Vorbereitung

- Öffnen Sie ein Terminalfenster (Konsole, xterm etc.) und erstellen Sie ein neues Verzeichnis `bspraktikum`, in das Sie dann hinein wechseln.
- Laden Sie das Archiv mit den Aufgaben-Dateien herunter, das geht mit `wget http://hm.hgesser.de/bs-ss2011/prakt/prakt04.tgz`
- Entpacken Sie das Archiv (`tar xzf prakt04.tgz`) und wechseln Sie in das Unterverzeichnis `prakt04`

12. Posix-Threads (2 Punkte)

Betrachten Sie das Programm `thread-beispiel.c`, das Sie mit dem Kommando

```
gcc -lpthread -o thread-beispiel thread-beispiel.c
```

übersetzen können. (Über die Option `-lpthread` binden Sie die *pthread*-Bibliothek zum Programm hinzu.)

a) Führen Sie das Programm aus. Notieren Sie, welche Informationen es ausgibt (nicht den vollen Text, nur eine Beschreibung).

b) Öffnen Sie ein zweites Terminalfenster und beobachten Sie dort die Prozess- und Thread-Liste mit dem Befehl

```
ps -Lf -C thread-beispiel
```

Sie können für eine regelmäßige Anzeige sorgen, indem Sie das Kommando mit `watch` ausführen; der richtige Befehl heißt dann

```
watch -n1 "ps -Lf -C thread-beispiel"
```

In der Ausgabe sehen Sie für (bis zu) drei Threads die Prozess-ID (PID), Parent Process ID (PPID) und Thread ID (LWP, Light Weight Process ID). Wenn die Ausgabe zu schnell verschwindet, ändern Sie in der Definition von `thread_function1` und `thread_function2` die Bedingung `i<10` in (z. B.) `i<30`.
Stellen Sie fest, welche Prozess-IDs in der Spalte PPID eingetragen sind – was sagt Ihnen das über Vater-Sohn-Verhältnisse zwischen dem „Haupt-Thread“ und den von ihm erzeugten Threads?

c) Öffnen Sie ggf. noch ein drittes Terminalfenster und schicken Sie versuchsweise einem der Threads (unter Angabe seiner Thread-ID aus der Spalte LWP!) ein STOP-Signal (`kill -STOP` *`LWPID`*).
Was geschieht dann mit dem Thread, dem Sie das Signal schicken, und was geschieht mit den übrigen Threads?
Schicken Sie anschließend einem anderen (!) Thread das CONT(inue)-Signal:

```
kill -CONT LWPID
```

Was geschieht dann mit den Threads?

d) Passen Sie das Programm so an, dass die `main`-Funktion die Threads nicht mit `pthread_join()` einsammelt, sondern direkt mit `exit()` verlässt. Entfernen Sie auch das letzte `sleep()` nach dem zweiten `pthread_create()`-Aufruf. Prüfen Sie, ob diese Änderung Auswirkungen auf die in separaten Threads laufenden Funktionen hat – erklären Sie Ihre Beobachtung.

13. Globale Variable in den Threads (3 Punkte)

Die Datei `zwei-threads.c` ähnelt der Datei `thread-beispiel.c`, enthält aber nur ein Gerüst für den Start der beiden Threads, außerdem fehlen die Verzögerungen. Dafür wird am Anfang eine globale Integer-Variable `i` definiert.

a) Füllen Sie die beiden Thread-Funktionen mit Inhalt: Eine der Funktionen soll die globale Variable initialisieren und dann zwei Sekunden warten (`sleep`); die andere soll zunächst kurz warten und dann den Wert der Variable ausgeben:
`printf("i=%d\n", i);`
Welchen Wert hatten Sie der Variable zugewiesen und welcher wird ausgegeben? Was sagt Ihnen das über die Speichernutzung der Threads?

b) Passen Sie Ihr Programm an: Vor den Thread-Definitionen initialisieren Sie die Variable bereits auf 0 (`int i=0;`). Im ersten Thread soll nun 1000 mal der Wert 1 addiert werden, im zweiten Thread 1000 mal der Wert 1 abgezogen werden – geben Sie im Hauptprogramm (vor `exit(0);`) den finalen Wert von `i` aus. Dieser sollte 0 sein. Wenn Sie das Programm starten, erhalten Sie mit großer Wahrscheinlichkeit auch den Wert 0. Lassen Sie es über eine Schleife ca. 500 bis 1500 mal laufen und prüfen Sie die Ausgaben. (Tipp: Lesen Sie die Manpage zu `sort`, insbesondere zur Option `-u`.) Wenn Sie als Ergebnis in jedem Durchlauf 0 erhalten haben, ändern Sie in beiden Schleifen die Anzahl von 1000 auf 10.000.
Eventuell erhalten Sie nicht immer den Wert 0. Warum? (Diese Frage sollen Sie auch dann zu beantworten versuchen, wenn Sie immer 0 erhalten.)

```
# zwei-threads.c

#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>  /* printf() */

int i;

void *thread_function1(void *arg) {  return 0;  }

void *thread_function2(void *arg) {  return 0;  }

int main(void) {
  pthread_t mythread1;
  pthread_t mythread2;
  /* Threads erzeugen */
  if ( pthread_create( &mythread1, NULL, thread_function1, NULL) ) {
    printf("Fehler bei Thread-Erzeugung."); abort();
  }
  if ( pthread_create( &mythread2, NULL, thread_function2, NULL) ) {
    printf("Fehler bei Thread-Erzeugung ."); abort();
  }
  /* Threads wieder einsammeln */
  if ( pthread_join ( mythread1, NULL ) ) {
    printf("Fehler beim Join."); abort();
  }
  if ( pthread_join ( mythread2, NULL ) ) {
    printf("Fehler beim Join."); abort();
  }
  exit(0);
}
```