Osnove Unixa in interneta

Eric Raymond, esr@thyrsus.com v2.4, 12. junij 2001

Na poljuden način so opisane osnove delovanja osebnih računalnikov, Unixu podobnih operacijskih sistemov in interneta.

Kazalo

1 Uvod

1.1 Namen tega pisanja

Ta spis je pisan za tiste uporabnike Linuxa in interneta, ki se učijo s poskušanjem. To je sicer neprekosljiv način, kako si pridobimo določene veščine, vendar pa včasih pusti nenavadne luknje v poznavanju osnov – luknje, ki nam zato otežujejo kreativno razmišljanje in učinkovito odpravljanje napak, saj nam manjka dober miselni model, kaj se pravzaprav dogaja.

V preprostem in jasnem jeziku bomo poskusili opisati, kako stvari delujejo. Ta opis je prirejen za ljudi, ki uporabljajo Unix ali Linux na osebnih računalnikih (PC). Na sistem se bomo navadno sklicevali kot na Unix, saj je večina opisanih stvari neodvisna tako od strojnega okolja, kot od posamične izvedbe Unixa.

Predpostavili bomo, da uporabljate osebni računalnik s procesorjem Intel ali z njim združljiv. Podrobnosti so nekoliko drugačne pri računalnikih s procesorji Alpha, PowerPC ali katerimi drugimi, osnovni koncepti pa ostajajo enaki.

Že povedanih stvari ne bomo ponavljali. To po eni strani pomeni, da morate brati pazljivo, po drugi pa tudi, da bo vse prebrano nekaj novega. Dobra misel je, da prvič sestavek samo preletite, potem pa se vračate in prebirate razdelke, dokler vam ni vse jasno.

Sestavek je delo, ki se neprestano razvija in dopolnjuje. Na željo bralcev občasno dodamo nove razdelke, tako da se splača vsake toliko časa vrniti in pogledati, kaj je novega.

1.2 Nove izdaje

Nove izdaje spisa so periodično objavljene v novičarskih skupinah <news:comp.os.linux.help>, <news:comp.os.linux.announce> in <news:news.answers>; ažurirane izdaje pa so objavljene tudi na različnih spletiščih in strežnikih FTP, med njimi tudi na strežniku Linux Documentation Project.

Najbolj ažurna izdaja je vedno na voljo na naslovu `http://www.linuxdoc.org/HOWTO/Unix-and-Internet-Fundamentals-HOWTO/index.html`. Ob slovenskem prevodu, ki ga ravnokar berete in je na voljo na naslovu `http://www.lugos.si/delo/slo/HOWTO-sl/Unix-and-Internet-Fundamentals-HOWTO-sl.html`, je spis preveden tudi v poljščino: `http://www.gszczepa.hg.pl/esr1iso2.htm`.

1.3 Odzivi in popravki

Vprašanja in komentarje lahko naslovite na pisca, Erica S. Raymonda <esr@thyrsus.com>. Pripombe in kritike so vedno dobrodošle. Še posebej dobrodošle so povezave na spletne strani, kjer so posamezni koncepti razloženi bolj v podrobnosti. Če mislite, da ste v besedilu našli napako, bomo veseli, če jo javite, da jo bomo lahko v naslednji izdaji popravili. Vnaprej hvala!

Pripombe na slovenski prevod naslovite na prevajalca: Primož Peterlin, `primoz.peterlin@biofiz.mf.uni-lj.si`.

1.4 Podobni spisi

Če to berete z namenom, da bi se naučili, kako postati heker, vas bo morda zanimal tudi spis „How To Become A Hacker FAQ“, dostopen na naslovu <http://www.tuxedo.org/~esr/faqs/hacker-howto.html>. Med drugim vsebuje tudi nekaj povezav na druge uporabne vire.

2 Osnovna anatomija vašega računalnika

Elektronsko vezje, s katerim računalnik dejansko računa, imenujemo procesor. Ob njem je v računalniku tudi notranji pomnilnik (v svetu DOS/Windows znan tudi kot „RAM“, uporabniki Unixa pa ga dostikrat imenujejo tudi „core“ – spomin na čase, ko je bil notranji pomnilnik dejansko sestavljen iz feritnih obročkov). Procesor in pomnilnik sta nastanjena na *matični plošči*, ki je osrčje računalnika.

Računalnik ima tudi zunanje enote: monitor in tipkovnico, disk in disketnik. Monitor, disk in disketnik imajo svoje *krmilnike* – elektronske kartice, ki so priključene na matično ploščo. Tipkovnica je tako enostavna enota, da ne potrebuje posebnega krmilnika, ampak je ta vgrajen kar v samo ohišje tipkovnice.

Pozneje bomo bolj v podrobnosti opisali, kako te enote delujejo. Za zdaj pa nekaj osnovnih stvari:

Vsi deli računalnika znotraj ohišja so priključeni na *vodilo*. Fizično je vodilo tisto, na kar priključimo krmilniške kartice (grafična kartica, krmilnik za disk, zvočna kartica). Vodilo je podatkovna avtocesta med procesorjem, zaslonom, diskom, in vsem ostalim.

Če ste v povezavi z osebnimi računalniki naleteli na izraze ISA, PCI ali PCMCIA, in niste vedeli, kaj pomenijo: to so vrste vodil. Vodilo ISA je razen nekaj podrobnosti enako, kot je bilo v prvotnem računalniku IBM PC leta 1980, zdaj ga srečamo čedalje bolj poredko. Vodilo PCI (Peripheral Component Interconnection) dandanes srečamo v vseh sodobnih računalnikih, vključno z novejšimi računalniki Macintosh. PCMCIA je izvedba vodila ISA z drobnejšimi priključki, ki se uporablja v prenosnikih.

Procesor, ki poganja vse v računalniku, ne more neposredno dostopati do drugih enot v računalniku – z njimi se mora sporazumevati po vodilu. Edina izjema pri tem je pomnilnik, do katerega ima procesor takorekoč takojšen dostop. Programi lahko tečejo, če so naloženi *v pomnilniku* (core).

Ko računalnik prebere program ali podatke z diska, to stori tako, da zahtevek za branje po vodilu pošlje krmilniku diska. Nekaj pozneje krmilnik – spet po vodilu – sporoči procesorju, da je podatke prebral z diska in jih shranil na zahtevano mesto v pomnilniku. Procesor lahko tedaj dostopa do podatkov v pomnilniku.

Tudi tipkovnica in monitor se s procesorjem sporazumevata po vodilu, vendar je to bolj preprosto. Več o tem bomo povedali pozneje. Za zdaj je dovolj, če razumete, kaj se zgodi, ko vključite računalnik.

3 Kaj se zgodi, ko računalnik vklopimo?

Izklopljen računalnik, v katerem ne teče noben program, je zgolj inerten kup elektronskih vezij. Prva stvar, ki jo mora računalnik ob vklopu opraviti, je pognati poseben program, ki se imenuje *operacijski sistem*. Njegova naloga je, da drugim programom pomaga tako, da nase prevzame umazane podrobnosti komuniciranja s strojno opremo.

Angleški izraz za zagon operacijskega sistema je „*booting*“, kar izvira iz besede „*bootstrap*“ (jermenček na škornju, denimo kavbojskem). Zagon računalnika iz začetnega mrtvila z lastno močjo je namreč nekaj podobnega, kot če bi se hoteli sami dvigniti v zrak, tako da bi se prijeli za jermenčke na škornjih in vlekli navzgor. Ne bo šlo. Računalnik pa se lahko zažene z lastno močjo, ker ima navodila za zagon zapisana v enem od svojih čipov, tako imenovanem BIOS (Basic Input/Output System, osnovni vhodno-izhodni sistem).

BIOS naroči procesorju, naj pogleda na dogovorjeno mesto, navadno na disk z najnižjo številko (tako imenovani *zagonski disk*), in na njem poišče poseben program, *zagonski nalagalnik*. V Linuxu se ta imenuje LILO. Zagonski nalagalnik se naloži v pomnilnik in požene. Njegova naloga je, da požene pravi operacijski sistem.

Nalagalnik mora zato poiskati *jedro* operacijskega sistema, ga naložiti v pomnilnik in pognati. Ob zagonu Linuxa vidite napis „LILO“, ki mu sledi vrsta pikic. Takrat LILO nalaga jedro operacijskega sistema. Vsaka pikica pomeni, da je naložil en *diskovni blok* kode v pomnilnik.

Zakaj je potreben vmesni korak z zagonskim nalagalnikom, namesto da bi BIOS kar sam naložil jedro v pomnilnik? Dva razloga sta za to. Prvič, BIOS za osebne računalnike IBM PC in njim podobne je bil napisan pred več kot dvajsetimi leti in v resnici ni sposoben uporabljati tako velikih diskov, kot so v rabi danes. In drugič, vmesna stopnja z zagonskim nalagalnikom nam takrat, ko imamo na disku nameščenih več operacijskih sistemov, ponuja možnost, da izberemo, katerega bomo zagnali.

Ko je jedro naloženo, se mora razgledati naokoli, poiskati preostanek strojne opreme in pripraviti na poganjanje programov. Najprej se loti natančnejšega evidentiranja priključenih naprav. Prečeše tisti del naslovnega prostora, ki se imenuje *vhodno-izhodna vrata* (angl. I/O port). To je dogovorjeni način sporazumevanja procesorja z vhodno-izhodnimi enotami. Iz odziva na V/I vratih jedro poskusi ugotoviti, katere naprave ima na voljo. Ta stopnja je znana kot „*autoprobing*“. Opazujemo jo lahko ob zagonu, ko jedro sproti izpisuje na zaslon, katere naprave je našlo.

Jedro Linuxa je zelo dobro pri prepoznavanju strojne opreme, boljše od večine drugih Unixov in veliko boljše od sistemov DOS ali Windows. Mnogi stari linuxovski mački so celo mnenja, da je spretnost Linuxa pri prepoznavanju strojnega okolja – kar posledično pomeni relativno enostavnost namestitve – bila ključna za to, da je od vseh prostih izvedb Unixa ravno Linux pridobil kritično maso uporabnikov.

S tem, ko se je jedro naložilo v pomnilnik in teče, zagona še ni konec. To je le *prva raven delovanja* (angl. *run level 1*). Po prvi stopnji, jedro preda nadzor posebnemu procesu, imenovanemu `init`, ki nadalje požene še več strežniških procesov.

