available to all sizes of clients from all sizes of servers or client/servers.

In a heterogeneous computing environment, clients, servers, and client/servers may be, for example, mainframes, minicomputers, workstations, or personal computers. Most services can be grouped into three major categories:

- distributed file system,
- distributed computing,

- messaging.

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

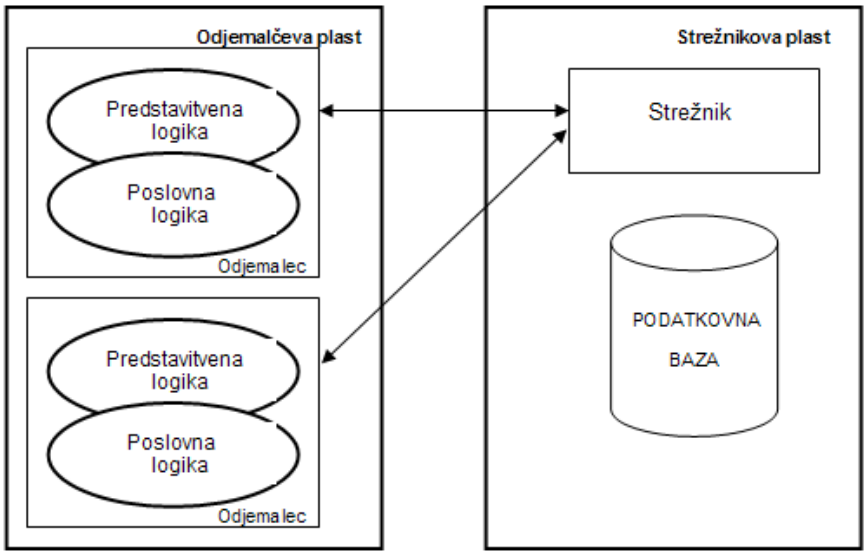
dostopajo preko omrežnih povezav. S tem program razpade na dve plasti – plast odjemalca in plast strežnika.

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.

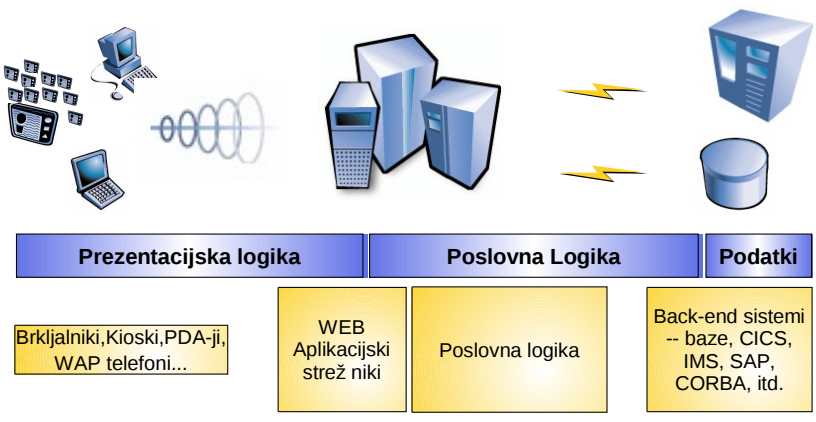
V dobi nove ekonomije bo potrebna popolna preobrazba programske opreme, kajti programska oprema bo postala storitev in ne več izdelek. Poleg tega so zaprti cikli, v katerih nastajajo nove različice programov, prepočasni za svet nove ekonomije.

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.

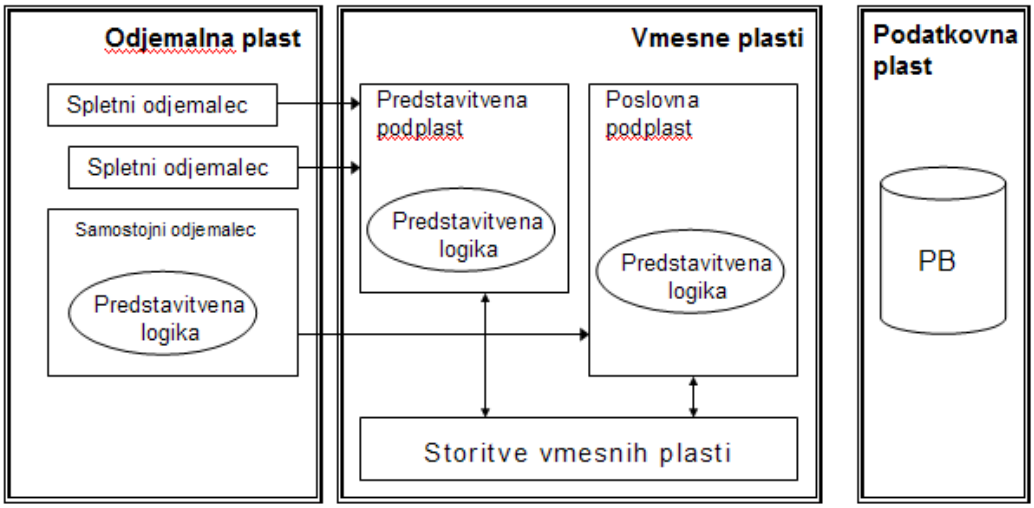
Troslojna / večslojna arhitektura

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.

Servisi na Internetu

The Internet Network

The Internet is simply a large, international network of computers and smaller computer networks. Despite its definition, the Internet does differ from a Local Area Network (LAN), which is a system consisting of one server and many clients, which access stored information on the server.

On the Internet, there is no central server. Instead, there are thousands of host machines which can both send and receive information to and from other hosts. In this way, each host on the Internet behaves as both a server and a client.

The actual physical network of the Internet consists of a massive web of transmission lines, each line capable of transfering a specified amount of data (this limitation is called bandwidth). This "web" of transmission lines is designed so that information from any site can reach its destination using thousands of different possible paths (see History of the Internet). When information is sent over this network, it is split into tiny packets.

Each packet of information travels to its destination using a different route. When all of the packets reach their destination, the packets are regrouped to form the original piece of information. If one or more of the packets doesn't successfully make its destination, the receiving site simply asks for another copy of those packets.

Internet Addresses

Two types of addresses are used on the Internet. There are **IP** (Internet protocol) addresses and **domain names**. IP addresses are numerical addresses used by computers on the Internet to route information (such as files or e-mail) through the various computers on the Internet.

Domain names are "common" names assigned to computers on the Internet. Domain names are used because numerical IP addresses (like 384.026.395) can be quite difficult for people to remember, and because of the fact that computers on the Internet often change their IP addresses, while their domain names usually remain the same.

Domain Names

As mentioned in the previous section, domain names are "common" names assigned to computers on the Internet. Each domain name has an IP address assigned to it, which is used by computers on the Internet. The domain names are used by the humans using the Internet because they are easier to remember than numerical IP addresses. They are also used because they are more permanent than IP addresses. A domain name consists of several parts: the top-level domain (TLD), the sub-domain, and the host name. The TLD is the broadest part of a domain name and comes last in the domain name. The sub-domain comes directly before the TLD. Preceeding the sub-domain can be any number of host names. In order to better demonstrate each part's function in the domain name, we will use the following example:

```
financing.hoover-bldg.acmeinc.com
```

In this case, the top-level domain (TLD) is com, which indicates the domain name belongs to a commercial business. The sub-domain is acmeinc, which indicates that the commercial organization that owns the domain name is most likely named Acme Incorporated. The first host name is hoover-bldg, which indicates that the specific computer being accessed is located in the Hoover Building of Acme Incorporated. The second host name, financing, indicates that the computer being accessed is in the financing department. Thus, we know that the computer on the Internet with the domain name, financing.hoover-bldg.acmeinc.com, is located in the financing department of the Hoover Building of Acme Incorporated, which is a commercial business.

Top-Level Domains (TLD's)

There are many top-level domains that you can encounter on the Internet. Some are national; that is, they may only be used by computers in a given poltical entity. Others are international, which may be used by any computer on the Internet.

Regional top-level domains are used only by computers in a given nation or political entity. They are two letters in length and are defined by the document, ISO 3166. Although the United States does have its own regional top-level domain (us), its structure is somewhat confusing and is not used very frequently. Instead, most U.S. computers on the Internet use international top-level domains (discussed below).

International top-level domains, or generic top-level domains, may be used by any computer on the Internet, regardless of its physical location. The following table shows the generic top-level domains currently avaiable, as well as their intended use:

Generic Top-Level Domains	Intended Usage

.gov	government
.edu	education
.com	commercial
.mil	military
.org	non-profit organizations
.net	networking providers
.int	international treat organizations and Internet databases

The **domain name system** (DNS) is used by computers on the Internet to translate domain names into their respective IP addresses. This system is used because computers on the Internet can only use IP addresses to route information, and since each domain name's IP address can change quite frequently, a central clearinghouse for the storage of every domain name's IP address is necessary. Every DNS host throughout the world has the same information. Whenever you tell your Internet software to connect to a certian domain name, it always checks with one of the DNS computers to get the correct IP address. Only then can it actually connect to the remote Internet site. If you had to install your own TCP software, you might have noticed an entry blank for your DNS server -- where you had to enter an IP address (instead of a domain name).

Electronic Mail

Advantages

You might ask, "What's the point in using Electronic-Mail (E-Mail) when I can just as easily use my phone, my fax machine, or the U.S. Postal Service?" Well, in an attempt to answer this question, some of the advantages of using E-Mail are listed below:

- **Speed** - The physical process of transferring E-Mail is virtually instantaneous.
- **Free** - The is no per-message charge when it comes to E-Mail.

- **Portability** - If you have a computer handy, you can dial your Provider and retrieve you E-Mail from anywhere in the world.

- **File Transfer** - E-Mail can be used to transfer binary files, including executable programs, graphics, sound, and data files.

- **Permanent Address -** If you're always "on the go," E-Mail can give you a permanent address. Using E-Mail, friends can always keep in touch, even if they don't have your most recent phone number.

While E-Mail may not replace the intimacy of a phone call (not yet), it can provide an easy way for friends and relatives to communicate without having to worry about outrageous long-distance charges.

E-Mail Addresses

In the same way that everyone with a mailbox has a postal address, everyone with an E-Mail account has an E-Mail address. These addresses may look unusual, but the rules governing the creation of E-Mail addresses are actually quite simple.

An E-Mail address consists of two parts: **the user ID** and the **domain**. A person's user ID is what they use to login to their Internet Provider.

While some user ID's resemble the person's actual name (Rob Jones = robjones, Steven McAffee = smcaffee), others are not as obvious (Marsha Thompson = mt295b). The domain is the name of the person's Internet Provider. When the user ID and domain are arranged in the form of an E-Mail address, they look like this:

mclark@finearts.caltech.edu

In this example, the user ID is mclark, which we can say stands for Matthew Clark. Everything after the @ symbol is the domain. This symbol is always pronounced as, "at." The periods separating the parts of the domain are pronounced as, "dot." Using these guidelines, the example would be pronounced as, "M Clark at Fine Arts dot CalTech dot E D U."

Finding Someone's E-Mail Address

Obviously, you have to know someone's E-Mail address before you can send them E-Mail. Fortunately, there are many ways of obtaining E-Mail addresses:

- Just Ask! - This is the most obvious and easiest way to get someone's E-Mail address. Just ask them to give it to you either by phone, in person, or by mail.
- Check an E-Mail Directory - Many companies, educational institutions, and organizations have their E-Mail directories available on the Internet.

