Vorbereitung

- Die Dateien zum heutigen Praktikumstermin laden Sie von der Vorlesungsseite herunter:
  `wget http://hm.hgesser.de/bs-ss2011/prakt09.tgz`
- Entpacken Sie das Archiv und wechseln Sie in das neue Unterverzeichnis `prakt09`.

20. POSIX-Threads und Semaphore in C (5 Punkte)

Lesen Sie, ausgehend von der Manpage `sem_overview`, die Dokumentation zu Semaphoren unter Linux und betrachten Sie dann das Programm `reader-writer.c`, das eine ausformulierte Fassung des Producer-Consumer-Beispiels aus der Vorlesung (Foliensatz 5, Folie 43) ist.

Im Folgenden finden Sie die wichtigsten Ausschnitte aus dem Code:

```
#define N 10                    // Groesse des Puffers

// Semaphor-Definitionen, siehe: man sem_overview
static sem_t mutex;             // kontrolliert Zugriff auf Puffer
static sem_t empty;             // zaehlt freie   Plaetze im Puffer
static sem_t full;              // zaehlt belegte Plaetze im Puffer

// Hauptprogramm des Producer-Prozesses -- wird als neuer Thread gestartet
void producer() {
  char item;
  while (item != NOCHAR) {
    item = produce_item();      // Erzeuge etwas für den Puffer
    sem_wait (&empty);          // Leere Plaetze dekrementieren bzw. blockieren
    sem_wait (&mutex);          // Eintritt in den kritischen Bereich
    enter_item (item);          // In den Puffer einstellen
    sem_post (&mutex);          // Kritischen Bereich verlassen
    sem_post (&full);           // Belegte Plaetze erhoehen, evtl. Consumer wecken
  }
}

// Hauptprogramm des Consumer-Prozesses -- wird als neuer Thread gestartet
void consumer() {
  char item;
  while (item != NOCHAR) {
    sem_wait (&full);           // Belegte Plaetze dekrementieren bzw. blockieren
    sem_wait (&mutex);          // Eintritt in den kritischen Bereich
    item = remove_item();       // Aus dem Puffer entnehmen
    sem_post (&mutex);          // Kritischen Bereich verlassen
    sem_post (&empty);          // Freie Plaetze erhoehen, evtl. Producer wecken
    consume_entry (item);       // Verbrauchen
  }
}

// Hauptprogramm
main() {
  pthread_t consumer_thread;  // Thread-Variablen fuer Consumer ...
  pthread_t producer_thread;  // ... und Producer
  sem_init(&mutex, 0, 1);     // Init: 1, Wertebereich: 0-1 (Mutex)
  sem_init(&empty, 0, N);     // Init: N, Wertebereich: 0-N
  sem_init(&full,  0, 0);     // Init: 0, Wertebereich: 0-N

  init_buffer(); printf ("Startwerte:\n"); list();
  // Threads erzeugen
  pthread_create( &producer_thread, NULL, (void*)&producer, NULL);
  usleep(100);
  pthread_create( &consumer_thread, NULL, (void*)&consumer, NULL);

  sleep(1); printf("\n\nEndwerte:\n"); list();
  // Aufraeumen:
  pthread_join( producer_thread, NULL ); pthread_join( consumer_thread, NULL );
}
```


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.

b) Verändern Sie das Hauptprogramm und die Funktion `list()`, so dass `list()` in eine Protokolldatei schreibt und regelmäßig aus dem Haupt-Thread heraus `list()` aufgerufen wird. Prüfen Sie, wie sich der Puffer füllt.