De Unix en Internet Basis HOWTO

door Eric S. Raymond,
vertaald door Ellen Bokhorst

v1.7, 6 maart 2000

Dit document beschrijft de praktische basis van PC-klasse computers, op Unix lijkende besturingssystemen, en het Internet in een niet-technische taal.

Inhoudsopgave

1 Introductie

1.1 Doel van dit document

Dit document is bestemd om Linux en Internet gebruikers, die al doende leren, te helpen. Ondanks dat dit een geweldige manier is om specifieke vaardigheden op te doen, laat het soms bijzondere hiaten in iemands basiskennis – hiaten die het moeilijk kunnen maken, creatief of effectief problemen op te lossen, door een gebrek aan een duidelijke denkwijze over wat er werkelijk aan de hand is.

Ik zal proberen in een duidelijke, eenvoudige taal te beschrijven hoe alles werkt. De presentatie zal worden afgestemd op mensen die Unix of Linux op PC-klasse hardware gebruiken. Niettemin zal ik hier meestal gewoon naar 'Unix' verwijzen, aangezien het meeste wat ik hier zal beschrijven gelijk is voor alle platformen en Unix varianten.

Ik ga ervan uit dat je een Intel PC gebruikt. De details verschillen een beetje als je met een Alpha of PowerPC of een andere Unix box werkt, maar de basisconcepten zijn hetzelfde.

Ik zal niets herhalen, dus je zult op moeten letten, maar dat betekent ook, dat je van ieder woord dat je leest, zult leren. Het is een goed idee dit vluchtig door te nemen als je dit voor het eerst leest; je zou eigenlijk terug moeten keren en het een paar keer herlezen, nadat je in je op hebt genomen wat je hebt geleerd.

Dit is een document dat geleidelijk ontstaat. Het is mijn bedoeling als reactie op feedback van gebruikers secties toe te voegen, dus je zou het periodiek opnieuw moeten bekijken.

2 Wat is nieuw

Nieuw in 1.2: De sectie 'Hoe bewaart mijn computer dingen in het geheugen?'. Nieuw in 1.3: De secties 'Wat gebeurt er als je inlogt?' en 'Eigenaarschap van een bestand, permissies en beveiliging'. Nieuw in 1.4: Wat preciezer geweest in wat de kernel doet vs. wat init doet. Nieuw in 1.7: De sectie over bestandspermissies gecorrigeerd en uitgebreid.

Andere versies bestonden uit het corrigeren van spelfouten en kleine redactionele wijzigingen.

2.1 Gerelateerde bronnen

Als je dit aan het lezen bent, om te leren hacken, zou je ook de *Hoe Wordt je een Hacker FAQ* `<http://www.tuxedo.org/~esr/faqs/hacker-howto.html>` moeten lezen. Er staan een aantal links in naar andere nuttige bronnen.

2.2 Nieuwe versies van dit document

Nieuwe versies van de Unix en Internet Fundamentals HOWTO zullen periodiek worden gepost naar *comp.os.linux.help* en *news:comp.os.linux.announce* en *news.answers*. Ze zullen ook naar diverse Linux WWW- en FTP-sites worden ge-upload, waaronder de homepage van de LDP.

Je kunt de laatste versie hiervan bekijken op het World Wide Web via de URL `<http://metalab.unc.edu/LDP/HOWTO/Unix-Internet-Fundamentals-HOWTO.html>`.

2.3 Feedback en correcties

Als je vragen of commentaar over dit document hebt, mail dan alsjeblieft naar Eric S. Raymond, via *esr@thyrsus.com*. Ik verwelkom alle suggesties of kritiek. Ik verwelkom vooral hyperlinks naar meer gedetailleerde uitleg over individuele concepten. Als je een fout in dit document tegenkomt, laat me dit dan alsjeblieft weten, zodat ik dit in de volgende versie kan corrigeren. Bedankt.

3 Basis anatomie van je computer

Je computer heeft binnenin een processorchip die het werkelijke werk verricht. Het heeft intern geheugen (wat DOS/Windows mensen "RAM" noemen en Unix mensen vaak "core" noemen. De processor en het geheugen bevinden zich op het *moederbord*, het hart van je computer.

Je computer heeft een scherm en toetsenbord. Het heeft harddisks en diskettestations. Het scherm en je disks hebben *controllerkaarten* die in het moederbord worden geplugd en de computer helpen deze apparaten die zich niet direct op het moederbord bevinden te besturen. (Je toetsenbord is te eenvoudig voor een aparte kaart; de controller is in de toetsenbordombouw zelf ingebouwd.)

We zullen later op een aantal details ingaan over hoe deze apparaten werken. Voor nu, zijn hier een aantal basiszaken om in gedachten te houden hoe ze samenwerken:

Alle ingebouwde delen van je computer zijn verbonden via een *bus*. Fysiek is de bus datgene waarin je je controllerkaarten plugt (de videokaart, de diskcontroller, een geluidskaart als je die hebt). De bus is de gegevenssnelweg tussen je processor, je scherm, je disk, en al het andere.

De processor, die ervoor zorgt dat al het andere aan de gang helpt, kan in werkelijkheid geen van de andere delen direct zien; het moet met hen via de bus communiceren. Het enige andere subsysteem waartoe het onmiddellijke toegang heeft, is geheugen (de core). Om programma's te laten draaien, moeten ze zich *in core* (in het geheugen) bevinden.

Wat er in werkelijkheid gebeurt, als je computer een programma of gegevens van disk leest, is dat de processor de bus gebruikt om een diskleesverzoek naar je diskcontroller te zenden. Wat later, gebruikt de diskcontroller de bus om de computer het sein te geven dat het de gegevens heeft ingelezen en het in een bepaald gebied in het geheugen heeft gezet. De processor kan dan de bus gebruiken ook naar dat geheugen te kijken.

Je toetsenbord en scherm communiceren ook via de bus met de processor, maar op een eenvoudiger manier. We zullen deze later bespreken. Voor nu weet je voldoende om te kunnen begrijpen wat er gebeurt als je je computer aanzet.

4 Wat gebeurt er als je een computer aanzet?

Een computer zonder een draaiend programma is gewoonweg een logge homp elektronica. Het eerste wat een computer moet doen als het wordt aangezet is een speciaal programma opstarten dat een *besturingssysteem* wordt genoemd. De taak van het besturingssysteem is te helpen bij het aan het werk krijgen van andere computerprogramma's door de lastige details met betrekking tot het besturen van de computerhardware af te handelen.

Het proces om het besturingssysteem aan de gang te krijgen wordt *booten* genoemd. (van origine was dit *bootstrapping* en zinsspeelde op de moeilijkheid om jezelf "aan je laarzen" op te trekken). Je computer weet hoe het moet booten omdat de instructies voor het booten in één van z'n chips zijn ingebouwd, de BIOS (of Basic Input/Output System) chip.

De BIOS chip vertelt het op een vaste plaats op de laagst-genummerde harddisk (de *bootdisk*) te zoeken naar een speciaal programma dat een *bootloader* wordt genoemd (onder Linux wordt de bootloader LILO genoemd). De bootloader wordt in het geheugen geladen en gestart. De taak van de bootloader is het echte besturingssysteem op te starten.

De loader doet dit door naar een *kernel* te zoeken, deze in het geheugen te laden en het te starten. Als je Linux opstart en je ziet op het scherm "LILO"gevolgd door een groep punten, is het de kernel aan het laden. (Iedere punt betekent dat het een ander *diskblok* van de kernelcode heeft geladen.)

(Het zou kunnen dat je je afvraagt waarom de BIOS de kernel niet direct laadt – waarom het twee-staps proces met de bootloader? Dit komt omdat de BIOS niet erg slim is. In feite is het erg dom, en Linux gebruikt het in het geheel niet na het opstarten. Het werd van origine geschreven voor primitieve 8-bits PC's met hele kleine disks, en het kan letterlijk niet voldoende toegang tot de disk krijgen om de kernel direct te kunnen laden. De bootloader-stap laat je ook één van de verscheidene besturingssystemen vanaf verschillende plaatsen van de disk opstarten, (in het onwaarschijnlijke geval dat Unix niet goed genoeg voor je is.)

Zodra de kernel opstart, moet het om zich heenzoeken om de rest van de hardware te vinden, en zorgen dat het gereed is om programma's te draaien. Het doet dit niet door rond te snuffelen in gewone geheugenlokaties maar eerder via *I/O poorten* – speciale bus adressen waar zich naar alle waarschijnlijkheid device controllerkaarten kunnen bevinden die luisteren in afwachting van commando's. De kernel snuffelt niet willekeurig rond; het heeft een heleboel ingebouwde kennis over wat het vermoedelijk waar kan vinden, en hoe controllers zullen reageren als ze aanwezig zijn. Dit proces wordt *autoprobing* genoemd.

