Inleiding tot GNU/Linux

Kris Luyten
kris.luyten@uhasselt.be

academiejaar 2005 - 2006

inleiding tot GNU/Linux

Kris Luyten

Limburgs Universitair Centrum
Expertisecentrum Digitale Media

De volledige versie van deze cursus is beschikbaar op http://research.edm.uhasselt.be/kris/courses/gnu-linux.

Inhoudsopgave

Lijst van Figuren

Lijst van Tabellen

Lijst van Code Listings

Hoofdstuk 1

Inleiding

> Given enough eyeballs, all bugs are shallow.
>
> *Eric S. Raymond: Linus's Law*

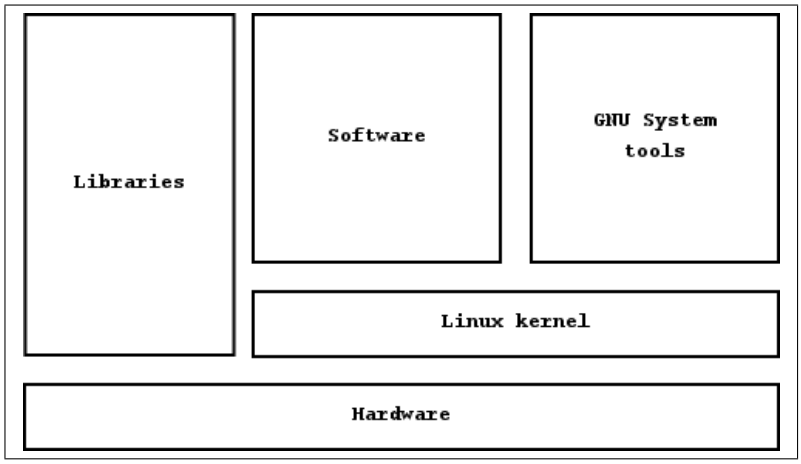
> Put on your pointy hat, grow a beard, drink Jolt Cola and come join in the fun.
>
> *Alan Cox*

Linux is een UNIX besturingssysteem, gestart door *Linus Torvalds* in 1991. Linus liet zich hierbij inspireren door het besturingssysteem Minix [Tan], speciaal ontwikkeld voor didactische doeleinden door Andrew S. Tanenbaum. Omdat Linux gebaseerd is op UNIX, zullen vele aspecten van het Unix besturingssysteem terug te vinden zijn in Linux. De meeste programma's die men kan gebruiken op Unix systemen hebben een Linux equivalent. UNIX is een besturingssysteem dat werd ontwikkeld in de jaren 70, en voortdurend verder ontwikkeld en uitgebreid is. Omdat UNIX het eerste besturingssysteem is dat grotendeels werd geschreven in een high-level programmeertaal, namelijk C, kon het werken op alle systemen, waarvoor een C compiler voorhanden was. Ook Linux is dus beschikbaar voor de meeste hardware.

UNIX is een multi-user, multitasking besturingssysteem; dit wil zeggen dat meerdere gebruikers op hetzelfde moment kunnen inloggen en dat elk van deze gebruikers meerdere programma's tegelijkertijd kunnen uitvoeren. Ook voor Linux zijn deze eigenschappen geïmplementeerd in het besturingssysteem.

Tegenwoordig bestaan er veel besturingssystemen die zijn afgeleid van UNIX. Er zijn de commerciële producten, zoals SunOS (nu gekend als Solaris), HP-UX, IBM AIX, en SGI Irix. En er zijn de UNIX versies waarvan de broncode verkrijgbaar is, en waarvan gebruikers die broncode mogen aanpassen aan hun eigen behoeften. Linux is het meest gekende voorbeeld, en is verkrijgbaar onder de zogenaamd GNU General Public License (GNU = GNU's not Unix)[1] Andere voorbeel-

[1] GNU General Public License, een software licentie voor vrij beschikbare herverdeelbare software,
zie http://www.gnu.org.



Figuur 1.1: Ruwe opbouw besturingssysteem Linux

den van zijn FreeBSD, OpenBSD, en NetBSD, die de BSD licentie hanteren; die licentie garandeert ook vrije beschikbaarheid en aanpasbaarheid van de broncode. Linux en de BSDs zijn eigenlijk slechts *kernels* van een besturingssysteem, en daar komt de gebruiker zelden rechtstreeks mee in aanraking. Naast die kernel levert het GNU project (en anderen) een verzameling van systeemprogramma's (de *GNU tools*), die bij alle Linux en BSD distributies terug te vinden zijn. Figuur 1.1 bevat een ruwe schets van de gehele structuur, samen met de GNU tools.

Het GNU project is gestart in 1985 door Richard Stallman, ook geïnspireerd door UNIX. Het project is begonnen met de ontwikkeling van de GNU tools, maar had nog geen kernel op het ogenblik dat Linux ten tonele verscheen. Linus Torvalds (en de ontwikkelaars van de BSD kernels) hebben van in het begin de nuttige complementariteit ingezien van hun eigen ontwikkelingen met die van het GNU project. En ook op vele commerciële UNIX systemen zijn de GNU tools beschikbaar. De GNU tools omvatten compilers, editors, shells, ... Verschillende van deze tools zullen later nog uitgebreid aan bod komen in deze cursus. Ondertussen is de GNU bibliotheek al uitgegroeid tot honderden toepassingsprogramma's.

De kernel en de GNU tools zijn alle tekstueel te gebruiken programma's, d.w.z., ze worden geactiveerd door bevelen in te tikken in een shell, of door het uitvoeren van shell scripts. Er bestaan echter ook volledige grafische gebruikersomgevingen voor Linux, waarvan KDE en GNOME de meest populaire zijn.

Het gebruik van Linux voor een praktische toelichting bij het vak Systeemprogrammatuur ligt voor de hand. Mede omdat Linux vrij beschikbare broncode voorziet is het van grote didactische waarde. We kunnen zien hoe de kernel van het besturingssysteem werkt, en hoe de aangeleerde technieken van besturingssystemen in de praktijk geïmplementeerd kunnen worden. Vooral in verband met het hoofdstuk over device drivers zal dit zeer handig blijken.

Al de hoger vermelde besturingssystemen gebruiken ongeveer dezelfde bevelen

als UNIX. In deze cursus worden commando's beschreven die overal moeten werken, maar sommige Unix besturingssystemen kunnen kleine afwijkingen vertonen. Om zeker te zijn over de syntax van het systeem waarop je werkt, kun je best de manual pages bekijken (syntax: man commando, bv.: `man chfn`. Je kan de man pagina verlaten door op de "q"-toets te drukken).

Hoofdstuk 2

Eerste kennismaking

2.1 Inloggen

De meeste grafische interfaces op Linux versies zijn gebaseerd op The X Window System[1]. Het is vaak zo'n grafische interface die je gebruikt om op de computer in te loggen. Ofwel moet er dan op de computer zelf of op een terminal (bij mainframes) worden ingelogd ofwel kan dit vanuit een pc gebeuren, indien de juiste software is geïnstalleerd om vanop afstand op andere UNIX computers te werken. De computer waarop Linux draait moet ook X Windows ondersteunen. Dit doet het door een zogenaamde X Server. De X Server is verantwoordelijk voor de grafische output. Een belangrijk voordeel van het gebruik van zo een X Server is dat de output ook naar andere schermen kan gestuurd worden, bijvoorbeeld naar een scherm in een naburige kamer[2] (of een naburig land!) dat niet rechtstreeks aan de schermuitgang van de computer zelf verbonden is. Meer uitleg over de beschikbare grafische omgevingen in GNU/Linux kan je vinden in hoofdstuk 5. Een andere manier om in te loggen kan via telnet. Dit kan op bijna ieder platform gebruikt worden om op een server in te loggen. De nadelen zijn dat telnet volledig karaktergeoriënteerd en zeer onveilig is. SSH (Secure Shell) is een veiliger alternatief. Indien mogelijk moet dit verkozen worden boven telnet. Eens je (via ssh of telnet) ingelogd bent, krijg je een prompt te zien zoals in figuur 2.1 en achter die prompt kan je bevelen typen. In hoofdstuk 4 vind je meer informatie over beschikbare bevelen.

```
kris@lumumba:~$
```

Figuur 2.1: De prompt

Voor dit opleidingsonderdeel wordt gebruik gemaakt van de computer `lumumba.luc.ac.be` of van een centrale server, die je toelaat een grafische omgeving op te starten. Om

[1] http://www.xfree.org,http://www.x.org

[2] Als er een netwerkverbinding is.

```
kris@lumumba:~$ passwd
Changing password for kris
Old password:
Enter the new password (minimum of 5, maximum of 127 characters)
Please use a combination of upper and lower case letters and numbers.
New password:
```

Figuur 2.2: Het passwd commando

te programmeren voor de Linux kernel zijn er vier andere PC's beschikbaar waarop je kan inloggen als root. Deze 4 machines zijn niet op het netwerk aangesloten.

Er zijn twee soorten gebruikers op een Linux systeem: de root en de gewone gebruikers. De root is de beheerder van het systeem. Deze heeft overal op het systeem toegang tot bestanden en kan deze verwijderen, aanpassen,... De gewone gebruiker kan enkel gegevens veranderen in de eigen home directory, alle directories daaronder en soms in enkele andere directories zoals de temp directory (/tmp). Zo is het de root die meestal de nieuwe software zal installeren, device drivers kan bijvoegen,... De speeltuin van de gewone gebruiker is beperkt tot zijn eigen home directory en wat daaronder hangt.

2.2 Gebruikersinformatie veranderen

2.2.1 Paswoord

Iedere gebruiker heeft een paswoord om zijn persoonlijke gebruikerstoegang te beveiligen. Om het paswoord te veranderen tik je `passwd` achter de prompt (zie figuur 2.2). De computer vraagt dan eerst om het oude paswoord, en daarna om het nieuwe. Dit laatste gebeurt tweemaal, zodat je niet per ongeluk een verkeerd paswoord kan intypen. Vergeet dit paswoord niet, anders kan je niet meer inloggen en moet je de systeembeheerder vragen om een nieuw paswoord.

2.2.2 Finger

Het systeem houdt informatie bij over iedere gebruiker. Andere gebruikers kunnen deze informatie opvragen via het zogenaamde finger protocol (let wel, deze service moet aanwezig zijn op het systeem alvorens men finger kan gebruiken). Om informatie over Kris Luyten op te vragen, typ je achter de prompt `finger kris@lumumba.luc.ac.be`. Hierbij is *kris* de login en *lumumba* de computer, waar hij zijn login heeft. Er verschijnt dan de volgende informatie op het scherm :

```
kris@lumumba:~$ finger kris
Login: kris                             Name: Kris Luyten
Directory: /home/kris                   Shell: /bin/bash
```

```
On since Tue Aug 13 15:23 (CEST) on pts/6 from edm-062.edm.luc.ac.be
Mail last read Tue Aug 13 15:19 2002 (CEST)
Project:
GNU/Linux World Domination!
Free all software!
Plan:
```

Hiervoor moet je wel de login naam kennen van de persoon die je zoekt. Om op een stuk van iemands echte naam te zoeken, kan je ook het `finger` commando gebruiken bv. `finger raymaekers@alpha.luc.ac.be`. Om deze mogelijkheid expliciet te vermijden gebruik je de -m switch. Op de meeste systemen is de echte naam van een gebruiker voornaam.achternaam. Om je eigen gebruikersinformatie aan te passen, gebruik je het `chfn` (change full name) bevel. Welke informatie precies kan worden opgegeven is afhankelijk van het systeem.

Op vele servers kan men, omwille van veiligheidsredenen, het `finger` bevel niet uitvoeren vanop een andere computer. Enkel indien je bent ingelogd op de server, kan je het `finger` bevel uitvoeren.

2.3 Informatie over bevelen

De volgende hoofdstukken voeren heel wat bevelen in om op een Linux systeem te gebruiken. Voor de meeste bevelen kan je een zogenaamde *man*-pagina opvragen. Om bijvoorbeeld de syntax en de betekenis van het `finger` bevel te kennen, typ je `man finger` in, dat je een overzicht geeft van onder andere de syntax van dat bevel, de betekenis ervan en de mogelijke opties.

Naast het `man` bevel om informatie over andere bevelen op te vragen, kan je ook `info` gebruiken om wat uitgebreidere informatie te verkrijgen. Tenslotte is er ook `apropos`, om van één of meer sleutelwoorden op te vragen welke bevelen hiermee verbonden zijn. Als je bijvoorbeeld een wiskundige functie wil plotten, maar je weet niet of er geschikte[3] programma's hiervoor op je computer staan, kan je het bevel `apropos` oproepen met als argument "plot" of "math" of beide. Als ik dit bevel op mijn systeem uitvoer krijg ik het volgende te zien:

```
[kris@localhost course]$ apropos plot
gnuplot              (1)  - an interactive plotting program
mathplot             (1)  - interactive (or batch) function grapher
mathplot [mathplot2ps] (1)  - interactive (or batch) function grapher
mathplot2ps          (1)  - interactive (or batch) function grapher
mathplot2ps [mathplot] (1)  - interactive (or batch) function grapher
pbmtoplot            (1)  - convert a portable bitmap into a Unix plot(5) file
.Vb 1  PDL::Graphics::PGPLOT [PDL::Graphics::PGPLOT] (3)  - PGPLOT enhanced interface
PDL::Graphics::PGPLOT::Window (3)  - A OO interface to PGPLOT windows
PDL::Graphics::PGPLOTOptions (3)  - Setting PGPLOT options
```

[3] `apropos` staat trouwens voor "apropriate search."

Opgave Probeer je eigen gebruikersinformatie op te vragen via `finger`. Verander deze informatie met `chfn` en vraag dan nog eens je eigen gebruikersinformatie op.

Hoofdstuk 3

Directories en bestanden

3.1 Directories

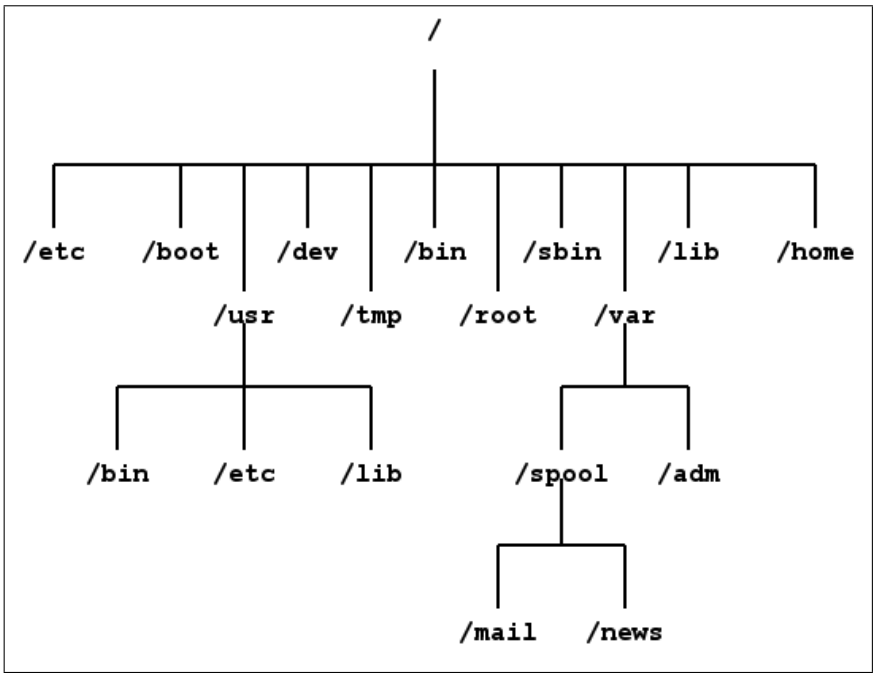
Bestanden in Linux zijn hiërarchisch geordend. Dit wil zeggen dat zij in een directory boom staan, die te vergelijken is met de directory structuur van DOS[1]. Er zijn een aantal afwijkingen: ten eerste maakt Linux het onderscheid tussen hoofdletters en kleine letters ("case-sensitive") in de namen van directories en bestanden: Sample.txt is niet hetzelfde als sample.txt. Verder is het niet het "\" teken dat directories van elkaar scheidt, maar "/". De subdirectory C van de directory src is dus src/C. UNIX (en dus ook Linux) systemen kennen ook geen drive aanduidingen zoals in DOS (zoals, bijvoorbeeld, A: en C:). Alle drives zijn toegankelijk via een subdirectory van de root (/) directory. Op de meeste systemen is de CD-ROM drive bijvoorbeeld toegankelijk via /cdrom/ of /mnt/cdrom/. Hoe dit praktisch gebeurt, vind je in sectie 3.6. De actieve directory vraag je op met het bevel `pwd` (pwd staat voor "print working directory").

3.2 De directory ordening onder Unix

Anders dan bij het Microsoft Windows besturingssysteem (waar er slechts enkele "verplichte" directories zijn zoals "Program Files", "Windows" en "Windows/System" bijvoorbeeld) is de directory structuur op een UNIX (en dus ook Linux) systeem voor een groot deel vooraf vastgelegd. Tabel 3.1 en tabel 3.2 geven een overzicht van de structuur van respectievelijk een UNIX systeem en een Linux systeem. In figuur 3.1 vind je een grafische voorstelling van de directory structuur, met als wortel van de boom de root directory.

Opgave Ter verduidelijking van de /proc directory (die je op Linux en een aantal andere UNIX systemen kan vinden) kan je de volgende commando's proberen:
`cat /proc/meminfo`

[1] Disk Operating System.



Figuur 3.1: Linux directory tree

/	de root directory
/etc	bevat data om het systeem op te starten. Deze directory bevat ook de meeste configuratiebestanden
/lib	bevat functie bibliotheken die gebruikt worden door de C compiler, evenals de gedeelde bibliotheken (zie sectie 10.3 voor meer uitleg over bibliotheken)
/tmp	bevat tijdelijke bestanden
/bin	bevat de meest noodzakelijke en gebruikte binaire en uitvoerbare bestanden
/usr	bevat al de rest
/usr/spool	bevat data die meestal in een queue staat, wachtend om verwerkt te worden, bijvoorbeeld printjobs
/usr/bin	bevat binaire en uitvoerbare bestanden zoals /bin
/usr/include	bevat de bron code die gebruikt wordt door `#include` statements
/usr/adm	bevat diagnostische informatie en de boekhouding van het systeem
/usr/lib	bevat diverse bibliotheken die door andere programma's aangeroepen worden

Tabel 3.1: Voorbeeld van een Unix directory structuur

/	de root directory
/etc	bevat data om het systeem op te starten. Deze directory bevat ook de meeste configuratiebestanden
/etc/passwd	bevat de gebruikers database (is *geen* directory, wel een bestand)
/etc/rc.d	bevat systeem-initialisatiescripts
/lib	bevat functiebibliotheken, die gebruikt worden door de C compiler, evenals de gedeelde bibliotheken (zie sectie 10.3 voor meer uitleg over bibliotheken)
/tmp	bevat tijdelijke bestanden
/var	bevat systeem definitie tabellen
/home	bevat de accounts van de gebruikers
/home/<*zet uw naam hier*>	dit is de user account van de gebruiker <*zet uw naam hier*>
/bin	bevat de meest noodzakelijke en gebruikte binaire en uitvoerbare bestanden
/sbin	bevat alle systeem-programma's
/usr	bevat al de rest die niet aanwezig is in de andere directories
/usr/bin	bevat binaire en uitvoerbare bestanden zoals /bin
/proc	bevat een pseudo-filesystem, dat dient als een interface naar kernel data structuren. Het wordt ondermeer gebruikt om proces informatie uit het geheugen te lezen.
/dev	bevat bestanden die de hardware voorstellen van het systeem

Tabel 3.2: Linux directory structuur

```
cat /proc/interrupts
cat /proc/partitions
cat /proc/devices
```

Door middel van het `cat` commando wordt de inhoud van de file naar het scherm weggeschreven.

3.3 Veranderen van directory

Het commando om van directory te veranderen is `cd` (change directory). Enkele voorbeelden zijn:

`cd ..`	Ga naar de onmiddellijk hogere directory in de directorytree ("parent directory");
`cd /`	Ga naar de hoogste directory (root);
`cd /home/craymaek`	Ga naar de home directory van de gebruiker craymaek;
`cd ~craymaek`	idem;
`cd ~`	Ga naar je eigen home directory;
`cd dir`	Ga naar de directory dir die onder de huidige directory staat;

Directory specificaties zijn niet steeds relatief ten opzichte van de huidige directory of absoluut (ten opzichte van de root directory).

Opgave Probeer zelf enkele directories te activeren. Je kan de huidige directory opvragen door het bevel `pwd`.

3.4 Directory aanmaken

Je kunt pas naar een directory gaan als deze directory bestaat. Om een directory te maken, moet je het commando `mkdir` (make directory) gebruiken met als parameter de directory die je wilt maken. Je kan alleen maar op de plaatsen waar je hiervoor toestemming hebt nieuwe directories aanmaken. Voor een gewone gebruiker is dat meestal alleen in de eigen home directory (`~/`) of de temp directory (`/tmp/`). Voor de beheerder van het systeem (root of *super user*) is dat nagenoeg overal.

Opgave Maak een directory oefeningen aan onder je home directory. Deze directory gaat nog verder worden gebruikt tijdens de oefeningen.

3.5 Directory-gegevens bekijken

Met het bevel `ls` (list) vraag je de inhoud van een directory op. Enkele voorbeelden zijn:

`ls`	Geef de inhoud van de huidige directory;
`ls oefeningen`	Geef de inhoud van de oefeningen directory;
`ls h*`	Geef de lijst van alle bestanden waarvan de naam met h begint;
`ls ?allo`	Geef de lijst van alle bestanden waarvan de naam met een willekeurig karakter begint en eindigt op "allo";

Vergeet niet dat ook hier directories relatief of absoluut kunnen zijn. Zoals je in de vorige voorbeelden kan zien is het ook mogelijk *joker*-tekens te gebruiken die door de shell ge-expand worden. Als je het teken `*` gebruikt, wil dit zeggen dat het met 0 of meer achtereenvolgende willekeurige tekens mag matchen. Het teken `?` wil zeggen dat er met 0 of 1 willekeurig teken gematcht mag worden. Zo matcht `ch*ter1*` bijvoorbeeld met chapter1, chaputer1, chter105689,... en `ch?ter1?` met chater12, chter1, choter1b,...

Het `ls` bevel heeft een reeks van parameters (switches). Enkele belangrijke bevelen zijn

`-a`	Geef alle bestanden weer, ook de verborgen "."bestanden;
`-l`	Geef de directory meer gedetailleerd weer;
`-R`	Ga ook alle subdirectories af;

Een bestand met een naam beginnend met een punt ".", is een speciaal bestand, meestal een configuratie-bestand of iets dergelijks. Een gewone listing toont deze bestanden niet. We moeten hiervoor expliciet opgeven dat we alle bestanden willen zien met de optie `-a`.

Deze switches kunnen ook worden gecombineerd tot bv. `ls -la`. Indien de directory te groot is, zodat je niet alles in één keer op het scherm kan zien, kun je | `more` achter het `ls`-bevel typen. Vb: `ls -la /home | more`. Dit geeft de directory listing pagina per pagina weer. Om naar de volgende pagina te gaan, moet je op de spatiebalk drukken. Bij sommige systemen moet je na de laatste pagina op "q" drukken (quit). Bijna alle bevelen die uitvoer op het scherm geven, kunnen met | `more` worden gecombineerd. Meer informatie over het "|" symbool vind je in sectie 3.7. Een alternatief voor `more` is `less`. Hiermee kan je zowel voorwaarts als achterwaarts scrollen, waardoor less een meer ingewikkeld bevel is dan more. Na iedere pagina moet een bevel worden gegeven om het vervolg te zien. De belangrijkste zijn *f* (forward, soms ook de PageDown toets), *b* (backward, soms ook de PageUp toets) en *q* (quit). Meer informatie over less staat in de man pages.

/dev/console	de monitor die aan de computer hangt
/dev/hd	de IDE randapparaten (harde schijven, cdrom), zo geeft /dev/hda2 de tweede partitie weer op de schijf hda
/dev/sd	de SCSI randapparaten
/dev/fd	de floppy drive
/dev/null	dit is de "vuilbak" van het systeem; alle data die men ernaar toe schrijft is voor altijd verloren. Indien het gebruikt wordt als invoerbestand wordt er een bestand gecreëerd van lengte 0

Tabel 3.3: belangrijke device bestanden

3.6 Randapparaten als bestanden

3.6.1 Hoe randapparaten voorgesteld worden?

Voordat je de directories kan gebruiken om de randapparaten aan te spreken, moet je eerst het geviseerde "device" *mounten*. Dit wil zeggen dat je een directory en het apparaat logisch moet verbinden. Dit kan je bijvoorbeeld doen als volgt: `mount -t iso9660 /dev/hdc /mnt/cdrom`. *iso9660* duidt op het soort bestandssysteem dat men mount (andere mogelijkheden zijn *vfat* voor fat bestandssystemen en *extf2* voor Linux extended filesystem 2 bestandsystemen, en nog een hele hoop andere). */dev/hdc* is een voorbeeld van een bestand dat de verbinding maakt met de eigenlijke hardware. Welk bestand voor welk randapparaat staat, is afhankelijk van de hardware configuratie van je computer. Het hoeft bijvoorbeeld niet `/dev/hdc` te zijn voor je cdrom. Op een andere PC had dat ook `/dev/hdb` kunnen zijn. Meestal kan je een floppy zelfs mounten zonder het type van bestandssysteem eraan mee te geven: `mount /dev/fd0 /mnt/floppy` bijvoorbeeld. `/dev/fd0` stelt een bestand voor dat een abstractie van de hardware is (meer bepaald de floppy disk) en we willen deze mounten op de directory `/mnt/floppy`. Let op: de directory `/mnt/floppy` moet bestaan om dit bevel uit te voeren.

In UNIX en Linux worden randapparaten en andere hardware dus als bestanden voorgesteld. Deze speciale bestanden vind je terug in de directory `/dev`. Om output naar deze randapparaten te sturen kan men dan simpelweg naar deze bestanden schrijven. Vb.: `cat /usr/share/sounds/generic.wav > /dev/dsp` stuurt de gegevens in het wave bestand naar de geluidskaart. We gebruiken hiervoor weer de redirectie operator >. Merk op dat het bestand waarnaar geschreven wordt in rechtstreeks contact staat met de eigenlijke hardware (via de kernel van het besturingssysteem om). Anderzijds kan men van deze bestanden ook informatie lezen. Zo geeft het bestand `/dev/mouse`[2] gegevens over de muis. Tabel 3.3 geeft een overzicht van de belangrijkste device bestanden.

[2]Dit is meestal een link naar een ander bestand!

3.6.2 Het mount bevel

We besteden nu wat meer aandacht aan de werking van het mount bevel [3]. De algemene syntax van dit bevel ziet er als volgt uit:

`mount` [opties] device mountpoint

device staat voor het apparaat of de partitie die je wil mounten. Mountpoint geeft aan waar je de data wil zien. Dat is een directory die al moet bestaan, en normaal gezien leeg is.

Enkele mogelijke opties zijn:

- **-a** mount alle filesystems in /etc/fstab (zie verder)
- **-t** Geeft het type filesystem aan (bvb vfat, iso9660, ext2, msdos,...)
- **-r** "read only": schrijven op het apparaat wordt volledig uitgeschakeld.
- **-o** gevolgd door verdere opties voor het filesystem. Bijvoorbeeld:
  - 'ro' voor read only (zelfde als -r)
  - 'noexec': zorgt ervoor dat programma's op het gemounte filesystem niet kunnen uitgevoerd worden. Dit kan nuttig zijn voor beveiliging.

Zie `man mount` voor de overige opties.

Enkele voorbeelden van het gebruik van het mount bevel:

```
mount -t vfat /dev/hda1 /mnt/windows
```

Dit bevel zorgt ervoor dat de inhoud van de eerste partitie op de eerste IDE-hd toegankelijk wordt in de directory /mnt/windows. Het veronderstelt dat deze partitie een FAT/FAT32 filesystem bevat.

```
~$ mount -t iso9660 -r /dev/hdc /mnt/cdrom
```

Dit zorgt ervoor dat je in /mnt/cdrom de inhoud van de CD in de CDROM drive aangesloten op secondary master ziet. Je kan ook met `mount -t iso9660 -o ro /dev/hdc /mnt/cdrom` hetzelfde effect bekomen.

```
mount -t vfat /dev/fd0 /floppy
```

Mount de DOS of MS Windows floppy die zich in de eerste floppydrive (de tweede drive is fd1) bevindt onder de directory /floppy. Je kan /dev/fd0 vergelijken met A: onder MS Windows.

Als je de `-t` optie weglaat, tracht mount automatisch het type van filesystem te bepalen. Een verdere hulp in het automatiseren van mount bevelen is de configuratiefile `/etc/fstab`.

[3] overgenomen uit [Kei02]

3.6.3 Het /etc/fstab bestand

Het configuratiebestand /etc/fstab bevat een aantal regels betreffende randapparaten die automatisch gemount kunnen worden. De regels die je terugvindt in fstab zien er als volgt uit:

```
device dir fstype opts df fs_passno
```

Het eerste veld is het te mounten apparaat of de partitie, bvb. /dev/hda1. Het tweede veld is het mountpoint (zoals bij het mount commando). Het derde veld is het type filesystem, bvb. iso9660, ext2 , msdos,... Het vierde veld bevat een aantal opties. Hierin moet de manier van mounten vermeld staan en eventueel die opties die via -o doorgegeven worden aan het mount programma. Hier moet ook de optie *'noauto'* vermeld worden: dit zorgt ervoor dat het filesystem in kwestie niet automatisch gemount wordt door *'mount -a'* bij de systeemstart. Het vijfde veld heeft betrekking tot backups en het dump programma. Het zesde veld bepaalt de volgorde waarin `fsck` (programma dat de integriteit van filesystems nagaat) de partities moet checken. Aan te raden is dit op 1 te zetten voor het / filesystem, 2 voor de andere filesystems.

Alle filesystems die vermeld zijn in /etc/fstab, behalve die met de noauto optie, worden bij het opstarten van het systeem automatisch gemount op de aangegeven mountpoints. Lijnen die beginnen met # worden als commentaar aanzien. Een paar voorbeelden zijn ook hier op hun plaats:

```
# /etc/fstab: static file system information.
#
# <file system> <mount point>   <type>  <options>                       <dump> <pass>
/dev/hda2       /               ext2    defaults,errors=remount-ro      0       1
/dev/hda5       none            swap    sw                              0       0
proc            /proc           proc    defaults                        0       0
/dev/hda1       /boot           ext2    rw                              0       2
/dev/hda6       /usr            ext2    rw                              0       2
/dev/hda7       /tmp            ext2    rw                              0       2
/dev/hda8       /var            ext2    rw                              0       2
/dev/hda4       /home           ext2    rw                              0       2
/dev/hdc1       /cdrom          iso9660 ro,user,noauto                  0       0
```

Deze computer bevat dus een aantal ext2 partities. Ook zie je hier het gebruik van een aantal opties. Het plaatsen van regels voor vaak gebruikte filesystems in /etc/fstab heeft het voordeel dat je bijvoorbeeld niet telkens `mount -t iso9660 /dev/hdb1 /mnt/cdrom` hoeft te typen om de CDROM te mounten, 'mount /cd' volstaat dan. Bij zo'n bevel gaat mount zelf kijken in /etc/fstab voor de nodige info.

3.7 Pipes

In het voorbeeld van het gebruik van het `more` bevel zagen we het symbool "|". Dit wordt een pipe (pijp) genoemd. In UNIX wordt het pipe symbool (|) gebruikt om twee of meerdere bevelen te combineren. Hierbij dient de uitvoer van een bevel als invoer voor het volgende bevel. De betekenis van dit symbool is makkelijk uit te leggen aan de hand van het voorbeeldje `ls -l /home | less`. Het bevel voor de pipe, namelijk `ls -l home` genereert veel uitvoer, meer dan in één keer op het scherm kan. Het bevel na de pipe, namelijk `less`, zorgt ervoor dat als men het invoer geeft langer dan het scherm dit deel per deel kan bekeken worden op het scherm. Nu moeten we een manier hebben om de uitvoer van `ls -l /home` als invoer aan `less` te geven. Hiervoor zorgt de pipe: "|". Het neemt de uitvoer van het bevel dat voor het symbool | staat en geeft het als invoer aan het bevel dat achter het symbool | staat.

Een ander voorbeeldje van het gebruik van pipes is `ls /home | sort | less`. In het eerste deel, voor de eerste pipe wordt een listing gegeven van al de bestanden en subdirectories in `/home`. Deze listing is de invoer voor het bevel `sort`. Dit bevel sorteert de lijnen invoer die het krijgt in alfabetische volgorde. Als uitvoer van het bevel `sort` kan je dus een gesorteerde lijst regels verwachten. Deze uitvoer wordt door de tweede pipe doorgegeven aan het laatste bevel, nl. `less`. Dit zorgt ervoor dat de gebruiker al de uitvoer kan doornemen op het scherm.

Het bevel `grep` wordt ook regelmatig gebruikt in combinatie met andere bevelen. `grep` drukt lijnen af die aan een meegegeven patroon voldoen. Hierbij zoekt `grep` in de bestanden die het aangeduid krijgt via de argumenten of de standaard invoer als er geen bestanden opgegeven zijn. Als we bijvoorbeeld alle zinnen willen zien uit het bestand `/etc/rc.d/rc.local` waarin het woord "echo" voorkomt, dan kan dit door het bevel: `grep echo /etc/rc.d/rc.local`. Als eerste argument geeft men het patroon mee waarnaar men zoekt en als tweede argument de plaats, waar grep dit moet gaan zoeken (zie hoofdstuk 6 voor meer uitleg). Nu kunnen we bijvoorbeeld ook via een pipe een lijst tekstlijnen doorgeven en grep gebruiken om de ongewenste lijnen weg te filteren. Zo kunnen we bijvoorbeeld alle bestanden die als bestandsnaam het patroon "oef" bevatten (in de huidige directory) opvragen door het bevel `ls | grep oef`.

Hoofdstuk 4

Enkele simpele operaties

4.1 Creëren van bestanden

Er is een simpel bevel om een *leeg* bestand te creëren in Linux, namelijk `touch`. `touch mijnbestand` maakt een leeg bestand aan met de naam `mijnbestand` indien dit bestand nog *niet* bestond. Anders verandert het gewoon de "modification time" van het bestand. Let op: een bestandsnaam mag geen karakters bevatten die een speciale betekenis hebben voor de shell. Concreet wil dit zeggen dat de volgende tekens niet mogen gebruikt worden in een bestandsnaam:

```
! @ # $ % ^ & * ( ) [ ] { } ' " \ / | ; < > `
```

Beschouw het volgende voorbeeld: Bij het finger bevel van paragraaf 2.2.2 kreeg je op de laatste regel "`No plan`" te zien. Op deze plaats kun je extra informatie geven, en hiervoor maak je een tekstbestand aan met de naam `.plan` (vergeet de "." niet, dit geeft aan dat dit een speciaal bestand is). De tekst in dit bestand zal afgebeeld worden op de plaats waar nu "No plan" staat. Je kan het bestand, weliswaar zonder enige inhoud, creëren door middel van `touch ~/.plan` De eenvoudigste manier om een bestand aan te maken met tekst erin is natuurlijk via een tekst editor. De meeste UNIX systemen hebben drie karaktergeoriënteerde tekst editors: *ed*, *vi* en *pico*. Hiervan is pico (onderdeel van het email en news programma pine) de eenvoudigste om te gebruiken, maar de minst krachtige. Een andere veel gebruikte editor is *emacs*. Onder software ontwikkelaars zijn emacs en vi (of vim, "vi improved") zeer populaire editors. Niet zozeer vanwege hun gebruiksvriendelijkheid (de leercurve mag wel steil genoemd worden) maar wel vanwege de efficiëntie die ermee kan bereikt worden. Een korte introductie tot vim kan je vinden in hoofdstuk 7.

Opgave Maak een `.plan` bestand aan door `pico .plan` te typen. Schijf een paar zinnen met behulp van de editor pico en sluit af met ctrl+x. Probeer nu je eigen gebruikersinformatie op te vragen met `finger`.

4.2 Bekijken van bestanden

Een tekstbestand kan met een teksteditor worden bekeken. Maar indien je enkel een bestand wil bekijken is een tekst editor hiervoor niet altijd het beste hulpmiddel. Belangrijke bestanden[1] kan je beter niet editeren als je niet precies weet wat je aan het doen bent. Je loopt dan immers het gevaar dat je niet meer kan inloggen, als je per ongeluk een aantal parameters een verkeerde waarde geeft. Het kan ook zijn dat je wel het recht hebt om een bestand te bekijken maar niet om dit bestand te wijzigen. Het `cat` bevel is handig om een bestand te bekijken. De syntax is `cat bestandsnaam`. Ook `more` kan gebruikt worden samen met `cat` door middel van een pipe. Hierbij kan `cat bestandsnaam | more` op de meeste systemen worden afgekort tot `more bestandsnaam`. Dit is omdat `more` ook op bestanden kan werken, die dan per pagina op de standaard uitvoer (het scherm) te zien zijn.

`cat` wordt vaak gecombineerd met `head` of `tail`. Hiermee kan je respectievelijk het begin en het einde van een bestand bekijken. Vb.:

`cat readme | head`	Toont de 10 eerste regels van het bestand readme;
`cat readme | head -n 5`	Toont de 5 eerste regels van het bestand readme;
`cat log | tail`	Toont de 10 laatste regels van het bestand log;
`cat log | tail -n 5`	Toont de 5 laatste regels van het bestand log;

`Cat` kan je ook gebruiken om twee of meerdere bestanden samen te voegen (concatenate), vb.: `cat bestand1 bestand2 > bestand3`. Dit wil zeggen dat de inhoud van `bestand1` en `bestand2` achter elkaar in `bestand3` moet worden geschreven. “>” is de redirectie operator en zegt dat de uitvoer niet naar de standaard-uitvoer (dit is meestal de console) gaat, maar naar het vermelde bestand (`bestand3` in dit geval).

Opgave Bekijk het bestand waarin jouw history staat (`.sh_history`), met behulp van het cat bevel. Indien je de *bash* gebruikt, moet je het bestand `.bash_history` gebruiken.

4.3 Copy, Move en Link

Een besturingssysteem is niet compleet als je bestanden niet kan copiëren of verplaatsen. Het bevel `cp` (copy) copiëert een bestand of directory. Om `bestand1` naar `bestand2` te copiëren moet je het volgende bevel geven: `cp bestand1 bestand2`.

[1] Onder “belangrijke bestanden” verstaan we configuratie bestanden zoals die bijvoorbeeld in de `/etc` directory voorkomen. Zonder een goede kennis van hun inhoud kan men deze beter niet zomaar veranderen.

De volledige inhoud van een directory kan worden gecopiëerd door gebruik te maken van het jokerteken "*". Het bevel `cp /etc/* ./` zal de volledige inhoud van de /etc directory naar de huidige directory copiëren (doe dit niet: de `/etc` directory is zeer uitgebreid!!!). Door de `-r` switch te gebruiken copiëer je een complete directory (inclusief de subdirectories), vb: `cp -r dir1 dir2`, dat `dir2` aanmaakt als copie van `dir1`. De switch `-r` geeft aan dat het bevel `cp` hier recursief moet werken.

Het verplaatsen van bestanden en directories is analoog aan het copiëren ervan. Hiervoor gebruik je het bevel `mv` (move). Met dit bevel hernoem je ook een bestand, vb.: `mv .pan .plan`.

In UNIX omgevingen kan men een bestand laten verwijzen naar een ander bestand. Dit bestand wordt een link genoemd. De meest gebruikte links zijn symbolische links. Als men een symbolische link wil editeren, wordt het oorspronkelijke bestand geëditeerd als dat bestaat. Het bevel om een symbolische link te maken is `ln -s bestand1 bestand2`.

