

MISH-DOCUMENTATION

Ou3- systemnära programmering



Handledare:

Fredrik Peteri, Joel Sandman, Elias Åström

DEN 20 OKTOBER 2020

ELIAS NIKO
C18eno@cs.umu.se

Innehållsförteckning

Mish-dokumentation	2
Introduktion	2
Problembeskrivning	2
Systembeskrivning	2
Algoritmbeskrivning	6
Anropsdiagram.....	7
Begränsningar	7
Kompilering och körning	7
Testkörningar	8
Resultat	10
Diskussion	10
Bilagor	10

Mish-dokumentation

Ou3- systemnära programmering

Introduktion

Ett Shell är det text-baserade program som körs då man öppnar en terminal. Programmet skriver ut en så kallad "prompt" och väntar på kommandon från användaren. Olika OS har olika Shell. till exempel Linux har "bash" som default Shell och Windows har "Windows Shell". Ett Shell är kommunikationsverktyg mellan användare och operativ systemet. Skillnaden mellan mish och bash beskrivs kortfattat i avsnittet begränsningar.

Problembeskrivning

Uppgiften går ut på att implementera ett minimalt Shell, **mish**. Det ska klara av exekvering av kommandon och program, pipor mellan program och omdirigering av standardinput (`stdin`) och output (`stdout`). Till hjälp finns det filerna `parser.c` och `parser.h` som tillhandahåller en funktion som läser av en textsträng och delar upp den i kommandon. Kommandon separeras med tecknet `|`, en så kallad "pipa".

Systembeskrivning

För att mish exekvera de externa kommandon och pipa mellan programmen så fungerar den på det sättet att först läser den kommandon med hjälp av `fgets()` från `stdin` och använder funktionen `parse()` som finns i `parser.c`. funktionen `parse(line, cmds)` användes så här att den tar två argument `line` och `cmds`. `line` är den rad användaren skriver i `stdin` och `cmds` är kommandon som ska separeras med `|`. `Parse()` returnerar ett tal som är antal kommandon.

- `read_command_line(argc, argv)`: Den här funktionen läser in kommando raden från tangentbord och använder funktionen `parse` för att dela upp den i kommandon. sedan anropar den funktionen `run_cmd(argc, comLine)` för att köra kommandon sparad i `comLine`.

```

mish% echo hello world
hello world
{
  Argv: ["echo", "hello", "world"]
  Argc: 3
  Infile: (null)
  Outfile: (null)
}
mish% cd ..
{
  Argv: ["cd", ".."]
  Argc: 2
  Infile: (null)
  Outfile: (null)
}
mish% cat mish.h | tail -5 > apa
{
  Argv: ["cat", "mish.h"]
  Argc: 2
  Infile: (null)
  Outfile: (null)
}
{
  Argv: ["tail", "-5"]
  Argc: 2
  Infile: (null)
  Outfile: apa
}
cat: mish.h: No such file or directory
mish%

```

Figur 1: skissen visar mishens anropsdiagram

- `run_cd(argc, cmds[])`: Denna funktion börjar med att iterera genom alla kommandon som finns i `cmds`. Varje kommando är antingen extern eller intern. Datatypen kommando har en integer attribut som kallas `intern` vilket används för att visa om ett kommando är extern eller intern. "cd" och "echo" är dem interna kommandon som måste implementeras och alla övriga kommandon måste räknas som externa. Som standard blir alla kommandon extern. Om första kommando är antingen cd eller echo, då anrops funktionerna cd eller echo. Annars är dem externa kommando och måste hanteras i ett barn process. Däremot anrops `ext_commands` för att köra de externa kommandona.
- `extern_command (argc, cmds[])`: Funktionen börjar med att deklarerar två fil deskriptorer och en integer variabel för pid nummer till varje process. Sedan loopar funktionen genom alla kommandon för att köra de.
-

Nedan följer ett exempel på hur exekvering av piporna fungerar i mishen Steg 1: mishen är redo:

Mish (run)
DATA 1
ENV 1
Pid: 1675
0: term
1: term
2: term

Figur 2: ready

Steg2:

