Praktikum 5: Semaphore

Die Lernziele in diesem Praktikum sind

- Verwenden von Pthread-Mutex und Bedingungsvariablen.
- Kennenlernen der POSIX-Semaphore durch Implentieren einer eigenen Semaphor-Bibliothek.
- Erstellen von statischen Bibliotheken.

Betriebssysteme stellen Semaphordienste als elementaren Mechanismus zur Prozesssynchronisation bereit. In der Literatur wird unter einem Semaphor weitestgehend einheitlich eine ganzzahlige Synchronisationsvariable verstanden, auf der die P- und V-Operation definiert sind. Die konkrete Realisierung in den bekannten Betriebssystemen sieht dagegen sehr unterschiedlich aus.

Linux hat zahlreiche Dienste zur Prozesssynchronisation. Häufig verwendet werden Bedingungsvariablen und Mutex aus der Pthread-Bibliothek. *Bedingungsvariablen* dienen ausschließlich zur Signalisierung. Ein *Mutex* synchronisiert Zugriff auf eine Ressource, muss jedoch vom selben Thread belegt (*lock*) und wieder freigegeben (*unlock*) werden, eignet sich also nicht zur Inter-Thread-Kommunikation. In C++ sind Mutex und Bedingungsvariablen als Klassen realisiert und im Header `<mutex>` und `<condition_variable>` verfügbar.

Ein *Semaphor* ist also gegeben aus einer Zählvariable, einer Bedingungsvariable zur Signalisierung und einem Mutex zum Synchronisieren des Zugriffs auf die Zählvariable und Bedingungsvariable.

In diesem Praktikum soll in C++ eine Semaphor-Bibliothek realisiert werden, die die gegebenen Mutex und Bedingungsvariablen benutzt. Die folgende Klasse beschreibt den Semaphor-Typ `my_sem`:

```
class my_sem {
    std::mutex mtx;
    condition_variable cond;
    unsigned int count;
};
```

Aufgabe 1

Basierend auf den Mutex und Bedingungsvariablen der Pthreads-Bibliothek soll eine Bibliothek namens `libsem.a` realisiert werden, welche eine einfachere Schnittstelle für Semaphordienste zu Verfügung stellt.

Die Bibliothek soll die folgenden Methoden analog zu der POSIX-Semaphor-Bibliothek enthalten.

1. `my_sem(int value)`
   Der Konstruktor initialisiert den Semaphor und setzt den Initialwert `value`.
2. `void wait()`
   `wait()` ist die P-Operation.

3. `void post()`
   `post()` ist die V-Operation.

Implementieren Sie die Bibliothek in der Datei `semops.cpp`. Compilieren Sie die Datei zu `semops.o` und benutzen Sie das Programm `ar` um die Bibliothek `libsem.a` zu erstellen.

Aufgabe 2
Im Export-Verzeichnis `/home/export` finden Sie auch ein C++-Programm namens `semtest.cpp` zum Test der von Ihnen bereitgestellten Bibliothek `libsem.a`. Das Programm erzeugt bzw. öffnet einen Semaphor mit Anfangswert 1 und durchläuft einen kritischen Abschnitt.

1. Schreiben Sie ein `Makefile`, welches die Bibliothek libsem.a und ein lauffähiges Programm `semtest` generiert.
2. Überprüfen Sie die korrekte Funktionsweise der Bibliothek libsem.a mit Hilfe des Programms semtest. Starten Sie dazu z `semtest` und prüfen Sie, dass nicht beide Threads gleichzeitig im kritischen Abschnitt sein können.