

Systemnära programmering HT19

Version: 1.0

Datum: 2019-10-01

CS-användare: ID18mim

mish

Obligatorisk uppgift 3

Mimmi Edholm

Innehåll:

1. Inledning	2
2. Användarhandledning	2
3. Systembeskrivning	3
3.1. Huvudprogram	5
3.2. Parser	5
3.3. Execute	5
3.4. Sighant	5
4. Testkörningar	6
4.1. Interna kommandon	6
4.2. Externa kommandon	6
Begränsningar	8
Diskussion	8

1. Inledning

Denna rapport kommer diskutera ett enkelt skal skrivet i C. Skalet kan utföra externa kommandon men även vissa interna såsom “echo” och “change directory”.

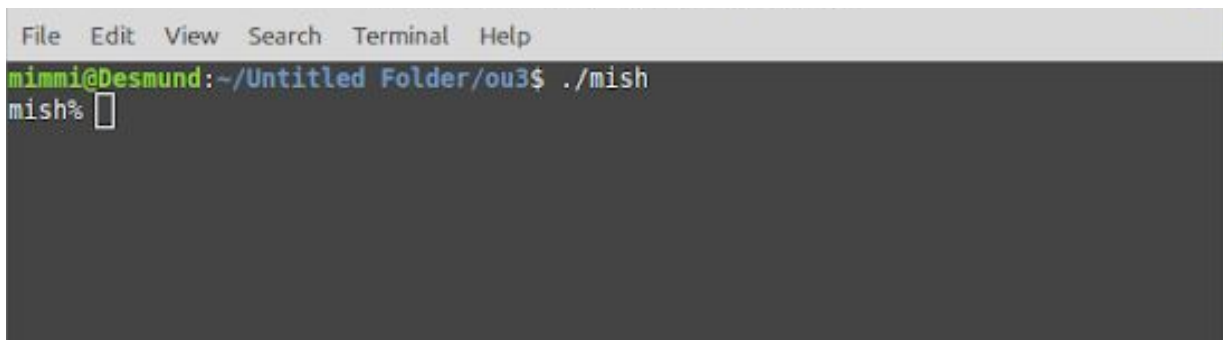
2. Användarhandledning

För att kompilera programmet används en makefil. Kompilatorn som används är GCC och de flaggor används är:

-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.

För att starta programmet skriv kommandot “./mish”.

För ett exempel på användargränssnitt se figur 1.

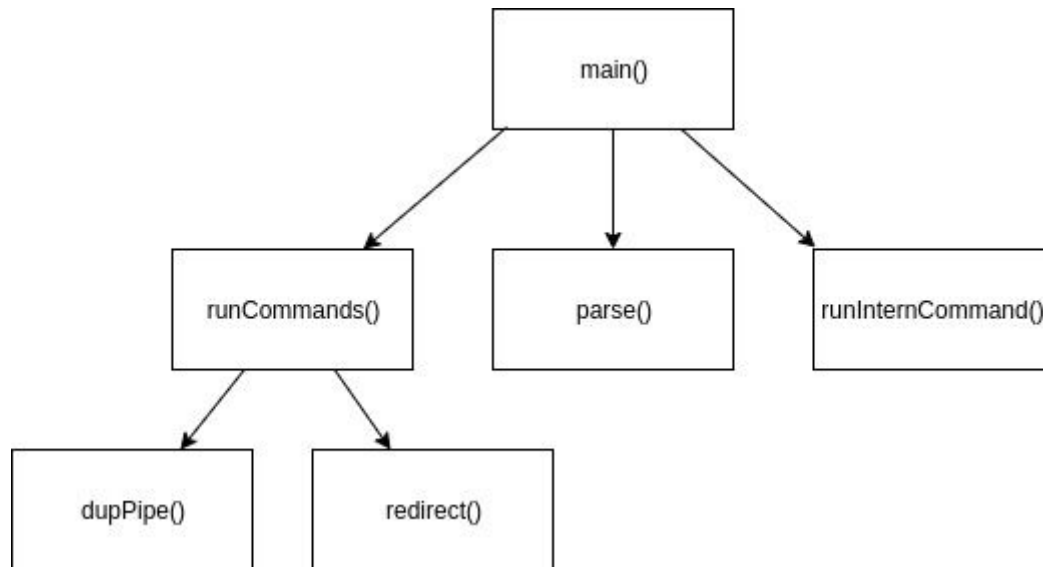
A screenshot of a terminal window. The title bar at the top shows 'File Edit View Search Terminal Help'. The terminal content shows a prompt 'mimmi@Desmund:~/Untitled Folder/ou3\$' followed by the command './mish'. Below this, the prompt changes to 'mish%' and there is a cursor. The background of the terminal is dark grey.

