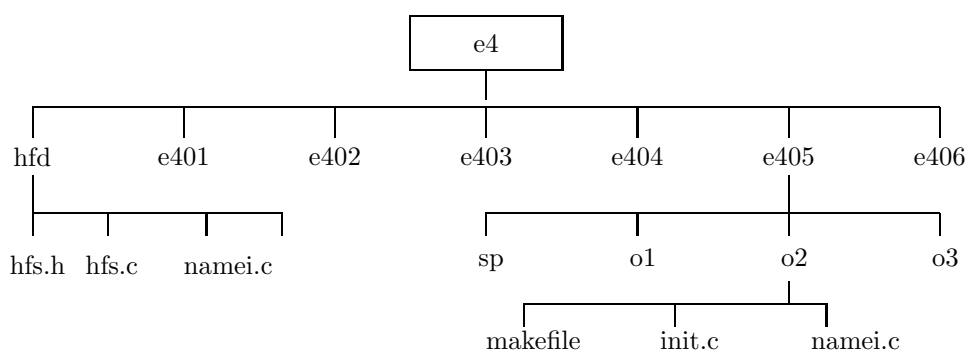


# BESTURINGSSYSTEMEN

Begeleidende tekst bij practica

## 1 Organisatie van de bestanden.

De login-directories van de verschillende groepen studenten bevinden zich in de klas directory **e4**. In deze klas directory is er ook een *hfd* directory aanwezig. In deze directory staan een aantal bestanden die tijdens de oefeningen moeten gebruikt worden.



In de login directory worden een aantal subdirectories gemaakt, één per oefening. In subdirectory **o1** komen de verschillende C-shell scriptbestanden. In subdirectories **o2** en **o3** komen de verschillende sourcebestanden, de **makefile**, de objectbestanden en het uitvoerbaar bestand.

In de **hfd** directory vindt men per oefening een include-bestand en een sourcebestand met het hoofdprogramma. Deze dingen moeten dus niet meer ingetikt worden; ze kunnen naar de eigen directory gecopiëerd worden.

De bestanden *hfs.c* en *hfs.h* horen bij oefening 2.

In directory **o2** worden sourcebestanden gecreëerd voor de verschillende elementen van de opgave.

## Permanente evaluatie

Bij permanente evaluatie wordt rekening gehouden met volgende elementen om tot een score te komen:

1. de bijhorende theorie op voorhand bestudeerd hebben;
2. een aantal opgaven voorbereid hebben;
3. op tijd komen;
4. tijdens de praktijkzitting: niet-storend aanwezig zijn;
5. tijdens de praktijkzitting: actief werken aan de opgaves;
6. bij het stellen van vragen blijk geven van toch al kennis te hebben van de behandelde materie; dus, voordat een vraag gesteld wordt, toch al zelf eens nagedacht hebben;
7. aandachtig luisteren wanneer er klassikaal een bijkomende verduidelijkende uitleg gegeven wordt;
8. valabele elementen aanbrengen tijdens een eventuele klassikale discussie;
9. nadat de basisopgave afgewerkt is, met evenveel enthousiasme aan de eventuele uitbreidingen werken;
10. geen plagiaat plegen;
11. na de praktijkzitting: tijdig het verslag met de eventuele oplossingen afgeven aan de begeleidende docent;
12. indien nodig, komen bijwerken aan bepaalde opgaves indien deze niet voldoende afgewerkt zijn tijdens de praktijkzittingen zelf;
13. het ingeleverde werk wordt door de begeleidende docent beoordeeld en eventueel van commentaar voorzien;
14. bij volgende opgaves rekening houden met eventueel ontvangen commentaar op vorige verslagen;
15. het praktijklokaal reglement respecteren (zie agenda, *algemeen reglement praktijklokalen*).

**Bijkomende richtlijnen** voor de praktijklokalen A215, A216, A217:

- absoluut rook-, eet-, drink- en snoepverbod (herhaling van punt 6 van het algemeen reglement praktijklokalen);
- toestellen en schermen zoveel mogelijk op hun plaats laten staan, dus niet verschuiven of verdraaien;
- netwerk-, muis-, toetsenbord- en voedingskabel laten zitten;
- geen eigen toestellen aansluiten op het netwerk;
- geen *De Nayer*-toestellen meenemen.

## 2 Multi-User operating system: UNIX

### 2.1 Gebruikers en enkele eenvoudige commando's

Op een unix systeem kunnen verschillende gebruikers tegelijk werken. Om te kunnen werken heeft men een loginnaam nodig (publiek gekend) en een geheim paswoord. Met behulp hiervan kan men inloggen. Wanneer dit succesvol gebeurd is, draait er een shell die de commando's zal interpreteren en uitvoeren.

**exit** : om terug af te loggen;

**passwd** : wijzigen van het paswoord: een paswoord moet bestaan uit 3 letters en 3 cijfers; tijdens het intikken verschijnen deze tekens niet op het scherm; ter controle moet daarom het paswoord tweemaal ingegeven worden;

**hostname** : geeft de naam van de computer waarop men ingelogd is;

**whoami** : de eigen loginnaam wordt getoond;

**who** : geeft een lijst van de ingelogde gebruikers op de lokale computer;

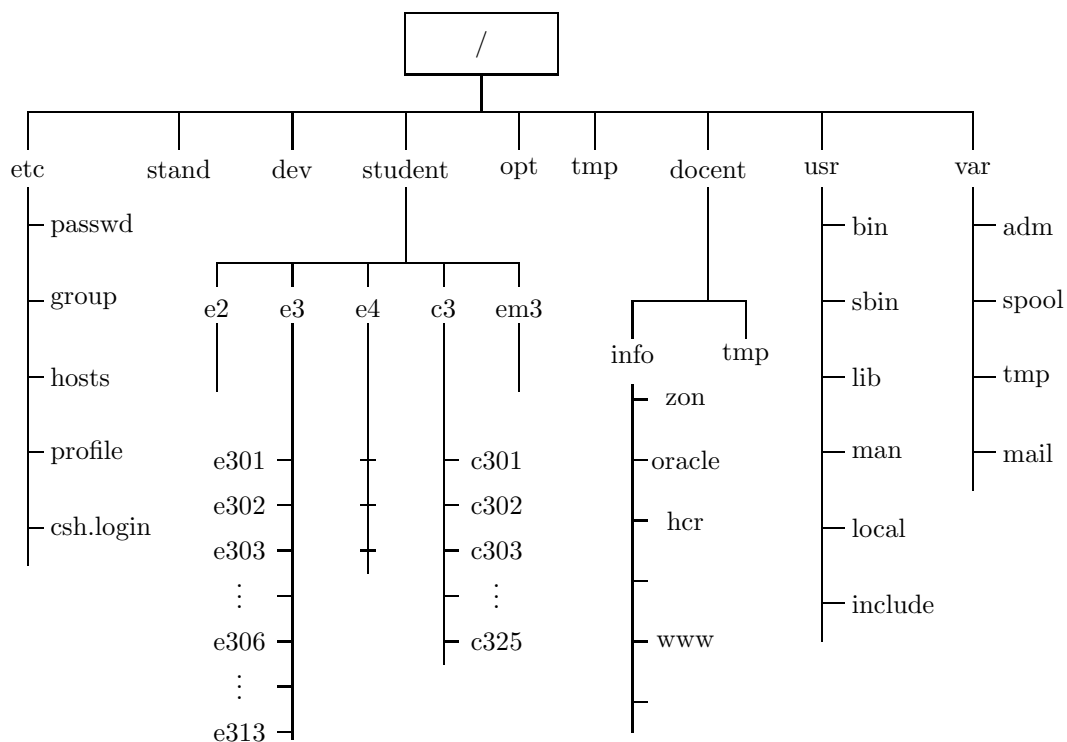
**rwho** : geeft een lijst van de ingelogde gebruikers op de verschillende computers van het netwerk;

**mail** : versturen van boodschappen en lezen van de binnengekomen berichten.

De loginnaam komt overeen met een userid. Deze relatie wordt gelegd in de `/etc/passwd` file. Hierin vindt men ook de default groep waartoe de gebruiker behoort, het geëncrypteerde paswoord, de home directory en de shell die moet opgestart worden bij inloggen. Deze user- en groupid kan met het commando `id` opgevraagd worden.

### 2.2 Bestanden structuur

De verschillende disks vormen samen één logisch bestandensysteem dat georganiseerd is in een boom:



Enkele commando's:

**ll** : toont de inhoud van een directory;

**more** : toont de inhoud van een bestand op het scherm;

**cp** : kopiëren van een bestand naar een ander bestand;

**rm** : verwijdert een bestand uit een directory;

**mv** : geeft een nieuwe naam aan een bestand;

**mkdir** : maakt een nieuwe directory;

**rmdir** : verwijdert een directory, wanneer deze geen files of subdirectories meer bevat;

**pwd** : drukt het pad van de actuele directory af;

**cd** : de actuele directory wijzigt.

Het protectie systeem is gebaseerd op drie klassen van gebruikers:

- eigenaar: gewoonlijk de gebruiker die het bestand creëerde;
- groep: de groep waartoe de eigenaar behoort;
- anderen: al de rest van de gebruikers

Per klasse kunnen drie toegangsprivileges gegeven worden:

<b>r</b>	het bestand mag gelezen worden
<b>w</b>	het bestand mag gewijzigd of verwijderd worden
<b>x</b>	het bestand mag uitgevoerd worden
	de directory mag doorzocht worden

Deze privileges kunnen opgevraagd worden met **ll**:

<b>-rw-r-----</b>	<b>1</b>	<b>hcr</b>	<b>info</b>	<b>102</b>	<b>Oct 3 11:03</b>	<b>jefke</b>
<b>type+protec</b>	<b>links</b>	<b>eig.</b>	<b>grp.</b>	<b>grootte</b>	<b>laatste_upd</b>	<b>naam</b>

De *eigenaar* (**hcr**) mag de file **jefke** lezen en schrijven, de gebruikers van de *groep* **info** kunnen deze file lezen; de *rest* van de gebruikers (o.a. studenten) hebben geen toegang tot deze file.

