UMEÅ UNIVERSITET
Institutionen för Datavetenskap
Rapport obligatorisk uppgift

Systemnära Programmering (C) 7.5 p
5DV088

Obligatorisk uppgift nr 3

Namn	Jonas Sjödin
E-post	josj0105@student.umu.se
Datum	6 oktober 2017
Kursansvarig	Mikael Rännar
Handledare	Didrik Lindqvist, Jakob Lindqvist, William Viktorsson

UMEÅ UNIVERSITET
Institutionen för Datavetenskap
Rapport obligatorisk uppgift

Innehåll

UMEÅ UNIVERSITET
Institutionen för Datavetenskap
Rapport obligatorisk uppgift

1 Inledning

Hur fungerar ett skal egentligen? I denna rapport kommer det att diskuteras hur jag skrivit ett enkelt unixshell i c vid namn 'mish'. unix-shellet har de inbyggda kommandona `cd` och `echo` och kan exekvera alla andra kommandon som är installerade på aktiv maskin. Skalet tar userinput som den delar upp i olika strängar och sedan exekverar de angivna kommandona med sina parametrar. Skalet stöder pipes '|' och omdirigering av input '<' och output '>'.

2 Syfte

Syftet med denna uppgift är att:

- Lära sig om fildeskriptorer och hur de appliceras i c med hjälp av funktioner som pipe() och dup().
- Lära sig om processer och hur ny processer exekveras i ett program i c med hjälp av funktioner som fork(), wait(), execvp.
- Lära sig om signaler och hur dessa appliceras i c.
- Lära sig hur ett shell fungerar i unixmiljö.

3 Interna kommandon

I denna uppgift finns två interna kommandon implementerade, `cd()` och `echo()`. De är enkla implementationer av standardfunktionerna med samma namn.

3.1 cd

Denna funktion byter working directory efter given input. För att gå till ett nytt directory skrivs tillexempel: `$ cd /path/to/directory)` för att gå till det directoryt. Då den använder sig av `parse()` för att parsa sin input kan den endast gå till directories som inte innehåller '|', '<' eller '>' då dessa kommer att klippa upp och dela strängen.

3.2 echo

Denna funktion skriver ut den inlästa userinputen formaterat, d.v.s. att tillexempel strängen "hello worldskrivs ut som "hello world". Då denna funktion också använder sig av `parse()` kommer den inte heller att acceptera tecknena '|', '<' och '>' då dessa kommer att klippa upp och dela strängen.

4 Systembeskrivning

4.1 Gränsyta

I denna del av rapporten kommer de olika filernas användningsområde att beskrivas samt vad de utför för kommandon för att ge en större inblick i hur koden fungerar.

4.1.1 mish.c & mish.h

Denna fil innehåller själva mish-skalet och `main`-funktionen. Funktionen blockerar i sitt standardutförande (när den inte kör kommandon) `SIGINT`. Den skriver först ut `mish%` till `stderr` och väntar på user-input via `stdin`. Sedan parsar funktionen (se 4.3 parser.c & parser.h) den inlästa user-inputen med hjälp av parser.c:s funktion `parse()`. Denna funktion delar upp inputen i olika `char`-arrayer samt tar ut antalet kommandon mm. Efter detta kollar funktionen om den inlästa inputen är ett internt kommando. Om `char`-arrayen är ett internt kommando utför den det kommandot, annars fortsätter funktionen.

Nu duplicerar `main`-funktionen `stdin` då den sedan kan komma att stängas vid omdirigering av output. Sedan skapar funktionen så många pipes i fildeskriptorer som behövs. Sedan forkas funktionen. Nu är inte `SIGINT` längre blockerad. Om den signalen skickas kommer den nu att terminera alla barnprocesser. Nu kommer en `for`-loop som körs lika många gånger som antaler kommandon. I denna loop kör barnet funktionen `child_action()`. Den kollar först om input ska omdirigeras (Om det är första gången funktionen körs). Sedan kollar den om det är sista gången som loopen körs. Om det är inte är det omdirigerar den output till nästa kommando. Om det är sista gången loopen körs kollar den om den ska omdirigera `stdout` till en fil och gör det isåfall. Sist exekverar barnet kommandot. Om det inte är sista gången loopen körs dirigerar föräldern nu input till pipens läsände, den pipe som barnet skriver sin output till.

Sedan ignoreras `SIGINT` igen och koden väntar med hjälp av funktionen `wait()` på att alla program ska köras klart. Sedan återställs stdin till plats 0 och programmet fortsätter att köras tills användaren väljer att terminera programmet med kommandot 'exit'.

4.1.2 sighant.c & sighant.h

sighant.c innehåller signalhanteringen av `SIGINT` som ska användas när ett kommando exekveras. Den stänger då ner alla barnprocesser sparade i mish.h:s globala pid-array.

