KAZALO

1.UVOD

To gradivo naj bi bilo zgoščen pripomoček za uporabnike operacijskega sistema UNIX, pri čemer so nujno opuščene številne podrobnosti, ki jih zasledimo v obširni dokumentaciji, ki sodi v opremo računalnika s tem operacijskim sistemom. Tako je za UNIX značilna cela vrsta uslužnostnih programov (utilities), ki jih povprečen uporabnik morda nikoli ne bo uporabil. Po zgledu nekaterih knjig se tudi v tem gradivu omejujemo na osnovna orodja. Za boljše razumevanje so v gradivu podani tudi osnovni koncepti sistema UNIX.

Knjiga predpostavlja, da ima bralec že določeno računalniško predznanje. Zaželeno je poznavanje popularnega operacijskega sistema DOS, saj so določeni pojmi pri obeh sistemih enaki ali podobni.

Prvi del knjige bralca takoj pouči z osnovnimi značilnostmi sistema UNIX in ga seznani z osnovnimi ukazi, ki že omogočajo uporabo sistema. V nadaljevanju so po abecedi nanizani in opisani standardni ukazi oziroma orodja, potrebna za interaktivno delo. Posebej pa je opisan urejevalnik teksta vi, ki je standarden pripomoček, namenjen pisanju programov in drugih tekstovnih datotek.

Sledijo poglavja, namenjena bolj zahtevnim uporabnikom. Bralec se seznani s pisanjem ukaznih datotek, ki pri UNIX lahko predstavljajo že kar prave programe.

Programerji malo bolj zahtevnih programov v jeziku C morajo tudi poznati interakcijo teh programov z operacijskim sistemom. V naslednjih poglavjih se zato seznanijo z ozadjem operacijskega sistema in z osnovnimi sistemskimi klici.

Dogovor:

Imena tipk,ukazov in drugih elementov bodo pisana *kurzivno* (na primer *ESC, backspace, copy*).

Tudi pogovor z računalnikom bo pisan *kurzivno*. Pri tem bodo izpisi računalnika pisani *nepoudarjeno*, tekst uporabnika pa ***poudarjeno***.

Prva uporaba nekega pojma (imena ukaza ali podobno) bo nakazana s **poudarjenim** izpisom.

Pripombe na pripravljeno gradivo in opozorila na (zanesljivo prisotne) napake so zelo zaželeni.

2.OSNOVNA NAVODILA UPORABNIKU

Pomembno opozorilo:

Pri tipkanju moramo biti pozorni na pravilno rabo malih oziroma velikih črk, ker lupina oba tipa razlikuje. Če se zmotimo, uporabljamo tipko ***backspace***.

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

login:

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

password:

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

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

Delo z računalnikom zaključimo z vtipkanjem ukaza

$ ***exit***

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

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

Primer:
*$ **lp pismo***

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

Če ukazno vrstico zaključimo z znakom ***&*** pred tipko *ENTER*, pomeni to zahtevek, da lupina sicer sproži izvajanje podanega ukaza (oziroma njemu ustreznega procesa), vendar se tudi takoj spet oglasi z najavko in čaka na nov ukaz. Prvi ukaz bo torej izvajan paralelno, pravzaprav v ***ozadju*** (background) naše interakcije z lupino.

Primer:
*$ **lp pismo &***
1234
$

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

Primer:
*$ **lp -m pismo***

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

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

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

Primer:
*$ **lp pismo; ls ***

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

Pogosto se zgodi, da želimo nek program predčasno prekiniti. To dosežemo s prekinitveno tipko, ki je običajno kombinacija ***CTRL/C***, lahko pa je v ta namen

uporabljena tudi tipka ***delete*** ali ***break*** ali ***rubout***. Proces, ki teče v ozadju, pa prekinemo z ukazom ***kill***, pri čemer moramo navesti že omenjeno oznako *PID* danega procesa.

Primer:
$ kill 1234

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

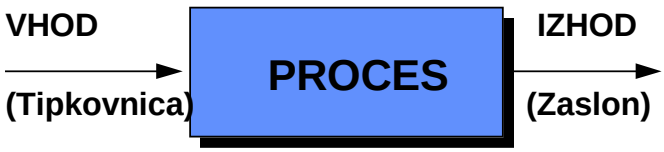
ukaz A1 A2 .. An

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

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

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

standardni vhod .....tipkovnica
standardni izhod .....zaslon
zapis napak.....zaslon



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

ukaz A1 A2 ..An <podatki

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

ukaz A1 A2 .. An > izhod

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

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

ukaz A1 A2 .. An 2> napake

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

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

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

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

3.DATOTEČNI SISTEM UNIX

UNIX pozna naslednje tipe datotek:

- navadne datoteke,
- direktorije,
- posebne datoteke,
- cevi (pipes).

Navadna datoteka je zbirka znakov, pomnjena na disku. Lahko vsebuje tekst, zapis programske kode ali kakšno drugo informacijo.

Direktorij je v bistvu datoteka s seznamom datotek in poddirektorijev, ki so zbrani skupaj z določenim razlogom (poseben projekt, lastnik ali kaj podobnega).

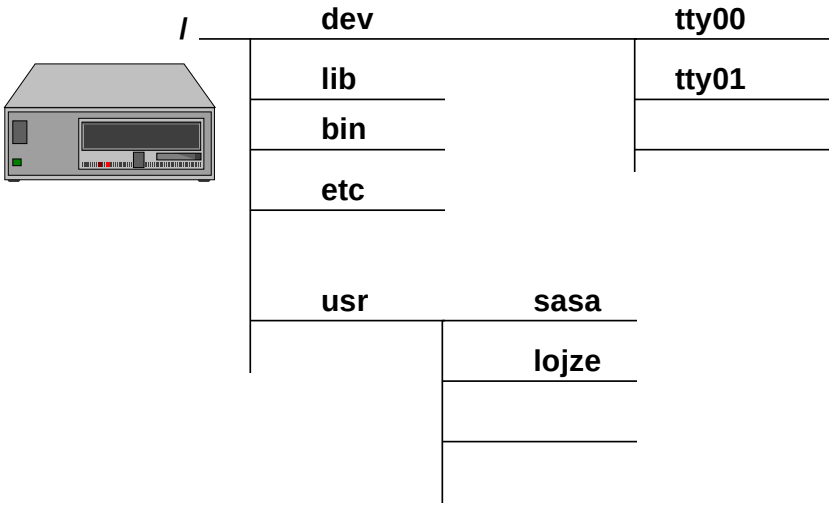
Posebna datoteka predstavlja fizično napravo, kot je na primer terminal. To je posebnost sistema UNIX, ki obravnava naprave na enak način kot datoteke. To omogoča uporabo enakih ukazov tako za datoteke kot za periferne naprave.

Cevi (*pipes*) so "datoteke" tipa FIFO (First In First Out), ki jih lahko uporabljamo za komunikacijo med pari programskih procesov.

Skupek datotek na nekem pomnilnem mediju je organiziran v ***datotečni sistem***.

Datotečni sistem je hierarhično zasnovan. Vsak direktorij vsebuje imena datotek in morebitnih poddirektorijev. Na vrhu drevesa direktorijev je ***osnovni direktorij*** (*root directory*), ki ga pri sistemu UNIX označujemo z znakom */*.

UNIX že sam po sebi predvideva določeno organiziranost drevesa poddirektorijev, ki je podobna naslednjemu primeru:



Vsak aktivni uporabnik sistema UNIX ima svoj ***domači direktorij*** (*HOME directory*), v katerega ga UNIX namesti avtomatično ob vstopu. Kasneje lahko uporabnik prehaja v druge direktorije. Trenutnemu direktoriju, v katerem se nahaja, pravimo ***delovni direktorij*** (*current work directory*). Tega lahko med delom spreminja z ukazom ***cd*** (change directory).

Do neke datoteke lahko pridemo, če poznamo ***pot*** (*path*) do nje. Ime poti je lahko ali ***absolutno*** ali ***relativno***. V prvem primeru je ime sestavljeno iz niza, ki ga začne oznaka osnovnega direktorija (*/*), tej pa sledi veriga imen poddirektorijev, ki se končno zaključi z imenom iskane datoteke. Ločilo med temi imeni je spet znak ***/.***

Primer:
/usr/janez/dopisi/pismo

Ime datoteke *pismo* leži v direktoriju *dopisi*, ki je poddirektorij direktorija *janez*. *janez* je poddirektorij direktorija *usr*. Slednji je poddirektorij osnovnega direktorija */*.

Relativno ime poti vsebuje podobno verigo imen, ki predstavlja pot do iskane datoteke začenši v delovnem direktoriju.

Primer:
dopisi/pismo

V primeru, da je naš delovni direktorij *janez*, pridemo s tem relativnim imenom poti do iste datoteke *pismo*.

Kot že rečeno, pomeni lupini znak / napotek za osnovni (root) direktorij. Podobno imamo posebne oznake tudi za tekoči oziroma delovni (current, working) direktorij (znak **.**) in predhodni (parent) direktorij (znak **..**). Znak ~ pa je lupini napotek za domači oziroma vstopni (HOME) direktorij.

Opozorilo:
Imena posameznih datotek in direktorij lahko sestavimo s poljubno kombinacijo znakov razen z znakom /. Odsvetuje se še uporaba znakov *# $ & < > presledek, backspace* in *tab*, ter vseh kontrolnih znakov. Znaki **+ - .** naj ne bodo na prvem mestu imen.

Delo z datotečnim sistemom je opisano v naslednjem poglavju.

4. NEKAJ OSNOVNIH UKAZOV ZA INTERAKTIVNO DELO

V tem poglavju bomo nanizali ukaze, ki jih lahko uporabljamo pri delu z datotekami, razvoju programov in pri drugih aktivnostih, ki pridejo v poštev pri normalnem delu z računalnikom. Poglavje je urejeno glede na vrste operacij. Bralec naj ga razume kot pomensko kazalo. Podrobnosti za posamezne ukaze, pa tudi konkretne zglede pa zasledimo v posebnem preglednem poglavju, kjer so ukazi nanizani po abecedi.

4.1. Rokovanje z datotekami

Med najbolj osnovne operacije sodi prepisovanje in brisanje datotek. Na voljo imamo več variant. Tako ukaz ***cat*** (concatenate) omogoča prepisovanje standardnega vhoda na standardni izhod. To je torej univerzalen program, s katerim bodisi ustvarjamo nove datoteke, jih prepisujemo ali izpisujemo na zaslon itd.

Podobno funkcijo ima ukaz ***cp*** (copy), ki pa za svoje delo uporablja datoteke, navedene v argumentih. Omogoča tudi prepisovanje več datotek v nek direktorij.

Omenimo še ukaz ***mv*** (move), ki pa ne napravi kopije originalnih datotek temveč dejansko premakne imena datotek na nov direktorij. Premik imena znotraj istega direktorija pomeni v bistvu preimenovanje datoteke (brisanje starega imena in tvorba novega).

Datoteke lahko brišemo z ukazom ***rm*** (remove).

Tekstovne datoteke lahko prikažemo na zaslon tudi z ukazom ***more*** in izpišemo s tiskalnikom z ukazom ***lp*** (line printer).

Ena izmed osnovnih operacij je pisanje ali popravljanje tekstovnih datotek. V ta namen ima uporabnik na voljo več urejevalnikov.V okolju UNIX je standarden urejevalnik ***v***i (visual editor). Osnovna navodila za njegovo uporabo so podana v posebnem poglavju. Med uporabniki sistemov UNIX je tudi precej razširjen urejevalnik ***emacs***.

Uporabnik lahko svojim datotekam spreminja zaščite pred branjem, brisanjem in izvajanjem. Te zaščite lahko veljajo tako za posameznega uporabnika, za skupino (grupo) uporabnikov ali za vse. Spreminjamo jih z ukazom ***chmod***.

4.2. Delo z direktoriji

Tudi direktorije lahko tvorimo, brišemo, preimenujemo itd. Tako lahko tvorimo nov direktorij z ukazom ***mkdir*** (make directory), preimenujemo ga lahko z že omenjenim ukazom ***mv*** (move), brišemo pa ga z ukazom ***rmdir*** (remove directory). Nek direktorij lahko zbrišemo le, če je prazen.

Prikaz vsebine nekega direktorija dobimo z uporabo ukaza ***ls*** (list). Prehajanje med direktoriji omogoča ukaz ***cd*** (change directory), z ukazom ***pwd*** (print working directory) pa dobimo informacijo, v katerem direktoriju se trenutno nahajamo. Kaj rado se namreč zgodi, da pozabimo, kje smo.

4.3. Komunikacije

Ker je UNIX v zasnovi večuporabniški sistem, lahko uporabniki med seboj komunicirajo na različne načine. Tako imajo na voljo program ***mail***, ki omogoča oddajanje in sprejemanje sporočil, torej *elektronsko pošto* med uporabniki.

V nadaljevanju si bomo ogledali nekaj najbolj preprostih oblik komunikacije med uporabniki. Za primer vzemimo, da uporabljajo računalniški sistem uporabniki Peter, Janez, Micka in Lojze.

Janeza najprej zanima, kdo vse v danem trenutku dela na računalniku. To ugotovi z ukazom ***who*** ki ima naslednjo obliko:

$ ***who***

Sistem izpiše podatke o vseh trenutno priključenih uporabnikih na naslednji način:

peter tty05 09:25
janez tty06 12:33

Za vsakega uporabnika pove tudi oznako njegovega terminala in čas, ko je ta začel delo na računalniku.

Janez bi Petru lahko sporočil kratko ovestilo na naslednji način:

$ write peter
Zdravo Peter
Sporoci mi svoje mnenje o predlaganem projektu
< CTRL/D >
$

V prvi vrstici je Janez z ukazom ***write*** nakazal, da hoče prenesti obvestilo. Druga beseda pa je ime uporabnika obvestila.

Sledi ena ali več vrstic z obvestilom. Obvestilo zaključimo skombinacijo tipk ***CTRL*** in ***D***.

Takoj zatem se na zaslonu Janeza pojavi najavna značka njegove lupine (znak ***$***) in lahko nadaljuje s svojim delom. Hkrati pa se na Petrovem terminalu pojavi naslednje obvestilo:

Message from janez tty06 [wed Jun 11 14:15:11]..
Zdravo Peter
Sporoci mi svoje mnenje o predlaganem projektu
EOF

Obvestilo torej vsebuje glavo, ki pove ime pošiljatelja, oznako terminala, na katerem ta dela ter trenutek oddaje obvestila. Temu sledi podatkovni del obvestila.

Neugodnost take komunikacije je, da se naslovniku (v našem primeru Petru) pojavi obvestilo na zaslonu sredi njegovega normalnega dela (na primer med urejanjem nekega teksta) in ga lahko zbega. Pred takimi, včasih nezaželenimi motilnimi obvestili se zasčitimo z ukazom

$ mesg n

Pošiljatelj (na primer Janez) bi v takem primeru dobil pri poskusu komunikacije (s Petrom) obvestilo:

Permission denied

Obratno funkcijo od ukaza ***mesg n i***ma ukaz ***mesg y.***

Vzemimo, da želi Peter odgovoriti Janezu. če je obvestilo, ki ga želi sporočiti, predolgo, oziroma ga že ima zapisanega v datoteki na primer z imenom *odgovor*, prenese ta tekst Janezu na naslednji način:

$ write janez <odgovor

Če oba uporabnika uporabljata ukaz ***write***, je s tem možen njun pogovor.

Vsekakor pa je za posredovanje daljših obvestil bolj primerna uporaba pošte oziroma ukaza ***mail***.

Uporaba elektronske pošte z ukazom *mail* ima tudi druge prednosti. Tako ni nujno, da sta na računalniku prisotna oba uporabnika istočasno. če ne prej, bo naslovnik sprejel obvestilo o pošti takoj, ko bo vstopil v sistem.

Samo posredovanje pošte je podobno kot pri ukazu *write* in ima na primer naslednjo obliko:

$ ***mail peter***
Zdravo Peter
sprejel sem tvoj odgovor in ga bom upostevaI
<CTRL/D>
$

Peter bo na svojem zaslonu ob prvi priliki dobil sporočilo "*You have mail*". Sprejemana pošta se pri posameznem uporabniku nabira v posebni datoteki, ki služi kot *poštni nabiralnik* (mailbox).

V nadaljevanju bomo spoznali, kako lahko že omenjeni program *mail* uporabimo za branje pošte, njeno shranjevanje, ponovno odgovarjanje in podobno.

Če preprosto pokličemo program *mail* z ukazom

$ mail

bomo dobili odgovor " *No mail*", če nas ne čaka nobena pošta. Vnašem primeru pa bi Peter imel na zaslonu naslednji zapis:

$ You have mail
$ ***mail***
From janez Wed Jun 11 13:58:22 1990
Zdravo Peter
sprejel sem tvoj odgovor in ga bom upošteval.
?

Zadnji znak ***?*** je pravzaprav *najavka* (prompt) programa *mail*, ki omogoča interaktivno delo. če programa mail ne poznamo, enostavno vtipkamo še en vprašaj in na zaslonu se bodo pojavile možnosti, ki jih kot uporabnik imamo na voljo. Na primer:

*?****?***
q quit
x exit without changing mail
p print
s [file] save (default mbox)
w [file] same without header
-- print previous
d delete
+ next (no delete)
m user mail to user

! cmd
?
Vidimo, da lahko obvestilo, ki smo ga prebrali, zbrišemo, izpišemo s tiskalnikom, lahko ga shranimo (z glavo ali brez) v ustrezno datoteko. Ukaz ***m*** (*mail*) omogoča tudi takojšnje pošiljanje pošte drugim uporabnikom.

Primer:
*?**fo micka lojze***

Sprejeto pošto smo posredovali še uporabnikoma Micki in Lojzetu. Ukaz ***fo*** pomeni *"forward"*.

Seveda se je lahko v "poštnem nabiralniku" nabralo tudi več obvestil. Z ukazom + lahko "odpiramo" obvestila drugo za drugim, jih shranjujemo, mečemo v koš , pošiljamo naprej in podobno.

Iz programa *mail* izstopimo z ukazom ***q*** ali ***x***. V drugem primeru se nam sprejeta pošta ne izbriše iz nabiralnika, čeprav smo (morda pomotoma) to prej zahtevali (z ukazom ***d***).

Običajno ima uporabnik na voljo tudi izpopolnjene poštne programe (na primer mailx, elm), ki pa jih v tem kratkem pregledu sistema UNIX ne bomo obravnavali. Uporaba teh programov se lahko v podrobnostih razlikuje, v osnovi pa je podobna navedeni uporabi programa *mail*.

5.UREJEVALNIK vi

vi je zaslonsko usmerjen urejevalnik teksta. Poženemo ga podobno, kot vrsto drugih urejevalnikov.

