Univerzitetna založba
Univerze v Mariboru

SISTEMSKA ADMINISTRACIJA V LINUXU

Iztok Fister, ml.
Iztok Fister

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko

Sistemska administracija v Linuxu

Avtorja
Iztok Fister, ml.
Iztok Fister

November 2023

Naslov
Title
Sistemska administracija v Linuxu
System Administration in Linux

Avtorja
Authors
Iztok Fister, ml.
(Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko)

Iztok Fister
(Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko)

Recenzija
Review
Matjaž Krnc
(Univerza na Primorskem, Fakulteta za matematiko, naravoslovje in informacijske tehnologije)

Grega Vrbančič
(Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko)

Lektoriranje
Language editing
Nuša Grah

Tehnična urednika
Technical editors
Iztok Fister, ml.
(Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko)

Jan Perša
(Univerza v Mariboru, Univerzitetna založba)

Oblikovanje ovitka
Cover designer
Jan Perša
(Univerza v Mariboru, Univerzitetna založba)

Grafične priloge
Graphics material
Fister, ml., Fister, 2023

Grafika na ovitku
Cover graphic
1295397, avtor: OpenClipart-Vectors, pixabay.com, CC0, 2023

Založnik
Published by
Univerza v Mariboru
Univerzitetna založba
Slomškov trg 15, 2000 Maribor, Slovenija
https://press.um.si, zalozba@um.si

Izdajatelj
Issued by
Univerza v Mariboru,
Fakulteta za elektrotehniko, računalništvo in informatiko
Koroška cesta 46, 2000 Maribor, Slovenija
https://feri.um.si, feri@um.si

Izdaja
Edition
Prva izdaja

Vrsta publikacije
Publication type
E-knjiga

Dostopno na
Available at
http://press.um.si/index.php/ump/catalog/book/824

Izdano
Published at
Maribor, november 2023

CIP - Kataložni zapis o publikaciji
Univerzitetna knjižnica Maribor

004

FISTER, Iztok, ml.
Sistemska administracija v Linuxu [Elektronski vir] / Iztok Fister, ml., Iztok Fister. - 1. izd. - E-knjiga. - Maribor : Univerza v Mariboru, Univerzitetna založba, 2023

Način dostopa (URL): https://press.um.si/index.php/ump/catalog/book/824
ISBN 978-961-286-796-6 (PDF)
doi: 10.18690/978-961-286-796-6
COBISS.SI-ID 170317315

ISBN	978-961-286-796-6 (pdf)
DOI	https://doi.org/10.18690/um.feri.11.2023
Cena *Price*	Brezplačni izvod
Odgovorna oseba založnika *For publisher*	prof. dr. Zdravko Kačič, rektor Univerze v Mariboru
Citiranje *Attribution*	Fister, ml., I., Fister, I. (2023). *Sistemska administracija v Linuxu*. Univerza v Mariboru, Univerzitetna založba. doi: 10.18690/um.feri.11.2023

Kazalo

1 | Uvod

Vsaka moderno zgrajena zgradba stoji na temeljih, ki predstavljajo vmesni člen med zemeljsko podlago ter gradbenimi bloki, s katerimi obremenjujemo temelje. Temelji so zatorej pomembni elementi, brez katerih ne moremo postaviti neke zgradbe, ki bi po eni strani kljubovala vsem naravnim pojavom od vremenskih neprilik do geoloških pojavov, kot so npr. potresi, in po drugi strani služila uporabnikom zgradbe.

Podobno kot je temelj pomemben v gradbeništvu, tudi operacijski sistem (angl. Operating System) [1] v računalniškem svetu predstavlja temelje za uporabo računalnika. Ta deluje kot posrednik med strojno opremo, kot npr. vhodne enote, pomnilniške enote ter programsko opremo, ki jo lahko razdelimo na sistemsko in uporabniško programsko opremo. Moderni operacijski sistem ima veliko nalog, med katerimi so tudi zagotavljanje povezanega in organiziranega okolja za upravljanje računalniških virov, uporabnikom pa omogoča učinkovito interakcijo s sistemom.

Glede na to, da vsak računalnik, pametni telefoni in tudi moderna televizija praktično uporablja operacijski sistem, lahko dandanes na tržišču najdemo veliko operacijskih sistemov, med katerimi omenimo

samo najpomembnejše:

- **Microsoft Windows**: je trenutno najpopularnejši operacijski sistem za namizne in prenosne računalnike, najdemo pa ga lahko tudi med strežniki. Operacijski sistem Windows ponuja uporabnikom prijazen vmesnik, obsežno združljivost programske opreme in širok nabor funkcij. Ta operacijski sistem za uporabo zahteva licenco.

- **macOS**: operacijski sistem je razvilo podjetje Apple za svoje računalnike Mac. Operacijski sistem MacOS je najbolj znan po svoji elegantni zasnovi, brezhibni integraciji z drugimi napravami Apple ter bogatem ekosistemu sistemske in uporabniške programske opreme. Zaznamuje ga logotip jabolka in prav tako spada med licenčne operacijske sisteme.

- **Linux**: je odprtokodni operacijski sistem, ki je zelo prilagodljiv in se pogosto uporablja v različnih okoljih, vključno s strežniki, superračunalniki, vgrajenimi sistemi in mobilnimi napravami. Obstaja veliko različnih distribucij Linux, kot so: Ubuntu, Fedora, Debian, CentOS in ostali. Vsaka izmed omenjenih distribucij ponuja svoje specifične funkcije in sisteme za upravljanje programskih paketov. Velika večina distribucij Linux je v celoti brezplačna, kar pomeni, da jih lahko snamemo s spleta in jih uporabljamo na poljubni napravi. Obstajajo tudi plačljive distribucije Linuxa, kot npr. Red Hat Enterprise distribucija Linuxa.

- **Družina operacijskih sistemov BSD**: v katero štejemo predvsem FreeBSD, NetBSD, OpenBSD. Ti operacijski sistemi izhajajo iz prvotnega Unix BSD, ki so ga razvili na kalifornijski univerzi v Berkeleyju. Družina BSD je znana po svoji stabilnosti, razširljivosti in upoštevanju standardov, zato je najprimernejša izbira za strežnike, omrežno opremo in vgrajene sisteme.

- **Android**: je odprtokodni operacijski sistem, ki temelji na jedru Linux in ga je razvilo podjetje Google predvsem za mobilne

naprave, kot so pametni telefoni in tablični računalniki. Postal je najbolj priljubljen mobilni operacijski sistem na svetu, saj ponuja obsežen ekosistem aplikacij in seveda povezavo med različnimi storitvami.

- **Operacijski sistem iOS**: je posebej zasnovan za mobilne naprave Apple, kot so: iPhone, iPad in iPod Touch. Ponuja varen in uporabniku prijazen vmesnik, optimizirano delovanje in dostop do širokega nabora aplikacij prek trgovine App Store.

Ta knjiga je v celoti posvečena operacijskemu sistemu Linux, ki je odprtokodni operacijski sistem in je od svojega nastanka korenito spremenil svet računalništva. Linux, ki ga je leta 1991 razvil Linus Torvalds na unvierzi v Helsinkih na Finskem, je postal zmogljiva in vsestranska odprtokodna platforma, ki deluje na številnih napravah, od strežnikov in namiznih računalnikov do mobilnih telefonov in tudi vgrajenih sistemov. S svojo robustno arhitekturo, prilagodljivostjo ter živahno skupnostjo razvijalcev in uporabnikov je Linux postal priljubljena izbira posameznikov, podjetij in organizacij po vsem svetu. Običajno se sploh ne zavedamo, na koliko napravah v tipičnem domu teče operacijski sistem Linux. Najdemo ga na mobilnih napravah, v televiziji, hladilniku, usmerjevalniku in celo pametni kameri.

Komu je ta knjiga namenjena?

Pričujoča knjiga je namenjena dodiplomskim študentom različnih smeri na Fakulteti za elektrotehniko, računalništvo in informatiko ter študentom višjih letnikov iste oz. drugih fakultet Univerze v Mariboru, ki se srečujejo s predmeti s področij sistemske administracije, skriptnega programiranja, administracije strežnikov in deloma tudi virtualizacije. Prav tako je knjiga namenjena vsem, ki bi želeli poglobiti ali osvežiti znanje o sistemski administraciji v Linuxu.

Struktura knjige

Knjiga je sestavljena iz osmih poglavij. Po uvodu sledi krajši pregled operacijskega sistema Linux vključno z opisom njegovih popularnih distribucij. V tretjem poglavju se bralci seznanijo z ukazno lupino **bash** na teoretičnem in tudi praktičnem nivoju. V četrtem poglavju je prikazano skriptno programiranje v Linuxu. V petem poglavju je poudarek na procesih v operacijskem sistemu Linux, medtem ko v šestem poglavju prikažemo delo z diski in datotečnimi sistemi na operacijskem sistemu Linux. Sedmo poglavje je posvečeno ukazom za delo z uporabniki v Linuxu. V zadnjem, tj. osmem poglavju, sledi povzetek knjige.

2 | Operacijski sistem Linux

Operacijski sistem Linux razvija velika skupnost razvijalcev iz celega sveta in za tem razvojem ne stoji kakšna večja korporacija. Jedro operacijskega sistema Linux predstavlja temeljno komponento, odgovorno za upravljanje virov strojne opreme in za komunikacijo med programsko in strojno opremo. Linux je v bistvu odprtokodno jedro operacijskega sistema, ki ima korenine v 90. letih prejšnjega stoletja. Njegov glavni avtor je Linus Torvalds, ki je še vedno glavni koordinator projekta in sprejema končne odločitve o nadgrajevanju sistema ter posodabljanju. Poleg Linusa pri projektu razvoja Linuxa sodeluje tisoče in tisoče posameznikov iz celotnega sveta, ki prihajajo iz akademskega okolja, industrije, različnih organizacij ali pa so to samo računalniški navdušenci [2]. Izvorno kodo Linuxa lahko spreminja kdorkoli, saj je v celoti tudi javno dostopna. Odprtokodna licenca omogoča vsem zainteresiranim svobodo pri pregledovanju, spreminjanju in distribuciji izvorne kode [3]. Zatorej lahko trdimo, da se pri Linuxu okrite napake hitreje odpravljajo, saj lahko kodo popravlja vsak. Ne glede na to da je večina distribucij Linux v celoti brezplačna, večja podjetja prispevajo svoj delež v obliki donacij, strojne opreme ali pa s strokovnim

znanjem vse s ciljem omogočiti nadaljnji razvoja tega operacijskega sistema.

2.1 Razlika med operacijskima sistemoma Linux in Unix

Unix je star že preko 50 let. V zbirnem jeziku ga je razvilo podjetje Digital Equipment Corporation (krajše DEC) na računalniku PDP/7 pod okriljem projekta Bellovih laboratorijev tedanjega lastnika AT&T. Leta 1973 so ga prepisali v programski jezik C [4]. Hitro se je razširil v akademskih krogih. Danes obstajata dve struji razvoja tega operacijskega sistema: na AT&T in BSD.

Linux je bil izdan v letu 1991 in to ni Unix, kot je trdil že njegov začetnik Linus Torvalds [5]. Čeprav sta si zelo podobna med seboj, težko rečemo, da gre za isti operacijski sistem. Linux temelji na jedru Unix, upošteva standarde POSIX[1] in je kompatibilen z večino programske opreme Unix. Največja razlika med njima je v tem, da je Linux odprtokodni operacijski sistem, ki ga skupaj razvija skupina razvijalcev iz vsega sveta. Zato podpira nove tehnološke rešitve, ki jih plačljive platforme na operacijskih sistemih Unix (npr. AIX IBM Unix, HP-UX HP UNIX, Oracle Solaris) ne podpirajo bodisi zaradi specifičnosti, bodisi zaradi politike podjetja, ki ga razvija.

Po drugi strani pa programska oprema Linux teče tudi na operacijskem sistemu Unix [6]. Na osnovi projekta GNU (angl. GNU's Not Unix), ki je lastnik splošne javne licence (angl. General Public Licence, krajše GNU), je večina programske kode Linux brezplačno prenosljiva na Unix. Spletni strežnik Apache, na primer, lahko zaganjamo tako na Linux, kakor tudi ne-Linux platformah. Obstajajo seveda tudi odstopanja od prostih licenc GPL. Operacijski sistemi, ki se razvijajo pod

[1] POSIX (angl. Portable Operating System Interface) je družina standardov, ki jih je določila računalniška organizacija IEEE za vzrževanje kompatibilnosti med operacijskimi sistemi.

okriljem Univerze v Berkeleyu (npr. FreeBSD, NetBSD in OpenBSD), niso povsem odprti, saj jih razvijajo razvijalci v sklopu univerze in je zato njihova uporaba omejena. V bistvu predstavljajo popolni operacijski sistem Unix.

Dandanes se oba omenjena operacijska sistema uporabljata v proizvodnih okoljih brez posebnih težav. Glavna razlika v uporabi obeh v teh okoljih je, da je vzdrževanje operacijskega sistema Unix vključeno v mesečno licenčnino, pri uporabi operacijskega sistema Linux pa smo odvisni od reakcij vzdrževalnih skupin, ki praviloma delajo brezplačno.

2.2 Distribucije

Distribucija Linux (okrajšano **distro**) predstavlja operacijski sistem skupaj z zbirko programov. Ta vključuje jedro in dodatni material pogosto v obliki sistema za vzrževanje paketov (angl. package management system). Vsem distribucijam Linux je skupno poreklo jedra, ves dodatni material pa se lahko zelo razlikuje. Distribucije Linux se razlikujejo glede na fokus, podporo in popularnost. Čeprav danes obstaja stotine neodvisnih distribucij Linux, večina teh temelji na distribucijah Debian, Red Hat in SUSE.

Razlike med posameznimi distribucijami Linux v bistvu niso tako velike in se nanašajo predvsem na način, "kako čim lažje namestiti operacijski sistem" oz. "kakšno število različnih knjižnic podpirati". V nadaljevanju predstavimo nekaj izbranih popularnih distribucij operacijskega sistema Linux, ki so predstavljene v Tabeli 2.1 in razvrščene po abecedi.

Trenutno stanje popularnosti posameznih distribucij Linux lahko zainteresirani bralec vidi na spletni strani DistroWatch.com[2].

[2] `https://distrowatch.com/`

Tabela 2.1: Najpopularnejše distribucije Linuxa.

Distribucija	Spletna stran	Komentar
Alpine	www.alpinelinux.org	Lahka, stabilna in varna
Arch	archlinux.org	Lahka in fleksibilna
Debian	www.debian.org	Najbližja GNU
Fedora	fedoraproject.org	Izhaja iz Red Hata
Gentoo	www.gentoo.org	V izvorni obliki, optimizacija
Manjaro	manjaro.org	Temelji na Arch Linux
Mint	linuxmint.org	Temelji na Ubuntu
Ubuntu	ubuntu.org	Sveža verzija Debian

2.2.1 Alpine Linux

Alpine Linux[3] imenujemo tudi kot lahka (angl. lightweight) in varnostno usmerjena distribucija, ki se uporablja predvsem za strežnike in vgrajene sisteme. Alpine je prav tako ena glavnih izbir za kreiranje osnovnih slik (angl. base image) operacijskega sistema v okolju Docker, saj je celotna slika velika samo okrog 5 MB. Za upravljanje paketov uporabljamo Alpine Package Keeper (APK). Izjemno majhna velikost, ki zaznamuje Alpine Linux, omogoča uporabo v okoljih z omejenimi viri, npr. na vgrajenih sistemih.

2.2.2 Arch Linux

Arch Linux[4] je, podobno kot Alpine, lahka distribucija, ki se da zelo prilagajati in prihaja s filozofijo "naredi sam". Za upravljanje paketov uporablja upravitelja paketov Pacman, medtem ko dodatni paketi živijo v arhivu Arch User Repository (AUR). Kot zanimivost-Arch temelji na sprotnih izdajah, kjer se uporablja model tekočih izdaj, ki zagotavlja stalne posodobitve in ne izdaj po različicah. Arch Linux je znan po obsežni in izčrpni dokumentaciji, ki uporabnikom omogoča

[3] `https://www.alpinelinux.org/`
[4] `https://archlinux.org/`

poglobljeno razumevanje sistema. Uporabnikom omogoča, da od samega začetka izdelajo lasten sistem po meri in ga prilagodijo svojim posebnim potrebam. Arch Linux ponavadi ni distribucija, ki je primerna za čiste začetnike.

2.2.3 Debian

Debian[5] je ena najstarejših in najvplivnejših distribucij Linux, znana po svoji stabilnosti, zanesljivosti in zavezanosti odprtokodni programski opremi. Uporablja orodje Advanced Package Tool (APT) za namestitev paketov, ki so v paketih **.deb**. Debian ima izjemno veliko in dejavno skupnost, ki zagotavlja obsežno podporo in obsežno shrambo programske opreme. Debian prav tako podpira številne strojne arhitekture, zato je vsestransko uporaben za različne sisteme; od namiznih računalnikov, strežnikov in tudi vgrajenih sistemov. Iz Debiana je nastalo kar nekaj pomembnih distribucij, med katere lahko štejemo Ubuntu in njegove izpeljanke.

2.2.4 Fedora

Fedora[6] je distribucija Linuxa, ki je sponzorirana s strani podjetja Red Hat, s poudarkom na najnovejših funkcijah in tehnologijah. Za upravitelja paketov se uporablja program DNF in paketi RPM. Ponuja redni cikel izdajanja z novimi različicami vsakih šest mesecev, kar zagotavlja dostop do najnovejše programske opreme. Fedora služi tudi kot testni poligon za preizkušanje prihodnjih funkcij, ki bodo morda na koncu vključene v Red Hat Enterprise Linux.

2.2.5 Gentoo

Gentoo[7] je izvorno zasnovana distribucija, znana po svoji prilagodljivosti, zmogljivosti in možnostih prilagajanja. Uporablja sistem za

[5] `https://www.debian.org/`
[6] `https://fedoraproject.org/`
[7] `https://www.gentoo.org/`

upravljanje paketov Portage, ki sestavlja programsko opremo iz izvorne kode. Uporabnikom omogoča, da prilagodijo svoj sistem s sestavljanjem programske opreme, optimizirane za njihovo strojno opremo. Uporablja model tekoče izdaje, ki zagotavlja stalne posodobitve brez ločenih različic. Uporabniki morajo ročno konfigurirati in optimizirati svoj sistem, kar zagotavlja natančen nadzor. Gentoo je distribucija, primerna za napredne uporabnike, ki dajejo prednost zmogljivosti, prilagajanju in praktičnemu pristopu k svojemu sistemu Linux. Gentoo se od drugih distribucij razlikuje po tem, da temelji na izvirniku, saj ponuja zelo prilagodljivo in optimizirano okolje, čeprav za ceno večje zapletenosti in večjega časovnega vložka pri namestitvi in vzdrževanju.