Opgave

- copiëer het bestand `.plan` in je home directory naar een bestand genaamd `plan` in de oefeningen directory. Copiëer de hele inhoud van de oefeningen directory naar een directory genaamd backup.
- Maak in de oefeningen directory een symbolische link naar `.plan`, genaamd `planlink`. Bekijk de inhoud van de oefeningen directory met `ls -l`. Is er een verschil tussen de twee bestanden?
- Editeer nu het bestand planlink en bekijk je eigen gebruikersinformatie met `finger`. Is er nu iets veranderd?

4.4 Directories en bestanden verwijderen

Bestanden kan je verwijderen met het bevel `rm` (remove), waarvan de syntaxis `rm bestandsnamen` is. Ook hier kan je het jokerteken gebruiken. Om een (lege!) directory te verwijderen, gebruik je het bevel `rmdir` (remove directory). Het verwijderen van alle bestanden en subdirectories van een directory en het verwijderen van een directory zelf kan worden gecombineerd door de -r switch: `rm -r dirname`. Zoals bij het copy-bevel geeft de `-r` switch ook hier aan dat `rm` recursief te werk moet gaan. Als je er zeker van bent dat je de bestanden wilt verwijderen, kan je de optie `f` (force) gebruiken. Door `rm -f naam` in te typen zal het rm bevel dit bestand verwijderen zonder nog eens expliciet toestemming hiervoor te vragen aan de gebruiker.

Opgave Verwijder de backup directory.

4.5 Rechten op bestanden en directories

In UNIX omgevingen behoort iedere gebruiker tot een *groep*. Aan de hand hiervan kan de beveiliging van het systeem worden opgezet. Ieder bestand in een UNIX systeem heeft namelijk een owner (user), een group en een beveiligingsmodus van de volgende vorm `TUUUGGGOOO`. Dit kan men in 4 delen opsplitsen.

`T`	type van het bestand, vb.: directory (d), gewoon bestand (-);
`UUU`	owner read (r), write (w) en execute (x);
`GGG`	group read (r), write (w) en execute (x);
`OOO`	others read (r), write (w) en execute (x);

Het bestand `public_html` in een home directory kan de volgende beveiligingsmodus hebben: `drwxr-xr-x` Dit wil zeggen dat dit bestand een directory is (d), en dat de owner dit bestand mag lezen (ls), schrijven (bestanden creëren en verwijderen) en uitvoeren (cd). Anderen (de group en alle andere gebruikers van het systeem) mogen het bestand lezen en uitvoeren, maar niet schrijven. Alleen de owner van een bestand (of directory) en de root-gebruiker hebben het recht om deze informatie te veranderen. De owner van de directory bepaalt wie in de directory bestanden mag verwijderen of aanmaken.

Het belangrijkste bevel voor de beveiliging verandert de beveiligingsmodus zelf: `chmod` (change modus). Dit bevel kan op twee manieren worden gebruikt: `chmod [ugoa] {+,-,=} [rwx] bestandsnamen` of `chmod mode bestandsnamen`. De eerste manier zorgt ervoor dat voor de user (u), group (g), others (o) of iedereen (a, all) een bepaalde permissie (Read, Write, eXecute) wordt toegevoegd (+) of afgenomen (-). Ook kunnen de permissies aan een bepaalde permissie worden gelijkgesteld (=). Om bijvoorbeeld andere gebruikers van dezelfde group, waar je zelf inzit, permissie te geven om het `.plan` bestand te lezen moet `chmod g+r .plan` worden uitgevoerd.

De tweede manier bekijkt de permissie op een binaire wijze. De permissie `rw-r-- r--` kan binair worden gezien als 110 100 100. Octaal is dit 644. Om nu het .plan bestand deze permissie te geven, kan het volgende worden ingetypt: `chmod 644 .plan`.

Om de owner van een bestand of directory te veranderen, gebruik je het bevel `chown` (change owner). Dit bevel gebruik je alleen maar als je 100% zeker bent van wat je doet. De syntax is `chown user bestandsnamen`. Alleen de nieuwe eigenaar van het bestand of directory en de systeembeheerder (super user) kunnen je opnieuw de eigenaar maken van deze bestanden. Meestal kan enkel de superuser `chown` toepassen.

De group verander je met het bevel `chgrp` (change group): `chgrp group bestandsnamen`.

Opgave Zorg ervoor dat alle gebruikers de directory listing van de oefeningen directory kunnen vragen en dat de gebruikers van de groep ook naar deze directory kunnen gaan. De owner moet zijn of haar schrijfpermissies behouden.

4.6 Archiveren van bestanden

Wanneer je met grote projecten werkt is het handig om de bestanden van dit project te archiveren. Hierbij worden de bestanden samengevoegd tot één bestand. Het bevel dat dit doet is `tar` (tape archiver). Historisch gezien wordt `tar` gebruikt om back-ups te maken op tape, maar het resultaat van `tar` kan eender welk bestand zijn. De syntaxis is: `tar [key] [bestand...]`. `key` kan de volgende waarden hebben:

`r`	archiveert bestanden;
`x`	extract bestanden;
`t`	geeft een lijst van de bestanden in de archive;
`c`	creëert een nieuw archive (impliceert "r");
`v`	verbose: geeft op de console weer wat er gebeurt;
`f`	de volgende parameter is de naam van de archive;

De public_html directory kan gearchiveerd worden met het bevel `tar -cf pages.tar public_html`. De switch `c` ("create") geeft aan dat er een tar-bestand wordt gecreëerd en `f` ("file") geeft aan dat we dit naar een bestand willen doen. Omdat de switch `f` aangeeft dat we naar een bestand gaan archiveren is het eerste argument dat verwacht wordt de bestandsnaam voor het resultaat, hier is dat `pages.tar`. Daarna verwacht `tar` al de directories en bestandsnamen die gearchiveerd moeten worden, hier is dat enkel de directory `public_html`. De directorystructuur wordt op een relatieve manier ook mee gearchiveerd. Het gemaakte bestand is niet gecomprimeerd: de bestanden zijn gewoon in de archive geplaatst. Om een bestand te comprimeren kan je gzip gebruiken. Er zijn meerdere GNU compressie programma's beschikbaar, zo is bzip2 een goed alternatief. Hiermee kan elk bestand worden gecomprimeerd. Het bevel `gzip pages.tar` verwijdert het bestand `pages.tar` en creëert het gecomprimeerde `bestand.tar.gz`. Het bevel `gunzip` decomprimeert het bestand weer. Deze bevelen kunnen natuurlijk ook gecombineerd worden. Om de public_html directory te archiveren en te comprimeren kan je het volgende intypen: `tar -c ./public_html/ | gzip > pubhtml.tgz`[2]. Door het tar bevel worden al de bestanden in de directory samengezet in een enkel bestand. De uitvoer van het tar bevel wordt rechtstreeks als invoer van het gzip bevel doorgegeven via de pipe. Meer uitleg over het gebruik van een pipe (|) vind je in sectie 3.7. `gzip` wijst door > te gebruiken pubhtml.tgz als uitvoerbestand aan (dit wordt ook wel "redirectie naar een bestand" genoemd). Hierbij moet

[2] tar heeft ook een optie om de compressie met gzip dadelijk toe te passen, namelijk de switch `z`, bijvoorbeeld `tar -czf pubhtml.tgz ./public_html/`

nog opgemerkt worden dat gzip slechts op individuele bestanden werkt. Als men geprobeerd had `gzip ./public_html/` in te typen zouden alle bestanden in de directory in aparte bestanden met .gz als extensie gecomprimeerd worden. Om het bestand `pubhtml.tgz` nu terug uit te pakken kan men `tar -xvzf pubhtml.tgz`[3] intypen (waarbij x aangeeft dat het om extractie gaat, v aangeeft dat er informatie over de operatie moet gegeven worden[4], z aangeeft dat het bestand eerst gedecomprimeerd moet worden met gunzip en f aangeeft dat het om een bestand gaat).

4.7 Coderen van bestanden

Moderne e-mail programma's zoals Ximiam Evolution, Microsoft Outlook en de Mozilla mail-client hebben ondersteuning voor MIME[5]. Dit maakt het verzenden van binaire bestanden zeer gemakkelijk. E-mail programma's, zoals *pine* en *elm* die standaard op UNIX systemen staan, kunnen slechts e-mails aan die gebruik maken van de eerste 128 ASCII karakters. (De nieuwere versies kunnen echter ook overweg met MIME bijlagen!) Om via zo'n e-mail-programma een binair bestand te sturen of om een binair bestand te sturen naar iemand die alleen maar zo'n e-mail-programma heeft, moet het binaire bestand op een speciale manier worden gecodeerd. Het programma dat een binair bestand encodeert, heet `uuencode`. Het programma dat een gecodeerd bestand terug omzet naar een binair bestand heet `uudecode`. Om een bestand te encoderen typ je het volgende bevel: `uuencode pages.tar.gz pages.tar.gz`. De eerste paramater geeft aan wat de naam van het bestand is dat moet geëncodeerd worden. De tweede parameter geeft de naam aan die het bestand moet krijgen bij het decoderen. Het is vaak verstandig om twee maal dezelfde parameter te gebruiken. Er is nu nog één probleem: het resultaat wordt naar het scherm geschreven in plaats van naar een bestand. Dit kan worden opgelost door redirectie te gebruiken: : `uuencode pages.tar.gz pages.tar.gz > pages.uue`. Dit wil zeggen dat het resultaat niet naar het scherm moet worden geschreven, maar naar het bestand met naam pages.uue. UNIX systemen behandelen het scherm namelijk op dezelfde manier als een bestand. Vaak wordt `uue` als extensie genomen om aan te geven dat het bestand is geëncodeerd met uuencode. Het bestand kan je nu in een teksteditor inlezen. Het decoderen gebeurt met het bevel `uudecode pages.uue`. Het maakt niet uit als er in het bestand nog andere informatie staat, zodat een uuencoded-bestand in een e-mail kan worden gezet.

Opgave Maak wat lege en niet lege bestanden aan in de oefeningen directory. Maak een archive van de oefeningen directory en comprimeer het resultaat. En-

[3] We kunnen dit ook in 2 stappen doen: eerst `gunzip` gebruiken om te unzippen en daarna pas untarren.

[4] de optie *v* is beschikbaar voor de meeste bevelen om informatie te geven over wat het bevek aan het doen is

[5] Multipurpose Internet Media Extensions

codeer het gecomprimeerde bestand. Verwijder de oefeningen directory en het gecomprimeerde bestand. Voer nu uudecode, gunzip en tar uit. De oefeningen directory zou nu terug in zijn oorspronkelijke staat hersteld moeten zijn.

Hoofdstuk 5

X Window System

GNU/Linux kent ook een grafische interface gebaseerd op het X Window systeem. Meestal spreken we kortweg over "X".

De inhoud van dit hoofdstuk is gebaseerd op de uitstekende "X Window Overview" HOWTO [Man01b].

5.1 Algemene opbouw

X biedt een abstractie aan waarmee grafische programma's gebouwd kunnen worden.

X is gebaseerd op een *client-server architectuur*. Alle programma's fungeren als cliënten, ze communiceren met de server door er requests heen te sturen en door de ontvangen informatie te verwerken. De X server heeft de volledige controle over het scherm, het toetsenbord en de muis en handelt aanvragen van de cliëntapplicaties af. Zo moeten applicaties niet weten welke grafische kaart, scherm, toetsenbord of muis je hebt om hun ding te doen, maar moeten ze enkel met de server kunnen communiceren. Op die manier vraagt een cliënt aan de server om "een lijn van hier naar daar" te tekenen of om "deze tekst in dit font" op het scherm te zetten. Voorts kan een cliënt aan de server vragen om verwittigd te worden wanneer er op de muis geklikt wordt.

Het echte revolutionaire aan deze architectuur is dat de cliënten zich *niet* op dezelfde computer moeten bevinden als de server! Zo is het perfect mogelijk om een film te laten renderen op een speciaal daartoe ontworpen machine (bijvoorbeeld een Silicon Graphics computer) en het resultaat daarvan op je GNU/Linux computer te bekijken. Al het werk wordt op de speciale machine gedaan, jij krijgt alleen het resultaat te zien. Het is ook niet noodzakelijk dat de cliënten op dezelfde machine draaien!

De X specificatie definieert hoe cliënten met de server moeten communiceren (en omgekeerd), het eventuele netwerkprotocol dat gebruikt moet worden en welke functies er aan programmeurs beschikbaar moeten worden gesteld zodat cliëntapplicaties geschreven kunnen worden. Deze specificatie is het resultaat van

het Athena project aan het MIT[1] en zag het levenslicht in 1984. In 1988 werd de ontwikkeling en distributie van X in handen gegeven van het zogenaamde *X consortium* [xco] die de X specificatie gratis verspreid. Er bestaan verschillende implementaties van X. Degene die onder GNU/Linux het meeste gebruikt wordt is XFree86[xfr].

5.2 Window managers

Tot nog toe hebben we een server die zorgt voor de invoer en uitvoer, en cliënt applicaties die met de server kunnen communiceren.

Zoals iedereen die al eens met een Grafische User Interface (GUI) gewerkt heeft weet, moet het mogelijk zijn om de "cliënt" windows te verplaatsen, van grootte te veranderen, te maximaliseren of te minimaliseren, ... Het merkwaardige aan X is dat X deze taken *niet* afhandelt. Er is een speciaal programma dat daarvoor instaat: de window manager.

De window manager is een cliënt programma dat het privilege heeft om te mogen beslissen waar vensters geplaatst moeten worden, hoe de gebruiker de positie en grootte van deze vensters kan controleren. Bovendien staat een window manager in voor het "kader" van de vensters: de titelbalk, het frame, de verschillende knopjes (sluiten, minimaliseren, maximaliseren, ...). Sommige window managers geven ook de mogelijkheid om je bureaublad verschillende *desktops* in te delen.

Nogmaals, een Windows Manager is *geen* deel van X maar is een speciaal cliënt programma. X biedt enkel manieren aan waarop Window Managers hun werk kunnen doen.

Er bestaan dan ook verscheidene window managers die verschillen in de manier waarop je de interactie met de vensters uitvoert, de decoratie, ... Sommige zijn zeer simplistisch (Twm) terwijl andere enorm grafisch zijn en enorm veel features bezitten (Enlightenment). De meeste window managers schommelen hier ergens tussen (fvwm, amiwm, icewm, windowmaker, afterstep, sawfish, kwm, ...). Het voordeel aan het grote aanbod van window managers is dat zelf kan kiezen hoe je het prettigste werkt.

5.3 Cliënt applicaties en widget libraries

In tegenstelling tot Microsoft Windows biedt X geen standaard manier om dingen zoals knoppen, scrollbars, comboboxes, ...(ook wel *widgets* genoemd) voor te stellen. Je kan er alleen de basisdingen mee (lijnen, cirkels, rechthoeken, ... tekenen en opvullen). Dat heeft als gevolg dat het elke cliënt applicatie vrijstaat om te bepalen hoe hij zijn knoppen etc. wil tekenen en laten gedragen.

Omdat het een heel werk is om zelf widgets te schrijven zijn er in de loop van de tijd heel wat *widget libraries* ontwikkeld. Dit zijn codebibliotheken die pro-

[1] MIT staat voor "Massachusettes Institute of Technology", wat een universiteit in Amerika is.

grammeurs kunnen aanspreken en waarin de verschillende widgets al gedefinieerd zijn. Net zoals er verschillende window managers bestaan (elk met hun eigen look and feel) zo bestaan er ook verschillende widget libraries. Voorbeelden zijn Motif , GTK[gtk] en Qt[qt].

Deze verscheidenheid heeft tot gevolg dat applicaties die met behulp van verschillende widget libraries geschreven zijn er ook verschillend uitzien en zelfs verschillend bediend moeten worden. Spijtig genoeg gaat dat dan ook ten koste van een consistente manier van werken: als je drie verschillende applicaties tegelijk draait kan het perfect zijn dat ze er allemaal anders uitzien en dat je ze anders moet bedienen.

5.4 Desktop environments

De verscheidenheid aan keuze in het X Windows Systeem kan zeer zeker als een aanwinst bekeken worden. Je zit niet opgescheept met één enkele voorgedefinieerde Windows Manager die niet werkt zoals jij het wil (je kan immers altijd een andere kiezen). Bovendien kunnen verschillende applicaties er verschillend uitzien en mogen ze de interactie met de gebruiker regelen zoals het hun goeddunkt (waardoor je weer een grotere keuze hebt en de applicatie kan kiezen die het beste bij je past). Als nadeel hebben we al opgemerkt dat de consistentie verloren gaat en dat daardoor de complexiteit van het werken met deze programma's verhoogt.

Een *desktop environment* probeert dat nadeel te verhelpen door alle noodzakelijke programma's (bestandsbeheerder, webbrowser, tekstverwerker, taakbalk, ...) onder een consistente interface aan te bieden. Programma's die tot een bepaalde desktop environment behoren maken allemaal gebruik van dezelfde widget library. Bovendien biedt een desktop environment meestal ook een window manager aan.

Onder GNU/Linux zijn de K Desktop Environment (KDE) en GNU Network Object Model Environment (GNOME) de bekendste en ook meest volwassen desktop environments. Er zijn echter ook andere zoals GNUStep, ROX, GTK+XFce, UDE, ... Elke Desktop Environment heeft zo zijn eigen filosofie en welke jij het beste vindt is vooral een kwestie van smaak.

Gebruikers van Microsoft Windows zullen het hele concept van een desktop environment nogal vreemd vinden omdat Microsoft Windows slechts over één de desktop environment en window manager beschikt en omdat deze in Microsoft Windows ingebakken zitten. Er bestaat dan nogal de neiging om bijvoorbeeld KDE en GNOME als twee verschillende versies van X te zien. Dat is echter niet zo! Het is perfect mogelijk om KDE te draaien en programma's die bij GNOME horen op te starten en omgekeerd! Meer nog, het is zelfs mogelijk om meerdere X sessies tegelijk te gebruiken!

Hoofdstuk 6

Reguliere Expressies

6.1 Wat zijn reguliere expressies?

Een reguliere expressie is een patroon, en dit patroon beschrijft een taal. In dit specifieke geval beschrijft het patroon een verzameling van strings. Naar analogie met arithmetische expressies is het mogelijk kleinere reguliere expressies te combineren tot een grote reguliere expressie met behulp van operatoren. Het alfabet dat hier gebruikt wordt is de verzameling van letters en digits. Zo zal een lijst van karakters omringd door `[` en `]` elk karakter tussen deze haken matchen. Bijvoorbeeld: [**af05**] matcht de letters *a*, *f* en de digits *0* en *5*. We kunnen ook een bereik gebruiken, bijvoorbeeld [**d-i**] matcht met de letters *d*, *e*, *f*, *g*, *h* en *i*. De verzameling karakters tussen `[` en `]` wordt ook wel een *character class* genoemd.

In dit hoofdstuk laten we zien hoe je reguliere expressies kan gebruiken in vele GNU programma's. Als je bijvoorbeeld naar een bepaald patroon zoekt in een tekst, en alle regels wil weergeven waarin dat patroon voorkomt, dan moet je reguliere expressies gebruiken. Deze expressies gaan na of een bepaalde sequentie beantwoordt aan het opgegeven patroon: dit wordt ook wel *pattern matching* genoemd. We laten aan de hand van `grep` en `egrep`[1] zien hoe we pattern matching met behulp van reguliere expressie kunnen gebruiken. `grep` staat trouwens voor *G*eneral *R*egular *E*xpression *P*arser. Andere programma's die regelmatig gebruik maken van reguliere expressies zijn de editors `ed`, `sed`, `vim` en `emacs` (om zoekopdrachten in een tekst te doen), de scripting talen `awk` en `perl`, en het bevel `find`. Een kleine waarschuwing is op zijn plaats: we gebruiken hier enkel `grep` en `egrep` ter illustratie, en bij andere applicaties kan de syntax voor het specifieren van een reguliere expressie wel lichtjes verschillen. Raadpleeg de gebruikershandleiding en de man-pagina's voor de juiste syntax. Zie ook [Man99, Hek97] voor meer uitleg.

[1] extended *grep*

6.2 grep en egrep

`grep` en `egrep` nemen als invoer een bepaald bestand, en geven als uitvoer de regels uit dat bestand waarin een opgegeven patroon voorkomt. De syntax is als volgt:

```
grep [options] <PATROON> [BESTAND(EN)...]
```

`egrep` is hetzelfde als `grep -E`. De optie `-E` zorgt er voor dat `grep` het patroon interpreteert als een "extended" reguliere expressie. Dit zijn krachtigere reguliere expressies (zie later).

Ter illustratie, stel dat we alle lijnen die het woord "Linux" bevatten in de (LaTeX bestanden van de) hoofdstukken "Programmeren onder Linux" en "Een device driver in Linux" willen zoeken. We zouden dan het volgende bevel intypen: `grep 'Linux' shellprogr.tex devicedr.tex`. Het resultaat hiervan is:

```
devicedr.tex:\chapter{Een device driver in Linux}
devicedr.tex:over de Linux kernel is te vinden op \url{http://www.kernelnotes.org/}.
devicedr.tex:Al de devices waarvan je Linux systeem gebruik kan maken
staan opgesomd in de /dev directory, zoals reeds vermeld
devicedr.tex:Er is een duidelijk verschil in aanpak wanneer we voor de
kernel moeten programmeren.  Het besturingssysteem Linux kent
devicedr.tex:we er in Linux mee rekening houden dat lezen van en schrijven
naar een randapparaat dezelfde
linprogr.tex:\chapter {Programmeren onder Linux}
linprogr.tex:Op zowat elk Linux systeem is er een \href{http://gcc.gnu.org/}{GNU
C compiler} aanwezig,
linprogr.tex:omdat deze de meest gebruikte compilers zijn op Linux systemen.
linprogr.tex:%voor het Linux platform.  Daarnaast zijn er (net als op
het Microsoft platform)
```

Telkens wordt eerst de naam van het bestand afgebeeld, en daarna de regel waarin het woord "Linux" voorkomt. Men kan ook het lijnnummer mee laten afbeelden door de optie `-n` mee te geven aan `grep`. Raadpleeg de man-pagina's voor alle opties.

Opmerking: het is altijd veiliger om quotes rond het patroon te zetten, anders probeert de shell het patroon te expanderen.

6.3 De mogelijke reguliere expressies met grep

Naast de voorbeelden uit het begin van dit hoofdstuk biedt grep nog heel wat andere mogelijkheden om reguliere expressies te vormen. We zetten deze hier op een rijtje met telkens enkele voorbeelden. Als voorbeeld bestand gebruiken we de Makefile die je vindt in appendix B.

[ ]: de character class; matcht een karakter uit een verzameling van karakters tussen [en], zoals reeds getoond in de inleding van dit hoofdstuk (zie sectie 6.1). Binnen in [en] kan de verzameling karakters voorafgegaan worden door ^. Dit betekent dan dat deze karakters *niet* mogen gematcht worden. Zo zal **b[^A-Da-d]l** ervoor zorgen dat de letters a, b, c, d, A, B, C en D niet mogen voorkomen als letter tussen *b* en *l*. De volgende woorden zouden dus mogelijk wel matchen: *bol*, *bOl*, *b5l*, *bil* en *bwl*.

. : matcht eender welk karakter;

bevel	`grep '[iI]...[xX]' Makefile`

output	`BIB = bibtex # for generating the BibTeX entries`
	`INDEX = makeindex # for generating the index`
	`$(INDEX) $(FILE)`
	`$(INDEX) $(FILE)`

Dit voorbeeld matcht elk woord dat begin met *i* of *I*, gevolgd door *juist 3* karakters en afgesloten door *x* of *X*. Zo matcht **k.st** ondermeer de woorden *kast*, *kbst*, *kost*, *k3st*,...

***** : het voorgaande karakter zal nul, één of meer keer herhaald worden. Let op hierbij: als je * gebruikt in de shell zal dit ge-expand worden naar alle mogelijke opeenvolgingen van karakters. Bij grep is dit anders; zo matcht **hoo*fd** de woorden *hofd*, *hoofd*, *hooofd*, *hoooofd*, *hooooofd*,...

bevel	`grep 'hoo*l' Makefile`
output	`# de hoofdfile van de tekst` `# de hoofdregel van deze Makefile`

bevel	`grep 'ho*l' Makefile`
output	

bevel	`grep 'ho*l*' Makefile`

output	`BIB = bibtex # for generating the BibTeX entries`
	`INDEX = makeindex # for generating the index`
	`# de hoofdfile van de tekst`
	`# de hoofdregel van deze Makefile`
	`linprogr.tex shellprogr.tex softinstall.tex\`
	`app-callback.tex app-makefile.tex bibliotheken.tex\`
	`oploefshell.tex editors.tex regexpr.tex`
	`# ruim de boel op, alles behalve de benodigde sources`

Het verschil tussen beide voorgaande voorbeelden is dat het tweede ook lijnen matcht waarin *nul* keer een "*l*" voorkomt.

^ : matcht het begin van een lijn

bevel	`grep '^#' Makefile`

output	`# De compilers:`
	`# de hoofdfile van de tekst`
	`# de flags voor de DVI compiler en de output file`
	`# welke bestanden mogen weg gedaan worden`
	`# de hoofdregel van deze Makefile`
	`# ruim de boel op, alles behalve de benodigde sources`

Dit matcht dus alle regels die met '#' beginnen.

bevel	`grep '^[BP][DI]' Makefile`
output	`PDF = pdfelatex # for a pdf file` `BIB = bibtex # for generating the BibTeX entries`

Opmerking: let op het verschil met het gebruik van ^ binnen `[` en `]`, daar wordt het gebruikt als negatie.

$: matcht het einde van een lijn

bevel	`grep 'file$' Makefile`

output	`DVI = latex # for a DVI file`
	`PS = dvips # for a postscript file`
	`PDF = pdfelatex # for a pdf file`
	`# de flags voor de DVI compiler en de output file`
	`# de hoofdregel van deze Makefile`

Dit matcht alle regels die eindigen met het woord "file".

\ : zorgt ervoor dat we ook de gereserveerde karakters kunnen gebruiken zoals +, ?, |, (,) en $.

bevel	`grep '\$(DELETABLE)' Makefile`

output	`rm -rfv $(DELETABLE) #verwijder intermediate files`
	`rm -rfv $(DELETABLE) #verwijder intermediate files`
	`rm -rfv $(CLEANABLE) $(DELETABLE)`

\{ \} : het voorgaande karakter van \{*n*\} wordt juist *n* keer gematcht. Bijvoorbeeld: de expressie **0[123456789]\{2\}/[0123456789]\{6\}** matcht een telefoonnummer van de vorm *044/124578*.
Opmerking: deze metakarakters werken *niet* bij egrep.

Een veel voorkomend patroon om te matchen is de willekeurige string, ongeacht de lengte of welke karakters erin voorkomen. Dit patroon kan men matchen met de reguliere expressie **.***.

6.4 Enkele grep opties

Naast de uitgebreide mogelijkheden om reguliere expressies te specifieren bij grep, heeft grep ook nog heel wat opties. Hier volgt een opsomming van de meest gebruikte opties:

-i : maakt de zoekactie case insensitive.

-v : inverteer de zoektocht; laat enkel de regels zien die *niet* matchen met het patroon.

-n : toon de regelnummers bij de gevonden regels.

-c : toont enkel het *aantal* gevonden regels per bestand.

-f BESTAND : gebruik de patronen in BESTAND (één per regel) om de zoekactie uit te voeren.

-w : toon enkel de regels waarin een heel woord gematcht wordt.

6.5 Extended reguliere expressies met egrep

Alhoewel de opdeling tussen `grep` en `egrep` eerder artificeel is, willen we toch aanduiden dat er verschillen zijn tussen de mogelijke expressies die men kan vormen. Hier volgt een lijst van extra mogelijkheden die men met `egrep` kan gebruiken.

+ : het voorgaande karakter wordt één of meerdere keren herhaald. Zo zal **ho+fd** matchen met de woorden *hofd*, *hoofd*, *hooofd*, *hoooofd*,...

? : het voorgaande karakter wordt nul of één keer gematcht. Zo zal **pee?r** matchen met de woorden *per* en *peer*.

| : Geeft een aantal *mogelijkheden* op om mee te matchen. Bijvoorbeeld; de reguliere expressie **(haar|zijn)** geeft aan dat zowel de zin *is dit zijn boek?* als *is dit haar boek?* moeten matchen. De zin *dit is hun boek!* zal echter niet matchen.

( ) : men noemt de haakjes ook wel de "groepeer" operatoren. Het vorige voorbeeld gaf reeds aan hoe deze gebruikt worden: om verschillende mogelijkheden te groeperen. De reguliere expressie **b(a|o|e)l** matcht enkel met de woorden *bal*, *bol* en *bel*.

\< \> : zorgt ervoor dat enkel de karakters op het begin van een woord of het einde van een woord gematcht worden.

bevel	`grep 'dvi\>' Makefile`
output	`DVIFILE = $(FILE).dvi` `CLEANABLE = *.[^e]ps *.pdf *.tex~ *.dvi *.ps.gz`

bevel	`grep '\<make' Makefile`
output	`INDEX = makeindex # for generating the index` `app-callback.tex app-makefile.tex bibliotheken.tex\`

tekens	grep	egrep	awk	vi
.	√	√	√	√
[ ]	√	√	√	√
^	√	√	√	√
$	√	√	√	√
\{ \}	√			
?		√	√	
+		√	√	
|		√	√	
( )		√	√	

Tabel 6.1: Ondersteuning voor reguliere expressies syntax.

bevel	`grep '\<the\>' Makefile`
output	`BIB = bibtex # for generating the BibTeX entries` `INDEX = makeindex # for generating the index`

6.6 Overzicht van de metakarakters

Tabel 6.1 toont welk tool en/of scripting taal de in dit hoofdstuk getoonde metakarakters kan gebruiken. De syntax en semantiek blijven ongeveer dezelfde voor de verschillende gevallen. Raadpleeg bij twijfel de man pagina.

6.7 e-mailadressen ontleden en zoeken

Gegeven is een bestand (`grmail.txt`) met als inhoud een lijst met namen geassociëerd aan email-adressen (zie listing 6.1). Sommige namen worden geassocieerd met meerdere email-adressen, omdat ze meer dan één keer voorkomen in het bestand. We voeren enkele zoekacties uit in dit bestand, met behulp van `grep` en `egrep`.

Listing 6.1: `grmail.txt`; bestand met emailadressen

```
drs. Chris Raymaekers chris.raymaekers@luc.ac.be
Jori Liesenborgs      jori.liesenborgs@luc.ac.be
Jori Liesenborgs      jori@lumumba.luc.ac.be
drs. Tom Van Laerhoven tom.vanlaerhoven@luc.ac.be
drs. Tom Van Laerhoven tom@lumumba.luc.ac.be
prof. dr. Karin Coninx karin.coninx@luc.ac.be
prof. dr. Eddy Flerackers    eddy.flerackers@luc.ac.be
drs. Kris Luyten      kris.luyten@luc.ac.be
```

```
drs. Kris Luyten      kris@lumumba.luc.ac.be
Jo Segers      jo.segers@luc.ac.be
drs. Jan Van den Bergh jan.vandenbergh@luc.ac.be
student endroew eldritsj endroew.eldritsj@student.luc.ac.be
student sjef vanknutsel sjef.vanknutsel@student.luc.ac.be

student soja boon soja.boon@luk.ac.be
student Jozef Lumumba lumumba@congo.org
```

- Geef alle emailadressen in de lijst die een emailadres hebben op de server *lumumba*. Het is duidelijk dat `grep 'lumumba' grmail.txt` onvoldoende is. We krijgen dan als output:

  ```
  Jori Liesenborgs        jori@lumumba.luc.ac.be
  drs. Tom Van Laerhoven  tom@lumumba.luc.ac.be
  drs. Kris Luyten kris@lumumba.luc.ac.be
  student Jozef Lumumba lumumba@congo.org
  ```

  De laatste hit is geen email adres op de server lumumba. We kunnen grep ook laten weten dat lumumba moet voorafgegaan worden door een `@`!

  `grep '@lumumba' grmail.txt` geeft ons:

  ```
  Jori Liesenborgs        jori@lumumba.luc.ac.be
  drs. Tom Van Laerhoven  tom@lumumba.luc.ac.be
  drs. Kris Luyten kris@lumumba.luc.ac.be
  ```

  Dit is wat we zochten!

- Verdergaand op de vorige vraag: we willen zoeken welke mensen een emailadres hebben op de server lumumba en aan het doctoreren zijn (*drs.*). We kunnen dit op twee manieren doen:

  1. De output van de vorige oplossing pipen naar een nieuwe grep:

     ```
     grep '@lumumba' grmail.txt | grep 'drs\.'
     ```

  2. Het grep bevel uitvoeren met een wat moeilijkere expressie:

     ```
     grep 'drs\..*@lumumba' grmail.txt
     ```

     Let op de `.*`: deze betekent dat een willekeurige reeks karakters kan gematcht worden.

- Tel de lege lijnen in de file met behulp van grep. Een lege lijn is een lijn zonder inhoud: er komt niets voor tussen het begin van de lijn (^) en het einde van de lijn ($). Met behulp van het patroon `^$` kunnen we dus lege lijnen vinden. Het aantal lijnen dat je vindt met behulp van grep kan je laten zien door de optie `-c` te gebruiken.

  `grep -c "^$"grmail.txt` geeft:

  ```
  2
  ```

Hoofdstuk 7

Editors

7.1 Inleiding

Er zijn zeer veel verschillende editors beschikbaar voor Linux. De juiste keuze is dan ook meestal een kwestie van smaak. De meeste beginnende Linux gebruikers zullen `pico` gebruiken om tekstbestanden te editeren. Pico is een editor die bij de emailclient `pine` hoort.

Pico is echter verre van een krachtige editor. Hij kan enkel de basisbewerkingen doen. Dit hoofdstuk bespreekt twee van de krachtigste editors die er te vinden zijn: *vim*[1] en *emacs*[2].

7.2 Vi/Vim

Vi was de eerste editor voor UNIX die niet meer lijn per lijn werkte, zoals `ed`, maar wel pagina-geöriënteerd was. Dit feit en het gebrek aan alternatieven onder UNIX gedurende de jaren 70 en 80, maakten vi tot de "standaard" editor van UNIX. Dit wil zeggen: de enige editor waarvan iedere leverancier van programma's voor UNIX *altijd* mag verwachten dat hij aanwezig is.

Vim staat voor *Vi im*proved en is als vrije software project opgestart door de Nederlander Bram Molenaar. Vim is een uitbreiding van Vi, met meer flexibiliteit en meer opties. Eerst behandelen we de basis, namelijk Vi, waarna we verder ingaan op Vim. De tekst die je nu aan het lezen bent is trouwens geschreven met behulp van Vim!

Je kan vim opstarten en een bestand laten laden door het commando: `vim test.txt`. Dit zorgt ervoor dat het bestand "test.txt" geladen wordt, of indien het bestand nog niet bestond, dat het gecreëerd wordt. Je sluit vim af door `:q` in te drukken (let wel, je moet hiervoor in command mode zijn, zie sectie 7.2.1. Om ervoor te zorgen dat vim de aangebrachte wijzigingen vergeet, druk je `:q!` en om

[1] http://www.vim.org
[2] http://www.gnu.org/software/emacs/, http://www.xemacs.org

ze te bewaren, gebruik je `:wq`.

7.2.1 Verschillende modes

Beginnende Vi gebruikers hebben nogal eens last van de verschillende modes waarin Vi toetsaanslagen interpreteert:

Command mode Deze dient voor het uitvoeren van commando's, zoals het bestand opslaan, vi afsluiten of een tekst inplakken. Standaard zit je in de command mode bij opstarten. En de `[ESC]` toets brengt vi vanuit elke mode naar de command mode.

Insert mode (of *input mode*, of *editeer mode*). Je gaat van command mode naar insert mode door `i` in te drukken. In deze mode kan je tekst intypen.

Ex mode (of *last line mode*). Je gaat van de command mode naar ex mode via `:`. Dit doet op de onderste lijn van je scherm een dubbele punt verschijnen en alle toetsaanslagen verschijnen nu daar. (Behalve de `[ESC]`.) Na deze dubbele punt kan je bevelen van de vi's voorganger "*ex*" ingeven. (Door vim uitgebreid met vele nieuwe bevelen.)

Visual mode (enkel in vim beschikbaar). Een `v` brengt vim van de command mode in de visual mode, waarin je met de pijltjestoetsen of navigatie-bevelen gedeelten van de tekst kan selecteren.

Search mode Een `/` vanuit command mode brengt vi naar de laatste lijn van het editeerscherm, waar je nu een zoekterm kan intypen. De `[ENTER]` toets is het signaal om het zoeken te beginnen.

7.2.2 Enkele standaard bevelen

Tabel 7.1 geeft een lijst van vi bevelen voor de command mode. Sommige van deze bevelen brengen vi naar zijn insert mode. Vi kan werken met een heleboel "buffers" waarin je tekst editeert. Deze buffers zijn zowel gehele bestanden, als tijdelijke opslagplaatsen waarmee je stukken tekst kan "knippen en plakken." Naast de bevelen uit tabel 7.1 zijn er nog vele andere; raadpleeg hiervoor de documentatie, of gebruik de on-line documentatie, via `:help`.

Een belangrijk aspect van deze commando's is dat men ze kan combineren, door de verschillende "codes" simpelweg na elkaar te plaatsen vooraleer op [ENTER] te drukken. Door een getal x voor een commando te zetten kan je dit ook x keer laten herhalen. Enkele voorbeelden hiervan zijn:

d5$ verwijdert tot op het einde van de huidige lijn en verwijdert ook de volgende vier lijnen.

y10w copiëert de volgende 10 woorden na de cursor.

d10w cut de volgende 10 woorden (copiéert en verwijdert).

y$ copiëert tot het einde van de huidige lijn.

yG copiëert tot het einde van het huidige bestand.

7.2.3 Zoeken en vervangen in Vim

Vim heeft een uitgebreide ondersteuning voor het gebruik van *reguliere expressies* om zoekacties uit te voeren en stukken tekst te vervangen. De mogelijkheden en een korte praktische introductie tot reguliere expressies vind je in hoofdstuk 6. Tabel 6.1 uit hoofdstuk 6 geeft al een idee van wat je allemaal kan doen met reguliere expressies in Vim.

Om in Vim naar een patroon van karakters te zoeken ga je eerst naar *command mode*. Zorg ervoor dat je dus eerst [ESC] gedrukt hebt vooraleer de zoekopdracht uit te voeren. We geven aan dat we naar een patroon willen zoeken door eerst '/' in te typen. Daarna typen we het patroon in dat we zoeken. Dus `/modes[ENTER]` brengt ons naar het eerste voorkomen van het woord "modes" achter de cursor. Het *volgende* voorkomen vind je door op de toets "*n*" te drukken; "*N*" brengt je naar het *vorige* voorkomen.

Naast simpele woorden kan men ook meer complexe zoekacties uitvoeren met behulp van reguliere expressies. Bijvoorbeeld, de zoekopdracht `/s...t` vindt het volgende woord met de letters *s* en *t* en daartussen exact drie andere willekeurige letters.

7.3 Emacs

Emacs is een behoorlijk uitgebreide editor. Het is een tegenhanger van vim. Er zijn mensen die vim aanhangen en er zijn mensen die emacs aanhangen. Een typische opmerking van een vim-aanhanger: *"Emacs is a great operating system. It lacks a good editor, though."*. Het eerste deel van die opmerking is niet eens zo gek, want je kunt er e-mail mee versturen, nieuwsgroepen, websites en manual pages mee bekijken en tetris spelen!

