Övningar på trådar

Det kommer att visa sig att det är mycket enklare att programmera med trådsom pratar med pipes/sockets ar när man en gång bemästrat IPC med processer och pipes/sockets. Vi ska dock formulera några övningsuppgifter som illustrerar detta. Vi kommer att utgå från följande grundläggande program som skapar två trådar och som fungerar med parametrar:

```
#include <pthread.h>
#include <stdio.h>
/* Parameters to print_function. */
struct char_print_parms
{
   /* The character to print. */
   char character;
   /* The number of times to print it.        */
   int count;
};
/* Prints a number of characters to stderr, as given by PARAMETERS,
    which is a pointer to a struct char_print_parms. */
void* char_print (void* parameters)
{
   /* Cast the cookie pointer to the right type. */
   struct char_print_parms* p = (struct char_print_parms*) parameters;
   int i;
   for (i = 0; i < p->count; ++i)
     fputc (p->character, stderr);
   return NULL;
}
/* The main program.     */
int main ()
{
   pthread_t thread1_id;
   pthread_t thread2_id;
   struct char_print_parms thread1_args;
   struct char_print_parms thread2_args;

   /* Create a new thread to print 30000 x's. */
   thread1_args.character = 'x';
   thread1_args.count = 30000;
   pthread_create (&thread1_id, NULL, &char_print, &thread1_args);
   /* Create a new thread to print 20000 o's. */
   thread2_args.character = 'o';
   thread2_args.count = 20000;
   pthread_create (&thread2_id, NULL, &char_print, &thread2_args);
   /* Make sure    the first thread has finished. */
   pthread_join    (thread1_id, NULL);
   /* Make sure    the second thread has finished. */
   pthread_join    (thread2_id, NULL);
   /* Now we can safely return.        */
   return 0;
}
```

Vi studerade detta program när vi först började med trådar, det skapar två trådar som vardera skriver ut på skärmen varsinn symbol. Vi ska modifiera detta program så att trådarna kan styras mer av huvudprogrammet

och vi ska styra trådarna med pipar istället. Övningen kommer att illustrera hur trådarna finns i samma adressrymd och därmed bara kan använda piparna direkt utan att stänga oanvända läs- eller skrivändar och liknande.

Övning 1. Skriv om ovanstående program så att endast en tråd startas och istället för att skicka styrparametrar via structen, skicka tre saker via en pipe: 1. Tecken att skriva ut. 2. Antal tecken att skriva ut. 3. En antal sekunder som tråden ska pausa innan den börjar skriva ut. Låt även tråden skriva ut ett tecken i sekunden istället för att bara vräka ut tecken. (Det blir lite mer överskådligt då.

Lösning. Lösningen illustreras i 4 steg som benämns ass1_stage1.c, ass1_stage2.c, ass1_stage3.c respektive ass1_stage4.c. I första stadiet skrivs bara ett program som skickar in en fildeskriptor i en tråd, det enda tråden gör är att skriva ut fildeskriptorn som blir 3 (= läsänden på en pipe som skapas av huvudtråden). Det ser ut så här:

```
#include <pthread.h>
#include <stdio.h>

int p1[2];

void* char_print (void* parameter)
{
  int fds; fds = *((int *)parameter);
  int i;

  printf("fds: %d.\n", fds);

  return NULL;
}
* The main program.     */
int main ()
{
  pthread_t thread1_id;
  pipe(p1);

  pthread_create (&thread1_id, NULL, &char_print, &(p1[0]));
  /* Make sure    the first thread has finished. */
  pthread_join    (thread1_id, NULL);

  return 0;
}
```

Observera att tråden bara behöver fildeskriptorn, som ju är ett vanligt heltal. Vid systemanropet `pipe(p1)` så skapas pipen och inga ändar behöver alltså stängas efter det anropet som vi normalt behöver göra då parallella processer skapas. Här kan vi alltå bara använda deskriptorn direkt. En testkörning resulterar i att talet 3 skrivs ut.

Vi gå vidare till stage2 där programmet utvidgats lite till att vi skickar något över pipen, vi läser in ett testtal, programmet ser ut så haär:

```
#include <pthread.h>
#include <stdio.h>

int p1[2];

void* char_print (void* parameter)
{
  int fds; fds = *((int *)parameter);
  int i, x;

  read(fds, &x, sizeof(int));
  printf("Got: %d.\n", x);

  return NULL;
}
```

```
/* The main program.     */
int main ()
{
  int x = 47;
  pthread_t thread1_id;
  pipe(p1);

  pthread_create (&thread1_id, NULL, &char_print, &(p1[0]));
  write(p1[1], &x, sizeof(int));
  pthread_join(thread1_id, NULL);
  return 0;
}
```

Detta program skriver ut `Got:  47`, vilket indikerar att talet 47 skickats från huvudprogrammet till tråden.