Primeri:
$vi
$vi file1

V prvem primeru vstopimo direktno v urejevalnik, v drugem pa le-ta odpre vhodno datoteko *file1* in prikaže na zaslon prvih nekaj vrstic.

Iz urejevalnika izstopimo z ukazom *" quit"*, ki ima naslednjo obliko

:q!

Urejevalnik *vi* ne spreminja direktno urejane datoteke. Napravi si kopijo v začasnem polju (buffer). Besedilo shranimo v imenovano datoteko šele z ukazom " write", ki ima naslednjo obliko

:w file1

Samo delo z urejevalnikom *vi* je novincu, ki je domač v okolju MSDOS, nenavadno. Zato se bomo omejili le na nekaj najbolj osnovnih ukazov. Tako velja opozoriti, da loči *vi* dva režima delovanja: **ukazni** in **vnašalni** režim (command mode, insert mode). V ukaznem pomeni pritisk na posamezne tipke ustrezno akcijo, v vnašalnem pa vnašamo besedilo. Prehod iz ukaznega v vnašalni režim lahko sprožimo z vtipkanjem ukaza ***i*** (insert) ali ***a*** (insert after cursor). Možni so tudi nekateri drugi ukazi za prehod v vnašalni režim. V ukazni režim se vrnemo s tipko ***ESC***.

Pogosto uporabljane ukaze podamo z eno tipko (kot na primer ***i***) ali pa s kombiniranjem dveh tipk (na primer *CNTRL* in *F*). Obstaja tudi skupina ukazov, ki jih moramo zaključiti s tipko ENTER . To so ukazi, ki jih začenjamo z znakom ***:*** (ali */*). Tak je na primer ukaz ***:q*** .

To, kar vidimo na zaslonu, je okno z vsebino medpomnilnika, v katerem hrani *vi* urejevano besedilo. Prazne vrstice na koncu datoteke so na zaslonu označene z znakom ~. Okno premikamo po medpomnilniku z naslednjimi ukazi:

CTRL/F	pomik (scroll) naprej za cel zaslon
CTRL/B	pomik nazaj za cel zaslon

Utripač (cursor) pomikamo po zaslonu z naslednjimi ukazi:

h	pomik v levo za en znak
j	pomik navzdol za eno vrstico
k	pomik navzgor za eno vrstico

l	pomik v desno za en znak
H	pomik v prvo vrstico na zaslonu
L	pomik v zadnjo vrstico na zaslonu
/niz	poišči naslednji nastop niza *niz*

Poleg teh osnovnih ukazov pozna *vi* še vrsto ukazov za pomike naprej oziroma nazaj do naslednje besede, stavka, paragrafa, do vrstice z dano tekočo številko ipd.

vi omogoča urejanje znakov in vrstic na zaslonu. Pregled važnejših tipk oziroma funkcij je naslednji:

x	brisanje znaka pod utripačem
X	brisanje znaka pred utripačem
dd	brisanje tekoče vrstice
rx	zamenjava znaka pod utripačem z znakom x
.	ponovitev zadnje urejevalnikove spremembe
i	prehod v vnašalni način (vrinjenje novega besedila pred znaka pod utripačem)
a	prehod v vnašalni način (vrivanje novega besedila za znak pod utripačem)

Vpisovanje novega teksta sprožimo s pritiskom tipke ***i*** in vstopimo v tekstovni način. V tekstovnem načinu lahko vpišemo oziroma vrinemo tudi po več vrstic. V ukazni režim prestopimo s pritiskom na tipko *ESC*.

Med urejanjem v ukaznem načinu imamo lahko do *vi* posebne zahtevke. Ti se začenjajo z dvopičjem in zaključujejo z *ENTER*. Pregled važnejših zahtevkov je naslednji:

:sh	izvajanje lupine (shell execution)
:r file	branje datoteke *file*
:w file	prepis teksta v datoteko *file*
:q	izstop (quit) brez pomnenja popravkov
:wq	prepis teksta v odprto datoteko in izstop

Uporabnik *vi* lahko prilagaja nastavitve urejevalnika svojim potrebam z ukazom *set* (ali okrajšano se). Tako z zahtevkom

:set all

dobimo prikaz trenutnih nastavitev. Z zahtevki

:set optionName
:set no optionName
:set optionName=value

lahko opcije setiramo, resetiramo ali (če je to predvideno) jim dodelimo potrebne vrednosti.

S temi zahtevki lahko na primer izberemo avtomatsko zamikanje vrstic, ignoriranje velikih in malih črk pri iskanju nizov, osveževanje zaslona med urejanjem, izbor tipa terminala (spremenljivka ***term***) itd.

Če je tip terminala neznan, tedaj urejevalnik *vi* ne more uporabljati zaslonsko usmerjenih funkcij.

Bolj izkušen uporabnik si lahko definira tudi nove funkcijske tipke z zahtevkom ***map***, ki ima naslednjo obliko

:map niz1 niz2

Pri tem je *niz1* niz, ki naj bi bil ekvivalenten zaporedju (nizu) tipk oziroma znakov *niz2*. *Niz1* ne sme biti daljši od 10 znakov, največkrat pa ga predstavlja le en znak. *Niz2* ima lahko do 100 znakov. S *CTRL/v* lahko v definicijo *map* vključimo znake, kot so *ENTER,ESC* in podobno.

Primer:
:map q :wqCTRL/v ENTER

V tem primeru tipka *q* skriva v sebi ukaz *:wq,* ki ga, kot je znano, zaključujemo z *ENTER*.

6. VAŽNEJŠI UKAZI UNIX

V nadaljevanju si bomo ogledali važnejše ukaze sistema UNIX. Zaradi preglednosti bodo le-ti razporejeni po abecednem vrstnem redu. Zaradi poenostavitve veljajo v podanem pregledu naslednji dogovori:

file	ime datoteke
file...	več imen datotek, ločenih s presledki
direktorij	ime direktorija, vključno s potjo do njega
options	opcijska stikala

V nekaterih primerih so kot argumenti podana tudi kakšna druga imena. Če kakšen argument ni obvezen, je podan v oglatem oklepaju.

Sami ukazi so podani v abecednem vrstnem redu. Nekateri ukazi so le informativno navedeni in si mora potrebno razlago poiskati uporabnik v ustrezni sistemski dokumentaciji.

at, batch ... Izvajanje ukazov v kasnejšem času

Nekaj oblik ukaza:
at time [date] [increment]
batch

Oba ukaza, ***at*** in ***batch***, bereta ukaze s standardnega vhoda. Ena od možnosti je, da standardni vhod preusmerimo na ukazno datoteko, v katero smo zapisali klice programov.

Ukaz *at* bo sprožil zahtevani program ob določenem času (time) oziroma datumu (date), lahko pa nekaj enot (minutes, hours, days, weeks, months ali years) za tem časom.

Ukaz *batch* pa bo sprožil zahtevani program takoj, ko bo računalnik dovolj razbremenjen.

Nekaj zgledov:
at now+1 day <file

Sistem bo sprožil ukazno datoteko *file* naslednji dan.

batch
lp pismo

Sistem bo zahteval izpis datoteke *pismo* na tiskalnik, ko sistem (ne tiskalnik!) ne bo več preobremenjen.

cat ... kopiranje in združevanje datotek

Oblika ukaza:

cat [-u][-s][-v][file]

cat bere datoteke, ki so navedene kot argumenti in jih zaporedno prepisuje na standardni izhod. Če ne navedemo nobene vhodne datoteke, bere *cat* s standarnega vhoda.

Osnovne opcije:

-u Znaki so posredovani v izhodno datoteko nemudoma, brez vmesnega pomnenja v medpomnilniku (buffer).

-s V primeru, da katera od navedenih vhodnih datotek ne obstaja, program ne izpiše obvestila o napaki.

-v Neizpisljivi znaki bodo prikazani v ustrezni vidni obliki.

Primeri:
cat file

Vsebina datoteke *file* bo prikazana na zaslonu.

cat file1 file2 >file3

Vsebina datoteke *file3* bo sestavljena iz zaporedja vsebin datotek *file1* in *file2.*

cat file1 > /dev/tty3

Vsebina datoteke *file1* bo izpisana na terminal *tty3.*

cc ... prevajalnik C

Oblika ukaza:
cc [options] file..

Prevajalnik obravnava navedene datoteke tako, da upošteva podaljšek, ki je dodan njihovemu imenu:

.c	Datoteko obravnava kot izvorni program v jeziku C
.o	Datoteka je objektni modul, ki naj ga obdela povezovalnik (linking loader)
.s	Datoteka vsebuje program v zbirnem jeziku

Če posebej ne zahtevamo drugače, požene prevajalnik še povezovalnik, ki tvori datoteko z izvršljivim programom.

Glavne opcije:
Nekatere od teh opcij uporabi direktno prevajalnik, druge posreduje povezovalniku.

-c	Prepreči povezovanje in tvori eno ali več objektnih datotek s podaljškom *.o*
-o ofile	Ime izvršljive izhodne datoteke bo *ofile* namesto privzetega (default) imena *a.out*.
-O	Aktivira optimizacijo tvorjene kode
-S	Izvede le prevajanje tako, da dobimo datoteko s programom v zbirnem jeziku s podaljškom imena *.s*
-E	Izvede le fazo predobdelave (preprocessing) in posreduje rezultat na standardni izhod.

Primeri:
cc main.c library.o

Prevede datoteko *main.c* in tvori izvršljiv modul, v katerega vključi tudi objektno datoteko *library.o*

cc prog.c -lx -o prog

Prevede datoteko *prog* in poišče klice zunanjih modulov z iskanjem le-teh v knjižnjici */lib/libx.a.* Na koncu sproži rezultirajoči izvršljivi program *prog.*

cc -E prog.c > file1.c

Izvrši predobdelavo in zapiše rezultirajoči izvorni program v datoteko *file1.c*

cd ... sprememba delovnega direktorija

Oblika ukaza:

cd [direktorij]

Opis:
cd omogoča prehod v poljubno lokacijo datotečnega sistema. Če ne navedemo nobenega argumenta, povzroči ukaz *cd* prehod na domači direktorij (home directory) oziroma na direktorij, naveden v spremenljivki *HOME*. Argument lahko vsebuje absolutno ime poti, če ga začenjamo z znakom /, ki označuje koren (root) datotečnega sistema. Če se ime poti ne začenja s tem znakom, predstavlja relativno pot, ki izhaja iz tekočega (delovnega) direktorija.

Primeri:
cd /usr/janez

Tekoči direktorij postane */usr/janez*. Pot do direktorija je bila podana v celoti izhajajoč iz osnovnega direktorija (root) datotečnega sistema.

cd dopisi/osebno

Pot do direktorija je bila podana relativno na tekoči direktorij. Če je ta bil na primer */usr/janez*, bo novi tekoči direktorij */usr/janez/dopisi/osebno*.

cd ..

Po drevesu direktorijev se dvignemo za en nivo. Če smo na primer bili v direktoriju */usr/janez/dopisi/osebno*, bo nov tekoči direktorij */usr/janez/dopisi.*

chgrp ... zamenjava lastniške skupine za datoteke ali direktorije

Oblika ukaza:
chgrp group file ...

Opis:
chgrp zamenja skupinski razpoznalnik (group *ID*) za vse navedene datoteke in direktorije. Operacijo lahko naredi le lastnik datotek oziroma direktorijev ali pa skrbnik sistema (superuser). Za novo skupinsko ime moramo uporabiti eno od imen, ki so navedena v datoteki */etc/group*.

Primer:
chgrp drugi

chmod ... sprememba zaščite datotek ali direktorijev

Oblike ukaza:
chmod mode file ..
chmod [who] [op] [permission..] file...

Opis:
chmod zamenja način (mode) zaščite (oziroma dovoljenj) navedenih datotek in direktorijev. Način lahko definiramo **absolutno** ali **simbolično**. V prvem primeru je to tromestno osmiško število. Prva številka določa dovoljenja lastniku, druga skupini, tretja pa velja za vse ostale uporabnike. V vsaki številki logično upoštevamo 1, če želimo dovoljenje za izvajanje (programov ipd.), 2 če dovolimo spreminjanje (zapisovanje) in 4, če dovolimo le branje datoteke ali direktorija.

Pri simbolični obliki pišemo namesto *who* (kdo) naslednje možnosti:

u	uporabnik
g	skupina (*group*)
o	drugi (*others*)
a	vsi (*all*)

op je lahko eden naslednjih operatorjev

+	dodamo tip dovolilnice
-	odvzamemo tip dovolilnice
=	natančno določimo dovoljenja

Tipi dovolilnic so naslednji:

r	branje (*read*)
w	pisanje (*write*)
x	izvajanje (*execution*)

Dovolilnice lahko spreminjata le lastnik datotek oziroma direktorijev ter skrbnik sistema.

Primeri:
chmod 640 file1

Lastnik datoteke *file1* jo lahko bere ali spreminja, člani njegove skupine jo lahko berejo, ostali pa do nje nimajo dostopa.

chmod 777 file2

Lastnik, člani skupine in vsi ostali lahko datoteko berejo, spreminjajo in izvajajo.

chmod ug+rw file3

Datoteki *file3* sta dodani dovolilnici za uporabnika in člane njegove skupine, ki lahko sedaj datoteko berejo in spreminjajo.

chmod +x file4

Datoteka *file4 (*program ali ukazna datoteka) postane izvršljiva.

chown ... zamenjava lastnika datoteke

Oblika ukaza:
chown owner file...

Opis:
chown zamenja lastnika datoteke ali direktorija. Operacijo lahko izvede le lastnik navedenih datotek ali skrbnik sistema. Novi lastniki so imena uporabnikov sistema, ki so navedena v datoteki */etc/passwd*.

Primer:
*chown janez **

S tem postaje *janez* lastnik vseh datotek v tekočem direktoriju.

cmp ... primerjava dveh datotek

Oblika ukaza:
cmp [-l][-s] file1 file2

Opis:
cmp primerja vsebini dveh navedenih datotek. V primeru različnosti pove številko zloga (byte) in vrstice, v kateri pride do razlike.

Opcije:
-l Izpis številke zloga (desetiško) in vsebine zlogov (osmiško) obeh datotek,

ki se razlikujejo.

-s Ne pošiljaj nobenih obvestil. Vrni ustrezno povratno kodo (exit code), ki je lahko uporabljena v ukazni (lupinini) proceduri;
0 za enaki datoteki, 1 za različni,
2 za primer nedostopnih ali manjkajočih datotek.

Primer:
cmp file1 file2

Izpiše ustrezno obvestilo v primeru razlik med datotekama *file1* in *file2*.

comm ... iskanje skupnih vrstic v dveh sortiranih datotekah

Oblika ukaza:
comm [-123] file1 file2

Opis:
Ukaz bere dve datoteki, *file1* in *file2* , ki morata biti sortirani. Rezultat ukaza je prikaz treh kolon. Prva vsebuje vrstice, ki nastopajo le v datoteki *file1* . Druga kolona vsebuje vrstice, ki nastopajo le v *file2*. Tretja kolona vsebuje vrstice, ki so prisotne tako v *file1* kot v *file2* .

Zastavice *1*, *2* in *3* preprečijo prikaz ustrezne kolone.

Primer:
comm -12 file1 file2

Prikazana bo le kolona z vrsticami, ki so skupne obema datotekama.

cp ... prepis datoteke

Oblike ukaza:
cp file1 file2
cp file1 [file2..] dirname

Opis:
Prva oblika ukaza povzroči prepis prvo navedene datoteke v drugoimenovano datoteko. Druga oblika ukaza povzroči prepis navedenih datotek v podan direktorij.

Primeri:
cp dopis1 dopis2 dopis3 dopisi

Datoteke *dopis1*, *dopis2* in *dopis3* se prekopirajo v poddirektorij *dopisi*.

Opozorilo:
Če ciljna datoteka z danim imenom že obstaja, bo njena vsebina zamenjana z novo. Datoteke ne moremo prepisati same v sebe.

cpio ... kopiranje datotečnih arhivov

Oblike ukaza:
cpio -o [acBvV] [-C bufsize] [[-O file] [-Mmessage]
cpio -i [BcdmrtuvVfsSb6k] [-C bufsize] [-I file] [-Mmessage]
cpio -p [adlmuvV] direktorij

Opis:
Arhivi datotek ASCII, ki jih tvorimo s ***cpio***, so prenosljivi med različnimi implementacijami sistema UNIX V. Poleg arhiviranja je ta ukaz torej primeren tudi za prenos datotek med sistemi.

Oblika *cpio -o* bere s standardnega vhoda seznam poti do datotek in te datoteke (skupaj s potjo do njih in z drugimi statusnimi informacijami) prepiše na standardni izhod.

Oblika *cpio -i* zahteva obratno operacijo. Iz (že prej formiranega) arhiva se restavrirajo datoteke in to le tiste, ki ustrezajo definiramemu vzorcu.

Kot je razvidno iz navedenih oblik ukaza *cpio*, imamo na voljo vrsto opcij. Pomen le-teh si lahko bralec ogleda v ustreznem sistemskem priročniku. Tu navedimo le nekaj zgledov:

ls|cpio -oc >../newfile

Seznam (listanih) datotek se prepiše na izhod, ki je preusmerjen (redirekcija z >) v datoteko *../newfile*. Opcija ***c*** zagotavlja prenosljivost te datoteke tudi na druge stroje (ASCII format glave).

cat newfile|cpio -icd "memo/a1" "memo/b\z"

cat posreduje vsebino (arhivske) datoteke *newfile* posreduje programu *cpio.* Ta izlušči datoteke, ki ustrezajo vzorcema *memo/a1* oziroma *memo/b*\z. Tvorijo se ustrezni poddirektoriji (opcija ***d***) pod tekočim direktorijem in vanje se vpišejo te datoteke.

cron ... izvrševanje časovno zamaknjenih ukazov

Oblika ukaza:
cron

Ta program normalno kliče sistemski proces *init* ob zagonu računalnika. ***cron*** spada med takoimenovane ***demonske*** procese. Bedi nad tem, kdaj je potrebno pognati programe, ki smo jih navedli z ukazi *at*, *batch* ali *crontab*.

crontab ... zahteva za izvajanje ukazov v regularnih intervalih

Oblike ukaza:
crontab [file]
crontab -r
crontab -l

Opis:
Ukaze lahko zapišemo v datoteko *file*. Za vsak ukaz, ki naj se izvaja v regularnih intervalih, uporabimo eno vrstico s šestimi podatki. Prvih pet (številk) definira interval (minute (0-59), ure (0-23), dan v mesecu (1-31), mesec v letu (1-12), dan v tednu (0-6)). zadnji podatek predstavlja ime ukaza. Podatki so ločeni s presledki. Namesto posameznih numeričnih podatkov lahko pišemo tudi zvezdico (pomeni za vse minute, ure itd) ali pa po dve številki, ločeni s pomišljajem (pomeni od - do).