2.2.6 Manjaro

Manjaro[8] je uporabniku prijazna distribucija, ki temelji na Arch Linuxu in ponuja ravnovesje med stabilnostjo in najsodobnejšo programsko opremo. Za upravitelja paketov uporablja orodje Pacman z dostopom do uradnih skladišč in tudi AUR. Ponuja model tekoče izdaje, ki zagotavlja pogoste posodobitve, hkrati pa ohranja stabilnost sistema. Zasnovan je za uporabnike, ki prehajajo z operacijskega sistema Windows ali macOS, in jim omogoča nemoteno učenje. Primeren je za uporabnike, ki iščejo prednosti prilagodljivosti sistema Arch Linux, vendar s poudarkom na prijaznosti do uporabnika.

2.2.7 Linux Mint

Linux Mint[9] je uporabniku prijazna distribucija, ki temelji na distribuciji Ubuntu in je zasnovana tako, da zagotavlja intuitivno namizje. Poudarja preprostost, enostavnost uporabe in znano postavitev, zaradi česar je privlačna izbira za novince v Linuxu ali tiste, ki prehajajo z drugih operacijskih sistemov. Omeniti velja, da Linux Mint temelji

[8] `https://manjaro.org/`
[9] `https://linuxmint.com/`

na distribuciji Ubuntu, uporablja njegov sistem za upravljanje paketov in ima koristi od njegovih repozitorijev programske opreme in podpore skupnosti. Linux Mint dodaja lasten nabor prilagoditev in privzetih aplikacij za izboljšanje splošne uporabniške izkušnje.

2.2.8 Ubuntu

Ubuntu[10] je uporabniku prijazna distribucija, ki temelji na distribuciji Debian in je znana po svoji enostavnosti uporabe in široki priljubljenosti. Uporablja sistem za upravljanje paketov APT in pakete Debian (DEB). Podpira tudi večino namiznih okolij, kot npr. GNOME, KDE, LXDE. Ubuntu je široko razširjen med uporabniki namiznih računalnikov in strežnikov, zlasti med tistimi, ki iščejo uporabniku prijazno izkušnjo z Linuxom.

2.2.9 Kako izbrati najbolj primerno distribucijo?

Velikokrat se uporabniki operacijskega sistema Windows, ki želijo presedlati na Linux, sprašujejo, s katero distribucijo začeti. Izbira najprimernejše distribucije Linux je odvisna od več dejavnikov, ki se razlikujejo z ozirom na posameznikove želje, zahteve in tehnično znanje.

V nadaljevanju predstavljamo nekaj izbranih aspektov, ki vplivajo na to odločitev. **Namen** je prvi pomemben aspekt, saj določa, za kakšna opravila bomo uporabljali Linux, tj. ali bo distribucija namenjena primarni uporabi, ali jo bomo koristili za postavitev strežnika? Naslednji aspekt je **uporabniška izkušnja**, kjer moramo znati oceniti uporabnikovo tehnično znanje in se vprašati, kakšno uporabniško izkušnjo želimo zagotavljati. Nekatere distribucije, kot sta npr. Ubuntu in Mint, dajejo prednost prijaznosti do uporabnika in enostavnosti uporabe. Take distribucije so idealne za začetnike ali uporabnike, ki prehajajo z drugih operacijskih sistemov. Po drugi strani pa distribucije, kot sta npr. Arch in Gentoo, ponujajo več možnosti prilagajanja, vendar

[10] `https://ubuntu.com/`

zahtevajo višjo raven tehničnega znanja. Dodatno je pomembno preveriti aspekt **združljivosti s strojno opremo**, kjer preverimo, ali izbrana distribucija podpira našo strojno opremo. Nekatere distribucije imajo boljšo združljivost s strojno opremo že v izhodišču, medtem ko lahko druge zahtevajo dodatno konfiguracijo ali nameščanje novih gonilnikov. Aspekt **skupnost in podpora** je potrebno vzeti v obzir pri iskanju najprimernejše distribucije, ko želimo preveriti, kako velika je skupnost in kako močna je podpora, saj oba faktorja vplivata na hitrejše učenje in odpravljanje napak. **Ekosistem programske opreme** je pomemben aspekt, kjer ocenimo repozitorije programskih paketov od distribucije do pravočasne posodobitve [7]. Tukaj moramo upoštevati tudi, ali distribucija zagotavlja potrebno sistemsko in applikativno programsko opremo, ki jo potrebujemo za svoje specifične naloge.

2.2.10 Kako namestiti Linux?

Dandanes namestitev Linuxa ne predstavlja velikih naporov že navadnim začetnikom. Skoraj vse distribucije so namreč opremljene z uradnimi vodiči, kjer so vsi koraki namestitve nazorno prikazani. Veliko distribucij ponuja grafično namestitev, kot npr. Fedora, Ubuntu, Mint, medtem ko npr. Alpine linux omogoča namestitev prek ukazne lupine. Za namestitev je najprej potrebno sneti uradno distribucijo s spleta, npr. v obliki datotek **.iso**. Zatem je potrebno določiti, na kakšen način bomo namestili izbrano distribucijo. Omenimo samo nekaj možnosti namestive, ki jih omogoča moderna tehnologija, tj. namestitev:

- kot edini operacijski sistem v računalniku,
- kot dual boot, kjer namestimo Linux vzporedno z že obstoječim operacijskim sistemom,
- s pomočjo virtualizacije (npr. VirtualBox, oz. VMware).

3 | Ukazna lupina Bash

Uporabniki komunicirajo z operacijskim sistemom Linux prek ukaznih lupin. Privzeta ukazna lupina je **bash** (angl. the Bourne again shell), za Unix pa so značilne še **sh** (angl. original Bourne shell), **ksh** (angl. Korn shell) in **csh**. Pri tem je **sh** nastajal pod okriljem AT&T, zato njegova koda ni bila nikoli razkrita. Podobno je s **csh**, ki je zaščitena pod licenco BSD. Ukazna lupina igra vlogo makro-procesorja za izvajanje ukazov, ki jih razčlenjuje interpreter ukazne vrstice (angl. command line interpreter). Poleg ukazov lahko interpreter ukazne vrstice procesira tudi kot stavčne konstrukte, zapisane v različnih skriptnih programskih jezikih [8]. Simbolično lahko ukazno lupino zapišemo kot vsoto dveh funkcionalnosti:

ukazna lupina = ukazni interpreter + interpreter programskega jezika

Tukaj prva funkcionalnost omogoča izvajanje ukazov, ki izkoriščajo možnosti operacijskega sistema, druga pa kombinira njegove različne možnosti v celoto.

Ukaze lahko posredujemo ukaznemu interpreterju na dva načina: interaktivno in neinteraktivno. Pri prvem načinu ta bere ukaze prek tipkovnice, medtem ko pri drugem iz skriptnih datotek. V neinteraktivnem načinu se ukazi lahko izvajajo sinhrono ali asinhrono. Pri

sinhronem izvajanju interpreter čaka, da se ukaz zaključi in šele potem začne z interpretiranjem novega. Pri asinhronem izvajanju interpreter nadaljuje z interpretiranjem naslednjega ukaza paralelno.

3.1 Osnove ukazne lupine bash

Vsak proces na operacijskem sistemu Linux je povezan vsaj s tremi kanali (angl. channels), ki lahko predstavljajo datoteke, omrežne povezave ali kanale, ki pripadajo drugim procesom. Standardne datoteke so STDIN, STDOUT in STDERR. Vsak ukaz Linux pričakuje vhod iz datoteke STDIN, izhode piše v datoteko STDOUT in napake v datoteko STDERR. Vhodno-izhodni model operacijskega sistema prireja vsakemu kanalu standardno vrednost tipa integer. Po dogovoru je datoteki STDIN prirejena vrednost 0, datoteki STDOUT vrednost 1 in datoteki STDERR vrednost 2. Privzeta vrednost 0 predstavlja tipkovnico, privzeti vrednosti 1 in 2 pa prikazovalnik.

Preusmeritve in cevovodi

Ukazni interpreter obravnava simbole <, > in >> kot preusmeritve. Simbol > preusmerja datoteko STDOUT v novo datoteko, kot na primer:

```
$ echo "Niz znakov." > file_1.txt
```

Z ukazom **echo** izpišemo zaporedje znakov "Niz znakov." na prikazovalnik s simbolom > pa ta izpis preusmerimo iz prikazovalnika v datoteko na disku **file_1.txt**, ki jo kreiramo na novo. S simbolom >> pripenjamo tekst na konec obstoječe datoteke, kot na primer:

```
$ echo "Niz znakov." >> file_1.txt
```

V primeru da datoteka **file_1.txt** ne obstaja, jo preusmeritev kreira na novo in doda tekst na njen začetek.

Cevovod označimo s simbolom | in omogoča povezovanje izhoda prvega ukaza z vhodom drugega, kot na primer:

```
$ ps -ef | grep httpd
```

Ukaz **ps** izpiše vse aktivne procese na sistemu Linux, ta izpis pa filtriramo z ukazom **grep** tako, da izpišemo samo tiste vrstice, ki vsebujejo niz 'httpd'.

Tudi simbol && omogoča sestavljanje cevovoda, vendar se tukaj drugi ukaz izvede samo v primeru, da se je prvi ukaz izvedel uspešno, kot na primer:

```
$ mkdir foo && cd foo
```

V primeru da se ukaz **mkdir** uspešno konča, se s pomočjo ukaza **cd** premaknemo v mapo *foo*.

Cevovod lahko ustvarimo tudi s simbolom ||, ki pa deluje nasprotno od simbola &&, tj. če se prvi ukaz cevovoda izvede neuspešno, se požene drugi, kot na primer:

```
$ cd foo || echo "No such directory."
```

Če se z ukazom **cd** ne moremo premakniti v mapo *foo*, izpišemo z ukazom **echo** sporočilo o napaki.

Simbol ; predstavlja ločilo med posameznimi ukazi, zapisanimi v isti vrstici, kot na primer:

```
$ mkdir foo; cd foo; ls -l
```

V tem primeru se ukazi izvajajo zaporedoma. V našem primeru najprej z ukazom **mkdir** ustvarimo mapo *foo*, ki z uporabo ukaza **cd** postane trenutna mapa. Z zadnjim ukazom **ls -l** izpišemo imena vseh datotek, ki jih ta mapa vsebuje v dolgem formatu.

Spremenljike in narekovaji

Spremenljivke so označene s simbolom **$**, ki predhodi njenemu imenu, ko se nanjo sklicujemo. Uporabljamo jih v ukazih oz. matematičnih izrazih. Njihova imena začenjamo s črko in nadaljujemo s črko, številko ali simbolom za podčrtaj _. Spremenljivko ustvarimo s pomočjo prireditvenega stavka (simbol =), kot na primer:

```
$ homedir='/home'
$ echo $homedir
$ /home
```

Prireditveni stavek pišemo brez presledkov, ker ukazni interpreter obravnava tekst, ki je razmejen s presledki, kot novi argument ukaza.

Za prirejanje niza znakov določeni spremenljivki uporabimo narekovaje, med katerimi lahko zapišemo besede, ki jih med seboj ločimo s presledki, kot na primer:

```
$ echo "To je niz."
```

Ukazni interpreter obravnava dvojne narekovaje enako kot enojne, na primer:

```
$ echo 'To je prav tako niz.'
```

Okoljske spremenljivke

Pri zagonu operacijskega sistema Linux nastavimo seznam argumentov ukazne lupine, ki ga imenujemo tudi okoljske spremenljivke (angl. environment variables). Ta seznam privzetih argumentov lahko pogledamo z ukazom **printenv**. Po dogovoru jih pišemo z velikimi črkami, čeprav to ni nujno, tj. uporabimo lahko tudi okoljske spremenljivke, pisane z malimi črkami.

Okoljske spremenljivke vključimo v imenski prostor **bash** ob zagonu ukazne lupine avtomatično prek skriptnih datotek ∼/**.profile** oziroma

~/**.bashrc**. Dinamično jih lahko dodajamo tudi s pomočjo ukaza **export**:

```
$ export EDITOR=nano
$ vipw
|zagon urejevalnika nano|
```

Ukaz **vipw** pogleda okoljsko spremenljivko **EDITOR** in zažene ustrezni urejevalnik.

3.2 Osnovni ukazi v Linuxu

Sintaksa ukazov na sistemu Linux je standardizirana, tj. vsak ukaz ima ime, stikala in argumente, ločene s presledki:

$ ime_ukaza -sw$_1$ ... **-sw**$_n$ **arg**$_1$ ... **arg**$_m$

Stikala **-sw**$_i$ označimo s simbolom '-' in so neobvezna. Z njimi vplivamo na način izvajanja ukaza. Z argumenti **arg**$_i$ se sklicujemo na imena objektov, nad katerimi se ukaz izvaja.

Pomoč o sintaksi ukazov nudi ukaz **man**:

```
$ man [ime_ukaza]
```

Tukaj se argument ukaza **man** nanaša na **ime_ukaza**. Večina operacijskih sistemov Linux vključuje strani **man** za dokumentacijo. Te strani nudijo veliko več kot običajna zastavica **-help** oz. **-h** pri klicanju ukaza. Strani s pomočjo **man** so lahko shranjene v različnih področjih na disku.

Kot primer uporabe ukaza v Linuxu vzemimo kopiranje datotek, ki ga poženemo kot:

```
$ cp a.txt b.txt
```

Ukaz zahteva dva argumenta, kjer prvi argument **a.txt** predstavlja izvorno, drugi argument **b.txt** pa ponorno ime datoteke.

Osnovni ukazi operacijskega sistema Linux so prikazani v Tabeli 4.1, iz

Tabela 3.1: Osnovni ukazi v Linuxu.

Delo z datotekami	**cd, ls, cp, mv, rm, find, more, tail, wc, grep, mkdir, touch, head, tail**
Dovoljenja in lastn.	**chmod, chown, sudo**
Delo z uporabniki	**useradd, usermod, userdel, passwd**
Delo s procesi	**ps, top, kill, nice**
Delo z diski	**df, fdisk, du, mount, unmount**
Tiskanje	**lpr, lpq, lprm**
Omrežji	**telnet, rlogin, ssh, ifconfig, netstat, route**
Ostalo	**man, date, who, vim, nano...**

katere lahko razberemo, da te lahko razdelimo glede na področje uporabe v ukaze za delo z datotekami, dovoljenji in lastništvom, uporabniki, procesi, diski, omrežji, s tiskalniki in ostalim. Med ostale ukaze prištevamo: ukaz za pomoč, urejevalnike in različna orodja (npr. ukaz **date** prikaže sistemski datum in čas).

3.2.1 Ukazi za delo z datotekami

Ta družina ukazov omogoča delo z datotekami in predstavlja osnovne veščine, ki jih mora osvojiti vsak novi uporabnik na operacijskem sistemu Linux. S pomočjo teh ukazov lahko uporabnik operira nad datotekami z naslednjimi akcijami:

- listanje,
- kopiranje,
- preimenovanje,
- brisanje.

Za listanje datotek v ukazni lupini uporabljamo ukaz **ls**, ki ga pogosto poganjamo s stikalom **-l**, ki omogoča izpis trenutnega direktorija v dolgem formatu, kot na primer:

```
$ ls -l a.txt
-rw-rw-r– 1 user group 1624232 Jun 26 20:27 a.txt
```

Kopiranje datotek omogoča ukaz **cp**, ki zahteva dva argumenta (tj. izvorno in ponorno ime datoteke), kot na primer:

```
$ cp a.txt b.txt
$ ls -l *.txt
-rw-rw-r– 1 user group 1624232 Jun 26 20:27 a.txt
-rw-rw-r– 1 user group 1624232 Jul  11 10:23 b.txt
```

Rezultat izvajanja ukaza **cp** lahko vidimo v direktoriju z ukazom **ls -l *.txt**, ki izpiše vse datoteke tipa **.txt**.

Če želimo preimenovati datoteko **a.txt** v **c.txt**, uporabimo ukaz **mv**, ki prav tako zahteva dva argumenta (tj. izvorno in ponorno datoteko), kot na primer:

```
$ mv a.txt c.txt
$ ls -l *.txt
-rw-rw-r– 1 user group 1624232 Jul  11 10:23 b.txt
-rw-rw-r– 1 user group 1624232 Jun 26 20:27 c.txt
```

Datoteke brišemo z ukazom **rm**, ki mu dodamo seznam datotek, ki jih želimo izbrisati, kot argumente ukaza, kot na primer:

```
$ rm a.txt b.txt c.txt
```

Ukaz zbriše tri datoteke: **a.txt**, **b.txt** in **c.txt**, če te seveda obstajajo v trenutnem direktoriju.

Datoteko na Linuxu najlažje kreiramo z ukazom **echo**, s katerim specifični tekst preusmerimo v izhodno datoteko, kot na primer:

```
$ echo "Tekst.» a.txt
$ ls -l *.txt
-rw-rw-r– 1 user group          7 Jul 11 10:35 a.txt
-rw-rw-r– 1 user group 1624232 Jul 11 10:23 b.txt
```

Poti do datotek

Do datotek datotečnega sistema Linux lahko dostopamo absolutno ali relativno. Pri absolutnem dostopu zapišemo celotno pot do določene datoteke začenši od korenskega direktorija /, kot na primer:

```
/home/user/Documents/a.txt
```

Pod do direktorija, v katerem se nahaja datoteka **a.txt**, začnemo opisovati s korenskim direktorjiem / in nadaljujejo s poimenovanji ustreznih poddirektorijev na poti (npr. **home**/**user**/**Documents**/).

Pri relativnem dostopu do datotek začnemo s trenutnim direktorijem. Tega na operacijskem sistemu Linux označimo s simbolom **./** in nadaljujemo opisovanje poti do želene datoteke z ustreznimi poddirektoriji, kot na primer:

```
./Documents/a.txt
```

Če predpostavimo, da se trenutno nahajamo v domačem direktoriju /**home**/**user**, lahko pot do datoteke **a.txt**, ki se nahaja v poddirektoriju **Documents** domačega direktorija, začnemo s simbolom za trenutni direktorij **./**.