**chmod** : het wijzigen van de protectiebits van een bestand;

**chown** : het wijzigen van de eigenaar van een bestand;

**chgrp** : het wijzigen van de groep-eigenaar van een bestand.

Commando's om bewerkingen op bestanden te doen:

**diff** : vergelijken van twee bestanden;

**sort** : sorteren van een bestand;

**grep** : doorzoeken van een bestand naar een tekenpatroon;

**find** : doorzoeken van een directorystructuur naar een bestand;

**wc** : tellen van aantal lijnen, woorden, tekens van een bestand;

**rev** : lijn per lijn in omgekeerde volgorde (**echo abcde | rev**);

**pr** : formatteren van een bestand;

**lp** : doorsturen van een bestand naar de printer-spooler;

**lpstat** : opvragen van de status van de spooler.

## 2.3 De vi-editor

Tekstbestanden kunnen gecreëerd en gewijzigd worden met vi. Deze editor heeft twee basis modes:

- commando mode
- tekst invoer mode

Wanneer men vi opstart komt men in commando mode. Overgaan naar tekst invoer mode kan met volgende commando's gebeuren:

i	invoeren van tekst voor de cursor positie
a	invoeren van tekst na de cursor positie
o	invoeren van tekst op de volgende lijn
O	invoeren van tekst op de vorige lijn
R	overschrijven van tekst

Tijdens tekst invoer mode kan alleen met de `backspace` toets naar links bewogen worden. Terugkeren naar commando mode gebeurt m.b.v. de `ESC` toets. In deze mode kan met de pijltjes over de tekst bewogen worden. ook kunnen `NEXT` en `PREV` gebruikt worden om de cursor over 24 lijnen naar onder of naar boven te bewegen. Andere positioneringen:

<i>lijn</i> <code>nrG</code>	naar de lijn met lijnnummer <i>lijn</i>
<code>G</code>	naar het einde van het bestand;
<code>^</code>	naar het begin van een lijn
<code>\$</code>	naar het einde van een lijn
<code>/zoekarg</code>	naar het eerst volgende voorkomen van <i>zoekarg</i>
<code>n</code>	naar het volgende voorkomen van <i>zoekarg</i>
<code>?</code>	naar het vorige voorkomen van <i>zoekarg</i>

Tekst verwijderen kan op verschillende manieren gebeuren:

<code>dd</code>	de lijn waarop de cursor staat
<code>dw</code>	het woord waarop de cursor staat
<code>D</code>	vanaf de cursor tot op het einde van de lijn
<code>x</code>	het teken op de positie waar de cursor staat
<code>r</code>	overschrijven van één teken (geen verwijdering)

Tekst kopiëren:

<code>yy</code>	de lijn waarop de cursor staat wordt in een buffer geplaatst
<code>p</code>	inhoud van de buffer wordt toegevoegd na de cursorlijn
<code>P</code>	inhoud van de buffer wordt toegevoegd voor de cursorlijn
<code>J</code>	samenvoegen van twee opeenvolgende lijnen

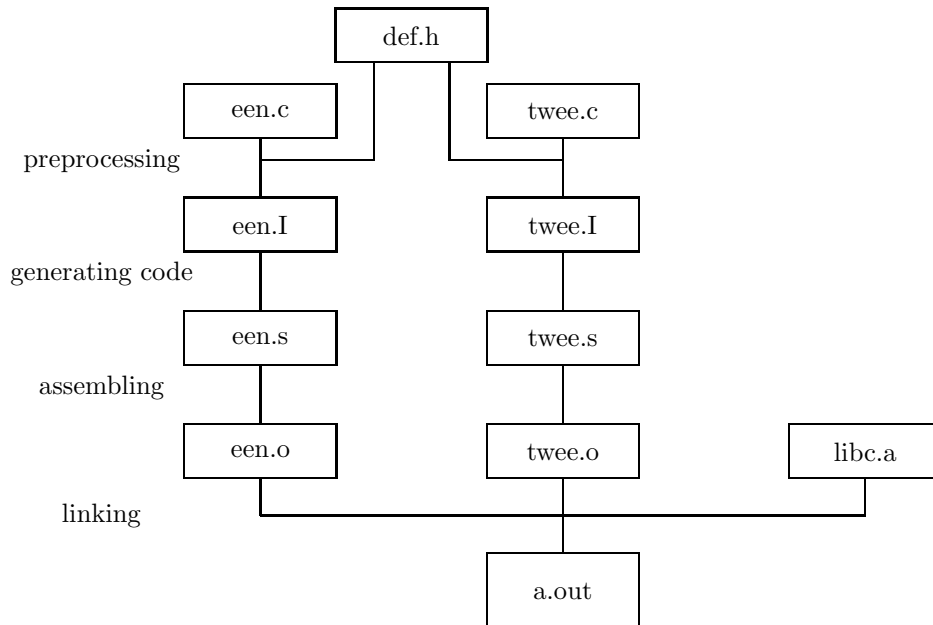
Het `yy` en `dd` kan voorafgegaan worden door een getal. In dat geval wordt de bewerking uitgevoerd op het gespecificeerde aantal lijnen vanaf de lijn waarop de cursor staat.

Algemene commando's:

<code>ZZ</code>	bewaar de veranderingen en verlaat de editor
<code>:wq</code>	bewaar de veranderingen en verlaat de editor
<code>:q!</code>	verlaat de editor zonder de veranderingen te bewaren
<code>.</code>	herhaal vorige actie
<code>u</code>	maak vorige actie ongedaan
<code>U</code>	herstel de lijn in haar vorige toestand

## 2.4 Programma ontwikkeling

De source van het programma moet eerst ingetikt worden m.b.v. bijvoorbeeld de vi-editor. Daarna moet het gecompileerd en gelinkt worden. Op de meeste Unix systemen zijn verschillende compilers beschikbaar: bijvoorbeeld *pc*: pascal, *cc*: c en *fc*: fortran. Een compiler werkt meestal in verschillende passen:



Vanuit de bronbestanden `een.c` en `twee.c` en een header file `def.h` wordt een executable gemaakt welke in het bestand `a.out` terecht komt.

Wanneer het programma niet foutloos werkt, kan beroep gedaan worden op de **GNU** debugger. Hiervoor moet het programma wel met de **-g** optie gecompileerd en gelinkt zijn. De debugger starten gebeurt met het bevel :

```
gdb [laadnaam] (te vinden in /opt/langtools/bin)
```

De debugger geeft aan de gebruiker aan dat zij klaar is om iets te doen met een prompt teken, wat in dit geval (**gdb**) is.

Een beperkte lijst van commands:

**q** (van quit) debugger beëindigen;

**l** display van de volgende 10 lijnen

**l -** display van de vorige 10 lijnen

**l van-lijnno,tot-lijnno** display van enkele lijnen uit het bronprogramma,  
met *lijnnummer* wordt de nummer van een lijn bedoeld in het bronprogramma;

**b lijnnummer** (van break) een breekpunt plaatsen op een lijn;

**d breekpuntnummer** (van delete) een breekpunt verwijderen;  
de breekpuntnummers kan u vinden met behulp van **info break**;

**r** (van run) het programma starten;

**c** (van continue) het programma laten verder werken tot het volgend breekpunt;

- s** (van single step) het programma lijn voor lijn uitvoeren, eventueel kan dmv. een getal het aantal lijnen opgegeven worden dat moet uitgevoerd worden;
- n** *aantal* (van next) het programma een *aantal* lijnen laten verder werken (miv. functie-oproepen), nodig om niet in functies van de standaard C-library te sukkelen;
- p** [/*formaat*] naam van variabele : opvragen van de waarde van een variabele  
*formaat* kan een van volgende letters zijn: d(ecimaal), f(loating point), x(hexadecimaal), c(haracter), a(dres);
- x** [/*Nuf*] expressie (naam van een variabele) : inhoud van opeenvolgende geheugenplaatsen  
*N* is het aantal opeenvolgende eenheden; *u* geeft de eenheid: b(ytes), h(alfword), w(ord), g(iant) en *f* is zoals hierboven;
- bt** trace van alle frames van de stack (m.i.v. argumenten);
- l bronbestandsnaam:lijnummer** veranderen van bronbestand:  
 dit bevel heb je alleen nodig indien het laadprogramma samengesteld is uit meerdere bronbestanden.

Wanneer een programma uit verschillende bronbestanden bestaat, kan de utility *make* gebruikt worden. Make doet beroep op een *makefile* waarin aangegeven wordt hoe een executable tot stand kan komen:

```
#makefile voor bovenstaand voorbeeld
all: a.out
CFLAGS=-g
LFLAGS=-g

een.o: een.c def.h
        cc -c $(CFLAGS) een.c

twee.o: twee.c def.h
        cc -c $(CFLAGS) twee.c

a.out: een.o twee.o
        cc -o a.out $(LFLAGS) een.o twee.o
```

Het doelbestand a.out wordt gemaakt op basis van twee object bestanden die zelf tot stand komen door een aantal bronbestanden te compileren.

