

# Systemnära Programmering (5DV088)

## Obligatorisk Uppgift 3

Marko Nygård, oi12mnd

1 november 2017

Kursansvarig: Mikael Rännar

Handledare: Jakob Lindqvist

Didrik Lindqvist

William Viktorsson

## Innehåll

<b>1</b>	<b>Problembeskrivning</b>	<b>1</b>
<b>2</b>	<b>Kompilering och Körning</b>	<b>1</b>
<b>3</b>	<b>Systembeskrivning</b>	<b>2</b>
3.1	Exekveringsflöde . . . . .	2
3.2	Processkommunikation . . . . .	4
<b>4</b>	<b>Exempelkörningar</b>	<b>6</b>

## 1 Problembeskrivning

Denna rapport beskriver ett arbete med att implementera ett simpelt shell-program kallat *mish.c*. Shallet ska kunna exekvera de interna kommandona *echo* och *cd*. *echo* skriver ut alla sina argument till stdout (terminalen), *cd* används för att byta den nuvarande katalogen. Shallet ska även kunna exekvera externa kommandon och program, omdirigera in- och output från dessa samt möjliggöra kommunikation mellan dessa via pipes.

Slutligen har även en signalhanterare som kan avsluta exekverande program implementerats.

## 2 Kompilering och Körning

För att kunna köra programmet behövs filerna *mish.c*, *mish.h*, *parser.c*, *parser.h*, *sighant.c*, *sighant.h*, *execute.c*, *execute.h* samt *Makefile*. Kompilering av programmet utförs genom att i terminalen navigera till katalogen där filerna är sparade och skriva *make*. Shallet startas genom att skriva *./mish* (inga argument antas). När programmet startat skrivs prompten *%mish* ut, vilket innebär att programmet kan ta emot input i form av ett eller flera kommandon med eventuella inparametrar. Om flera kommandon ska exekveras ska dessa separeras av pipptecken *|*. Om användaren vill omdirigera outputen av ett kommando ska tecknet *>* finnas mellan kommandot och namnet på output-filen. På motsvarande sätt ska tecknet *<* anges om omdirigering av inputen önskas.

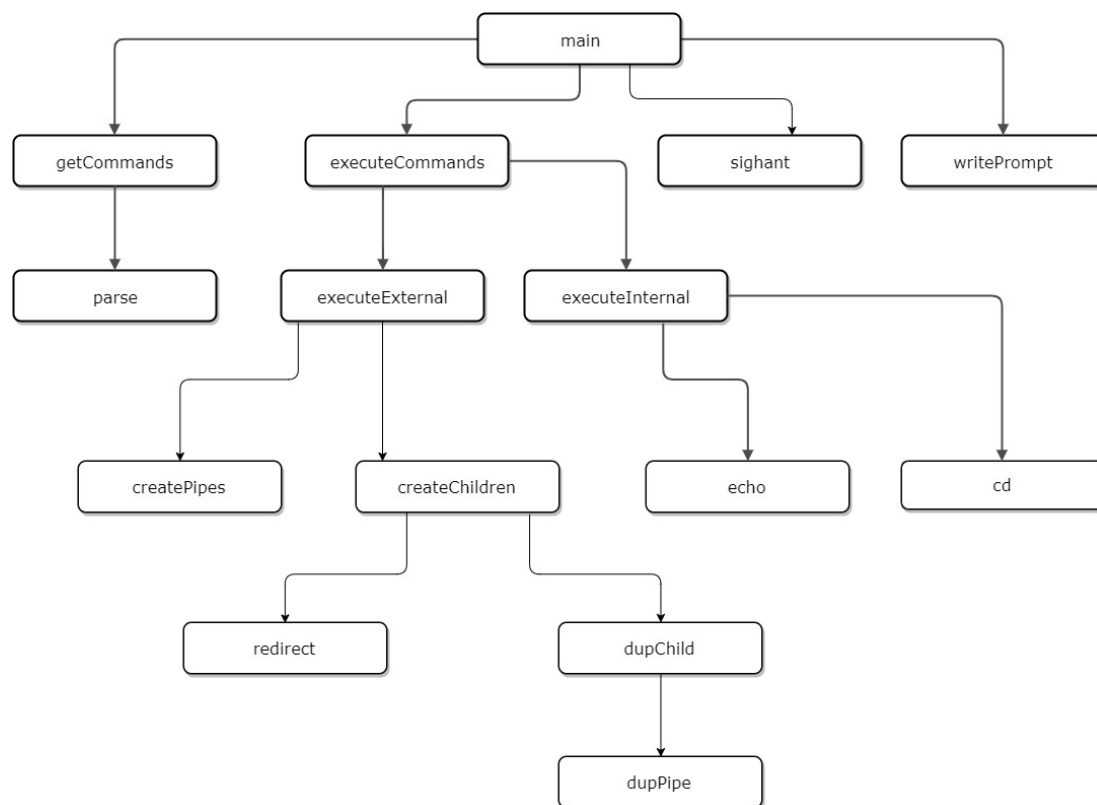
Det interna kommandot *echo [ARGS]* skriver ut alla dess argument *ARGS* på en ny rad i terminalen. Kommandot *cd newDirectory* byter den nuvarande katalogen till dess argument *newDirectory* om den existerar. Anges fler än ett argument ignoreras dessa.