Figur 1. Prompt.

När programmet startat så kan användaren skriva in kommandon, till största del så fungerar skalet som hur man kan förvänta sig att till exempel bash fungerar. För att avsluta en process används signalen sigint.

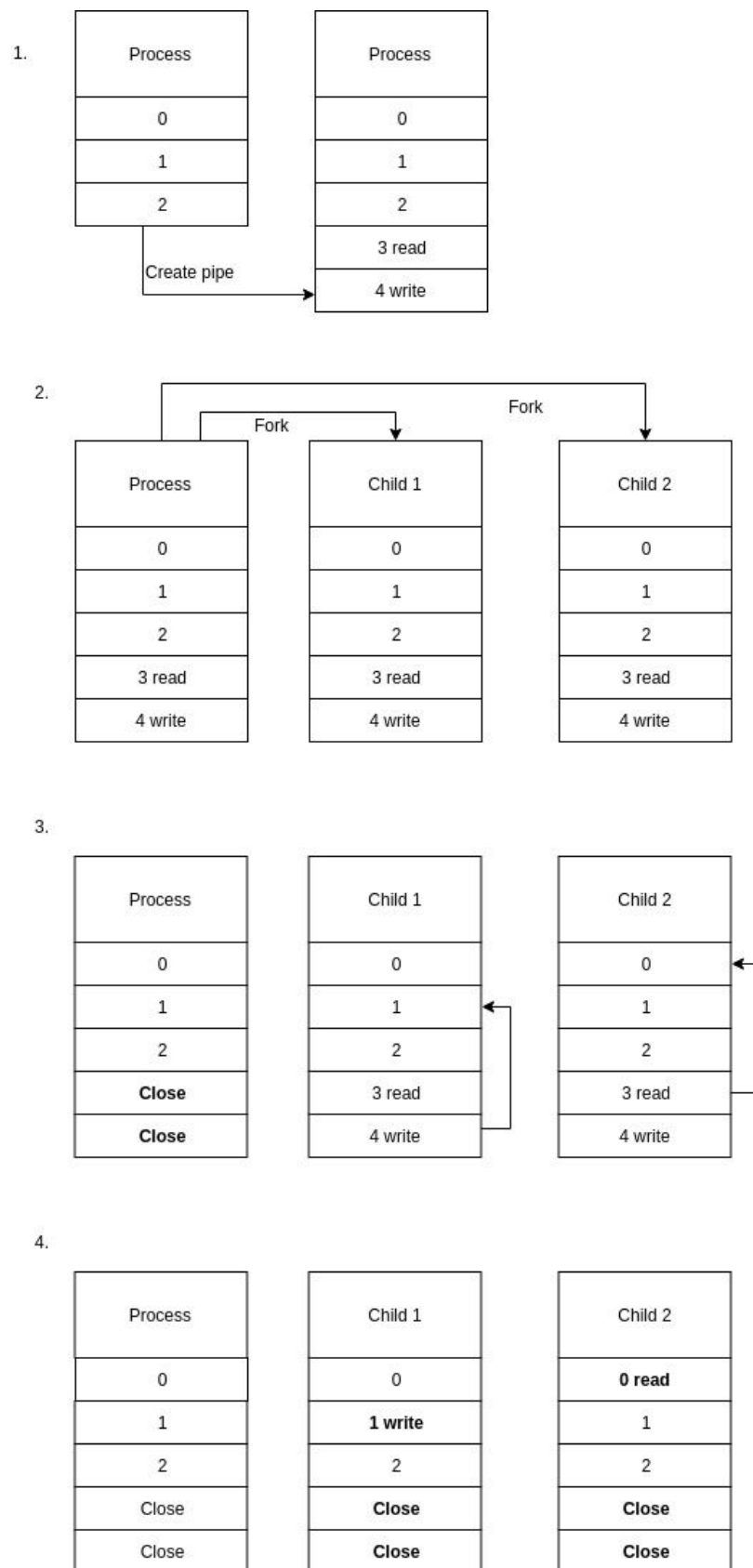
3. Systembeskrivning

Programmet består av fyra C-filer. Varje C-fil beskrivs i en headerfil. Funktionsanropen beskrivs i följande anropsdiagram, se figur 2.



Figur 2. Anropsdiagram.

De processer som skapas kan kommunicera med varandra genom så kallade pipor. De är arrayer med två värden som tar input till plats ett och output till plats noll. Man kan då omplacera standard output och standard input hos processer så de får en enkelriktad kanal med data mellan varandra. Varje process har en egen referens till pipan. För ett illustrerat exempel se figur 3.



Figur 3. Pipes.

3.1. Huvudprogram

Huvudprogrammet mish skriver ut en prompt som användaren interagerar med. Filen har en main-funktion och funktionerna runCommands() och runInternCommand(). Kommandoraden som skrivs in skickas in som parameter i parse. Parse returnerar då en array av kommandon, mish kollar om första kommandot är "cd" eller "echo". Om första kommandot är internt har antagandet gjorts att det som kommer efter är argument till kommandot, funktionen runInternCommand() anropas. Om det första kommandot inte är internt så exekveras kommandoraden genom funktionen runCommand(). Varje barnprocess som skapas kollar kommandots in och ut-fil om de måste redirectas. runCommand() tar hand om att stänga alla pipor som inte ska vara öppna. Om ett kommando inte kunde utföras skrivs ett felmeddelande tillsammans med en ny prompt ut.

3.2. Parser

