Embedded Systems Engineering

Operating Systems

practicumhandleiding

Jos Onokiewicz, Marc van der Sluys

Klassen: ES2, ES2D

9 februari 2017

versie 3.3

Hogeschool van Arnhem en Nijmegen

Embedded Systems Engineering

Documenthistorie

Versie	Datum	Wie	Wijzigingen
1.0		Jos Rouland	-
2.0		Jos Rouland	Geen. Uitgevoerd door Jos Onokiewicz
2.1	26-08-2013	Jos Onokiewicz	Layout verbeterd. Alle opdrachten in één hoofdstuk geplaatst (hoofdstuk 2). Utilities htop en tmux laten gebruiken.
3.0	26-02-2015	Jos Onokiewicz	Vernieuwde opdrachten. Onder andere gebruik queue (dynamisch geheugengebruik).
3.1	31-08-2015	Jos Onokiewicz	Signal en driveropdracht aangepast. Handleiding tmux toegevoegd. Qt-Creator als IDE te gebruiken.
3.2	20-10-2015	Jos Onokiewicz	
3.3	09-02-2017	Marc van der Sluys	Vele kleine aanpassingen en verduidelijkingen. Meer Engels in code. Opgave 10.2,3 -> Bijlage 1, Opgave 11.1,2. Verstrek code via #OO i.p.v. Bijlage 1, 3. Gebruik man page i.p.v. Bijlage 4.

Inhoud

Inleiding

1.1 Vereiste voorkennis

Een goede kennis en vaardigheid in het programmeren met C is vereist, zoals geleerd in het vak INF1. Je dient vaardigheid te bezitten op het terrein van het onderkennen van functies met een geschikte parameterlijst en in het modulair programmeren. Zorg voor een hoge codekwaliteit: kies informatieve namen voor variabelen en functies, en pas een goede consequente lay-out toe! Vermijd het gebruik van “magic numbers” door #define’s met informatieve namen te gebruiken.

1.2 Uitvoering van de opdrachten

Er bestaat een groot aantal C-functies zowel in de standaardbibliotheek als voor het Linux operating system. Tijdens de voorbereiding en uitvoering van het practicum is het noodzakelijk een reference manual te kunnen raadplegen (man pages, internet en/of boeken). Bestudeer ook uitvoerig de codevoorbeelden in het OPS dictaat als voorbereiding op het practicum.

Een deel van de opdrachten wordt gedaan op de Linux command line. Neem even tijd om hiervoor een teksteditor te kiezen (b.v. nano, emacs, vi, gedit, nedit, …). De IDE die voor andere opdrachten gebruikt wordt is QtCreator zoals toegepast in INF2, zodat we gebruik kunnen maken van de opgedane ervaringen in INF2. Omdat we geen gebruik maken van C++ en Qt voor een GUI moeten we gebruik maken van de volgende instelling voor C-codeprojecten: Non-Qt project, Plain C project.

Alle opdrachten moeten individueel worden uitgevoerd en toegelicht.

1.3 Eisen voor de afronding van de opdrachten

Er hoeven voor dit practicum geen verslagen te worden gemaakt. De vereiste correcte werking van de programma’s moet kunnen worden aangetoond met een demo. Hierbij kunnen vragen over de werking, achtergrond en context worden gesteld.

Aanvullende eisen:

- **Maak voor elke opdracht een eigen directory aan met de naam opdracht1, opdracht2, enz (mkdir <dirname>).**
- **Voorzie elke file met main() van een commentaarkop met je naam, opdrachtnummer en een datum.**
- **Zorg ervoor dat de namen van variabelen en functies en de lay-out van de programma's voldoen aan de 'C stijl- en programmeerrichtlijnen' van Jos Onokiewicz.**
- **De source code moet kunnen worden toegelicht.**
- **Beantwoord de gestelde vragen in je logboek, deze moeten kunnen worden toegelicht.**

1.4 Planning inleverdata van opdrachten

De uiterlijke inleverdata (lesweek) voor de practicumopdrachten Operating Systems zijn hieronder aangegeven. Het eindcijfer voor het practicum moet minimaal 6.0 bedragen om een eindcijfer voor de onderwijseenheid Operating Systems te krijgen en dus om de studiepunten (7,5 EC) te halen.

Voltijd en deeltijd (2 x 8 weken)

opdr.		1		2	3	4		5		6	7		8		9		10
week	1	2	3	4	5	6	7	8		1	2	3	4	5	6	7	8

De opdrachten geven belangrijke oefening voor de OPS tentamenvragen waarin C-code geschreven moet worden. Het is dus sterk aan te bevelen om de planning voor de inlevering van de opdrachten te volgen.

2 Voorbereiding opdrachten

2.1 Practicumvoorbereiding

Linux moet beschikbaar zijn. Advies: **Ubuntu 16.04 LTS**. Er is een image beschikbaar met een voorgeïnstalleerde versie van Ubuntu met alle benodigde software (zie OnderwijsOnline). Gebruik van je favoriete Linux-smaak (in een VirtualBox, als dual- of single-boot) wordt toegejuicht!

Voor de toe te passen IDE moet **QtCreator** geïnstalleerd zijn. Ga naar de Digia site om de juiste versie voor de Linux installer te downloaden.

Voor het zichtbaar maken van de uitvoering van tasks door het Linux operating systeem gaan we gebruik maken van het programma **htop**. Dit is onder Ubuntu te installeren met: **sudo apt-get install htop**. In htop kunnen we met <F4> een selectie maken van wat we willen zien door bijvoorbeeld (een deel) van de naam van een programma (task) in te voeren.

In een aantal opdrachten moeten we gelijktijdig gebruik maken van meerdere terminals. Om dit overzichtelijk te houden maken we gebruik van **tmux** (terminal multiplexer). Dit is onder Ubuntu te installeren met: **sudo apt-get install tmux**. Zie Bijlage 3 voor informatie.

Voor het testen van het geheugengebruik maken we gebruik van **valgrind**. Dit is onder Ubuntu te installeren met: **sudo apt-get install valgrind**. In valgrind vinden we ook **callgrind** een profiler en **helgrind** om naar threads te kunnen kijken.

Voor het gebruik van command-line parameters gebruiken we getopt. Dit is onder Ubuntu te installeren met: **sudo apt-get install getopt**.

2.2 Referenties C programmeren en Linux

Voor het gebruik van C-functies van de standaard bibliotheek en de standaard Linux API kunnen veelal de voorbeelden uit het OPS dictaat gebruikt worden. Soms is het noodzakelijk om de parameters in detail te kennen, om zo de vereiste functionaliteit in te stellen.

Informatie over de taal C is te vinden in:

- Sectie 3 van de manpages (man 3 <function name>)
- http://en.cppreference.com/w/c
- http://en.wikipedia.org/wiki/C_standard_library

Nuttige documentatie van Linux system calls:

- Sectie 2 van de manpages (man 2 <function name>)
- http://linux.die.net/man/ (section 2)

De meeste sites hebben zoekfuncties waarmee naar een bepaalde functie gezocht kan worden.

2.3 Compileren

De te maken opdrachten moeten op verschillende manieren gecompileerd worden. Dit heeft te maken met het feit dat we de verschillende aspecten van compileren moeten leren kennen.

2.3.1 Compileren op de command line

Het compileren van programma's via de command line gaat met het commando **gcc**:

```
gcc [options] <file-list>
```

De options worden niet afgedwongen door de compiler, ze zijn niet verplicht (vandaar dat ze tussen [ ] staan). Gebruik echter voor het compileren van een executable **altijd** de optie –Wall:

```
gcc -Wall -o progvb file1.c file2.c file3.c
```

Achter `-o` staat de naam van de output file, dus de naam van de executable (de file met de binaire machinecode voor de processor). De file list bevat de lijst met sourcecode files, die gecompileerd en gelinkt moeten worden. Deze lijst bevat geen headerfiles. Als je wiskundige functies in je programma gebruikt, moet je de optie **`-lm`** meegeven om de **math library** mee te linken. Voor het gebruik van de POSIX threadbibliotheek moet **`-lpthread`** meegenomen worden.

