VSO
Vzdrževanje sistemske programske opreme

Andrej Štrancar

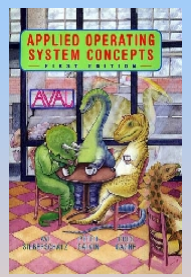
Snov predavanj:

- Razvoj operacijskih sistemov
- Programiranje v lupini **bash**
- Procesi, sočasnost, sinhronizacija
- Internet, internetni protokoli, storitve
- Varnost, kriptiranje, elektronski podpis
- Datotečni sistemi, primeri: UNIX, NTFS
- Porazdeljeni sistemi, datotečni sistem NFS

Vaje:

- Spoznavanje Linuxa
- Programiranje v lupini **bash**
- Medprocesna komunikacija; **java / c**
- Internet: TCP/IP
- Elektronska pošta, FTP
- Varnost: certifikati, VPN
- Delo na daljavo: telnet, SSH, *remote desktop*
- Porazdeljen datotečni sistem NFS, SAMBA

Literatura:



Applied Operating System Concepts
Silberschatz, Galvin, Gagne
John Wiley & Sons, Inc. ISBN 0-471-36508-4

- **Zapiski (prosojnice) s predavanj**
- **Skripta za SPO; Saša Divjak**

Obveznosti, ocena:

- Opravljene vaje
- Pisni izpit ali seminar (po dogovoru)
- Ustni izpit

- Skupna ocena je povprečje ocene vaj, pisnega izpita/seminarja in ustnega izpita.

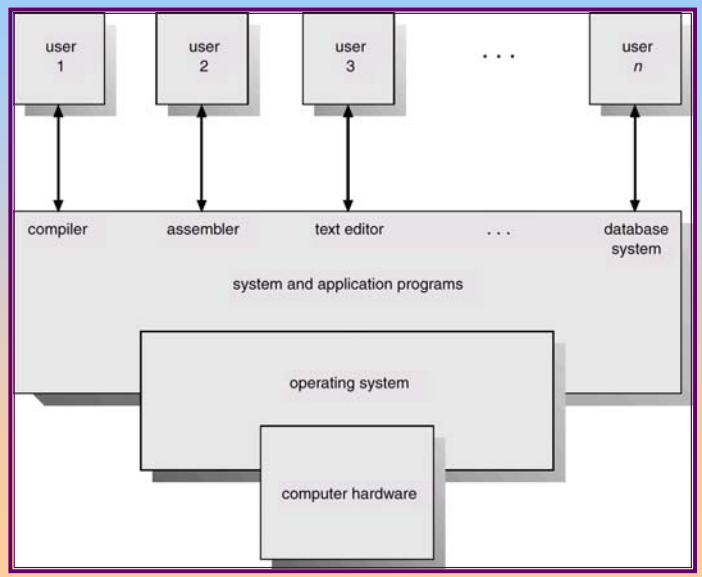
Kaj je operacijski sistem?

- Program, ki je posrednik med uporabnikom računalnika in računalnikovo strojno opremo.
- Namen operacijskega sistema:
  - Izvajanje uporabnikovih programov in uporabniku olajšati reševanje nalog.
  - Narediti računalniški sistem enostaven za uporabo.
- Izrabiti strojno opremo na učinkovit način.

Komponente računalniškega sistema

1. Strojna oprema (hardware) – zagotavlja osnovne računske vire (CPE, pomnilnik, V/I naprave).
2. Uporabniški programi – uporabniški programi opisujejo, kako uporabljati te računske vire za reševanje računskih problemov.
3. Uporabniki (ljudje, stroji, drugi sistemi).
4. Operacijski sistem – nadzoruje in koordinira uporabo strojne opreme med različnimi uporabniškimi programi za različne uporabnike.

Komponente računalniškega sistema



Lastnosti OS :

- OS nadzoruje / usmerja uporabo strojne opreme
- dodeljuje in upravlja vire (resource allocation)
- olajšuje delo ⇒ povečuje učinkovitost ...

- ustvariti prijazno okolje, kjer bo uporabnik lažje izvajal svoje programe
- omogoča učinkovito uporabo strojne opreme

Na začetku :

- računalniki so bili fizično zelo veliki
- upravljali in poganjali so jih z enega, centralnega mesta – konzole
- programer je bil tudi operater

postopek:

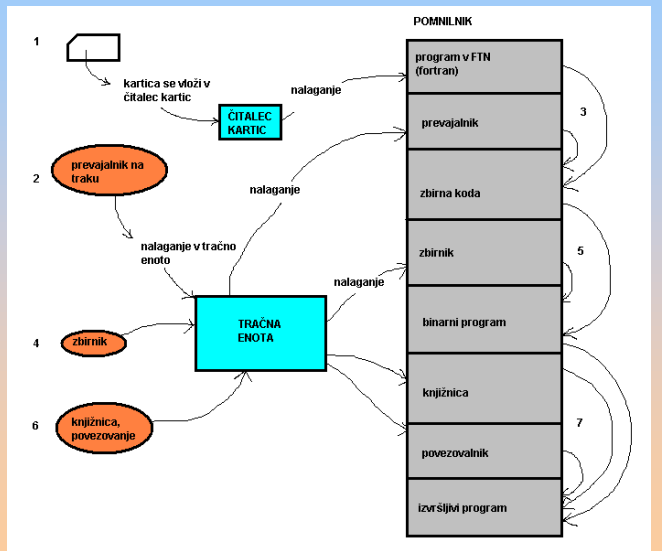
- programer je napisal program v strojnem jeziku
- poskrbel je za nalaganje v pomnilnik preko konzole
- vpisal je začetni naslov v programski števec
- pognal je izvajanje programa
- opazoval je dogajanje na konzoli
- če je prišlo do napake, je izvajanje ustavil in odčital vsebine registrov, ki jih je lahko tudi ročno spreminjal
- izpisal je podatek

Kasneje :

strojna oprema :
- čitalci kartic
- boljši tiskalniki
- enote z magnetnim trakom

programska oprema:
- zbirniki (assembler)
- programi: nalagalnik, ki je omogočil hitrejši prenos podatkov iz kartice v pomnilnik
- knjižnice
- povezovalniki, ki olajšujejo programiranje
- gonilniki (driverji), ki olajšujejo delo z napravami
- prevajalniki

Postopek :



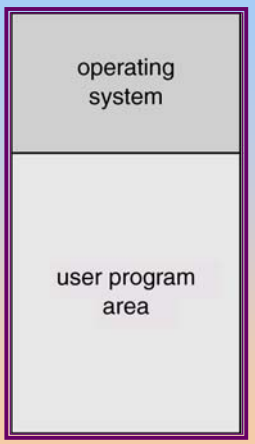
Enostavna paketna obdelava

- Profesionalni operaterji
- Podobna opravila združujejo v pakete (manjši vzpostavitveni čas)
- Avtomatično nizanje opravil – to dela operacijski sistem.
- Rezidentni monitor
  - Najprej prevzame nadzor monitor
  - Nadzor se prenese na posel
  - Ko se opravilo konča, se nadzor vrne monitorju

Rezidentni monitor

- ali naj aktivira nalagalnik
- kako so v paketu razmejeni programi
- katerega od prevajalnikov naj naloži
- kdaj naj požene assembler
- kdaj naj požene uporabniški program...

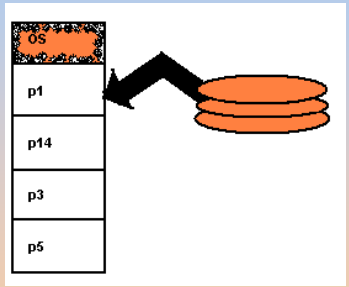
kontrolne kartice



Multiprogramska paketna obdelava

Prekrivanje računskih in V/I operacij

- v pomnilniku je OS ter eden ali več poslov, ki so prišli z diska
- OS enega izbere, npr. p_1 in ga začne izvajati
- če se p_1 ustavi in čaka na nek dogodek, OS izbere drug posel in ga začne izvajati, npr. p_{14}
- ko je p_1 spet pripravljen, lahko procesor nadaljuje izvajanje p_1



Vprašanja ob multiprogramiranju

- kako se odločiti, kateri posel naložiti v delovni pomnilnik (razvrščanje poslov)?
- razvrščanje na procesorju = CPU scheduling ali procesor scheduling?
- Kako upravljati s pomnilnikom, fragmentacija?
- Zaščita?
- Komunikacija, sinhronizacija?

Večopravilni sistemi z dodeljevanjem časa

- **multiprogramiranje** – preklaplanje (dodeljevanje procesorja procesom) mora biti dovolj hitro
  - časovno dodeljevanje
- uporaba navideznega pomnilnika
- menjavanje (swapping) – umikanje programov v celoti na disk
- datotečni sistem
- upravljanje z diskom
- varnost / zaščita
- komunikacija / sinhronizacija . . .

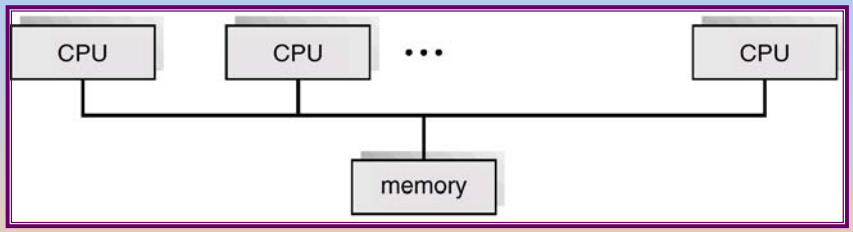
Vzporedni sistemi

- Večprocesorski sistemi z več kot eno CPE, ki so tesno povezane.
- *Tesno povezan sistem* – procesorji imajo skupen pomnilnik; običajno komunicirajo preko skupnega pomnilnika.
- Prednosti vzporednih sistemov:
  - Povečana *propustnost*
  - Ekonomičnost
  - Povečana zanesljivost
    - Manj občutljivi na napake zaradi odpovedi
    - Odpoved ene CPE zgolj zmanjša zmogljivost

Vzporedni sistemi

- *Simetrično multiprocesiranje*
  - Vsak procesor poganja identično kopijo operacijskega sistema.
  - Hkrati se lahko izvaja več procesov.
  - Večina modernih OS podpira simetrično multiprocesiranje
- *Asimetrično multiprocesiranje*
  - Vsakemu procesorju se dodeli določen posel; obstaja glavni procesor, ki drugim dodeljuje posle.
  - Bolj pogosto v zelo velikih sistemih

Simetrično multiprocesiranje



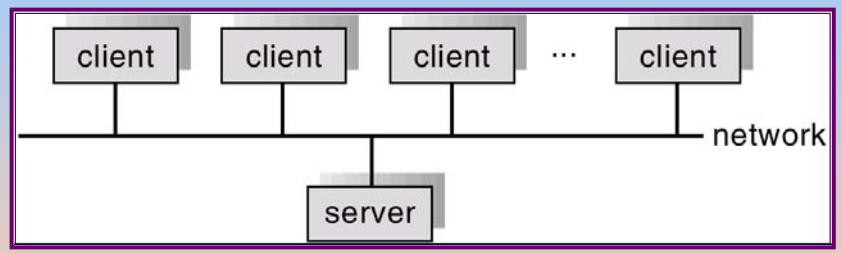
Porazdeljeni sistemi

- Računaje porazdelimo na več fizičnih procesorjev.
- *Šibko povezan sistem* – vsak procesor ima svoj lokalni pomnilnik; komunikacija poteka po različnih tipih povezav kot so vodila ali telefonske žice – odvisni so od mreže.
- Prednosti porazdeljenih sistemov.
  - Deljenje virov
  - Večja hitrost računanja – porazdelitev bremena
  - Zanesljivost
  - Omogočajo komunikacijo

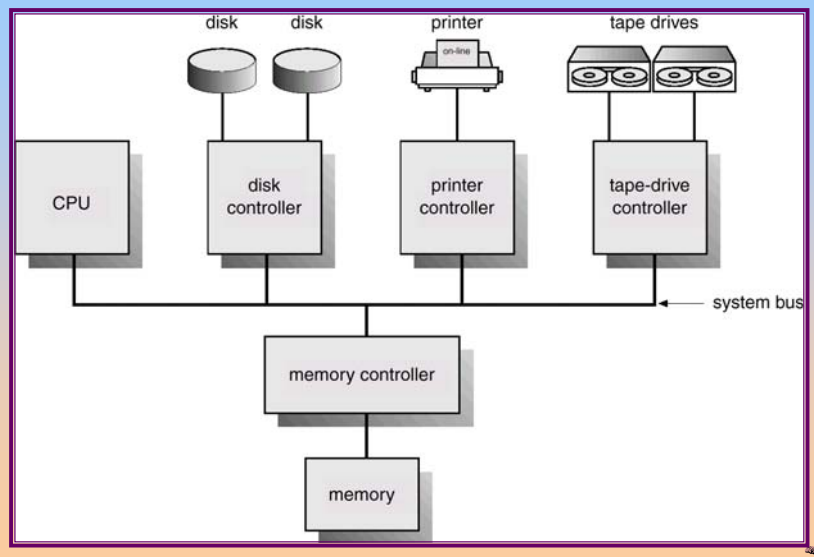
Porazdeljeni sistemi

- Funkcionalno odvisni od računalniške mreže.
- Hitre lokalne mreže (LAN), širše (prostrane) mreže (WAN)
- Lahko so organizirani kot odjemalec-strežnik ali enak z enakim (peer-to-peer).

Struktura odjemalec-strežnik



Računalniški sistem in njegovo delovanje



Delovanje sistema

- V/I naprave in CPE delajo vzporedno.
- Vsak krmilnik V/I naprav ima medpomnilnik – buffer.
- CPE kopira (premika) podatke med medpomnilniki V/I naprav in glavnim pomnilnikom
- V/I naprava ima dostop samo do medpomnilnika v krmilniku.
- Krmilnik obvesti CPE, da se je neka operacija končala tako, da sproži *prekinitev*.

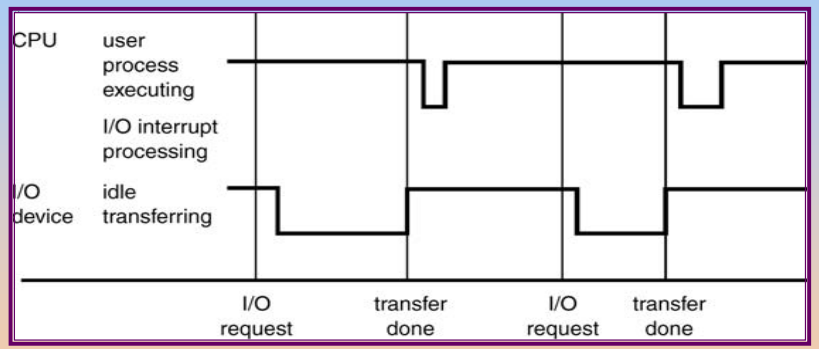
Prekinitve

- Prekinitev sproži ustrezni prekinitveni servisni program. Kje v pomnilniku se ta program nahaja, je določeno s *prekinitvenim vektorjem*, ki vsebuje naslov prvega ukaza prekinitvenega servisnega programa.
- Zagotoviti je potrebno transparentnost – shraniti naslov, kjer je bilo izvajanje prekinjeno - *povratni naslov*.
- Poznamo tudi programske prekinitve – *pasti,* ki se sprožijo ob določenih napakah ali na zahtevo uporabnika.
- Moderni OS je voden s prekinitvami (*interrupt* driven).

Prekinitve

- Ob prekinitvah je potrebno shraniti *stanje CPE,* da zagotovimo nemoteno nadaljevanje prekinjenega programa – *transparentnost prekinitev*.
- Ugotoviti, katera naprava je sprožila prekinitev – *prepoznavanje naprave*:
  - *Programsko izpraševanje*
  - *Vektorske prekinitve*
- Izvede se ustrezen prekinitveni servisni program

Dogajanje ob prekinitvah



Zagon računalniškega sistema

- Požene se program v ROM-u, ki naloži sistem. Na ta program - *nalagalnik* kaže *reset vektor*.
- Nalagalnik mora znati najti in naložiti OS v RAM
- Požene prvi *proces,* npr. init. Ta proces požene še druge procese, tudi grafični uporabniški vmesnik / ***lupino***.

Lupina *bash*

- Linux ima nameščenih več različnih lupin:
  - Bourne shell (sh), C shell (csh), Korn shell (ksh), TC shell (tcsh), Bourne Again shell (bash).
- Najpogostejša lupina je "bash". Bash je komptibilna s sh in združuje uporabne lastnosti lupin Korn shell (ksh) in C shell (csh).
- Ustreza standardu IEEE POSIX P1003.2/ISO 9945.2.
- Glede na sh je izboljšana; glede programiranja in interaktivne uporabe.

Programi ali Skripti ?

- bash ni zgolj zelo dobra lupina za delo v ukazni vrstici, vsebuje tudi močan skriptni jezik. Uporaba skriptov omogoča uporaba zmogljivosti lupine in avtomatizacijo mnogih opravil, ki bi sicer zahtevala izvršitev mnogo ukazov.
- Razlike med programskimi in skriptnimi jeziki:
  - Programski jeziki so v splošnem mnogo močnejši in hitrejši od skriptnih. Programski jeziki se prevajajo v izvršljive (strojne) programe. Izvršljiv strojni progam je težko prenosljiv na druge operacijske sisteme.
  - Skriptni jezik se ne prevaja v strojne programe. Programi v skriptnih jezikih se interpretirajo; interpretira jih **tolmač** (interpreter). Tolmač izvaja vsak skriptni ukaz posebej. Zato so skriptni programi počasnejši. Njihova glavna prednost je prenosljivost na druge sisteme. bash je skriptni jezik; drugi popularni skriptni jeziki so Perl, Lisp, Tcl….

Prvi program v bash

- Programe pišemo v urejevalniku besedil; so ASCII datoteke. V Linuxu je več urejevalnikov besedil, skoraj vedno sta na voljo vsaj:
  - `vi, emacs (ali xemacs).`
- Poženemo urejevalnik vi:
  - `$ vi &`

  prvi program ima le dve vrstici:
  - `#!/bin/bash`
    `echo "Hello World"`
- S prvo Linuxu povemo, da mora datoteko (skript) interpretirati z bash. Datoteko poimenujemo, npr. hello.sh. Datoteko moramo narediti izvršljivo; nato jo poženemo:
  - `$ chmod 700 hello.sh`
  - `$./hello.sh`

  `Hello World`

Drugi program v bash

- Program prekopira vse datoteke (v trenutnem direktoriju) v direktorij *trash,* nato pa ta direktorij skupaj z njegovo vsebino zbriše. To dosežemo z zaporedjem ukazov:
  - ```
    $ mkdir trash
    $ cp * trash
    $ rm -rf trash
    $ mkdir trash
    ```
- Namesto tipkanja zaporedja ukazov lahko napišemo program v bash:
  - ```
    $ cat trash
    #!/bin/bash
    # this script deletes some files
    cp * trash
    rm -rf trash
    mkdir trash
    echo “Deleted all files!”
    ```

Spremenljivke

- Načeloma so vse spremenljivke v bash nizi, vendar obstajajo ukazi za matematične operacije, ki omogočajo računanje oziroma uporabo spremenljivk kot števil.
- Spremenljivk ni potrebno deklarirati, spremenljivka se ustvari, ko ji dodelimo vrednost.
- Primer
  - ```
    #!/bin/bash
    STR=“Hello World!”
    echo $STR
    ```
- Vrstica 2 ustvari spremenljivko STR spremenljivka je niz z vrednostjo "Hello World!". Ko spremenljivko uporabimo, pred njenim imenom dodamo znak '$'. Včasih lahko pride pri imenih spremenljivk do dvoumnosti; takrat uporabimo npr. ${SPR}.
- Primer
  - ```
    STR=INŽENIR
    STRING=vaje
    echo $STRING   …  vaje
    echo ${STR}ING   …  INŽENIRING
    ```

Pozor !

- Tip spremenljivk je en sam; spremenljivka vedno vsebuje niz. Če so v nizu samo števke, lahko s spremenljivko računamo, kot da bi spr. predstavljala število…
  - count=0
  - count=Sunday
- V zgornjem primeru smo spremenljivki spremenili *tip*, najprej je bila število, sedaj je niz. To lahko zelo oteži razumevanje (popravljanje) skripta; priporočljivo je, da tipa spremenljivke ne spreminjamo.
- \ je ubežni znak z njim vnašamo znake, ki imajo sicer poseben pomen.
  - ```
    $ ls *
    ls: *: No such file or directory
    ```

Enojni in dvojni narekovaji

- Če želimo spremenljivki prirediti niz, ki vsebuje presledke ali druge posebne znake, moramo niz pisati med enojnimi ali dvojnimi narekovaji.
- Uporaba dvojnih narekovajev pomeni, da se spremenljivke v nizu nadomestijo s svojimi vrednostmi
  - ```
    $ var=“test string”
    $ newvar=“Value of var is $var”
    $ echo $newvar
    Value of var is test string
    ```
- Uporaba enojnih narekovajev je ‘močnejša’, niz je točno tak, kot smo ga podali
  - ```
    $ var=’test string’
    $ newvar=’Value of var is $var’
    $ echo $newvar
    Value of var is $var
    ```


    ```

Ukaz export

- Ukaz export spremenljivko 'izvozi', dostopna bo potomcem procesa:

```
$ x=hello
$ bash        # Run a child shell.
$ echo $x     # Nothing in x.
$ exit        # Return to parent.
$ export x
$ bash
$ echo $x
hello         # It's there.
```

- Če potomec spremeni x, se sprememba ne odrazi pri očetu procesa. To lahko preverimo na naslednji način:

```
$ x=ciao
$ exit
$ echo $x
hello
```

Spremnljivke okolja

- Obstajata dve vrsti spremenljivk:
  - Lokalne spremenljivke
  - Spremenljivke okolja
- Spremenljivke okolja postavi sistem, do njih lahko pridemo z ukazom env. Te spremenljivke imajo posebne vrednosti, npr.