Domači direktorij uporabnika ima poseben pomen na sistemu Linux, zato ga lahko označimo krajše s simbolom ∼, ki označuje absolutno pot do njegovega domačega direktorja. Absolutno bi lahko zapisali pot do datoteke **a.txt** v domačem direktoriju **Documents** tudi kot:

```
∼/Documents/a.txt
```

S simbolom **../** se na sistemu Linux sklicujemo na starševki direktorij trenutnega direktorija, na primer:

```
$ ls ../∼
```

Ukaz izpiše vse domače direktorije uporabnikov na sistemu Linux.

4 | Skriptno programiranje v Linuxu

Ukaz **sh** predstavlja interpreter skriptnega jezika **bash**, ki omogoča avtomatizacijo opravil, ki bi jih drugače morali reševati s tipkanjem ukazov v ukazni lupini. Skripta **bash** je standardno orodje za avtomatizacijo poslov na operacijskem sistemu Linux, ki pa ga v zadnjem času močno izpodriva skriptno programiranje v programskih jezikih Python in Ruby. Pri tem velja, če je skripta **bash** dolga do 50 vrstic, je že čas, da uporabimo možnosti skriptnega programiranja v programskih jezikih Python ali Ruby [6].

Standardna orodja za procesiranje teksta na operacijskem sistemu Linux so: **grep**, **sed** in **awk**. Vsa omenjena orodja obdelujejo datoteke s sekvenčnim bralnikom vhodnih vrstic, ki jih filtrirajo s pomočjo regularnih izrazov in naprej modificirajo. Ukaz **grep** je namenjen filtriranju tistih vhodnih vrstic, ki se ujemajo z določenimi vzorci znakov. Ukaz **sed** je sekvenčni urejevalnik, ki omogoča iskanje in modificiranje vhodnih vrstic s pomočjo programskih ukazov. Ukaz **awk** je popoln programski jezik, ki lahko procesira vhodni tekst, dela primerjave ali izvaja aritmetične operacije nad izbranim nizom znakov [9].

Python je visokonivojski programski jezik, ki se pogosto uporablja tudi v sistemski administraciji Linux [10]. Njegova enostavna sintaksa je idealna izbira za reševanje problemov sistemskih administratorjev, pisanje skript in razvoj orodij v okolju Linux. Z uporabo tega programskega jezika lahko sistemski administratorji poenostavijo svoje delo, povečujejo učinkovitost in dosežejo globlji vpogled v delovanje sistema Linux.

4.1 Osnove skriptnega programiranja bash

Skripta **bash** je tekstovna datoteka tipa **.sh**, ki vsebuje niz ukazov. Interpreter skriptnega jezika **bash** jih bere zaporedoma in jih izvaja tako, kot da bi bili vnešeni prek tipkovnice. Skupaj z ukaznim interpreterjem predstavljata močan vmesnik do operacijskega sistema Linux. Na sistemu Linux največkrat uporabljamo interpreter skriptnega jezika **bash**.

Tipična struktura skripte **kdo_sem.sh** je naslednja:

```
#!/bin/bash
whoami
```

Ukaz v prvi vrstici je pragmatični komentar, ki pokliče interpreter skriptnega jezika **bash**. Pragmatični komentar se začenja s simbolom #, podobno kot vsi stavki, ki označujejo komentar, in se nadaljuje s simbolom !. Ta stavek se ne interpretira, ampak pomeni navodilo ukaznemu interpreterju, da naj za interpretiranje skripne datoteka uporabi interpreter skriptnega jezika **bash**. Ukaz v drugi vrstici **whoami** je regularen ukaz Linux in ga ta pošlje v izvajanje ukaznemu interpreterju.

Skripto **bash** lahko poženemo na dva načina:

1. s pomočjo interpreterja skriptnega jezika **sh**:

```
$ sh kdo_sem.sh
user
```

2. neposredno z imenom:

```
$ ./kdo_sem.sh
user
```

V prvem primeru zaženemo novo instanco interpreterja skriptnega jezika **sh**, kjer interpretiramo skriptno datoteko **kdo_sem.sh**. Skripta mora v drugem primeru imeti dovoljenje za izvajanje. Dovoljenje za izvajanje dodamo z ukazom:

```
$ chmod +x kdo_sem.sh
```

Simbol ./ pove ukaznemu interpreterju, da se skripta **kdo_sem.sh** nahaja v trenutnem imeniku. Rezultat izvajanje skriptne datoteke je v obeh primerih enak in izpiše ime uporabnika, prijavljenega na sistem Linux.

Vnos in izpis podatkov

Za vhodno-izhodne operacije uporablja skripta **bash** več ukazov. Izhodni niz znakov lahko izpišemo na dva načina, tj. z ukazom **echo** oz. **printf**. Ukaz **echo** je robustnejši in omogoča izpis niza znakov v neformatirani obliki, medtem ko je ukaz **printf** kompleksnejši in omogoča formatirani izpis. Kot primer vzamimo primerjavo uporabe obeh:

```
$ echo "\tena\tdva\ttri\n"
\tena\tdva\ttri\n
```

oziroma:

```
$ printf "\tena\tdva\ttri\n"
ena    dva    tri
```

V prvem primeru dobimo transparentni izpis niza znakov. V drugem primeru pa se simbola \t in \n interpretirata in imata podoben po-

men, kot pri uporabi funkcije **printf** v programskem jeziku C [11], tj. simbol \t je tabulator, medtem ko simbol \n označuje preskok v novo vrstico.

Za vnos podatkov s tipkovnice lahko uporabimo ukaz **read**, kot na primer v skriptni datoteki **enter.sh**:

```
#!/bin/sh
echo -n "Vnesi ime: "
read ime
if [-n "$ime"]; then
   echo "Pozdravljen $ime!"
   exit 0
else
   echo "Ime ni bilo vnešeno."
   exit 1
fi
```

Po pragmatičnem komentarju skripta **enter.sh** izpiše niz znakov, s katerim zahteva od uporabnika vnos imena. Ta niz izpišemo z ukazom **echo -n**, kjer stikalo **-n** pomeni, da skočimo v novo vrstico po samem izpisu. Ukaz **read** prebere ime prek tipkovnice. Če vnesemo pravilno ime, kar testiramo z vejitvenim stavkom **if [-n "$name"]**, kjer stikalo **-n** označuje neprazen niz, ga izpišemo. V primeru da je vnešen prazen niz, pa izpišemo sporočilo o napaki.

Primer izvajanja skripte izgleda takole:

```
$ sh enter.sh
Vnesi ime: Jože
Pozdravljen, Jože!
```

Argumenti ukazne lupine

Skripte **bash** lahko zaganjamo v ukazni lupini tudi z argumenti, na katere se sklicujemo znotraj skript s spremenljivkami, katerih ime pre-

stavlja zaporedno številko argumenta. Prvi argument označimo kot **$1**, naslednjega kot **$2** itd. Poseben pomen ima spremenljivka **$0**, ki označuje celotno ukazno vrstico. Spremenljivka $# označuje število argumentov ukazne vrstice.

Primer uporabe argumentov skriptnih datotek prikazuje naslednja skripta **preimenuj.sh**:

```
#!/bin/sh
mv $1 $2
```

Skripto **preimenuj.sh** zaženemo, kot kaže naslednji primer:

```
$ chmod +x ./preimenuj.sh
$ ./preimenuj a.txt b.txt
```

V prvem ukazu dodamo skripti pravice za izvajanje, v drugi vrstici pa skripto poženemo neposredno z imenom. Pri tem interpreter skripnega jezika **bash** priredi prvi argument **a.txt** spremenljivki **$1**, drugi argument **b.txt** pa spremenljivki **$2**.

Aritmetični izrazi

Kompleksne podatkovne strukture in aritmetika nista največji prednosti skriptnega jezika **bash**, čeprav je osnovna aritmetika še vedno podprta. Vse spremenljivke v skriptah **bash** se obravnavajo kot nizi znakov, zato interpreter skriptnega jezika ne loči med številčno vrednostjo (npr. 1) in pripadajočim znakom (npr '1'). Razlika med obema je v načinu uporabe v samih skriptah, kjer se aritmetični izrazi izračunavajo znotraj okroglih oklepajev, ali drugače **$((...))**.

Poglejmo primer uporabe, ki nazorno odslikava to razliko:

```
#!/bin/sh
a = 1
b = $(($a+4))
echo "$a+$b"
echo "$(($a+$b))"
```

Rezultat skripte sta naslednji dve vrstici:

```
1+5
6
```

Prvi izpis obravnava spremenljivki **$a** in **$b** kot niza znakov, kjer se simbol plus ne obravnava kot operator za združevanje dveh nizov, ampak se enostavno izpiše. Pri drugem izpisu pa izpisujemo rezultat aritmetične operacije seštevanja. Seveda je rezultat aritmetične operacije pri prirejanju spremenljivke **$b** še vedno shranjen v obliki niza znakov in ne številčne vrednosti.

Nadzor poteka

Nadzor poteka izvajamo v skriptnem jeziku **bash** s pomočjo različnih variant 'if-then-else' stavčnega konstrukta, ki mu lahko dodamo tudi vejitev 'else-if'. Vejitveni stavek zaključimo z rezervirano besedo **fi**.

Sintakso vejitvenega stavka v splošni obliki lahko zapišemo kot:

```
if [pogoj_1]; then
    stavčni_blok_1
elif [pogoj_2]; then
    stavčni_blok_2
else
    stavčni_blok_3
fi
```

Oglati oklepaji **[]** pri testiranju pogojev predstavljajo okrajšavo za ukaz **test** in niso potrebni v originalni sintaksi stavčnega konstrukta. Rezultat testiranja pogoja lahko dobimo z logično ali numerično operacijo. Razlika med testiranjem obeh vrst operatorjev je prikazana v Tabeli 4.1, iz katere lahko razberemo, da nize znakov primerjamo

Tabela 4.1: Operatorji za testiranje pogojev v skriptnem jeziku **bash**.

Logični	Numerični	Rezultat testiranja
$x = y$	x **-eq** y	Pravilno, če je x enak y
$x != y$	x **-ne** y	Pravilno, če je x ni enak y
$x < y$	x **-lt** y	Pravilno, če je x manjši y
$x <= y$	x **-le** y	Pravilno, če je x manjši ali enak y
$x > y$	x **-gt** y	Pravilno, če je x večji y
$x >= y$	x **-ge** y	Pravilno, če je x večji ali enak y

z logičnimi simboli (npr. $=$, $<$ in $>$), numerične spremenljivke pa s simboličnimi operatorji (npr. **-eq**, **-lt** in **gt**).

Lastnosti datotek so pod operacijskim sistemom Linux obširne, zato zanje pri testiranju pogojev določenih lastnosti uporabljamo kar stikala, prikazana v Tabeli 4.2. Naj omenimo, da smo izpostavili samo

Tabela 4.2: Stikala za testiranje različnih lastnosti datotek.

Stikalo	Rezultat testiranja
-d *file*	Pravilno, če *file* obstaja in je mapa
-e *file*	Pravilno, če *file* obstaja
-f *file*	Pravilno, če *file* obstaja in je normalna
-r *file*	Pravilno, če ima uporabnik pravico branja
-s *file*	Pravilno, če *file* obstaja in ni prazna
-w *file*	Pravilno, če ima uporbnik pravico pisanja

najpogostejša stikala, ki so v uporabi.

Primer testiranja lastnosti datoteke *a.txt* na pravico pisanja prikazuje skriptna datoteka **za_branje.sh**:

```
#!/bin/sh
file = a.txt
if [-w $file]; then
   echo "Uporabnik ima pravico do pisanja datoteke $file."
elif [-e $file]; then
   echo "Uporabnik nima pravice do pisanja datoteke $file."
else
   echo "Datoteka $file ne obstaja."
fi
```

Zagon skripte **za_branje.sh** bi v primeru da datoteka **a.txt** obstaja in ima uporabnik pravico do pisanja vanjo, izgledal takole:

```
$ ./za_branje
Uporabnik ima pravico do pisanja datoteke a.txt.
```

Zančni stavki

Zančni stavki v skriptah **bash** omogočajo ponavljanje bloka stavkov, kjer lahko pogoj ustavljanja določimo odvisno od uporabljenega zančnega stavka. Skripte **bash** podpirajo tri zančne stavke: **for**, **while** in **until**. Stavek **for** podpira dve obliki sintakse. Sintaksa prve oblike je naslednja:

```
for item in [LIST]
do
   [UKAZI]
done
```

Tukaj blok stavkov zapišemo med rezervirani besedi **do**...**done**, pogoj ustavljanja pa definiramo z množico elementov, ki jo lahko zapišemo eksplicitno (kot npr. LIST=1,2,3) ali implicitno (kot npr. rezultat izvajanja določenega ukaza (npr. **ls -l**), pri čemer množico LIST določa število izpisnih vrstic).

Sintaksa druge oblike stavka **for** je bližje programerjem v programskem jeziku C in je prikazana v naslednjem primeru:

```
for ((item=1;item<100;item++)); do
    echo "Vrstica: $item"
done
```

Sintaksa stavka **while** je zelo podobna kot v programskem jeziku C in je naslednja:

```
while [ POGOJ_PRAVILNO ]; do
    [UKAZI]
done
```

Tudi tukaj blok stavkov označimo podobno kot pri stavku **for**. Zanka se izvaja, dokler je pogoj ustavljanja POGOJ_PRAVILNO pravilno.

Sintaksa stavka **until** je enaka sintaksi stavka **while**, z drugimi besedami:

```
until [ POGOJ_NEPRAVILNO ]; do
    [UKAZI]
done
```

Razlika med obema stavkoma je v tem, da se v drugem primeru blok stavkov izvaja, dokler je pogoj ustavljanja POGOJ_NEPRAVILNO nepravilno.

4.2 Regularni izrazi, grep, awk, in sed

4.2.1 Regularni izrazi

Regularni izrazi (krajše **regex**) so standardni vzorci, s katerimi razčlenjujemo in obdelujemo nize znakov. Podprti so v večini modernih

programskih jezikov. Lahko jih uporabljamo v nekaterih ukazih operacijskega sistema Linux, kot npr. **grep** in **vi/vim**. Kot taki ne predstavjajo skriptnega jezika, ampak omogočajo primerjanje vhodnega iskalnega niza z iskanim vzorcem. Pri tem iščemo ujemanje iskalnega niza z iskanim vzorcem, ki se lahko ujema z nizom na poljubni poziciji. Po uspešnem ujemanju evaluator regularnih izrazov vrne vse nize znakov, ki se ujemajo z vzorcem.

V splošnem se znaki v regularnem izrazu ujemajo s samim seboj. Tako se vzorec "To je vzorec." ujema z vsemi iskalnimi nizi, kjer se ta pojavi, neglede na njegovi pozicijo v nizu. Iskalni niz, na primer:

"Poljuben tekst pred. To je vzorec. Poljuben tekst potem."

se ujema z iskalnim vzorcem "To je vzorec.", pri čemer je pomembna tudi velikost črk.

Pri definiciji regularnih izrazov uporabljamo posebne znake, s katerimi sestavljamo iskalni vzorec. Seznam najvažnejših posebnih znakov je prikazan v tabeli 4.3. Za popoln seznam lahko zainteresirani bralec pogleda publikacijo [12].

Tabela 4.3: Posebni znaki v regularnih izrazih.

Znak	Ujemanje
.	Ujema se z vsemi znaki
$[A-Z]$	Ujema se z vsemi znaki A-Z
^	Ujema se z začetkom vrstice
$	Ujema se s koncem vrstice
$\backslash w$	Ujema se z vsemi črkami in številkami, tj. A-Za-z0-9_
$\backslash d$	Ujema se s številkami 0-9
|	Ujema se z vzorcem na levi ali desni
$(expr)$	Ujemanje skupine elementov
?	Dovoljuje nič ali eno ujemanje predhodnega elementa
$*$	Dovoljuje nič, eno ali več ujemanj predhodnega elem.
$+$	Dovoljuje nič ali več ujemanj predhodnega elem.

Naslov IPv4 je sestavljen iz 4 bajtov, ki označujejo števila intervalu [0,255]. Če bi želeli ujemanje iskalnega niza z iskanim vzorcem, bi ustrezen regularni izraz zapisali na zelo enostaven način kot:

```
^(?:[0-9]1,3)3[0-9]1,3$
```

Če želimo najti ujemanje naslova v formatu IPv4, je potrebno preveriti števila [0-9], zaključena s piko tri krat in končati s končnim bajtom, tj. z največ tremi števili (predpis {1,3}) v intervalu [0-9]. Seveda pa je regularni izraz preenostaven, saj obravnava trimestna števila v intervalu [0,999] namesto v intevalu [0,255]. Če želimo upoštevati tudi pravilne vrednosti posameznih bajtov, je potrebno spremeniti regularni izraz kot:

```
^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)3(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$
```

Dokaz pravilnosti delovanja regularnega izraza je prepuščen bralcu samemu. Naj omenimo, da za testiranje regularnih izrazov obstaja celotna paleta evaluatorjev, kot na primer **https://regex101.com/**.

4.2.2 Iskalnik vzorcev grep

Ukaz **grep** išče iskane vzorce v vhodnih nizih znakov in izpisuje vse vrstice, ki se ujemajo z njimi. Ime ukaza je okrajšava za **g**lobal **r**egular **e**xpression search and **p**rint. Ta je bil originalno razvit na operacijskem sistemu UNIX in ima naslednjo sintakso:

grep -options <regex> file

kjer z zastavicami **options** krmilimo obnašanje ukaza, **regex** označuje regularni izraz, s katerim definiramo iskalni vzorec in **file** predstavlja pot do vhodne datoteke, nad katero izvajamo ujemanje vzorcev. Najpomembnejše zastavice **options** so prikazane v Tabeli 4.4.

Primer uporabe ukaza **grep** je prikazan v obliki naslednjega cevovoda:

Tabela 4.4: Stikala pri ukazu **grep**.

Stikalo	Pomen
-c	Izpiše število vrstic z ujemanjem
-i	Ne upošteva velikost črk pri ujemanju
-o	Namesto celotne vrstice izpiše samo iskani vzorec
-v	Izpiše vrstice, ki se ne ujemajo z iskanim vzorcem
-P	Uporabi regularne izraze v obliki Perl

```
$ ls -l | grep -o *.txt | wc -l
5
```

Tukaj prvi ukaz izpiše vse datoteke v trenutnem direktoriju v dolgi obliki. Izhodne vrstice prvega ukaza se posredujejo ukazu **grep**, ki posreduje zadnjemu ukazu samo tiste vrstice, ki vsebujejo iskani vzorec **.txt**, tj. imena vseh tekstovnih datotek. Ukaz **wc -l** prešteje število izhodnih vrstic in ga izpiše. V našem primeru najdemo v trenutnem direktoriju pet tekstovnih datotek.