Na het compileren kan je binary worden uitgevoerd door:

```
./progvb
```

2.3.2 Compileren met make

Bestaat een programma uit een groot aantal modules die in aparte files zijn ondergebracht, dan spaart het gebruik van de make utility veel tijd. De make utility leest een speciale file (met als defaultnaam **Makefile**) die informatie bevat over hoe en uit welke files een executable programma moet worden samengesteld. De utility houdt zelf bij of er wijzigingen in een bepaalde file hebben plaatsgevonden sinds de laatste keer dat de executable werd gemaakt en herhaalt alleen die activiteiten, die opnieuw moeten worden uitgevoerd. De make utility biedt programmeurs een belangrijke hulp bij het onderhoud van (groepen van) programma's. We moeten daarbij niet alleen denken aan programma sources.

In de praktijk bestaat een applicatieprogramma vaak uit een aantal verschillende files, zoals C-sources, header files, assembler files en libraries in object code. Nadat elke source file vertaald is in een assembler file, worden de assembler files omgezet in object files. Ten slotte worden alle object files door de linker tot executeerbare object code gelinkt.

Omdat de verschillende files een bepaalde samenhang vertonen, moeten we alert zijn op de consequenties van veranderingen in een of meer files. De volgende vragen kunnen we ons bijvoorbeeld stellen:

- Welke source files moeten opnieuw gecompileerd worden, als een constante in een bepaalde header file wordt gewijzigd?
- Welke andere source files moeten opnieuw gecompileerd worden, als een bepaalde source file opnieuw gecompileerd is en in welke volgorde?

De verschillende stappen die tot een executeerbaar programma moeten leiden, worden in een zogenaamde makefile (ASCII-file) opgeslagen. Deze makefile draagt default de naam **Makefile**. Deze file wordt door de make utility gebruikt indien deze utility wordt opgestart met

```
make
```

Make voert alleen die stappen uit die noodzakelijk zijn. Make moet daarvoor drie gegevens hebben:

- de afhankelijkheden van ieder target van de verschillende source files
- de verschillende commando's die uitgevoerd moeten worden om een file (target) te maken
- het tijdstip waarop de verschillende files veranderd/gecreëerd zijn

Het eerste en tweede gegeven moet de programmeur aangeven in de makefile en het derde gegeven leest make uit de datering- en tijddata (time stamp) van de files. Als de time stamp van een te maken file (target) ouder is dan van een of meer van de files waarvan de target afhankelijk is, moet de target conform de opgegeven commando's geregenereerd worden.

Commentaarregels worden in de makefile voorafgegaan door een #. De afhankelijkheid wordt opgegeven door naam van de target file op te geven, gevolgd door een dubbele punt (:) met daarachter de lijst van files waaruit de target file moet worden opgebouwd. Deze files worden van elkaar gescheiden door spaties. Elk commando dat aangeeft hoe een target file wordt gemaakt, staat op een nieuwe regel en moet worden voorafgegaan door **een TAB** (geen spaties!). Let erop of je editor correct is ingesteld voor dit doel.

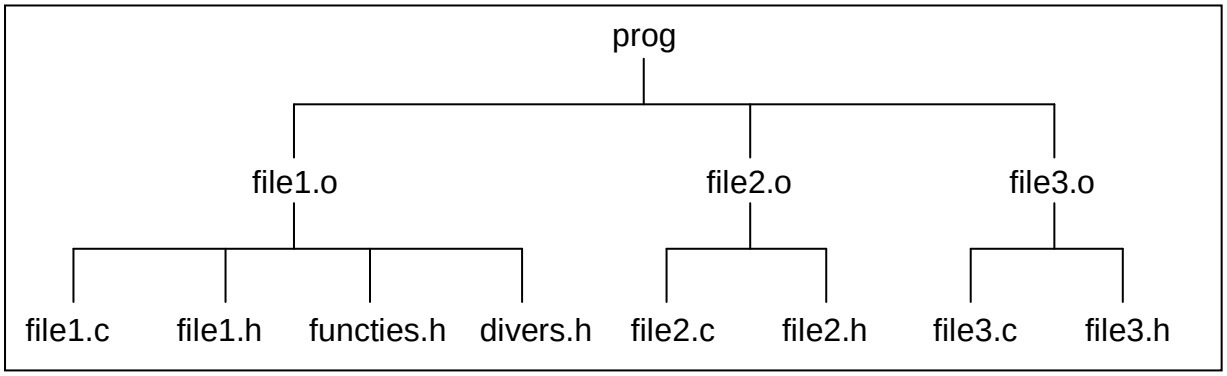


Fig. 1 Afhankelijkheid van files

Stel dat we een make file willen maken voor een project1 waarbij de afhankelijkheid van de files wordt aangegeven in fig. 1, dan komt de inhoud van de make file Make-project1 er als volgt uit te zien:

```
# This is the makefile for project1.
prog: file1.o file2.o file3.o
    gcc -Wall -o prog file1.o file2.o file3.o

file1.o: file1.c file1.h functions.h diverse.h
    gcc -Wall -c file1.c
# The option -c compiles only and avoids (premature) linking
file2.o: file2.c file2.h
    gcc -Wall -c file2.c

file3.o: file3.c file3.h
    gcc -Wall -c file3.c
```

2.3.3 Compileren met QtCreator

Verschillende IDE's kunnen worden gebruikt. We kiezen voor QtCreator. Hiermee kunnen we ook C-programma's compileren en debuggen. We moeten hiervoor wel een C-project aanmaken. QtCreator zorgt dan zelf voor de administratie in een .pro file voor het build management door files aan het project toe te voegen. Let erop dat we zelf in de .pro file de extra benodigde libraries voor de linker moeten aangeven. Tevens kunnen we in QtCreator verschillende plug-ins gebruiken zoals Valgrind. Let erop dat we standaard per QtCreator C-project maar één executable kunnen genereren. Zie Bijlage 4: QtCreator installeren en gebruik.

2.4 Voorbereiding op het practicum

De opdrachten zijn gerelateerd met de volgende stof:

- De belangrijkste Linux- en shell-commando's, taakbeheer, het gebruik van het root account en het lezen van manpages vind je in het dictaat *Efficient use of the Linux command line in the Bash shell*.

- Zie §3.1-§3.35, §5.3-§5.15 en §5.18 van het boek *Linux for programmers and users* voor meer informatie over Linux-commando's.
- In §11.6 van dat boek vind je uitgebreide informatie over de utility make, die het leven van een programmeur kan veraangenamen. In §11.8 wordt de debugger behandeld. Deze heb je nodig bij het opsporen van hardnekkige run-time fouten.

3 Opdrachten

Voor het uitvoeren van de opdrachten moeten eerst de voorbereidingen gedaan zijn zoals beschreven in hoofdstuk 2. Per opdracht is aangegeven hoe we moeten compileren. Maak voor iedere opdracht een aparte subdirectory aan, bijvoorbeeld met:

```
$ mkdir ex01
$ cd ex01
$ nano ex01.c
```

Opdracht 1. Command line parameters

Compileren: met de command line. Zie §6.1 en §6.2 van het OPS-dictaat.

De functie main() heeft de volgende interface voor het gebruik van command-line parameters (zie §6.1):

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

Voor het omgaan met een standaard command-line opbouw maken we gebruik van de volgende GNU C-library die dit makkelijk maakt: **getopt**. Geef de juiste linker-optie op in de command line om met deze library te linken. Gebruik het tweede uitgebreide codevoorbeeld (**`getopt_long()`** onder EXAMPLES) in de getopt manpage (**`man 3 getopt`**) als template voor deze opdracht. Compileer en run de code met verschillende command-line opties (b.v. **`-a b -c arg1 --create arg2`**) en verklaar wat er gebeurt.

a. Pas de code aan aan de volgende eisen:
   - zowel de standaard options als de long options moeten gebruikt kunnen worden.
   - indien geen command-line parameters worden meegegeven moet een usage mededeling gegeven worden.
   - de options **`-h`** en **`--help`** (zonder argument) geven informatie over alle te gebruiken command-line options.
   - de options **`-f <filename.txt>`** en **`--file <filename.txt>`** (verplicht argument) openen de tekstfile en tonen de eerste regel op stdout (hint: **`man fgets`**).
   - test of de file bestaat (tip: **`man fopen`** of **`man access`**). Als dit niet het geval is volgt een foutmelding op stderr. Gebruik hiervoor perror() (en errno; zie §6.2, **`man 3 perror`** en **`man errno`**).
   - test of de extensie van de filenaam gelijk is aan **`.txt`** (tip: **`man strncpy, man strlen`** en **`man strcmp`**). Indien niet correct, volgt een informatieve foutmelding op stderr.