De meeste meldingen die je tijdens het booten ziet, komen voort uit het autoprobing-proces van de hardware via de I/O poorten, door de kernel, uitzoekend wat er beschikbaar is en het zichzelf aanpast aan je computer, beter dan de meeste andere Unixes en *veel* beter dan DOS of Windows. In feite, denken vele Linux gebruikers van de oude stempel dat de slimheid van Linux's onderzoeken tijdens de systeemstart (wat het relatief gemakkelijk maakt om het te installeren) een belangrijke reden was voor de doorbraak van de vrije-Unix experimenten door een kritische massa gebruikers aan te trekken.

Maar de kernel volledig geladen en draaiend krijgen is niet het einde van het bootproces; het is pas de eerste fase (soms *run level 1* genoemd). Na deze eerste fase, geeft de kernel de controle over aan een speciaal proces, genaamd 'init', welke verscheidene beheertaken verricht.

De eerste taak van het init-proces is meestal een controle om er zeker van te zijn dat je disks OK zijn. Disk bestandssystemen zijn kwetsbaar; als ze door een hardware-fiasco of een plotselinge stroomuitval zijn beschadigd, zijn er goede redenen om herstelstappen te ondernemen voordat je Unix weer helemaal in orde is. We zullen later op enkele van deze details ingaan, als we het gaan hebben over 11.6 (hoe het mis kan gaan met bestandssystemen).

De volgende stap van init is het opstarten van verscheidene *daemons*. Een daemon is een programma zoals een print spooler, een mail listener of een WWW-server die zich in de achtergrond verscholen houdt, in afwachting om iets te doen. Deze speciale programma's moeten vaak verscheidene verzoeken die met elkaar in conflict kunnen raken, coördineren. Er zijn daemons, omdat het vaak makkelijker is om een programma te schrijven dat continue draait en bekend is met alle verzoeken dan het zou zijn om te proberen te voorkomen dat een schare kopieén (waarbij ieder een verzoek verwerkt en ze allen tegelijkertijd draaien) elkaar niet in de weg zitten. De bepaalde verzameling daemons die je systeem start kan variëren, maar hier zal bijna altijd een print spooler bij zijn (een portier daemon voor je printer).

Zodra alle daemons zijn gestart, bevinden we ons op *run level 2*. De volgende stap is de voorbereiding op gebruikers. Init start een kopie van een programma met de naam `getty` om je console in de gaten te houden (en misschien meer kopieën om dial-in seriële poorten in de gaten te houden). Dit programma zorgt dat de `login`-prompt op je console te voorschijn komt. We zijn nu op *run level 3* en gereed om in te loggen en programma's te draaien.

5 Wat gebeurt er als je inlogt?

Als je inlogt (een naam en wachtwoord geeft) identificeer je jezelf aan `getty` en de computer. Het draait vervolgens een programma met de naam (natuurlijk genoeg) `login`, welke controleert of je geautoriseerd bent om de computer te gaan gebruiken. Als dat niet zo is, zal je poging om in te loggen worden verworpen. Als het wel zo is, zal login een aantal huishoudelijke zaken verrichten en vervolgens een commando-interpreter, de *shell*, opstarten. (Ja, `getty` en `login` zouden één programma kunnen zijn. Ze zijn gescheiden om historische redenen waarvan het niet waard is ze hier te vermelden).

Bij deze wat meer over wat het systeem doet, voordat het je een shell presenteert; je zal het voor het begrip later nodig hebben, wanneer we het gaan hebben over bestandspermissies. Je identificeert jezelf met een loginnaam en een wachtwoord. De loginnaam wordt opgezocht in een bestand met de naam /etc/password, welke bestaat uit een reeks regels, waarvan ieder een gebruikersaccount beschrijft.

Één van deze velden is een versleutelde versie van het account-wachtwoord. Datgene wat je invoert als een account-wachtwoord wordt op exact dezelfde wijze versleuteld, en het `login`-programma controleert of die twee overeenkomen. Terwijl de conversie van de ingetikte versie naar de versleutelde versie eenvoudig is, hangt de beveiliging van deze methode af van het feit, of het erg moeilijk is om dit proces om te draaien. Dus ook al kan iemand de versleutelde versie van je wachtwoord zien, ze kunnen je account niet gebruiken. (Als je je wachtwoord vergeet, betekent dit bovendien dat er geen manier is waarop je het kunt herstellen, je hebt alleen de mogelijkheid een ander wachtwoord te kiezen).

Zodra je succesvol bent ingelogd, krijg je alle privileges, welke met die individuele account zijn verbonden. Misschien dat je ook als onderdeel van een *group* (groep) wordt herkend. Een groep is een door de systeembeheerder bij naam genoemde verzameling gebruikers. Een groep kan privileges hebben, die onafhankelijk zijn van de privileges van de leden van die groep. Een gebruiker kan onderdeel uitmaken van meerdere groepen. (zie de sectie hieronder over 11.5 (), voor details over hoe Unix-privileges werken).

(Noot: alhoewel je normaal gesproken naar gebruikers en groepen bij naam refereert, worden ze intern in werkelijkheid als numerieke ID's opgeslagen. Het wachtwoordbestand deelt je gebruikersnaam in naar een gebruikers-ID; het `/etc/group` bestand deelt de groepsnamen in naar numerieke groep-ID's. Commando's die te maken hebben met accounts en groepen zorgen automatisch voor de vertaling.

Het record van je account bevat ook je *home-directory*, de plaats in het Unix-bestandssysteem waar je persoonlijke bestanden voorkomen. Tenslotte, stelt het record van je account ook je *shell* in, de commando-interpreter dat door `login` zal worden opgestart, om je commando's te accepteren.

6 Wat gebeurt er als je programma's vanuit de shell draait?

De shell is de interpreter voor de Unix-commando's die je intikt; het wordt een shell genoemd omdat het omhulsel is van de kernel en het de kernel verbergt. De normale shell geeft je de '$' prompt die je na het inloggen ziet (tenzij je het hebt aangepast om iets anders te doen). We zullen het hier niet over shell-syntax hebben en de makkelijke zaken die je op het scherm kunt zien; in plaats daarvan zullen we een blik achter de schermen werpen over wat er vanuit het gezichtspunt van de computer gebeurt.

Na het booten en voordat je een programma draait, kun je aan je computer denken als een dierentuin vol met processen die allen wachten om iets te kunnen doen. Ze wachten allemaal op *events*. Een event kan zijn dat je een toets indrukt of een muis beweegt. Of een event kan een datapakket zijn, dat via het netwerk binnenkomt, als je computer op een netwerk is aangesloten.

De kernel is één van deze processen. Het is een speciale, omdat het bepaalt wanneer de andere *gebruikersprocessen* kunnen draaien, en het is normaal gesproken het enige proces met directe toegang tot de hardware van de computer. In feite moeten gebruikersprocessen een verzoek indienen aan de kernel als ze toetsenbordinvoer willen ophalen, naar je scherm willen schrijven, van of naar disk willen schrijven, of gewoon alles willen doen anders dan in het geheugen vermalen van bits. Deze verzoeken staan bekend als *system calls*.

Normaal gesproken gaat alle I/O via de kernel dus het kan de bewerkingen regelen en voorkomen dat processen elkaar in de weg zitten. Van een paar speciale gebruikersprocessen is het toegestaan dat ze de kernel ongemerkt voorbijgaan, gewoonlijk doordat er directe toegang tot I/O poorten wordt gegeven. X-servers (de programma's die op de meeste Unix boxen grafische schermverzoeken van andere programma's afhandelen) zijn hier het meest algemene voorbeeld van. Maar we zijn nog niet bij de X server aangekomen; je kijkt naar een shell-prompt op een character console.

De shell is gewoon een gebruikersproces, en niet een bijzonder speciaal proces. Het wacht op je toetsaanslagen, luistert (via de kernel) naar de toetsenbord I/O poort. Als de kernel ze ziet, echoot hij ze naar je scherm en geeft ze vervolgens door aan de shell. Als de kernel een 'Enter' ziet geeft het een regel tekst door aan de shell. De shell probeert deze toetsaanslagen als commando's te interpreteren.