De eerste lijn is een commentaarlijn (aangeduid door het *#* teken). De tweede lijn geeft het vooropgestelde (*default* voor de kenners) doel weer, wanneer geen argument weergegeven wordt bij het oproepen van make. Dus de bevelen *make* en *make all* hebben hetzelfde effect. Op de twee volgende lijnen worden de variabelen CFLAGS en LFLAGS gedefiniëerd. Deze worden in de volgende bevelen gebruikt om aan te geven hoe de compilatie en linking moet gebeuren.

Daarna volgen de afhankelijkheden en de bevelen nodig om van het bronbestand of -bestanden het doelbestand te maken:

```
<doel> : afhankelijk van {<bron>}
TAB-teken UNIX-bevel
[ {TAB-teken UNIX-bevel} ]
```

Deze UNIX-bevelen worden enkel uitgevoerd wanneer één van de < *bron* >bestanden jonger (recentere modificatietijd) is dan het < *doel* >bestand, of wanneer het < *doel* >bestand nog niet bestaat.

## 2.5 Het proces systeem

Enkele commando's:

**ps:** produceert een lijst van processen (opties: **-u** *loginnaam*, **-e**, **-f**, **-l**);

PID	TTY	TIME	COMMAND
13849	pts/2	0:00	ssh
14153	pts/2	0:00	ps

**nice:** verandert de prioriteit van een proces (bijv. **nice +4 ps -l**);

**time:** chronometreert de uitvoering van een proces (bijv. **time ps -ef**);

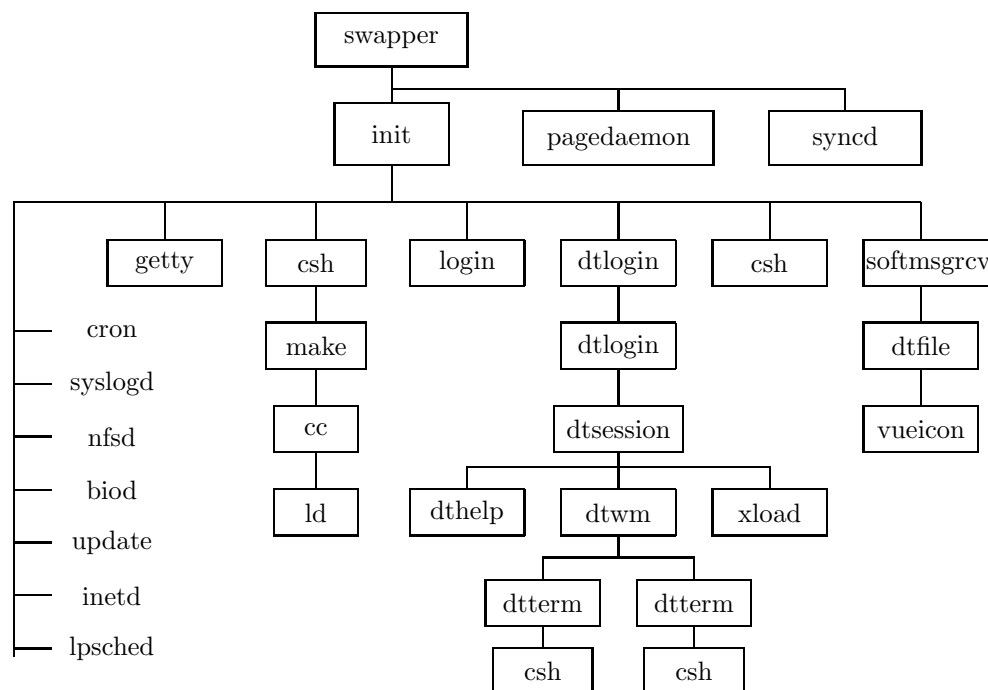
**kill:** stuurt een signaal naar een proces, meestal met de bedoeling het proces te laten stoppen (lijst met mogelijke signalen: **kill -l**) (processen via naam: **pkill a.out**);

**CTRL C**: breekt een proces af (op sommige toetsenborden **BREAK**);

**CTRL Z**: onderbreekt een proces;

**fg/bg:** een proces wordt voortgezet in de voorgrond/de achtergrond.

Het proces systeem is georganiseerd als een boom. Op proces 0 na, wordt elk proces gecreëerd door een ouderproces:



Enkelvoudige commando's en primitieven van de shell vormen bouwstenen om complexere commando's te bouwen:

&	het proces wordt in de achtergrond gestart
	pipe: de output van het eerste proces is de input van het tweede proces
<	haalt de gegevens (standaard input) uit het aangegeven bestand
>	stuurt de resultaten (standaard output) naar het aangegeven bestand (eventueel wordt dit eerst gecreëerd)
>>	voegt de resultaten achteraan aan een bestaand bestand bij
tee	stuurt de input zowel naar standaard output als naar het aangegeven bestand

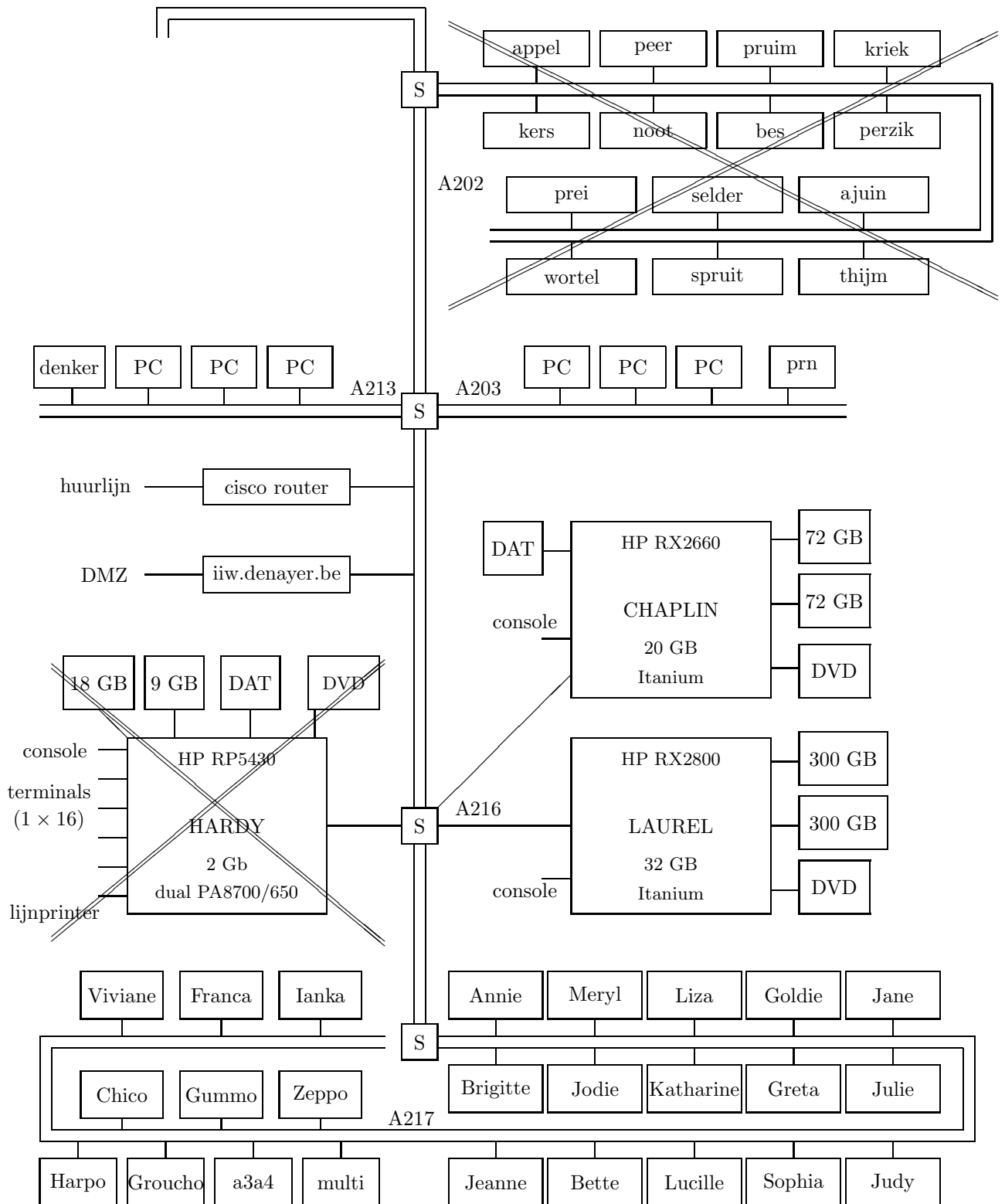


## 2.6 Het netwerk systeem

Naar de gebruiker toe wordt het netwerk vooral gebruikt voor het transport van bestanden tussen DOS en UNIX via ftp. Dit gebeurt met de volgende procedure.