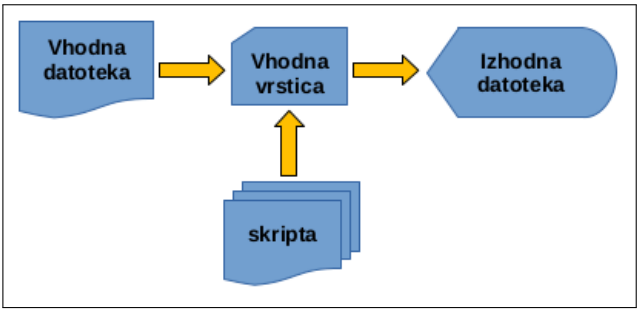
4.2.3 awk

Skriptni interpreter programskega jezika **awk** je nastal okrog leta 1970 in je dobil ime po začetnicah priimkov njegovih avtorjev [13]. Sintaksa jezika je zelo podobna programskemu jeziku C.

Prednosti uporabe programskega jezika **awk** so naslednje [14]:

- je podatkovno orientiran,
- uporablja regularne izraze pri ujemanju vzorcev,
- uporabniku omogoča nadzor prek skriptnih datotek.

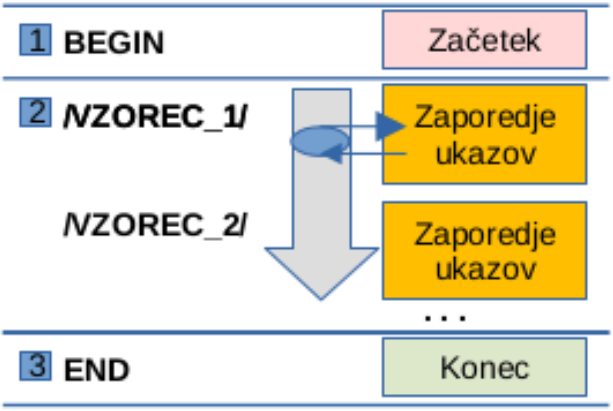
Princip delovanja interpreterja programskega jezika **awk** je prikazan na Sliki 4.1, iz katere lahko razberemo, da ta obdeluje vhodne podatke, ki jih dobi prek standardnega vhoda STDIN zaporedno (tj. vrstico za vrstico) in jih preoblikovane pošilja na standardni izhod STDOUT. S



Slika 4.1: Princip delovanja ukazov **awk** in **sed**.

pomočjo preusmerjanja lahko standardni vhod oz. izhod preusmerimo v datoteko na disku.

Preoblikovanje vhodnih vrstic poteka na podlagi ukazov, ki jih interpreter dobi prek skriptnih datotek v obliki *vzorec/akcija*. Programski model interpreterja **awk** prikazuje Slika 4.2, iz katere lahko razberemo,



Slika 4.2: Programski model skriptnega interpreterja awk.

da ta sestoji iz treh komponent:

1. bloka **BEGIN**,
2. glavne zanke sestavljene iz zaporedij *vzorec/akcija*,
3. bloka **END**.

Bloka **BEGIN** in **END** se izvedeta samo enkrat, tj. prvi ob zagonu skripte, drugi pa ob njenem zaključku. Podatkovni tok je usmerjen v glavno zanko, ki se zažene za vsako vrstico posebej. Vsaka vhodna vrstica se med obdelavo primerja z vzorcem in če interpreter najde ujemanje, to ustrezno preoblikuje v skladu s predpisano akcijo. Akcije v programskem jeziku **awk** so lahko programski stavki oz. funkcije. V primeru da skripta ne podpira več vzorcev, se vse vrstice preoblikujejo na podlagi privzete akcije.

Sintaksa ukaza **awk**, s katerim poženemo interpreter, je naslednja:

```
$ awk [options] script filename
```

S stikali **options**, ki niso obvezna, nadzorujemo vedenje interpreterja, argument **script** določa blok stavkov **awk** in **filename** ime vhodne datoteke. Najpogosteje uporabljena stikala ukaza **awk** so prikazana v tabeli 4.5. Argument **script** lahko interpreterju podamo na dva na-

Tabela 4.5: Stikala pri klicu ukaza **awk**.

Stikalo	Pomen
-f	Vpeljuje ime skriptne datoteke.
-F	Sprememba separatorja polja.
-v	Inicializacije spremenljivk oblike $var = vrednost$.

čina: (1) eksplicitno (zaporedje programskih stavkov znotraj enojnih narekovajev ') ali (2) implicitno (uporaba stikala **-f** in **imena** skriptne datoteke). Skriptna datoteka tipa **.awk** je naslednje oblike:

```
#!/bin/awk
blok stavkov
```

V pragmatičnem komentarju v prvi vrstici najprej poženemo interpreter skripnega jezika **awk**. Temu sledi blok programskih stavkov zapisanih v programskem jeziku **awk**. Podobno kot skripte **bash** tudi skripte **awk** lahko uporabljajo argumente vhodne vrstice, na katere se sklicujemo s spremenljivkami $1, $2 . . .

Primer zagona interpreterja **awk** je naslednji:

```
$ awk '/SLO/' list.txt
Jože Novak, Vrtna 5, 2000 Maribor, SLO
```

Ukaz izpiše vse vrstice datoteke **list.txt**, ki se ujemajo z vzorcem 'SLO'. V skriptah **awk** so vsi vzorci zapisani med dvema simboloma / (tj. v obliki **'/vzorec/'**).

V prejšnjem primeru smo interpreter **awk** zagnali s prvo komponento *vzorec/akcija*, tj. vzorcem. Primer zagona interpreterja, ki se nanaša na akcijo, pa je naslednji:

```
$ awk '{ print $1}' list.txt
Jože
```

Skripta izpiše prvi argument vsake vrstice, ki se v našem omejenem prikazu nanaša na ime Jožeta Novaka, stanujočega na Vrtni 5 v Mariboru (Slovenija).

V zadnjem primeru zaženimo interpreter **awk** z uporabo skripte **skripta.awk**. Skripto zapišemo v naslednji obliki:

```
#!/bin/awk
BEGIN {}
{ print $1 }
END {}
```

Kot lahko ugotovimo iz definicije skripte **skripta.awk**, sta začetni in zaključni blok prazna, skripta pa izvaja isti blok stavkov kot v prejšnjem primeru. Zagon interpreterja z uporabo skripte **awk** je

naslednji:

```
$ awk -f skripta.awk list.txt
Jože
```

Tudi v tem primeru je rezultat izvajanja skripte ostal podoben kot pri izvajanju ukazov. V splošnem uporabljamo skripte **awk** v primeru, da je število programskih stavkov preveliko (npr. večje kot pet) oz. so ti stavki prekompleksni, da bi jih lahko zapisali v ukazno vrstico.

Skripte **awk** predstavljajo močno orodje za sistemskega administratorja, saj omogočajo uporabo:

- spremenljivk,
- aritmetično-logičnih operatorjev,
- aritmetično-logičnih izrazov,
- vejitvenega stavka **if**,
- zančnih stavkov **for**, **while** in **do**,
- zapisov in polj.

Skripte **awk** v sistemski administraciji velikokrat uporabljamo v primerih, ko želimo analizirati obnašanje določenih programov prek sistemov za beleženje aktivnosti, incidentov oz. informacij o klicih. Še posebej primerne so za hitro preoblikovanje velike količine izhodnih rezultatov v agregatne tabele oblike Latex.

4.2.4 sed

Sekvenčni urejevalnik teksta **sed** je skovanka dveh angleških besed **s**equential+**ed**, kjer se **ed** nanaša na izvorni vrstični urejevalnik teksta na sistemu Unix. Podobno kot **awk** je tudi **sed** vrstično orientiran in pričakuje vhod iz standardne datoteke STDIN ter pošilja preoblikovane vrstice na standarni STDOUT. Uporablja regularne izraze, uporabniki pa ga krmilijo z ukazi neposredno prek tipkovnice oz.

zbranimi v skriptnih datotekah. Ukazi **sed** niso konstrukti programskega jezika, ampak predstavljajo ukaze urejevalnika za preoblikovanje teksta. Za razliko od interpreterja **awk**, ki je usmerjen na elemente v vrstici, katerih pri tem ne preoblikuje, pa **sed** omogoča tudi modificiranje teh elementov samih.

Sintaksa sekvenčnega urejevalnika **sed** je naslednja:

```
$ sed [options] script filename
```

V prikazani sintaksi ukaza se spremenljivka **options**, ki ni podana obvezno, nanaša na stikala, s katerimi krmilimo vedenje urejevalnika, spremenljivka **skript** lahko predstavlja niz ukazov urejevalnika znotraj enojnih narekovajev oz. ime skriptne datoteke, v kateri te ukaze shranimo, in se spremenljivka **filename** nanaša na ime vhodne datoteke, ki jo obdelujemo.

Stikala, ki jih najpogosteje uporabljamo pri klicu ukaza **sed**, so prikazana v Tabeli 4.6.

Tabela 4.6: Stikala pri klicu ukaza **sed**.

Stikalo	Pomen
-f	Vpeljuje ime skriptne datoteke.
-e	Vpeljuje nize 'ukaz' za preoblikovanje elementa.
-n	Preprečuje izpis izhodne vrstice.

Ukazi urejevalnika **sed** so prikazani v Tabeli 4.7. Kot lahko razberemo

Tabela 4.7: Ukazi za preoblikovanje elementov vhodne vrstice.

Stikalo	Pomen
/pattern/s/match/sub/g	Nadomesti vzorec 'match' z vzorcem 'sub'.
/pattern/d	Izbriši vrstico.
/pattern/a <line>	Dodaj <**line**> za tekočo vrstico.
/pattern/i <line>	Vstavi <**line**> pred tekočo vrstico.

iz Tabele 4.7, so tudi ukazi **sed** orejeni v obliki *vzorec/akcija*, saj

vsakemu ukazu predhodi regularni izraz /**pattern**/, s katerim se mora ujemati vhodna vrstica, ki jo želimo preoblikovati. Tega nadaljuje ukaz za urejanje, tj. akcija (npr. **s**, **d**, **a**, **i**), ukazni niz pa zaključijo parametri tega ukaza.

Enostaven primer uporabe ukaza **sed** je naslednji:

```
$ sed 's/SLO/Slovenija/' list.txt
Jože Novak, Vrtna 5, 2000 Maribor, Slovenija
```

Sekvenčni urejevalnik **sed** je v vhodni vrstici zamenjal vzorec 'SLO' z vzorcem 'Slovenija'.

V istem ukazu lahko kličemo tudi več ukazov urejanja. To lahko naredimo na dva načina. Prvi način klica več ukazov urejanja je povezovanje več ukazov s simbolom **;**, kot na primer:

```
$ sed 's/SLO/Slovenija/; s/ITA/Italija/' list.txt
```

Urejevalnik bo v tem primeru obravnaval vse vrstice, ki vsebujejo vzorec 'SLO' ali vzorec 'ITA' in jih ustrezno preoblikoval.

Drugi način izkorišča uporabo stikala **-e**. V našem primeru bi isto funkcionalnost kot zgoraj dosegli na naslednji način:

```
$ sed -e 's/SLO/Slovenija/' -e 's/ITA/Italija/' list.txt
```

Tukaj urejevalniku ukaze za preoblikovanje teksta podamo eksplicitno prek dveh ukazov in stikala **-e**.

Najenostavnejša pot urejanja vhodnega teksta pa predstavlja uporaba skriptnih datotek. Predpostavimo, da želimo našo vhodno datoteko **list.txt** spremeniti z naslednjimi ukazi, zbranimi v datoteki **scriptfile**:

```
$ cat scriptfile
s/SLO/Slovenija/
s/ITA/Italija/
s/AUT/Avstrija/
s/CRO/Hrvatska/
s/HUN/Madžarska/
```

Urejevalnik teksta **sed** poženemo z naslednjim ukazom:

```
$ sed -f scriptfile list.txt
```

Rezultat delovanja skriptne datoteke je preoblikovanje vseh vrstic, ki vsebujejo kratico naše oz. kratice naših obmejnih držav, v regularno ime države.

Skripte **sed** pogosto uporabljamo v primerih, ko je potrebno spreminjati vsebino konfiguracijskih tekstovnih datotek. Večina teh sprememb zahteva enostavne operacije dodajanja, brisanja in spreminjanja določenega polja v konfiguracijski datoteki (npr. imena uporabnika). Spremembo istega polja je pogosto potrebno narediti v več konfiguracijskih datotekah. Če naredimo spremembo samo v eni konfiguracijski datoteki, ne pa v vseh, lahko pride do odpovedi določene sistemske funkcije.

4.3 Sistemska administracija v Pythonu

Python je odprtokodni splošno-namenski programski jezik, ki združuje različne stile programiranja: objektnoorientirani, proceduralni in funkcijski [15]. Ponuja dobro interaktivno okolje in ima enostavne vmesnike do drugih programskih jezikov (npr. C/C++, Java, Fortran). Interaktivni vmesnik omogoča pisanje skript podobno kot **bash** in **awk**. Poglejmo si enostavno skripto **helloworld.py** v Pythonu:

```
#!/usr/bin/python
print "Hello world."
```

Če želimo skripto izvajati kot ukaz, je potrebno poskrbeti za njene pravice izvajanja, z drugimi besedami:

```
$ chmod +x helloworld.py
$ ./helloworld
Hello world.
```

Uporaba Pythona za skriptno programiranje je, kot lahko vidimo iz primera, zelo enostavna. Kljub vsemu pa za naprednejšo uporabo tega potrebujemo določene osnove programskega jezika. V nadaljevanju poglavja zato podajamo osnove Pythona, s katerimi lahko tudi začetnik hitro osvoji principe programiranja v tem jeziku.

4.3.1 Značilnosti programskega jezika Python

Sintaksa programskega jezika Python je poenostavljena, saj ne vsebuje deklaracij spremenljivk oz. eksplicitne uporabe stavčnih blokov. Namesto eksplicitne deklaracije spremenljivk tukaj uporabljamo implicitno deklariranje, kjer tip spemenljivke določimo iz konteksta prireditvenega stavka. Po drugi strani blok stavkov programer določa implicitno s pomočjo zamikanja. Poglejmo primer programa **sample.py**, zapisanega v tem programskem jeziku:

```
1  #!/usr/bin/python
2  x = 5
3  y = "Hello"
4  z = 2.45
5  if z == 2.45 or y == "Hello":
6      x = x + 1
7      y = y + " world".
8  print y
```

Po zagonu programa lahko pričakujemo naslednji rezultat:

```
./sample
Hello world.
```

Iz prikazanega primera lahko razberemo, da interpreter Python določi tipe premenljivk **x**, **y** in **z** ob prirejanju ustreznih vrednosti. Tako spremenljivka **x** dobi tip **integer**, spremenljivka **y** tip **string** in spremenljivka **z** tip **float**. Vejitveni stavek **if** uporablja za primerjanje enakosti simbol ==. Z zamikanjem programer določi, da se blok stavkov v vrsticah 6 in 7 izvrši, če je pogoj logične operacije pravilno. Logični operatorji niso simboli, ampak rezervirane besede **and**, **or** in **not**. Ukaz za izpis je **print**, podobno kot pri skriptah **bash**.

Tipi spremenjivk

Python loči osnovne in sestavljene podatkovne tipe. Osnovne podatkovne tipe, ki smo jih spoznali že pri obravnavanem primeru, so naslednji: **integer**, **float** in **string**. Spremenljivkam tipa **string** lahko prirejamo nize znakov znotraj enojnih ali dvojnih narekovajev. Narekovaje lahko uporabljamo tudi znotraj niza (npr. **"Matt's"**).

Sestavljeni podatkovni tipi so trije: **tuple**, **list** in **dictionary**. Elementi tipa **tuple** so lahko mešanih tipov in predstavljajo enostavno urejeno zaporedje, ki je namenjeno samo za branje, kot na primer:

```
tuple = ( 'abcd', 789, 1.23, 'Jože', 72.4 )
```

Elemente tipa **list** lahko spreminjamo, dodajamo in brišemo, kot na primer:

```
list = [ 'abcd', 789, 1.23, 'Jože', 72.4 ]
```

Tip **dictionary** predstavlja tabelo **hash**, kjer je vsak element sestavljen iz parov *'ključ:vrednost'*. Ključ je tipa **string**, vrednost pa je katerakoli izmed enostavnih podatkovnih tipov, kot na primer:

```
dictionary = { 'ime':'Jože', 'ident':6789, 'oddelek':'prodaja' }
```

Domet spremenljivk

Prirejanje spremenljivk v Pythonu postavi referenco na objekt. Zato imena spremenljivk nimajo predpisanih tipov, ampak jih dobivajo iz konteksta. Njihov domet je blok stavkov, v katerem se pojavi definicija. Referenco na objekt zbrišemo, ko ta pade izven bloka, v katerem smo spremenljivko definirali. Za sproščanje pomnilnika poskrbi t. i. garbage collection samodejno.

Komentarji

Programski jezik Python podpira dve vrsti komentarjev. Začetek komentarja tipično označimo s simbolom #, ki pomeni, da interpreter preostanek vrstice ignorira. Python pa dovoljuje tudi t. i. *dokumentacijske nize*, ki jih vključujemo v prvo vrstico definicije funkcij oz. razredov, kot na primer:

```
def moja_funkcija (x, y):
  """To je dokumentacijski niz.
  Ta fukcija ima naslednje funkcionalnosti..."""
  # To je normalen komentar...
```

Do dokumentacijskih nizov dostopamo na podoben način kot pri uporabi ukaza **man** na operacijskem sistemu Linux. Seveda v interpreterju Python namesto ukaza **man** uporabimo interpreterjev ukaz **help**, kot na primer:

```
>>> help(moja_funkcija)
To je dokumentacijski niz.
Ta fukcija ima naslednje funkcionalnosti...
```

4.3.2 Konstrukti programskega jezika Python

Programski jezik Python omogoča več stavčnih konstruktov, s katerimi nadzorujemo vedenje programa. Funkcije omogočajo razširi-

tve uporabnosti programskega jezika, še posebej, če jih vključujemo v knjižnice. V nadaljevanju poglavja opisujemo omenjene možnosti programskega jezika Python podrobneje.