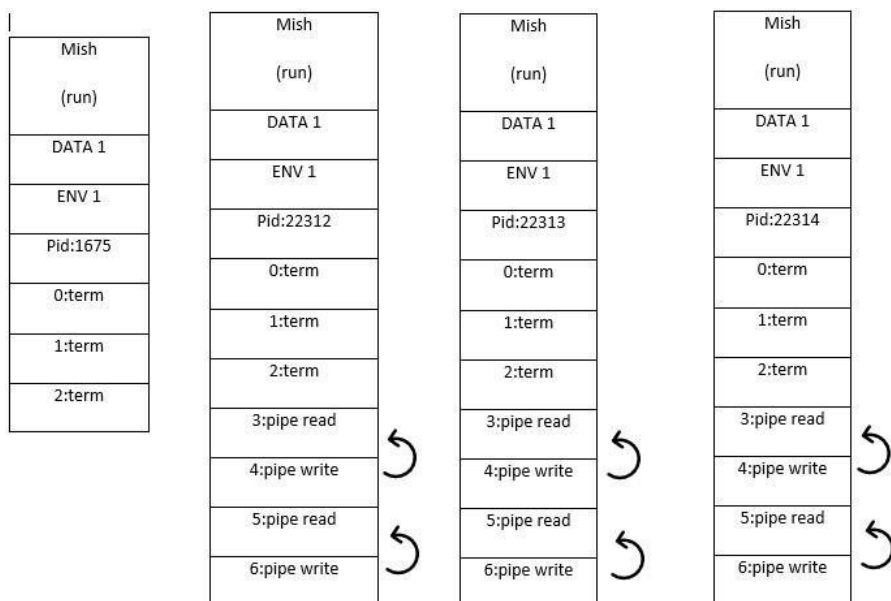
Mishen börjar på att pipa kommandona genom att pipa kommandona och lägga till nya file deskriptorer:

Mish (run)
DATA 1
ENV 1
Pid:1675
0:term
1:term
2:term
3:pipe read
4:pipe write
5:pipe read
6:pipe write

Figur 3: pipe

Steg 3:

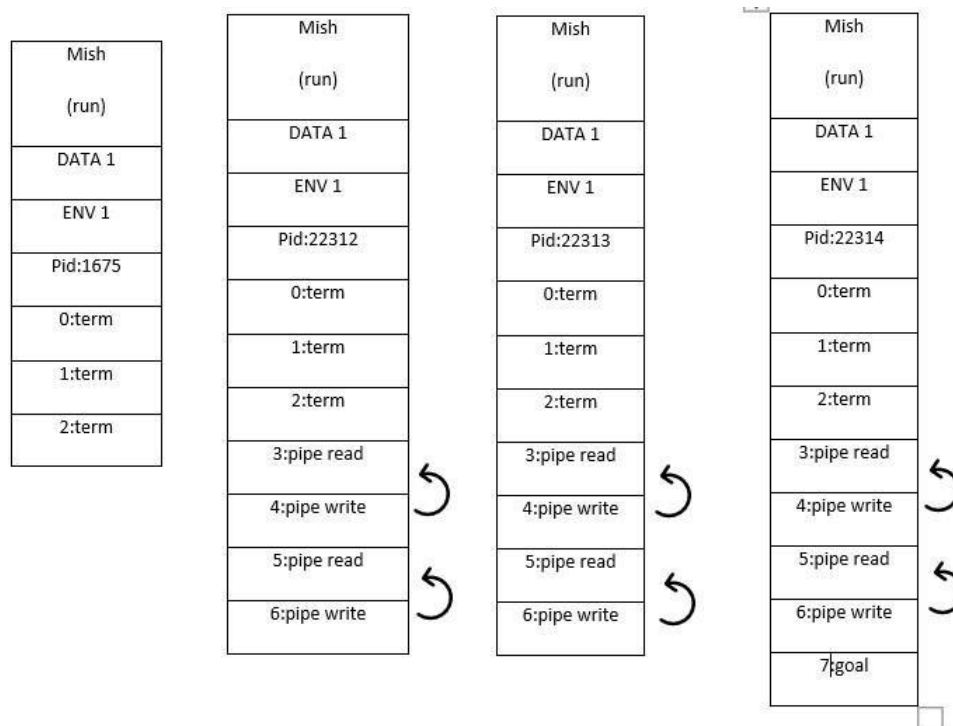
I det här steg så mishen utför för för att ska barn process för att exekvera de kommandona och stänger av fil deskriptorerna. I det här steg mishen väntar på barn processer att utföra sina jobb. OBS! i figuren istället för (run ska vara (wait).