Op PC:

1. inbrengen diskette/USB-stick in a: of x: drive
2. opstarten: *ftp chaplin* (naam van een UNIX machine, dus abott is ook mogelijk)
3. hier moet ingelogd worden op chaplin: loginnaam: .....  
paswoord: .....
4. de prompt is ftp>
5. *drive a:* (3.5 inch) OF *drive x:* (USB-stick)
6. lcd \dos-directory (naar keuze)
7. cd unix-directory
8. van DOS->UNIX: *mput \*.c* (bijvoorbeeld)
9. van UNIX->DOS: *mget \*.c* (bijvoorbeeld)
10. andere nuttige bevelen:  
  
    **help** : lijst van commands  
    **mkdir** : dos-directory (op de diskette dus)  
    **mkdir** : unix-directory ( op chaplin)  
    **lpwd** : print directory van dos  
    **pwd** : print directory van unix  
    **ldir** : inhoud lokale directory van dos  
    **ls** : inhoud remote directory van unix  
    **dir** : lange inhoud remote directory van unix
11. transfer gebeurt in een bepaalde mode, is op te vragen met status  
  
    **bin** : binair (geen enkele conversie van tekens)  
    **ascii** : vertaling:  
        UNIX->DOS (get) : \n wordt \n\r  
        DOS->UNIX (put) : \n\r wordt \n
12. drive c:
13. stoppen met *quit* of *bye*.



Figuur 2.1: Configuratieschema

### 3 Oefening 1: C-shell

Bij het opstarten van de C-shell worden drie bestanden uitgevoerd:

**/etc/csh.login** : globale definities, door de SA ingesteld, alleen bij de login shell;

**.cshrc** in de home directory: eigen definities, telkens een shell gestart wordt;

**.login** in de home directory: eigen definities, alleen bij de login shell.

Ter illustratie, maak een eigen *.cshrc* bestand in de home directory met volgende inhoud:

```
set prompt="$$.\!% "
```

Door een sub-shell op te starten (*csh*), wordt de prompt een getal bestaande uit twee delen: het eerste deel de proces-id van het shell proces en het tweede deel de volgnummer van het bevel in deze sub-shell. In deze sub-shell kan nog een shell gestart worden: *.cshrc* wordt opnieuw uitgevoerd, we krijgen een nieuw eerste deel in de prompt te zien.

Het beëindigen van een shell-sessie kan met *exit*, of **CTRL-D**. Men keert dan terug naar de ouder-shell. Het beëindigen van de login shell-sessie kan met *exit*, *logout* of **CTRL-D**. In dit geval wordt het bestand *.logout* in de home-directory uitgevoerd.

#### Voorbereiding.

```
% mkdir o1
% cp /student/e4/hfd/bs/* o1
% cd o1
% chmod 0750 *.csh
```

#### 3.1 Variabelen

Er zijn twee soorten:

**omgevingsvariabelen** : deze zijn globaal in het shell proces en worden doorgegeven naar alle kindprocessen. Voor deze namen gebruikt men hoofdletters. Initialisatie met het *setenv* bevel. Het bevel *env* geeft een lijst van alle omgevingsvariabelen.

**shell variabelen** : deze zijn lokaal in het shell proces (ze worden niet doorgegeven naar kindprocessen). Voor deze namen gebruikt men kleine letters. Initialisatie met het *set* bevel of met *@* (spaties zijn nodig); variabele expansie met *\$*.

```
% set term=hp
% echo term
term
% echo $term
hp
% @ i = 4
% @ j = 4 * $i
% echo $j
16
```

Het bevel *set* geeft een lijst van alle shell variabelen.

Merk op dat de omgevingsvariabelen *USER*, *TERM* en *PATH* steeds dezelfde inhoud hebben als de shell variabelen *user*, *term* en *path*. (De syntax voor *path* is echter verschillend).

Een variabele verwijderen kan met *unset* of *unsetenv*.

```
% set lok = appel
% setenv IKKE jos
% echo $IKKE $lok
jos appel
% dtterm &
% echo $IKKE
% echo $lok
% unsetenv IKKE
% echo $IKKE
IKKE: Undefined variable.
```

In de csh in het nieuwe window is wel IKKE maar niet lok gekend (probeer `% echo $lok $IKKE`).

### 3.1.1 Meer over shell variabelen

`$varnaam`, `${varnaam}` worden tijdens interpretatie geëxpandeerd naar de inhoud van de variabele. De “{” symbolen zijn optioneel en dienen enkel om de naam te scheiden van andere woorden. Soms kan een variabele uit een lijst van woorden of waarden bestaan. Eén element hieruit selecteren kan door middel van `${varnaam[selector]}`. `${#varnaam}` geeft dan het aantal woorden in de lijst weer.

`$$` geeft de proces-id van de ouder-shell. Met `$<` kan een lijn van standaard input gelezen worden. Met `${?varnaam}` kan getest worden of een variabele gezet is of niet.

```
% set x=z
% echo $xy
xy: Undefined variable
% echo ${x}y
zy
% set a=(appel peer banaan)
% echo $a
appel peer banaan
% echo $a[2]
peer
% echo $#a
3
% echo $$
20345
% echo -n "Tik iets in : " ; set i="$<"
Tik iets in : abc def ghi
% echo $i
abc def ghi
% echo $?i
1
% unset i
% echo $?i
0
```

Ingebouwde shell variabelen:

**autologout** : automatische logout na een inactiviteit gedurende de gespecificeerde tijd

**cwd** : de current working directory

**home** : de home directory (zie `/etc/passwd`, één lijn hieruit: `grep e406 /etc/passwd`)

**path** : lijst van directories waarin uitvoerbare programma's gezocht worden

**prompt** : keuze van de prompt-string

**history** : aantal bevelen die bijgehouden worden in de history buffer

**savehist** : aantal bevelen die bijgehouden worden na een logout voor een volgende shell sessie in het .history bestand in de home directory

**status** : exit-status van het laatste bevel; een abnormale beëindiging van een built-in bevel heeft exit-status **1**, een normale beëindiging waarde **0**.

**argv** : de argumenten van de bevelijn: \$argv[1] is het eerste argument, \$argv[2] het tweede, ...; deze kunnen afgekort worden tot **\$1**, **\$2**, ... \$argv kan vervangen worden door **\$\***. **\$0** bevat de naam van het bevel.

```
% lst.csh flup marie bever
% echo $status
% lst.csh 6 abc
% echo $status

#!/usr/bin/csh
echo $*
echo $0
echo $1
echo $argv[$#argv]
if ( $1 =~ [0-9] ) then
    echo "Stoppen met status " $1
    exit $1
endif
```

## 3.2 Meta-tekenen

De eerste reeks heeft met de syntax te maken:

;  
( )  
<, >  
|  
&  
||  
&&

: scheiding van bevelen die in sequentie moeten uitgevoerd worden  
: groeperen van een reeks bevelen, en uitvoeren in een sub-shell  
: herdirectie van standaard input en standaard output  
: pijplijn  
: bevel wordt uitgevoerd in een background proces  
: tweede bevel wordt alleen uitgevoerd als het eerste mislukt  
: tweede bevel wordt alleen uitgevoerd als het eerste lukt

```
ls ; pwd
(cd .. ; ls ) ; pwd
ls > flup ; wc < flup
ls | wc ; who | cut -d' ' -f1 | sort | uniq
who | uniq &
lst.csh 0 1 || pwd ; lst.csh 1 2 || pwd
lst.csh 0 1 && pwd ; lst.csh 1 2 && pwd
```

De tweede reeks zorgt voor het expanderen van bestandsnamen

\*  
?  
[ ]  
~

: een reeks van tekens, inclusief de null string  
: één teken  
: één van de tekens tussen de haakjes of wanneer een range gespecificeerd is (mbv '-' ) één van de tekens uit de range  
: de padnaam van de home directory van de specificeerde user

```
ls f*
ls ??.csh
ls [a-i]*
ls ~e406 ; ls ~
```

Wanneer een meta-teken zijn originele betekenis moet behouden, moeten quote-tekenen gebruikt worden:

\  
'

: het volgende teken behoudt zijn originele betekenis  
: de tekens in de string tussen twee single quotes behouden hun originele betekenis

” : zoals single quotes, maar bevel en variabele expansie gebeurt wel.

Bevel expansie gebeurt met backquotes(`): het resultaat van het bevel (een aantal woorden) kan toegekend worden aan een variabele.

```
% set a = 3
% echo $a \ $a ' $a' " $a"
% set i='pwd'
% set j='`pwd`'
% set k="`pwd`"
% echo $i $j $k
/student/e4/e406/csh `pwd` /student/e4/e406/csh
```

### 3.3 Expressies

De volgende operatoren zijn mogelijk op numerieke variabelen:

|| && | ^ & <= >= < > << >> + - \* / % ! ()

Testen op al of niet gelijkheid van strings gebeurt met == en !=. Bij =~ en !~ is de rechterzijde een patroon (met \*'s, ?'s en [...]’s) waarop de linkerzijde vergeleken wordt.

In een expressie kan ook een status van een file opgevraagd worden: -l *filenaam* waarbij de letter l één van de volgende kan zijn:

d	is de filenaam een directory ?	r	heb ik lees toegang tot deze file ?
e	bestaat de filenaam ?	w	heb ik schrijf toegang tot deze file ?
f	is de filenaam een gewone file ?	x	kan ik deze file uitvoeren ?
o	ben ik eigenaar van de file ?	z	bevat deze file 0 bytes ?

