Seminarium II

Andra seminariet behandlar speciella övningsuppgifter som gavs ut på kurswebben tidigare. Programmen som presenteras är förlag till lösningar på övningsuppgifterna. Genomgående används systemanropet

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

för att kontrollera att de olika situationer och släktförhållanden mellan processer som verkligen uppstår i programmen som som det krävdes i uppgiftsformuleringarna. Anrop till `sleep()` har ibland lagts in i de processer som skapats.

Grundprogrammen

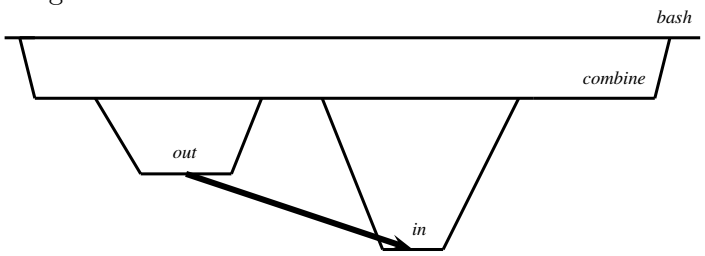
Programmen *in* och *out* (givna nedan)

```
// 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));
}
```

används i ett program som benämns *combine* (som skapar barn som omvandlas till över av *in* och *out*) är i centrum och det är kring uppgifter kring dessa program som löses nedan.

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:



Lösningsförslag: Programmet *combine* innehåller inga kontroller av systemanropen som utförs, det var lättare att genomföra utvecklingen av lösningen när dessa lades till. Här är lösningen:

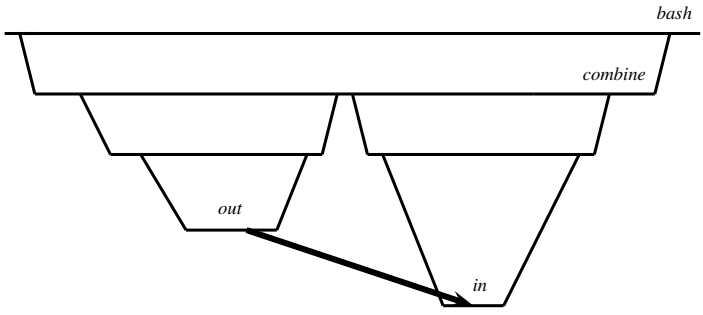
```
// Programnamn: uppg1
main()
{
  int fds[2], test; pipe(fds);

  if(!fork())
  {
    test = close(1); assert(test==0);
    test = dup(fds[1]); assert(test==1);
    test = close(fds[0]); assert(test==0);
    test = close(fds[1]); assert(test==0);
    sleep(1);
```

```
    execlp("./out", "./out", "10", NULL);
    fprintf(stderr,"execlp did not work\n");
  }
  system("ps -o pid,ppid,pgid,sess,comm");
  wait(0);

  if(!fork())
  {
    test = close(0); assert(test==0);
    test = dup(fds[0]); assert(test==0);
    test = close(fds[0]); assert(test==0);
    test = close(fds[1]); assert(test==0);
    sleep(1);
    execlp("./in", "./in", NULL);
    fprintf(stderr,"execlp did not work\n");
  }

  test=close(fds[0]); assert(test==0);
  test=close(fds[1]); assert(test==0);
  system("ps -o pid,ppid,pgid,sess,comm");

  wait(0);

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

En testkörning har följande utseende:

```
$ ./uppg1
  PID  PPID  PGID  SESS COMMAND
 4316  3577  4316  4316 bash
 4362  4316  4362  4316 uppg1
 4363  4362  4362  4316 uppg1
 4364  4362  4362  4316 sh
 4365  4364  4362  4316 ps
  PID  PPID  PGID  SESS COMMAND
 4316  3577  4316  4316 bash
 4362  4316  4362  4316 uppg1
 4366  4362  4362  4316 uppg1
 4367  4362  4362  4316 sh
 4368  4367  4362  4316 ps
10
$
```

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:



Lösningsförslag:

```
// Programnamn: uppg2
main()
{
  int fds[2], test; pipe(fds);
```

```
  if(!fork())
  {
    if(!fork())
    {
      test = close(1); assert(test==0);
      test = dup(fds[1]); assert(test==1);
      test = close(fds[0]); assert(test==0);
      test = close(fds[1]); assert(test==0);
      sleep(1);
      execlp("./out", "./out", "10", NULL);
      fprintf(stderr,"execlp did not work\n");
    }
    test = close(fds[0]); assert(test==0);
    test = close(fds[1]); assert(test==0);
    wait(0);
    exit(0);
  }
  system("ps -o pid,ppid,pgid,sess,comm"); wait(0);

  if(!fork())
  {
    if(!fork())
    {
      test = close(0); assert(test==0);
      test = dup(fds[0]); assert(test==0);
      test = close(fds[0]); assert(test==0);
      test = close(fds[1]); assert(test==0);
      sleep(1);
      execlp("./in", "./in", NULL);
      fprintf(stderr,"execlp did not work\n");
    }
    test = close(fds[0]); assert(test==0);
    test = close(fds[1]); assert(test==0);
    wait(0);
    exit(0);
  }

  test=close(fds[0]); assert(test==0);
  test=close(fds[1]); assert(test==0);
  system("ps -o pid,ppid,pgid,sess,comm"); wait(0);
}
```

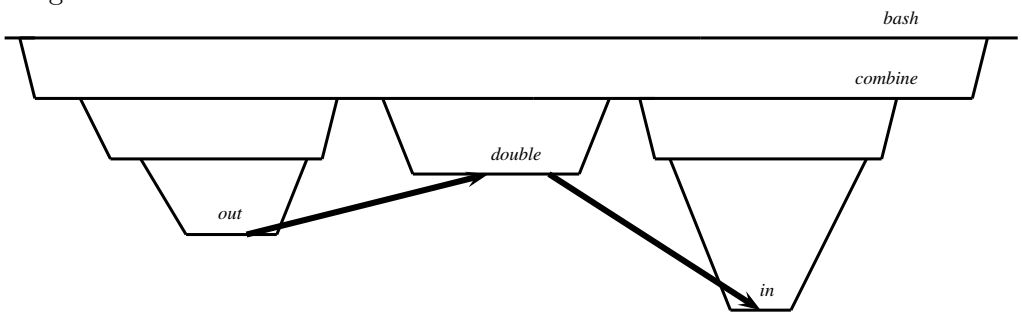
Och en testkörning:

```
$ ./uppg2
  PID  PPID  PGID  SESS COMMAND
 3674  3577  3674  3674 bash
 5956  3674  5956  3674 uppg2
 5957  5956  5956  3674 uppg2
 5958  5956  5956  3674 sh
 5959  5957  5956  3674 uppg2
 5960  5958  5956  3674 ps
  PID  PPID  PGID  SESS COMMAND
 3674  3577  3674  3674 bash
 5956  3674  5956  3674 uppg2
 5961  5956  5956  3674 uppg2
 5962  5956  5956  3674 sh
 5963  5961  5956  3674 uppg2
 5964  5962  5956  3674 ps
10
$
```

3. Låt nu *combine* skapa ytterligare ett barn som går in mitt emellan barnbarnen och som dubblerar det som skickas, tag koden från 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



Lösningsförslag:

```
//Programnamn: uppg3
main()
{
  int fds1[2], fds2[2], test;
  pipe(fds1); pipe(fds2);

  if(!fork())
  {
    test=close(fds2[0]); assert(test==0);
    test=close(fds2[1]); assert(test==0);
    if(!fork())
    {
      test = close(1); assert(test==0);
      test = dup(fds1[1]); assert(test==1);
      test = close(fds1[0]); assert(test==0);
      test = close(fds1[1]); assert(test==0);
      sleep(1);
      execlp("./out", "./out", "10", NULL);
      fprintf(stderr,"execlp did not work\n");
      exit(1);
    }
    test = close(fds1[0]); assert(test==0);
    test = close(fds1[1]); assert(test==0);
    wait(0);
    exit(0);
  }
  system("ps -o pid,ppid,pgid,sess,comm");
  wait(0);

  if(!fork())
  {
    test=close(fds1[1]); assert(test==0);
    test=close(fds2[0]), assert(test==0);

    test=close(0); assert(test==0);
    test=dup(fds1[0]); assert(test==0);
    test=close(fds1[0]); assert(test==0);

    test=close(1); assert(test==0);
    test=dup(fds2[1]); assert(test=1);
    test=close(fds2[1]); assert(test==0);

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

```
    fprintf(stderr,"execlp did not work.\n");
    exit(1);
  }
  system("ps -o pid,ppid,pgid,sess,comm");
  wait(0);

  if(!fork())
  {
    test=close(fds1[0]); assert(test==0);
    test=close(fds1[1]); assert(test==0);
    if(!fork())
    {
      test = close(0); assert(test==0);
      test = dup(fds2[0]); assert(test==0);
      test = close(fds2[0]); assert(test==0);
      test = close(fds2[1]); assert(test==0);
      sleep(1);
      execlp("./in", "./in", NULL);
      fprintf(stderr,"execlp did not work\n");
      exit(1);
    }
    test = close(fds2[0]); assert(test==0);
    test = close(fds2[1]); assert(test==0);
    wait(0);
    exit(0);
  }

  test=close(fds1[0]); assert(test==0);
  test=close(fds1[1]); assert(test==0);
  test=close(fds2[0]); assert(test==0);
  test=close(fds2[1]); assert(test==0);
  system("ps -o pid,ppid,pgid,sess,comm");
  wait(0);
}
```

Och en testkörning:

```
$ ./uppg3
  PID  PPID  PGID  SESS COMMAND
 3674  3577  3674  3674 bash
 5799  3674  5799  3674 uppg3
 5800  5799  5799  3674 uppg3
 5801  5799  5799  3674 sh
 5802  5800  5799  3674 uppg3
 5803  5801  5799  3674 ps
  PID  PPID  PGID  SESS COMMAND
 3674  3577  3674  3674 bash
 5799  3674  5799  3674 uppg3
 5804  5799  5799  3674 double <defunct>
 5805  5799  5799  3674 sh
 5806  5805  5799  3674 ps
  PID  PPID  PGID  SESS COMMAND
 3674  3577  3674  3674 bash
 5799  3674  5799  3674 uppg3
 5807  5799  5799  3674 uppg3
 5808  5799  5799  3674 sh
 5809  5807  5799  3674 uppg3
 5810  5808  5799  3674 ps
20
$
```