Init najprej pregleda, ali je zapis na diskih videti normalen. Če je, jih priklopi. Lahko pa se zgodi, da ni – morda je z diskom v resnici kaj narobe ali pa samo sistem nazadnje ni bil pravilno ugasnjen in so podatki na disku ostali nekako napol zapisani. To se rado zgodi, če sredi dela zmanjka elektrike. Jedro poskusi zapise na disku spraviti v red in če misli, da je uspelo, disk priklopi. Pozneje si bomo v razdelku 11.3 (Kako gredo stvari narobe) ogledali spravljanje zapisov v red bolj v podrobnosti.

V naslednjem koraku `init` požene vrsto specializiranih strežniških procesov. Angleški izraz zanje je *daemon*. To so programi, ki tečejo, ne da bi karkoli brali s tipkovnice ali pisali na zaslon. Vsak od njih skrbi za svoje opravilo, na primer tiskanje, izvajanje periodičnih opravil in podobno. Vsak od njih mora koordinirati potencialno konfliktne zahtevke. Zakaj vsakega zahtevka ne obdeluje svoj proces? Zaradi enostavnosti. Bolj enostavno je napisati strežniški proces, ki ves čas teče in ve vse o zahtevkih, kot pa skrbeti, da si cel trop procesov, ki obdelujejo vsak svoj zahtevek, tečejo pa lahko tudi vsi hkrati, ne hodi po prstih. Nabor strežniških procesov, ki se zaženejo ob zagonu, se razlikuje od sistema do sistema; skoraj vedno pa je med njimi tiskalniški strežnik.

Še en korak naprej se `init` pripravi na gostitev uporabnikov, in požene program, imenovan `getty`. Ta pazi, kaj se dogaja na konzoli, torej tipkovnici. Dodatne kopije istega programa morda pazijo tudi na terminalske in telefonske linije. Getty je tisti, ki izpiše pozivnik `login`, ki ga srečamo ob prijavi. Ko tečejo vsi strežniški procesi, in procesi `getty` pazijo na tipkovnico in vse terminalske linije, smo na *drugi ravni delovanja* (*run level 2*). Na tej točki se lahko prijavimo v sistem in poženemo svoje programe.

A nismo še končali. Ko sistem teče, je naslednji korak zagon strežnikov za različne omrežne storitve (sprejemanje in pošiljanje pošte, streženje spletnim zahtevkom ipd.). Ko se zaženejo ti, smo na *tretji ravni delovanja* (*run level 3*), sistem pa je v celoti pripravljen za delo.

4 Kaj se zgodi, ko se prijavimo v sistem?

Ko vtipkamo uporabniško ime pri pozivniku, ki ga izpiše `getty`, se s tem predstavimo računalniku. Getty naš primer brž preda naslednjemu programu z imenom `login` in umre. Proces `login` nas povpraša po geslu, in ko ga podamo, preveri, ali smo upravičeni do dostopa do računalnika. Če nismo, bo poskus prijave zavrnjen. Če smo, pa `login` postori še nekaj administrativnih opravil, potem pa požene *ukazno lupino*. V resnici pravzaprav ni potrebno, da sta `getty` in `login` različna programa. To je ostalina iz nekdanjih časov, vendar pa o razlogih tu ne bi izgubljali besed.

Še nekaj več o tem, kaj sistem počne, preden požene ukazno lupino – to bomo potrebovali pozneje, ko bomo govorili o dovolilnicah za datoteke. Računalniku se predstavimo z uporabniškim imenom in geslom. Ob prijavi `login` preveri, ali v datoteki `/etc/passwd` obstaja vrstica za podano uporabniško ime. Vsakemu uporabniškemu računu ustreza zapis (fizično ena vrstica) v tej datoteki.

Zapis o posameznem uporabniškem računu je razdeljen na polja. Eno od teh polj vsebuje šifrirano obliko gesla za uporabniški račun (ponekod so šifrirana gesla hranjena ločeno, v datoteki `/etc/shadow`, ki ima strožje omejitve za branje – to še dodatno pripomore k večji varnosti sistema). Geslo, ki ga vtipkate, se šifrira z enakim postopkom, nato pa `login` preveri, ali se šifrirano geslo ujema s šifriranim geslom v datoteki `/etc/passwd`. Varnost tega postopka leži v šifrirnem postopku – gesla se šifrirajo z algoritmom, s katerim je šifriranje bistveno lažje od dešifriranja. Zato nekomu, ki je izvedel naše šifrirano geslo, to še vedno ne omogoča dostopa do sistema. Po drugi strani pa tudi mi sami, če smo geslo slučajno pozabili, nimamo praktično nobene možnosti, da bi ga izvedeli – sistemski upravnik ga lahko kvečjemu spremeni v neko drugo geslo.

Ko smo uspešno prijavljeni, nam pripadejo vse pravice, ki pritičejo našemu uporabniškemu računu. Poleg tega smo lahko tudi člani ene ali več *skupin uporabnikov*. Skupina uporabnikov – ustvari jo lahko upravnik sistema – ima pravice, neodvisne od pravic posameznih članov. Več o 11.2 (pravicah) do dela z datotekami bomo povedali pozneje.

(Čeprav se na uporabniška imena in skupine navadno sklicujemo z njihovimi imeni, so v resnici shranjena kot števil čne identifikacijske vrednosti. Datoteka z gesli `/etc/passwd` vsebuje preslikavo med uporabniškimi imeni in identifikacijskimi številkami (angl. user identification, UID), datoteka `/etc/group` pa preslikavo med imeni skupin in identifikacijskimi številkami skupin (group identification, GID). Vsi ukazi, ki imajo opraviti z uporabniškimi ra čuni ali skupinami, opravijo pretvorbo avtomati čno.)

Zapis o uporabniškem računu v datoteki `/etc/passwd` vsebuje tudi podatke o *domačem imeniku* posameznega uporabnika. To je prostor v strukturi imenikov, namenjen našim osebnim datotekam. In, ne nazadnje, zapis o uporabniškem računu vsebuje tudi podatek o tem, katero ukazno lupino uporabljamo, in ta se bo pognala vedno, ko se prijavimo v sistem.

5 Kaj se zgodi, ko iz ukazne lupine poženemo program?

Ukazna lupina je tolmač, ki ga Unix uporablja za ukaze, ki jih vtipkamo. Lupina ji pravimo zato, ker ovija in skriva jedro operacijskega sistema. Pomembna lastnost Unixa je, da sta jedro in ukazna lupina dva lo čena programa, ki se sporazumevata prek razmeroma majhnega števila sistemskih klicev. To omogo ča obstoj več različnih ukaznih lupin, od katerih vsaka ustreza svojemu okusu glede uporabniških vmesnikov.

Običajno se nam ukazna lupina po prijavi oglasi s pozivnikom `$` (razen če tega nismo ukrojili po svoje). Tu ne bomo izgubljali besed o skladnji ukazne lupine in enostavnih trikih, ki jih lahko pri čaramo na zaslon – namesto tega si bomo s stališča računalnika ogledali, kaj se dogaja v ozadju.

Po zagonu, in preden poženemo kak program, si lahko ra čunalnik predstavljamo kot cel zverinjak procesov, ki vsi čakajo, da se bo zgodilo nekaj, na kar se bodo odzvali. Čakajo na *dogodke* (angl. event). Dogodek je lahko pritisk tipke ali premik miške. Če je računalnik vključen v omrežje, je dogodek lahko tudi podatkovni paket, ki je prispel iz omrežja.

Eden teh procesov je tudi jedro. Jedro je poseben proces, saj dolo ča, kdaj lahko tečejo *uporabniški procesi*, navadno pa je tudi edini proces, ki lahko dostopa do strojne opreme. Uporabniški procesi morajo zato nasloviti zahtevek jedru, če želijo prebrati vnos s tipkovnice, izpisati znak na zaslon, zapisati ali prebrati podatke z diska, in bolj ali manj karkoli razen mletja podatkov v pomnilniku. Takim zahtevkom jedru pravimo *sistemski klici*.

Praviloma gredo vse vhodno-izhodne operacije prek jedra, tako da lahko to razporedi zahtevke in prepre či, da bi ti drug drugemu hodili v zelje. Izjemoma je procesu dovoljeno, da zaobide jedro in neposredno komunicira z vhodno-izhodnimi vrati. Strežnik X (program, ki v ve čini sistemov Unix obravnava zahtevke drugih programov za pisanje po zaslonu) je ena takih izjem. A nismo še pri strežnikih X – za zdaj se ukvarjamo s pozivnikom ukazne lupine v tekstovnem načinu.

Ukazna lupina je zgolj uporabniški proces, in ni č kaj posebno odlikovan. Čaka na pritisk tipk, tako da (prek jedra) bere vhodno-izhodna vrata tipkovnice. Ko jedro zazna pritisk tipke, ga izpiše na zaslon. Če je to tipka **Enter**, jedro poda celotno natipkano vrstico ukazni lupini. Ta jo zatem poskusi raztolma čiti kot niz ukazov.

Denimo, da smo z namenom, da bi pognali program, ki izpiše seznam datotek, vtipkali `ls`. Ukazna lupina uporabi vgrajena pravila in z njihovo pomo čjo ugotovi, da želimo izvesti datoteko `/bin/ls`. Zato s sistemskim klicem posreduje jedru zahtevek, naj požene datoteko `/bin/ls`. To ga požene kot proces potomec ukazne lupine (ki je zahtevala njegovo ustvarjenje), in na potomca prestavi pravice ua dostop do tipkovnice in zaslona. Dokler `/bin/ls` ne opravi svojega, ukazna lupina spi, na zaslon pa se izpisuje izhod procesa potomca.

Ko `/bin/ls` konča, novico o zaključku s sistemskim klicem *exit* sporoči jedru. Jedro prebudi spečega roditelja in mu vrne pristojnosti za dostop do tipkovnice in zaslona. Ukazna lupina izpiše pozivnik in spet čaka na novo vrstico ukazov.

Medtem, ko se ukaz `ls` izvaja, se lahko dogajajo še druge reči (predpostavimo, da gre za zelo dolg seznam datotek): poženemo lahko drugo virtualno konzolo, se prijavimo, poženemo igrico, in podobno. Ali, če je računalnik vključen v omrežje, lahko medtem, ko se izvaja `/bin/ls`, pošilja ali sprejema pošto.

6 Kako delujejo enote in prekinitve?

Tipkovnica je zelo enostavna vhodna enota: enostavna, ker generira majhne količine podatkov, in to zelo počasi (za računalniške pojme). Ko pritisnemo tipko, elektronika v tipkovnici po kablu sporoči zahtevek po *strojni prekinitvi*.

