Vorbereitung

- Booten Sie den Rechner unter Linux, melden Sie sich an und öffnen Sie ein Terminalfenster (`gnome-terminal`, `xterm` etc.).
- Die Dateien zum heutigen Praktikumstermin laden Sie von der Vorlesungsseite herunter: `wget http://fhm.hgesser.de/bs-ss2008/prakt07.tgz`
- Entpacken Sie das Archiv (`tar xzf prakt07.tgz`) und wechseln Sie in das neue Unterverzeichnis `prakt07`.

1. POSIX-Threads und Synchronisation in C

Unter *http://www.thinkbrown.com/programming/pthreads.pdf* (Kopie: *http://fhm.hgesser.de/bs-ss2008/pthreads.pdf*) finden Sie eine Einführung in die Programmierung mit POSIX-Threads (pthreads). Lesen Sie darin die Seiten 1-5 (ohne den Anfang von Absatz 4). Das Beispielprogramm, das dort auf Seite 4/5 abgedruckt ist, läuft unter Linux nicht; eine modifizierte Version liegt im Archiv als `reader-writer.c`:

```
#include <pthread.h>
#include <stdio.h>                      // fuer printf()
#include <unistd.h>                     // fuer usleep()

char buffer;                            // Buffer der Groesse 1
int buffer_has_item = 0;                // Buffer am Anfang leer
pthread_mutex_t mutex;                  // Mutex fuer Zugriff auf Buffer

int make_new_item() { return 0; };      // neues Item erzeugen
void consume_item (int i) { return; }; // Item verbrauchen

void writer_function(void) {            // Erzeuger
  while(1) {
    pthread_mutex_lock( &mutex );
    if ( buffer_has_item == 0 ) {
      buffer = make_new_item();
      buffer_has_item = 1; }
    pthread_mutex_unlock( &mutex );
    usleep(100);                        // 100 msec warten
  }
}

void reader_function(void) {            // Verbraucher
  while(1) {
    pthread_mutex_lock( &mutex );
    if ( buffer_has_item == 1 ) {
      consume_item( buffer );
      printf ( "%d \n",buffer );        // Lebenszeichen ausgeben
      buffer_has_item = 0; }
    pthread_mutex_unlock( &mutex );
    usleep(100);                        // 100 msec warten
  }
}

main() {
  pthread_t reader;
  pthread_mutex_init(&mutex, NULL);     // Mutex initialisieren
                                        // ein neuer Thread: Erzeuger
  pthread_create( &reader, NULL, (void*)&reader_function, NULL);
  writer_function();                    // Verbraucher im Haupt-Thread
}
```

Sie übersetzen das Programm mit

```
gcc -o reader-writer reader-writer.c -lpthread
```

und können es dann ausprobieren. Die `usleep`-Aufrufe in den beiden Threads dienen dazu, dem jeweils anderen Thread eine Chance zu geben, den Mutex zu erhalten.

a) Modifizieren Sie das Programm, so dass es mit beliebig vielen Erzeuger- und Verbraucher-Threads arbeitet wie viele, legen dann zwei Variablen (oder Konstanten)`NO_READERS` und `NO_WRITERS` im Programm fest. Außerdem soll der Buffer nicht die Länge 1 haben, sondern bis zu 32 Elemente aufnehmen.

b) Passen Sie das Original-Programm (mit zwei Threads) so an, dass es statt `pthread_mutex_lock` die Funktion `pthread_mutex_trylock` verwendet. Lesen Sie in der Manpage zu `pthread_mutex_trylock` nach, wie sich diese beiden Funktionen unterscheiden, und achten Sie darauf, dass ein Thread nur in den kritischen Bereich (Zugriff auf den Buffer) eintreten darf, wenn er den Mutex auch wirklich erhalten hat.

   (Sollten Sie eine hier angegebene Manpage auf Ihrem Rechner nicht finden, können Sie auch im Internet danach suchen.)

2. Semaphore mit Mutexen simulieren

POSIX-Threads können auch mit Semaphoren synchronisiert werden; dafür gibt es die Funktionen `sem_init`, `sem_post` (entspricht `signal`) und `sem_wait` (entspricht `wait`). Lesen Sie zur Information die zugehörigen Manpages.

a) Sie können das Semaphor-Verhalten aber auch mit Hilfe von Mutexen simulieren. Schreiben Sie zwei Funktionen `mysem_signal` und `mysem_wait` (ohne Argumente), die erwarten, dass es eine Integer-Variable `sem` und einen `pthread_mutex_t sem_mutex` gibt. Sie verwenden den Mutex, um den Zugriff auf die Semaphor-Variable (`sem`) zu schützen. Um das Warte- und Signalisier-Verhalten umzusetzen, werden Sie noch einen weiteren Mutex benötigen.

   (Pthread-Mutexe erlauben es nicht, einen gelockten Mutex in einem fremden Thread zu unlocken – deswegen lässt sich diese Art der Semaphor-Implementation nicht mit Posix-Mutexen realisieren.)

   Sie können statt in C auch in C++ programmieren und eine generische Semaphor-Klasse mit Methoden `sem_signal` und `sem_wait` schreiben, wenn Sie C++ bevorzugen.

b) [Bonusaufgabe] POSIX-Threads bieten noch andere Synchronisationsmethoden, etwa Bedingungsvariablen (siehe Manpages zu `pthread_cond_wait` u.a.). Erstellen Sie ein lauffähiges Programm, das diese alternative Semaphor-Implementation testet. Dabei sollten Sie mindestens vier Threads erzeugen, die eine nur 2x vorhandene Ressource (Semaphor-Initialwert 2) nutzen wollen.

[☛ Abgabe per Mail] Die von Ihnen erstellten Programmdateien schicken Sie bitte per E-Mail an `hans-georg.esser@hm.edu`. Vergessen Sie nicht, Ihren Quellcode gut zu dokumentieren.

Wenn Sie beim Testen feststellen, dass sich Ihr Programm nicht wie gewünscht verhält, Sie dies aber nicht beheben können, geben Sie in der Mail (oder in Kommentaren im Programm) an, inwiefern Ihr Programm sich falsch verhält und was Sie als Ursache vermuten.