Om ctrl+C trycks under exekvering avbryts samtliga processer som körs och en ny prompt skrivs ut. För att avsluta shallet, tryck ctrl+D när inga processer körs.

## 3 Systembeskrivning

### 3.1 Exekveringsflöde

I figur 1 finns ett anropsdiagram som beskriver hur programmets funktioner beror på varandra på ett överskådligt sätt.



**Figur 1** – Anropsdiagram för funktionerna som används av *mish*.

*main*-funktionen börjar med att skapa en signalhanterare *sighant* som har i uppgift att stänga barnprocesser när ctrl+C trycks. Sedan körs huvudalgoritmen, vilken grovt sett ser ut som i Algoritm 1.

---

**Algorithm 1** Huvudalgoritm

---

```
1: while Signal ctrl+D not received do
2:   Write a prompt
3:   Get user command(s)
4:   if Command is internal then
5:     Execute command
6:   else if Command is external then
7:      $n$  = number of commands
8:     Create  $n - 1$  pipes
9:     Create  $n$  child processes
10:    Parent closes pipes, waits for children
11:    for every child do
12:      if Infile is given then
13:        Redirect input
14:      end if
15:      if Outfile is given then
16:        Redirect output
17:      end if
18:      if  $n > 1$  then
19:        Draw pipes between child processes
20:      end if
21:      Execute commands
22:    end for
23:  end if
24: end while
```

---

Mellan varje anrop skrivs en ny prompt ut med *writePrompt* för att visa att ett nytt kommando kan ges. Användarens kommando( $n$ ) registreras med *getCommands* som parsar kommandoraden med hjälp av *parse* om kommandoraden inte är *NULL*. *parse* delar upp kommandoraden i enskilda kommandon och sparar ner dem i en vektor med structar som innehåller information om kommandot (kommando-namn, antal argument osv).

Det nästa som händer är att *executeCommands* anropas. Denna funktion kontrollerar först om det är ett internt kommando (*cd* eller *echo*) eller ett externt kommando (med hjälp av funktionen *isInternal*, inte utmarkerad i figur 1). I det första fallet anropas *executeInternal* som utför det angivna interna kommandot. I det andra fallet anropas *executeExternal*, vilken skapar  $n - 1$  pipor med *createPipes* och  $n$  barnprocesser med *createChildren*, där  $n$  är antalet kommandon. Efter att barnen skapas stänger föräldern alla sina pipor och väntar på att barnen ska exekvera.