Naloga operacijskega sistema je, da pazi na takšne zahtevke. Za vsako vrsto prekinitve obstoja *prekinitveni servisni program*, ki je del jedra operacijskega sistema, ki prebere in vse potrebne podatke o prekinitvi – v tem primeru kodo pritisnjene ali spuščene tipke – in jih shrani, dokler ne pridejo na vrsto za obdelavo.

Prekinitveni servisni program za tipkovnico ima razmeroma lahko delo – kode tipk prepiše na sistemsko območje v pomnilniku. Tam bodo počakale, da jih pregleda program, ki ga je jedro trenutno pooblastilo za branje s tipkovnice.

Tudi bolj zapletene vhodne enote, denimo disk ali omrežna kartica, delujejo podobno. Prej smo omenili, da diskovni krmilnik po vodilu sporoči, ko je zahtevek za branje z diska izveden. V resnici diskovni krmilnik sproži zahtevek za prekinitev. Prekinitev prestreže ustrezni servisni program, ta pa prepiše podatke v pomnilnik in poskrbi, da bo program, ki je sprožil zahtevo po prebiranju diska, podatke tudi našel.

Prekinitve so razvrščene po *prednostnih ravneh*, s čimer je določen vrstni red obdelave prekinitev. Prekinitve z nizko pomembnostjo (denimo tipkovnica) morajo počakati, da se najprej obdelajo morebitne prekinitve z visoko pomembnostjo (ura, disk). Unix daje prednost dogodkom, ki morajo biti obdelani hitro, da lahko sistem teče čim bolj gladko.

Med sporočili ob zagonu ste morda opazili oznake *IRQ*, ki so jim sledile različne številke. Morda veste tudi, da je eden od pogostih načinov, kako računalnik slabo nastavimo, ta, da dvema enotama določite, naj uporabljata isti IRQ – ne veste pa povsem, zakaj je tako.

Tu je odgovor. IRQ je kratica za „interrupt request“ (angl. zahtevek za prekinitev). Operacijski sistem ob zagonu ugotovi, katera enota bo uporabljala prekinitev s katero številko, in ji priredi ustrezni prekinitveni servisni program. Če bi dve napravi nenadzorovano uporabljali isti signal IRQ, bi zahtevke z ene od naprav obdelal napačen servisni program. V najboljšem primeru to pomeni vsaj to, da je enota blokirana, nekatere operacijske sisteme pa lahko tak odziv včasih zmede tako hudo, da se zrušijo.

7 Kako zmore računalnik opravljati več stvari naenkrat?

Saj jih v resnici ne. Računalnik lahko obenem opravlja le eno opravilo (ali *proces*) naenkrat. Pač pa lahko računalnik izjemno hitro preklaplja med opravili, tako da mnogo počasnejšim ljudem ustvari privid hkratnega izvajanja. Temu pravimo tudi *sistem z dodeljevanjem časa*.

Eno od opravil jedra je tudi dodeljevanje procesorskega časa procesom. Ta del jedra se imenuje *razporejevalnik opravil* (angl. scheduler), in v sebi hrani vse potrebne podatke za vse uporabniške procese v zverinjaku. Vsako stotinko sekunde se v jedru sproži prekinitev; prestreže jo razporejevalnik in začasno ustavi proces, ki trenutno teče, ter požene naslednji proces s seznama.

Stotinka sekunde se morda ne sliši dosti, vendar pa današnji mikroprocesorji v tem času izvedejo na tisoče strojnih ukazov, ki lahko postorijo kar nekaj. Tako da, tudi če obenem teče veliko procesov, lahko vsak od njih vseeno opravi kar nekaj v dodeljenem mu času.

V praksi program ni vedno deležen vsega razpoložljivega časa. Če med izvajanjem prispe zahtevek za prekinitev, jedro ustavi trenutno opravilo in požene prekinitveni servisni program. Šele ko ta opravi svoje, se začne tekoče opravilo spet izvajati. Plaz prekinitev visoke pomembnosti lahko tako povsem zaustavi normalno obdelavo – na srečo pa pri sodobnih Unixih redko pride do tega.

Hitrost, s katero se program izvaja, je v resnici zelo redko omejena s procesorskim časom, ki mu je na voljo (izjema so seveda računsko intenzivne operacije, denimo generiranje zvoka ali tridimenionalne grafike). Dosti pogosteje so zastoji pogojeni s tem, da program čaka na podatke z diska ali omrežja.

Operacijski sistem, ki podpira več hkratnih procesov imenujemo *večopravilni sistem*. Družina operacijskih sistemov Unix je bila zasnovana z mislijo na večopravilnost in je v tem pogledu zelo dobra – dosti bolj učinkovita kot Windows ali Mac OS, pri katerih so večopravilnost dodali pozneje, in je izvedena slabše. Učinkovita in zanesljiva večopravilnost je v veliki meri zaslužna za uspeh Linuxa kot omrežnega strežnika.

8 Kako preprečimo, da bi procesi drug drugemu hodili v zelje?

Razporejevalnik opravil skrbi za časovno ločitev procesov, torej za to, da si naenkrat lasti procesor le en proces. Poleg tega moramo procese ločiti tudi prostorsko, tako da procesi uporabljajo le kos pomnilnika, ki jim je dodeljen, ne pa tudi prostora, dodeljenega drugim procesom. Celo če predpostavimo, da bi programi poskušali biti kooperativni, ne moremo dopustiti, da bi napaka v enem od njih poškodovala druge. Naloge operacijskega sistema v zvezi z dodeljevanjem prostora so znane pod imenom *upravljanje pomnilnika* (angl. memory management).

Vsak proces v zverinjaku potrebuje svoj kos pomnilnika, torej prostor, v katerem bo izvajal svojo kodo ter hranil spremenljivke in rezultate. Lahko si zamislimo, da del tega prostora zaseda *programski segment*, ki hrani programske ukaze v strojnem jeziku in iz katerega lahko samo beremo, ter *podatkovni segment*, ki vsebuje vse spremenljivke, uporabljene v programu, in v katerega je mogoče tudi pisati. Podatkovni segment se res razlikuje od enega procesa do drugega; programski pa ne nujno: če dva procesa poganjata isto programsko kodo, Unix varčuje s prostorom in samodejno poskrbi, da je programski segment naložen v pomnilnik samo enkrat.

8.1 Virtualni pomnilnik: preprosta razlaga

Učinkovitost ravnanja s prostorom je pomembna, saj je pomnilnik drag. Včasih ga nimamo dovolj, da bi naenkrat držali v pomnilniku vse programe, ki se izvajajo, še posebej, če je med njimi kak velik program, denimo strežnik X. Unix težavo zaobide s tehniko, ki je znana kot *virtualni pomnilnik*. Jedro ne poskuša držati v pomnilniku celotne kode in podatkov za posamezni proces, ampak samo relativni manjši *delovni nabor*. Preostanek pomnilniške slike procesa je shranjen na posebnem prostoru na disku (tako imenovani *izmenjalni prostor*).

V preteklosti, ko so imeli računalniki malo pomnilnika, je „včasih“ iz prejšnjega odstavka pomenilo „skoraj vedno“. Dandanes pomnilnik ni več tako drag, kot je bil včasih, in celo računalniki s spodnjega konca lestvice ga premorejo kar precej. Na sodobnih enouporabniških računalnikih, ki imajo 64 ali več megabajtov pomnilnika, je mogoče poganjati okna X in običajno paleto opravil, ne da bi bilo treba uporabiti izmenjalni prostor.

8.2 Virtualni pomnilnik: podrobnosti

V prejšnjem razdelku smo stvari namenoma poenostavili. Drži, da programi obravnavajo pomnilnik kot dolgo vrsto pomnilniških naslovov, ki lahko presega fizični pomnilnik, in da to iluzijo vzdržujemo z diskovnim izmenjavanjem. Vendar pa premore običajni računalnik nič manj kot pet različnih vrst pomnilnika, in razlike med njimi so pomembne, kadar moramo iz računalnika iztisniti kar največ. Da bi resnično razumeli, kaj se dogaja v računalniku, moramo razumeti tudi, kako delujejo posamezne vrste pomnilnika.

Pet vrst pomnilnika je: procesorski registri, notranji predpomnilnik (na samem čipu), zunanji predpomnilnik (na ločenem čipu), glavni pomnilnik, ter disk. Razlog za toliko različnih vrst pomnilnika je en sam: hitrost se plača. Našteli smo jih od najhitrejšega do najpočasnejšega, oziroma od najdražjega do najcenejšega. Registri so najhitrejša in najdražja oblika pomnilnika, do njih pa lahko dostopamo približno milijardokrat na sekundo. Disk je najpo časnejša in najcenejša oblika; do podatkov na njem lahko dostopamo približno stokrat na sekundo.

Sledi seznam z vrednostmi za običajni namizni računalnik, kot so bile aktualne spomladi leta 2000. Hitrost in velikost bosta s časom naraščali, cene pa padale, vendar pa lahko pričakujete, da bodo razmerja med njimi ostala bolj ali manj enaka. In ta razmerja določajo pomnilniško hierarhijo.

Disk

Velikost 13.000 MB Hitrost: 100 KB/s

Pomnilnik

Velikost 256 MB Hitrost: 100 MB/s

Zunanji predpomnilnik

Velikost 512 KB Hitrost: 250 KB/s

Notranji predpomnilnik

Velikost 32 KB Hitrost: 500 KB/s

Register

Velikost 28 B Hitrost: 1000 KB/s

Celotnega računalnika ne moremo zgraditi iz najhitrejše vrste pomnilnika. Dosti predrago bi bilo – in celo če to ne bi bilo res, hitri pomnilnik ni trajen. Brž ko izključimo napajanje, izgubi svojo vrednost. Zato morajo računalniki imeti tudi neko vrsto trajnega pomnilnika – npr. disk – ki ohrani svojo vrednost tudi, ko izklopimo napajanje. Med hitrostjo procesorjev in hitrostjo diskov pa zeva ogromna vrzel. Srednje tri ravni v hierarhiji procesorjev – notranji predpomnilnik, zunanji predpomnilnik in glavni pomnilnik – obstajajo zgolj, da premostijo to vrzel.