Figur 4:(Fork, close, wait)

Steg 4:

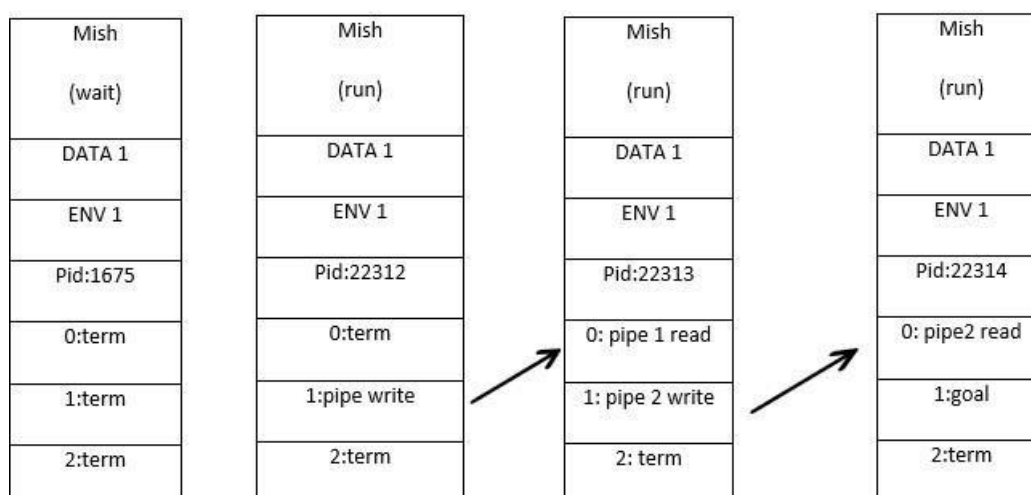
I detta steg så medan föräldrar processen väntar på sina barn så, barn processen med pid nummer 22314 öppnar filen `goal.c`



Figur 5: open

Steg 5:

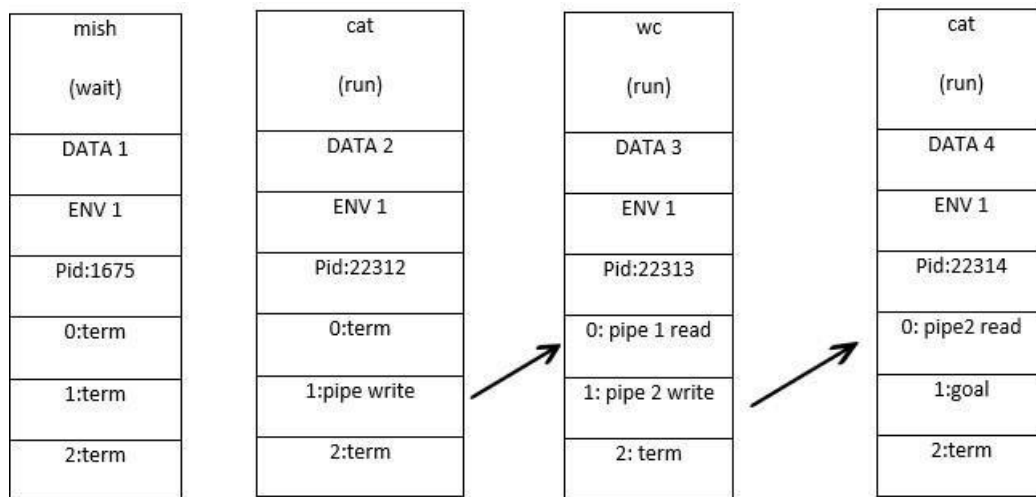
I detta steg så skapas kopia av en fil deskriptor, det använder det minsta deskriptor för den nya deskriptor.



Figur 6: dup, close

Steg 6:

I detta steg så kallas `execvp` funktionen för att använda den barn processen för att utföra de kommandon som ska göras som externa kommandon.



Figur 7: exec

Steg 7:

(exit)

Mish (run)
DATA 1
ENV 1
Pid: 1675
0: term
1: term
2: term

Figur 8: exit

Algoritmbeskrivning

Pipe:

För att separera kommandona och skapa nya file deskriptor så används pipe. Pipe ägnar nya file deskriptor åt de kommandona som existerar

Mish (run)
DATA 1
ENV 1
Pid:1675
0:term
1:term
2:term
3:pipe read
4:pipe write
5:pipe read
6:pipe write

Figur 9: pipe

Fork:

Fork skapar barnprocesser för existerade kommandona. Barnprocessernas skillnad med föräldrar processen är PID numret. Om skapandet av fork var lyckad så returneras PID för barnprocessen i förälderns process och barnet returnerar 0. Om skapandet av for misslyckades så returneras ett negativt nummer (-1) i förälderns process.