Parser var ett givet program för uppgiften som delar upp en kommandorad i kommandon, argument och pipor. Det görs i funktionen parse(). Parse sparar all data i en struct. Bland annat in och ut-fil.

3.3. Execute

Heatherfilen till execute var given och skulle implementeras. Filen består av två funktioner, dupPipe() och redirect(). De är funktioner för att underlätta att hantera pipor och redirecta i huvudprogrammet. dupPipe() tar in en pipa, en flagga och en destination, för att sedan utföra dup2(). Som nämnt tidigare i 3.1. så tar inte dupPipe() hand om att stänga pipor, det sköts i huvudprogrammet. redirect() tar in ett filnamn, en flagga och en destination, och som i dupPipe() så utför den dup2(). Men först öppnas filen, då ser funktionen till att filen skapas om den inte redan finns och får rätt rättigheter.

3.4. Sighant

Sighant tar hand om signalen SIGINT och använder kill() och waitpid() för att se till att alla processer dör.

4. Testkörningar

Nedan följer simpla testkörningar för att visa skalet utföra kommandon. Testerna är utförda i en Linuxmiljö.

4.1. Interna kommandon

De interna kommandon som kan utföras är change directory, cd, och echo. För exempel på cd se figur 4. Utan argument tar cd användaren till hemdirectoryt.

```
mimmi@Desmund:~$ cd Untitled\ Folder/ou3
mimmi@Desmund:~/Untitled Folder/ou3$ ./mish
mish% cd
mish% ls
Desktop      Downloads    Pictures     Templates    Videos
Documents    Music       Public      'Untitled Folder'
```

Figur 4. Change directory to home directory.

Det andra interna kommandot som kan köras är echo. Kommandot skriver ut argumenten. För ett exempel på echo se figur 5.

```
mimmi@Desmund:~/Untitled Folder/ou3$ ./mish
mish% echo Hello There
Hello There
mish%
```

Figur 5. Echo "Hello There".