Emacs start je op met het bevel `emacs`. Op deze manier start emacs op met een *buffer* met de naam `*scratch*`. Emacs laadt bestanden namelijk in buffers. Je kan ook één of meerdere bestanden die je wil editeren als argument meegeven. Zo zal `emacs mijntext.txt oef27.sh` de bestanden tt mijntext.txt en `oef27.sh` in twee buffers laden die geassocieerd worden met deze bestanden. Deze buffers kunnen dan naar het eigenlijke bestand op de schijf geschreven worden.

commando	betekenis
:q	sluit de huidige buffer af
:q!	afsluiten zonder de laatst gedane wijzigingen op te slaan
:w	het huidige bestand opslaan
:w *filename*	sla het bestand op met bestandsnaam *filename*
:qw	afsluiten en het bestand opslaan
:e *filename*	open het bestand *filename* in een aparte buffer
:next	ga naar de volgende buffer
:previous	ga naar de vorige buffer
:new abc	maak op het huidige editeerscherm plaats voor een nieuw buffer om het bestan "abc" te editeren
:split	laat meerdere buffers op het scherm zien
:help *commando*	Open een nieuwe buffer waar de help informatie voor *commando* wordt getoond.
a	begin met tekst in te voegen achter de huidige positie van de cursor
i	begin met tekst in te voegen voor de huidige positie van de cursor
A	begin met tekst in te voegen aan het einde van de huidige lijn
I	begin met tekst in te voegen aan het begin van de huidige lijn
o	begin met een nieuwe lijn tekst in te voegen onder de huidige lijn
O	begin met een nieuwe lijn tekst in te voegen boven de huidige lijn
dd	wis een lijn
d	delete commando, doet een "cut" operatie
cw	*change word*: verwijder het woord na de cursor, en vervang het door wat je vana nu intypt
C	vervang de rest van de lijn door wat je vanaf nu intypt
y	copiëer (*yank*) de geselecteerde tekst
p	plak (*paste*) de geselecteerde tekst *achter* de cursor
P	plak de geselecteerde tekst *voor* de cursor
j	ga één lijn naar onder
k	ga één lijn naar boven
h	ga één teken naar links
l	ga één teken naar rechts
$	ga naar het einde van de regel
^	ga naar het begin van de regel
G	ga naar het einde van de tekst
1G	ga naar het begin van de tekst
13G	ga naar lijn 13 in de tekst
ma	markeer deze lijn als "a"
'a	spring naar de lijn die voorheen als "a" gemarkeerd is
n	voer de vorige zoekopdracht opnieuw uit
.	voer de vorige opdracht opnieuw uit
w	ga naar het volgende woord
v	ga naar "visual mode", de gebruiker kan nu een stuk tekst selecteren

Tabel 7.1: Veel gebruikte bevelen in vim.

7.3.1 Toetsencombinaties

Het Vi concept van "modes" (zie sectie sectie 7.2.1) bestaat niet in Emacs. Deze editor vereist speciale toetstencombinaties om bevelen uit de niet-editeer modes van Vi te geven. Deze combinaties gebruiken de **control**-toets (`[Ctrl]`) en de **meta**-toets (meestal `[Alt]`).

7.3.2 Tekst bewerken

In het edit-venster kan op normale wijze tekst worden getypt. Aan het eind van de regel wordt overgesprongen naar de volgende regel. Aan het eind van de vorige regel wordt een teken geprint, zodat je duidelijk ziet dat er een automatische *enter* staat.
De volgende toetsen kunnen worden gebruikt in de edit-modus:

C-f:	Cursor een teken naar voren
C-b:	Cursor een teken naar achter
C-n:	Cursor naar de volgende regel
C-p:	Cursor naar de vorige regel

Navigeren door de tekst kan ook met de pijltjestoetsen. Tevens zijn de volgende toetsen aanwezig voor het scrollen:

C-v:	Een scherm omlaag
M-v:	Een scherm omhoog

Voor het wissen van de geschreven tekst kunnen de volgende toetsen gebruikt worden:

M-[backspace]:	Wis woord voor de cursor
M-d:	Wis woord achter de cursor

Daarnaast kan nog de `C-k` toets gebruikt worden voor het wissen van de tekst op een regel. Eenmaal `C-k` indrukken, wist de tekst, een tweede maal deze combinatie gebruiken wist ook het NewLine-karakter.

7.3.3 Meerdere windows

Als een groot tekstbestand bewerkt wordt, kan het handig zijn met twee vensters tegelijk te werken. Elk venster kan zich dan op een andere plaats in het document bevinden. Om het huidige venster te splitsen, wordt het commando `C-x 2` gebruikt. Een tweede venster verschijnt, met als inhoud het oorspronkelijke document. Beide vensters worden up-to-date gehouden tijdens het typen. Gebruik de toetsencombinatie `C-M-v` om in het niet-actieve venster te scrollen. Om tussen de twee windows te schakelen, gebruikt men `C-x o` Om het tweede window te sluiten en het actieve window open te laten, wordt de combinatie `C-x 1` gebruikt.

7.3.4 Zoeken in Emacs

Het kan nodig zijn in een document een bepaald woord of bepaalde karakter combinatie te zoeken. Dit kan bereikt worden met het `C-s` commando. `C-s` vraagt om invoer, door middel van de I-search (incremental search) prompt onderaan het scherm. Tijdens het intypen van de gezochte tekst, begint emacs al meteen met zoeken. Stel, er wordt gezocht naar het woord *'window'* in een document. Als achter de I-search-prompt de $'w'$ wordt ingetypt, springt de cursor direct naar de eerst voorkomende $'w'$ in de tekst. Zo verder met de rest van het woord. Is het woord gevonden, dan is het mogelijk het volgende woord te zoeken door nogmaals de combinatie `C-s` op te geven.

7.3.5 Bestanden openen en bewaren

Om in emacs een bestand te openen, wordt er gebruik gemaakt van het commando `C-x C-f [bestandsnaam]`. Geef dus eerst de combinaties `C-x` en `C-f`, en typ vervolgens de naam van het te openen bestand. Mocht het ingetypte bestand niet bestaan, dan wordt deze aangemaakt. Het opslaan van bestanden geschiedt met het commando `C-x C-s`. Onder in beeld verschijnt de mededeling *'Wrote [pad/bestandsnaam]'*. Mocht de gebruiker alvorens het bestand op te slaan een bevestiging willen geven, dan kan de toetsencombinatie `C-x s` gebruikt worden. Emacs stelt de vraag: *'Save to [pad/bestandsnaam]?'*. Na een bevestiging door middel van de $'y'$-toets wordt het bestand pas opgeslagen. Het is ook mogelijk het huidige bestand op te slaan onder een andere bestandsnaam en wel met het commando `C-x C-w`. Emacs vraagt na gebruik van bovenstaande toetsencombinatie om de nieuwe naam van het tekstbestand. Typ deze in en bevestig met <enter>.

7.3.6 Hulp opvragen

Emacs is voorzien van een uitgebreid hulp-systeem. Dit systeem is bereikbaar via de `C-h` combinatie. Een aantal `C-h` combinaties:

C-h k:	Hulp voor toetsencombinaties
C-h t:	Opent TUTORIAL bestand (een les emacs)
C-h ?:	Laat alle C-h combinaties zien

De tutorial (`C-h t`) is aan te raden. Deze behandelt de in deze sectie genoemde opdrachten en veel meer. De tutorial is in het Engels geschreven.

7.3.7 Overige nuttige toetsencombinaties

Een aantal toetsencombinaties is het kennen waard. Deze worden kort besproken.

C-x u Undo, maakt de laatst gebruikte toetsencombinatie ongedaan

C-g Cancel operation, Keert onmiddellijk terug naar het huidige document zonder de toetsencombinatie uit te voeren. Stel, de combinatie `C-h` is ingetypt, maar

de gebruiker wenst op het laatste moment toch geen helpscherm te zien, kan deze met de `C-g` combinatie ongedaan gemaakt worden.

7.3.8 Emacs afsluiten

Als de gebruiker klaar is met het bewerken of bekijken van tekst, kan emacs worden afgesloten. Dit wordt gedaan met de toetsencombinatie `C-x C-c`. Als er bestanden open staan die gewijzigd zijn, maar nog niet opgeslagen, vraagt emacs of dit alsnog moet. Emacs wordt hierna afgesloten en de gebruiker keert weer terug naar de command-prompt.

Hoofdstuk 8

Shells en shell scripting

8.1 Shell

Een shell is een programma dat zorgt voor de "command line interface" (CLI) tussen de gebruiker en het besturingssysteem (d.w.z., de kernel en de GNU tools). De shell interpreteert elk bevel van de gebruiker en voert het uit; d.w.z., het zet het bevel om in een reeks van "system calls" voor het besturingssysteem. UNIX heeft verschillende soorten shells. De belangrijkste (historisch gesproken) zijn: de Bourne shell, de Korn shell en de C shell.

De Bourne shell (sh) is de originele UNIX shell en is daarom beschikbaar in alle UNIX versies. Deze shell is zeer geschikt voor shell programmeren, maar is niet erg gebruiksvriendelijk.

De C shell (csh) is krachtiger dan de Bourne shell. Hoewel veel mensen deze shell niet zo goed vinden als de Bourne shell, wordt deze shell door veel C programmeurs gebruikt omdat de syntax sterk lijkt op de C syntax.

De Korn shell (ksh) combineert de voordelen van de Bourne shell en de C shell en is compatibel met de Bourne shell.

Modernere shells die op deze shells zijn gebaseerd, zijn o.a. de *Bourne again shell* (bash) en *tcsh*. Elke shell heeft een aantal initialisatie-bestanden. Dit wil zeggen, bestanden met shell-bevelen die altijd uitgevoerd worden wanneer de gebruiker een nieuwe shell start. Typisch zijn de namen van die initialisatie-bestanden gevormd uit de naam van de shell met daarachter "*rc*" (wat oorspronkelijk stond voor "resource configuration"). Zo heeft een typische Linux-installatie voor de bash shell de bestanden `/etc/bash.bashrc` (voor "system wide" initialisatie) en een verborgen bestand `.bashrc` in de home directory van elke gebruiker. Het system wide initialisatie-bestand wordt eerst uitgevoerd en daarna de `.bashrc` van de gebruiker.

Dit hoofdstuk is gebaseerd op sh, bash en ksh. Vergelijkbare technieken bestaan ook voor csh en tcsh, soms met kleine verschillen o.a. in de syntax.

8.2 Aliases

Gebruikers kunnen (reeksen van) bevelen die ze vaak gebruiken, afkorten met het `alias` bevel. (En opslaan in hun `.bashrc`.) Een voorbeeld is `alias ll="ls -l"`. Als dit bevel is uitgevoerd levert `ll` hetzelfde resultaat als `ls -l` voordien.

Om het aantal alias-definities in het bestand `.login` te tellen kunnen we het volgende bevel uitvoeren: `cat .login | grep alias | wc -l`. Het eerste bevel toont het bestand met omgevingsvariabelen (zie sectie 4.2). Op dit resultaat wordt dan het `grep` bevel toegepast. Dit bevel weerhoudt alleen de lijnen waarin het woord "alias" voorkomt. Tenslotte telt `wc` (word count) het aantal lijnen (dankzij de optie -l). Het resultaat van de hele regel is dus het aantal aliases die in .login zijn gedefiniëerd. Als je met de *Bourne Again Shell* werkt moet je het bestand `.bashrc` doorzoeken in plaats van `.login`.

8.3 Programmeren in de Shell

De shell geeft niet alleen bevelen door aan het besturingssysteem, maar kan ook *shell scripts* interpreteren en uitvoeren. Een shell script is een bestand dat een aantal shell bevelen bevat. Het bestand is uitvoerbaar gemaakt en zijn eerste regel geeft aan welke shell het bestand moet interpreteren. Deze eerste regel is eigenlijk een commentaar-lijn en begint met het commentaar-teken (#), gevolgd door een uitroepteken en het volledige pad van het shell-programma. Bijvoorbeeld: `#!/bin/sh` is de eerste lijn van alle voorbeelden en oefeningen van deze cursus en ze zegt dat de Bourne shell de interpreter is van het script.

Dit tekstbestand kan op twee manieren worden uitgevoerd:

1. Duid expliciet de interpreter van het bestand aan: `/bin/sh scriptname`.
2. Laat het bestand interpreteren door de huidige shell: `./scriptname`. Hiervoor moet het script eerst uitvoerbaar gemaakt worden met behulp van `chmod` (zie ook paragraaf 4.5).

8.4 Variabelen

Zoals de meeste talen kent ook shell scripting variabelen. En zoals in de meeste geïnterpreteerde talen hebben shell variabelen geen expliciet aangegeven type. Een nieuwe variabele wordt aangemaakt door een nieuwe naam te gebruiken. Daarom is het bij scripting zeer belangrijk om het script op fouten te controleren. Enkele voorbeelden zijn.

```
aantal=5
voornaam=Chris
achternaam=Raymaekers
```

Het is zeer belangrijk dat er voor en na = geen spaties worden geplaatst, anders weet de shell niet hoe het commando moet worden geïnterpreteerd. De waarde

van een variabele wordt opgevraagd door er een dollarteken ($) voor te plaatsen. Vb:

```
naam="$voornaam $achternaam"
```

Het `echo` bevel schrijft de waarde op het scherm. Ook hier gebruik je het dollarteken:

`echo naam`	resultaat: `naam`
`echo $naam`	resultaat: `Chris Raymaekers`

Anderzijds leest het `read` bevel een waarde in, bekijk het voorbeeld in listing 8.1.

Listing 8.1: het read bevel.

```
#!/bin/sh
eentekst="Shell programmeren is fijn"
echo $eentekst
echo "Typ nu zelf een regel in"
read eentekst
echo $eentekst
```

Dit voorbeeld laat zien hoe de variabele `eentekst` eerst een initiële waarde krijgt en hoe die in de volgende stap uitgeschreven wordt. Daarna wordt er gevraagd om zelf een regel in te typen. Die wordt ingelezen in de variabele `eentekst` en daarna uitgeschreven. Een ander voorbeeld vind je in listing 8.2

Listing 8.2: het read bevel., voorbeeld 2.

```
#!/bin/sh
echo "geef je voor- en achternaam, gevolgd door je universiteit,"
echo "je richting en je hobbies"
read voornaam achternaam universiteit richting hobbies
echo "voornaam: $voornaam"
echo "achternaam: $achternaam"
echo "universiteit: $universiteit"
echo "richting: $richting"
echo "hobbies: $hobbies"
```

Als je bij het programma voorgesteld in listing 8.2 de invoer "`Kris Luyten LUC informatica mensen irriteren met domme opmerkingen`" intypt, zal je de volgende uitvoer bekomen:

```
voornaam: Kris
achternaam: Luyten
```

```
universiteit: LUC
richting: informatica
hobbies: mensen irriteren met domme opmerkingen
```

De Linux shells kennen ook een reeks ingebouwde variabelen, waarvan de waarde kan worden opgevraagd.

`$#`	Het aantal argumenten dat aan het script is meegegeven.
`$?`	De exit waarde van het commando dat het laatste is uitgevoerd.
`$0`	De naam van het shell script (dit is impliciet het eerste argument).
`$*`	De positionele parameters $1, $2,... die de parameters bevatten. Vervang * door het gewenste parameter nummer.
`$@`	De volledige lijst van parameters (een "array" van alle argumenten).
`$$`	Het ID van het huidige proces.

Deze ingebouwde variabelen kunnen gebruikt worden in shell scripts, zoals getoond in listing 8.3.

Listing 8.3: Ingebouwde variabelen.

```
#!/bin/sh
echo the name for this script is $0 and it has $# arguments
echo the first argument of this script is $1
echo the fifth argument of this script is $5
```

Nu kunnen we desgewenst ook argumenten meegeven als we shell scripts gaan schrijven.

8.5 Speciale karakters

Elke shell heeft een reeks van speciale karakters. Deze paragraaf bespreekt er vier belangrijke: double quotes ("), single quotes ('), back slash (\) en back tick (`).

Double quotes worden gebruikt om een string die spaties bevat uit te schrijven of toe te kennen aan een variabele. Als de double quotes worden weggelaten, interpreteert de shell ieder woord als een nieuw bevel. Hierdoor zouden ernstige fouten kunnen ontstaan bij de uitvoering van het script. De waarden van variabelen worden nog steeds ingevuld.

Single quotes zijn sterker dan double quotes. Een string die tussen single quotes staat wordt letterlijk geïnterpreteerd. Indien slechts een gedeelte van een expressie letterlijk moet worden genomen, dan kan men best double quotes gebruiken in combinatie met de back slash. Dit teken zorgt er namelijk voor dat het volgend teken letterlijk wordt genomen. Vb.:

```
echo "Hello $LOGNAME"          resultaat: Hello craymaek
echo 'Hello $LOGNAME'          resultaat: Hello $LOGNAME
echo "Hello \$LOGNAME"         resultaat: Hello $LOGNAME
```

Met behulp van de back tick (‘) en het `expr` commando kan het resultaat van een bewerking aan een variabele worden toegekend. Dit wordt getoond in listing 8.4.

Listing 8.4: Toekenning aan variabelen.

```
een=1
twee=‘expr $een + 1‘
echo $twee
datum=‘date‘
echo $datum
```

De back tick wordt ook gebruikt om een string *uit te voeren*, d.w.z., de shell interpreteert de string als een shell bevel. Bijvoorbeeld, het bevel `info=‘ls‘` stelt de variabele info niet gelijk aan “ls”, maar aan de inhoud van de huidige directory.

8.6 Vergelijken van expressies

Het commando om een logische expressie te evalueren is `test`. De syntax is `test expressie`. Vaak worden vierkante haakjes gebruikt in plaats van het test commando: `[ expressie ]`. Het resultaat is in beide gevallen hetzelfde. Tabel 8.1, tabel 8.2, tabel 8.3 en tabel 8.4 geven de verschillende expressie weer. Enkele voorbeelden van expressies zijn:

- `[ 5 -eq 6 ]` geeft vals
- `[ -n lala ]` geeft waar
- `[ -d /dev ]` geeft waar
- `[ -f /home ]` geeft vals

Deze expressies komen pas helemaal tot hun recht bij conditionele expressies en condities voor het uitvoeren van herhalingslussen. Dit wordt besproken in de volgende secties.

8.7 Conditionele expressies

De Bourne shell kent twee conditionele expressies: *if* en *case*. De syntax van de if expressie is:

Operator	Betekenis
int1 -eq int2	int1 = int2 (equal)
int1 -ge int2	int1 $\geq$ int2 (greater than or equal)
int1 -gt int2	int > int2 (greater than)
int1 -le int2	int1 $\leq$ int2 (less than or equal)
int1 -lt int2	int1 < int2 (less than)
int1 -ne int2	int1 $\neq$ int2 (not equal)

Tabel 8.1: Integer expressies

Operator	Betekenis
str1 = str2	str1 = str2 (equal)
str1 != str2	str1 $\neq$ str2 (not equal)
str1	str1 is niet null
-n str1	de lengte van str1 is groter dan 0
-z str1	de lengte van str1 is 0

Tabel 8.2: String expressies

Operator	Betekenis
-d *bestandsnaam*	bestand (*bestandsnaam*)*is een directory* [a]
-f *bestandsnaam*	bestand is een gewone file
-r *bestandsnaam*	bestand kan worden gelezen door het proces
-s *bestandsnaam*	bestand heeft een lengte groter dan 0
-w *bestandsnaam*	bestand kan worden beschreven door het proces
-x *bestandsnaam*	bestand kan worden uitgevoerd door het proces
-e *bestandsnaam*	bestand bestaat

[a] in UNIX is een directory in weze ook een bestand

Tabel 8.3: Bestand expressies

Operator	Betekenis
! expr1	Negatie van expr1
expr1 -a expr2	expr1 en expr2
expr1 -o expr2	expr1 of expr2

Tabel 8.4: Logische expressies

Listing 8.5: Syntax van de if expressie.

```
if [ expressie ]
then
  statements
elif [ expressie ]
then
  statements
else
  statements
fi
```

Een voorbeeldje van het gebruik van een if-statement in een shell script vind je in listing 8.6.

Listing 8.6: Voorbeeld van een if expressie.

```
if [ $var = "Yes" ]
then
  echo "Value is Yes"
elif [ $var = "No" ]
then
  echo "Value is No"
else
  echo "Invalid value"
fi
```

De case expressie is vergelijkbaar met het switch bevel van C. De syntax is beschreven in listing 8.7.

Listing 8.7: Syntax van de case expressie.

```
case word in
  str1)
    statements;;
  str2 | str3)
    statements;;
  *)
    statements;;
esac
```

Een voorbeeldje van het gebruik van een case-expressie in een shell script vind je in listing 8.8.

Listing 8.8: Voorbeeld van een case expressie.

```
case $1 in
  01 | 1) echo "Month is January";;
```

```
  02 | 2) echo "Month is February";;
  03 | 3) echo "Month is March";;
  04 | 4) echo "Month is April";;
  05 | 5) echo "Month is May";;
  06 | 6) echo "Month is June";;
  07 | 7) echo "Month is July";;
  08 | 8) echo "Month is August";
  09 | 9) echo "Month is September";;
  10) echo "Month is October";;
  11) echo "Month is November";;
  12) echo "Month is December";;
  *) echo "Invalid Parameter";;
esac
```

8.8 Iteraties

Een iteratie in een shell script kan op drie verschillende manieren worden geïmplementeerd: for, while en until.

8.8.1 for

De syntax van de for expressie is:

Listing 8.9: Syntax van de for expressie.

```
for var in list
do
  statements
done
```

Een voorbeeldje van het gebruik van een for expressie in een shell script vind je in listing 8.10. Het script in listing 8.10 schrijft de huidige directory uit.

Listing 8.10: Voorbeeld van een for expressie.

```
#!/bin/sh
dirlist=‘ls‘
for file in $dirlist
do
  echo $file
done
```

Stel dat je in een directory heel wat tar.gz files staan hebt die je allemaal wil decomprimeren en uitpakken, dan kan je daarvoor een soortgelijk script schrijven. Een voorbeeldje hiervan zie je in listing 8.11.

Listing 8.11: untarren en gunzippen m.b.v. een for lus

```
#!/bin/sh
for file in ‘ls *tar.gz‘
do
  tar -xvzf $file
done
```

Merk op dat ‘`ls *tar.gz`‘ een verzameling bestanden teruggeeft waarover geïtereerd wordt. Je kan in plaats van het ls bevel uit te voeren, het te expanden patroon meegeven: `for file in *tar.gz`. De shell zal dit dan eerst expanden tot all tar.gz files in de huidige directory. In feite hoeft dit helemaal niet in een bestand

gezet te worden. We zouden ook het script kunnen intypen achter de shell prompt, waarbij we de enters vervangen door `;`. Zo zou je in plaats van listing 8.11 in een bestand te plaatsen ook kunnen intypen:

```
for file in ‘ls *tar.gz‘; do tar -xvzf $file ; done
```

Dit heeft hetzelfde effect.

8.8.2 while

De syntax van de while expressie is:

Listing 8.12: Syntax van de while expressie.

```
while expressie
do
  statements
done
```

Een voorbeeldje van het gebruik van een while expressie in een shell script vind je in listing 8.13.

Listing 8.13: Voorbeeld van een while expressie.

```
#!/bin/sh
i=1
while [ $i -le 10 ]
do
  echo $i
  i=‘expr $i + 1‘
done
```

Het script in listing 8.13 telt tot 10 en schrijft de getallen uit.

8.8.3 until

Tenslotte is er nog de until expressie waarvan de syntax beschreven wordt in listing 8.14. De until expressie voert de code tussen `do` en `done` uit tot de conditie die achter `until` staat voldaan is. Als deze conditie voldaan is voordat de lus voor de eerste keer uitgevoerd is, wordt de lus gewoon niet uitgevoerd.

Listing 8.14: Syntax van de until expressie.

```
until expressie
do
  statements
done
```

Een voorbeeldje van het gebruik van een until expressie in een shell script vind je in listing 8.15.

Listing 8.15: Voorbeeld van een until expressie.

```
#!/bin/sh
i=1
until [ $i -gt 10 ]
do
  echo $i
  i=‘expr $i + 1‘
done
```

Ook het voorbeeld uit listing 8.15 telt tot 10 en schrijft de getallen uit.

8.9 Functies

Scripts voor de Bourne shell kunnen ook functies bevatten. De syntax is voorgesteld in listing 8.16.

Listing 8.16: Syntax voor script functies.

```
functienaam () {
  statements
}
```

De functie kan worden opgeroepen met de functienaam. Ook kunnen er parameters worden doorgegeven: `functienaam param1 param2 ...`. Deze parameters kunnen dan in de functie worden aangeroepen door middel van de positionele parameters ($1, $2, ...). Een voorbeeldje van een functie die argument1 argument2-keer met zichzelf optelt (zoals een gewone vermenigvuldiging dus) vind je in listing 8.17. Merk op dat de functie voor de aanroep moet gedefinieerd worden.

Listing 8.17: Voorbeeld van een script functie.

```
#!/bin/sh
```

```
telkeerop(){
 result=$1
 teller=$2
 until [ $teller -eq 1 ]
 do
   result=‘expr $result + $1‘
   teller=‘expr $teller - 1‘
 done
 echo $result
}

telkeerop 5 7
```

8.10 Scripts onderbreken

Een iteratie (for, while en until) kan worden onderbroken door middel van het bevel `break`. Het hele script kan beëindigd worden met het bevel `exit`.

Een laatste bevel is shift. Hiermee worden de positionele parameters één plaats opgeschoven. Een voorbeeld hiervan vind je in listing 8.18. Dit voorbeeld schrijft alle parameters uit. De parameters kunnen ook meerdere plaatsen (n) plaatsen worden opgeschoven door `shift n`.

Listing 8.18: Voorbeeld van een shift opdracht.

```
#!/bin/sh
while [ $# -ne 0 ]
do
  echo $1
  shift
done
```

8.11 eval

Bekijk het stukje code in listing 8.19. We zouden verwachten dat de uitvoer van dit stukje code “`blaaiwaai`” is, maar we krijgen “`$bla`” als output te zien.

Listing 8.19: Zonder gebruik van eval.

```
#!/bin/bash
bla=blaaiwaai
wa=bla
z='$'$wa
echo $z
```

De reden hiervoor is eenvoudig. De expressie `z='$'$wa` kent aan z niet de inhoud van de variabele `$wa` toe, maar wel de string “`$bla`”. `$wa` evalueert namelijk als de string “bla”, en wordt aan het karakter ’$’ vastgehangen.

We hebben dus iets nodig dat de *waarde van de waarde van een variabele* teruggeeft: `eval`. Dit stelt ons in staat om zeer krachtige, dynamische programma's te bouwen. Alleen wordt een script door het gebruik van `eval` niet alleen krachtiger maar ook complexer en dus ook moeilijker leesbaar en moeilijker te debuggen. Listing 8.20 Laat zien hoe `eval` werkt. Na het uitvoeren van het scriptje in listing 8.20 krijgen we wel de verwachte uitvoer "`blaaiwaai`".

Listing 8.20: Met gebruik van eval

```
#!/bin/bash
bla=blaaiwaai
wa=bla
eval z='$'$wa
echo $z
```

8.12 Opgave

1. Leg uit wat het script in listing 8.21 doet. Maak gebruik van de man paginas voor de opdrachten die je niet kent.

2. Schrijf een script dat een reeks van bestanden aanvaardt en naar een back-up directory copiëert. Zorg ervoor dat de copies read-only zijn en maak een gezipt archive van de back-up directory. Vergeet niet om alle tijdelijke bestanden en directories op te ruimen.

3. Maak een script dat een reeks van bestanden aanvaardt en voor ieder bestand een functie aanroept. Deze functie schrijft de gegevens van het bestand (ls -l) en de inhoud uit.

4. Schrijf een script dat een directory als parameter krijgt en de inhoud per email naar de gebruiker stuurt. Tip: bekijk de man pages van mail.

5. Maak een script dat een lijst van email-adressen meekrijgt en als laatste argument een tekst. Zorg ervoor dat deze tekst doorgestuurd wordt naar alle email-adressen.

6. Maak een script dat als argument de login van een gebruiker meekrijgt en een melding op het scherm geeft wanneer dat die gebruiker inlogt. Dit wil zeggen dat het script moet blijven werken totdat de gebruiker inlogt, waarna het een melding geeft en dan stopt. Tip: u kan hiervoor het `sleep` en het `grep` commando gebruiken.

7. Breid het vorige programma uit zodat je een lijst van loginnamen kan meegeven om te melden. Het script is dan bedoeld om vanaf het starten ervan altijd te blijven lopen.

8. Maak een script dat als invoer shell-commando's krijgt en deze zelf uitvoert, waarbij er een minimale boekhouding gedaan wordt. Schrijf de naam van het process, uitvoeringstijd en starttijd in een bestand, samen met de gebruiker die het process opgestart heeft. Dit programma moet altijd blijven werken. Gebruik `read` voor de invoer. Zorg ervoor dat je het script kan beëindigen door quit in te typen.

Listing 8.21: Wat doet dit script?

```
#!/bin/bash
if [ -d ~/dustbin ]
then
  :
else
   mkdir ~/dustbin
fi

i=1
currentdir=‘pwd‘
mkdir -p ~/dustbin$currentdir
movedate=‘date +%s‘
while [ $i -le $# ]
do
 if [ -e $1 ]
   then
     mv $1 ~/dustbin$currentdir/$1.$movedate
     echo moving $1 to dustbin "("$1 "->" $1.$movedate")"
   fi
 shift
done
```

Hoofdstuk 9

Gnu Awk

GNU bevat een hoop programma's om taken te automatiseren. Deze programma's zijn meestal heel goed om één taak uit voeren. En uitzondering hierop is *gawk*, de GNU versie van *awk*. *awk* is namelijk een scripting taal die kan gebruikt worden om verschillende soorten taken te automatiseren. *awk* is genoemd naar de drie auteurs (Aho, Weinberger en Kernighan).

Deze cursus behandelt *gawk* omdat deze versie wordt meegeleverd bij de meeste Linux distributies. De verschillen tussen de verschillende *awk* versies zijn echter miniem. De principes die in deze cursus worden uitgelegd gelden dus grotendeels voor alle *awk* versies.

9.1 Principes van AWK

AWK is een scripting taal die gebruikt wordt om informatie uit tekstfiles te verwerken. De programmeur moet zelf geen files openen, lezen of sluiten; dit zal *awk* automatisch doen. De verschillend regels van de tekst files worden zelfs opgedeeld in tekstvelden.

awk kan worden opgestart vanuit de commandolijn, zoals in dit eerste voorbeeld:

```
gawk '{print NF ": " $0}' file.txt
```

Stel dat de file "file.txt" de volgende inhoud heeft:

```
Dit is een eerste voorbeeld.
Dit voorbeeld bevat      3         regels.
zoals een test!
```

Dan is het resultaat van het script:

```
5: Dit is een eerste voorbeeld.
5: Dit voorbeeld bevat      3         regels.
3: zoals een test!
```

Het script telt dus de woorden in iedere regel in de tekst file en schrijft dit uit. De invoer van het *awk* script is het resultaat van `cat file.txt`, de inhoud van deze file. *awk* zal vervolgens iedere regel van de invoer in velden onderverdelen. Omdat de verschillende velden door "whitespace" (een combinatie van spaties en tabs) worden gescheiden, komen de verschillende velden overeen met de woorden van het script. Vervolgens wordt met het `print` commando, het aantal velden van iedere regel (aangegeven door de variable `NF`) en de regel zelf (aangegeven door de variable `$0`) uitgeprint. Het is ook mogelijk om de verschillende velden uit te printen:

```
gawk '{print $1 " ... " $NF}' file.txt
```

Het resultaat is dan:

```
Dit ... voorbeeld.
Dit ... regels.
zoals ... test!
```

Ingewikkeldere *awk* scripts worden in een file gezet. Op deze manier kan een script meerdere keren worden uitgevoerd. Bv. het script listing 9.1 zal hetzelfde resultaat geven als het vorige voorbeeld, wanneer het wordt aangeroepen zoals afgebeeld in listing 9.2.

Listing 9.1: count.awk

```
{print $1 " ... " $NF}
```

Listing 9.2: count met awk

```
gawk -f count.awk file.txt
```

De meeste van de voorbeelden in deze tekst zulllen gebruik maken van een voorbeeld password file (genaamd "voorbeeld"). Dit is de file (`/etc/passwd`) waarin UNIX systemen alle gebruikersinformatie opslaan:

```
root:x:0:0:Super User:/root:/bin/sh
chris:x:101:20:Chris Raymaekers:/home/chris:/bin/tcsh
john:x:102:20:John Doe:/home/john:/bin/csh
jane:x:103:20:Jane Doe:/home/jane:/bin/sh
peter:x:104:30:Peter Icshello:/home/peter:/bin/sh
```

Iedere regel in deze file geeft informatie over één gebruiker. Het eerste veld geeft de gebruikersnaam aan. In het tweede veld staat het geëncrypteerde password van deze gebruiker (x geeft aan dat het password in een meer beveiligde file staat). De 2 volgende velden geven een getal aan dat de gebruiker uniek aanduidt en het getal dat zijn groep[1] aanduidt (gebruikers worden gegroepeerd, zodat bepaalde rechten

[1] Een groep heeft ook een gebruiksvriendelijke naam, zoals "users". Deze informatie staat in `/etc/groep`.

aan een groep van gebruikers kunnen worden toegekend). Het vijfde veld bevat de volledige naam van de gebruiker. Uiteindelijk komen de home directory en de shell van de gebruiker (zie ook hoofdstuk 8).

Als we deze file willen splitsen in meerdere regels, dan moeten we aangeven dat de verschillende velden niet gescheiden worden door whitespace, maar door : (de field separator is :). Dit kan aan *awk* worden aangegeven door de optie `-F ":"` aan awk mee te geven. Het is echter vervelend om dit iedere keer aan te geven als *awk* wordt aangeroepen. Een *awk* script kan echter een initialisatieblok bevatten, waarin dit soort variabelen wordt geïnitialiseerd:

```
BEGIN { FS = ":" }
{print $5}
```

De aanroep

```
gawk -f users.awk voorbeeld
```

heeft als resultaat:

```
Super User
Chris Raymaekers
John Doe
Jane Doe
Peter Icshello
```

Het blok, aangegeven door `BEGIN`, is het initialisatieblok en wordt dus slechts eenmaal aangeroepen. Het andere blok (op de volgende lijn) vormt het eigenlijke script en wordt voor iedere regel van het voorbeeldbestand aangeroepen. Een overzicht van de belangrijkste voorgedefinieerde variabelen (zoals `FS`) wordt gegeven in tabel 9.1.

Variabele	Betekenis	Standaardwaarde
FILENAME	Naam van het invoerbestand	
FS	Input field separator	whitespace
NF	Aantal velden in het huidige record	
NR	Aan records (lijnen) die al zijn gelezen	
ORS	Output record separator	Newline (enter)
RS	Input record separator	Newline (enter)

Tabel 9.1: Voorgedefinieerde variabelen

Op dezelfde manier kunnen we een blok specificeren dat enkel op het einde wordt uitgevoerd:

```
BEGIN { FS = ":" }
{print $5}
END { print NR " gebruikers"}
```

De uitvoer van dit script geeft:

```
Super User
Chris Raymaekers
John Doe
Jane Doe
Peter Icshello
5 gebruikers
```

Ook *awk* scripts kunnen commentaar bevatten: het teken # geeft aan dat de rest van de regel commentaar bevat.

9.2 Output

Tot nu toe werd in alle voorbeelden gebruik gemaakt van het `print` commando. Dit commando print de de gebruikte variabelen rechtstreeks uit (en voegt een newline toe). Eventuele getallen worden tot op 6 cijfers na de comma afgeprint. Het is echter ook mogelijk om zelf het formaat van de output te kiezen. Dit gebeurt met de functie `printf( format-specifier, variabele1, variabele2, ...)`. De "format-specifier" is een string (reeks van karakters) die aangeeft hoe de variabelen *variabele1, variabele2, ...* worden weergegeven. De lijst van de meest belangrijke formatie specifiers staat in tabel 9.2.

Een voorbeeld van deze specifiers is:

```
printf( "%i %f %e\n", 45, 3.2, 10 )
printf( "%g %g\n", 10, 10000000000 )
printf( "%c %c %s\n", 64 "a", "abc" )
```

Resultaat:

```
45 3.2 1.00000E+01
10 1.00000E+10
@ a abc
```

Vaak volstaat echter het `print` commando.

9.3 Input

De voorgaande voorbeelden printen alle lijnen uit. Soms willen we echter de invoer gaan filteren: we willen niet uit alle record gegevens gaan uitschrijven. De verschillende velden kunnen gecontroleerd worden met een aantal vergelijksoperatoren (zie ook tabel 9.3). Voorbeeld listing 9.3 schrijft de namen van de gebruikers uit, waarvan de groeps ID (veld #4) gelijk is aan 20.

Listing 9.3: Schrijf gebruikersnamen uit met groep ID=20

```
BEGIN { FS = ":" }
$4 == 20 {print $5}
```

Variabele	Betekenis
\n	newline (enter)
\t	tab
\"	"
\\	\
%i	integer (geheel getal)
%f	floating point (reëel getal)
%e	floating point in wetenschappelijke notatie
%g	%f of %e (wat het kortste is)
%c	ASCII karakter
%s	string (reeks van karakters)

Tabel 9.2: Format specifiers

De uitvoer van het script in listing 9.3 geeft:

```
Chris Raymaekers
John Doe
Jane Doe
```

Operator	Betekenis
==	gelijk
! =	niet gelijk
>	groter dan
>=	groter dan of gelijk aan
<	kleiner dan
<=	kleiner dan of gelijk aan
~	Voldoet aan een reguliere expressie(zie paragraaf 9.5)
! ~	Voldoet niet aan een reguliere expressie(zie paragraaf 9.5)

Tabel 9.3: Vergelijkingsoperatoren

Verschillende testen kunnen met behulp van logische operatoren worden gecombineerd (zie tabel 9.4).

9.4 Variabelen

In *awk* kunnen ook variabelen worden gedefinieerd om berekeningen en tussentijdse waarden op te slaan. Een variabele moet niet expliciet worden gedefinieerd. De naam is case-sensitive; pas dus goed op dat je de naam van een variabele altijd op dezelfde manier schrijft, anders kun je per ongeluk een nieuwe variabele aanmaken. Een nieuwe variabele wordt steeds op 0 geïnitialiseerd.

Operator	Betekenis
&&	logische en
||	logische of
!	logische niet
()	combineren van logische expressies

Tabel 9.4: Logische operatoren

Voorbeeld listing 9.4 telt alle woorden in het voorbeeldbestand "file.txt" (een overzicht van de belangrijkste wiskundige operatoren staat in tabel 9.5)

Listing 9.4: Tel alle woorden

```
{ teller += NF }
END { print teller }
```

Operator	Betekenis
+	optelling
−	aftrekking
*	vermenigvuldiging
/	deling
%	rest na deling
∧	macht
=	toekenning

Tabel 9.5: Wiskundige operatoren

Met behulp van een de functie `split`, kan een veld worden opgesplitst in meerdere velden. Voorbeeld listing 9.5 schrijft de achternaam uit van alle gebruikers van de voorbeeld password file.

Listing 9.5: Schrijf de achternaam uit

```
BEGIN { FS = ":" }
$3 != 0 { split( $5, naam, " " ); print naam[2] }
```

Merk op dat `split` het resultaat in een array opslaat. Het verschil tussen arrays in *awk* en veel andere talen is dat de indices van *awk* arrays beginnen bij 1.

9.5 Reguliere expressies

Vaak wil je gaan zoeken naar een string die op een andere string lijkt, maar niet identiek hetzelfde is. Dit kan worden bereikt door het gebruik van reguliere expressies. Een reguliere expressie is een string die metakarakters bevat (zie tabel 9.6 en hoofdstuk 6 voor meer informatie).