Ukaz ***crontab*** s stikalom ***-r*** odstrani (*crontab*) datoteko *file* iz sistema, s stikalom ***-l*** pa zahtevamo listanje omenjene ***crontab*** datoteke.

csh ... interpreter ukazov s sintakso, podobno jeziku C

Oblika ukaza:
csh [options]

Opis:
csh je takoimenovana lupina, ki interpretira ukaze. Najprej izvede ukaze, ki so zapisani v ukazni datoteki ***.login***. Ko zahtevamo izstop z ukazom ***exit*,** izvrši pred izstopom še ukaze, ki so zapisani v ukazni datoteki ***.logout***.

Enostavni ukaz sestavlja zaporedje besed. Pri tem prva beseda določa ukaz, ki naj bo izveden. Zaporedje enostavnih ukazov lahko ločimo z znakom | in tako tvorimo ***cevovod*** (*pipeline*). (Standardni) izhod vsakega ukaza v cevovodu predstavlja (standardni) vhod v naslednji ukaz v cevovodu.

Ena vrstica lahko vsebuje tudi zaporedje več enostavnih ukazov ali cevovodov, ki so med seboj ločeni s podpičji. Taka zaporedja ukazov se izvajajo zaporedno.

Enostavne ukaze ali cevovode lahko izvajamo tudi v ozadju (in background), če vrstico s takimi ukazi zaključimo z znakom ***&***.

cu ... klic drugega sistema UNIX

Oblike ukaza:
cu [options] telno
cu [options] systemname

Opis:
S tem ukazom pokličemo drug operacijski sistem UNIX, krmilimo interaktivno konverzacijo in prenos datotek ASCII. Pri tem je *telno* številka telefona, *systemname* pa sistemsko ime UUCP. Podrobnosti o tem ukazu zasledimo v ustreznem sistemskem priročniku.

cut ... izločenje izbranih polj iz vrstic datoteke

Oblike ukaza:
cut -c list [file1, file2...]
cut -f list [-d char][-s][file1 file2..]

Opis:
S tem ukazom izločimo kolone v navedenih datotekah. Če datotek ne navedemo, bo program bral standardni vhod. Tako lahko *cut* uporabimo kot filter. *List* je seznam celih števil, ločenih z vejicami. Z znakom **-** med števili določamo območje.

Pomen posameznih opcij je naslednji:

clist Niz *list*, takoj za znakom *c* določa pozicije znakov, ki jih obdržimo v vsaki vrstici. Tako opcija -c7-72 pomeni, da obdržimo znake med sedmo in 72. kolono.

-flist S to opcijo izberemo polja, med katerimi je kot ločilo uporabljen znak *char*, definiran s stikalom *-d.*

-s Vrstice, ki ne vsebujejo delimiterja *char*, določenega z opcij*o -d,* ne bodo upoštevane.

date ... prikaz in sprememba datuma in časa

Oblike ukaza:
date [+format]
date MMDDhhmm[YY]

Opis:
Ukaz brez parametrov ali ukaz, pri katerem se parameter začne z znakom +, prikaže datum in uro v podanem formatu.
Format je podan z nizom, v katerem bodo posamezna polja, podana z znakom %, ki mu sledi posebna črka, nadomeščena z ustrezno trenunto vrednostjo.

Velja:

%d dan v mesecu
%m mesec
%y leto
%H ura
%S sekunda
%n preskok vrstice

Primer:
date '+DATUM: %d/%m/%y%n%URA: %H-%M-%S'

povzroči izpis datuma in časa v naslednji obliki:

DATUM: 30/09/66
URA: 12-33-18

Druga oblika ukaza je namenjena nastavitvi datuma in časa. Pri tem velja:

MM mesec (podan številčno)
DD dan dneva v mesecu
hh ura
mm minuta
YY leto

Primer:
date 10080645

Datum bo nastavljen na 8. oktober, čas pa na 6.45.

Ker je načeloma UNIX večuporabniški sistem, lahko datum spremeni le pooblaščeni uporabnik- skrbnik sistema (superuser).

devnm ... identifikacija imena naprave

Oblika ukaza:
/etc/devnm [names]

Opis:
Ukaz identificira posebne datoteke, ki so pridružene datotečnemu sistemu v navedenem direktoriju.
Primer:
Če je v direktoriju */usr* montirana */dev/hd1* (trdi disk), bo ukaz

/etc/devnm /usr

povzročil izpis

/usr/hd1

df ... Prikaz prostora na disku

Oblika ukaza:
df [-f] [-t] [-v] [fileSystem..]

Opis:
df ugotovi število prostih blokov in i-vozlov v navedenih datotečnih sistemih. Če ne navedemo noben datotečni sistem, poda *df* informacije o prostoru na vseh datotečnih sistemih, ki so trenutno montirani na sistem.

Opcije:

-f Poda le število prostih blokov
-t Poleg prostih blokov in i-vozlov poda tudi skupno (total) število blokov, ki so alocirani v vsakem datotečnem sistemu.
-v Procentualni izpis uporabljenih blokov ter števila zasedenih in prostih blokov.

Primeri:
df -t

Podaj število prostih blokov in ivozlov ter število alociranih blokov za vsak montirani datotečni sistem.

df /dev/rp01

Prikaži število prostih blokov in i-vozlov za */dev/rp01*.

diff ... primerjava dveh tekstovnih datotek

Oblika ukaza:
diff [-befh] file1 file2

Opis:
Ukaz povzroči primerjavo dveh datotek. Namesto enega od imen(*file1* oziroma *file2*) lahko pišemo pomišljaj. V tem primeru bo *diff* primerjal datoteko s standardnim vhodom. Če pomeni eno od imen direktorij, bo *diff* primerjal datoteko z istoimensko datoteko v tem direktoriju. Izstopna koda programa *diff* je 0 za primer enakih datotek, 1 v primeru različnih ter 2 v primeru napak (naprimer da ene od datotek ni).

Opcija *b* povzroči ignoriranje vodečih presledkov v vsaki vrstici.

dircmp ... primerjava direktorijev

Oblika ukaza:
dircmp [-d][-s][-wn] direktorij1 direktorij2

Opis:
Ukaz povzroči primerjanje dveh direktorijev. Izpišejo se imena datotek, ki nastopajo le v enem direktoriju. Izpiše se tudi, če imajo datoteke, ki nastopajo v obeh direktorijih, isto vsebino.

Opcija *-d* povzroči izvedbo operacije *diff* na vseh istoimenskih datotekah, ki niso identične. Navedimo pri tem, da je posebnost UNIXa tudi v tem, da ima lahko ista datoteka več imen. Opcija *-s* pa prepreči izpis datotek z identičnimi imeni.

Opcija *-wn* spremeni dolžino izpisne vrstice na *n* znakov.

du ... prikaz izkoriščenosti diska

Oblika ukaza:
du [-afrsu] [file1,..]

Opis:
du poda število zasedenih blokov za vsako datoteko, ki je nanizana med argumenti. Če smo navedli direktorij, pregleda *du* rekurzivno tudi vse poddirektorije. Če nismo navedli nobenega argumenta, pregleda *du* vse direktorije začenši z osnovnim (root) direktorijem.
Glavne opcije:
-s Zahtevamo le skupno število blokov, ki jih zasedajo navedene datoteke oziroma navedeni direktoriji.
-a Zahtevamo izpis števila zasedenih blokov za vsako datoteko v sklopu imenovanih direktorijev oziroma poddirektorijev.
-r Zahtevamo izpis obvestil o napakah, če katera od datotek ali poddirektorijev ne more biti pregledana (na primer da je zaščitena).

Primer:
du -s /usr/janez /usr/lojze

Prikaži globalno zasedenost datotečnega sistema z datotekami v poddirektorijih *janez* in *lojze.*

echo ... izpis argumentov na standardni izhod

Oblika ukaza:
echo [argument...]

Opis:
Argumenti bodo izpisani na standardni izhod. Med njimi bo presledek. Zaključeni so običajno s skokom v novo vrstico. Med argumenti lahko navajamo tudi posebne znake po zgledu na jezik C. Pomen najvažnejših posebnih znakov je naslednji:

\n skok v novo vrstico
\c izpis vtrsice brez prehoda v naslednjo vrstico
\t tabulatorski znak
\\ \
\num osmiška koda ASCII znaka

env ... nastavljanje okolja za izvajanje ukazov

Oblika ukaza:
env [-] [name=value]...[command [arguments..]]

Opis:
Ta ukaz omogoča prikaz in modifikacijo trenutnega okolja (environment). Zatem lahko sledi izvedba ukaza s spremenjenim okoljem. Spremembe okolja se dodajo stanju obstoječega okolja. Če pa uporabimo opcijski znak **-**, se bo navedeni ukaz izvedel ob upoštevanju navedenih argumentov, stanje okolja pred tem ukazom pa ne bo upoštevano.

Ukaz *env* brez kakršnegakoli argumenta povzroči le prikaz trenutnega okolja.

ex ... urejevalnik

Opis:
ex je urejevalnik, ki je nadgrajen urejevalniku ***ed***, njemu pa je nadgrajen urejevalnik ***vi.***

false ... vrne izstopno kodo, različno od 0

Oblika ukaza:
false

Ta ukaz uporabljamo v ukaznih datotekah.

find ... poišče datoteke, ki ustrezajo danemu kriteriju

Oblika ukaza:
find seznam_poti izraz

Opis:

Ukaz ***find*** poišče datoteke, ki ustrezajo navedenemu kriteriju. Rekurzivno pregleda poddirektorije za vsako pot v seznamu poti. Ta ukaz rabimo na primer, če iščemo datoteke, ki jih želimo brisati.

Primer:
find / -name core -atime +7 -exec rm {} \;

Ukaz briše vse datoteke *core* (te se formirajo pri nenormalnih prekinitvah programov), ki niso bile uporabljene zadnjih 7 dni (normalno se uporabljajo pri iskanju napak v takih programih).

finger ... Informacija o uporabnikih

Oblika ukaza:
finger [-bfilpqsw] [login_name ...]

Opis:
Ukaz ***finger*** izpiše najavno ime, polno ime, ime terminala in status vsakega imenovanega uporabnika. Pove še druge podatke, kot je na primer čas vstopa (login time) posameznih uporabnikov, lokacijo, morda telefonsko številko itd.

file ... razpoznava tipa datoteke

Oblike ukaza:
file [-m] file...
file [-m] -f file1

Opis:
file pregleda vsebino navedenih datotek in skuša iz te vsebine klasificirati njihov tip. Za ASCII datoteke skuša ugotoviti tudi jezik, v katerem so pisane.

Če je podana opcija *-f*, bo program *file* jemal imena datotek iz seznama, zapisanega v datoteki *file1*. Če smo uporabili opcijo *-m*, bo čas zadnjega dostopa do pregledovanih datotek ažuriran na tekoči čas, sicer pa bo ostal ta atribut datotek nespremenjen.

Primer:
*file * >pregled*

Ugotovi tipe datotek v tekočem direktoriju in izpiše ustrezno informacijo v datoteko pregled.

find ... iskanje datoteke

Oblika ukaza:
find pathName1 pathName2 .. izraz

Opis:
Rekurzivno pregleda vse navedene poti in išče datoteke, katerih ime zadošča regularnemu izrazu. Elemente v izrazu lahko kombiniramo z oklepaji, lahko jih negiramo (!), verižimo ali uporabljamo disjunkcijo (-o). Elementi v izrazu so lahko naslednji:

-name file1	Element je TRUE za neko datoteko z imenom *file1;*
-links n	Element je TRUE za datoteko ki ima *n* vezav (links);
-user userName	Element je TRUE za datoteko, katere lastnik je *userName;*
-size n	TRUE, če datoteko sestavlja *n* blokov;
-inum n	TRUE za datoteko z i-vozlom številka *n*;
-exec commandName	TRUE, če izvedba ukaza *commandName* vrne vrednost 0;
-ok commandName	Podobno kot v prejšnjem primeru, s tem, da je izvedba ukaza *commandName* pogojena z interaktivno potrditvijo uporabnika z *"y"*;
-print	Je vedno TRUE. povzroči izpis tekoče poti. Tako dobimo izpis vseh datotek, ki izpolnjujejo ostale pogoje v izrazu.
-newer file1	TRUE, če je obravnavana datoteka novejša od navedene datoteke *file1*.

ftp ... prepis datotek med različnimi računalniki

Oblika ukaza:
ftp [-v] [-d] [-I] [-n] [-g] [host]

Opis:
Ukaz ***ftp*** je Internetov ukaz za prepisovanje datotek na oddaljene računalnike ali iz njih. Sam *ftp* je v bistvu strežni program, ki omogoča komunikacijo z oddaljenim računalnikom. Običajno navedemo v ukazni vrstici tudi ime oddaljenega računalnika (*host*). V vsakem primeru se program javi z najavko

ftp>

Na to najavko moramo vpisati ustrezen zahtevek. Na voljo imamo več ukazov *ftp*. Najvažnejši so naslednji:

account [passwd]	Oddaljenemu računalniku posredujemo geslo
append local_file [remote_file]	Lokalno datoteko prilepimo oddaljeni datoteki
bye	Prekinemo delo z *ftp*
cd direktorij	Na oddaljenem računalniku menjamo direktorij
chmod mode file1	Oddaljeni datoteki menjamo zaščito
get remote_file [local_file]	Kopiramo datoteko z oddaljenega računalnika na naš lokalni računalnik
lcd [direktorij]	Menjamo direktorij na lokalnem računalniku
mdelete [remote_files]	Brisanje datotek na oddaljenem računalniku
mkdir dir_name	Tvori direktorij na oddaljenem računalniku
open host [port]	Vzpostavi komunikacijo z oddaljenim strežnikom ftp (če je že nisi prej vzpostavil z imenom v oddaljenega računalnika v ukazni vrstici ftp...)
put local_file [remote_file]	Prepis lokalne datoteke na oddaljeni računalnik
pwd	Izpis tekočega direktorija na oddaljenem računalniku
size file1	Izpis velikosti datoteke na oddaljenem računalniku

grep ... iskanje nizov in regularnih izrazov v neki datoteki

Oblika ukaza:
grep [-bchlnsvy] [-e izraz] [file...]

Opis:
grep pregleduje datoteke, ki so navedene v ukazni vrstici. Če nismo navedli nobene datoteke, pregleduje standardni vhod. V navedenih datotekah išče vrstice, ki zadoščajo podanemu izrazu. Te vrstice izpisuje na standardni izhod. Če zahtevamo pregled več datotek, izpiše pred vrsticami ime datoteke, kateri te vrstice pripadajo.

Glavne opcije:

-v Izpis vrstic, ki ne zadoščajo pogoju;
-c Izpis le števila vrstic, ki ustrezajo pogoju;
-l Zgolj izpis imen datotek, ki vsebujejo vrstice, ki zadoščajo pogoju oziroma izrazu;
-n Pred vsako izpisano vrstico bo dodana tekoča številka vrstice v datoteki;
-b Pred vsako izpisano vrstico bo izpisana številka bloka, v katerem je bila najdena;
-s Prepreči izpis opozoril o neobstoječih datotekah ali o datotekah z zaščito pred branjem.
-y Primerjava ne bo občutljiva na male črke-velike črke.

Opozorilo:
Pri uporabi metaznakov *$,~,[,{},|,(,) in* \ moramo biti previdni, ker imajo poseben pomen za samo lupino. Najbolje je zato izraz vstaviti med apostrofa (').

Primeri:
grep 'janez' dopis

Izpiše vse vrstice, ki vsebujejo niz *janez*.

grep -n '[uU]nix' f.~

Izpiše vse vrstice, ki vsebujejo izraz *unix* ali *Unix*. Pri tem pregleda vse datoteke, ki začenjajo s črko " f". Pred izpisom vsake vrstice izpiše še ime datoteke, Kateri pripada, ter tekočo številko vrstice v tej datoteki.

id ... izpis identifikatorjev in imena uporabnika in skupine

Oblika ukaza:
id

Opis:
Program izpiše uporabnikovo in gupno identifikacijsko število in ime za proces (normalno lupino), ki ta ukaz kliče.

init ... iniciacija operacijskega sistema

Oblika ukaza:
/etc/init [0123456SsQqabc]

Opis:
Glavna naloga tega ukaza je tvorba procesov na osnovi informacij, ki se nahajajo v datoteki */etc/inittab*.

V vsakem trenutku se računalnik nahaja na enem od osmih možnih ***nivojev delovanja*** (run level). Z ukazom *init* in s primernim parametrom lahko privilegirani uporabnik te nivije menjava. na voljo ima naslednje možnosti:

0	Zaustavitev operacijskega sistema (shut down)
1	Sistem naj preide v enouporabniški režim
2	Sistem naj preide v mnogouporabniški režim
3	Poženemo demonske procese in procese za *"remote file sharing"*
5	Zaustavitev operacijskega sistema, namenjeno serviserjem
6	Zaustavitev in ponoven zagon operacijskega sistema v skladu z informacijo v /etc/inittab
q,Q	Kontrola /etc/inittab
s,S	Prehod v enouporabniški režim

kill ... ukinjanje nekega procesa

Oblika ukaza:
kill [-signo] procesId...

Opis:
kill pošlje signal *15* (končaj) navedenim procesom. Številke tekočih procesov lahko dobimo z ukazom *ps*. Če navedemo kot številko procesa število *0,* bo zaključni signal poslan vsem procesom v procesni grupi. Uporabnik lahko ukinja le procese, katerih lastnik je. Vse procese pa lahko ubija le sistemski skrbnik .

Z ukazom *kill* lahko procesom pošiljamo tudi druge signale tako, da za pomišljajem navedemo številko signala (signo).
Zanesljivo ukinitev procesa dosežemo s klicem *kill -9....*

line ... branje ene vrstice

Oblika ukaza:
line

Opis:
Ukaz povzroči branje ene vrstice s standardnega vhoda in izpis tega na standardni izhod. Vrne izstopno kodo 1. Pogosto uporabljamo ta ukaz znotraj ukaznih datotek.

ln ... tvorba datotečne povezave (link)

Oblika ukaza:
ln [-f] file1 [file2...] cilj

Opis:
link je *povezava* (entry) v nekem direktoriju, ki navaja neko datoteko. Isto datoteko lahko navaja več povezav. Pri tem so imena te (iste) datoteke lahko tudi različna ali enaka (v različnih direktorijih).

Če je *cilj* direktorij, potem lahko eno ali več datotek povežemo na ta direktorij.

Opcija *-f* prepreči interaktivno potrjevanje navezovanja.