Laten we ervan uitgaan dat je 'ls' en Enter intikt om de Unix directorylijst aan te roepen. De shell volgt zijn interne regels om er achter te komen dat je het uitvoerbare commando in het bestand '/bin/ls' wilt uitvoeren. Het genereert een system call door de kernel /bin/ls als een nieuw *kind* proces op te starten en het toegang te geven tot het scherm en toetsenbord via de kernel. Vervolgens gaat de shell slapen, in afwachting tot ls is beëindigd.

Als /bin/ls klaar is, vertelt het de kernel dat het klaar is door een *exit* system call aan te roepen. De kernel schudt vervolgens de shell wakker en vertelt het dat het verder kan gaan met de uitvoering. De shell roept een andere prompt aan en wacht op een andere regel invoer.

Er kunnen zich echter andere dingen afspelen als 'ls' wordt uitgevoerd, (we moeten er van uit gaan dat je een zeer lange directorylijst laat weergeven). Je zou bijvoorbeeld naar een andere virtuele console kunnen schakelen, daar inloggen en het spel Quake opstarten. Of, veronderstel dat je bent aangesloten op het Internet, dan zou je computer mail kunnen verzenden of ontvangen op het moment dat /bin/ls wordt uitgevoerd.

7 Hoe werken apparaten en interrupts?

Je toetsenbord is een zeer eenvoudig invoerapparaat; gewoonweg, omdat het zeer langzaam kleine hoeveelheden gegevens genereert (voor computerstandaards). Als je een toets indrukt of loslaat, wordt die event doorgeseind aan de toetsenbordkabel om een *hardware interrupt* te doen ontstaan.

Het is de taak van het besturingssysteem om dergelijke interrupts in de gaten te houden. Voor iedere mogelijke soort interrupt, zal er een *interrupt handler* zijn, een deel van het besturingssysteem welke enige gegevens die ermee zijn geassocieerd verbergt (zoals je toetsindruk/toetsloslaat waarde), totdat het kan worden verwerkt.

Wat de interrupt handler voor je toetsenbord eigenlijk doet, is de toetswaarde in een systeemgebied vlakbij de onderkant van het geheugen posten. Daar zal het ter inzage beschikbaar zijn als het besturingsssysteem de controle overgeeft aan het programma, waarvan op dat moment verondersteld wordt dat het van het toetsenbord aan het lezen is.

Complexere invoerapparaten zoals disk- of netwerkkaarten werken op een vergelijkbare manier. Hierboven refereerde we naar een diskcontroller die de bus gebruikte om te seinen dat er aan een diskverzoek was beantwoord. Wat er in werkelijkheid gebeurt, is dat de disk een interrupt veroorzaakt. De disk interrupt handler kopieert de ontvangen gegevens vervolgens naar het geheugen, voor later gebruik door het programma dat het verzoek deed.

Iedere soort interrupt heeft een geassocieerde *prioriteiten niveau*. Lagere-prioriteit interrupts (zoals toetsenbord events) moeten wachten op hogere-prioriteit interrupts (zoals kloktikken of disk events). Unix is ontworpen om hoge prioriteit te geven aan de soort events die snel moeten worden verwerkt om ervoor te zorgen dat de response van de computer vlot verloopt.

In de opstartmeldingen van je OS, zou het kunnen dat je verwijzingen tegenkomt naar *IRQ*-nummers. Het kan zijn dat je je er bewust van bent dat één van de algemene manieren om je hardware onjuist te configureren is, dat twee verschillende apparaten dezelfde IRQ proberen te gebruiken, zonder dat je exact begrijpt waarom.

Hier is het antwoord. IRQ staat voor "Interrupt Request". Het besturingssysteem moet bij het opstarten weten welke genummerde interrupts elk hardware-apparaat zal gebruiken, zodanig dat het de juiste handlers met ieder daarvan kan associëren. Als twee verschillende apparaten dezelfde IRQ proberen te gebruiken, zullen interrupts soms naar de verkeerde handler worden gezonden. Dit zal gewoonlijk op z'n minst het apparaat doen vastlopen, en kan het OS soms genoeg in de war brengen dat het vastloopt of crasht.

8 Hoe doet mijn computer verscheidene dingen tegelijkertijd?

Dat doet 't in werkelijkheid niet. Computers kunnen slechts één taak (of *proces*) tegelijkertijd verrichten. Maar een computer kan zeer snel van taak wisselen, en langzame menselijke wezens voor de gek houden door ze te laten denken dat het verscheidene dingen tegelijkertijd doet. Dit wordt *timesharing* genoemd.

Een van de taken van de kernel is het beheren van timesharing. Het heeft een onderdeel dat de *scheduler* wordt genoemd, die alle informatie intern bijhoudt over alle andere (niet-kernel) processen in je dierentuin. Iedere 1/60 van een seconde, gaat er in de kernel een tijdklok af die een klokinterrupt genereert. De scheduler stopt het proces welke erop dat moment ook draait, onderbreekt het op z'n plaats, en geeft de controle over aan een ander proces.

1/60 van een seconde klinkt misschien niet als veel tijd. Maar voor de tegenwoordige microprocessors is het voldoende om tienduizenden machine-instructies uit te voeren, die flink wat werk kunnen verrichten. Dus zelfs als je veel processen hebt lopen, kan ieder ervan heel wat in zijn tijdsfragment volbrengen.

In praktijk kan het zijn dat een programma niet zijn volledige tijdsfragment krijgt. Als een interrupt vanaf een I/O device binnenkomt, stopt de kernel in feite de huidige taak, draait de interrupt handler, en keert daarna terug naar de huidige taak. Een storm hoge-prioriteiten interrupts kan de normale verwerking verdringen; dit wangedrag wordt *thrashing* genoemd en is gelukkig onder moderne Unixes zeer moeilijk te forceren.

In feite is de snelheid van programma's slechts zeer zelden beperkt door de hoeveelheid machinetijd die ze kunnen krijgen (er zijn een paar uitzonderingen op deze regel, zoals het genereren van geluid of 3-D graphics). Veel vaker, worden vertragingen veroorzaakt als het programma op gegevens van een diskdrive of netwerkverbinding moet wachten.

Een besturingssysteem dat volgens vaste regel veel gelijktijdige processen kan ondersteunen, wordt "multitasking"genoemd. De Unix familie besturingssystemen is vanaf het begin af aan ontworpen voor multitasking en is daar erg goed in – veel effectiever dan Windows of de Mac OS, waarbij multitasking als een latere overweging is ingesloten en ze dit nogal armzalig doen. Efficiënte, betrouwbare multitasking is een belangrijk deel van wat Linux superieur maakt voor netwerken, communicaties, en Web service.

9 Hoe zorgt mijn computer ervoor dat processen elkaar niet in de weg zitten?

De scheduler van de kernel zorgt voor het verdelen van de tijd over de processen. Je besturingssysteem moet ze ook qua ruimte verdelen, zodanig dat processen niet in elkaars werkgeheugen kunnen gaan zitten. Zelfs als je er van uit gaat dat alle programma's proberen samen te werken, wil je niet dat een bug in het ene programma de andere programma's kan beschadigen. Datgene dat je besturingssysteem doet om dit probleem op te lossen wordt *geheugenbeheer* genoemd.

Ieder proces in je dierentuin heeft z'n eigen gebied in het geheugen nodig, als een plaats van waaruit het z'n code kan uitvoeren en variabelen en resultaten in op kan slaan. Je kunt je dit voorstellen als een alleen leesbaar *code segment* (waar de instructies van het proces in staan) en een schrijfbaar *data segment* (waarin alle variabelen van het proces zijn opgeslagen). Het data segment is waarlijk uniek voor ieder proces, maar als twee processen dezelfde code uitvoeren, herschikt Unix ze automatisch zodanig dat ze een enkel codesegment delen als een efficiëntie maatregel.

Efficiëntie is belangrijk, omdat geheugen duur is. Soms heb je niet genoeg om het geheel aan programma's die op de computer worden gedraaid, vast te houden, vooral als je een groot programma zoals een X server gebruikt. Om dit te ontduiken, gebruikt Unix een strategie die *virtueel geheugen* wordt genoemd. Het probeert niet alle code en data voor een proces in het geheugen te behouden. In plaats daarvan, blijft het werken met een relatief kleine *werkset*; de rest van de stand van het proces blijft achter in een speciaal *swap space* gebied op je harddisk.