Metakarakter	Komt overeen met
\	escape karakter: plaats dit voor een metakarakter dat je letterlijk wilt gebruiken
∧	begin van een string
$	einde van een string
/∧$/	een lege lijn
.	een willekeurig karakter
[ABC]	A, B of C
[A-Ca-C]	A,B,C, a, b of c
[∧ABC]	ieder karakter, behalve A, B en C
Stoel—Tafel	"Stoel" of "Tafel"
[ABC][DEF]	A,B of C, gevolgd door D, E of F
[ABC]*	0 of meerdere opeenvolgingen van A, B of C
[ABC]+	1 of meerdere opeenvolgingen van A, B of C
[ABC]?	A, B, C of niets
()	Combineert reguliere expressies

Tabel 9.6: Metakarakters

Reguliere expressies zijn zeer krachtig in het verwerken van teksten en worden in Unices veel gebruikt. Deze cursustekst geeft slechts een inleiding, aangezien dit een zeer uitgebreid onderwerp is en volledige boeken[2] zijn geschreven over het gebruik van reguliere expressies.

Bijvoorbeeld: de strings "linkermuisknop" en "rechtermuisknop" voldoen aan `/(linker|rechter)muisknop/` De strings "muisknop", "middelste muisknop" en "Linkermuisknop" voldoen echter niet. Deze twee eerste strings voldoen echter aan `/(linker|rechter)?muisknop/`

Omdat `?` aangeeft dat `(linker|rechter)` ten hoogste éénmaal mag voorkomen. De volledige string moet echter niet voldoen aan de reguliere expressie. Het is voldoende dat een deel van de string voldoet aan de (volledige!) reguliere expressie. Daarom voldoen bovenstaande strings aan `/muisknop/`

Stel dat we nu willen dat alleen "muisknop" voldoet aan de reguliere expressie, dan moeten we aangeven dat de string begint met "muisknop": `/^muisknop/` Als we in het voorbeeld password file willen controleren welke gebruikers `csh` of `tcsh` als shell hebben, moeten we in principe enkel controleren welke velden overeenkomen met "csh":

```
BEGIN { FS = ":" }
/csh/ { print $5 }
```

De uitvoer van dit script geeft dan:

```
Chris Raymaekers
```

[2] Een uitstekend boek is *Mastering regular expressions* (Jeffrey Friedl, O'Reilly).

```
John Doe
Peter Icshello
```

De gebruikers "Chris Raymaekers" (tcsh) en "John Doe" (csh) voldoen hier inderdaad aan, maar "Peter **Icsh**ello" heeft `sh` als shell. De "csh" in zijn naam komt overeen met de reguliere expressie. Er zijn 2 manieren waarop we dit kunnen oplossen:

```
BEGIN { FS = ":" }
/(\/bin\/)t?(csh)/ { print $5 }
```

```
BEGIN { FS = ":" }
$NF ~ /csh/ { print $5 }
```

De eerste oplossing controleert of `/bin/csh` of `/bin/tcsh` in de regel voorkomt. De haakjes zijn niet nodig, maar werden toegepast om de oplossing iets leesbaarder te maken. De tweede oplossing controleert enkel of `csh` in het laatste veld voorkomt. Alhoewel beide oplossing correct zijn, is de tweede oplossing beter:

- De tweede oplossing controleert enkel het veld dat relevant is.
- De tweede oplossing maakt gebruik van een eenvoudigere reguliere expressie en wordt daarom sneller uitgevoerd.
- De tweede oplossing is beter leesbaar en daarom gemakkelijker om uit te breiden en te onderhouden.

9.6 Conditionele expressie

awk kent ook constructies die je in de meeste scripting en programmeertalen tegenkomt, zoals `if`.

Stel dat we een databank file (genaamd "bestelling") van bestellingen hebben die er als volgt uitziet:

```
Computer:50000:1:2
Boek:1000:4:1
```

Het eerste veld geeft een beschrijving van het artikel, het tweede geeft de eenheidsprijs zonder BTW, het derde veld geeft aan hoeveel artikels werden besteld, het laatste veld bevat een code die aangeeft hoeveel BTW moet worden aangerekend. Het volgende script (listing 9.6, genaamd "factuur.awk") maakt een factuur van deze file.

Listing 9.6: factuur.awk

```
BEGIN {
# de velden worden gescheiden door :
   FS = ":"
# print de hoofding
```

```
   printf( "%-15s | %-13s | %-11s | %-7s | %-7s\n", "Artikel", "Eenheidsprijs",
      "Hoeveelheid", "Prijs", "BTW inc" )
   printf( "---------------------------------" )
   printf( "---------------------------------\n" )
}

{
# totale prijs voor dit artikel
   prijs = $2 * $3
# tel dit op bij het volledige totaal
   totaal += prijs;
# bereken de BTW
   if ( $4 == 1 )
      btw = prijs * 1.15 #15%
   else
      btw = prijs * 1.21 #21%
# tel dit op bij het volledige totaal
   totaalbtw += btw
# schrijf het rapport uit
   printf( "%-15s | %13i | %11i | %7i | %7i\n", $1, $2, $3, prijs, btw )
}

END {
   printf( "---------------------------------" )
   printf( "---------------------------------\n" )
# print de totalen
   printf( "%-18s: %6i BF (%7.2f euro)\n", "Totaal zonder BTW", totaal,
      totaal / 40.3399 )
   printf( "%-18s: %6i BF (%7.2f euro)\n", "Totaal met BTW", totaalbtw,
      totaalbtw / 40.3399 )
}
```

Het resultaat van de uitvoer is dan:

```
Artikel         | Eenheidsprijs | Hoeveelheid | Prijs   | BTW inc
-----------------------------------------------------------------
Computer        |         50000 |           1 |   50000 |   60500
Boek            |          1000 |           4 |    4000 |    4600
-----------------------------------------------------------------
Totaal zonder BTW :  54000 BF (1338.63 euro)
Totaal met BTW    :  65100 BF (1613.79 euro)
```

Met behulp van het programma `sort` kunnen we ervoor zorgen dat de artikels in alfabetische volgorde staan:

```
sort bestelling | awk -f factuur.awk
```

Het resultaat van deze oproep is:

```
Artikel         | Eenheidsprijs | Hoeveelheid | Prijs   | BTW inc
```

```
-----------------------------------------------------------------
Boek            |          1000 |           4 |    4000 |    4600
Computer        |         50000 |           1 |   50000 |   60500
-----------------------------------------------------------------
Totaal zonder BTW :  54000 BF (1338.63 euro)
Totaal met BTW    :  65100 BF (1613.79 euro)
```

De getallen die bij de format specifiers van `printf` staan, geven de (minimale) breedte van de verschillende velden aan. De velden worden standaard rechts uitgelijnd; een minteken geeft aan dat een veld links moet worden uitgelijnd. `7.2f` betekent dat 7 karakters moeten worden voorzien voor het uitschijven van een reëel getal, waarvan 1 karakter voor de decimale punt en 2 karakters voor de cijfers na het decimale punt.

awk kent ook een conditionele operator: de if constructie van het vorige voorbeeld kan afgekort worden tot:

```
btw = ( $4 == 1 ) ? prijs * 1.15 : prijs * 1.21
```

9.7 Iteraties

awk kent ook enkele lus-structuren:

```
for ( initialisatie ; test ; increment )
    ...

for ( teller in array )
    ...

while ( test )
    ...
```

De eerste en derde iteratie constructie komen ook voor in C, Java en JavaScript. De tweede constructie is een constructie die in sommige scripting talen bestaat en kan gebruikt worden om alle waarden in een array te doorlopen (zie ook sectie 9.4).

Met behulp van een iteratie kan code die de horizontale lijn uitschrijft in het vorige voorbeeld geschreven worden als:

```
for ( i = 0 ; i < 65 ; i++ )
   printf( "-" )
printf( "\n" )
```

9.8 Functies

Zoals de meeste talen kent *awk* functies om veel gebruikte code te groeperen.

In het voorbeeld getoond in listing 9.6 wordt tweemaal een horizontale lijn getekend. We kunnen hiervoor een functie definiëren.

```
function printline( count )
{
   for ( i = 0 ; i < count ; i++ )
      printf( "-" )
   printf( "\n" )
}
```

De functie kan dan op de volgende manier worden opgeroepen in het script:

```
printline( 65 )
```

Met behulp van het `return` commando kan een functie een waarde teruggeven.

9.9 Voorbeeld

Deze sectie geeft nog een extra voorbeeld, waarin enkele constructies worden herhaald. Ook wordt extra informatie gegeven over de werking van arrays.

Stel dat op een computer een log file staat van de volgende vorm:

```
boot:disk2 failed
network:telnet:connection refused from beezer.mysite.be
network:ftp:connection refused from beezer.mysite.be
login:John:bad password
network:telnet:connection refused from www.mysite.be
```

De systeembeheerder wil statistieken maken van de log file. Er moet bijgehouden hoeveel "boot" (opstart) errors, hoeveel "login" errors en hoeveel "network" errors zijn opgetreden. Verder moeten statistieken worden bijgehouden over de oorzaken van deze errors. Het script houdt ook bij hoeveel onbekende fouten er zijn opgetreden, zodat het script kan worden aangepast als een nieuw soort fout optreedt.

Listing 9.7: Statistieken van een log file

```
BEGIN {
   FS = ":"
}

$1 == "boot" { #boot error
   boot++;
   known++
}

$1 == "login" { #login error
   login++;
   logins[$2]++; #user who caused the error
   known++
}
```

```
$1 == "network" { #network error
   network++;
   services[$2]++ #service that reported the error
   split( $NF, messages, " " );
   sites[ messages[4] ]++ #site that caused the error
   known++
}

{
   print $0 #print all errors
   total++
}

END {
   printf( "\nSummary\n" )
   print ( "-------" )
   printf( "Boot: %i error(s)\n", boot )

   printf( "Login: %i error(s)\n", login )
   for ( user in logins ) #print login errors
      printf( "  %s: %i bad login(s)\n", user, logins[ user ] )

   printf( "Network: %i error(s)\n", network )
   for ( service in services ) #print network errors
      printf( "  %s: %i connection(s) refused\n", service, services[ service ] )
   for ( site in sites ) #print network errors
      printf( "  %i connection(s) refused from %s\n", sites[ site ], site )

   if ( total - known > 0 ) #there are unknown errors
      printf( "%i unknown error(s)\n", total - known )
}
```

Merk op dat in tegenstelling tot programmeertalen *awk* willekeurige waarden kan gebruiken als index van een array (dit is typische voor scripting talen zoals *awk* en *perl*).

Het uitvoeren van dit script geeft het volgende resultaat.

```
boot:disk2 failed
network:telnet:connection refused from beezer.mysite.be
network:ftp:connection refused from beezer.mysite.be
login:John:bad password
network:telnet:connection refused from www.mysite.be

Summary
-------
Boot: 1 error(s)
Login: 1 error(s)
   John: 1 bad login(s)
```

```
Network: 3 error(s)
   telnet: 2 connection(s) refused
   ftp: 1 connection(s) refused
   2 connection(s) refused from beezer.mysite.be
   1 connection(s) refused from www.mysite.be
```

Hoofdstuk 10

Programmeren onder Linux

10.1 Beschikbare middelen

Op zowat elk Linux systeem is de C compiler van de GNU Compiler Compiler Collection aanwezig: *gcc*, die ANSI C (en C++) compileert. De volgende lijst geeft een overzicht van andere programma's om software mee te schrijven:

ar om bibliotheek bestanden te creëren;

as om object bestanden te creëren (assembler);

bison om parsing tabellen te genereren;

flex lexical analyzer;

cpp C code preprocessor;

ld link editor (dynamisch libraries aanmaken);

make, automake, autoconf om automatisch programma's mee te creëren, hercreëren, updaten en onderhouden. Zie ook appendix B;

patch, diff om patches op broncode aan te maken en toe te passen;

gdb GNU debugger.

10.2 De GNU C Compiler

We zullen ons enkel concentreren op het gebruik van *gcc* en *g++*, omdat deze de meest gebruikte compilers zijn op Linux systemen. gcc staat voor Gnu C Compiler, en g++ is de GNU C++ compiler; vaak kan je met gcc zowel C als C++ compileren. Een volledig overzicht van alle opties (switches) die bruikbaar zijn met gcc zou ons te ver brengen, daarom zullen we ook enkel de meest relevante opties verduidelijken en aangeven.

Om het versienummer van de compiler te weten te komen kan men de switch `-v` gebruiken: `gcc -v`. De uitvoer ziet er dan als volgt uit, afhankelijk van de versie van gcc en de installatie:

```
Reading specs from /usr/lib/gcc-lib/i586-mandrake-linux/2.95.3/specs
gcc version 2.95.3 19991030 (prerelease)
```

Met de `-o` switch kan je aangeven welk uitvoerbestand het moet gebruiken om het programma weg te schrijven, bijv.: `gcc -o disaster.out disaster.c` compileert het bestand `disaster.c` en genereert het uitvoerbestand `disaster.out` van de gecompileerde code. Stel dat we het zeer eenvoudige programma hello.c in listing 10.1 zouden compileren met `gcc hello.c`, zonder de -o switch te gebruiken. Dan genereert gcc standaard een uitvoerbestand `a.out`, dat men kan uitvoeren met `./a.out`. Met `gcc -o hello hello.c` wordt het uitvoerbare bestand `hello`, wat we kunnen uitvoeren door `./hello`.

Listing 10.1: "hello world" programma

```
/* hello.c */
int main()
{
  printf("hello world\n");
}
```

De switch `-Wall` zorgt voor meer uitgebreide foutmeldingen en waarschuwingen. Hier is een lijstje met nog andere switches voor gcc:

`-o file`	plaats de uitvoer in `file`
`-ansi`	ondersteun al de ANSI standaard C programma's
`-include file`	verwerk `file` eerst, en daarna pas het gewone invoerbestand
`-static`	verbiedt de compiler dynamisch linken toe te staan
`-shared`	zorgt ervoor dat de compiler een gedeelde bibliotheek genereert (zie ook sectie 10.3)
`-Idir`	voeg directory `dir` toe om erin naar include files te zoeken
`-W`	print extra waarschuwingen
`-Wall`	Laat *zoveel mogelijk* waarschuwingen zien
`-g`	produceer debug informatie, handig voor `gdb` (kan ook met `-ggdb`)
`-O`	optimaliseer de uitvoer: probeert de grootte te beperken en uitvoeringssnelheid te maximaliseren

Opgave Schrijf zelf een programma; implementeer bijvoorbeeld uw favoriete sorteer-algoritme met behulp van gcc.

10.3 Bibliotheken

Deze sectie geeft een beknopte inleiding tot het programmeren met bibliotheken in Linux en een opsomming van de voordelen ervan bij het gebruik in eigen appli-

caties. Een bibliotheek is een bestand dat code en data bevat die op zichzelf niet uitvoerbaar zijn, maar gelinkt moeten worden aan een applicatie. Het gebruik van bibliotheken kan ervoor zorgen dat applicaties meer uit aparte modules bestaan, sneller te compileren zijn en bovendien gemakkelijker up-to-date gehouden kunnen worden. Bibliotheken stimuleren ook herbruikbaarheid: andere applicaties kunnen er ook gebruik van maken, zonder dat de bibliotheek gehercompileerd hoeft te worden.

Indien het tijdstip waarop een bibliotheek wordt geladen als criterium wordt genomen, kunnen we drie soorten onderscheiden: *statische bibliotheken* (static libraries), *gedeelde bibliotheken* (shared libraries) en *dynamisch geladen bibliotheken* (DL libraries). Verder wordt er ook een onderscheid gemaakt tussen verschillende formaten. We hanteren hier echter enkel het *ELF formaat*, het Executable and Linking Format, omdat dit tegenwoordig het meest gebruikt wordt. Voor een vollediger overzicht en meer voorbeelden verwijzen we naar [Whe00].

10.3.1 Statische bibliotheken

Een statische bibliotheek bestaat uit een collectie van object-bestanden en wordt volledig in het uitvoerbare programma gecopiëerd tijdens compilatie. Programmeurs kunnen de bibliotheek aan hun applicaties linken zonder dat deze bibliotheek gehercompileerd moet worden, iets wat behoorlijk wat tijd uitspaart. Ze zijn ook bruikbaar indien men andere gebruikers wil toelaten hun applicatie aan jouw bibliotheek te linken zonder dat ze beschikken over de broncode ervan. Een laatste voordeel is dat ze in theorie bij de uitvoer iets sneller zijn dan de andere soorten bibliotheken. In de praktijk blijkt dit echter zelden het geval te zijn wegens allerlei andere complicaties.

Ondanks deze voordelen worden statische bibliotheken niet zoveel meer gebruikt als vroeger. De voordelen van gedeelde bibliotheken overheersen namelijk, zoals in de volgende sectie zal blijken. Het volgende voorbeeld toont hoe een statische bibliotheek wordt aangemaakt aan de hand van enkele object bestanden die met het programma *ar* (sectie 10.1) tot één archief gecombineerd worden :

```
ar rcs mijn_bibliotheek.a bestand1.o bestand2.o
```

De opties `rcs` staan respectievelijk voor het toevoegen van object bestanden (insert with **r**eplacement), het aanmaken van een archief (**c**reate) en het creëren of overschrijven van een object bestand index.

10.3.2 Gedeelde bibliotheken

Gedeelde bibliotheken worden geladen zodra een gelinkte applicatie wordt opgestart. De eigenlijke bibliotheekcode wordt dus niet in de binary meegecompileerd maar slechts als verwijzing in de applicatie bewaard. Gedeelde bibliotheken hebben de volgende interessante eigenschappen:

- Een bibliotheek kan bijgewerkt of aangevuld worden terwijl applicaties die *niet-backwards-compatibele versies van de bibliotheek* gebruiken nog steeds ondersteund worden. Dit wordt mogelijk gemaakt door de manier waarop bibliotheken beheerd worden, namelijk door het onderscheid te maken tussen verschillende namen van een bibliotheek, zoals verder wordt uitgelegd. De verschillende versies kunnen dus naast elkaar blijven bestaan.

- Een bibliotheek, of zelfs specifieke functies in die bibliotheek, kunnen overridden worden bij de uitvoer van een gelinkte applicatie.

- Dit alles is mogelijk terwijl andere applicaties runnen en gebruik maken van bestaande bibliotheken.

Om deze eigenschappen te kunnen ondersteunen is men verplicht een aantal richtlijnen te volgen. Het is noodzakelijk het verschil te kennen tussen de verschillende namen van de bibliotheek, en verder moet men ook enige notie hebben van waar de bibliotheek in kwestie uiteindelijk geplaatst moet worden.

Elke gedeelde bibliotheek heeft een drietal namen:

- De **soname** begint met de prefix "lib", gevolgd door de naam van de bibliotheek, de letters ".so", en afgesloten door een punt en de versienummer die wordt verhoogd telkens de interface van de bibliotheek gewijzigd werd. Meestal is de *soname* een symbolische link naar de *echte naam*.

- Met de **echte naam** wordt het bestand aangeduid dat de eigenlijke bibliotheekcode bevat. Het formaat voegt aan de *soname* een punt, het minor nummer, nog een punt, en het optionele release nummer. De twee extra nummers laten gebruikers weten wat de exacte versie is van een geïnstalleerde bibliotheek, zodat ook meerdere versies geïnstalleerd kunnen worden zonder verwarring te creëren.

- De **linker naam** wordt gebruikt door de compiler wanneer deze de bibliotheek wil gebruiken. Ze wordt gevormd door de *soname* zonder enige versienummer. Meestal is dit ook een symbolische link naar een *soname* of naar een *echte naam*.

Voordat getoond wordt hoe een gedeelde library gemaakt kan worden, eerst nog een paar opmerkingen over de compatibiliteit tussen verschillende versies van een bibliotheek. Zoals eerder vermeld moet de "soname" aangepast worden indien een nieuwere versie van de bibliotheek binair-incompatibel is met de oudere versie. [Whe00] somt een aantal oorzaken op, voor zowel C als C++, waardoor de ABI (Application Binary Interface) van een bibliotheek verandert en een bibliotheek dus incompatibel wordt met oudere versies.

Als eenvoudig voorbeeld worden eerst een tweetal object bestanden gecreëerd (bestand1.o en bestand2.o), en daarna een gedeelde bibliotheek die beiden bevat.

```
g++ -fPIC -g -c -Wall bestand1.cpp
g++ -fPIC -g -c -Wall bestand2.cpp
g++ -shared -Wl,-soname,libmijn_bib.so.1 -o libmijn_bib.so.1.0.1
     bestand1.o bestand2.o -lc
```

Naast de gekende parameters gebruikten we `-fPIC`, een vereiste bij gedeelde bibliotheken om "Position Independent Code" te genereren, `-c` om object bestanden te creëren en `-shared` om aan te geven dat we een gedeelde bibliotheek willen. `-Wl` laat toe enkele parameters aan de linker door te geven, gescheiden door komma's zoals hier de optie `-soname` met de naam die we gekozen hebben. De parameter `-lc` lijkt nieuw, maar geeft aan dat de bibliotheek libc meegelinkt moet worden.

Als nu nog de nodige symbolische links aangemaakt worden voor de *soname* en de *linker naam* kan de bibliotheek gelinkt worden aan een applicatie:

```
g++ [...] -lmijn_bib
```

Zodra echter de bibliotheek door meerdere applicaties gebruikt moet kunnen worden, is het noodzakelijk dat ze ergens op een centrale plaats staat. Volgens twee standaarden (de *GNU standard* en de *Filesystem Hierarchy Standard*[1]) is `/usr/local/lib` de aangewezen plaats voor bibliotheken die mogelijk nog fouten bevatten, en in `/usr/lib` zouden de latere, stabielere versies moeten komen. Het programma **ldconfig** zorgt ervoor dat een verwijzing naar de bibliotheek in een speciaal bestand wordt toegevoegd zodat er snellere toegang tot de bibliotheek mogelijk is (zie verder).

10.3.3 Dynamisch geladen bibliotheken

Dynamisch geladen bibliotheken of DL bibliotheken worden geladen op eender welk tijdstip dat verschilt van het moment dat een applicatie wordt opgestart. Dit biedt de mogelijkheid tot het creëren van plug-ins of modules. Het tijdstip waarop de module geladen wordt kan dan immers uitgesteld worden totdat de inhoud ervan voor het eerst gebruikt wordt.

Het formaat van een DL bibliotheek verschilt niet van dit van statische en gedeelde bibliotheken, enkel het tijdstip waarop ze geladen wordt. Een speciale API zorgt voor het openen van een DL bibliotheek, het opzoeken van symbolen, het afhandelen van fouten en het sluiten van de bibliotheek. Deze interface is echter niet op elk platform hetzelfde [Whe00], we behandelen hier dus enkel de interface ondersteund door Linux.

De DL bibliotheek API (gedefinieerd in de dlfcn.h) biedt de volgende vier functies:

```
void* dlopen(const char* filename, int flag); // Opent een bibliotheek.
void* dlsym(void* handle, char* symbol);      // Geeft de waarde van een symbool.
char* dlerror();                              // Een beschrijving van de foutmelding.
int dlclose(void* handle);                    // Sluit de bibliotheek.
```

[1] http://www.pathname.com/fhs

De parameter "flag" in de functie *dlopen* kan drie waarden aannemen: `RTLD_LAZY` zorgt ervoor dat symbolen enkel *resolved* worden zodra ze worden aangeroepen vanuit de code van de bibliotheek. `RTLD_NOW` verlegt dit tijdstip tot vóór de functie dlopen() is afgelopen. Tenslotte, `RTLD_GLOBAL` mag optioneel ge-or'd worden met de twee vorige waarden en geeft aan dat alle externe symbolen die in de bibliotheek gedefinieerd werden ter beschikking gesteld worden van bibliotheken die achteraf nog geladen worden.

Listing 10.2 haalt een pointer naar een functie uit de bibliotheek "libm" en gebruikt ze om de cosinus van 2.0 te berekenen (voor meer uitleg over pointers naar functies, zie appendix A).

Listing 10.2: dllib.c

```
#include <stdlib.h>
#include <stdio.h>
#include <dlfcn.h>

int main(int argc, char **argv) {

  void *handle;
  double (*cosine)(double);
  char *error;

  handle = dlopen ("/lib/libm.so", RTLD_LAZY);
  if( !handle) {
    fputs(dlerror(), stderr);
    exit(1);
  }

  cosine = dlsym(handle, "cos");
  if((error = dlerror()) != NULL) {
    fputs(error, stderr);
    exit(1);
  }

  printf ("%f\n", (*cosine)(2.0));
  dlclose(handle);

  return 0;
}
```

10.3.4 Relevante utilities

Het gereedschap dat je nodig hebt om aan de slag te gaan en je eigen libraries te creëren bestaat uit de volgende utilities: ldconfig, ldd, nm. Deze worden meestal standaard mee geïnstalleerd bij je GNU/Linux distributie.

10.4 Versiebeheer

Tijdens het werken aan grotere software projecten komen we vaak de volgende problemen tegen:

1. De software doorloopt verschillende stadia (versies). Om op ieder ogenblik in staat te zijn bepaalde versies te reproduceren moeten we weten welke wijzigingen door wie, waarom en wanneer zijn aangebracht.

2. Wanneer verschillende personen tegelijk aan hetzelfde deel van de software willen werken, moet dit kunnen zonder dat ze elkaar in de weg lopen.

3. De benodigde hoeveelheid geheugenruimte om de verschillende versies bij te houden moet zo minimaal mogelijk gehouden worden. Ipv van elke versie een aparte kopie bij te houden, zouden alleen de verschillen tussen de versies bijgehouden moeten worden.

Om deze problemen op te lossen, zijn er versie-beheer pakketten ontwikkeld. De twee meest populaire pakketten op Linux zijn Revision Control System (RCS) en Concurrent Version System (CVS). CVS is een front-end van RCS waaraan extra features zijn toegevoegd:

- Terwijl RCS de wijzigingen en versies bestuurt vanuit een enkele directory, breidt CVS de besturingsstructuur uit tot een hieïarchische verzameling van directories. De directory waarin de wijzigingen en versies worden bijgehouden, wordt de repository genoemd. RCS dient dus meer om de verschillende versies van aparte files te beheren en CVS dient meer om gehele projecten te beheren.

- CVS ondersteunt gedistribueerde projecten.

10.4.1 Basisbewerkingen

Doorgaans worden RCS-bestanden bewaard in de subdirectory "RCS". De eerste stap in RCS is dan meestal ook de volgende: `mkdir RCS`. Daarna kan een bestand onder RCS-controle geplaatst worden door het bestand *in te checken*: dit gebeurt met het commando `ci bestand`. Hiermee wordt een bestand met als naam `bestand,v` gemaakt in de RCS directory. Dit bestand, `bestand,v`, wordt een RCS-bestand genoemd en zal alle toekomstige revisies van bestand bewaren. Wanneer ci voor de eerste keer wordt uitgevoerd op een bestand, wordt er gevraagd de inhoud van het bestand te beschrijven. Daarna deponeert ci het bestand in het RCS-bestand als revisie 1.1.

Wanneer we een bestand dat onder RCS-controle staat alleen willen lezen, bv om iets in op te zoeken of om te compileren, kunnen we het bestand als volgt *uit checken*: `co bestand`. RCS haalt nu een kopie van het bestand uit het RCS-bestand waarvan de permissies op *read-only* staan.

Wanneer we een bestand dat onder RCS-controle staat willen bewerken, dus lezen en schrijven, moeten we het bestand als volgt *uit checken*: `co -l bestand`. RCS haalt nu een kopie van het bestand uit het RCS-bestand waarvan de permissies op *read-write* staan. De optie `-l` dient om het bestand te vergrendelen of te "locken" en voorkomt dat meerdere personen tegelijk hetzelfde bestand kunnen bewerken.

Wanneer je een werkbestand wilt vergelijken met de laatste revisie kan je volgende opdracht gebruiken: `rcsdiff bestand`. Een andere nuttige opdracht is `rlog`. Rlog toont een pak nuttige logberichten zoals, RCS pathname, default branch, totaal aantal revisies, status van de file, ... Je kan deze logberichten onder andere gebruiken om de geschiedenis van een bestand te achterhalen.

10.4.2 Trefwoordvervanging

In RCS kan je trefwoordvariabelen in werkbestanden plaatsen. Trefwoordvariabelen zijn variabelen die nadat een bestand is ingechecked, door RCS uitgebreid worden tot revisieaantekeningen. Revisieaantekeningen vermelden bijvoorbeeld de auteur die een bestand het laatst heeft bewerkt, of de datum waarop het bestand voor het laatst is bewerkt geweest. Als je revisienotities wilt maken door middel van trefwoordvervanging voer je onderstaande procedure uit:

1. typ in het werkbestand een van de trefwoorden;
2. schrijf het bestand in;
3. schrijf het bestand weer uit. Tijdens het uitschrijven, breidt de opdracht `co` elk trefwoord uit;
4. bij een volgende inschrijving en uitschrijving van een bestand worden de bestaande trefwoordwaarden bijgewerkt.

Trefwoorden

De volgende lijst bevat enkele trefwoordvariabelen die gebruikt kunnen worden in RCS bestanden, samen met een korte beschrijving van hun betekenis:

- $Author$: gebruikersnaam van de persoon die de revisie heeft ingeschreven.
- $Date$: de datum en het tijdstip van inschrijving.
- $Headers$: een titel die de volledige padnaam van het RCS-bestand vermeldt, evenals het revisienummer, de datum, de auteur, de status en (indien vergrendeld) de persoon die het bestand heeft vergrendeld.
- Id: idem als $Headers$, maar de volledige padnaam van het RCS-bestand wordt niet vermeld.

- $Locker$: gebruikersnaam van de persoon die het bestand heeft vergrendeld. Als het bestand niet vergrendeld is, is deze waarde leeg.
- Log: het bericht dat bij het inschrijven is ingegeven om het bestand te beschrijven, voorafgegaan door de RCS-bestandsnaam, het revisienummer, de auteur en de datum. Logberichten worden telkens aangevuld en worden niet overschreven.
- $RCSfile$: de RCS-bestandsnaam zonder padnaam.
- $Revision$: het toegekende revisienummer.
- $Source$: de RCS-bestandsnaam, inclusief padnaam.
- $State$: de status die met de optie `-s` van `ci` of `rcs` is toegekend.

10.4.3 Revisienummering

Tenzij anders is aangegeven, werken RCS-opdrachten doorgaans met de laatste revisie. Enkele opdrachten hebben de optie `-r` waarmee je een revisienummer kunt opgeven. Revisienummers bestaan uit maximum vier velden: release, level, branch en sequence, maar de meeste revisies bestaan alleen uit release en level.

Op de volgende manier kan je revisie 1.4 uitschrijven: `co -l -r1.4 hfdst01`. Wanneer je dit bestand weer inschrijft, wordt de nieuwe revisie gemarkeerd als 1.5. Maar stel dat de bewerkte kopie als de volgende versie moet worden ingeschreven, dan geef je de volgende opdracht: `ci -r2 hfdst01`. Met deze opdracht maak je revisie 2.1. Je kan ook vertakkingen maken vanuit een oudere revisie. Met de volgende opdracht maak je revisie 1.4.1: `ci -r1.4.1 hfdst01`.

10.4.4 Toestanden opgeven

Vooral in programmeeromgevingen, willen we de toestand kennen van een reeks revisies. Men zegt dat een RCS-bestand zich in verschillende toestanden kan bevinden. RCS toestanden zijn gebaseerd op de status van het bestand binnen het project. Zo kan een RCS-bestand een hele cyclus doorlopen, beginnend van experimenteel naar release, totdat het uiteindelijk een stabiele, of m.a.w. werkende, toestand bereikt. Toestanden worden door de gebruiker gedefinieerd en hebben geen speciale interne betekenis. Aan een RCS-bestand kan je een bepaalde toestand toekennen, door het bestand te markeren met een tekenreeks die de *toestand* beschrijft. De standaardtoestand van een bestand bij initialisatie is **Exp** (experimenteel). Andere keuzen zijn **Stab** (stable) of **Rel** (release).

Hoofdstuk 11

Processen en Threads

Linux is een *multi-tasking* besturingssysteem, wat wil zeggen dat er op ieder moment meer dan één programma "tegelijk" kan draaien. "Tegelijk" staat tussen aanhalingstekens, omdat een enkele processor fysisch gezien slechts de bevelen van één enkel programma kan uitvoeren; maar het besturingssysteem wisselt ettelijke malen per seconde van programma (dit heet *scheduling*), zodat het lijkt alsof die programma's echt allemaal tegelijk aan het draaien zijn. Indien het een multi-process systeem betreft, is het natuurlijk *wel* mogelijk dat er meer dan één proces tegelijk draait.

Dit hoofdstuk introduceert de concepten achter multi-tasking, en bespreekt de meest courante bevelen om de boekhouding van al die verschillende taken te beheren.

11.1 Definitie processen

Een proces is een programma in uitvoering. Het programma zelf is niet meer dan een bestand dat uitvoerbare code bevat. Als het programma uitgevoerd wordt, dan laadt het besturingssysteem die uitvoerbare code in het geheugen. Maar het besturingssysteem voorziet voor elk proces niet alleen plaats voor de uitvoerbare code die uit het bestand komt, maar ook voor de gegevens die tijdens de uitvoering aangemaakt worden:

- de *stack*: hierop komen de parameters en adressen voor functieoproepen.
- de *heap*, die dient voor dynamisch gecreëerde variabelen.
- de *registers*, waar een proces zijn huidige stack wijzer, instructie-wijzer, en register-inhouden bewaart, op het ogenblik dat de scheduler de uitvoering ervan stopt, en een ander proces inlaadt.
- de *file descriptors*. Dit zijn pointers naar geopende bestanden, communicatie-kanalen, enz.

- de *page table*. Dit zijn pointers naar de geheugen-pagina's die het proces gebruikt.

Het geheel van alle hierboven vermelde gegevens van een programma in uitvoering noemen we de *context* van het proces. Bij scheduling schrijft het besturingssysteem de context weg naar een daarvoor voorzien stuk van het geheugen, en het proces gaat over van de "actieve" toestand naar de toestand "klaar". Dit wegschrijven van de context noemt men een *context switch*. De omgekeerde operatie gebeurt wanneer het proces door de scheduler weer geselecteerd wordt om actief te worden.

Het leven van een proces bestaat uit *drie* stadia:

1. geboorte;
2. uitvoering;
3. dood.

De uitvoering van een proces kan verder opgedeeld worden in de drie stadia *klaar* ("READY"), *wachtend* ("SLEEPING"), en *actieve uitvoering* ("RUNNING") [SG94, Das01]. Een proces heeft eveneens een unieke identiteit aangegeven door zijn **PID** (*process identifier*). Een proces kan zelf kind-processen creëren, of zichzelf vervangen door een ander proces, met behulp van een "*fork*" instructie (zie `man fork`) of een "*clone*" (`man clone`). Kind-processen hebben andere PIDs, indien het proces zichzelf vervangt behoudt het zijn PID.

In multi-tasking systemen zoals GNU/Linux zijn er verschillende processen gelijktijdig in uitvoering. Een proces in uitvoering heeft CPU-tijd nodig om zijn instructies uit te voeren, een deel geheugen om de nodige data in te bewaren en het uitvoerbare gedeelte te laden, en afhankelijk van het doel van het proces ook andere resources zoals I/O.

In het eerste deel van dit hoofdstuk bekijken we hoe processen worden gecreëerd en behandeld door GNU/Linux. We starten op het niveau van de systeemadministratie. Daarna kijken we hoe we met behulp van `fork`, `exec`, `wait` en de *signals* interface zelf processen kunnen implementeren.

11.2 Het oerproces

init is het oerproces:[1] het proces dat als eerste opstart, en waar alle andere processen rechtstreeks of onrechtstreeks kinderen van zijn. Het proces **init** wordt opgestart als eerste stap na het booten van de Linux kernel. Als eerste proces heeft init als PID **1**. De hoofdtaak van init bestaat er uit andere processen te creëren: zo zal het eerst in het bestand `/etc/inittab` kijken en de daar beschreven processen opstarten.

[1] Zie `man init` voor meer informatie.

Figuur 11.1: Van init tot shell.

Het is bijvoorbeeld belangrijk dat er een proces opgestart wordt voor het inloggen van gebruikers. Dit gebeurt als volgt: `init` creëert een **getty** proces. Dit proces "beheert" de console (een voorbeeld van een zogenaamd "tty device," waar "tty" staat voor de mechanische "teletype," de oeroude voorgangers van de scherm-terminals): het print onder andere via de console uitvoer, en parst wat de gebruiker intypt. `getty` opent zelf nog een andere proces: het **login** proces. `getty` leest de login-naam die de gebruiker intypt en start het `proces` op, met als argument de login-naam die de gebruiker ingetypt heeft. `login` zal het paswoord van de gebruiker opvragen en controleren. Nadat de gebruiker met succes inlogt op het systeem, krijgt hij een **shell** proces. Figuur 11.1 stelt de procedure grafisch voor.

Opgave

1. Stel: je hebt je root paswoord verloren, maar wil toch toegang met beheerders rechten tot je systeem. Zoek op hoe je het initiële proces dat de kernel creëert kan veranderen om toegang tot het systeem te krijgen, zonder hoeven in te loggen. **tip**: bekijk de man-pagina van bootparam.

11.3 `ps`, de proces-administratie

ps is een applicatie die de status weergeeft van alle processen die momenteel in uitvoering zijn. Je ziet al die processen met het bevel `ps -A`. Wil je enkel je eigen processen zien dan voer je gewoon `ps` of `ps -U jouwloginnaam` uit. Een voorbeeld van de output van `ps -U kris` ("kris" is de login-naam van een gebruiker) zie je in figuur 11.2. Let op de hoofdingen van de kolommen in de uitvoer getoond in figuur 11.2. Deze geven aan welke informatie elke kolom voorstelt.

Een volledige overzicht van *alle* processen krijg je met `ps -aux`. Een verkort overzicht van de uitvoer van zo'n `ps -aux` zie je in figuur 11.3.

Door middel van de optie "-f" kan `ps` ook de hierarchie van processen tonen, figuur 11.4.

Het bevel `top` geeft een overzicht van de actieve processen dat elke seconde wordt geactualiseerd, in plaats van de momentopname die `ps` levert. Meer informatie vind je in de man pagina's van top: `man top`. Op vele systemen is ook een grafische versie beschikbaar van top, zoals bijvoorbeeld `gtop` in de GNOME desktop.

```
  PID TTY          TIME CMD
 1276 ?        00:01:06 enlightenment
 1306 ?        00:00:00 root-tail
 1316 ?        00:00:00 E-Exec.epplet
 1318 ?        00:00:06 E-Cpu.epplet
 1320 ?        00:00:05 E-NetFlame.eppl
 1345 ?        00:00:02 oafd
 1351 ?        00:00:01 wombat
 1357 ?        00:00:00 bonobo-moniker-
 1387 ?        00:00:00 evolution-alarm
 9579 ?        00:00:03 xterm
 9580 pts/1    00:00:00 bash
 9806 ?        00:00:33 gkrellm
 9810 ?        00:00:00 gkrellm
10520 ?        00:00:00 run-mozilla.sh
10525 ?        00:00:32 mozilla-bin
10532 ?        00:00:00 mozilla-bin
10533 ?        00:00:00 mozilla-bin
10534 ?        00:00:00 mozilla-bin
10535 ?        00:00:00 mozilla-bin
13784 ?        00:00:00 mozilla-bin
13801 ?        00:00:00 xterm
13802 pts/0    00:00:00 bash
15322 pts/0    00:00:00 make
15326 pts/0    00:00:00 make
15327 pts/0    00:00:00 sh
15603 ?        00:00:03 gvim
23867 ?        00:00:00 gkrellm
25033 pts/0    00:00:00 sh
25034 pts/0    00:00:00 make
25035 pts/0    00:00:00 sh
25039 pts/0    00:00:00 sh
25040 pts/0    00:00:00 make
25277 pts/0    00:00:08 perl5
25280 pts/1    00:00:00 ps
```

Figuur 11.2: De output van `ps -U kris`

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.2   1356   528 ?        S    10:05   0:04 init [5]
root         2  0.0  0.0      0     0 ?        SW   10:05   0:00 [keventd]
root         3  4.4  0.0      0     0 ?        SW   10:05   9:52 [kapm-idled]
root         4  0.0  0.0      0     0 ?        SW   10:05   0:02 [kswapd]
root         5  0.0  0.0      0     0 ?        SW   10:05   0:00 [kreclaimd]
root         6  0.0  0.0      0     0 ?        SW   10:05   0:00 [bdflush]
root         7  0.0  0.0      0     0 ?        SW   10:05   0:00 [kupdated]
root         8  0.0  0.0      0     0 ?        SW<  10:05   0:00 [mdrecoveryd]
root      1164  0.0  0.0      0     0 ?        SW   10:05   0:00 [scsi_eh_0]
   ...
root      1169  0.0  0.1   1324   448 tty1     S    10:05   0:00 /sbin/mingetty tty1
root      1170  0.0  0.1   1324   448 tty2     S    10:05   0:00 /sbin/mingetty tty2
root      1171  0.0  0.1   1324   448 tty3     S    10:05   0:00 /sbin/mingetty tty3
  ...
kris     13801  0.0  1.1   5312 2848 ?        S    13:21   0:01 xterm -ls
kris     13802  0.0  0.6   2664 1616 pts/0    S    13:21   0:00 -bash
kris     15603  0.5  2.6 10544 6772 ?        S    13:22   0:07 gvim processes-threads.tex
kris     25918  0.0  1.7 14288 4576 ?        S    13:44   0:00 gkrellm
kris     25920  0.0  1.7 14288 4576 ?        S    13:44   0:00 gkrellm
kris     25921  0.0  1.7 14288 4576 ?        S    13:44   0:00 gkrellm
kris     25922  0.0  0.2   2600   756 pts/1    R    13:44   0:00 ps aux
```

Figuur 11.3: De uitvoer van `ps -aux`.