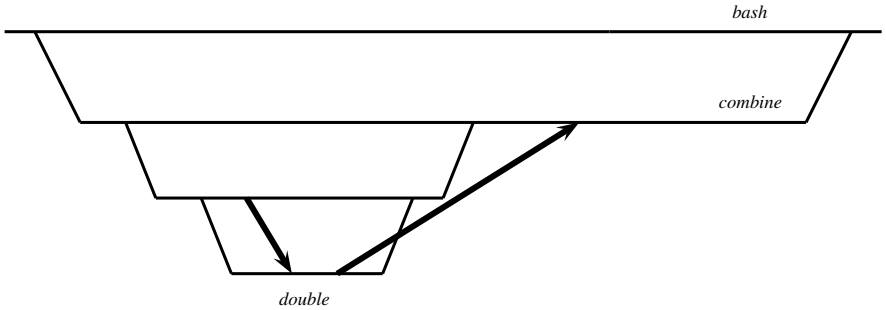
4. Skriv nu om programmet så att alla barn och barnbarn till *combine* kör parallellt. Verfiera med ett välplacerat anrop till `system(ps ... )` att barnen och barnbarnen verkligen kör parallellt. Det kan även behövas välplacerade anrop till `sleep()` för att säkert se att det fungerar.
Lösningsförslag:

```
// Programnamn: uppg4
main()
{
  int fds1[2], fds2[2], test;
  pipe(fds1); pipe(fds2);

  if(!fork())
  {
    test=close(fds2[0]); assert(test==0);
    test=close(fds2[1]); assert(test==0);
    if(!fork())
    {
      test = close(1); assert(test==0);
      test = dup(fds1[1]); assert(test==1);
      test = close(fds1[0]); assert(test==0);
      test = close(fds1[1]); assert(test==0);
      sleep(1);
      execlp("./out", "./out", "10", NULL);
      fprintf(stderr,"execlp did not work\n");
      exit(1);
    }
    test = close(fds1[0]); assert(test==0);
    test = close(fds1[1]); assert(test==0);
    wait(0);
    exit(0);
  }
  //system("ps -o pid,ppid,pgid,sess,comm");
  //wait(0);

  if(!fork())
  {
    test=close(fds1[1]); assert(test==0);
    test=close(fds2[0]), assert(test==0);

    test=close(0); assert(test==0);
    test=dup(fds1[0]); assert(test==0);
    test=close(fds1[0]); assert(test==0);

    test=close(1); assert(test==0);
    test=dup(fds2[1]); assert(test=1);
    test=close(fds2[1]); assert(test==0);

    execlp("./double","./double", NULL);
    fprintf(stderr,"execlp did not work.\n");
    exit(1);
  }
  //system("ps -o pid,ppid,pgid,sess,comm");
  //wait(0);

  if(!fork())
  {
    test=close(fds1[0]); assert(test==0);
    test=close(fds1[1]); assert(test==0);
    if(!fork())
    {
```

```
      test = close(0); assert(test==0);
      test = dup(fds2[0]); assert(test==0);
      test = close(fds2[0]); assert(test==0);
      test = close(fds2[1]); assert(test==0);
      sleep(1);
      execlp("./in", "./in", NULL);
      fprintf(stderr,"execlp did not work\n");
      exit(1);
    }
    test = close(fds2[0]); assert(test==0);
    test = close(fds2[1]); assert(test==0);
    wait(0);
    exit(0);
  }

  test=close(fds1[0]); assert(test==0);
  test=close(fds1[1]); assert(test==0);
  test=close(fds2[0]); assert(test==0);
  test=close(fds2[1]); assert(test==0);
  system("ps -o pid,ppid,pgid,sess,comm");
  wait(0);
  wait(0);
  wait(0);
}
```

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. Detta behövs eftersom* `execlp` *kräver strängar som parametrar.*)

   Vi ger en liten lösning till detta som också utgör ett förslag på vad som kan komma att tas med i tillåtna hjlpmedel. Vi skriver ett program som illustreras av följande tidsdiagram:



   Ett program/process som vi kallar *combine* som skapar ett barn som skickar sitt processid via en pipe till *double* som dubblerar och skickar det tillbaka till *combine* som lser in det det få och skriver ut resultatet.

