

# **Laboration 3 - mish**

Laborationsrapport

<b>Namn</b>	Emil Söderlind
<b>CS</b>	id15esd
<b>Kurs</b>	Systemnära programmering HT18
<b>Kursansvarig</b>	Mikael Rännar
<b>E-brev</b>	Soderlindemil@gmail.com

# Contents

<b>1 Inledning</b>	<b>2</b>
1.1 Introduktion . . . . .	2
1.2 Problembeskrivning . . . . .	2
<b>2 Åtkomst och användarhandledning</b>	<b>3</b>
2.1 Exempelkörning . . . . .	3
2.2 Signalhantering . . . . .	4
<b>3 Systembeskrivning</b>	<b>4</b>
3.1 Anropsdiagram & algoritmbeskrivning . . . . .	4
3.2 Kommunikation mellan processer . . . . .	7
<b>4 Lösningens begränsningar</b>	<b>10</b>

# 1 Inledning

## 1.1 Introduktion

Under kursen Systemnära programmering HT18 har ett antal programmeringslaborationer utförts. Denna rapport behandlar den tredje laborationen av fyra med namn "mish". Mish står för "minimal-shell" och går ut på att implementera ett eget skal som kan hantera flertalet kommandon via pipor i separata processer.

## 1.2 Problembeskrivning

```
[mish% cat mish.h | wc  
      45      89     798  
[mish% cat mish.c | wc | tail -1  
      299      797    7707  
mish%
```

Figure 1: Exempelkörning av mish-skalet

Ett skal är ett text-baserat program som körs när användaren öppnar en terminal. Skalet skriver ut en prompt (t.ex. "mish: ") och invänder användarens kommandon. Se figur 1 för en exempelkörning av det implementerade mish-skalet.

Målet med laborationen har varit att implementera ett shell/skal som dels hanterar kommandona **cd** och **echo** med en egen implementation och hanterar övriga kommandon genom att vidarebefordra argumenten till den "ursprungliga" implementationen av respektive kommando. Skalet kan hantera exekvering av kommandon/program, pipor mellan program samt omdirigering av stdin och stdout.

Varje pipa "|" mellan kommandon omdirigerar det vänstra kommandots stdout till det högra kommandots stdin osv i en lång kommando-kedja. Om användaren trycker [Ctrl] + [C] så stängs kommandonas processer av och prompten skrivs ut.

## 2 Åtkomst och användarhandledning

Lösningen består av följande filer med tillhörande header-filer:

- **Makefile**

Makefile som kompilerar skalet

- **mish.c**

Består av main-funktion samt stora delar av skalets logik

- **parser.c**

Given fil som hanterar parsning av kommandon från användaren.

- **sighant.c**

Hanterar signalhantering

- **execute.c**

Består av funktionalitet som rör omdirigering av stdin/stdout samt in/ut-omdirigering från fil.

### 2.1 Exempelkörning

För att kompliera programmet skriver användaren följande medan hen står i samma mapp som filerna:

**make** (Följt av enter)

För att därefter köra skalet skriver hen:

**./mish** (Följt av enter)

Därefter skriver programmet ut prompten "mish: " och kommandon och pipor kan skrivas in och köras då användaren trycker enter. T.ex. följande kommando:

**cat mish.h | tail -1 | wc** (Följt av enter)

Ovan kommando-kedja består av cat, tail samt wc. Cat använder sin första input-parameter "mish.h" och skriver ut dennes innehåll till sin stdout. Tail tar därefter in cat:s stdout som sin stdin och skriver vidare den första raden (flagga -1) av stdin och ger den till wc:s stdin som räknar antalet rader, tecken och ord. Det ger följande output:

På liknande sätt kan användaren skriva in både fler och färre kommandon samt omdirigera så att första kommandot tar stdin från en fil samt sista kommandot skriver ut sin stdout till en fil.

## 2.2 Signalhantering

Användaren kan skicka signalen SIGINT till skalet från en annan terminal med "kill -INT <skalets-PID>", skalet stänger därefter av de kommandon som kör och prompten skrivas ut. Ovanstående kan också framställas genom att användaren trycker [Ctrl] + [C] medan kommandon körs via skalet.

