UMEÅ UNIVERSITET 11 oktober 2018

Laboration 3

Systemnära Programmering, HT 2018, 15hp

Mish

Namn	Thomas Sarlin
E-mail	thsa0043@gapps.umu.se
Huvudlärare	Mikael Rännar
Handledare	Klas af Geijerstam, Joel Sandman & Elias Åström

Innehåll

1 Problembeskrivning

2 Inledning

I den tredje laborationen på kursen systemnära programmering så ska det implementeras ett minimal skalprogram (shell). Ett skalprogram tar in kommandon från användaren och exekverar dessa i den ordning dessa skrivs in. Lösningen ska kunna hantera exekvering av kommandon samt program. Ifall flera kommandon/program anropas ska pipor skapas mellan dessa för att omdirigera standrard input och output. Programmet ska även hantera in och utfiler. All information som ges av användaren kommer ska hanteras med parser.c som returnerar en hanterbar struct.

3 Åtkomst och användarhandledning

3.1 Åtkomst

För handledare så ligger filerna i mappen /id15tsn/edu/Systemnaera/ou3.

3.2 Användarhandledning

3.2.1 Kompilering

För att använda programmet behöver det först kompileras. Detta görs enklast med något av följande kommandon:

```
make
make all
```

Efter körning kan man enklast rensa upp i sin mapp med något av följande kommandon:

```
make clear
make clearAll
```

clear tar bort alla .o filer som skapats och *clearAll* tar även bort den körbara filen som skapats med make.

3.2.2 Körning

När kompileringen är klar körs programmet genom att köra kommandot:

```
./mish
```

Vilket startar programmet och ifall programmet är igång så visas följande prompt:

```
mish%
```

Nu kan du köra kommandon precis som i andra skalprogram till exempel

```
mish% cat Makefile
```

som skriver ut innehållet i Makefile till stdout.

För att köra flera kommandon och skapa pipor emellan dessa så används | för att indikera pipor. Se nedan

```
mish% cat Makefile | sort
```

Där resultatet från Makefile skickas vidare till sort istället för att skrivas till stdout.

Ifall du vill skriva eller läsa från/till en fil så används < och > för att indikera in och utfilerna. Se nedan

```
mish% sort < Makefile
mish% cat Makefile > Makefile_copy.txt
```

Det övre kommandot tar innehållet från Makefile till cat genom att ersätta stdin med input från filen Makefile.
Det nedre kommandot skriver resultatet från cat Makefile till filen Makefile_copy.txt.

4 Algoritmbeskrivning

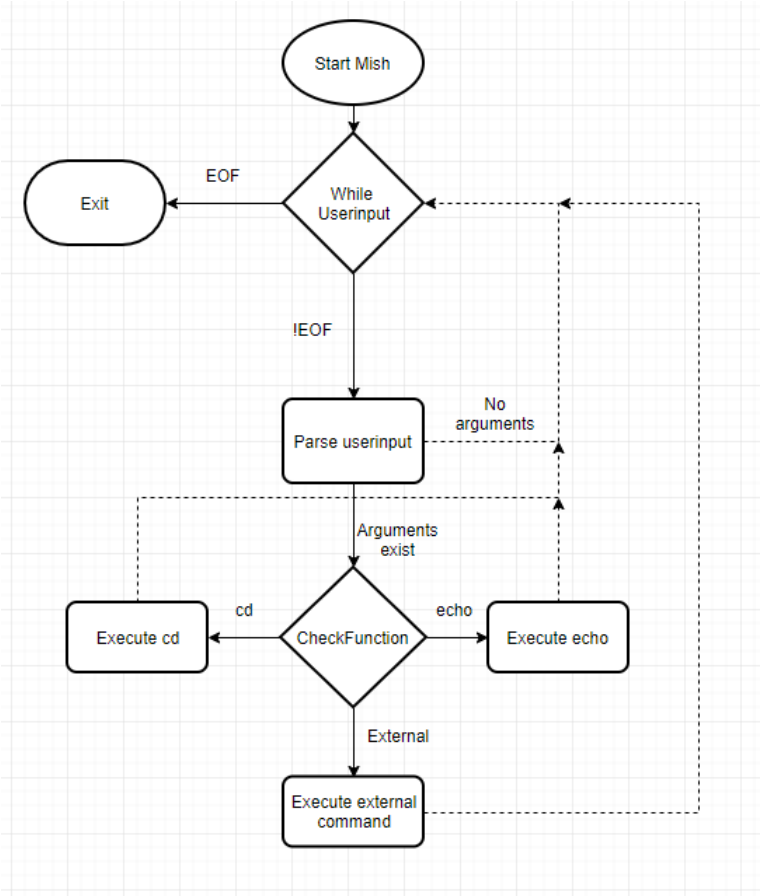
Figur 1: Algoritm för programmet Mish

```
while user enters commands
    n = number of commands
    if external command/commands
        create n-1 pipes
        for i < n do fork
            if fork
                redirect stdin/stdout to pipes/files
                close pipes
                execute command i
        close pipes
        wait for n status messages
    if internal command
        execute internal command
```

5 Systembeskrivning

5.1 Flödesschema

Figur 2: Flödesschema Mish



5.2 Filer

5.2.1 mish.c/mish.h

Mish är huvuddelen av systemet, själva programloopen som tar input från användaren och kallar på rätt funktioner för att genomföra det användaren kräver av programmet. Använder sig av parser.h för att hantera indata från användaren och kallar i sin tur på andra includes såsom external, cd samt echo.