Als het proces draait, probeert Unix vooruit te lopen op hoe de werkset zal wijzigen en heeft slechts hetgeen het nodig heeft in het geheugen. Dit doeltreffend doen is zowel gecompliceerd als lastig, dus ik zal niet proberen het hier allemaal te beschrijven, – maar het hangt af van het feit dat code en dataverwijzingen geneigd zijn in clusters te gebeuren, waarbij het aannemelijk is dat iedere nieuwe cluster naar een oude cluster in de buurt ervan verwijst. Dus als Unix de code of data die het vaakst (of meest recent) wordt gebruikt, in de buurt houdt, zal het er gewoonlijk in slagen tijd te besparen.

Merk op dat in het verleden, dat "Soms"twee paragrafen terug "Bijna altijd"was, – de grootte van het geheugen was kenmerkend klein gerelateerd aan de grootte van uitvoerende programma's, dus er werd frequent geswapt. Geheugen is tegenwoordig veel minder duur en zelfs de goedkoopste computers hebben er heel veel van. Op moderne single-user computers met 64MB of meer geheugen, is het mogelijk om X te draaien en een typische mix taken zonder ooit te swappen.

Zelfs in deze gelukkige situatie, heeft het deel van het besturingssysteem dat *geheugenbeheer* wordt genoemd, nog steeds belangrijk werk te doen. Het moet ervoor zorgen dat programma's alleen hun eigen datasegmenten kunnen wijzigen –dat wil zeggen, voorkomen dat door onjuiste of kwaadwillige code in het ene programma de data in een ander programma overhoop wordt gehaald. Om dit te doen, houdt het een tabel met gegevens en codesegmenten bij. De tabel wordt bijgewerkt als een proces om meer geheugen verzoekt of geheugen vrijgeeft (het laatste meestal als het stopt).

Deze tabel wordt gebruikt om commando's door te geven naar een gespecialiseerd deel van de onderliggende hardware met de naam *MMU* of *memory management unit*. MMU's zijn bovenop moderne processor chips gebouwd. De MMU heeft de speciale mogelijkheid om geheugengebieden af te schermen, dus een buiten-degrens verwijzing zal worden geweigerd en een speciale interrupt veroorzaken.

Als je ooit een Unix melding "Segmentation fault", "core dumpedöf iets vergelijkbaars te zien krijgt, is dit wat er precies is gebeurd; een poging van het uitvoerende programma om toegang tot het geheugen (de core) buiten zijn segment te verkrijgen, heeft een fatale interrupt veroorzaakt. Dit duidt op een bug in de programmacode; de *core dump* die het achterlaat bestaat uit diagnostische informatie met de bedoeling de programmeur te helpen het op te sporen.

Er is nog een aspect om ervoor te zorgen processen tegen elkaar te beschermen, buiten het opdelen van het geheugen dat ze benaderen. Je zal ook de toegankelijkheden van de bestanden willen beheren, zodat een programma met fouten of een kwaadwillig programma kritieke delen van het systeem niet kan beschadigen. Daarom bestaan er onder Unix 11.5 (bestandspermissies), die we later zullen bespreken.

10 Hoe bewaart mijn computer zaken in het geheugen?

Waarschijnlijk weet je al dat alles op een computer als een reeks bits (binary digits) wordt bewaard; je kunt je dit voorstellen als een heleboel aan- en uitschakelingen). We zullen hier uitleggen hoe deze bits worden gebruikt om de letters en nummers weer te geven.

Voordat we hier op in kunnen gaan, moet je enig begrip hebben van de *woordgrootte* van je computer. De woordgrootte is de voorkeursgrootte van je computer voor het manoeuvreren van eenheden informatie; technisch gezien is het de lengte van de *registers* van je processor, wat de opslaggebieden zijn die je processor gebruikt voor de arithmetische en logische berekeningen. Dit is wat mensen bedoelen, als ze schrijven over computers met bit-grootte (waarbij ze het hebben over, bijvoorbeeld “32-bit” of “64-bit”computers).

De meeste computers (waaronder 386, 486, Pentium en Pentium II PC’s) hebben een woordgrootte van 32 bits. De oude 286 computers hadden een woordgrootte van 16. Mainframes van de oude stijl hadden vaak 36-bit woorden. Een paar processors (zoals de Alpha welke van DEC was en nu van Compaq) hebben 64-bit woorden. Het 64-bit woord zal de volgende vijf jaar meer algemeen worden; Intel is van plan de Pentium II door een 64-bit chip met als codenaam ‘Merced’, en nu officieel genaamd de ‘Itanium’, te vervangen.

De computer ziet je geheugen als een reeks woorden genummerd van 0 tot één of ander groot nummer, waarvan de waarde afhankelijk is van de grootte van je geheugen. Die waarde is beperkt door je woordgrootte. Daarom moeten oudere computers, zoals 286’rs, zich door moeizame bochten wringen om grote hoeveelheden geheugen te adresseren. Ik zal ze hier niet beschrijven; ze bezorgen oudere programmeurs nog steeds nachtmerries.

10.1 Getallen

Getallen worden voorgesteld als woorden of stel woorden, afhankelijk van de woordgrootte van je processor. Een 32-bit computerwoord is de meest algemene grootte.

Rekenkunde van gehele getallen (arithmetic integer) nadert, maar heeft niet werkelijk een mathematisch grondtal twee. Het minst significante bit is 1, vervolgens 2, dan 4 enzovoort als in zuiver binair. Maar nummers met een teken worden voorgesteld in *twee-complement* notatie. Het meest significante bit is een *tekenbit* welke de kwantiteit negatief maakt, en ieder negatief nummer kan worden verkregen uit de corresponderende positieve waarde door alle bits om te draaien. Daarom komen integers op een 32-bit computer voor in het bereik -2ˆ31 +1 tot 2ˆ31 -1 (waar ˆ de ‘machts’-bewerking , 2ˆ3=8 is). Het 32e bit wordt gebruikt voor het teken.

Een aantal computertalen geeft je toegang tot *unsigned arithmetic* (rekenkunde zonder teken) welke is gebaseerd op grondtal 2, met verder alleen positieve nummers en nul.

De meeste processors en een aantal programmeertalen kunnen omgaan met *floating-point* getallen (deze mogelijkheid is in alle recente processor-chips ingebouwd). Floating-point getallen geven je een veel breder bereik aan waarden dan gehele getallen (integers) en geven je de mogelijkheid breuken uit te laten drukken. De wijze waarop dit wordt gedaan verschilt en is te gecompliceerd om hier in detail te bespreken, maar in het algemeen lijkt 't veel op de zogenoemde 'wetenschappelijke notatie', waarbij men op zou kunnen schrijven `1.234 * 10^23`; de codering van het getal is geplitst in een *mantisse* (1.234) en een exponent (23) voor een macht tot de tiende vermenigvuldiging.

10.2 Tekens

Tekens worden normaal geproken voorgesteld door een string van zeven bits, gecodeerd in het zogenoemde ASCII (American Standard Code for Information Interchange). Op moderne computers, bestaat ieder van de 128 ASCII-tekens uit de laagste zeven bits van een 8-bit *octet*; octets zijn verpakt in geheugenwoorden, zodat (bijvoorbeeld) een string van zes tekens slechts twee geheugenwoorden in beslag kan nemen. Typ 'man 7 ascii' achter je Unix-prompt, voor een ASCII-code tabel.

De voorgaande paragraaf was op twee manieren misleidend. De minst belangrijke is dat de term 'octet' formeel correct is, maar in feite zelden wordt gebruikt; de meeste mensen refereren naar een octet als een *byte* en verwachten dat een byte acht bits groot is. Strikt genomen, is de term 'byte' algemener; er waren bijvoorbeeld 36-bit computers met 9-bit bytes (alhoewel die er waarschijnlijk nooit meer zullen zijn).

Het belangrijkste is dat niet iedereen in de wereld ASCII gebruikt. In feite kan niet iedereen in de wereld gebruik maken van ASCII. Hoewel het prima werkt voor Amerikaans-Engels, ontbreken er veel geaccentueerde letters en andere speciale tekens in die gebruikers van andere talen nodig hebben. Zelfs Britisch-Engels heeft er problemen mee dat er een pound-teken in ontbreekt om geldbedragen mee uit te drukken.

Er zijn verscheidene pogingen ondernomen iets aan dit probleem te doen. Allen maken gebruik van het extra hoge bit dat ASCII niet heeft, waarbij ASCII wordt gemaakt tot de lage helft van een 256-character set. Het meest gebruikte hiervan is de zogenoemde 'Latin 1' chararacter set (wat formeler met de naam ISO 8859-1). Dit is de standaard character set voor Linux, HTML, en X. Microsoft Windows maakt gebruik van een gewijzigde Latin-1. Hieraan zijn een boel tekens toegevoegd, zoals de rechter en linker dubbele aanhalingstekens op plaatsen van de originele Latin-1, die om historische redenen niet zijn ingevuld. Zie de *demoroniser* `<http://www.fourmilab.ch/webtools/demoroniser>` page voor de problemen die dit veroorzaakt).