Ukazu ***ln*** je inverzen ukaz *rm*, ki briše eno od povezav na datoteko. Fizično brisanje datoteke nastopi šele, ko odstranimo z nje zadnjo *povezavo*.

lp ... tiskanje datotek " v ozadju"

Oblika ukaza:
lp [-c] [-r] [-m] [-n] [file ..]

Opis:
lp uvrsti v izpisno vrsto vsebino navedenih datotek. Če ne navedemo nobene datoteke, sledi tiskanje s standardnega vhoda.

Glavne opcije:

-m Po koncu tiskanja bo uporabnik dobil ustrezno obvestilo po poštnem servisu.
-c Program si naredi kopijo datotek, ki naj bodo posredovane tiskalniku. To je potrebno v primeru, da bi med čakanjem na izpis želeli original datotek spreminjati.
-r Briše datoteke, ki jih je tiskal.

Primer:
lp *.c

Tiska vsebine datotek z izvirnimi programi v jeziku C.

lpstat ... prikaz statusa tiskalnika

Oblika ukaza:
lpstat [opcije]

Opis:
Brez podanih opcij dobimo stanje zahtevkov za tiskanje, ki so jih dali uporabniki sistema.

ls ... prikaz seznama datotek v navedenem direktoriju

Oblika ukaza:
ls [opcije] [dirname]

Opis:
ls poda spisek datotek, ki so v navedenem direktoriju. Če tega direktorija ne navedemo, navede datoteke v tekočem direktoriju.

Glavne opcije:

-l Za vsako datoteko in direktorij bo podan razširjen opis. Tako bodo navedene zaščite, število vezi (links), lastnik in skupina, katerima datoteka pripada, velikost datoteke v bytih ter datum zadnje spremembe datoteke. Zaščite (oziroma dvolilnice) so podane s tremi nizi s po tremi znaki. Prvi niz pomeni dovolilnice uporabniku, drugi velja za skupino in tretji za vse ostale uporabnike. Vsak niz lahko sestavljajo znaki ***r*** (dovoljenje branja), ***w*** (dovoljenje spreminjanja) ter ***x*** (dovoljenje izvajanja). če katero od teh dovoljenj ne velja, je na njegovem mestu v nizu znak -. Pred to skupino devetih znakov je še znak, ki pove, ali je to navadna datoteka (znak **-**), direktorij (znak ***d***) ali posebna datoteka (znak ***c*** ali ***b***).

-p Poudari direktorije tako, da za imenom zapiše znak\.

-a Navedi tudi datoteke in direktorije, katerih ime začne s piko (**.**). Tak primer datoteke je *.profile*.

-uuuu Namesto datuma zadnje spremembe prikaži datum zadnje uporabe datoteke.

Primer:
ls -ap

Navedi vse datoteke in poddirektorije v tekočem direktoriju, tudi tiste, ki začenjajo s piko. Direktorije poudari z znakom \.

mail ... elektronska pošta

Opomba:
To orodje je bilo opisano v uvodnih poglavjih.

man ... interaktivni priročnik

Oblika ukaza:
man [-afbcw][-tproc][-ppager][-ddir][-Tterm][section][title]

Opis:

Z ukazom *man* zahtevamo izpis navodil , običajno o nekem ukazu. Ta program torej predstavlja interaktivni priročnik (manual pages). Bolj verzirani uporabniki si lahko tudi sami tvorijo svoje priročnike (na primer za neko razvito aplikacijo).

Pri uporami ukaza *man* je title ime poglavja, ki ga iščemo.

Primer:
man man

Dobili bomo napotke, kako uporabljamo interaktivni priročnik.

mesg ..dovoljenje ali prepoved sprejemanja obvestil na zaslon

Oblika ukaza:
mesg [y][n]

Opomba:
To orodje je bilo opisano v uvodnih poglavjih.

mkdev .. klic ukaznih datotek za dodajanje perifernih naprav

Oblika ukaza:
mkdev file1

S tem ukazom kličemo sistemske ukazne datoteke, ki dodajajo periferne naprave našemu datotečnemu sistemu. Podrobnosti se razlikujejo od sistema do sistema. Ta ukaz ni namenjen navadnim uporabnikom.

Primeri:
mkdev mouse
mkdev hd [[disk] [controller|adapter]][lun]

mkdir ... tvorba novega direktorija

Oblika ukaza:
mkdir [-m mode] directory

Opis:
mkdir tvori nove direktorije, katerih imena so predlagana kot argumenti ukaza. Za tvorjene direktorije veljajo vsa dovoljenja za lastnika, skupino in ostale uporabnike. Kasneje jih lahko omejimo z ukazom *chmod.* Lahko pa ta dovoljenja z opcijo ***-m*** določimo ob sami tvorbi direktorija.

Primer:
mkdir dopisi

Če smo bili v tekočem direktoriju */usr/janez*, je ta ukaz tvoril direktorij */usr/janez/dopisi.*

mkfs .. tvorba datotečnega sistema

Oblika ukaza:
mkfs [y|n][-f fstype] special blocks [:inodes][gap inblocks]

Opis:
S tem ukazom tvorimo na posebni datoteki *special* (oziroma ustrezni napravi) nov datotečni sistem.

mknod ... tvorba posebnih datotek

Oblike ukaza:

mknod name [c|b] major minor
mknod name p
mknod name s
mknod name m

Opis:
S tem ukazom odpremo ime v direktoriju ter ustrezni vozel za eno posebno datoteko. Pritem je *name* ime datoteke. Prvi primer velja za blokovne naprave(opcija b; diski, trakovi) ali za znakovne naprave (opcija c; terminali in druge naprave). Številčni podatek ***major*** definira tip naprave, podatek ***minor*** pa njeno številko.

Preostale tri oblike odpirajo vozel za cevi (pipes; opcija p), semaforje(s) oziroma skupen spomin (shared memory;opcija m).

more ... prikaz navedenih datotek

Oblika ukaza:
more [-d] [-f] [-l] [-vrstica] [+/pogoj] file1 ...

Opis:
more prikaže datoteko na zaslon in sicer v zaporednih skupinah po 22 vrstic. Med opcijskimi stikali lahko navedemo vrstico, pri kateri naj začne prikaz. Lahko navedemo tudi pogoj. V tem primeru se bo prikaz datoteke začel pri prvi vrstici, ki zadošča temu pogoju. Po izpisu 22 vrstic čaka *more* na naš ukaz. Vtipkamo lahko:

ENTER	Sledi prikaz naslednje vrstice;
presledek	Sledi prikaz naslednje skupine 22 vrstic;
ni	Prikazanih bo *n* naslednjih vrstic. Pri tem je *n* celoštevilčno število;
ns	Preskok *n* vrstic in prikaz naslednje skupine vrstice (naslednje ekranske slike);
nf	Preskok *n* skupin po 22 vrstic (ekranskih slik) in prikaz naslednje skupine;
q:	Izstop iz *more.*
=	Prikaz številke tekoče vrstice;
n/pogoj	Poišči *n*-ti natop izpolnjenega pogoja;
v:	Poženemo urejevalnik *vi* začenši s tekočo vrstico. Pri izstopu iz urejevalnika prevzame nadzor spet program *more*;
h:	Prikaz navodil;
!ukaz:	Izvede ukaz, ki je podan kot argument;
..	Ponovi izvedbo zadnjega ukaza;
i:n:	Prikaže *i*-to naslednjo datoteko , če je bilo v ukazni vrstici

navedenih več datotek.

mount ... montaža datotečne strukture v datotečni sistem

Oblika ukaza:
mount [-v][-r][-f fstyp] special device

Opis:
S tem ukazom povemo sistemu, da smo dodali prenosljiv medij (na primer disketo) in naj ta medij razume kot imenovano posebno datoteko (*special device*). Ta ukaz lahko uporabi le sistemski administrator. Opcija *-r* pove, da lahko medij le beremo.

mv ... premik datoteke in direktorijev

Oblike ukaza:
mv file1 file2
mv file1 [file2 ..] directory

Opis:
Prva oblika povzroči premik prvoimenovane datoteke v drugoimenovano. S tem se bistvu zamenja ime datoteke. Druga oblika premakne datoteke v drug direktorij in pri tem ohrani njihova imena. Razlika med *mv* in *cp* je v tem, da *mv* briše originalne datoteke.

Primer:
mv DOPIS dopis

Datoteka *DOPIS* se bo odslej imenovala *dopis.*

Opozorilo:
mv file1 file2

V primeru, da *file2* že obstaja, bo zaradi tega ukaza njegova stara vsebina izgubljena.

newgrp ... sprememba skupine (group) uporabniku

Oblika ukaza:
newgrp [-] group

Opis:
Uporabnik lahko s tem klicem zamenja skupino. Pri tem ostaja v istem delovnem direktoriju. Če je skupina ščitena z geslom (password), ga bo moral podati pred zamenjavo skupine.

news ... prikaz novic

Oblika ukaza:
news [-a][-n][-s][zadeva]

Ukaz naj bi uporabnika informiral o tekočih dogodkih. Brez navedbe argumentov bi dobili prikaz najnovejših novic Na voljo pa imamo še naslednje možnosti:

-a Izpisane bodo vse zadeve oziroma informacije
-n Izpisana modo le imena informacij
-s Izpisano bo le število novih informacij

nice ... sprememba prioritete izvajanja nekega ukaza

Oblika ukaza:
nice [-število] command [arguments]

Opis:
Ukaz ***nice*** krmili prednost uvrščanja (scheduling) ukaza *command [arguments]*. Vsakemu programskemu procesu je namreč dodeljena neka vrednost " nice", ki določa (obratnosorazmerno) prioriteto tega procesa. Če v sklopu ukaza *nice* podamo število, se bo to prištelo " vrednosti" nice danega ukaza.

Sistemski skrbnik lahko pospeši izvajanje svojih ukazov s tem, da jim dodaja negativne vrednosti (na primer -4).

Primer:
*nice -10 grep janez **

Ukaz *grep janez* * se bo izvajal z nizko prednostjo.

nl ... filter za številčenje vrstic

Oblika ukaza:
nl [opcije] file

Opis:
nl omogoča številčenje vrstic navedene datoteke. Če te ne navajamo, številči vrstice na standardnem vhodu. Pri tem deli sprejeti tekst na logične strani. Vsaka stran ima glavo, telo in konec. V vhodni datoteki označimo dele logične strani z vrsticami, ki vsebujejo (le) naslednje znake:

\: glava (header)
\:\: telo (body)
\: konec (footer)

Normalno se številčenje začenja na vsaki logični strani. številčimo lahko vse vrstice na strani, zgolj vrstice v glavi, v telesu ali v koncu, zgolj vrstice , ki vsebujejo definirani niz ipd.

Glavne opcije:

-p	Pri prehodu na novo stran se številčenje ne resetira;
-wnum	Pri tem je *num* celo število, ki pove, koliko znakov imajo tekoče številke v vrstici;

pack,pcat,unpack ... pakiranje in razpakiranje datoteke

Oblike ukaza:
pack [-] ime...
pcat ime...
unpack ime...

Opis:
pack komprimira navedene datoteke. Učinek stisnjenja zavisi od enakomernosti razporeditve bytov v izvorni datoteki. Pri pakiranju izvorna datoteka izgine, tvori pa se stisnjena datoteka, ki ima ime prejšnje, podaljšano z znaki " .z". Če določimo argument " -", bo program *pack* izpisal različne statistične podatke.

unpack ekspandira datoteke, ki so bile pakirane z ukazom *pack.* Razpakirana datoteka (z originalnim imenom *ime*) bo nadomestila njeno pakirano verzijo.

pcat ukaz ima podobno funkcijo kot ukaz *cat,* le da velja za izpis pakiranih datotek na standardni izhod. S tem ukazom lahko torej razpakiramo neko pakirano datoteko, ne da bi uničili njeno pakirano verzijo. Ukaz lahko uporabimo tudi za prikaz vsebine pakirane datoteke na zaslon.

Primeri:
*pack *.c*

Vse datoteke s programi v jeziku C bodo pakirane.

pcat file1

Datoteka z imenom *file1,* ki je bila pakirana v datoteko file1.z, bo prikazana na zaslonu.

pcat file1 >file2

Prej navedena datoteka bo razpakirana v datoteko z imenom file2.

passwd ... spremeni vstopno geslo

Oblika ukaza:
passwd geslo

Opis:
Vsak uporabnik lahko določi geslo, ki ga bo sistem zahteval pri vstopu. To geslo hrani sistem zakodirano v datoteki */etc/passwd.*

Primer:
passwd user12

V tem primeru bo novo geslo enako *user12*.

Opozorilo:
Sistemski skrbnik lahko spreminja datoteko ***/etc/passwd***, ki vsebuje gesla.

paste ... zlivanje vrstic navedenih datotek

Oblike ukaza:

paste file1 file2...
paste [-s] [-dlist] file1 file2...

Opis:
V prvih dveh oblikah dobimo zlivanje istoležnih vrstic v vhodnih datotekah *file1*, *file2*, ... Izhod zlivanja je usmerjen v standardni izhod. Za razliko od ukaza *cat*, s katerim lahko dosežemo vertikalno zlivanje datotek, dobimo s tem ukazom horizontalno zlivanje.

" Lepilo" med zlivanimi vrsticami je normalno tabulatorski znak, ki pa ga lahko zamenjamo z nizom, ki neposredno sledi stikalu *-d.*

ping ... preizkus delovanja oddaljenega računalnika

Oblika ukaza:
ping [-r] [-v] host [packetsize][pcount]

Opis:
Ukaz ping naj bi uporabljali pri testiranju računalniške mreže, meritvah in upravljanju. Pogosto ga uporabljamo zato, da vidimo, ali se nek oddaljeni računalnik javlja. Ker je ta ukaz precej potraten, ga raje ne uporabljajmo prepogosto.

pr ... prikaz datotek na standardni izhod

Oblika ukaza:
pr [opcije] [file..]

Opis:
pr lista vsebino navedenih datotek na standardni izhod, če pa teh ni, posreduje vsebino standardnega vhoda. Na vsaki strani bo glava z datumom in uro, imenom datoteke in številko strani.

Glavne opcije:

-p Zahtevamo prikaz strani na zaslonu, stran za stranjo. Za vsako stranjo čaka *pr* na tipko *ENTER*. Ta opcija velja, če je kot standardni izhod mišljen zaslon terminala.

-h" niz" S to možnostjo spremenimo glavo (header) strani s poljubnim nizom;

-n Vsaka vrstica teksta naj bo opremljena s tekočo številko;

-wn Ta opcija določi število znakov (*n*) v vrsticah in s tem širino strani;

-lk S tem določimo število vrstic (*k*) na strani;

-d Zahtevamo dvojni presledek med vrsticami.

-t Preprečimo izpis glave in prazne vrstice med posameznimi stranmi.

Primer:
pr -p -n -t -l23 file1

Prikaži vsebino datoteke *file1* brez glav posameznih strani. Vsaka stran ima 23 vrstic in vsaka vrstica naj bo oštevilčena. Po prikazu vsake strani počakaj na tipko *ENTER.*

ps ... prikaz stanja procesov

Oblika ukaza:
ps [opcije]

Opis:
ps tabelarično prikaže različne informacije o stanju aktivnih procesov v sistemu. V vsaki vrstici izpiše ime procesa, sicer pa je vsebina posameznih kolon tabele odvisna od glave tabele. Tako kolona *UID* vsebuje številčno oznako uporabnika (user identification number), *PID* je številčna oznaka procasa, *PPID* pa oznaka očeta danega procesa. *PRI* je nivo prednosti procesa, *TIME* podaja čas izvajanja procesa, *CMD* pa je ime procesa.

Glavne opcije:

-e Prikaži informacije za vse procese, ne samo za tiste, ki jih je aktiviral uporabnik, ki je sedaj vtipkal ukaz ps.

-f Prikaži vse možne informacije.

pwd ... izpis delovnega, tekočega direktorija

Oblika ukaza:
pwd

Opis:
pwd poda popolno pot do tekočega direktorija.

rm ... brisanje datoteke

Oblike ukaza:
rm [option] file ...
rm [option] dirname

Opis:
rm briše eno ali več imenovanih datotek ali celoten direktorij. *rm* zbriše navedene datoteke, če zanje obstaja le še ena *povezava* (link). Če pa je povezav več, zmanjša le te za imenovano povezavo iz tekočega direktorija.

Glavne opcije:

-f Če je katera od navedenih datotek zaščitena pred spreminjanjem, rm zahteva potrditev pred brisanjem, v kolikor ne navedemo opcije *-f*, ki povzroči brisanje brez potrjevanja.

-r Če je podana ta opcija, briše *rm* tudi navedene direktorije in rekurzivno vse datoteke in poddirektorije, ki jih imenovani direktorij vsebuje.

-i Zahtevamo interaktivno potrditev pred brisanjem posameznih datotek.

Primer:
rm -ri dopisi

Pregleda vse datoteke v poddirektoriju *dopisi* ali v njegovih poddirektorijih. Po morebitni potrditvi vsako datoteko ali poddirektorij zbriše.

rmdir ... brisanje direktorija

Oblika ukaza:
rmdir directory...

Opis:
rmdir zbriše navedene direktorije. Posamezne direktorije lahko tako zbrišemo le, če so prazni. Tako je ta ukaz manj nevaren kot ukaz *rm -r direktorij*.

sed ... stream editor

Oblika ukaza:
sed [-n][-e script][-f sfile][files]

Opis:
sed kopira imenovane datoteke (ali standardni vhod) na standardni izhod. Pri tem datoteke predela (edit) v skladu z zapisom ukazov *script*. Ob uporabi opcije *-f* bere te urejevalne ukaze z datoteke *sfile*. Opcija *-n* prepreči izhod. Zapis *script* vsebuje urejevalne ukaze naslednje oblike:
[naslov [,naslov]] funkcija [argumenti]

Posamezne urejevalne ukaze lahko ločimo s podpičjem. Normalno se pri izvedbi *sed* ciklično prepisujejo vrstice v vzorčni prostor. Izvedejo se vsi ukazi, ki s svojim naslovom izberejo ta vzorčni prostor. Končno se preurejena vsebina prepiše iz vzorčnega prostora na standardni izhod.

naslov je lahko decimalno število vhodne vrstice, znak *$*, ki predstavlja zadnjo vhodno vrstico ali nek " regularen izraz".

