

NVS5 Projektübungen – 1

Schuljahr 2020/21

Dr. Günter Kolousek

Version 1.0 (2020-11-24)

Inhaltsverzeichnis

1 Überblick	1
1.1 Projektedaten	1
1.2 Software	3
2 Benotung	3
2.1 Umfang und Tiefe der Ausführung	4
2.2 Fehlerbehandlung	4
2.3 Ausgaben, Einhaltung der Coding Conventions, Kommentare	5
2.4 Repository	5
2.5 Ausarbeitung	5
2.6 Prüfungsgespräch	6
3 Prüfung und Prüfungsstoff	6
4 Beispielthemen	7

1 Überblick

Es geht bei diesen Projektübungen darum, kleine Projekte mit Prozessen, Threads und Synchronisation für verschiedene Anwendungsszenarien zu entwickeln und damit zu zeigen, dass die praktischen Fähigkeiten zum Implementieren verteilter Systeme erworben wurden.

1.1 Projektedaten

- Repository (siehe auch Abschnitt 2.4) inkl. Ausarbeitung in elektronischer Form: 2021-01-06T23:59
- Abgabe der Ausarbeitung in schriftlicher Form (hard copy): 2021-01-12
- Prüfungsdatum (siehe Abschnitt 2.6): 2021-01-20

Hier folgt eine Liste der weiteren notwendigen Bedingungen:

- Zur Programmierung ist die Version C++17 zu verwenden und das Projekt muss unter Linux zu übersetzen sein. Warnungen sollte es keine geben! Testen!!!
- Meson (Version 0.56+) ist mit dem von mir zur Verfügung gestellten Template zu verwenden (siehe Abschnitt Repository)!!!
- Das Projekt ist unter die Boost-Lizenz zu stellen.
- Die Funktionsweise der eigentlichen Aufgabenstellung ist **gut** mittels Ausgaben zu **zeigen**!!! Das ist wichtig! Wird *diese* Anforderung nicht erfüllt, dann gilt die Implementierung als **nicht** funktionsfähig!!!!

Die Formatierung jeglicher Ausgabe *kann* mittels der Bibliothek `fmt` erfolgen (siehe zur Verfügung gestelltes `template.tar.gz`) zu erfolgen.

- Eine der Aufgabenstellung entsprechende textbasierte Benutzerschnittstelle ist zur Verfügung zu stellen. Das bedeutet, dass die Programme mittels Kommandozeilenparametern gesteuert bzw. mittels Kommandozeilenoptionen konfiguriert werden. So eine Art der Benutzerschnittstelle setzt natürlich auch eine Hilfe mittels der Optionen `-h` bzw. `--help` inklusive einer aussagenkräftigen Fehlermeldung im Fehlerfall voraus!

Es **muss** die Bibliothek `CLI11` verwendet werden (siehe `template.tar.gz`).

- Für das Loggen **ist** die header-only Bibliothek `spdlog` zu verwenden (siehe `template.tar.gz`).
- Es dürfen Bibliotheken wie im Abschnitt 1.2 angeführt verwendet werden.
- Die Coding Conventions sind einzuhalten!
- Die Abgabe erfolgt indem die Software **spätestens** bis zum Abgabezeitpunkt (siehe 1.1) im Repository mit dem Tag `v1.0` versehen wird.

1.2 Software

Software	Zweck
backward-cpp	beautiful stack trace pretty printer
Catch2	Unit-Tests
CLI11	CLI
cppfsm	Finite State Machine for C++
cpp-httplib	HTTP/HTTPS Bibliothek
cpp-subprocess	Prozesse
cpp-peglib	PEG Parser Bibliothek
criterion	Benchmarking
fmt	Formatieren von Strings
glob	File globbing...
inja	Template Engine
json	JSON for Modern C++
magic_enum	Better handling von enums
pprint	Hübsche Ausgabe von C++ Datentypen
pystring	Nützliche String-Funktionen
rang	Färbige Ausgabe am Terminal
spdlog	Logging
tabulate	Tabellen im Terminal
taskflow	Abarbeitung paralleler Tasks
toml++	Konfiguration auf Basis von TOML-Dateien
tfile	Einfaches Handling von Dateien
sqlite_orm	ORM für SQLite

Es handelt sich hier fast ausschließlich um header-only Bibliotheken. Wir gehen davon aus, dass diese Bibliotheken **nicht** in den Repositories abgelegt werden, sondern sich außerhalb des Repositories im Dateisystembaum abgelegt sind. Das von mir zur Verfügung gestellte Template beinhaltet entsprechende Meson-Optionen.