# 3 Systembeskrivning

I denna del beskrivs lösningen, dess algoritm och anropsdiagram/sekvensdiagram illustreras.

## 3.1 Anropsdiagram & algoritmbeskrivning

Lösningens anropsdiagram återfinns i figur 2. När skalet körs startas main-funktionen som kallar på loopRunShell-funktionen. LoopRunShell-funktionen kallar på signalhanterings-funktionen så att signalen SIGINT kan hanteras. När förälder-processen tar emot en SIGINT-signal så "dödas" de aktiva barn-processerna (aktiva kommandona), föräldrar väntar ut så att barnen "dör" och kan därefter skriva ut prompten igen för mer kommandon.

LoopRunShell-funktionen loopar därefter och kallar på runShell i varje loop-varv. RunShell skriver ut prompten till användaren, använder sig av den givna parser.c för att sedan avgöra om den inskrivna kommandokedjan (t.ex. "**cat mish.c | tail -1 | wc**") består av ett internt kommando (cd/echo) eller ett externt kommando.

I fallet med ett internt kommando kallas respektive funktion med tillhörande argument (internal\_cd/internal\_echo). Om runShell identifierar externa kommandon så skapas (antal kommandon i kommando-kedjan minus 1 stycken) pipor och runCommand kallas för varje kommando.

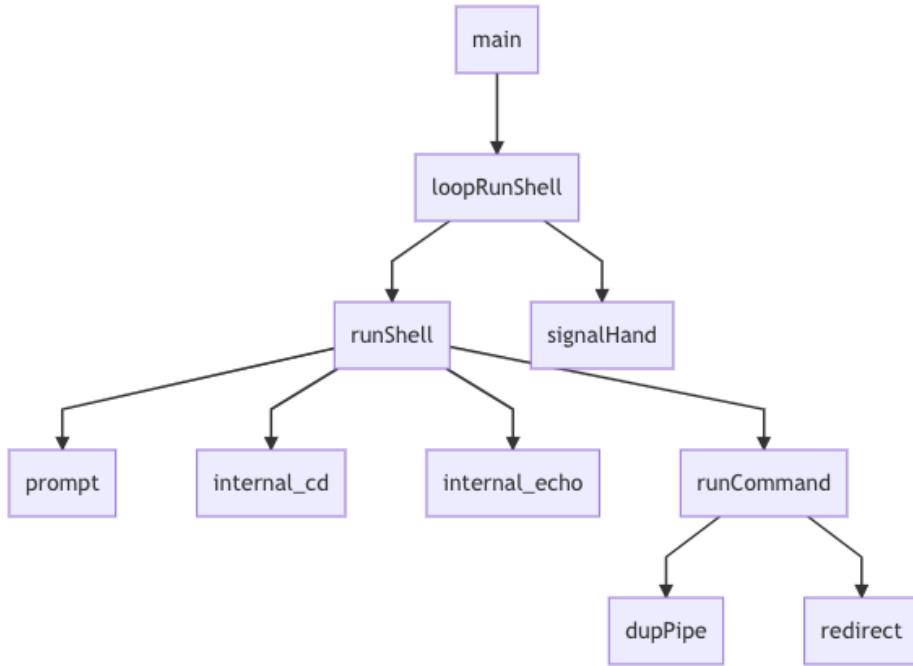


Figure 2: Anropsdiagram

RunCommand skapar en ny barn-process för varje barn och sparar deras Process-ID i en lista i föräldra-processen. Därefter kopplar varje barn-process om dennes pipor till att ta in (stdin) från föregående process och ge ut (stdout) till nästa process i kommando-kedjan. Observera att första och sista kommandon i kedjan blir specialfall.

Efter att barn-processerna kopplat om deras pipor kör de execvp utifrån deras vardera kommando-argument. Under tiden ställer sig föräldra-processen och väntar på att barn-processerna blir klara med sin exekvering, detta utifrån den sparade Process-ID listan.

Om föräldra-processen tar emot en SIGINT-signal kallas en funktion som "dödar" samtliga barn-process-ID:n och väntar sedan på att de "dör".