```
$ echo $SHELL
/bin/bash
$ echo $PATH
/usr/X11R6/bin:/usr/local/bin:/bin:/usr/bin
```

- Spremenljivke okolja so definirane v skriptih /etc/profile, /etc/profile.d/ in ~/.bash_profile. To so inicializacijski skripti, ki se izvedejo ob zagonu lupine. Ob izhodu iz lupine se izvede skript ~/.bash_logout

Spremenljivke okolja

- HOME: Privzeti parameter (domač direktorij) ukaza **cd**.
- PATH: Pot za iskanje ukazov. Vsebuje seznam direktorijev (ločenih z dvopičji), v katerih lupina išče programe, ukaze...
- Skripte običajno poganjamo na naslednji način:
  - $ ./script_name
- Če določimo PATH=$PATH:. Smo v PATH vključili trenutni direktorij in lahko tipkamo kar:
  - $ script_name
- Običajno (zaradi varnosti) imamo izvršljive datoteke v posebnem direktoriju
  - $ mkdir ~/bin

  v ~/.bash_profile lahko vključimo:
  - PATH=$PATH:$HOME/bin
  - export PATH

  in dodamo mapo v PATH ob zagonu lupine.

Spremenljivke okolja

- LOGNAME: ime uporabnika
- HOSTNAME: ime računalnika.
- PS1: oblika najavke
  - ✓ \t hour
  - ✓ \d date
  - ✓ \w current directory
  - ✓ \W last part of the current directory
  - ✓ \u user name
  - ✓ \$ prompt character

  Primer

```
[janez@homelinux janez]$ PS1='zivjo \u *'
zivjo janez* _
```

Ukaz Read

- Ukaz read omogoča interakcijo z uporabnikom.
- Primer

```
#!/bin/bash
echo -n “Enter name of file to delete: ”
read file
echo “Type 'y' to remove it, 'n' to change your mind ... ”
rm -i $file
echo "That was YOUR decision!"
```

- Vrstica 3 ustvari spremenljivko file, njena vrednost je niz znakov, ki ga vnese uporabnik. Do spremenljivke dostopamo kot običajno, pred njenim imenom pišemo '$'.

Substitucija ukazov

- Vzvratni narekovaj “`” je drugačen od enojnega “’”. Uporablja se ga za nadomeščanje (substitucijo) z rezultatom ukaza: `command`.

```
$ LIST=`ls`
$ echo $LIST
hello.sh read.sh
PS1=“`pwd`>”
/home/janez/didaktika/>
```

- Običajno substitucijo ukazov izvedemo na preglednejši način $(command), ki pa ni združljiv s sh.

```
$ LIST=$(ls)
$ echo $LIST
hello.sh read.sh
rm $( find / -name “*.tmp” )
$cat > backup.sh
#!/bin/bash
OF=/home/janez/backup-$(date +%d-%m-%y).tar.gz
tar -czf $OF $HOME
```

Aritmetika

- Za izvajanje aritmetičnih operacij lahko uporabimo ukaz **let**:

```
$ let X=10+2*7
$ echo $X
24
$ let Y=$X+2*4
$ echo $Y
32
```

- Aritmetični izraz lahko ovrednotimo z $[expression] ali $((expression))

```
$ echo “$((123+20))”
143
$ VALUE=$[123+20]
$ echo “$[10*$VALUE]”
1430
```

Aritmetične operacije

- Podprte operacije: +, -, /, *, %
- Primer

```
#!/bin/bash
echo -n “Enter the first number: ”; read x
echo -n “Enter the second number: ”; read y
add=$(($x + $y))
sub=$(($x - $y))
mul=$(($x * $y))
div=$(($x / $y))
mod=$(($x % $y))
# print out the answers:
echo “Sum: $add”
echo “Difference: $sub”
echo “Product: $mul”
echo “Quotient: $div”
echo “Remainder: $mod”
```

Pogojni stavki

- Osnovna oblika pogojnega stavka je naslednja:
  - if [*expression*];
    then
      statements
    elif [*expression*];
      then
        statements
    else
        statements
    fi
  - Sekciji (in ukaza) za elif (else if) in else sta neobvezni

Primerjave

- Primerjave so lahko naslednjih tipov: primerjava nizov, numerična primerjava, datotečni operatorji in logični operatorji. Izrazi imajo obliko [expression]:
- Primerjave nizov:
  - = ali sta niza enaka?
  - != ali niza **nista** enaka?
  - -n ali je dolžina niza večja od 0?
  - -z ali je dolžina niza enaka 0?
- Primeri:
  - [s1 = s2] (true če sta s1 in s2 enaka, sicer false)
  - [s1 != s2] (true če s1 in s2 **nista** enaka, sicer false)
  - [s1] (true če s1 ni prazna spr., sicer false)
  - [-n s1] (true če je s1 dolžina s1 večja od 0, sicer false)
  - [-z s2] (true če je dolžina s2 enaka 0, sicer false)

Primerjave

- Primerjave števil:
  - -eq enak kot
  - -ge večji ali enak
  - -le manjši ali enak
  - -ne različen kot
  - -gt večji
  - -lt manjši
- Primeri:
  - [n1 -eq n2] (true če sta n1 in n2 enaki, sicer false)
  - [n1 -ge n2] (true če n1 >= n2, sicer false)
  - [n1 -le n2] (true če n1 <= n2, sicer false)
  - [n1 -ne n2] (true če sta n1 in n2 različna, sicer false)
  - [n1 -gt n2] (true če n1 > n2, sicer false)
  - [n1 -lt n2] (true če n1 < n2, sicer false)


- ```
  #!/bin/bash
  echo -n “Enter your login name: "
  read name
  if [ “$name” = “$USER” ];
  then
       echo “Hello, $name. How are you today ?”
  else
       echo “You are not $USER, so who are you ?”
  fi
  ```
- ```
  #!/bin/bash
  echo -n “Enter a number 1 < x < 10: "
  read num
  if [ “$num” -lt 10 ];        then
     if [ “$num” -gt 1 ]; then
         echo “$num*$num=$(($num*$num))”
       else
         echo “Wrong insertion !”
       fi
  else
       echo “Wrong insertion !”
  fi
  ```

Primerjave

- Datotečni operatorji:
  - -d preveri ali je direktorij
  - -f preveri ali je datoteka
  - -e preveri ali datoteka obstaja?
  - -r preveri ali je dovoljeno branje?
  - -s preveri ali je dolžina datoteke večja kot 0
  - -w preveri ali je dovoljeno pisanje?
  - -x preveri ali je dovoljeno izvajanje
- Primeri:
  - [-d fname] (true če je fname directory, sicer false)
  - [-f fname] (true če je fname datoteka, sicer false)
  - [-e fname] (true če fname obstaja, sicer false)
  - [-s fname] (true če je dolžina fname večja od 0, else false)
  - [-r fname] (true če je za fname dovoljeno branje, sicer false)
  - [-w fname] (true če je za fname dovoljeno pisanje, sicer false)
  - [-x fname] (true če je fname dovoljeno izvajati, sicer false)

Primer

- ```
  #!/bin/bash
  if [ -f /etc/fstab ];
  then
      cp /etc/fstab .
      echo “Done.”
  else
      echo “This file does not exist.”
      exit 1
  fi
  ```
- Vaja. Napišite skript, ki sprejme ime datoteke in:
  - Preveri, če datoteka obstaja
  - Če obstaja, naredi kopijo z imenom + **.bak** (če .bak že obstaja, vprašaj ali ga želi uporabnik zamenjati).
  - Rezultat je datoteka .bak.

Izrazi

- Logični operatorji:
  - ! negacija (NOT) logičnega izraza
  - -a logična konjunkcija (AND) dveh logičnih izrazov
  - -o logična disjunkcija (OR) dveh logičnih izrazov
- ```
  #!/bin/bash
  echo -n “Enter a number 1 < x < 10:”
  read num
  if [ “$num” -gt 1 -a “$num” -lt 10 ];
  then
      echo “$num*$num=$(($num*$num))”
  else
      echo “Wrong insertion !”
  fi
  ```

Izrazi

- Logični operatorji:
  - && logična konjunkcija (AND) dveh logičnih izrazov
  - || logična disjunkcija (OR) dveh logičnih izrazov
- ```
  #!/bin/bash
  echo -n "Enter a number 1 < x < 10: "
  read num
  if [ “$num” -gt 1 ] && [ “$num” -lt 10 ];
  then
      echo “$num*$num=$(($num*$num))”
  else
      echo “Wrong insertion !”
  fi
  ```


  ```

Primer 1

```
$ cat iftrue.sh
#!/bin/bash
echo “Enter a path: ”; read x
if cd $x; then
    echo “I am in $x and it contains”; ls
else
  echo “The directory $x does not exist”;
  exit 1
fi
```

```
$ iftrue.sh
Enter a path: /home
I am in /home and it contains
janez franc …
```

```
$ iftrue.sh
Enter a path: miha
The directory miha does not exist
```

Parametri

- Pozicijski parametri dobijo vrednostni argumentov skripta, ko je bil ta pognan. Pozicijski parameter “N” lahko naslovimo z “${N}”, ali “$N” če je “N” samo ena števk 0-9.
- Posebni parametri
  - $# je število vseh parametrov
  - $0 je ime skripta
  - $* vsi parametri skupaj
  - $@ vrne niz besed; parametrov…

```
$ cat sparameters.sh
#!/bin/bash
echo “$#; $0; $1; $2; $*; $@”
$ sparameters.sh ena dve
2; ./sparameters.sh; ena; dve; ena dve; ena dve
```

Trash (koš)

```
$ cat trash.sh
#!/bin/bash
if [ $# -eq 1 ];
then
    if [ ! –d “$HOME/trash” ];
    then
      mkdir “$HOME/trash”
    fi
    mv $1 “$HOME/trash”
else
  echo “Use: $0 filename”
  exit 1
fi
```

Stavek Case

- Uporaben namesto večkratnega if stavka. Primer uporabe je izvajanje različnih blokov stavkov glede na neko vrednost.
  - Vrednost, ki se uporabi, je lahko izraz
  - Blok stavkov se končuje z dvojnim podpičjem;
  - *) blok se izvrši, če se ne izvrši noben drug (default)

```
case $var in
val1)
    statements;;
val2)
    statements;;
*)
    statements;;
esac
```

Primer (case.sh)

```
#!/bin/bash
echo -n “Enter a number 1 < x < 10: ”
read x
case $x in
    2) echo “Value of x is 2.”;;
    3) echo “Value of x is 3.”;;
    4) echo “Value of x is 4.”;;
    5) echo “Value of x is 5.”;;
    6) echo “Value of x is 6.”;;
    7) echo “Value of x is 7.”;;
    8) echo “Value of x is 8.”;;
    9) echo “Value of x is 9.”;;
    1 | 10) echo “wrong number.”;;
    *) echo “Unrecognized value.”;;
esac
```

Zanke (iteracije)

- Stavek **for** lahko za iteracijo skozi seznam vrednosti spremenljivke.
-
```
for var in list
do
  statements
done
```
- Stavki se izvršijo za vse vrednosti var s seznama.
  -
```
#!/bin/bash
let sum=0
for num in 1 2 3 4 5
do
  let “sum = $sum + $num”
done
echo $sum
```

Zanke (iteracije)

-
```
#!/bin/bash
for x in paper pencil pen; do
  echo “The value of variable x is: $x”
  sleep 1
done
```
- Če seznam vrednosti izpustimo, dobi var po vrsti vrednosti pozicijskih parametrov ($1, $2, $3,…)
  -
```
$ cat for1.sh
#!/bin/bash
for x
do
  echo “The value of variable x is: $x”
  sleep 1
done
$ for1.sh ena dve
The value of variable x is: ena
The value of variable x is: dve
```

Primer 1 (old.sh)

```
#!/bin/bash
# Move the command line arg files to old directory.
if [ $# -eq 0 ] #check for command line arguments
then
  echo “Usage: $0 file …”
  exit 1
fi
if [ ! –d “$HOME/old” ]
then
  mkdir “$HOME/old”
fi
echo The following files will be saved in the old directory:
echo $*
for p in $* #loop through all command line arguments
do
  mv $p “$HOME/old/”
done
ls -l “$HOME/old”
```

Primer 2 (args.sh)

```
#!/bin/bash
# Invoke this script with several arguments: “one two three“
if [ ! -n “$1” ]; then
   echo “Usage: $0 arg1 arg2 ..." ; exit 1
fi
echo ; index=1 ;
echo “Listing args with \”\$*\”:”
for arg in “$*” ; do
   echo “Arg $index = $arg”
   let “index+=1” # increase variable index by one
done
echo “Entire arg list seen as single word.”
echo ; index=1 ;
echo “Listing args with \”\$@\”:”
for arg in “$@” ; do
   echo “Arg $index = $arg”
   let “index+=1”
done
echo “Arg list seen as separate words.” ; exit 0
```

Uporaba polj v zankah

- Lupina bash omogoča uporabo polj. Uporaba je možna na dva načina:
  - `pet[0]=dog`
    `pet[1]=cat`
    `pet[2]=fish`
  - `pet=(dog cat fish)`
- Največje število elementov je 1024. Do posameznih elementov pridemo z uporabo `${arrayname[i]}`
  - `$ echo ${pet[0]}`
    `dog`
- Do vseh elementov pa z uporabo *:
  `echo ${arraynames[*]}`
- Na ta način lahko polja kombiniramo s **for** zanko:
  - `for x in ${arrayname[*]}`
    `do`
    `...`
    `done`

‘C-jevska’ for zanka

- Alternativna oblika zanke **for** je
  - `for (( EXPR1 ; EXPR2 ; EXPR3 ))`
    `do`
    `statements`
    `done`
- Najprej se ovrednoti izraz EXPR1. EXPR2 se vrednoti v vsakem obhodu zanke, dokler izraz ne postane neresničen (false). Vsakič ko je EXPR2 resničen (true), se izvršijo stavki in se ovrednoti (izračuna) EXPR3.
- `$ cat for2.sh`

```
#!/bin/bash
echo –n “Enter a number: ”; read x
let sum=0
for (( i=1 ; $i<$x ; i=$i+1 )) ; do
  let “sum = $sum + $i”
done
echo “the sum of the first $x numbers is: $sum”
```

Razhroščevanje

- V bash sta na razpolago dve možnosti, s katerima pridemo do uporabnih podatkov za razhroščevanje
  - **-x** : prikaže vsako vrstico, z ovrednotenimi spremenljivki, tik preden se vrstica izvede
  - **-v** : prikaže vsako vrstico (tako kot je napisana), tik preden se vrstica izvede
- Uporaba: #!/bin/bash –v ali #!/bin/bash –x ali #!/bin/bash –xv
  - `$ cat for3.sh`

```
#!/bin/bash –x
echo –n “Enter a number: ”; read x
let sum=0
for (( i=0 ; $i<=$x ; i=$i+1 )) ; do
  let “sum = $sum + $i”
done
echo “the sum of the first $x numbers is: $sum”
```

Razhroščevanje

```
$ for3.sh
+ echo –n ‘Enter a number: ’
Enter a number: + read x
3
+ let sum=0
+ (( i=0 ))
+ (( 0<=3 ))
+ let ‘sum = 0 + 0’
+ (( i=0+1 ))
+ (( 1<=3 ))
+ let ‘sum = 0 + 1’
+ (( i=1+1 ))
+ (( 2<=3 ))
+ let ‘sum = 1 + 2’
+ (( i=2+1 ))
+ (( 3<=3 ))
+ let ‘sum = 3 + 3’
+ (( i=3+1 ))
+ (( 4<=3 ))
+ echo ‘the sum of the first 3 numbers is: 6’
the sum of the first 3 numbers is: 6
```

Stavek While

- Stavek while je namenjen izvajanju bloka ukazov dokler je določen pogoj resničen (true). Zanka se neha izvajati takoj, ko pogoj postane neresničen (false).
- while *expression*
  do
  statements
  done
- $ cat while.sh

```
#!/bin/bash
echo –n “Enter a number: ”; read x
let sum=0; let i=1
while [ $i –le $x ]; do
  let “sum = $sum + $i”
  ((i=$i+1))
done
echo “the sum of the first $x numbers is: $sum”
```

Meni

-

```
#!/bin/bash
clear ; loop=y
while [ “$loop” = y ] ;    do
  echo “Menu”;  echo “====”
  echo “D: print the date”
  echo “W: print the users who are currently logged on.”
  echo “P: print the working directory”
  echo “Q: quit.”
  echo
  read –s choice
  case $choice in
    D | d) date ;;
    W | w) who ;;
    P | p) pwd ;;
    Q | q) loop=n ;;
    *) echo “Illegal choice.” ;;
  esac
  echo
done
```

Išči vzorec in odpri z vi

```
$ cat grep_edit.sh
#!/bin/bash
# Edit argument files $2 ..., that contain pattern $1
if [ $# -le 1 ]
then
  echo “Usage: $0 pattern file …” ; exit 1
else
  pattern=$1 # Save original $1
  shift  # shift the positional parameter to the left by 1
  while [ $# -gt 0 ] # New $1 is first filename
  do
    grep “$pattern” $1 > /dev/null
    if [ $? -eq 0 ] ; then # If grep found pattern
      vi $1 # then vi the file
    fi
    shift
  done
fi
$ grep_edit.sh while *.sh
```

Continue

- Ukaz `continue` povzroči skok na naslednjo iteracijo zanke, ostali ukazi v trenutni iteraciji se preskočijo.

```
#!/bin/bash
LIMIT=19
   echo
   echo "Printing Numbers 1 through 20 (but not 3 and
   11)"
a=0
while [ $a -le "$LIMIT" ]; do
  a=$(($a+1))
  if [ "$a" -eq 3 ] || [ "$a" -eq 11 ]
  then
   continue
  fi
  echo -n "$a "
done
```


Break

- Ukaz `break` prekine izvajanje zanke, skok ven.

```
#!/bin/bash
LIMIT=19
   echo
   echo "Printing Numbers 1 through 20, but something
   happens after 2 … "
a=0
while [ $a -le "$LIMIT" ]; do
  a=$(($a+1))
  if [ "$a" -gt 2 ]
  then
    break
  fi
  echo -n "$a "
done
echo; echo; echo
exit 0
```


Zanka Until

- Struktura until je zelo podobna strukturi while. Zanka se izvaja, dokler pogoj ne postane resničen.
- `until [expression]`
  `do`
  `  statements`
  `done`
- `$ cat countdown.sh`

```
#!/bin/bash
echo "Enter a number: "; read x
echo ; echo Count Down
until [ "$x" -le 0 ]; do
  echo $x
  x=$(($x -1))
  sleep 1
done
echo ; echo GO !
```


Delo z nizi

- Na področju dela z nizi ponuja bash precej možnosti:
  - `${#string}` vrne dolžino niza
  - `${string:position}` vrne podniz `$string` od mesta `$position`
  - `${string:position:length}` vrne podniz dolžine `$length` znakov v nizu `$string` od mest `$position`
- Primer

```
$ st=0123456789
      $ echo ${#st}
      10
      $ echo ${st:6}
      6789
      $ echo ${st:6:2}
      67
```


Substitucija parametrov

- Manipulacije / razširitve spremenljivk
  - `${parameter-default},` Če parameter nima vrednosti, uporabi default.
    - `$ echo ${username-`whoami`}`
      `janez`
      `$ username=simon`
      `$ echo ${username-`whoami`}`
      `simon`
  - `${parameter=default},` Če parameter nima vrednosti, vrednost postavi na default.
    - `$ echo ${username=`whoami`}`
      `$ echo $username`
      `janez`
  - `${parameter+value},` Če parameter ima vrednost, uporabi `value`, sicer pa prazen niz.
    - `$ echo ${username+franc}`
      `franc`

Substitucija parametrov

- `${parameter?msg},` Če parameter ima vrednost, jo uporabi, sicer izpiši `msg`
  - `$ value=${total?'total is not set'}`
    `total: total is not set`
    `$ value=${total?'total is not set'}`
    `$ echo $value`
    `10`

Primer

```
#!/bin/bash
    directory=${1-`pwd`}
    find $directory * -type d –maxdepth 0
    exit 0
```

Funkcije

- Skripti lahko vsebujejo funkcije. Tako lahko progra razbijemo v manjše dele, skript pa je lažje berljiv.

```
#!/bin/bash
hello()
{
echo “You are in function hello()”
}

echo “Calling function hello()…”
hello
echo “You are now out of function hello()”
```

- Funkcijo smo klicali po imenu `hello()` v vrstici: `hello` . Ko se ta vrstica izvede, bash poišče vrstico z `hello()`. Najde jo na vrhu in izvede njene ukaze….

Funkcije

```
#!/bin/bash
function check() {
if [ -e "/home/$1" ]
then
  return 0
else
  return 1
fi
}

echo “Enter the name of the file: ” ; read x
if check $x
then
  echo “$x exists !”
else
  echo “$x does not exist !”
fi.
```

Skript 1: Izbor naključne karte (RANDOM)

```
#!/bin/bash
# Count how many elements.

Suites="Clubs Diamonds Hearts Spades"
Denominations="2 3 4 5 6 7 8 9 10 Jack Queen King Ace"

# Read into array variable.
suite=($Suites)
denomination=($Denominations)

# Count how many elements.
num_suites=${#suite[*]} #array length
num_denominations=${#denomination[*]}
echo -n "${denomination[$((RANDOM%num_denominations))]} of "
echo ${suite[$((RANDOM%num_suites))]}
exit 0
```

Skript 2: Spremeni vsa imena datotek tako, da uporabljajo samo male črke (tr)

```
#!/bin/bash
for filename in *
# Traverse all files in directory.
do
  fname=`basename $filename`
  # Change name to lowercase.
  n=`echo $fname | tr A-Z a-z`
  if [ "$fname" != "$n" ]
  # Rename only files not already lowercase.
  then
    mv $fname $n
  fi
done
exit 0
```

Skript 3: Primerjava datotek (cmp)

```
#!/bin/bash
ARGS=2 # Two args to script expected.
if [ $# -ne "$ARGS" ]; then
  echo "Usage: `basename $0` file1 file2" ; exit 1
fi
if [[ ! -r "$1" || ! -r "$2" ]] ;  then
  echo "Both files must exist and be readable." ; exit 2
fi
cmp $1 $2 &> /dev/null
# /dev/null buries the output of the "cmp" command.
# Also works with 'diff', i.e., diff $1 $2 &> /dev/null
if [ $? -eq 0 ] # Test exit status of "cmp" command.
then
  echo "File \"$1\" is identical to file \"$2\"."
else
  echo "File \"$1\" differs from file \"$2\"."
fi
exit 0
```

Procesi

- Koncept procesa
- Operacije nad procesi
- Sočasni, sodelujoči procesi
- Medprocesna komunikacija

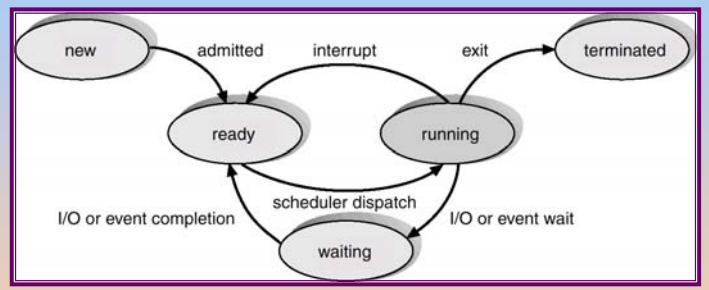
Koncept procesa

- Proces – program v izvajanju; izvajanje procesa je zaporedno.
- Aktivnosti, povezane s procesi (OS):
  - Ustvarjanje (kreiranje) / uničevanje uporabniških in sistemskih procesov
  - Porazdeljevanje procesov
  - Mehanizmi za sinhronizacijo, komunikacijo, zaščito…
- Proces ni le program, sestavljen je iz pasivnega in aktivnega dela:
  - **Pasivni:** programska koda
  - **Aktivni**
    - Programski števec, registri
    - sklad
    - Podatkovni del (data segment)

Stanja procesa

- Proces prehaja med različnimi stanji
  - **nov** (new), če se je ravnokar pričel (ustvarjen - created)
  - **teče** (running), če se ravnokar izvajajo njegovi ukazi
  - **čaka** (waiting), če čaka na nek dogodek (npr. konec V/I aktivnosti, signal), da bi lahko spet stekel. Pravimo tudi, da spi oz. da je blokiran (blocked, sleeping).
  - **pripravljen** (ready), če mu manjka le še procesor.
  - **končan** (terminated), če je zaključil izvajanje programa.