Latin-1 omvat de meeste Europese talen, waaronder Engels, Frans, Duits, Spaans, Italiaans, Nederlands, Norweegs, Zweeds, Deens. Dit is echter ook nog niet voldoende, en als resultaat is er een hele serie Latin-2 tot Latin-9 character sets voor zaken zoals Grieks, Arabisch, Hebreeuws, Spaans en Serbo-Kroatisch. Zie de *ISO alphabet soup* `<http://www.utia.cas.cz/user_data/vs/documents/ISO-8859-X-charsets.html>` page voor details.

De laatste oplossing is een zeer grote standaard genaamd Unicode (en z'n identieke tweeling ISO/IEC 10646-1:1993). Unicode is identiek aan Latin-1 in z'n laagste 256 slots. Hierboven bevat het in 16-bit ruimte Grieks, Cyrillisch, Armeens, Hebreeuws, Arabisch, Devanagari, Bengaals, Gurmukhi, Gujarati, Oriya, Tamils, Telugu, Kannada, Malayalam, Thaais, Lao, Georgiaans, Tibetaans, Japanees-Kana, de volledige set met moderne Koreaanse Hangul, en een éénduidige set met Chinese/Japanese/Koreaanse (CJK) ideogrammen. Zie de *Unicode Home Page* `<http://www.uicode.org/>` voor details.

11 Hoe bewaart mijn computer zaken op disk?

Als je een harddisk onder Unix bekijkt, zie je een structuur met benoemde directory's en bestanden. Normaal gesproken zul je niet dieper hoeven te kijken, maar het zou handig zijn om te weten wat er zich afspeelt als er zich een diskcrash voordoet en je bestanden moet proberen te redden. Helaas is er geen goede manier om de diskorganisatie bekeken vanaf het bestandsniveau te beschrijven, dus ik zal het vanaf de kant van de hardware moeten beschrijven.

11.1 Low-level disk en bestandssysteemstructuur

Het oppervlaktegebied van je disk, waar de gegevens worden opgeslagen, is onderverdeeld op iets dat lijkt op een dartbord, – in circulaire sporen die weer als taartstukken zijn onderverdeeld in sectoren. Omdat sporen dichtbij de buitenrand uit een grotere oppervlakte bestaan dan die dichtbij de as in het midden van de disk, bevinden er zich op de buitenste sporen meer sectorstukken dan op de binnenste sporen. Iedere sector (of *diskblok*) heeft dezelfde grootte, die onder moderne Unixes in het algemeen 1 binary K (1024 8-bit woord) groot is. Ieder diskblok heeft een uniek adres of *diskbloknummer*.

Unix verdeelt de disk in *diskpartities*. Iedere partitie is een onafgebroken span blokken die gescheiden wordt gebruikt ten opzichte van enige andere partitie, als een bestandssysteem of als swap space. De oorspronkelijke redenen voor partities had te maken met het herstellen bij een crash in een wereld van veel langzamere en veel meer fouten bevattende disks; door de grenzen ertussen neemt vermoedelijk de kans af dat het deel van je disk ontoegankelijk of beschadigd raakt door een willekeurige slechte plek op de disk. Tegenwoordig is het veel belangrijker dat partities voor alleen lezen kunnen worden gedeclareerd (om te voorkomen dat een indringer kritieke systeembestanden kan wijzigen) of met diverse bedoelingen over een netwerk kunnen worden gedeeld wat we hier niet zullen bespreken. Met de laagst-genummerde partitie wordt vaak speciaal omgegaan, als een *boot partitie,* waar je een kernel kunt plaatsen om te worden geboot.

Iedere partitie bestaat óf uit *swap space* (wordt gebruikt om 9 (virtueel memory) aan te vullen óf een *bestandssysteem*, dat wordt gebruikt om bestanden vast te houden. Swap-space partities worden behandeld als een lineaire reeks blokken. Aan de andere kant hebben bestandssystemen een manier nodig om bestandsnamen naar reeksen diskblokken in te delen. Omdat bestanden groter worden, kleiner worden en gedurende de tijd wijzigen, zullen de gegevensblokken van een bestand niet bestaan uit een lineaire reeks maar het kan zijn dat ze over de partitie verspreid zijn (waar het besturingssysteem dan ook een vrij blok kan vinden als het er één nodig heeft).

11.2 Bestandsnamen en directory's

Binnen elk bestandssysteem, wordt de indeling van namen naar blokken verwerkt door een structuur die een *i-node* wordt genoemd. Er is een pool van deze zaken dichtbij de "onderkant" (laagst-genummerde blokken) van ieder bestandssysteem (de allerlaagste worden gebruikt voor huishoudelijke zaken en etiketteringsdoeleinden die hier niet zullen worden beschreven). Iedere i-node beschrijft een bestand. Datablokken van een bestand komen voor boven de inodes (in hoger genummerde blokken).

Iedere i-node bestaat uit een lijst met de diskbloknummers in het bestand dat het beschrijft. (In werkelijkheid is dit niet helemaal waar, slechts correct voor kleine bestanden, maar de rest van de details zijn hier niet van belang). Merk op dat de i-node *niet* de naam van het bestand bevat.

Namen van bestanden komen voor in *directorystructuren*. Een directorystructuur deelt namen in naar i-node nummers. Daarom kan een bestand in Unix meerdere echte namen hebben (of *hard links*); het zijn slechts meerdere directory-ingangen die naar dezelfde inode verwijzen.

11.3 Mount points

In het eenvoudigste geval, komt je volledige Unix bestandssysteem voor in slechts één diskpartitie. Ondanks dat je dit op een aantal kleine persoonlijke Unix systemen zal tegenkomen, is het niet gebruikelijk. Het is waarschijnlijker dat ze verspreid zijn over verscheidene diskpartities, mogelijk op verschillende fysieke disks. Dus het kan zijn dat er op je systeem bijvoorbeeld een kleine partitie voorkomt met de kernel, een wat grotere waar de OS utilites op voorkomen, en een veel grotere waar de home directory's op voorkomen.

De enige partitie waar je onmiddellijk na de systeemstart toegang toe zal hebben is je *rootpartitie*, welke (bijna altijd) degene is waarvan je bootte. Het bevat de root directory van het bestandssysteem, de top node waaraan al het andere hangt.

De andere partities in het systeem moeten aan deze root zijn gekoppeld opdat je volledige, uit meerdere-partities bestaande bestandssysteem toegankelijk is. Ongeveer halverwege het bootproces zal Unix deze niet-rootpartities toegankelijk maken. Het zal iedere partitie aan een directory op de rootpartitie *mounten*.

Als je bijvoorbeeld een Unix directory met de naam '/usr' hebt, is het waarschijnlijk een mount point naar een partitie die veel geïnstalleerde programma's op je Unix bevat maar niet is vereist gedurende de initiële boot.

11.4 Hoe een bestand wordt opgezocht

We kunnen nu van boven naar onder het bestandssysteem bekijken. Als je een bestand opent (zoals, laten we zeggen, `/home/esr/WWW/ldp/fundamentals.sgml`) gebeurt er het volgende:

Je kernel begint bij de root van je Unix bestandssysteem (in de root partitie). Het zoekt van daar uit naar een directory met de naam 'home'. Gewoonlijk is 'home' een mountpoint naar een grote gebruikerspartitie elders, dus zal het daar naartoe gaan. In de top-level directorystructuur van deze gebruikerspartitie, zal het naar een ingang met de naam 'esr' zoeken en een inode-nummer extraheren. Het zal naar die i-node gaan, opmerken dat het een directorystructuur is, en 'WWW' opzoeken. *Die* i-node extraherend, zal het naar de corresponderende subdirectory gaan en 'ldp' opzoeken. Dat zal het weer naar een andere directory-inode brengen. Die openend, zal het een i-node nummer voor 'fundamentals.sgml' vinden. Die inode is geen directory, maar bevat in plaats daarvan de lijst met diskblokken die met het bestand overeenkomen.

11.5 Eigenaarschap van een bestand, permissies en beveiliging

Om te voorkomen dat programma's per ongeluk of met opzet gegevens in de weg zitten, heeft Unix *permissies*. Deze werden oorspronkelijk ontworpen om timesharing te ondersteunen door meerdere gebruikers op dezelfde computer tegen elkaar te beschermen, in de tijd dat Unix hoofdzakelijk op dure gedeelde minicomputers werd gedraaid.