I figur 3 nedan återfinns ett översiktligt sekvensdiagram över skalets algoritm, diagrammet finns också på följande länk: [Länk till sekvensdiagram](#).

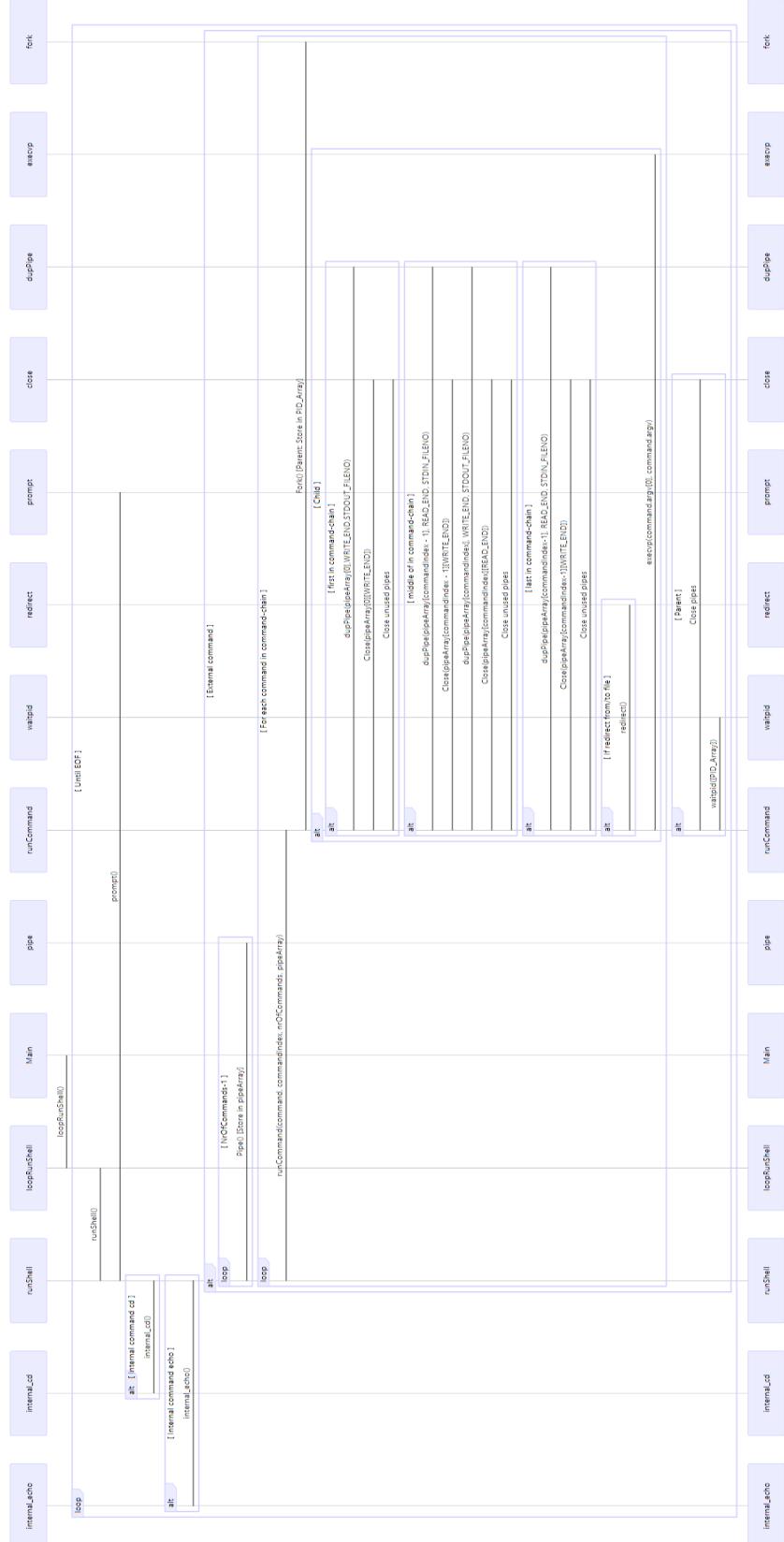


Figure 3: Anropsdiagram  
6

## 3.2 Kommunikation mellan processer

Skalet består av en förälder-process som skapar barn-processer för varje kommando, kopplar ihop barn-processernas pipor och eventuellt omdirigerar deras stdin/stdout till/från fil. Därefter kör varje barn-process execvp() med det tillhörande kommandot. Notera att skalet kan köra de interna kommandona **cd**/**echo** och hanterar då inte pipor eller omdirigeringar.