Stanja procesa



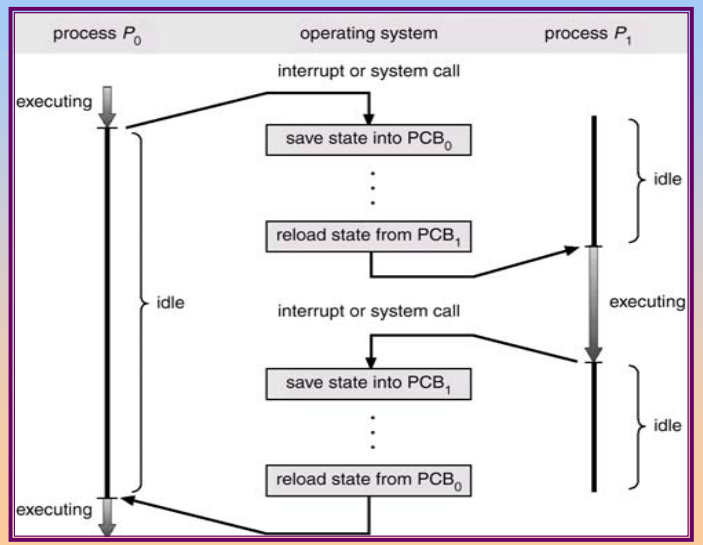
Kontrolni blok procesa

Process Control Block (PCB)

Blok vsebuje vrsto podatkov o procesu:

- Stanje procesa
- Programski števec
- Vsebina registrov
- Podatki za upravljanje s pomnilnikom
- Podatki za dodeljevanje CPE; prioriteta
- 'Računovodski' podatki
- Stanje V/I naprav

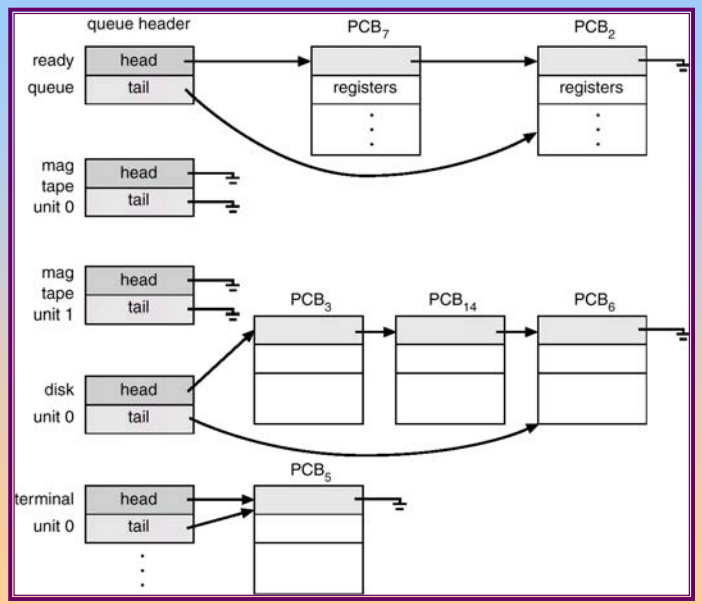
CPE preklaplja med procesi (multiprogramiranje, dodeljevanje časa)



Vrste (Queues)

- Vrsta vseh procesov – vsi trenutno obstoječi procesi.
- Vrsta pripravljenih (Ready queue) – procesi v delovnem pomnilniku, ki so pripravljeni na izvajaje.
- Vrste za V/I naprave (Device queues) – procesi, ki čakajo na določeno V/I napravo.
- Procesi se selijo med različnimi vrstami.

Vrsta pripravljenih in vrste za nekaj V/I naprav



Menjava okolja (context)

- Ko CPE preklopi iz enega procesa na drugega, mora OS shraniti stanje starega procesa v njegov PCB in naložiti shranjeno stanje za nov proces.
- Čas, ki je potreben za zamenjavo okolja, je izgubljen; sistem med preklapljanjem ne dela koristnega dela.
- Čas za preklop je odvisen od podpore strojne opreme.

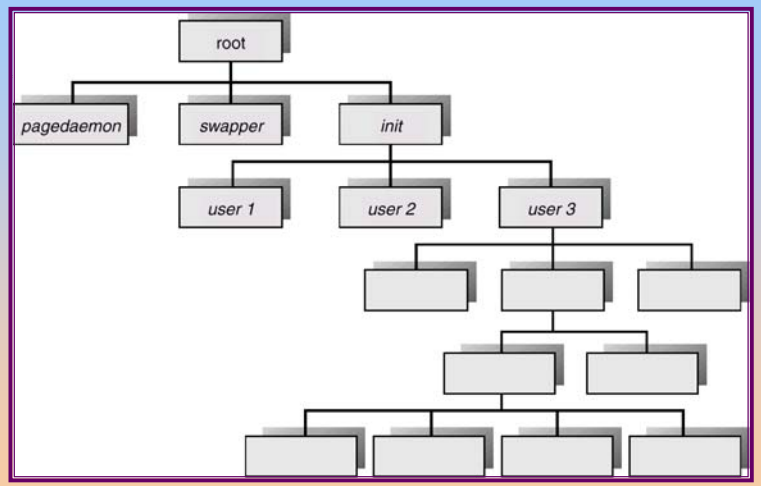
Ustvarjanje procesa (creation)

- **Kako nek proces ustvari nek nov proces?**
  - Izvede nek sistemski klic, npr. v UNIX-u je to fork().
- **Kako dobi proces vire (resource), kot so procesorski čas, pomnilnik, datoteke, V/I naprave,...?**
  - neposredno od OS
  - od očeta
- **Kako sin dobi vhodne podatke?**
  - od očeta
- **Kaj se zgodi z očetom, ko je sin ustvarjen?**
  - nadaljuje izvajanje
  - oče se ustavi (zaspi) in čaka, da se sin konča.
- **Kakšen je novoustvarjeni sin?**
  - sin je sprva »kopija« očeta, kasneje pa morda poskrbi za novo kodo (UNIX) ali
  - sin že takoj vsebuje svojo programsko kodo

Primer - UNIX

```
#include <stdio.h>
main()
{
    int pid, status, sinPID;
    ...
    pid=fork();
    if (pid!=0)
    {
        printf("Jaz sem oče št. %d; moj oče je %d", getpid(), getppid());
        sinPID=wait(&status);
        printf("sin številka %d se je končal.", sinPID);
    }
    else /* (B) */
    {
        printf("Jaz sem sin številka %d, moj oče je %d", getpid(), getppid());
        ...
        exit(status);
    }
}
```

UNIX – drevo procesov



Uničenje procesa (termination)

- **Proces konča samega sebe** (**exit**).
  - Očetu, ki čaka (**wait**), pošlje signal (SIGCHILD), in čaka, da ga bo oče sprejel.
  - Vrnejo se viri, ki si jih je proces prilastil.
- Oče lahko zahteva uničenje sina (**abort**).
  - Sin je prekoračil dodeljene vire.
  - Sin ni več potreben.
  - Oče namerava končati.
    - Večina OS ne dovoli, da bi sinovi obstajali, če se oče konča.
    - Verižno (kaskadno) uničenje.

Sodelujoči procesi

- Proces je *neodvisen* takrat, kadar ne more vplivati na izvajanje drugih procesov v sistemu in, kadar drugi procesi ne morejo vplivati nanj.
- *Sodelujoči* procesi lahko vplivajo na druge procese in tudi drugi procesi lahko vplivajo nanje
- Potrebni so mehanizmi za:
  - Komunikacijo med sodelujočimi procesi
  - Sinhronizacijo (uskladitev) njihovega delovanja
  - Zaščita
- Primer: proizvajalec - porabnik

Proizvajalec - porabnik

- **Procesa imata skupne spremenljivke:**

```
var n ...................../* velikost vmesnika */
type item = ...../* tip podatkov */
var buffer = array[0...n-1] of item;
in, out: 0...n-1;
```

- **Proizvajalec:**

```
repeat
        .........
        nextp:= proizvedi_naslednjega;
        .........
        while (in+1)mod n=out (poln) do no_op;
        buffer[in]:=nextp;
        in:=(in+1)mod n;
until false;
```

- **Porabnik:**

```
repeat
        .........
        while in=out (prazen) do no_op;
        nextc:=buffer[out];
        out:=(out+1)mod n;
until false;
```


Medprocesna kominukacija (IPC)

- IPC je mehanizem za komunikacijo in sinhronizacijo med procesi..
- realiziran je s sistemom prenašanja sporočil (message passing); procesom se ni treba opirati na skupne spremenljivke.
- IPC nudi vsaj dve ključni operaciji:
  - **send**(*message*) – velikost sporočila je lahko fiksna ali ne
  - **receive**(*message*)
- Če želita *P* in *Q* komunicirati, morata:
  - Med seboj vzpostaviti *povezavo - link*
  - Izmenjati podatke s *send/receive*
- Povezava je lahko implementirana
  - fizično (skupni pomnilnik, vodila, omrežje…)
  - logično (logične lastnosti)

Vprašanja glede implementacije povezave:

- Kako se povezava vzpostavi?
- Ali lahko povezava služi več kot dvema procesoma hkrati?
- Koliko povezav je lahko med vsakim parom procesov, ki komunicirata ?
- Ali je povezava enosmerna ali dvosmerna?
- Ali je povezava neposredna ali posredna (prek skupnega medija - nabiralnik)?
- Ali je sporočilo fiksne (lažja implementacija *send* in *receive*) ali spremenljive dolžine (lažje za uporabniško programiranje)?
- Ali se pošilja kopija sporočila ali njegovega naslova (referenca)?

Neposredna komunikacija

- Procesa morata drug drugega (simetrična) ali samo pošiljatelj sprejemnika (asimetrična) eksplicitno poimenovati:
  - **send** (*P, message*) – pošlji poročilo procesu P
  - **receive**(*Q, message*) – sprejmi sporočilo od procesa Q
- Lastnosti povezave
  - povezava je vzpostavljena avtomatsko
  - povezava je vzpostavljena med natanko enim parom procesov
  - povezava med dvema procesoma je natanko ena
  - povezava je lahko enosmerna, vendar je običajno dvosmerna.

Posredna komunikacija

- Sporočila se pošiljajo in sprejemajo preko *nabiralnikov* (mailbox).
  - send (A, message) – pošlji sporočilo nabiralniku A.
  - receive(A, message) – sprejmi sporočilo od nabiralnika A.
  - Procesa lahko komunicirata, če si delita nabiralnik.
- Lastnosti povezave
  - povezava je lahko vzpostavljena le, če si procesa delita nabiralnik
  - povezava je lahko vzpostavljena med več procesi (tekma za sporočila).
  - Vsak par procesov lahko ima več povezav.
  - Povezava je lahko enosmerna ali dvosmerna.

Tekma za sporočila

- Proces P pošlje sporočilo v nabiralnik A, medtem ko R in Q oba izvedeta operacijo **`receive(A,m)`**
  - Q začne z `receive` in prebere sporočilo iz A, vendar ga od tam še ne odstrani. Preden ga odstrani, je prekinjen – CPE dobi R.
  - tudi R izvede `receive` in dobi sporočilo – oba imata sporočilo, NAPAKA!!
- Rešitve:
- sistem izbere proces, kateremu bo poslal sporočilo
  - dovoli začetek obeh `receive` operacij, pravilno se zaključi le ena
  - Pošiljatelju (lahko) sporoči, komu je sporočilo dodelil
- dovolimo povezavo med največ dvema procesoma (na to mora paziti uporabnik – slabost: omejevanje uporabnika)
- dovolimo največ eno izvajanje operacije `receive` naenkrat (kritična sekcija, obstaja več rešitev)

Sinhronizacija

- Pošiljanje sporočil je lahko sinhrono ali asinhrono.
- **sinhrono** (blocking)
- **asinhrono** (non-blocking)
- **send**
  - Sinhron, če je pošiljatelj blokiran, dokler sporočilo ne prispe do sprejemnika/nabiralnika
- **receive**
  - Sinhron, če je sprejemnik blokiran, dokler sporočilo ne prispe do njega/v njegov nabiralnik
- Zmenek: pošiljatelj in sprejemnik sta sinhrona

Lastništvo nad nabiralniki

- Lastnik nabiralnika je proces:
  - Nabiralnik je del procesa, je pritrjen na proces (del naslovnega prostora)
  - Lastnik nabiralnika je proces, ki nabiralnik kreira
  - Lastnik lahko sprejema sporočila iz svojega nabiralnika
  - Ostali procesi lahko pošiljajo sporočila v nabiralnik
  - Ko se lastnik nabiralnika konča, nabiralnik izgine
- Lastnik nabiralnika je OS:
  - tak nabiralnik je neodvisen
  - OS omogoča kreiranje, pošiljanje, sprejemanje, uničenje…
  - Proces, ki nabiralnik kreira, je avtomatično njegov lastnik. Lastnik lahko lastništvo dodeli še drugim procesom…

Kopičenje sporočil

- Povezava vsebuje vrsto, ki začasno shrani poslano sporočilo. Glede na kapaciteto vrste razlikujemo povezave:
  - brez kopičenja (zero capacity, no buffering): povezava ne more shraniti nobenega sporočila. Če taka povezava ni prosta (Q še ni prejel prejšnjega sporočila), se P blokira, preden pošlje (*send* mora preverjati prostost povezave; lahko je tudi sinhron);
  - z omejenim kopičenjem (bounded capacity): povezava lahko shrani *n* sporočil (n je končen). Če jih je <n, lahko P pošilja asinhrono, sicer se mora blokirati, dokler se ne sprosti prostor. Kritična sekcija;
  - z neomejenim kopičenjem (unbounded capacity): vrsta ima potencialno neomejeno kapaciteto (virtualno neskončno), OS po potrebi podaljša sicer končno vrsto – *send* je lahko vedno asinhron.

Pošiljanje kopij ali referenc?

- prenos kopije sporočila
  - OS kopira sporočilo v vrsto, ki je v naslovnem prostoru sistema.
  - Ko Q zahteva sporočilo, bo OS prekopiral celo sporočilo iz vrste v naslovni proctor Q.
  - Slabost: sporočilo se kopira dvakrat.
- prenos naslova sporočila
  - OS kopira v vrsto naslov sporočila.
  - Ko bo Q zahteval sporočilo, bo OS prekopiral celo sporočilo m iz naslovnega prostora P v naslovni prostor Q.
  - sporočilo se kopira samo enkrat
  - NEVARNOST: sporočila ne smemo spreminjati do konca *receive*. Če je *send* asinhron, mora P to upoštevati. Če se P konča pred *receive*, mora OS poskrbeti za ohranitev sporočila.

Sinhronizacija proceov

- Ozadje, motivacija
- Problem kritičnih sekcij
- Strojna sinhronizacija
- Semaforji

Ozadje, motivacija

- Sodelujoči procesi se med seboj lahko ovirajo, saj si direktno delijo logični naslovni prostor (kodo in podatke).
- Za ohranjanje konsistentnosti podatkov potrebujemo mehanizme za sinhronizacijo procesov.
- Rešitev s skupnim pomnilnikom za problem skupnega vmesnika dopušča, da je v vmesniku kvečjemu $n - 1$ elementov. Rešitev, ki omogoča izrabo vseh N mest, ni preprosta.
  - Ena možnost je, da dodamo celoštevilsko spremenljivko (števec) **counter**, ki bo predstavljala trenutno število zasedenih komponent v vmesniku.

Skupni vmesnik

- Skupni podatki

```
#define BUFFER_SIZE 10
typedef struct {
   . . .
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
int counter = 0;
```

Skupni vmesnik

- Proizvajalec

```
item nextProduced;

while (1) {
      while (counter == BUFFER_SIZE)
            ; /* do nothing */
      buffer[in] = nextProduced;
      in = (in + 1) % BUFFER_SIZE;
      counter++;
}
```

Skupni vmesnik

- Porabnik

```
item nextConsumed;

while (1) {
      while (counter == 0)
            ; /* do nothing */
      nextConsumed = buffer[out];
      out = (out + 1) % BUFFER_SIZE;
      counter--;
}
```

Skupni vmesnik

- Stavka

  counter++;
  counter--;

  se morata izvesti *atomarno*.

- Atomarna operacija se mora izvesti v celoti, med njo ne sme priti do prekinitve oziroma menjave okolja.

Skupni vmesnik

- Stavek "**count++**" je na strojnem nivoju lahko implementiran kot:

  register1 = counter
  register1 = register1 + 1
  counter = register1

- Stavek "**count--**" pa kot:

  register2 = counter
  register2 = register2 – 1
  counter = register2

Skupni vmesnik

- Če proizvajalec in porabnik z vmesnikom delata sočasno, se lahko zgodi, da se strojne operacije prepletejo.
- Prepletanje je odvisno od razvrščevalnika in trenutka, ko pride do menjave okolja.

primer 1:

T0: a1 .....R1=5
T1: a2 .....R1=6
..
T2: a3 .....counter=6
.....(menjava okolja)
T3: b1 .....R2=6
..
T4: b2 .....R2=5

T5 :b3 .....counter=5
---
rezultat (pravilen): count=5

Skupni vmesnik

primer 2:

a1 .....R1=5
a2 .....R1=6
.....(menjava okolja)
b1 .....R2=5

b2 .....R2=4
.....(menjava okolja)
a3 .....counter=6
.....(menjava okolja
b3 .....counter=4
---
rezultat (nepravilen): 4

primer 3:

a1 .....R1=5
a2 .....R1=6
.....(menjava okolja)
b1 .....R2=5

b2 .....R2=4

b3 .....counter=4
.....(menjava okolja)
a3 .....counter=6
---
rezultat (nepravilen): 6

Tekma (Race Condition)

- **Tekma**: situacija, ko več procesov sočasno spreminja iste podatke in je rezultat izvajanja odvisen od določenega vrstnega reda, po katerem procesi dostopajo do podatkov, je neke vrste tekma (race condition) in ni zaželena.
- Da bi se izognili tej situaciji, je potrebno zagotoviti, da le en proces naenkrat spreminja spremenljivko *counter*. Če se a in b ne prepletata, je rezultat pravilen. Zato moramo zagotoviti, da se bosta izvajala kvečjemu a oz. b; dokler ne konča eden, se drugi ne sme zagnati. Pravimo, da ukazi a oz. b tvorijo KRITIČNO SEKCIJO v svojem programu.

Problem kritičnih sekcij

- Danih je *n* procesov: P_0, P_1, ... ,P_{n-1}
- Vsak od procesov ima del kode, imenovan **kritična sekcija** (critical section), v kateri lahko proces spreminja skupne spremenljivke, nadgrajuje tabele, piše datoteko.
- Pomembna stvar takega sistema je, ko en proces izvaja svojo kritično sekcijo, ne sme noben drug proces izvajati svoje. Torej je izvajanje kritičnih sekcij procesov časovno vzajemno izključujoče (mutual exclusion).
- Problem kritičnih sekcij je izdelati protokol, ki ga procesi lahko uporabljajo pri sodelovanju. Vsak proces mora zaprositi za dovoljenje za vstop v svojo kritično sekcijo. Del kode, ki implementira to prošnjo, se imenuje **vstopna sekcija** (entry section). Za kritično sekcijo se nadaljuje **izstopna sekcija**(exit section).

Rešitev problema kritične sekcije

- **Vzajemno izključevanje**: če proces izvaja svojo kritično sekcijo, noben drug proces ne more izvajati svoje.
- **Napredovanje**: če noben proces ne izvaja svoje kritične sekcije in obstaja nekaj procesov, ki želijo vstopiti v svojo kritično sekcijo, potem le ti procesi lahko odločijo, kateri bo vstopil v svojo kritično sekcijo kot naslednji.
- **Omejeno čakanje**: proces mora v končnem času vstopiti v svojo kritično sekcijo.
  - Predpostavimo, da se osnovni ukazi strojnega jezika (load, store, test, add, ...) izvedejo atomarno (nedeljivo) – morajo se izvesti do konca, preden se upošteva želena prekinitev.
  - Poleg tega pa mora biti rešitev neodvisna od vrednosti n (se pravi števila procesov) in tehnoloških značilnosti računanja.

Kako rešiti problem?

- Za začetek se omejimo le na 2 procesa, P_0 in P_1
- Splošna struktura procesa P_i:

```
do {
    vstopna sekcija
        kritična sekcija
    izstopna sekcija
        ....
} while (1);
```

- Za sinhronizacijo procesa lahko uporabljata skupne spremenljivke.

Algoritem 1

- Skupne spremenljivke:
  - **int dovoljenje**;
    na začetku **dovoljenje = 0**
  - **dovoljenje = i** $\Rightarrow P_i$ lahko vstopi v kritično sekcijo (i=0,1)
- Proces P_i

```
do {
    while (dovoljenje!= i) ;
        kritična sekcija
    dovoljenje = j;
        ….
} while (1);
```

- Ta rešitev zagotavlja vzajemno izključevanje, ne zagotavlja pa vedno napredovanja. Npr., ko Pi zapusti svojo kritično sekcijo, da dovoljenje drugemu. Recimo, da drugi tega ne želi več, Pi pa hoče ponovno v svojo kritično sekcijo. Ker je možno le izmenično vstopanje, imamo problem.

Algoritem 2

- Problem algoritma 1 je, da ne vsebuje dovolj informacije o stanju vsakega od procesov. Vsebuje le informacijo o tem, kateri proces lahko vstopi v kritično sekcijo.
  - **boolean zahteva[2]**;
    na začetku **zahteva[0] = zahteva[1] = false.**
  - **zahteva[i] = true** $\Rightarrow P_i$ je pripravljen na vstop v kritično s.
- Struktura procesa P_i

```
do {
    zahteva[i] := true;
    while (zahteva[j]) ;
            kritična sekcija
    zahteva [i] = false;
        ….
} while (1);
```

- Zagotavlja vzajemno izključevanje, vendar ne zagotavlja omejenega čakanja. Npr. P0 postavi zahteva[0]=true (hoče v kritično sekcijo), vendar takoj zatem pride do prekinitve (npr. timer). Procesor dobi P1 in tudi on postavi zahteva[1]:=true, ker hoče v svojo kritično sekcijo.