Linux in drugi Unixi uporabljajo *virtualni pomnilnik*. To pomeni, da se operacijski sistem vede, kot da bi imel na razpolago dosti več pomnilnika, kot pa ga je dejansko na voljo. Fizični pomnilnik se obnaša kot vrsta „oken“ na mnogo večjem „virtualnem“ pomnilniku, katerega večina je v katerem koli trenutku shranjena na disku, na posebnem prostoru, imenovanem *izmenjalni prostor*. Ne da bi se uporabniški procesi tega zavedali, operacijski sistem sproti prenaša bloke podatkov (takemu bloku pravimo tudi *page*, slov. stran) iz pomnilnika na disk in nazaj, in tako ustvarja iluzijo velikega pomnilnika. Končni rezultat je, da je tak virtualni pomnilnik dosti večji, pa niti ne dosti počasnejši od fizičnega.

Koliko počasnejši je virtualni pomnilnik od fizičnega je odvisno od tega kako dobro algoritmi za izmenjevanje v operacijskem sistemu predvidijo porabo pomnilnika. Na srečo večina sklicev na pomnilniške lokacije, ki si sledijo v kratkem času, bere oziroma piše v pomnilniške lokacije, ki so tudi prostorsko blizu skupaj. Ta lepa lastnost je znana kot *lokalnost*, oziroma *lokalnost sklicevanja*. Če bi, nasprotno, bili pomnilniški sklici naključno raztreseni po celotnem pomnilniškem prostoru, bi morali ob vsakem sklicu na pomnilniško lokacijo prebrati podatke z diska, in virtualni pomnilnik bi bil enako počasen kot disk. Ker pa programi kažejo lokalnost, lahko operacijski sistem opravi pomnilniška sklicevanja z razmeroma malo branja z diska.

Izkustveno je bilo ugotovljeno, da je najučinkovitejša metoda za široko paleto vzorcev rabe pomnilnika nadvse preprosta: algoritem se imenuje LRU (angl. Least Recently Used, „tisti, ki najdlje ni bil rabljen“). Mehanizem za virtualni pomnilnik naloži diskovni blok v svoj *delovni nabor*, ko se za to pokaže potreba. Če fizičnega pomnilnika ni več na voljo, iz njega zbriše blok, ki najdlje ni bil rabljen. Vse različice Unixa, kot tudi večina drugih operacijskih sistemov, ki uporabljajo virtualni pomnilnik, uporablja to ali ono različico algoritma LRU.

Virtualni pomnilnik je prvi člen v premostitvi vrzeli v hitrostih diska in procesorja, in ga izrecno upravlja operacijski sistem. Podobna, čeprav nekaj manjša vrzel zeva tudi med hitrostjo glavnega pomnilnika in hitrostjo procesorja. Notranji in zunanji predpomnilnik rešujeta ta problem s tehniko, ki je podobna pravkar opisani.

Tako kot se fizični glavni pomnilnik obnaša kot vrsta oken v izmenjalnem prostoru na disku, se tudi zunanji predpomnilnik obnaša kot okna v glavnem pomnilniku. Zunanji predpomnilnik je hitrejši od glavnega (250 milijonov dostopov na sekundo proti 100 milijonom), a manjši. Računalnik - natančneje, upravljalnik pomnilnika - izvaja v njem algoritem LRU na blokih podatkov iz glavnega pomnilnika. Iz zgodovinskih razlogov se tu pomnilniška enota imenuje „vrstica“ (angl. line) namesto „stran“ (angl. page).

Nismo še končali. Še zadnji korak v pospešitvi je notranji predpomnilnik. Ta izvaja algoritem LRU na blokih podatkov iz zunanjega pomnilnika. Je še hitrejši in še manjši - tako majhen pravzaprav, da je kar del mikroprocesorskega čipa.

Če bi radi napisali čim hitrejše programe, je dobro, da vemo te podrobnosti. Programi tečejo tem hitreje, čim večjo lokalnost imajo, saj je tedaj algoritem LRU bolj učinkovit. Najenostavnejši način, da dosežemo, da so programi hitri, je, da so majhni. Če programa ne zavira kopica branj in pisanj z diska ali omrežja, bo navadno tekel s hitrostjo najmanjšega predpomnilnika, v katerega ga lahko shranimo.

Če ne moremo napraviti celotnega programa tako majhnega, se v časih splača potruditi in časovno kritične dele napisati tako, da so čim bolj lokalni. Podrobnosti tehnik za takšna fina uglaševanja presegajo ta navodila; do takrat, ko jih boste potrebovali, boste verjetno že dovolj domači s prevajalnikom, da boste mnoge od njih odkrili sami.

8.3 Enota za upravljanje pomnilnika

Celo če imamo na voljo dovolj fizičnega pomnilnika, da diskovno izmenjevanje ni potrebno, ima del operacijskega sistema, zadolžen za *upravljanje pomnilnika*, še vedno pomembno nalogo. Paziti mora na to, da lahko vsak program spreminja le svoj podatkovni segment – preprečiti mora torej, da bi okvarjen ali zlonameren program poškodoval podatke, ki pripadajo drugemu programu. Zato vodi knjigovodstvo o uporabljenih podatkovnih in programskih segmentih. Vsakič, ko program zahteva dodatni pomnilnik, ali pa ko sprosti pomnilnik (slednje se navadno zgodi, ko program zaključi z delom), mora ažurirati tabelo.

Tabela se uporablja za posredovanje ukazov specializiranemu kosu strojne opreme, imenovanem MMU (angl. memory management unit, enota za upravljanje pomnilnika). Sodobni mikroprocesorji imajo enoto MMU že integrirano na sam procesorski čip. Enota MMU ima možnost, da „ogradi“ posamezna območja pomnilnika, tako da so poskusi poseganja izven tega območja zavrnjeni in izzovejo posebno vrsto prekinitve.

Če ste v Unixu že kdaj naleteli na napako „Segmentation fault, core dumped“ ali kaj podobnega – to je to. Proces je poskusil poseči po delu pomnilnika izven svojega podatkovnega segmenta, kar mu je operacijski sistem preprečil in ga prisilno zaključil. Takšno obnašanje je posledica napake v programu – datoteka `core` s pomnilniško sliko procesa ob smrti, ki jo operacijski sistem ob tej priliki zapiše na disk, je diagnostična informacija, ki naj bi bila v pomoč programerju pri iskanju napake.

Poleg omejitve pomnilnika obstaja še en vidik varovanja procesov pred drugimi procesi. Nadzor želimo imeti tudi nad dostopom do datotek, tako da okvarjen ali zlonameren program ne more poškodovati katere od ključnih datotek na disku. Zato pozna Unix 11.2 (dovolilnice za datoteke), o katerih bomo več povedali pozneje.

9 Kako hrani računalnik podatke v pomnilniku?

Verjetno veste, da je v računalniku vse shranjeno kot zaporedje bitov (angl. binary digit, dvojiška števka - mislimo si jih lahko kot stikala, ki so lahko vključena ali izključena). Tu bomo razložili, kako lahko z biti predstavimo v računalniku številke in črke.

Preden se spustimo v razlago, si moramo razjasniti še pojem *velikosti strojne besede*. Velikost strojne besede je najprimernejša velikost za premikanje informacije sem ter tja. Tehnično je enaka velikosti *registrov* - pomnilnih celic procesorja, ki služijo za aritmetične in logične operacije. Ko govorimo o 32- ali 64-bitnih računalnikih, mislimo na to.

Večina računalnikov - vključno z osebnimi računalniki s procesorji 386, 486, Pentium ipd. - uporablja 32-bitno strojno besedo. Starejši osebni računalniki (286) so uporabljali 16-bitno strojno besedo. Stari veliki računalniki so pogosto uporabljali 36-bitno strojno besedo. Nekaj procesorjev, kot denimo Alpha družbe DEC (ki jo je medtem kupil Compaq), uporablja 64-bitno strojno besedo. Uporaba te bo postala v prihodnjih letih še pogostejša: pri Intelu načrtujejo nadomestitev sedanje serije Pentium z novimi 64-bitnimi procesorji Itanium.

Računalnik si predstavlja pomnilnik kot zaporedje besed, oštevilčenih od nič do neke velike vrednosti, določene s količino pomnilnika, ki jo imamo na voljo. Tudi količina pomnilnika, ki jo lahko procesor naslavlja, je povezana z velikostjo strojne besede - to je bil vzrok, da je bil na starejših računalnikih 286 dostop do večjih količin pomnilnika izjemno zapleten. Teh težav tukaj ne bomo obnavljali, saj starejših programerjev ne želimo spominjati na te more.

9.1 Števila

Cela števila so predstaljena bodisi kot strojne besede, bodisi kot pari strojnih besed, odvisno od velikosti strojne besede. Najbolj običajna predstavitev celih števil je 32-bitna strojna beseda.

Celoštevilčna aritmetika je podobna, ne pa povsem enaka dvojiški aritmetiki v matematiki. Bit z najnižjo vrednostjo pomeni enico, naslednji dvojko, še naslednji štirico in tako dalje, kot v dvojiškem sistemu. Predznačena cela števila pa so predstavljena kot *dvojiški komplementi*. Negativno celo število dobimo iz ustreznega pozitivnega celega števila tako, da invertiramo vse bite le-tega in prištejemo ena. Obseg celih števil na 32-bitnih računalnikih je zato od -2^31 do 2^31 -1. Z oznako ^ smo označili potenciranje: npr. 2^3 = 8. Dvaintrideseti bit se uporablja za oznako predznaka.

Nekateri programski jeziki puščajo tudi možnost *nepredznačene aritmetike*, kar pomeni navaden dvojiški sistem, pri katerem imamo na razpolago naravna števila in nič.

Večina procesorjev in programskih jezikov zmore tudi aritmetične operacije s *plavajočo vejico*. V vse novejše mikroprocesorje je sposobnost za takšno računanje že vgrajena. Števila s plavajočo vejico ponujajo mnogo širši razpon od celih števil, z njimi pa lahko izrazimo tudi ulomke. Načini, kako računamo z njimi, se med seboj nekoliko razlikujejo, vsi pa so preveč zapleteni, da bi o njihovih podrobnostih razpravljali na tem mestu. Njihova skupna značilnost je, da so podobna takoimenovanemu znanstvenemu zapisu, pri katerem število zapišemo kot, denimo 6,022 * 10^26. Pri tem smo število razdelili na *mantiso* (6,022) in eksponent (26) z osnovo 10 (10^26 je število s šestindvajsetimi ničlami).