Die beiden Ausnahmen sind `pystring` und `backward-cpp`: Hier ist jeweils die beiden Dateien direkt in das `src` bzw. das `include` Verzeichnis zu kopieren.

2 Benotung

Die zu erreichende Note hängt von den folgenden Faktoren ab:

- Umfang und Tiefe der Implementierung (30%)
- Fehlerbehandlung (10%)
- Ausgaben, Einhaltung der Coding Conventions, Kommentare (10%)
- Repository: Commits, Issues (10%)
- Ausarbeitung (30%)
- Einhaltung der Richtlinien (10%)
- optionales Prüfungsgespräch

2.1 Umfang und Tiefe der Ausführung

Ich gehe davon aus, dass das Programm so entwickelt wird, dass es dem Niveau eines 5. Jahrganges entspricht (siehe Lehrplan). Erfüllt die Abgabe nicht diese Anforderung, dann führt dies automatisch zu einer **negativen** Beurteilung (siehe §15 LBVO)!

Der Umfang jeder Aufgabenstellung (siehe Abschnitt 4) ist an sich gering.

Abgesehen davon: Je mehr *Umfang* und je mehr an *Tiefe*, desto besser für die Note!!!

- ad *Umfang*: Eine minimale Umsetzung hinsichtlich des Umfanges liegt vor, wenn genau die Anforderung aus dem Abschnitt 4 umgesetzt wird. In diesem Fall kann man davon ausgehen, dass die Anforderungen wie in §15 LBVO in den wesentlichen Bereichen überwiegend erfüllt worden sind. Natürlich kann man selbst den Umfang erhöhen, indem man weitere (sinnvolle) Features einbaut.

Werden von mir angegebene Bibliotheken richtig eingesetzt, dann zählt dies *ebenso* als eine Vergrößerung des *Umfanges*!

Hier ein paar Beispiele für den Einsatz von Bibliotheken:

- Es kann durchaus sinnvoll sein, die Optionen zusätzlich oder auch alternativ zur Spezifikation über die Kommandozeile in einer Konfigurationsdatei abzulegen. Dafür könnte man TOML (siehe `toml++`) oder auch JSON (siehe `json`) verwenden.
- Ergebnisse oder auch den Zustand eines Prozesses könnten in einer Datei (siehe `tf::file`) oder auch in einer Datenbank (siehe `sqlite_orm`) abgelegt werden.
- Die Ausgabe könnte farbiger gestaltet werden (siehe `rang`) oder auch Tabellen (siehe `tabulate`) ausgegeben werden.
- Es könnte durchaus Sinn machen zwei verschiedene Implementierungen einer Lösung bzgl. Laufzeit zu vergleichen und dafür die Software `criterion` zu verwenden.
- Zur Ausgabe von Testdaten (z.B. beim Logging) kann man für C++ Datentypen die `pprint` einsetzen.
- Kommt es zu unerwarteten Abstürzen, dann wäre `backward-cpp` durchaus hilfreich.
- ...
- ad *Tiefe*: Man kann sich natürlich weiterführend in die Thematik einarbeiten, dies in der Ausarbeitung beschreiben (Zitierungen nicht vergessen) und in der Implementierung umsetzen.

Mir ist durchaus bewusst, dass die Problemstellungen nicht alle exakt den gleichen Schwierigkeitsgrad aufweisen. Dies wird von mir bei der Benotung berücksichtigt!

2.2 Fehlerbehandlung

Die Programme sollten, wenn möglich, mit jeder Art von Fehlersituation zurechtkommen und je nach Art des Fehlers diesen entweder

- maskieren und loggen
- loggen und nochmals probieren
- loggen und abbrechen

Für das Loggen siehe den folgenden Abschnitt.

2.3 Ausgaben, Einhaltung der Coding Conventions, Kommentare

Es sind sinnvolle Ausgaben vorzunehmen und Logginginformationen (spdlog ist zu verwenden) auszugeben, die die Funktionalität der Anwendung **klar** demonstrieren!

Weiters sind im Code aussagekräftige Kommentare zu verfassen, wo dies sinnvoll ist.

Ausgaben und Kommentare können entweder in Deutsch oder Englisch verfasst werden. Weder bei den Ausgaben noch bei den Kommentaren dürfen jeweils die Sprachen gemischt werden.

Die vorgegebenen Coding Conventions sind einzuhalten (wie dies im Unterricht besprochen wurde).