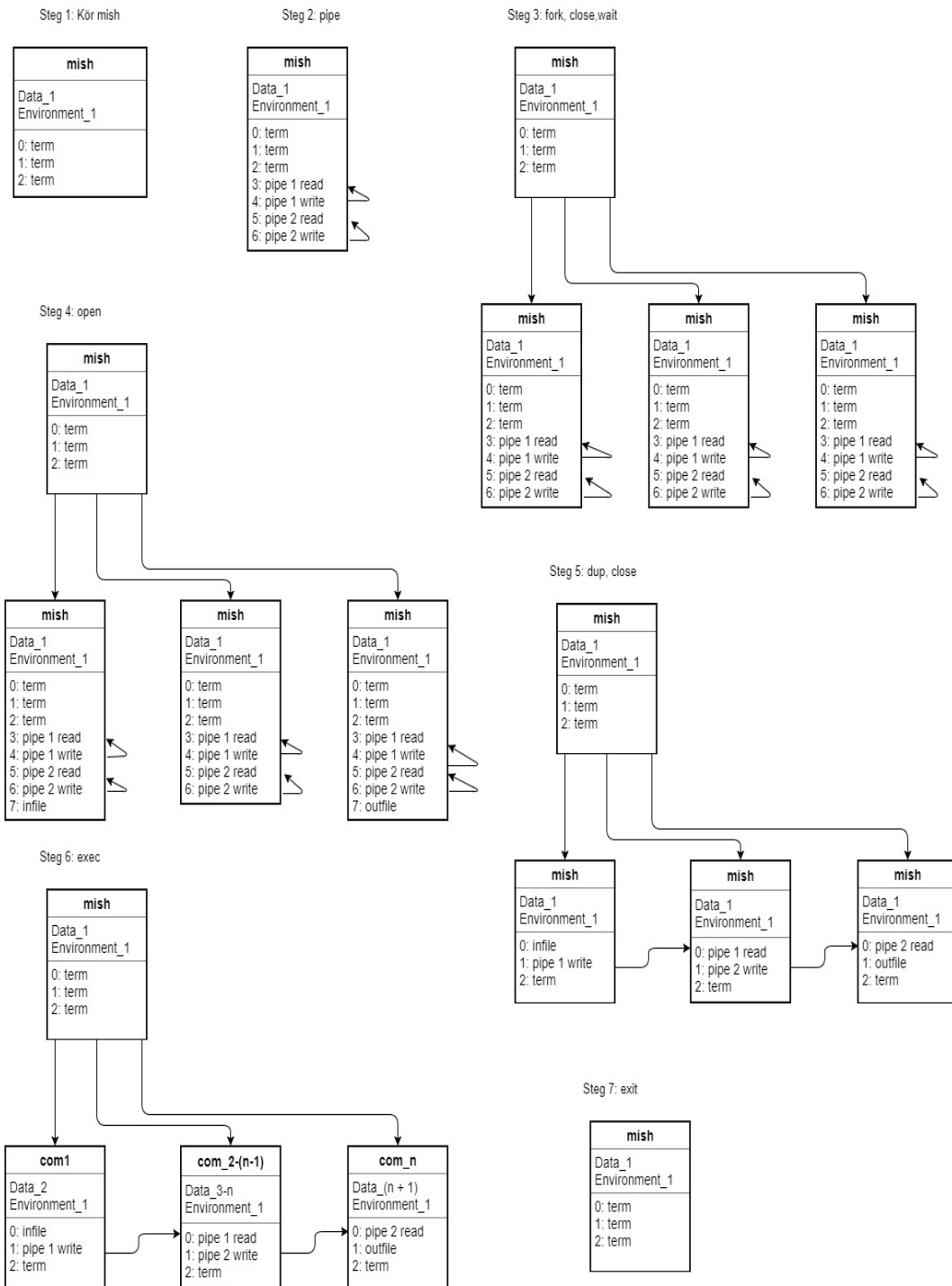
Barnen börjar med att kontrollera om omdirigering av I/O behöver utföras. Om så är fallet, görs detta med funktionen *redirect*. Efter det hanteras kommunikationen mellan barnen med hjälp av *dupChild*, som beroende på barnnummer använder *dupPipe* för att dup-a och stänga relevanta fildeskriptorer. Till sist exekverar barnprocesserna sitt

respektive kommando.