### 3.4 Controle structuren

#### 3.4.1 foreach-end

```
% for.csh
#!/usr/bin/csh
set a = ( aap beer das geit )
foreach i ( $a)
    echo -n $i " : "
    echo $i | rev
end
```

#### 3.4.2 if-endif

```
% if.csh
% if.csh .. flup
#!/usr/bin/csh
if ( $#argv == 0 ) then
    echo "Graag argumenten"
    exit 1
endif
echo $*
foreach i ( $* )
    if ( -d $i ) then
        echo "$i is een directory"
    else if ( -r $i ) then
        echo "$i is een file"
    else
        echo "$i is nog iets anders"
    endif
end
```

### 3.4.3 while-end

% while.csh

```
#!/usr/bin/csh
@ i = 1
while ( $i < 20 )
    @ j = $i % 2
    if ( $j == 0 ) then
        echo $i
    endif
    @ i++
end
```

### 3.4.4 goto

% goto.csh

Met *shift* worden de elementen in **a** één positie naar links verschoven.

```
#!/usr/bin/csh
set a = ( 1 2 3 4 5 )
nog:
    echo $a
    shift a
    if ( $#a > 0 ) then
        goto nog
    endif
echo $?a
```

### 3.4.5 switch-endsw

% sw.csh

```
#!/usr/bin/csh
set antw=ongeldig
while ( $antw == "ongeldig" )
    echo "1: aanpassing inode"
    echo "2: aanpassing file"
    echo "3: toegang file"
    echo -n "Keuze : "
    set keuze=$<
    if ( ($keuze<1) || ($keuze>3) ) continue
    set antw = geldig
end
switch ( $keuze )
case 1 :
    set a=c
    breaksw
case 2 :
    set a=l
    breaksw
case 3 :
    set a=u
    breaksw
endsw
ls -l$a
```

### 3.4.6 onintr

```
% onintr.csh

Druk of CTRL C en be-
kijk de variabele status
met echo $status.

#!/usr/bin/csh
set inter=nee
onintr opvang
@ i = 1
start:
    while ( $inter == nee )
        echo -n "."
        @ i++
    end
    @ j = $i % 256
    echo $i, $j
    exit( $i )
opvang:
    echo
    echo "het wordt tijd"
    set inter=ja
    goto start
```

## 3.5 Projectie en selectie

Voor sommige opgaven is het *cut* bevel nodig:

```
% cut -d: -f1,3 /etc/passwd
% cut -c1-9 /etc/passwd
% cat /etc/passwd | cut -d: -f1,5
```

In het eerste bevel wordt het scheidingsteken (:) gedefinieerd; op basis daarvan wordt het eerste en het derde veld (f1,3) van elke lijn van de paswoord file geprojecteerd.

In het tweede bevel worden de eerste 9 tekens van elke lijn van de paswoord file geprojecteerd.

Een ander nuttig bevel is *grep*:

```
% grep '^e4' /etc/passwd
% grep 'se$' /etc/passwd
% grep -l goto *.csh
```

Met het eerste/tweede bevel worden alle lijnen die respectievelijk beginnen/eindigen met de letters “e4”/“se”, uit de paswoord file geselecteerd.

Het derde bevel geeft een lijst van bestandsnamen waarin het woord “goto” voorkomt.

Voorbeeld:

```
% prosel.csh

#!/usr/bin/csh
set f=/student/e4/hfd/*.csh
set d='grep -l goto $f'
@ j = 2
foreach i ( $d )
    echo $i:h $i:r $i:e $i:t
    set e='echo $i | cut -d/ -f$j'
    echo $e
    @ j++
end
```

De resulterende padnaam wordt opgedeeld in een aantal delen:

<b>h</b>	(head) het hoofd zonder laatste component
<b>r</b>	(root) de padnaam zonder suffix
<b>e</b>	het suffix (csh)
<b>t</b>	(tail) de laatste component, d.i. de bestandsnaam



Ook wordt van elke padnaam de eerste component (**student**) gegeven.

### 3.6 Vervangen en weglaten van tekens

Voor het vervangen van alle voorkomens van een teken in een bestand door een ander teken, is er **tr** (translate).

```
% e2a.csh
#!/usr/bin/csh
set i = 'grep hcr /etc/group | cut -d: -f1'
set j = 'echo $i | tr e a'
echo $i "->" $j
```

Een lijn uit `/etc/passwd`:

```
spin:avLafUiA9z4gU:1343:99:Spinmaster:/apache/home:/usr/bin/csh
```

Zo'n lijn bevat zeven velden, met het dubbelpunt als scheidingsteken:

```
loginnaam : paswoord : user-id : group-id : naam : homedir : login-shell
```

Een lijn uit `/etc/group`:

```
spin::99:spin,hcr,bco
```

Zo'n lijn bevat vier velden, met het dubbelpunt als scheidingsteken. Het vierde veld bevat de groepsleden, een lijst van loginnamen gescheiden door komma's.

```
groepnaam : paswoord : group-id : groepsleden
```

Het verwijderen van alle voorkomens van een teken in een bestand kan met behulp van de **-d** optie.

```
% wegcr dosfile
#!/usr/bin/csh
echo $$
tr -d '\015' < $1 > /tmp/fghj$$
mv /tmp/fghj$$ $1
```

### 3.7 Opgaven

1. Een C-shell programma met een bestandsnaam als argument. Indien het argument overeenkomt met een tekstbestand, wordt de inhoud van het bestand op het scherm getoond. Indien het argument overeenkomt met een directory, wordt een lijst met de in deze directory aanwezige bestanden eindigend op ".csh", aan de gebruiker getoond samen met een volgnummer. De gebruiker kan hieruit met behulp van het volgnummer een bestand kiezen. De inhoud van het gekozen bestand wordt op het scherm getoond.
2. In de `/student/e4/hfd/tmp/oef2` directory zijn een aantal subdirectories aanwezig. In elk van deze subdirectories zitten een aantal bestanden waarvan de naam bestaat uit 3 tekens, een volgnummer bestaande uit 2 cijfers en getal bestaande uit drie cijfers. Bijvoorbeeld, de bestanden `abc01200`, `abc04200`, `abc12200`, `abc83200`. De eerste drie tekens van een bestandsnaam zijn gelijk aan de naam van de subdirectory.

Schrijf een C-shell programma met 1 parameter (een getal van drie cijfers), dat deze subdirectories met de bestanden copieert naar een directory **oef2** in uw working directory, waarbij bij elke bestandsnaam in elke subdirectory de laatste drie cijfers vervangen worden door de opgegeven parameter.

Bij een oproep met parameter 750 moet dus een directory `oef2` in uw working directory gecreëerd worden met de verschillende subdirectories en daarin bijvoorbeeld bestanden met naam `abc01750`, `abc04750`, `abc12750`, `abc83750`.

3. Uitbreiding: in de subdirectories van de `/student/e4/hfd/tmp/oef3` directory zitten bestanden met verschillende eindgetallen. Het C-shell programma met twee argumenten copieert nu alleen die bestanden waarvan het eindgetal (laatste drie cijfers) gelijk is aan de tweede parameter. Dit eindgetal moet ook weer vervangen worden door de eerste parameter. Indien een bestand door u niet leesbaar is, moet het ook niet gecopieerd worden.

Extra (Jeroen): indien er niets in de subdirectory gecopieerd wordt, mag de subdirectory zelf ook niet in het resultaat opgenomen worden.

4. Een C-shell programma dat in een directory informatie van bestanden toont die in een bepaalde maand aangepast geweest zijn. Het programma heeft één of twee argumenten: het eerste argument is de naam van een maand (eerste drie letters); het eventuele tweede argument is de naam van een directory:

```
zoek.csh maand [directory]
```

Indien geen tweede argument gegeven is, dan is de directory gelijk aan de huidige directory. Het programma gaat na (op basis van de output van `ll`) of er in de directory bestanden aanwezig zijn die in de gegeven maand aangepast zijn:

- indien een gewoon bestand: toon de eerste lijnen van dit bestand (more)
  - indien directory: doorzoek de directory of er in de directory zo'n bestanden zijn en toon van die bestanden de eerste lijnen.
  - indien een ander type bestand : niets doen
5. Een C-shell programma om te zoeken in de `passwd` file naar gebruikers waarvan de user-id in een gegeven range ligt, display ook de bijhorende loginnaam. De range wordt ingegeven via argumenten bij het commando, bijvoorbeeld `oef5.csh 1300 1310`.  
Uitbreiding: bepaal of de corresponderende homedirectory (`~`) door u doorzoekbaar is (x). Merk op dat sommige uid's overeenkomen met verschillende loginnamen. In dat geval moet alleen de homedirectory van de eerste loginnaam op doorzoeken getest worden.
  6. Een C-shell programma om te zoeken in de group file naar groepen waarvan de group-id in een gegeven range ligt, display ook de bijhorende groupnaam en de het aantal leden. De range wordt ingegeven via argumenten bij het commando, bijvoorbeeld `oef6.csh 25 30`.  
Uitbreiding: bepaal of de homedirectory (`~`) van elk groepslid door u doorzoekbaar is (x).
  7. Een C-shell programma om een signaal te sturen naar een aantal processen die aan een bepaalde voorwaarde voldoen, bijvoorbeeld waarvan de naam van het in uitvoering zijnde programma gelijk is aan "smbd".

## 4 Oefening 2: Implementatie van een beperkt filesystem.

**Doel:** Het bouwen van een eenvoudig filesystem, bestaande uit een superblok, een reeks inodes en een reeks data blokken, waarmee een aantal systeem oproepen kunnen gesimuleerd worden. De grootte van een blok is 32 bytes. Het geheel wordt in een UNIX file gestockeerd.

