ulm university universität
uulm

Institut für Numerische Mathematik

Dr. Andreas F. Borchert

3. Mai 2019
Blatt 2

Parallele Programmierung mit C++ (SS 2019)

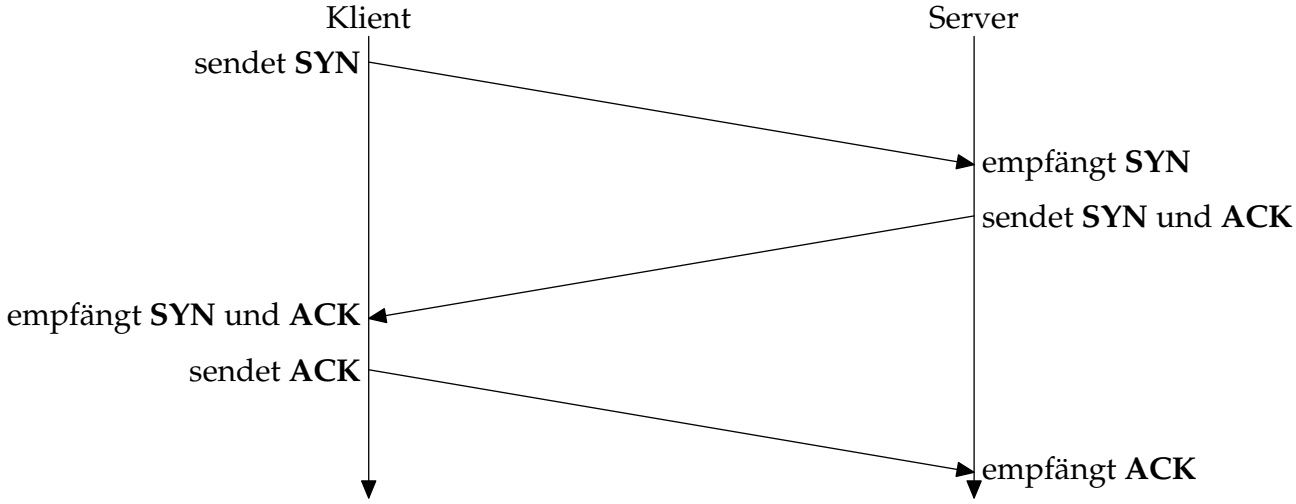
Abgabe bis zum 10. Mai 2019, 14:00 Uhr

Lernziele:

- Modellierung paralleler Prozesse mit CSP

Aufgabe 2: 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 zusammen mit dem Netzwerk erfolgreich terminieren (in CSP mit Hilfe von *SKIP*). So könnte ein beispielhafter Verlauf aussehen:

```
theon$ trace -ap handshake.csp
Acceptable: {connect, listen}
listen
Acceptable: {connect}
connect
Acceptable: {client_sends_SYN}
client_sends_SYN
Acceptable: {server_receives_SYN}
server_receives_SYN
Acceptable: {server_sends_SYN_and_ACK}
server_sends_SYN_and_ACK
Acceptable: {client_receives_SYN_and_ACK}
client_receives_SYN_and_ACK
Acceptable: {client_sends_ACK}
client_sends_ACK
Acceptable: {client_established, server_receives_ACK}
client_established
Acceptable: {server_receives_ACK}
server_receives_ACK
Acceptable: {server_established}
server_established
OK
theon$
```

Hinweis: Während es bei dem Klienten und Server offensichtlich ist, wann sie terminieren, ist dies beim Netzwerk weniger klar. Hier bietet es sich an, den $\Box$-Operator in Verbindung mit *SKIP* zu verwenden.