- naast de optie **-f** en **--file** kan de extra optie **-e** en **--end** gegeven worden. In dat geval moet niet de eerste regel maar de laatste regel in de opgegeven tekstfile afgedrukt worden.
- Hints (uit de getopt manpage):
  i. In de struct **long_options[]**, gebruik het eerste veld voor de long option (zonder --), het tweede om aan te geven of een argument gewenst of zelfs verplicht is, zet het derde op 0 en geef in het vierde element de short option op (zie de regel met "create" als voorbeeld). De laatste regel bevat vier keer een 0.
  ii. Het derde argument van de functie **getopt_long()** is een string van alle short options. Een : achter een karakter geeft aan dat een argument verplicht is. Deze short options moeten overeenkomen met die in de **struct long_options[]**.
  iii. In de switch/case structuur wordt een actie ondernomen voor de verschillende short options.
  iv. Als een argument bij een optie wordt gegeven (b.v. een filename) dan is deze opgeslagen in de variabele **optarg**.

b. Voeg aan de parameterlijst van main() de environment pointer toe:

```
int main(int argc, char* argv[], char* envp[])
```

Druk alle environment parameters af in een lus met printf(). Vergelijk de output met die van het commando `$ env | less` op de command line.

Opdracht 2. Multitasking

Opdracht 2.1: syntax check

Compileren: met make en een zelfgemaakte Makefile. Zie het dictaat §5.9-5.13.

Ontwerp een programma display.c dat een karakter een aantal keren achter elkaar op het scherm afdrukt (dus niet telkens op een nieuwe regel). De methode van afdrukken, het aantal keren dat een karakter moet worden afgedrukt en het af te drukken karakter moeten via de command line opgegeven worden, bv.:

```
./display  e  1500  A
```

De algemene syntax van het commando is:

```
./display  <print type >  <# of times >  <print character>
```

De afdrukwijze mag uit slechts één karakter bestaan en wel:

e Het afdrukken geschiedt met de system call

```
(void) system ("/bin/echo –n …");
```

De optie **n** zorgt ervoor dat de newline wordt onderdrukt, zodat de karakters achter elkaar op het scherm worden geschreven.

Op de plaats van de puntjes moet de betreffende letter komen te staan. Daar de letter een variabele is, moet de string worden gemaakt. Het handigst gaat dit als volgt:

```
char echoCommand[] = "/bin/echo -n  ";
                     /* Note the two spaces after the n */
echoCommand[13] = cLetter;
```

Werkt de optie newline-onderdrukking niet op jouw (Apple) PC, probeer dan

```
char echoCommand[] = "bash –c \"echo -n  \"";
echoCommand[17] = cLetter;
```

p Het gebufferd afdrukken geschiedt met de C-functie printf().
Per keer wordt er één karakter afgedrukt (zonder nieuwe regel).

w Het niet-gebufferd afdrukken geschiedt met de C-functie write().
Per keer wordt er één karakter afgedrukt.

Het aantal keer moet een geheel positief getal zijn.

Het afdrukkarakter moet één karakter zijn (letter, cijfer of leesteken).

Download de file excercise_02.tar.gz op #OnderwijsOnline voor de code voor deze opgave. Pak de .tar.gz file uit in de directory `ex02/` met

```
$ tar xfz file.tar.gz
```

Bekijk de code goed. Wij hoeven hieronder alleen de file testFunctions.c aan te passen. Het hoofdprogramma roept de volgende drie functies aan, die in de file displayFunctions.c worden opgeborgen:

1. SyntaxCheck () stelt vast of de syntax van de aanroep van het programma correct is en geeft een foutmelding terug.
   Bij een syntaxfout wordt de functie DisplayError() aangeroepen en wordt het programma beëindigd. In het andere geval wordt de functie Work() aangeroepen.
2. DisplayError() toont een foutboodschap en geeft de correcte syntax van de aanroep van het programma.
3. Work() stuurt in een lus hetzelfde karakter een aantal malen naar het beeldscherm. In de lus wordt het afdrukkarakter telkens één keer op het display afgedrukt. Dit moet naar keuze op een van de 3 aangegeven manieren geschieden.

De definitie van ErrCode staat in een afzonderlijke header file errorCode.h:

```
typedef enum  {NO_ERR, ERR_PARS, ERR_TYPE, ERR_NR, ERR_CHAR}
              ErrCode;
```

De prototypes van de drie functies met documentatie (zie de code in excercise_02.tar.gz) staan in de header file displayFunctions.h:

```
ErrCode SyntaxCheck(int argc, char **argv);
void DisplayError(ErrCode errCode);
void Work(char prType, unsigned long int numberOfTimes, char
prChar);
```

De functie SyntaxCheck() maakt op zijn beurt gebruik van testfuncties die ondergebracht worden in de file testFunctions.c. De bijbehorende functieprototypes komen te staan in de file testFunctions.h.

Tips: Kopieer de file **excercise_02.tar.gz** van #OO en vul de ontbrekende onderdelen in. Dit betreft de functies **Work(), TestType(), TestNr()** en **TestChar().**
Gebruik **Makefile** voor het compileren en linken van het programma. Wat doet het commando **`make -j4`**? (Tip: zie **`man make`**).
Gebruik de standaardfunctie **strtoul()** (om een string om te zetten in een unsigned long integer en) om te controleren of de string daadwerkelijk een integer voorstelt. Zie **`man strtoul`** voor de syntax.

De functie strtoul() vertaalt (het begindeel van) een string in een unsigned long integer. Hierbij wordt iGrondtal als het grondtal van het gewenste talstelsel gebruikt. Dit grondtal mag 0 of 2 t/m 36 bedragen. De string mag worden voorafgegaan door white spaces optioneel gevolgd door '-' of '+'. Grondtal 0 wordt opgevat als grondtal 10, behalve als het volgende cijfer in de string 0 is. In dat geval wordt als grondtal 8 genomen. De cijfers 0 t/m 9 worden geaccepteerd als correcte cijfers. Bij grondtal 2 worden alleen de cijfers 0 en 1 geaccepteerd. Bij grondtal 16 zijn de cijfers 0 t/m 9 en de letters a t/m f (en A t/m F) correct (en wordt een voorafgaande 0x geaccepteerd) en bij 36 ook de andere letters van het alfabet.

Het beginadres van de te vertalen string is pcAdres. Het adres van het eerste karakter in de string dat niet wordt geaccepteerd, wordt opgeslagen op de locatie ppcStop. Het aanroepend programma moet dus het adres opgeven waar de functie strtoul() het adres kan opbergen van de plaats waar deze met vertalen is gestopt. Als de hele string een geldig getal is, wijst deze pointer naar het einde van de string, dus naar '\0'. Als voor ppcStop NULL wordt opgegeven, wordt het stopadres weggegooid.

Als de te vertalen string niet met een geldig cijfer begint, wordt 0 als antwoord teruggegeven aan het aanroepend programma. Als de te vertalen string met een '-' teken begint, wordt de absolute waarde van het getal teruggegeven.

Test het programma display.
Wat gebeurt er door het indrukken van Ctrl-C tijdens het uitvoeren van display? Wat gebeurt er als je Ctrl-Z indrukt? Wat gebeurt er als je vervolgens `$ fg` ingeeft? Wat betekent het commando fg? Hoe kun je de documentatie van fg vinden via de command line?

Opdracht 2.2: multitasking en prioriteiten

Start het ontwikkelde proces display twee keer concurrent op via de command line van de shell:

```
./display e 1500 . & ./display e 1500 +
```

Bekijk met htop de uitvoering en verklaar je waarnemingen in je logboek. Wat gebeurt er wanneer je de ampersand vervangt door een puntkomma? Verklaar!