2.4 Repository

Das Repository muss wieder auf github gehostet werden und folgendermaßen benannt sein: <nachname>_project_1 (alles in Kleinbuchstaben und ohne Sonderzeichen!!!).

Dieses Repository muss ein Meson-Projekt beinhalten, das auf dem von mir zur Verfügung gestellten Template basiert, wobei der Ordner `build` **leer** zu sein hat.

Bzgl. Commits:

- jeweils eine logische Einheit, also etwas was man als Patch verwenden kann bzw. ein Schritt den man wieder zurücknehmen können soll.
- Commit-Meldungen sollen kurz und prägnant sein und sollen ausdrücken für was dieser Commit steht.

Der Schreibstil sollte aktiv sein, also: "Add this and that" **anstatt** "This and that was added".

Issues sind anzulegen und zu bearbeiten.

Es ist ein Readme anzulegen, das kurz das Projekt beschreibt.

2.5 Ausarbeitung

- Inhalt: Kurze Beschreibung des Hintergrunds (um was geht es, wie sieht die Lösung aus), des Aufbaus (wie wurde die Lösung umgesetzt) und der Bedienung in eigenen Worten und kommentierten Source-Code-Beispielen.
- Umfang: Deckblatt, Inhaltsverzeichnis, Inhalt, ggf. Literaturverzeichnis

- Dateiformat: Textformat in einer Auszeichnungssprache: \LaTeX oder Emacs Org-Mode (z.B. auch mittels pandoc). **Zusätzlich** ist die Ausarbeitung elektronisch als PDF *und* als hard-copy abzugeben (siehe Abschnitt Projektedaten).

2.6 Prüfungsgespräch

Ich werde, wenn notwendig, d.h. *in Abhängigkeit* des Projektverlaufes, ein kurzes Prüfungsgespräch mit Ihnen führen, das den abgedeckten Stoff des gewählten Beispiels, die eigentliche Umsetzung zum Inhalt sowie den unter Abschnitt Prüfung und Prüfungsstoff festgelegten Umfang zur Folge hat.

Wird dieses Projekt *nicht* oder *nicht in genügender Form* abgegeben, dann wird eine praktische Leistungsfeststellung abzulegen sein.

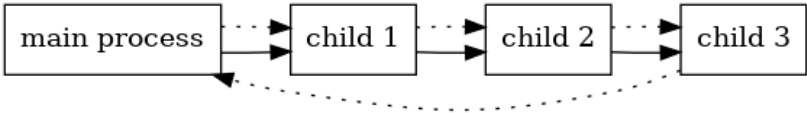
Wird auch die verlangte Leistungsfeststellung nicht abgelegt, dann muss ich im Semester die Note mit "gestundet" festsetzen.

3 Prüfung und Prüfungsstoff

Stoffumfang der Prüfung ist folgender Teil des heuer durchgenommenen Stoffes, angegeben durch die Foliensätze:

- 10_processes
- 11_threads
- 12_threads2
- 13_synchronization
- 14_condition_synchronization
- 15_sync_mechanisms
- 16_threadsafe_interfaces
- 17_dist_sync
- 18_task_based_programming
- 19_parallel_programming
- 20_threads_perfmem
- 21_encoding
- 22_data_interoperability
- 23_character_encoding

