Laboration 2

Uppgift 1. *Eget mikro-shell*

Skriv ett *C*-program som använder `execlp` i en barnprocess för att utföra det kommando som användaren skriver in. Hur avancerat kan du få ditt shell? Du ska kunna ge vilka kommandon som helst, du får inte skriva ett skal som bara klarar av ett visst antal kommandon. Dock behöver du inte kunna bli superuser. Kommandot `cd`, är ett så kallat *inbyggt kommando* som måste hanteras på ett speciellt sätt, se en speciell ledning längre ner. Shellet ska, för varje kommando som ges, utföra kommandot i en barnprocess med hjälp av ett `exec()`-kommando. Detta innebär att det kan bli krångel med utskrifter, föräldraprocessen måste hela tiden använda `wait()` för att utskrifterna ska skötas ordentligt. Det kommer troligtvis att bli en del svårigheter med utskrifterna, vi kommer inte att kräva perfektion i den här kursen avseende utskrifterna.

Körexempel:

```
> ./a.out
Ange kommando > ls -l
total 36
-rwxr-xr-x    1 sand     users        9854 2005-09-06 14:10 a.out
drwxr-xr-x    2 sand     users        4096 2005-09-06 14:08 dir1
-rw-r--r--    1 sand     users           0 2005-09-06 14:19 fillista
-rw-r--r--    1 sand     users         437 2005-09-05 13:43 lab3_1.c
-rw-r--r--    1 sand     users         437 2005-09-05 13:56 lab3_2.c
-rw-r--r--    1 sand     users        1738 2005-09-05 15:17 lab3_3.c
-rw-r--r--    1 sand     users        1616 2005-09-06 14:10 myshell.c
-rw-r--r--    1 sand     users           0 2005-09-05 15:12 outfile
-rw-r--r--    1 sand     users         702 2005-09-05 15:21 statusfile
```

Ledning: Basera arbetet på programmet angivet i listing 3.4 på sidan 51 i *Advanced Linux Programming*.) Du kan också, från stora kommunikationsövningen se till att använda konstruktionen `execlp("/bin/sh", "sh", "-c", commandstr ...` för att ha ett lätt sätt att kunna köra ett kommando som användaren skriver in.

En ledning till: Hur är det med kommandot `cd`? Hur kan du få det att fungera? Du kan läsa i *bash*-manualen om så kallade inbyggda kommandon (`man bash` gå ner till rubriken "`SHELL BUILTIN COMMANDS`", ett par tusen rader ner i manualsidan). Dessa är inte processer som anropas av skalet i vanlig mening. För att få kommandot `cd` att fungera i ditt shell behöver du använda kommandot `chdir()`, frågan är, ska du göra detta i en barnprocess som med alla andra kommandon?

Att fundera på: Vad ska vi egentligen tycka om `gets()`? Googla på nätet så får du se några intressanta synpunkter om `gets()`.

Teoriavsnitt (läs igenom före laborationen, skriv in och testa olika program särskilt programmen med `fork()`.): Advanced Linux Programming kap 1.2, 1.5, 2.1, 2.2, 3.1, 3.2, 3.3, 3.4.

Uppgift 2. *Egen server*

Tag nu skalet som du skrivit ovan och lägg in det i en client-serverapplikation. Servern ska skrivas som en demon som tar kommandon från klienten. Server och klient ska kommunicera via en *UNIX* domain socket. De kör förstås på samma maskin, än så länge, så vi ska i princip ha en applikation som liknar körningen av ett vanligt kommandoskal. Skriv också ett stopskript till din demon, alltså ett enkelt bashscript som läser in processid från demonen och skickar en kill-signal till demonen så

att den avslutas. Det är då viktigt att demonen, då den startar, lagrar sitt processid i en textfil som är läsbar för stopscriptet. Det är OK att skriva stopscriptet som ett vanligt *C*-program också, det viktigaste är att man inte ska behöva hålla reda på vilket processid demonen har.

För att kontrollera att demonen startar och avslutar och verkligen är en demon rekommenderas kommandot

```
ps -e -o pid,ppid,comm,sess
```

kolla att demonen startar och avlutas med hjälp av dina script, kolla att demonen har förälder 1 och kör i en egen session.

OBS: Det är viktigt att varje inloggning mot servern resulterar i en dialog med servern som är en självständig session som är oberoende av andra körande sessioner, det betyder att man ska kunna ha flera parallella inloggningar och att varje inloggning är en självständig klientbetjäning, det innebär till exempel att två inloggade klienter kan ha två olika arbetskataloger.

Uppgift 3. Byt nu ut *UNIX*-domainsocketen till en internet socket och gör laborationen distribuerad, dvs kör servern på en virtuell dator och kör en klient på en annan virtuell dator. För detta behöver du kunna köra två parallella virtuella datorer och de ska kunna kommunicera via ett internt lokalt nätverk i Virtualbox. Det är också viktigt att servern och klienten startas med kommandoradsargument, om du inte gjort det tidigare så gör det nu, alltså ta in strängvektorn `argv[]` och behandla starten av programmen (servern och klienten) med den.

Redovisning: Laborationerna redovisas inte i denna kurs. De examineras via en datortentamen. På datortentamen kommer uppgifter som kräver att man arbetat med laborationernas innehåll. För att laborationerna ska vara ett mått på hur väl man tagit till sig kursen är det viktigt att lösningar till laborationer hemlighålls. Ni får tala fritt om innehållet i övningar, men laborationernas lösningar behöver ni alltså hemlighålla för varandra. Vi kan alla tala fritt om det abstrakta innehållet i laborationerna, men aldrig titta på varandras kod och absolut aldrig flytta laborationskod från en studentdator till en annan. Vi kan också diskutera hur laborationerna ska fungera på seminarier och bufferttillfällen.