Algoritem 3

- Ta algoritem je kombinacija idej prejšnjih dveh algoritmov.
- Process P_i

```
do {
    zahteva[i]:= true;
    dovoljenje = j;
    while (zahteva[j] and dovoljenje = j) ;
        kritična sekcija
    zahteva[i] = false;
        ….
} while (1);
```

- Če poskušata oba procesa vstopiti istočasno, bo dovoljenje postavljeno na i in na j skoraj istočasno, vendar se bo ena od vrednosti prepisala z drugo in tako bo le en proces dobil dovoljenje za vstop v svojo kritično sekcijo.
- Ta rešitev je korektna, obstaja formalni dokaz (Silbershatz)

Rešitev za n procesov (Bakery Algorithm)

- Pred vstopom v kritično sekcijo proces dobi številko. Proces z najnižjo številko lahko vstopi v svojo kritično sekcijo naslednji.
- Ta algoritem pa ne zagotavlja, da dva procesa ne dobita iste številke. Rešitev: če procesa P_i in P_j dobita isto številko in velja $i < j$, potem P_i vstopi prvi; sicer vstopi prvi P_j.
- Shema številčenja procesov vedno generira števila v naraščajočem zaporedju; npr., 1,2,3,3,3,3,4,5...

Bakery algoritem

- Notacija: < ≡ leksikografska ureditev parov (vstopnica, št procesa)
  - $(a,b) < (c,d)$ če $a < c$ ali $a = c$ in $b < d$
  - max $(a_0,\ldots, a_{n-1})$ je največje izmed števil $a_0,\ldots, a_{n-1}$
- Skupni podatki

  boolean izbira[n];

  int stevilo[n];

  Podatkovni strukturi na začetku inicializiramo na **false** in zaporedoma

Bakery algoritem

```
do {
   izbira[i] = true;
   stevilo[i] = max(stevilo[0], stevilo[1], …, stevilo [n – 1])+1;
   izbira[i] = false;
   for (j = 0; j < n; j++) {
            while (izbira[j]) ;
            while ((stevilo[j] != 0) && (stevilo[j],j) < (stevilo[i],i)) ;
   }
     kritična sekcija
    stevilo[i] = 0;
      ….
} while (1);
```

Strojna sinhronizacija

- Onemogočanje prekinitev
- Atomarno primerjanje in spreminjanje spremenljivk.

```
boolean TestAndSet(boolean &target) {
   boolean rv = target;
   target = true;

   return rv;
}
```

Vzajemno izključevanje s Test-and-Set

- Skupni podatki:

  boolean cakaj = false;

- Process P_i

```
do {
   while (TestAndSet(cakaj)) ;
      kritična sekcija
   cakaj = false;
      ….
}
```

- **Lahko se zgodi, da Pi čaka, da Pj izstopi iz svoje kritične sekcije, medtem pa pride proces Pk, ki tudi čaka na proces Pj. Ko Pj konča, vstopi v svojo kritično sekcijo Pk in Pi še vedno čaka. Čakanje ni omejeno!**

Semaforji

- semafor S je orodje za sinhronizacijo procesov oz. usklajevanje njihovega dela.
- semafor *S* – je celoštevilčna spremenljivka
- lahko jo (poleg inicializacije) spreminjata le dve standardni **atomarni** operaciji, ki sta nad njo definirani:

wait (*S*):

```
while S≤ 0 do no-op;
  S--;
```

signal (*S*):

```
S++;
```

- ***Ti dve operaciji je uvedel Dijkstra in sta bili originalno označeni z:***
  - ***P (proberen, to test) .....wait***
  - ***V (verhogen, to increment) .....signal***

Problem kritičnih sekcij *n* procesov

- Vsak od procesov P1,...,Pn ima svojo kritično sekcijo. Ti procesi imajo skupen semafor, **mutex** (**mut**ual **ex**clusion), ki je inicializiran na 1.

```
semaphore mutex; // na začetku mutex = 1
```

- Struktura procesa *Pi:*

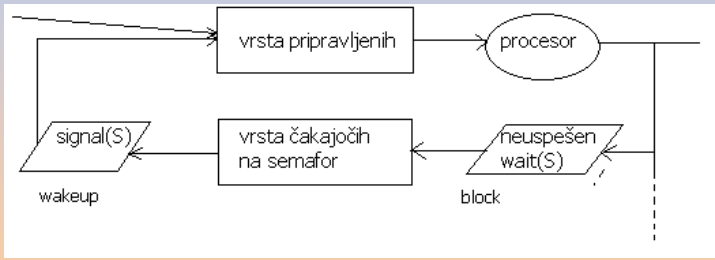
```
do {
   wait(mutex);
      kritična sekcija
   signal(mutex);
      …
} while (1);
```

Učinkovitost semaforjev

- Vse dosedanje metode za reševanje problema kritičnih sekcij uporabljajo metodo nekoristnega čakanja (**busy waiting**).
- Dokler je nek proces v svoji kritični sekciji bodo vsi drugi procesi, ki želijo vstopiti v svojo kritično sekcijo, krožili v vstopni sekciji. Tak tip semaforja se imenuje tudi števni semafor (spinlock).
- Definiramo dve operaciji:
  - **block** uspava proces, ki ga izvede.
  - **wakeup(*P*)** obudi proces **P** (postavi ga v vrsto pripravljenih).

Semaforji z vrsto čakajočih

- Spremenimo definicijo **wait** in **signal** operacij.
- Ko proces izvaja wait operacijo in ugotovi, da je trenutna vrednost semaforja S negativna, naj se proces raje blokira (uspava) in se postavi v vrsto čakajočih na semafor. Ko nek drug proces izvede ukaz **signal**, se blokiran proces spet obudi.



Implementacija

- Za implementacijo take definicije semaforjev lahko definiramo semafor kot zapis:

```
typedef struct {
  int value;
  struct process *L;
} semaphore;
```

- Vsak semafor ima celoštevilsko vrednost (števec) in seznam procesov (vrsta čakajočih procesov na semafor).

  Običajna inicializacija: **S.value=1;**
  S.L=0;

Implementacija

- Operacije semaforjev so lahko sedaj definirane takole:

wait(*S*):

```
S.value--;
if (S.value < 0) {    /* P ne bo smel nadaljevati */
        dodaj ta proces v vrsto S.L;
        block;
}
```

signal(*S*):

```
S.value++;
if (S.value <= 0) {   /* v S.L je vsaj en blokiran proces */
        odstrani nek proces P iz vrste S.L;
        wakeup(P);
}
```

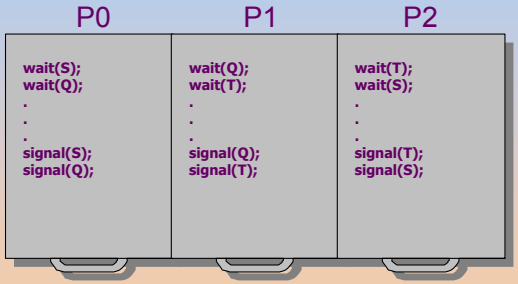
Uporaba semaforjev za sinhronizacijo

- Izvedi operacijo *B* v P_j šele potem, ko se je operacija *A* v procesu P_i že izvedla.
- Uporabimo semafor *flag*, ki je na začetku 0
- Program:

P_i	P_j
⋮	⋮
A	*wait*(*flag*)
signal(*flag*)	*B*

Smrtni objem in stradanje

- **Smrtni objem** – dva ali več procesov čakata v nedogled na dogodek, ki se lahko pojavi le v enem izmed čakajočih procesov.
- Primer:



Recimo, da P0 izvede wait(S), P1 izvede wait(Q) in nato P2 izvede wait(T). Ko P0 izvede wait(Q), mora čakati, dokler P1 ne izvede signal(Q). Podobno mora P1 čakati, ko izvede wait(T), dokler P2 ne izvede signal(T),... Ker operacije signal ne morejo biti izvedene, se procesi P0, P1 in P2 znajdejo v smrtnem objemu.

Smrtni objem in stradanje

- Množica procesov je v smrtnem objemu, če vsak proces iz množice čaka na dogodek, ki ga lahko izvede le nek drug proces iz te množice. Problemi bi nastali, če bi več procesov hkrati uporabljalo skupne vire, npr. :
  wait(x).....zaseži(x)
  signal(x)....sprosti_vir(x)
- Tu bi si nato nek proces lahko prilastil vir in ga ne bi več mogel sprostiti, ker bi se znašel v smrtnem objemu.
- Drug problem, ki je povezan s smrtnimi objemi, je **stradanje** (starvation). To je situacija, kjer proces čaka neskončno dolgo znotraj semaforja. Rešitev je npr. **staranje procesov.**

Vrste semaforjev

- *Binarni* semafor – vrednost 0 ali 1; lažja implementacija.
- *Števni* semafor – celoštevilčna vrednost neomejena.
- Števne semaforje lahko implementiramo z binarnimi.
- S1 (dostop do C) in S2 (semafor za vrsto čakajočih) sta binarna semaforja, inicializirana na 1 in 0
- *wait*

```
wait(S1);
C--;
if (C < 0) {                    // virov je zmanjkalo, postavi v vrsto čak.
                                signal(S1);
                                wait(S2);
}
else signal(S1);
```

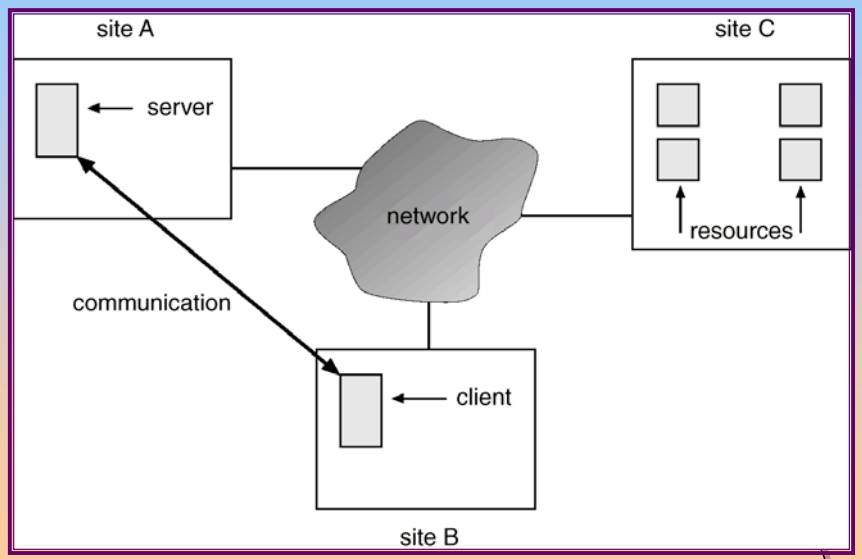
- *signal*

```
wait(S1);
C ++;
if (C <= 0) {                   // ali je kdo v vrsti čakajočih
        signal(S2);
}
signal(S1);
```

Mrežne strukture

- Ozadje
- Tipi računalniških mrež
- Komunikacija
- Komunikacijski protokoli
- Internet
- HTTP, FTP, SMTP, POP
- Varnost, kriptiranje

Porazdeljen sistem



Motivacija

- Skupni (deljeni) viri
  - Skupna raba in tiskanje datotek na oddaljenih računalnikih
  - Obdelava podatkov v porazdeljeni podatkovni bazi
  - Uporaba oddaljenih naprav
- Večja hitrost računanja – *porazdelitev bremena*
- Zanesljivost – izpad računalnika se zazna, ostali delujejo. Sistem (včasih) lahko deluje dalje…
- Komunikacije – pošiljanje sporočil

Mrežni operacijski sistemi

- Uporabniki se zavedajo, da je računalnikov v sistemu več. Dostop virov na oddaljenih računalnikih je mogoč z uporabo:
  - Prijavo (remote login) na določen oddaljen računalnik.
  - Prenos podatkov z oddaljenega na lokalni računalnik z uporabo npr. FTP (File Transfer Protocol).

Porazdeljeni operacijski sistemi

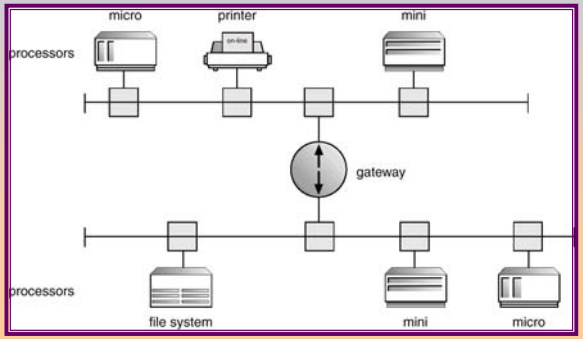
- Uporabniki se ne zavedajo, da je računalnikov več. Dostop do oddaljenih virov je podoben dostopu do lokalnih.
- Migracija podatkov – prenos celotnih datotek ali le delov datotek, ki so potrebni za trenutno opravilo.
- Migracija računanja – včasih je bolj smiselno prenesti samo računanje (obdelavo) na računalnik, na katerem so podatki.

Porazdeljeni operacijski sistemi

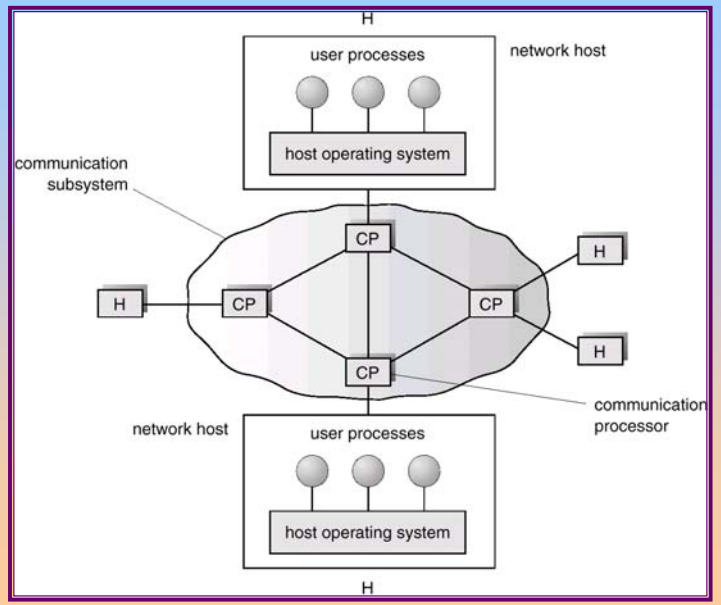
- Migracija procesov – celoten ali del procesa se izvaja na različnih računalnikih.
  - Uravnoteženje bremena – procese porazdelimo tako, da so računalniki enakomerno obremenjeni.
  - Hitrejše računanje –procesi se lahko izvajajo vzporedno na več računalnikih.
  - Glede na strojno opremo – določeni procesi so bolj primerni za določen tip CPE.
  - Prisotnost programske opreme – določena programska oprema je lahko na voljo le na nekaterih računalnikih.
  - Dostop do podatkov – če je količina podatkov velika, je smiselno proces izvesti na računalniku, na katerem podatki so; prenos velike količine podatkov ni smiseln.

Tipi računalniških mrež

- Lokalna (krajevna) mreža LAN (Local-Area Network) – primerna za manjše razdalje.
  - Hitrost ≈ 100 megabitov/sekundo.
  - Prenos je hiter in poceni.
  - Vozlišča:
    - Delovne postaje / osebni računalniki
    - Nekaj večjih / zmogljivejših strežnikov.



WAN (Wide-Area Network). Internet.



Komunikacija

V zvezi s *komunikacijsko* mrežo se pojavijo naslednja vprašanja:

- **Imena, razreševanje imen v naslove**: Kako dva procesa lahko najdeta drug drugega in začneta komunicirati?
- **Strategije usmerjanja (routing)**. Kako sporočila poslati po mreži?
- **Povezovalne strategije**. Kako procesa pošljeta zaporedje večih sporočil?

Imena, razreševanje imen v naslove

- Imenski sistemi v mreži
- Sporočila so naslovljena z identifikacijsko številko procesa (process-id).
- Identifikacija procesa na oddaljenem sistemu s parom

  <host-name, identifier>.

- *Domain name service* (DNS) – določa imensko strukturo in način razreševanja imen (Internet).

Strategije usmerjanja (routing)

- **Fiksno usmerjanje**. Pot med *A* in *B* je določena vnaprej; spremeni se samo ob izpadih.
  - Običajno je to najkrajša pot, zato so stroški komuniciranja relativno nizki.
  - Ne more se prilagajati spremembam v obremenjenosti povezav.
  - Sporočila se vedno sprejmejo v istem zaporedju, kot so poslana.
- **Navidezna zveza**. Pot med *A* in *B* je določena za trajanje ene seje. Pri različnih sejah se lahko pot med istima računalnikoma spremeni.
  - Izboljšano prilagajanjem spremembam v obremenjenosti povezav.
  - Sporočila se vedno sprejmejo v istem zaporedju, kot so poslana.

Strategije usmerjanja (routing)

- **Dinamično usmerjanje**. Pot med *A* in *B* se določi šele, ko je sporočilo poslano.
  - Običajno se uporabi povezava, ki je takrat najmanj obremenjena.
  - Prilagaja se spremembam v obremenjenosti povezav – izogne se pošiljanju po obremenjenih povezavah.
  - Vrstni red sporočil se (lahko) spremeni. Vsakemu sporočilu je potrebno dodati zaporedno številko.

Povezovalne strategije

- **Preklapljanje vodov (circuit switching)**. Zveza se vzpostavi za trajanje celotne komunikacije (telefonsko omrežje).
- **Preklapljanje sporočil (message switching)**. Začasna zveza se vzpostavi za trajanje enega sporočila (pošta).
- **Preklapljanje paketov (packet switching)**. Sporočila se razdeli v pakete enake dolžine. Pošilja se posamezne pakete. Vsak paket lahko potuje po različni poti.

- Preklapljanje vodov zahteva vzpostavitveni čas, vendar se zveza vzpostavlja le enkrat (manj izgubljenega časa). Obdobja z malo prenosa… Drugi dve strategiji zahtevata manjši vzpostavitveni čas, vendar se izgubi več časa, ker se zveza vzpostavi za vsako sporočilo/paket posebej.

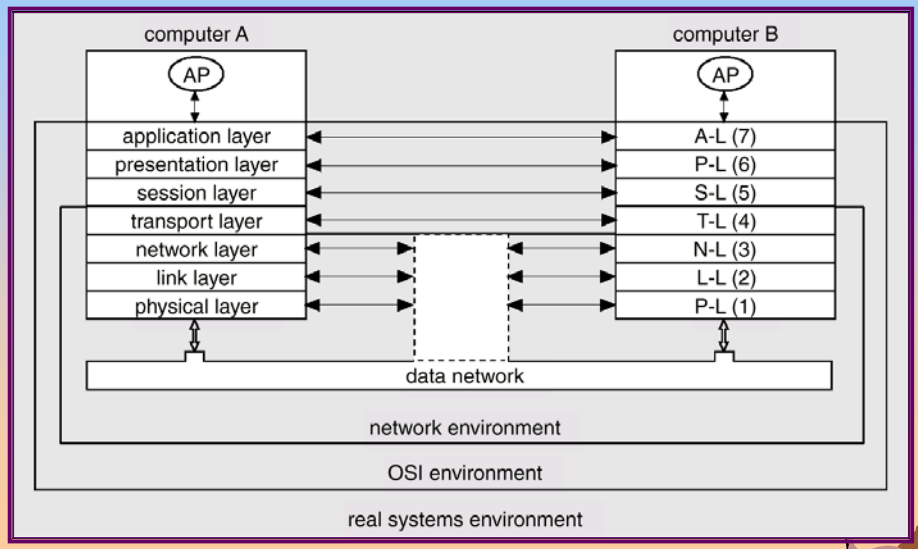
Komunikacijski protokoli

- Fizični sloj – mehanski in električni standardi.
- Povezovalni sloj – deluje nad okvirji *(frames)*, vključuje detekcijo (in korekcijo) napak, do katerih pride v fizičnem sloju.
- Omrežni sloj – zagotavlja povezave in usmerja pakete v komunikacijski mreži. Naslovi odhodnih paketov, dekodiranje naslovov paketov, ki prihajajo, usmerjanje glede na obremenjenost posameznih vodov.

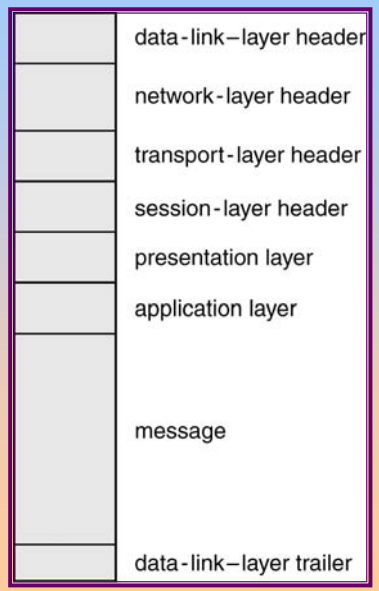
Komunikacijski protokoli

- Prenosni sloj – prenos sporočil, razdelitev na pakete, vrstni red paketov,…
- Sejni sloj – komunikacija na nivoju proces – proces.
- Predstavitveni sloj – premosti razlike med različnimi formati zapisa podatkov…
- Aplikacijski sloj – stik z uporabnikom, prenos datotek FTP, elektronska pošta, *remote login*…

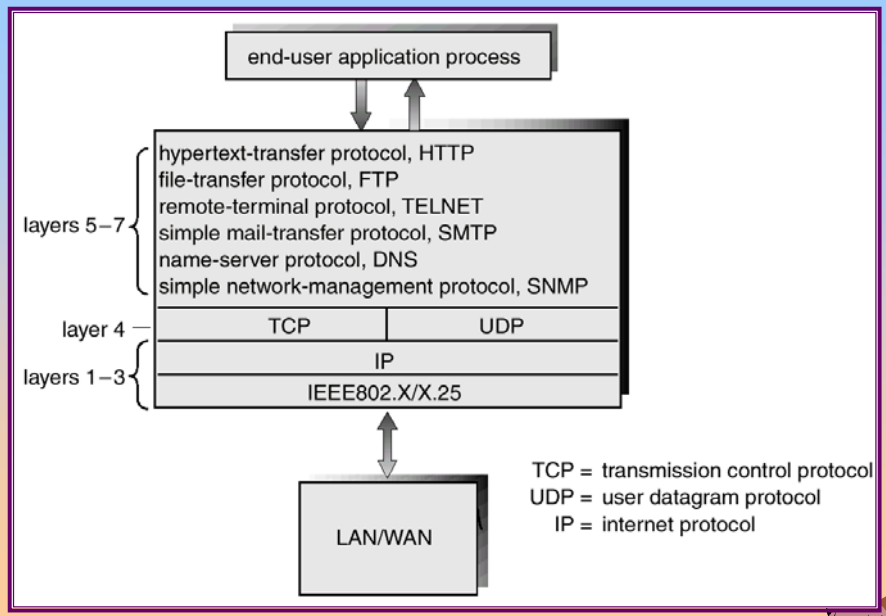
Komunikacija po modelu ISO



Sporočilo (gleda na ISO model)



Internet: TCP/IP



Internet: TCP/IP

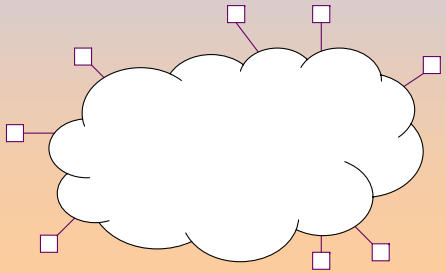
- Na teh protokolih temelji večina internetnih aplikacij – orodja za elektronsko pošto, brskalniki, orodja za prenos datotek...
- Mrežni sloj (IP):
  - Sprejema pakete podatkov z višjih slojev in jih preko mreže pošlje do ciljnega sistema; ta je lahko v drugi mreži.
  - Pri prenosu se lahko paketi pomešajo, ali pa sploh ne pridejo do cilja. Sloji nad mrežnim pakete uredijo po vrsti in ugotovijo, če se je kateri izgubil.
- Prenosni sloj (TCP / UDP):
  - Omogoča aplikacijam, da se povežejo med sabo in izmenjujejo podatke. Zagotavlja mrežne usluge, ki jih koristijo aplikacije.
  - Primer: prenos datagramov brez potrjevanja (UDP) za prenos video ali avdio signala; zanesljiva povezava za prenos elektronske pošte ali datotek (TCP).