5.2.2 external.c/external.h

External ansvarar för all hantering av externa kommandon samt de pipor och forks som krävs för att koppla kommandona och exekvera dessa.

5.2.3 cd.c/cd.h

Ansvarar för byte av directory.

5.2.4 echo.c/echo.h

Ansvarar för utskrift av an input från användaren, denna version av echo tar även in och utfiler.

5.2.5 execute.c/execute.h

Ansvarar för att omdirigera stdin/stdout till pipor alternativt fildeskritors.

6 Resultat

Det hela slutade i en fungerande mini-shell som är fullt användbar till den utsträckning som laborationen indikerar.

- Programmet
    1. Använder parser.c och parser.h
    2. Använder execute.c och execute.h
    3. Klarar testerna på inlämningssidan.
    4. Kontrollerar systemanrop och hanterar dessa om det skulle uppstå felanrop.
    5. Felutskriften sker på stderr.
    6. Hanterar infiler.
    7. Hanterar utfiler.
    8. Har CD och ECHO implementerat.
    9. Hanterar externa kommandon och anrop till program.
    10. Hanterar flera externa kommandon och anrop till program.
    11. Skapar en process för varje externt kommando.
    12. Skapar och använder pipes.
    13. Hanterar SIGINT.
    14. Lämnar inte processer kvar ifall programmet avbrutits (zombies).
- Makefile finns, som separatkompilerar de olika filerna.
- Valgrind indikerar inga minnesläckor.
- Kod
    1. Indenterad
    2. Kommenterad
    3. Funktioner uppdelade
    4. Radlängd begränsad
    5. Bra namngivning

7 Diskussion/reflektion

Jag stötte på endel problem på vägen men överlag så gick laborationen väldigt smidigt om man valde att gå på föreläsningarna. Direkt man inte gick på en föreläsning så missade man en viktig del till laborationen (indikerar på hur bra lektionerna är upplagda). Till exempel så missade jag lektionen med signalhantering vilket lämnade mig ganska ovetandes om hur man hanterar zombies / processer som fortsätter gå då *mish* avslutas.
Att komma igång och förstå hur man skapade sin execute.c så att den fungerade som tänkt var ännu en grej som var rätt klurig.
Jag tycker att uppgiften är riktigt rolig och man känner sig väldigt duktig då man väl avklarat uppgiften. Detta är väldigt viktigt i kurser där programmering är fokus. Ifall man känner att man gjort något stort så blir man motiverad att dyka djupare i processer och skal.
Jag är väldigt glad över klassen som varit väldigt öppna med att bolla ideér om man kört fast.

8 Testkörningar

8.1 Test från labspeccen

8.1.1 Test 1 - kolla att det bara finns en mish-process

```
./mish
mish%

bash% ps -u
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
ThomasS+   238  0.0  0.0  10556   584 tty1     S    14:21   0:00 ./mish
ThomasS+   239  0.0  0.0  17380  1924 tty2     R    14:22   0:00 ps -u
```

8.1.2 Test 2 - kolla att det bara finns en mish-process

```
mish% /bin/nosuchcommand
/bin/nosuchcommand: No such file or directory
mish%
bash% ps -u
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
ThomasSa+   238  0.0  0.0  10556   612 tty1     S    14:21   0:00 ./mish
ThomasSa+   241  0.0  0.0  17380  1920 tty2     R    14:24   0:00 ps -u
```

8.1.3 Test 3 - Inga zombies

```
mish% sleep 60 | sleep 60 | sleep 60

bash% ps -u
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
ThomasS+   209  0.0  0.0  10556   612 tty1     S    14:06   0:00 ./mish
ThomasS+   232  0.3  0.0  13956   816 tty1     S    14:15   0:00 sleep 60
ThomasS+   233  0.0  0.0  13956   820 tty1     S    14:15   0:00 sleep 60
ThomasS+   234  0.0  0.0  13956   820 tty1     S    14:15   0:00 sleep 60
ThomasS+   235  0.0  0.0  17380  1924 tty2     R    14:15   0:00 ps -u

bash% kill -INT 209
bash% ps -u
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
ThomasS+   209  0.0  0.0  10556   612 tty1     S    14:06   0:00 ./mish
ThomasS+   236  0.0  0.0  17380  1924 tty2     R    14:15   0:00 ps -u
```

8.2 Egna test

8.2.1 Test 1 - Skapa fil

```
bash% ls *.txt
ls: cannot access '*.txt': No such file or directory
mish% cat Makefile > apa.txt
bash% ls *.txt
apa.txt
```

8.3 Läs från fil

```
mish% tail -n 5 < apa.txt
clearAll:
        rm *.o mish
valgrind:
        valgrind ./mish
```

8.4 Många kommandon, en process kvar efter avslutat kommando

```
mish% cat < apa.txt | sort | tail -n 1 | wc -l | cat
1
mish% ps -u
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
ThomasS+   275  0.0  0.0  10556   608 tty2     S    14:36   0:00 ./mish
ThomasS+   284  0.0  0.0  17380  1924 tty2     R    14:40   0:00 ps -u
```