Urejevalni ukaz brez naslova velja za kakršenkoli vzorec. En naslov izbira le vrstice, ki temu naslovu ustrezajo. Dva naslova pa predstavljata območje ustrezajočih vzorcev.

sh ... klicanje interpreterja ukazov sh

Oblika ukaza:
sh

Opis:

Lupina ***sh*** predstavlja interpreter standardnega ukaznega jezika. Izvaja ukaze, ki jih tipkamo na terminalu ali ki jih bere z neke (ukazne) datoteke.

shutdown ... Zaustavitev operacijskega sistema

Oblika ukaza:
shutdown

Opis:
S tem ukazom zahtevamo varen zaključek vseh trenutno tekočih procesov.

sleep ... suspenz izvajanja za nek interval

Oblika ukaza:
sleep time

Opis:
sleep uporabljamo za proženje neke komande po določenem času ali v določenih intervalih. Taki uporabi ponazorujeta spodnja primera:

```
(sleep 105; command)&
while true
do
        command
        sleep 37
done
```

sort ... sortiranje in zlivanje datotek

Oblika ukaza:
sort [-cmu][+pos1][-pos2][-b][-r][-tch][-o file1] file...

Opis:
Izhod programa ***sort*** je ali standardni izhod ali datoteka, ki smo jo navedli za opcijskim stikalom *-o*. Če ne podamo nobenega argumenta, razvrsti *sort* standardni vhod v naraščajočem zaporedju. Pri tem za sortirni ključ uporablja celotno vrstico. Če navedemo neko datoteko, jo *sort* sortira. Če navedemo več datotek, jih zlije in sortira.

Sortiranje lahko poteka tudi po več ključih. Vsako vrstico lahko delimo na polja, pri čemer je ločilo med polji znak *ch*, podan za opcijskim stikalom *-t* .

Naj bodo *m1, n1, m2, n2* cela števila. Tedaj ukaz tipa

sort -c +m1.n1 -m2.n2 file1*

sortira datoteko *file1*. Vsako vrstico deli v polja, ločena z znakom *. Prvi znak sortirnega ključa dobimo s preskokom *m1* polj ter nato v polju *(m1+1) n2* znakov. Zadnji znak ključa je predhodnik znaka, ki ga dobimo s preskokom *m2* polj in nato *n2* znakov od začetka vrstice. Če opustimo lociranje zadnjega znaka v ključu, se smatra, da ključ poteka do konca vrstice.

Glavne opcije:

-c Izvede le kontrolo, če je datoteka sortirana.
-m Ob predpostavki, da so vhodne datoteke že urejene, jih *sort* le zlije v eno.
-u *sort* posreduje izhodu le en izvod vrstic, ki imajo enake ključe.
-b Ignoriranje vodečih presledkov pred posameznimi polji.
-r Sortiranje v obratnem vrstnem redu.
-tch *ch* je ločilni znak med posameznimi polji. Normalno je ta znak presledek.

Opciji *+pos1* in *-pos2* imata že omenjeno obliko *m.n*. Če podatek *.n* manjka, pomeni, da začenjamo s prvim znakom v polju *(m+1).*

Primeri:
sort +1 -2 file1

Sortirana bo datoteka *file1*. Pri tem je kot ključ uporabljeno drugo polje v vrsticah.

sort -r -o file3 +1.0 -1.2 file1 file2

Vsebini datotek *file1* in *file2* bosta zliti in sortirani v datoteko *file3* v obratnem vrstnem redu. Kot sortirni ključ se uporablja drugo polje *(m=1)* in sicer prvi znak *(n=0).*

split ... delitev datoteke na več delov

Oblika ukaza:
split [-n] [file1 [name]]

Opis:
split tvori množico datotek. Vsaka bo imela le *n* vrstic datoteke *file1*. Če *n* opustimo, bodo posamezne datoteke vsebovale po 1000 vrstic. Posamezne datoteke dobijo

imena *nameaa, nameab,nameac,... ,namezz*. Če imena datoteke ne podamo, bodo posamezne rezultirajoče datoteke imele imena *xaa, xab,... ,xzz*.

Primer:
split -100 janez

Dobili bomo datoteke *janezaa, janezab,..* ,dolge po 100 vrstic.

stty ... nastavitev opcij terminala

Oblika ukaza:
stty [-a][-g][opcije]

Opis:
Ukaz brez argumentov povzroči prikaz nastavitev važnejših opcij terminala. Uporaba opcije *-a* povzroči izpis vseh opcij. Opcija *-g* povzroči izpis v šestnajstiškem sistemu. Sledi pregled glavnih opcij za nastavitev terminala:

Glavne opcije:

parenb (-parenb)	Omogoči (prepreči) generacijo oziroma detekcijo parnega bita.
parodd (-parodd)	Izbira lihe ali sode parnosti.
cs7 cs8	Izbira števila bitov v znakih.
300 600 1200 ...	Izbira hitrosti terminala.
term termtype	Izbira tipa terminala.

su ... postani sistemski administrator ali drug uporabnik

Oblika ukaza:
su [-] [ime [argumenti...]]

Opis:
su omogoča direkteno spremembo v *sistemskega administratorja* (super user) brez posrednega izstopa z *exit* in ponovnega vstopa kot *root*. Isti klic lahko uporabimo za prehod na drugega uporabnika. V obeh primerih moramo poznati tudi geslo novega uporabnika.

Brez znaka - bomo pri spremembi v *sistemskega uporabnika* ali v uporabnika *ime* obdržali staro uporabnikovo okolje. Če pa navedemo pomišljaj, se bo tudi okolje nastavilo tako, kot bi se pri regularnem vstopu (login).

" argumenti" lahko vsebujejo stikalo *-c* in ukaz (z argumenti), ki naj se izve v okolju sistemskega uporabnika ali navedenega uporabnika. Ta klic pa je le začasen. Po njegovi izvedbi smo spet v svojem okolju.

tail ... prikaz repa neke datoteke

Oblika ukaza:
tail [-[num][lbc][f]] [file]
tail [+[num][lbc][f]] [file]

Opis:
tail prikaže zadnji del navedene datoteke. Običajno je prikazanih zadnjih 10 vrstic. Lahko pa zahtevamo prikaz zadnjih *num* vrstic (*l*), blokov (*b*) ali znakov (*c*), kar definiramo z ustreznim opcijskim stikalom. Če namesto znaka - uporabimo +, bo *tail* prikazal zadnji del datoteke za *num* vrsticami, bloki ali znaki.

Primer:
tail -50c janez

Prikazanih bo zadnjih 50 znakov datoteke *janez*.

tar ... arhiviranje datotek

Oblika ukaza:
tar [key][files]

Opis:
S tem ukazom shranjujemo datoteke na arhivski medij (disketa, trak). Z njim lahko datoteke tudi restavriramo. Akcijo krmili argument *key*. Ta je lahko sestavljen iz različnih opcij:

Glavne opcije:

c Tvorimo nov arhiv. Imenovane datoteke se bodo zapisale na začetek arhiva.
r Imenovane datoteke bodo zapisane na konec obstoječega arhiva.
x Imenovane datoteke bodo izvlečene iz arhiva. Če smo navedli direktorij, bo restavrirana tudi njegova vsebina. Če nismo navedli imena datotek, bo restavrirana celotna vsebina arhiva.
t Zahtevamo listanje (prikaz vsebine) arhiva oziroma izpis imen navedenih datotek, če jih arhiv vsebuje.

tee ... izhod iz cevi

Oblika ukaza:
tee [-i][-a][-u][file]...

Opis:
tee kopira svoj vhod na izhod in tudi na navedeno datoteko. Če ga uporabimo v verigi ukazov, ki so povezani s cevmi (pipes), lahko tako pomnimo podatke, ki tečejo po taki cevi.

Glavne opcije:

-a Izhod iz *tee* se doda (append) v navedeno datoteko.
-I Ignoriranje prekinitev (interrupts)
-u Izhod bo *" unbuffered"*.

Primer:
ls -l | tee pregled | wc -l

ls tvori spisek datotek v tekočem direktoriju in ga posreduje programu *tee*. Ta pomni ta spisek v datoteki pregled in ga hkrati posreduje programu *wc*. Slednji v bistvu prešteje število vrstic (in s tem datotek) in to izpiše na zaslon.

telnet ... povezava z oddaljenim računalnikom

Oblika ukaza:
telnet [host[port]]

Opis:
Ta ukaz omogoča najavo na oddaljen računalnik. Ni nujno, da tudi na tem računalniku teče operacijski sistem UNIX, mora pa biti podprt s protokolom TELNET.

test ... oceni logični izraz

Oblika ukaza:
test izraz [izraz]

Opis:
test ocenjuje izraze, ki jih sestavljamo s spodaj navedenimi operatorji. Glede na oceno vrne izstopno (exit) vrednost 0 (TRUE) ali različno od 0 (FALSE). Običajno uporabljamo ta ukaz v lupininih ukaznih procedurah (shell procedures) v povezavi z ukazi *if* in *while*.

Glavni operatorji:

-r file	TRUE, če datoteka *file* obstoja in je dovoljeno njeno branje.
-w file	TRUE, če *file* obstaja in je dovoljeno njeno spreminjanje.
-x file	TRUE, če datoteka *file* obstoja in je izvedljiva.
-f file	TRUE, če *file* obstaja in je to navadna datoteka.
-s file	TRUE, če file obstaja in ima dolžino večjo od 0.
-z st1	TRUE, če je dolžina niza *st1* enaka 0.
-n st1	TRUE, če je dolžina niza *st1* večja od 0.
st1=st2	TRUE, če sta niza *st1* in *st2* enaka.
st1!=st2	TRUE, če niza *st1* in *st2* nista enaka.
st1	TRUE, če niz st1 ni prazen.
n1 -eq n2	TRUE, če sta celi števili *n1* in *n2* enaki.
	Podobno delujejo operatorji *-ne, -gt, -ge, -lt, -le.*
!	negacija izraza
-a	konjunkcija izrazov
-o	disjunkcija izrazov
()	grupiranje izrazov.

time ... merjenje časa izvajanja ukaza

Oblika ukaza:
time command

Opis:
Izvede se podani ukaz *command*. Zatem sledi izpis poročila, koliko časa je bilo pri tem porabljenega.

touch ... ažuriranje časov dostopa in spremembe datoteke

Oblika ukaza:
touch [-amc][mmddhhmm[yy]] files

Opis:
S tem ukazom lahko spremenimo čase dostopa (opcija *-a*) oziroma spremembe (opcija *-m*) navedenih datotek. Opcija *-c* prepreči tvorbo imenovane datoteke, če ta še ne obstaja.

tr ... preslikava znakov

Oblika ukaza:
tr [-cds][niz1][niz2]

Opis:
Program tr kopira standardni vhod na standardni izhod. Pri tem lahko izbrane znake briše ali nadomešča z drugimi. Prebrane znake, ki se ujemajo z znaki v nizu *niz1*, zamenja z istoležnimi znaki v nizu *niz2*. Pomagamo si lahko še z naslednjimi opcijami:

-d Brisanje znakov, ki se ujemajo z znaki v nizu 1.
-s Krčenje vseh nizov ponavljajočih se znakov, ki so navedeni v nizu 2, na en sam znak.

Pri navajanju nizov lahko ukazni zapis poenostavljamo z naslednjimi okrajšavami:

[a-z] Določanje območja znakov, v našem primeru je to območje od znaka a do znaka z.
[a*n] Pomeni n-kratno ponavljanje znaka *a*. To okrajšavo uporabljamo pri definiranju niza 2.

Z znakom \ , kateremu sledi številka z vodečo ničlo, definiramo znake z njihovim osmiškim ekvivalentom kode ASCII.

true ... vrne izhodno kodo 0

Oblika ukaza:
true

Opis:
true vrne izhodno kodo 0 in je tako komplementaren ukazu *false*. Uporabljamo ga v ukaznih datotekah.

tty ... prikaz imena (poti do) terminala

Oblika ukaza:
tty [-s]

Opis:

Ta ukaz izpiše pot (pathname) uporabnikovega terminala. Opcija *-s* prepreči ta izpis. V tem primeru je edini rezultat tega ukaza njegova izstopna koda. Ta je 0, če je standardni vhod terminal, sicer pa je 1.

umask ... nastavitev običajnih zaščit datotek in direktorijev

Oblika ukaza:
umask [zaščite]

Opis:
Z ***umask*** lahko določimo, katere dovolilnice naj veljajo pri tvorbi nove datoteke. V splošnem velja za zaščito osmiška koda 666, (dovoljenje za branje in spreminjanje za vse), za nove direktorije pa običajno velja maska dovolilnic 777 (branje, spreminjanje in iskanje za vse). Novo masko dovolilnic dobimo tako, da od obstoječe odštejemo argument z osmiško podano zaščitno kodo (iz dovoljenj izločimo zaščite!!). Če argumenta (osmiške kode) ne podamo, bo *umask* izpisal trenutno veljavno masko.

Primer:
umask 066

Novo tvorjene datoteke in direktoriji ne bodo imeli dovoljenja za branje in pisanje za skupino in ostale. Datoteke bodo torej imele osmiško kodo dovolilnic 600, direktoriji pa 711.

uname ... izpis imena sistema

Oblike ukaza:
uname [-snrvma]
uname [-S systemName]

Opis:
Ukaz povzroči prikaz imena sistema, ki ga uporabljamo. Pri tem imamo na voljo več opcij:

-s Izpis imena sistema
-n Izpis imena vozla, po katerem je sistem razpoznaven v komunikacijski mreži.
-r Izpis izdaje (release) operacijskega sistema
-v Izpis verzije operacijskega sistema
-m Izpis imena računalnika (stroja)
-a Izpis vseh podatkov

uniq ... ugotovi ponovljene vrstice v datoteki

Oblika ukaza:
uniq [-udc [+n] [-n]]] [file1 [file2]]

Opis:
Če ne navedemo nobenega argumenta, posreduje ***uniq*** standardni vhod na standardni izhod, pri tem pa prepiše le eno kopijo morebitno enakih in zaporednih vrstic. Če podamo imena datotek, uporablja *uniq* nvedeno vhodno (ter izhodno) datoteko.

Glavne opcije:

-u Ne bo prikaza podvojenih vrstic
-d Samo podvojene vrstice bodo prikazane
-c Prikazane bodo enkratne vrstice ter po ena kopija večkratnih. V razliko alternativi ukaza brez opcij bo tu pred vsako vrstico izpisano število nastopov enake vrstice.

Primer:
sort naslovi | uniq -c
Ugotovi ponavljane vrstice, četudi v originalni datoteki naslovi niso zaporedne.

uptime ... Koliko časa deluje sistem

Oblika ukaza:
uptime

Opis:
S tem ukazom dobimo podatek, koliko časa sistem že deluje, kdo na njem dela, lahko pa izpiše tudi njegovo obremenitev.

uucp, uulog, uuname ... sodelovanje med sistemi UNIX

Oblike ukaza:
uucp [options] files.. file2
uulog [options] -ssystem

uulog [options] system
uulog [options] -fsystem
uuname [-l][-c]

Opis:
Ukaz ***uucp*** kopira navedene datoteke v ciljno datoteko. Pri tem je lahko ime posameznih datotek navedeno na običajen način s potjo do datoteke, lahko pa ima obliko:

systemName!pathName

Pri tem je *systemName* eno od imen sistemov, ki jih *uucp* pozna. Sistemsko ime je lahko tudi bolj kompleksno oziroma sestavljeno iz seznama sistemskih imen naslednje oblike:

systemName!systemName!..!systemName!pathName

V tem primeru želimo prenos datotek po navedeni prenosni poti med navedenimi sistemi. Glavne opcije, ki jih pri tem ukazu lahko uporabimo:

-m Obvesti uporabnika, ko je prenos datotek končan
-nuser Obvesti uporabnika oddaljenega (remote) sistema, ko je prenos končan.

uustat ... izpis in krmiljenje statusa uucp

Oblike ukaza:
uustat[-m]
uustat[-a]
uustat[-q]
uustat[-kjobid]
uustat[-ssystem][-uuser]

Opis:
-m Prikaz statusa dostopnosti vseh strojev
-a Izpis vseh čakajočih poslov (jobs)
-q Prikaz čakajočih poslov za vsak stroj. Izpis pove, koliko ukaznih datotek čaka za vsak sistem.
kjobid Ubijanje (kill) zahtevka *uucp* za posel (job) z identifikatorjem *jobid*.
-ssys Prikaz statusa zahtevkov za oddaljeni sistem sys
-uuser Prikaz stanja zahtevkov,podanih s strani uporabnika *user*.

uux ... izvajanje ukazov z enega UNIXa na drugem

Oblika ukaza:
uux [options] command

Opis:
Z ***uux*** povzročimo izvajanje ukaza na navedenem sistemu. Argumenti ukaza lahko vsebujejo imena datotek, ki lahko vsebujejo tudi imena sistemov (in ustreznih poti), na katerih se te datoteke nahajajo.

Primer:
uux “diff usg!/usr/dan/file1 pwba!/a4/dan/file2 >!~/dan/file.diff"

Ukaz bo prevzel datoteki *file1* in *file2*, ki sta locirani na strojih *usg* oziroma *pwba*. Izvedel bo ukaz *diff* in posredoval rezultat v datoteko *file.diff* na lokalnem poddirektoriju *dan.*

vi ... urejevalnik tekstovnih datotek

Oblika ukaza:
vi [file1]

Opis:
vi je osnovni urejevalnik tekstovnih datotek. Opisan je v posebnem poglavju.

wait ... čakanje na zaključek procesov v ozadju

Oblika ukaza:
wait

Opis:
Ukaz povzroči čakanje zaključka vseh procesov, ki smo jih sprožili z ukazno vrstico, ki se je končala z znakom *&*.

wall .. izpis obvestila vsem uporabnikom

Oblika ukaza:
wall

Opis:
wall bere obvestilo s standardnega vhoda (do nastopa End of File). Nato to obvestilo pošlje vsem trenutnim uporabnikom sistema.

wc ... štetje besed v datoteki

Oblika ukaza:
wc [-lwc] [file...]

Opis:
wc šteje znake, vrstice ali besede v navedenih datotekah. Če ne navedemo nobene datoteke, obravnava *wc* standardni vhod.

Glavne opcije:

-l štetje vrstic
-w štetje besed
-c štetje znakov

Primer:
ls -l | wc -l

Določi število datotek, ki jih vsebuje tekoči direktorij.

who ... prikaz aktivnih uporabnikov

Oblike ukaza:
who [-ulpdbrtas] [file1]
who am i

Opis:
who poda ime, terminal in čas vstopa za vse uporabnike, ki so navezani na sistem. *who am I* poda podatke o uporabniku, ki je tak ukaz vtipkal. Če kot argument navedemo datoteko */etc/wtmp*, dobimo sled vseh vstopov (login) in izstopov (exit) od nastanka navedene datoteke.

Glavne opcije:

-l Podan bo spisek prostih terminalov.
-b Podan bo datum in čas zagona operacijskega sistema.
-t Podan bo datum in čas zadnje spremembe sistemskega časa.

