Beispiel `07_sync2`

Dr. Günter Kolousek

22. Oktober 2018

1 Allgemeines

- **Backup** nicht vergessen!
- Im ersten Beispiel gibt es genaue Anweisungen zum Aufbau und der Durchführung eines Beispiels. Bei Bedarf nochmals durchlesen!
- Trotzedem hier noch zwei Erinnerungen:
  - Regelmäßig Commits erzeugen!
  - Backup deines Repos nicht vergessen (am Besten nach jedem Beispiel)!!!
- Verwende das bereitgestellte Archiv `template.tar.gz` zum Erstellen eines Meson-Projektes. Es enthält die notwendigen Anpassungen zur Verwendung von Threads.
- Die Anpassungen der Datei `.hgignore` sollten schon erledigt sein, sodass das Verzeichnis `build` nicht versioniert wird.
- In diesem Sinne ist jetzt ein neues Verzeichnis `07_sync2` anzulegen.

2 Aufgabenstellung

Aufgabe ist es, ein einfaches Producer/Consumer Programm `loadsim` mit einer Queue zu schreiben. Bei dem Producer handelt es sich um einen Boss, der in variablen Abständen (0-1s) Arbeitspakete in die Queue stellt. Insgesamt soll es 3 Consumer (Worker) geben, die für die Abarbeitung der Arbeitspakete eine variable Zeitspanne (1-10s) benötigen. Los geht's!

3 Anleitung

Alle Dateien dieses Programmes sind in `src` bzw. in `include` abzulegen.

1. Entwickle eine Klasse `WorkPacket`. Diese enthält lediglich eine Nummer (`id`), die das WorkPacket identifiziert. Diese Nummer wird in dem Konstruktor übergeben. Für die `id` ist eine Getter-Methode zu implementieren.

   Mehr enthält diese Klasse nicht. D.h. diese ist sehr einfach, soll als sogenanntes Value Object verwendet werden.

   Diese Klasse kann durchaus in einer einzigen Headerdatei `work_packet.h` entwickelt werden.

2. Entwickle weiters selber eine thread-safe Klasse `WorkQueue` (Dateien `work_queue.h` und `work_queue.cpp`), die über die beiden Methoden `push` und `pop` verfügt. Als Parameter bzw. Returnwert treten `WorkPacket` - Instanzen auf. D.h. die Instanzen von `WorkPacket` werden kopiert (per-value übergeben). Auf mögliche Probleme beim Allozieren von Speicher wird daher nicht Rücksicht genommen.

   Die `WorkQueue` ist von der Größe her nicht beschränkt.

   Für die Implementierung kann eine `std::queue` verwendet werden.

   Übungszweck

   - thread-safe unbeschränkte Queue (unbounded) auf der Basis von Bedingungsvariablen implementieren.

3. Implementiere weiters ein Programm `loadsim`, das in diese Queue einfach nur fortlaufend Arbeitspakete im zeitlichen Abstand von 0.5 Sekunden einstellt und nach jedem Hinzufügen eine Meldung auf der Konsole ausgibt:

   ```
   B: Submitted work packet 0
   B: Submitted work packet 1
   B: Submitted work packet 2
   B: Submitted work packet 3
   ```

   Beendet wird dieses Programm durch Abbruch mittels `Ctrl-C`.

4. Implementiere in deinem Programm jetzt eine Funktion `void worker(int id, WorkQueue& q)`, die

   - eine `id` für den Worker, also 1,2,… als Parameter bekommt
   - von der übergebenen Queue zuerst ein Arbeitspaket haben will und dieses auch mit einer Meldung auch kommentiert, dann das Arbeitspaket abholt, dieses in einer Sekunde abarbeitet und im Anschluss den Erfolg wieder in einer Meldung kommentiert. Dann beginnt die Arbeit wieder von vorne. All dies soll in einer Endlosschleife passieren.

   Starte zwei solcher Worker als Threads. Die Ausgabe wird dann in etwa folgendermaßen aussehen:

```
WW21: Want work packet: Want work packet

B:  Submitted work packet 0
W2: Got work packet 0
B:  Submitted work packet 1
W1: Got work packet 1
W2: Processed work packet 0
W2: Want work packet
```

Es stellt sich halt jetzt die Frage, warum die ersten Zeilen komisch aussehen und die anderen Zeilen ganz in Ordnung sind. Weißt du warum?

Behebe dieses Manko!

Das Ergebnis sollte jetzt auf jeden Fall so etwas wie das Folgende sein:

```
W1: Want work packet
W2: Want work packet
B:  Submitted work packet 0
W1: Got work packet 0
B:  Submitted work packet 1
W2: Got work packet 1
W1: Processed work packet 0
W1: Want work packet
B:  Submitted work packet 2
```

Übungszweck

- Thread-safe Implementierung einer Queue

5. Erweitere jetzt die Simulation so, dass die Arbeitspakete beim

   - Boss in einer zufälligen Zeit im Intervall $[0, 1)$ Sekunden erzeugt werden und
   - Worker in einer zufälligen Zeit im Intervall $[1, 10)$ Sekunden verarbeitet werden.

   Starte außerdem 3 Worker-Threads.

   Erweitere weiters die Ausgabe, dass diese folgendermaßen aussieht (beachte die Zeiten und die "Waiting…" Zeilen):

```
W2: Want work packet
W1: Want work packet
W3: Want work packet
B:  Waiting to submit work packet 0
B:  Submitted work packet 0 (0.69s)
W2: Got work packet 0
B:  Waiting to submit work packet 1
```

```
B:  Submitted work packet 1 (0.84s)
W1: Got work packet 1
B:  Waiting to submit work packet 2
B:  Submitted work packet 2 (0.87s)
W3: Got work packet 2
B:  Waiting to submit work packet 3
B:  Submitted work packet 3 (0.54s)
B:  Waiting to submit work packet 4
B:  Submitted work packet 4 (0.72s)
W3: Processed work packet 2 (1.6s)
```

Übungszweck

- Zufällige Zeiten in `C++` verwenden.

6. Im Konstruktor wird jetzt die Größe der Queue angegeben, wobei davon *ausgegangen* wird, dass die Größe größer als 0 ist. D.h. es wird auf eine Bounded Queue umgebaut.

   Der Konstruktor kann in bewährter Manier in der Headerdatei implementiert werden.

   Es können nicht mehr Pakete in der Queue gespeichert werden als diese groß ist.

   Die Größe der Queue soll über die Kommandozeile als Argument mitgegeben werden können.

   Übungszweck

   - Besseres Verstehen der Bedingungsvariable durch Erweitern auf eine Bounded Queue.