9.2 Znaki

Znaki so navadno predstavljeni kot nizi sedmih bitov, kodirani po kodnem razporedu ISO 646/ASCII (American Standard Code for Information Interchange, ameriški standardni kod za izmenjavo informacij). V sodobnih računalnikih je 128 znakov, ki jih kodira standard ASCII, kodiranih s spodnjimi sedmimi biti *okteta* (osembitnega zloga oz. bajta). Oktete lahko zlagamo v strojne besede - beseda s šestimi črkami tako zavzema dve strojni besedi v pomnilniku. Razpored kodnega standarda ASCII dobimo z ukazom `man 7 ascii`.

Prejšnji odstavek vsebuje dve nepopolnosti. Manjša od njiju je uporaba izraza oktet - čeravno je tehnično točen, ga skoraj nihče ne uporablja, ampak oktetom kratkomalo pravijo *bajt* ali zlog in predpostavljajo, da so bajti osembitni. Strogo vzeto je bajt sicer širši pojem - starejši 36-bitni računalniki so denimo računali z devetbitnimi bajti, vendar pa danes - in verjetno nikoli več - praktično ni v rabi računalnikov, ki ne bi uporabljali osembitnih bajtov.

Večja nepopolnost se nanaša na izbor kodnega nabora ASCII. V resnici si namreč večina sveta z njim ne more kaj dosti pomagati. V naboru ASCII, ki je sicer povsem dober za angleško rabo v ZDA, manjkajo mnoge črke, ki jih uporabljajo drugi narodi. Celo v Veliki Britaniji je njegova raba omejena, saj ne pozna znaka za funt.

Težavo so poskušali odpraviti na več načinov. Večina uporablja osmi bit, ki ga ASCII ne uporablja, in tako pride do nabora z 256 znaki, katerega spodnjo polovico predstavlja ASCII. Najpogosteje uporabljan med temi nabori je tako imenovan Latin-1 (formalno ISO 8859-1). Ta je tudi privzeti nabor znakov v Linuxu, HTML in oknih X. Microsoft Windows uporablja mutirano izvedenko nabora Latin-1, v kateri so na mestih, ki jih Latin-1 iz zgodovinskih razlogov pušča prosta, dodani znaki kot na primer levi in desni dvojni narekovaji. Nekaj o težavah, ki jih to povzroča, lahko preberemo na strani `<http://www.fourmilab.ch/webtools/demoroniser/>`.

Latin-1 je povsem primeren za večino zahodnoevropskih jezikov, ne pa tudi za slovenščino. Ta si skupaj z drugimi srednje- in vzhodno-evropskimi jeziki (bošnjaščina, češčina, hrvaščina, lužiška srbščina, madžarščina, moldavščina, poljščina, romunščina, slovaščina in srbščina) deli nabor Latin-2 (ISO 8859-2). Še drugi jeziki in pisave (cirilica, grščina, hebrejščina, arabščina) uporabljajo ostale nabore iz družine ISO 8859. Več o tem na strani `<http://czyborra.com/charsets/iso8859.html>`.

Končno rešitev predstavlja 16-bitni standard Unicode oziroma ISO/IEC 10646-1:1993. V prvih 256 znakih se Unicode ujema s standardom ISO 8859-1. Naslednji znaki kodirajo pismenke, potrebne za zapis grščine, cirilice, armenščine, hebrejščine, arabščine, devanagarija, bengalščine, gurmukščine, orijščine, tamilščine, tajščine, laoščine, gruzijščine, tibetanščine, japonske katakane, celotnen nabor korejskih hangulskih pismenk in unificiranih kitajsko/japonsko/korejskih (CJK) pismenk. Več podrobnosti na strani `<http://www.unicode.org/>`.

10 Kako so shranjeni podatki na disku

Če si v Unixu pogledamo disk, vidimo drevo poimenovanih imenikov in datotek. Navadno nas globlji pogled ne zanima, včasih - denimo, ko se nam zruši disk in bi radi rešili podatke na njem - pa je uporabno vedeti tudi, kaj se skriva za tem. Na žalost ni nobenega dobrega načina, kako opisati organizacijo diska od ravni datotek navzdol, tako da bomo ubrali obratno smer, od strojne opreme navzgor.

10.1 Nizkonivojska struktura diska in datotečnega sistema

Površina diska je razdeljena približno tako, kot si sledijo polja pri tarči za pikado: na koncentrične *steze*, vsaka od njih pa še naprej, radialno, na *odseke*. Ker so steze na zunanjem robu diska daljše kot tiste bližje osi, so navadno razdeljene na več odsekov kot notranje. Odseki ali *diskovni bloki* so enako veliki - pri sodobnih Unixih navadno 1 kB (1024 osembitnih besed). Vsak diskovni blok ima svoj naslov, številko diskovnega bloka.

V Unixu je disk razdeljen na *razdelke*. Vsak razdelek vsebuje zaporedje diskovnih blokov, ki ga uporabljamo neodvisno od drugih razdelkov na disku. Razdelke lahko uporabimo bodisi kot *datotečni sistem*, bodisi kot *izmenjalni prostor*. Razlogi za razdelitev diska na razdelke segajo še v čase diskov, ki so bili mnogo počasnejši in so se pogosteje kvarili. Meje med razdelki so morebitno napako omejile na posamezni razdelek, tako da je preostanek diska ostal dostopen. Dandanes so pomembnejše druge lastnosti razdelkov. Na posameznem razdelku lahko dovolimo samo branje, pisanja pa ne, in tako preprečimo, da bi morebitni vsiljivci spreminjali kritične sistemske datoteke. Posamezne razdelke lahko tudi po krajevnem omrežju delimo z drugimi računalniki, česar pa tu ne bomo opisovali v podrobnosti. Razdelek z najnižjo zaporedno številko je posebej odlikovan. To je *zagonski razdelek*. Ob zagonu se z njega prebere jedro.

Posamezni razdelek lahko uporabimo bodisi kot *izmenjalni prostor* (ki se uporablja za *virtualni pomnilnik*), bodisi za *datotečni sistem*, kjer so shranjene datoteke. Izmenjalni prostor jedro obravnava kot linearno zaporedje blokov, pri datotečnem sistemu pa je vse skupaj nekoliko bolj zapleteno, saj potrebujemo evidenco o tem, kateri diskovni bloki pripadajo posamezni datoteki. Ker se datoteke s časom daljšajo, krajšajo in spreminjajo, je malo verjetno, da bo datoteki dodeljeni zaporedje diskovnih blokov. Bolj verjetno je, da bodo razpršeni po celotnem razdelku, kjer bo pa č operacijski sistem našel prostor. Takemu pojavu razpršenosti blokov pravimo tudi *fragmentacija*.

10.2 Imena datotek in imeniki

Vsaka datoteka in vsak imenik v datotečnem sistemu je opisan s podatkovno strukturo, imenovano *inod* (angl. in-ode). Kazalo inodov najdemo pri „dnu" (diskovni bloki z nižjimi številkami) datotečnega sistema (čisto najnižji bloki se uporabljajo za oznake in administrativne zadeve, o katerih tu ne bomo razpravljali). Podatkom (datotekam in imenikom) so namenjani diskovni bloki z višjimi številkami.

Vsak inod vsebuje seznam diskovnih blokov, ki pripadajo posamezni datoteki (kar sicer ni čisto res, oziroma je res samo za majhne datoteke, ampak s takimi podrobnostmi se tu ne bomo ukvarjali). Posebej bi poudarili, da inod *ne vsebuje* imena datoteke ali imenika.

Ime datoteke (ali imenika) je shranjeno v *imeniški strukturi*. To kazalo je enostavnejše od kazala inodov, saj obsega le preslikavo iz imen na številko inoda. Zdaj razumemo, zakaj imajo lahko v Unixu datoteke več pravih imen (ali *trdih povezav*), saj ni nobenega razloga, zakaj ne bi moglo več imen kazati na isti inod.

11 Priklopne točke

V najpreprostejšem primeru je celotni datotečni sistem Unixa shranjen na enem samem razdelku. Čeprav na tako situacijo naletimo včasih na majhnih sistemih za osebno rabo, ni običajna. Navadno se sistem razteza prek več razdelkov, včasih celo prek več diskov. Tako imamo lahko na primer en čisto majhen razdelek za jedro, večji razdelek za druge datoteke, ki spadajo k operacijskemu sistemu, in ogromen razdelek z uporabniškimi datotekami.

Edini razdelek, do katerega imamo dostop takoj ob zagonu računalnika, je *korenski razdelek*. Ta je skoraj vedno tisti razdelek, s katerega smo zagnali sistem. Vsebuje *korenski imenik* datotečnega sistema - vrh razvejene drevesne strukture imenikov.

Da lahko dostopamo do podatkov na ostalih razdelkih, morajo biti ti pridruženi korenskemu. Unix omogoči dostop do njih nekje sredi zagonskega postopka z operacijo, ki ji pravimo *priklop* (angl. mount). Razdelke priklopimo na obstoječi imenik na korenskem razdelku.

Če imamo, denimo, imenik `/usr`, je to verjetno *priklopna točka* za razdelek, ki vsebuje številne programe Unixa, vendar pa noben od njih ni nujno potreben ob zagonu.

11.1 Pot do datoteke na disku

Zdaj lahko na datotečni sistem spet pogledamo od vrha. Ko odpremo datoteko, denimo `/home/esr/WWW/ldp/fundamentals.sgml`, se zgodi naslednje.

Jedro začne iskati pri korenu datotečnega sistema (ta leži vedno na korenskem razdelku), in išče imenik /home. Navadno je /home le priklopna točka za velik razdelek z uporabniškimi programi, ki leži kje drugje, zato jedro sledi na ta razdelek. V vrhnji imeniški strukturi razdelka z uporabniškimi podatki jedro poišče vnos esr in prebere številko pripadajočega inoda. Ko inodu sledi, ugotovi, da gre za imeniško strukturo, in v njej poišče vnos WWW. Ko sledi temu inodu, pride spet do podimenika in v njem poišče ldp, kar ga privede do še enega imeniškega inoda. Odpre ga in v njem poišče inod za fundamentals.sgml. Ta inod ni imenik, ampak vsebuje seznam diskovnih blokov, ki pripadajo datoteki s podanim imenom.

11.2 Lastništvo datotek, dovolilnice in varnost

Da programi namerno ali nenamerno ne bi poškodovali podatkov, katerih ne smejo poškodovati, ima Unix urejen sistem *dovolilnic*. Prvotno - v času, ko je Unix tekel predvsem na velikih in dragih miniračunalnikih - so se uporabljale na sistemih z dodeljevanjem časa, na katerih so uporabnike varovale pred drugimi uporabniki.