4.1.3 parser.c & parser.h

Denna funktion var given för denna uppgift. Den går igenom den inlästa strängen av användaren och sparar den i en struct. Den sparar där alla inlästa strängar, antalet pipor, var input ska omdirigeras och var output ska omdirigeras.

4.1.4 execute.c & execute.h

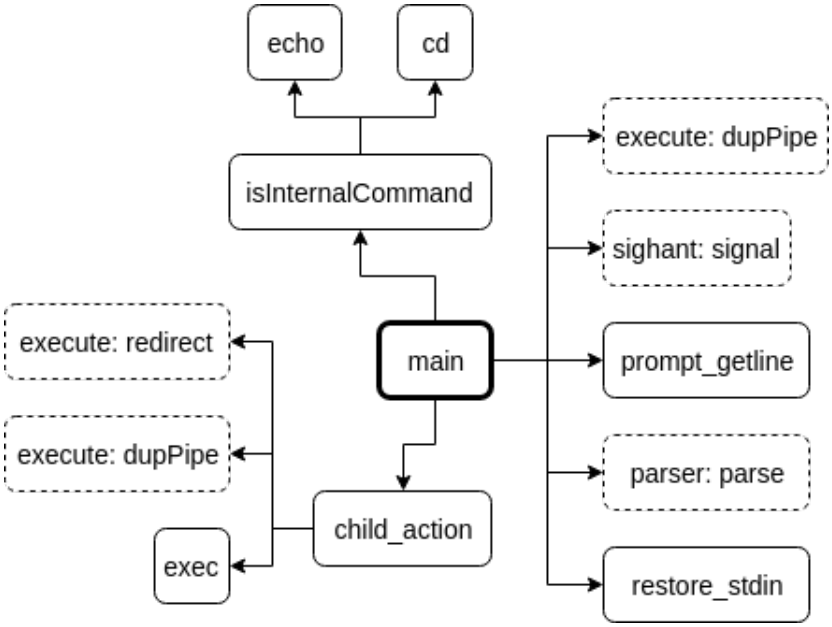
Till denna funktion var header-filen given. Den innehåller en funktion `dupPipe()`som duplicerar en pipe till en file descriptor och stänger sedan båda pipe-änderna. Den innehåller även funktionen `redirect()` som omdirigerar en file descriptor till en textfil. Om output ska omdirigeras måste filen inte finnas. Om input ska omdirigeras måste filen finnas.

4.2 Makefile

Denna makefil kompilerar koden med gcc och flaggorna:

```
-std=c11 -Wall -pedantic -ansi -std=gnu11 -Wall -Wextra -Werror -Wmissing-declarations
-Wmissing-prototypes -Werror-implicit-function-declaration -Wreturn-type -Wparentheses
-Wunused -Wold-style-definition -Wundef -Wshadow -Wstrict-prototypes -Wswitch-default
-Wunreachable-code -g
```