Je moet onze beschrijving van gebruikers en groepen in de sectie 5 (Wat gebeurt er als je inlogt?) even terughalen, om bestandspermissies te kunnen begrijpen. Ieder bestand heeft een gebruiker en een groep als eigenaar. Dit zijn in eerste instantie degenen die het bestand aanmaakten; ze kunnen met de programma's chown(1) en chgrp(1) worden gewijzigd.

De basis-permissies die met een bestand kunnen worden geassocieerd zijn 'read' (permissie de gegevens erin te lezen), 'write' (permissie het te wijzigen) en 'execute' (permissie het als een programma uit te voeren). Ieder bestand heeft drie sets met permissies; één voor de eigenaar ervan, één voor iedere gebruiker uit de groep ervan, en één voor alle anderen. De 'privileges' die je krijgt, als je inlogt, bestaan uit de mogelijkheden om die bestanden waarvan de permissie-bits overeenkomen met je gebruikers-ID of één van de groepen waartoe je behoort, te lezen, schrijven en uit te voeren.

Laten we eens een aantal listings van bestanden op een hypothetisch Unix-systeem bekijken om te bezien hoe deze op elkaar inwerken en hoe Unix ze toont. Hier is er één:

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

Dit is een gewoon gegevensbestand. De listing vertelt ons dat 'esr' de eigenaar ervan is en dat het werd aangemaakt met de groep 'users'. Waarschijnlijk plaatst de computer waar we ons nu op bevinden, alle gewone gebruikers standaard in deze groep; andere groepen die je vaak zal zien op timesharing computers zijn 'staff', 'admin', of 'wheel' (om vanzelfsprekende redenen zijn groepen op een single-user systeem of op

PC's niet zo belangrijk). Op je UNIX-systeem kan het een andere standaardgroep zijn, misschien één die is benoemd naar je gebruikers-ID.

De string '-rw-r–r–' stelt de permissie-bits voor van het bestand. Het allereerste streepje is de positie van de directory-bit; het zou een 'd' tonen als het bestand een directory was. De eerste posities daarna zijn de drie plaatsen met gebruikerspermissies, het tweede drietal de permissies van de groep en het derde drietal zijn de permissies voor alle anderen (worden vaak 'world' permissies genoemd). Bij dit bestand mag de eigenaar van het bestand 'esr' het bestand lezen en beschrijven, andere mensen in de groep 'users' mogen het lezen, en alle anderen in de wereld mogen het lezen. Dit is een typische set permissies voor een gewoon gegevensbestand.

Laten we nu eens kijken naar een bestand met totaal andere permissies. Dit bestand is de GCC, de GNU C-compiler.

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

Dit bestand behoort toe aan een gebruiker genaamd 'root' en een groep genaamd 'bin'; het kan alleen door root worden beschreven (gewijzigd), maar door iedereen worden gelezen of uitgevoerd. Dit is een typische eigenaarschap en set permissies voor een voorgeïnstalleerd systeemcommando. Op een aantal Unixes bestaat de 'bin' groep om systeemcommando's te groeperen (de naam is een historisch overblijfsel, een afkorting van 'binary'). Het kan zijn dat er onder jouw Unix in plaats daarvan een 'root' groep wordt gebruikt (niet geheel hetzelfde als de 'root' gebruiker!).

De 'root' gebruiker is de conventionele naam voor numeriek gebruiker ID 0, een speciaal, account met privileges waarmee alle andere privileges kunnen worden overheerst. Root-toegang is handig maar gevaarlijk; een typische fout, als je als root bent ingelogd, kan kritieke systeembestanden in de war schoppen wat met hetzelfde commando, uitgevoerd met een gewoon gebruikersaccount, niet kan.

Omdat het root-account zo krachtig is, is waakzaamheid bij de toegang ertoe geboden. Je root-wachtwoord is het enige meest kritieke stukje beveiligingsinformatie op je systeem, en dat is wat alle crackers en indringers, die ooit zullen komen, zullen proberen te verkrijgen.

(over wachtwoorden: Schrijf ze niet op – en kies geen wachtwoorden uit die makkelijk te raden zijn, zoals de voornaam van je vriendin/vriend/echtgenote. Dit is een verbazingwekkende algemene slechte gewoonte waarmee crackers worden geholpen...). In het algemeen, kies geen woord uit het woordenboek; er zijn programma's genaamd 'woordenboekkrakers' die naar waarschijnlijke wachtwoorden zoeken door woordenlijsten met algemene keuzes door te nemen. Een goede techniek bestaat uit een combinatie van een bestaand woord, een cijfer, een ander woord, zoals 'shark6cider' of 'jump3joy'; dat zal er voor zorgen dat de zoekruimte voor een woordenboekkraker te groot is. Maak echter geen gebruik van deze voorbeelden – crackers zouden daar vanuit kunnen gaan en ze in hun woordenboeken kunnen plaatsen.

Laten we nu eens een derde situatie bekijken:

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

Dit bestand is een directory (merk de 'd' op in het eerste permissie slot). We zien dat het alleen door esr kan worden beschreven, maar door alle anderen kan worden gelezen en uitgevoerd.

Leespermissie geeft je de mogelijkheid de directory weer te geven – dat wil zeggen, de namen van bestanden en directory's die erin staan te zien. Schrijfpermissie geeft je de mogelijkheid bestanden in de directory aan te maken en te verwijderen. Als je je een lijst namen van bestanden en subdirectory's in de directory herinnert, zijn deze regels begrijpelijk.

Execute-permissie op een directory betekent dat je via de directory bestanden en daaronderliggende directory's kunt openen. Als resultaat geeft het je permissie de inodes in de directory te benaderen. Een directory met de execute-bit volledig uitgeschakeld zou nutteloos zijn.

Zo nu en dan zal je een directory tegenkomen die voor iedereen leesbaar, maar niet voor iedereen uitvoerbaar is; dit betekent dat een willekeurige gebruiker alleen bij de daaronder liggende bestanden en directory's kan komen als het de exacte namen kent (de directory kan niet worden weergegeven).

Het is belangrijk eraan te denken dat lees, schrijf- of permissie om uit te voeren op een directory onafhankelijk staat van de bestandspermissies en daaronderliggende directory's. In het bijzonder betekent schrijftoegang op een directory dat je nieuwe bestanden erin aan kunt maken en bestaande bestanden kunt verwijderen, maar het geeft je niet automatisch schrijftoegang tot bestaande bestanden.

Laten we als laatste eens kijken naar de permissies van het login-programma.

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

Dit heeft de permissies zoals we ze zouden verwachten van een systeemcommando – behalve dan die 's', waar de execute-bit van de eigenaar zou moeten staan. Dit is de zichtbare manifestatie van een speciale permissie genaamd de 'set-user-id' of *setuid bit*.

De setuid bit is normaliter verbonden met programma's die gewone gebruikers root-privileges toe moeten kennen, maar dan op een gecontroleerde wijze. Als het op een uitvoerbaar programma is ingesteld, krijg je de privileges van de eigenaar van dat programmabestand terwijl het programma namens jou draait, of 't nu wel of niet met jou overeenkomt.

Net als het root-account, zijn setuid programma's handig maar gevaarlijk. Iedereen die een setuid programma met als eigenaar root, kan verwerpen of kan wijzigen, kan het gebruiken om een shell met root-privileges voort te brengen. Om deze reden, wordt op de meeste Unixes het setuid-bit uitgezet, als het bestand voor schrijven wordt geopend. Veel aanvallen op de Unix-beveiliging proberen bugs te exploiteren in setuid programma's om ze te kunnen verwerpen. Beveiligings bewuste systeembeheerders zijn daarom extra voorzichtig met deze programma's en installeren met tegenzin nieuwe programma's.

Er zijn een paar belangrijke details die we verdoezelde toen we hiervoor de permissies bespraken; namelijk hoe de eigenaar en de permissies van de groep worden toegekend, wanneer een bestand of directory voor 't eerst wordt aangemaakt. De groep is een probleem omdat gebruikers deel uit kunnen maken van meerdere groepen, maar één daarvan (aangegeven in het record van de gebruiker in `/etc/passwd`) is de *standaardgroep* van de gebruiker en hierin zullen normaal gesproken de bestanden staan die door de gebruiker zijn aangemaakt.