Internet: TCP/IP

- Povezovalni in fizični sloj
  - Sloji, ki so pod mrežnim slojem, ne spadajo v domeno TCP/IP; zagotavljati morajo povezavo med računalniki mreži, protokol pa s TCP/IP ni določen.
  - Pogosti so naslednji protokoli: IEEE 802.3 (ethernet), IEEE 802.5 (token ring), SLIP/PPP, …
  - Bistveno je, da nizkonivojski protokol omogoča višjim slojem pošiljanje IP paketov.

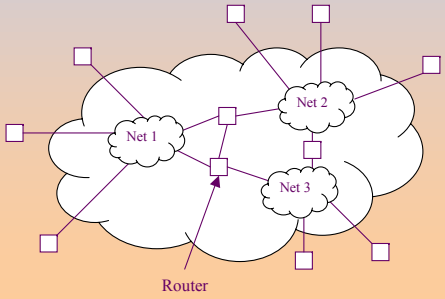
Internet

- Navidezno omrežje
  - Internet je v resnici omrežje omrežij, ki so med sabo povezana. Omrežja se med sabo razlikujejo, uporabnik in aplikacije se razlik ne ‘zavedajo’. Internet vidijo kot enotno mrežo, v kateri je veliko število računalnikov. To omogoča:
    - Univerzalen način naslavljanja
    - Univerzalne usluge
  - Vse podrobnosti o fizičnih mrežah so uporabnikom in aplikacijam prikrite



Internet

- Navidezno omrežje
  - Internet je v resnici omrežje omrežij, ki so med sabo povezana. Omrežja se med sabo razlikujejo, uporabnik in aplikacije se razlik ne ‘zavedajo’. Internet vidijo kot enotno mrežo, v kateri je veliko število računalnikov. To omogoča:
    - Univerzalen način naslavljanja
    - Univerzalne usluge
  - Vse podrobnosti o fizičnih mrežah so uporabnikom in aplikacijam prikrite



Protokol IP *(Internet Protocol)*

- Temeljni protokol interneta. Odgovoren je za promet podatkovnih paketov med računalniki in med omrežji.
- Upravlja naslovni del vsakega podatkovnega paketa in na ta način poskrbi, da pridejo od izvornega računalnika prek omrežij do naslovljenega računalnika, na podlagi njegovega številčnega naslova IP.
- Vsak paket obravnava neodvisno, kar pomeni, da mora vsak paket vsebovati vse informacije o naslovu. Zato tudi te naslove imenujemo naslove IP.
- Omrežje TCP/IP je medij za prenos paketov IP od izvornega do naslovljenega računalnika.
- Promet podatkovnih paketov med omrežji usmerjajo, na podlagi naslovov v paketih t.i. usmerjevalniki (router), ki navadno povezujejo različna omrežja v Internet.

Protokol TCP (Transmission Control Protocol

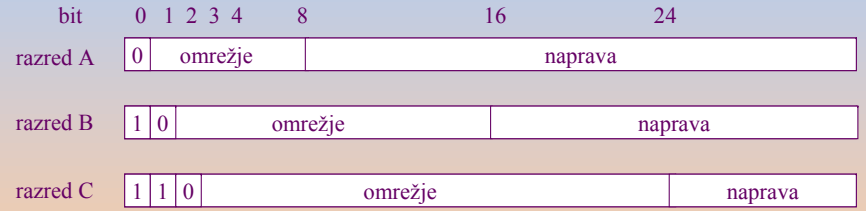
- Upravlja pakiranje podatkov v pakete, ki jih usmerjevalniki usmerjajo po različnih poteh prek Interneta.
- TCP je odgovoren za nadzorovanje pravilnega prenosa podatkovnih paketov od odjemalca k strežniku. Podatkovni paketi se lahko v vmesnih omrežjih izgubijo.
- Protokol TCP zagotavlja zanesljive omrežne povezave, ki odpravljajo pomanjkljivosti protokola IP.

Naslavljanje

- Naslavljanje je določeno s protokolom **IP.**
- Vsaka naprava, ki je priključena v internet ima svojo 32-bitno številko - **IP naslov (številko)**.
- Dve ali več naprav ne more imeti istega IP naslova.
- IP naslov je razdeljen v dva dela: številko omrežja in številko naprave.
  - **Številka omrežja:** določa omrežje, v katero je računalnik priključen.
  - **Številka naprave:** določa napravo v omrežju.
  - IP Naslov, sestavljen iz štirih zlogov, zapišemo s pikami med številkami, npr. 192.160.15.39
- Protokol TCP naslovu IP doda še 16-bitno številko vrat (port).

Razredi IP naslovov

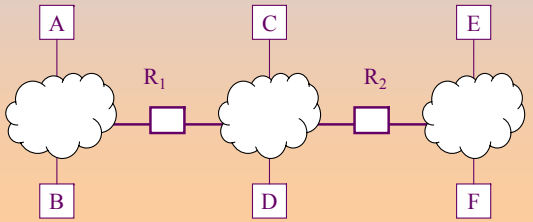
- Pri IP naslovih ločujemo tri različne vrste delitve na omrežje in napravo, razrede A, B in C



Razreševanje naslovov

- Protokoli višjih slojev (TCP in IP) uporabljajo za naslavljanje IP naslove.
  - Shema “navideznega omrežja".
  - Prikrije aparaturne podrobnosti.
- Omrežna oprema mora uporabljati **naslove, definirane na nivoju aparaturne opreme** (hardware address), npr. MAC (Media Access Control) v ethernet omrežju.
- IP naslove je pred pošiljanjem paketa potrebno razrešiti v naslove na nivoju aparaturne opreme.
- Razreševanje je lokalno v posamezni mreži. Protokol ARP (Address Resolution Protocol)

- Razreševanje naslovov
  - **A** razreši IP naslov v MAC naslov **B**
  - **A** ne more razrešiti IP naslova **F.**
    - A (po IP naslovu) ugotovi, da F ni v istem omrežju, zato paket posreduje usmerjevalniku R1.
      - IP naslov R1 razreši v njegov MAC naslov in mu pošlje paket
    - R1 (po IP naslovu) ugotovi, da F ni v omrežju, v katerega je povezan, sporočilo posreduje usmerjevalniku R2.
      - IP naslov R2 razreši v njegov MAC naslov in mu pošlje paket
    - R2 ugotovi (preko IP naslova), da je priključen v isto omrežje kot F, IP naslov F razreši v MAC F in pošlje paket.

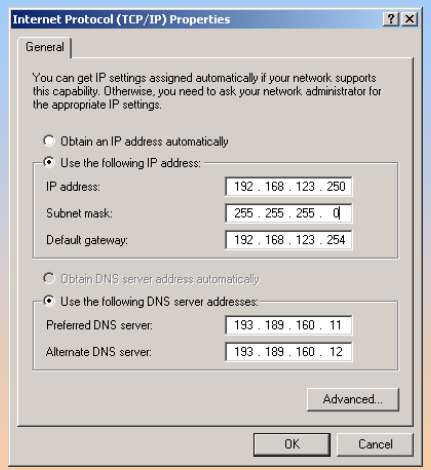


- ARP – dinamično razreševanje
  - Za razreševanje IP naslovov se uporabi omrežje
  - Dve možnosti:
    - ***Strežnik*** *– IP naslove razrešuje strežnik.*
    - **Porazdeljeno** – pošiljatelj pošlje sporočilo vsem napravam v omrežju (lokalnem); ‘lastnik’ IP naslova mu odgovori in mu pošlje MAC naslov.
      - Računalnik hrani kombinacije IP naslov – MAC v tabeli; po določenem času se par briše.
      - Windows in Linux: **arp -a**

Sistem domen (DNS - domain name system)

- IP naslove si je težko zapomniti
- Prevajanje simboličnih imen v IP naslove
- Strežniki DNS – tabele za preslikovanje
- razor.arnes.si
  - Osnovni DNS prepusti prevajanje WEP za domeni .si
  - Tabele za poddomeno arnes.si so v Arnesu => tolmačenje preda ustreznemu strežniku
  - Pridemo do IP naslova 193.2.1.80
- ping razor.arnes.si

Primer konfiguracije TCP/IP



- IP naslov: 192.168.123.250
- naslov razreda C
- naslove 192.168.123.0 – 192.168.123.255 razreši sam (ARP)
- pakete za vse ostale naslove se pošlje prehodu (usmerjevalniku) na naslovu 192.168.123.254
- za prevajanje imen v IP naslove se uporabi DNS strežnik na naslovu 193.189.16.11 ali 193.189.160.12

Aplikacijski sloj

- WWW – Svetovni splet
  - HTML - osnove
  - HTTP
- FTP

WWW – Svetovni splet

- Svetovni splet (WWW) so razvili Tim Berners-Lee in drugi znanstveniki na inštitutu CERN (European center for nuclear research), v poznih 1980 in zgodnjih 1990 letih.
- WWW ustreza modelu odjemalec strežnik in uporablja povezave TCP za prenos vsebine spletnih strani od strežnika do odjemalca.
- WWW uporablja hipertekst oziroma zapis HTML (HyperText Markup Language). Ta omogoča interaktivno dostopanje do množice dokumentov (strani).
- Dokument lahko vsebuje
  - Besedilo (hipertekst), Grafiko, Zvok, Animacije, Video
- Dokumenti so povezani s hiperpovezavami

WWW – hiperpovezave (povezave)

- Dokument (lahko) vsebuje povezave (kazalce) do drugih dokumentov.
- Povezava je predstavljena kot “aktivno območje"
  - Grafika - gumb
  - Besedilo – poudarjeno, podčrtane, obarvano..
- Ko izberemo določeno povezavo, odjemalec prenese ustrezen dokument s strežnika in ga prikaže.
- Povezave so lahko neveljavne.
- Povezava ni nič drugega kot ime dokumenta na določenem strežniku.
- Dokument lahko nekdo premakne ali izbriše, povezava pa ostane…

WWW - dokumenti

- Dokumentom WWW pravimo (spletne) strani.
- Začetna (vstopna) stran neke organizacije se imenuje **domača stran**.
- Spletna stran lahko vsebuje različne tipe podatkov; določati mora:
  - Vsebino (content) – Sami podatki
  - Vrsto vsebine (type of content) – Vrsta podatkov, npr.: besedilo, slika…
  - Povezave do drugih dokumentov
- Spletne strani so zapisane v določenem formatu **mark up language** (označevalni jezik), ki ga 'razumejo' vsi odjemalci.
- Različni odjemalci – brskalniki, lahko strani prikazujejo na različne načine, vendar so ti zelo podobni.
- To omogoča odjemalcem, ki ne tečejo v grfičnem okolju, da npr. ne prikazujejo slik.
- Standard se imenuje **HyperText Markup Language** (HTML).

WWW - HTML

- HTML določa
  - Grobo strukturo dokumenta
  - Ukaze za primerno oblikovanje strani, ki jih brskalnik izvede.
  - Hipertekstne povezave – Povezave do drugih dokumentov
  - Dodatne informacije o vsebini dokumenta
- Dokument se deli v dva dela:
  - **Glava (Head)** vsebuje podrobnosti o vsebini.
  - **Telo (Body)** vsebina dokumenta.
- Spletna stran je besedilo ASCII z dodatnimi oznakami (tag) HTML, ki odjemalcu predstavljajo navodila za oblikovanje prikaza strani.
  - Oblikovan odsek strani se začne z <TAGNAME>
  - Konec odseka označimo z </TAGNAME>

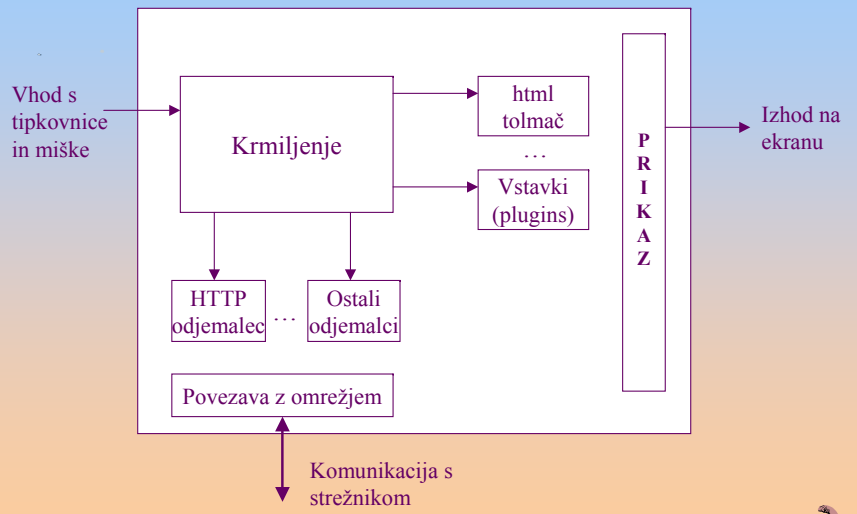
WWW – primer HTML

```
<html>

<head>
<title>Primer HTML</title>
</head>

<body>
<p>
An image:
<img src="constr4.gif"
width="144" height="50">
</p>

<p>
A moving image:
<img src="hackanm.gif"
width="48" height="48">
</p>

<p>
Note that the syntax of inserting a moving image
    is no different from that of a non-moving
    image.
</p>
</body>
</html>
```

WWW – model odjemalec-strežnik

- Brskalnik je odjemalec, WWW (web) server je strežnik.
- Odjemalec:
  - Odjemalec vzpostavi TCP povezavo s strežnikom.
  - Odjemalec pošlje zahtevo za določeno spletno stran, ki jo želi prikazati.
  - Odjemalec vsebino strani sprejme preko TCP povezave in jo prikaže v oknu brskalnika.
  - Strežnik prekine TCP povezavo.
- Vsak element (slika, avdio) potrebuje svojo TCP povezavo.
- HyperText Transport Protocol (HTTP) določa ukaze, ki jih odjemalec (brskalnik) pošilja strežniku in odgovore, ki jih strežnik pošlje nazaj odjemalcu.

HTML odjemalec



HTML odjemalec

- Modul za izpisovanje na ekran.
- HTML tolmač za oblikovanje HTML dokumentov.
- Vstavki za prikaz drugih vsebin (npr. Shockwave ali Real Audio)
- HTTP odjemalec za prenos HTML dokumentov z WWW strežnika.
- Odjemalci za druge protokole (npr. ftp)
- Modul za obdelavo vhoda s tipkovnice, miške.

- Internet Explorer, Mozilla, Konqueror, Opera, Lynx, …

HTML strežnik

- WWW strežnika čaka, da odjemalec vzpostavi TCP povezavo na določenih vratih (80).
- Ko je povezava vzpostavljena, strežnik preko nje sprejme HTTP ukaz, ki ga pošlje odjemalec.
- Pošlje zahtevano vsebino.

- IIS, Apache, Xitami, …

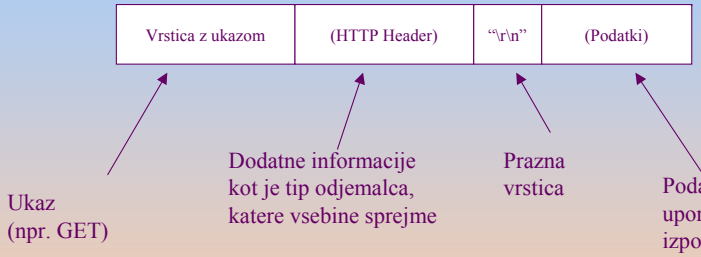
WWW- HTTP protokol

- Ko uporabnik natipka **http://www.yahoo.com/Recreation/Sports/Soccer/index.html**, odjemalec WWW strežniku preko TCP povezave pošlje HTTP ukaz GET.
- Za naveden primer je oblika ukaza GET naslednja:

```
GET /Recreation/Sports/Soccer/index.html HTTP/1.0
User-Agent: InternetExplorer/5.0
Accept: text/html, text/plain, image/gif, audio/au
"\r\n"
```

WWW – ukazi HTTP

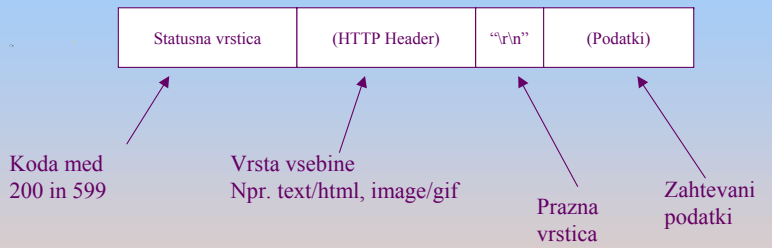
- HTTP ukaze odjemalec pošilja strežniku.



  - Obstaja več HTTP ukazov
    - **Get** – uporabimo za prenos strani s strežnika
    - **Head** – prenese se samo glava dokumenta, npr. za preverjanje povezav pri iskalnikih
    - **Post** – pošiljanje podatkov strežnikov (obrazci)
    - **Put / Delete** – vzdrževanje spletnih strani, običajno odjemalci ne podpirajo.

WWW – odgovori HTTP

- HTTP odgovore strežnik pošilja odjemalcu.



  - Statusna vrstica vrne številko, ki predstavlja uspešno / neuspešno izvedbo zadnjega HTTP ukaza.
    - 200 – 299 Uspešno
    - 300 – 399 Preusmeritev – dokument je premaknjen
    - 400 – 499 Napaka odjemalca – neznan ukaz, nedovoljen dostop,…
    - 500 – 599 Napaka strežnika – interna napaka, preobremenjen

FTP - File Transfer Protocol

- Osnovna funkcija FTP je izmenjava datotek preko interneta.
- Omogoča tudi:
  - Uporabnikom omogoča upravljanje z računalnikom na daljavo.
  - Pred uporabniki prikrije razlike v datotečnih sistemih. Format datotek je na različnih sistemih različen. Prilagoditi je potrebno tudi imena datotek, npr. glede na omejitve dolžine imen.
  - Prenos datotek med računalniki mora biti zanesljiv in čim hitrejši. FTP omogoča tudi prenos velikih datotek po kosih.

FTP - File Transfer Protocol

- FTP ustreza modelu odjemalec/strežnik
- FTP odjemalec uporabniku omogoča interakcijo s strežnikom. Na voljo so različni ukazi za dostopanje do datotek na strežniku..
- Odjemalci so različnih izvedb:
  - Preprosti programi, ki podpirajo ukazno vrstico. Npr. v ukazni vrstici
    C:\ ftp ftp.maths.tcd.ie
  - Integrirani v brskalnike, npr.. Netscape Navigator, Internet Explorer.
- FTP zagotavlja podobne usluge kot jih podpira večina datotečnih sistemov. Izpis vsebine direktorijev, ustvarjanje datotek, prenos datotek, brisanje datotek…
- FTP uporablja TCP povezave, običajno so za FTP uporabljena vrata 21.

FTP - ukazi

Ukaz	Opis
ftp maths.tcd.ie	vzpostavi povezavo do strežnika
ls	izpis vsebine direktorija
cd	zamenjaj direktorij
bin	preklopi v binarni način; prenos datotek, ki niso besedilo
get	Prenesi datoteko s strežnika na lokalni računalnik
reget	Nadaljuj prenos datoteke s strežnika na lokalni računalnik
put	Prenesi datoteko z lokalnega računalnika na strežnik
mget	multiple get
mput	multiple put
bye	konec seje

FTP – nadaljevaje prenosa

- Prenos datotek se lahko prekine še preden je prenesena celotna datoteka.
  - Razlog je lahko v napaki na strani odjemalca ali strežnika, lahko se prekine TCP povezava…
- FTP omogoča, da se prenos nadaljuje od tiste točke, kjer je bil prekinjen; ni potrebno zopet prenašati celotne datoteke.
- FTP to zagotovi z uporabo označb - **restart markers,** ki si jih izmenjujeta odjemalec in strežnik.
- Te označbe odjemalec shrani v posebno datoteko. Ko želi nadaljevati s prenosom, označbe pošlje strežniku.

Elektronska pošta

- Protokoli
  - SMTP
  - POP, IMAP
  - Priponke
- Varnost
- Požarni zid

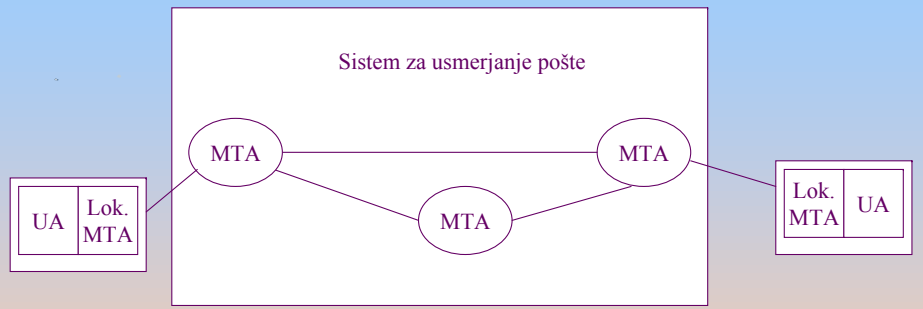
Namen elektronske pošte

- Ray Tomlinson 1971
- Prenos sporočil med dvema (ali večimi) uporabnikoma
- Sporočilo lahko vsebuje:
  - Besedilo
  - Multimedijska vsebina – Slika, Avdio ali Video
  - Izvršljivi programi, aplikacije
- Elektronska pošta je v bistvu prenos datoteke od pošiljatelja k naslovniku na zahtevo pošiljatelja.
- Pri HTTP je obratno, datoteka se s strežnika prenese na zahtevo naslovnika.
- Datoteka - sporočilo, ki se prenaša, je **besedilo;** tudi priponke se pretvorijo v 7-bitno ASCII besedilo.