### Structuur:

1. Superblok (blok 0):  
totaal aantal inodes en data blokken : 2 shorts (4 bytes) gevolgd door twee bitmaps:  
    één voor de vrije inodes: ( grootte:12 bytes) 96 bits  
    één voor de vrije blokken: (grootte:16 bytes) 128 bits  
(wanneer in de bitmap de N-e bit de waarde 1 heeft, is de N-e inode resp. blok vrij).
2. Inode (twee per data blok):  
uid en link: 2 chars; mode en size: 2 shorts; samen 6 bytes  
(mode bevat het type en de protectiebits, zie stat system call);  
4 directe verwijzingen naar data blokken: 8 bytes;  
1 indirecte verwijzing (2 bytes) naar blok met max 16 blokadressen.
3. Data blok: 32 bytes.
4. Zo'n datablok bevat eventueel een deel van een Directory:  
(vier entries van 8 bytes per datablok) een entry bevat de inode nummer (1 byte) en de naam (7 bytes).

### Akties:

- I** : Initialisatie <aantal\_inodes> <aantal\_diskblokken> : creëert een filesystem waarbij superblok, inode 1 (rootinode), en de rootdirectory (“.” en “..”) geïnitieerd zijn;
- S** : Status <naam> : het afdrukken van de status van een bestand, d.i. inhoud van de inode
- F** : File <naam> <mode> <grootte> <opvul\_char> : creatie van een file, die opgevuld wordt met het opvul karakter, zodat de size < grootte > is (max 128+512 bytes)
- M** : Mkdir <naam> : het maken van een directory entry, met mode 0777
- U** : Unlink <naam> : het verwijderen van een naam (eventueel is dit een *lege* directory en moeten de “.” en “..” entries ook verwijderd worden)
- L** : Link <bestaande\_linknaam> <nieuwe\_linknaam> : het creëren van een extra link
- Z** : Zien : een compacte dump van het filesystem
- P** : Proces : een compacte dump van de user-structuur van het proces

**Uitbreidingen:** De <naam> die bij bovenstaande acties moet meegegeven worden, is steeds een volledige padnaam, d.w.z. wordt geïnterpreteerd vanaf de root van het filesystem. Door in het simulatieproces een current en een rootdirectory te definiëren, kan de *chdir* en *chroot* system call gesimuleerd worden. Wanneer aan het proces ook een eigen <uid> toegekend wordt, kan het protectiemechanisme getest worden:

- A** : Umask <nieuwe\_umask> : veranderen van de u\_mask van het proces
- E** : Eigenaar <nieuwe\_uid> : veranderen van de current eigenaar van het proces
- C** : Chdir <naam> : veranderen van de current directory van het proces
- R** : Rdir <naam> : veranderen van de root directory van het proces
- N** : Mode <nieuwe\_mode> <naam> : veranderen van de mode van een bestand
- D** : Duid <nieuwe\_uid> <naam> : veranderen van de uid van een bestand

**Testprogramma.** Om deze routines te testen, is een hoofdprogramma geschreven. Ook is de routine *zien* reeds geïmplementeerd. De sources zijn in de **hfd** directory terug te vinden: *hfs.h* en *hfs.c*.

**Demo+verslag:** eerste demo (o.a. init) in 5e labozitting.  
 Lesweek 9, tijdens de labozitting: listing van het programma + verklarende tekst

**Voorbeelden van de compacte dump.** i 4 6

```

4 6 70000000000000000000000000000000 7c000000000000000000000000000000
1 : 0 2 41ff 16 : 1 0 0 0 0
2 : 0 0 0 0 : 0 0 0 0 0
3 : 0 0 0 0 : 0 0 0 0 0
4 : 0 0 0 0 : 0 0 0 0 0
1 : 1 . 0 0 0 0 0 0 1 . . 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 : 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
3 : 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4 : 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
5 : 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
6 : 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

f /jef 666 45 x

m /tmp

```

4 6 10000000000000000000000000000000 0c000000000000000000000000000000
1 : 0 3 41ff 32 : 1 0 0 0 0
2 : 13 1 81b6 45 : 2 3 0 0 0
3 : 11 2 41ff 16 : 4 0 0 0 0
4 : 0 0 0 0 : 0 0 0 0 0
1 : 1 . 0 0 0 0 0 0 1 . . 0 0 0 0 0 2 j e f 0 0 0 0 3 t m p 0 0 0 0
2 : x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x
3 : x x x x x x x x x x x x x 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4 : 3 . 0 0 0 0 0 0 1 . . 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
5 : 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
6 : 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

f /tmp/louis 444 40 a

u /jef

```

4 6 40000000000000000000000000000000 60000000000000000000000000000000
1 : 0 3 41ff 32 : 1 0 0 0 0
2 : 0 0 0 0 : 0 0 0 0 0
3 : 11 2 41ff 24 : 4 0 0 0 0
4 : 11 1 8124 40 : 5 6 0 0 0
1 : 1 . 0 0 0 0 0 0 1 . . 0 0 0 0 0 0 j e f 0 0 0 0 3 t m p 0 0 0 0
2 : x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x
3 : x x x x x x x x x x x x x 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4 : 3 . 0 0 0 0 0 0 1 . . 0 0 0 0 0 4 l o u i s 0 0 0 0 0 0 0 0 0 0
5 : a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a
6 : a a a a a a a a 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

```

1  /*
   * hfs.h : globale definities voor een filesysteem simulatie programma
3  *
   */
5
   #define NAAMFS "klein"
7  #define SYSLEN 80
   #define MAXINODE 12
9  #define MAXBLOK 16
   #define BLOKSIZE 32
11 #define INOSIZE 16
   #define BLKADDR 5
13 #define BLKINDIR (BLKADDR-1)
   #define NAMELEN 7
15 #define DIRLEN (NAMELEN+1)

17 /* foutcodes */
   #define GEENACC 801
19 #define GEENDIR 802
   #define REEDS 803
21 #define GEENBLO 804
   #define GEENINO 805
23 #define NOGNIET 806
   #define NIETLEEG 807
25

   /* toegangsrechten */
27 #define RRR 4
   #define WWW 2
29 #define XXX 1

31 #define SetIalloc(n) sbk.s_fi[((n)-1)/8] &= ~(1<<(7-(((n)-1)%8)))
   #define SetIfree(n) sbk.s_fi[((n)-1)/8] |= (1<<(7-(((n)-1)%8)))
33 #define IsIfree(n) (sbk.s_fi[((n)-1)/8] & (1<<(7-(((n)-1)%8))))
   #define SetBalloc(n) sbk.s_fb[((n)-1)/8] &= ~(1<<(7-(((n)-1)%8)))
35 #define SetBfree(n) sbk.s_fb[((n)-1)/8] |= (1<<(7-(((n)-1)%8)))
   #define IsBfree(n) (sbk.s_fb[((n)-1)/8] & (1<<(7-(((n)-1)%8))))
37 #define GetUid(n) (((n)>>4) & 0xF)
   #define GetGid(n) ((n) & 0xF)
39

   typedef struct superblok
41 {
       short s_inode;
43       short s_blok;
       unsigned char s_fi[MAXINODE];
45       unsigned char s_fb[MAXBLOK];
   } Superblok ;
47

   typedef struct inode
49 {
       unsigned char i_uid; /* 4bits userid; 4 bits groupid */
51       unsigned char i_link;
       unsigned short i_mode;
53       unsigned short i_size;
       unsigned short i_blok[BLKADDR];

```

```

55 } Inode ;

57 typedef struct dir
58 {
59     char    d_ino;
60     char    d_naam[NAMELEN];
61 } Dir ;

63
64 typedef struct user
65 {
66     short    u_uid;
67     short    u_gid;
68     short    u_mask;
69     short    u_error;
70     short    u_cdir;
71     short    u_rdir;
72     short    u_pdir;    /* parent directory, gezet door namei */
73     short    u_diract;  /* offset in directory, voor nieuwe entry */
74     Dir u_dirent;
75 } User ;

77 EXT Superblok sbk;
78 EXT User u;
79 EXT char sysnaam[SYSLEN];
80 EXT int verbose;

81
82 /* prototypes */
83
84 void zien(void);
85 void initproc(void);
86 void makfile(char *cpn, short mode, short size, char opvul);
87 void lnk(char *cpn, char *new);
88 void unlnk(char *cpn);
89 void makdir(char *cpn);
90 void curd(char *cpn, int wortel);
91 void fstatus(char *cpn);
92 void chmode(char *cpn, short mode);
93 void chuid(char *cpn, char uid, char gid);
94 void mkfs(short ninode, short nblok);
95
96 short namei(char *cp);
97 void SchrijfSuperBlok(void);
98 void SchrijfInode(int ino, Inode *ip);
99 void SchrijfBlok(int blkno, char *bp);
100 Superblok* LeesSuperBlok(void);
101 Inode* LeesInode(int ino, Inode *ip);
102 char* LeesBlok(int blkno, char *bp);
103 void DrukSuperBlok(Superblok *sb);
104 void DrukInode(Inode *ip);
105 void DrukBlok(char *bp);
106 void DrukProc(void);

```

```

/*
2  * hfs.c : een filesysteem simulatie programma
      *
      *               aanpassingen 15 oktober 2004
4  */

