ulm university universität
uulm

Institut für Numerische Mathematik

Dr. Andreas F. Borchert

17. April 2015
Blatt 1

Parallele Programmierung mit C++ (SS 2015)

Abgabe bis zum 24. April 2015, 14:00 Uhr

Lernziele:

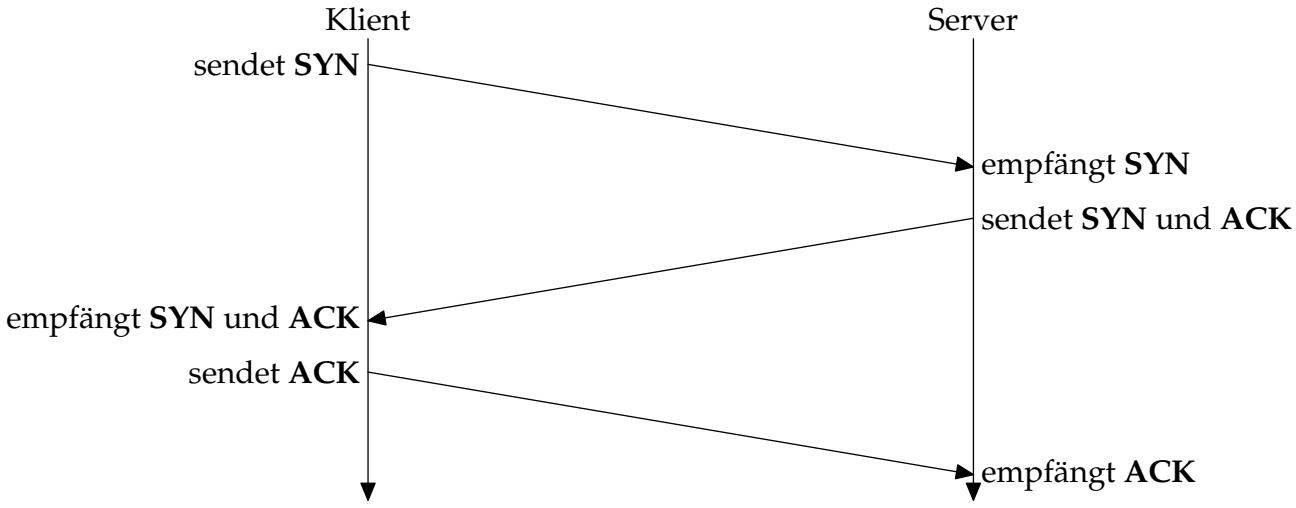
- Modellierung einfacher deterministischer Prozesse mit CSP
- Implementierung und Benutzung einer Klasse in C++

Hinweise:

Melden Sie sich für diese Veranstaltung in SLC an. Ohne diese Anmeldung können Sie keine Lösungen elektronisch einreichen. Sobald Sie sich angemeldet haben, wird es bis zu einer Stunde brauchen, bis Sie anschließend Lösungen einreichen können.

Aufgabe 1: Three-Way Handshake

Modellieren Sie den Beginn einer TCP-Verbindung mit Hilfe von CSP:



Der Klient beginnt mit dem Systemaufruf *connect*, sendet dann ein SYN-Paket, wartet dann auf das SYN-und-ACK-Paket, sendet dann ein ACK, worauf es die Verbindung als etabliert

betrachtet. Der Server beginnt mit dem Systemaufruf *listen*, wartet dann auf den Eingang des SYN-Pakets, schickt dann das SYN-und-ACK-Paket, wartet dann auf das ACK und betrachtet dann die Verbindung als etabliert. Modellieren Sie dabei auch das dazwischenliegende Netzwerk, das jeweils ein Paket transportiert. Achten Sie dabei darauf, dass die Schnittmenge der Alphabete des Klienten und des Servers leer ist. Sobald der Klient bzw. der Server die Verbindung als etabliert betrachten, können sie jeweils erfolgreich terminieren (in CSP mit Hilfe von *SKIP*).

Eine Ergänzungsmöglichkeit wäre der Fall, dass der Server noch nicht *listen* aufgerufen hat, aber schon ein SYN-Paket erhält. In diesem Falle würde er ein RST-Paket schicken (*connection refused*).