De prioriteit van een taak is standaard 0. We gaan deze waarde veranderen en onderzoeken wat het effect is. De prioriteit mag alleen verlaagd worden. (De superuser mag de prioriteit wel verhogen.) Door de nice waarde te verhogen (tot maximaal +19) wordt de prioriteit verlaagd.

```
nice -19 ./display e 1500 . & nice -10 ./display e 1500 - &
./display e 1500 +
```

Verklaar je waarnemingen in je logboek.

Opdracht 2.3: printf

Test het proces display met afdruktype 'p'. Analyseer de verschillen t.o.v. afdruktype 'e' door display concurrent op te starten, met en zonder prioriteitverschillen.
Geef een verklaring in je logboek.
Bepaal de grootte van de printbuffer door een voldoend groot aantal karakters gebufferd weg te schrijven. Kies meer dan 2000 karakters om weg te schrijven.

Hint: printf() schrijft via een geheugenbuffer naar de display. Er wordt pas werkelijk naar de display geschreven bij een '\n' en als het geheugenbuffer vol is terwijl er nog data in moet worden opgeborgen. De inhoud van het printbuffer wordt ook op het display afgedrukt als het programma wordt afgesloten.

Neem aan het einde van het hoofdprogramma de functieaanroep sleep(3) op om 3 sec te wachten en experimenteer met het aantal af te drukken karakters. Zie **`man 3 sleep`** voor meer informatie.

Opdracht 2.4: write

Test het proces display met afdruktype 'w'. Analyseer de verschillen t.o.v. afdruktype 'e' en 'p' door display concurrent op te starten, met en zonder prioriteitverschillen.
Geef een verklaring in je logboek.
Maakt write() gebruik van een buffer?

Opdracht 3. Afsplitsen van identieke taken: fork(), wait()

Compileren: met make en een zelfgemaakte Makefile. Zie OPS-dictaat §6.3.

Kopieer de directory ex02 naar ex03, door vanuit ex02/ de volgende commando's uit te voeren:

```
make –j4          # generate the .o files
cd ..             # go up one directory
cp –r ex02 ex03   # copy the directory
cd ex03           # cd into the new dir
```

Als je in de directory ex03 bent, pas je de Makefile aan door de volgende regels (*rules*) te verwijderen (dus ook de bijbehorende commando's):

- testFunctions.o: …
- clean: …
- cleanall: …

Hierdoor kunnen we testFunctions.o niet overschrijven of verwijderen!

Doe verder het volgende:

```
$ rm testFunctions.c            # no longer needed!
$ mv errorCode.h errorCodes.h   # rename ('move') the file
```

Wijzig het hoofdprogramma display.c als volgt:

- Splits een identieke child af met een nice-waarde die de nice increment groter is dan die van de parent. Deze child drukt het tweede karakter af (m.b.v. Work()).
- Vervolgens splitst de parent (display) opnieuw een identieke child af met een nice-waarde die tweemaal de nice increment hoger is dan die van de parent. Deze tweede child drukt het derde karakter af (m.b.v. Work()).
- De parent drukt (m.b.v. Work()) het eerste karakter af (en heeft de laagste nice-waarde).
- Ten slotte wacht de parent op de exit-actie van de twee children. Gebruik dus twee keer de functie wait().

Proces display wordt bv. als volgt opgestart:

```
./display e 1500 6 + - .
```

Hierin is 'e' het afdruktype en 1500 het aantal keren dat de karakters '+', '0' en '-' elk worden afgedrukt. Het getal 6 is de nice increment die aangeeft dat de nice waarde voor het afdrukken van het tweede karakter 6 hoger is dan van het eerste (de standaard waarde) en dat die van het derde karakter weer 6 hoger is dan die van het tweede. Hierna volgen de drie af te drukken karakters zelf.

Voer de volgende aanpassingen uit:

- Pas het syntax checking aan het nieuwe commandoformaat aan. Wijzig hiertoe de functie SyntaxCheck() in de file displayFunctions.c. Gebruik hierbij de bestaande testfuncties uit de file testFunctions.h (en testFunctions.o).
- Voeg aan de file errorcodes.h de foutcode ERR_NICE toe.

Degene die een extra uitdaging niet uit de weg gaat, kan het programma display.c zodanig ontwerpen dat er een willekeurig aantal karakters (dus 1, 2, 3 of meer dan 3) worden afgedrukt, steeds met een nice waarde die het opgegeven getal hoger is dan die bij het vorige karakter. De algemene syntax van het commando is dan:

```
./display  <print type >  <# of times >  <nice increment>
<print character> [...]
```

Tip: Voor het bepalen en veranderen van de prioriteit van een taak kunnen de functies getpriority() en setpriority() worden gebruikt. Voor het veranderen van de huidige prioriteit kan ook de functie nice() worden gebruikt.

Test programma display.

Opdracht 4. Afsplitsen van nieuwe taken

Compileren: met make en een zelfgemaakte Makefile. Zie OPS-dictaat §6.3.

Kopieer vanuit je directory ex03 de Makefile, de source files display.c en displayFunctions.c en de header files errorCodes.h en displayFunctions.h naar je directory ex04. Wijzig hierin de namen:

- display.c in parent.c
- displayFunctions.c in syntaxCheck.c
- displayFunctions.h in syntaxCheck.h

en pas de Makefile aan. De header file errorCodes.h hoeft niet te worden gewijzigd.

Wijzig de file **parent.c** als volgt:

- Het hoofdprogramma parent.c roept de functie SyntaxCheck() aan voor een beperkte syntax check. Deze functie wordt opgeslagen in de file syntaxCheck.c.
- **Parent** splitst drie children af die achtereenvolgens elk (m.b.v. een exec-call) het proces **display** in directory ex02 opstarten met:
  - de nice-waarde van de parent,
  - een nice-waarde met één keer de nice increment hoger
  - en een nice-waarde met twee keer de nice increment hoger.

  De drie op deze wijze opgestarte display processen drukken respectievelijk het eerste, het tweede en het derde karakter af. Hints:
  - kies execl() – zie **`man 3 exec`**
  - geef als eerste argument het relatieve of absolute pad op naar de uit te voeren file
  - het tweede argument is argv[0]. Wat is het laatste argument?
- Print een foutmelding als de exec-call mislukt. Gebruik hiervoor perror() – zie **`man 3 perror`**.

    - Hint: wat gebeurt er na een succesvolle exec-call?
- Nadat de drie children zijn opgestart, wacht de parent op de exit-actie van deze drie children. Gebruik dus drie keer de functie wait().
- Bij een syntaxfout wordt het programma, c.q. de betreffende child beëindigd.

Wijzig de file **syntaxCheck.c** als volgt:
- Verwijder de functie Work().
- Wijzig de functie SyntaxCheck zodanig dat deze een beperkte syntax check op de meegegeven parameters uitvoert. Deze syntax check heeft alleen betrekking op het aantal parameters en de nice increment, want de andere parameters worden immers door display getest.

Pas de header file syntaxcheck.h in overeenstemming met syntaxcheck.c aan.

Het proces parent wordt bv. als volgt opgestart:

```
./parent e 1500 6 + - .
```

In het geval de student een programma ontwerpt dat geschikt is voor een willekeurig aantal children (0 …. n) is de algemene syntax van het commando:

```
./parent  <print type >  <# of times >  <nice increment>
<print character> [...]
```

Het proces parent maakt gebruik van het ongewijzigde proces display en de bijbehorende ongewijzigde functies in de file displayFunctions.c!

Test programma parent.

Extra uitdaging voor de snelle studenten: vervang de parallelle processen door threads. Wijzig hiertoe de functie Work() te in de threadfunctie WorkThread() en start in het hoofdprogramma voor elk af te drukken karakter een thread op.

Opdracht 5. Queue implementatie afronden en testen

Compileren: met bijgevoegde Makefile.

Maak gebruik van de C-code in **excercise_05.tar.gz** op #OO. Bestudeer de implementatie van de code voor de queue. Hier is gebruik gemaakt van een Circular Singly Linked List. Bron: http://ocw.utm.my/course/view.php?id=31, onderwerp 11b: Queue, Linked List implementation. Zie ook Figuur 5.5 in het OPS-dictaat.