Hanteringen av pipor, barnprocesser och kommunikationen mellan dem är mer utförligt beskriven i stycke 3.2.

### **3.2 Processkommunikation**

I figur 2 finns ett diagram som översiktligt visar hur barnprocesser skapas och kommunicerar.



**Figur 2** – Processdiagram för skapandet och kommunikation mellan barnprocesser skapade av *mish*.

Låt  $n$  vara antalet kommandon som användaren angivit. Föräldern börjar med att skapa  $n - 1$  pipor.  $n$  barnprocesser skapas med *fork()*.

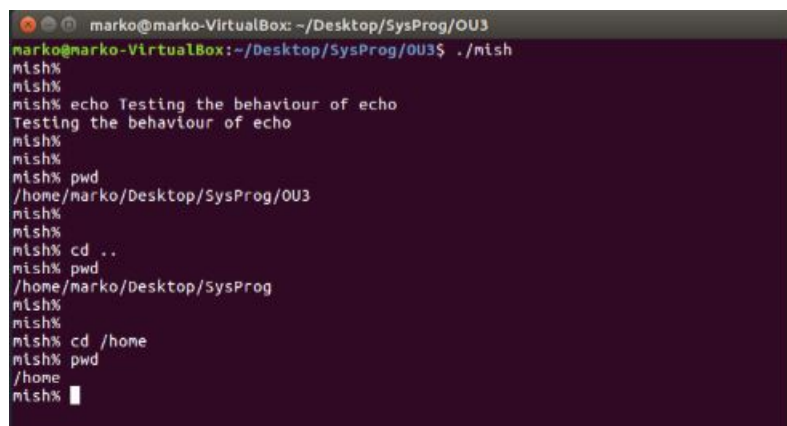
När föräldraprocessen skapat alla barn kan den stänga alla pipor den inte använder (dvs. alla utom de som hör till terminalen). Till sist väntar den på att alla barnen exekverat klart.

Barnprocesserna öppnar in/output-fil om den/de är angiven/angivna. Nästa steg är att låta barnprocesserna kommunicera med varandra via de pipor som skapades innan.

Det första barnet behöver endast dup-a sin skriv-ända, och på motsvarande sätt behöver sista barnet endast dup-a sin läs-ända. Om  $n > 2$  behöver de övriga barnen dup-a både skriv- och läs-ändan. När detta gjorts kan barnen stänga de pipor som inte används och exekvera sina kommandon.

## 4 Exempelkörningar

Flera tester utfördes för att kontrollera programmets korrekthet. Först testades det interna kommandot *echo* genom att ge den ett antal argument och kontrollera att de skrivs ut på en ny rad i terminalen. Efter det testades det andra interna kommandot *cd* för att byta current working directory, samt det externa kommandot *pwd* för att skriva ut det (mellan anropen skrivs några blankrader ut för att göra utskrifterna tydligare). Resultatet kan ses i figur 3.



```
marko@marko-VirtualBox: ~/Desktop/SysProg/OU3
marko@marko-VirtualBox:~/Desktop/SysProg/OU3$ ./mish
mish%
mish%
mish% echo Testing the behaviour of echo
Testing the behaviour of echo
mish%
mish%
mish% pwd
/home/marko/Desktop/SysProg/OU3
mish%
mish%
mish% cd ..
mish% pwd
/home/marko/Desktop/SysProg
mish%
mish%
mish% cd /home
mish% pwd
/home
mish%
```

**Figur 3** – *mish* körs och exekverar sina interna kommandon, samt kör det externa kommandot *pwd*.

Enligt figuren ser det ut som att båda funktionerna fungerar korrekt, dessutom fungerar det att anropa ett enskilt externt kommando.