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 2 handshake.csp
```

Aufgabe 3: Unzuverlässiges Netzwerk

Modellieren Sie die Beziehung eines Klienten mit einem Dienst und einem unzuverlässigem Netzwerk mit einer Bandbreite von drei Paketen, die parallel behandelt werden können. Der Klient schickt bis zu drei Mal ein Anfragepaket über das Netzwerk (*send_request*). Wenn keine Antwort innerhalb einer Zeitschranke eintrifft (das wird durch die Ereignisklasse *timeout* modelliert), dann wird ein Anfragepaket erneut verschickt. Wenn der Dienst über das Netzwerk ein Anfragepaket erhält (*receive_response*), dann bearbeitet er die Anfrage und sendet eine Antwort zurück (*send_response*), die, falls das Netzwerk nicht ausfällt, vom Klienten empfangen wird (*receive_response*).

Dieses Modell findet Anwendung bei DNS (*domain name service*), das auf Basis von UDP-Paketen (UDP = *User Datagram Protocol*) arbeitet. UDP-Pakete können verlorengehen oder

auch mehrfach ankommen. Entsprechend muss jeder DNS-Klient darauf vorbereitet sein, die Anfragen zu wiederholen, wenn innerhalb einer Frist keine Antwort kommt.

So könnte ein beispielhafter Verlauf aussehen, bei dem das erste Anfragepaket beim Senden verloren geht und beim zweiten Versuch das Antwortpaket des Dienstes vom Netzwerk verschluckt wird:

```
theon$ trace -ap dnsreq.csp
Acceptable: {send_request}
send_request
Acceptable: {timeout}
timeout
Acceptable: {send_request}
send_request
Acceptable: {receive_request, timeout}
receive_request
Acceptable: {process_request, timeout}
process_request
Acceptable: {send_response, timeout}
send_response
Acceptable: {timeout}
timeout
Acceptable: {send_request}
send_request
Acceptable: {receive_request, timeout}
receive_request
Acceptable: {process_request, timeout}
process_request
Acceptable: {send_response, timeout}
send_response
Acceptable: {receive_response, timeout}
receive_response
Acceptable: {process_response}
process_response
OK
theon$
```

Sollten alle drei Versuche schiefgehen, sollte der Klient mit dem *failure*-Ereignis aufhören:

```
theon$ trace -ap dnsreq.csp
Acceptable: {send_request}
send_request
Acceptable: {timeout}
timeout
Acceptable: {send_request}
send_request
Acceptable: {timeout}
timeout
Acceptable: {send_request}
```

```
send_request
Acceptable: {timeout}
timeout
Acceptable: {failure}
failure
OK
theon$
```

Hinweise: Es empfiehlt sich, zuerst mit einem funktionierenden Netzwerk zu beginnen. Dann sollten Sie die Bandbreite des Netzwerks erhöhen, so dass mehrere Pakete gleichzeitig über das Netzwerk transportiert werden können. Sie können das z.B. mit mehreren konkurrierenden Servern und Klienten testen.

Im nächsten Schritt kehren Sie wieder zur ursprünglichen Variante mit einem Klienten und einem Dienst zurück, ergänzen aber den Klienten dahingehend, dass das *timeout*-Ereignis berücksichtigt wird und bis zu drei Versuche unternommen werden. Diese Erweiterungen können inkrementell erfolgen mit *Client1*, der einen Versuch unternimmt und nach dem *timeout* mit *failure* endet. Darauf aufbauend kann dann *Client2* geschrieben werden, der es einmal selbst versucht und bei einem *timeout* das weitere dem *Client1* überlässt usw. Beachten Sie hierbei, dass auch Antworten auf frühere Anfragen nach einem Timeout kommen können. Wenn Sie alles am Ende mit *SKIP* schön beenden wollen, müssen Sie darauf achten, dass da mindestens das Alphabet aus der Schnittmenge der Alphabeten zwischen dem Netzwerk und dem Klienten explizit angegeben wird.

Dann sollten Sie einen verlässlichen Netzwerktransport mit einem Pakete verlierenden Netzwerktransport nicht-deterministisch mit dem $\sqcap$-Operator verknüpfen.

Ihre Lösung können Sie einreichen mit

```
submit pp 3 udp.csp
```

Viel Erfolg!