De interface is overeenkomstig de C++11 STL class **queue**. De namen van de C-functies zijn overeenkomstig de C++ namen maar zijn voorzien van een post fix tekst Queue. Elke functies in C moet een unieke naam hebben omdat hier geen C++ function overloading gebruikt kan worden.

De code is te compileren met de bijgevoegde Makefile.

a. **Schets** de opbouw van een queue voorbeeld met drie nodes en de gebruikte pointers in de code.
b. Om memory leakage vast te stellen gebruiken we **Valgrind** voor alle volgende opgaven:

```
$ valgrind --leak-check=full ./queue
```

   Hoe zien we dat er geen of wel memory leakage is opgetreden? Als er memory leakage is geconstateerd probeer deze dan op te lossen in de code.
c. Schrijf de functie **sizeQueue()** die het aantal nodes in de queue returnt. Gebruik deze functie in showQueue() om de grootte van een queue te melden en test dit met een lege queue en met enkele queues met verschillende grootte. Welke big-O-notation O(1), O(n), of O(??) is op sizeQueue() voor zijn performance van toepassing? Waarom?
d. Schrijf de functie **deleteQueue()** die de gehele queue moet teruggeven aan het systeem met free() voor elke node in de queue. Test deze functie in main(). Wat moet de waarde van pLastNode worden?
e. Vul de functie **createQueue()** aan met de controle of de pointer naar de queue ongelijk is aan NULL. Indien de pointer een waarde heeft moet eerst de queue waarnaar verwezen wordt verwijderd worden met deleteQueue(). Test deze functie in main().
f. Creëer een memory leak in je code, en toon m.b.v. Valgrind aan dat dit gelukt is, en dat je dit kunt oplossen.

Opdracht 6. Signals voor synchronisatie

Compileren: met QtCreator. Zie OPS-dictaat H9.

Opdracht 6.1: signals ontvangen

Ontwerp een signal-ontvangend programma **getsignal.c** (project GetSignal) dat in een oneindige lus elke seconde steeds hetzelfde cijfer niet-gebufferd (gebruik write()) naar het beeldscherm stuurt. Gebruik de functie **sleep()** (**`man 3 sleep`**) voor een wachttijd van 1 s. Maak een geschikte signal-handling routine (SHR) overeenkomstig de nieuwere POSIX standaard.

Indien een bepaald signal (kies een signal-waarde tussen 20 en 31) wordt ontvangen, moet het volgende numerieke karakter op het beeldscherm geschreven worden. Begin met een '1'. Na het eerste signal wordt dit vervangen door '2', na het volgende signal door '3', enz. Na '9' moet '0' weer komen.

Gebruik voor de karakters in het programma altijd de concrete waarden ‘0’, ‘1’, etc. Vermijd dus hexadecimale codes. Let erop dat je gewoon bij een karakter een waarde kan optellen om een nieuw karakter te verkrijgen. Voorbeeld: ‘5’ + 1 levert een ‘6’.

Start de taak getsignal. Open een tweede terminalvenster op je systeem voor het opzoeken van de process id van proces getsignal heeft. Dit kan met het commando

```
ps -a
```

Het is nu mogelijk een signal via de command line van de shell naar de taak getsignal te sturen (zie **`man 1 kill`**):

```
kill -<signal code> <process id>
```

Test de werking van getsignal.

Een proces dat op de foreground draait, kan gestopt worden met Ctrl-C (SIGINT). Elk proces kan worden gestopt door signal 9 (SIGKILL) naar dat proces te zenden.

Opdracht 6.1a:
Voeg de volgende functionaliteit toe aan getsignal: wanneer getsignal wordt opgestart moet het zijn pid op het scherm afdrukken voor het aan de loop begint. Op die manier hoeven we niet steeds ps aan te roepen.

Opdracht 6.2: signals zenden

Ontwerp een programma **sendsignal.c** (project SendSignal) dat periodiek (bijvoorbeeld elke 4 s) een signal naar de ontvangende taak getsignal stuurt. Gebruik de functies kill() (zie **`man 3 kill`**) en sleep(). Sendsignal ontvangt de PID van het proces getsignal via een command line parameter:

```
./sendsignal <PID of getsignal>
```

Start eerst het proces getsignal (in de achtergrond) op en ga na welk pid getsignal heeft. Draag dit pid via een command-line parameter over aan het proces sendsignal.

Test de werking van sendsignal.

Opdracht 7. Communicatie via named pipes

Compileren: met QtCreator. Zie OPS-dictaat §7.5.

Maak via de command line een named pipe (FIFO) met de naam IDpipe.

Kopieer getsignal.c en sendsignal.c uit je directory ex06 naar ex07. Noem deze files respectievelijk **getsigsendpid.c** en **sendsiggetpid.c**. Zorg ervoor dat het proces getsigsendpid zijn process id via de fifo IDpipe verstuurt. Proces sendsiggetpid leest de PID uit deze fifo.

Test de werking van getsigsendpid en sendsiggetpid door beide processen met enige tussentijd na elkaar op te starten.

Maakt het wat uit als je deze processen in omgekeerde volgorde opstart?

Opdracht 8. Redirectioneren met unnamed pipes

Compileren: met QtCreator. Zie OPS-dictaat §7.2, 7.3.

Opdracht 8.1: proces filter met terminal invoer/uitvoer

Ontwerp een filterprogramma **filter.c** (project Filter) dat van ingevoerde karakters (string) de hoofdletters omzet in kleine letters (hint: **`man 3 tolower`**). De invoer van karakters vindt plaats via het toetsenbord, de uitvoer via de display. Afhankelijk van de gebruikte system calls kunnen hiervoor de file pointers stdin en stdout of de file descriptors 0 en 1 worden gebruikt. Als je een regel hebt ingetypt en op Enter drukt, moet de outputregel worden getoond en om een nieuwe inputregel worden gevraagd. Als een ESC-karakter (escape) wordt ingevoerd, moet het filterprogramma beëindigd worden. Op Linux-systemen gebeurt dit met **Ctrl-[**. De ASCII-code van ESC is 0x1B. Gebruik een #define om dit aan te geven.

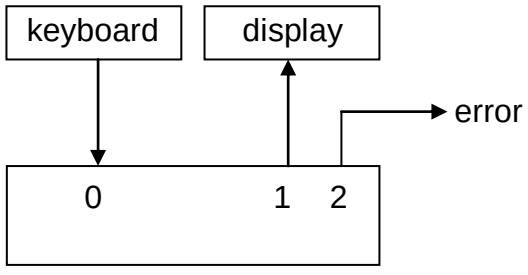
Test de werking van proces 'filter'.

Opdracht 8.2: Parent proces dat via unnamed pipes met proces filter communiceert

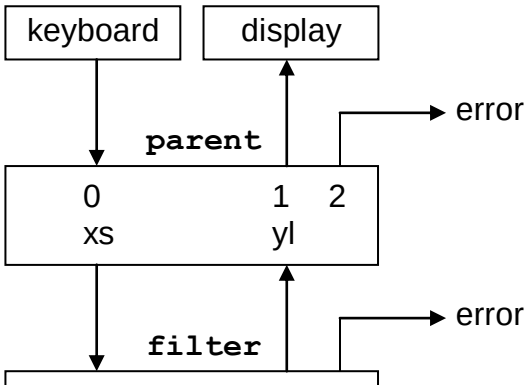
Ontwerp een programma **parent.c** dat de filtertaak filter (van opdracht 8.1) opstart als zijn child en ervoor zorgt dat het via een unnamed pipe data naar deze child kan sturen en via een tweede unnamed pipe data van deze child kan ontvangen (zie de figuur). Zet een kopie van de executable van filter in de directory van deze opdracht.

Parent leest data via stdin van het toetsenbord en zendt deze via zijn eerste unnamed pipe naar zijn child filter. De parent ontvangt het resultaat van zijn child filter via zijn tweede unnamed pipe en geeft dit via stdout op het beeldscherm weer. Bedenk dat unnamed pipes alleen bereikt kunnen worden via file descriptors (niet via file pointers). Mogelijk moet aan filter.c achter printf() of putchar() de system call fflush(stdout) worden toegevoegd voor correcte werking van parent.

begin/eindsituatie



werksituatie



Test de werking van proces 'parent'.