- Look at the Return Address - If someone has sent you E-Mail in the past, just look at the "From:" line in the header. The address given there is most likely that person's E-Mail address.

Use an Online E-Mail Search Service - There are several Internet sites on the World Wide Web that serve as clearing houses for E-Mail addresses.

Mailing Lists

On the Internet, a mailing list is a discussion group that promulgates its member's messages via e-mail. This is accomplished by using a listserver, which is a computer that automatically receives messages from members of the discussion group, and sends all received messages to every member of the group at a specified time interval (hourly, daily, weekly, etc.). These messages are usually compiled and sent to each user as one large e-mail message, but they can also be sent individually. Most of the specific details can be set by the mailing list administrator, and often by each individual member of the list.

The topic of discussion on a mailing list varies from list to list. There are mailing lists that discuss everything from gardening to television shows to computer programming. Regardless of the topic, it is nearly always advisable to keep your posts on the assigned topic of that group. Most mailing lists are small in size when compared to Usenet newsgroups. Mailing lists are also usually more controlled than newsgroups. Mailing lists do have their disadvantages, however. Because all messages are sent as e-mail, one must read through every message, regardless of the subject.

Finding Mailing Lists

With thousands of mailing lists to choose from, it can be quite difficult to find the one you're looking for. If you know of any web pages devoted to the topic you are looking for, check those pages to see if a mailing list is mentioned. You can also ask other people on the Internet who are interested in the desired topic if they know of any mailing lists. The Massachusetts Institute of Technology FTP server, rtfm.mit.edu, has several lists of known mailing lists available to the public, which can be quite helpful when trying to find just the right mailing list.

Smiley's and Emoticons

Writing a personal letter using nothing but a standard alpha-numeric keyboard is an extremely difficult task. This is because people usually communicate using more than just words. We convey emotions, facial expressions, and other subtle body movements that help add "flavor" to our sentences. Believe it or not, but this can all be accomplished in the world of e-mail, as well.

To aid the art of e-mail communications, early Internet users developed a system of characters using different combinations of standard ASCII characters that represent human emotions.

These characters are called smiley's, or emoticons (pronounced ee-MOTE-eh-cons). A smiley is read by leaning your head down toward your left shoulder and looking at the characters sideways, reading from left to right. For example, a simple happy face is:

:-)

Although you can use smiley's anywhere in your message, try not to over use these characters -- too many can make your message hard to read.

Here are some common smiley's:

Emotions and Comments

:-)	Happy/Funny
;-)	Winking/Sarcastic
:-\	Undecided
:-(	Sad
:-o	Oops!

Remember, there is no "official" set of smiley's, so be creative! If you need help, check out the Unofficial Smiley Dictionary.

In addition to smiley's and emoticons, you can also add emotion to your actual e-mail text! For example, you can SHOUT on-line, or add_emphasis_ to a word, if you like. Shouting is accomplished simply by typing the part to be shouted in all caps. Because typing in all caps is considered shouting in the Internet world, you need to be sure that you don't leave your caps lock key on while typing e-mail messages, because everyone will think that you're shouting at them. It's just not polite.

As previously mentioned, emphasis can also be placed on a word or group of words. This is done by placing an underscore (the _ character - typed by "shifting" your dash key on most keyboards) before and after the word or groups of words to be emphasized. For example, "I' m _only_ 43 years old." In this example, the word, "only," would be emphasized. If you were to say the example aloud, you would raise the pitch of your voice slightly on the word, "only." Although the "underscore-emphasis" technique isn't used as often as the "all-caps-shouting" method, both can be found quite frequently on the Internet.

Internet Abbreviations

Some abbreviations commonly used in e-mail and other messages on the Internet are as follows:

BTW -	By the way
c-ya -	See you later
FAQ -	Frequently Asked Questions
FWIW -	For what it's worth

IMHO -	In my humble opinion
LOL -	Laughing out loud
OTOH -	On the other hand
RTFM -	Read the @%&!^ manual! (Read the FAQ)
TIA -	Thanks in advance

Telnet

What is Telnet?

Telnet is an Internet application that allows you to connect to remote Internet sites. Telnet acts much in the same way as your modem communications software, only it communicates with other computers over the Internet, instead of your standard phone lines.

Telnet Software

To use Telnet, you first need **a Telnet client application**. There are many good Telnet applications available for downloading on the Internet, many of them free.

HINT: Many Web browsers will automatically start your Telnet software when they encounter the telnet:// prefix protocol. To use this feature, you need to tell your browser where it can find your Telnet application. If using Netscape, this information can be entered from the General Preferences menu.

Using Telnet

To connect to a remote Internet site, you first need to know either its domain name or IP address, as well as the port you wish to connect to.

For the following examples, we'll pretend that we're trying to connect to port 600 of the Internet site flower.example.org. Some Telnet applications display a dialogue box with entry spaces for the Internet site (where you enter the domain name) and port number. Others simply display one entry box, where you usually enter the domain name (or IP address) and the port number in one of the following formats:

flower.example.org:600 (most common format)

or

flower.example.org 600

In most cases, once your computer has connected with the remote Internet site, you will be asked to enter a login ID and a password. If you are a user on the Internet site, simply enter your user ID where it asks for a login: and your password where it asks for a password (be careful: your password will not actually be displayed when you type it, so make sure you type

it in correctly). If you entered some information incorrectly, you will receive the message login incorrect. Just try again. If you still receive the message, you are probably not entering the right information. Most Internet sites will disconnect you if you can't supply a valid login and password after three tries.

If you are not a user on the Internet site, you might be able to login under the user ID guest or newuser (if this is the case, the password is usually the same as the user ID - or the Internet site might even tell you what password to use). Once you have successfully logged in, you will probably be given either a menu or a Unix prompt. At this point, you can browser through the contents of the remote Internet site or, depending on your user level (or security level), manipulate the files and/or directories on that site.

File Transfer Protocol (FTP)

What is FTP?

File Transfer Protocol (FTP) is a protocol used to transfer ASCII or BINARY files between two computers on the Internet. The primary function of FTP on the Internet is in the use of anonymous FTP sites. An anonymous FTP site is simply an Internet site accessed via FTP that has a large number of files available for downloading. They are called **anonymous** because you login anonymously. Simply enter the word anonymous at the "Name:" prompt and your full e-mail address at the "Password:" prompt.

Some anonymous FTP sites are small, consisting of only a dozen or so files related to a specific topic. Others are huge shareware archives with thousands of files. It should be noted that the increasing influence of the World Wide Web has resulted in most file archives being available over the Web, as well as FTP.

FTP Clients

To directly access an FTP site on the Internet, you will need some kind of FTP client. While most Web browsers have built-in FTP clients, stand-alone FTP applications also exist. Having such an application is an advantage if you don't like waiting for your web browser to load when all you want to do is access an FTP site.

Archie File Search

Because there are so many anonymous FTP sites on the Internet, it would be impossible for you to search through each one individually in hopes of finding the file you want. To assist you in your file search, a service called Archie is available. Archie is a public search engine whose database currently maintains a list of about 900 Internet anonymous FTP sites of approximately 2.1 million files containing 170 Gigabytes (170,000,000,000 bytes) of information.

There are many Archie servers throughout the world that can be accessed either by e-mail or telnet. None of these servers actually store all of the files available via anonymous FTP. Instead, they simply keep track of the directories and filenames located at each anonymous FTP site.

Types of Searches

Name Searches

To perform a name search, simply give Archie a search string (there are four search formats, discussed below). Archie will then list all FTP sites that have the word(s) you're searching for in their directories or filenames. For example, if you searched for the word "equestrian," you might receive the following result:

```
Host plaza.aarnet.edu.au
Location: /usenet/FAQs
DIRECTORY drwxr-xr-x       512  15-May-1996 16:53:25  rec.equestrian

Host sun.rediris.es
   Location: /docs/faq/rec
     DIRECTORY drwxr-xr-x       512  05-Mar-1996 15:07:17  equestrian

Host ucdavis.ucdavis.edu

   Location: /pub/U.C.Davis.directory/Campus.Departments.and.Offices/E
        FILE -rw-r--r--       122  29-Oct-1995 10:44:02
EQUESTRIAN.CENTER.....................................752-2372
```

We now know that what appears to be a UseNet FAQ for the newsgroup rec.equestrian can be downloaded via anonymous FTP from the FTP site plaza.aarnet.edu.au, among others. The third result (from ucdavis.ucdavis.edu) appears to be part of an e-mail directory.

As previously mentioned, there are four ways of entering a search string for an Archie name search. A brief description of each is as follows:

search substring - Simply enter one or more words to be found anywhere in the Archie database (either as a filename or directory)

search substring (case sensitive) - Same as above, except this search will only produce results that match the same case

(capitalization) as the entered search string.

exact - Use this option if you already know the exact filename of the program you're looking for. Archie will search only through

filenames for an exact match. This is the fastest search, but it is only useful if you already know exactly what you're looking for.

regex - This stands for Regular Expressions, which is a complex system within Archie that allows you to use certain characters and

syntax to "mold" your search to yield specific results. A complete guide to using Archie Regular Expressions is available.

File Description Searches

Archie maintains a Public Domain Software Description Database, which contains descriptive information about the actual content of some files

(mostly text files) that can be downloaded via anonymous FTP. This type of search is most useful if you're trying to find information about

RFC's, Unix utilities, or technical Internet documents.

Site Searches

Two commands fall under ths category. The list command results in a list of all anonymous FTP sites currently in Archie's database. The site command produces a list of all files known to Archie for a specific FTP site.

How to Access Archie

There are many Archie servers located throughout the world. Each can be accessed several different ways. You can perform an interactive Archie search by logging into an Archie site via telnet or by using an Archie client. A remote Archie search can be done by using e-mail. Each of these methods is discussed below:

Archie Client

This is perhaps the easiest way of using Archie for anyone with a SLIP/PPP Account. An Archie client is a piece of software that automatically telnets to your favorite Archie server, enters the proper commands, and neatly displays the results on your screen. Most programs allow you to automatically download any file in the Archie result screen using your FTP Client.

Telnet to Archie

Accessing Archie via telnet is probably one of the most difficult ways of getting things done, but a brief explanation of how to go about it has been included just in case. The first step is to telnet to one of the Archie servers. A good rule of thumb is to pick the one that is geographically closest to your location. Once you are connected to the server, simply enter the word archie at the login prompt (login:). No password is necessary. You will then receive a prompt that looks like this: archie:.

The appearance of this prompt marks the beginning of your interactive Archie session. At this point, you can enter any of the commands accepted by Archie. Although a complete Archie operation manual is available, a brief summary of the essential commands is listed below:

find <pattern> - Where <pattern> is the word(s) you want to search for within the Archie database. Will result in a list showing the filename, directory, and FTP site where each match can be found. The find command can be used with a searchable substring (optional case sensitive), exact pattern, or regular expressions (see above).

help <topic> - Initiates an Archie help session.

manpage - Produces the entire Archie operating manual. (very long!)