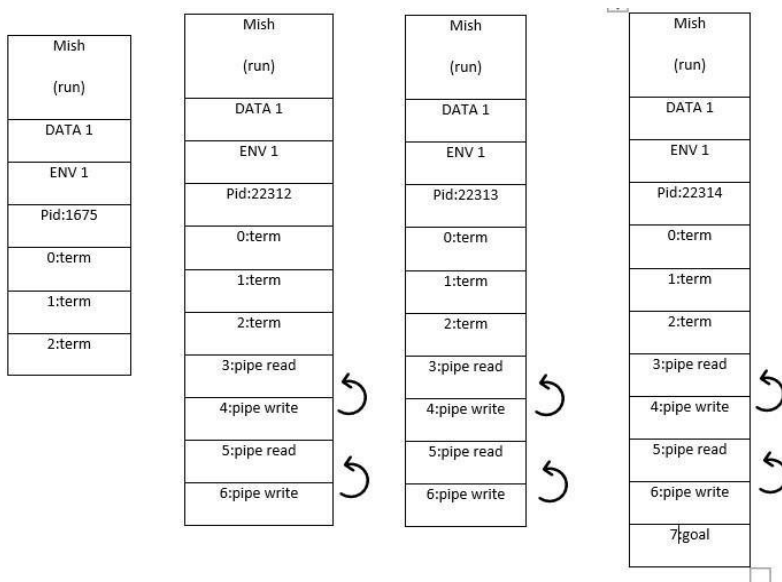
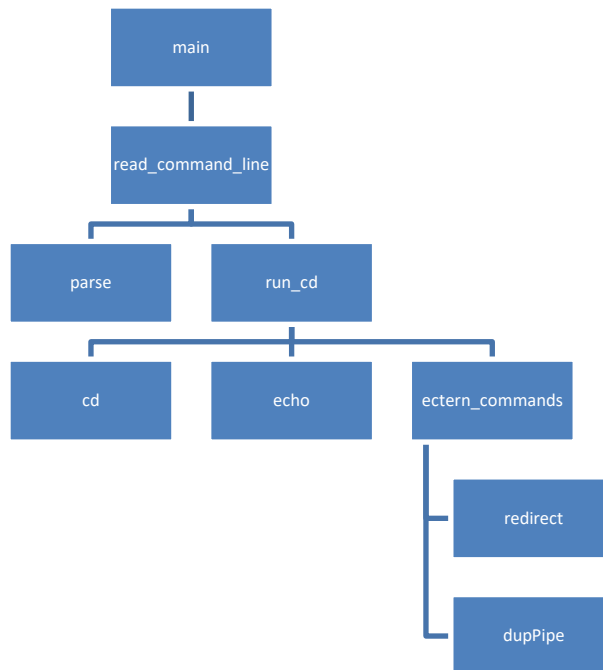


Figure 10: fork

Anropsdiagram



Figur 11: anropsdiagram

Begränsningar

Mish kan inte klara kombination av extern och interna kommandorad som har pipor i mellan, detta är en begränsning för mish. Mish kan inte omredigera standardinput och standard output ur interna kommandon.

Kompilering och körning

För kompilera filerna så behöver man filerna i en directory. Sedan med hjälp av terminalen kan man kompilera filerna genom att använda **make**. Nedan följer ett exempel på hur **mish** kompileras:

```

lollipop@lollipop: /mnt/c/users/admin/desktop/dropbox/uni/snp/ou3/mish
lollipop@lollipop: /mnt/c/users/admin/desktop/dropbox/uni/snp/ou3/mish$ make
gcc -g -Wall -Wextra -Werror -Wmissing-declarations -Wmissing-prototypes -Werror -Wimplicit-function-declaration -Wreturn-type -Wparentheses -Wunused -Wold-style-definition -Wundef -Wshadow -Wstrict-prototypes -Wswitch-default -Wunreachable-code -c -o mish.o mish.c
gcc -g -Wall -Wextra -Werror -Wmissing-declarations -Wmissing-prototypes -Werror -Wimplicit-function-declaration -Wreturn-type -Wparentheses -Wunused -Wold-style-definition -Wundef -Wshadow -Wstrict-prototypes -Wswitch-default -Wunreachable-code -o mish mish.o parser.o execute.o sighant.o
lollipop@lollipop: /mnt/c/users/admin/desktop/dropbox/uni/snp/ou3/mish$ ./mish
mish%
  