Opdracht 9. Threading, shared queue en semaforen

Compileren: met QtCreator.

In deze opdracht moeten we gebruik maken van de queue-software zoals gebruikt en afgerond in opdracht 5. Maak een programma ShareQueue dat in een aantal threads gebruik maakt van een shared queue (globale variabele). Maak geen gebruik van een shared memory module.

Maak de volgende threads:

- een thread die om de 2 seconden **data** (d.w.z., een node) **schrijft** in de shared queue (producer)
- een thread die om de 3 seconden **data schrijft** in de shared queue (producer)
- een thread die om de 4 seconden **data schrijft** in de shared queue (producer)
    - gebruik voor bovenstaande threads steeds **dezelfde producerfunctie**!
- een thread die om de 15 seconden de inhoud van shared queue **wegschrijft** naar een file (append) en toont op stdout. De queue moet na afloop van het tonen leeg gemaakt worden (consumer).

Eisen:

- zorg ervoor dat de data-inhoud per thread verschillend is, zodat we aan de getoonde uitvoer van de laatste thread kunnen zien of dit goed verloopt.
- gebruik sleep() in de herhaalussen om de gewenste herhaaltijd te verkrijgen.
- pas een semafoor of semaforen toe om de kritieke acties correct uit te voeren.
- pas een signal-handling routine toe om met Ctrl-C de herhaallussen in alle gebruikte threads te stoppen. Hier hebben we voor alle threads nog een shared variabele nodig die in de SHR op een andere waarde wordt gezet.

Bekijk de uitvoer van het programma en noteer je bevindingen.

Opdracht 10. Software device driver

Compileren: met make. Zie H6 van het dictaat Linux device drivers.

In deze opdracht breiden we een memory device driver uit. Compileer het voorbeeld van de memory device driver uit **excercise_10.tar.gz** van #OO. Toon met uitvoer-voorbeelden aan dat de verschillende programma's die deze driver gebruiken correct werken.

Voeg aan de driver de **ioctl** optie 'r' toe die de string in de inputbuffer omgekeerd in de outputbuffer zet. Daarvoor moet de laatste buffer ook eerst geleegd worden. Toon met een programma dat deze driver gebruikt aan dat deze driver dit ook correct uit-voert.

Bijlage 1: Hardware device driver

Deze bijlage bevat opdrachten die niet verplicht zijn bij OPS.

Opdracht 11. Hardware device driver

Compileren: met make. Zie H6 van het dictaat Linux device drivers.

Opdracht 11.1: device driver zonder interrupts

Maak een Linux device driver voor het I/O-bordje van fig. 1.2 van het dictaat "Linux device drivers". Noem de source code van de te ontwerpen driver IOdriver.c en de bijbehorende device file IOfile. De applicatie useIO om via deze device driver te communiceren met het I/O-bordje, is gegeven (zie #OO).

Maak een directory IObordje aan om alle files op te slaan. Kopieer de files runmake en Makefile van #OO in deze directory. Deze zijn nodig om de device driver te compileren. Kopieer ook de files useIO.c en useIO van #OO.

Volg de onderstaande stappen nauwkeurig op!
1. Zorg ervoor dat Oracle VM VirtualBox (en Ubuntu) uitgeschakeld zijn.
2. Sluit het I/O-bordje aan.
3. Ga naar 'Apparaatbeheer' van Windows.
   Kijk onder 'Poorten' op welke poort het I/O-bordje aanwezig is (bv. COM2). Bij 'Eigenschappen' moet hier als Fabrikant 'FTDI' zijn aangegeven).
4. Start VirtualBox op.
5. Selecteer Ubuntu zonder deze op te starten!
6. Ga naar 'Settings' en kies 'Seriële poorten'.
7. Vink aan 'Seriële poort activeren' en kies de bij Windows gevonden COM-poort voor het I/O-bordje (bv. COM2).
8. Noteer het IRQ-nummer (bv. 3) en het adres van de I/O-poort (bv. 0x2F8).
9. Klik op 'OK'.
10. Start Ubuntu.

In Ubuntu kun je via 'Apparaten', USB-apparaten controleren of hier inderdaad 'HAN-ESE Linux Device Board' voorkomt.

Bovenstaande werkzaamheden hoeven slechts één keer op je laptop te worden uitgevoerd. Een volgende keer kan volstaan worden met het aansluiten van het I/O-bordje en het hierna opstarten van VirtualBox en Ubuntu.

Het I/O-bordje

Het I/O bordje communiceert via een USB-poort met je laptop en wel met een snelheid van 19.200 bps, 8-bits karakters, geen pariteit en 1 stopbit. Het bevat voor de communicatie 4 switches (0 t/m 3) en 4 leds (0 t/m 3). Als de tekst 'Linux Driver Board HAN ESE' gewoon leesbaar is, bevindt de 'meestwaardige' led led3 zich boven en de 'minstwaardige' schakelaar, switch0 onder. Midden boven bevindt zich een resetdrukknop waarmee het I/O-bordje zonodig kan worden gereset.

Telkens als er een switch van stand verandert, wordt de nieuwe informatie van de switches serieel via de USB-poort naar de laptop gestuurd.

De UART

Een COM-poort (die een seriële RS-232 interface heeft) wordt aangestuurd via een aparte chip, een UART (Universal Asynchronous Receiver/Transmitter). Dit is een chip die asynchroon (met start- en stopbits) serieel communiceert en gelijktijdig kan zenden en ontvangen. Het universele zit in het feit dat de chip programmeerbaar is voor bv. 5-, 6-, 7- of 8-bits karakters met of zonder even (of oneven) pariteit. Verder is instelbaar met welke bitsnelheid wordt gecommuniceerd en welke interrupts er eventueel worden doorgegeven aan de microprocessor.

Bij gebruik van een USB-poort als COM-poort wordt er door een bijbehorende driver een RS-232-interface, en daarmee ook een UART, gesimuleerd. In een laptop is dat een 16550A-chip. De 16550D is een licht verbeterde versie van de 16550A. Zie bijlage 2 voor meer informatie over de UART. Uitgebreide informatie vind je in bv. http://www.ti.com/lit/ds/symlink/pc16550d.pdf.

Informatie die naar de leds wordt gestuurd, gaat via het 8-bits Transmitter Holding Register van de UART. Wordt in dit register geschreven, dan wordt de inhoud ervan via de USB-poort naar het I/O-bordje gestuurd en de betreffende leds gaan aan (bij een 1) of uit (bij een 0). Zetten we bv. 00001001 in dit register, dan zullen led3 en led0 aan gaan en led2 en led 1 uit.

Informatie van de schakelaars wordt door het I/O-bordje via de USB-poort naar de laptop gezonden en daar opgeslagen in het 8-bits Receiver Buffer Register van de UART. Staat hier bv. 00001000 dan betekent dit dat switch3 ON is en de overige drie switches OFF.

De gegeven applicatie useIO

Haal de file useIO.c en useIO op vanaf #OO. Deze applicatie wordt opgestart nadat de device driver is geladen en de device file is aangemaakt. Raadpleeg §10.1.7 hiervoor. Opstarten van de applicatie geschiedt met

```
./useIO IOfile
```

waarbij IOfile de naam is van de device file (niet de device driver).
Wil je eventueel wijzigingen aanbrengen in de applicatie, dan kan deze worden gecompileerd met

```
gcc -Wall -o useIO useIO.c
```

De applicatie useIO:

- opent de opgegeven device file voor lezen en schrijven
- installeert een signal handler voor signal 12
- voert de volgende commando's uit:

s	sluit de device file
r	leest de huidige stand van de schakelaars en drukt die af op het scherm

e geeft de interrupt van de UART door, die optreedt als er een nieuwe schakelaarstand wordt ontvangen
d blokkeert de interrupts van de UART
1000 stuurt het commando naar het I/O-bordje om led3 aan te zetten en de overige leds uit (het gaat hier om een voorbeeld van een string met vier enen en nullen)

Als de receive interrupt is enabled (en de device driver daarvoor is ingericht), zorgt de applicatie ervoor dat de informatie van de leds onmiddellijk (dus zonder r-commando) op het scherm wordt afgebeeld.