Accessing Archie by E-Mail

Because Archie is used quite frequently by the Internet community, you will often find that using Archie interactively is rather sluggish. This is because you are always in line behind others trying to use the same Archie system. So, let's say you just want a quicker way to search Archie's database. Or, perhaps you only have access to e-mail and can't use Archie using any of the other methods. There's a simple solution to both of these problems -- access Archie by e-mail!

All you have to do is simply send an e-mail message to the Archie server of your choice, and the next time you check your e-mail (the process usually takes a few minutes, although it can sometimes take as much as a day), you will have an e-mail message from Archie with all the results from your search. To use Archie via e-mail, follow these steps:

Address a new e-mail message to archie@<Archie server> (where <Archie server> is the address of any of the Archie servers - i.e. archie.au).

Leave the subject line blank.

Use any Archie commands in the body of the message. Hint: To simply search the Archie database for one or more words, use “find <pattern>”, where <pattern> is the word(s) that you want to search for.

Send the e-mail!

You should receive a reply from the Archie server within a few minutes, although it can sometimes take as much as a day.

This process is done using a listserver and is completely automatic. In other words, no humans are involved in this process, so any problems you encounter are due to problems with the Archie server. A complete list of Archie commands that can be used is available.

Search For Anonymous FTP Sites on the Web

With the increasing influence of the World Wide Web on the Internet, many anonymous FTP sites now have their entire file archies available on the Web. Most of these also have easy-to-use search engines that search file names and (sometimes) file descriptions for an entered search pattern. A useful service called Shareware.com (http://www.shareware.com) can be used to search several of the existing on-line FTP sites at once.

Internet Chat

Chat rooms have always been popular on the Internet, but not until recently have they become quite so popular with the general Internet public, previously frequented most often by "hard core" Internet users with a little too much time on their hands. Now, it seems everyone wants to take part in the online chatting experience. There are four major types of online chat that we will discuss: IRC Chat, one of the most popular forms of online chat, Web-based chat, chat rooms exclusive to commercial providers, such as AOL or Prodigy, and finally MUD chat rooms, which are somewhat less popular now than they were a couple years ago.

Internet Relay Chat

Internet Relay Chat, or IRC, was developed in 1988. IRC is now used by thousands of people every day as a virtual meeting place in which to discuss various issues of the day. Users enter specific channels in which a specific subject or theme is usually predominant. Channels can be either public, where anybody on the Internet can join, or private, in which two people can carry on a private conversation. For more extensive information on the usage of IRC chat, feel free to refer to the Internet Relay Chat Help Site at irchelp.org.

In order to use IRC, you must have an IRC client (software). Popular IRC clients are mIRC for Windows and IRCle for Mac users. Once you have downloaded and installed this software, you will need to be able to provide the following information: your IRC server, your e-mail address, your name, and your nick name, or nick.

IRC Servers

In order to use IRC, you will have to tell your IRC client which IRC server to connect to. There are plenty of servers across the world, all connected to form one large IRC network. It

is usually a good idea to choose the server geographically closest to you. This should provide for a good and fast connection, with minimal lag time (delays). IRC servers are individually operated by IRC operators (or ops, for short). In some cases, you may be denied access to an IRC server based on the decision of an IRC operator to deny access either to you personally (usually unlikely), or perhaps your domain (in which case you can really do nothing about it). If this happens, simply try another IRC server.

Nicknames

While using IRC, you will be referred to not by your real name or by your e-mail address. Instead, you will be known throughout the IRC network by a one-word IRC nickname, such as "coolguy" or "smartgal," for example. Although you can essentially choose any nickname that you want, it is usually not a good idea to pick a nickname which is likely to be already used, such as "john" or "mike," which are both very common names. Some IRC clients will ask for you to provide two nicknames, one being a backup in the case that your first choice is already being used.

IRC Channels

In order to provide some form of organization within the IRC network, many channels exist, in which a particular subject or theme is usually predominant. Channels are operated by channel operators, or channel ops for short. Channel ops have the final word on all matters within thechannel: who can enter the channel, who can talk, and so on. A channel can either be private, in which only specified users can join and the conversation is kept private from the rest of the IRC community, or public, in which anybody can join the channel (unless specifically banned by the channel op) and take part in the discussion. Names are given to channels that usually represent their topic of discussion or purpose. Within the IRC network, all channel names begin with the pound symbol (#). Examples of some channel names are #newbies and #hamradio.

Web-based Chat

There are many Internet chat rooms that exist right on the World Wide Web, run by various organizations and web sites. Web-based chat rooms sometimes offer real-time updates of chat activity, while others require that you reload the screen periodically to keep up with the discussion. Posting is usually entered through standard submission forms within your Web browser. For links to some of the Web-based chat rooms available on the Internet, check out a fairly comprehensive list maintained by Yahoo!.

Provider-Exclusive Chat

Some commercial on-line providers, such as American Online, give users access to various chat rooms exclusive to that on-line service. In other words, these rooms are not available to everyone on the Internet. Rather, they can only be accessed by members of that specific on-line service.

For more information on any such chat opportunities, please contact your own commercial on-line provider for details.

MUD Chat

Although being somewhat replaced by IRC, Multi-user Dungeons, or MUD's, still constitute a major portion of on-line chat usage. MUD's are actually quite similar to IRC, in that real-time chatting capabilites are provided, but MUD's are usually accessed by telnetting to a remote Internet site and logging in with a special nickname, which sometimes is assigned a permanent password. MUD's are most frequently used these days as homes to real-time games, which are usually somewhat like role-playing games. For some of the MUD's available on the Internet, feel free to visit a fairly comprehensive list maintained by Yahoo!.

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:

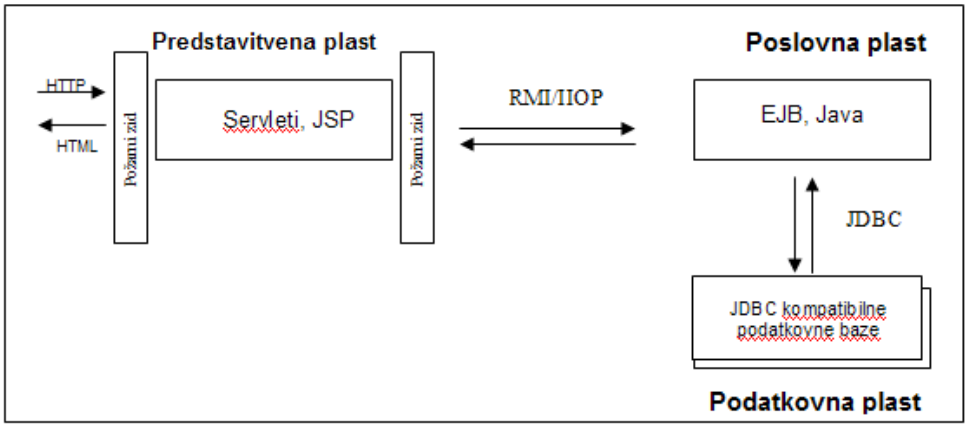
- je programski jezik, ki je enostaven (posebej za nekoga , ki pozna C/C++), predmetno usmerjen, varen, robusten, porazdeljen, večopravilen, interpretiran, arhitekturno neodvisen in prenosljiv
- poleg jezika je pomen Jave tudi v tem, da nudi vsa potrebna razvojna orodja (prevajalnik, tolmač prevedene 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:

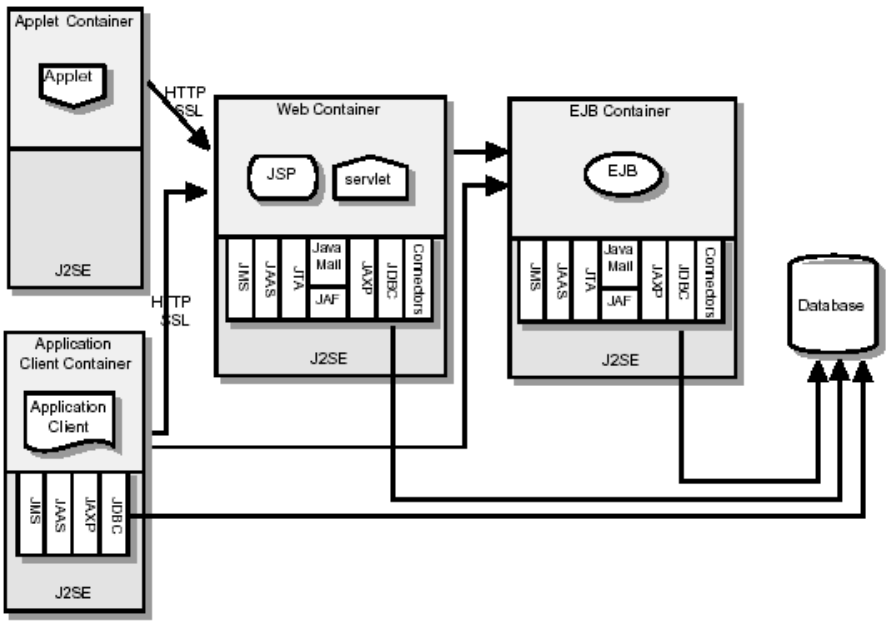
Naloge posameznih plasti so razdeljene na sledeči način:

- 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.

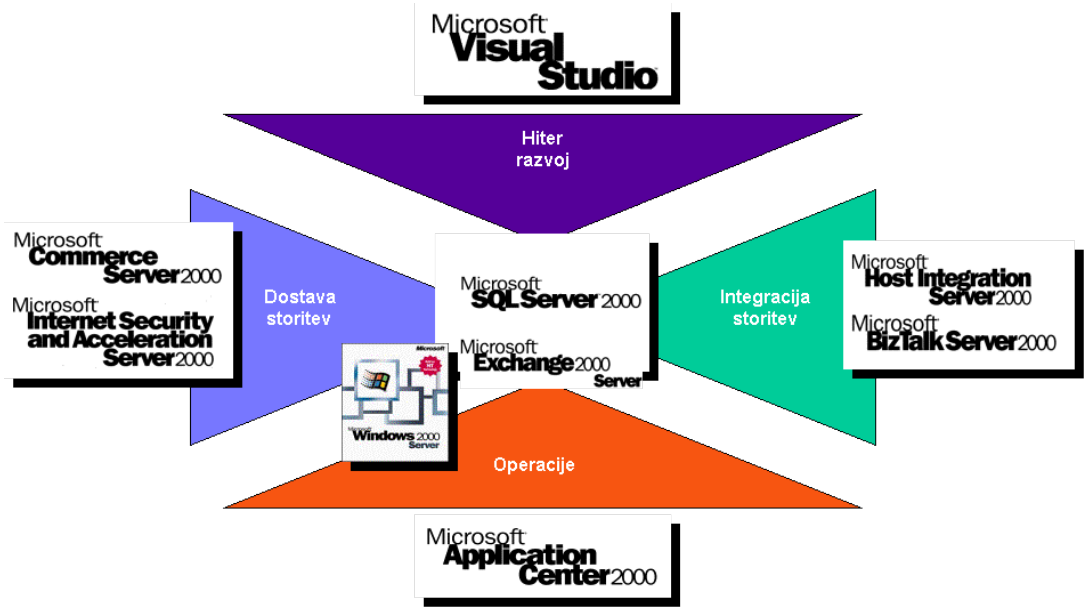
- **Exchange Server 2000**: omogoča sporočanje kadarkoli in kjerkoli in upravljanje s sporočili. Nudi storitve za upravljanje stikov in nalog, za vodenje debatnih skupin, itd.