Vejitveni stavek if

Sintaksa sestavljenega stavka **if** je v programskem jeziku Python naslednja:

```
if <condition_1> :
    <block_of_statements_1>
elif <condition_2> :
    <block_of_statements_2>
else :
    <block_of_statements_3>
```

Sestavljeni stavek **if** nudi razvejitve glede na več pogojev. Vsak izmed pogojev <**condition_i**> moramo zaključiti s simbolom **:**.

Zančni stavki

Programski jezik Python dovoljuje uporabo dveh vejitvenih stavkov: **for** in **while**. Sintaksa stavka **for** je naslednja:

```
for s in S :
    <block_of_statements>
```

Stavek **for** izvede blok stavkov <**block_of_statements**> za vsak element $s \in S$. Če definiramo množico kot $S = \{1, 2, 3, 4, 5\}$, se bo zanka izvedla petkrat.

Sintaksa stavka **while** je naslednja:

```
while <condition> :
    <block_of_statements>
```

Pri tem stavku se bo blok stavkov <**block_of_statements**> izvajal, dokler je pogoj <**condition**> pravilno.

Funkcije

Funkcije v programskem jeziku Python definiramo v skladu z naslednjo sintakso:

```
def ime_funkcije (arg1, arg2,..., argvN):
    <block_of_statements>
    return value
```

Funkcijo definiramo z rezervirano besedo **def**, ki ji sledi ime skupaj z množico argumentov znotraj okroglih oklepajev. Rezultat funkcije posredujemo s pomočjo stavka **return**, ki implicitno določi tudi njen tip. Tipi argumentov se prav tako določajo implicitno ob klicu same fukncije. Določeni argumenti imajo lahko privzete vrednosti, ki jih določamo pri definicije same funkcije.

Primer definicije enostavne funkcije **multiply** za množenje dveh števil **x** in **y** je naslednji:

```
def multiply (x, y = 2):
    return x*y
```

Funkcijo lahko pokličemo na dva načina. V prvem primeru specificiramo oba argumenta funkcije eksplicitno, kot na primer:

```
>>> multiply (5, 4)
20
```

V drugem primeru pa izkoristimo privzeto vrednost drugega argumenta in funkcijo pokličemo kot:

```
>>> multiply (5)
10
```

V prvem primeru smo izvedli matematično operacijo $5 \cdot 4 = 20$, v drugem pa $5 \cdot 2 = 10$.

4.3.3 Moduli za sistemsko administracijo

Moduli v programskem jeziku Python združujejo več programov s končnico **.py**, ki se nanašajo na določeno problemsko domeno. Prednosti funkcij v modulih je predvsem ponovna uporabljivost kode. Moduli omogočajo uporabo imenskih prostorov, kar pomeni, da se lahko imena funkcij znotraj modulov tudi podvajajo [16].

Fukcije modulov lahko kličemo na dva načina. Prvi način uporablja stavek **from** kot:

```
from module import function
function()
```

Tukaj v izvajalno okolje programa naložimo samo funkcijo **function**, ki se nahaja v modulu **module**. Drugi način naloži v izvajalno okolje celoten modul, do posamezne funkcije tega pa dostopamo prek imenskega prostora modula, kot na primer:

```
import module
module.function()
```

Prednost drugega načina je, da modul naložimo samo enkrat, funkcije modula pa uporabljamo ves čas trajanja programa.

Za manipulacijo z operacijskim sistemom Linux je bilo tako razvito večje število modulov, ki zelo olajšajo delo sistemskega administratorja.

Seznam najpomembnejših modulov je predstavljen v Tabeli 4.8 Pri-

Tabela 4.8: Najpomembnejši moduli za delo s sistemom Linux.

Modul	Opis
sys, **os**	Večina sistemskih vmesnikov.
glob	Delo z datotekami.
socket	Medsebojna komunikacija med procesi.
treading, _tread	Sinhronizacija niti.
time, timeit	Dostop do časovnih funkcij.
signal	Nadzor nad procesi.

mer listanja trenutnega direktorija bi v Pythonu izvedli z naslednjim nizom ukazov:

```
>>> import os
>>> os.system(ls -l)
total 30
drwxr-xr-x 2 user user 4096 Jun 28 14:59 Documents
...
```

Kot lahko vidimo iz zadnjega primera, lahko s pomočjo funkcije **system** modula **os** izvajamo iste ukaze kot neposredno v ukazni lupini Linux.

5 | Procesi in delo s procesi

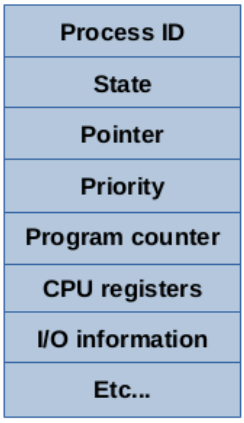
Proces je abstraktni pojem, ki ga uporabljamo v teoriji operacijskih sistemov za predstavitev izvajalnega programa [6]. Ta v času izvajanja zaseda sistemske vire, kot so procesorski čas, primarni pomnilnik in vhodno/izhodne enote. Procesi so lahko sistemski ali uporabniški, čeprav v smislu porabe sistemskih virov med njimi ni razlik, tj. sistem jih obravnava kot enakovredne.

5.1 Komponente procesa

Proces sestoji iz naslovnega prostora in množice sistemskih struktur, povezanih v t. i. programski nadzorni blok (angl. Program Control Block, krajše PCB). Naslovni prostor predstavlja množica pomnilniških strani, ki jih jedro operacijskega sistema Linux prireja določenemu procesu. Te strani sestojijo iz kode aplikacijskih programov oz. programov iz knjižnic, spremenljivk, sklada in ostalih dodatnih informacij, ki jih proces potrebuje med izvajanjem.

Programski nadzorni blok vzdržuje jedro operacijskega sistema Linux

oz. njegova komponenta razporejevalnik opravil (angl. scheduler) in sestoji iz več vrst informacij (Slika 5.1).



Slika 5.1: Programski nadzorni blok.

Najpomembnejše informacije v programskem nadzornem bloku so naslednje [6]:

- identifikacija procesa,
- trenutni status,
- prioriteta,
- programski števec,
- omejitve pomnilnika,
- registri,
- informacije o porabi vhodno-izhodnih virov,
- itd.

V nadaljevanju opisujemo omenjene informacije o procesih v programskem nadzornem bloku podrobneje.

Identifikacija procesa

Identifikacija procesa sestoji iz naslednjih informacij:

- identifikacije procesa (angl. Process ID, krajše PID),
- identifikacije starša procesa (angl. Parent Process ID, krajše PPID),
- identifikacije lastnika procesa (angl. User ID, krajše UID),
- identifikacije skupine lastnikov procesa (angl. Group User ID, krajše GUID),
- specifikacije nadzornega terminala.

PID je unikatna identifikacijska številka, ki jo jedro operacijskega sistema Linux pridruži vsakemu procesu. Identifikacija procesa PID=1 je rezervirana za jedro operacijskega sistema, vsi drugi procesi pa so njegovi *kloni*. Nobeden izmed procesov Unix oz. Linux ne more nastati iz nič, ampak se mora obstoječi proces klonirati in zagnati kot novi proces. Pri kloniranju dobi originalni proces naziv starša (angl. parent), novi proces pa naziv potomca (angl. child). Povedano drugače, starš dobi atribut PPID, potomec pa svoj unikatni PID.

UID se nanaša na lastnika procesa, to je osebo, ki je proces ustvarila. Glede na to, da ga je ta kloniral, hkrati postane tudi lastnik procesa potomca. GID je identifikacija skupine, kateri pripada lastnik. Skupina uporabnikov je pojem v sistemski administraciji Linuxa, ki koncept lastniških pravic razširja, saj je uporabnik običajno član več skupin.

Večina nedemonskih (angl. non-daemon) procesov na operacijskem sistemu Linux ima pridruženo nadzorno ukazno lupino, s katero določamo povezave na standardni vhod STDIN, standardni izhod STDOUT in standardno napako STDERR.

Trenutni status

To polje opisuje trenutni status procesa. O različnih statusih procesov govorimo več v nadaljevanju tega poglavja.

Prioriteta

Prioriteta vpliva na vrstni red obravnavanja procesov v t. i. opravilni vrsti (angl. scheduling queue). Tudi o prioritetah govorimo več v nadaljevanju.

Programski števec

Programski števec hrani naslov naslednje instrukcije, ki jo mora izvesti določen proces.

Omejitve pomnilnika

To polje vsebuje informacije o upravitelju pomnilniškega sistema, ki ga uporablja operacijski sistem Linux. Te informacije se nanašajo na tabelo strani, segmentno tabelo itd.

Registri

V to področje shranjujemo registre centralno-procesne enote (angl. Central Process Unit, krajše CPU) v primeru prekinitev. Registri vključujejo: akumulator, bazni register in splošno namenske registre.

Informacije o porabi vhodno-izhodnih virov

Te informacije vsebujejo seznam datotek, ki jih je odprl določen proces.

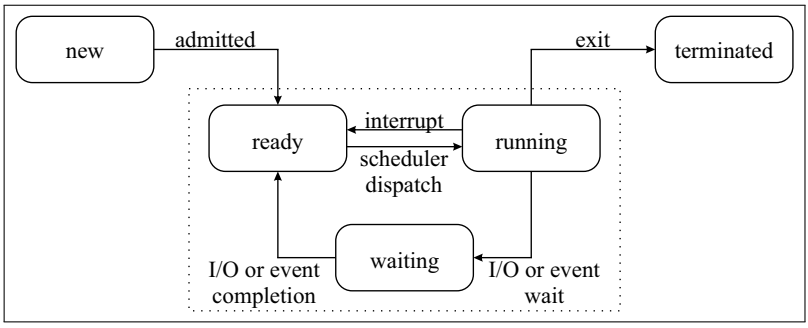
5.2 Življenjski cikel procesov

Vsak process ima svoj življenjski cikel, skozi katerega gre prek več stanj. Kreiranje novega procesa povzroči sistemski klic funkcije **fork**. Ta klonira originalni proces, pridobi unikatno identifikacijsko številko PID in tudi PPID starša, ki ga je kloniral.

Ob zagonu sistema se naloži proces **init** s PID=1, ki je odgovoren za izvajanje zagonskih skript, pomembno vlogo pa ima pri upravljanju s procesi. Ko želi določen proces končati, požene funkcijo **_exit**, ki javi jedru operacijskega sistema, da je ta proces pripravljen končati in mu pošlje ustrezno izhodno kodo. Če gre za uspešen zaključek procesa, je ta koda postavljena na nič.

Preden proces zaključi, jedro operacijskega sistema zahteva, da konec procesa potrdi tudi njegov starševski proces. Če je starševski proces že zaključil, postane potomec sirota (angl. orphan) in ga mora zaključiti jedro.

Življenjski cikel procesa na sistemu Linux je prikazan na Sliki 5.2, iz



Slika 5.2: Življenjski cikel procesa na sistemu Linux.

katere lahko razberemo, da se potomec po kreiranju postavi v opravilno vrsto razporejavalnika poslov (status "ready"). Ko mu razporejevalnik poslov preda nadzor nad procesorjem, dobi status "running". Ko določena časovna rezina poteče, ga ta ponovno postavi v opra-

vilno vrsto. Če zahteva podatke s perifernih naprav (vhodno/izhodna operacija) oz. čaka na kakšen drug dogodek (npr. zaključek drugega procesa), ga razporejavanik opravil postavi v vrsto čakajočih procesov (status "wait"). Ko sproži zahtevo po zaključku, stanje procesa postane "terminated".

Vseh stanj procesov na sistemu Linux sistemski administrator ne more opazovati, ker so nekatera avtomatsko posledica razporejevalnika opravil. Zato na samem sistemu ta običajno lahko opazuje stanja procesov, prikazana v Tabeli 5.1. Stanje **Runnable** označuje, da je proces pri-

Tabela 5.1: Stanja procesov na operacijskem sistemu Linux.

Stanje	Opis
Runnable	Proces je pripravljen na izvajanje.
Sleeping	Proces čaka na določen vir oz. dogodek.
Zombie	Proces je v fazi zaključka.
Stopped	Proces je bil prekinjen administrativno.

pravljen na izvajanje in čaka na časovno rezino razporejevalnika opravil (stanje "ready"). V stanju **Sleeping** je proces, ki čaka na zaključek vhodno/izhodne operacije oz. določen dogodek (stanje "wait"). Proces postane **Zombie**, ko ta sporoči jedru operacijskega sistema, da želi končati, Proces pade v stanje **Stopped**, ko njegovo izvajanje prepoveduje administrator s pošiljanjem signalov STOP oz. TSTP, in se lahko nadaljuje, ko dobi signal CONT.

5.3 Signali

Signali so prekinitvene zahteve, ki se sprožajo na procesnem nivoju. Operacijski sistem Linux pozna okrog 30 različnih signalov, ki se uporabljajo na različne načine, kot na primer:

- signale lahko pošiljajo procesi, ki komunicirajo med seboj;
- izvor signala je lahko terminalski gonilnik, ki sporoča zaključek,

prekinitev oz. začasno zaustavitev procesa (kombinacija tipk <Ctrl-C> ali <Ctrl-Z>);

- administrator pošilja signale, ko želi zaključiti procese na različne načine (ukaz **kill**):
- jedro operacijskega sistema Linux sporoča, da se je proces zaključil oz. da se je končala določena vhodno-izhodna operacija;
- pomnilnik lahko generira signale zaradi strojnih oz. programskih napak.

Proces, ki dobi signal, lahko tega obravnava (angl. catch), če ima za to pripravljeno prekinitveno rutino (angl. interrupt routine) oz. konča s pomnilniškim izpisom jedra (angl. core dump) v primeru, da te rutine nima pripravljene. Proces ima možnost signal tudi blokirati (angl. block), ko ne želimo prekinjati delovanja programa.

Nekaj najpomembnejših signalov, skupaj z njihovimi opisi, prikazuje Tabela 5.2. Velja dogovor, da signale lahko pišemo v daljši ali krajši obliki. Pri daljši obliki imenu signala dodamo predpono SIG. Signal HUP v dolgi obliki bi lahko npr. zapisali tudi kot SIGHUP. V nadaljevanju uporabljamo samo krajše oblike imen signalov.

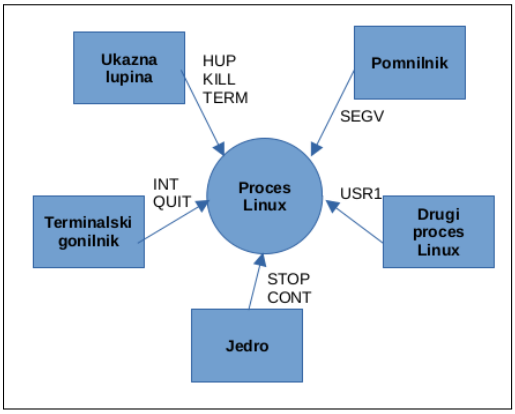
Signal HUP ima dva pomena: (1) signalizira demonskem procesu, da se ta požene znova, ali (2) če je demonski proces sposoben posodobitve novih nastavitev, lahko ta proces zaženemo z novimi nastavitvami s pošiljanjem tega siganala, ne da bi ta proces prekinili. Signal INT je posledica terminalskega gonilnika in se generira, ko ta naleti na kombinacijo tipk <Ctrl-C>, ta pa zahteva zaključek trenutne operacije. Signal TERM zahteva zaključek izvajanja procesa in je podoben signalu QUIT, s to razliko, da QUIT generira pomnilniški izpis jedra, če ta ne podpira prekinitvene rutine. Signala KILL in STOP ni mogoče blokirati oz. prezreti. Prvi signal prekine delujoči proces, drugi pa ga začasno zaustavi, dokler ga signal CONT ponovno ne aktivira. Signala BUS in SEGV sta posledici sistemskih napak: prvi se pojavi zaradi napak pri branju podatkov s perifernih naprav, drugi pa zaradi

Tabela 5.2: Signali na operacijskem sistemu Linux.

Št.	Ime	Opis	Privzeta akcija
1	HUP	Hangup	Terminate
2	INT	Interupt	Terminate
3	QUIT	Quit	Terminate
9	KILL	Kill	Terminate
10	BUS	Bus error	Terminate
11	SEGV	Segmentation fault	Terminate
15	TERM	Software termination	Terminate
17	STOP	Stop	Stop
18	TSTP	Keyboard stop	Stop
19	CONT	Continue after stop	Ignore
28	WINCH	Window changed	Ignore
30	USR1	User defined 1	Terminate
31	USR2	User defined 2	Terminate

dostopa do lokacij primarnega pomnilnika, ki procesu niso bile dodeljene. TSTP je programska verzija signala STOP in ga lahko interpretiramo tudi kot zahtevo po zaustavitvi procesa. Običajno je posledica terminalskega gonilnika in se sproža pri procesiranju kombinacije tipk <Ctrl-Z>. WINCH pošiljajo terminalski emulatorji, ko spremenimo configuracijske parametere terminala (npr. povečamo število vrstic prikazovalnika). Signala USR1 in USR2 nimata določenega pomena in ju običajno uporabljajo alikacijski strežniki za svoje potrebe.

Vrste signalov glede na izvor so prikazane na Sliki 5.3, iz katere lahko ugotovimo, kdo je vir pošiljanja določene vrste signalov. Kot smo omenili že v začetku poglavja, imamo pet izvorov generiranja signalov na operacijskem sistemu Linux.



Slika 5.3: Vrste signalov na operacijskem sistemu Linux.

5.4 Ukazi za nadzor procesov

Ukaze za nadzor procesov lahko rezdelimo v tri skupine, tj. ukaze za:

- pošiljanje signalov,
- spreminjanje prioritet procesov,
- nadzor procesov.

V nadaljevanju poglavja obravnavamo posamezne skupine ukazov podrobneje.

5.4.1 Pošiljanje signalov

Ukaz **kill** omogoča pošiljanje poljubnega signala izvajalnemu procesu. Že iz njegovega imena je jasno razvidno, da je ukaz prvenstveno namenjen pošiljanju signala TERM, kar je tudi njegova privzeta vrednost. Sintaksa ukaza je naslednja:

```
$ kill [-signal] PID
```

Stikalo **-signal** označuje številko signala oz. njegovo simbolično ime (tabela 5.2) in se **PID** nanaša na identifikacijo procesa. Če poženemo ukaz **kill** s privzeto vrednostjo TERM (tj. brez stikala **-signal**), ni zagotovila, da se bo ustrezen proces tudi v resnici končal. Zato je potrebno pognati ukaz v naslednji obliki:

```
$ kill -9 PID
```

Omenjeni ukaz zagotavlja, da se bo proces zagotovo končal, saj mu v tem primeru pošljemo signal za brezpogojni zaključek procesa KILL.