4.2. Externa kommandon

Skalet ska klara av att köra flera processer samtidigt med hjälp av pipor. Figur 6 visar ett exempel på en kommandorad som kommer få programmet att forka tre gånger och skapa två pipor. Kommandot “cat mish.c | grep fork | wc” skriver ut filen mish.c, hittar alla användningar av ordet “fork” och wc räknar dem.

```
mish% cat mish.c | grep fork | wc
      7      22     168
mish% 
```

Figur 6. Pipe test.

mish upptäcker om en fil ska redirectas med hjälp av structen den får av parse(). När programmet ska redirecta en fil, kallar den på redirect(). För ett exempel av användande av redirect, se figur 7.

```
mish% cat < mish.c > mishCopy.txt
mish% cat mishCopy.txt
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <errno.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>
#include <signal.h>
#include <fcntl.h>

#include "mish.h"
#include "parser.h"
#include "sighant.h"
#include "execute.h"
```

Figur 7. Redirect test.

Programmet ska inte ge några “zombies”, alltså processer som inte avslutas och hänger kvar och blivit särkopplade från föräldraprocessen. För att testa så detta inte inträffar utfördes följande test. I första terminalen kördes mish, som fick utföra tre sleep- kommandon, alltså “sleep 60 | sleep 60 | sleep 60” och sedan avbröts med “kill -SIGINT mish-id”, där mish-id är id:et för mish-processen. I andra terminalen, kördes kommandot “ps aux | grep sleep”. Alltså, skriv ut alla processer och hitta processen sleep. På detta sätt kan man se till att skalet dödar alla processer. För ett exempel av detta test se figur 8.


```
File Edit View Search Terminal Help
mish% sleep 60 | sleep 60 | sleep 60
rc
mish% sleep 60 | sleep 60 | sleep 60
mish%

File Edit View Search Terminal Help
mimmi@Desmund:~/Untitled Folder/ou3$ ps aux | grep sleep
mimmi  13794  0.0  0.0  15972  992 pts/2    S+   16:47   0:00 grep --color=au
to sleep
mimmi@Desmund:~/Untitled Folder/ou3$ ps aux | grep mish
mimmi  13756  0.0  0.0  4524   820 pts/0    S+   16:46   0:00 ./mish
mimmi  13868  0.0  0.0  15972  1104 pts/2    S+   16:48   0:00 grep --color=au
to mish
mimmi@Desmund:~/Untitled Folder/ou3$ kill -SIGINT 13756
mimmi@Desmund:~/Untitled Folder/ou3$ ps aux | grep sleep
mimmi  13900  0.0  0.0  15972  1012 pts/2    S+   16:49   0:00 grep --color=au
to sleep
mimmi@Desmund:~/Untitled Folder/ou3$
```

Figur 8. Zombie test.

Begränsningar

De begränsningar som finns i denna lösning är såklart att det inte fungerar lika smidigt som ett skal som till exempel bash. Det går till exempel inte att få åtkomst till sin kommandohistorik vilket det brukar underlätta att ha. Även att bara två interna kommandon är implementerade och echo kan inte redirectas. Det finns alltså mycket förbättringspotential och rum för finslip, men i grunden gör mish vad man kan förvänta sig att det ska kunna göra.

Diskussion

Huvudprogrammet har rätt få funktioner och därför får varje funktion relativt stort ansvar. Koden hade såklart blivit mer lättläst med fler funktioner, men jag anser att det inte är ett alltför stort problem i detta fall. Funktionerna i execute underlättar såklart läsbarheten men kunde ha varit en del av huvudprogrammet.

Uppgiften har gått ut på att förstå stora koncept men man måste även ha haft koll på mindre detaljer och för en mindre erfaren programmerare har det varit mycket att hålla reda på. Det har varit väldigt givande och gett större insikt i hur filer hanteras i C och Unix, och för det mesta har det varit roligt att lösa uppgiften.