- **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)



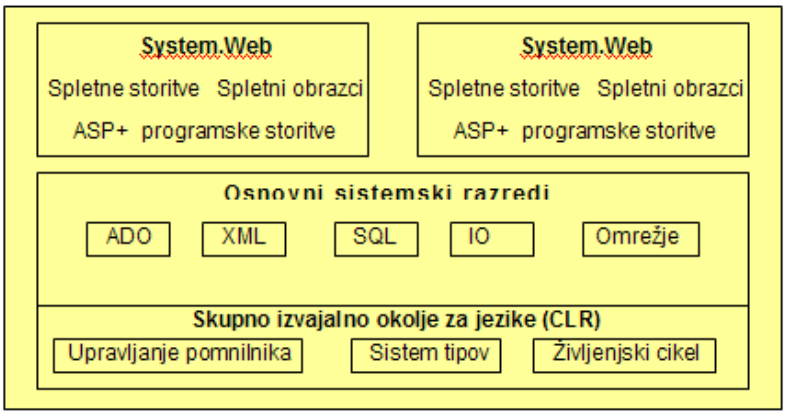
Slika: Vloga .NET strežnikov

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.

Primerjava J2EE in .NET

Sunova platforma J2EE in Microsoftovo ogrodje .NET sta namenjena isti ciljni publiki – razvijalcem sodobnih poslovnih aplikacij in lahko rečemo, da sta si iz arhitekturnega in konceptualnega vidika precej podobni. Pokrivata tehnologije za sloj uporabniškega vmesnika, za sloj spletnih storitev, poslovne logike, vmesnega sloja in za stanje podatkov.

	J2EE	.NET
Tip tehnologije	Standard	Izdelek
Število ponudnikov	Več kot 30	Microsoft
Interpreter	JRE	CLR
Podprti programski jeziki	Java	C#, C++, Visual Basic .NET, Cobol*, Perl*…
Debeli odjemalci	Swing, JFC	WinForms
Lahki odjemalci	JSP, Servleti	ASP.NET, WebForms
Komponentni model	EJB	.NET Component Service
Protokol za komponente	SOAP, IIOP	SOAP
Sporočilni sistem	JMS	MSMQ
Imeniški sistem	JNDI	ADSI
Dostop do podatkovnih baz	JDBC	ADO.NET
Dostop do obstoječih sistemov	CORBA Connectors	HIS (dodatek .NET)

Postavlja se vprašanje, za katero rešitev se odločiti? Za merilo lahko vzamemo različne faktorje in vidike posamezne rešitve:

Podpora programskim jezikom

Izbira platforme glede podpore programskim jezikom je najlažja – J2EE podpira le Javo in tudi v bližnji prihodnosti ni predvideno, da bi neposredno podpirala kakšen drugi jezik, medtem pa platforma .NET podpira množico jezikov, vendar ne Jave. Napovedana je tudi podpora za npr. Cobol, vendar le v različici, prilagojeni za .NET. To pomeni določene spremembe notacije in pravila .NET, ki se jih bo potrebno držati. Ravno direktna uporaba obstoječe kode torej ne bo možna, je pa seveda tak prenos kode dosti lažji, kakor če bi morali popolnoma zamenjati jezik.

Nekatera podjetja (IBM, BEA) so v svoje pakete, ki implementirajo J2EE vgradile tudi podporo za druge jezike, vendar pa ne pod tehnologijo J2EE. Dostop do funkcij drugih jezikov je v J2EE možen z uporabo JNI (Java Native Interface) ali CORBA modela.

Vmesna koda

Platforma Java že od samega začetka izvorne kode ne prevaja direktno v strojno kodo, pač pa v tako imenovano zlogovno ali vmesno kodo. To kodo nato izvaja Javin navidezni stroj (JVM). Navidezni stroj stoji za vsakim bajtnim ukazom in lahko programom po potrebi prepreči dostop do nekaterih sistemskih virov, npr. datotečnega sistema. V prvih različicah je JVM kodo interpretiral, posledica pa so bile slabe zmogljivosti. Sedaj pa JVM-ji kodo pred izvajanjem prevedejo. Govorimo o Just-in-time prevajanju, kar izvajanje vmesne kode močno pospeši in ga postavlja ob bok drugim prevedenim jezikom.

Podobno je tudi pri ogrodju .NET, kjer je osnova skupno izvajalno okolje za programske jezike (CLR). To okolje vpeljuje vmesni jezik (IL – Intermediate Language), v katerega se prevede izvorna koda. Tako lahko uporabimo praktično katerikoli jezik, vendar so vsi jeziki podvrženi manjšim prilagoditvam in omejitvam, kar prednost podpore več jezikov nekoliko omeji (npr. CLR ne podpira večkratnega dedovanja, zato tudi C++ preveden v IL tega ne podpira). Tako bo bolje, da za stvari, ki jih bomo razvijali na novo uporabimo kak nov, sodoben jezik (C#).

Oba koncepta, JVM in CLR se dobro obneseta pri zagotavljanju varnosti. Zaradi koncepta t.i. peskovnika (torej nadzorovanega izvajanja programov), nudita veliko stabilnejšo in predvsem varnejšo platformo od direktnega prevajanja v strojno kodo. S tem je Microsoft tiho priznal uspešnost koncepta nadzorovanega izvajanja kode in ga še dopolnil s svojimi izvirnimi rešitvami (predvsem podporo za različne programske jezike).

Prenosljivost

Pod prenosljivost razumemo možnost prenosa aplikacije na drugo platformo. V praksi to običajno pomeni možnost prenosa na drug operacijski sistem na eni strani in možnost prenosa

na aplikacijski strežnik drugega proizvajalca na drugi strani. Slednje označujemo tudi kot neodvisnost od proizvajalca.

Platforma J2EE je bila že od vsega začetka zamišljena popolnoma neodvisno in v tem je tudi ena njenih največjih prednosti pred ogrodjem .NET. Le to je tradicionalno vezano samo na Microsoftov operacijski sistem in o kakšni prenosljivosti med proizvajalci in platformami tu ne moremo govoriti.

Neodvisnost J2EE je bila reklamirana kot »napiši enkrat, poganjaj vsepovsod«, vendar je praksa pokazala, da je popolno prenosljivost težko doseči, zato je pametno, da razvijalci sistem kljub vsemu stestirajo v vseh okoljih, v katerih ga želijo izvajati.

Tudi neodvisnost od proizvajalca je v J2EE v veliki meri dosežena. Vedeti pa moramo, da imajo aplikacijski strežniki različnih proizvajalcev (več kot 30) vgrajene razne dodatne funkcije, katerih uporaba oteži prenosljivost. Vseeno pa je veliko lažje prenesti aplikacijo skladno z J2EE iz aplikacijskega strežnika enega proizvajalca v drugega, kakor pa prenesti aplikacijo iz .NET v J2EE ali obratno.

Prenosljivost aplikacij in neodvisnost od proizvajalca sta pomembni za podjetja, ki so neodvisni proizvajalci programske opreme (ISV) in morajo produkt, ki so ga napisali, prodati večim strankam, z različnimi operacijskimi sistemi. V drugi vrsti pa so to podjetja, ki morajo zaradi rasti večkrat menjati platformo strojne opreme.

Povezljivost

Povezljivost nam pove, na kakšne način je mogoče naše aplikacije povezati s sistemi, ki ne delujejo v izbranem ogrodju. Pri povezljivosti nudita obe ogrodji precej. J2EE podpira arhitekturo CORBA, ki velja za enega od najpomembnejših načinov za dostop do obstoječih sistemov in integracijo aplikacii.Poleg tega ponuja J2EE tudi t.i. Connectors, ali konektorje, za enostavno integracijo komercialnih sistemov. Z podporo XML in pripadajočih standardov pa se nam olajša elektronsko poslovanje in IAI (Internet Application Integration).

Ogrodje .NET pa je v tej podpori še močnejše. Standardi SOAP, UDDI in WDSL so podprti v samem jedru ogrodja. Slabši, oziroma težavnejši pa je dostop do sistemom na platformah, ki niso Microsoftove (lahko pa dokupimo Host Integration Server)

Odjemalna stran

Za izdelavo tako imenovanih lahkih odjemalcev ponuja Java kombinacijo strežniških strani (JSP) in Servletov, ogrodje .NET pa ASP.NET in WebForms. Slednja omogočata izdelavo nivoja uporabniškega vmesnika, ki je neodvisen od fizičnega tipa odjemalca (Osebni računalnik, telefon, itd). Ogrodje je namreč odgovorno za izbiro najboljše možne predstavitve vsebine na določenem tipu odjemalca. Avtomatska izbira načina predstavitve skriva veliko pasti, verjetno pa bo Microsoft poskrbel za najboljšo možno predstavitev na lastnih odjemalcih (npr. Internet Explorerju), kar vodi k monopolizaciji tržišča.

J2EE podobne neodvisnosti od tipa odjemalca trenutno nima in edina možnost je uporaba kombinacije XML / XSL

Vmesni sloj

Pri storitvah za vmesni sloj je podobnost med modeloma velika. Za dostop do relacijskih podatkovnih baz podpira J2EE JDBC, ogrodje .NET pa ODBC. Oba načina sta precej nizko nivojska, podpirata pa številne napredne funkcije (npr. upravljanje povezav),

Microsoft že nekaj časa razvija druge načine dostopa do podatkov – z ogrodjem .NET je bil uveden ADO.NET, kjer je protokol modela COM zamenjan s SOAP.

Java prav tako nudi višje nivojski dostop do podatkov, z uporabo entitetnih zrn komponentnega modela EJB, ki omogočajo preslikavo objektnega modela v trajno stanje, glede na izbran model. Drugi način, podprt v Javi, so Java Data Objects (JDO), konceptualno podobni ADO, vendar precej naprednejši v funkcijah, saj omogočajo tudi integracijo z obstoječimi sistemi.

Način dostopa do storitev poimenovanja je pri J2EE izveden preko vmesnika JNDI (Java Naming and Directory Interface), ki je neodvisen od dejansko uporabljenega imeniškega sistema in definira enoten vmesnik za dostop do takih sistemov, ne glede na njihovo funkcijo (seznami datotek, seznami računalnikov, seznami elektronskih naslovov, itd.). Ogrodje .NET pa temelji na Microsoftovemu imeniškemu sistemu Active Directory in na vmesniku ADSI (Active Directory Services Interface), za dostop do imenika.

Pri podpori komunikacije nudi J2EE vmesnik JMS (Java Message Service), ki omogoča preko abstraktnega vmesnika dostop do poljubnega sporočilnega sistema. Funkcionalnost je podobna Microsoftovemu Message Queue, seveda pa je zasnova Jave fleksibilnejša in neodvisna od konkretnega produkta.

Pri dostopu do obstoječih sistemov nudi J2EE poleg podpore standardom kot so CORBA in IIOP, tudi Java Connectors, ki omogočajo integracijo z velikimi komercialnimi sistemi (tudi sistemi ERP). Microsoft ponuja tu sicer Host Integration Server, ki pa ni del ogrodja .NET.