Ukaz **killall** je specifičen, saj deluje različno na sistemu Unix kot na sistemu Linux. Na Linuxu ga poganjamo, ko želimo zaključiti vse procese, na katere se skličemo eksplicitno z imenom, kot na primer:

```
$ sudo killall httpd
```

Posledica tega ukaza je, da zaključi vse procese web strežnika Apache.

5.4.2 Spreminjanje prioritet procesov

Na operacijskem sistemu Linux določimo prioriteto procesov na podlagi koncepta prijaznosti (angl. niceness), tj. kako razporejevalnik opravil obravnava določen proces v odvisnosti od drugih procesov na sistemu. Višja vrednost pomeni nižjo prioriteto in obratno. Vrednosti prioritet na Linuxu so cela števila v intervalu $[-20,19]$. Negativne vrednosti pomenijo višje prioritete.

Pri kreiranju procesa ta dobi prioriteto od svojega starša. Lastnik procesa lahko prioriteto samo zvišuje, kar pomeni, da se do ostalih procesov obnaša prijazno. Administrator lahko prioriteto procesa tudi znižuje. Vrednost prioritete se nanaša samo na porabo CPU, ne pa tudi na hitrost izvajanja vhodno-izhodnih operacij.

Začetno vrednost prioritete določimo z ukazom **nice**, pozneje pa lahko to spremenimo z ukazom **renice**, kot na primer:

```
$ nice -n 5 /usr/bin/moj_program
$ renice -n 5 PID_mojega_programa
```

V prvem ukazu poženemo program **moj_program** z vrednostjo prioritete 5, v drugem pa že delujočemu procesu **PID_mojega_programa** spremenimo prioriteto na 5. Privzeta vrednost prioritete za uporabniške procese je 0.

5.4.3 Nadzor procesov

Procese na operacijskem sistemu Linux običajno nadzorujemo s tremi orodji, tj. ukazi **ps**, **top** in **htop**. Čeprav so funkcionalno podobni, pa posedujejo svoje specifične lastnosti, ki jih opredeljujemo v nadaljevanju. Če želimo pogledati, kaj se dogaja s procesi na nižjem nivoju, lahko uporabimo ukaz **strace**.

Ukaz ps

Predstavlja osnovno orodje za nadzor procesov na operacijskem sistemu Linux in ima bogato zgodovino že na operacijskem sistemu Unix. Njegova prednost je, da prikazuje večino lastnosti delujočih procesov pod Linuxom, kot npr. PID, UID, prioriteto, stanje procesa, nadzorni terminal ipd. Ta prikaz procesov je statičen, kar pomeni, da lahko sistemski administrator vidi samo trenutno stanje, ne pa tudi resničnega dogajanja na sistemu skozi čas (Slika 5.4).

```
$ ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 Jun30 ?        00:00:03 /sbin/init splash
root           2       0  0 Jun30 ?        00:00:00 [kthreadd]
root           3       2  0 Jun30 ?        00:00:00 [rcu_gp]
root           4       2  0 Jun30 ?        00:00:00 [rcu_par_gp]
root           5       2  0 Jun30 ?        00:00:00 [slub_flushwq]
root           6       2  0 Jun30 ?        00:00:00 [netns]
root           8       2  0 Jun30 ?        00:00:00 [kworker/0:0H-events_highpri]
root          10       2  0 Jun30 ?        00:00:00 [mm_percpu_wq]
root          11       2  0 Jun30 ?        00:00:00 [rcu_tasks_rude_]
root          12       2  0 Jun30 ?        00:00:00 [rcu_tasks_trace]
root          13       2  0 Jun30 ?        00:00:00 [ksoftirqd/0]
```

Slika 5.4: Rezultati ukaza **ps**.

Pomanjkljivost ukaza **ps** je, da z leti postaja prekompleksen, saj v izpisu vidimo vse trenutno delujoče procese na sistemu, tj. celotno opravilno vrsto razporejevalnika opravil. Seveda se z razvojem računalniške tehnologije število delujočih procesov hitro povečuje.

Ukaz top

Ukaz **top** je naprednejši, saj omogoča sistemskemu administratorju prikaz slike procesov v realnem času. Prikaz procesov se namreč posodablja vsake 2–3 sekunde, prikazani pa so samo tisti procesi, ki uporabljajo največ CPU. Na ta način lahko sistemski administrator lažje ugotovi, kateri proces potencialno povzroča probleme na sistemu.

Primer izvajana ukaza **top** prikazuje Slika 5.5. Iz slike lahko razbe-

```
top - 09:29:26 up 3 days, 17:11,  1 user,  load average: 0,08, 0,23, 0,26
Tasks: 402 total,   1 running, 401 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0,7 us,  0,5 sy,  0,0 ni, 98,4 id,  0,4 wa,  0,0 hi,  0,0 si,  0,0 st
MiB Mem :  15572,3 total,   3378,1 free,   5188,3 used,   7005,8 buff/cache
MiB Swap:   2048,0 total,   2048,0 free,      0,0 used.   9914,8 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   1784 iztok     20   0 6028536 406752 146876 S  12,0   2,6  24:15.48 cinnamon
   1245 root      20   0   24,7g 224644 142368 S   5,3   1,4  25:27.88 Xorg
   3269 iztok     20   0 1131,3g 310264 142580 S   2,0   1,9  48:21.43 signal-+
 203075 iztok     20   0  696264  47640  37972 S   2,0   0,3   0:00.23 gnome-s+
  61444 iztok     20   0 1073164 101800  53956 S   1,7   0,6   0:43.18 nemo
    403 root     -51   0       0      0      0 S   1,0   0,0   6:06.44 irq/105+
 168478 iztok     20   0 3483056 703832 111928 S   0,7   4,4   9:26.59 Isolate+
 201108 iztok     20   0  797700  78944  52560 S   0,7   0,5   0:04.13 xed
   2078 iztok     20   0   12,5g   1,0g 286924 S   0,3   6,4 92:29.03 firefox+
   2131 iztok     20   0  545976  45656  35004 S   0,3   0,3   0:06.49 gnome-t+
 203038 iztok     20   0   15872   4244   3376 R   0,3   0,0   0:00.07 top
      1 root      20   0  166556  12000   8440 S   0,0   0,1   0:03.74 systemd
```

Slika 5.5: Rezultati ukaza **top**.

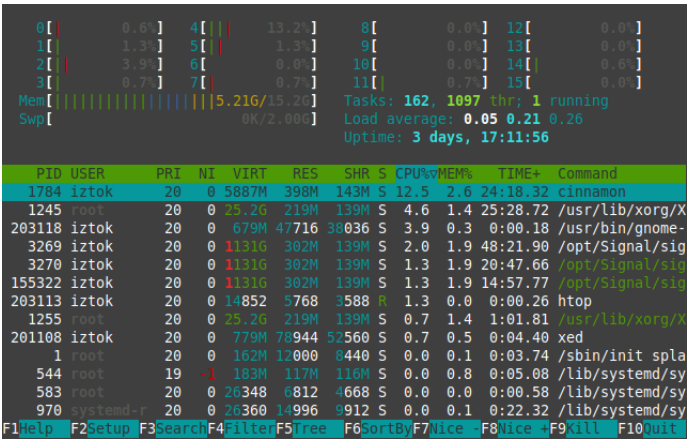
remo celotno statistiko delovanja opravilne vrste, skupaj z obremenitvijo sistema glede na število uporabnikov, porabe CPU in primarnega pomnilnika.

Ukaz htop

Ukaz **htop** izboljšuje možnosti ukaza **top** predvsem s stališča uporabnika. Sistemski administrator vidi namreč poleg trenutnega stanja

procesov na sistemu tudi zasedenost CPU po posameznih jedrih, porabo primarnega pomnilnika in izmenjevalnega prostora (angl. swap) v realnem času. Vsi ti prikazi so implementirani v psevdografični obliki.

Primer izvajanja ukaza **htop** prikazuje Slika 5.6. Dodatna možnost



Slika 5.6: Rezultati ukaza **htop**.

ukaza **htop** je konfiguriranje izpisa, ki omogoča uporabniku, da pride do tistih poglavitnih informacij o sistemu, ki ga zanimajo v določenem trenutku.

Ukaz strace

Ukaz **strace** omogoča sledenje sistemskih klicev in signalov in je zelo uporaben pri odkrivanju napak v fazi razvoja programske opreme. Sintaksa ukaza je naslednja:

```
$ strace [options] arguments
```

Osnovna uporaba ukaza **strace** je izpis vseh sistemskih klicev in signalov skozi opazovano obdobje, ki jih sprožimo z argumentom **ls**, kot na primer:

```
$ strace ls
execve("/usr/bin/ls", ["ls"], 0x7ffd77ac6fe0 /* 66 vars */)=0
brk(NULL) = 0x56548c52e000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffed1e9d3d0)=-1 EINVAL (Invalid argument)
access("/etc/ld.so.preload", R_OK)=-1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC)=3
newfstatat(3,, st_mode=S_IFREG|0644, st_size=95883, ..., AT_EMPTY_PATH)=0
mmap(NULL, 95883, PROT_READ, MAP_PRIVATE, 3, 0)=0x7f42ebf74000
close(3)=0
```

Izpis ukaza je zelo nerazumljiv, zato pogosto poganjamo sumarni izpis, kot na primer:

$ strace -c ls

% time	seconds	usecs/call	calls	errors	syscall
0,00	0,000000	0	5		read
0,00	0,000000	0	1		write
0,00	0,000000	0	9		close
0,00	0,000000	0	18		mmap
0,00	0,000000	0	2	2	access
...					
100,00	0,000000	0	78	5	total

Iz izpisa ukaza **strace -c** lahko razberemo, da je bilo v opazovanem obdobju 78 sistemskih klicev, 5 od teh se je končalo napačno.

6 | Diski in datotečni sistemi

Tradicionalni trdi diski še vedno ostajajo prevladujoči medij za trajno shranjevanje podatkov. V zadnjem času jim predstavljajo veliko konkurenco predvsem polprevodniški pogoni (angl. Solid State Drive, krajše SSD). Njihova prednost je hitrost dostopa, vendar hramba podatkov na tem mediju ni trajna. Operacijski sistem Linux danes omogoča več načinov dela z diskovnimi pogoni odvisno od zahtev uporabnikov, kot na primer: različnih implementacij diskovnih polj (angl. Redundant Arrays of Inexpensive Disks, krajše RAID), upraviteljev logičnih pogonov (angl. Logical Volume Manager, krajše LVM), sistemov za virtualizacijo diskov in nenazadnje različnih datotečnih sistemov (angl. filesystem).

Datotečni sistemi skrbijo za predstavitev in organizacijo sistemskih virov za shranjevanje. Poleg lastnih datotečnih sistemov (npr. **ext3** in **ext4**) podpira Linux dandanes tudi omrežne datotečne sisteme (npr. Networking File System, krajše NFS) in tuje datotečne sisteme, kot so tabela za alokacijo datotek (angl. File Allocation Table, krajše FAT) na MS-DOS oz. datotečni sistem nove tehnologije (angl. New

Technology File System, krajše NTFS).

V tem poglavju se najprej osredotočimo na delo z diski, nadaljujemo pa s proučevanjem lastnosti datotečnih sistemov, pri čemer izpostavljamo prednosti in slabosti pri uporabi različnih datotečnih sistemov.

6.1 Delo z diski

Osnovno vprašanje, s katerim se sistemski administrator pogosto srečuje v praksi, je: kaj vse je potrebno izvršiti pri dodajanju novega diska na sistemu Linux. V nadaljevanju poglavja se osredotočamo na ukaze, ki jih nudi operacijski sistem Linux za delo z diski in izpostavljamo njihove možnosti.

6.1.1 Dodajanje novega diska

Novi disk (npr. fizični disk SATA oz. SSD, USB disk SATA oz. SSD, virtualni disk na sistemu VirtualBox oz. VMware), ki ga želimo namestiti na operacijski sistem Linux, je potrebno: najprej identificirati, nato kreirati particijo in datotečni sistem ter na koncu še priklopiti. Korake, potrebne pri dodajanju novega diska, opisujemo v nadaljevanju poglavja podrobneje.

Identifikacija diskov

Identifikacija diska pomeni, da sistem Linux tega najprej razpozna, obenem pa določi njegove osnovne podatke, kot so: tip diska, njegova maksimalna velikost, številka model ipd. To akcijo na operacijskem sistemu Linux lahko poženemo z ukazom **fdisk**, kot na primer:

```
$ sudo fdisk -l
Disk /dev/sdb: 931,51 GiB, 1000204886016 bytes, 1953525168 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
```

Kot lahko razberemo iz izpisa diskov, imamo na operacijskem sistemu disk **/dev/sdb**, ki nima obstoječe particije in je neformatiran. Če želimo omenjeni disk uporabiti na sistemu Linux, je potrebno najprej ustvariti novo particijo.

Kreiranje particij na diskih Linux

Kreiranje particije poteka po naslednjem postopku:

```
$ sudo fdisk /dev/sdb
Command (m for help): n
```

S prvim ukazom poženemo **fdisk**, ki se oglasi v interaktivnem načinu z ukazno lupino. Z ukazom **n** izberemo formatiranje primarne particije oz. logičnega pogona. Primarna particija omogoča namestitev operacijskega sistema. Na sistemu imamo lahko največ štiri primarne particije, ki omogočajo t. i. dual boot. Logični pogoni so namenjeni shranjevanju podatkov, njihovo število pa ni omejeno.

Ko ustvarimo novo particijo, jo lahko pogledamo z naslednjim ukazom:

```
$ sudo fdisk -l
Disklabel type: gpt
Disk identifier: C0DD4701-93E2-4944-BB8D-DCACF32AF272
Device      Start        End    Sectors   Size Type
/dev/sdb1    2048 1953509375 1953507328 931,5G Linux filesystem
```

Iz izpisa lahko razberemo, da je nova particija tipa **gpt** (angl. **g**reat **p**artition **t**able), ki omogoča sistemu obravnavo velikih diskovnih pogonov.

Kreiranje datotečnega sistema

Preden lahko particijo na novem disku začnemo uporabljati, jo moramo ustrezno formatirati, tj. ustvariti ustrezni datotečni sistem. Danes na Linuxu najpogosteje uporabljamo datotečni sistem **ext4**, ki ga

ustvarimo z naslednjim ukazom:

```
$ sudo mkfs -t ext4 /dev/sdb1
```

Priklapljanje formatirane particije na Linux

Particijo priklopimo s pomočjo ukaza **mount** na naslednji način:

```
$ mount /dev/sdb1 /mnt
```

Ukaz priklopi datotečni sistem znotraj particije **/dev/sdb1** v direktorij **/mnt**.

6.1.2 Ukazi za delo z diski

Za delo s particijskimi tabelami na disku (tj. kreiranje, modificiranje in brisanje particij) nudi operacijski sistem Linux več orodij, kot na primer: **fdisk**, **cfdisk**, **sfdisk**, **parted** in **gparted**. Za vse diske velikosti $<$ 2TB uporabljamo Windows particijsko tabelo z zagonskim zapisom (angl. Master Boot Record, krajše MBR). Večji diski zahtevajo particijsko tabelo **gpt**, ki jo moramo kreirati z orodjem **parted** oz. **gparted**.

Sintaksa ukaza **fdisk** je naslednja:

```
$ sudo fdisk [options] [disk]
```

Ukaz **fdisk** je prioriteten, zato ga lahko izvajajo samo prioritetni uporabniki (npr. **root**) oz. vsi regularni uporabniki v administrativnem načinu delovanja prek ukaza **sudo**. Argumenti ukaza **options** so prikazani v Tabeli 6.1, iz katere lahko razberemo, da ukaz brez stikal požene ukazno lupino **fdisk**, v kateri operiramo s particijsko tabelo interaktivno. Ukazi ukazne lupine **fdisk** so prikazani v Tabeli 6.2. Orodje **cfdisk** je podobno ukazu **fdisk** in prav tako omogoča kreiranje, brisanje in modificiranje particij na diskih, vendar za komunikacijo administratorja z diskom uporablja grafični tekstovni vmesnik.

Tabela 6.1: Stikala ukaza **fdisk**.

Stikalo	Opis
-l	Izpiši particijsko tabelo.
-d	Backup/restore particijske tabele.
	Zaženi ukazno lupino **fdisk**.

Tabela 6.2: Ukazi ukazne lupine **fdisk**.

Ukaz	Opis
-n	Ustvari novo particijo.
-t	Spremeni tip particije.
-p	Izpiši particijsko tabelo.
-w	Zapiši spremembe na disk.

Za razliko od prej omenjenih je **sfdisk** popolnoma neinteraktivno orodje za manipulacijo particijskih tabel s pomočjo skriptnih datotek. Sintaksa ukaza je naslednja:

$ sudo sdisk [options] [disk] [<script]

Ukaz podpira podobna stikala kot **fdisk** in **cdisk**, vendar nadomešča interaktivni način uporabe prej omenjenih orodij s skriptnimi datotekami v predpisanem formatu, do katerih dostopa s preusmeritvijo sistemske datoteke STDIN.

Ukaz za preverjanje sistemskih datotek **fsck** (angl. **f**ile **s**ystem **c**heck) preverja datotečni sistem zaradi morebitnih napak. Imamo dva načina preverjanja: interaktivni in neinteraktivni. V interaktivnem načinu delovanja ukaz sprašuje administratorja ali popravi sektor na disku z napako ali ne. V neinteraktivnem načinu ukaz popravi vse sektorje na disku samodejno.

6.2 Datotečni sistemi Linux

Tipi datotečnih sistemov Linux

Na operacijskem sistemu Linux srečujemo več tipov datotečnih sistemov. Ti so odvisni od vrste operacijskega sistema oz. velikosti diska, na katerega datotečni sistem nameščamo. Najpogosteje srečujemo datotečne sisteme, prikazane v Tabeli 6.3. Linux podpira še

Tabela 6.3: Tipi datotečnih sistemov na Linuxu.

