Beispiel `04_sync2`

Dr. Günter Kolousek

14. Dezember 2020

1 Allgemeines

- 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** nicht vergessen!
- In diesem Sinne ist jetzt ein neues Verzeichnis `04_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

1. Entwickle eine Klasse `WorkPacket` in einem header-only Modul `work_packet`. Ab jetzt kannst du gerne `#pragma once` verwenden, da die aktuellen `C++`-Compiler unter Windows, Linux und MacOS dies unterstützen, auch wenn dies nicht im `C++`-Standard enthalten ist.

   Diese Klasse enthält lediglich eine Nummer (`id`), die das WorkPacket identifiziert (nicht veränderbar!). Diese Nummer wird in dem Konstruktor übergeben. Für die `id` ist eine Getter-Methode zu implementieren. Wie du weißt soll eine Getter-Methode keine Nebeneffekte haben, d.h., dass diese Methode `const` ist…

Mehr enthält diese Klasse nicht. D.h. diese ist sehr einfach und 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 einen Konstruktor verfügt und den 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 soll eine `std::queue` verwendet werden. Schaue dir spezielle die Datenstrukturen `std::set`, `std::unordered_set` und `std::vector` an.

3. Implementiere weiters das 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
```

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)
```

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. Die Größe soll vom Typ `size_t` sein!

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. D.h. die Methode `put` ist ebenfalls und analog zu `take` zu verändern.

7. Die Größe der Queue soll über die Kommandozeile als Argument mitgegeben werden können. Wird das Programm ohne Argument aufgerufen, dann soll es zu folgender Ausgabe kommen:

```
$ loadsim
size is required
Run with --help for more information.
$ loadsim -h
Boss and worker simulation
Usage: loadsim [OPTIONS] size

Positionals:
  size UINT REQUIRED          Size of the queue

Options:
  -h,--help                   Print this help message and exit

$ loadsim -3
Could not convert: size = -3
Run with --help for more information.
```

4 Übungszweck dieses Beispiels

- `#+pragma once` einsetzen
- `const` für Variablen einsetzen
- `const` für Methoden einsetzen
- Wiederholung von synchronisierten Ausgaben
- Wiederholung von Threads mit Parametern per Referenz starten
- Datenstrukturen aus der Containers-Library der Standardbibliothek verwenden, im Speziellen: `std::queue`.
- thread-safe unbeschränkte Queue (unbounded) auf der Basis von Bedingungsvariablen implementieren.
- Thread-safe Implementierung einer Queue

- Wiederholung der Verwendung zufälliger Zeiten in `C++`.
- Besseres Verstehen der Bedingungsvariable durch Erweitern auf eine Bounded Queue.
- `size_t` für Größenangaben
- Wiederholen und festigen: Einfache textbasierte Kommandozeilen-basierten Programme mit `CLI11`