I nedan figur 4-8 illustreras då följande kommando-kedja körs:

```
"cat mish.h | tail -1 | wc"
```

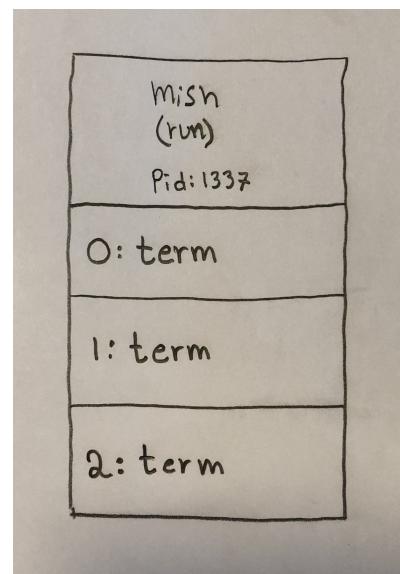


Figure 4: Skapandet av föräldra-process

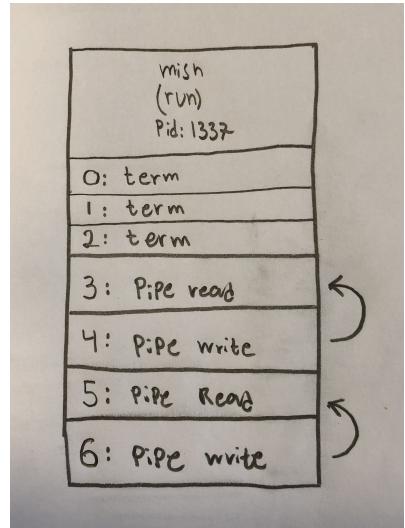


Figure 5: Föräldern skapar pipor

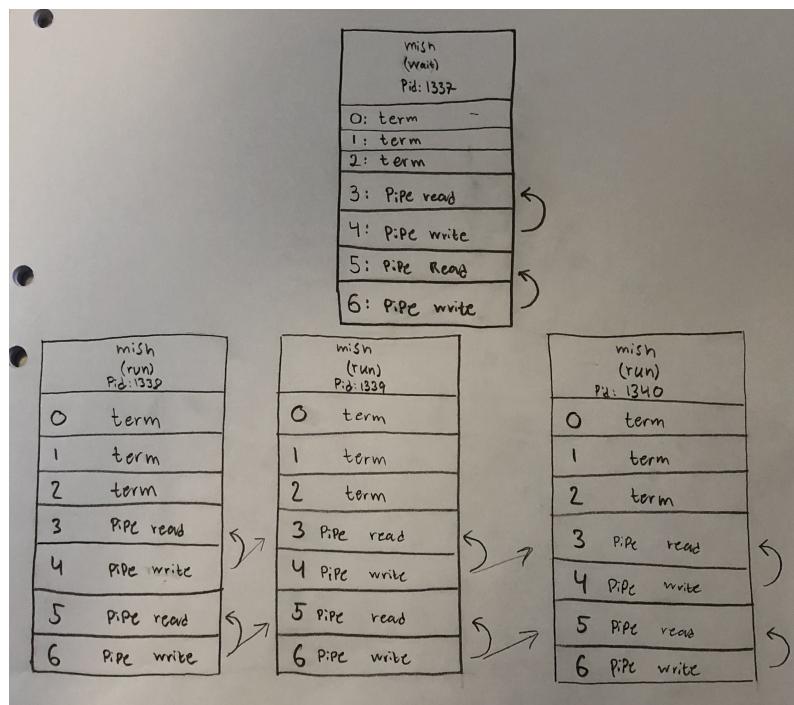


Figure 6: Föräldern kallar på fork() tre gånger och börjar vänta

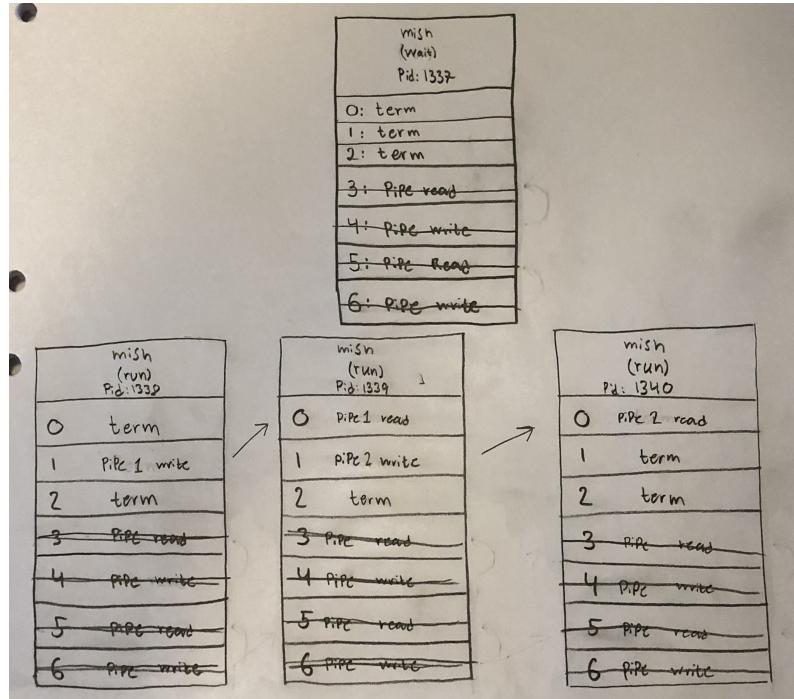