   Koden ser ut så här:

```
// Programnamn: uppg5
main()
{
  int fds1[2], fds2[2], test, a;
  pipe(fds1); pipe(fds2);

  if(!fork())
  {
    char pidstr[20];
    if(!fork())
    {
      test = close(0); assert(test==0);
      test = dup(fds1[0]); assert(test==0);
```

```
        test = close(fds1[0]); assert(test==0);
        test = close(fds1[1]); assert(test==0);
        test = close(1); assert(test==0);
        test = dup(fds2[1]); assert(test==1);
        test = close(fds2[1]); assert(test==0);
        test = close(fds2[0]); assert(test==0);
        sleep(1);
        execlp("./double", "./double", NULL);
        fprintf(stderr,"execlp did not work\n");
        exit(1);
      }

      test = close(1); assert(test==0);
      test = dup(fds1[1]); assert(test==1);
      test = close(fds1[1]); assert(test==0);
      test = close(fds1[0]); assert(test==0);
      test = close(fds2[0]); assert(test==0);
      test = close(fds2[1]); assert(test==0);
      sleep(1);
      sprintf(pidstr,"%d",getpid());
      execlp("./out","./out",pidstr,NULL);
      fprintf(stderr,"execlp did not work\n");
      exit(1);
    }
    system("ps -o pid,ppid,pgid,sess,comm");

    read(fds2[0],&a,sizeof(int)); printf("pid * 2: %d.\n", a);

    wait(0);
  }
```

och en testkörning ser ut så här:

```
$ ./uppg5
  PID  PPID  PGID  SESS COMMAND
 3674  3577  3674  3674 bash
 6260  3674  6260  3674 uppg5
 6261  6260  6260  3674 uppg5
 6262  6260  6260  3674 sh
 6263  6261  6260  3674 uppg5
 6264  6262  6260  3674 ps
pid * 2: 12522.
$
```

Exempelprogram som kommer att finnas med på tentan

På förra seminariet lades upp en lista på exempelkod som jag föreslår ska vara tillåtna hjälpmedel på tentamen, vi utökar ni listan på tillåtna hjälpmedel/exempelkoder med ovanstående och då får vi följande lista (som ska utökas ännu mer):

Användbara systemanrop.

```
fork() exit() wait() getpid() getppid() execl() execlp() pipe() open() close()
```

Hitta en includefil (och mer information) för ett systemanrop.

```
$ man <systemanropet>
```

Skapa en parallell process.

```
pid_t pid;
pid = fork();
switch(pid)
{
  case -1: fprintf(stderr,"Fork failed.\n"); exit(1);
  case 0: childcode(); exit(0); //Körs i en parallell barnprocess.
```

```
    default: parentcode1(); wait(0); //Körs parallellt med barnprocessen.
  }
  parentcode2(); //Körs efter barnprocessen avslutat.
```

Skapa ett barn och ett barnbarn där barnet skickar sitt processid till barnbarnet som i sin tur skickar det multiplicerat med 2 till föräldern.

```
// Programnamn: uppg5
main()
{
  int fds1[2], fds2[2], test, a;
  pipe(fds1); pipe(fds2);

  if(!fork())
  {
    char pidstr[20];
    if(!fork())
    {
      test = close(0); assert(test==0);
      test = dup(fds1[0]); assert(test==0);
      test = close(fds1[0]); assert(test==0);
      test = close(fds1[1]); assert(test==0);
      test = close(1); assert(test==0);
      test = dup(fds2[1]); assert(test==1);
      test = close(fds2[1]); assert(test==0);
      test = close(fds2[0]); assert(test==0);
      sleep(1);
      execlp("./double", "./double", NULL);
      fprintf(stderr,"execlp did not work\n");
      exit(1);
    }

    test = close(1); assert(test==0);
    test = dup(fds1[1]); assert(test==1);
    test = close(fds1[1]); assert(test==0);
    test = close(fds1[0]); assert(test==0);
    test = close(fds2[0]); assert(test==0);
    test = close(fds2[1]); assert(test==0);
    sleep(1);
    sprintf(pidstr,"%d",getpid());
    execlp("./out","./out",pidstr,NULL);
    fprintf(stderr,"execlp did not work\n");
    exit(1);
  }
  system("ps -o pid,ppid,pgid,sess,comm");

  read(fds2[0],&a,sizeof(int)); printf("pid * 2: %d.\n", a);

  wait(0);
}
```