Primer:
who | grep janez

Ugotovi, če j*anez* uporablja sistem.

write ... pisanje sporočila drugemu uporabniku

Oblika ukaza:
write userName [terminalName]

Opis:
S tem ukazom lahko pošiljamo sporočila drugemu, trenutno navezanemu uporabniku. Ta je obveščen, kdo mu pošilja obvestilo, nato pa sledi prepis vrstic pošiljatelja k imenovanemu naslovniku. Če naslovnik uporablja več terminalov, moramo poleg imena naslovnika podati tudi ime terminala, na katerega posredujemo tekst.

xinit ... inicializacija sistema X Window

Oblika ukaza:
xinit [[client] options][--[server][display]options[

Opis:
Program ***xinit*** sproži strežnik sistema X Windows in uvodni program - klijent. Ta je normalno emulator terminala. Ko ta uvodni klijent konča, ukine *xinit* tudi strežni program in tudi sam konča.

7. PROGRAMIRANJE V LUPINI

7.1. Osnovni pojmi

Zaradi poenostavljanja v tem poglavju enačimo pojem ***ukaz*** s pojmom ***program*** in ne nazadnje s pojmom ***programski proces***.

V resnici seveda velja, da je *program* koda nekega programa, ki je pomnjena v neki datoteki, ki ima primerno ime. To ime uporabljamo kot ***ukaz*** za proženje ***programa***. *Programski proces* pa je (poenostavljeno gledano) izvajanje tega programa.

Lupina lahko tudi bere in izvaja zaporedja ukazov (ukazne procedure), pomnjena v ***ukaznih datotekah*** (script files). Ta pristop uporabimo, če je delo računalnika pretežno paketno in manj interaktivno. Ukazni jezik (command language) lupine omogoča:

- ❒ Izvajanje programov: Le-te lahko izvajamo zaporedno ali paralelno.
- ❒ Tvorbo datotek: Tvorimo lahko nove datoteke ali dopolnjujemo obstoječe.
- ❒ Posredovanje argumentov: Programom, ki so klicani znotraj ukazne datoteke, lahko posredujemo argumente, ki so podani v ukazni vrstici.
- ❒ Pogojno izvajanje programov: Izvajanje programov je pogojeno z uspešnostjo izvedbe predhodnih programov. Preddefinirane procedure: sestavimo lahko ukazne datoteke (s***cr***ipt files), ki definirajo zaporedja programov, ki naj se izvedejo. Parametrizacijo: V ukaznih datotekah lahko uporabimo spremenljivke. Tako bo njihova uporabnost bolj splošna.

Uporabnik ima na voljo več lupin. Danes so najbolj poznane naslednje

csh	C shell
sh	Bourne shell
ksh	Korn shell

V prvi rubriki je navedeno ime posameznega programa (oziroma lupine), s katerim lupino izberemo (oziroma poženemo).

Bourne shell je bila prva standardna lupina, ki je omogočala *programiranje ukazov* UNIX. Še vedno je nekakšen standard.

C shell oponaša jezik C, primerna pa je predvsem za interaktivno delo z računalnikom.

Korn shell skuša združevati prednosti prej omenjenih lupin. Tako lahko vse ukazne datoteke, pisane za *Bourne shell*, uporabimo tudi v okolju lupine *Korn* (ne pa obratno).

Ne glede na to, katero lupino smo izbrali, lahko v njej uporabljamo ***uslužnostne programe*** UNIX (utilities), kot je na primer *cp* ali *rm*.

Izvajanje ukazov, pomnjenih v ukaznih datotekah lahko dosežemo na več načinov. Najbolj preprosto je z ukazom tipa

sh file [arguments]

Pri tem je *file* ime datoteke, v kateri je vpisano zaporedje ukazov. Temu zaporedju recimo še ***ukazna procedura*** (script file***)***.

V isti vrstici lahko podamo morebitne ***argumente***, ki jih lupina uporablja v ustreznih vrsticah izvajane ukazne procedure. Argumenti v ukazni vrstici nadomeste v trenutku izvedbe ukazne procedure formalne ***pozicijske parametre*** $1, $2,..

Če neko ukazno datoteko pogosto uporabljamo, je bolje, da ji v dovoljenja dodamo izvedljivost, na primer z ukazom oblike

chmod +x file

V tem primeru lahko nato sprožimo ukazno datoteko kar z naslednjo obliko:

file [arguments]

Primer:
Naj vsebuje datoteka *"poglej"* naslednjo vrstico:

who | grep $1

V tem primeru je ukaz

poglej janez

ekvivalenten ukazu

who | grep janez.

Ukazne datoteke lahko vsebujejo različne programske konstrukte. Tako lahko vsebujejo spremenljivke, ki so vrednotene kot nizi. Spremenljivke začenjajo s črko in lahko vsebujejo še črke, številke in podčrtaje.

Spremenljivkam prirejamo vrednosti s pomočjo enačajev. Na primer:

user= janez

Vrednost spremenljivke dobimo z uporabo njenega imena, pred katerega zapišemo znak *$*.

Na primer ukaz

echo $user

bi povzročil na zaslonu izpis niza *janez*.

Spremenljivke na primer omogočajo pripravnejše zapise pogosto uporabljanih (in morda dolgih) nizov.

Poznamo tudi več sistemskih spremenljivk. Nekatere važnejše sistemske spremenljivke oziroma njih vrednosti so:

$HOME	Privzeti (default) argument ukaza *cd*. Predstavlja pot do vstopnega (login) direktorija uporabnika.
$PATH	Iskalni seznam poti do ukazov,
$PS1	Najavni niz (normalno znak $)
$PS2	Najavni niz, če računalnik pričakuje vnos (normalno znak >).
$?	Desetiški niz z izhodnim statusom zadnje izvajane komande. To lahko uporabljamo v programskih konstruktih *if, while* itd.
$#	Število pozicijskih parametrov
$*	Vrednost, ki pomeni " kakršenkoli niz"

7.2. Programski konstrukti

Preprost ukaz (simple command) v kakršnikoli lupini je zaporedje več besed, ločenih s presledki. Prva beseda v tem zaporedju določa ukaz, ki naj bo izveden. Običajno so besede, ki sledijo, ukazni argumenti.

Zaporedje stavkov sestavljajo preprosti stavki, ki so med seboj ločeni s podpičji (tako kot to velja na primer v programskem jeziku C ali pascal). Stavki se bodo izvedli zaporedno, eden za drugim.

Cevovod ukazov (pipeline) je zaporedje ukazov, ločenih z znakom | **.** Vsak od teh ukazov ima svoj standarni vhod in standardni izhod. Standardni izhod predhodnega ukaza je po ***cevi*** (pipe) navezan na standardni vhod naslednjega ukaza. Vsi ukazi potekajo vzporedno kot ločeni oziroma konkurenčni procesi. Mehanizem takega cevovoda ukazov zmanjšuje potrebo po uporabi (začasnih) datotek.

V ukazne datoteke lahko seveda pišemo tudi komentarje. Vrstice s komentarjem se začenjajo z dvopičjem ali z znakom #.

Programski jeziki vseh lupin predvidevajo tudi uporabo krmilnih programskih struktur, s katerimi dobimo ***sestavljene stavke***. Tako poznajo vsi stavek ***if***. Slovnična pravila pa se od lupine do lupine razlikujejo.

Pred obravnavo oblik za pogojno krmiljenje ukaznih procedur se spomnimo, da posamezni programi ob svojem izstopu vračajo izstopno kodo, ki normalno kaže uspešnost njihovega izvajanja. Velja dogovor, da uspešno zaključen program vrne kodo 0. V sklopu vrste uslužnostnih programov zasledimo tudi programe *true*, *false* in *test*. Opisani so v abecednem pregledu ukazov UNIX. Program *true* vrne vedno kodo 0, program *false* vrne kodo 1, program *test* pa uporabljamo za test nekega pogoja in vrne kodo, ki ustreza stanju tega pogoja.

Bralec naj bi že poznal vsaj programski jezik pascal, še raje pa jezik C. Zato bo opis konstruktov za krmiljenje programov v ukaznih procedurah zelo zgoščen. Po vrsti si bomo najprej ogledali posamezne programske konstrukte , ki veljajo za lupini Korn in Bourne. Na koncu bomo podali še oblike, veljavne za lupino C.

7.3. Sestavljeni stavki lupin Bourne in Korn

Pogojna razvejitev ***if*** ima v primeru lupin Korn in Bourne naslednjo splošno obliko:

```
if zaporedje stavkov1
 then
        zaporedje stavkov2
 else
        zaporedje stavkov3
 fi
```

Lupina izvede *zaporedje stavkov1* in glede na izhodni status zadnjega ukaza v tem zaporedju izvede *zaporedje stavkov2* ali (morebitno) *zaporedje stavkov3*. Pogosto se kot *zaporedje stavkov1* uporablja lupinin ukaz ***test***, ki je bolj podrobno opisan v poglavju o uslužnostnih programih lupine.

Primer:

```
if test "$1" =-x
```

```
then
        echo ukaz $0 je bil klican s parametrom -x
fi
```

Formalni parameter *$1* smo dali med narekovaje in s tem zagotovili, da bo programu *test* posredovan vsaj prazen niz, če njemu ustrezni realni parameter ne bi bil definiran. V takem primeru bi sicer *test* javil napako. Zanimiva je tudi uporaba parametra *$0*, ki predstavlja prvo besedo ukaza, torej ukaz sam.

Primer:
Imejmo ukazno datoteko z naslednjo vsebino:

```
echo "Vpisi svoje ime:\c"
read  ime
if [-n "$ime"] then
    echo "vpisal si ime $ime \n"
else
    echo "Nisi vpisal imena, ponovi \n"
    exit 1
fi
```

Druga vrstica v sestavljenem stavku bere niz s tipkovnice in ga vpiše v spremenljivko *ime*. Tretja vrstica preverja, če je vnešeni niz poln. V primeru, da imena nismo vnesli, se stavek zaključi z izstopom v predzadnji vrstici. Vrednost izstopne spremenljivje (exit status) bo enak 1. Vrednost, različna od 1 običajno pomeni neregularen konec nekega programa ali procedure.

Konstrukt ***case*** ima naslednjo obliko:

```
case name   in
   vzorec1)     zaporedje ukazov1;;
   vzorec2)     zaporedje ukazov2;;
  ...
esac
```

Lupina primerja spremenljivko *name* z vsakim vzorcem in, če se ta ujema, izvede ustrezno zaporedje ukazov. Ker * ustreza kakršnemukoli nizu, ga lahko uporabimo kot " privzeti" (default) vzorec.

Vsaka alternativa se zaključuje z dvojnim podpičjem. To ima enako funkcijo kot besedica *break* v programskem jeziku C. Prepreči torej nezaželeno izvajanje stavkov v alternativah, ki slede.

Primer:

```
case $odgovor in
```

```
  d|D|da|Da)   echo pritrdilno;;
  n|N|ne|Ne)   echo nikalno;;
   *)  echo napacen odgovor;;
esac
```

Sestavljeni stavek ***while*** ima naslednjo obliko:

```
while zaporedje ukazov1
do
    zaporedje ukazov2
done
```

Lupina ciklično izvaja *zaporedje ukazov1* in primerja izstopni status zadnjega ukaza v zaporedju. Če je ta status enak 0, izvede še *zaporedje ukazov2*, sicer cikel prekine.

Primer:

```
while test $1
do
    ...
   shift
done
```

Ta zgled podaja še eno značilnost lupine. Ukaz ***shift*** preimenuje pozicijske parametre *$2, $3*,.. v *$1, $2*,... in pri tem izgubi staro vrednost *$1*.

Primer:

```
while true
do
   who
   ps
done
```

Tak program predstavlja neskončno zanko, v kateri se izmenično izvajata programa *who* in *ps*. Spomnimo se še enkrat, da *true* ni ključna beseda, je le klasično ime ustreznega ukaza oziroma programa.

Zanka ***for*** ima naslednjo splošno obliko:

```
for name   in w1 w2 ...
  do
        zaporedje ukazov
  done
```

Ukazi v zaporedju so ločeni z RETURN ali s podpičjem. *name* je spremenljivka, ki zaporedno zavzema vrednosti *w1, w2,...* Ciklično izvajanje *zaporedja ukazov* se neha, ko porabimo zalogo besed *w1, w2,.. .*

Primer:

```
for x in *
do
   if test -d  $x
   then
      echo $x je direktorij
   else
      echo \$x je datoteka
   fi
done
```

Taki ukazni datoteki lahko navajamo kot argumente zaporedje imen datotek, za katere želimo vedeti, ali so to imena direktorijev ali ne.

7.4. Sestavljeni stavki lupine C

Stavek ***if*** ima v splošnem obliko:

```
if izraz then
   zaporedje stavkov1
else
  zaporedje stavkov2
endif
```

Ta sestavljeni stavek se v primeri z lupino Korn ali Bourne ne razlikuje bistveno. Razlika je le v zadnji besedi (endif).

Sestavljenemu stavku *case* iz lupin Korn in Bourne ustreza v lupini C stavek ***switch***, ki ima naslednjo splošno obliko:

```
switch (beseda)
   case vzorec1
      zaporedje stavkov1
      breaksw
   case vzorec2
      zaporedje stavkov2
      breaksw
```

```
   default:
      zaporedje stavkov3
      breaksw
endsw
```

Ta oblika je torej precej bolj podobna znanemu stavku *switch* v programskem jeziku C.

Stavek ***while*** ima v lupini C naslednjo splošno obliko:

```
while izraz
   zaporedje stavkov
end
```

Očitno je oblika pri lupinah Korn in Bourne močnješa, saj dopušča namesto preprostega izraza za besedico while kar zaporedje stavkov (pri čemer se vrednoti zadnji stavek v zaporedju).

Stavku *for* z lupin Korn in Bourne ustreza v lupini C stavek ***foreach***, ki ima naslednjo obliko:

```
foreach  name (seznam besed)
   zaporedje ukazov
end
```

Tudi tu se *zaporedje stavkov* ponavlja tako, da po vrsti spremenljivka *name* zavzema vredosti besed iz *seznama besed*.

7.5. Drugi programski ukazi

Ukaz ***echo*** izpiše na standardnem izhodu niz. Velja v vseh lupinah.

Primer:
```
echo Zacetek kopiranja datotek
```

Ukaz ***exit*** prekine izvajanje ukazne procedure. Kot parameter lahko navedemo izstopno vrednost

Primer:
```
exit 1
```

Ukaz ***break*** omogoča predčasni izhod iz sestavljenih stavkov for ali while. Velja le za lupini Korn in Bourne.

Ukaz ***read*** nastavi vrednost navedene spremenljivke na niz, ki ga vtipkamo. Ukaz velja le za lupini Korn in Bourne.

Primer:
read ime

Ukaz ***set*** uporablja lupina C za branje ene vrstice Pri tem moramo na desni strani stavka *set* uporabiti znaka ***$<***
Primer:

echo "Ali hoces nastaviti datum [y/n/q]"
set odgovor = $<

Posebnosti lupin Korn in C (ne pa Bourne) sta še, da omogočata **pomnenje** uporabljanih ukazov (history). To omogoča kasnejše sklicevanje na že uporabljeni ukaz, kar je predvsem pri interaktivni uporabi dolgoveznih zaporednih stavkov ali cevovodov še kako zaželeno. Z ukazom ***history*** dobimo na zaslonu oštevilčen izpis zadnjih ukazov. Ponovno izvedbo takega ukaza lahko v lupini Korn zahtevamo s klicajem in številko ukaza

Primer:

!6

Spet se bo izvedel ukaz, ki je v "zgodovini ukazov" naveden pod številko 6.

Posebnost lupin C in Korn je še, da omogočata prirejanje novega imena (alias) nekemu enostavnemu ukazu, zaporedju ali cevovodu ukazov. V ta namen uporabimo ukaz ***alias***. To možnost izkoriščamo predvsem pri pogosto uporabljanih ukazih. Interaktivno delo je tako nekoliko poenostavljeno.

8. SISTEMSKI KONCEPTI UNIX

Povprečen uporabnik računalniškega sistema se bo pri svojem delu zadovoljil s skromnim poznavanjem nekaj osnovnih ukazov lupine. Resnejše delo pa zahteva vsaj informativno poznavanje operacijskega sistema. S tem si lahko delovno okolje na računalniku bolj racionalno uredimo, kar se ne nazadnje kaže v bolj učinkovitem delu, pa tudi izrabi računalnika. Določene operacije lahko na računalniku izvaja le pooblaščeni uporabnik (administrator sistema), ki ga UNIX pozna *kot super - uporabnika* (super-user).

Poznavanje ozadja operacijskega sistema omogoča tudi razvoj novih programskih orodij, ki potrebujejo za izvedbo svoje naloge pomoč operacijskega sistema. Interakcijo med takimi orodji in operacijskim sistemom dosežemo s takoimenovanimi ***sistemskimi klici*** (System calls).

Ena od nalog operacijskega sistema je, da nudi uporabniku računalniškega sistema udobno okolje, v katerem sistem prevzema upravljanje komponent strojne opreme. Del operacijskega sistema UNIX, ki opravlja te funkcije imenujemo ***jedro*** (kernel). Jedro je nameščeno v delovnem pomnilniku računalnika in mora zato biti majhno. Veliko funkcij, ki jih pri drugih operacijskih sistemih izvaja jedro, so pri UNIX vgrajene kot normalni programi, rezidenčni na disku in brez posebnih privilegijev.

Na hitro si oglejmo interno delovanje UNIX in sicer s poudarkom na upravljanju uporabnikovih programov. Omenili smo že pojem *programskega procesa*. S tem izrazom razumemo izvajanje nekega zaporedja instrukcij na dani množici podatkov. Proces mora nekdo tvoriti (create) in po določenem času umre ali pa ga nekdo " ukine" (kill). V obdobju življenja procesa se le-ta lahko znajde v različnih stanjih. Tako poznamo v grobem stanje izvajanja. Proces se lahko znajde tudi v stanju čakanja.

V mnogouporabniškem okolju je v vsakem trenutku prisotnih več procesov. V primeru UNIX lahko najdemo najmanj 1 proces za vsakega navezanega uporabnika, prisotnih pa je tudi več " sistemskih" procesov.

Pri enoprocesorskih računalnikih s sistemom UNIX se v danem trenutku lahko izvaja le en proces, drugi pa so pripravljeni, da pridejo na vrsto, ali pa čakajo na kakšen dogodek, kot je na primer sprejem znaka s tipkovnice, zaključek prenosa nekega bloka podatkov na disk, prihod nekega signala od nekega drugega procesa ipd.

V intervalih čakanja na nadaljevanje izvajanja hrani operacijski sistem vse informacije, ki so potrebne za nemoteno nadaljevanje danega procesa. Tako se ohranja vsebina pomnilnika(instrukcije in podatki), vsebine strojnih registrov, stanje odprtih datotek in drugi podatki, ki so morda uporabniku skriti (accounting,..).