```
 PID TTY      STAT   TIME COMMAND
1201 pts/1    S      0:00 -bash
2532 pts/1    R      0:00  \_ ps af
 354 pts/0    S      0:00 -bash
2521 pts/0    S      0:00  \_ xdvi.bin -name xdvi Linux-2002-v2.1
2524 pts/0    S      0:00  |   \_ gs -sDEVICE=x11
2528 pts/0    S      0:02  \_ vim processes-threads.tex
```

Figuur 11.4: (Een deel van) de uitvoer van `ps -af`.

11.4 Systeem-processen en daemons

In een besturingssysteem draaien heel wat processen zonder dat de gebruiker ze expliciet zelf heeft opgestart. Sommige van deze diensten roep je *onrechtstreeks* aan wanneer je je eigen programma's opstart. Andere zijn "achtergrond-processen," die reeds bestaan van bij het opstarten van het systeem, maar daarna meestal slapend hun tijd hebben doorgebracht, wachtend op een aanvraag voor één van de diensten die ze kunnen leveren. In UNIX heten die achtergrond-processen ook *daemons*, vandaar dat hun naam vaak eindigt met de letter "`d`." Je ziet er ongetwijfeld een paar in de uitvoer van een `ps aux` bevel.

Welke daemons beschikbaar zijn, en welke scripts nodig zijn om ze op te starten, te stoppen en te configureren, kom je te weten in de `/etc/init.d` directory. De meeste van die scripts aanvaarden de argumenten `start`, `stop`, `status` en `restart`. Het is natuurlijk niet de bedoeling dat je die scripts telkens bij opstarten van je computer zelf aanroept. Het `init` proces doet dat uiteindelijk voor jou, omdat het de scripts in de `/etc/rcX.d` (of `/etc/rc.d/rcX.d`) directory oproept, waarbij "`X`" in Linux staat voor een getal tussen 0 en 6, het zogenaamde *runlevel*. Je kan het besturingssysteem namelijk opstarten op verschillende niveaus, met meer en meer functionaliteit naarmate je een hoger runlevel kiest. Het runlevel bepaal je ofwel met een parameter bij het opstarten van je systeem, ofwel in het `/etc/inittab` bestand. (Zoek de lijn met de parameter `initdefault`.)

In elke `/etc/rcX.d` directory vind je scripts waarvan de naam begint met de letter "`K`" of met de letter "`S`," gevolgd door een getal tussen 0 en 99, en uiteindelijk de naam van de dienst. (Eigenlijk staan in de `/etc/rcX.d` directory geen scripts, maar *symbolische links* naar de scripts in de "`/etc/init.d`" directory.) De getallen geven de *volgorde* weer waarin het `init` proces de scripts oproept: de "`K`"-scripts worden opgeroepen bij het *verlaten* van een runlevel om diensten te "killen," de "`S`"-scripts bij het *betreden* van een runlevel, om diensten te "starten." Je kan ook nog na opstarten van het systeem van runlevel veranderen; dit doe je met het `telinit` bevel.

Een lijst van enkele veelgebruikte systeem-diensten vind je in tabel 11.1.

inetd, xinetd	netwerk-diensten.
atd, cron, anacron	diensten die op bepaalde tijdstippen moeten lopen.
automount	automatisch mounten van wegneembaar geheugen (schijven, CDs, floppies).
sshd, telnetd ftpd, httpd	ondersteuning van verschillende communicatie-protocols over een netwerk-verbinding.
lpd	printer-diensten.
syslogd, klogd	loggen van debug-boodschappen van programma's en kernel-taken.

Tabel 11.1: Veelgebruikte systeem-diensten

11.5 Processen programmeren

Deze sectie gaat dieper in op het programmeren van processen en communicatie tussen processen.

11.5.1 Proces creatie: `system`, `exec` en `fork`

system We starten met een eenvoudig voorbeeld van de functie `system` uit de header `stdlib.h`. De handtekening van `system` ziet er als volgt uit:

```
int system (const char * nieuwproces);
```

De functie `system` neemt het argument **nieuwproces** en laat het uitvoeren door een shell door "`/bin/sh -c nieuwproces`" op te roepen. De functie zal pas terugkeren als deze aanroep terugkeert. Dit is ineens het zwakke punt van system: het is een **blocking** functie aanroep. Een voorbeeldje ervan vind je in listing 11.1

Listing 11.1: proces1.c

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
  printf("Wil je een echo?\n");
  system("echo \"proces1.c vraagt: Wil je een echo?\"");
  printf("zo zie je maar\n");
  return 0;
}
```

als men het volgende voorbeeld uitvoert, listing 11.2 krijg je te zien dat `system` een *kind-proces* opstart van het proces: `system` wordt in de meeste UNIX systemen dan ook geïmplementeerd met behulp van `fork` (sectie 11.5.1) en `exec` (sectie 11.5.1). Je krijgt namelijk te zien dat "ps -faux" een kind-proces is van proces2, zoals getoond in figuur 11.5. Merk op dat de `printf`-oproepen die voor de `system`-aanroep voorkomen ook wel pas na de `system`-output getoond worden, gewoon om dat het bevel dat als argument aan `system` meegegeven wordt sneller output produceert. Andersom kan echter *niet* op de manier getoond in listing 11.1 en listing 11.2. Hier kan een mouw aan gepast worden door telkens een '&' achter het bevel te plaatsen; bijvoorbeeld `ps -faux &` in plaats van `ps -faux`.

Listing 11.2: proces2.c

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
```

```
system zal wachten
USER       PID TTY   STAT START   TIME COMMAND
root         1 ?     S    22:09   0:04 init [5]
root         2 ?     SW   22:09   0:00 [keventd]
....
kris      1887 S   23:25   0:04 gvim proces-programmeren.tex
kris      1982 ?     S    23:47   0:00  \_ /bin/bash -c (lists/proces2) >/tmp/v326724/0 2>&1
kris      1999 ?     S    23:47   0:00      \_ lists/proces2
kris      2000 ?     R    23:47   0:00          \_ ps -faux
...
system heeft gewacht
```

Figuur 11.5: Gedeeltelijk output van proces2

```
  printf("system zal wachten\n");
  system("ps -faux");
  printf("system heeft gewacht\n");
  return 0;
}
```

Oefeningen

1. Schrijf twee processen die elkaar oproepen met behulp van system. Bekijk met `ps -faux` en `top` wat er gebeurt op het systeem. Stop de processen met behulp van het `kill` bevel of de toetsencombinatie Ctrl-C. Bekijk ook het geheugen verbruik.

2. Pas de voorbeeldjes in listing 11.1 en listing 11.2 aan door '&' toe te voegen aan het `system` argument.

exec In tegenstelling tot `system` creëert de functie `execve` geen kind proces, maar *vervangt het huidige proces*. De handtekening van `execve` ziet er als volgt uit:

```
int execve (const char *filename, char *const argv [], char *const envp[]);
```

Deze functie kan je vinden in `unistd.h`. Merk op dat deze functie *niet* terugkeert indien ze succesvol uitgevoerd wordt: als deze functie slaagt wordt het proces van waaruit de functie uitgevoerd wordt immers vervangen door een ander proces. Het geheugen en de resources die toegekend werden aan het originele proces worden nu helemaal in beslag genomen door het proces dat met behulp van `execve` gestart was. Het eerste argument van `execve`: `filename` geeft aan welk proces het huidige proces zal vervangen. Het tweede argument: `argv` is een rij van argumenten die

we aan het nieuwe proces willen meegeven. Merk op dat het eerste argument de naam van het proces zelf moet zijn. Met andere woorden, de lijst die men meegeeft in `argv` moet dezelfde zijn als een C-programma die lijst als argument van de main functie (`int main(int argc, char *argv[])`) zou ontvangen. Het laatste argument: `envp` is bedoeld om informatie voor de omgeving mee te geven door middel van "sleutel=waarde" paren. Je kan de mogelijke sleutels hiervoor vinden in de man-pagina van `environ`.

Listing 11.3 toont hoe de functie kan gebruikt worden en de output ervan zie je in figuur 11.6.

Listing 11.3: execve1.c

```
#include <stdio.h>
#include <unistd.h>

int main(int argc, char* argv[])
{
  char *commando = "/bin/echo";
  char *argumenten[] = {"echo", "\"execve1.c vraagt: Wil je een echo?\"", 0 };
  char *omgeving[] = {"PATH=/bin", 0};

  printf("Wil je een echo?\n");
  execve(commando, argumenten, omgeving);
  printf("zo zie je maar\n");
  return 0;
}
```

Merk op dat de laatste `printf` *niet* uitgevoerd wordt. Het proces is dan immers reeds vervangen door het proces `echo`. Om dit wat duidelijker te maken, beschouw een variatie van dit voorbeeld in listing 11.4 met de output getoond in figuur 11.6. Het proces werd vervangen bij de eerste `execve`, zodat de lus nooit een keer kon uitgevoerd worden.

Listing 11.4: execve2.c

```
#include <stdio.h>
#include <unistd.h>

int main(int argc, char* argv[])
{
  char *commando = "/bin/echo";
  char *argumenten[] = {"echo", "\"execve1.c vraagt: Wil je een echo?\"", 0 };
  char *omgeving[] = {"PATH=/bin",0};

  while(1)
  {
    printf("Wil je een echo?\n");
    execve(commando, argumenten, omgeving);
    printf("zo zie je maar\n");
```

```
Wil je een echo?
"execve1.c vraagt: Wil je een echo?"
```

Figuur 11.6: De output van execve1.c (listing 11.3)

```
  }
  return 0;
}
```

Rond `execve` is een hele familie functies opgebouwd, de `exec`-functies:

- `int execl( const char *path, const char *arg, ...);`[2]
- `int execlp( const char *file, const char *arg, ...);`
- `int execle( const char *path, const char *arg , ..., char * const envp[]);`
- `int execv( const char *path, char *const argv[]);`
- `int execvp( const char *file, char *const argv[])`

Meer informatie over over deze functies vind je in de man-pagina's.

Oefeningen

1. Herwerk de code listing 11.3: vervang execve door execlp. Raadpleeg de man pagina's voor de nodige informatie.
2. Schrijf twee processen die elkaar oproepen met een funtie uit de exec-familie. Bekijk met `ps -faux` en `top` wat er gebeurt op het systeem. Stop de processen met behulp van het `kill` bevel of de toetsencombinatie Ctrl-C. Bekijk ook het geheugen verbruik.

fork De derde mogelijkheid om processen te creëeren is de `fork` functie oproep. `fork` maakt een kind-proces aan dat alleen verschilt van het originele proces verschilt in PID en de resources die toegewezen zijn aan het proces. Met andere woorden: het nieuwe proces is een exacte *kopie* van de adresruimte van het ouderproces, afgezien van de PID en de toegewezen resources. `fork` geeft dan de PID van het kind-proces terug. In het kind-proces zelf zal `fork` 0 teruggeven, zodat men in de code kan herkenne wanneer men met een kind-proces te maken heeft. De handtekening van `fork` ziet er als volgt uit:

[2] In C kan je `...` gebruiken in de functie handtekening om aan te geven dat de functie een willekeurig aantal argumenten kan ontvangen. In dit geval wordt een NULL pointer als een sentinel gebruikt om het einde van de argumenten-lijst aan te geven. `execve` vereist dat het laatste argument een NULL pointer is voor een goede werking.

```
Hier splits ik mij
ouder proces zegt: kind proces met PID 3300 aangemaakt
kind proces spreekt tot u
```

Figuur 11.7: De output van fork1.c (listing 11.5)

```
pid_t fork(void);
```

Deze functie kan gebruikt worden mits de headers `sys/types.h` (voor `pid_t`) en `unistd.h` geinclude worden.

In veel gevallen zal het ouder-proces dadelijk na de fork call ofwel wachten tot het kind-proces beëindigd is (met behulp van `wait`), ofwel zichzelf vervangen door een ander proces met behulp van een `exec` functie oproep. Een eerste voobeeld van het gebruik van fork is te vinden in listing 11.5, de output ervan is te vinden in figuur 11.7. Merk op dat fork non-blocking is als je dit voorbeeld zelf probeert.

Listing 11.5: fork1.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main(int argc, char* argv[])
{
  pid_t pnumber;
  pnumber=-2;
  printf("Hier splits ik mij\n");
  pnumber = fork();
  switch(pnumber){
    case -1 :
      printf("fork() is mislukt\n");
      break;
    case 0 :
      printf("kind proces spreekt tot u\n");
      break;
    default :
      printf("ouder proces zegt: kind proces met PID %d aangemaakt", pnumber);
  }
  return 0;
}
```

Soms dient het ouder-proces te wachten totdat de uitvoering van de kinderen eindigt. Hiervoor wordt gebruik gemaakt van de `wait` functie, gedefinieerd in de header `sys/wait.h`. De `wait` functie wacht totdat een van de kind-processen is beëindigd, en geeft de PID terug van het kind-proces dat afgelopen is.

In een uitgebreider voorbeeld van het gebruik van fork, implementeren we een äccounting system"waarbij een programma als een soort prompt zal werken die

informatie over de uitgevoerde commando's bijhoudt in een bestand. Merk op dat `fork` veel gebruikt wordt om nadien het kindproces te vervangen door een andere proces image met behulp van een van de `exec`-functies.

11.6 IPC: inter-proces communicatie

Sockets, pipes (FIFOs), message queues, gedeeld geheugen, ... "Everything is a file" in UNIX, dus elk van deze communicatie-primitieven heeft een toegang via de speciale "files" in de `/dev` directory. De communicatie verloopt voor de deelnemende processen (of threads) door te schrijven en te lezen uit deze bestanden.

11.7 Threads

11.7.1 Inleiding en definitie

UNIX kende oorspronkelijk alleen maar processen. Maar meer en meer programma's maken gebruik van een verwante techniek om code uit te voeren: *threads*. Een thread noemt men vaak ook een "lichtgewicht proces," omdat éénzelfde proces verscheidene threads kan herbergen, en die threads *delen* de page table, de heap, en de file descriptors. Een context switch voor een thread is dus veel lichter dan die voor een proces. In Linux is het onderscheid tussen proces en thread echter niet meer zo duidelijk: de `clone` functie laat toe te kiezen welke stukken van de context gedeeld worden door de taak die de functie oproept, en de taak die aangemaakt wordt. Het hele spectrum tussen "zwaar proces" en "uitermate lichte thread" is dus mogelijk.

De lichtheid van de context switch is echter niet de enige, en zeker niet de belangrijkste reden waarom threads zijn ingevoerd. Andere voordelen van threads zijn dat ze snel aangemaakt en vernietigd kunnen worden, en dat ze heel snel met elkaar gegevens kunnen uitwisselen omdat ze dezelfde geheugenruimte binnen één proces delen. (Dit brengt natuurlijk ook nadelen met zich mee: een fout in één thread kan alle threads laten verongelukken!) Door die snelle creatie en communicatie zijn threads heel geschikt binnen "server"-toepassingen: klanten van een server sturen een boodschap met een verzoek om een bepaalde dienst voor hen uit te voeren, en de server heeft een thread om die verzoeken te ontvangen, en elk nieuw verzoek uit te delen naar een thread die de eigenlijke uitvoering moet realiseren. Threads zorgen dus voor een grotere scaleerbaarheid en respons-tijd dan een server die uit één enkel proces bestaat.

Zowat alle moderne programma's die een grote en wisselende belasting van diensten "in parallel" moeten aanbieden maken gebruik van threads. De webserver Apache[3], de Linux kernel zelf, gegevensbanken, browsers, enz. Figuur 11.8 toont

[3]Het threaded model voor servers is zeker niet de standaard. Er bestaat nog veel discussie over het nut van threads voor server applicaties. Meer informatie vind je op `http://httpd.apache.org/docs/misc/perf-tuning.html#preforking`

```
10520 ?        00:00:00 run-mozilla.sh
10525 ?        00:00:32 mozilla-bin
10532 ?        00:00:00 mozilla-bin
10533 ?        00:00:00 mozilla-bin
10534 ?        00:00:00 mozilla-bin
10535 ?        00:00:00 mozilla-bin
13784 ?        00:00:00 mozilla-bin
```

Figuur 11.8: Verschillende threads binnen het Mozilla programma.

in de uitvoer van `ps` de verschillende threads die het programma `Mozilla` op een bepaald ogenblik gebruikt tijdens een surf-sessie: elke thread heeft in Linux zijn eigen proces PID, omdat Linux, zoals vroeger vermeld, eigenlijk geen verschil maakt tussen processen en threads.

11.7.2 POSIX threads

In deze sectie geven we een overzicht en voorbeelden van het gebruik van POSIX (Portable Operating System Interface) threads: het standaard threading model gebruikt op de meeste Unix systemen. We laten in de programmeertaal C zien hoe men imperatieve programma's multi-threaded kan maken. Tevens kijken we naar het effect van het gebruik van POSIX threads op de uitvoeringstijd van programma's. De lezer zal merken dat programmeren met threads een andere denkwijze vereist en een groter gevaar tot neveneffecten met zich meebrengt.

De nodige functies om met POSIX threads te werken vindt men in de pthread bibliotheek. De functies die door deze bibliotheek aangeboden worden, zijn gedefinieerd in de header `pthread.h`. Dit wil zeggen dat we tegen de pthread bibliotheek moeten linken bij het compileren; dit kan door de optie `-lpthread` mee te geven als een argument van gcc bij het compilatieproces.

We bekijken een voorbeeldje van het gebruik van threads in listing 11.6. De bedoeling is dat we de functie `foo` parallel laten uitvoeren binnen hetzelfde programma. Dit kan door meerdere threads aan te maken. Stel dat we de functie `foo` twee keer tegelijkertijd kunnen uitvoeren, dan maken we twee threads aan hiervoor. Een thread, voorgesteld door het type `pthread_t`, kan aangemaakt worden door de functie `pthread_create`. Deze functie heeft de handtekening: `int pthread_create(pthread_t *thread, pthread_attr_t *attr, void *(*tfunctie)(void *), void *args)`. Deze functie neemt als parameters een referentie naar de thread die gecreëerd dient te worden, de thread attributen, de functie die door de thread dient uitgevoerd te worden en de parameters die aan de functie doorgegeven dienen te worden. `pthread_t` is in feite de thread identifier, de handle die naar de eigenlijke thread verwijst. De functie die door de thread dient uitgevoerd te worden moet voldoen aan een bepaalde signatuur: als return waarde een void pointer en als parameter een void pointer: `void *thread_function(void *)`. Op deze manier kunnen eender welke argumenten doorgegeven worden aan de

functie, en eender welk return type teruggegeven worden. Een voorbeeld van parameter passing naar functies die op hun eigen thread uitgevoerd worden kan de lezer vinden in listing 11.7 (let wel, de syntax is correct in het voorbeeld, de semantiek echter niet).

Listing 11.6: threads1.c

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

void *foo(void *param){
   int k=0;
  while(1)
    printf("b");
  return NULL;
}

int main(int argc, char* argv[]){
  pthread_t dedraad,nogeendraad;
  int i=0;
  if(pthread_create(&dedraad, NULL, foo, NULL)){
   printf("Fout bij het aanmaken van de tweede thread");
   abort();
  }
  if(pthread_create(&nogeendraad, NULL, foo, NULL)){
    printf("Fout bij het aanmaken van de derde thread");
    abort();
  }

  while(1)
    printf("a");

  if(pthread_join(dedraad, NULL)){
   printf("Fout bij het joinen van de tweede thread");
   //abort();
  }
  if(pthread_join(nogeendraad, NULL)){
    printf("Fout bij het joinen van de derde thread");
    abort();
  }

  exit(0);
}
```

Het programma, beschreven in listing 11.6, kan je compileren met gcc als volgt:`gcc -D_REENTRANT threads1.c -o threads1 -lpthread`, waarbij `-lpthread` aangeeft dat er tegen de POSIX threads bibliotheek moet gelinkt worden. `D_REENTRANT` geeft aan dat de compiler ermee rekening moet houden dat sommige stukken co-

de op hetzelfde moment *meerder keren* in uitvoering kunnen zijn. Merk op dat als het programma uitgevoerd wordt, er **niet twee, maar drie** threads gebruikt worden. De uitvoering van het programma zelf (de main-method) heeft ook zijn eigen thread. Tijdens de uitvoering kan je merken dat de scheduler de threads afwisselend aan bod laat komen voor een bepaalde tijdspanne.

Opgave

1. Het starten en de volgorde van uitvoering van threads is niet-deterministisch. Daarom is het makkelijk onbewust fouten te maken. In listing 11.7 vind je zo een programma dat een ernstige fout bevat. Zoek deze fout en verklaar waarom de code verkeerd is bij gebruik van meerdere threads. **Tips**:

   - De main-method heeft ook zijn eigen thread
   - De threads stoppen op verschillende momenten

Listing 11.7: Ongewenste neveneffecten

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

struct theparam{
  char k;
};


void *foo(void *param){
   struct theparam* blaai=(struct theparam *)param;
  int i;
  for(i=0; i<1000;i++)
    printf("[%c]", blaai->k);
  return NULL;
}

int main(int argc, char* argv[]){
  pthread_t dedraad,nogeendraad, nogene;
  int i=0;
  struct theparam p1,p2,p3;
  p1.k='b';
  p2.k='c';
  p3.k='d';

  if(pthread_create(&dedraad, NULL, foo, &p1)
     || pthread_create(&nogeendraad, NULL, foo, &p2)
    || pthread_create(&nogene, NULL, foo, &p3)){
    printf("Fout bij het aanmaken van een van de threads");
    abort();
```

```
  }

  for(i=0;i<250;i++)
    printf("a");

  exit(0);
}
```

2. Schrijf een programma dat de functie `void *fac(void *arg)` dat iteratief de faculteit van een getal berekent. Het getal waarvan de faculteit berekent dient te worden wordt meegegeven in `void *arg`, via de traditionele thread creatie functie: `pthread_create`. De gebruiker kan interactief getallen ingeven, waarvan de faculteiten worden berekend. De functie `fac` wordt berekend op zijn eigen thread. Voeg `sleep(3)` toe aan de fac functie alvorens de faculteit te berekenen.
   tips: Inlezen doe je met `fgets`. `atoi` zet een char array om naar een integer.

3. Implementeer het probleem van de concurrente uitvoering van een producer en consumer routine. Maak hiervoor een producer functie en een consumer functie, die elk via een eigen thread opgestart worden. De `buffer` en `teller` worden globaal gedeclareerd. De pseudo code van de producer ziet er als volgt uit:

```
while(TRUE)
    produceer een item
    while(teller==MAX)
      ;
    buffer[in] = item;
    in = (in+1)%MAX;
    teller += 1;
```

   . De pseudo code voor de consumer zie er als volgt uit:

```
while(TRUE)
    while(teller==0)
      ;
    item = buffer[uit];
    uit = (uit+1)%MAX;
    teller -= 1;
    doe iets met item
```

   Welke problemen zie je?

11.7.3 Joining threads

Een belangrijke functie die niet in de vorige sectie besproken is, is de functie `int pthread_join(pthread_t thread, void **return_waarde)`. Deze functie maakt het mogelijk op de thread meegegeven als eerste argument te wachten. Het tweede argument van de functie vangt de return waarde van de thread op. Met andere woorden: waar `pthread_create` een thread creëert in het programma, daar zorgt `pthread_join` dat de twee threads (de main thread en de afgesplitste thread) weer samengevoegd worden.

Merk op dat het voorbeeld uit listing 11.7 wel in orde zou zijn indien we `pthread_join` gebruiken. Bovendien stelt deze functie ons in staat om de return waarde van een threaded functie op te vangen. Als simpel voorbeeld passen we listing 11.7 aan en gebruiken `pthread_join` om de threads samen te voegen alvorens het programma eindigt.

Listing 11.8: `pthread_join` voegt de threads samen

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

struct theparam{
  char k;
};


void *foo(void *param){
   struct theparam* blaai=(struct theparam *)param;
  int i;
  for(i=0; i<1000;i++)
    printf("[%c]", blaai->k);
  return param;
}

int main(int argc, char* argv[]){
  pthread_t dedraad,nogeendraad;
  void *resultaat, *resultaat2;
  int i=0;
  struct theparam p1,p2;
  p1.k='b';
  p2.k='c';

  if(pthread_create(&dedraad, NULL, foo, &p1)
     || pthread_create(&nogeendraad, NULL, foo, &p2)){
    printf("Fout bij het aanmaken van de een thread");
    exit(EXIT_FAILURE);
  }
```

```
  for(i=0;i<250;i++)
    printf("a");

  if(pthread_join(dedraad, &resultaat)||
    pthread_join(nogeendraad, &resultaat2)){
    printf("Kan threads niet samenvoegen");
    exit(EXIT_FAILURE);
  }
  printf("Resultaat 1:%c, Resultaat 2:%c",
       ((struct theparam *)resultaat)->k,
      ((struct theparam *)resultaat2)->k);

  exit(EXIT_SUCCESS);
}
```

11.7.4 Semaforen en Mutexen

In de eerste oefeningen omtrent threads hebben we gezien dat er ongewenste neveneffecten opduiken, daar threads dezelfde resources (ondermeer geheugenruimte) delen. Om de toegang tot deze gedeelde resources te synchroniseren, kan men gebruik maken van twee verschillende technieken: **semaforen** en **mutexen**. Mutex staat voor *mutual exclusion*, en kan in termen van semaforen geïmplementeerd worden. Een mutex zorgt ervoor dat slechts één thread per keer de gedeelde resource kan aanspreken. Een semafoor is in feite een teller die de toegang tot een stuk code bewaakt. Het is duidelijk dat een mutex als een semafoor met een binaire waarde kan gezien worden. In deze sectie zullen we ons concentreren op semaforen. Kiezen tussen het gebruik van een semafoor of mutex hangt veelal af van het probleem dat men dient op te lossen. Met behulp van semaforen, kan men makkelijk zelf mutexen aanleren en gebruiken. De volgende bibliotheken zijn meestal beschikbaar voor semaforen op GNU/Linuxsystemen:

Mutexen en Semaforen voor threads : POSIX Realtime Extension (`semaphore.h`);

Semaforen voor processen : System V semaforen;

We zullen enkel de eerste gebruiken, deze zit dan ook in de bibliotheek libpthread.

In `semaphore.h` wordt de struct `sem_t` gedefinieerd. Instanties van deze struct zullen gebruikt kunnen worden als semafoor; als de "teller". Er zijn vier functies uit `semaphore.h` die belangrijk zijn teneinde gebruik te kunnen maken van semaforen:

`sem_init` : Deze functie wordt gebruikt om een semafoor aan te maken. De volledige handtekening van deze functie ziet er als volgt uit:
`int sem_init(sem_t *s, int share, unsigned int value)`
Het argument `share` moet steeds 0 zijn[4]. Het argument `value` geeft de

[4]Een niet 0 waarde betekent dat de semafoor niet enkel tussen threads, maar ook tussen processen gedeeld kan worden. Deze functionaliteit wordt echter nog niet ondersteund door alle implementaties.

initiële waarde van de semafoor weer, in de meeste gevallen is dit ook 0.

`sem_post` : Deze functie verhoogt de semafoor. Dit is een **atomaire** functie.De volledige handtekening van deze functie ziet er als volgt uit:
`int sem_post(sem_t *s)`

`sem_wait` : Deze functie wacht totdat de semafoor minstens 1 is. Met andere woorden, totdat er minstens één keer `sem_post` opgeroepen is. Als dit het geval is zal de uitvoering *niet* stoppen bij `sem_post`. Indien de semafoor echter 0 is, zal de functie wachten totdat er minstens één semafoor gepost wordt door `sem_post`. `sem_wait` zal de semafoor verlagen indien de functie slaagt. De volledige handtekening van deze functie ziet er als volgt uit:
`int sem_wait(sem_t *s)`

`sem_destroy` : Deze functie ruimt de semafoor op die met behulp van `sem_init` gecreëerd was. De volledige handtekening van deze functie ziet er als volgt uit:
`int sem_destroy(sem_t *s)`

Als voorbeeld van een semafoor tonen we hoe zogenaamde "dirty-read" conflicten kunnen voorkomen. We spreken van een dirty-read als een variabele gelezen wordt, terwijl er naar dezelfde variabele geschreven wordt. Een probleem dat trouwens zeer belangrijk is bij vele systemen, denk maar aan concurrente toegang tot een database. Het voorbeeld in listing 11.9 imiteert een banktransactie, waar "bedragA" het aantal euro is dat op de bankrekening A staat, en "bedragB" het bedrag op bankrekening B. Er zijn twee threads die gelijktijdig toegang hebben tot deze bankrekeningen (in de vorm van de gedeelde variabelen `bedragA` en `bedragB`). De functie `bankier1` verhoogt en verlaagt regelmatig beide bankrekeningen, De functie `bankier2` stort regelmatig een bedrag van de ene naar de andere bankrekening. De semaforen vermijden dat een thread A eerst de waardes leest, waarna een andere thread B de waardes verandert, en waarna thread A dan weer de waardes verandert uitgaande van de originele waardes die hij gelezen had *voor* de actie van thread B. Dit zou betekenen dat de actie van B in feite ongedaan gemaakt werd. We lossen dit probleem op door `bedragA` en `bedragB` te beschermen door semaforen.

Listing 11.9: Voorbeeld van het gebruik van semaforen

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <semaphore.h>


double bedragA, bedragB;
sem_t safetrans;
```

```
void *bankier1(void *arg)
{
  int i=0, teken=1;
  double a,b, verhoging;
  for(i=0;i<10;i++)
  {
    sleep(1.1);
    sem_wait(&safetrans);
      a=bedragA;
      b=bedragB;
      verhoging= teken * (random()%100);
      teken = 0-teken;
      a+=verhoging;
      b+=verhoging;
      bedragA = a;
      bedragB = b;
    sem_post(&safetrans);
    printf("Bankier 1: Verhoogt met %g euro\n", verhoging);
    printf("Rekening A = %g, Rekening B = %g\n", bedragA, bedragB);
  }
}

void *bankier2(void *arg)
{
  int i=0,teken=1;
  double a,b, verplaatsing;
  for(i=0;i<10;i++)
  {
    sleep(1.2);
    sem_wait(&safetrans);
      a=bedragA;
      b=bedragB;
      verplaatsing= teken * (random()%100);
      teken = 0-teken;
      a+=verplaatsing;
      b-=verplaatsing;
      bedragA = a;
      bedragB = b;
    sem_post(&safetrans);
    printf("Bankier 2: %g euro verplaatst van A naar B\n", verplaatsing);
    printf("Rekening A = %g, Rekening B = %g\n", bedragA, bedragB);
  }
}

int main(int argc, char* argv[])
{

  pthread_t ent1, ent2;
```

```
  void *tresult;
  bedragA=100, bedragB=100;
  if(sem_init(&safetrans,0,0)){
    printf("Kan de semafoor niet aanmaken\n");
    exit(EXIT_FAILURE);
  }
  if((pthread_create(&ent1, NULL, bankier1, NULL)||
    pthread_create(&ent2, NULL, bankier2, NULL))!=0)
  {
    printf("Kan de threads niet aanmaken\n");
    exit(EXIT_FAILURE);
  }

  printf("Voer de transacties uit...\n");
  sem_post(&safetrans);
  if((pthread_join(ent1, &tresult)||
    pthread_join(ent2, &tresult))!=0){
    printf("Kan de threads niet terug samenvoegen\n");
    exit(EXIT_FAILURE);
  }
  printf("Rekening A = %g, Rekening B = %g\n", bedragA, bedragB);
  exit(EXIT_SUCCESS);
}
```

Opgave

1. Zoek op en leg uit in termen van semaforen: *spin-lock*
2. Zoek op en leg uit: *clone()*.
3. Verwijder de semaforen uit het programma weergegeven in listing 11.9. Leg uit waar de uitvoering mis kan lopen. Leg de invloed van de volgende statements in verband met mogelijke dirty-reads uit:

   ```
   a=bedragA;
   b=bedragB;
   ...
   bedragA=a;
   bedragB=b;
   ```

4. Los het probleem van de concurrente uitvoering van een producer en consumer routine met behulp van semaforen (vorige oefeningen reeks).
5. Implementeer het *dining philosophers* probleem met behulp van semaforen. Hou *geen* rekening met mogelijke *deadlocks*.

Hoofdstuk 12

De Linux kernel

12.1 Inleiding

Er bestaan verschillende definities en uiteenlopende meningen over wat men nu juist het besturingssysteem van een computer noemt. Een veel gebruikte definitie is dat het besturingssysteem het stuk software is dat altijd aanwezig is in het geheugen van de computer en dat verantwoordelijk is voor de besturing allocatie van resources voor andere programma's en eveneens instaat voor de controle van andere programma's. De software die altijd in het geheugen aanwezig is noemt men ook wel de kernel.

De Linux kernel is vrij beschikbaar voor verschillende processoren. De Linux kernel kan gebruikt worden voor onder andere de volgende processoren: Alpha, ARM, IA64, i386, MIPS, PPC en Sparc. Deze kernel is geschreven in de programmeertaal C. Sommige stukken die afhankelijk zijn van de architectuur van het systeem zijn ook in assembler geschreven.

12.2 Een Linux kernel compileren en installeren

Je kan jezelf enkel het label power-user opkleven als je zelf je kernel "stookt". Er zijn verschillende redenen om dit te doen: de performantie stijgt daar de code speciaal voor jouw processor architectuur gecompileerd wordt, nieuwere kernels ondersteunen meer hardware, bugs van oudere kernel versies worden gefixt in nieuwe versies,... Het is alsof je de motor van je auto vervangt door een beter model. GNU/Linux geeft je de mogelijkheid zelf de "motor" van je systeem kosteloos te upgraden.

Je eigen kernel maken uit de Linux broncode is een zeer leerrijke ervaring: je leert de werking van je systeem beter te doorgronden, en je kan eens een kijkje nemen in de interne werking van het besturingssysteem.

12.2.1 De kernel broncode bekomen

De broncode van de Linux kernel kan men bekomen op http://www.kernel.org. Naast code voor de kernel kan men er ook allerlei patches vinden die specifieke veranderingen aanbrengen aan de broncode van de kernel. Deze code is vrij beschikbaar; je kan er dus de code van de kernel gratis afhalen.

De broncode wordt ingepakt in een gecomprimeerd tar archief. Om de kernel zelf te bouwen uit die broncode moet je deze eerst uitpakken. In sectie 4.6 en sectie 14.2 kan je terugvinden hoe je zulke archieven kan uitpakken. De installatie loopt echter wel anders dan de installatie van normale software zoals voorgesteld in sectie 14.2.

12.2.2 De kernel patchen

Optioneel kan men de Linux kernel "patchen". Dit wil zeggen dat men wijzigingen of extra's aan de code van de Linux kernel toevoegt. Zo bestaat er bijvoorbeeld de "kdb" patch die een debugger aan de Linux kernel toevoegt. Een andere patch is de realtime-preempt patch die de Linux kernel meer geschikt maakt voor realtime toepassingen. Of de patch van Con Kolivas, die de Linux kernel tunen voor optimale performantie voor desktop gebruikers.

In feite bestaat een patch uit een diff bestand: dit is een bestand die de verschillen beschrijft tussen de originele broncode en de aangepaste broncode. Op deze manier kan de ontwikkelaar de standaard kernel afhalen van kernel.org en op deze code verder werken. Als de ontwikkelaar dan de aangepaste kernel af heeft, kan deze het verschil van zijn aangepaste versie met de originele versie berekenen met `diff`. De output van `diff` is in feite het patch bestand.

Stel dat je bijvoorbeeld de Con Kolivas patch wil uitvoeren op de kernel, dan haal je eerst de patch af. Soms is de patch gecomprimeerd met bzip2 of gzip, dan moet je natuurlijk eerst deze patch decomprimeren met de overeenkomstige tools. Bij de CK patch is dit bijvoorbeeld: `bunzip2 patch-2.6.15-ck2.bz2`. Je kan de gedecomprimeerde patch dan verhuizen naar de root dir van de Linux source code: `mv patch-2.6.15-ck2 linux/`. Nu staat de patch in dezelfde directory als de linux kernel broncode. De volgende stap is naar de directory gaan (`cd linux/`) en daarna uiteindelijk de patch toepassen op de broncode: `patch -p1 < patch-2.6.15-ck2`. `patch` is het bevel dat gebruikt wordt om een diff bestand toe te passen op de broncode. De optie `-p1` geeft aan dat patch de eerste directory die aangegeven wordt niet mag interpreteren. Met andere woorden: patch doet dit door telkens de volgende \ te verwijderen uit het pad dat aangegeven is in de diff. Als de diff de directory `linux/drivers/scsi` aangeeft, dan geeft de optie `-p0` de identieke directory en de optie `-p1` geeft `drivers/scsi`. Je kan patch ook de optie `--dry-run` meegeven die de patch in simulatie uitvoert. Alles lijkt er dus op dat de patch uitgevoerd wordt, maar in feite worden geen bestanden aangepast. Hiermee kan je vermijden dat een patch die zou falen, gedeeltelijk uitgevoerd zou worden.

Even de bevelen die we uitgevoerd hebben om de kernel te patchen op een rijtje zetten:

```
bunzip2 patch-2.6.15-ck2.bz2
mv patch-2.6.15-ck2 linux/
cd linux/
patch -p1 --dry-run < patch-2.6.15-ck2 # Als dit foutloos verloopt...
patch -p1 < patch-2.6.15-ck2 # Voer dan dit uit
```

Nu is de kernel gepatcht! Tenminste, als er geen fouten waren opgetreden.

12.2.3 De kernel configureren

Als de broncode klaar is om gecompileerd te worden, moet je nog een configuratiestap uitvoeren, zodat de kernel de functionaliteiten die het systeem nodig heeft voorziet.