Da bi razumeli dovolilnice za datoteke, se moramo spomniti opisa uporabnikov in skupin v razdelku 4 (Kaj se zgodi, ko se prijavimo v sistem). Vsaka datoteka pripada določenemu lastniku in določeni skupini. Na začetku ti zavzameta vrednosti, kot ju ima tisti, ki je datoteko ustvaril, pozneje pa ju lahko spremenimo z ukazoma chown(1) in chgrp(1).

Osnovna dovoljenja, ki jih lahko vsebuje dovolilnica, so dovoljenje za branje, dovoljenje za pisanje (tudi brisanje ali spreminjanje) in dovoljenje za izvajanje (če je datoteka program). Dovolilnica vsebuje tri nabore dovoljenj: za lastnika; za kogarkoli drugega v skupini, ki ji datoteka pripada; ter za kogarkoli drugega. Pravice, ki jih pridobimo ob prijavi v sistem, so pravice do branja, pisanja in izvajanja tistih datotek, katerih dovolilnice se ujemajo z našo identifikacijsko številko ali katero od identifikacijskih številk skupin, ki jim pripadamo, ali pa datotek, ki so dostopne vsem.

Oglejmo si izpis hipotetičnega sistema, da vidimo, kako stvari delujejo in kako Unix prikaže dovolilnice:

```
snark:~$ ls -l notes
-rw-r--r--   1 esr      users         2993 Jun 17 11:00 notes
```

To je navadna podatkovna datoteka. Izpis seznama kaže, da je njen lastnik esr, skupina, ki ji pripada, pa users. Najverjetneje sistem, v katerem je bila ustvarjena, privzeto postavi vsakega uporabnika v to skupino. Druge skupine, na katere tudi naletimo na večuporabniških sistemih, so staff (osebje), admin (uprava) in wheel (sistem). Na enouporabniških sistemih skupine nimajo posebnega pomena. Na nekaterih sistemih je privzeta skupina lahko drugačna - dostikrat je kar enaka uporabniškemu imenu.

Niz znakov -rw-r-r- je *dovolilnica* za dano datoteko. Prvi minus kaže imeniški bit. Če bi bil namesto datoteke imenik, bi tam pisalo d. Sledijo mu tri mesta, ki kažejo dovoljenja lastnika (rw-); sledijo tri mesta, ki kažejo dovoljenja skupine (r-) in nazadnje so tri mesta z dovoljenji za vse ostale (r-). To datoteko lahko bere in spreminja samo lastnik (esr); berejo jo lahko vsi člani skupine users, in berejo jo lahko tudi vsi ostali. Takšna dovolilnica je precej običajna za navadne datoteke.

Zdaj pa še zgled datoteke z nekaj drugačno dovolilnico. Ta datoteka je GCC, prevajalnik GNU C.

```
snark:~$ ls -l /usr/bin/gcc
-rwxr-xr-x   3 root     bin          64796 Mar 21 16:41 /usr/bin/gcc
```

Datoteka pripada uporabniku root in skupini bin. Spreminja jo lahko samo lastnik, bere in izvaja pa kdorkoli. Takšna dovolilnica je običajna za privzeto nameščene sistemske ukaze. Skupina bin je na nekaterih Unixih lastnica sistemskih ukazov (`bin` je okrajšava od „binaren", kar naj bi namigovalo, da gre za programe). Na vašem sistemu morda ta datoteka pripada skupini `root` (razlikovati moramo med uporabnikom `root` in istoimensko skupino!).

Uporabnik `root` z identifikacijsko številko 0 je poseben uporabniški račun s pravico, da zaobide vse varnostne mehanizme. To je obenem zelo uporabno in zelo nevarno. Napaka pri tipkanju, ko smo prijavljeni kot `root`, lahko poškoduje kritične sistemske datoteke, ki se jih isti ukaz, pognan z navadnega uporabniškega računa, ne bi dotaknil.

Ker je račun `root` tako močan, mora biti dostop do njega varovan zelo skrbno. Geslo za uporabnika `root` je najbolj kritičen element varnosti sistema, in zatorej informacija, ki jo bodo morebitni vlomilci najprej iskali.

Ko smo že pri geslih - ne zapisujte si jih nikamor, in ne izbirajte gesel, ki jih je enostavno uganiti, kot denimo ime dekleta, fanta, žene... To je osupljivo pogosta slaba navada, ki vlomilcem izjemno olajša delo. V splošnem se izogibajte besedam iz slovarja - obstajajo programi, ki si pri ugibanju gesel pomagajo s slovarji. Dobra so gesla sestavljena iz besede, ki ji sledi števka, in še ena beseda, denimo ‚shark6cider' ali ‚jump3joy'. S tem razširimo prostor možnih besed toliko, da navadno ugibanje z iskanjem po slovarju ni več uporabno. Seveda ne uporabite podanih zgledov - vlomilci so jih zdaj verjetno že dodali v svoje slovarčke. Zelo uporabna navodila za spisal tudi Mark Martinec z IJS, najdemo jih na naslovu `<http://www.ijs.si/dobro_geslo.html>`.

Pa še tretji zgled.

```
snark:~$ ls -ld ~
drwxr-xr-x  89 esr      users          9216 Jun 27 11:29 /home2/esr
snark:~$
```

Ta datoteka je imenik, kar spoznamo po oznaki `d`, s katero se začne dovolilnica. Vidimo tudi, da lahko nanj piše le lastnik (`esr`) berejo in izvajajo pa vsi.

Dovoljenje za branje nam dovoljuje, da izpišemo seznam datotek (in podimenikov) v danem imeniku. Dovoljenje za pisanje pomeni, da lahko v danem imeniku ustvarjamo in brišemo datoteke - slednje le, če imamo tudi dovoljenje za pisanje izbrane datoteke. Če se spomnimo, da je imenik seznam imen datotek in podimenikov, se nam bodo ta pravila zdela smiselna.

Dovoljenje za izvajanje pri imenikih pomeni, da lahko posežemo v imenik in v njem odpiramo datoteke in podimenike. Dejansko nam dovoli dostop do inodov v tem imeniku. V imeniku, ki nima dovoljenja za izvajanje, ne moremo početi nič.

Občasno srečamo imenike, ki jih lahko vsi „izvajajo", ne pa tudi berejo. To pomeni, da lahko kdorkoli dostopa do datotek in podimenikov v tem imeniku, vendar le, če pozna njihovo celotno ime, saj izpis seznama datotek ni dovoljen.

Pomembno je, da se zavedamo, da so dovolilnice za dani imenik povsem neodvisne od dovolilnic za datoteke, ki jih ta imenik vsebuje. Tako nam dovoljenje za pisanje dovoljuje ustvarjanje novih datotek in spreminjanje obstoječih, vendar slednje le, če ni v nasprotju z dovolilnicami za te datoteke. Nikakor pa z dovoljenjem za pisanje v imenik ne dobimo avtomatično tudi dovoljenja za pisanje v vse datoteke v njem.

Za konec si poglejmo dovolilnico za sam program `login`.

```
snark:~$ ls -l /bin/login
-rwsr-xr-x   1 root     bin         20164 Apr 17 12:57 /bin/login
```

Datoteka ima dovolilnico, kot jo pričakujemo za programe - z edino razliko, da na mestu, kjer bi pričakovali dovoljenje, da jo lastnik izvaja (torej `x`), stoji `s`. Tako v seznamu datotek izgledajo datoteke, ki imajo posebno dovoljenje, imenovano „set-user-id" ali na kratko *setuid*.

Dovoljenje setuid navadno izdamo programom, ki morajo navadnim uporabnikom izjemoma dovoliti katero od pravic uporabnika `root`, vendar seveda nadzorovano. Ko poženemo program, program teče s pravicami lastnika programa in ne s pravicami tistega, ki je program pognal, kot je običajno.

Kot uporabniški račun `root` so tudi programi setuid uporabni, a nevarni. Kdorkoli, ki si lahko podvrže ali spremeni program setuid, katerega lastnik je `root`, lahko požene novo ukazno lupino s pravicami uporabnika `root`. Zato vsi sodobni sistemi Unix pobrišejo bit `setuid` tisti trenutek, ko datoteko odpremo za pisanje. Veliko vdorov v sisteme Unix je povezanih z izkoriščanjem varnostnih vrzeli v programih setuid. Sistemski skrbniki, ki se zavedajo teh težav, so zato pri njih posebej previdni in neradi nameščajo nove.

Površno smo pri razpravi o dovolilnicah preskočili nekaj pomembnih podrobnosti, namreč, kako se novo ustvarjeni datoteki določi dovolilnica, lastnik in skupina. Lastnik je tisti, ki je datoteko ustvaril, pri skupini pa stvari niso tako jasne, saj je uporabnik lahko član več skupin. Vendar pa je med njimi le ena *privzeta skupina* (tista, ki je navedena v `/etc/passwd`), in vse novo ustvarjene datoteke bodo pripadale tej skupini.

Zgodba z začetnimi dovolilnicami je nekoliko bolj zapletena. Privzeta dovoljenja lahko spreminjamo s spremenljivko `umask` v okolju. Spremenljivka `umask` določa, kateri biti naj bodo *izključeni*, ko ustvarimo novo datoteko. Običajna vrednost na večini sistemov je `-----w-` oziroma 002, ki uporabnikom izven privzete skupine ne dovoljuje spreminjanja datoteke. Več o spremenljivki `umask` lahko preberete v priročniku za ukazno lupino.

Tudi izbira skupine pri imeniku je nekoliko zapletena. Na nekaterih Unixih vsak novo ustvarjeni imenik postane član privzete skupine tistega, ki je imenik ustvaril (dogovor izvira iz sistemov Unix System V), na drugih pa podeduje skupino od nadrejenega imenika (kot v sistemih BSD). V sodobnih sistemih Unix, vključno z Linuxom, je privzeta prva možnost, drugo pa lahko vklopimo z nastavitvojo bita `set-group-ID`, kar storimo z ukazom `chmod g+s`.

11.3 Kako gredo stvari lahko narobe

Že prej smo povedali, da so datotečni sistemi občutljive zadeve. Zdaj vemo, da moramo včasih prehoditi precej dolgo pot prek vnosov v imenikih in tabelah inodov, da pridemo do datoteke. Zdaj pa si predstavljajmo, da kakšen od sektorjev na disku postane slab.