Skalabilnost (performanse)

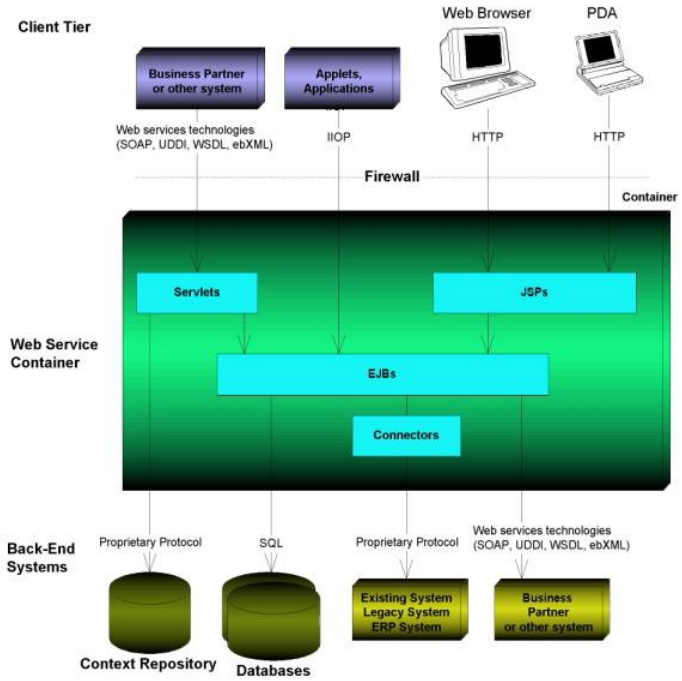
S skalabilnostjo je mišljena zmožnost sistema, da obvladuje dodatno breme (povečanje števila uporabnikov) z linearnim povečanjem porabe virov. V okviru problematike skalabilnosti sistema se običajno srečujemo s vprašanji obvladovanja porabe virov računalnika, razporejanja podatkov in funkcionalnosti po računalnikih.

Z uporabo standardiziranih testov, kot je Transaction Performance Council Benchmark, lahko izmerimo najvišje breme, ki ga naš sistem še prenese (merjeno v številu transakcij na minuto).

Podpora spletnim storitvam

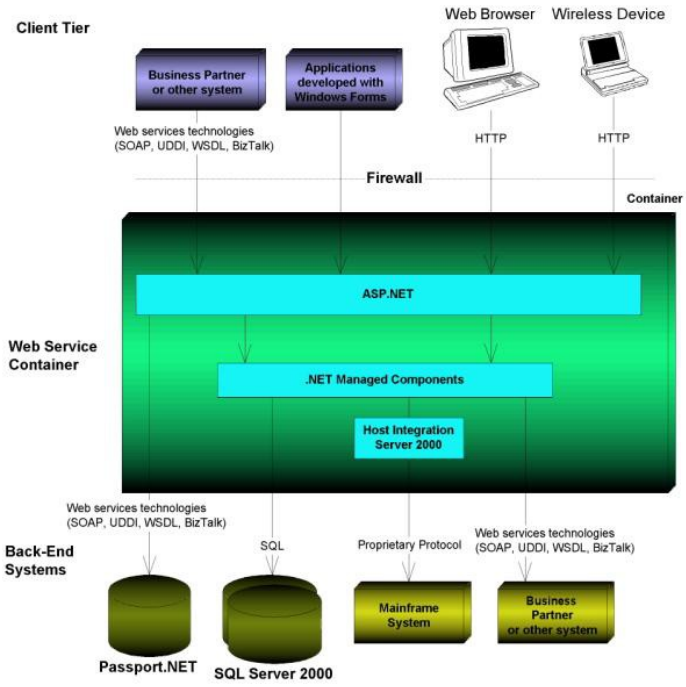
Podpora spletnim storitvam je eden izmed važnejših vidikov pri načrtovanju prihodnje infrastrukture za elektronsko poslovanje.

J2EE nudi podporo spletnim storitvam preko vmesnika JAXP (Java API for XML Parsing), kar sicer ni idealna rešitev, ker zahteva veliko ročnega dela. Zato so neodvisni proizvajalci ponudili celo vrsto J2EE kompatibilnih orodij za gradnjo in razvoj spletnih storitev: npr preko 20 implementacij protokola SOAP, več UDDI vmesnikov API (npr. IBM UDDI4J). Računa se, da bodo te knjižnice kmalu standardizirane in vključene preko JAX.



Slika: Razvoj spletnih storitev z Java2 Enterprise Edition

Prav tako tudi ogrodje .NET ponuja rešitve za gradnjo spletnih storitev in to s pomočjo čarovnikov v Visual Studio .NET. Slaba lastnost pa je nezadostna podpora za ebXML, ki je pomemben člen za polno uveljavitev in resnično odprtost spletnih storitev. Microsoft na tem področju uporablja BizTalk, ki pa uporablja svoj lastni SOAP, s čimer pa je spet pod vprašanjem resnična neodvisnost spletnih storitev.



Slika: Razvoj spletnih storitev z .NET

Orodja

Pri razvojnih orodjih imamo pri ogrodju .NET eno samo možnost – to je MS Visual Studio .NET. To pa ni nobena pomanjkljivost, ker je to zbirka zmogljivih in učinkovitih orodij. Visual Studio .NET podpira vse programske jezike podprte v prejšnjih verzijah Visual Studia (razen Jave), poleg tega pa podpira tudi nov objektno orientiran jezik C#.

Na področju razvojnih orodij lahko rečemo, da je Microsoft v prednosti, saj nam nudi celotno zbirko orodij z enim samim izdelkom, medtem ko moramo za doseganje iste funkcionalnosti na področju J2EE poseči po izdelkih več proizvajalcev (IBM-ovo orodje Visual Age for Java, Borland-ov JBuilder, itd)

Unikatna identifikacija

Unikatna identifikacija uporabnika bo ključni element spletnih storitev. Vizija je v tem, da bi odpadlo večkratno vpisovanje istih podatkov (uporabniških imen, gesel, številk kreditnih kartic, itd) ob prijavljanju na različne spletne strani. Z enostavnostjo prijave se poveča obisk spletne strani in omogoči lažja personalizacija vsebine.

Microsoft tu ponuja storitev Passport.NET. To je shramba podatkov potrebnih za identifikacijo, ki jo vodi Microsoft. Takoj se seveda pojavlja vprašanje zasebnosti podatkov, oziroma efekta »Veliki brat«, čeprav Microsoft trdi, da ne bodo izvajali nad podatki nobenega poizvedovanja ali kakršnega koli podatkovnega rudarjenja. Težko si je predstavljati, da bodo

podjetja kot so Visa, MasterCard in podobna, dopustila Microsoftu kontrolo nad informacijami o njihovih strankah.

Sunova J2EE vizija je decentralizirana, porazdeljena zbirka shranjevanja podatkov za unikatno identifikacijo. Vsak uporabnik lahko izbere svojega ponudnika za shranitev podatkov identifikacije. Shramba je lahko specializirana samo na eno skupino uporabnikov (npr. finance, medicina…), ki jo je ustvarila skupina več partnerjev in se s tem izognemo nezaupanju uporabnikov.

Prednost Microsoftove vizije je v tem, da je Passport že uporabna storitev, medtem ko mora Sun še svojo idejo standardizirat (APIji ki bodo spletnim storitvam omogočili uporabo unikatne identifikacije). Prednost je tudi še v tem, da pri Microsoftu ni vprašanja, kdo je uradni nosilec shrambe podatkov, medtem ko lahko pride pri J2EE pri temu do zmešnjave.

Stroški

Nedvomno pomemben vidik izbire med J2EE in .NET so stroški. Analiza stroškov je morda najbolj težavna naloga od vseh, saj je potrebno upoštevati veliko vidikov. Na eni strani govorimo o neposrednih stroških programske opreme, podpornih orodij in strojne opreme. Na drugi strani pa ne smemo zanemariti stroškov vzdrževanja, uporabe programske opreme in seveda stroškov šolanja.

Na prvi pogled je pri neposrednih stroških programske opreme v prednosti Microsoft, ki bo dobavljal celotno infrastrukturo skupaj z operacijskim sistemom in podpornimi orodji. Pri J2EE pa je potrebno posebej kupiti aplikacijski strežnik, katerih cene lahko gredo v vrtoglave višine. Vendar so to rešitve namenjene za res velike projekte za sisteme mainframe in za platforme Unix, kjer seveda ogrodje .NET odpove. Po drugi strani pa za rešitve na platformi Win32 tudi J2EE ponuja cenovno ugodne rešitve in pa tudi množico javno dostopnih, »open source« izdelkov. Cena programske opreme ne sme biti odločujoči faktor pri izbiri med J2EE in .NET, ker je vendarle to majhen delež celotnih stroškov vpeljave nove arhitekture (TCO).

Nedvomno visoki pa bodo stroški šolanja. Ti so primerljivo visoki pri obeh ogrodjih. Ne smemo pozabiti, da je ogrodje .NET rez s preteklostjo, kar pomeni, da obstoječi Windows razvijalci pri učenju ogrodja .NET ne bodo v prednosti. Nekatere analize celo kažejo, da se izkušen J2EE razvijalec prej nauči uporabe .NET, kakor izkušen Windows razvijalec.

Zaključek

Odločitev o izbiri med J2EE in .NET bo torej težka, vendar bo prehod prej kot slej neizbežen.

Java 2 Enterprise Edition je uveljavljena arhitektura, s preverjenimi in delujočimi aplikacijskimi rešitvami ter predvsem z veliko podporo različnih podjetij. Medtem pa je arhitektura .NET še v beta različici in je zanimiva za obstoječe Microsoftove razvijalce (predvsem Microsoftovce po »veroizpovedi«). Pri obeh platformah je podpora spletnim storitvam nova in zato še ne popolnoma zrela, prav tako pa je nepreverjena še tehnologija konektorjev JCA pri J2EE in jezik C# pri .NET

Nekaj argumentov, ki govorijo v prid .NET proti J2EE:

- .NET ima odlično marketinško podporo Microsofta
- .NET ponuja izbiro in s tem nevtralnost pri programskih jezikih
- .NET napoveduje odlično podporo razvijalcem s Visual Studio .NET
- .NET je tesno povezan z operacijskim sistemom
- .NET že ima delujočo rešitev za storitev unikatne identifikacije (shared context) – Passport
- .NETova zgodba o spletnih storitvah je prišla na trg prej in prevzela razmišljanje razvijalcev

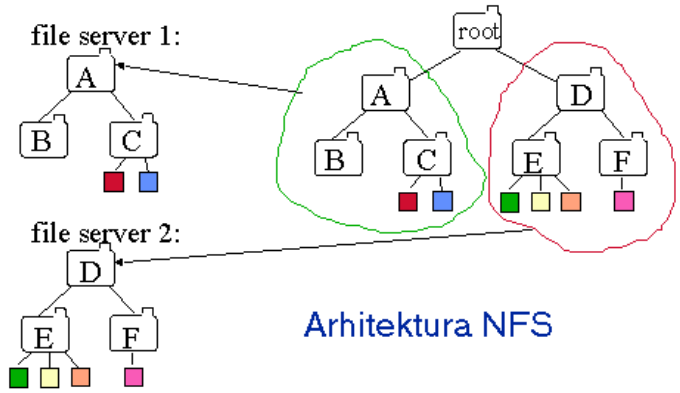
Nekaj argumentov, ki govorijo v prid J2EE proti .NET:

- J2EE je bila sprejeta v večini vodilnih podjetij (Sun, IBM, BEA, Borland, itd)
- spletne storitve lahko z J2EE razvijamo že danes
- J2EE je neodvisna od strojne podlage
- J2EE je neodvisna od operacijskega sistema (Windows, Unix, mainframe)
- veliko število Java razvijalcev (trenutno v svetu 2,5 milijona)
- večina neodvisnih razvijalcev programske opreme (ISV) bo verjetno prešla na J2EE, ker si s tem zagotavljajo široko izbiro strank

Za obe platformi pa velja, da bo potrebno vložiti ogromna sredstva v izobraževanje. Po ocenah analitikov bodo sredstva izobraževanja za uporabo .NET ogrodja primerljiva s tistimi za uporabo J2EE platforme. Kajti novi ali spremenjeni programski jeziki, novi aplikacijski vmesniki in nove knjižnice postavljajo Microsoftove razvijalce v pozicijo, ko jim obstoječe znanje ne koristi veliko.