Er zijn 3 verschillende manieren om deze configuratie te doorlopen. Eerst moet je naar de root directory gaan van de broncode (`linux/`). Nu kan je kiezen tussen de drie verschillende manieren: `make config`, `make menuconfig` of `make xconfig`. Deze drie manieren bieden dezelfde functionaliteit, maar telkens een andere gebruikersinterface op deze functionaliteit:

- `make config`: stap voor stap wordt elke optie van de kernel geconfigureerd. Je dient *alle* stappen te doorlopen en op alle opties expliciet een antwoord te geven. Deze configuratie gebeurt in de console.
- `make menuconfig`: er wordt in de console een interface opgebouwd[1] die je een overzicht geeft van alle opties. Hier kan je dan de nodige opties kiezen configureren.
- `make xconfig`: dit is een volledige grafische interface die je een overzicht geeft van alle opties. Hier kan je dan de nodige opties kiezen en configureren.

De meeste opties hebben drie mogelijkheden:

1. Uit
2. Compileer de optie als een module. De nodige code voor deze optie kan dan nadien met behulp van `modprobe` of `insmod` in de kernel geladen worden. Voor meer uitleg over modules zie hoofdstuk 13.
3. Compileer de code voor de optie in de kernel image zelf. Nu is de code steeds beschikbaar als de kernel geladen wordt. Nadeel is dat de kernel dan ook groter zal zijn.

Indien je de eerste keer zelf de kernel configureert, neem dan voldoende tijd om de uitleg bij de opties te lezen. Je kan best ook een juiste beschrijving van je computer hardware bij de hand hebben.

[1] De ncurses bibliotheek wordt gebruikt om de interface op te bouwen.

12.2.4 De kernel en bijbehorende modules compileren

Eenmaal de configuratie achter de rug is kan men beginnen aan het compileren van de kernel.

Men de kernel image bouwen met behulp van het bevel `make`. De kernel image en alle modules worden nu gecompileerd en de image wordt in de directory `arch/<architectuur>/boot/`. Voor de i386 architectuur vind je de kernel image terug in: `arch/i386/boot/bzImage`.

12.2.5 De nieuwe kernel installeren

Eens de kernel image klaar is, kan deze geïnstalleerd zodat de kernel kan geboot worden. Daarvoor voer je het bevel `make install` uit. Dit kopieert de kernel image naar de directory `/boot`. Verder dien je nog de kernel modules te installeren met behulp van het commando `make modules_install`. Enkel nog de nieuwe kernel image toevoegen aan de bootloader, en de kernel is klaar om bij de volgende start van de computer gebruitk te worden. Er zijn twee bootloaders die veel gebruikt worden hiervoor, en die meestal standaard mee geïnstalleerd worden bij de installatie van de Linux distributie: LILO (Linux Loader) en GRUB (GRand Unified Bootloader). Raadpleeg de documentatie van de bootloader die op je systeem aanwezig is om de nieuwe kernel bij in het bootmenu te zetten. Recente kernels automatiseren dit en voegen dus automatisch je nieuwe kernel aan de bootloader toe.

12.3 CPU allocatie in de Linux kernel

12.3.1 Inleiding

12.3.2 Enkele evoluties in de Linux kernel scheduler

Preemptive scheduling

$O(1)$

12.4 Memory management in de Linux kernel

Hoofdstuk 13

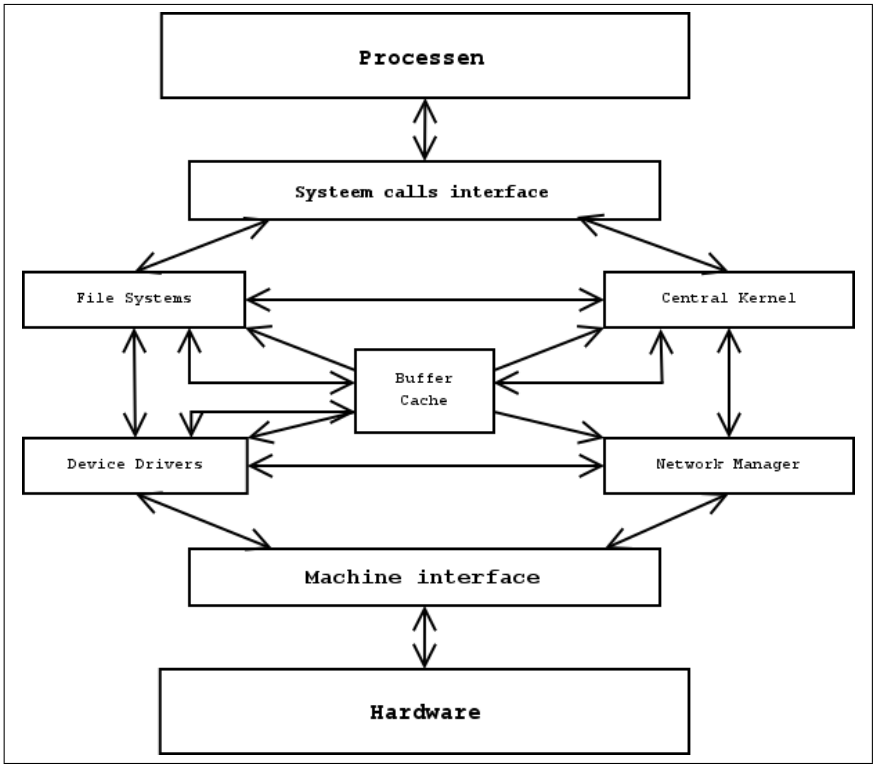
Een device driver in Linux

13.1 Randapparaten

De kern van een besturingssysteem is de kernel. De kernel is een stuk code dat steeds in het geheugen aanwezig is en verantwoordelijk is voor procesbeheer, geheugenbeheer, bestandsbeheer en hardwarebeheer. Om het systeem toe te laten met randapparatuur te communiceren gebruikt het device drivers. Dit kunnen modules zijn die in de kernel geladen worden of code die reeds met de kernel mee gecompileerd is en die toelaat om gegevens van de randapparatuur (de zgnd. devices) te lezen en/of ernaar te schrijven. Device drivers zijn de "software interfaces" om met een bepaald randapparaat te werken. Let wel: dit randapparaat hoeft *niet* een fysisch hardware apparaat te zijn; het kan bijvoorbeeld ook een random data generator of een virtuele console zijn. We hebben twee soorten devices: *character* devices en *block* devices. Character devices lezen de data enkel sequentieel en worden niet gebufferd. Block devices bieden random access aan en hun data wordt wel gebufferd. Een harde schijf, CDROMS,... zijn voorbeelden van block devices. Een console, de geluidskaart en de seriële poort zijn voorbeelden van character devices. In feite wordt er nog een onderscheid gemaakt met de network devices, zodat er eigenlijk 3 soorten devices zijn. Network devices werken dan ook op een totaal andere manier dan character en block devices. Op netwerk devices wordt in deze cursus echter niet dieper ingegaan.

De kernel heeft eigenlijk twee belangrijke interfaces: een interface voor de applicaties en een interface voor de hardware. Op deze manier kan het een uniforme toegang tot processen en hardware aanbieden wat de uitbreidbaarheid en onderhoudbaarheid van het besturingssysteem in grote mate ten goede komt. Een afbeelding van de kernel architectuur met de twee interfaces vind je in figuur 13.1.

Al de devices waarvan je Linux systeem gebruik kan maken staan opgesomd in de /dev directory, zoals reeds vermeld in sectie 3.1. In listing 13.1 kan je zien dat de letter *b* op het begin van een regel erop wijst dat we met een block device te maken hebben (in dit geval is dat de floppy) en dat de letter *c* staat voor een character



Figuur 13.1: Interne kernel architectuur

type device en protecties				*major nummer device driver*	*minor nummer id apparaat*		
brw-rw----	1	root	disk	3,	0	May 5 1998	/dev/hda
brw-rw----	1	root	disk	3,	1	May 5 1998	/dev/hda1
brw-rw----	1	root	disk	3,	64	May 5 1998	/dev/hdb
crw-rw----	1	root	sys	14,	4	May 5 1998	/dev/audio
crw-rw----	1	root	sys	14,	20	May 5 1998	/dev/audio1

Tabel 13.1: Gestructureerde kijk op /dev listing

device (in dit geval de DSP[1] van de geluidskaart). De nummers die volgen op de owner en group zijn de major en minor nummer. /dev/fd0 heeft major 2 en minor 0 en /dev/dsp heeft major 14 en minor 3. De major nummer duidt op een bepaald randapparaat en de minor op een instantie hiervan. We kunnen ook zelf een file aanmaken waar later een randapparaat kan gemapt worden met behulp van het `mknod` commando. Dit commando heeft de syntax: `mknod name type major minor`. Indien we de file /dev/fd0 zelf zouden aanmaken, dan zouden we het commando `mknod /dev/fd0 b 2 0` moeten uitvoeren.

Listing 13.1: Listing van de /dev directory

```
[kris@localhost course]$ls -l /dev/
...
brw-rw----   1 root    floppy    2,   0 Sep 27 12:31 /dev/fd0
...
crw-rw----   1 root    audio    14,   3 Sep 27 12:31 /dev/dsp
```

De betekenis van de /dev listing wordt overzichtelijk voorgesteld in de tabel 13.1

13.2 Programmeren voor de kernel

Er is een duidelijk verschil in aanpak wanneer we voor de kernel moeten programmeren. Het besturingssysteem Linux kent twee verschillende modes: *user-mode* en *supervisor-mode*. De eerste mode is de "normale" mode; de mode waarin we normaal gezien als gebruiker van het systeem werken. De user-mode heeft meer restricties in gebruik van stukken geheugen en aanspreken van de hardware bijvoorbeeld. De tweede mode, supervisor-mode, is de mode waarin we in dit hoofdstuk gaan werken. De supervisor-mode laat ons toe heel het geheugen te gebruiken, dus ook in kernel-space te werken en rechtstreeks hardware aan te spreken. Vooraleer we ons op het harde coderen gooien nog een opmerking: het is een algemene vuistregel dat wanneer men iets niet in de kernel-space hoeft te programmeren, men

[1] Digital Signal Processor

dat gewoon *niet* doet. Het voorbeeldje gegeven in sectie 13.3.1 is louter didactisch, en doorbreekt deze vuistregel.

13.3 Een eerste kernel module

13.3.1 Hello world

We beginnen aan onze eerste kernel module: "Hello world" voor de kernel. Listing 13.2 geeft een voorbeeldje van de module.

Listing 13.2: modhelloworld.c

```
/* modhelloworld.c; a kernel module version */

#include <linux/module.h>

static int __init led_init(void)
{
  /* printk is the kernel version of printf */
  printk(KERN_DEBUG "Hello world - this is your kernel speaking...\n");
  return 0;
}

static void __exit led_exit(void)
{
  printk(KERN_DEBUG "Goodbye dear world!\n");
}

module_init(led_init);
module_exit(led_exit);
```

Het include statement is het minimale include statement voor module-code. `module.h` bevat define statements en andere include statements die de nodige parameters at compile time aanreiken om een geschikte module voor de kernel te maken.

Wat ongetwijfeld eerst opvalt is het gebruik van `printk` in plaats van `printf`. `printk` biedt de mogelijkheid boodschappen vanuit kernelspace naar de gebruiker te sturen. De prefix KERN_DEBUG bepaalt de belangrijkheid van de boodschap. Eigenlijk kent printk verschillende "niveaus". Als printk blijkbaar niets uitschrijft probeer dan "KERN_INFO" vooraan bij in de string te zetten, werkt dat nog niet probeer dan "KERN_NOTICE" enzoverder tot en met "KERN_EMERG", dit is het meest kritische niveau van printk. Soms schrijft printk ook niet naar de console, maar naar een file, gecontrolleerd door `syslogd` en aangegeven in `/etc/syslog.conf` De volgende prefixes kunnen gebruikt worden door `printk`:

KERN_EMERG : het systeem is onbruikbaar

KERN_ALERT : probleem dat onmiddellijke afhandeling vereist

KERN_CRIT : kritisch probleem

KERN_ERR : foutmeldingen

KERN_WARNING : waarschuwingen

KERN_NOTICE : normale maar significante gebeurtenis

KERN_INFO : informatieve boodschappen

KERN_DEBUG : debug-niveau boodschappen

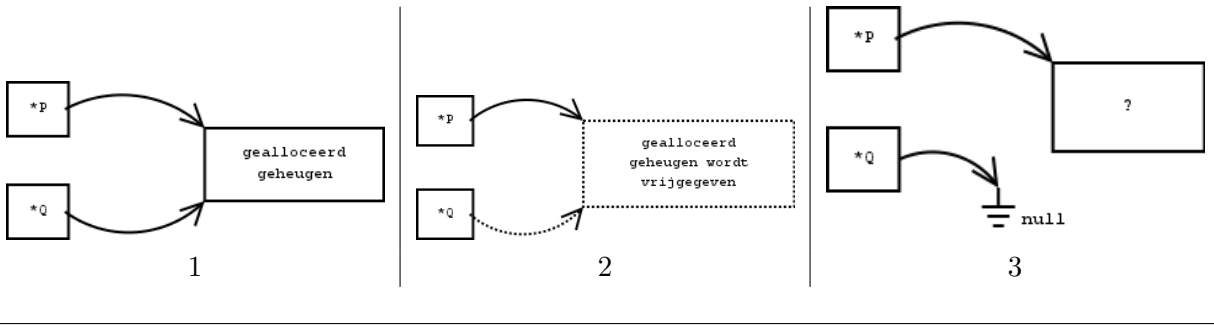
`printk` kan geen floating point getallen wegschrijven. Het wordt trouwens (zeer) sterk afgeraden om met floating point getallen te werken wanneer men voor de kernel programmeert: de floating point staat kan verloren gaan omdat de kernel de staat van de floating point unit niet opslaat. Deze extra informatie wordt niet opgeslaan om performantie rederenen.

De twee functies, `module_init` en `module_exit`, tenslotte zijn bijna zelf-verklarend: De functie gemarkeerd met `module_init` wordt door de kernel aangeroepen als de module in de kernel geladen wordt. De functie gemarkeerd met `module_exit` wordt aangeroepen als de module uit de kernel gehaald wordt om alles op te kuisen. Zeker als men voor kernel programmeert moeten we zeer voorzichtig omspringen met geheugengebruik. Een dangling pointer (zie figuur 13.2; stap 1 laat de pointers naar het geheugen wijzen, in stap 2 wordt het geheugen waar q naar wijst verwijderd, maar p blijft in stap 3 naar dezelfde geheugenplaats wijzen waar nu mogelijk rommel in staat.) is een onvergeefbare fout voor een kernel programmeur, want dit brengt een zeer groot risico voor een kernel crash met zich mee! De kleinste fout die men maakt in code bestemd voor de kernel heeft meestal het vastlopen van de computer als resultaat. Een crash van de kernel is moeilijk op te vangen en te ontleden. In plaats van een *core dump* zoals bij een crash van een gewone applicatie zal de kernel enkel een "Oops message" geven als het misloopt (dan weet je dat het hoogstwaarschijnlijk tijd is om de computer te herstarten). Het ontleden van zo een "Oops message" valt buiten deze cursus. Als de functie gemarkeerd met `module_init` als return value een 0 teruggeeft betekent dit dat de module succesvol geladen is.

Tenslotte moeten we ons programma nog gecompileerd krijgen. Om deze module te compileren gebruiken we het kernel buildsysteem. Bij vorige versies van de Linux kernel werd het zelfstandig compileren van kernel modules nog gebruikt, maar omwille van de talrijke problemen die dit veroorzaakte, wordt nu aangeraden het kernel buildsysteem te gebruiken. Het gebruik van het kernel buildsysteem zorgt ervoor dat de kernel module telkens met exact dezelfde GCC opties gecompileerd wordt.

Om je eigen kernel module te compileren, voeg je onderstaande `Makefile` toe:

```
kernel_release=‘uname -r‘
obj-m := hello.o
```




```
void *p, *q;
q=malloc(N);
p=q;
free(q);
q=NULL;
```

code die een dangling pointer veroorzaakt

Figuur 13.2: Ontstaan van een dangling pointer

```
all:
    make -C /lib/modules/${kernel_release}/build/ SUBDIRS=$(PWD)
```

De volgende opdrachtregel geeft aan hoe we de module moeten compileren:

```
make V=1
```

We zien dat de GCC compiler met een aanzienlijk aantal parameters opgeroepen wordt. Waarvoor dienen deze parameters?

-Wall : geeft aan dat de compiler *alle* warnings moet laten zien tijdens het compileren;

-D__KERNEL__ : De GCC vlag -D is een preprocessor define en komt dus overeen met een "#define" regel in de broncode. Het definiëeren van __KERNEL__ geeft aan dat we voor de kernel aan het compileren zijn;

-DMODULE : geeft aan dat we een module aan het compileren zijn;

-I/usr/src/linux/include : De GCC vlag -I geeft aan dat de compiler in die directory voor de includes moet zoeken. Dit kan verschillend zijn voor verschillende distributies;

-O2 : een kernel module moet met een optimisatie level van O2 of hoger gecompileerd worden omdat dit ervoor zorgt dat inlined functies geëxpandeerd worden in de kernel.

Als de compilatie gelukt is hebben we de module `hello.ko`. Switch nu naar een console en log in als root. Je kan de module nu laden door `insmod hello.ko` in te typen. Let op, dit moet in een console gebeuren omdat printk geen output naar X-terminals geeft. Als je `insmod hello.ko` ingetypt hebt krijg je normaal het volgende te zien:

```
[root@localhost lists]# insmod hello.ko
Hello world - this is your kernel speaking...
[root@localhost lists]#
```

Hoe kan je nu zien of de module wel degelijk geladen is? `/proc/modules` bevat een lijst van alle geladen modules, dus als we hier een cat van doen zien we of "`hello`" erbij is:

```
[root@localhost lists]# cat /proc/modules
hello 3180 0 - Live 0xf95f9000
aes 29792 1 - Live 0xf960e000
dm_crypt 12520 1 - Live 0xf95f4000
rfcomm 40056 1 - Live 0xf95fc000
hidp 16576 2 - Live 0xf95cf000
l2cap 24836 10 rfcomm,hidp, Live 0xf95e7000
bluetooth 49476 5 rfcomm,hidp,l2cap, Live 0xf95d9000
...
piix 10468 0 [permanent], Live 0xf8830000
generic 4356 0 [permanent], Live 0xf881d000
ide_core 130656 4 ide_cd,ide_generic,piix,generic, Live 0xf884d000
unix 28176 1116 - Live 0xf8823000
[root@localhost lists]#
```

Bovenaan het lijstje prijkt nu onze module. Om de module te verwijderen typ je `rmmod hello` in. Dan krijg je normaal de boodschap van `module_exit` te zien:

```
[root@localhost lists]# rmmod hello
bye bye
[root@localhost lists]#
```

13.3.2 Uitgebreider voorbeeld in het proc filesystem

Nu de fundamenten gelegd zijn, kunnen we een stapje verder gaan. We gaan een kernel module maken die een teller bijhoudt voor elke keer dat er informatie opgevraagd wordt over deze module. We zorgen ervoor dat de module ook een entry in de `proc` directory [2] heeft zodanig dat we via het `cat` commando informatie

[2] Het /proc filesystem is origineel gemaakt om makkelijk informatie over processen op te vragen. Tegenwoordig wordt het ook gebruikt om alle relevante informatie over de kernel op te vragen

kunnen opvragen. Bij het programmeren van deze module moeten we de input-output omdraaien. Wat voor de gebruiker van het systeem *data lezen* is, is voor de kernel *data schrijven* en omgekeerd. Dus als we gegevens van onze module opvragen is dat voor ons lezen, maar voor de module zal dat schrijven betekenen. Door middel van het `cat` commando *lezen* we dus de module.

Onze module, `proctext-2.6.c`, is afgebeeld in listing 13.3. Merk op dat er een include statement is bijgekomen: `proc_fs.h` omdat we het proc filesystem gebruiken.

In de functie `module_init` registreren we dus onze module bij het `/proc` systeem. Hiervoor gebruiken we de `create_proc_read_entry` functie. Deze functie verwacht als belangrijkste parameters de naam waaronder de informatie in de `/proc` directory moet verschijnen en de functie die instaat voor de inhoud van deze entry.

```
   create_proc_read_entry
 (
 const char *name,              →naam van de module
 mode_t mode,                   →protection mask, 0 voor default gedrag
 struct proc_dir_entry *base,   →definieert de set van operaties, hier zijn dat er geen dus
                                schrijven we NULL
 read_proc_t *read_proc,        →de funtie die opgeroepen wordt om informatie van de mo-
                                dule te verkrijgen
 void *data                     →Dit altijd op NULL zetten
 );
```

Een entry in het `/proc` system is in dit geval handig omdat we geen entry hebben in de `/dev` directory. Hiervoor zouden we met `mknod` eerst een entry moeten maken. Nu wordt er in de `/proc` directory automatisch een entry gemaakt als we de module laden. Merk op dat de module die we hier schrijven ook nog geen "echte" device driver is, het stuurt namelijk niets aan. Om de entry weer uit het `/proc` system te verwijderen gebruiken we de functie `remove_proc_entry` in de functie gemarkeerd met `module_exit`.

Listing 13.3: proctext.c (kernel 2.6)

```
#include <linux/module.h>
#include <linux/config.h>
#include <linux/init.h>
#include <linux/proc_fs.h>

MODULE_LICENSE("GPL");

static int read_proc( char *buffer,
      char **buffer_location,
      off_t offset,
      int buffer_length,
     int *eof,
     void *data)
```

```
{
  int len;
  static int count = 1;
  static char my_buffer[80];

  if(offset>0)
    return 0;


  len = sprintf(my_buffer,
        "You have asked me that already %d times!\n",
        count);
  count++;
  *buffer_location = my_buffer;
  *eof = 1;
  return len;
}

static int __init proctext_init()
{
  create_proc_read_entry("teller", 0, NULL, read_proc, NULL);
  return 0;
}


static void __exit proctext_exit()
{
  remove_proc_entry("teller", NULL);
}

module_init(proctext_init);
module_exit(proctext_exit);
```

Wat gebeurt er nu als we de module inserteren en dan de opdracht `cat /proc/teller` uitvoeren? De module reageert hierop door de functie op te roepen die we geregistreerd hebben met `create_proc_read_entry`, namelijk `procfile_read`. De output die moet verschijnen als reactie op `cat /proc/teller` wordt aangewezen door `buffer_location`. `len` geeft aan hoeveel bytes er in feite gebruikt werden. De offset is altijd gelijk aan 0 in dit voorbeeldje, omdat we data niet in chunks (verschillende afgelijnde delen) maar in één keer versturen. Als de read functie opgeroepen wordt met een offset>0 geven we niets terug, omdat met een offset groter dan 0 een volgende chunk bedoeld wordt. Daar onze data niet zo groot is, is het makkelijk om die in één keer door te geven. Als deze echter wat groter wordt kunnen we zo de data wel in stukken versturen (en zo de processor beter delen met andere processen).

13.4 Devices als files

13.4.1 De API

Tot nu toe hebben we nog geen echte device drivers geschreven, maar enkel modules die in de kernel geladen worden. Als we een echte device driver willen schrijven moeten we er in Linux mee rekening houden dat lezen van en schrijven naar een randapparaat dezelfde software architectuur gebruikt als operaties op bestanden (dit wordt ook weerspiegeld in het normale gebruik van het besturingssyteem, zie ook sectie 3.1 en sectie 3.6). Dit wil zeggen dat de verzameling geldige functies om met bestanden te werken ook geldt voor device drivers. Bij het schrijven van zulk een device driver moeten we aangeven welke functies uit deze verzameling we gaan gebruiken. Al de mogelijke functies voor het werken met bestanden zijn samengepakt in een structure:

```
struct file_operations {

  loff_t       (*llseek)            (struct file *, loff_t, int);
  ssize_t      (*read)              (struct file *, char __user *, size_t, loff_t *);
  ssize_t      (*aio_read)          (struct kiocb *, char __user *, size_t, loff_t);
  ssize_t      (*write)             (struct file *, const char __user *, size_t, loff_t*);
  ssize_t      (*aio_write)         (struct kiocb *, const char __user *, size_t, loff_t);
  int          (*readdir)           (struct file *, void *, filldir_t);
  unsigned int (*poll)              (struct file *, struct poll_table_struct *);
  int          (*ioctl)             (struct inode *, struct file *,
                                    unsigned int, unsigned long);
  int          (*unlocked_ioctl)    (struct file *,
                                    unsigned int, unsigned long);
  int          (*compat_ioctl)      (struct file *,
                                    unsigned int, unsigned long);
  int          (*mmap)              (struct file *, struct vm_area_struct);
  int          (*open)              (struct inode *, struct file *);
  int          (*flush)             (struct file *);
  int          (*release)           (struct inode *, struct file *);
  int          (*fsync)             (struct file *, struct dentry *, int);
  int          (*aio_fsync)         (struct kiocb *, int);
  int          (*fasync)            (int, struct file *, int);
  int          (*lock)              (struct file *, int, struct file_lock *);
  ssize_t      (*readv)             (struct file *, const struct iovec *,
                                    unsigned long, loff_t *);
  ssize_t      (*writev)            (struct file *, const struct iovec *,
                                    unsigned long, loff_t *);
  ssize_t      (*sendfile)          (struct file *, loff_t *, size_t,
                                    read_actor_t, void *);
  ssize_t      (*sendpage)          (struct file *, struct page *, int,
                                    size_t, loff_t*, int);
  unsigned long(*get_unmapped_area) (struct file *, unsigned long,
                                    unsigned long, unsigned long, unsigned long);
  int          (*check_flags)       (int);
  int          (*dir_notify)        (struct file *, unsigned long);
  int          (*flock)             (struct file *, int, struct file_lock *);
```

};
De code voor de device driver declareert een variabele van het type `file_operations` en zet vervolgens alle functies die *niet* ondersteund zullen worden door de device driver op NULL. De functies die wel ondersteund worden door de device driver, worden als verwijzing naar de echte implementatie in de code van de device driver wel in de structure geplaatst. Let erop dat de structure *pointers naar functies* bevat; op deze manier is het mogelijk nieuwe device drivers te schrijven waarvan de kernel niet "op de hoogte" was. Door enkel verwijzingen in `file_operations` door te geven naar de functies die ter ondersteuning van de device driver geprogrammeerd zijn, is het niet nodig dat de code statisch met de kernel gelinkt was. De functionaliteiten die een device driver aanbiedt worden bepaald door een tabel met wijzers (pointers) naar functies die bijgehouden en doorgegeven wordt. Functies worden door middel van hun verwijzingen (die terug te vinden zijn in die tabel) opgeroepen en niet "statisch" via hun naam (anders zou deze reeds kernel compile-time moeten gekend zijn!). In appendix A is wat meer uitleg verschaft over pointers naar functies.

13.4.2 Gebruikte entries

De entries in de `file_operation` structure die voor ons het meest interessant zijn, zijn ongetwijfeld read, write, open en close. Op ioctl wordt kort nog teruggekomen in sectie 13.4.3. Voor de read en write functies moeten we oppassen of we nu in kernel space of in user space aan het lezen en/of schrijven zijn. Het besturingssyteem is hier zeer strikt in! We kunnen gebruik maken van de twee functies **copy_to_user** en **copy_from_user**, met de definities:

```
unsigned long copy_to_user(void *to, void *from, unsigned long size)
unsigned long copy_from_user(void *to, void *from, unsigned long size)
```

De oorsprong en het doel van de kopieer operatie worden aangegeven, samen met de grootte van de te kopiëren data. In feite zijn deze twee functies de "uitgebreide" versie van `get_user` en `put_user` die enkel geschikt zijn voor kleine hoeveelheden data (een enkele primitieve variabele) te kopiëren. De `copy_`*`xx`*`_user` is beter geschikt voor grotere hoeveelheden data.

De functie `copy_to_user` laat toe om data van de kernel space naar de user space te kopiëren. De functie `copy_from_user` doet het omgekeerde: het kopiëert data van de user space naar de kernel space. Indien de functies succesvol uitgevoerd worden zullen ze 0 teruggeven. Indien dit niet het geval is geven ze een niet nul waarde terug, en wil dit meestal zeggen dat de toegang tot de doel-memory space ontzegd is.

13.4.3 I/O control: ioctl

Een zeer belangrijke entry in de structure is de `ioctl` verwijzing. `ioctl` laat toe om I/O-apparaten in te stellen zonder hiervoor nieuwe functies te hoeven schrijven. Meer zelfs; het laat toe instellingen van een I/O-apparaat te lezen en/of te veranderen terwijl de driver reeds in gebruik is. Dit is zeer handig voor device drivers die permanent in de kernel aanwezig zijn. Wij zullen hier echter geen gebruik meer van maken.

13.5 Een device driver voor de keyboard LEDs

Deze sectie is gebaseerd op de tekst van Michiel Ronsse [Ron00].

13.5.1 De opdracht

Om de opgedane kennis in dit hoofdstuk in de praktijk om te zetten, gaan we een device driver voor de 3 LEDs[3] van het toetsenbord schrijven. Eerst moeten we hiervoor het geschikte device voor aanmaken in de `/dev` directory, en dit doen we met het `mknod` commando, namelijk: `mknod /dev/led c x y`. Merk op dat het hier over een character device driver gaat. De waarden die voor x en y gebruikt dienen te worden, kunnen met `printk` aan de gebruiker getoond worden.

Het is de bedoeling dat we door + of - kunnen aangeven of een ledje uit of aan moet zijn. `echo +++ > /dev/led` zal de 3 ledjes laten oplichten. Als je `echo +-+ > /dev/led` uitvoert mag het middenste led niet oplichten, maar de twee buitenste wel.

13.5.2 Verduidelijking

Als u aandachtig de vorige sectie (sectie 13.5.1) heeft gelezen, ziet u dat voor de led driver er niet veel file operations nodig zijn: u hoeft enkel een read en write functie te definiëren naast de vereiste open en close (release) functies. In de `file_operations` structure zullen er dus 4 niet-NULL entries terug te vinden zijn.

poorten

Om de eigenlijke hardware aan te sturen moeten we de juiste "poorten" van de hardware kennen en zorgen dat we het device met de juiste major nummer aanspreken. Communicatie van het systeem met de hardware verloopt via deze poorten. De data poort van het toetsenbord staat op IO adres `0x60`, de status poort op IO adres `0x64`. Voor de LEDS tenslotte moeten we IO adres `0xED` gebruiken om de

[3] Light Emitting Diodes; op het toetsenbord bedoelen we hiermee "Num Lock", "Caps Lock" en "Scroll Lock"

LEDs op het toetsenbord aan te sturen. De volgende defines kan je gebruiken in de code om aan te geven welke adressen je voor het keybord moet gebruiken:

```
#define DATA      0x60      /* data poort v/d keyboard controller */
#define STATUS    0x64      /* status poort v/d keyboard controller */
#define SETLEDS   0xED      /* commando om leds aan te sturen */
```

Niet alleen voor het toetsenbord zijn er poorten voorzien, maar ook voor de andere randapparatuur natuurlijk. Zo heeft data poort van de printer IO adres `0x378`. De IO adressen van deze poorten zijn hardware afhankelijk en we werken hier met een PC compatibele architectuur! Voor verschillende klassen van hardware (devices) zijn bereiken van poorten voorzien. Voor de hard disk controller is bijvoorbeeld het bereik `0x320` tot `0x32F` gereserveerd en voor de parallelle poort het bereik `0x378` tot `0x37F`.

Low level poort input en output

Nu we de abstractie van *poorten* kennen in de hardware is het nodig om enkele operaties te leren kennen om echt datatransport mogelijk te maken. Er worden functies voorzien om datatransport in de kernel naar en van de hardware mogelijk te maken: namelijk `outb`, voor output naar de poort toe en `inb` voor input van de poort [4]. De definities van `inb` en `outb` in `include/asm/io.h` zijn als volgt:

```
inline void outb(char value, unsigned short port)
inline unsigned int inb(unsigned short port)
```

Bij `outb` geven we als argumenten de data (d.i. een byte) en het poort nummer mee aan de functie. Bij `inb` geven we enkel het poort nummer en verwachten we een byte, gelezen van die poort terug. `inb` en `outb` proberen zo snel mogelijk respectievelijk te lezen of te schrijven naar de poort, maar sommige hardware heeft liever dat dit niet gebeurt, maar dat er gewacht kan worden vooraleer te lezen/schrijven. Omdat we hier niet onderhevig zijn aan timing constraints, zoals bij realtime systemen e.d. wel het geval is, kunnen we hier gebruik maken van de "pauserende" tegenhangers van `outb` en `inb` namelijk `outb_p` en `inb_p`. Deze twee functies worden in `include/asm/io.h` als volgt gedfiniëerd:

```
inline void outb_p(char value, unsigned short port)
inline unsigned int inb_p(unsigned short port)
```

[4] Er wordt zo een hele "familie" van functies voorzien, te vinden in `include/asm/io.h`:
outb, outw, outl, outsb, outsw, outsl voor poort *output*
inb, inw, inl, insb, insw, insl voor port *input*
outb_p, outw_p, outl_p, inb_p, inw_p, inl_p voor *paused I/O*

driver registreren

Anders dan bij het vorige voorbeeld, moeten we nu de module niet registreren bij het proc filesystem, maar wel als driver registreren. Dit moet in de functie `module_init` gebeuren.

Vooraleer we een driver kunnen registeren, dienen we eerst een major nummer aan te vragen. Hiervoor gebruiken we de functie `alloc_chrdev_region`, waaraan we als parameters eerst en vooral een pointer naar een variable van type `dev_t` meegeven, waarin de gealloceerde device nummer geplaatst zal worden. Verdere parameters zijn de eerst te gebruiken minor nummer, het aantal gewenste device nummers en de naam.

Nu we een device major nummer van de kernel gekregen hebben, kunnen we de driver registeren. Met de `cdev_alloc` functie, kunnen we een variable van type `cdev` alloceren. Via de `ops` member van deze struct kunnen we onze een verwijzing naar de `file_operations` structure koppelen aan de driver. De uiteindelijke registratie zelf, doen we met de `cdev_add` functie, waaraan we als parameters de zonet gealloceerde variabele van type `cdev`, de device nummer en het aantal te associeren devices meegeven (in deze volgorde).

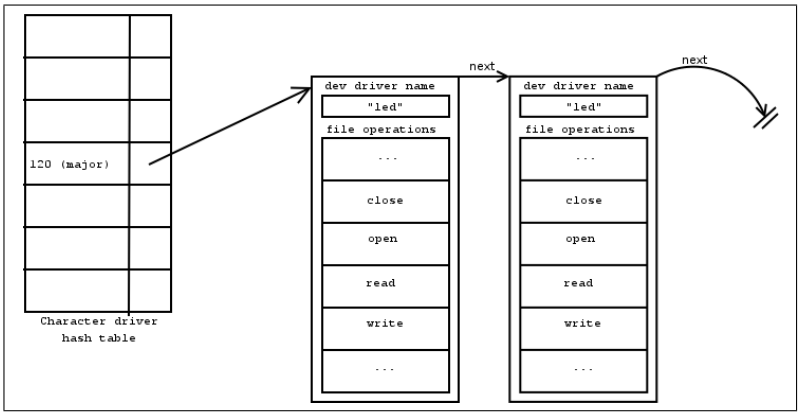
Bij de registratie van deze drivers wordt door de kernel een *character device switch table* bijgehouden zoals getoond in figuur 13.3. Hiervoor wordt een hash table (`chrdevs`) gebruikt die als entries variabelen van het type `struct char_device_struct` heeft. Deze structure wordt als volgt gedefinieerd:

```
struct device_struct
{
   struct char_device_struct *next;
   unsigned int major;
   unsigned int baseminor;
   int minorct;
   char name[64];
   struct file_operations *fops;
};
```

De major nummer wordt gebruikt als key voor de hash table, zodanig dat met behulp van de `major_to_index` functie en het doorlopen van een lijst, de `list_entry→fops` de file operaties teruggeeft en `list_entry→name` de naam van het device teruggeeft.

error codes

Om beter te kunnen checken op errors en soorten errors, is er de header file `errno.h`. Deze include zelf meestal `asm/errno.h`, bekijk deze file om meer te weten te komen over de betekenis van de error codes (meestal kan je de header files vinden in `/usr/src/linux/include/`, dit kan lichtjes verschillen afhankelijk van het systeem). Die laat je bijvoorbeeld toe de return value van `register_chrdriver`



Figuur 13.3: Character device switch table

te controleren. Als de code `-EBUSY` wordt teruggegeven dan weet je dat het device dat of de resource die je probeert aan te spreken bezet is, en dat de operatie ten gevolge mislukt is. Indien bij `write_leds` of `read_leds` iets misloopt moet er de error code `-EFAULT` teruggegeven worden. Let op de negatie van de error codes. Zo kan simpel gecheckt worden of een return waarde < 0 is, en aan de hand daarvan concluderen dat er "iets" is misgelopen[5].

13.5.3 De code

Framework

Om te beginnen laten we een framework voor de applicatie zien in listing 13.4. Dit zijn de functies en structures die je moet vervolledigen om tot een werkende device driver te komen. Daar we als major nummer voor het device 120 bij `mknod` hadden meegegeven, kunnen we dit best ook definiëren in de source code. `#define LED_DEV 120` bij de andere defines plaatsen volstaat zodat we `LED_DEV` kunnen gebruiken in plaats van 120.

Listing 13.4: Het framework voor de LEDs device driver

```
#define LED_DEV 120     /* major nummer van /dev/led */
#define DATA   0x60     /* data poort v/d keyboard controller */
#define STATUS 0x64     /* status poort v/d keyboard controller */
#define SETLEDS 0xED    /* commando om leds aan te sturen */

MODULE_LICENSE("GPL");

#include <asm/io.h>
```

[5]Dit is enkel zo omdat de device drivers in plain C worden geprogrammeerd. Als u een programma in C++ of Java schrijft is het "bad programming practice" foutwaarden mee te geven met de return waarde. Het gebruik van excepties is te verkiezen boven deze methode van fout indicatie en afhandeling.