Če imamo srečo, je napaka nastala na podatkovnem delu, kar pomeni samo nekaj izgubljenih podatkov v datoteki. Če je nimamo, je lahko napaka nastala v imeniški strukturi ali pa v tabeli inodov. To pomeni, da je cel podimenik sicer ostal na disku, vendar ne moremo do njega. Ali še slabše, struktura lahko ostane okvarjena, tako da kaže na napačne inode ali diskovne bloke. Takšna vrsta okvare je zoprna, ker se zaradi napačne evidence širi tudi izven prvotnega območja in kvari podatke na disku.

Na srečo so tovrstne težave s tem, ko diski postajajo vse bolj zanesljivi, vse redkejše. Vseeno pa Unix vsake toliko časa preveri integriteto datotečnega sistema, da bi odkril morebitne težave. Sodobni sistemi Unix opravijo hiter preizkus integritete sistema ob zagonu vsakič, preden priklopijo razdelek. Vsakih nekaj ponovnih zagonov pa opravijo temeljitejši preizkus, ki traja nekaj minut.

Če vse to daje vtis, da je Unix strahotno zapleten in nagnjen k napakam, bo morda v majhno tolažbo to, da ti preizkusi ob zagonu navadno ujamejo in popravijo napake, *še preden* te zadobijo katastrofalne razsežnosti. Drugi operacijski sistemi navadno tega ne počnejo, s čimer je zagon sicer nekoliko hitrejši, vendar pa se lahko naenkrat znajdete pred dosti bolj zavoženim sistemom, ko ga morate ročno popraviti (če sploh imate pri roki Norton Utilities ali kaj podobnega).

Eden od trendov v trenutnih zasnovah Unixa so *datotečni sistemi z dnevnikom* (angl. journalling file system). Ti vse diskovne transakcije organizirajo tako, da je sistem v zajamčeno konsistentnem stanju, ki ga je mogoče restavrirati ob zagonu. To bo preizkuse integritete ob zagonu znatno skrajšalo.

12 Kako delujejo programski jeziki?

Ogledali smo si že, 5 (kaj se zgodi, ko poženemo program?). Vsak program se na koncu izvede kot zaporedje bajtov, ki so ukazi v *strojnem jeziku* procesorja. Vendar pa se ljudje ne znajdemo najbolje v strojnem jeziku - zadnje čase je to postalo redko celo med hekerji.

Skoraj vsi programi, pisani za Unix, so dandanes napisani v enem od *višjih programskih jezikov*. Izjema je le nekaj malega programov v jedru, napisanih za podporo strojni opremi. (Izraz *višji programski jezik* je bolj ali manj zgodovinska ostalina - poudaril naj bi razliko od *nižjih programskih jezikov*. Slednjo skupino sestavljajo izključno *zbirni jeziki* za različne procesorje. Zbirni jezik predstavlja zgolj človeku razumljiv zapis strojnega jezika.)

Višjih programskih jezikov je več vrst. Da bi razumeli razlike med njimi, se moramo najprej zavedati, da mora biti *izvorna koda* programa, torej tisto, kar programer napiše in je mogoče popravljati, na tak ali drugačen način prevedena v *strojni jezik*, ki ga lahko računalnik edinega izvaja.

12.1 Prevajani programski jeziki

Najpogostejši so *prevajani programski jeziki*. Izvorno kodo prevede v strojni jezik poseben program, ki mu pravimo - logično - *prevajalnik*. Ko ta enkrat ustvari ustrezno strojno kodo, ne prevajalnika, niti izvorne kode ne potrebujemo več. Program lahko poganjamo na drugem računalniku, kjer ne eden ne drugi nista na voljo. Večina programja se razpečuje kot prevedeni programi, in izvorne kode zanje nikoli ne vidite.

Prevedeni programi tečejo zelo hitro in navadno ponujajo najpopolnejši dostop do operacijskega sistema, je pa relativno težko programirati v njih.

Programski jezik C, v katerem je napisan tudi Unix sam, je skupaj s svojo izpeljanko C++ brez dvoma najpomembnejši med prevajanimi programskimi jeziki. Fortran, programski jezik, ki se uporablja predvsem v znanosti in tehniki, spada tudi med prevajane jezike, vendar pa je dosti starejši in manj popoln od C. Drugih prevajanih jezikov v Unixu takoreko č ne srečamo, drugod pa je ponekod za finančne in bančne programe zelo priljubljen cobol.

Svojčas so bili prevajani jeziki pogostejši, vendar pa jih je večina zamrla, ali pa so ostali omejeni na raziskovalna okolja. Če ste novopečeni programer v Unixu in uporabljate prevajan programski jezik, je dandanes to zelo verjetno C ali C++.

12.2 Tolmačeni programski jeziki

Tolmačeni programski jeziki med svojim tekom potrebujejo poseben program, *tolmač*, ki izvorno kodo programa sproti prevaja v izračune in sistemske klice. Koda je tako raztolmačena vsakič znova, ko program poženemo, za delovanje programa pa potrebujemo tudi tolmača.

Tolmačeni jeziki tečejo počasneje kot prevedeni, in so pri dostopu do funkcij operacijskega sistema in strojne opreme pogosto omejeni. Po drugi strani pa je v njih lažje programirati, in so do napak v programu navadno bolj prizanesljivi od prevajanih jezikov.

Veliko pripomočkov v Unixu, vključno z ukazno lupino, `bc(1)`, `sed(1)` in `awk(1)` so manjši tolmačeni jeziki. Tudi basic je navadno tolmačen, in TCL tudi. Zgodovinsko gledano je bil najpomembnejši tolmačeni jezik lisp, ki je uvedel mnoge nove zamisli in izboljšave. Dandanes sta verjetno najpomembnejša čista tolmačena jezika v Unixu ukazna lupina in v urejevalnik Emacs vgrajeni lisp.

12.3 Jeziki, prevajani v psevdokodo

Po letu 1990 se čedalje bolj uveljavlja hibridna oblika jezikov, ki so tako prevajani kot tolmačeni. Izvorna koda programov, napisanih v enem od teh jezikov, je prevedena, vendar ne v strojni jezik, temveč v kompakten strojno berljiv zapis, imenovan *psevdokoda* ali *p-koda*. Ko program poženemo, poženemo tolmač za psevdokodo, ki tolmači program.

Psevdokoda je tolmačena skoraj tako hitro kot program, preveden v strojni jezik, obenem pa ti jeziki ohranjajo prožnost in moč, ki jo dajejo tolmačeni jeziki.

Med jezike, prevajane v psevdokodo, spadajo python, perl in java.

13 Kako deluje internet?

Da bi lažje razumeli, kako deluje internet, si oglejmo, kaj se dogaja pri tipični internetni operaciji - denimo takrat, ko brskalnik usmerimo na spletno stran dokumentacijskega projekta Linuxa. Angleški izvirnik spisa, ki ga berete, dobimo na naslovu:

```
http://www.linuxdoc.org/HOWTO/Unix-and-Internet-Fundamentals-HOWTO/index.html
```

Interno gre v resnici za datoteko `HOWTO/Unix-and-Internet-Fundamentals-HOWTO/index.html` v imeniku s spletnimi stranmi na računalniku `www.linuxdoc.org`.

13.1 Imena in naslovi

Da bi prebrali spletno stran, mora brskalnik vzpostaviti povezavo s spletnim strežnikom, torej z *gostiteljskim računalnikom*, ki hrani omenjeno stran. Prvi korak pri tem je, da ugotovi, kje se sploh nahaja računalnik z *internetnim imenom* `www.linuxdoc.org`. Lokacija gostiteljskega računalnika je podana z njegovim *naslovom IP* (kaj je IP, bomo razložili pozneje).

Naslov IP ugotovi brskalnik tako, da najprej pokliče drug program, *imenski strežnik*. Ta lahko teče na našem računalniku, bolj verjetno pa je, da na kakem drugem računalniku v omrežju. Ko ste računalnik pripravljali za delo z vašim ponudnikom internetnih storitev, ste skoraj gotovo morali nekje vpisati tudi naslov imenskega strežnika, ki teče v omrežju vašega ponudnika internetnih storitev.

Imenski strežniki med seboj komunicirajo, ter izmenjujejo in ažurirajo vse podatke, potrebne za pretvorbo internetnih imen v naslove IP. Imenski strežnik, na katerega smo naslovili zahtevek za naslov IP računalnika `www.linuxdoc.org`, je morda moral zahtevek posredovati naprej trem ali štirim drugim imenskim strežnike, vendar pa se to zgodi zelo hitro, navadno prej kot v sekundi. V naslednjem razdelku si bomo ogledali podrobnosti imenskih strežnikov.

Na koncu imenski strežnik vrne brskalniku podatek, da je številka IP za računalnik `www.linuxdoc.org` enaka `152.19.254.81`. Opremljeni s tem podatkom lahko naš računalnik neposredno izmenjuje podatke z računalnikom `www.linuxdoc.org`.

13.2 Sistem domenskih imen

Celotno omrežje programov in zbirk podatkov, ki sodelujejo pri pretvarjanju internetnih imen računalnikov v naslove IP se imenuje *sistem domenskih imen* (angl. Domain Name System, DNS). Kratica DNS je precej pogosta - dostikrat se za imenske strežnike uporablja izraz „strežnik DNS“. Ogledali si bomo, kako sistem deluje.

Internetna imena računalnikov sestavljajo besede, ločene s pikami. *Domena* je skupina računalnikov, ki si delijo skupno pripono internetnega imena. Domene so lahko del širših domen. Računalnik `www.linuxdoc.org` je denimo del domene `.linuxdoc.org`, ta pa je del domene `.org`.

Za vsako domeno je določen *primarni imenski strežnik*, ki pozna naslove IP za vse računalnike v dani domeni. Za primer okvare primarnega imenskega strežnika so lahko določeni rezervni ali *sekundarni imenski strežniki*, ki vskočijo, če je potrebno. Sekundarni strežniki avtomatično usklajujejo podatke v svojih tabelah s podatki na primarnem strežniku vsakih nekaj ur. Tako se spremembe v tabelah, ki jih izvedemo na primarnem strežniku, samodejno prenesejo naprej.