4 Beispielthemen

Nummer	Thema
1	Simulation einer verteilten Synchronisation mit einem zentralen Koordinator: Es gibt einen Koordinator- und n Workerthreads. Jeder Workerthread will in zufälligen Abständen (zwischen 3 und 5 Sekunden) in den kritischen Abschnitt eintreten. Im kritischen Abschnitt verbleibt ein Worker 4 Sekunden. n ist über die Kommandozeile zu konfigurieren.
2	Simulation einer verteilten Synchronisation basierend auf dem verteiltem Algorithmus: Es gibt insgesamt n Workerthreads. Jeder Workerthread will in zufälligen Abständen (zwischen 3 und 5 Sekunden) in den kritischen Abschnitt eintreten. Im kritischen Abschnitt verbleibt ein Worker 4 Sekunden. n ist über die Kommandozeile zu konfigurieren.
3	Simulation einer verteilten Synchronisation basierend auf dem Tokenring-Algorithmus: Es gibt insgesamt n Workerthreads. Jeder Workerthread will in zufälligen Abständen (zwischen 3 und 5 Sekunden) in den kritischen Abschnitt eintreten. Im kritischen Abschnitt verbleibt ein Worker 4 Sekunden. n ist über die Kommandozeile zu konfigurieren.
4	Wahlalgorithmus im Ring basierend auf dem Chang-Roberts Algorithmus: Es gibt insgesamt n Workerthreads. Jeder Workerthread bekommt beim Starten eine zufällige ID und alle 5 Sekunden wird ein neuer Wahlvorgang gestartet. n ist über die Kommandozeile zu konfigurieren.
5	<p>Ein Programm wird mit einer ganzen, positiven Zahlen n gestartet. Dieser gestartete Hauptprozess startet einen Kindprozess, dieser Kindprozess startet wieder einen Kindprozess,... bis insgesamt n Kinprozesse laufen. Der letzte Kindprozess sendet ein Signal an den Hauptprozess. Beim Empfang dieses Signals sendet der Hauptprozess ein "KILL" Signal an sein Kind. Dieses wiederum sendet daraufhin ein "KILL" Signal an sein Kind, usw. Jeder Elternprozess wartet auf seinen Kindprozess und beendet sich danach selber.</p>  <pre> graph LR MP[main process] -.-> C1[child 1] C1 --> C2[child 2] C2 -.-> C3[child 3] C3 -.-> MP </pre> <p>(Die Bibliothek <code>cpp-subprocess</code> darf nicht verwendet werden)</p>

Fortsetzung nächste Seite

Nummer	Thema
6	<p>Ein Programm wird mit zwei ganzen, positiven Zahlen n und m gestartet. Es werden vom Hauptprozess n Kindprozesse gestartet, die wiederum je m Kindprozesse starten (Annahme: $n = 3, m = 2$):</p> <pre> graph TD main[main process] --> child1[child 1] main --> child2[child 2] main --> child3[child 3] child1 --> child11[child 1.1] child1 --> child12[child 1.2] child2 --> child21[child 2.1] child2 --> child22[child 2.2] child3 --> child31[child 3.1] child3 --> child42[child 4.2] child11 -.-> child1 child12 -.-> child1 child21 -.-> child2 child22 -.-> child2 child31 -.-> child3 child42 -.-> child3 </pre> <p>Die Prozesse werden gemäß der Angabe erzeugt. Nachdem ein Prozess alle Kindprozesse angelegt hat wartet dieser darauf, dass sich diese beenden und beendet sich daraufhin selbst. Der Hauptprozess wartet 2 Sekunden und sendet danach je ein Signal an seine Kindprozesse, die daraufhin je ein Signal an ihre Kindprozesse weiterschicken. Haben die Blattprozesse ein Signal erhalten, dann beenden sie sich. (Die Bibliothek <code>cpp-subprocess</code> darf nicht verwendet werden)</p>
7	<p>Schreibe bitte eine Lösung für das folgende Problem: Die Berechnung der Kreiszahl π soll auf 2 Prozesse aufgeteilt werden. Die Berechnung soll als Näherung basierend auf der Leibniz' Formel erfolgen.</p> <p>Ein Programm wird mit einer ganzen, positiven Zahl $n \geq 2$ gestartet. Vom Hauptprozess werden insgesamt 2 Kindprozesse gestartet, die vorerst lediglich warten, dass diese vom Hauptprozess ein Signal zum Starten der Berechnung erhalten. Danach berechnet der erste Prozess alle positiven Terme $\leq n$ und der zweite Prozess alle negativen Terme $\leq n$. Das jeweilige Teilergebnis wird dem Elternprozess so mitgeteilt, dass dieses Teilergebnis in eine Datei geschrieben wird. Danach schickt der Client ein Signal an den Elternprozess. Der Elternprozess wird auf das jeweilige Signal warten und danach den entsprechenden Kindprozess beenden. Hat der Elternprozess beide Teilergebnisse aus den jeweiligen Dateien gelesen, dann berechnet dieser daraus das Endergebnis und gibt dieses auf <code>stdout</code> aus und beendet sich danach. (Die Bibliothek <code>cpp-subprocess</code> darf nicht verwendet werden)</p>
8	<p>Simulation einer Datenübertragung von ganzen Zahlen basierend auf der Kodierung Signed LEB128 wobei Zahlen (zufällig zwischen -100000 und 100000; über die Kommandozeile konfigurierbar) zwischen zwei Threads übertragen werden sollen. Es sollen permanent Zahlen im Sekundentakt übertragen werden, wobei die Übertragung als String stattfinden soll. Es sind <code>promise</code> und <code>future</code> Paare zur Kommunikation zu verwenden.</p>