6  #include <sys/types.h>
   #include <sys/stat.h>
8  #include <stdio.h>
   #include <stdlib.h>
10 #include <string.h>
   #include <fcntl.h>
12 #define EXT
   #include "hfs.h"
14
   extern int  optind;
16  extern char *optarg;
   int verbose;
18
   main( int    argc, char  *argv[] )
20 {
       int      cc;
22   char      str[SYSLEN] ;
       int      a1, a2, a3, i, k;
24   char      *ptr;
       char      *cpn;
26
       strncpy(sysnaam, NAAMFS, SYSLEN);
28   while ( (cc=getopt(argc, argv, "vf:h")) != EOF )
   {
30       switch (cc)
       {
32         case 'f':
           strncpy(sysnaam, optarg, SYSLEN);
34           break;
           case 'v':
36             verbose++;
           break;
38         case 'h':
           default:
40             fprintf(stderr, "gebruik: ./hfs [-v] [-f _naam]\n");
           break;
42       }
   }
44   if ( access(sysnaam, 0) >= 0 )
   {
46       LeesSuperBlok();
       DrukSuperBlok(&sbk);
48   }
   /* initialisatie van de user structuur */
50   u.u_uid = u.u_gid = 0;   u.u_cdir = u.u_rdir = 1;   u.u_mask = 026;
   while ( fgets(str, SYSLEN, stdin) != NULL )
52   {
       if ( (cpn = strchr(str, '\n')) != NULL )
54         *cpn = '\0';

```

```

    if ( verbose )
56         printf("Gelezen: %s\n", str);
    i = 1;
58     while ( str[i] == '_' )
        i++;
60     cpn = &str[i];
    switch(str[0])
62     {
    case 'f' :
64         while ( str[i] != '_' )
            i++;
66         str[i] = '\0';
        if ( verbose )
68         {
            for(k=0; k<=i; k++)
70                 printf("%2c", str[k]);
            printf(" : %d\n", i);
72         }
        i++;
74         a2 = strtol(&str[i], &ptr, 8);          /* mode */
        a3 = strtol(ptr+1, &ptr, 10);          /* lengte */
76         makfile(cpn, (short)a2, (short)a3, *(ptr+1) );
        break ;
78     case 'm' :
        makdir(cpn);
80         break ;
    case 's' :
82         fstatus(cpn);
        break ;
84     case 'u' :
        unlnk(cpn);
86         break ;
    case 'l' :
88         while ( str[i] != '_' )
            i++;
90         str[i] = '\0';
        i++;
92         lnk(cpn, &str[i]);
        break ;
94     case 'z' :
        zien() ;
96         break ;
    case 'i' :
98         a1 = strtol(cpn, &ptr, 10) ;          /* aantal inodes */
        if ( a1%2 ) a1++;
100         a2 = strtol(ptr+1, &ptr, 10) ;          /* aantal blokken */
        mkfs((short)a1, (short)a2) ;
102         break ;
    case 'a' :
104         u.u_mask = strtol(cpn, &ptr, 8) ; /* mask */
        break ;
106     case 'e' :
        u.u_uid = strtol(cpn, &ptr, 10) ; /* uid */
108         break ;

```



```

110         case 'g' :
            u.u_gid = strtol(cpn, &ptr, 10) ; /* gid */
            break ;
112         case 'c' :
            curd(cpn, 0);
114             break ;
            case 'r' :
116                 curd(cpn, 1);
                    break ;
118         case 'n' :
            a1 = strtol(&str[i], &ptr, 8) ;    /* mode */
120             chmode(ptr+1, (short)a1);
                break ;
122         case 'd' :
            a1 = strtol(&str[i], &ptr, 10) ; /* uid */
124             a2 = strtol(ptr+1, &ptr, 10) ;    /* aantal blokken */
                chuid(ptr+1, (char)a1, (char)a2);
126                 break ;
            case 'p' :
128                 DrukProc();
                    break ;
130         case 'q' :
            exit(0);
132         default :
            case '?' :
134                 printf("i_f_m_l_u_ : s_p_z_q_ : a_e_d_n_c_r_\n") ;
                    printf("i_aantal_inodes_aantal_blokken_a_mask_\n") ;
136                 printf("f_naam_mode_size_teken_uid_g_gid_\n") ;
                    printf("m_naam_d_uid_gid_naam_\n") ;
138                 printf("l_naam_naam_n_mode_naam_\n") ;
                    printf("u_naam_c_naam_\n") ;
140                 printf("s_naam_r_naam_\n") ;
                    break ;
142             }
            if ( verbose )
144                 DrukProc();
                memset(str,0,SYSLEN);
146     }
}
148
149 void DrukProc(void)
150 {
151     printf("U: Id_%d,%d_Mask_%.3o_Err_%d_Cd_%d_Rd_%d_Pd_%d",
152         u.u_uid, u.u_gid, u.u_mask, u.u_error, u.u_cdir, u.u_rdir, u.u_pdir);
153     printf("_diract_%d_Entry_|%.2d|%-7.7s|\n",
154         u.u_diract, u.u_dirent.d_ino, u.u_dirent.d_naam);
155 }
156
157 void zien(void)
158 {
159     char buf[BLOKSIZE];
160     Superblok *sb;
161     Inode *ip;
162     char *bp;

```

```

164     int fd, i, j, k, nin, nbuf;

165     fd = open(sysnaam, O_RDONLY);
166     if ( fd == -1 )
167     {
168         fprintf(stderr, "Er is nog niets te zien\n", sysnaam);
169         return;
170     }
171     if ( read(fd, buf, BLOKSIZE) <= 0 )
172     {
173         fprintf(stderr, "Er is nog niets te zien\n", sysnaam);
174         return;
175     }
176     sb = (Superblok *)buf;
177     nin = sb->s_inode/2; nbuf = sb->s_blok;
178     DrukSuperBlok(sb);
179     for(i=1; i<=nin; i++)
180     {
181         read(fd, buf, BLOKSIZE);
182         ip = (Inode *)buf;
183         for (k=1; k>=0; k--, ip++ )
184         {
185             printf("%2d: ", 2*i-k);
186             DrukInode(ip);
187         }
188     }
189     for(i=1; i<=nbuf; i++)
190     {
191         read(fd, buf, BLOKSIZE);
192         bp = (char *)buf;
193         printf("%2d: ", i);
194         DrukBlok(bp);
195     }
196     close(fd);
197 }

198 void DrukSuperBlok(Superblok *sb)
199 {
200     int i;
201
202     printf("%2d %2d", sb->s_inode, sb->s_blok);
203     for (i=0; i<MAXINODE; i++) printf("%.2x", sb->s_fi[i]);
204     printf("  ");
205     for (i=0; i<MAXBLOK; i++) printf("%.2x", sb->s_fb[i]);
206     printf("\n");
207 }

208 void DrukInode(Inode *ip)
209 {
210     int j;
211
212     printf("%.2x %2d %4x %3d: ", ip->i_uid, ip->i_link,
213                                     ip->i_mode, ip->i_size);
214     for(j=0; j<BLKADDR; j++)

```

```

        printf("%2d", ip->i_blok[j]);
218     printf("\n");
    }
220
    void DrukBlok(char *bp)
222 {
        int j;
224
        for (j=0; j<BLOKSIZE; j++)
226             if ( *(bp+j) < 32 )                /* non-printable tekens */
                printf("%2x", *(bp+j) );
228             else
                printf("%2c", *(bp+j) );
230     printf("\n");
    }

```

Resultaat van ino = namei(cpn);

- ino heeft de waarde 0: het bestand bestaat (nog) niet
  - u.u\_pdir: inodenummer van de directory waarin het bestand gecreëerd wordt
  - u.u\_diract: offset in de datablokken van de directory waar de nieuwe inodenummer en de naam van het bestand kan toegevoegd worden
  - u.u\_dirent.d\_naam: laatste component van de padnaam van het bestand dat gecreëerd wordt
- ino heeft de waarde van de inode van het gevonden bestand
  - u.u\_pdir: inodenummer van de directory waarin het bestand gevonden is
  - u.u\_diract: offset in de datablokken van de directory waar de **dir**-entry van het gevonden bestand staat
  - u.u\_dirent.d\_naam: laatste component van de padnaam van het gevonden bestand
  - u.u\_dirent.d\_ino: inodenummer van dit bestand

```

/*
2   *      namei.c          met eventuele stub : om te kunnen testen
   */
4
   #include <sys/types.h>
6   #include <sys/stat.h>
   #include <stdio.h>
8   #include <stdlib.h>
   #include <fcntl.h>
10  #include <string.h>
   #define EXT extern
12  #include "hfs.h"

14
   #ifndef STUB
16  short namei(char *cpn)
   {
18      static Inode    ipb;
        Inode    *ip;
20      char    *cpl;
        short wdir = 0;
22

```

```

printf("└direct└offset└in└dir└└"); scanf("%hd%c", &u.u_direct);
24 printf("└parent└inode└nummer└└"); scanf("%hd%c", &u.u_pdir);
printf("└inode└nummer└└"); scanf("%hd%c", &wdir);
26 if ( wdir != 0 )
{
28     ip = LeesInode(wdir, &ipb);
    if ( verbose && ip )
30     {
        printf("%3d└└", wdir);
32         DrukInode(ip);
    }
34     u.u_dirent.d_ino = wdir;
}
36 else
{
38     ip = (struct inode *)NULL;
    u.u_dirent.d_ino = 0;
40 }
/* afsplitsen van de laatste component van de padnaam */
42 cpl = strchr(cpn, '/');
if ( cpl == NULL )
44     cpl = cpn;
else
46     cpl++;
memset(u.u_dirent.d_naam, '\0', NAMELEN);
48 strncpy(u.u_dirent.d_naam, cpl, NAMELEN);
if ( verbose )
50     printf("dirent└%d└%-7.7s└└└direct└%d└└└pdir└%d└n",
        u.u_dirent.d_ino, u.u_dirent.d_naam, u.u_direct, u.u_pdir);
52 u.u_error = 0;
return wdir;
54 }
#else
56 short namei(char *cpn)
{
58     printf("namei└bestand└└%s└n", cpn);
    return 1;
60 }
#endif

```

## 5 Oefening 3: Proces-beheer.

### Programma 1:

Het parent programma (P) berekent de tijd verlopen sinds boottime en de effectief gebruikte CPU tijd in user- en in systemmode (*times()*), schijft deze drie waarden uit en creëert twee kinderen (K1 en K2).

K1 voert het programma uit dat gegeven wordt door de omgevingsvariabele **PROG** uit.