Stadium 3 (stage3) innebär att vi skickar mycket mer information via pipen, vi skickar, tecken, antal gången som det ska skrivas ut och antal sekunder som tråden ska ska sova innan utskrifterna påbörjas. Det ser ut så här:

```
#include <pthread.h>
#include <stdio.h>

int p1[2];

void* char_print (void* parameter)
{
  int fds; fds = *((int *)parameter);
  char character; int count; int pause;

  read(fds, &character, sizeof(char));
  read(fds, &count, sizeof(int));
  read(fds, &pause, sizeof(int));

  printf("Got: %c, %d, %d.\", character, count, pause);


  //for (i = 0; i < p->count; ++i)
  //  fputc (p->character, stderr);


  return NULL;
}
/* The main program.     */
int main ()
{
  char character;
  int count, pause;
  pthread_t thread1_id;
  pipe(p1);
  pthread_create (&thread1_id, NULL, &char_print, &(p1[0]));
  printf("Tecken: "); scanf("%c", &character);
  write(p1[1], &character, sizeof(char));
  printf("Antal: "); scanf("%d", &count);
  write(p1[1], &count, sizeof(int));
  printf("Paus: "); scanf("%d", &pause);
  write(p1[1], &pause, sizeof(int));

  pthread_join(thread1_id, NULL);
  return 0;
}
```

testkörningen bestå i att användaren matar in de tre uppgifterna och sedan skriver tråden ut att dessa kommit in.

Detta program utvidgas sedan lätt till slutresultatet:

```
#include <pthread.h>
#include <stdio.h>

int p1[2];

void* char_print (void* parameter)
{
  int i, fds; fds = *((int *)parameter);
  char character; int count; int pause;

  while(read(fds, &character, sizeof(char)))
  {
    read(fds, &count, sizeof(int));
    read(fds, &pause, sizeof(int));

    sleep(pause);
    for (i = 0; i < count; ++i)
    {
      sleep(1);
      fputc (character, stderr);
    }
  }

  return NULL;
}
/* The main program.     */
int main ()
{
  char character;
  int count, pause;
  pthread_t thread1_id;
  pipe(p1);

  pthread_create (&thread1_id, NULL, &char_print, &(p1[0]));
  printf("Tecken: "); scanf("%c", &character);
  write(p1[1], &character, sizeof(char));
  printf("Antal: "); scanf("%d", &count);
  write(p1[1], &count, sizeof(int));
  printf("Paus: "); scanf("%d", &pause);
  write(p1[1], &pause, sizeof(int));

  pthread_join(thread1_id, NULL);

  return 0;
}
```

Denna trådläser från pipen men har en möjlighet att uppfatta att pipen stängs, vi kommer att använda det i senare versioner.

Övning 2. Utvidga programmet ovan så att två trådar startar och skriver ut två tecken samtidigt. Du måste då synkronisera användningen av pipen så att inte båda trådarna läser från den på en gång. Det är ok att gå över till att ange asciikoder för tecken så slipper vi problem med inmatning av text/siffror.

Lösning.

```
#include <pthread.h>
#include <stdio.h>
```

```
int p1[2];
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;

void* char_print (void* parameter)
{
  int i, fds; fds = *((int *)parameter);
  int character; int count; int pause;

  while(1)
  {
    pthread_mutex_lock(&m);
    read(fds, &count, sizeof(int));
    if(!count)break;
    read(fds, &character, sizeof(int));
    read(fds, &pause, sizeof(int));
    pthread_mutex_unlock(&m);

    sleep(pause);
    for (i = 0; i < count; ++i)
    {
      sleep(1);
      fputc (character, stderr);
    }
  }
  pthread_mutex_unlock(&m);

  return NULL;
}
/* The main program.     */
int main () {
  char character;
  int count, pause; pipe(p1);

  pthread_t thread1_id;
  pthread_create (&thread1_id, NULL, &char_print, &(p1[0]));
  pthread_t thread2_id;
  pthread_create (&thread2_id, NULL, &char_print, &(p1[0]));

  printf("Antal (0 fr att avsluta): "); scanf("%d", &count);
  write(p1[1], &count, sizeof(int));
  printf("Tecken: "); scanf("%d", &character);
  write(p1[1], &character, sizeof(char));
  printf("Paus: "); scanf("%d", &pause);
  write(p1[1], &pause, sizeof(int));

  printf("Antal (0 fr att avsluta): "); scanf("%d", &count);
  write(p1[1], &count, sizeof(int));
  printf("Tecken: "); scanf("%d", &character);
  write(p1[1], &character, sizeof(char));

  count=0; write(p1[1],&count,sizeof(int));
  write(p1[1],&count,sizeof(int));
  printf("\n\n");

  pthread_join(thread1_id, NULL);  pthread_join(thread2_id, NULL);
  return 0;
}
```