Fortsetzung von vorheriger Seite

Nummer	Thema
9	Simulation einer Datenübertragung von ASCII Zeichen wobei jeweils 9 ASCII Zeichen (Bytes) zu einem Block zusammengefasst werden. Jeder Block soll eine Fehlererkennung mittels zweidimensionaler Parität sicherstellen. Hiermit erweitert sich der Block um ein Byte. Blöcke werden im Sekundentakt von einem Server- an einen Clientthread gesendet. Es sind future und promise Paare zur Kommunikation zu verwenden. Der Grundvorrat an ASCII Zeichen ist per Kommandozeile konfigurierbar.
10	Simulation einer Datenübertragung von ASCII Zeichen wobei jeweils 9 ASCII Zeichen (Bytes) zu einem Block zusammengefasst werden. Jeder Block soll eine Fehlererkennung mittels zweidimensionaler Parität sicherstellen. Hiermit erweitert sich der Block um ein Byte. Blöcke werden im Sekundentakt von einem Server- an einen Clientthread gesendet. Es sind future und promise Paare zur Kommunikation zu verwenden. Der Grundvorrat an ASCII Zeichen ist per Kommandozeile konfigurierbar.
11	Simulation der Leitungskodierung NRZ-I: Ein Senderthread sendet zufällige ASCII Zeichen (Bytes) an einen Empfängerthread wobei die einzelnen Bits mittels NRZ-I kodiert werden und als String übertragen werden. Der Empfängerthread dekodiert die Daten und gibt diese aus. Zur Kommunikation zwischen den beiden Threads ist eine Queue zu implementieren und zu verwenden. Der Grundvorrat an ASCII Zeichen ist per Kommandozeile konfigurierbar.
12	Simulation der Leitungskodierung AMI: Ein Senderthread sendet zufällige ASCII Zeichen (Bytes) an einen Empfängerthread wobei die einzelnen Bits mittels AMI kodiert werden und als String übertragen werden. Der Empfängerthread dekodiert die Daten und gibt diese aus. Zur Kommunikation zwischen den beiden Threads ist eine Queue zu implementieren und zu verwenden. Der Grundvorrat an ASCII Zeichen ist per Kommandozeile konfigurierbar.
13	Simulation der Leitungskodierung MLT-3: Ein Senderthread sendet zufällige ASCII Zeichen (Bytes) an einen Empfängerthread wobei die einzelnen Bits mittels MLT-3 kodiert werden und als String (ASCII mit 0en und 1en) übertragen werden. Der Empfängerthread dekodiert die Daten und gibt diese aus. Zur Kommunikation zwischen den beiden Threads ist eine Queue zu implementieren und zu verwenden. Der Grundvorrat an ASCII Zeichen ist per Kommandozeile konfigurierbar.
14	Simulation der Blockkodierung 4B5B: Ein Senderthread sendet zufällige ASCII Zeichen (Bytes) 4B5B kodiert als String (ASCII mit 0en und 1en) an einen Empfängerthread. Der Empfängerthread dekodiert die Daten und gibt diese aus. Zur Kommunikation zwischen den beiden Threads ist eine Queue zu implementieren und zu verwenden. Der Grundvorrat an ASCII Zeichen ist per Kommandozeile konfigurierbar.