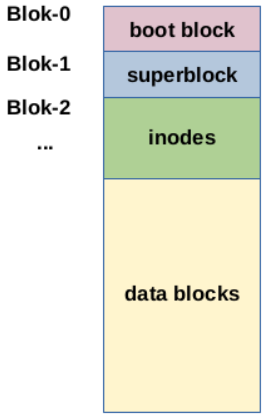
Tip	Značilnosti
ext2, **ext3**, **ext4**	Stabilnost in zanesljivost.
reiserfs	Manjše datoteke (vgrajeni sistemi).
xfs	Velike datoteke.
zfs	Upravitelj logičnih pogonov[11].

posebno vrsto datotečnega sistema, tj. izmenjevalni datotečni sistem (angl. swap), s katerim jedro operacijskega sistema navidezno razširja velikost primarnega pomnilnika. Vsak proces na sistemu Linux se izvaja v svojem navideznem naslovnem prostoru, ki ga mora jedro operacijskega sistema transformirati v realni naslovni prostor pred izvajanjem. Ta navidezni naslovni prostor hrani jedro v izmenjevalnem datotečnem sistemu **swap**.

Poleg datotečnih sistemov na fizičnih diskih, priključenih na operacijski sistem lokalno podpira Linux tudi omrežne datotečne sisteme. Do teh dostopamo prek omrežja. Najpogosteje uporabljamo:

- omrežni datotečni sistem (angl. Network File System, krajše NFS),
- splošni internetni datotečni sistem (angl. Common Internet File System, krajše CIFS).

[11] Upravitelj logičnih pogonov (angl. Logical Volume Manager, LVM) omogoča povezovanje več fizičnih diskov v en logični disk.



Slika 6.1: Fizična organizacija datotečnih sistemov Linux.

NFS omogoča porazdeljevanje omrežnih virov (npr. datotek in tiskalnikov) med sistemi Linux in Unix. Tukaj poznamo več verzij tega protokola (npr. NFSv3, NFSv4), ki se med seboj razlikujejo predvsem po nivoju varnosti, ki jo nudijo. Protokol CIFS omogoča, da se strežniki Windows predstavljajo sistemom Linux kot njihovi enakovredni partnerji. Za to funkcionalnost moramo na operacijskem sistemu Linux zagnati dva demona: **smbd** in **nmbd**. Ta dovoljujeta izmenjavo datotek in uporabo tiskalnikov Windows tako, kot če bi bili priključeni lokalno na strežnik Linux.

Fizična organizacija datotečnih sistemov Linux

Sodobna jedra operacijskih sistemov tvorijo abstraktni vmesnik do različnih datotečnih sistemov, kjer so shranjene datoteke. Datotečni sistemi Linux so organizirani tako, kot je to prikazano na Sliki 6.1. Kot lahko razberemo iz omenjene slike, je datotečni sistem na fizičnem nivoju razdeljen na bloke istih velikosti (npr. 4 Kbajtov). Prvi blok (tj. Blok-0 na sliki) je t. i. zagonski blok (angl. boot block), v katerem se nahaja zaganjalnik jedra operacijskega sistema (angl.

boot loader). Naslednji blok je t. i. glavni blok (angl. superblock), ki je metastruktura z različnimi informacijami, kot so na primer: tip datotečnega sistema, število blokov, število indeksnih blokov (angl. inodes), možnosti diska in podobno. Število indeksnih blokov je unikatno za določen tip particije in se določi ob kreiranju datotečnega sistema. Z njimi opisujemo objekte datotečnega sistema, kot so datoteke in direktoriji. Vsak inode je seznam atributov in lokacij podatkovnih blokov, ki sestavljajo podatke določene datoteke. Direktorij je seznam indeksnih blokov, ki jim priredimo imena datotek.

Logična organizacija datotečnih sistemov Linux

Datotečni sistem Linux je na logičnem nivoju organiziran kot drevo. Koren drevesa predstavlja glavni direktorij, ki ga označimo s simbolom / in vsebuje minimalno število direktorjev in sistemskih datotek, potrebnih za nalaganje in delovanje operacijskega sistema. Vsak direktorij lahko vsebuje datoteke in direktorije. Z imeni direktorijev označujemo, kateri tip datotek shranjujemo v njih. Konvencija poimenovanja direktorijev ni povsem stadardizirana in lahko na različnih distribucijah operacijskega sistema Linux oz. Unix lahko pomenijo drugo. Kljub vsemu pa so nekatera poimenovanja direktorijev skupna vsem.

Najpomembnejši sistemski direktorji so naslednji:

- /**etc**,
- /**bin**, /**sbin**,
- /**var**,
- /**usr**,
- /**tmp**,
- /**opt**,
- /**home**.

Direktorij /**etc** vsebuje datoteke, s katerimi konfiguriramo nastavitve sistemskih programov. Direktorij /**bin** oz. /**sbin** vsebuje ukaze pomožnih sistemskih programov, s katerimi razširjamo moč operacijskega sistema (npr. strežnik Apache). V direktoriju /**var** shranjujemo specifične sistemske podatke. Zaradi normalnega delovanja operacijskega sistema je priporočljivo, da je ta direktorij priključen kot ločen datotečni sistem, saj s tem ne ovira delovanja sistema v primeru odpovedi. Direktorij /**usr** je namenjen shranjevanju standardnih uporabniških programov, ki se ne poganjajo ob zagonu sistema. V direktorij /**tmp** običajno shranjujemo začasne datoteke, ki jih potrebujejo pomožni sistemski programi. Direktorij /**opt** vzdržuje opcijske programske produkte, katerih uporaba ne upošteva sistemskih standardov (tj. uporabljajo se nekonvencionalno). V direktoriju /**home** shranjujemo domače direktorije za posameznega uporabnika. Vsak direktorij uporabnika je določen z njegovim imenom (npr. domači direktorij uporabnika **user** je /**home**/**user**).

Datoteke na sistemu Linux

Vsaka datoteka na operacijskem sistemu Linux je določena z imenom in atributi. Izpis imena datoteke skupaj z atributi (tj. dolgi izpis) prikažemo z naslednjim ukazom:

```
$ ls -l a.txt
-rw-rw-r--  100  user group  930  Jul 7 11:19  a.txt
type and perm.  link count  owner and group  size  date and time  name
```

Dolgi izpis datoteke lahko združimo v šest izpisnih polj, ki predstavljajo:

- tip in dostopne pravice,
- število odprtih povezav,
- lastnika datoteke,
- velikost datoteke,

- datum in čas zadnje spremembe,
- ime datoteke.

Prvi od desetih znakov v izpisu datoteke predstavlja tip datoteke (Tabela 6.4). Normalna datoteka sestoji iz zaporedja bajtov, ki jih lahko

Tabela 6.4: Tipi datotek v Linuxu.

Simbol	Opis
-	Normalna datoteka (angl. regular file)
d	Direktorij
c	Gonilnik znakovnih naprav
b	Gonilnik blokovnih naprav
s	Lokalna vtičnica (angl. Local socket)
p	Imenovani cevovod (angl. named pipe)
l	Simbolična povezava (angl. symbolic link)

interpretiramo kot tekst oz. binarne nize. Direktoriji vsebujejo normalne datoteke in direktorije. Kreiramo jih z ukazom **mkdir**, brišemo pa z ukazom **rmdir**. Gonilniki znakovnih oz. blokovnih naprav predstavljajo vmesnik med napravo in jedrom operacijskega sistema. Razlika med znakovnimi in blokovnimi napravami je enota prenosa, saj znakovne naprave prenašajo zaporedja znakov (npr. tipkovnica, modem ipd.), medtem ko blokovne naprave operirajo z bloki (npr. diski, tračne enote ipd.). Gonilniki so shranjeni v sistemskem direktoriju **/proc** v obliki **/dev/naprava**. Gonilnik **/dev/tty0** na primer, kominicira s serijskimi vrati 0, na katerih je običajno nameščen prikazovalnik. Lokalne vtičnice omogočajo komunikacijo med procesi na lokalnem gostitelju. Tudi imenovani cevovodi omogočajo kominikacijo med dvema procesoma, ki tečeta na istem gostitelju. Tako lokalne vtičnice kakor tudi imenovani cevovodi so zastarele tehnologije, ki jih danes zamenjujo omrežni cevovodi. Simbolične povezave (tudi mehke povezave) vsebujejo sklice na datoteke z imeni. Simbolično povezavo kreiramo z ukazom **ln -s** in brišemo z ukazom **rm**.

Naslednjih devet znakov v izpisu datoteke predstavljajo dostopne pravice. Te pravice so zbrane v kombinaciji trikrat po tri znake, kjer vsak od treh znakov pomeni pravico do branja (znak 'r'), pisanja (znak 'w'), oz. izvajanja (znak 'x'), za lastnika, skupino in ostale uporabnike. Če pravica do datoteke ni dovoljena, je na ustreznem mestu zapisan znak '-'. Datoteka, označena z zaporedjem znakov '-rw-rw-r–', označuje normalno datoteko (tip datoteke), do katere ima lastnik oz. skupina uporabnikov pravico branja in pisanja, vsi ostali pa lahko datoteko samo berejo.

Število odprtih povezav (angl. link count) določa število uporabnikov, ki trenutno dostopa do datoteke. V našem primeru je to število 100, kar pomeni, da do datoteke trenutno dostopa 100 uporabnikov. Lastnik datoteke je označen z imenom uporabnika *user* in skupino *group*, ki ji pripada. Velikost datoteke je izmerjena v bajtih. Polje datum in čas (angl. *date and time*) označujeta čas zadnje spremembe datoteke. Zadnje polje v izpisu datoteke je njeno ime (npr. **a.txt**).

Pri kreiranju datotek oz. direktorijev dobi objekt, ki ga kreiramo, privzete pravice. Te pravice dodelimo na sistemu Linux z ukazom **umask**. Dostopne pravice iz privzetega dostopa do datotek izračunamo tako, da od vrednosti 666 odštejemo vrednost **umask**, na primer:

666-002=664

Vrednost 664 interpretiramo oktalno, kjer prvo oktalno število določa pravice za lastnika ($6_8 \equiv 110_2 \Rightarrow$ 'rw-'), drugo za skupino ($6_8 \equiv 110_2 \Rightarrow$ 'rw-') in tretje za ostale ($4_8 \equiv 100_2 \Rightarrow$ 'r--'). Skupaj oktalno število 664 transformiramo v naslednje zaporedje znakov:

$664_8 \equiv 110110100_2 \Rightarrow$ '-rw-rw-r--'

Začetni simbol '-' v posledici logičnega izraza označuje normalno datoteko.

Dostopne pravice iz privzetega dostopa do direktorijev izračunamo tako, da od vrednosti 777 odštejemo vrednost **umask**, kot na pri-

mer:

777-002=775

Vrednost 775 interpretiramo oktalno, kjer prvo oktalno število določa pravice za lastnika ($7_8 \equiv 111_2 \Rightarrow$ 'rwx'), drugo za skupino ($7_8 = 111_2 \Rightarrow$ 'rwx') in tretje za ostale ($5_8 \equiv 101_2 \Rightarrow$ 'r-x'). Skupaj oktalno število 775 transformiramo v naslednje zaporedje znakov:

$775_8 \equiv 111111101_2 \Rightarrow$ 'drwxrwxr-x'

Začetni simbol 'd' v posledici logičnega izraza označuje direktorij.

Ukaz **umask** določamo običajno v konfiguracijskih datotekah **.bashrc** oz. **.profile**, ki se poganjata ob zagonu ukazne lupine **bash**.

6.2.1 Ukazi za delo z datotečnimi sistemi

Ukazi za kreiranje datotečnih sistemov

Naprednejši način kreiranja datotečnih sistemov je ukaz **mkfs**, ki mu sledi prefiks **.** z imenom datotečnega sistema, ki ga želimo kreirati. Če želimo videti, katere možnosti datotečnih sistemov omogoča operacijski sistem Linux, vtipkamo ukaz **mkfs** in ga zaključimo z dvema zakoma <tab>, ali drugače:

```
$ mkfs<tab><tab>
mkfs          mkfs.btrfs    mkfs.exfat    mkfs.ext3     mkfs.fat      mkfs.hfsplus
mkfs.minix    mkfs.ntfs     mkfs.vfat     mkfs.bfs      mkfs.cramfs   mkfs.ext2
mkfs.ext4     mkfs.hfs      mkfs.jfs      mkfs.msdos    mkfs.reiserfs mkfs.xfs
```

Kot lahko vidimo iz odgovora ukaza, podpira Linux poleg lastnih datotečnih sistemov (**ext2**, **ext3** in **ext4**) tudi druge, kot na primer **ntfs**, **msdos** in **minix**.

Sintaksa ukaza **mkfs** je naslednja:

$ sudo mkfs[.datotečni_sistem] particija

V sintaksi ukaza pomeni **.datotečni_sistem** izbrani datotečni sistem. Če tega ne izberemo, je privzeta vrednost ukaza **.ext2**.

Izmenjalni datotečni sistem kreiramo z naslednjim ukazom:

```
$ sudo mkswap swapfile
```

Argument ukaza **swapfile** je pot do ustrezne datoteke, ki služi kot izmenjalni datotečni sistem. To datoteko lahko kreiramo tudi kot RAM disk, s čimer pospešimo čas izvajanja.

Ukazi za priklapljanje datotečnih sistemov

Formatirane datotečne sisteme lahko priklapljamo na operacijski sistem Linux na dva načina: (1) z uporabo datoteke **fstab** ali (2) z uporabo ukaza **mount**. Prvi način priklaplja particijo ob zagonu sistema. Disk priklapljamo običajno z njegovo identifikacijo, ki jo vidimo s pomočjo ukaza **blkid**, kot na primer:

```
$ blkid
dev/sda1: UUID="C0DD4701-93E2-4944-BB8D-DCACF32AF272"
```

Zapis avtomatskega priklopa diska v datoteki **/etc/fstab** pa je naslednji:

#<file-system>	<m.p.>	<type>	<options>	<dump>	<pass>
UID=C0DD4701-...	/	ext4	errors=remount-ro	0	1

Format datoteke **/etc/fstab** določa: (1) datotečni sitem, ki ga priklapljamo, (2) točko priklopa (angl. mount point), (3) tip datotečnega sistema (angl. type), (4) različne možnosti ob priklapljanju, (5) dump on=1, off=0, (6) fsck on=1, off=0.

Vse priklopljene datotečne sisteme na operacijskem sistemu Linux lahko vidimo z naslednjim ukazom:

```
$ mount
/dev/sda1 on / type ext4 (rw,relatime,errors=remount-ro)
/dev/sdb1 on /home type ext4 (rw,relatime,errors=remount-ro)
...
```

V našem primeru operiramo z dvema particijama **/dev/sda1** in **/dev/sdb1** kreiranima na dveh različnih trdih diskih **/dev/sda** in **/dev/sdb**, kjer se v prvi particiji nahaja korenski direktorij /, v drugi pa domači direktorij uporabnikov **/home**.

Sintaksa ukaza **mount** v splošnem je naslednja:

```
$ mount [options] device directory
```

Najpomembnejši stikali **options** ukaza sta **-t** in **-o**. S prvim stikalom se sklicujemo na tip datotečnega sistema (npr. **-t ext4**), z drugim na različne možnosti priklapljanja. Argument **device** se nanaša na napravo, ki jo želimo priklopiti (npr. **/dev/sda1**), argument **directory** pa določa točko priklopa na datotečni sistem (npr. **mnt**).

Datotečni sistem lahko po uporabi izklopimo z ukazom **umount**. Preden pa lahko izvedemo izklop, moramo preveriti, koliko procesov trenutno uporablja datotečni sistem. To preverjanje omogoča ukaz **fuser**, ki ga poženemo, kot na primer:

```
$ fuser -m /home
/home:
```

Če je število procesov nič, lahko datotečni sistem normalno izklopimo z naslednjim ukazom:

```
$ umount -l /home
```

Ukaz **umount** z opcijo **-l** (oz. –**lazy**) počaka, dokler datotečni sistem ni več v uporabi. Če želimo datotečni sistem izklopiti v trenutku, nam to omogoča ukaz:

```
$ umount -f /home
```

Opcija **-f** (oz. –**force**) povzroči izklop datotečnega sistema na silo.

Pri zagonu sistema izmenjalno datoteko priključimo avtomatično s pomočjo datoteke **fstab**, kot na primer:

```
$ cat /etc/fstab
/swapfile   none   swap   sw   0   0
```

Pri zagonu operacijskega sistema Linux jedro uporabi ime datoteke **swapfile** v korenskem direktoriju / kot izmenjalni datotečni sistem, ki nima prikjučne točke na korenski datotečni sistem (tj. mount point **none**). Tip datoteke bo **swap** z opcijo **sw**, ki pove, da bo ta datoteka predmet ukazov **swapon**/**swapoff**, s katerim bomo nadzorovali aktivnost izmenjevalnega procesa. Preostali dve možnosti sta za izmenjevalni datotečni sistem nepomembni in sta zato postavljeni na 0.

Zelo enostavno lahko priključimo omrežni datotečni sistem na sistemu Linux s pomočjo protokola **ssh** in ukaza **sshfs**. Ukaz **sshfs** je orodje za varno priklapljanje direktorijev na oddaljenih strežnikih na lokalni računalnik. Prenos podatkov poteka prek varne povezave **ssh**. Sintaksa ukaza je naslednja:

```
$ sshfs -f [-o <options>] user@host:/path_to_directory /mnt
```

Oddaljeni strežnik določimo kot niz **user@host:/path_to_directory**, kjer se **user** nanaša na uporabniško ime na oddaljenem strežniku, **host** določa njegovo ime DNS oz. naslov IP, /**path_to_directory** pa določa absolutno pot do direktorija, do katerega dostopamo lokalno prek priključne točke /**mnt**.

Ukazi za nadzor datotečnih sistemov

Za preverjanje velikosti datotečnega sistema uporabljamo ukaz **df**. Primer uporabe ukaza je naslednji:

```
$ df -h /home
Filesystem   Size   Used   Avail   Use%   Mounted on
/dev/sda4    869G   345G   480G    42%    /home
```

Ukaz smo uporabili s stikalom **-h**, ki izpiše velikost datotečnega sistema v človeku prijazni obliki (angl. human readable form), kar pomeni, da namesto števila zasedenih blokov ukaz izpiše velikost v gigabajtih.

Ukaz **du** (angl. disk usage) omogoča uporabnikom pogledati informacije o uporabi diska na hiter način. Nanaša se na določen direktorij in podpira več argumentov. Primer uporabe ukaza je prikazan kot:

```
$ df -sh /home
345G /home
```

Ukaz prikaže skupno velikost direktorija /**home** (stikalo **-s**) v uporabniku prijazni obliki (stikalo **-h**).