```

Figur 8: kompilering av mish

Testkörningar

Test1: Ett exempel av kommandon `ls` och `echo` via `mish`:

note! För att avsluta `mish` så användaren ska trycka `ctrl + d` eller via en annan terminal via `killall mish`

```
lollipop@lollipop: /mnt/c/users/admin/desktop/dropbox/uni/snp/ou3/mish
mish% ls
Makefile          execute.h  fork.c  m      mish.h  parser.h  sighant.c  test
c18eno_ou3_rapport.pdf execute.o  goal    mish  mish.o  parser.o  sighant.h  test.c
execute.c          foo.txt   goal.c  mish.c parser.c parser_ex.c sighant.o
mish% echo mish
mish
mish% echo Hello World
Hello World
mish% lollipop@lollipop: /mnt/c/users/admin/desktop/dropbox/uni/snp/ou3/mish$
```

Test2: kolla om bara en `mish` process vara aktiv

Terminal 1:

```
lollipop@lollipop: /mnt/c/users/admin/desktop/dropbox/uni/snp/ou3/mish
lollipop@lollipop: /mnt/c/users/admin/desktop/dropbox/uni/snp/ou3/mish$ ./mish
mish%
```

Terminal 2:

```
lollipop@lollipop: /mnt/c/users/admin/desktop/dropbox/uni/snp/ou3/mish
lollipop@lollipop: /mnt/c/users/admin/desktop/dropbox/uni/snp/ou3/mish$ ps -u
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
lollipop    7  0.0  0.0  16884  3272 tty1      S   15:01   0:00 -bash
lollipop   84  0.0  0.0  17016  3476 tty2      S   15:24   0:00 -bash
lollipop  154  0.0  0.0  10556   588 tty1      S   17:06   0:00 ./mish
lollipop  162  0.0  0.0  17380  1912 tty2      R   17:10   0:00 ps -u
lollipop@lollipop: /mnt/c/users/admin/desktop/dropbox/uni/snp/ou3/mish$
```

Test 3: `mish` ska kunna hantera en commando som inte finns:

Terminal 1:

```
lollipop@lollipop: /mnt/c/users/admin/desktop/dropbox/uni/snp/ou3/mish
lollipop@lollipop:/mnt/c/users/admin/desktop/dropbox/uni/snp/ou3/mish$ ./mish
mish% nosuchcommand file.txt
nosuchcommand: No such file or directory
mish%
```

Terminal 2:

```
lollipop@lollipop: /mnt/c/users/admin/desktop/dropbox/uni/snp/ou3/mish
lollipop@lollipop:/mnt/c/users/admin/desktop/dropbox/uni/snp/ou3/mish$ ps -u
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
lollipop    7  0.0  0.0  16884  3272 tty1      S   15:01   0:00 -bash
lollipop   84  0.0  0.0  17016  3476 tty2      S   15:24   0:00 -bash
lollipop  154  0.0  0.0  10556   608 tty1      S   17:06   0:00 ./mish
lollipop  166  0.0  0.0  17380  1916 tty2      R   17:15   0:00 ps -u
lollipop@lollipop:/mnt/c/users/admin/desktop/dropbox/uni/snp/ou3/mish$
```

Test 3: inga zombie-process kvar:

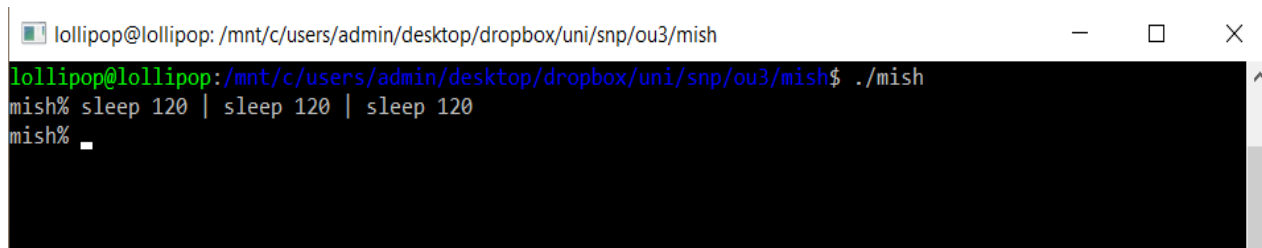
Terminal 1: innan kill -INT PID

```
lollipop@lollipop: /mnt/c/users/admin/desktop/dropbox/uni/snp/ou3/mish
lollipop@lollipop:/mnt/c/users/admin/desktop/dropbox/uni/snp/ou3/mish$ ./mish
mish% sleep 120 | sleep 120 | sleep 120
```