Zdaj pa k pomembnemu delu. Imenski strežnik za dano domeno *ne pozna* naslovov računalnikov v drugih domenah, niti ne v lastnih poddomenah. Kar mora poznati, so naslovi imenskih strežnikov za te domene. V našem zgledu primarni imenski strežnik za domeno `.org` *ne pozna* naslova računalnika `www.linuxdoc.org`, niti nobenega drugega računalnika v domeni `.linuxdoc.org`, pozna pa naslov imenskega strežnika za domeno `.linuxdoc.org`, in tega lahko vpraša za naslov kateregakoli računalnika v tej poddomeni.

Sistem domenskih imen je drevesno urejen. Povsem pri vrhu so korenski strežniki. Vsakdo pozna naslove IP korenskih strežnikov - vgrajeni so že v programje DNS. Korenski imenski strežniki poznajo naslove imenskih strežnikov za vrhnje domene, kot denimo `.com`, `.org` ali `.si`, ne pa naslovov vseh računalnikov v teh domenah. Imenski strežniki za vrhnje domene poznajo naslove imenskih strežnikov za domene neposredno pod njimi, in tako naprej.

Sistem domenskih imen je bil zasnovan z zamislijo, da minimiziramo količino podatkov o obliki drevesa, ki ga mora poznati vsak računalnik v omrežju. Po drugi strani lahko spremembe v pod-drevesih izvedemo enostavno s spremembami v tabelah primarnega strežnika za to poddomeno.

Ko izdamo zahtevek za naslov računalnika z imenom `www.linuxdoc.org`, se zgodi naslednje. Lokalni imenski strežnik, kateremu smo izdali zahtevek, povpraša korenski strežnik za naslov strežnika za domeno `.org`. Ko ga izve, tega povpraša za naslov strežnika za domeno `.linuxdoc.org`, in tega, ko ga izve, povpraša za naslov računalnik `www.linuxdoc.org`.

Večinoma pa tako dolga pot ni potrebna. Ko je enkrat izvedel naslov, imenski strežnik nekaj časa lokalno hrani tabelo preslikav med internetnimi imeni in naslovi IP, in ga ob naslednjih poizvedbah postreže kar iz lokalne tabele. Zato, ko obiščemo novo spletišče, navadno samo na začetku dobimo sporočilo „Looking up host“, pri naslednjih dostopih do strani na istem strežniku pa gre hitreje. Imenski strežnik zbranih tabel ne hrani za vse večne čase, ampak imajo podatki omejen rok trajanja. Ko ta preteče, mora ponovno opraviti celotno pot poizvedbe. To je pomembno, da imenski strežnik ne hrani napačnih podatkov za naslove, ki so se medtem morda spremenili. Naslov je takoj vržen iz tabele tudi v primeru, ko je računalnik na tem naslovu nedosegljiv.

13.3 Paketi in usmerjevalniki

Brskalnik bi rad na koncu spletnemu strežniku posredoval ukaz, s katerim bi prebral spletno stran:

```
GET /HOWTO/Unix-and-Internet-Fundamentals-HOWTO/index.html HTTP/1.0
```

Zgodi se naslenje. Ukaz se zapre v *paket* - blok bitov, ki je analogen telegramu, in ki vsebuje tri pomembne podatke: *naslov pošiljatelja* (torej naslov IP našega računalnika), *naslov prejemnika* (v danem zgledu `152.19.254.81`) in pa *številko storitve* oziroma *številko vrat*. Slednja je v našem zgledu enaka 80, ki je dogovorjena vrednost za vse spletne poizvedbe.

Računalnik pošlje paket po omrežju (krajevnem omrežju ali telefonskem vodu do ponudnika internetnih storitev), dokler ne pride do specializiranega računalnika, ki mu pravimo *usmerjevalnik*. Usmerjevalnik hrani v pomnilniku zemljevid interneta - ne celotnega, ampak del, ki opisuje omrežno okolico - in pozna poti do usmerjevalnikov za druge omrežne okolice v internetu.

Paket bo verjetno na svoji poti do cilja potoval prek več usmerjevalnikov. Ti so premeteni - hranijo tudi podatke o tem, po kako dolgem času drugi usmerjevalniki potrdijo sprejem paketa, in pakete usmerjajo po tistih linijah, ki so hitrejše. Opazijo tudi, kadar kakšen usmerjevalnik ali kabel odpove, in v takih primerih poiščejo alternativne poti.

Urbano izročilo pravi, da je bil internet zasnovan tako, da bi preživel jedrsko vojno. To sicer ni res, vendar pa je zasnova interneta izjemno dobra v tem, da ob nezanesljivi strojni opremi in povezavah zagotavlja zanesljivo komunikacijsko pot. To je zato, ker so podatki, potrebni za delovanje omrežja, razpršeni po tisočih usmerjevalnikih, namesto da bi bili skoncentrirani le v nekaj ogromnih in ranljivih centralah (tako kot denimo telefonsko omrežje). Zato napake ostanejo lokalizirane, omrežje pa se prilagodi tako, da poišče obvoze.

Ko paket prispe do naslovnega računalnika, ta uporabi podatek o številki vrat, in posreduje paket spletnemu strežniku. Ta ve, komu poslati odgovor, saj je na paketu označen tudi naslov pošiljatelja. Ko spletni strežnik kot odgovor vrne zahtevani spis, se tudi ta razdeli na pakete. Velikost paketov je odvisna od prenosnega sredstva v omrežju in vrste storitve.

13.4 TCP in IP

Da bi razumeli, kako delujejo prenosi več paketov, moramo vedeti, da internet dejansko uporablja za prenos dva *protokola*, enega vrh drugega.

Na spodnji ravni, *IP* (angl. Internet Protocol, internetni protokol), je rešeno, kako posamezni paket prenesti od pošiljateljevega naslova do naslovnikovega - zato so ti naslovi znani tudi kot *naslovi IP*. Vendar pa IP ni zanesljiv protokol - če se paket na poti slučajno izgubi, tega ne pošiljatelj, ne naslovnik morda nikoli ne bosta izvedela. V omrežni terminologiji pravimo takim protokolom *brezpovezavni* protokoli: pošiljatelj enostavno pošlje paket naslovniku in ne pričakuje potrditve prejema.

IP je hiter in cenen. Včasih je hitra in cenena, pa čeprav nezanesljiva komunikacija čisto v redu. Če po omrežju igramo igro Doom ali Quake, je vsaka krogla predstavljena s paketom IP. Če se sem ter tja kakšna izgubi, ni prevelike škode.

Zanesljivost zagotavlja, zgornja raven, *TCP* (angl. Transmission Control Protocol, protokol za nadzor prenosa). Ko se računalnika dogovorita za povezavo TCP (kar opravita s paketi IP), prejemnik ve, da mora v dogovorjenem času pošiljatelju poslati potrdilo o prejetem paketu. Če pošiljatelj potrdila ne prejme, ta paket pošlje še enkrat. Nadalje, pošiljatelj pakete TCP oštevilči, tako da jih lahko prejemnik sestavi v pravilnem vrstnem redu, čeprav so morda prispeli pomešani (kar se rado zgodi, kadar se omrežna povezava med prenosom prekine).

Paketi TCP/IP vsebujejo tudi nadzorno vsoto, s katero lahko prejemnik ugotovi, ali so paketi prispeli nepoškodovani. Nadzorna vsota se izračuna tako, da če je poškodovana bodisi sama nadzorna vsota, bodisi preostanek paketa, bo ponoven izračun le-te zelo verjetno izkazal napako. S stališča nekoga, ki uporablja imenske strežnike in TCP/IP, so internetne povezave videti kot zanesljiva oblika povezave med vrati na pošiljateljevem računalniku in vrati na naslovnikovem računalniku, in se mu ni treba ukvarjati z razdelitvijo sporočil na pakete in sestavljanjem le-teh, nadzornimi vsotami in ponovnim pošiljanjem okvarjenih paketov. Za vse to poskrbijo že ravni pod njim.

13.5 Uporabniški protokol HTTP

Vrnimo se k našemu zgledu. Brskalniki in spletni strežniki se sporazumevajo v *uporabniškem protokolu*, ki teče nad TCP/IP. Slednjega uporablja za enostavno pošiljanje podatkovnih nizov tja in nazaj. Protokol se imenuje HTTP (angl.

Hyper-Text Transfer Protocol, protokol za prenos nadbesedila), in pri zgledu GET malo prej smo že videli, kako je videti ukaz.

Ko ukaz GET prispe do vrat številka 80 računalnika www.linuxdoc.org, bo posredovan spletnemu strežniškemu programu, ki streže vratom 80. Večina internetnih storitev je izvedenih s strežniškimi programi, katerih vsak streže svoja vrata - čaka na promet na danih vratih in izvaja prispele ukaze.

Če je pri zasnovi interneta kakšno splošno pravilo, je to, da so vsi deli čim bolj enostavni in čim bolj dostopni ljudem. HTTP, kot tudi sorodni protokoli, denimo SMTP (angl. Simple Mail Transfer Protocol, preprost protokol za prenos pošte), uporablja preproste besedilne ukaze, ki jih zaključi z znakom za novo vrstico.

To je sicer nekoliko neučinkovito - v nekaterih okoliščinah bi bila uporaba gosto kodiranega binarnega protokola hitrejša. Vendar pa so izkušnje pokazale, da prednosti ukazov, ki jih človek enostavno razume, odtehtajo malenkosten prihranek pri učinkovitosti, ki bi ga pridobili z binarnimi protokoli.

Zatorej je tudi odgovor, ki ga strežnik vrne po protokolu TCP/IP, besedilo. Začetek odgovora bo nekaj podobnega kot to (nekaj vrstic zaglavja je izpuščenih):

```
HTTP/1.1 200 OK
Date: Sat, 10 Oct 1998 18:43:35 GMT
Server: Apache/1.2.6 Red Hat
Last-Modified: Thu, 27 Aug 1998 17:55:15 GMT
Content-Length: 2982
Content-Type: text/html
```

Glavi sledi prazna vrstica, tej pa besedilo spletne strani. Ko je celotno besedilo poslano, se zveza prekine, brskalnik pa izriše stran. Podatki v glavi mu pri tem pomagajo - vrstica Content-Type pove, da gre za nadbesedilni spis v obliki HTML.

14 Dodatno branje

Na naslovu <http://www.linuxdoc.org/HOWTO/Reading-List-HOWTO/> je seznam knjig, ki obravnavajo teme, ki smo se jih tu dotaknili, bolj v podrobnosti. Morda si boste želeli ogledati tudi spis How To Become A Hacker na naslovu <http://www.tuxedo.org/~esr/faqs/hacker-howto.html>.