Hochschule Niederrhein	PE2 / SE2	Dozenten
Fachbereich 03	Übung 13	Rethmann / Brandt
Bachelor Informatik	SoSe 2021	Stockmanns / Quix

Verteilte Systeme in C++

Lernziele

Vertiefen der Kenntnisse über Sockets und Remote Procedure Calls und anwenden der Kenntnisse auf ein etwas größeres Beispiel.

Aufgabe 13: (Sockets)

Implementieren Sie die Methoden `send` und `recv` zu der in der Vorlesung vorgestellten Wrapper-Klasse `Socket`. Beachten Sie dabei, dass Nachrichten nicht beliebig groß sein dürfen:

- Daher müsste die `send`-Methode die Nachricht zerstückeln und mittels mehrerer `write`-Aufrufe verschicken. Dazu ist es notwendig, den Rückgabewert von `write` zu überprüfen. Der eventuelle Rest, der nicht gesendet werden konnte, muss durch weitere `write`-Aufrufe versendet werden.
- Werden viele `send`-Aufrufe sehr schnell hintereinander abgesetzt, kann es sein, dass Daten zusammen in einem einzigen Paket verschickt werden. Die Nachrichten müssen getrennt werden und bei einem `recv`-Aufruf soll nur eine der Nachrichten geliefert werden. Um n Nachrichten zu lesen, sind also n Aufrufe von `recv` nötig.

Zusatz-Aufgabe Z13: (Chat)

Erstellen Sie ein Chat-Programm mit Sockets oder mit Apache Thrift. In der Vorlesung konnte leider nicht besprochen werden, wie bei Thrift auf dem Client eine Callback-Funktion angelegt werden kann, die vom Server aus aufrufbar ist. Um dennoch einen Chat mittels Thrift-RPC implementieren zu können, kann der Client mittels einer als *oneway* deklarierten Funktion dem Server seine Nachricht schicken, ein zweiter Thread auf dem Client fragt beim Server in regelmäßigen Abständen, z.B. jede Sekunde einmal, nach neuen Nachrichten.

Der Server speichert alle Nachrichten in einer Message-Queue, einer größenbeschränkten Liste. Wenn die Liste komplett gefüllt ist und eine weitere Nachricht gespeichert werden muss, wird einfach der älteste Eintrag der Liste gelöscht, um Platz für die neue Nachricht zu schaffen. Mögliches Interface in IDL:

```
struct Message {
    1:i32 _id;
```

```
    2:string _user;
    3:string _msg;
}

exception DuplicateIDException {
    1:string _msg;
}

service Chat {
    bool addClient(1:string id)
        throws  (1:DuplicateIDException ex);
    bool eraseClient(1:string id);
    oneway void sendMsg(1:string user, 2:string msg);
    list<Message> getMessages(1:i32 last_id);
}
```

Auf dem Server soll die Klasse `TThreadedServer` genutzt werden. Dadurch können mehrere Clients gleichzeitig RPC-Calls ausführen. Achten Sie auf mögliche Race-Conditions. Die Message-Queue wird von allen Threads gemeinsam genutzt und muss daher Thread-Safe sein.