Kako deluje elektronska pošta?

- Oba, pošiljatelj in naslovnik uporabljata odjemalca elektronske pošte (UA – User Agent). Ta mora podpirati tako pošiljanje kot sprejemanje elektronske pošte:
  - Ustvarjanje / Prikaz sporočil
  - Prenos sporočil do sistema za obdelavo (prenos) sporočil..
- **Sistem za usmerjanje pošte** (MHS - Mail Handling System) usmerja sporočila od pošiljatelja do naslovnika.
- Sistem za usmerjanje pošte je v bistvu mreža medsebojno povezanih **prenosnih agentov** (MTA - Mail Transfer Agents) oziroma **poštnih strežnikov** (Mail Servers).

Kako deluje elektronska pošta?



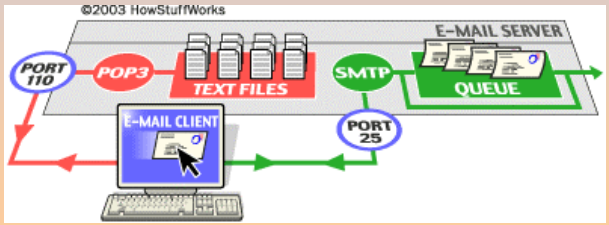
- Oblika sporočil, ki se prenašajo med uporabniškima agentoma je standardizirana.

Kako deluje elektronska pošta?

- Za prenos elektronskih sporočil sta zadolžena dva protokola.
- Prvi protokol prenaša sporočila med poštnim odjemalcem (UA) in lokalnim prenosnim agentom (MTA - lokalni pošti strežnik).
- To velja za sprejemanje in oddajanje sporočil. Pogosta sta dva protokola (odvisno od lokalnega poštena strežnika), ki omogočata prenos od lokalnega poštnega strežnika do odjemalca:
  - POP (Post Office Protocol)
  - IMAP (Internet Message Access Protocol)
- Drugi protokol prenaša sporočila med MTAji, imenuje se SMTP(Simple Mail Transport Protocol).

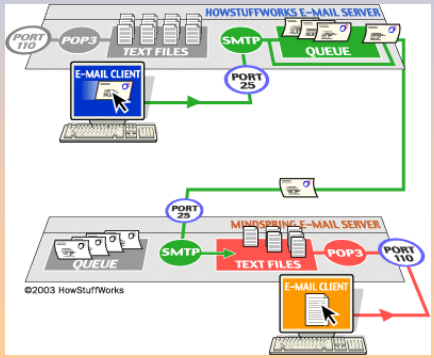
Poštni strežnik

- Dva strežnika, eden za pošiljanje, drugi za sprejemanje
  - Razlog za uporabo dveh različnih protokolov, enega za pošiljanje sporočil in prenos med MTAji in drugega za prenos sporočil od lokalnega strežnika do odjemalca med lokalnim strežnikom in odjemalcem, je v tem, da uporabniki svoje računalnike izklapljajo.
  - Sporočila s poštnega strežnika se prenesejo na zahtevo odjemalca; prenos sproži pošiljatelj – odjemalec pri naslovniku je lahko izklopljen.
  - Če je naslovnikov računalnik izklopljen, bodo sporočila ostala shranjena v MTAju, dokler uporabnik sporočila ne bo dobil.
  - Pomembno je tudi dejstvo, da v primeru nedostopnosti lokalnega poštnega strežnika ostanejo shranjena v drugih MTAjih, dokler sporočil ni možno predati naprej.



Pošiljanje elektronske pošte

- Pri pošiljanju mora pošiljatelj navesti naslovnika in druge parametre; npr. ali zahteva potrditev sprejema…
- Sporočilo mora biti v 7-bitnem ASCII zapisu; gre torej za besedilo.
- Naslovi imajo obliko mailbox@location.com, mailbox določa uporabnika, ki bo sporočilo dobil, del za @ določa poštni strežnik naslovnika.



Format sporočila

- Glava sporočila vsebuje:

Polje	Opis
To:	Naslovnik
Cc:	Ostali naslovniki
Bcc:	Ostali naslovniki…
From:	Naslov pošiljatelja
Reply-To:	Naslov za odgovor na sporočilo; običajno enak naslovu pošiljatelja.
Received:	Zaporedje MTAjev, ki so posredovali sporočilo med potjo do naslovnika.
Content-Length	Dolžina sporočila

Priponke - MIME

- Multimedia Mail Extensions
- HTTP dopušča zgolj prenos 7-bitnega ASCII besedila
- Priponke (običajno) niso besedilo; MIME jih prekodira
  - uuencode / uudecode
  - 6 bitov + 32; 3 bajti -> 4 ASCII znaki
- Dodatna polja v glavi sporočila

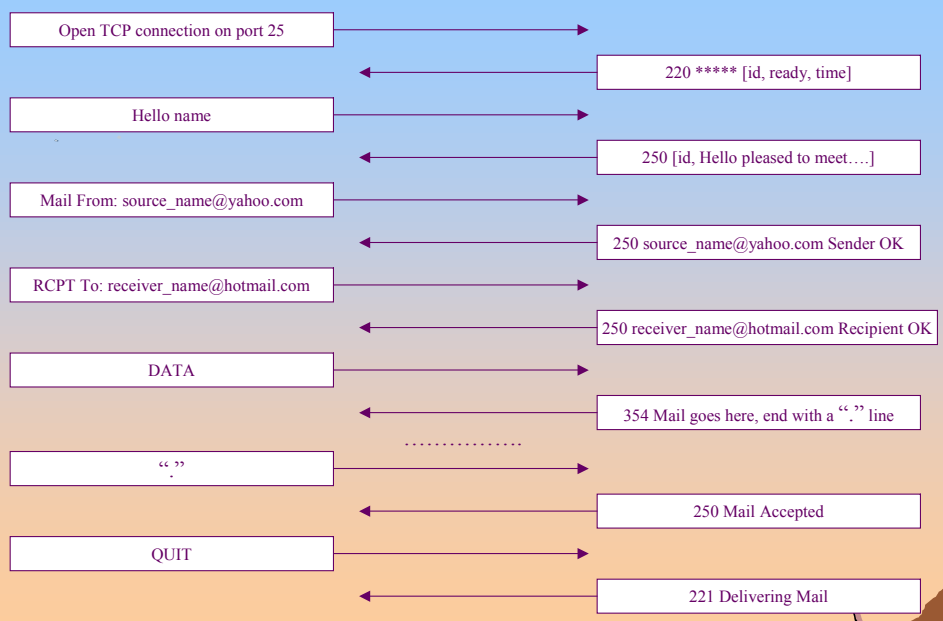
Primer sporočila s priponko

```
Return-Path: <andrej.strancar@fri.uni-lj.si>
Received: from Scenic (nexus.fri.uni-lj.si [193.2.76.116])
        by postar.fri.uni-lj.si (Postfix) with SMTP id 59922BC91
        for <andrej.strancar@fri.uni-lj.si>; Sat, 16 Oct 2004 11:54:28 +0200
    (CEST)
Message-ID: <000901c4b366$21bb5de0$744c02c1@Scenic>
Reply-To: "Andrej Strancar" <andrej.strancar@fri.uni-lj.si>
From: "Andrej Strancar" <andrej.strancar@fri.uni-lj.si>
To: "Andrej Strancar" <andrej.strancar@fri.uni-lj.si>
Subject: GUMB
Date: Sat, 16 Oct 2004 11:54:30 +0200
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="----
    =_NextPart_000_0006_01C4B376.E5311B10"
X-Priority: 3
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook Express 6.00.2900.2180
X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2900.2180

This is a multi-part message in MIME format.
```

Primer sporočila s priponko

```
------=_NextPart_000_0006_01C4B376.E5311B10
Content-Type: text/plain; format=flowed;
        charset="iso-8859-1"; reply-type=response
Content-Transfer-Encoding: 7bit

Posiljam sliko gumba za iskanje.
LP, Andrej
------=_NextPart_000_0006_01C4B376.E5311B10
Content-Type: image/gif; name="search_button.gif"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="search_button.gif"

R0lGODlhLwAUALMAAHN2c2vZawACAP7//kasRq/brwGYAQhTCJekl+Lq4gMxAyqkKgB1AMzhzC4z
LkhSSCH5BAAAAAAALAAAAAAvABQAAAT/UMhJq704zxK6/2AojqFxSIGhrmzrvrAbnEIa3zg+oyyx
5DifjmYjDl4FQ+E4SBqMDZUR+TM0kM9Bymjb1VSJhAAAUCwfD4fCEDiujQhAk7FUyAFGwCF/YBNV
Aw0CJgdLAgyIBglLAFlldQMlAgsND3x8fjxKkVVnBCoLAw+RewN6YQ6mawwKcAtteplfKksJn0tH
n20CDQmtA2Gio7FSTEexXjYqAGGFAxN0lUsORgVhk6Z9xQACw33JLQiihisJxnim4g4GpEoEl9my
NhybAp4+ogKpCHyRdAkHYMFD9mfTMXvGjDhgwC6Bt14HFpiLJGDgt4ImHkjoo4DCCYYGMyQwEMBx
EIM0AlaRNDDyoqYVIIHESBQDnMybOWzi3PnCC6EDQIMKHUq0qNGhGplqXTohAgA7

------=_NextPart_000_0006_01C4B376.E5311B10—

.
```


SMTP

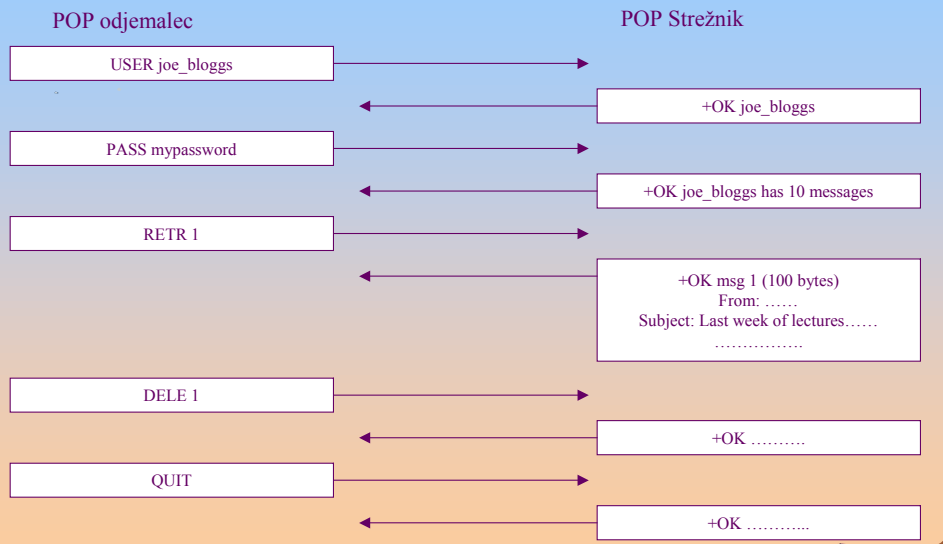


Prenos sporočil s strežnika (POP3)

- POP (Post Office Protocol) omogoča prenos sporočil s poštnega strežnika do odjemalca.
  - POP omogoča (zahteva) avtentikacijo
  - Preprost protokol, odjemalec s strežnikom komunicira preko TCP povezave na vratih **110**.
  - Odjemalec s strežnikom komunicira s preprostimi ukazi, ki omogočajo prenos in brisanje sporočil na strežniku. Možnih je še nekaj ukazov za pregled, kaj je na strežniku…
  - Trenutno je aktualna različica 3.0 – POP3.

USER – uporabniško ime
PASS - geslo
QUIT – izhod, konec seje
LIST – izpis sporočil, ki so na strežniku
RETR – prenos sporočila
DELE – brisanje sporočila
TOP – prikaz nekaj začetnih vrstic sporočila

Primer prenosa sporočila s strežnika - POP



IMAP

- Novejši protokol za komunikacijo poštni odjemalec - poštni strežnik **Internet Message Access Protocol** (**IMAP**).
- IMAP uporablja vrata **143**.
- IMAP je funkcionalno boljši POP
  - Boljša avtentikacija.
  - Uporabniki lahko uporabljajo več nabiralnikov
  - Več operacij za delo s sporočili (iskanje, iskanje po ključnih besedah…)
  - Sporočila ostanejo na strežniku; uporabnik lahko do njih dostopa ok koderkoli.
- Manj popularen: sporočila so na strežniku – veliko prostora na strežnikih ponudnika internetnih storitev (ISP)

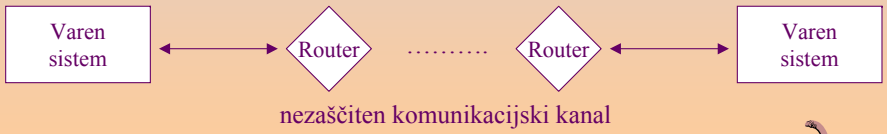
Varnost

- Pri komuniciranju preko računalniške mreže se pojavijo vprašanja v povezavi z varnostjo in zasebnostjo.
- Zelo velika omrežja, kot je internet, so ranljiva za napade. Nihče interneta ima pod nadzorom.
- Paketi se pogosto pošiljajo nezaščiteni.
- Napadalec lahko prebere sporočilo (e-pošto) pri njeni poti skozi omrežje MTAjev.
- Napadalec lahko ponaredi identiteto pošiljatelja; pošilja pošto v tujem imenu.
  - To je zelo enostavno narediti; na vrata 25 MTAja je potrebno le poslati nekaj ukazov SMTP.
- V sporočila s priponkami je mogoče podtakniti priponke s škodljivo vsebino – viruse, črve…
- Napadalec je lahko kdorkoli; iz druge države, bivši zaposleni, zaposleni v istem podjetju…

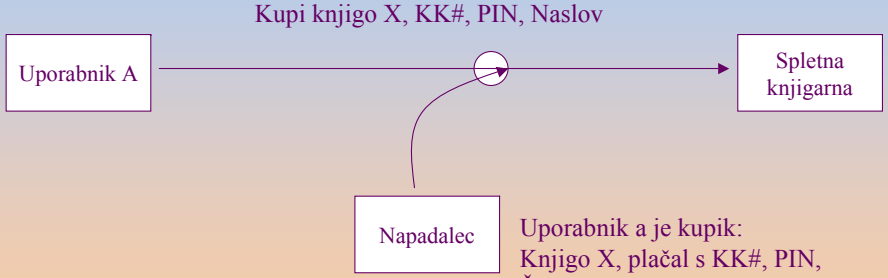
Vrste napadov

- **Pasivni napad** (podobno kot prisluškovanje telefonskim pogovorom)
- Na ta način napadalec pride do informacij, ki jih kasneje lahko uporabi.
- Npr. podatki o kreditni kartici.
- Take napade je praktično nemogoče zaznati.

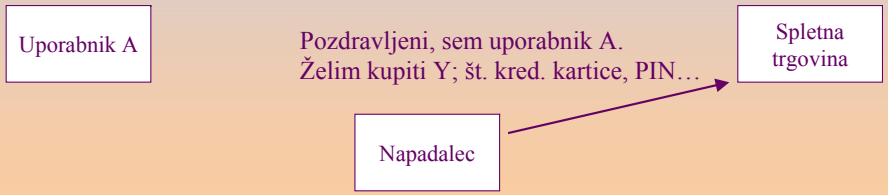
- **Aktivni napad** v primeru, ko napadalec spreminja ali duplicira podatke



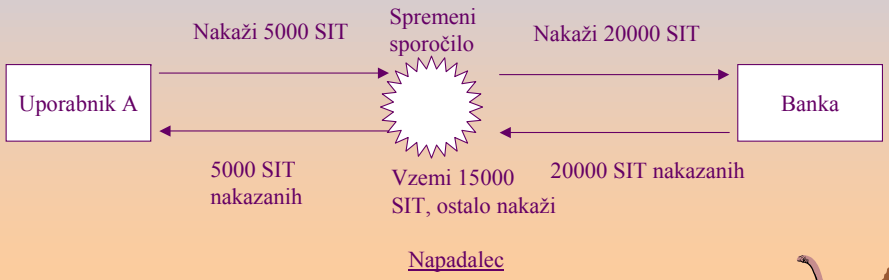
- Vrste napadov
  - **Prisluškovanje** (Pasivni napad)
  - Napadalec pridobi kopije sporočil
  - To se doseže s pregledovanjem prometa po mreži



- Vrste napadov
  - **Prevzem identitete**
  - Pošiljanje ali sprejemanje sporočil v imenu napadenega; pri tem se uporabi lažna identiteta, ukradeni podatki… Kako? Kraja gesel, PINov,št. kreditnih kartic, drugih podrobnosti s prisluškovanjem.
  - Uporaba pridobljenih podatkov…



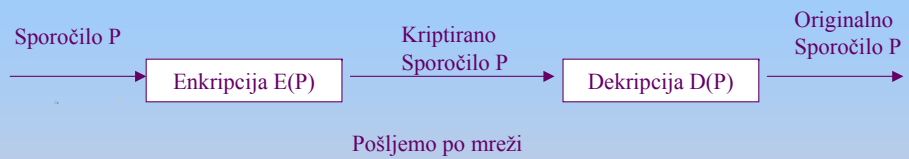
- Vrste napadov
  - **Spreminjanje sporočil**
  - Prestrezi sporočilo in ga spremeni
  - Spremenjeno sporočilo pošlji pravemu naslovniku
  - Vključuje:
    - Določene zamenjave podatkov, brisanje podatkov
    - Kreiranje drugačnega sporočila



Kriptiranje

- Enkripcija
  - Kodiranje podatkov na tak način, da jih lahko *dekriptira* samo tisti, ki ima **ključ**
  - Poznana že iz rimskih časov
- Simetrična
  - Pri enkripciji in dekripciji se uporablja isti ključ
  - Ključ poznata samo pošiljatelj in naslovnik
  - Težave z distribucijo ključa
  - Hitri postopki
- Nesimetrična (javni ključ)
  - Dva ključa E-enkripcija, D-dekripcija
  - E – **javni ključ**; pozna ga lahko 'vsakdo'
  - D – **zasebni ključ**; pozna ga samo lastnik
  - Počasni, zapleteni postopki

Kriptiranje



- Simetrična kriptografija
  - E=D=E
  - P→E(P)…→E(E(P))
  - Data Encryption Standard (DES), Tripe-DES (3DES), International Data Encryption Algorithm (IDEA), SAFER, Blowfish
- Asimetrična
  - P→E(P) … →D(E(P))
  - Rivest, Shamir, and Adleman (RSA), Digital Signature Standard (DSS), Elliptic Curve Cryptography (ECC)

Kriptiranje – RSA

- Izberemo p,q: veliki praštevili, npr. 1024 bitov
  - $n = pq$
  - $z = (p-1)(q-1)$
- Izberemo d: nima skupnih deliteljev s številom *z*.
- Izberemo e: $ed \bmod z = 1$

- P → $C=P^e \bmod n$ *kriptiranje; E = (e, n)*
- C → $P=C^d \bmod n$ *dekriptiranje; D = (d, n)*

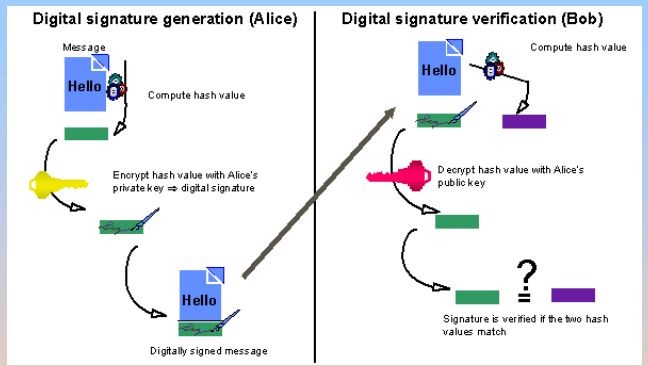
- *Razbijanje kode: n poznan; zelo težko ugotovimo p in q*

Kriptiranje – zgled RSA

- p = 3, q = 11
- n = 33, z = 20
- d = 7, e = 3
- P → $C=P^3 \bmod 33$ *enkripcija*
- C → $P=C^7 \bmod 33$ *dekripcija*

	P	P^3	C	C^7	$C^7 \bmod 33$
T	21	9261	21	1801088541	21
E	06	216	18	612220032	06
K	12	1728	12	35831808	12
S	19	6859	28	13492928512	19
T	21	9261	21	1801088541	21

Elektronski podpis



- Hash – povzetek sporočila
  - Majhna verjetnost, da imata različni sporočili enako hash vrednost
- Kriptiraj hash vrednost z zasebnim ključem
- Pošlji
- Sprejemnik ponovno izračuna hash vrednost
- Dekriptiraj sprejet elektronski podpis
- Primerjaj hash vrednosti; preveri ujemanje
- Možno, če E(D(P)) = D(E(P))

Kriptiranje - SSL

- Secure Socket Layer
- Simetrično kriptiranje na transportnem sloju
- Delovanje:
  - Odjemalec pošlje strežniku podatke o podprtih načinih kriptiranja
  - Strežnik pošlje odjemalcu svoj **javni ključ**
  - Odjemalec generira naključni ključ, ki se bo uporabljal v seji, ga kriptira s strežnikovim javnim ključem in pošlje strežniku.
  - Vzpostavi se varna povezava, podatki se kriptirajo z dogovorjenim ključem
  - Ključ velja za čas ene seje

Datotečni sistemi

- Struktura datotečnih sistemov
- Implementacija datotečnih sistemov
- Imeniki
- Metode alokacije
- Upravljanje s prostim prostorom
- Učinkovitost, zmogljivost
- Napake
- Dnevniško vodeni (journaling) datotečni sistemi
- NFS

Struktura datotečnih sistemov

- Struktura datotek
  - Logična enota za hranjenje podatkov
  - Množica povezanih podatkov
- Datotečni sistem se nahaja na sekundarnem pomnilniku (disk).
- Organiziran je v slojih.
- *Datotečni nadzorni blok* – struktura, ki vsebuje informacijo o datoteki.