Porazdeljeni datotečni sistemi (NFS)

Sistemu NFS pravijo včasih porazdeljen datotečni sistem, vendar je bolj pravilno, da mu rečemo mrežni datotečni sistem. NFS je običajno poseben dodatek k nabavljenemu operacijskemu sistemu. Omogoča programom na računalnikih - klijentih transparenten dostop (branje in pisanje) do datotek na računalnikih z ustreznim strežnikom NFS.



Arhitektura NFS

Organizacija NFS je simetrična tako, da je lahko dani računalnik lahko istočasno klijent in strežnik.

Pri obravnavi NFS moramo ločiti med pojmoma produkt NFS in protokol NFS.

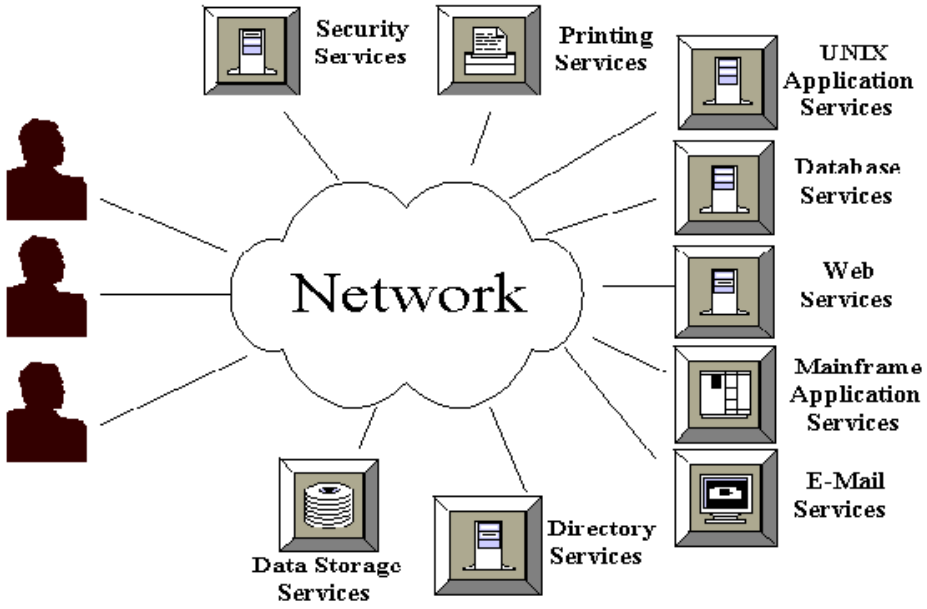
Sistem NFS naj bi sicer dopuščal ekvivalentne operacije, ki jih imamo na lokalnem datotečnem sistemu, vendar nekatere omejitve le veljajo. Med bolj znanimi je brisanje odprte datoteke. Pri lokalnih datotečnih sistemih bi taka datoteka še fizično obstajala, dokler jo nek proces drži odprto. Tega pa strežnik NFS ne more vedeti. Delno se taki problemi rešujejo z več inteligence na strani klijentov. Pa še pri tem veljajo omejitve (na primer da sta proces, ki datoteko odpre, in proces, ki datoteko briše, na istem računalniku - klijentu). Prej navedeni problem se na strain klijenta na primer rešuje z začasnim preimenovanjem datoteke (damo ji neko "sintetično" ime, na primer .xx12345) . Datoteke pod iskanim imenom ne poznamo več, vendar fizično še obstaja. Tako zaznamovano (in nevidno) datoteko nato zbrišemo pri njenem zaprtju, lahko pa jih briše tudi neka procedura ob ponovnem zagonu računalnika.

Podobne probleme imamo tudi pri spreminjanju lastništva in zaščit datotek. Probleme povzroča tudi pomnenje atributov direktorijev, ki so tako kot navadne datoteke pomnjeni v medpomnilniku klijenta. Problemi so lahko tudi zaradi slabe časovne sinhronizacije računalnikov (eden od atributov datoteke je tudi čas njene tvorbe ali dostopa).

Seveda so tudi performanse (propustnost) sistema NFS praviloma slabše od performans lokalnih datotečnih sistemov.

Okolje porazdeljenega računanja (DCE)

Naš računalniški sistem je omrežje

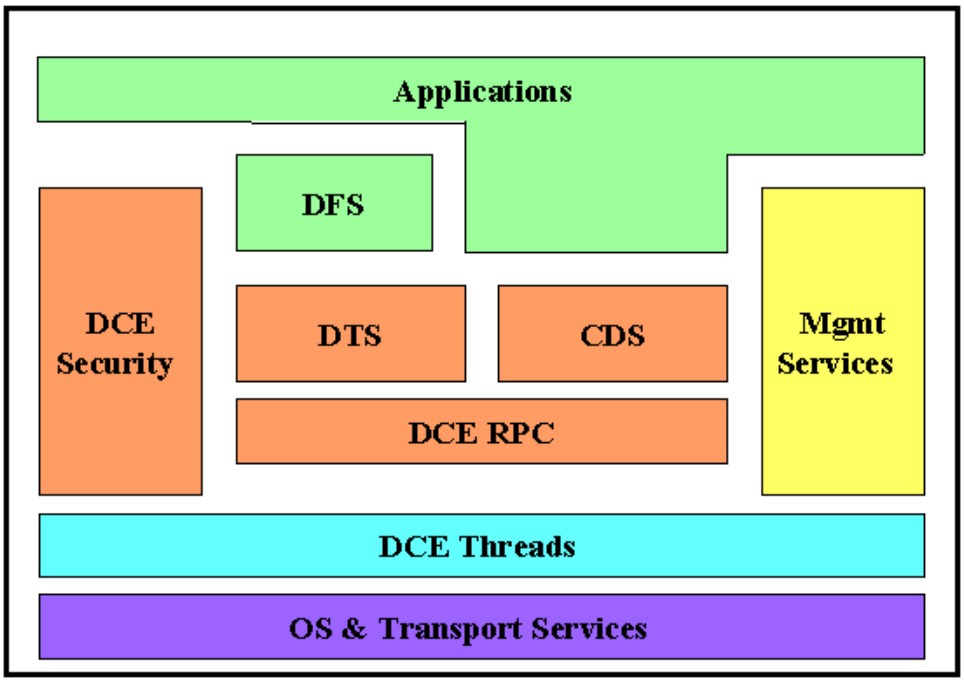


Okolje porazdeljenega računanja (Distributed Computing Environment, DCE) zagotavlja tehnologijo za razvoj aplikacij, ki delajo (in sodelujejo) na različnih platformah, neodvisno od proizvajalca opreme**. DCE** torej ni aplikacija, pač pa **standardizirana množica orodij za podporo aplikacijam**

Servisi DCE

- Name (Principal) resolution service - Cell Directory Services
- Distributed Processing services - Remote Procedure Calls
- Time synchronization service - Distributed Time Service
- Security (authentication, authorization, encryption) service

Komponente DCE



DTS is a secure service to keep system clocks synchronized.
Important in a transaction environment to enable workable transaction queuing and rollback.

Remote Procedure Call (RPC) is a DCE core component that provides a secure, standardized method for asking another machine to perform processing without an explicit login.

CDS (Cell Directory Service): Database of cell information, security information and a name resolution system which provides a mapping of names to resource locations.

Services and systems can be defined independently of specific hardware addresses so that multiple machines can provide replication of data for service reliability

X Windows in Arhitektura klijent strežnik

Introduction

A client-server architecture is a general mechanism for handling a shared resource that several programs may want to access simultaneously.

In the case of X, the shared resources are the drawing area and the input channel. If every process was allowed to write on it at its will, several processes may want to draw at the same place, resulting in an unpredictable chaos.

Thus, only one process is allowed to get access to the drawing area: the X server.

The processes wanting to draw stuff or get inputs send requests to the X servers. They are "clients".

They do this over a communication channel. The X server performs the requests for its clients, and sends them back replies. It may also send messages without explicit client's requests to keep them informed of what is going on. These messages sent by the server on its own behalf are called "events".

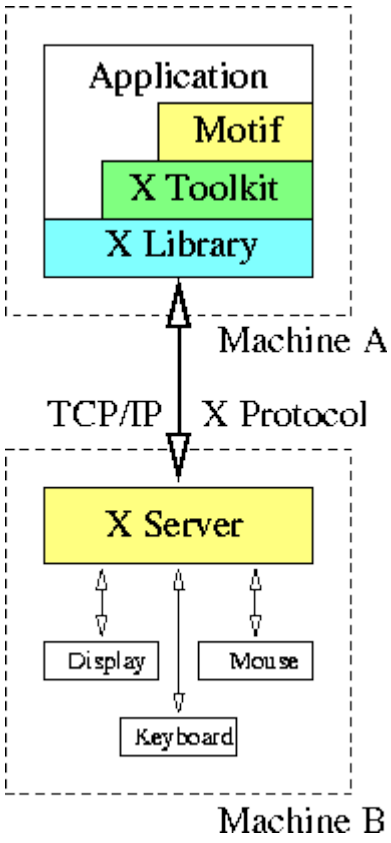
Structure of the X client-server architecture

As we already mentioned, the server and a client communicates over a communication channel. This channel is composed of two layers: the low-level one, which is responsible for carrying bytes in a reliable way (that is with no loss nor duplication). This link may be among others a named pipe in the Unix environment, a DECNet link and of course a TCP/IP connection.

The upper layer use the byte-transport channel to implement a higher-level protocol: the X protocol. This protocol says how to tell the server to request window creation, graphics drawing, and so on, and how the server answers and sends events. The protocol itself is separated into different parts:

- How to connect and how to break a connection,
- how to represent the different data types,
- what are the requests and what they mean and
- what are the replies and what they mean.

The client-server paradigm of the X Window System



On each workstation of X-Terminal an X server is running. The X server' task is to draw on its associated display and to deal with the keyboard, the mouse, and perhaps other input devices. Therefore, the X server is a strongly device-dependent piece of software.

Applications are telling the X server what it should draw and the X server tells the applications about keyboard and mouse events.

The X server acts as a server process fulfilling drawing requests etc. from the clients, the applications. The client-server communication happens via TCP/IP and is therefore network-transparent. The client and server processes could run on the same machine, but they don't have to.

The communication between the X server and its applications is based on the X Protocol.