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/ioport.h>
#include <linux/errno.h>
#include <linux/major.h>
#include <asm/uaccess.h>

static void set_leds(){}
static ssize_t read_leds(struct file * file, char * buf,
                         size_t count, loff_t *ppos){}
static ssize_t write_leds(struct file * file, const char * buf,
                          size_t count, loff_t *ppos){}
static int open_leds(struct inode * inode, struct file * file){}
static int close_leds(struct inode * inode, struct file * file){}
static struct file_operations led_fops = {};
static int __init led_init(void){}
static void __exit led_exit(void){}
module_init(led_init);
module_exit(led_exit);
```

Om hiermee te kunnen werken hebben we eerst nog wat extra informatie over de functie declaraties nodig.

read_leds

parameters

struct file * file : uit deze "file" vraagt de applicatie om te lezen;

char * buf : de buffer waarin het resultaat geplaatst moet worden;

size_t count : het aantal bytes dat de applicatie vraagt om te lezen;

loff_t *ppos : de offset waar we in de buffer aan het lezen zijn, de "leeswijzer";

return waarde *ssize_t*: indien >0, het aantal bytes dat gelezen is, anders wordt een error waarde teruggegeven. Deze functie mag maar *1* karakter teruggeven! Een foutief resultaat moet met `-EFAULT` aangeduid worden. Anders zal er 1 teruggegeven worden als er een karakter gelezen is, anders 0 ("einde bestand").

write_leds

parameters

struct file * file : naar deze "file" vraagt de applicatie om te schrijven (hebben we niet nodig voor onze device driver);

const char * buf : wat er ons gevraagd wordt om te schrijven;

size_t count : het aantal bytes dat de applicatie vraagt om te schrijven;

loff_t *ppos : de offset waar we in de buffer aan het schrijven zijn, de "leeswijzer";

return waarde *ssize_t*: indien >0, het aantal bytes dat geschreven is, anders wordt een error waarde teruggegeven;

open_leds

parameters

struct inode * inode zullen we niet hoeven te gebruiken;

struct file * file zullen we niet hoeven te gebruiken;

return waarde *int*: indien dit <0 is wordt er een error waarde teruggegeven;

close_leds

parameters

struct inode * inode zullen we niet hoeven te gebruiken;

struct file * file zullen we niet hoeven te gebruiken;

return waarde *int*: indien dit <0 is wordt er een error waarde teruggegeven;

De eerste code

Nu we de grondvesten gelegd hebben, rest er ons alleen nog wat code voor de bodies van de functies te schrijven. In listing 13.5 wordt een eerste korte uitleg gegeven over wat er juist in welke body moet komen. Merk op dat de enige functie die echt met de hardware zal communiceren `set_leds()` is.

Listing 13.5: De eerste code voor de LED device driver

```
#define LED_DEV 120     /* major nummer van /dev/led */
#define DATA   0x60     /* data poort v/d keyboard controller */
#define STATUS 0x64     /* status poort v/d keyboard controller */
#define SETLEDS 0xED    /* commando om leds aan te sturen */

MODULE_LICENSE("GPL");

#include <asm/io.h>
#include <linux/module.h>
#include <linux/config.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/ioport.h>
#include <linux/errno.h>
#include <linux/major.h>
#include <asm/uaccess.h>
```

```
#define FALSE 0

/* Aantal gebruikers, mag hoogstens 1 worden in deze oefening*/
static int led_users = 0;
/* Houdt de stand van de leds bij */
static int led_on[3] = {FALSE, FALSE, FALSE};

static void set_leds()
{
 /* We zetten het ingelezen patroon om in een geschikte
  * vorm om aan outb_p mee te geven. Vervolgens
  * zeggen we dat we de LEDs willen gaan aansturen
  * waarop we wachten totdat deze klaar zijn om aangestuurd te worden.
  * In de laatste stap zetten we de LEDs volgens het patroon.
  *
  * Dit is de enige functie die echt met de hardware via de poorten
  * commmuniceert!
  */
}


static ssize_t read_leds(struct file * file, char * buf,
                         size_t count, loff_t *ppos)
{
 /* We gaan in deze functie de stand van de leds uitlezen.
  * We lezen deze vervolgens uit door de offset *ppos
  * te verhogen en de LED stand uit te lezen van
  * de array led_on.
  * Hierbij moet copy_to_user gebruikt worden om
  * de gevraagde data in user space te krijgen.
  */
}

static ssize_t write_leds(struct file * file, const char * buf,
                          size_t count, loff_t *ppos)
{
 /* Dit is het moeilijkste gedeelte van het geheel.
  * In deze functie wordt de stand van de LEDs geschreven
  * Dus iets wat we invoeren vanuit user space moet naar
  * kernel space gekopieerd worden. Hierbij moet
  * copy_from_user gebruikt worden.
  */
}

static int open_leds(struct inode * inode, struct file * file)
{
 /* Zorg ervoor dat slechts 1 gebruiker de driver kan openen,
  * gebruik makend van led_users.
  * Gebruik ook de voorgedefinieerde module usage counter
```

```
  */
}

static int close_leds(struct inode * inode, struct file * file)
{
 /* Hier wordt de device driver gesloten. Denk eraan
  * dat dit een "illegale" operatie is als voorafgaand
  * de driver niet geopend was!
  * Gebruik ook de voorgedefinieerde module usage counter
  */
}

static struct file_operations led_fops =
{
 /* Hier vul je al de file operations in
  * die deze device driver ondersteunt,
  * de overige entries worden NULL gezet.
  * Merk op dat deze device driver maar 4
  * soorten operaties ondersteunt (lezen,
  * schrijven, openen en sluiten)
  */
};

static int __init led_init(void)
{
  /* Hier gebeurt het registreren van de character device
   * driver. Zorg ervoor dat hier ook de juiste fout-
   * afhandeling plaatsvindt en gepaste boodschappen
   * uitgeschreven worden.
   */
  return 0;
}

static void __exit led_exit(void)
{
 /* Hier wordt de character device driver verwijderd.
  * Zorg ook hier voor de juiste foutafhandeling
  * en de gepaste boodschappen
  */
}

module_init(led_init);
module_exit(led_exit);
```

Vooral bij `write_leds` is het uitkijken geblazen! Dit is een functie die data van user space naar kernel space moet kopiëren om de van hem verwachte functie te vervullen. Besef dat het zeer gevaarlijk is nu buiten het gealloceerde geheugen te schrijven. Omdat we met super user rechten werken in kernel space is het mogelijk dat u daardoor data van andere processen overschrijft of andere data

spaces, die niet voor de device driver gereserveerd zijn, vervuilt. Dit kan leiden tot een paginafout. We moeten er ook voor zorgen dat we niet meer bytes als `count` kopiëren, anders lezen we teveel uit de user space, wat weer tot een paginafout kan leiden.

Nog enkele kleinigheden in verband met de programmeertaal C die u dient te weten vooraleer u zelf kan beginnen te coderen:

- *Bitgewijze AND (`&`) operator*:
  deze operator kan aangewend worden om een bit te testen, bijvoorbeeld:
  - `x = bbyte&0x02` zet x op 2 als de 2e bit van bbyte aan staat, anders 0. We kunnen meestal ook gewoon `bit2 = bbyte&2` schrijven hiervoor.
  - `y = bbyte&4` (binair is 4 nl. 000100) zet de stand van y op 4 als de 3e bit van bbyte aanstaat, anders op 0.
- *De Left-Shift operator* <<:
  verschuift de bits in de variabele een opgegeven aantal bits naar links. Hiervoor worden er rechts nullen ingeschoven. Als we rekening moeten houden met het teken en we hebben een negatieve waarde worden er enen in plaats van nullen ingeschoven.

De startcode

Na al deze informatie is het nu mogelijk om de device driver zelf af te maken. De start code wordt u gegeven in listing 13.6. U moet de code vervolledigen en werkend krijgen. U moet ook alle code begrijpen en kunnen uitleggen. Met andere woorden moet u vragen kunnen beantwoorden in verband met de structuur, opbouw en betekenis van de code.

Listing 13.6: De startcode voor de LED device driver

```
#include <linux/ioport.h>
#include <linux/fs.h>
#include <linux/module.h>
#include <linux/config.h>
#include <linux/init.h>
#include <linux/cdev.h>
#include <linux/errno.h> /* Definitie van EFAULT, EBUSY, EIO, ... */
#include <asm/io.h>
#include <asm/uaccess.h>

#define DRIVER_NAME "led"
#define KB_PORT_DATA  0x60      /* data poort v/d keyboard controller */
#define KB_PORT_STATUS 0x64     /* status poort v/d keyboard controller */
#define KB_COMMAND_SETLEDS 0xED   /* commando om leds aan te sturen */
#define TRUE 1
#define FALSE 0
```

```
MODULE_LICENSE("GPL");

dev_t dev;
struct cdev *led_cdev;


static int led_on[3]   = {FALSE, FALSE, FALSE}; /* status v/d LEDs */



static void set_leds(void){
  int pattern = (led_on[1]<<2) + (led_on[0]<<1) + led_on[2];

  outb_p(KB_COMMAND_SETLEDS, KB_PORT_DATA);
  while (inb_p(KB_PORT_STATUS)&2)
    ;
  outb_p(pattern, KB_PORT_DATA);
}


static ssize_t read_leds(struct file * file, char * buf,
                         size_t count, loff_t *ppos){
  char kar;

  kar = (*ppos<3) ? (led_on[*ppos] ? '+' : '-') : (*ppos==3) ? '\n' : 0;

}


static ssize_t write_leds(struct file * file, const char * buf,
                          size_t count, loff_t *ppos){
  int i,start,stop;


  if (*ppos<3) {
    start = *ppos;
    stop  = *ppos+count;
    if (stop > 3) stop = 3;
    for (i=start; i<stop; i++)
      switch(temp[i-start]){
       case '+': led_on[i] = TRUE; break;
       case '-': led_on[i] = FALSE; break;
      }
  }

  set_leds();

  /* zorg ervoor ppos aan te passen! */
}
```

```
static int open_leds(struct inode * inode, struct file * file){

  return 0;
}


static int close_leds(struct inode * inode, struct file * file){

  return 0;
}


static struct file_operations led_fops = {

 /* Hier vul je al de file operations in
  * die deze device driver ondersteunt,
  * de overige entries worden NULL gezet.
  * Merk op dat deze device driver maar 4
  * soorten operaties ondersteunt (lezen,
  * schrijven, openen en sluiten)
  */
  .owner = THIS_MODULE,
  .read = read_leds,
  .write = write_leds,
  .open = open_leds,
  .release = close_leds,
};


static int __init led_init(void)
{
  int result;

  result = alloc_chrdev_region(&dev, 0, 1, DRIVER_NAME);
  if (result < 0) {
    return result;
  }
  printk(KERN_INFO DRIVER_NAME ": Allocated major %d, minor %d\n", MAJOR(dev), MINOR(dev));

  led_cdev = cdev_alloc();
  led_cdev->ops = &led_fops;
  led_cdev->owner = THIS_MODULE;
  result = cdev_add(led_cdev, dev, 1);
  if (result < 0)
    unregister_chrdev_region(dev, 1);
  return result;
}
```

```
static void __exit led_exit(void) {
  cdev_del(led_cdev);
  unregister_chrdev_region(dev, 1);
}

module_init(led_init);
module_exit(led_exit);
```

13.5.4 Praktisch gebruik van de device driver

Het gebruik van de device driver ontwikkeld in dit hoofdstuk is enkel geïllustreerd door het shell commando `echo +-+ > /dev/led`. Wat als je nu deze device driver wil gebruiken vanuit een ander programma? Door de voorstelling van devices als files, kunnen we ook de traditionele file-operaties gebruiken om een device driver aan te sturen. We kunnen eerst een "open" operatie op de device driver uitvoeren, waarna we met een "write" functie naar het device kunnen schrijven.

We kunnen de device driver vanuit een C programma met de functie `fopen` openenen en met behulp van `fprintf` kan er data naar geschreven worden. Listing 13.7 illustreert hoe dit gecodeerd kan worden.

Listing 13.7: Aansturen van het device vanuit een C programma

```
#include <stdio.h>
int main(int argc, char *argv[])
{
  FILE *file = fopen("/dev/led", "w+");
  if(!file)
  {
    fprintf(stderr, "Kan leds niet aansturen\n");
    return -1;
  }
  else
  {
      fprintf(file, "%s\n", "+++");
      fclose(file);
     return 1;
  }
}
```

Nu is het mogelijk om programma's te maken die de LEDs van het toetsenbord gebruiken. Enkele mogelijkheden zijn:

- een programma dat binnenkomende emails meldt door een LED te laten knipperen
- een programma dat de LEDs laat knipperen op de maat van muziek

- een programma dat de processor belasting laat zien aan de hand van de LEDs

Opgave Schrijf een programma dat uit een bestand een sequentie van LED standen leest en deze uitvoert. Zorg ervoor dat de overgang van de vorige naar de volgende toestand met een redelijk tijdsinterval rekening houdt. Zoek zelf de nodige functies hiervoor op.

13.6 Verdere informatie

Voor meer informatie over het programmeren van device drivers worden de volgende bronnen aangeraden: [Ron00], [AJ00], [Pom99], [LDP] en [SM99].

Hoofdstuk 14

Hoe software installeren

Het is onvermijdelijk dat je Linux systeem na verloop van tijd nood heeft aan nieuwe software: omdat nieuwe versies van geïnstalleerde pakketten uitkomen, omdat je veiligheidslekken wil dichten, omdat je nieuwe programma's wil installeren, enz. Omdat Linux vrije software is, en dus beschikbaar in broncode vorm, bestaan er twee manieren om software te installeren:

1. *Installatie van binair gedistribueerde software.* Alle bestaande distributeurs van vrije software hebben hiervoor software-ondersteuning voorzien, *en* archieven waar je de binaire pakketten kan afhalen. Sectie 14.1 geeft meer uitleg.

2. *Installatie van zelf te compileren broncode.* In de loop van de jaren is er een zekere uniformiteit ontstaan over hoe je een pakket van broncode configureert, compileert en installeert. Sectie 14.2 geeft meer uitleg.

14.1 Binaire software distributie

Dit is de klassieke manier waarop de meeste software-leveranciers hun producten aanbieden. Linux heeft gevordere ondersteuning voor dit soort installaties (zie later in deze sectie), maar ook nog een aantal gebreken die ervoor zorgen dat het hele proces geen "no brainer" is. De belangrijkste gebreken zijn:

- Linux heeft niet erg veel aandacht voor *backwards compatibility*, d.w.z., het kan heel goed voorvallen dat een nieuwe versie van een pakket niet meer binair compatibel is met zijn voorganger, en dat andere software die gebruik maakt van het geïnstalleerde pakket problemen kan geven.

- Er zijn een grote hoop servers in de wereld waar je van zowat alle beschikbare Linux software binaire installatie-pakketten kan afhalen. Maar Linux distributies zijn onderling nog niet in die mate gestandaardiseerd dat je er zeker van kan zijn dat je van gelijk welke server gelijk welk binair installatie-pakket kan afhalen en het zonder problemen zal kunnen installeren en configureren.

- Linux heeft veel verschillende releases van stabiele kernels. En veel ondersteunende bibliotheken hebben ook een hoop versies. Je kan die wel naast elkaar installeren, maar het blijft een feit dat andere leveranciers van software het niet makkelijk hebben om te garanderen dat de binaire bestanden die ze ter beschikking stellen ook zullen werken op jouw systeem.

Er zijn natuurlijk niet alleen gebreken aan de verdeling van Linux software; de volgende secties stellen de sterke punten voor.

14.1.1 Red Hat Package Manager

Red Hat is een bedrijf dat Linux distributies verzorgt. Om het de gebruikers te vergemakkelijken heeft het een formaat ontworpen om software te installeren op een systeem, namelijk *RPM*[1], het *Red Hat Package Management systeem*. De software zit verpakt in een bestand dat de extensie `.rpm` draagt. Dit bestand kan met behulp van het `rpm` programma geïnstalleerd worden. De syntax van het `rpm` programma is `rpm [opties] package-naam`. Bijvoorbeeld, het pakket `hugs-1.33-2.i386.rpm` installeer je met `rpm -i hugs-1.33-2.i386.rpm`. Let wel dat men voor de meeste software installaties als beheerder (root of superuser) op het systeem moet ingelogd zijn. In het pakket zit ook informatie over het pakket, namelijk: *hugs* staat voor de naam van het programma dat in het pakket verpakt zit, *1.33* staat voor het versie-nummer, *2* staat voor het release-nummer, en *i386* geeft aan dat het pakket bedoeld is voor een intel 386 processor-architectuur. In het `RPM` bestand zit niet alleen een binaire, uitvoerbare versie van het programma, maar ook de documentatie, en informatie voor de `RPM` gegevensbank. Deze gegevensbank houdt bij welke `RPM` bestanden er reeds geïnstalleerd zijn, welke versies, en welke andere pakketten ervan afhankelijk zijn.

Om een stuk software te updaten met behulp van rpm, bijvoorbeeld versie 1.47 van hugs, kan men `rpm -U hugs-1.47-1.i386.rpm` intypen. De "`-U`" staat hier voor "update." Indien je niet weet of de software reeds aanwezig is op het systeem kan men ook `rpm -Ui hugs-1.47-1.i386.rpm` intypen wat de software gewoon installeert indien deze niet reeds aanwezig was op het systeem.

Een *RPM* pakket kan ook "ondervraagd" worden over zijn inhoud, via `rpm -qip package-naam`. Om te verifiëren of er niet met het package geknoeid is alvorens het te installeren kan men `rpm -V package-naam` gebruiken (let op, de V is een hoofdletter).

Bij installatie komt er informatie over het geïnstalleerde pakket terecht in de *RPM* gegevensbank: dat is een bestand dat deze informatie bijhoudt, en dat ook via het `rpm` bevel geconsulteerd kan worden. Bijvoorbeeld, `rpm -q hugs` vertelt je welke versie van hugs beschikbaar is. `rpm -qa|grep hugs` geeft alle gegevensbank-matches terug waarin het woord "hugs" voorkomt.

Software verwijderen is even eenvoudig als installeren. `rpm -e hugs` verwijdert het hugs pakket, ongeacht welke versie er geïnstalleerd is. `rpm` geeft je een

[1] http://www.rpm.org.

verwittiging in dien bij het verwijderen de afhankelijkheden met andere pakketten zouden verbroken worden.

`rpm` heeft ook een *verbose* optie: indien je meer informatie wenst afgedrukt te zien tijdens installatie, dan voeg je een "v" (kleine letter) toe aan de `rpm` opties.

`rpm` is een commando-lijn tool; RedHat en andere verdelers hebben er grafische interface-programma's rond gebouwd, eventueel in combinatie met support of betaalde ondersteuning voor automatische updates vanaf de servers van de verdeler. RedHat heeft zijn `up2date`, Mandrake zijn `urpmi`, Ximian zijn `red-carpet`.

14.1.2 De apt-tools

De meeste commerciële Linux-verdelers gebruiken het *RPM* formaat, maar de onafhankelijke Debian distributie biedt al vele jaren zijn eigen systeem aan, *apt* (Advanced Package Tool). Het is de combinatie van volgende componenten:

1. *`.deb` bestanden.* Een .deb bestand bevat alle informatie en software van een bepaald pakket: de uitvoerbare bestanden, de gedeelde bibliotheken, de documentatie, en de informatie die de afhankelijkheden van dit pakket met andere pakketten beschrijft. In principe is dit dezelfde inhoud als een *RPM* bestand; er bestaan tools zoals `alien` om de conversie te maken.

2. *`dselect` en `dpkg`.* Indien je op je systeem een hoop `.deb` bestanden hebt staan, dan helpen deze programma's je om een selectie te maken van welke bestanden je wil installeren, om de afhankelijkheden tussen je geselecteerde pakketten en de reeds geïnstalleerde pakketten te controleren, en om de installatie zelf ook effectief uit te voeren. Beide programma's zijn "laag niveau," dit wil zeggen dat je erg veel mogelijkheden hebt om alles precies te doen zoals je wenst, maar dat je daar een prijs voor betaalt in termen van complexiteit.

3. *`apt`.* Dit programma verhelpt aan bovenvermelde complexiteit, en is een tool bovenop `dselect` en `dpkg`, met een gebruiksvriendelijkere interface. Het verbindt de gebruiker met de software gegevensbank van Debian (of één van zijn talrijke mirrors wereldwijd).

4. *Oplossen van afhankelijkheden.* De kwaliteit van automatische installatie hangt niet alleen af van de software die de eigenlijke installatie doet (`apt`, `dselect` en `dpkg`), maar zeker evenveel van de kwaliteit van het *geheel* van de distributie. Deze laatste kwaliteit kan je subjectief meten aan de hand van hoevaak je manueel moet ingrijpen in het installatie-proces, omdat verschillende te installeren pakketten niet goed geconfigureerd zijn, of omdat er gebreken zijn in de afhankelijkheden.

5. *Gegevensbank van geïnstalleerde pakketten.* Dit is volledig gelijkaardig aan de RPM gegevensbank, maar nu voor `.deb` pakketten.

Debian scoort erg goed op het gebied van het consistent houden van de afhankelijkheden tussen software pakketten. Deze afhankelijkheden worden getest en opgelost door een team van enkele honderden vrijwilligers wereldwijd, die elk instaan voor de correcte configuratie van de `.deb` bestanden van één of meerdere projecten. Dit is zeer tijdsrovend werk, en Debian is dan ook de "traagste" verdeler van software. Maar de kwaliteit is doorgaans wel erg hoog, *en* Debian is beschikbaar op erg veel platformen (11 voor de 3.0 versie). Vandaar dat het `apt` systeem ook populair begint te worden bij andere verdelers.

Om met `apt` nuttig werk te kunnen doen moet je een bron hebben die je .deb bestanden kan aanleveren. Dat kan een CD zijn, maar in veel gevallen is een FTP of HTTP server het handigste om on-line nieuwe software te installeren, of nieuwe versies van reeds geïnstalleerde software. `apt` verwacht dat je in de `/etc/apt` directory een bestand `sources.list` hebt staan, waarin je informatie over de .deb servers bewaart. In zijn eenvoudigste vorm bevat dat bestand slechts één lijn, bijvoorbeeld:

```
deb http://ftp.kulnet.kuleuven.ac.be/debian/ woody main non-free contrib
```

(De server op `http://ftp.kulnet.kuleuven.ac.be` heeft een bijna volledige copie van de basis-server op `http://ftp.debian.org`, en is voor toegang in België erg geschikt. Er zijn er ook nog andere.) De lijn hierboven zegt ook dat ik enkel pakketten wens uit de `woody` versie van Debian, en daaruit enkel de `main`, `non-free` en `contrib` sub-archieven.

Apt heeft enkele deel-programma's:

- `apt-get`. (Werkt enkel met permissies van de superuser.) Hiermee doe je installatie van software, via de server(s) uit `/etc/apt/sources.list`. De typische procedure bestaat uit:

  1. `apt-get upgrade`. Dit brengt de lijst van beschikbare pakketten, zoals je die in je .deb gegevensbank op je eigen computer hebt staan, in overeenstemming met wat er op de server(s) ter beschikking is.
  2. `apt-get install pakket-xyz`. Dit doet de eigenlijke installatie. Eerst kijkt het of alle afhankelijkheden in orde zijn, en indien dit niet zo is zegt het welke pakketten extra nodig zijn, hoeveel plaats ze zullen innemen, en vraagt het of je de voorgestelde installatie wil laten doorgaan. Als je dit bevestigt, dan zie je eerst de pakketten opgehaald worden van de server, waarna ze geïnstalleerd en geconfigureerd worden.

  (Her-)configuratie en verwijderen van pakketten behoort ook tot de opties van `apt-get`. Voor alle details consulteer je best de man pagina.

- `apt-cache`. (Heeft geen superuser toelating nodig.) Hiermee kan je opvragingen doen aan je lokale .deb gegevensbank: welke versie van een bepaald pakket je hebt; tot welk pakket een bepaald programma behoort; wat de afhankelijkheden zijn; enz.

`apt` installeert zijn gegevensbank normaal gezien in `/var/cache/apt`, in binaire vorm. In die directory zie je ook de `archives` directory: die bevat alle .deb bestanden die je al geïnstalleerd hebt, en dat kan na verloop van tijd wel een wat plaats op je harde schijf in beslag nemen. Je kan dat archief verwijderen met `apt-get clean`.

`/var/lib/apt/list` is een andere standaard plaats waar `apt` gegevens bijhoudt, namelijk die van wat er op de server(s) ter beschikking is aan pakketten.

In beide bovenvermelde directories staan `lock` bestanden: je kan maar op één plaats tegelijk met `apt` bezig zijn met de gegevensbanken (lokaal of van de server), anders loop je het risico om ze inconsistent te maken.

14.2 Broncode pakket installeren

Veel software voor Linux systemen is beschikbaar in de vorm van broncode vergezeld van een *Makefile*. Deze Makefile beschrijft de afhankelijkheden en vereisten van de code en geeft aan hoe de code moet gecompileerd worden. De Makefile kan de volgende soorten lijnen bevatten:

- lege lijnen;
- commentaar lijnen, wordt altijd voorafgegaan door `#`;
- lijnen die afhankelijkheden aangeven;
- lijnen die suffixen voor bestandsnamen vastleggen;
- lijnen die commando's bevatten;
- macro definities, bijv.: `define naam\ string\ endef` of `naam =string`;
- include statements;

Een voorbeeld van een Makefile met wat uitleg vind je in appendix B.

Meestal zal de broncode verpakt zijn in een gecomprimeerd archief-bestand (zie sectie 4.6), d.w.z., een tar.gz, .tgz (dit is hetzelfde als tar.gz) of tar.bz2 bestand. Pak dit bestand uit, met `tar zxvf bestand.tar.gz` of `bzip2 -d bestand.tar.bz2; tar xvf bestand.tar`. Lees dan eerst het `README` bestand, dat bijna altijd in het archief aanwezig is. De README bevat informatie over het pakket, zoals de versie, de features, de gekend problemen, de adressen waar je de ontwikkelaats van het pakket kan bereiken, enz. Meestal is er ook een `INSTALL` bestand, met daarin de instructies voor de installatie van de broncode. In het meest typische geval moet je de volgende stappen doorlopen:

1. `./configure`, indien dit beval erbij zit moet men het uitvoeren. Het zal controleren of het systeem voldoet aan de vereisten van de software (alle nodige bibliotheken en tools zijn aanwezig) en de Makefiles generen;

2. `make`, dit bevel start de compilatie, en je krijgt de uitvoer ervan op je scherm te zien;

3. `make check`, dit is een niet verplichte tussenstap die de huidige toestand nakijkt (of alles wel in orde is);

4. `make install`, uit te voeren als *als superuser*, installeert de gecompileerde en gelinkte bestanden op je systeem;

5. `make clean`, dit ruimt de bestanden op die ontstaan zijn bij de compilatie, en die na de installatie overbodig zijn;

Hoofdstuk 15

Linux Advocacy - mini-HOWTO

Deze tekst werd geïnspireerd door Jon "maddog" Hall toen hij reageerde op een verzoek op feedback op richtlijnen voor het pleiten voor Linux tijdens de NetDay-activiteiten http://www.netday.org. Hij reageerde positief op de richtlijnen en merkte op dat deze de basis waren voor een lijst van "gedragsregels" die de Linuxgemeenschap vooruit zouden helpen.

Het originele (Engelstalige) document is beschikbaar in HTML formaat op
http://www.datasync.com/ rogerspl/Advocacy-HOWTO.html. Dit is de Nederlandse vertaling door Wim 'vvim' Deprez, wim@lumumba.luc.ac.be (9 september 2002) van de Linux Advocacy mini-HOWTO om richtlijnen en ideen te verschaffen die u kunnen helpen met uw inspanningen om voor Linux te pleiten.

15.1 Introductie

De Linuxgemeenschap is al een tijd bekend voor vele applicaties, Linux is een stabiel, betrouwbaar, robuust (hoewel niet perfect) product. Spijtig genoeg zijn er nog steeds veel mensen, waaronder belangrijke beleidsvormers, die nog niet weten van het bestaan van Linux en diens mogelijkheden.

Als Linux en de vele andere componenten die een Linux distributie maken, hun volle vermogen moeten bereiken, dan is het cruciaal dat we uitreiken naar toekomstige "klanten" en pleiten (wees tegelijk zorgzaam om niet te veel te beloven) voor het gebruik van Linux voor geschikte applicaties. De reden waarom vele bedrijfsproducten het zo goed hebben gedaan in de markt, is niet de technische superioriteit van het product, maar het marketingstalent van het bedrijf.

Als u dankbaar gebruik maakt van Linux en iets zou willen bijdragen tot de Linuxgemeenschap, overweeg dan alstublieft om één of meer van de ideeën uit deze

mini-HOWTO toe te passen en help zo anderen om meer over Linux te leren.

15.2 Pleiten voor Linux

- Deel uw persoonlijke ervaringen met Linux (zowel de goede als de slechte). Iedereen weet dat software bugs en beperkingen heeft en als we enkel lovende opmerkingen over Linux hebben, zijn we niet eerlijk. Ik hou ervan om te vertellen dat ik vier keer heb moeten rebooten (waarvan drie geplande) in drie jaar tijd.

- Als iemand een probleem met Linux heeft dat zou kunnen opgelost worden, stel dan voor om hem of haar door te wijzen naar gepaste informatie (webpagina's, artikels, boeken, specialisten, ...). Als u zelf de voorgestelde oplossing niet gebruikt hebt, zeg dat dan ook.

- Als u beschikbaar bent voor presentaties over Linux, registreer u dan bij het Linux Speakers Bureau [http://noframes.linuxjournal.com/lsb/listing.html].

- Bied aan om nieuwe Linux-gebruikers te helpen bij hun eerste stappen. Zorg er ook voor dat ze over de kennis beschikken om hun systeem efficiënt te gebruiken.

- Sommige mensen geloven nog steeds dat Linux en verwante besturingssystemen enkel in text-mode werken. Wijs hen op de aanwezigheid van grafische applicaties zoals de Gimp [http://www.gimp.org/].

- Probeer om iedere week op een "newbie" posting te reageren. Zoek de moeilijke vragen, u kan de enige zijn die antwoord en het is mogelijk dat u tegelijkertijd nog iets bijleert. Als u niet overtuigd bent dat u met het juiste antwoord kan reageren, vind dan iemand die dat wel kan.

- Zoek kleine bedrijven die zich bezig houden met het ontwikkelen van software en stel hen voor om een presentatie over Linux te maken.

- Als de kans er is, maak dan een presentatie voor uw werkgever zijn Information Technology groep.

- Doe mee aan activiteiten van de gemeenschap, zoals NetDay [http://www.netday.org/]. Terwijl uw eerste prioriteit het meewerken aan het succes van zulks een activiteit is, kan u de kans grijpen om anderen te laten weten wat Linux voor hen kan doen.

- Overweeg altijd de standpunten van degene die u Linux probeert te "verkopen". Steun, betrouwbaarheid, interoperabiliteit en kost zijn allemaal factoren die moeten overdacht worden, voor er een beslissing wordt genomen. Van de genoemden weegt kost meestal het minste door.

- De beschikbaarheid van steun wordt meestal genoemd als een belangrijke factor voor de aanschaf van Linux. Bedrijven zoals Caldera [http://www.caldera.com/], Cygnus Solutions [http://www.cygnus.com/], Red Hat [http://www.redhat.com/], en S.u.S.E. [http://www.suse.com/] bieden steun voor bepaalde of alle componenten van een typische Linux distributie. Daarbij komt nog de Linux Consultants HOWTO [http://www.linuxdoc.org/HOWTO/Consultants-HOWTO.html], die een lijst van bedrijven bevat die commerciële steun omtrent Linux verschaffen. Natuurlijk is 1 van de beste hulp te vinden op comp.os.linux en de Linux nieuwsgroep hoofdgroepen.

- Maak duidelijk dat de productie van open-source software [http://www.opensource.org/] plaats vind in een omgeving van openlijke samenwerking tussen systeemarchitecten, programmeurs, schrijvers, alfa/bèta testers en gebruikers die vaak resulteren in goed gedocumenteerde en krachtige producten zoals Apache [http://www.apache.org/], GNU Emacs [http://www.gnu.org/], Perl [http://www.perl.com/] en de Linux kernel [http://www.linuxhq.com/].

- Sta op en zorg dat u geteld wordt! Registreer u bij de Linux Counter [http://counter.li.org/].

- Bericht succesvolle inspanningen van het promoten van Linux aan Linux International (li@li.org) en gelijkaardige organisaties.

- Vind een nieuw thuis voor Linux CD-ROMs en boeken die u niet langer nodig hebt. Geef ze aan iemand die geïnteresseerd is in Linux, een openbare bibliotheek of een computerclub op school. Een boek en zijn CD-ROM zullen het meest geschikt zijn voor de bibliotheek. Zorg er wel voor dat het publiek beschikbaar maken van de CD-ROM niet in strijd is met een licentie overeenkomst of auteursrechten. Informeer het personeel van de bibliotheek dat het materiaal op de CD-ROM vrij verdeelbaar is. Controleer of het uiteindelijk beschikbaar is in de rekken.

- Geef bij aanschaf van boeken over software verdeeld met Linux de voorkeur aan boeken die geschreven zijn door de auteurs van de software. De royalties die de auteurs ontvangen van de boekverkoop zouden best wel eens de enigste inkomsten zijn die zij krijgen voor hun inspanningen.

- Moedig Linux-gebaseerde sites aan om zich kenbaar te maken aan de "Powered by Linux" [http://sunsite.nus.edu.sg/pub/LDP/powered.html] pagina en stel voor om banners die Linux [http://www.linux.org/], Apache [http://www.apache.org/], GNU [http://www.gnu.org/], Perl [http://www.perl.com/] ... promoten op hun site te zetten.

- Neem deel! Als u dankbaar gebruik maakt van open-source software [http://www.opensource.org/ overweeg dan om bij te dragen tot de free software-gemeenschap door:

  - gedetailleerde bug reports te maken
  - documentatie te schrijven
  - illustraties te creëren
  - management talenten te gebruiken
  - uitbreidingen voor te stellen
  - technische steun te verschaffen
  - software bij te dragen
  - benodigdheden te doneren
  - financile steun te bezorgen.

  Het Linux Documentation Project [http://www.linuxdoc.org/] voorziet een lijst [http://www.linuxdoc.org/devel.html] van Linux en gerelateerde projecten.

- Ten laatste: hou in gedachte dat we allemaal met oneindig veel belangrijker kwesties te maken hebben, dan het kiezen van een computeromgeving.

15.3 Gedragsregels

- Als vertegenwoordiger van de Linuxgemeenschap, neemt u op een professionele manier deel aan discussies in mailing lists en newsgroups. Doe niet mee aan scheldpartijen of het gebruik van vulgaire taal. Zie uzelf als een lid van een virtueel genootschap met Mr. Torvalds als uw Chief Executive Officer. Uw woorden kunnen het beeld van de lezer over de Linuxgemeenschap verhogen of verlagen.

- Vermijd altijd overdrijvingen en niet gestaafde uitspraken. Het is onprofessioneel en zal resulteren in nutteloze discussies.

- Een zorgzame, weldoordachte reactie op een posting zal niet enkel inzicht verschaffen aan uw lezers, maar zal ook hun respect vermeerderen voor uw kennis en bekwaamheid.

- Geef niet toe aan uitlokkingen. Te veel threads verlagen zich tot “Mijn O/S is beter dan uw O/S” argumenten. Beschrijf nauwkeurig de kwaliteiten van Linux en laat het daarbij.

- Onthoud altijd dat als u iemand beledigd of disrespectvol bent ten opzichte van iemand, deze negatieve ervaring gedeeld kan worden met vele anderen. Als u iemand beledigd, probeer het dan alstublieft weer goed te maken.

- Concentreer op het gene dat Linux te bieden heeft. Er is geen nood aan het onderuithalen van de concurrentie. Linux is een goed, krachtig product dat op zichzelf staat.

- Respecteer het gebruik van andere besturingssystemen. Hoewel Linux een geweldig platform is, beantwoordt het niet aan de noden van iedereen.
- Verwijs naar een ander product met de juiste naam. Men wint niets bij pogingen om een bedrijf of zijn producten te ridiculiseren door het gebruik van "creatieve spelling". Als wij respect voor Linux verwachten, moeten wij ook andere producten respecteren.
- Geef de eer aan degenen die het verdienen. Linux is niet meer dan een kernel. Zonder de inspanningen van mensen die betrokken zijn bij het GNU project [http://www.gnu.org/gnu/linux-and-gnu.html], het MIT, Berkeley en anderen - te veel om op te noemen - zou de Linux kernel niet erg bruikbaar zijn voor de meeste mensen.
- Dring er niet op aan dat Linux het enige antwoord is voor een bepaalde toepassing. Net zoals de Linuxgemeenschap de vrijheid liefhebben die Linux hen verschaft, zou het opdringen van Linux als enige oplossing anderen van hun vrijheid beroven.
- Er zullen gevallen zijn waarvoor Linux niet het antwoord is, wees de eerste om deze gevallen te herkennen en een andere oplossing aan te bieden.

15.4 User Groups

- Werk mee aan een lokale user group. Een index [http://www.linuxdoc.org/links/#lug] van de Linux User Group registries is deel van het Linux Documentation Project [http://www.linuxdoc.org/]. Als er geen user group bestaat in uw buurt, begin er dan één.
- De Linux User Group HOWTO [http://www.linuxdoc.org/HOWTO/User-Group-HOWTO.html] omvat verschillende punten betreffende het starten van een user group en bediscussieerd de belangrijkheid van het pleiten voor Linux als één van de doelen van een user group.
- Zorg voor sprekers voor organisaties die geïnteresseerd zijn in Linux.
- Publiceer over uw activiteiten in de lokale media.
- Bied aan om een Linux systeem te configureren naar de noden van lokale organisaties. Uiteraard moet het installatieproces ook de training van de user-gemeenschap omvatten, zodat zij het systeem kunnen gebruiken, samen met adequate documentatie voor het onderhoud van het systeem.
- Discussieer over de Linux Advocacy mini-HOWTO tijdens een vergadering. Brainstorm en leg nieuwe ideen voor.

15.5 Verkopersrelaties

- Als u nadenkt over de aanschaf van nieuwe hardware, vraag de verkoper dan over Linux support en de ervaringen van andere gebruikers van het product in een Linux omgeving.
- Overweeg om verkopers die Linux gebaseerde producten en diensten verkopen te steunen. Moedig hen aan om hun product te laten opnemen in de Linux Commercial HOWTO [http://www.linuxdoc.org/HOWTO/Commercial-HOWTO.html].
- Steun verkopers die een deel van hun inkomen doneren aan organisaties zoals de Free Software Foundation [http://www.gnu.org/help/help.html], het Linux Development Grant Fund [http://li.org/li/fund/grants.shtml], het XFree86 Project [http://www.xfree86.org/donations.html] of Software in the Public Interest [http://www.debian.org/donations.html]. Indien mogelijk, doe dan een persoonlijke donatie aan deze of andere organisaties die open-source software steunen [http://www.opensource.org/]. Vergeet niet dat sommige werkgevers een passend gift programma aanbieden.
- Als u een applicatie nodig heeft die niet ondersteund is op Linux, contacteer dan de verkoper en vraag een native Linux versie aan.

15.6 Mediarelaties

- Linux International verzamelt krantenknipsels [http://www.li.org/li/resources/pressclippings.shtr die Linux, GNU en andere vrij verkrijgbare software vermelden. Als u zo een artikel tegenkomt, zend dan alstublieft de volgende informatie naar clippings@li.org:
  - Naam van de publicatie
  - Het contact adres van de uitgever
  - Naam van de auteur
  - Het contact adres van de auteur
  - Titel van het artikel
  - Paginanummer vanwaar het artikel begint
  - De URL als het online beschikbaar is
  - Een samenvatting van het artikel en een eigen opinie
- Als u gelooft dat Linux niet eerlijk behandeld werd in een artikel, een bespreking of het nieuws, zend dan de details, samen met de bovenstaande informatie, naar li@li.org zodat een geschikte reactie naar de uitgever gestuurd kan worden. Als u de uitgever zelf contacteert, wees dan professioneel en zeker van uw feiten.

- Als u betrokken bent bij een Linux gerelateerd project, verstrek dan persberichten aan geschikte nieuwsdiensten op een regelmatige basis.

Hoofdstuk 16

Linux op het web

Het Internet is het belangrijkste distributie kanaal voor Linux en software voor het Linux systeem. Deze sectie geeft een aantal links naar websites waar men software kan afhalen of meer informatie kan vinden. De volgende links zijn een greep uit het ruime aanbod, en slechts enkele categoriën. Voor meer uitgebreide lijsten kan je surfen naar http://www.freshmeat.net, http://linux.davecentral.com en naar http://www.linux.com. Je kan ook op de nieuwsgroep *be.comp.os.linux* terecht met je vragen.

16.1 Distributies

Er zijn verschillende Linux distributies beschikbaar, die men van het web kan afhalen. Dit zijn de bekendste distributies:

RedHat Linux http://www.redhat.com

Mandrake Linux http://www.mandrake.com

Debian Linux http://www.debian.org

Suse Linux http://www.suse.de

Slackware Linux http://www.slackware.com

Caldera Linux http://www.caldera.com

16.2 Kantoor-applicaties

Onder kantoorapplicaties verstaan we tekstverwerkingsprogramma's, spreadsheets en presentatie-programma's.

OpenOffice http://www.openoffice.org

KOffice http://www.koffice.org/

Gnumeric http://www.gnome.org/projects/gnumeric/

Abiword http://www.abiword.org/

klat http://lumumba.luc.ac.be/jori/klat/klat.html

16.3 Software ontwikkeling

Programmeren De lijst met alle beschikbare software om aan software ontwikkeling te doen is zeer lang. We beperken ons hier tot degene die bruikbaar kunnen zijn voor de richting Informatica-Kennistechnologie zoals gegeven aan het LUC.

Free Pascal http://www.freepascal.org

KDevelop http://www.kdevelop.org

gcc http://gcc.gnu.org/

gdb http://sources.redhat.com/gdb/

Borland JBuilder http://www.borland.com/jbuilder/

Qt http://www.trolltech.com/products/qt/

SWI Prolog http://swi.psy.uva.nl/projects/SWI-Prolog/

GNU Prolog http://pauillac.inria.fr/~diaz/gnu-prolog/

Scheme http://www.swiss.ai.mit.edu/projects/scheme/

Hugs http://www.haskell.org/hugs/

IBM JSDK 1.3 http://www.ibm.com/java/jdk/index.html

Sun JSDK 1.3 http://java.sun.com/j2se/1.3/

Freebuilder http://www.freebuilder.com/

Gegevensbanken Er zijn ook vele gegevensbanken beschikbaar voor Linux.

MySQL http://www.mysql.com/

PostgreSQL http://www.postgresql.org/

Sybase http://www.sybase.com/linux/

IBM DB2 http://www-4.ibm.com/software/data/db2/linux/

16.4 Meer informatie

Linux Documentation Project http://www.linuxdoc.org/

UK.LINUX.ORG http://www.linux.org.uk/

Brave GNU World http://www.bgw.org/tutorials/

GNU's Not Unix! http://www.gnu.org/

Bash Reference Manual http://www.gnu.org/manual/bash/index.html

Linux Online http://www.linux.org/

Linux.com http://www.linux.com/

ddj Linux http://www.ddj.com/topics/linux/

Bijlage A

Pointers naar functies

De volgende code listing laat een paar voorbeelden zien van functies die door middel van een verwijzing naar de functie als parameter doorgegeven worden aan andere functies. Een functie is eigenlijk niet meer als een verwijzing naar een adres waar de code van de desbetreffende functie daadwerkelijk begint. Functies zijn in feite een beetje zoals datastructuren: omdat zij ook in het geheugen opgeslagen worden is er ook een beginadres hiervoor te vinden.

Listing A.1: callbackf.c

