Übungsblatt 9

Dr. Matthias Frank, Dr. Matthias Wübbeling

Ausgabe Mittwoch, 6. Dezember 2023
Abgabe bis
Freitag, 15. Dezember 2023, 23:59 Uhr
Vorführung vom 18. bis zum 22. Dezember 2023

Alle Programme müssen unter **Ubuntu 22.04** kompilierbar bzw. lauffähig und ausreichend kommentiert und mit **Makefile** (C, Assembler) versehen sein, um Punkte zu erhalten. Als Compiler sollen **clang** (C) und **nasm** (Assembler) verwendet werden. Die Lösungen sind bei Ihrem Tutor während Ihrer Übungsgruppen vorzuführen. Alle Gruppenmitglieder sollten die Abgabe erklären können. Die Abgabe erfolgt mittels Ihres Git-Repositories und der vorgegebenen Ordnerstruktur.

Die Punkte der Aufgaben sind relevant für die Zulassung. Die Punkte der Bonusaufgaben werden auf Ihren Punktestand addiert, werden aber nicht auf die für die Zulassung benötigten Punkte addiert.

Abgabestruktur: Jede Abgabe, die die folgende Struktur nicht umsetzt, wird **NICHT** bepunktet bzw. mit 0 Punkten bewertet

- die zu korrigierenden Lösungen müssen bis zur Deadline auf dem `master`-Branch liegen. Lösungen auf anderen Branches werden nicht gewertet
- alle Lösungen müssen in der vorgegebenen Ordnerstruktur (**blattXX/aufgabeYY**) abgelegt werden, wobei `XX` und `YY` durch die jeweiligen Nummern des Zettels und der Aufgaben ersetzt werden sollen. Der Name soll exakt nur aus diesen Zeichen bestehen und achtet auf Kleinschreibung
- sämtliche Aufgaben, mit Ausnahme der theoretischen Aufgaben, die eine PDF erfordern, benötigen zwingend ein **Makefile**. Abgaben ohne dieses werden nicht gewertet

Hinweis:	Manche Browser sind nicht dazu in der Lage die in die PDFs eingebetteten Dateien anzuzeigen. Mittels eines geeigneten Readers, wie dem Adobe Reader, lassen sich diese jedoch anzeigen und verwenden.

Aufgabe 1 Read-Write-Locks mit Mutexes und Condition Variables (4 + 2 = 6 Punkte)

Mutexes und Condition Variables sind zwei grundlegende Synchronisationskonstrukte, auf die sich andere Synchronisationsaufgaben abbilden lassen. Ziel dieser Aufgabe ist es, mit Hilfe von Mutexes und Condition Variables eine eigene Implementierung von Read-Write Locks zu realisieren.

a)

Implementieren Sie eine eigene Read-Write-Lock Bibliothek, die folgende Funktionen bereitstellt:

```
// Initialisiert die Read-Write-Lock-Datenstruktur 'lock'.
int rwlock_init(struct rwlock* lock);
// Fordert einen read-lock für 'lock' an.
int rwlock_lock_read(struct rwlock* lock);
// Fordert einen write-lock für 'lock' an.
int rwlock_lock_write(struct rwlock* lock);
// Gibt einen read-lock auf 'lock' frei.
int rwlock_unlock_read(struct rwlock* lock);
// Gibt einen write-lock auf 'lock' frei.
int rwlock_unlock_write(struct rwlock* lock);
```

Als Hilfsmittel dürfen Sie lediglich Mutexes und Condition Variables aus der PThread-Bibliothek verwenden.

b)

Erweitern Sie ihre Bibliothek so, dass Anforderungen eines Write-Locks Priorität gegenüber Anforderungen von Lese-Locks bekommen. Es gilt folgende Regel: Anforderungen eines Lese-Locks dürfen nur erfüllt werden, wenn keine Anforderung für einen Write-Lock mehr wartet.

c) (2 Punkte)

Schreiben Sie ein Testprogramm, das mit Hilfe mehrerer Threads die korrekte Funktionalität Ihres Read-Write-Locks demonstriert. Entwerfen Sie Ihr Testprogramm so, dass Sie damit **alle** Semantiken Ihrer Implementierungen aus den Teilen a) und b) demonstrieren können.

Aufgabe 2 Matrixmultiplikation mit pthread (3 Punkte)

Entwerfen Sie ein C-Programm, das zwei Matrizen mittels `pthread` multipliziert. Dabei soll die Anzahl erzeugter Threads der Anzahl der Zeilen entsprechen. Sie können dabei von quadratischen Matrizen ausgehen, wobei die Größe per Kommandozeile übergeben werden soll. Die Matrizen sollen dabei während der Laufzeit mit Zufallszahlen gefüllt werden.

Achten Sie dabei darauf, die erzeugten Matrizen, sowie das Ergebnis, in der Kommandozeile auszugeben. Anschließend messen Sie die Zeit, die ihr Programm für die Multiplikation zweier 10x10 Matrizen benötigt und vergleichen Sie diese mit einer herkömmlichen Implementierung ohne Threads.

Dokumentieren Sie ihre Ergebnisse anhand von mindestens 5 Durchläufen in einer PDF-Datei.

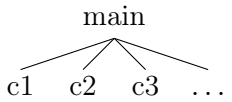
Hinweis: Als Grundgerüst mit bereits existierender `print_matrix`-Funktion bietet sich hierfür Aufgabe 2 von Blatt 4 an.

Wir empfehlen zunächst die Matrixmultiplikation ohne Threads zu implementieren und diese anschließend zu parallelisieren.

Aufgabe 3 Prozessbäume (4 Punkte)

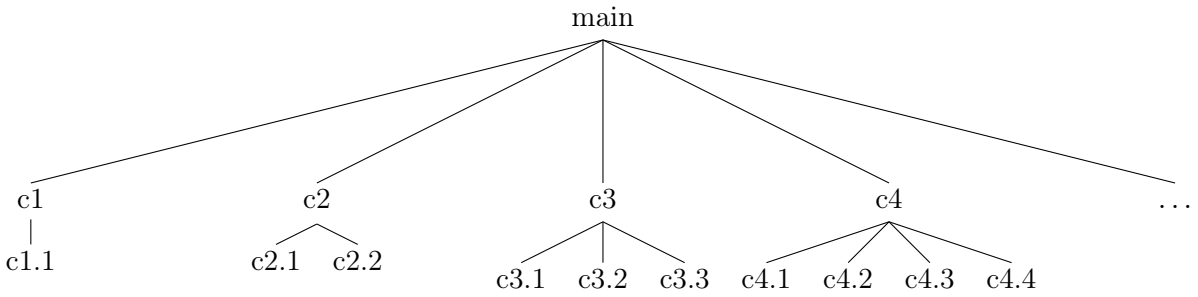
In dieser Aufgabe bekommen Sie Prozessbäume gegeben. Sie sollen ein C-Programm schreiben, das genau diesen Prozessbaum erstellt. An jeder Stelle, wo sie Ellipsen (...) sehen, sollen Sie das Programm bis 5 laufen lassen. Geben Sie außerdem in jedem Prozess den Namen des Prozesses sowie die Prozess-ID und die Prozess-ID des Elternprozesses aus. Der Start-Prozess heißt `main`, die Kinder beginnen mit c. Wenn ein Prozess ein Kind bekommt, hängt er an den Namen des Kindes einen Punkt an sowie die Nummer des Kindes, beginnend bei 1.

a)



b)

c)



d)