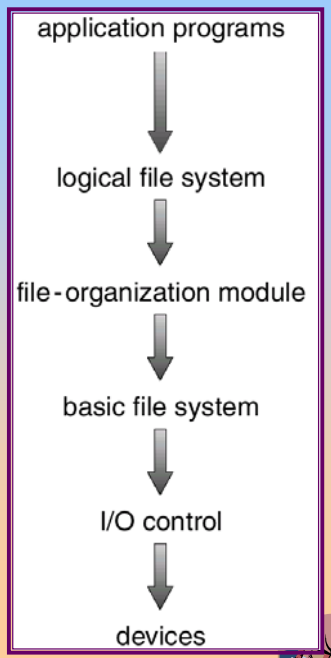
Razslojen datotečni sistem

Logični datotečni sistem
Imena datotek, direktoriji…

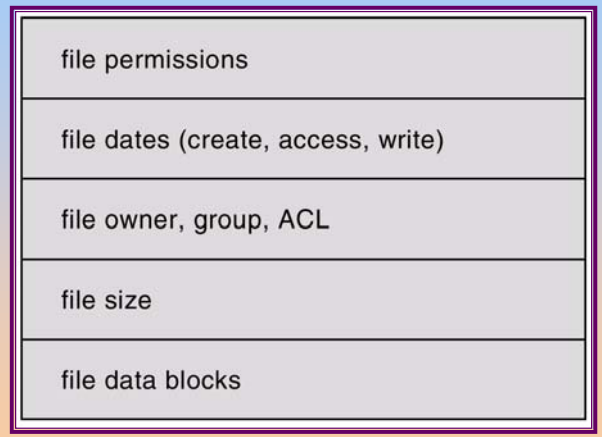
Datotečna organizacija
Zaveda se datotek, logičnih blokov (0-N) in fizičnih blokov…

Osnovni datotečni sistem
Pretvarja fizične št. Blokov v npr. (površina, cilinder, sled, sektor)

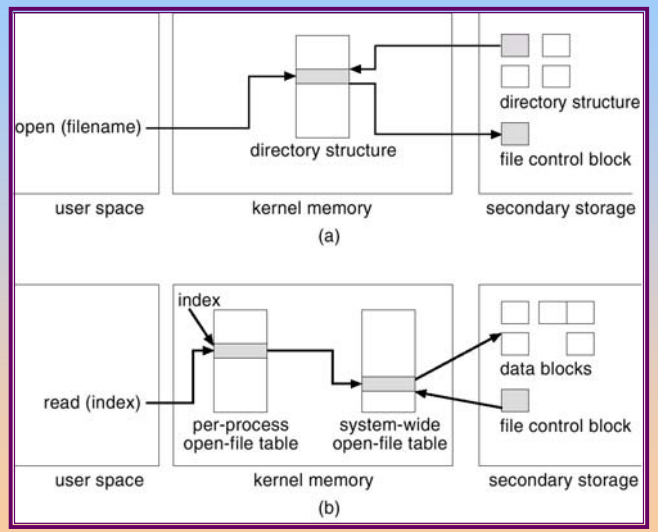
V/I nadzor
Gonilniki, prekinitve…



Tipični datotečni nadzorni blok



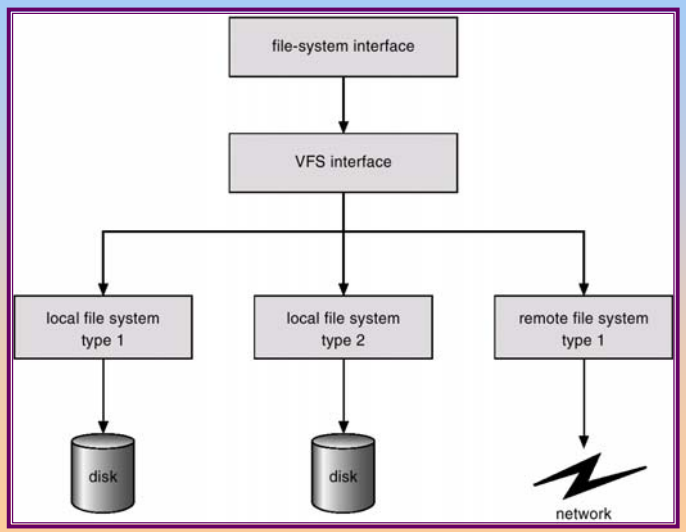
Strukture datotečnega sistema (ki so v pomnilniku)



Navidezni datotečni sistem

- Navidezni datotečni sistem (VFS) je objektno orientiran način implementacije datotečnih sistemov.
- VFS omogoča poenoten način dostopanja (skozi API), ki nje neodvisen odvisen od vrste datotečnega sistema.
- API dela z vmesnikom VFS, ne s specifično vrsto datotečnega sistema

Shematski prikaz navideznega datotečnega sistema



Implemantacija imenikov

- Linearen seznam imen datotek in ustreznih kazalcev na bloke podatkov.
  - Preprosto za programiranje
  - Počasnost pri izvajanju
- Razpršena tabela – linearen seznam z razpršenimi podatki.
  - Zmanjša čas iskanja v imeniku
  - *kolizije* – dve imeni se ‘razpršita’ v isto mesto (razpršilna funkcija ima pri obeh enako vrednost)
  - Velikost je fiksna

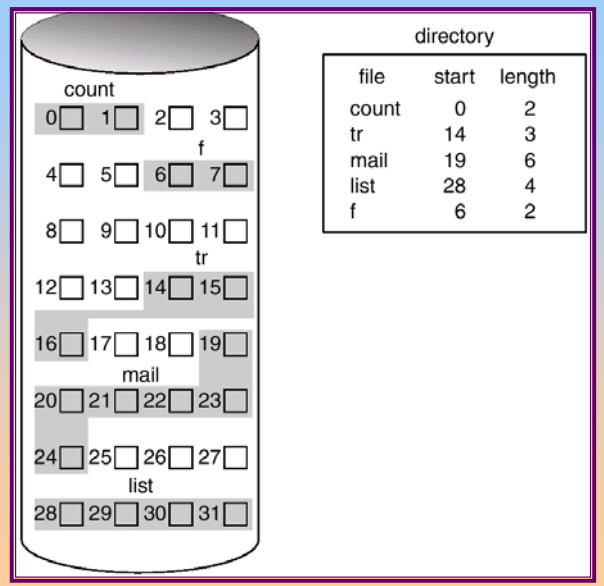
Alokacija prostora

- Način alokacije določa, kako se bloki na disku dodeljujejo datotekam:
- Kontinuirana alokacija
- Povezana alokacija
- Indeksirana alokacija

Kontinuirana alokacija

- Vsaka datoteka zaseda množico zaporednih blokov na disku.
- Preprostost – Potrebujemo le začetno lokacijo (št. Prvega bloka in dolžino (število blokov).
- Naključni dostop.
- Potrata prostora (problem dinamične alokacije).
- Datotek ni mogoče povečavati

Kontinuirana (zaporedna) alokacija prostora na disku



Povezana alokacija

- Vsaka datoteka je povezan seznam blokov na disku: bloki so po disku poljubno razmetani.

Povezana alokacija

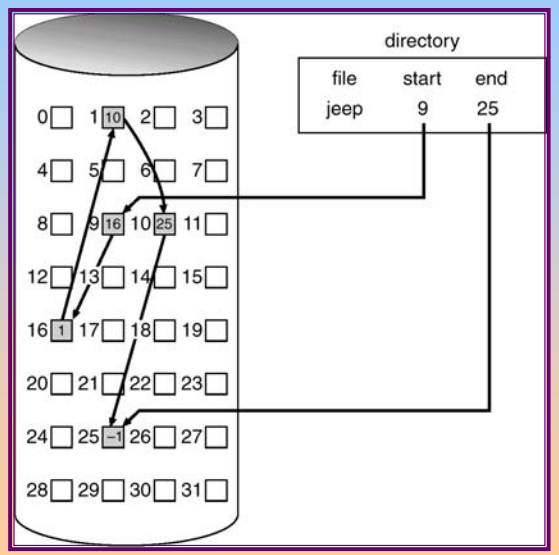
- Preprostost – samo začetni naslov (št. bloka)
- Sistem za upravljanje s prostim prostorom – ni izgube prostora
- Ni naključnega dostopa
- Preslikava

LA/512 → Q, R

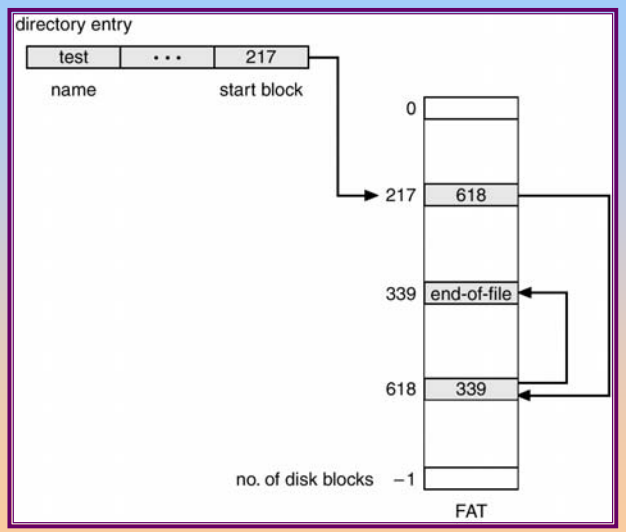
Blok, do katerega želimo dostopati je Q-ti blok v povezanem seznamu blokov, ki predstavljajo datoteko. Odmik v bloku = R

File-allocation table (FAT) – alokacija prostora ki jo uporabljata MS-DOS in OS/2.

Povezana alokacija



Alokacijska tabela



Indeksirana alokacija

- Vsi kazalci so združeni v *indeksnem bloku.*
- Logično pogled.

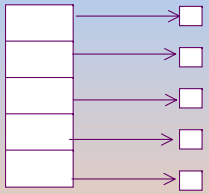
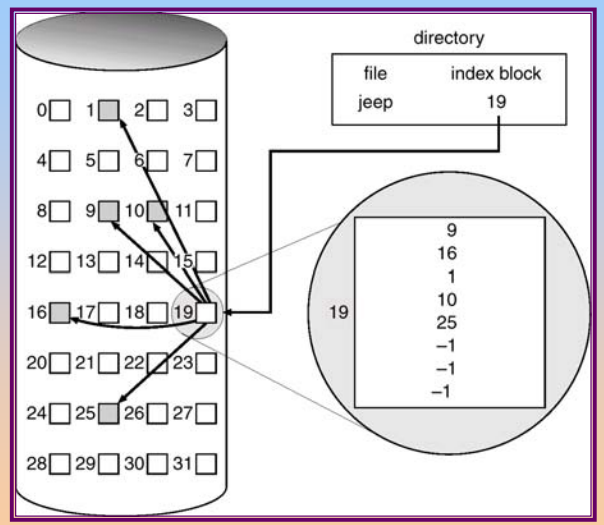


Tabela indeksov

Primer indeksirane alokacije



Indeksirana alokacija

- Potrebuje tabelo indeksov
- Naključen dostop
- Dinamični dostop brez zunanje fragmentacije, indeksni blok predstavlja izgubljen prostor.
- Preslikava iz logičnega naslova datoteke v fizičnega v datoteki z največjo dolžino 256K besed in velikostjo bloka 512 besed. Dolžina tabele indeksov je 1 blok.

LA/512 → Q, R

Q = odmik v tabeli indeksov
R = odmik v bloku

Indeksirana alokacija

- Preslikava iz logičnega naslova v fizičnega v datoteki neomejene dolžine (velikost bloka je 512 besed).
- Povezana shema – povezati bloke, v katerih je tabela indeksov (brez omejitve velikosti).

LA / (512 x 511) → Q_1, R_1

Q_1 = blok tabele indeksov
R_1 se uporabi tako:

R_1 / 512 → Q_2, R_2

Q_2 = odmik v bloku tabele indeksov
R_2 odmik v bloku datoteke:

Indeksirana alokacija

- Dvonivojski indeksi (največja velikost datoteke je 512^3)

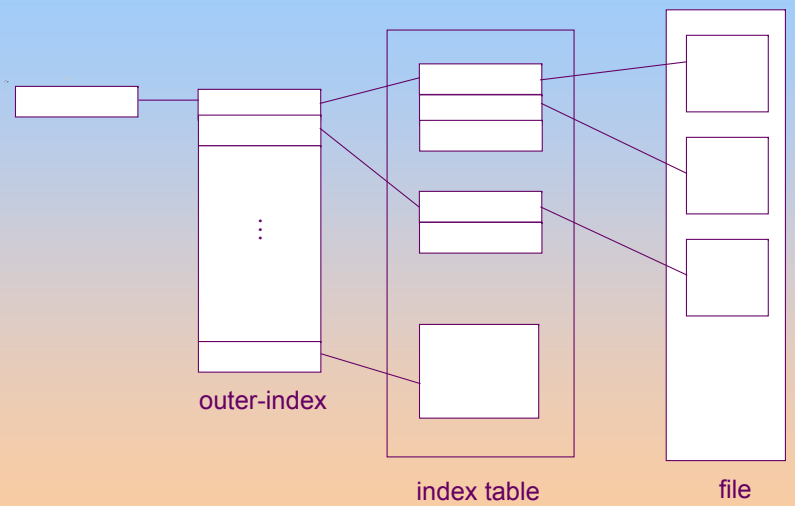
LA / (512 x 512) → Q_1, R_1

Q_1 = odmik v zunanji tabeli indeksov
R_1 se uporabi tako:

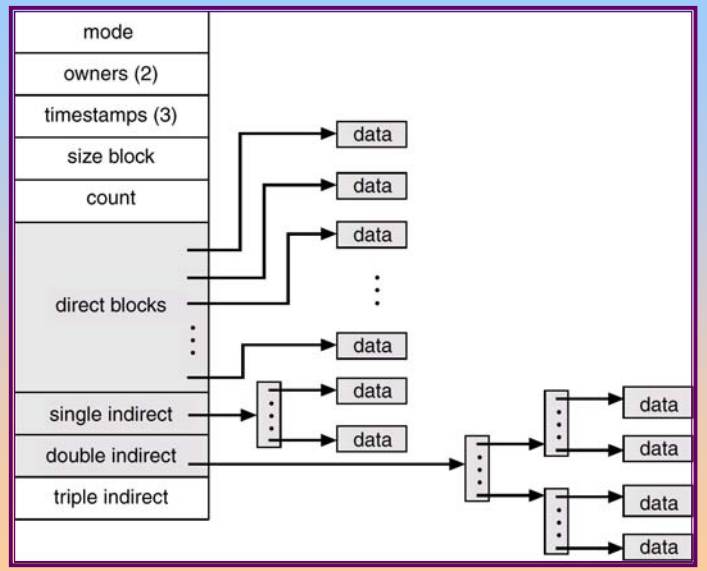
R_1 / 512 → Q_2, R_2

Q_2 = odmik v bloku tabele indeksov
R_2 odmik v bloku datoteke:

Indeksirana alokacija

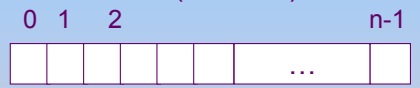


Kombinirana shema: UNIX (4KB na blok)



Upravljanje s prostim prostorom

- Bitni vektor - *Bit vector* (*n* blokov)



$$bit[i] = \begin{cases} 0 \Rightarrow \text{blok}[i] \text{ prost} \\ 1 \Rightarrow \text{blok}[i] \text{ zaseden} \end{cases}$$

Izračun števila blokov

(dolžina besede) *
(število besed z vrednostjo 0) +
Odmik prvega bita z vrednostjo 1

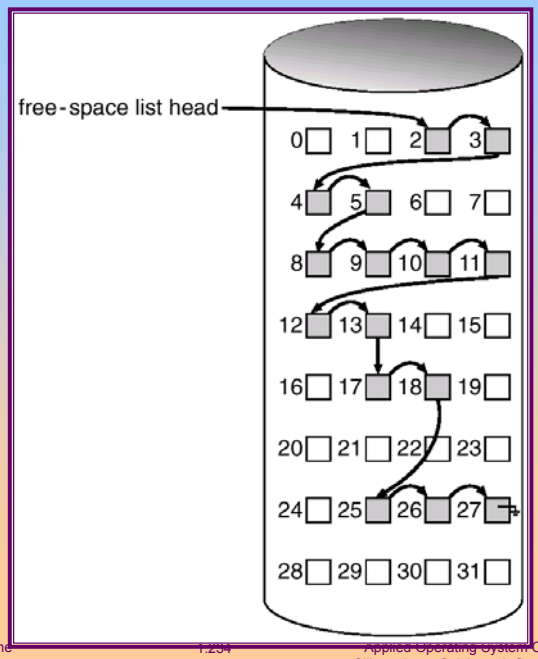
Upravljanje s prostim prostorom

- Bitni vektor zahteva dodaten prostor. Primer:
  - velikost bloka = 2^{12} bajtov
  - velikost diska = 2^{30} bajtov
  - $n = 2^{30}/2^{12} = 2^{18}$ bitov (ali 32KB)
- Preprosto dobimo kontinuirane datoteke
- Povezan seznam (free list)
  - Težko dobimo kontinuiran prostor
  - Ni izgube prostora
- Grupiranje
- Štetje

Upravljanje s prostim prostorom

- Zaščititi je potrebno:
  - Kazalec na seznam prostih blokov
  - Bitni vektor
    - Mora biti shranjen na disku
    - Kopija v primarnem pomnilniku se lahko razlikuje od tiste na disku.
    - Ni dovoljeno, da bi za blok[*i*] veljalo: bit[*i*] = 1 v primarnem pomnilniku in bit[*i*] = 0 na disku.
  - Rešitev:
    - Postavi bit[*i*] = 1 na disku
    - Alociraj blok[*i*]
    - Postavi bit[*i*] = 1 v pomnilniku

Povezan seznam prostih blokov



Izraba prostora, hitrost dostopa

- Izraba prostora je odvisna od:
  - Načina alokacije prostora in izvedbe direktorijev
  - Količine podatkov o datoteki, ki so spravljeni v vnosu v direktoriju
- Hitrost dostopa
  - predpomnilnik – rezerviramo del glavnega pomnilnika za predpomnjenje pogosto zahtevanih blokov
  - free-behind in branje vnaprej – optimizacija sekvenčnega dostopa

Napake

- Preverjanje konsistentnosti – primerja podatke v imeniški strukturi s podatkovnimi bloki na disku in poskuša odpraviti nekonsistentnosti.
- Podatke z diska *arkiviramo* (back up) na druge naprave (trakovi, CD, …).
- Izgubljene podatke obnovimo iz arhiva.

Datotečni sistemi z dnevniškim vodenjem

- **Datotečni sistemi z dnevniškim vodenjem** (journaling) na disk zapisujejo nameravane spremembe, šele nato pa se te izvršijo - **transakcije**.
- Vse transakcije se vpisujejo v dnevnik - **log**. Transakcija se smatra za **izvršeno,** ko je uspešno vpisana v dnevnik – ni rečeno, da se je že vpisala tudi v datotečni sistem.
- Transakcije se iz dnevnika prepišejo v datotečni sistem; ko se spremembo vpiše v datotečni sistem, se transakcijo odstrani iz dnevnika.
- Če pride do sesutja datotečnega sistema, transakcije, ki jih je potrebno izvršiti ostanejo v dnevniku.

Sun NFS (Network File System)

- Implementacija in specifikacija sistema (programskega) za dostopanje do datotek preko računalniške mreže (LAN, WAN).
- Implementacija je del Solarisa in SunOS (Sunove delovne postaje) in temelji na protokolu UDP in Ethernetu.

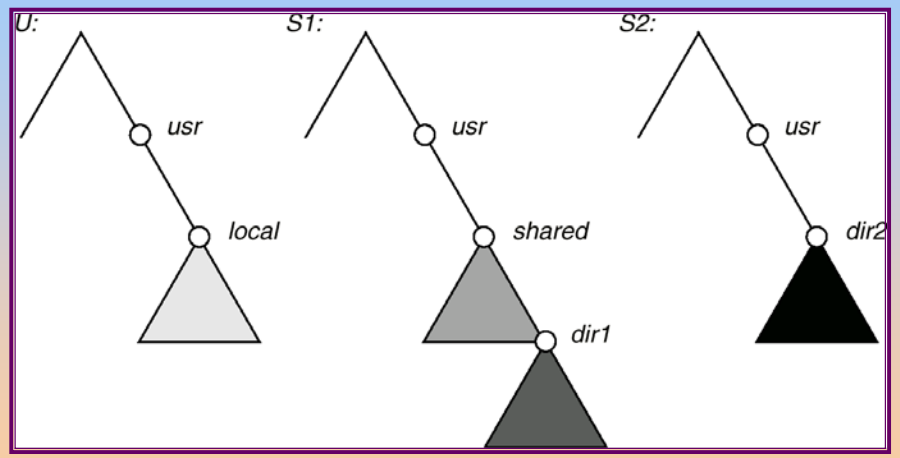
NFS

- Povezane računalnike si predstavljamo kot množico neodvisnih računalnikov z neodvisnimi datotečnimi sistemi. Omogoča skupno rabo datotek na različnih računalnikih.
  - Oddaljen direktorij je mogoče priklopiti na lokalni direktorij. Oddaljen direktorij je viden kot poddrevo lokalnega datotečnega sistema (zamenja morebitno lokalno poddrevo).
  - Ob priklopu oddaljenega direktorija je potrebno navesti ime gostitelja in ime oddaljenega direktorija. Do datotek v oddaljenem direktoriju se dostopa unako kot do lokalnih.
  - Subject to access-rights accreditation, potentially any file system (or directory within a file system), can be mounted remotely on top of any local directory.

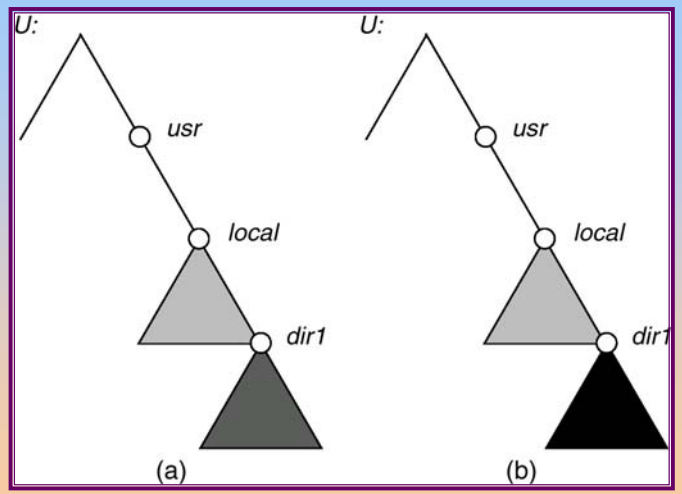
NFS

- NFS je namenjen delovanju v heterogenih sistemih različnih računalnikov, operacijskih sistemov in omrežij. NFS je od vsega naštetega neodvisen.
- To neodvisnost so dosegli z uporabo klicev oddaljenih procedur RPC in temelji na protokolu XDR (External Data Representation).
- NFS razlikuje med uslugami, ki jih zagotavlja mehanizem priklopa (mount) in dejanskimi uslugami dostopa do oddaljenih datotek.