The X Protocol is a low-level protocol which defines how the applications have to formulate requests to the X server, and how the X server tells the applications about events. No application programmer is supposed to know the details of the X Protocol. He never uses it directly, but always by means of libraries providing handy functions for creating, moving, and destroying windows, etc.

The library functions then formulate the requests to the X server using the X11 Protocol. The most basic library is the *X Library* (Xlib, the library is linked to an application using `-lX11`), in which a window is just a rectangular area on the display. The *X Toolkit* (Xt) defines and implements *widgets*, which are graphical objects and can be grouped together to form the grahical user interface (GUI) of an application. The X Toolkit is built upon the X Library. It never directly sends/receives X Protocol data to/from the network.

Usually an application's interface to the X Window System is the X Toolkit.

Most applications do not use directly X Library functions. Another layer upon the X Toolkit, the Motif toolkit, provides a very nice set of widgets. But an application can't be implemented by just unsing Motif function and widgets, but also need to use directly X Toolkit functions. The functions used in our sample application and having the `Xt` prefix are X Toolkit functions. The functions with `Xm` prefix and the widget classes with `xm`

prefix are from the Motif toolkit.

The window manager

Modern window systems provide a border decorations for the main windows of its applications making it possible for the user to easily activate, resize, move, iconify, etc. the applications' main window. In the X Window System, this is done by window managers, which, in the context set up above, just are special applications.

Window managers may run on different machines as the X server and the other applications. Each display is associated exactly one window manager, but there exists a great variety of these window managers. We just quote a few.

`twm`. *Tom's window manager*. One of the first window managers of the X Window System. It's very basic.

`fvwm`. The *free virtual window manager*. A further development of the `twm` posessing the possibility of having different virtual screens. Mostly used on Linux systems.

`olwm`. The *Open Look Window Manager*. Used by Sun and AT&T.

`mwm`. The *Motif Window Manager* which is part of the Motif distribution.

`dtwm`. The *Desktop Window Manager* of the Common Desktop Environment (CDE). In some aspects similar to `mwm`.

`kdewm`. The window manager of the Kool Desktop Environment (KDE) project, which is apparently replacing the `fvwm` on Linux systems.

The multiplexing mechanism

Consider one of the several buttons of one of the several applications running on a certain display on behalf of an X server.

If the user presses the mouse key at a certain position on the screen, usually only the button widget in whose window the position of the mouse pointer is, should be informed. Other widgets in the application containing the button under consideration and even other applications are usually not interested in the keypress event.

Therefore, the X server just should inform the button widget about the keypress event. Each widget on the screen is associated to an X window, which is merely just a rectangular area on the screen. The X server has information on the geometry of the X windows, and therefore knows in which window the event happened. Via the application context structure XAppContext it also knows the application where the window belongs to. Therefore it can send the keypress event information together with an ID of the window where it happened directly to the involved application without having to bother the other applications. The

application's X Library and X Toolkit receive the keypress event and the window ID and can translate this ID to the button widget. Then, the X Toolkit informs the button widget and an action and/or callback are triggered.

Medprocesna komunikacija

Interprocess communication (IPC) is a means by which two or more processes can communicate by exchanging data.

Interprocess communication protocols (IPCs) allow any two programs to send and receive messages, commands, and responses.

The programs can be running in the same or different environment.

Named pipes, sockets, and remote-procedure calls (RPCs) are three of the most commonly used IPC protocols in today's client/server environments

Named Pipes

Named pipes is the native IPC protocol that is very similar to writing to a file, except that the "file" is referred to as a "named pipe" that can be shared by many different processes at once.

Pipes can be unidirectional (write or read) or bidirectional (write and read), blocking or non-blocking, and dedicated to a single process or used by many processes.

Remote-Procedure Calls

RPCs are the interprocess communication protocol primarily used in Unix environments to do program-to-program communication.

Standardized by X-Open (Open Systems Foundation)

RPCs are implemented as a compiler feature supported by platform-specific RPC run-time libraries that allow the program to make remote-procedure calls in the same manner as a local procedure call would be made.

The run-time library is responsible for "finding" the remote procedure, establishing the connection, and handling the communication.

Sockets

The socket mechanism was originally developed and implemented as part of BSD Unix. The idea is that two processes wishing to communicate set up a pair of "sockets" to create a communication channel between them.

The socket mechanism is the most powerful IPC method, because it allows communication between processes that are running on the same computer or on different computers.

The use of a socket for communication often follows the client/server model.

One method of communication between server and client processes is to design the server following these steps:

- Create the socket.
- Assign a name to the socket.
- Attach a connection to the socket.
- Transfer data via the socket.
- Clean up the socket after use.
- The connection to a socket also uses a socket.

The connecting socket used for a client follows similar steps, except that the assignment of a name is not always necessary.

Vtičnice (Sockets) : BSD Unix, Windows, Java

Writing your own client/server applications using Java

There are two communication protocols that one can use for socket programming: datagram communication and stream communication.

Datagram communication:
The datagram communication protocol, known as UDP (user datagram protocol), is a connectionless protocol, meaning that each time you send datagrams, you also need to send the local socket descriptor and the receiving socket's address. As you can tell, additional data must be sent each time a communication is made.

Stream communication:
The stream communication protocol is known as TCP (transfer control protocol). Unlike UDP, TCP is a connection-oriented protocol. In order to do communication over the TCP protocol, a connection must first be established between the pair of sockets. While one of the sockets listens for a connection request (server), the other asks for a connection (client). Once two sockets have been connected, they can be used to transmit data in both (or either one of the) directions.

Now, you might ask what protocol you should use - UDP or TCP? This depends on the client/server application you are writing.

UDP is an unreliable protocol - there is no guarantee that the datagrams you have sent will be received in the same order by the receiving socket. On the other hand, TCP is a reliable protocol; it is guaranteed that the packets you send will be received in the order in which they were sent.

In short, TCP is useful for implementing network services - such as remote login (rlogin, telnet) and file transfer (FTP) - which require data of indefinite length to be transferred. UDP is less complex and incurs fewer overheads. It is often used in implementing client/server applications in distributed systems built over local area networks.

Programming sockets in Java

In this section we will answer the most frequently asked questions about programming sockets in Java. Then we will show some examples of how to write client and server applications.

Note: In this tutorial we will show how to program sockets in Java using the TCP/IP protocol only since it is more widely used than UDP/IP. **Also:** All the classes related to sockets are in the java.net package, so make sure to import that package when you program sockets.

How do I open a socket?

If you are programming a client, then you would open a socket like this:

```
Socket MyClient;
MyClient = new Socket("Machine name", PortNumber);
```

Where Machine name is the machine you are trying to open a connection to, and PortNumber is the port (a number) on which the server you are trying to connect to is running. When selecting a port number, you should note that port numbers between 0 and 1,023 are reserved for privileged users (that is, super user or root). These port numbers are reserved for standard services, such as email, FTP, and HTTP. **When selecting a port number for your server, select one that is greater than 1,023!**

In the example above, we didn't make use of exception handling, however, it is a good idea to handle exceptions. (From now on, all our code will handle exceptions!) The above can be written as:

```
Socket MyClient;
try {
  MyClient = new Socket("Machine name", PortNumber);
}
catch (IOException e) {
   System.out.println(e);
}
```

If you are programming a server, then this is how you open a socket:

```
ServerSocket MyService;
try {
  MyServerice = new ServerSocket(PortNumber);
}
catch (IOException e) {
  System.out.println(e);
}
```

When implementing a server you also need to create a socket object from the ServerSocket in order to listen for and accept connections from clients.

```
Socket clientSocket = null;
try {
  serviceSocket = MyService.accept();
}
catch (IOException e) {
  System.out.println(e);
}
```

How do I create an input stream?

On the client side, you can use the DataInputStream class to create an input stream to receive response from the server:

```
DataInputStream input;
try {
  input = new DataInputStream(MyClient.getInputStream());
}
catch (IOException e) {
  System.out.println(e);
}
```

The class DataInputStream allows you to read lines of text and Java primitive data types in a portable way. It has methods such as read, readChar, readInt, readDouble, and readLine,. Use whichever function you think suits your needs depending on the type of data that you receive from the server.

On the server side, you can use DataInputStream to receive input from the client:

```
DataInputStream input;
try {
  input = new DataInputStream(serviceSocket.getInputStream());
}
catch (IOException e) {
```

```
    System.out.println(e);
  }
```

How do I create an output stream?

On the client side, you can create an output stream to send information to the server socket using the class PrintStream or DataOutputStream of java.io:

```
PrintStream output;
  try {
    output = new PrintStream(MyClient.getOutputStream());
  }
  catch (IOException e) {
    System.out.println(e);
  }
```

The class PrintStream has methods for displaying textual representation of Java primitive data types. Its Write and println methods are important here. Also, you may want to use the DataOutputStream:

```
DataOutputStream output;
  try {
    output = new DataOutputStream(MyClient.getOutputStream());
  }
  catch (IOException e) {
    System.out.println(e);
  }
```

The class DataOutputStream allows you to write Java primitive data types; many of its methods write a single Java primitive type to the output stream. The method writeBytes is a useful one.

On the server side, you can use the class PrintStream to send information to the client.

```
  PrintStream output;
  try {
    output = new PrintStream(serviceSocket.getOutputStream());
  }
  catch (IOException e) {
    System.out.println(e);
  }
```

Note: You can use the class DataOutputStream as mentioned above.

How do I close sockets?

You should always close the output and input stream before you close the socket.

On the client side:

```
try {
   output.close();
   input.close();
   MyClient.close();
}
catch (IOException e) {
   System.out.println(e);
}
```

On the server side:

```
try {
   output.close();
   input.close();
   serviceSocket.close();
   MyService.close();
}
catch (IOException e) {
   System.out.println(e);
}
```

Examples

In this section we will write two applications: a simple SMTP (simple mail transfer protocol) client, and a simple echo server.

Let's write an SMTP (simple mail transfer protocol) client -- one so simple that we have all the data encapsulated within the program. You may change the code around to suit your needs. An interesting modification would be to change it so that you accept the data from the command-line argument and also get the input (the body of the message) from standard input. Try to modify it so that it behaves the same as the mail program that comes with Unix.

```
import java.io.*;
import java.net.*;

public class smtpClient {
   public static void main(String[] args) {
```

```
// declaration section:
// smtpClient: our client socket
// os: output stream
// is: input stream

      Socket smtpSocket = null;
      DataOutputStream os = null;
      DataInputStream is = null;

// Initialization section:
// Try to open a socket on port 25
// Try to open input and output streams

      try {
         smtpSocket = new Socket("hostname", 25);
         os = new DataOutputStream(smtpSocket.getOutputStream());
         is = new DataInputStream(smtpSocket.getInputStream());
      } catch (UnknownHostException e) {
         System.err.println("Don't know about host: hostname");
      } catch (IOException e) {
         System.err.println("Couldn't get I/O for the connection to: hostname");
      }

// If everything has been initialized then we want to write some data
// to the socket we have opened a connection to on port 25

      if (smtpSocket != null && os != null && is != null) {
         try {

// The capital string before each colon has a special meaning to SMTP
// you may want to read the SMTP specification, RFC1822/3

            os.writeBytes("HELO\n");
            os.writeBytes("MAIL From: k3is@fundy.csd.unbsj.ca\n");
            os.writeBytes("RCPT To: k3is@fundy.csd.unbsj.ca\n");
            os.writeBytes("DATA\n");
            os.writeBytes("From: k3is@fundy.csd.unbsj.ca\n");
            os.writeBytes("Subject: testing\n");
            os.writeBytes("Hi there\n"); // message body
            os.writeBytes("\n.\n");
            os.writeBytes("QUIT");

// keep on reading from/to the socket till we receive the "Ok" from SMTP,
// once we received that then we want to break.

            String responseLine;
            while ((responseLine = is.readLine()) != null) {
```

```
                System.out.println("Server: " + responseLine);
                if (responseLine.indexOf("Ok") != -1) {
                  break;
                }
              }

// clean up:
// close the output stream
// close the input stream
// close the socket

              os.close();
              is.close();
              smtpSocket.close();
            } catch (UnknownHostException e) {
              System.err.println("Trying to connect to unknown host: " + e);
            } catch (IOException e) {
              System.err.println("IOException:  " + e);
            }
        }
    }
}
```

When programming a client, you must follow these four steps:

- Open a socket.
- Open an input and output stream to the socket.
- Read from and write to the socket according to the server's protocol.
- Clean up.