Het verhaal met initiïle permissie-bits is iets gecompliceerder. Een programma dat een bestand aanmaakt, zal er om te beginnen de permissies aan toekennen. Maar deze zullen door een variabele in de gebruikersomgeving met de naam *umask* worden gewijzigd. De umask geeft aan welke permissie-bits worden *uitgezet* als een bestand wordt aangemaakt; de meest algemene waarde, en op de meeste systemen de standaard, is ——-w- of 002, waarmee de world-write bit wordt uitgezet. Zie de documentatie van het umask commando in de manual-page van je shell voor details.

De initiële directorygroep is ook wat gecompliceerd. Op een aantal Unix-systemen krijgt een nieuwe directory de standaardgroep van de gebruiker die het aanmaakt (dit is de System V conventie); op andere systemen krijgt de directory als eigenaar de groep van de parent-directory waarin het is aangemaakt (dit is de BSD-conventie). Op een aantal moderne Unixen, waaronder Linux, kan die laatste worden geselecteerd door het instellen van de set-group-ID op de directory (chmod g+s).

Er is een nuttige discussie over bestandspermissies in het artikel van eric Goebelbecker *Take Command* `<http://www2.linuxjournal.com/cgi-bin/frames.pl/lj-issues/issue21/tc21.html>`.

11.6 Hoe het mis kan gaan

Eerder lieten we doorschemeren dat bestandssystemen kwetsbare zaken kunnen zijn. Nu weten we dat je je door iets wat een willekeurig lange reeks directory en i-node verwijzingen kan zijn, moet manoeuvreren. Veronderstel nu dat je harddisk een slechte plek ontwikkelt?

Als je geluk hebt, zullen er alleen maar wat bestandsgegevens beschadigd raken. Als je geen geluk hebt, is het mogelijk dat een directorystructuur of i-node nummer wordt verknoeid en een volledige subtree van je systeem in het niets terechtkomt – of nog erger, in een verknoeide structuur resulteert die op meerdere manieren naar hetzelfde diskblok of een inode verwijst. Dergelijke corruptie kan door normale bestandsbewerkingen worden verspreid, door gegevens te ruïneren, welke zich niet op de originele slechte plek bevonden.

Gelukkig, zijn dit soort onvoorziene omstandigheden heel ongewoon geworden, aangezien diskhardware betrouwbaarder is geworden. Nog steeds betekent het dat je Unix een periodieke integriteits-controle uit zal willen voeren op het bestandssysteem om er zeker van te zijn dat er niks aan scheelt. Moderne Unixes voeren op iedere partitie tijdens het opstarten vlak voor het mounten een snelle integriteits-controle uit. Iedere paar reboots doen ze een veel grondiger controle die een paar minuten langer in beslag neemt.

Als dit alles klinkt alsof Unix verschrikkelijk complex en vatbaar voor storingen is, kan het geruststellend zijn te weten dat deze opstartcontroles kenmerkend normale problemen afvangen en corrigeren *voordat* ze echt noodlottig worden. Andere besturingssystemen hebben deze faciliteiten niet, wat het booten een beetje versnelt, maar je veel serieuzer onder pressie zet als je probeert met de hand te herstellen. (en in de eerste plaats in de veronderstelling dat je een kopie van de Norton Utilities of iets dergelijks hebt...).

12 Hoe werken computertalen?

We hebben reeds besproken 6 (hoe programma's werken). Ieder programma moet uiteindelijk als een stroom bytes, die uit instructies in de *machinetaal* van je computer bestaan, worden uitgevoerd. Maar menselijke wezens kunnen niet zo erg goed met machinetaal overweg; dit is inmiddels zelfs onder hackers een zeldzame zwarte magie.

Bijna alle Unix code behalve een kleine hoeveelheid directe hardware-interface ondersteuning in de kernel zelf is tegenwoordig geschreven in een *hogere programmeertaal*. (Het 'hogere' in deze term is een historisch overblijfsel om het te onderscheiden van 'lagere' *assembleertalen*, wat in wezen doorzichtige verpakkingen om machinecode heen zijn.)

Er zijn verscheidene verschillende soorten hogere programmeertalen. Teneinde hierover te kunnen meepraten, is het handig om in gedachten te houden dat de *broncode* van een programma (de door de mens gemaakte, te wijzigen versie) een soort vertaalslag moet ondergaan naar machinecode om het door de machine uit te kunnen laten voeren.

12.1 Gecompileerde programmeertalen

De meest gebruikelijke soort programmeertaal is een *gecompileerde programmeertaal*. Gecompileerde talen worden omgezet naar uitvoerbare bestanden met uitvoerbare machinecode door een speciaal programma dat (logisch genoeg) een *compiler* wordt genoemd. Als het uitvoerbare bestand éénmaal is aangemaakt, kun je het direct opstarten zonder nog naar de sourcecode te kijken. (De meeste software wordt geleverd met gecompileerde uitvoerbare bestanden aangemaakt aan de hand van code die je niet ziet.)

Gecompileerde programmeertalen geven een uitstekende performance en hebben complete toegang tot het OS, maar zijn ook moeilijk om in te programmeren.

C, de programmeertaal waarin Unix zelf is geschreven, is hiervan verreweg de belangrijkste (met z'n variant

C++). FORTRAN is een andere gecompileerde programmeertaal, nog steeds in gebruik door technici en wetenschappers, maar jaren ouder en veel primitiever. In de Unix wereld zijn er in belangrijke mate geen andere gecompileerde programmeertalen in gebruik. Daarbuiten wordt COBOL op velerlei gebied gebruikt voor financiële en zakelijke software.

Er werden veel andere gecompileerde programmeertalen gebruikt, maar de meeste daarvan zijn uitgestorven of zijn strikte onderzoekshulpmiddelen. Als je een nieuwe Unix ontwikkelaar bent die een gecompileerde programmeertaal gebruikt, zal dit zeer waarschijnlijk C of C++ zijn.

12.2 Geïnterpreteerde talen

Een *geïnterpreteerde programmeertaal* is afhankelijk van een interpreter programma dat de broncode inleest en vertaalt in berekeningen en system calls. De bron moet iedere keer dat de code wordt uitgevoerd, opnieuw worden geïnterpreteerd (en de interpreter moet aanwezig zijn).

Geïnterpreteerde programmeertalen hebben de neiging langzamer te zijn dan gecompileerde talen en hebben vaak beperkte toegang tot het onderliggende besturingssysteem en de hardware. Aan de andere kant zijn ze gericht op gemakkelijker programmeren en vergevingsgezinder betreft codeerfouten dan gecompileerde talen.

Veel Unix utility's, inclusief de shell en bc(1) en sed(1) en awk(1), zijn doeltreffende kleine geïnterpreteerde talen. BASIC varianten zijn gewoonlijke geïnterpreteerde talen. Zo ook Tcl. Historisch gezien, is de belangrijkste geïnterpreteerde programmeertaal LISP geweest (een belangrijke verbetering ten opzichte van zijn meeste opvolgers). Tegenwoordig wordt Perl op velerlei gebied gebruikt en wint zo langzamerhand meer aan populariteit.

12.3 P-code talen

Sinds 1990 is een soort hybride programmeertaal die zowel van compilatie als interpretatie gebruik maakt, in toenemende mate belangrijk geworden. Bij P-code programmeertalen wordt net als bij gecompileerde programmeertalen de bron vertaald naar een compacte uitvoerbare bestandsvorm, hetgene wat je in feite uitvoert, maar die vorm is geen machinecode. In plaats daarvan is het *pseudocode* (of *p-code*), die gewoonlijk veel eenvoudiger maar krachtiger is dan echte machinetaal. Als je het programma draait, interpreteer je de p-code.

P-code kan bijna zo snel worden uitgevoerd als een gecompileerd uitvoerbaar bestand (p-code interpreters kunnen zeer eenvoudig, klein en snel zijn) Maar p-code programmeertalen kunnen de flexibiliteit en kracht van een goede interpreter behouden.

Belangrijke p-code talen zijn onder andere Python en Java.

13 Hoe werkt het Internet?

Om je te helpen begrijpen hoe het Internet werkt, zullen we datgene bekijken wanneer je een typische Internet operatie uitvoert – – een browser richtend op de voorpagina van dit document op z'n thuisbasis op het web bij het Linux Documentatie project. Dit document is

`http://metalab.unc.edu/LDP/HOWTO/Fundamentals.html`

wat betekent dat het voorkomt in het bestand LDP/HOWTO/Fundamentals.html onder de World Wide Web export-directory van de host metalab.unc.edu.