Neodvisni datotečni sistemi



Priklop v NFS



Priklopi Kaskadni priklopi

NFS Protokol priklopa (mount)

- Vzpostavi začetno logično povezavo med strežnikom in odjemalcem.
- Operacija priklopa (mount) zajema ime oddaljenega direktorija, ki ga želimo priklopiti in ime strežnika, na katerem se oddaljen direktorij nahaja.
  - Zahteva po priklopu se preslika v ustrezni RPC in posreduje priklopnemu strežniku (mount server) na strežnem računalniku.
  - *Export list* – določa lokalne datotečne sisteme, ki jih strežnik dovoljuje priklopiti, skupaj z imeni računalnikov, ki jim je te datotečne sisteme dovoljeno priklopiti.
- Po zahtevi po priklopu (mount request) ob predpostavki, da odjemalec zahtevan datotečni sistem lahko priklopi, strežnik vrne ročico datoteke (*file handle)*—ključ za nadaljnje dostope.
- Ročica datoteke – identifikacija datotečnega sistema in številka vozla (inode number) direktorija.
- Operacija priklopa na strani strežnika ne naredi ničesar, zgolj spremeni odjemalčev pogled…

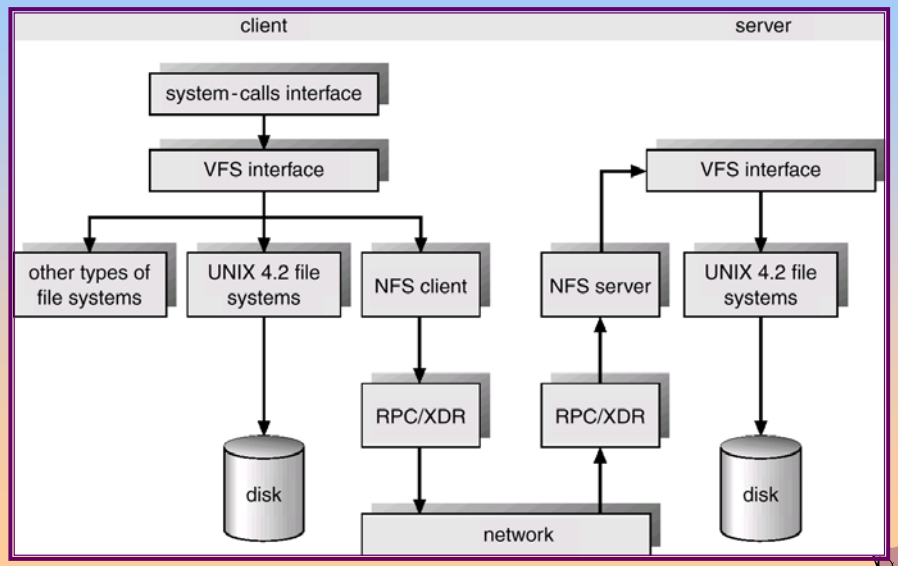
NFS Protokol

- Zagotavlja množico RPCjev za delo z oddaljenimi datotekami. Podprte so naslednje operacije:
  - Iskanje datoteke v direktoriju
  - Branje množice vnosov v direktoriju
  - Delo s povezavami in direktoriji
  - Dostop do atributov
  - Branje in pisanje datotek
- Strežnik NFS ne pomni stanja o odjemalcih; je torej brez stanja (*stateless)*; vsaka zahteva mora vsebovati vse potrebne podatke.
- Spremenjeni podatki se morajo najprej zapisati na disk strežnika, šele nato se vrne rezultat odjemalcu. (izgubimo prednosti predpomnjenja)
- Protokol NFS ne omogoča mehanizmov za nadzor sočasnosti.

Trije sloji NFS arhitekture

- UNIX-ov vmesnik datotečnega sistema (temelji na klicih **open, read, write**, in **close**, in opisnikih-deskriptorjih datotek).
- Sloj *navideznega datotečnega sistema* (VFS)– Razlikuje med lokalnimi in oddaljenimi datotekami, dodatno razlikuje med lokalnimi glede na tip njihovega datotečnega sistema.
  - VFS ob dostopu do lokalnih datotek aktivira ustrezne operacije glede na datotečni sistem, na katerem je določena datoteka.
  - Pri dostopu do oddaljenih datoteh pokliče ustrezno proceduro RPC.
- Uslužnostni sloj (service layer) NFS– spodnji nivo arhitekture; implementira protokol NFS.

Shematski prikaz arhitekture NFS



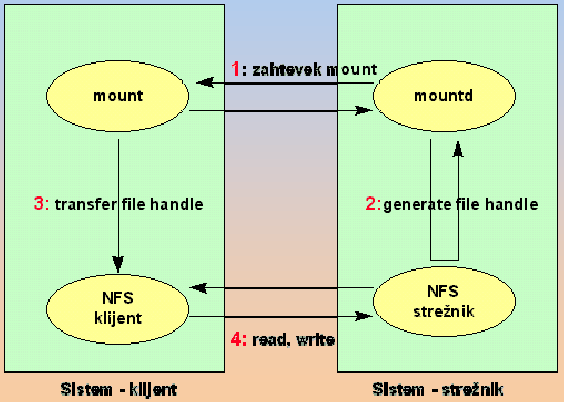
NFS – preslikava poti

- Pot je potrebno prelomiti v lokalni in oddaljeni del???Performed by breaking the path into component names and performing a separate NFS lookup call for every pair of component name and directory vnode.
- Da je iskanje hitrejše, se na strani odjemalca uporablja predpomnjenje v-vozlov za imena oddaljenih direktorijev.

Priklop na strežnik NFS

•Pri zagonu sistema se v strežniku pužene (poleg demonskega procesa ***nfsd***) tudi demonski proces ***mountd***.

•Na strani klijenta uporabimo ukaz ***mount.***



/etc/mount -f NFS [,opcije] hostName:pathName directory

UNIX - Datotečni sistem

- UNIXov datotečni sistem pozna dva objekta: datoteke in direktorije.
- Direktoriji so datoteke, ki imajo določen format; torej je v resnici objekt samo datoteka.

Bloki in fragmenti

- Večino datotečnega sistema predstavljajo *podatkovni bloki*.
- 4.2BSD uporablaj dve velikosti blokov za datoteke brez indirektnih blokov:
  - All Vsi bloki, ki pripadajo datoteki so večje velikosti (npr. 8K), izjema je zadnji blok.
  - Velikost zadnjega bloka je primeren večkratnik *velikosti fragmenta* (npr. 1024) – glede na velikost datoteke.
  - Datoteka veliosti 18,000 bajtov bi zasedala dva bloka velikosti 8x in fragment velikosti 2KB (ki ne bi bil popolnoma zaseden).

Bloki in fragmenti

- Velikost *bloka* in *fragmenta* se določi ob kreiranju datotečnega sistema glede na nameravano uporabo:
  - Če pričakujemo majhne datoteke, izberemo majhno velikost fragmenta.
  - Če pričakujemo ponavljajoče prenose velikih datotek, izberemo večjo velikost bloka.
- Največje razmerje med velikostjo bloka in fragmenta 8 : 1; najmanjša velikost bloka je 4K (tipični izbiri sta 4096 : 512 in 8192 : 1024).

I-vozli

- Vsaki datoteki pripada i-vozel (*inode)* — zapis, ki hrani podatke o določeni datoteki na disku.
- I-vozel vsebuje tudi 15 kazalcev na bloke, ki vsebujejo podatke datoteke.
  - Prvih 12 kaže na *direktne bloke*.
  - Naslednji trije kažejo na *indirektne bloke*
    - Prvi kaže na *single indirect block* — indeksni blok, ki vsebuje naslove blokov s podatki.
    - Drugi je *double-indirect-block pointe*r, naslov bloka, ki vsebuje kazalce na bloke, v katerih so kazalci na podatkovne bloke.
    - Zadnji je *triple indirect* pointer, ki pa ni potreben; datoteke velikosti do 2^{32} bajtov nimajo dovolj blokov, da bi bil ta kazalec potreben; odmik v datoteki je 32-biten.

Direktoriji

- I-vozli imajo polje, ki določa ali gre za datoteko ali direktorij.
- Vnosi v direktorijih so spremenljive dolžine (različno dolga imena datotek); vsak vnos vsebuje najprej dolžino vnosa, ime datoteke in številko ustreznega i-vozla.
- Uporabnik dostopa do datotek preko njihovih imen (in poti), s sistemskega vidika je datoteka definirana z i-vozlom.
  - Jedro mora pot in ime datoteke preslikati v ustrezen i-vozel.
  - Preslikava poteka preko direktorijev.

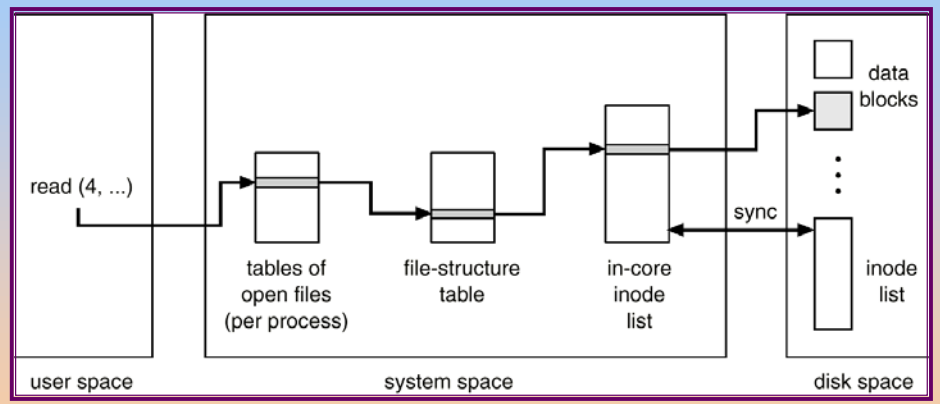
Direktoriji

- Najprej se določi začetni direktorij:
  - Če je prvi znak v poti "/", je začetni direktorij korenski direktorij.
  - Za vse druge začetne znake je začetni direktorij trenutni direktorij.
- Postopek iskanja se nadaljuje dokler ni dosežen konec poti in pridemo do ustreznega i-vozla.
- Ko pridemo do i-vozla, se alocira datotečna struktura, ki kaže na ta i-vozel.
- 4.3BSD za izboljšanje zmogljivosti datotečnega sistema uporablja predpomnjenje zadnjih nekaj parov pot – i-vozel.

Preslikava datotečnega deskriptorja

- Sistemski klici za odpiranje datotek kot parameter potrebujejo (zahtevajo) datotečni deskriptor.
- Jedro uporablja datotečni deskriptor kot indeks v tabeli odprtih datotek **trenutnega procesa**.
- Vsak vnos v tabeli vsebuje kazalec na datotečno strukturo.
- Datotečna struktura kaže na i-vozel.
- Ker je tabela odprtih datotek končno dolga (dolžina se določi ob zagonu sistema), je število sočasno odprtih datotek v sistemu omejeno.

Strukture datotečnega sistema



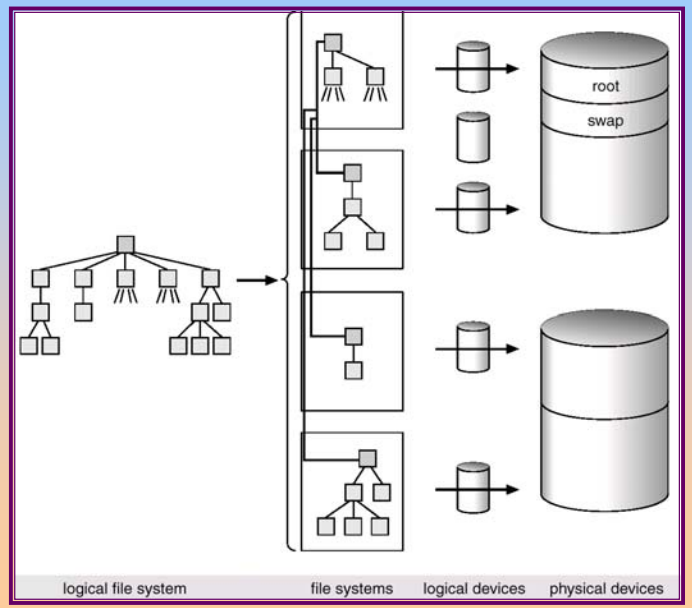
Strukture na disku

- Enotni datotečni sistem, ki ga vidi uporabnik, lahko sestoji iz večih fizičnih datotečnih sistemov, vsak na svoji napravi.
- Razdelitev fizične naprave na več datotečnih sistemov prinaša več koristi:
  - Različni sistemi lahko podpirajo različno uporabo.
  - Poveča se zanesljivost
  - Možno je izboljšati učinkovitost z nastavitvijo različnih parametrov.
  - Program ne more zasesti vsega prostora z eno veliko datoteko…
  - Hitrejše arhiviranje in obnavljanje particij, večja preglednost arhiva.

Strukture na disku

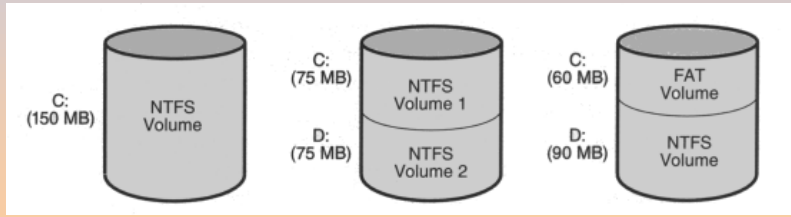
- *Korenski datotečni* sistem je vedno dostopen na določeni napravi.
- Druge datotečne sisteme je mogoče *priklopiti* — integrirati v hierarhijo direktorijev korenskega datotečnega sistema.
- Jedro za identifikacijo datotek uporablja par a <*št. logične naprave, št. i-vozla*>.
  - Št. Logične naprave določa datotečni sistem.
  - i-vozli so številčeni zaporedno….
- Naslednja slika prikazuje kako je struktura direktorijev razdeljena v datotečne sisteme, ki se nahajajo na logičnih napravah, ki so particije fizičnih naprav.

Preslikava datotečnega sistema na fizične naprave



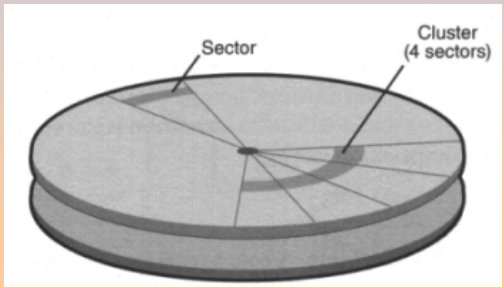
NTFS

- Osnovna struktura datotečnega sistema NTFS je zvezek
  - Ustvari se ga z orodjem *XP disk administrator utility*.
  - Temelji na particijah.
  - Lahko zaseda del diska, cel disk, lahko pa tudi več diskov.
- Vsi *metapodatki* (metadata), kot so podatki o zvezku, so shranjeni v običajnih datotekah.



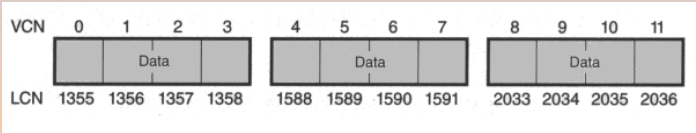
NTFS

- NTFS kot osnovno enoto alokacije prostora na disku uporablja *grozde* (clusters).
  - Grozd je sestavljen iz več (potenca 2) sektorjev.
  - Velikost grozda se določi ob formatiranju NTFS in znaša od enega do 8 sektorjev (0.5 – 4 KB). Notranja fragmentacija je zato razmeroma majhna (v primerjavi z npr. FAT-16).



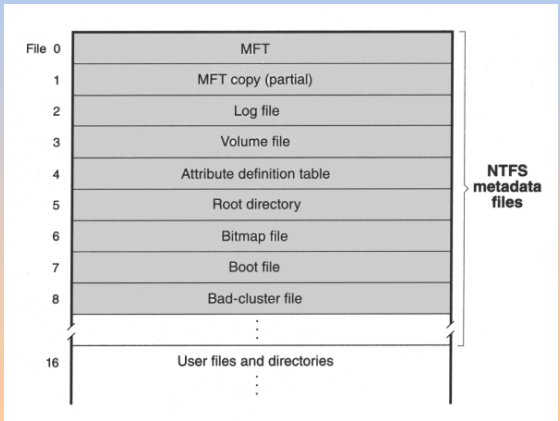
NTFS

- Do podatkov v posameznih datotekah pridemo preko takoimenovanih VCN (Virtual Cluster Numbers), ki se za vsako posamezno datoteko štejejo od 0 naprej.
- VCN na disku niso nujno zvezni, preslikujejo se v skupine zveznih LCN.



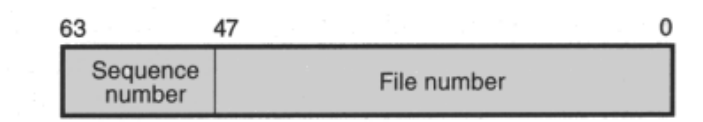
NTFS

- Srce strukture NTFS v zvezku (na disku) je glavna tabela datotek (**MFT**, Master File table).
- Poleg tabele MFT je na zvezku še množica metadata datotek.



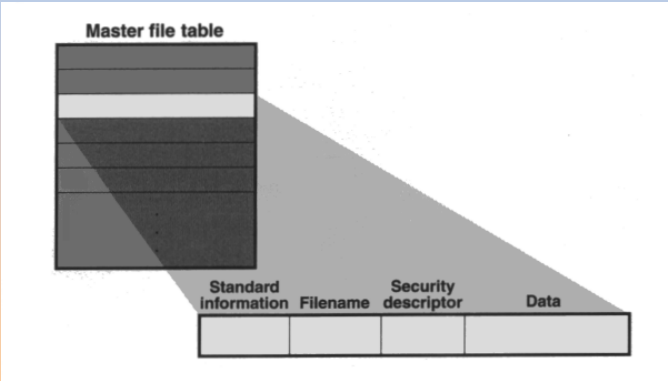
NTFS

- Datoteko v zvezku NTFS določa 64-bitna referenca datoteke. Ta vsebuje številko datoteke in sekvenčno številko. Številka datoteke je ustreza poziciji datotečnega zapisa v MFT (-1).
- Sekvenčno število se poveča pri vsaki uporabi datotečnega zapisa in ga uporabljamo za konsistenčne teste.



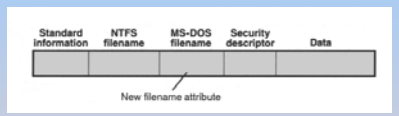
NTFS

- Vsak datotečni atribut pomnimo kot ločen tok bajtov v datotekah. Strogo rečeno, **NTFS** ne bere ali piše datotek, **bere ali piše tokove atributov**.

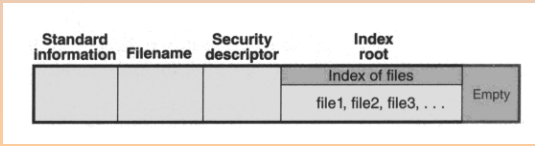


NTFS

- MS DOS imena datotek so shranjena v istem datotečnem zapisu kot dolga imena datotek

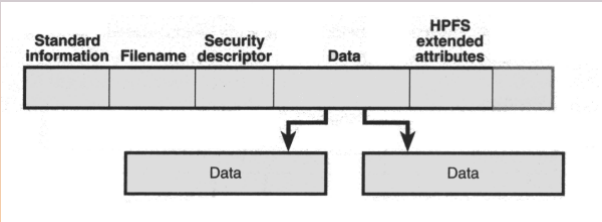


- Če je vebina nekega atributa pomnjena kar v datotečnem zapisu, je to **rezidenčni atribut**.
- Če je datoteka majhna, pomnimo vse atribute in tudi njihove vrednosti (podatke) v datotečnem zapisu. Dostop do (zelo) kratkih datotek je zato hiter.



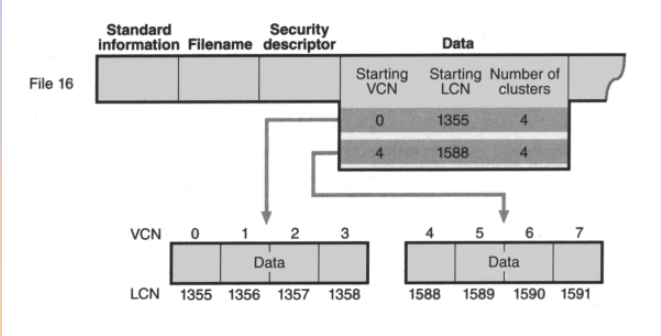
NTFS

- Za atribute z dolgimi vrednostmi (na primer za datoteke z več podatki) uvede NTFS izven tabele MFT nekj kB velik podaljšek (pravimo mu tudi **extent**, run), v katerem pomni vrednost atributa (v našem primeru podatke datoteke).
- Atributi, katerih vrednosti so pomnjene izven MFT, so **nerezidenčni atributi**.



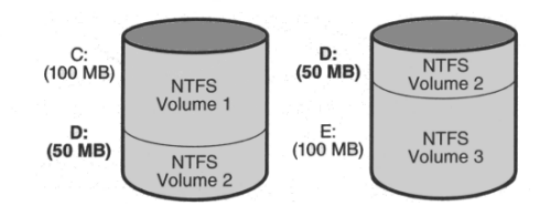
NTFS

- Slika prikazuje, kako zaglavje podatkovnega atributa hrani preslikavo med virtualnimi in logičnimi številkami grozdov.



NTFS

- **Volume set** Volume set je en logični zvezek, ki ga sestavlja do 32 področij prostora na enem ali več diskih.
- Z orodjem "Windows NT Disk Administrator" združimo ta področja v enoten volume set. in ga formatiramo za izbrani datotečni sistem.

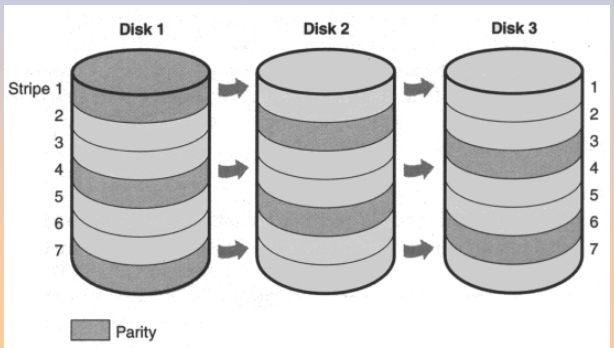


NTFS

- **Stripe set** (stripe = proga) je zaporedje več particij (vsaka particija na svojem disku), ki skupaj predstavljajo en logični zvezek. Vse particije morajo biti enako velike.

NTFS

- Stripe set s parnostjo (=RAID5) zagotavlja tolerantnost do napak tako, da rezervira ekvivalent enega diska za vsak stripe.
- Možno je tudi zrcaljenje (RAID1)



NTFS – stiskanje podatkov

- Za stisnjene datoteke, NTFS razdeli podatke datotek v enote - *compression units*, ki so bloki 16 zaporednih grozdov.
- Za ‘redke’ datoteke, NTFS uporablja drugačno tehniko.
  - Grozdi, v katerih so same ničle se v resnici ne zapišejo na disk.
  - Namesto tega se v (običajno zveznem) zaporedju VCNje pusti presledke...
  - Če NTFS pri branju datoteke zazna presledek med VCNji, NTFS v uporabnikov medpomnilnik vpiše ustrezno število ničel.