Učinkovitost diskov lahko sistemski administrator pogleda s pomočjo ukaza **iostat**, kot na primer:

```
$ iostat
...
Device            tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd
sda              0,24         2,00         5,64         0,00     776915    2188605          0
sdb              0,32         5,95         3,68       570,95    2310600    1428780  221755648
sr0              0,00         0,00         0,00         0,00          2          0          0
```

7 | Uporabniki

Čeprav se dandanes vse bolj uporabljajo integrirane storitve za delo z uporabniškimi računi, kot na primer LDAP in Active Directory, operacijski sistem Linux omogoča osnovne ukaze za delo z uporabniškimi računi in njihovimi pravicami. Ti ukazi predstavljajo temelj sistemske varnosti.

V nadaljevanju poglavja obravnavamo ukaze za dodajanje in odstranjevanje uporabniških računov in pregledamo datoteke, v katerih hranimo informacije o sistemski varnosti.

7.1 Definicija uporabniških računov

Vsak uporabnik na sistemu Linux ne predstavlja nič drugega kakor 32-bitno število, znano pod imenom User ID (krajše UID). Vse definicije uporabnikov, ki jih pozna sistem Linux, so shranjene v datoteki uporabniških računov /**etc/passwd**. Format te datoteke je naslednji:

```
$ cat /etc/passwd |grep root
root:x:0:0:root:/root:/bin/bash
```

username *password* *UID* *GID* *Comment* *Home* *Shell*

Definicija uporabniškega računa sestoji iz uporabniškega imena (angl. username), gesla (angl. password), identifikacije (angl. User ID, krajše UID),

identifikacije skupine uporabnikov (angl. Group ID, krajše GID), komentarja (angl. Comment), domačega direktorija (angl. home directory) in specifikacije ukazne lupine (angl. command shell), ki se požene ob prijavi uporabnika na sistem.

Uporabniško ime je dolgo največ 32 znakov in mora biti unikatno na sistemu Linux. Geslo je označeno s simbolom **x** in je kriptirano v ločeni datoteki **/etc/shadow**. Vsebino te datoteke lahko vidi samo uporabnik **root**. UID je 32-bitno število (največje število $2^{32} - 1$). Število UID=0 je razervirano za uporabnika **root**. Števila UID od 0–999 so rezervirana za sistemske račune, vsa ostala števila od 1000 naprej pa za regularne uporabniške račune. GID je prav tako celoštevilčna 32-bitna vrednost. Podobno kot pri UID je tudi GID=0 rezerviran za skupino **root**. Uporabniki so lahko člani več skupin, vendar morajo imeti eno privzeto skupino. Polje komentar omogoča vnos poljubnega komentarja, ki ga vnese sistemski administrator pri definiciji uporabniškega računa. Domači direktorij označuje področje na disku namenjeno shranjevanju matičnih podatkov uporabnika. Ukazna lupina določa ime ukaznega interpreterja, ki se požene pri prijavi uporabnika na sistem. Privzeti ukazni interpter je **bash**.

Skupine uporabnikov hranimo v datoteki **/etc/group**, ki ima naslednji format:

```
$ cat /etc/group |grep root
root : x : 0 :
groupname password GID grouplist
```

Skupina sestoji iz imena skupine (angl. groupname), gesla (angl. password), identifikacije skupine (angl. Group ID, krajše GID) in seznama skupin, ki ji skupina pripada, ločenih s simbolom **,**. Običajno gesla za skupine ne uporabljamo. Seznam skupin ostaja pri definiciji uporabniškega računa prazen.

7.2 Ukazi za delo z uporabniškimi računi

Dodajanje uporabniških računov na operacijskem sistemu Linux omogoča ukaz **useradd**, ki ga običajno poganjamo z naslednjimi parametri:

```
$ sudo useradd -c "Privzeti uporabnik" -d /home/user -m -s /bin/bash user
```

Stikalo **-c** doda komentar **"Privzeti uporabnik"** v datoteko **/etc/passwd**. Kot domači direktorij izbere področje **/home/user** (stikalo **-d**). S stikalom **-m** ustvarimo domači direktorij, če ta še ne obstaja, stikalo **-s** pa kopira vse datoteke iz direktorija **/etc/skel** v domači direktorij.

Spremembe, ki jih je povzročilo delovanje ukaza **useradd** v datoteki **/etc/passwd**, so naslednje:

```
$ cat /etc/passwd |grep user
user:x:1001:1001:Privzeti uporabnik:/home/user:/bin/bash
```

V datoteki **/etc/passwd** se pojavi novi vpis definicije uporabnika **user**, njegovo UID=1001, GID=1001, komentarjem, ustvarjenim domačim direktorijem in specifikacijo ukazne lupine **bash**.

V datoteki **/etc/group** najdemo definicijo za skupino **user** z GID=1001. Dodatnih skupin pri skupini **user** še ni definiranih.

```
$ cat /etc/group |grep user
user:x:1001:
```

Delovanje stikala **-s** opazimo, če zlistamo vsebino na novo ustvarjenega domačega direktorija **/home/user**.

```
$ sudo ls -la /home/user
total 32
drwxr-x---  3 user user 4096 Jul 11 12:16 .
drwxr-xr-x  5 user user 4096 Jul 11 12:16 ..
-rw-r--r--  1 user user  220 Jan 6 2022   .bash_logout
-rw-r--r--  1 user user 3771 Jan 6 2022   .bashrc
drwxr-xr-x  5 user user 4096 Dec 17 2022  .config
-rw-r--r--  1 user user   22 Sep 8 2011   .gtkrc-2.0
-rw-r--r--  1 user user  516 Dec 17 2013  .gtkrc-xfce
-rw-r--r--  1 user user  807 Jan 6 2022   .profile
```

Opazimo lahko, da smo pri kreiranju novega uporabniškega imena kreirali tudi privzete zagonske datoteke **.bashrc** in **.profile**. Vsebino omenjenih datotek lahko naknadno priredimo svojim potrebam (npr. nastavitev **umask**).

Geslo za nov uporabniški račun **user** definiramo s pomočjo ukaza **passwd** na naslednji način:

```
$ sudo passwd user
[sudo] password for root: <root password>
New password: <user password>
Retype new password: <user password ponovno>
passwd: password updated successfully
```

Z ukazom **usermod** lahko spreminjamo polja v datoteki **/etc/passwd**, kot na primer:

```
$ sudo usermod -c "Obstoječi uporabnik"user
```

Ukaz spremeni komentar v uporabniškem računu uporabnika **user** v datoteki **/etc/passwd** kot:

```
$ cat /etc/passwd |grep user
user:x:1001:1001:Obstoječi uporabnik:/home/user:/bin/bash
```

Uporabniški račun na sistemu Linux zbrišemo z ukazom **userdel**, kot na primer:

```
$ sudo userdel user
```

Ukaz zbriše uporabniški račun iz datoteke **/etc/passwd** in kriptirano geslo iz **/etc/shadow**. Če želimo zbrisati tudi domači direktorij, pokličemo ukaz s stikalom **-r**, kot na primer:

```
$ sudo userdel -r user
```

7.3 Ukazi za delo s skupinami uporabnikov

Ukazi te družine omogočajo dodajanje, spreminjanje in brisanje skupin uporabnikov. Pojem skupina uporabnikov uporabljamo za organizacijo in administracijo uporabniških računov v Linuxu. Osnovni namen skupine je definirati množico privilegijev, kot so dostopne pravice branja, pisanja in izvajanja danega vira, ki so porazdeljeni med uporabnike neke skupine. Skupino uporabnikov dodajamo na dva načina: (1) z imenom skupine ali (2) s specifikacijo GID. Sintaksa ukaza za dodajanje skupine prek imena je enostavna:

```
$ sudo groupadd moja_skupina
```

Ukaz vpiše novo skupino **moja_skupina** v datoteko **/etc/group**. Vpis lahko pogledamo z ukazom **cat**, kot na primer:

```
$ cat /etc/group |grep moja
moja_skupina:x:1001:
```

Sintaksa dodajanja skupine uporabnikov prek identifikacije skupine GID je naslednja:

```
$ sudo groupadd -g 1010 moja_skupina
$ cat /etc/group |grep moja
moja_skupina:x:1010:
```

Ukaz doda novo skupino **moja_skupina** in ji priredi identifikacijo skupine 1010. Za dodajanje uporabniškega računa v skupino uporabnikov uporabimo naslednji ukaz:

```
$ sudo useradd -g moja_skupina user
```

Spremembo v datoteki **/etc/group** prikazuje naslednji ukaz:

```
$ cat /etc/group |grep moja
moja_skupina:x:1010:user
```

Skupino uporabnikov **moja_skupina** brišemo z ukazom:

```
$ sudo groupdel moja_skupina
```

Da smo skupino tudi v resnici zbrisali, preverimo za naslednjim ukazom:

```
$ getent group | grep moja_skupina
groupdel: group 'moja_skupina' does not exist
```

V primeru da skupine **moja_skupina** ni v datoteki **/etc/group**, dobimo ustrezen komentar.

8 | Zaključek

V tem učbeniku sta avtorja poskušala na enostaven in priročen način približati možnosti uporabe operacijskega sistema Linux bralcem, še posebej tistim začetnikom, ki bi želeli presedlati na Linux z drugih operacijskih sistemov ali pa se z njim vsaj pobliže spoznati. Učbenik je na kratko predstavil principe delovanja in praktične uporabe tega zmogljivega operacijskega sistema, ki je od skromnih začetkov kot študentski projekt pripeljal do tega, da postaja hrbtenica zahtevnih strežniških sistemov, vgrajenih sistemov in poglavitno orodje za razvijalce programske opreme. Na ta način ta operacijski sistem še naprej oblikuje prihodnost tehnologije in posledično spreminja svet.

Tukaj smo predstavili značilnosti operacijskega sistema Linux vključno z njegovo zgodovino, potegnili pa smo tudi vzporednice Linuxa z Unixom. Razumevanje te zgodovine bralcem omogoča vpogled v filozofijo delovanja, na kateri temelji Linux kot odprtokodni projekt, ki ga soustvarja množica posameznikov po celem svetu. Bralcem smo predstavili najpopularnejše distribucije tega operacijskega sistema, izpostavili njihove prednosti in slabosti. S primerjavo distribucij, kot so Ubuntu, Fedora, Manjaro in Arch Linux, smo želeli, da bi bralci bolje razumeli, katero distribucijo izbrati, da kar najbolje ustreza različnim primerom uporabe v praksi.

V učbeniku smo podrobno prikazali možnosti ukazne lupine **bash** ter skriptno programiranje s pomočjo **bash**, **sed**, **awk** in **Python**. Prikazali smo praktične primere za avtomatizacijo opravil, manipulacijo z besedilom in

obdelavo podatkov. Ta znanja so bistvena za vsakega sistemskega administratorja, saj ta omogočajo racionalizacijo delovanja, večjo učinkovitost in nemoteno delovanje naših sistemov Linux.

Poleg pisanja skript smo raziskali ključne vidike dela z uporabniškimi računi, diski in datotečnimi sistemi. Razumevanje upravljanja uporabniških računov in omejevanja dostopnih pravic je ključnega pomena za vzdrževanje varnega in organiziranega sistema. Upravljanje diskov in datotečnih sistemov omogoča učinkovito dodeljevanje in uporabo pomnilniških virov.

Ko zaključujemo naše popotovanje po svetu upravljanja operacijskih sistemov Linux, se moramo zavedati, da se to področje nenehno razvija. Operacijski sistem Linux se še naprej prilagaja zahtevam nove tehnologije in z vsako novo izdajo ponuja nove funkcije, izboljšane varnostne ukrepe in večjo zmogljivost. Zato je bistveno, da sistemski administratorji ostanejo radovedni in da so sposobni vseživljenjskega učenja.

Čeprav ta knjiga zagotovlja močne temelje za upravljanje operacijskih sistemov Linux, je to za potencialnega sistemskega administratorja šele začetek poti. Svet Linuxa je obsežen in se nenehno širi, saj ponuja neskončne možnosti in izzive, ki jih je potrebno raziskovati vedno znova. Z uporabo znanja in veščin, pridobljenih v tej knjigi, ste dobro pripravljeni, da se spopadete z nešteto scenariji ter prispevate k robustnemu delovanju in zanesljivosti sistemov Linux.

Naprednejnejša sistemska administracija, delo z omrežji, upravljanje spletnih in poštnih strežnikov ostajajo predmet naslednjega učbenika.

Literatura

[1] A. S. Tanenbaum in A. S. Woodhull, *Operating systems: design and implementation*. Prentice Hall Englewood Cliffs, 1997, zv. 68.

[2] A. Košir, R. Maurer, R. Papež, P. Peterlin in M. Tomšič, *Linux z namizjem KDE: priročnik za delo z operacijskim sistemom Linux*. Pasadena, 2003.

[3] R. M. Stallman, "Free Software, Free Society," 2002.

[4] B. W. Kernighan in D. M. Ritchie, *The C Programming Language*, 1th. Englewood Cliffs, NJ: Prentice Hall, 1978, ISBN: 0-13-110163-3.

[5] G. Moody, *Rebel code : the inside story of Linux and the open source revolution*. Cambridge, Mass. : Perseus Pub., 2001.

[6] E. Nemeth, G. Snyder, T. R. Hein, B. Whaley in D. Mackin, *UNIX and Linux System Administration Handbook (5th Edition)*, 5th. Addison-Wesley Professional, 2017, ISBN: 0134277554.

[7] D. Legay, A. Decan in T. Mens, "On package freshness in linux distributions," v *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, 2020, str. 682–686.

[8] C. Ramey in B. Fox, *Bash Reference Manual: Reference Documentation for Bash*, 5.2. Free Software Foundation, Inc., 2022.

[9] D. Dougherty, *Sed and AWK*. USA: O'Reilly & Associates, Inc., 1991, ISBN: 0937175595.

[10] N. Gift in J. Jones, *Python for Unix and Linux System Administration*. O'Reilly Media, Inc., 2008, ISBN: 0596515820.

[11] B. W. Kernighan in D. M. Ritchie, *The C Programming Language*, 2nd. Prentice Hall Professional Technical Reference, 1988, ISBN: 0131103709.

[12] J. Goyvaerts in S. Levithan, *Regular Expressions Cookbook*. Sebastopol, CA: O'Reilly Media, Inc., 2009, ISBN: 0596515820.

[13] A. V. Aho, B. W. Kernighan in P. J. Weinberger, *The AWK Programming Language*. USA: Addison-Wesley Longman Publishing Co., Inc., 1987, ISBN: 020107981X.

[14] D. Dougherty in A. Robbins, *Sed and AWK*, 2st. Sebastopol, CA: O'Reilly Media, Inc., 1997, ISBN: 978-1-565-92225-9.

[15] M. Lutz, *Learning Python*, 2. izd. USA: O'Reilly & Associates, Inc., 2003, ISBN: 0596002815.

[16] N. Gift in J. Jones, *Python for Unix and Linux System Administration*. O'Reilly Media, Inc., 2008, ISBN: 0596515820.

Seznam slik

Seznam tabel

SISTEMSKA ADMINISTRACIJA V LINUXU

IZTOK FISTER, ML., IZTOK FISTER

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, Slovenija
iztok.fister1@um.si, iztok.fister@um.si

Pričujoča knjiga je namenjena tako začetnikom, ki želijo pridobiti osnovno znanje in veščine, potrebne za upravljanje operacijskega sistema Linux, kakor tudi izkušenim sistemskim administratorjem, ki želi osvežiti svoje znanje. Knjiga je razdeljena na tematsko organizirana poglavja, ki se začnejo z uvodom v sistem Linux in se postopoma poglobijo v njegove osnovne komponente. Bralci se bodo naučili pisanja skript Bash, izkoriščanja moči skriptnih programov sed in awk, učinkovitega upravljanja uporabnikov, optimizacije diskov in datotečnih sistemov ter konfiguriranja omrežnih virov. V knjigi so številni primeri iz resničnega sveta, ki zagotavljajo praktična spoznanja in utrjujejo obravnavane koncepte. Ne glede na to, ali gre za začetnika ali izkušenega administratorja, ta knjiga ponuja jasno in pragmatično znanje za obvladovanje upravljanja sistema Linux.

DOI
https://doi.org/10.18690/um.feri.11.2023

ISBN
978-961-286-796-6

Ključne besede:
operacijski sistem Linux,
sistemska administracija,
skriptno programiranje,
sed,
awk

Univerzitetna založba Univerze v Mariboru

DOI
https://doi.org/
10.18690/um.feri.11.2023

ISBN
978-961-286-796-6

Keywords:
finite element method,
plane line structures,
computer analysis of
structures,
static response analysis,
modeling,
graphical representations
of results

SYSTEM ADMINISTRATION IN LINUX

IZTOK FISTER, ML., IZTOK FISTER

University of Maribor, Faculty of Electrical Engineering and Computer Science,
Maribor, Slovenia
iztok.fister1@um.si, iztok.fister@um.si

This book is intended for beginners who want to acquire the basic knowledge and skills needed to manage the Linux operating system, as well as for experienced system administrators who want to refresh their knowledge. The book is divided into thematically organised chapters that start with an introduction to Linux and gradually delve deeper into its basic components. Readers will learn how to write Bash scripts, harness the power of sed and awk scripts, manage users efficiently, optimise disks and file systems, and configure network resources. The book includes numerous real-world examples that provide practical insights and reinforce the concepts covered. Whether a beginner or an experienced administrator, this book provides clear and pragmatic knowledge for mastering Linux administration.

University of Maribor Press

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko

Knjiga o administraciji operacijskega sistema Linux je izčrpen in koristen vir znanja, primeren za vse, od začetnikov do izkušenih strokovnjakov. Avtorja sta uspešno poenostavila kompleksne koncepte Linuxa, kar olajša razumevanje in obvladovanje tega operacijskega sistema. Knjiga nudi trdne temelje za upravljanje Linux sistemov, z osredotočenostjo na ukazno lupino Bash, skriptiranje, upravljanje procesov, diske in uporabniške račune.

izr. prof. dr. **Matjaž KRNC**
Univerza na Primorskem

Menim, da je učbenik pripravljen sistematično in dobro predstavi osnove sistemske administracije operacijskega sistema Linux.

doc. dr. **Grega VRBANČIČ**
Univerza v Mariboru