Praktikum Simulative Untersuchung von Protokollfunktionen des Internet

Aufgabenblatt 1 [1]

Vorbereitung

Da Sie nur 3 Zeitstunden zur Verfügung haben, um die praktischen Aufgaben zu bewältigen, ist es wichtig, dass Sie sich gut auf den praktischen Termin vorbereiten. Daher ist die folgende Vorbereitung **vor jedem** Labortermin obligatorisch:

1. Versuchen Sie sich das Grundwissen zu der jeweiligen Aufgabenstellung (mittels Büchern, Wikipedia, etc.) anzueignen, sodass Sie in der Lage sind, die einzelnen Programmiertätigkeiten in den Kontext des untersuchten Verfahrens einzuordnen.

2. Lesen Sie sich das aktuelle Aufgabenblatt (einschließlich der praktischen Aufgaben) gut durch und notieren Sie sich etwaige Fragen, um diese gleich zum Anfang des Labortermins stellen zu können.

3. Lösen Sie die Vorbereitungsaufgaben. Die Vorbereitungsaufgaben sind von **jedem einzelnen** Studierenden zu bearbeiten (nicht nur einmal pro Gruppe) und vor dem praktischen Termin dem Betreuer per Email zu schicken.

4. Überlegen Sie, wie die praktischen Aufgaben gelöst werden könnten.

Eine sorgfältige Vorbereitung und die Bearbeitung der Aufgabenblätter ist zwingende Voraussetzung für die erfolgreiche Teilnahme am Projektseminar.

Die praktischen Lösung der Aufgabenblätter sind spätestens einen Labortermin nach dem entsprechenden praktischen Termin dem Betreuer zu zeigen. Wir behalten uns vor, die Vorbereitung durch mündliche Rücksprachen zu überprüfen.

Aufgabe 1: C++
Warum ist es “common practice” nach dem Freigeben von Speicher den entsprechenden Pointer auf `NULL` zu setzen? Betrachten Sie bitte folgendes Beispiel:

```
1 cMessage* msg = new cMessage();
2 delete msg;
3 msg = NULL;
4 delete msg;
```

Was ist der Effekt des Kommandos in Zeile 4, wenn Zeile 3 wie gezeigt ausgeführt wird und wenn Zeile 3 auskommentiert würde?

Aufgabe 2: C++
Warum wird der folgende Codeschnipsel vom Compiler nicht akzeptiert? Wie müssten Sie den Code ändern, damit er akzeptiert würde? Wann wird der Destructor von `msg` aufgerufen?

```
1 {
2     cMessage msg;
3     msg.setName("MyMessage");
4 }
5 send(msg, "out");
```

[1] Stand: 12. Oktober 2023

Aufgabe 3: C++
Wird der folgende Codeschnipsel vom Compiler akzeptiert und wird das Programm stets korrekt laufen? Begründen Sie bitte Ihre Antwort. Überlegen Sie dafür, welche Typen die Ausdrücke `msg` und `&msg` haben und welchen Zweck die `delete`-Operation hat.

```
1 cMessage msg;
2 msg.setName("MyMessage");
3 delete &msg;
```

Aufgabe 4: C++
In OMNeT++ benutzt man in der Regel Vererbung, um von der Klasse `cMessage` eigene Nachrichtentypen mit zusätzlichen Feldern abzuleiten. Das OMNeT++-Framework liefert jedoch jeweils nur Pointer auf Nachrichten vom Typ `cMessage*` zurück.

Nehmen Sie an, Sie hätten mittels `receive()` eine Nachricht erhalten, von der Sie wissen, dass diese vom Typ `NetworkPacket*` ist und unter anderem die Methode `getDestAddr()` zur Verfügung stellt. Wie können Sie diese Funktion aufrufen, wenn `receive()` standardmäßig immer einen Pointer vom Typ `cMessage*` zurückliefert, wie z.B. in folgendem Codeschnipsel:

```
1 cMessage* msg = receive();
```

Aufgabe 5: OMNeT++
OMNeT++ bietet zwei verschiedene Möglichkeiten, die Funktionalität von Modulen zu implementieren: die Coroutinen-basierte mit `activity()` und die ereignisgesteuerte mit `handleMessage()`. Was ist der Unterschied zwischen den beiden Varianten? Warum benötigt das Modul für `activity()` einen eigenen Stack?

Praktischer Teil

Die praktischen Aufgaben sind innerhalb des Labortermins von jeder Gruppe zu erfüllen. Die Antworten auf die zugehörigen Fragen sind einmal pro Gruppe zu beantworten.

Denken Sie bitte bei den praktischen Aufgaben immer an aussagekräftige Debug-Ausgaben. Diese erleichtern die Fehlersuche ungemein. Hierbei können Sie sich z.B. aktuelle Belegungen von Variablen ausgeben lassen, oder prüfen ob ihr Code überhaupt angesprungen wird.

Aufgabe 6: Simulationen kompilieren und ausführen
In der Einführung wurde Ihnen ein einfaches Beispiel (die erste Aufgabe) vorgeführt. Die Quelldateien für diesen Versuch finden Sie im Unterverzeichnis `protsim/protsim01`. Analog dazu befinden sich die restlichen Quellen der Aufgaben in `protsim/protsim02...12`. Jede praktische Aufgabe wird also in einem eigenen Verzeichnis bearbeitet. Dabei kommt es in späteren Versuchen vor, dass Teile des Quellcodes in die nächste Aufgabe zu übernehmen sind.

1. Wechseln Sie nun mit `cd protsim/protsim01` in das entsprechende Verzeichnis.
2. Führen Sie `./build.sh` aus, um die nötigen Makefiles zu generieren und das Projekt zu bauen.
3. Starten sie nun die ausführbare Simulation mit `./run.sh`
4. Wie lange braucht ein Paket vom Sender zum Empfänger?
5. Können Sie sich anhand der Dateien `networks/ExampleNetwork.ned` und `userapps/ExampleSender.cc` erklären, wie es zu dieser Zeit kommt?

Aufgabe 7: Simulationen ändern
Ändern Sie das obige Beispiel so ab, dass

1. der Empfänger (`userapps/ExampleReceiver.cc`) die Pakete nicht löscht, sondern mittels der Methode `setName()` auf den Namen `Pong` umbenennt und dann zurückschickt;
2. der Sender (`userapps/ExampleSender.cc`) das ID-Feld der Nachricht mit der Funktion `setId()` auf einen zu initialisierenden und mit jeder Nachricht zu inkrementierenden Wert setzt;
3. der Sender die `Pong`-Nachricht empfängt, den Namen und das ID-Feld mit `getName()` und `getId()` ermittelt und ausgibt, diese Nachricht anschließend löscht und erst dann ein neues Paket generiert und losschickt.

Kompilieren Sie bitte das geänderte Programm und führen Sie es aus. Wie lange dauert es, bis ein Paket zurück beim Sender ist? In welcher Datei werden die mit `NetworkPacket` assoziierten zusätzlichen Funktionen wie `getDestAddr()` definiert?