These steps are pretty much the same for all clients. The only step that varies is step three, since it depends on the server you are talking to.

Now let's write a server. This server is very similar to the echo server running on port 7. Basically, the echo server receives text from the client and then sends that exact text back to the client. This is just about the simplest server you can write. Note that this server handles only one client. Try to modify it to handle multiple clients using threads.

```
import java.io.*;
import java.net.*;
```

```
public class echo3 {
   public static void main(String args[]) {

// declaration section:
// declare a server socket and a client socket for the server
// declare an input and an output stream

      ServerSocket echoServer = null;
      String line;
      DataInputStream is;
      PrintStream os;
      Socket clientSocket = null;

// Try to open a server socket on port 9999
// Note that we can't choose a port less than 1023 if we are not
// privileged users (root)

      try {
        echoServer = new ServerSocket(9999);
      }
      catch (IOException e) {
        System.out.println(e);
      }

// Create a socket object from the ServerSocket to listen and accept
// connections.
// Open input and output streams

      try {
        clientSocket = echoServer.accept();
        is = new DataInputStream(clientSocket.getInputStream());
        os = new PrintStream(clientSocket.getOutputStream());

// As long as we receive data, echo that data back to the client.

        while (true) {
          line = is.readLine();
          os.println(line);
        }
      }
      catch (IOException e) {
        System.out.println(e);
      }
   }
}
```

Zgrešene predpostavke

The network connecting clients and servers is a crucial component of a client/server system and can often be a bottleneck.

Client/Server systems are more difficult to maintain than and network failures are serious.

Essentially everyone, when they first build a distributed application, makes the following 7 assumptions. All prove to be false in the longrun and all cause big trouble and painful learning experiences.

- The network is reliable
- Latency is zero
- Bandwidth is infinite
- The network is secure
- Topology doesn't change
- There is one administrator
- Transport cost is zero

Naloge in postopki sistemskega administratorja

Pregled nalog

System administration regardless if you are running UNIX or NT requires common tasks to be performed on a daily, weekly, and monthly basis. Some of these tasks include

- adding/deleting user accounts
- adding new hosts to the network
- backing up file systems
- installing patches and new software
- monitoring system resources
- running cable for the network

- making sure your boss can read his/her E-mail
- writing scripts to automate your work
- controlling printing
- searching for security holes in your network

Please note that all of these physical jobs are important, but the hardest part about system administration is making the right decision when it comes to blatant or unintentional mis-use of a system. That answer only comes with experience.

User support

1. Highly visible aspect of the system administrator's job but must be relegated to the background in favor of systemwide issues
2. Gather information about the problem from the user
3. Rule out possibilities, theorize, and tweak things till they work

Maintain hardware

- Computers, X terminals, printers, modems, other peripherals

Maintain software

- Operating system, applications, services
- Install upgrades of operating system and applications
- Provide support for installation and maintenance of various packages, such as TeX, MathCAD, and `elm`
- Maintain user accounts, including logins, files, and printer access

Maintain computers and networks

- Physical network
- Server configurations

Maintain security

- Monitor the network, servers, and other workstations for suspicious access
- Install security patches on different operating systems

- Install security features on the network to deny access
- Install virus protection software on PCs
- Upgrade all the systems to guard against any perceived threats based on advisories and other information

Maintain information services

- E-mail
- News
- Gopher
- World-wide web

Train users

- Resolve problems associated with system use (restoring *lost* files and answering questions about email)
- Give seminars to users on the use of newly installed software
- Prepare small handouts on newly installed software and upgrades to be distributed to the users

Maintain system documentation and library

- Maintain local documentation and records on
- Hardware installation or removal
- Software installation or removal
- Documentation on new software and hardware installation and troubleshooting
- Keep track of manuals and documentation in the system library

Miscellaneous

- Order new hardware and software
- Perform backups

Rezervne kopije

Doing back-ups and restores is probably a system administrator's most hated job. Users constantly delete their files and need them restored ASAP and there is always the occasional disk or power crash. These two reasons by themselves should be enough warning to keep backups.

There are many ways to do back-ups with UNIX. Some good commands to learn are tar, cpio, dd, pax, and mt. All of these have their cool features.

Avtomatozacija nalog

Automating tasks saves a system administrator hours of time. It also keeps the load on the computer and network to a minimum by running programs off hours. Some of the things you would want to automate would be backing up file systems, condensing of system logs, restarting the web server, or emptying out /tmp.

The most common UNIX automation is the cron. The /var/spool/cron is a file that is unique for each user. By default you do not have one. To create a cron type crontab -e (the -e is for edit). The format of this file is very specific. The fields in order are:

minute: 0-59
hour: 0-23 (note military time)
day: 1-31
month: 1-12
weekday: 0-6 (0 is Sunday...)

For example, the follow example will print "Hello world!" to the console everyday at 7 am. Note the * matches all values for that field.

```
0 7 * * * echo "Hello world!" 2>&1 /dev/console
0 8,9,10 * * * echo "Are you still up?" 2 >1& /dev/console
```

With the use of a comma the above example prints "Are you still up?" at 8, 9, 10 am.
The three flags you can use with crontab:
crontab -e ==> Edit/Create your crontab
crontab -l ==> List the entries in your crontab
crontab -r ==> Delete all entries in your crontab file

Note: Super-User can perform any of these operations on anyone's crontab whereas a user can only use them on his/her own crontab file.

Windows NT does not come with any out of the box software for doing automated tasks, but there are many easy to use programs on the net that can assist you, for example, WINAT that comes with the "Windows NT Resource Kit". Assuming you didn't buy the Resource kit you can use the command line interface and execute an AT command. This command can call

batch jobs that run your program and then reschedule your job to run for the next time. This is Windows NT's answer for cron jobs. Here is the syntax:

```
at [\\computername] time [/every:date[,...] | /next:date[,...] ] command
```

to remove a scheduled job at #id number /delete

For example, to run a program called "restart" on yellow everyday at 7am on the 13, 14, and 15 of every month type the following.

```
at \\yellow 7:00 /every:13,14,15 "restart"
```

Nadzor sistemskih virov

Resources on a computer can become a problem when they aren't monitored correctly. The "Bottlenecks" are

- CPU performance
- Memory usage
- Disk performance
- Network performance (Discussed in Network Administration)

There are many share ware/free ware and commercial programs out there to monitor and alert you that there may be a problem. Windows NT comes with Performance Monitor, which when configured correctly can alert you to potential problems.

In UNIX the first thing you should check is the load on the machine by typing *uptime* Uptime will give you Time, Time since last reboot, # of users, #of jobs in run queue in last 1, 5, 15 minutes.

Any loads over 2.0-3.0 should be investigated.

To investigate use *ps*. PS will give you a list of all active processes running on the system. With certain flags you will get different output, for example, ps -a returns:

```
PID TTY      TIME CMD
7199 pts/9   12:45 netscape
7196 pts/5    0:09 rlogin
6488 pts/6    0:00 ps
7195 pts/5    0:00 rlogin
8466 pts/7    0:01 vi
```

Memory information can be found with *ps* and *swap -l*.To interactively watch memory usage use *vmstat*, or *osview*. Another outside program that is extremely useful is *top*. It lists all process and can be configured to update itself at a given interval.

Monitoring disk performance by invoking *iostat -v* This gives you reads and wites per sec, Kilobyte reads and writes per sec, queue length, ave. Transactions being serviced, ave. Service time, % time the queue is empty, % time disk busy.

Windows NT roles all of the above commands into a GUI interface called Performance Monitor (Perf Mon) located in the Administrative Tools Group. By default the Perf Mon doesn't monitor anything, but by clicking the "+" button you get a objects listing of the following:

Browser, Cache, Logical Disk, Memory, NBT Connections, Objects, Paging File, Physical Disk, Process, Processor, Redirector, Server, Server Work Queues, System, Telephony, and Thread.

Select the object than specify the Counter you want to watch and click ADD. When you are done click DONE. You can save this log for future reference.

Another feature to use is the Windows NT Diagnostics package that is located with Perf Mon under administrative Tools Group. This gives you general info about your NT box and its resources.

One last "home grown" package to help you is the Task Manager, this gives you cpu and memory usage etc. Right Click on the task bar and select Task Manager.

Organiziranje in vzdrževanje datotečnega sistema

Windows NT and Solaris' file structures are very different. Windows prefers a deep file structure under the winnt directory (c:\winnt). Solaris on the other hand prefers a file structure that is split over many directories from the root (\). I will start with the names of the Solaris file structure and their definitions.

/ This is the root file directory housing the following directories.

/bin Bin houses the executable files for the system others are /usr/bin, /sbin, and /usr/ubc.

/dev This is the device directory, it is used for the physical disks, tape drives, and terms.

/etc This directory is mainly for configuration files, most importantly the system boot scripts.

/lost+found Houses files that were lost due to an unexpected system failure, these files are found when fsck is performed.

/lib Holds libraries for programming languages like C and fortran

/mnt This is an empty directory reserved for mounting remove file systems like a CD Rom or other partition. /tmp Used as a scratch directory and swap space for memory, remember to clear this directory every once and awhile.

/home Used to house user's home directories.

/usr Used for local programs and packages.

/usr/adm Administrative tools like Top, and any locally created tools.

/usr/include Houses C header files, I/O, and math libraries.

/usr/lib More programming libraries along with the libraries for X.

/usr/local Locally created/used executables for public access.

/usr/man The man files for the system are in this directory in order of importance.

/var Houses printer termcaps, printing, and mail files.

/var/spool Used for spooling printing, mail, and cron jobs.

/var/spool/mail Mail directory for user's mail file used for incoming mail.

NT has a some what different structure. Top level directories that are important are:

c:\ top level directory that houses the paging file.

c:\program files For putting common programs for users.

c:\winnt (or wherever you install nt) This directory houses all the important stuff:

c:\winnt\inf Holds all the inf files that configure some programs.

c:\winnt\profiles This is where your users environment settings are held.

c:\winnt\repair Houses your original system configuration.

c:\winnt\system This directory contains most system files (*.sys) for controlling the system and some fonts.

c:\winnt\system32 Contains 32-bit "old DOS" commands and some other non essential commands