Introductie tot Makefiles

Mattias Holm & Kristian Rietveld

Universiteit Leiden
The Netherlands

Aanwijzingen practicum

- Readability/existence check uitvoeren voordat fork() plaats zal vinden.
- Controleren return value execv() blijft nodig.

Aanwijzingen practicum

- Readability/existence check uitvoeren voordat fork() plaats zal vinden.
- Controleren return value execv() blijft nodig.

- Permissiecheck met fstat(): goed over nadenken.
  - Eigenlijk programmeer je een eigen versie van access().

Source code eisen

- Waar letten we op bij het beoordelen van de kwaliteit van de source code?
  - Structuur
  - Indentatie (is deze consistent?)
  - Memory handling
  - Commenting

Commenting

- Goede code is te begrijpen zonder commentaar.
- Soms zijn comments toch nodig.
- Wanneer wel, wanneer niet?

Commenting (cont.)

- Wanneer niet:
  - Namen van variabelen uitleggen die al duidelijk zijn.
  - Een enkele regel code precies uitleggen, terwijl dat zonder comment ook duidelijk is.
  - Etc...

Commenting (cont.)

- Wanneer wel:
  - Als een bepaalde beslissing of trade-off is gemaakt. Waarom?
  - Documentatie per functie.
  - “Secties” in grote source file aanduiden.
  - Als een fragment code niet meteen duidelijk is.
  - FIXME/TODO comments.
  - Als iets tegenintuitief is.

Commenting (cont.)

```
int ret_val;   // Store return value

/* increment counter */
i++;

/* FIXME: verify all members are freed */
free (object);

/* call function execv */
execv (path, argv);

if (...)
  ...
else
  // cannot bubble down any more, heap order satisfied
  break;
```

Introductie tot Makefiles

Mattias Holm & Kristian Rietveld

Universiteit Leiden
The Netherlands

make

- Make wordt voornamelijk gebruikt om het compileren van software te automatiseren.
- Eigenlijk: het genereren van files gebaseerd op andere files.
- De file wordt alleen gegenereerd als dat nodig is:
  - De file bestaat nog niet, of
  - de gegenereerde file is ouder dan de files waarop het is gebaseerd.

make rules

- Een Makefile bestaat uit "rules".
- Deze hebben de volgende vorm:

```
target:    dependencies
           one or more commands
```

- Belangrijk! Gebruik tabs en geen spaties.

```
test:      test.c test.h
           gcc -Wall -o test test.c
```

- Invocatie: make test (make <target>)

Speciale targets

- Het belangrijkste target is "all".
- Hier kan je aangeven welke targets moeten worden gegenereerd als je make aanroept zonder argumenten.

```
all:        test

test:       test.c test.h
            gcc -Wall -o test test.c
```

Automatic variables

- Er zijn een aantal speciale variabelen die je in make rules kan gebruiken:
  - $@ bevat naam van het target
  - $< bevat naam van de eerste dependency
  - $^ bevat naam van alle dependencies

Generieke "rules"

- Je wilt niet voor elke C file een aparte make rule schrijven.
- Het is mogelijk om een generieke rule te schrijven die wordt gebruikt als er geen expliciete rule is gevonden.

```
%.o:        %.c
            gcc -Wall -g -c $<
```

Variabelen

- Zelf kun je ook variabelen introduceren.
- Vaak wordt dit gebruikt om de compiler flags te specificeren.
- Merk op dat de naam tussen haakjes moet staan bij gebruik.

```
CFLAGS = -Wall -g

test:     test.c
          gcc $(CFLAGS) -o test test.c

test:     test.c
          gcc $(CFLAGS) -o $@ $<
```

Iets completer voorbeeld

```
CFLAGS = -Wall -g
OBJECTS = main.o feature1.o feature2.o

all:        test

test:       $(OBJECTS)
            gcc $(CFLAGS) -o $@ $^

%.o:        %.c
            gcc $(CFLAGS) -c $<
```

Wildcard operator

- In plaats van zelf alle files op te geven, kan je dit ook automatisch doen.
- Dit kan met de wildcard operator:

```
FILES = $(wildcard %.c)
```

- De meeste grote projecten gebruiken dit *niet*, om te voorkomen dat files onbedoeld in de executable terecht komen.

Substitution references

- In make kan je een handige operator gebruiken om de extensie van een lijst van files te veranderen.
- Syntax:

```
DEST = $(VAR:A=B)
```

- In variabele VAR, vervang elke "A" met "B".

Substitution refs (vervolg)

- Dit wordt vaak gebruikt om extensies te wijzigen.

```
OBJECTS = $(FILES:.c=.o)
LATEX_SOURCE = $(FILES:.pdf=.tex)
```

- Zoals we nodig hebben in ons voorbeeld:

```
FILES = $(wildcard %.c)
OBJECTS = $(FILES:.c=.o)
```

If expressie

- In make kan je if expressies gebruiken.
- Vaak gaat het om string vergelijkingen of kijken of een bepaalde variabele is gedefinieerd.

```
ifeq ($(DEBUG), 1)
      CFLAGS="-Wall -g"
else
      CFLAGS="-Wall -s -O3"
endif
```

- Je kan variabelen definiëren op de make command line: *make DEBUG=1*

Handige make opties

- -n print commando's zonder uit te voeren
- -C <dir> draai make in opgegeven directory
- -j N draai N jobs tegelijkertijd (nuttig op multi-core machines)

Makefiles genereren

- Schrijft iedereen dan zelf Makefiles?
- Nee, er zijn weer tools om deze automatisch te genereren:
  - cmake
  - autotools (autoconf, automake)
  - qmake
  - etc.

Practicum

- Practicum in zaal 411.