Inför datortentan II

Detta är andra dokumentet i en följd av dokument som med syfte att klargöra kursmål och examinationsformen på datortentan. I det här dokumentet presenteras ytterligare övningsuppgifter som ni bör göra inför övningar och seminarier i kursen, men allra senast förstås inför själva datortentan.

Genomgående behöver systemanropet

```
system("ps -o pid,ppid,pgid,sess,comm");
```

göras för att illustrera de olika situationer och släktförhållanden mellan processer som verkligen uppstår i era program. För att alla processer ska synas av detta anrop kan det vara nödvändigt att lägga in anrop till `sleep()` i de processer som skapas. Till exempel kommer barnet i koden

```
if(!fork())
{
  sleep(1);
  exit(0);
}
system("ps -o pid,ppid,pgid,sess,comm"); //anrop 1
wait(0);
system("ps -o pid,ppid,pgid,sess,comm"); //anrop 2
```

att synas säkert wid anrop 1 ovan. I anrop 2 så kommer inte barnprocessen att synas eftersom den avslutats då med `exit(0)` och eftersom föräldern gör `wait` innan anrop 2 så kommer inte barnet att synas vid anrop 2. Om vi inte har `sleep(1)` så kan det hända att barnprocessen inte hinner skapas innan föräldern gör anrop 1 så då syns inte barnet i vare sig anrop 1 eller anrop 2.

Här är en provkörning av ovanstående kod:

```
  PID  PPID  PGID  SESS COMMAND
 4399  3164  4399  4399 bash
 4466  4399  4466  4399 a.out
 4467  4466  4466  4399 a.out
 4468  4466  4466  4399 sh
 4469  4468  4466  4399 ps
  PID  PPID  PGID  SESS COMMAND
 4399  3164  4399  4399 bash
 4466  4399  4466  4399 a.out
 4470  4466  4466  4399 sh
 4471  4470  4466  4399 ps
```

som vi ser syns barnet som tydligen har processid 4467 vid första utskriften som kommer i repons till första aropet. Vid andra anropet är barnet borta.

Expanderad problemställning

Studera nedanstående tre program:

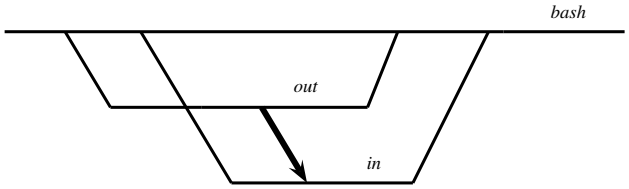
```
// Programnamn: in
main(int argc, char *argv[])
{
  int a;
  read(0,&a,sizeof(int));
  printf("%d\n",a);
}
```

```
// Programnamn: out
main(int argc, char *argv[])
{
  int a = atoi(argv[1]);
  write(1,&a,sizeof(int));
}
```

De här programmen skapar varsinn process. Den första (`in`) läser från sin stdin och skriver ut på skärmen (med `printf()`) det som kommer in. Processen som skapas när programemt `in` kör förväntar sig att få in ett heltal. Den andra (`out`) skriver ut i textform det som den får in som kommandoradsargument. Dessa båda program är mycket liknande de i avsnittet om relationer mellan körande processer. En körning vid kommandoradsprompten ger följande utskrift:

```
$ ./out 10 | ./in
10
$
```

En enkel utskrift av ett tal. Vi kan rita följande tidsdiagram för att beskriva ovanstående:



Överst ser vi föräldraprocessen som som alltid vid kommandoradsanrop av det här slaget ä skalet som härbärgerar anropet, det är *bash* själv. Undertill ser vi två barnprocesser, den ena benämns *out* och den andra benämns *in*. Den tjocka pilen från *out* till *in* representerar den pipe som skapas av kommandoraden.

Det vi ska göra med detta enkla första exempel är att ta in de båda processerna *in* och *out* som barn till en ny förälder som kallas *combine*. Vi ska låta *combine* skapa båda barnen och den mellanliggande pipen och även göra anropet

```
system("ps -o pid,ppid,pgid,sess,comm");
```

för att beskriva situationen. Vi ser på hur *combine* kan skrivas:

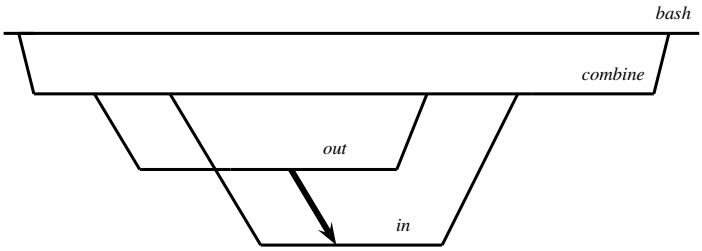
```
// Programnamn: combine
main()
{
  int fds[2]; pipe(fds);

  if(!fork())
  {
    close(1); dup(fds[1]); close(fds[0]); close(fds[1]);
    sleep(1);
    execlp("./out", "./out", "10", NULL);
  }

  if(!fork())
  {
    close(0); dup(fds[0]); close(fds[0]); close(fds[1]);
    sleep(1);
    execlp("./in", "./in", NULL);
  }

  system("ps -o pid,ppid,pgid,sess,comm");

  close(fds[0]); close(fds[1]);
  wait(0); wait(0);
}
```

