Fakultät für Informatik und Mathematik
HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN MÜNCHEN

Betriebssysteme, SoSe 2020

Praktische Übung 6: Conditions, Barriers, Monitore

Ziele des Labortermins

- Umsetzung von Synchronisation für einfache Datenstrukturen
- Bedingungen und Barrieren verstehen und einsetzen

Aufgabe 6.1: Verständnisfragen

a) Erklären Sie den Unterschied zwischen grob-grunalerem und fein-granularem Locking.

b) Gibt es eine Limitierung der Threads, die auf eine Condition warten?

c) Gibt es eine Limitierung der Threads, die an einer Barrier teilnehmen?

d) In welchen Situationen ist die Verwendung von Spinlocks der Verwendung von blockierenden Locks vorzuziehen?

Aufgabe 6.2: Verwendung von Conditions (praktisch)

a) Machen Sie sich mit POSIX Thread Conditions (`pthread_cond_*` vertraut.

b) Was ist die Rolle des Mutex-Paramters?

c) Implementieren Sie folgende Funktionalität:

- Ihr Programm startet 128 Threads, die jeweils einen gemeinsamen Zähler inkrementieren.

- Ihr Hauptprogramm wartet in einer Schleife darauf dass alle Threads terminieren indem es den Zähler beobachtet.
- Um nicht permanent den Zähler zu überwachen, soll eine Condition Variable verwendet werden: Das Hauptprogramm wartet dabei auf die Signalisierung der Threads, dass sie den Counter inkrementiert haben.

Aufgabe 6.3: Implementierung einer Barriere (praktisch)

Implementieren Sie eine Barriere entsprechend der Vorlesung wie folgt:

- Definieren Sie einen Datentyp `barrier_t` der die Funktionalität kapselt.
- Der Datentyp `barrier_t` soll die Anzahl der Threads in dieser Barrier angeben.
- Daneben enthält der Datentyp weitere Variablen zur Synchronisierung.
- Implemenetieren Sie die Funktion zur Erstellung einer Barriere `int barrier_init(barrier_t *b, int N)`.
- Implementieren Sie die Funktion `void barrier(barrier_t *barrier)` so dass jeder Thread beim Aufruf davon blockiert bis der N-te Thread sie aufruft.

Aufgabe 6.4: Verwendung von Barrieren (praktisch)

Erweitern Sie die Array-Funktion aus der vorherigen Übung dahingehend, dass es zwei Bearbeitungsphasen gibt, die parallel ohne Verwendung des Fork-Join-Modells bearbeitet werden sollen, mit der Hilfe Ihrer Barrieren-Implementierung:

```
extern int A[];
int x = 0;
for (int i = 0; i < 10000; i++) {
  x += A[i];
}
x /= 100;
for (int i = 0; i < 10000; i++) {
  A[i] = A[i] * x;
}
```