2G1520 Operativsystem

Labb 2 – smallShell v2.01

Wilhelm Svenselius 830412-0218
wsv@kth.se

Denna sida ersätts med förtryckt försättsblad vid inlämning.

Uppgiftsbeskrivning

Uppgiften går ut på att skriva en primitiv kommandotolk. Primitiv betyder i det här fallet att tolken är begränsad till fyra funktioner: Byta katalog, starta förgrundsprocess, starta bakgrundsprocess och avsluta sig själv. Tack vare att många "inbyggda" kommandon i UNIX egentligen består av lösa program blir kommandotolken ganska användbar, trots sin enkla uppbyggnad.

Kommandotolken skriver ut information om process-ID för de processer den startar. Vid avslut skriver den ut process-ID och, för förgrundsprocesser, hur lång exekveringstiden var. Skulle processen ha avslutats på onormalt vis skrivs även information om detta ut. Alla kommandon utöver de inbyggda *cd* (byt katalog) och *exit* (avsluta) körs i separata processer.

Kommandot *cd* skiljer sig en smula från de facto standard genom att växla till den katalog som anges av miljövariabeln *HOME* ifall den angivna sökvägen inte existerar. Skulle miljövariabeln *HOME* vara ogiltig eller inte existera, skrivs ett felmeddelande ut.

Programmet är skrivet i ANSI C och kompilerar utan varningar eller fel med *gcc -xc -Wall* på såväl SunOS 5.9 (*ripper.it.kth.se*) (gcc 2.95.2) som Windows/Cygwin 1.5.18-1 (gcc 3.4.4).

Programuppbyggnad

Programmet är uppdelat i flera funktioner där *main* sköter inläsning av användardata och processhantering. En dynamiskt allokerad buffert används för att läsa in kommandoraden. Därefter "städas" denna från onödiga mellanslag, specialtecken och annat med funktionen *cleanup*. Ytterligare en hjälpfunktion, *argsplit*, delar upp kommandoraden i stycken och returnerar en array med pekare till varje stycke. Den resulterande arrayen är lämplig att använda som parameter i anrop till *execvp.*

Eftersom det bara finns två kommandon hanteras detta med hjälp av en enkel *if-elseif-else*-sats. Det inbyggda kommandot *exit* är trivialt, *cd* får lite specialbehandling i och med att den osplittade kommandoraden skickas till systemanropet *chdir*. Detta för att *cd* skall kunna hantera katalognamn med mellanslag. Skulle anropet till *chdir* misslyckas kommer den nya katalogen istället att bli användarens hemkatalog, förutsatt att en giltig sådan är angiven i miljövariabeln *HOME*. Denna del av uppgiften är lite tveksam i mitt tycke, eftersom det innebär att användaren kan hamna i en annan katalog än vad han hade väntat sig (se andra exempelutskriften, nedan).

Förgrundprocesser hanteras på enklaste möjliga vis med en kombination av *vfork* och *execvp*. *vfork* snarare än *fork* används eftersom *exec* sker omedelbart; det finns alltså inget behov av att kopiera föräldraprocessens sidor. Efter att processen har avslutats (väntan sker med hjälp av en funktion *wait_ch* som är en anpassad version av *wait_child* från laboration 1) skrivs information om körtid ut, samt anledningen till att processen avslutades.

Bakgrundsprocesser hanteras *inte* genom någon signalhanterare, trots att uppgiftstexten antydde att detta var den förväntade lösningsmetoden (det står dock ingenstans att signalhanterare måste användas) utan genom en s k sentry-process som väntar på att barnprocessen skall avslutas. Detta innebär att vi slipper synkroniseringsproblematiken (risk för avbrutna systemanrop, etc) som signalhantering innebär.

När programmet avslutas (genom *exit*) skickas en *SIGKILL* till alla fortfarande aktiva barnprocesser, detta ifall användaren avslutar kommandotolken innan dess barnprocesser har kört klart. En alternativ lösning hade varit att vänta in alla barnprocesser, men detta medför en risk för låsning ifall någon barnprocess inte svarar.

Programutskrift

Följande testutskrift kommer från *ripper.it.kth.se*, med operativsystemet SunOS 5.9. Programmet *idiot* är en oändlig loop som anropar *sleep(1)*.

```
[it03_wsv@ripper it03_wsv]$ cd opsys
[it03_wsv@ripper opsys]$ ls
idiot.c  smash.c
[it03_wsv@ripper opsys]$ gcc -xc -Wall smash.c -o smash
[it03_wsv@ripper opsys]$ ./smash
> ls
*** Foreground process [4993] started.
idiot.c  smash  smash.c
*** Process [4993] terminated normally.
*** Runtime: 19 msec.
> gcc -xc -Wall idiot.c -o idiot
```

```
*** Foreground process [4994] started.
*** Process [4994] terminated normally.
*** Runtime: 489 msec.
> ls
*** Foreground process [5000] started.
idiot  idiot.c  smash  smash.c
*** Process [5000] terminated normally.
*** Runtime: 19 msec.
> ps
*** Foreground process [5001] started.
   PID TTY      TIME CMD
  5001 pts/16   0:00 ps
  4992 pts/16   0:00 smash
  4955 pts/16   0:01 bash
*** Process [5001] terminated normally.
*** Runtime: 68 msec.
> ./idiot &
> *** Background process [5003] started.
ps
*** Foreground process [5004] started.
   PID TTY      TIME CMD
  5002 pts/16   0:00 smash
  4992 pts/16   0:00 smash
  4955 pts/16   0:01 bash
  5003 pts/16   0:00 idiot
  5004 pts/16   0:00 ps
*** Process [5004] terminated normally.
*** Runtime: 68 msec.
> ls
*** Foreground process [5005] started.
idiot  idiot.c  smash  smash.c
*** Process [5005] terminated normally.
*** Runtime: 19 msec.
> exit
*** Goodbye.
Killed
[it03_wsv@ripper opsys]$ ps
   PID TTY      TIME CMD
  4955 pts/16   0:01 bash
  5006 pts/16   0:00 ps
[it03_wsv@ripper opsys]$
```