Fortsetzung nächste Seite

Fortsetzung von vorheriger Seite

Nummer	Thema
15	<p>Simulation einer Aufzugssteuerung mit genau 3 Etagen (je ein Thread) und einem Aufzug. Das System soll mittels der Kommandos CALL <floornr> (Rufe Aufzug zu Etage floornr, MOVE <floornr> (Bewege Aufzug zu Etage floornr) verfügen (mittels eines REPL-Interpreters). Z.B.</p> <pre>>>> CALL 3 >>> CALL 2 >>> MOVE 1 >>> MOVE 3</pre> <p>Daraufhin wird der Aufzug in die Etage 3 gerufen, während sich dieser zu Etage 3 bewegt, wird der Aufzug auch zur Etage 2 gerufen, in Etage 3 steigt wer und fährt zur Etage 1, danach bewegt sich der Aufzug zur gewünschten Etage 2, es steigt wer ein und fährt zur Etage 3. Die Zeit zwischen den Etagen soll z.B. 3s betragen (per Kommandozeile konfigurierbar).</p>
16	<p>Simulation des Reader/Writer Locks mit 3 Reader- und 2 Writerthreads (jeweils per Kommandozeile konfigurierbar). Schreibe dazu eine Klasse RWLock mit den Methoden read_lock() und write_lock(), die jeweils einen Read- bzw. Write-Lock anfordern und der Methode unlock(), die den entsprechenden Lock wieder zurückgibt (ohne dafür die Funktionalität shared_lock / unique_lock zu verwenden).</p>
17	<p>Entwickle eine Anwendung, die Arbeitspakete auf Basis von packaged_task durch Workerthreads abarbeitet: Der Hauptthread erstellt z.B. 10 Instanzen (abhängig von der Kommandozeile) folgender Gestalt packaged_task<double>([x=i](){return x*x;}); mit $i \in [0,10)$, die in eine geeignet erstellte Task-Queue (Klasse TaskQueue oder Instanz eines geeigneten Templates, Methoden push und pop) gestellt werden. Z.B. 3 Workerthreads (abhängig von der Kommandozeile) (Klasse Worker mit überladenen Operator ()) holen sich nacheinander Pakete aus der Task-Queue, arbeiten diese ab und stellen future<double> Objekte in eine Result-Queue (Klasse ResultQueue oder Instanz eines geeigneten Templates). Der Hauptthread legt alle Workerthreads an, holt sich die Ergebnisse aus der Result-Queue und gibt die ermittelten Ergebnisse auf stdout aus.</p>
18	<p>Simulation mit z.B. 5 Threads (abhängig von der Kommandozeile) einer (wiederverwendbaren) Barrier (siehe "The Little Book of Semaphores", Abschnitt 3.6).</p>
19	<p>Simulation mit z.B. 5 Sitzen (abhängig von der Kommandozeile), die das Barbershop-Problem (siehe "The Little Book of Semaphores", Abschnitt 5.2).</p>
20	<p>Simulation des River crossing-Problem (siehe "The Little Book of Semaphores", Abschnitt 5.7). Zeitliche Parameter der Simulation sollen per Kommandozeile konfigurierbar sein.</p>

Fortsetzung nächste Seite

Fortsetzung von vorheriger Seite

Nummer	Thema
21	Simulation des Santa Claus-Problem (siehe "The Little Book of Semaphores", Abschnitt 5.5). Zeitliche Parameter der Simulation sollen per Kommandozeile konfigurierbar sein.
22	Simulation einer Ampelsteuerung: 1 Ampel (Thread) mit 4 Straßeneinmündungen (je ein Thread) inkl. per Kommandozeile spezifizierter Auslastung der Straßeneinmündungen (d.h. wie viele neue Autos je Zeiteinheit ankommen)
23	<p>Entwickle eine Klasse <code>Reservoir</code>, die über die Methoden <code>void charge(double val)</code>, <code>void discharge(double val)</code>, <code>double value()</code> sowie über <code>Reservoir(double curr, double max, double min)</code> verfügt.</p> <p>Diese Klasse stellt ein Reservoir dar, das anfänglich mit einem aktuellen Wert <code>curr</code> befüllt ist. Der Wert darf nicht unter <code>min</code> und nicht über <code>max</code> gehen. Würde der Wasserstand zu hoch bzw. zu nieder werden, wird gewartet. Schreibe dafür eine Simulation, deren Konfiguration mittels Kommandozeilenparameter gesteuert werden kann.</p>
24	<p>Implementiere eine Klasse <code>Monitor</code>, die einen Monitor implementieren soll und dazu über die beiden Methoden <code>void enter()</code> und <code>void leave()</code> verfügt. Bei Aufruf von <code>enter()</code> soll der aktuelle Thread in den Monitor eintreten. Bei Aufruf von <code>leave</code> wird der kritische Abschnitt wieder verlassen. Es muss sichergestellt sein, dass nur der Thread den Monitor korrekt verlassen darf, der diesen auch betreten hat. D.h. die Klasse soll eben wie der Synchronisationsmechanismus <code>Monitor</code> funktionieren.</p> <p>Schreibe weiters eine Klasse <code>Worker</code>, dessen Instanzen aufrufbar sind und im Konstruktor eine Referenz auf einen <code>Monitor</code> als auch ein <code>Callable</code> zur Verfügung gestellt bekommen. Wird eine Instanz von <code>Worker</code> aufgerufen, dann führt diese 2 Mal das im Konstruktor übergebene <code>Callable</code> aus (allerdings <code>threadsafe</code> mit Hilfe des <code>Monitors</code>) und wartet jedes Mal 200ms. Lege n Threads an (abhängig von der Kommandozeile) und das <code>Callable</code> soll jeweils "w1 working..." (bzw. w2, w3,...) ausgeben.</p>