Om vi provkör detta program så får vi utskriften

```
 PID  PPID  PGID  SESS COMMAND
 4399  3164  4399  4399 bash
 4884  4399  4884  4399 combine
 4885  4884  4884  4399 combine
 4886  4884  4884  4399 combine
 4887  4884  4884  4399 sh
 4888  4887  4884  4399 ps
10
```

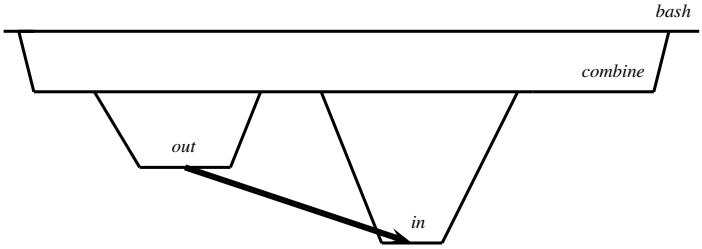
en enkel utskrift av talet 10 alltså. Vi ser att processen *combine* har två barn, det ena kommer att vara *in* och det andra kommer att vara *out* även om de båda benämns *combine* av operativsystemet.

Ett tidsdiagram kan se ut så här

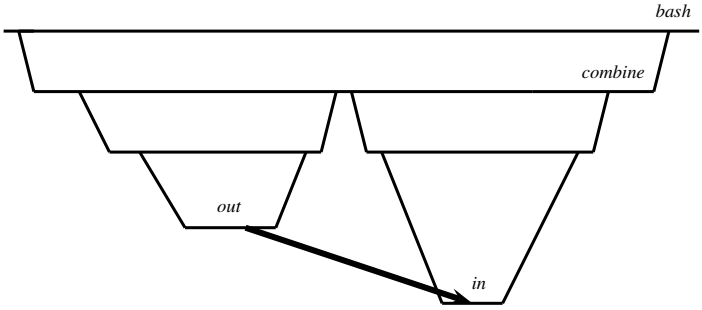


alltså precis samma grundstruktur som det tidigare, fast vi har lagt allting i en barnprocess till *bash* som heter *combine* som i sin tur skapar de båda barnprocesserna *in* och *out*. Med detta program ska vi skapa ett antal övningar:

1. Modifiera programmet så att barnprocesserna inte kör parallellt utan att den ena avslutas innan den andra påbörjas. Kontrollera att processerna inte kör parallellt genom att göra anropet `system("ps -o pid,ppid,pgid,sess,comm")` vid väl valda tidpunkter. Programmet kan då beskrivas av följande tidsdiagram:



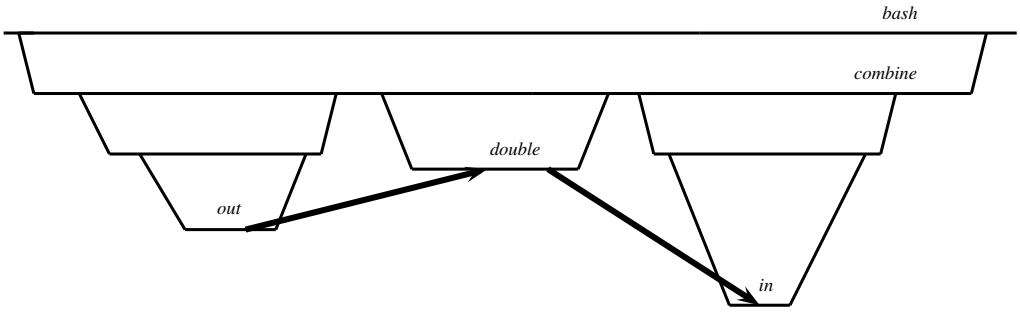
2. Låt nu processerna *in* och *out* köras som barnbarn (i form av *kusiner*) till combine istället, ni ska alltså körningen beskrivas av följande tidsdiagram:



3. Låt nu *combine* skapa ytterligare ett barn som går in mitt emellan barnbarnen och som dubblerar det som skickas, tag koden frå processen som heter `double` och som gavs på kursmötet om relationer mellan processer. *Ledning:* för att anropa den krävs användning av `execlp` och anropet blir

   ```
   execlp("./double","./double",NULL);
   ```

   och då måste programkoden hörande till `double` ligga i arbetskatalogen för *P1*. Vi får då följande tidsdiagram



   (Här behövs alltså ytterligare en pipe.)
4. Skriv nu om programmet så att alla barn och barnbarn till *combine* kör parallellt. Verfiera med ett bra placerat anrop till `system(ps ... )` att barnen och barnbarnen verkligen kör parallellt. Det kan även behövas välplacerade anrop till `sleep()` för att säkert se att det fungerar.
5. Skapa en egen variant av barn och barnbarn som kommunicerar med varandra på detta sätt. Istället för att skicka talet 10, låt processerna istället skicka ett processid av det första barnet som skickar någonting. (*Ledning: För att göra om ett processid, som är ett heltal, kan ni använda anropet* `sprintf(str,"%d",pid);` *för att få in heltalet som är processid:t i en sträng.*)