0x0b – Übungen

’Westwheat’

Threads, Mutex, ref

Starten Sie zwei Threads und füllen Sie in der Workerfunktion einen (globalen) int-vector jeweils mit n Zufallszahlen.

- Starten Sie mit kleinem n (<5) und schützen Sie den Zugriff auf den Vektor nicht. Funktioniert das?
- Erhöhen Sie nun die Anzahl n solange, bis das Programm abstürzt. Warum ist das so?
- Schützen Sie den Vektor beim Einfügen mit einem lock_guard (thread-safe). Funktioniert es jetzt wieder?
- Anstatt den Vektor global zu definieren, definieren Sie ihn lokal und übergeben Sie ihn als Referenz der Workerfunktion.

0x0b – Übungen

’Coldwall’

Threads, Mutex, ref

- Überlegen Sie sich eine geeignete parallele Strategie mit mehreren Threads und summieren Sie die Zahlen 1..n parallel und thread-safe in einer gemeinsamen lokalen Summenvariablen auf.
- Messen Sie die Zeit jeweils für keine Parallelisierung, für zwei Threads, drei Threads usw. Ab wie vielen Threads wird es nicht mehr schneller bzw. wird es überhaupt schneller?

0x0b – Übungen

’Deepbutter’

Mutex, lock, unlock, unique_lock

- Starten Sie zwei Threads in main und lassen Sie sie sicher (!) in einen Wartezustand laufen.
- Signalisieren Sie aus main heraus dem ersten Thread, dass er weiterlaufen kann und warten auf das Ende der beiden Threads.
- Nach Erhalt des Signals im ersten Thread signalisieren dem zweiten Thread, dass er nun weiterlaufen kann.
- Die beiden Threads haben keine besonderen Aufgaben, es geht hier lediglich um eine wohldefinierte Reihenfolge der Abarbeitung. Verwenden Sie kein sleep, sondern lösen Sie das Problem durch die richtige Verwendung der Befehle.

0x0b – Übungen

’Smoothrock’

file IO

- Implementieren Sie eine "read_all_lines" Funktion, die eine Textdatei komplett einliest und die Zeilen in einem Vektor von Strings zurückgibt.
- Implementieren Sie analog eine "write_all_lines" Funktion, die einen übergebenen Vektor von Strings in eine Textdatei schreibt.