Denna testutskrift är gjord i Cygwin 1.5.18-1 på Windows XP Pro SP2 och demonstrerar *cd*:

```
Wilhelm Svenselius@Lind ~
$ env | grep HOME
HOMEPATH=\Documents and Settings\Wilhelm Svenselius
HOME=/home/Wilhelm Svenselius
HOMEDRIVE=C:

Wilhelm Svenselius@Lind ~
$ ls
THIS_IS_MY_HOMEDIR  smash.c

Wilhelm Svenselius@Lind ~
$ gcc -xc -Wall smash.c -o smash.exe

Wilhelm Svenselius@Lind ~
$ smash
> pwd
*** Foreground process [3596] started.
/home/Wilhelm Svenselius
*** Process [3596] terminated normally.
*** Runtime: 108 msec.
> cd ..
> ls
*** Foreground process [4012] started.
Wilhelm Svenselius
*** Process [4012] terminated normally.
*** Runtime: 16 msec.
> cd Lukas Grimfors
*** Directory doesn't exist, trying home directory.
> ls
*** Foreground process [3752] started.
THIS_IS_MY_HOMEDIR  smash.c  smash.exe
*** Process [3752] terminated normally.
*** Runtime: 16 msec.
> cd ..
> cd Wilhelm Svenselius
```

```
> ls
*** Foreground process [2684] started.
THIS_IS_MY_HOMEDIR  smash.c  smash.exe
*** Process [2684] terminated normally.
*** Runtime: 16 msec.
> rm THIS_IS_MY_HOMEDIR
*** Foreground process [2952] started.
*** Process [2952] terminated normally.
*** Runtime: 142 msec.
> exit
*** Goodbye.
Killed
```

Förberedelseuppgifter

1. Motivera varför det ofta är bra att exekvera kommandon i en separat process.

Man vill inte att ett kommando som kanske gör något fult och kraschar tar med sig resten av programmet, om det går att undvika. Ofta (särskilt i GUI-applikationer) är det också önskvärt att ha användargränssnittet och arbetsrutinerna exekverandes parallellt så att gränssnittet förblir responsivt under tiden arbete pågår.

2. Vad händer om man inte i kommandotolken exekverar wait() för en barn-process som avslutas? Hur skall man utläsa `SIGSEGV`?

En barn-process som fortfarande är igång då föräldraprocessen avslutas kommer att överföras till *init* (dvs, den får *init* som förälder) om den svarar. Svarar den inte blir den en s k "zombieprocess" och ligger kvar och slösar resurser. *SIGSEGV* är signalen för segmenteringsfel, dvs processen som tar emot signalen har gått utanför sitt minnesområde. Vanligaste orsaken till detta är programmerarfel.

3. Varför kan man inte blockera `SIGKILL`?

För att det skulle förta hela syftet med *SIGKILL*, nämligen att ta livet av processer som inte uppför sig ordentligt. Kunde man blockera *SIGKILL* skulle en "elak" process kunna göra sig själv odödlig.

4. Hur skall man utläsa deklarationen `void (*disp)(int))`?

En funktionspekare *disp* till en funktion som returnerar *void* och tar en *int* som enda argument.

5. Vilket systemanrop använder `sigset(3C)` troligtvis för att installera en signalhanterare?

Den använder troligtvis *signal*.

6. Studera körningsexemplet och förklara varför man inte har bytt "working directory" till `/home/ds/robertr` när man avslutat miniShell:et?

Kommandot *cd* anropar *chdir*, vilket byter ut processens arbetskatalog. Processen som startade *miniShell* har också en arbetskatalog, så när miniShell avslutas återställs arbetskatalogen till vad den var tidigare. Varje process har en egen uppsättning miljövariabler, där bland annat arbetskatalogen ingår.

Källkod

Källkoden är bifogad i "*for dummies*"-version med pedagogisk färgkodning och extra stor stil. Den återfinnes även i AFS, där i kompilerbart format, med adressen:

```
/afs/it.kth.se/home/it03/it03_wsv/opsys/smash.c
```

Förslag på kompilatoranrop:

```
gcc -xc -Wall smash.c -o smash
```

Utvärdering

Jag uppfattade labben som enklare än labb 1, visserligen är jag nu aningen mer van vid systemprogrammering i UNIX men uppgiften kändes mindre omfattande och de nödvändiga kommandona ur systembiblioteken var enklare och färre

(åtminstone om man hoppar över signalhanteringen). Jag uppskattar särskilt att uppgiften medger ett brett spektrum av olika lösningar på de respektive delmomenten.

Kanske vore det vettigt att byta plats på laborationerna 1 och 2 till nästa års kurs? Jag uppfattade att tvåan krävde mer kunnande om C och mindre om UNIX' systembibliotek, vilket kan göra den lättare att ta till sig för en nybörjare än den första labben. Dessutom är tvåan kortare om man ser till kodstorleken.

Jag bedömer att den här laborationen tog mig sex timmar att slutföra, inklusive skrivandet av denna rapport.

Bilagor

1. Källkod “smash.c”