13.1 Namen en lokaties

Het eerste wat je browser moet doen is een netwerkverbinding tot stand brengen met de computer waarop het document voorkomt. Om dat te kunnen doen, moet het eerst de netwerklokatie van de *host* metalab.unc.edu traceren ('host' is een afkorting voor 'host machine' of 'netwerk host'; metalab.unc.edu is een typische *hostname*). De corresponderende lokatie is in werkelijkheid een nummer dat een *IP-adres* wordt genoemd (het 'IP' deel van deze term wordt later uitgelegd).

Om dit te doen, ondervraagt je browser een programma dat een *name-server* wordt genoemd. De name-server kan zich op je computer bevinden, maar het is waarschijnlijker dat het draait op een service machine waarmee de jouwe communiceert. Als je je bij een ISP aanmeldt, zal een deel van je setupprocedure bijna zeker inhouden dat je Internet software het IP-adres van een name-server op het netwerk van de ISP bekend wordt gemaakt.

De name-servers op verschillende computers praten met elkaar over alle informatie die nodig is om hostnamen te herleiden, uit te wisselen en up to date te houden (ze in te delen naar IP adressen). Het zou kunnen dat je name-server drie of vier verschillende sites in het netwerk ondervraagt tijdens het herleidingsproces van metalab.unc.edu, maar dit gebeurt gewoonlijk erg snel (in minder dan een seconde).

De name-server zal je browser laten weten dat het IP adres van Metalab 152.2.22.81 is; nu het dit weet, zal je computer in staat zijn om op directe wijze bits met metalab uit te wisselen.

13.2 Packets en routers

Wat de browser wil doen is een commando naar de Webserver op Metalab zenden dat er ongeveer zo uitziet:

```
GET /LDP/HOWTO/Fundamentals.html HTTP/1.0
```

Dit is wat er gebeurt. Het commando wordt veranderd tot een *pakket*, een blok bits zoals een telegram dat met drie belangrijke zaken wordt ingepakt; het *bronadres* (het IP-adres van je computer), het *bestemmingsadres* (152.2.22.81), en een *service-nummer* of *poortnummer* (80, in dit geval) dat aangeeft dat het een World Wide Web verzoek is.

Je computer seint het pakket over (via een modem verbinding naar je ISP, of lokale netwerk) totdat het bij een gespecialiseerde computer aankomt, die een *router* wordt genoemd. De router heeft een indeling van het Internet in zijn geheugen – niet altijd compleet, maar één die je netwerkomgeving volledig beschrijft en weet hoe het bij de routers in andere omgevingen op het Internet moet komen.

Je pakket kan verscheidene routers passeren op weg naar zijn bestemming. Routers zijn slim. Ze kijken hoelang het duurt voor andere routers beantwoorden dat ze een pakket hebben ontvangen. Ze gebruiken die informatie om het verkeer via snelle links te adresseren. Ze gebruiken het om op de hoogte te zijn als andere routers (of een kabel) van het netwerk zijn uitgevallen, en compenseren dit zo mogelijk door het zoeken naar een andere route.

Er is een stedelijke legende die zegt dat het Internet is ontworpen om een nucleaire oorlog te overleven. Dit is niet waar, maar het ontwerp van het Internet is extreem goed in het verkrijgen van betrouwbare performance buiten onconventionele hardware in een onzekere wereld. Dit is te danken aan het feit dat zijn intelligentie via duizende routers in plaats van een paar massieve schakelingen (zoals het telefoon-netwerk) wordt gedistribueerd. Dit betekent dat storingen goed kunnen worden gelokaliseerd en het netwerk ze kan omzeilen.

Zodra je pakket bij zijn bestemmingsmachine aankomt, gebruikt die machine het service-nummer om het pakket aan de webserver door te geven. De webserver weet waarnaar het de reply moet zenden door het bron IP-adres van het pakket te bekijken. Als de webserver dit document retourneert, zal het in een aantal pakketjes worden opgebroken. De grootte van de pakketjes zal overeenkomstig de transmissiemedia in het netwerk en de soort service variëren.

13.3 TCP en IP

Om te begrijpen hoe meerdere-pakket transmissies worden afgehandeld, is het nodig dat je weet dat het Internet in werkelijkheid twee protocollen gebruikt, de één bovenop de ander gestapeld.

Het lagere niveau, *IP* (Internet Protocol), weet hoe het individuele pakketjes vanaf een bron-adres naar een doel-adres moet krijgen (daarom worden ze IP-adressen genoemd). IP is echter niet betrouwbaar; als een pakketje verdwaalt of zoekraakt, kan het zijn dat de bron- en bestemmingsmachine dit nooit zullen weten. In netwerkjargon, is IP een *connectieloos* protocol; de zender stuurt gewoon een pakket naar de ontvanger en verwacht geen bevestiging.

Hoe dan ook, IP is snel en goedkoop. Soms is snel, goedkoop en onbetrouwbaar OK. Als je via een netwerk Doom of Quake speelt, stelt iedere kogel een IP-pakketje voor. Als een aantal daarvan verdwaald raken, is dat OK.

Het bovenste niveau, *TCP* (Transmission Control Protocol), geeft je betrouwbaarheid. Als twee computers een TCP connectie tot stand brengen (wat ze doen door gebruik te maken van IP), weet de ontvanger dat het bevestigingen van de pakketjes die het ziet naar de zender moet terugsturen. Als de zender geen bevestiging te zien krijgt binnen een bepaalde pauzeperiode voor een pakketje, zendt het dat pakketje opnieuw. Verder geeft de zender ieder TCP pakket een opeenvolgend nummer, welke de ontvanger kan gebruiken om je pakketjes weer samen te voegen voor het geval ze niet in de juiste volgorde verschijnen. (Dit kan gebeuren als netwerkkoppelingen gedurende een verbinding worden verbroken of hersteld).

TCP/IP pakketjes bevatten ook een controle-totaal om detectie te activeren van gegevens die door slechte links zijn beschadigd. Dus vanuit het gezichtspunt van iemand die TCP/IP en name-servers gebruikt, lijkt het een betrouwbare manier om stromen bytes tussen hostname/service-nummer paren door te geven. Mensen die zich bezig houden met het schrijven van netwerkprotocollen hoeven zich bijna nooit bezig te houden met het verpakken van informatie in pakketjes, het weer samenvoegen van pakketjes, foutencontrole, controleren van totalen, en herverzending dat op dat niveau plaatsvindt.

13.4 HTTP, een applicatieprotocol

Laten we nu naar ons voorbeeld teruggaan. Web browsers en servers spreken een *applicatieprotocol* dat bovenop TCP/IP wordt uitgevoerd, door het eenvoudigweg als een manier te gebruiken om reeksen bytes heen en weer door te geven. Dit protocol wordt *HTTP* (Hyper-Text Transfer Protocol) genoemd en we hebben reeds een commando ervan gezien – het GET zoals hierboven getoond.

Als het GET commando naar de webserver van metalab.unc.edu met service nummer 80 gaat, zal het naar een *server daemon* luisterend op poort 80, worden verzonden. De meeste Internetdiensten worden door serverdaemons uitgevoerd die niets anders doen dan poorten in de gaten houden, uitkijken naar en uitvoeren van inkomende commando's.

Als het ontwerp van het Internet over de gehele linie gelijk zou zijn, is het dat alle delen zo eenvoudig en mens-toegankelijk mogelijk zou moeten zijn. HTTP, en daaraan gerelativeerde protocollen (zoals het Simple Mail Transfer Protocol, *SMTP*, dat wordt gebruikt om elektronische mail tussen hosts te verplaatsen) maken in het algemeen gebruik van eenvoudige afdrukbare-tekst commando's die met een carriage-return/line feed eindigen.

Dit is marginaal inefficient; in een aantal omstandigheden zou je meer snelheid kunnen krijgen door gebruik te maken van een waterdicht-gecodeerd binair protocol. Maar ervaring heeft uitgewezen dat de voordelen van commando's die gemakkelijk door menselijke wezens zijn te beschrijven en begrijpen zwaarder wegen dan enig bijkomstig voordeel in efficiëntie dat je zou kunnen krijgen ten koste van het maken van lastige en onbevattelijke zaken.

Daarom is wat de serverdaemon je terugzendt via TCP/IP ook tekst. Het begin van de response zal er ongeveer zo uitzien (een paar van de headers zijn achtergehouden):

```
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
```

Deze headers zullen worden gevolgd door een lege regel en de tekst van de webpage (waarna de verbinding wordt verbroken). Je browser toont enkel die pagina. De headers geven aan hoe (in het bijzonder vertelt de Content-Type header het dat de geretourneerde gegevens in feite HTML zijn).