```
setenv PROG /usr/bin/date | progpap = getenv("PROG");
```

K2 berekent het aantal seconden sinds 1/1/1970 (*time()*), schrijft dit getal in hexadecimale vorm uit en in leesbaar datumformaat (*ctime()*), en roept de routine *pause()* op.

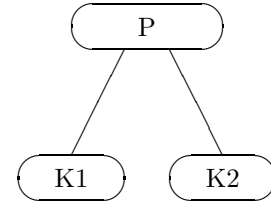
P wacht (in *wait*) totdat K1 een *exit* doet. Het proces schrijft de terugkeerwaarde en de status uit. Dan stuurt P signaal SIGUSR1, mbv. *kill* naar K2, en wacht terug (in *wait*).

K2 reageert met de default actie op het signaal en doet dus een *exit*.

Gevolg daarvan is dat P terug uit *wait* komt. Het proces schrijft de terugkeerwaarde en de status uit. Om de waarde van deze status te begrijpen, kan best de waarde van SIGUSR1 ook uitgeschreven worden. Het doet dan opnieuw een *times* system call, schrijft de resultaten hiervan uit en doet dan zelf een *exit*.

Bij elke creatie van een proces, wordt de proces-id van het nieuwe proces en de parent-proces-id uitgeschreven. Na elke *wait* wordt de terugkeerwaarde en de bijhorende status uitgeschreven.

Bij de printf's van P wordt de tekst niet voorafgegaan door een tab, bij de printf's van K1 door één tab, en bij de printf's van K2 door twee tabs.

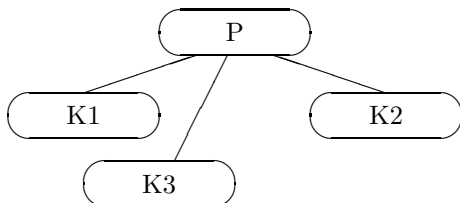


### Programma 2: bouwt verder op programma 1:

Het programma heeft één argument: het aantal seconden dat het programma maximaal mag draaien. Deze functionaliteit kan gerealiseerd worden met de *alarm* system call.

In de parent wordt na het ontdekken van het einde van K1 een oneindige lus gestart: verwittigen van K2 via SIGUSR1, wachten op antwoord van K2, creëren van een proces K3, wachten op het einde van K3. Indien het verwittigen van K2 niet lukt, wordt de oneindige lus afgebroken.

In K2 wordt een programmalus voorzien die NKEER (bijv. gelijk aan 50) uitgevoerd wordt.



In plaats van de default actie in K2, vangt K2 het signaal SIGUSR1 op. K2 meldt aan P mbv SIGUSR2 dat er terug een kind (K3) mag gecreëerd worden dat het programma aangegeven door **PROG** uitvoert. Het proces K2 wacht (in *pause()*) totdat er terug een signaal SIGUSR1 binnenkomt van P.

In de printf's in K3-achtigen wordt de tekst voorafgegaan door één tab.

Resultaat is een heleboel creaties van een K3 proces dat PROG (bijv. /usr/bin/date) uitvoert, waarvoor het K2 proces elke keer de "toelating" geeft.

In dit programma moeten signal catchers voorzien worden. De code in deze catchers moet zo beperkt mogelijk zijn: het uitschrijven van de proces-id en een kort bericht en het eventueel herinstalleren van de signaal catcher.

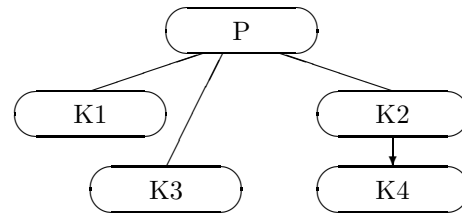
Net zoals in programma 1 wordt bij elke creatie van een proces, de proces-id van het nieuwe proces en de parent-proces-id uitgeschreven. Na elke *wait* wordt de terugkeerwaarde en de bijhorende status uitgeschreven.

**Programma 3:** bouwt verder op programma 2:

Bij het ontvangen van SIGUSR1 van P wordt in de catcher routine van K2 een kind K4 gecreëerd dat `/usr/bin/who am i` uitvoert. De tekst in de printf's van K4 wordt voorafgegaan door drie tabs.

Vier mogelijke versies bij het starten van P:

- d:** default actie op SIGCLD in K2
- i:** ignore actie op SIGCLD in K2
- s:** specifieke actie op SIGCLD in K2:  
het uitschrijven van een boodschap
- S:** specifieke actie op SIGCLD in K2:  
maar zombie-kind (K4) laten verdwijnen

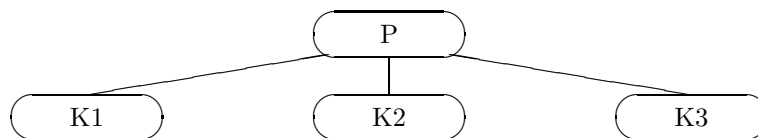


**Programma 4:**

Het parent programma P is een scheduler die via *round robin* een aantal kind-processen na elkaar laat uitvoeren. Elk kind proces voert een oneindige lus uit waarin het wacht op signaal SIGUSR1. Wanneer dit signaal ontvangen wordt, schrijft het kind in een oneindige lus een specifieke letter uit. Na een *time-slice* wordt door de parent het kind-proces in uitvoering gestopt, door middel van signaal SIGUSR2 en wordt een volgend kind proces opgestart door middel van signaal SIGUSR1. Wanneer het laatste kind gestopt is, wordt terug het eerste kind gestart.

Het parent proces heeft drie argumenten: het aantal kind processen, de lengte van de time-slice en de letter voor het eerste kind proces.

Voor het kind proces moet een apart main programma geschreven worden, dat in de parent na de fork ge-exec-ed wordt. Op die manier kan het kind proces afzonderlijk uitgetest worden. Nadat een kind gecreëerd is, schrijft het zijn proces-id en dat van het ouderproces uit. Wanneer de proces-id oneven is, wordt een nieuwe proces groep gestart (*setpgp()*). Een kind proces mag maximaal 300 seconden draaien. Na deze tijd wordt naar de parent signaal SIGQUIT gestuurd (zie verder).



In een tweede versie moet er ook van kind proces gewisseld worden wanneer op **BREAK** (of **CTRL-C**) gedrukt wordt.

Signaal SIGQUIT (**CTRL-\**) kan gebruikt worden om het parent proces te onderbreken: dit proces zal dan naar alle gemaakte kindprocessen SIGTERM sturen en daarna zelf een exit uitvoeren.

**Demo+verslag:** tijdens de laatste praktijkzitting.

Het verslag bevat een listing van de geschreven routines en een korte tekst (+ figuren) waarin de verschillende programma's besproken worden.

Naam:

---

1. `w`  
Hoelang draait het systeem al?
  
2. `/etc/swapinfo`  
Hoeveel verschillende swapspaces zijn er?  
  
Wat is de totale grootte van de swapruimte?
  
3. `vmstat -s`  
Wat is er het meeste gebeurd: CPU context switches of device interrupts?
  
4. `vmstat -z`  
Beantwoord straks de laatste vraag.
5. `vmstat -S`
6. `vmstat 2`  
Hoe kan je de getalwaarde in de kolom *sy* (system calls) of *in* (device interrupts) verhogen?
  
7. `top`  
Hoeveel CORE's zijn er en wat is hun activiteitspercentage?
  
  
8. `cp /student/e4/hfd/lang lang`
9. maak een bijkomende terminal en start `lang`
10. `ps -lu e4xx`  
Wat is het proces met de laagste prioriteit?
  
11. `ps -fu e4xx`  
Teken de procesboom.

12. `cd /dev`

13. `ll *mem`

Wat is de reden waarom *mem* en *kmem* dezelfde major device nummer hebben?

14. `cd dsk && ll`

15. `cd ../rdsd && ll`

Wat is het verschil tussen deze twee lijsten?

16. `cd .. ; ll /dev/tty`

Wat is de major en minor device nummer?

17. `cd` naar homedirectory

18. `cat > /dev/tty`

appel peer citroen

CTRL-D

Hoe komt het dat iedereen dat op het eigen scherm ziet?

19. `stty -a`

Wat is de grootte van het venster waarin je werkt?

20. `stty erase q`

21. `vi iets`

i ltqs ESC

22. `size lang`

Wat is de grootte van de geïnitieerde data en van bss?

23. `nm lang`

24. `nm -g lang`

Hoe groot is de functie *afdruk*?

25. `chatr lang` Geef een opvallende karakteristiek van deze executable?

26. Hoeveel system calls zijn er tijdens deze sessie uitgevoerd?

27. Hoeveel *pages* zijn er outgepaged, wat is het CPU gebruik van de page stealer?