Celoto teh podatkov imenujemo ***slika procesa***. To je v bistvu stanje nekega navideznega računalnika v danem trenutku. Sam proces pa predstavlja izvajanje te slike. Med izvajanjem procesa mora biti slika v centralnem pomnilniku. Tu ostaja tudi v času izvajanja drugih procesov, dokler nek proces z višjo prednostjo ne povzroči prepis (dela) slike na disk.

Med izvajanjem procesa je le-ta predstavljen v pomnilniku s tremi logično ločenimi sekcijami. Prva sekcija vsebuje ***kodo*** izvajanega programa (text section). To kodo lahko uporabljajo tudi drugi procesi, ki predstavljajo izvajanje enakega programa (na primer pri večkratni uporabi urejevalnika besedil). Taka sekcija mora biti zaščitena pred pisanjem. Drugi dve sekciji sta lastni posameznim procesom. Drugo predstavljajo ***podatki*** (data section), tretjo pa ***sklad*** (stack section), ki med izvajanjem procesa raste ali pada.

Proces nastane takrat, ko nek drug, že aktiven proces izvaja sistemski klic ***fork().*** To pomeni, da mora en proces obstajati že ob zagonu sistema UNIX, ta pa povzroči verižno reakcijo tvorbe novih procesov. Rojenemu procesu pravimo ***otrok***. Dodeljena mu je natančna kopija slike originalnega procesa - očeta. razlikuje se le po številčni identifikacijski oznaki ***PID***. Funkcija *fork()* vrne procesu - očetu *PID* procesa - otroka. Otroku pa vrne vrednost *0*. Proces - oče lahko vpliva na otroka, saj tako pozna njegov PID.

Sistemski programer piše programe, ki normalno obdelujejo podatke na datotekah, uporabljajo razne periferne naprave in podobno. Na voljo ima vrsto rutin, ki so zbrane bodisi v ***standardni vhodno - izhodni*** sistemski knjižnjici (stdio) in v drugih knjižnjicah. Poleg tega lahko kliče rutine, ki so sestavni del jedra operacijskega sistema. Klicem rutin, rezidenčnih v jedru, pravimo ***sistemski klici***.

Rutine, ki so sestavni del standardne vhodno- izhodne knjižnjice (stdio), omogočajo izvajanje takoimenovanih " visoko nivojskih" vhodnih in izhodnih operacij. Značilno za te rutine je, da skrivajo v sebi tudi vmesno pomnenje (buffering) ter vhodne oziroma izhodne konverzije numeričnih podatkov.

Na tem mestu jih le naštejmo. Sicer pa jih bralec najde v ustreznih priročnikih za programiranje v jeziku C.

fopen()	Odpiranje datoteke za branje ali pisanje. Ukaz vrne kazalec na odprt podatkovni tok(stream). Funkcija ima dva argumenta. Prvi je niz, ki vsebuje ime datoteke, drugi pa znak, ki pove, ali datoteko odpiramo za branje (r), pisanje (w) ali dodajanje (a kot *append*).
fclose()	Zapora datoteke oziroma njej ustreznega podatkovnega toka. Kot argument navedemo kazalec na odprti podatkovni tok.
getc()	Branje znaka iz odprte datoteke. Argument je kazalec na njej ustrezen podatkovni tok.

putc() Pisanje znaka v odprto datoteko. Prvi argument je kazalec na podatkovni tok. Drugi argument je znak, ki ga zapisujemo.

fgets() Branje ene vrstice z odprte datoteke. Prvi argument je naslov polja, v katerega beremo, drugi je dolžina tega polja, tretji je kazalec na odprto datoteko.

fputs() Zapis vrstice oziroma niza v datoteko. Ima dva argumenta: naslov polja z nizom, ki se zaključuje z znakom 0, ter kazalec na datoteko.

fread() Branje datoteke brez deljenja njene vsebine na znake ali vrstice. Klic ima 4 argumente: Kazalec na podatkovno strukturo, dolžino podatkovne strukture v bytih, število podatkov, ki naj bodo prebrani ter kazalec na odprti podatkovni tok (datoteko).

fwrite() Zapis podatkov v datoteko. Klic rutine je analogen klicu bralne rutine *fread()* in ima prav tako 4 parametre.

fscanf() Formatirano branje podatkov. Prvi argument je kazalec na podatkovni tok. Drugi je niz, ki določa format branja, sledijo naslovi spremenljivk, ki jih beremo v skladu z določenim formatom.

fprintf() Formatiran zapis podatkov. Prvi argument je kazalec na podatkovni tok, drugi argument je niz, ki definira format zapisa. Slede imena spremenljivk, ki jih želimo zapisati na datoteko v določenem formatu.

sscanf() Branje formatiranih podatkov iz navedenega niza. Namesto kazalca na datoteko navedemo naslov niza, sicer pa je klic rutine analogen klicu fscanf().

sprintf() Zapis formatiranih podatkov v navedeni niz. Klic je podoben klicu rutine *fprintf()* , le da namesto kazalca na datoteko navedemo niz, v katerega želimo zapisati formatirane podatke.

lseek() Premik v odprti datoteki na zahtevano mesto. Prvi argument klicane rutine predstavlja kazalec na odprt podatkovni tok, drugi *je long integer*, ki mu pravimo *odmik* (offset). Določa odmik v bytih. Referenčno točko za ta odmik definira tretji argument, ki je enak 0 (odmik od začetka datoteke), 1 (odmik od trenutne pozicije) ali 2 (odmik od konca datoteke).

Kot že rečeno, imamo na voljo tudi sistemske klice, ki so včasih manj " prijazni", zato pa bolj direktni. V nadaljevanju sledi opis važnejših sistemskih klicev, ki jih bomo združili po namembnosti.

9.SISTEMSKI KLICI UNIX

9.1. Uvodna razlaga

Sistemski klici so zahtevki operacijskemu sistemu, da opravi za naš program neko storitev (servis). Tako je na primer ***read()*** sistemski klic, s katerim zahtevamo, da operacijski sistem napolni neko polje (buffer) s podatki, branimi z diska ali neke druge periferne naprave. Uporaba sistemskih klicev je nujna, da ne pride v računalniku do zmede. Le operacijski sistem oziroma njegovo jedro sme namreč imeti direkten dostop do sistemskih podatkovnih struktur. Sistemske klice uporabljamo za upravljanje z datotečnim sistemom, za nadzor programskih procesov in za njihovo medsebojno komuniciranje. V nadaljevanju bomo spoznali najvažnejše klice, pri čemer pa je potrebno poznavanje programskega jezika C, ki je v okolju UNIX standarden.

Sistemski klic izvedemo enako kot klic katerekoli funkcije v našem programu. če je sistem pri njegovi obravnavi naletel na napako, običajno vrne vrednost -1 in pomni kodo napake v eksterni spremenljiki ***errno***.

9.2. Delo z datotečnim sistemom

Sistemski klici omogočajo tvorbo, odpiranje in zapiranje datotek, branje in pisanje datotek, naključni dostop, pridruževanje (aliasing) in odstranjevanje datotek, branje in spreminjanje statusa datotek.Pri teh operacijah uporabljamo bodisi niz, ki predstavlja absolutno ali relativno pot do datoteke, bodisi celoštevilčno oznako, ki ji pravimo ***številka datoteke*** (file descriptor) in ki določa ustrezni vhodno- izhodni kanal. Spomnimo se, da številke 0, 1 in 2 utrezajo standardnemu vhodu, izhodu in izpisu napak. Kanal je povezava med nekim procesom in datoteko, ki jo vidi proces kot neformatiran tok bytov. V prejšnjem poglavju smo za delo z datotekami uporabljali visokonivojske klice standardnih rutin. Pri njih smo namesto številke datoteke (file descriptor) imeli ***kazalec na podatkovni tok*** (file pointer). Razlika je tudi v tem, da je številka datoteke kratko celo število.

Pri vhodno-izhodnih operacijah mora proces določiti številko datoteke za ustrezni vhodno-izhodni kanal, polje (buffer), v katerega naj bi bili podatki iz datoteke brani (oziroma pisani), ter velikost tega polja.

Sistemski klic ***creat()*** tvori in odpre novo izpisno datoteko.

Sintaksa tega klica je naslednja:

int creat(fileName,mode)
*char *fileName;*
int mode;

Pri tem je *fileName* kazalec na niz, ki vsebuje ime (in pot do) datoteke, *mode* pa vsebuje kodo dovolilnic (access permissions). Klic *creat()* vrne celoštevilčno vrednost, ki jo nato uporabljamo kot številko datoteke (file descriptor).

Podoben je sistemski klic ***open(),*** s katerim odpremo datoteko bodisi za branje, pisanje ali branje in pisanje. Sintaksa tega klica je naslednja:

#include <fcntl.h>
.....
int open(fileName,optionFlags,mode)
*char *fileName;*
int optionFlags,mode;

Tudi ta funkcija vrne številko datoteke (številko kanala), v primeru neuspeha pa vrednost *-1*. Pri tem pomeni *optionFlags* tip kanala. Na voljo so naslednje možnosti, ki jih lahko smiselno tudi kombiniramo:

O_RDONLY	Datoteka *fileName* je odprta za branje.
O_WRONLY	Datoteka *fileName* je odprta za pisanje.
O_APPEND	Vsak zapis v datoteko bo dodan na njen konec.
O_CREAT	Če datoteka še ne obstaja, bo imela dovoljenja v skladu s kodo *mode*, sicer pa bodo veljala stara dovoljenja.
O_TRUNC	Če datoteka že obstoja, bo njena vsebina pozabljena.
O_EXCL	V kombinaciji z O_CREAT ne bo prišlo do odpiranja datoteke, če ta že obstaja.

Obratna operacija je zapiranje kanala oziroma datoteke. To dosežemo s klicem *close()*. Ta ima naslednjo sintakso:

int close(fileDescriptor)
int fileDescriptor;

Pri tem *fileDescriptor* določa trenutno odprti kanal, ki ga želimo zapreti.

Branje datoteke oziroma zapis vanjo omogočata sistemska klica *read()* oziroma *write()*. Običajno ju uporabljamo za zaporedno branje oziroma pisanje. Njuna sintaksa je naslednja:

int read(fileDescriptor,bufferPointer,transferSize)

int fileDescriptor,
*char *bufferPointer;*
unsigned transferSize;

int write(fileDescriptor,bufferPointer,transferSize)
int fileDescriptor,
*char *bufferPointer;*
unsigned transferSize;

Pri tem *fileDescriptor* določa številko kanala, *bufferPointer* kaže na polje, v katerem naj bi bili prenašani podatki, *transferSize* pa pove v bistvu velikost tega polja. Normalno obe funkciji vračata število prenešenih bytov ali pa vrednost *-1*, ki pomeni neuspelo operacijo.

V nekaterih primerih je zaželen naključni dostop do posameznih podatkov v datoteki. Pomagamo si s sistemskim klicem ***lseek().*** Med vhodno-izhodnimi operacijami namreč sistem UNIX uporablja datotečni kazalec (FILE POINTER), ki je tipa *long integer*. Ta predstavlja v bistvu število bytov od začetka datoteke do naslednjega (še neprenešenega) byta v datoteki. Ta kazalec je torej ***odmik*** (offset) naslednjega byta v datoteki. Klic *lseek()* omogoča vsiljeno spremembo vrednosti tega kazalca. Sintaksa tega klica je naslednja:

long lseek(fileDescriptor,offset,whence)
int fileDescriptor;
long offset;
int whence;

Funkcija *lseek()* vrne novo vrednost datotečnega kazalca, ob neuspehu pa vrne negativno vrednost. Parameter *whence* (od kod) pove, kako uporabiti parameter *offset*. Na voljo so naslednje možnosti:

whence	**nova pozicija**
0	odmik offset velja od začetka datoteke
1	odmik offset velja od trenutne pozicije
2	odmik velja od konca datoteke

V nekaterih primerih potrebujemo podvojitev kanala za neko odprto datoteko. Najbolj pogosto naletimo na tako zahtevo po tvorbi novega programskega procesa s klicem ***fork(),*** ki ga bomo spoznali kasneje. To dosežemo s klicem funkcije ***dup(),*** ki vrne številko kanala- dvojnika. Sintaksa te funkcije je naslednja:

int dup(fileDescriptor)
int fileDescriptor;

Datotečni sistem UNIX dopušča, da poimenujemo isto datoteko z več imeni. Temu recimo ***poimenovanje*** (aliasing). Poimenovanje doda datoteki še eno ime, pri tem pa

vsa imena naslavljajo isto podatkovno strukturo. V povezavi s tem moramo poznati še nekaj pojmov. Ti so:

i-vozel — ***i-vozel*** (i-node) je podatkovna struktura, ki vsebuje vse podatke o datoteki, razen njenega imena. Tako vsebuje podatke o dolžini datoteke, o njenih zaščitah, lastništvu, času zadnjega dostopa in zadnje spremembe in kazalce na bloke, kjer se nahajajo resnični podatki. Dostop do datoteke je možen le preko *i-vozla*, kar pa nadzoruje jedro UNIX (kernel).

i-seznam — Vsi *i-vozli* so pomnjeni v polju datotečnega sistema, ki mu pravimo ***i-seznam*** (i-list).

i-število — Dostop do določenega *i-vozla* v *i-seznamu* dosežemo s številom, ki mu pravimo ***i-število.*** Vsaka datoteka v datotečnem sistemu ima torej svoje *i-število*.

vez — Vsakemu *i-številu*, ki imenuje nek *i-vozel*, pravimo ***vez*** (link). V sklopu *i-vozla* je polje *links*, ki pove, koliko vezi trenutno naslavlja dani *i-vozel*.

direktorij — To je datoteka posebne vrste, ki usklajuje ASCII imena datotek z njim ustreznimi *i-vozli*. Glede na to, da potekajo številke *i-vozlov* od 1 naprej, pomeni vrednost 0 v direktoriju, da je ustrezno mesto (entry) v direktoriju še prosto.

Poimenovanje neke datoteke pomeni torej, da sistem poišče v direktoriju prazno mesto in vanj vpiše novo ime datoteke. V rubriko z *i-vozli* pa vrednost 0 nadomesti z vrednostjo *i-vozla* poimenovane datoteke. Vse to dosežemo s klicem funkcije ***link(),*** ki ima naslednjo sintakso:

```
int link(originalName,aliasName)
char *originalName, *aliasName;
```

Pri tem sta *originalName* in *aliasName* kazalca na niza, ki vsebujeta obstoječe in pridruženo ime datoteke.

Inverzna operacija je ***odstranjevanje*** imena, ki ga dosežemo s klicem funkcije ***unlink().*** Ta ima naslednjo sintakso:

```
int unlink(fileName)
char *fileName;
```

Pri tem je f*ileName* ime datoteke, ki ga želimo odstraniti. Klic *unlink()* povzroči brisanje imena iz ustreznega direktorija ter zmanjšanje polja *links* v *i-vozlu* dane datoteke. če pade vrednost *links* na 0, pomeni to, da je datoteka izgubila tudi zadnje

ime in je s tem zbrisana. *i-vozel* se sprosti, sprostijo pa se tudi bloki, ki so pomnili podatke datoteke.

Včasih potrebujemo v programih informacijo o statusu posamezne datoteke oziroma njej ustreznega *i-vozla*. To lahko dobimo na primer s klicem sistemskih funkcij ***stat()*** oziroma ***fstat(),*** pri čemer pa moramo poznati format podatkovne strukture *i-vozla.* Ta je opisan v datoteki */usr/include/sys/stat.h* in uporablja tipe, ki si definirani v datoteki */usr/include/sys/types.h*. Zato je sintaksa sistemskih klicev *stat()* oziroma *fstat()* naslednja:

```
#include <sys/types.h>
#include <sys/stat.h>
....
int stat(fileName,statBuf)
char *fileName;
struct stat *statBuf;

int fstat(fileDescriptor,statBuf)
int fileDescriptor;
struct stat  *statBuf;
```

Datoteko lahko torej določimo s *fileName*, ki je kazalec na niz z imenom datoteke, lahko pa uporabimo številko kanala, ki jo pomnimo v spremenljivki *fileDescriptor*. V obeh primerih dobimo v strukturi *statBuf* ustrezne informacije o stanju datoteke.

V programih lahko tudi spreminjamo dostopnost do datotek in njih lastništvo s klici ***access(), chmod()*** oziroma ***chown().***

Klic *access()* ima naslednjo sintakso:

```
int access(fileName, accessMode)
char  *fileName;
int accessMode;
```

V *accessMode* zapišemo zahtevane dovolilnice, pri čemer imamo naslednje možnosti:

vrednost	**pomen**
04	branje
02	pisanje
01	izvajanje
00	obstoj

Ob uspešni vzpostavitvi dovoljenj vrne funkcija *access()* vrednost 0, sicer pa vrne -1.

9.3. Kontrola periferij

Periferne naprave krmilijo programski moduli, ki jim pravimo ***gonilniki naprav*** (device drivers). Ti so sestavni del ***jedra*** (kernel) operacijskega sistema UNIX. Programer ima na voljo sistemski klic ***ioctl()***, s katerim določa in spreminja status neke periferne naprave. Pri tem uporablja ukaze, ki so specifični za posamezen terminal in so definirani v datoteki */usr/include/termio.h*. Klic *ioctl()* se med različnimi verzijami sistema UNIX razlikuje. Podrobnosti najdemo v ustreznih priročnikih. Pri posameznih verzijah sistema UNIX imamo na voljo še druge sistemske klice za kontrolo perifernih naprav.

9.4. Podatki o uporabnikih

Vsak uporabnik sistema ima enoumno ***vstopno ime*** (login name). To ime uporabljamo pri vstopu v sistem, pri uporabi elektronske pošte, pri označevanju izpisov na skupnem tiskalniku itd. To ime lahko dobimo s sistemskim klicem ***getlogin(),*** ki vrne kazalec na niz z imenom. V nekaterih verzijah UNIX imamo v ta namen kak drug klic (na primer *cuserid()*).

Vsak programski proces v sistemu ima pridružena še dva številčna identifikatorja: *real user ID* in *effective user ID*. Realna številka uporabnika označuje uporabnika, ki izvaja nek proces. Efektivna številka je normalno enaka realni. Spremenimo pa jo, če prevzemamo dovoljenja, ki veljajo za nekega drugega uporabnika. Obe oznaki lahko dobimo s sistemskima klicema ***getuid()*** oziroma ***geteuid().***

Uporabnikovo oznako (UID), tako realno kot efektivno,lahko spremenimo s klicem ***setuid().*** Ta klic omogoča sistemskemu administratorju spreminjanje njegove identitete.