Zie #OO voor useIO.c.
Merk op dat de system **functie signal()** vervangen is door de **functie sigaction()**, samen met de **struct sigaction**. Deze is moderner, betrouwbaarder en thread safe en wordt aanbevolen.

Eisen aan de device driver IOdriver

Ontwerp de device driver voor het bordje zo dat deze met de applicatie communiceert met 8-bits karakters. Hierbij zijn de bits 4 t/m 7 “don’t cares”.

Benodigde includes voor de device driver:

```
#include <linux/cdev.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/ioport.h>
#include <linux/ioctl.h>
#include <asm/uaccess.h>
#include <asm/io.h>
#include "IOdriver.h"
```

I/Odriver.h bevat de declaraties van de functies die in de device driver worden gebruikt.

Activiteiten van de functies van de device driver:

init Vraag een devicenummer aan voor IOdriver, bepaal hieruit major en minor nummers en druk deze af met

```
printk (KERN_INFO "Major number = %d\n", IOdriver_major);
printk (KERN_INFO "Minor number = %d\n", IOdriver_minor);
```

Maak een struct cdev aan voor het device en registreer deze.
Test of de 8 adressen van de UART vrij zijn. Zo niet, geef deze dan vrij.
Vraag de 8 adressen van de UART aan.
Initialiseer de UART (de kristalfrequentie van de UART is 1,8432 MHz):
selecteer de divisor latches
zet de bitsnelheid op 19.200 bps
selecteer de dataregisters en
zet de karakterlengte op 8 bits, geen pariteit, 1 stopbit, geen break

disable alle interrupts (reset Interrupt enable register)
zet de UART in de 16450 mode
zet de besturing van de modemsignalen uit
zet led2 en led1 aan en led3 en led0 uit

exit Zet alle leds uit.
Verwijder de struct cdev van het device.
Geef het device nummer van de device driver vrij.
Geef de 8 adressen van de UART vrij.

open Zet led1 en led0 aan en led3 en led2 uit.

close Zet led3 en led2 aan en led1 en led0 uit.

read Lees de waarde van de Receiver Buffer van de UART, vertaal die in een string van (vier) enen en nullen en plaats deze in de user buffer.

write Lees de string uit de user buffer, vertaal die in de stuurbyte voor de leds en plaats die byte in de Transmitter Holding Buffer van de UART.

ioctl Geen actie.

Raadpleeg bijlage 2 voor de gegevens van de UART (of download de datasheet van de 16550D voor gedetailleerde informatie).

Raadpleeg verder §10.1.7 voor uitleg omtrent extra benodigde system calls.
Bestudeer natuurlijk ook het dictaat 'Linux device drivers'.

Extra benodigde system calls

```
int check_region (unsigned long startadres,
                  unsigned long aantal);
```

Hier wordt getest of <aantal> I/O-adressen vanaf <startadres> vrij zijn. Dit is het geval als de returnwaarde 0 is. Anders is de returnwaarde negatief.

```
void release_region (unsigned long start, unsigned long len);
```
Deze functie geeft <aantal> I/O-adressen vanaf <startadres> vrij.

```
struct resource *request_region (unsigned long startadres,
                                 unsigned long aantal,
                                 char *drivernaam);
```

Deze functie wijst <aantal> I/O-adressen vanaf <startadres> toe aan de driver met de naam waarnaar de pointer <drivernaam> wijst. Als dit is gelukt, wordt de NULL pointer als returnwaarde teruggegeven.

```
int  inb  (I/O-adres);
void outb (databyte, I/O-adres);
```

De returnwaarde van de functie inb() is de gelezen byte uit de locatie <I/O-adres>.
De functie outb() plaatst de waarde van <databyte> op de locatie <I/O-adres>.

Compileren en laden van de device driver

Als je als root bent ingelogd, kun je 'sudo' in de volgende commando's weglaten. Het compileren van de device driver IOdriver.c gaat als volgt:

```
./runmake IOdriver
```

Zorg ervoor dat alle foutmeldingen en waarschuwingen verdwenen zijn voordat de device driver wordt geladen.

Het laden van de device driver gaat met

```
sudo insmod IOdriver.ko
```

Als het goed is, moeten nu led2 en led1 aan gaan en led3 en led0 uit.

Het achterhalen van major en minor nummer van de device driver gaat met:

```
dmesg
```

Stel dat gevonden wordt dat major nummer = 250 en minor nummer = 0, dan moet de special device file worden aangemaakt met

```
sudo mknod /dev/IOfile c 250 0
sudo chmod 666 dev/IOfile
```

chmod is nodig opdat ook de applicatie kan lezen en schrijven in de device driver via de device file.

De device driver kan worden verwijderd met

```
sudo rmmod IOdriver
```

In dit geval moeten alle leds uit gaan.

De applicatie wordt opgestart met

```
./useIO IOfile
```

Nu moeten led1 en led0 aan gaan en led3 en led2 uit.

Bij het commando **r** wordt de stand van de schakelaars op het scherm afgedrukt als een string van vier karakters (nullen en enen).
Met een stringcommando van vier enen en nullen, bv. 0101, worden de leds aangestuurd.

Met het commando **s** wordt de applicatie beëindigd en moeten led3 en led2 aan en led1 en led0 uit gaan.

Opdracht 11.2: device driver met receiver interrupt

Compileren: met make.

Om onze device driver met interrupts te laten werken moeten de volgende includes worden toegevoegd:

```
#include <linux/interrupt.h>
#include <linux/irqreturn.h>
#include <linux/types.h>
#include <linux/signal.h>
#include <linux/sched.h>
```

Voeg aan de device driver een interrupt handler toe die de volgende acties uitvoert:

Test of de interrupt een receive interrupt van de UART is.
Zo ja:
    Reset de receiver interrupt van de UART.
    Zend signal 12 naar de taak waarvan de pid is ontvangen.
    Return de waarde IRQ_HANDLED.
Zo nee, return de waarde IRQ_NONE.

Voor het zenden van een signal moet je een pointer naar een task_struct declareren:

```
struct task_struct *pTask;
```

Het zenden van een signal gaat dan als volgt:

```
pTask = current;
send_sig (12, pTask, 0);
```

Breid ioctl-functie van de device driver uit met de volgende acties bij een ontvangen commando:

1 test of het benodigde interruptnummer vrij is
  zo nee, geef het vrij en installeer de interrupt handler
  reset de interrupts van de UART
  reset de receiver ready bit
  enable de data ready interrupt

0 disable alle UART-interrupts
  geef het interruptnummer vrij

Bijlage 2: 16550D UART

Registers	**Adres (t.o.v. beginadres van UART)**
Reciever Buffer Register	0 (bij read)
Transmitter Holding Register	0 (bij write)
Interrupt Enable Register	1
Interrupt Identification Register	2 (bij read)
FIFO Control Register	2 (bij write)
Line Control Register	3
Modem Control Register	4
Line Status Register	5
Modem Status Register	6
Scratch Register	7
Divisor Latch (LSB)	0
Divisor Latch (MSB)	1

Soms hebben twee registers hetzelfde adres. Er wordt dan onderscheid gemaakt door te lezen of te schrijven in dat adres. Echter ook de Divisor Latches worden via de adressen 0 en 1 bereikt terwijl die al zijn uitgegeven. Hier wordt de truc toegepast dat er een extra adresbit wordt gebruikt en wel in de vorm van bit 7 van de Line Control Register, de DLAB (Divisor Latch Address Bit). Als deze 1 is, worden de Divisor Latches geadresseerd op de adressen 0 en 1. Is deze 0, dan wordt een van de dataregisters, resp. de Interrupt Enable Register geadresseerd.

De bits 0 en 1 van het Line Control Register selecteert de grootte van de datakarakters:

bit1	bit 0	karakter
0	0	5 bits
0	1	6 bits
1	0	7 bits
1	1	8 bits

bit 2 = 0: 1 stopbit
bit 2 = 1: 2 stopbits (bij 6-, 7- of 8-databits)
bit 3 = 0: geen pariteit
bit 3 = 1: pariteit enabled
bit 4 = 0: oneven pariteit (mits enabled)
bit 4 = 1: even pariteit (mits enabled)
bit 5 = 0: geen vaste pariteit
bit 5 = 1: vaste pariteit = 0 als bit 3 = 1 en bit 4 = 0
vaste pariteit = 1 als bit 3 = 1 en bit 4 = 1
bit 6 = 0: geen break signaal
bit 6 = 1: break signaal (transmit data blijft continu 0, totdat bit 6 = 0)
bit 7 = DLAB (zie hierboven)

Divisor Latch = UART-frequentie / (16 x bitsnelheid)

De UART-frequentie bedraagt 1,8432 MHz.