4.3 Anropsdiagram

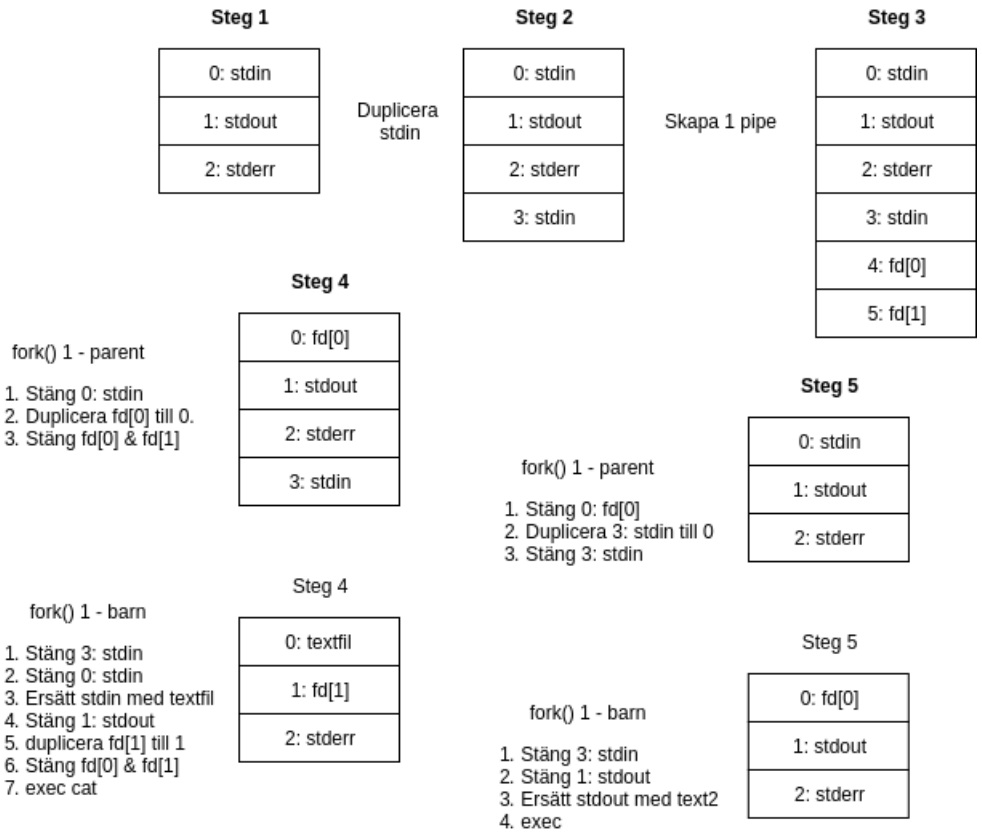


Figur 1: anropsdiagram

4.4 Processfigur

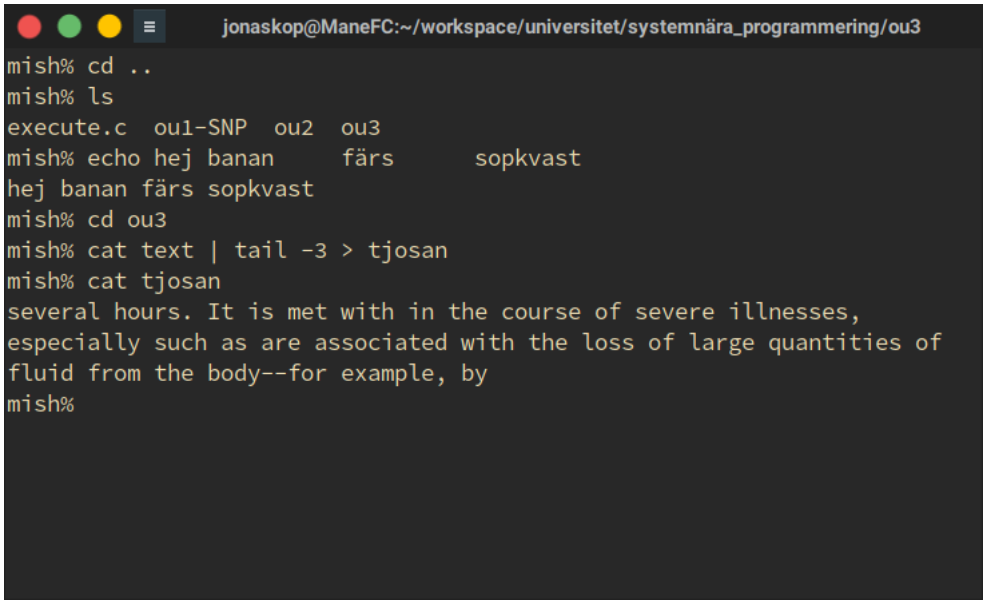
Detta är en figur hur processen och kommunikationen ser ut i exempelkommandot

```
$ cat < textfil | tail -2 > text2
```



Figur 2: Bild över hur processerna kommunicerar

5 Testkörning



Figur 3: Testkörning av mish

6 Förbättringar

En förbättring av programmet hade varit accepterande av piltangenterna. Tillexempel om användaren trycker på vänster piltangent ska skrivpekaren flyttas vänster och om användaren trycker på höger piltangent ska skrivpekaren flyttas åt höger. Även stöd för att komma ihåg tidigare kommandon hade varit bra så att pil upp och pil ner hade gjort så att användaren bläddrade igenom tidigare exekverade kommandon. En annan bekvämlighet är om <tab >hade autofyllt inputsträngen som i många andra shell. Detta är de största bristerna i mitt ska enligt mig.

7 Diskussion & Avslutning

Under utvecklingen har jag stött på vissa problem med fildeskriptorer vilket har resulterat i att jag fått googla och kolla man-sidor för att förstå mer. Jag tycker att denna kursen har visat bra var ny information letas vilket jag använt mycket i denna uppgift. Annars har utvecklingen gått relativt bra.

Jag är nöjd med mitt skal. Jag hade kunnat förbättra vissa delar (Som jag talar om i 6. Förbättringar) och det kommer jag troligtvis att göra för att lära mig mer om linux och unix. Jag tycker att denna uppgift har varit rolig och otroligt givande. Jag har lärt mig mycket om shell-skal i allmänhet samt nya användningsområden och funktioner av denna laboration. Jag har även lärt mig hur man ska använda fork på ett korrekt sätt och hur man hanterar fildeskriptorer och skriver till dessa. Detta projekt har väckt ett sug att lära sig mer om unix som miljö och har gett mig en betydligt mycket bredare grund att stå på.