```
#include<stdio.h>

int function1(void){
  printf("this is function 1\n");
}

int function2(int (*p)()){
  printf("function 2 printing:\n");
  p();
  printf("function 2 printed\n");
}

double somfun(int n, double (*f)(int k)){
  double s = 0;
  int i;
  for (i=1;i<=n;i++)
    s += f(i);
  return s;
}

double term(int k){
  return 2*k+1/k;
}

int main(void){
```

```
  double som;
  int (*p)();
  int function1();
  p = function1;
  function1();
  function2(p);
  function2(function1);
  p();
  som = somfun(3, term);
  printf("%lf\n", som);
  return 0;
}
```

Als we het programma van listing A.1 compileren en uitvoeren krijgen we als output:

```
this is function 1
function 2 printing:
this is function 1
function 2 printed
function 2 printing:
this is function 1
function 2 printed
this is function 1
13.000000
```

We kunnen `p` laten wijzen naar `function1`, door p als volgt te definiëren: `int (*p)();`. Nu heeft `p` de juiste typering om er `function1` aan toe te kennen: `p = function1;`. Vervolgens kan `function2` als parameter een variabele ontvangen die op dezelfde manier als `p` en dus ook `function1` gedefinieerd is. M.a.w. we geven `p` mee als argument aan `function2`, en in de body van deze function wordt p dan opgeroepen. Anderzijds kunnen we ook gewoon `function1` als parameter meegeven aan `function2`. In feite wordt dan het adres van de functie meegegeven, zodanig dat als men dan een verwijzing naar dat adres neemt, men eigenlijk aan de eerste statement van de functie zelf zit.

Tenslotte wordt er nog een praktisch voorbeeldje gegeven in listing A.1 door de functie `term` als argument aan `somfun` mee te geven. Probeer zelf eens een andere implementatie voor `term` te verzinnen en dan `somfun` terug op te roepen. Je zal zien dat het resultaat in `som` dan ook verandert.

Bijlage B

Voorbeeld van een Makefile

Het programma make is zeer handig om een "onderhoudbaar" en makkelijk compilatie proces te voorzien. Het gebruikt hiervoor de zogenaamde *Makefile*. In het voorbeeld (listing B.1) kan je een aantal macro-definities zien zoals `PDF = pdfelatex`. Alles wat achter # staat is commentaar. De macro-definitie PDF wordt opgeroepen door $(PDF). Het $ teken geeft aan dat het de variabele moet "expanden". Een lijn als `Linux2001v2.pdf:Linux2001v2.tex ...` wil zeggen dat als er een verandering waargenomen is in een van de TEX files (algemener één van de files achter het dubbel punt) de pdf-file moet geupdate worden. Dit geeft aan dat er een afhankelijkheid bestaat tussen het bestand voor het dubbelpunt en de bestanden achter het dubbelpunt. Deze regel is de "hoofdregel" van de Makefile, als je het commando `make` intypt zal het standaard de eerste regel van de vorm `all : <bronA> <bronB> ...` nemen. Hier is dit gedefinieerd als `all: Linux2001v2.pdf`. De update gebeurt door de lijnen die op de afhankelijkheidsregel volgen uit te voeren. Als je `make clean` intypt zal alleen de sectie die begint met `clean:` in de Makefile uitgevoerd worden. Idem voor `make html`.

Listing B.1: de Makefile voor dit document

```
# De compilers:
DVI = latex # for a DVI file
PS = dvips # for a postscript file
PDF = pdflatex # for a pdf file
THUMB = thumbpdf
BIB = bibtex # for generating the BibTeX entries
INDEX = makeindex # for generating the index

# de hoofdfile van de tekst
FILE = Linux-2006-v2.2
TEXFILES = basiscommandos.tex devicedr.tex\
  dir-and-files.tex inleiding.tex linonweb.tex\
  linprogr.tex shellprogr.tex softinstall.tex\
  app-callback.tex app-makefile.tex bibliotheken.tex\
  oploefshell.tex editors.tex regexpr.tex linuxcourse.bib\
```

```
  bijdragen.tex grmail.txt edm.sty awk.tex rcs.tex\
  processes-threads.tex proces-programmeren.tex X.tex\
  threads-programmeren.tex kernel-compile-install.tex\
  kernel.tex advocacy.tex gnu-lic.tex


# de flags voor de DVI compiler en de output file
DVI2PSFLAGS = -o $(FILE).ps
DVIFILE = $(FILE).dvi

# welke bestanden mogen weg gedaan worden
DELETABLE = *.aux *.toc *.out *.ind *.lof *.idx *.ilg *.lot\
  *.blg *.bbl *.lol *.lo *.brf *.*~ *.tpt
CLEANABLE = *[^e]ps *pdf *.tex~ *.dvi *.ps.gz *.*~ *.bak


SUBDIR = figs


all: $(FILE).pdf $(FILE).ps

# de hoofdregel van deze Makefile
$(FILE).pdf: $(FILE).tex
  make pdf

$(FILE).ps: $(FILE).tex $(TEXFILES)
  make ps

$(FILE).html: $(FILE).tex $(TEXFILES)
  make html

pdf:
  @make -C figs
  $(PDF) $(FILE) #pdflatex moet je een paar keer uitvoeren
  $(PDF) $(FILE)
  $(BIB) $(FILE)
  $(PDF) $(FILE)
  $(THUMB) $(FILE).pdf
  $(INDEX) $(FILE).idx
  $(PDF) $(FILE)
  $(PDF) $(FILE)
  rm -rfv $(DELETABLE) #verwijder intermediate files

ps:
  @make -C figs
  $(DVI) $(FILE) #latex moet je een paar keer uitvoeren
  $(DVI) $(FILE)
  $(BIB) $(FILE)
  $(DVI) $(FILE)
```

```
  $(INDEX) $(FILE).idx
  $(DVI) $(FILE)
  $(DVI) $(FILE)
  $(PS) $(DVI2PSFLAGS) $(DVIFILE)
  rm -rfv $(DELETABLE) #verwijder intermediate files
  cp $(FILE).ps temp.ps
  rm -fv $(FILE).ps.gz
  gzip $(FILE).ps
  mv temp.ps $(FILE).ps

html:
  @make figs
  @make ps
  latex2html -mkdir -dir html-versie\
    -address "editor: <a href=\"mailto:kris.luyten@luc.ac.be\">Kris Luyten</A>, experimentele
    -local_icons -numbered_footnotes -show_section_numbers -split 5 $(FILE).tex

# ruim de boel op, alles behalve de benodigde sources
clean:
  @make -C figs clean
  rm -rfv $(CLEANABLE) $(DELETABLE)
```

Let erop dat de regels voor de Makefile met een tab moeten beginnen!

Nu werkt dit voorbeeldje met LaTeX files als input en een pdf file als output. Hetzelfde kan gedaan worden voor een hoop object files, waarvoor c files moeten gecompileerd worden waarvan een programma afhangt. Een voorbeeld hiervan vind je in listing B.2.

Listing B.2: Een simpele Makefile

```
quantumsearch: main.o q.o bubble.o
   gcc -o quantumsearch main.o q.o bubble.o

main.o: main.c
   gcc -c main.c -o main.o

q.o: q.c q.h
   gcc -c q.c -o q.o

bubble.o: bubble.c bubble.h
   gcc -c bubble.c -o bubble.o
```

De listing B.2 zegt dat de (executable) file `quantumsearch` afhangt van `main.o`, `q.o` en `bubble.o`. (door de regel `quantumsearch: main.o q.o bubble.o`). De volgende regel in listing B.2 geeft aan wat er moet gebeuren om `quantumsearch` terug up-to-date te krijgen. We geven op hun beurt ook aan waarvan `main.o`, `q.o` en `bubble.o` afhangen. Indien blijkt dat één van deze bestanden niet up-to-date is, zullen eerst de bijbehorende opdrachten geactiveerd worden, waarna verder gegaan wordt met `quantumsearch` bij te werken.

De twee voorbeelden, listing B.1 en listing B.2 , geven duidelijk aan dat Makefiles handig zijn in allerhande software projecten. Ze worden vooral gebruikt om software projecten bestaande uit meerdere bestanden van broncode op een efficiënte manier te compileren, het overbodige compileer werk vermijdend. Het zal de up-to-date files niet hercompileren, en enkel de files die aangepast zijn hercompileren. Een Makefile kan dus gebruikt worden wanneer we een bepaalde output willen verkrijgen, afhankelijk van één of meerder input files.

Bijlage C

Oplossingen oefeningen

Oplossingen oefeningen uit sectie 8.12:

Listing C.1: oplossing oef 2

```
#!/bin/bash
if [ ! -d ~/backup ]
then
  mkdir ~/backup
fi

movedate=‘date +%s‘
tar -c $@ | bzip2 > ~/backup/$movedate.tar.bz2
chmod uago-wx ~/backup/$movedate.tar.bz2
```

Listing C.2: oplossing oef 3

```
#!/bin/bash
show_me_your_stuff(){
  ls -l $1
  cat $1
}

i=1
while [ $i -le $# ]
do
  show_me_your_stuff $1
  shift
done
```

Listing C.3: oplossing oef 4

```
#!/bin/bash
blaai=/tmp/attachment.tar.gz
if [ -d $1 ]
then
```

```
  tar -cv $1 | gzip > $blaai
  mail `whoami` < $blaai
else
 echo "$1 is geen directory"
fi
```

Listing C.4: oplossing oef 5

```
#!/bin/bash
eval text='$'$#
maxnr=$#
i=1
while [ $i -lt $maxnr ]
do
 eval receiver='$'$i
 mail $receiver < $text
 i=`expr $i + 1`
done
```

Listing C.5: oplossing oef 6

```
#!/bin/bash
until who | grep "$1" >/dev/null
do
  sleep 10
done

echo "$1 is ingelogd"
```

Listing C.6: oplossing oef 7

```
#!/bin/bash
while true
do
    i=1
    while [ $i -le $# ]
   do
      eval user='$'$i
     iser=`who | grep "$user"`
     if [ -n "$iser" ]
     then
      echo "user $user is er"
     else
      echo "user $user is er niet"
     fi
     i=` expr $i + 1 `
   done
   sleep 10
done
```

Listing C.7: oplossing oef 8

```
#!/bin/bash
while true
do
   echo "geef een commando:"
  read jobname
  if [ "$jobname" = "quit" ]
  then
    exit
  fi
  starttijd=‘exec date‘
  echo "$jobname starts at $starttijd" >> jobaccounting
  $jobname
  eindtijd=‘exec date‘
  echo "$jobname ends at $eindtijd" >> jobaccounting
done
```

Bijlage D

GNU Free Documentation License

A-1 Voorwoord

Dit is een vertaling gebaseerd op de Engelse en Duitse versies.
Dit is een officieuze vertaling van de GNU Free Documentation License (GFDL), Version 1.1. Ze wordt niet door de Free Software Foundation uitgegeven, en legt niet de wettelijke verspreidingsvoorwaarden vast voor documenten die de GFDL gebruiken; dat doet enkel de originele Engelse tekst van de GFDL (http://www.gnu.org/copyleft/fdl.htm

De originele versie van de GFDL bevindt zich op http://www.gnu.org/copyleft/fdl.html
Het doel van deze Licentie is om een handleiding, boek, of ander geschreven document "vrij" te maken: om iedereen de effectieve vrijheid te verzekeren om het te kopiren en te herverspreiden, met of zonder wijzigingen, zowel commercieel als niet-commercieel. In tweede instantie bezorgt deze Licentie de auteur en uitgever een manier om krediet te krijgen voor hun werk, terwijl ze toch niet verantwoordelijk gesteld kunnen worden voor wijzigingen aangebracht door anderen.

Deze Licentie is een soort "copyleft", wat betekent dat afgeleide werken van dit document zelf ook vrij moeten zijn in dezelfde betekenis. De Licentie is een aanvulling op de GNU General Public License, die een copyleft licentie is voor vrije software.

We hebben deze Licentie ontworpen om te gebruiken voor handleiding voor vrije software, omdat vrije software vrije documentatie nodig heeft: een vrij programma zou moeten begeleid worden door handleidingen die dezelfde vrijheid leveren als de software. Maar deze Licentie is niet beperkt tot software-handleidingen; ze kan gebruikt worden voor elke tekstueel werk, ongeacht het onderwerp of het feit of het gepubliceerd wordt als gedrukt boek. We raden deze Licentie vooral aan voor instructie- of referentie-werken.

A-2 Toepasbaarheid en definities

Deze Licentie is van toepassing op elke handleiding of ander werk dat een nota van de auteursrechterlijke eigenaar bevat die vermeldt dat het werk kan verspreid worden onder de voorwaarden van deze Licentie. Het "Document", hieronder, verwijst naar elk dusdanige handleiding of werk. Ieder lid van het publiek is een licentiehouder, en wordt aangesproken als "u".

Een "Gewijzigde Versie" van het Document betekent elk werk dat het Document bevat, of een gedeelte ervan, ofwel letterlijk gekopiëerd, ofwel met wijzigingen en/of vertaald in een andere taal.

Een "Secundaire Sectie" is een benoemde appendix of een front-matter sectie van het Document dat uitsluitend handelt over de relatie tussen de uitgevers of auteurs van het Document enerzijds en het algemene onderwerp van het Document (of gerelateerde onderwerpen) anderzijds, en bevat niets dat rechtstreeks te maken heeft met dat algemene onderwerp. (Bijvoorbeeld, als het Document gedeeltelijk een wiskundeboek is, dan mag een Secundaire Sectie geen wiskunde beschrijven.) De relatie zou kunnen zijn: een historische connectie met het onderwerp of met aanverwante materie, of een wettelijk, commercieel, filosofisch, ethisch of politiek standpunt terzake.

De "Invariante Secties" zijn zekere Secundaire Secties wiens titels aangeduid zijn als zijnde deze van Invariante Secties, in de nota die zegt dat het Document vrijgegeven is onder deze Licentie.

De "Schutblad-teksten" zijn korte tekstpassages die als Voorpagina-Tekst of als Achterflap-Tekst zijn vermeld in de nota die zegt dat het Document vrijgegeven is onder deze Licentie.

Een "Transparante" kopie van het Document is een machine-leesbare kopie, gepresenteerd in een formaat wiens specificatie beschikbaar is voor het brede publiek, wiens inhoud kan bekeken en ge-editeerd worden op een onmiddellijke en voor de hand liggende wijze, met algemene tekst-editors, of (voor beelden bestaande uit pixels) met algemene bitmap-programma's, of (voor tekeningen) met algemene tekenprogramma's, en dat geschikt is als invoer voor tekstverwerkers of voor automatische vertaling naar een waaier van formaten geschikt voor invoer voor tekstverwerkers. Een kopie gemaakt in een doorgaans Transparant bestandsformaat wiens opmaak ontworpen is om verdere aanpassing door lezers moeilijk te maken of te ontmoedigen is niet Transparant. Een kopie die niet "Transparant" is wordt "Ondoorzichtig" genoemd.

Voorbeelden van geschikte formaten voor Transparante kopiën omvatten norma-

le ASCII zonder opmaak, Texinfo invoerformaat, LaTeX invoerformaat, SGML of XML gebruik makende van een publiek beschikbare DTD, en standaarden-getrouwe eenvoudige HTML geschikt voor menselijke aanpassing. Ondoorzichtige formaten omvatten PostScript, PDF, proprietaire formaten die alleen kunnen gelezen en ge-editeerd worden met proprietaire tekstverwerkers, SGML of XML waarvoor de DTD of bewerkings-tools niet algemeen beschikbaar zijn, en machine-gegenereerde HTML die sommige tekstverwerkers produceren en die alleen geschikt is voor uitvoer-doeleinden.

De "Titelpagina" betekent, voor een gedrukt boek, de titelpagina zelf, plus al de volgende pagina's nodig om, in leesbare vorm, het materiaal te bevatten dat door deze Licentie verplicht aanwezig moet zijn in de titelpagina. Voor werken in formaten die geen eigenlijke titelpagina hebben, betekent "Titelpagina" alle tekst in de meest prominente nabijheid van de titel, voorafgaand aan het begin van de eigenlijke tekst.

A-3 Letterlijk kopiëren

U mag het Document kopiren en verspreiden op elk medium, zowel commercieel als niet-commercieel, onder voorwaarde dat deze Licentie, de auteursrechten-tekst, en de licentie-tekst die zegt dat deze Licentie van toepassing is op het Document, gereproduceerd worden in alle kopien, en dat u geen enkele andere voorwaarde aan deze Licentie toevoegt. U mag geen technische maatregelen nemen om het lezen of verder kopiren van de kopiën die u maakt of verspreidt, te bemoeilijken of te controleren. U mag echter wel compensaties aanvaarden in ruil voor kopiën. Indien u een voldoende groot aantal kopiën verspreid dan moet u ook de voorwaarden in Sectie A-4 navolgen.

U mag ook kopiën uitlenen, onder dezelfde voorwaarden als hierboven bepaald, en u mag kopiën openbaar vertonen.

A-4 Kopiëren in grote hoeveelheden

Indien u gedrukte kopiën van het Document publiceert in een oplage van meer dan 100, en de licentie-tekst van het Document vereist Schutblad-teksten, dan moet u de kopiën insluiten in een schutblad dat, duidelijk en leesbaar, al deze Schutblad-teksten bevat: Voorpagina-tekst op de voorpagina, en Achterflap-teksten op de achterpagina. Beide pagina's moeten ook duidelijk en leesbaar uzelf identificeren als de uitgever van deze kopiën. De voorpagina moet de hele titel tonen, met al zijn woorden even prominent en leesbaar aanwezig. U mag ander materiaal aan de voor- en achterpagina toevoegen. Kopiren met wijzigingen aan de voor- en achterpagina alleen, kunnen beschouwd worden als letterlijke kopien, zolang zij de titel van het Document behouden en aan bovenstaande voorwaarden voldoen.

Indien de vereiste teksten voor voor- of achter-pagina te volumineus zijn om leesbaar te blijven, dan moet u de eerst-vermelde teksten op de eigenlijke voor- en achterpagina plaatsen (zoveel als er redelijkerwijs op passen), en de rest op naburige pagina's.

Indien u Ondoorzichtige kopiën van het Document verdeelt in een oplage van meer dan 100, dan moet u ofwel een machine-leesbare Transparante kopie leveren bij elke Ondoorzichtige kopie, of in elke Ondoorzichtige kopie een publiekelijk bereikbare computer-netwerk-locatie vermelden waar een volledige Transparante kopie van het Document ter beschikking is, vrij van toegevoegd materiaal, en waartoe het algemene netwerk-gebruikende publiek toegang heeft om anoniem en kosteloos af te halen, gebruik makende van publieke-standaard netwerk-protocols. Indien u deze laatste optie gebruikt, en wanneer u begint met de verspreiding van Ondoorzichtige kopiën in grote oplagen, dan moet u gerede voorzorgen nemen om te verzekeren dat deze Transparante kopie toegankelijk blijft op deze locatie tot ten minste een jaar na de laatste keer dat u een Ondoorzichtige kopie van deze uitgave heeft verspreid onder het publiek (rechtstreeks of door uw agenten of verdelers).

Het is aangeraden, maar niet verplicht, dat u de auteurs van het Document contacteert ruim vooraleer u een groot aantal kopiën herverspreid, om hen de kans te geven u te voorzien van een geactualiseerde versie van het Document.

A-5 Wijzigingen

U mag een Gewijzigde Versie van het Document kopiren en verspreiden onder de voorwaarden van Secties 2 en 3 hierboven, op voorwaarde dat u de Gewijzigde Versie vrijgeeft onder precies deze zelfde Licentie, waarbij de Gewijzigde Versie de rol van Document krijgt, om aldus de verdeling en wijziging van de Gewijzigde Versie in licentie te geven aan gelijk wie een kopie ervan bezit. Bovendien moet u de volgende dingen doen in de Gewijzigde Versie:

1. Gebruik in de Titel-pagina (en op de voor- en achter-pagina's, indien aanwezig) een verschillende titel dan het Document, en van alle vorige versies (die, indien bestaande, zouden moeten aanwezig zijn in de Geschiedenis-sectie van het Document). U mag dezelfde titel gebruiken als een vroegere versie indien de oorspronkelijke uitgever van die versie de toelating geeft.

2. Vernoem op de Titel-pagina, als auteurs, een of meer personen of entiteiten die verantwoordelijk zijn voor het auteurschap van de wijzigingen in de Gewijzigde Versie, tezamen met ten minste vijf van de belangrijkste auteurs van het Document (al de belangrijkste auteurs, indien het er minder dan vijf zijn).

3. Vermeld als uitgever op de Titel-pagina de naam van de uitgever van de Gewijzigde Versie.

4. Behoud al de auteursrechterlijke teksten van het Document.

5. Voeg een aangepaste auteursrechterlijke tekst toe van uw wijzigingen bij de andere auteursrechterlijke teksten.

6. Voeg een licentie-tekst toe, onmiddellijk achter de auteursrechterlijke teksten, die het publiek de toelating geeft om de Gewijzigde Versie te gebruiken onder de voorwaarden van deze Licentie, onder de vorm zoals getoond in het Addendum hieronder.

7. Behoud in die licentie-tekst de volledige lijst van Invariante Secties en de vereiste Schutblad-teksten die in de licentie-tekst van het Document voorkomen.

8. Voeg een ongewijzigde versie van deze Licentie toe.

9. Behoud de sectie met als titel "Geschiedenis", alsook die titel zelf, en voeg er een paragraaf aan toe die, op zijn minst, melding maakt van de titel, jaar, nieuwe auteurs, en uitgever van de Gewijzigde Versie zoals gegeven op de Titel-pagina. Indien er geen sectie met als titel "Geschiedenis" bestaat, maak er dan één aan die melding maakt van de titel, jaar, nieuwe auteurs, en uitgever van de Gewijzigde Versie zoals gegeven op de Titel-pagina, en voeg dan een een paragraaf toe die de Gewijzigde Versie beschrijft zoals in de vorige zin aangegeven.

10. Behoud de netwerk-locatie, indien bestaande, die in het Document aangegeven is voor publieke toegang tot een Transparante kopie van het Document, en alsook de netwerk-locaties aangegeven in het Document voor de vorige versies waarop het gebaseerd was. Deze verwijzingen mogen geplaatst worden in de "Geschiedenis" sectie. U mag een netwerk-locatie schrappen foor een werk dat gepubliceerd is ten minste vier jaar vòòr het Document zelf, of na toelating van de oorspronkelijke uitgever van de versie waarnaar het verwijst.

11. Behoud in elke sectie met de titel "Dankwoord" of "Erkenning" de titel van die sectie, en behoud in de sectie al de inhoud en de toon van elk van de bijgedragen dankwoorden en/of uitingen van erkenning.

12. Behoud al de Invariante Secties van het Document, met ongewijzigde tekst en titels. Sectie-nummers en dergelijke worden niet beschouwd als deel uitmakend van de titel.

13. Schrap elke sectie met als titel "Goedkeuringen". Zo'n sectie mag niet aan de Gewijzigde Versie toegevoegd worden.

14. Geef geen enkele bestaande sectie de titel “Goedkeuringen”, of breng geen enkele sectie-titel in conflict met een Invariante Sectie.

Indien de Gewijzigde Versie nieuw front-matter secties bevat of appendices die in aanmerking komen om als Secundaire Secties beschouwd te worden, en die geen materiaal bevatten dat gekopiëerd is van het Document, dan mag u naar uw eigen goeddunken enkel of alle van deze secties als Invariant aanwijzen. Hiertoe voegt u hun titels toe aan de lijst van de Invariant Secties in de licentie-tekst van de Gewijzigde Versie. Deze titels moeten anders zijn dan alle andere sectie titels.

U mag een sectie met als titel “Goedkeuringen” toevoegen, op voorwaarde dat het niets anders bevat dan goedkeuringen van uw Gewijzigde Versie d oor verschillende partijen, bijvoorbeeld, bewijzen van peer review, of dat de tekst door een organisatie goedgekeurd is als de gezaghebbende definitie van een standaard.

U mag een passage van maximaal vijf woorden toevoegen aan de Voorpagina-tekst, en een passage van maximaal 25 woorden aan de Achterflap-tekst, aan het einde van de lijst van voorpagina- en achterflap-teksten in de Gewijzigde Versie. Slechts één passage van de Voorpagina-tekst en één van de Achterflap-tekst mag toegevoegd worden door gelijk welke entiteit (of door haar bemiddeling). Indien het Document reeds een voorpagina- of achterflap-tekst bevat, die vroeger door u of door bemiddeling door dezelfde entiteit op wiens gezag u optreedt, dan mag u geen nieuwe toevoegen; maar u mag wel de oude vervangen, na expliciete toelating van de vorige uitgever die de oude passage heeft toegevoegd.

De auteur(s) en uitgever(s) van het Document geven met deze Licentie geen toelating om hun namen te gebruiken voor publicitaire doeleinden, of om goedkeuring te laten gelden of te vooronderstellen voor gelijk welke Gewijzigde Versie.

A-6 Documenten combineren

U mag het Document combineren met andere documenten die met deze Licentie zijn verspreid, onder de voorwaarden voor gewijzigde versie besproken in Sectie A-5 hierboven, op voorwaarde dat u in de combinatie alle Invariante Secties van al de oorspronkelijke documenten insluit, ongewijzigd, en ze alle in de licentie-tekst opsomt als Invariante Sectie van uw gecombineerde werk.

Het gecombineerde werk moet slecht één kopie van deze Licentie bevatten, en meerdere identieke Invariante Secties mogen vervangen worden door één enkele kopie. Indien er meerdere Invariante Secties zijn met dezelfde naam maar verschillende inhoud, dan maakt u de titel van elk van deze secties uniek door er aan het eind, en tussen haakjes, de naam van de oorspronkelijke auteur of uitgever (indien deze gekend zijn), of anders een uniek nummer, aan toe te voegen. Maak

dezelfde aanpassingen aan de sectie-titels in de lijst van de Invariante Secties in de licentie-tekst van het gecombineerde werk.

In het gecombineerde werk moet u alle secties , die in de oorspronkelijke documenten de titel "Geschiedenis" droegen, combineren in één sectie met als titel "Geschiedenis"; hetzelfde geldt voor de secties met als titel "Dankwoord", of "Erkenning". U moet alle secties met als titel "Goedkeuringen" verwijderen.

A-7 Verzamelen van documenten

U mag een verzameling maken van het Document en andere documenten die onder deze licentie zijn verspreid, en de individuele kopiën van deze Licentie in de afzonderlijke documenten vervangen door een enkel kopie die in de verzameling wordt ingesloten, op voorwaarde dat u al de regels volgt van deze Licentie aangaande het letterlijk kopiren van elk van de documenten.

U mag een afzonderlijk document uit zo'n verzameling afzonderen, en het afzonderlijk verspreiden onder deze Licentie, op voorwaarde dat u een kopie van deze Licentie in het afgezonderde document insluit, en dat u deze Licentie volgt in alle voorwaarden aangaande het letterlijk kopiren van dat document.

A-8 Samenvoegen met onafhankelijke werken

Een compilatie van het Document of zijn afgeleide documenten met andere afzonderlijke en onafhankelijke documenten of werken, in of op een opslag- of verdeelmedium, telt niet in zijn geheel als een Gewijzigde Versie van het Document, op voorwaarde dat geen compilatie auteurs-recht geclaimd wordt op de compilatie. Zo'n compilatie wordt een "samenvoeging" genoemd, en deze Licentie is niet van toepassing op de andere op zichzelf staande werken die met het Document worden samengevoegd, op basis van het samenvoegen zelf, indien zijzelf geen afgeleide werken van het Document zijn.

Indien de Voorpagina-tekst eisen van sectie A-4 van toepassing zijn op deze kopiën van het Document, en indien het Document minder dan een vierde van de gehele samenvoeging uitmaken, dan mogen de Voorpagina- en Achterflap-teksten van het Document geplaatst worden op schutbladen die enkel het Document omvatten binnenin de hele samenvoeging. Anders moeten zij op de schutbladen van de hele samenvoeging geplaatst worden.

A-9 Vertaling

Vertaling wordt beschouwd als een soort van wijziging, dus mag u vertalingen van het Document verspreiden onder de voorwaarden van sectie A-5. Het vervangen van Invariante Secties door vertalingen vereist speciale toelating van de auteursrechterlijke eigenaars, maar u mag vertalingen van sommige of van alle Invariante Secties toevoegen bovenop de oorspronkelijke versies van deze Invariante Secties. U mag een vertaling van deze Licentie toevoegen, op voorwaarde dat u ook de oorspronkelijke Engelse versie van deze Licentie toevoegt. In geval van niet-overeenstemming tussen de vertaling en de oorspronkelijke Engelse versie van deze Licentie is het de oorspronkelijke Engelse versie die van toepassing is.

A-10 Beëindiging

U mag het Document niet kopiren, wijzigen, in licentie geven, of verspreiden, tenzij zoals uitdrukkelijk voorzien in deze Licentie. Elke andere poging tot kopiren, wijzigen, in licentie geven, of verspreiden van het Document is ongeldig, en zal onmiddellijk uw rechten onder deze Licentie beëindigen. Andere partijen, echter, die via u onder deze Licentie kopiën of rechten hebben verkregen zullen hun licenties niet beëindigd zien zolang ze zelf in volle overeenstemming blijven met de Licentie.

A-11 Toekomstige revisies van deze licentie

De Free Software Foundation kan van tijd tot tijd nieuwe, gereviseerde versie verspreiden van de GNU Free Documentation License. Zulke nieuwe versies zullen naar de geest gelijkaardig zijn aan de huidige versie, maar kunnen in details afwijken om nieuwe problemen of bezorgdheden aan te pakken. Zie http://www.gnu.org/copyleft/.

Iedere versie van de Licentie krijgt een onderscheidende versienummer. Indien het Document aangeeft dat een specifiek genummerde versie van deze Licentie “of elke latere versie” van toepassing is, dan heeft u de keuze om de bewoordingen en voorwaarden te volgen van ofwel die specifieke versie of van elke latere versie die verspreid is (niet als draft) door de Free Software Foundation. Indien het Document geen versienummer van deze Licentie aangeeft, dan mag u gelijk welke versie kiezen die ooit verspreid is (niet als draft) door de Free Software Foundation.

Hoe deze licentie te gebruiken voor uw documenten?

Om deze Licentie te gebruiken in een document dat u geschreven heeft, voegt u een kopie van de Licentie aan uw document toe en zet de volgende auteursrechten- en licentie-tekst net na de titel pagina:

```
Copyright (c)  YEAR  YOUR NAME.
```

```
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.1
or any later version published by the Free Software Foundation;
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
A copy of the license is included in the section entitled "GNU
Free Documentation License".
```

Indien u geen Invariante Secties heeft, schrijf dan "zonder Invariante Secties" in plaats van aan te duiden welke secties invariant zijn. Indien u geen Voorpagina-teksten heeft, schrijf dan "geen Voorpagina-teksten"; en op gelijkaardige wijze voor de Achterflap-teksten.

Indien uw document niet-triviale voorbeelden bevat van programma-code, dan raden wij aan om deze voorbeelden in parallel te verspreiden onder uw keuze van vrije software licentie, zoals de GNU General Public License, opdat ze zouden kunnen gebruikt worden in vrije software.

Bibliografie

[Aiv00] Tigran Aivazan. *Linux Kernel Internals*. World Wide Web, http://www.linuxdoc.org, 2000.

[AJ00] Alessandro and Jonathan. *Linux Device Drivers, 2nd edition*. O'Reilly, 2000. 136

[Bar96] Daniel Barlow. *The Linux gcc HOWTO*. LDP, http://www.linuxdoc.org, 1996.

[Bow98] Ivan T. Bowman. Conceptual architecture of the linux kernel. Technical report, University of Waterloo, http://plg.uwaterloo.ca/~itbowman/papers/CS746G-a1.html, 1998.

[BST98] Ivan T. Bowman, Saheem Siddiqi, and Meyer C. Tanuan. Concrete architecture of the linux kernel. Technical report, University of Waterloo, http://plg.uwaterloo.ca/~itbowman/papers/CS746G-a2.html, 1998.

[Das01] Sumitabha Das. *Your UNIX, The Ultimate Guide*. McGrawHill, 2001. 89

[GG] Arne Georg Gleditsch and Per Kristian Gjermshus. The linux kernel hackers' guide. Worl Wide Web. http://www.linuxdoc.org/LDP/khg/HyperNews/get/khg.html.

[gtk] Gimp toolkit (gtk). www.gtk.org. 36

[Hay98] Mark Hays. *POSIX Threads Tutorial*. http://www.math.arizona.edu/swig/pthreads/, 1998.

[Hek97] Jessica Perry Hekman. *Linux in a nutshell, a desktop quick reference*. O'Reilly, 1997. 37

[JB98] Jack Tacket Jr. and Steve Burnett. *Special Edition Using Linux (fourth edition)*. 1998.

[Kei02] Jan Keirse. *Het LinuX/UniX Console Handboek*. World Wide Web, http://users.pandora.be/tux/linuxbook.html, 2002. 24

[LDP] The linux documentation project. World Wide Web. http://www.linuxdoc.org. 136

[Man99] Steve Mansour. A tao of regular expressions. World Wide Web, 1999. 37

[Man01a] MandrakeSoft. *MandrakeLinux Reference Manual*, April 2001. http://www.linux-mandrake.com.

[Man01b] Daniel Manrique. X Window System Architecture Overview HOWTO. Technical report, Linux Documentation Project, mei 2001. 34

[Pom99] Ori Pomerantz. *Linux Kernel Module Programming Guide*. LDP, http://www.linuxdoc.org, 1999. 136

[qt] Q toolkit (qt). www.trolltech.com/producs/qt. 36

[Rob00] Daniel Robbins. Posix threads explained; a simple and nimble tool for memory sharing. 2000.

[Ron00] Michiel Ronsse. *Cursus besturingssystemen, Linux-practica: handleiding*, 2000. 124, 136

[SG94] Abraham Silberschatz and Peter B. Galvin. *Operating System Concepts (fourth edition)*. Addison-Wesley, 1994. 89

[SM99] Richard Stones and Neil Matthew. *Beginning Linux Programming (second edition)*. Wrox, 1999. 136

[Tan] Andy Tanenbaum. Minix. http://www.cs.vu.nl/~ast/minix.html. 11

[Wal97] Sean Walton. *Linux Threads Frequently Asked Questions*. http://www.ibiblio.org/pub/Linux/docs/faqs/Threads-FAQ/html/, 1997.

[War99] Brian Ward. *The Linux Kernel HOWTO*. LDP, http://www.linuxdoc.org, 1999. http://www.linuxdoc.org.

[Whe00] David A. Wheeler. *Program Library HOWTO*. LDP, http://www.linuxdoc.org, 2000. http://www.linuxdoc.org. 81, 82, 83

[WT95] Tom Wagner and Don Towsley. *Getting Started With POSIX Threads*. p://dis.cs.umass.edu/~wagner/threads_html/tutorial.html, 1995.

[xco] X consortium. www.x.org. 35

[xfr] The XFree86 Project. www.xfree86.org. 35

Over dit document

Deze tekst is geschreven in LaTeX, een verzameling macro packages gebouwd rond de typesetting taal TeX[1]. Om dit naar het dvi-formaat om te zetten is er het `latex` programma gebruikt dat het bron bestand als argument aannam. Deze dvi is vervolgens geconverteerd naar het postscript[2] formaat met behulp van `dvips`. Een pdf[3] bestand is uit de bron tekst gegenereerd door `pdfelatex`. Alle tekeningen zijn gemaakt met dia[4]. Eén tabel is gemaakt met Gnumeric[5], gebruik makend van de latex2e output mogelijkheid.

De laatste versie van deze cursus is te vinden op http://research.edm.uhasselt.be/kris/courses/systeemprogrammatuur/. Via CVS[6] kan je ook de LaTeX source files verkrijgen. U kan hier anoniem op inloggen en het paswoord "hiephoi" gebruiken. De volgende commando's zijn hiervoor nodig (aangenomen dat CVS op uw systeem geïnstalleerd is).

```
export CVSROOT=:pserver:anoncvs@lumumba.uhasselt.be:/home/kris/CVSROOT
cvs login
cvs checkout courses/linux/course
```

In een Microsoft Windows omgeving vervangt u `export` door `set`. Alle opmerkingen, kritiek, eventuele bijdragen zijn zeer welkom bij:
`kris.luyten@uhasselt.be`

Veel dank aan degenen die dit al gedaan hebben (zie ook volgend blad).

Laatste update: 5 februari 2006

[1] http://www.tug.org
[2] http://www.adobe.com
[3] http://partners.adobe.com/asn/developer/technotes/acrobatpdf.html
[4] http://www.lysator.liu.se/~alla/dia
[5] http://www.gnome.org/gnumeric
[6] http://www.cvshome.org

Auteurs en bijdragen

Deze tekst kwam tot stand met de bijdragen en hulp van de volgende mensen:

Herman Bruyninckx

- grondige correcties
- suggesties
- aanvullingen

Jori Liesenborgs

- grondige reviews en correcties
- suggesties

Tom Van Laerhoven

- Bibliotheken
- correcties

Bart Moelans

- GNU FDL update in LaTeX
- Emacs
- correcties

Panagiotis Issaris

- Herwerking hoofdstuk device drivers voor 2.6 kernel
- suggesties en correcties

Annika Jansen

- Grondige spellingscontrole

Chris Raymaekers

- originele auteur

- GNU Awk scripting

Stijn Vansummeren

- Hoofdstuk over X

Karin Coninx

- grondige reviews en correcties

Jan Van den Bergh

- aanpassingen
- correcties

Jan Keirse • bijdrage over mounten

Sander Devrieze • Figuur Cover

Fabian Di Fiore

- correcties

Saki Arkoudopoulos

- correcties

Jo Segers

- correcties

Chris Vandervelpen

- correcties

Gioti Arkoudopoulos

- RCS

Tom Haber

- correcties

Simon Vandemoortele

- correcties

Tom Moers

- correcties

Lieven Marchand

- correcties

William Van Haevre

- correcties

Index