Bijlage 3: Handleiding tmux

Met het commado **tmux** kan een aantal terminals in windows en panes (onderdeel van een window) geopend worden.

Kennismaking met tmux:

http://vimeo.com/69185909

Zie voor een uitgebreidere gebruikshandleiding:

http://blog.hawkhost.com/2010/06/28/tmux-the-terminal-multiplexer/

Een tmux session heeft een naam en bestaat uit meerdere windows die weer opgedeeld kunnen worden in panes. Elke window heeft een volgnummer en kan een informatieve naam gegeven worden.

Enkele belangrijke commando's:

tmux	Start tmux session met 1 window
tmux new -s <name>	Start tmux session met 1 window met session naam <name>
tmux ls	Toon alle tmux sessions (overzicht session namen)
tmux kill-session	Stop tmux session
tmux attach -t <name>	Start tmux session met naam <name> (indien beschikbaar)
Ctrl-B "	Opdelen in twee panes horizontaal
Ctrl-B %	Opdelen in twee panes verticaal
Ctrl-B pijltje	Ga naar pane in richting pijltje
Ctrl-B c	Creëer nieuw window
Ctrl-B ,	Geef currrent window een (informatieve) naam
Ctrl-B 1..9	Spring naar window 1..9
Ctrl-B o	Ga naar de volgende pane
Ctrl-B x	Verwijder current pane uit window
Ctrl-B z	Maak current pane zo groot als window en herstel vorige view
Ctrl-B d	Detach session, session is weer op te starten met tmux attach -t <naam>
Ctrl-B ?	Toon alle keybindings

Een tmux session is in zeer veel details in te richten. Het is mogelijk tmux session in een bash script te configureren.

Let erop dat je een English (US) keyboard layout gebruikt.

Bijlage 4: QtCreator installeren en gebruik

Installeren QtCreator
Van QtCreator komen regelmatig nieuwe versies ter beschikking. We gaan gebruik maken van versie 5.5.0 (augustus 2015). Op de site van Digia kunnen we een installer die past bij ons operating system Linux downloaden: http://www.qt.io/download/.

Voor het installeren kan Digia vragen naar het wel of niet-commerciële gebruik van de tooling. We moeten keuzes maken die altijd moeten bevestigen dat we Open Source (niet commercieel) gaan ontwikkelen. Bij het installeren wordt gevraagd informatie over **een Digia-account op te geven, dit moeten we met de skip-optie overslaan.**

Werkdirectory OPS maken
Alle opdrachten moeten in een zogenaamde werkdirectory geplaatst worden. Noem deze directory OPS. Deze directory moet voordat we met QtCreator aan de slag gaan bestaan.

Instellen editor
Start QtCreator. Kies in de bovenbalk **Tools** en kies daarin **Options...** onderdeel **Text editor**, **Behavior**, **Tabs and Indentation:** onderdeel **Tabs policy:** Spaces only, Tab size = 3, Indent size = 3. Maak naast Behavior de keuze voor **Display**. Vink de volgende functies aan: Display line numbers, Highlight current line en Highlight matching parenthesis. Druk op de knop Okay om af te ronden.

Instellen inspringen coderegels
QtCreator ondersteunt het consequent inspringen van C-coderegels (indentation). Op #OO staat daarvoor een configuratie file: CodeStyle-v1.0.xml. Zet deze in de werkdirectory INFT1. Kies in de bovenbalk **Tools** en kies daarin **Options...** onderdeel **C++**, Code style, Import de configuratie file CodeStyle-v1.0.xml. Zie voor gebruik de volgende paragraaf.

We kunnen ons beperken tot een klein aantal gebruiksopties. Let erop dat bij het onjuist gebruik en/of instellen van gebruiksopties we niet meer goed QtCreator voor onze doeleinden kunnen gebruiken. QtCreator geeft de mogelijkheid voor een zeer groot aantal platforms applicaties te kunnen ontwikkelen. We hoeven maar met een klein aantal functies van QtCreator te gaan werken. We moeten echter wel de juiste aanpak hanteren om niet te "verdwalen" in de ontwikkelomgeving.

C-project maken
Een C-programma-file moet opgenomen worden in de interne administratie van QtCreator van een C-project. Voor elke opdracht moeten we een project creëren die in een eigen directory staat. In een project staat de informatie voor de compiler hoe deze de executable moet gaan genereren. De volgende stappen doen dit voor ons:

1. Start QtCreator (click op het geïnstalleerde QtCreator icoon op de desktop)
2. Click het **Welcome** icoon aan de linker kant, kies daarna de **Projects** knop en de **New Project** knop.
3. Dan verschijnt de **New Project** window. Kies in de Projects kolom het type:

Non-Qt-project.

4. Kies in de tweede kolom **Plain C project**.
5. Kies rechtsonder de knop **Choose...**
6. Elk project heeft een unieke naam. Kies voor Name: **VBopdracht**. Geef in **Create in:** de naam van het pad naar de directory **OPS**. Deze is met **Browse...** te vinden. Kies rechtsonder de knop **Next**.
7. Kies bij **Kit Selection** de knop **Next**.
8. Kies bij **Project Management** de knop **Finish**.
9. Dan verschijnt het standaard programma 'Hello world' in de editor. Links zien we een kolom met een explorer voor de C-projectadministratie. Daarin staan onder andere de C-programma files. Standaard is dit één file: **main.c**.

C programma compileren en runnen

In QtCreator kunnen we linksonder kiezen voor het bovenste groene pijltje om een nog niet gecompileerd programma te compileren en daarna te runnen (alleen mogelijk als er geen C-taalfouten in zitten). Dit kan ook met <Ctrl><r>. Een derde mogelijkheid is dit via de bovenste menubalk met de keuze **Build** te doen. De eerste twee manieren zijn gewoon wat sneller voor standaard gebruik.

Let erop dat als we een programmatekst veranderd hebben we deze eerst moeten opslaan. Met <Ctrl><s> wordt de programma tekst opgeslagen. We kunnen in een mededeling aanvinken dat QtCreator dat altijd automatisch gaat doen.

Mededelingen voor compile time errors en warnings verschijnen onder de editor het subwindow **4 Compile Output**.

Debuggen

We kunnen een programma stap voor stap doorlopen door een debugger te gebruiken. Uiteraard moet het programma daarvoor wel te compileren zijn. Tijdens het stap voor stap doorlopen kunnen we naar de actuele waarden van variabelen kijken. Kies voor debuggen het tweede groen pijltje linksonder. Met de functietoetsen kunnen we het volgende:

F5	Start or continue debugging
Shift + F5	Exit debugger
F10	Step over
F11	Step into
Shift + F11	Step out
F9	Toggle breakpoint
Ctrl + F6	Run to selected function
Ctrl + F10	Run to line
F12	Reverse direction

Multithreaded code is moeilijk te debuggen!

Inspringen van coderegels

Een consequente en overzichtelijke lay-out vergroot de leesbaarheid van code. Dit is noodzakelijk om comfortabel en efficiënt code te kunnen ontwikkelen. QtCreator heeft de mogelijkheid om minimaal het inspringen consequent te laten uitvoeren. We moeten CodeStyle-v1.0.xml geïnstalleerd hebben (zie vorige paragraaf). Kies voor

<Ctrl><a> en <Ctrl><i> om dit uit te voeren. Met <Ctrl><a> selecteren we de gehele codetekst. Het is ook mogelijk een blok tekst te selecteren en alleen deze met <Ctrl><i> consequent te laten inspringen. Let erop dat de resterende lay-out afspraken met de hand moeten worden uitgevoerd. Denk hierbij bijvoorbeeld aan de consequente plaatsing van de { } voor codeblokken.