Velja še, da lahko skupina uporabnikov tvori ***grupo***. Taka skupina uporabnikov ima lahko na primer določene privilegije do dostopa do datotek, ki za ostale uporabnike sistema ne veljajo. Tudi tu dodeljuje sistem skupinam realno in efektivno grupno oznako, ki je dana v obliki celega števila. Obe oznaki dobimo s klicema ***getuid()*** oziroma ***geteuid().***

9.5. Kontrola časa

V vseh verzijah UNIX dobimo trenutni čas s sistemskim klicem ***time().*** Kot argument moramo navesti spremenljivko tipa *long integer*. V njej bo vrnil *tekoči čas*. Pogosto pa želimo imeti ta čas v obliki ASCII. To lahko dosežemo s pomočjo nekaj dodatnih rutin:

V datoteki *time.h* imamo podano strukturo *tm*. Ta struktura ima celoštevilčne (int) elemente tm_sec, tm_min, tm_hour,.., v katero prepakira rutina *localtime()* lokalni čas. Podobno bi dobili v tej strukturi ob uporabi rutine *gmtime()* univerzalni čas. Končno lahko uporabimo rutino *asctime(),* ki na osnovi podatkov v strukturi *tm* formira niz ASCII s časom oziroma datumom.

Primer:

```
longint t;
 .....
time(t);
printf(" time:\%s" ,asctime(localtime(t)));
```

Včasih si želimo, da bi naš programski proces za določen čas " zaspal". V ta namen imamo sistemski klic ***sleep(),*** ki ima en sam argument. Ta predstavlja sekunde " spanja" procesa.

Uporaben je tudi klic ***alarm(),*** ki ima prav tako en argument. Če je ta večji od nič, pomeni, da želimo čez toliko sekund dvigniti alarmni signal *SIGALRM*. S klicem *alarm(0),* torej z argumentom z vrednostjo 0, nastop alarma preprečimo. Kako uporabimo alarmni signal, zasledimo v poglavju, ki opisuje procesne signale.

Končno omenimo še, da imamo na voljo sistemski klic ***times(),*** s pomočjo katerega lahko ugotovimo, koliko časa je porabil proces. Ta sistemski klic vrne ustrezno informacijo v podatkovno strukturo tipa *tms*, ki je definirana v datoteki *sys/times.h*.

9.6. Procesni signali

Signali so ***programske prekinitve*** (software interrupts), ki so posredovani procesom, normalno z namenom, da jih obveste o nekem nenavadnem dogodku v njihovem okolju. Proces lahko na tak signal reagira na tri načine:

- ❒ Signal lahko ignorira.
- ❒ Signal lahko *ujame* in reagira v skladu s proceduro, ki jo v ta namen definiramo. Procedura se izvede, ko pride tak signal.

❒ Izvede se *normalna* (default) akcija, ki jo sistem predvideva ob nastopu takega signala.

Različne verzije sistema UNIX imajo različno število signalov. Nekaj pa jih je le skupnih. Nekateri od teh so:

SIGHUP	Ta signal je posredovan procesom, katerih terminal je bil izklopljen.
SIGINT	Prekinitveni zahtevek s tastature terminala
SIGILL	Nelegalna instrukcija
SIGFPE	*" Floating point"* napaka: delitev z 0, prekoračitev in podobno
SIGKILL	*Kill*. Ta signal lahko ignoriramo, ujamemo ali blokiramo.
SIGSYS	Napačen argument v sistemskem klicu
SIGPIPE	Pisanje v cev, ki je nihče ne bere
SIGALRM	Signal " alarmne ure"

Sistemski klic, ki pošlje nek signal, se imenuje ***kill()*** in ima dva argumenta. Prvi argument, *PID* je celo število, procesna številka (process identifier). Določa proces, kateremu pošiljamo signal. drugi argument je ena od prej navedenih oznak in definira signal, ki ga pošiljamo danemu procesu. če je *PID* enak 0, pošiljamo signal vsem procesom naše skupine. Na voljo pa je še nekaj drugih kombinacij. Negativna vrednost *PID* se uporablja za posredovanje signala vsem drugim procesom.

Kot že rečeno, lahko v programskem procesu zahtevamo ignoriranje nekega signala. To dosežemo s sistemskim klicem oblike

signal(signame,SIG_IGN);

Navedeni signal bo ignoriran do nadaljnjega. Še bolj atraktivno pa je definiranje lastne rutine, ki naj se izvede ob nastopu nakega signala. S klicem oblike

signal(SIGINT, handler)

na primer dosežemo, da se bo ob nastopu signala SIGINT izvedla rutina *handler(),* ki smo jo sami napisali in vključili v kodo našega programa.

9.7. Kontrola programskih procesov

Če želimo sprožiti nov, dodaten proces, moramo v našem programu najprej uporabiti klic ***fork(),*** ki ima naslednjo sintakso

int fork()

S tem klicem povzročimo tvorbo novega procesa (otroka), ki je (ob rojstvu) enak svojemu očetu, od katerega se razlikuje le po svoji, specifični procesni številki (*PID*). Ker od trenutka rojstva oba procesa (oče in otrok) živita ločeno, lahko spoznata, kdo je kdo (oče ali otrok) po tem, da očetu vrne klic fork vrednost *PID* procesa-otroka, otroku pa vrne vrednost 0.

Omenimo tu še klica ***getpid()*** in ***getppid(),*** ki sta po sintaksi podobna klicu *fork()*. Pri tem vrne *getpid() PID* kličočega procesa, *getppid()* pa mu vrne *PID* njegovega očeta.

Kljub temu, da poslej oba procesa živita ločeno, delna vez med njima le še obstaja. Tako lahko proces-oče uporabi klic ***wait(),*** ki povzroči čakanje očeta na konec (življenja) procesa otroka. Sintaksa klica *wait()* je naslednja:

int wait(status)
*int *status;*

Pri tem je *status* kazalec na celoštevilčno vrednost, v katero shrani UNIX vrednost, ki jo vrača ob svojem koncu proces-otrok.

Konec procesa (ki je normalno otrok nekemu drugemu procesu) dosežemo s klicem ***exit(),*** ki ima naslednjo sintakso:

void exit (status)
int status;

Pomen spremenljivke *status* smo že pojasnili pri opisu klica *wait()*.

Operacijski sistem UNIX nudi na voljo več sistemskih klicev za nadzor in krmiljenje programskih procesov. Tako imamo celo družino klicev tipa ***exec(),*** ki imajo vsi za nalogo, da neko (izvršljivo) programsko datoteko preslikajo v proces. S tem ukazom normalno procesu- otroku nadomestimo " program" podedovan od očeta, s "programom", branim z diska. Za zgled si poglejmo sintakso enega od teh klicev:

int execv(fileName,argv)
* char *fileName, *arg[];*

Pri tem je *fileName* ime izvršljive (programske) datoteke, ki naj se transformira v proces. V polju *arg* pa so argumenti, ki jih temu procesu posredujemo.

V razliko od ostalih sistemskih klicev se ob uspešnem zaključku sistemskega klica tipa *exec()* nadzor ne vrne na naslednjo instrukcijo kličočega procesa temveč na začetek programa, ki ustreza imenovani programski datoteki. Novi proces tako nadomesti proces, ki je vseboval klic tipa *exec()*.

10.ADMINISTRACIJA SISTEMA

Administracija sistema predstavlja vse aktivnosti, ki pomenijo vzdrževanje (predvsem sistemske) programske opreme na danem računalniku. Pri večuporabniških kih sistemih naj bi se s tem ukvarjala pooblaščena oseba. Pri sistemu UNIX pravimo takemu pooblaščenemu operaterju **skrbnik sistema** (super-user). Leta lahko uporablja tudi ukaze, ki drugim niso dopustni.

Skrbnik sistema ki ima vstopno ime ***root***, ima vse možne privilegije in lahko torej zbriše katerikoli datoteko ali **ukine** (kill) katerikoli programski proces. Samo on lahko spremeni tekoči datum z ukazom *date*. Samo on lahko montira nov datotečni sistem z ukazom *mount* itd.

Vlogo skrbnika sistema lahko prevzamemo na več načinov, poznati pa moramo ustrezno geslo, ki se definira pri sami instalaciji sistema. Normalno zahteva sistem to geslo pri vsakem zagonu računalnika. Pogovor v tem primeru poteka preko sistemskega terminala (konzole). Kasneje, med samim obratovanjem, pa lahko bodisi vstopimo v računalnik preko kateregakoli terminala, kot vsak drug uporabnik, le pravo ime in geslo sistemskega skrbnika moramo vpisati. Tretja možnost je, da za sam vstop uporabimo navadno (neprivilegirano) ime in geslo, nato pa uporabimo ukaz ***su*** in se tako prelevimo v "sistemskega skrbnika". Seveda moramo tudi v tem primeru poznati njegovo geslo.

Naloge sistemskega skrbnika (superuser tasks) so:

- ❒ Zagon in izklop sistema,
- ❒ Vzdrževanje datotečnega sistema,
- ❒ Dodeljevanje dovolilnic in gesel uporabnikom,
- ❒ Skrb za varnost sistema,
- ❒ Dodajanje perifernih naprav oziroma njihove programske podpore,
- ❒ Uglaševanje sistema.

Pri tem si sistemski skrbnik pomaga s primernimi ukazi, ki smo jih nanizali v prejšnjih poglavjih. Delo si lahko olajša s pripravo in uporabo primernih ukaznih procedur. Pogosto pa ima na voljo še dodatne programe za administracijo sistema, ki pa se od primera do primera razlikujejo in jih zato tu ne bomo navajali. Oglejmo si še nekatere podrobnosti, vezane na prej naštete aktivnosti.

10.1. Zagon sistema

Zagon sistema se od primera do primera razlikuje, vendar se pri tem normalno najprej vzpostavi v **enouporabniški režim**. Sistem nato preveri **verodostojnost datotečnega sistema**. V primeru nasilnega izklopa računalnika, na primer izpada električnega napajanja, je namreč velika verjetnost, da posamezne datoteke, ki so bile uporabljane, niso pravilno zaprte in ne vsebujejo pravilnih podatkov. Sistem skuša v takem primeru stanje urediti. V najslabšem primeru si mora sistemski operater pomagati s kopijo datotečnega sistema (back-up), če jo je seveda napravil.

Ob zagonu lahko tudi definiramo tekoči datum in (s klicem programa ***init***) preidemo v **mnogouporabniški** režim. Program *init* ugotovi iz podatkov v datoteki */etc/inittab*, koliko uporabnikov predvideva sistem, katere periferne enote so na voljo itd. Po iniciaciji se na zaslonu pojavi normalni zahtevek *" login"*, na katerega moramo odgovoriti z ustreznim vstopnim imenom in geslom.

10.2. Izklop sistema

Računalnika z operacijskim sistemom UNIX ne smemo enostavno izklopiti ali "resetirati", čeprav morda v danem trenutku sami delamo na njem. Izklop sistema zahteva prav tako ustrezen postopek, ki posreduje na vse terminale ustrezna opozorila, ukine vse programske procese, se prepriča, če so vse vhodno- izhodne operacije zaključene in demontira datotečne podsisteme. Temu šele sledi fizični izklop računalnika. Postopek za izklop računalnika je običajno zapisan v ukazni datoteki ***/etc/shutdown***.

10.3. Gesla in dovolilnice uporabnikov

Kot je znano, imajo vsi uporabniki sistema UNIX ustrezno vstopno ime in geslo ter pripadajo ustrezni skupini (group). Ti podatki so pomnjeni v datotekah */etc/passwd* oziroma */etc/group.*

Datoteka *passwd* v direktoriju */etc* je navadna ASCII datoteka, v kateri je vsakemu uporabniku namenjena ena vrstica. Pomen posameznih elementov in format take vrstice kaže naslednji primer:

janez:ABCD1234:203:10:Janez Kovac:/aa/elektro/janez:/bin/sh

Vrstica vsebuje polja, ločena z dvopičji. Za programirano uporabo te datoteke je v *pwd.h* opisan format njenih podatkov. Ta ima (lahko) naslednjo obliko:

```
struct passwd {
    char  *pw_name;
    char  *pw_passwd;
    int  pw_uid;
    int  pw_gid;
    int  pw_quota;
    char *pw_comment;
    char *pw_gecos;
    char  *pw_dir;
    char  *pw_shell;
};
```

Pomen teh polj je naslednji:

pw_name	Vstopno ime uporabnika (login name)
pw_passwd	Zakodirano geslo uporabnika
pw_uid	Številčna oznaka uporabnika
pw_gid	Številčna oznaka uporabnika, ki velja ob njegovem vstopu
pw_quota	Neuporabljeno polje
pw_comment	Neuporabljeno polje
pw_gecos	V tem polju je polno ime uporabnika ter drugi podatki.To polje je v bistvu razdeljeno v 4 podpolja, ločena med seboj z vejicami.
pw_dir	Domači (home) direktorij uporabnika
pw_shell	Program, ki naj se požene pri vstopu uporabnika. To je normalno njegova lupina.

Navadni uporabniki morajo imeti visoke identifikacijske številke, sistemski administrator, ki ima vstopno ime ***root***, pa ima obvezno identifikacijsko številko " 0".

Tudi datoteka */etc/group* je ASCII in vrstice v njej imajo naslednjo obliko.

elektro::20:janez,lojze, pepe

Pri tem pomeni prvi element ime skupine. Temu sledi geslo skupine, kar pa se redko uporablja (zato dve zaporedni dvopičji). Sledi identifikacijska številka skupine. V zadnjem polju je seznam članov skupine. Njihova imena so ločena z vejicami.

Vpis novega uporabnika tako terja naslednji postopek:

- ❒ Vpis uporabnikovega imena in drugih podatkov v datoteko passwd,
- ❒ Uporabniku moramo dodeliti domači direktorij
- ❒ Po potrebi vpišemo uporabnika v skupinsko datoteko.

Tudi za vpisovanje novih uporabnikov ima sistemski operater na voljo ustrezen program, ki pa se od sistema do sistema razlikuje.

Končno sodi v skrb za uporabnike tudi beleženje njihove porabe računalniškega sistema. V ta namen se izvajajo v ozadju (in background) ustrezni **zakriti sistemski procesi** (daemon processes), sistemski administrator pa ima na voljo ustrezna programska orodja za uporabo zbiranih podatkov.

10.4. Skrb za varnost sistema

Na varnost sistema deloma vpliva že dejstvo, da lahko nevarne posege dela le **sistemski administrator** (superuser) in da imajo vsi uporabniki svoja gesla. Med samim delovanjem sistema skrbe za varnost različni zakriti procesi, sistemski administrator pa ima na voljo primerne programe, s katerimi spremlja zasedenost diskov oziroma datotečnega sistema, zasedenost računalnika s procesi in s tem njegovo zasičenost. Po potrebi skrbi varnostni podsistem za generiranje ustreznih poročil in za komprimiranje informacij v teh poročilih.

V skrb za varnost sistema sodi tudi periodično shranjevanje **rezervne kopije** (back-up) celotnega datotečnega sistema ali njegovega dela. Eno od orodij, ki ga lahko pri tem uporabimo, je na primer ***cpio***, ki smo ga že opisali. Primerno je, da vsebino rezervne kopije verificiramo, na kopiji pa označimo naslednje:

- Ime računalnika,
- ime datotečnega sistema.
- datum zapisa rezervne kopije,
- ime osebe, ki je znapravila kopijo,
- število blokov na mediju (trak, disk,..),
- število vseh medijev za to kopijo in tekočo številko medija.

Restavracija podatkov iz rezervne kopije v datotečni sistem je lahko **popolna** ali **delna**, pač glede na trenutne potrebe. Tudi samo kopiranje v rezervno kopijo ima lahko različne nivoje. Bolj poredko, na primer 1 krat mesečno, napravimo kopijo celotnega sistema. Vmes pa kopiramo le najnovejše datoteke, na primer nastale oziroma spremenjene v zadnjem tednu. (inkrementalni "backup")

10.5. Dodajanje perifernih naprav

Te aktivnosti nastopijo pri dopolnitvi računalniškega sistema znovimi ali drugačnimi perifernimi napravami, kot so diski, terminali in podobno. V sistem je potrebno vgraditi ustrezno programsko podporo. Sem sodijo **gonilniki perifernih naprav**

(device drivers). Ti se razlikujejo od naprave do naprave in od sistema do sistema. Ustrezne postopke moramo prebrati v specializiranih priročnikih. Skupno vsem sistemom UNIX je, da obravnavajo periferne naprave kot **posebne datoteke**. Tako datoteko moramo tvoriti v poddirektoriju *dev i*nv ta namen uporabimo ukaz *mknod.* Ta tvori za novo napravo ustrezen **i-vozel**. Ustrezne gonilne programe (drivers) pa moramo uvesti tudi v samo kodo **jedra** *(*kernel) operacijskega sistema.

10.6. Uglaševanje sistema

Operacijski sistem UNIX lahko optimiziramo bodisi s ciljem, da povečamo njegovo propustnost, bodisi da želimo minimizirati porabo njegovih spominskih zmogljivosti. Ponovno uglaševanje sistema je smiselno, če se število uporabnikov bistveno poveča, če smo sistemu dodali spominske kapacitete ali če moramo nenadoma reševati bistveno drugačne probleme (na primer da uporabljamo nenavadno velike datoteke).

Sistemski operater lahko spreminja število medpomnilnikov (buffers), reorganizira sezname v datotečnem sistemu, spreminja parametre uvrščanja procesov (scheduling).

Pri svojem delu si pomagamo z dodatnimi sistemskimi programi, ki omogočajo tako ustrezno diagnosticiranje kot tudi spreminjanje parametrov. Uglaševanje sistema je seveda od sistema do sistema različno in podrobnosti zasledimo v specializiranih sistemskih priročnikih.

11.LITERATURA

L.Reiss, J.Radin: UNIX System Adminstration Guide, Osborne Mc Graw Hill 1993, ISBN 0-07-881951-2

W.Brecht: Verteilte Systeme unter UNIX, Eine praxisorientierte Einfuehrung Vieweg 1992, ISBN 3-528-05194-9

R. Morgan, H. McGilton: Introducing UNIX System V, McGraw-Hill Software series for computer professionals,NewYork 1987

R. Thomas, L. R. Rogers, J. L. Yates: Advanced Programmer's Guide to UNIX System V; McGraw-Hill, Berkeley, California,1986

D. A. Curry: Using C on the UNIX System, O'Relly \& Associates, Purdue Research Foundation 1989

M. J. Rochkind: Advanced UNIX programming, Prentice Hall Software series Englewood Cliffs, New Yersey 1985

S. R. Bourne: The Unix system; Addison Wesley Publ.Comp., London 1982,

M. Matteuzzi, P. Pellizzardi: Ambiente Unix; gruppo editoriale Jackson 1985,

UniSoft Systems: Uniplus+ System V, Release 2; User Guide

SCO UNIX System V/386: Operating System, System Administrator's Guide