Nästa test avgör om omdirigeringen av I/O fungerar. Funktionen *ls* fick skicka sin output till en testfil, *cat* tog den som input och skrev ut innehållet i terminalen. Resultatet kan ses i figur 4.



```
marko@marko-VirtualBox: ~/Desktop/SysProg/OU3
mish% cd marko/Desktop/SysProg/OU3
mish%
mish%
mish% ls > testfile.txt
mish%
mish%
mish% cat testfile.txt
execute.c
execute.h
execute.o
Makefile
mish
mish.c
mish.h
mish.o
mishold.c
parser.c
parser.h
parser.o
sighant.c
sighant.h
sighant.o
Spec.pdf
SysProgOU3.pdf
testfile.txt
mish%
```

**Figur 4** – Test av omdirigering av I/O i *mish*.

Enligt figuren verkar omdirigeringen fungera.

Sedan testades att processer kan kommunicera med varandra. *ls* fick skicka sin output till *cat* som skickade sin output vidare till *wc* med flagga *-l* som skriver ut radantalet i sin input. Vidare testades så att omdirigering av I/O fungerar i kombination med processkommunikation. Båda resultaten finns i figur 5.

```
marko@marko-VirtualBox: ~/Desktop/SysProg/OU3
mish% ls|cat|wc -l
19
mish% cat < testfile.txt|wc > wc.txt
mish% cat wc.txt
18      18      169
mish%
```

**Figur 5** – Test av kommunikation mellan processer i *mish*.

Mellan figur 4 och 5 skapades en till textfil i mappen, så radantalet är ett större än det som *cat* skrev ut i figur 4. Vidare kan ses att processer kan kommunicera samtidigt som omdirigering av I/O sker.

Nästa test kontrollerar att signalhanteraren fungerar som den ska. Tre *sleep*-kommandon exekverades, varpå *ctrl+C* trycktes. Detta bör leda till att en ny prompt skrivs ut, vilket kan ses i figur 6.

```
marko@marko-VirtualBox: ~/Desktop/SysProg/OU3
mish% sleep 60|sleep 60|sleep 60
^Cmish% ps -u
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
marko    2237  0.0  0.2 31364  5960 pts/17   Ss   09:56   0:00 bash
marko    3329  0.0  0.0  4372  1432 pts/17   S+   10:19   0:00 ./mish
marko    3402  0.0  0.1 45960  3664 pts/17   R+   10:24   0:00 ps -u
mish%
```

Figur 6 – Test av signalhanteraren i *mish*.

För att illustrera att alla barnprocesser stoppats, anropades *ps*. Som kan ses är det endast *bash*, *mish* och *ps* själv som är igång.

Det sista testet tittar på vad som händer när *mish* kör ytterligare instanser av *mish*. I den sista anropas *echo* och *ps* för att se att inget märkligt händer. Sedan stängs alla processer utom den första, och *ps* anropas igen. Sedan stängs *mish* och kontrollen återgår till terminalen. Händelseförloppet kan ses i figur 7.

```
marko@marko-VirtualBox: ~/Desktop/SysProg/OU3
mish% ./mish
mish% ./mish
mish% ./mish
mish%
mish% echo Running mish inside mish inside mish inside mish??
Running mish inside mish inside mish inside mish??
mish%
mish% ps -u
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
marko    2237  0.0  0.2 31364  5960 pts/17   Ss   09:56   0:00 bash
marko    3329  0.0  0.0  4372  1432 pts/17   S+   10:19   0:00 ./mish
marko    3416  0.0  0.0  4372   668 pts/17   S+   10:24   0:00 ./mish
marko    3417  0.0  0.0  4372   748 pts/17   S+   10:24   0:00 ./mish
marko    3418  0.0  0.0  4372   800 pts/17   S+   10:24   0:00 ./mish
marko    3424  0.0  0.1 45960  3612 pts/17   R+   10:25   0:00 ps -u
mish%
mish%
mish%
mish% ps -u
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
marko    2237  0.0  0.2 31364  5960 pts/17   Ss   09:56   0:00 bash
marko    3329  0.0  0.0  4372  1432 pts/17   S+   10:19   0:00 ./mish
marko    3425  0.0  0.1 45960  3628 pts/17   R+   10:26   0:00 ps -u
mish%
marko@marko-VirtualBox:~/Desktop/SysProg/OU3$
```

Figur 7 – *mish* kör flera instanser av *mish*.