Figure 7: Barn-processerna dup/close:ar fildeskriptorer

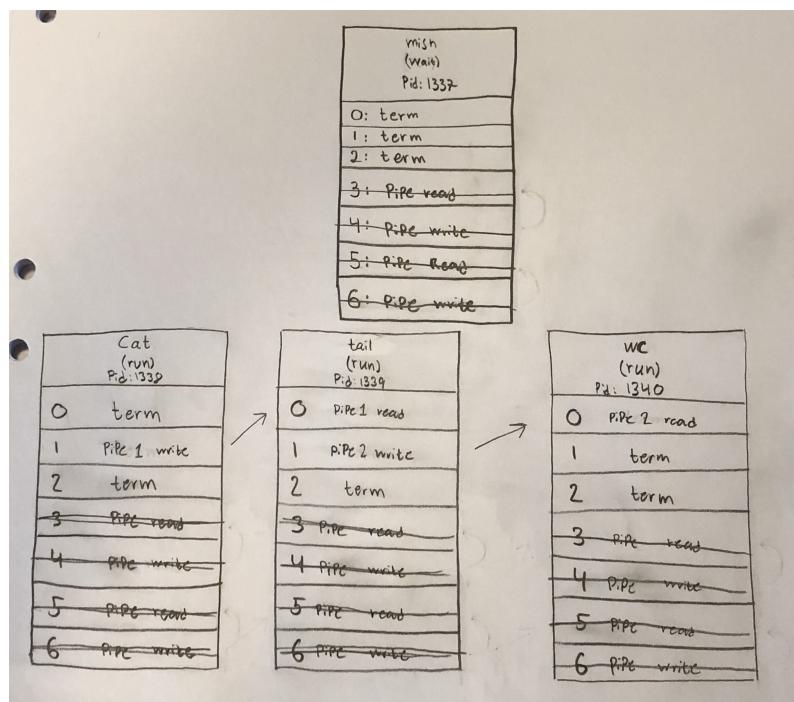


Figure 8: Barn-processerna kör respektive kommando (cat/tail/wc)

## 4 Lösningens begränsningar

Skalets interna kommandon **cd/echo** hanterar inte att köras med pipor eller omdirigeringar. Den smidiga genvägen som återfår tidigare skrivna kommando genom uppåtpil, finns dessvärre inte implementerad.