Ihre Lösung können Sie einreichen mit

```
submit pp 1 handshake.csp
```

Aufgabe 2: Pipeline

Modellieren Sie mit CSP eine Produzenten/Konsumenten-Konfiguration mit einer zwischenliegenden Pipeline. Der Produzent produziert und sendet das Produzierte fortwährend. Der Konsument empfängt und verbraucht ohne Unterlass. Im einfachsten Fall hat die Pipeline nur die Kapazität eines Produkts, das zuerst vom Produzenten empfangen und dann vom Konsumenten empfangen wird. D.h. wenn Sie einen Ablauf $\langle produce, send, produce \rangle$ haben, dann kann dieser nur mit *receive* fortgesetzt werden, da ein weiteres *send* aufgrund der begrenzten Kapazität der Pipeline nicht möglich ist.

Modellieren Sie eine Pipeline mit einer Kapazität von drei Paketen, d.h. bei einem Ablauf kann die Zahl der *send*-Ereignisse die Zahl der *receive*-Ereignisse um bis zu drei übersteigen.

Ihre Lösung können Sie einreichen mit

```
submit pp 2 pipe.csp
```

Aufgabe 3: Nim-Spiel

Zu implementieren ist eine einfache Variante des Nimm-Spiel in C++. Gegeben sei eine Anzahl von Stäbchen, die pseudo-zufällig aus dem Bereich $10 \ldots 19$ gewählt werden soll. Dem menschlichen Spieler ist es dann überlassen, zu entscheiden, wer beginnen soll. Es wird abwechselnd gezogen. Wer dran ist, darf zwischen einem und drei Stäbchen wegnehmen. Wer das letzte oder die letzten Stäbchen wegnimmt, hat gewonnen.

Hier ist ein Beispiel für einen Spielverlauf:

```
thales$ TestNim
*** Nimm-Spiel ***
Ein Haufen von 19 Staebchen liegt vor.
In jedem Zug koennen zwischen 1 und 3 Staebchen entfernt werden.
Wer die letzten Staebchen entfernt, hat gewonnen.
Wer soll anfangen? 1=Sie 2=Computer
```

```
1
Noch sind es 19 Staebchen.
Ihr Zug: 3
Noch sind es 16 Staebchen.
Der Computer nimmt 2 Staebchen.
Noch sind es 14 Staebchen.
Ihr Zug: 2
Noch sind es 12 Staebchen.
Der Computer nimmt 1 Staebchen.
Noch sind es 11 Staebchen.
Ihr Zug: 3
Noch sind es 8 Staebchen.
Der Computer nimmt 1 Staebchen.
Noch sind es 7 Staebchen.
Ihr Zug: 3
Noch sind es 4 Staebchen.
Der Computer nimmt 1 Staebchen.
Noch sind es 3 Staebchen.
Ihr Zug: 3
Herzlichen Glückwunsch, Sie haben gewonnen!
thales$
```

Bei der Lösung ist eine Klasse zu erstellen, deren Objekte den Stand eines Nimm-Spiels repräsentieren können. Einen Vorschlag für diese Schnittstelle, die Sie übernehmen können, steht als *Nim.hpp* zur Verfügung.

Wenn Sie die vorgegebene Schnittstelle verwenden, brauchen Sie nur noch die beiden anderen Dateien einzureichen:

```
submit pp 3 Nim.cpp TestNim.cpp
```

Wenn Sie eine eigene Fassung von *Nim.hpp* haben, sollten Sie diese mit einreichen:

```
submit pp 3 Nim.hpp Nim.cpp TestNim.cpp
```

Hinweis: Ein einfacher Pseudo-Zufallszahlengenerator steht mit der Funktion *rand()* zur Verfügung, der ganze Zahlen liefert, die mit dem Modulo-Operator in geeignete Bereiche abgebildet werden kann. Damit bei jedem Aufruf die Folgen unterschiedlich sein können, empfiehlt es sich, die Folge mit Hilfe z.B. der aktuellen Zeit zu initialisieren:

```
#include <cstdlib>
#include <ctime>

srand(time(0)); // seed random generator
int rand_val = rand() % 100; // pseudo-random value out of [0,99]
```

Viel Erfolg!