Terminal 2:

```
lollipop@lollipop: /mnt/c/users/admin/desktop/dropbox/uni/snp/ou3/mish
lollipop@lollipop:/mnt/c/users/admin/desktop/dropbox/uni/snp/ou3/mish$ ps -u
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
lollipop    7  0.0  0.0  16884  3272 tty1      S   15:01   0:00 -bash
lollipop   84  0.0  0.0  17016  3476 tty2      S   15:24   0:00 -bash
lollipop  154  0.0  0.0  10556   608 tty1      S   17:06   0:00 ./mish
lollipop  168  0.0  0.0  13956   812 tty1      S   17:16   0:00 sleep 120
lollipop  169  0.0  0.0  13956   816 tty1      S   17:16   0:00 sleep 120
lollipop  170  0.0  0.0  13956   812 tty1      S   17:16   0:00 sleep 120
lollipop  172  0.0  0.0  17380  1916 tty2      R   17:16   0:00 ps -u
lollipop@lollipop:/mnt/c/users/admin/desktop/dropbox/uni/snp/ou3/mish$ kill -INT 154
lollipop@lollipop:/mnt/c/users/admin/desktop/dropbox/uni/snp/ou3/mish$ ps -u
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
lollipop    7  0.0  0.0  16884  3272 tty1      S   15:01   0:00 -bash
lollipop   84  0.0  0.0  17016  3488 tty2      S   15:24   0:00 -bash
lollipop  154  0.0  0.0  10556   608 tty1      S   17:06   0:00 ./mish
lollipop  173  0.0  0.0  17380  1916 tty2      R   17:16   0:00 ps -u
lollipop@lollipop:/mnt/c/users/admin/desktop/dropbox/uni/snp/ou3/mish$
```

Terminal 1: after kill -INT 154



```
lolliop@lolliop: /mnt/c/users/admin/desktop/dropbox/uni/snp/ou3/mish
lolliop@lolliop:/mnt/c/users/admin/desktop/dropbox/uni/snp/ou3/mish$ ./mish
mish% sleep 120 | sleep 120 | sleep 120
mish% 
```

Resultat

Resultatet av lösningen blev till slut ett minimalt Shell som kan exekvera båda interna och externa kommandon. De två interna kommandon som **mish** kan exekvera är `cd` och `echo`. `cd` omdirigerar till olika kataloger och `echo` skriver ut sitt argument till `stdout`. Dessutom körs de i huvud processen utan att `mish` behöver skapa en ny process. Däremot för de externa kommandona behöver `mish` skapa en så kallad Child-process och exekvera externa kommandon där istället.

Diskussion

Mish är ett simpelt Shell som inte kan exekvera avancerade kommandon. Däremot kan det exekvera `cd` som byter katalog och `echo` som skriver sitt argument på `stdout`. I denna laboration har jag lärt oss om hur `execvp()` och andra `exec` funktionerna fungerar, hur `fork` görs och hur interna kommandon och externa kommandon gör jobbet. Det som var svårt för mig i denna laboration var att hur ska jag hantera de externa kommandon och signal hantering. Hur ska man göra att när man gör `kill` för en föräldrar process så ska alla child dödas.

Bilagor

<https://www.cambro.umu.se/access/content/group/57250HT19-1/Laboration%203/mish-specification.pdf>

<http://www8.cs.umu.se/kurser//5DV088/HT08/tentor/Losn-071106.pdf> <https://www.geeksforgeeks.org/exec-family-of-functions-in-c/>

<https://www.cambro.umu.se/access/content/group/57250HT19-1/Exempel%20fr%C3%A5n%20F%C3%B6rel%C3%A4sningar/F08/mysignal.c>