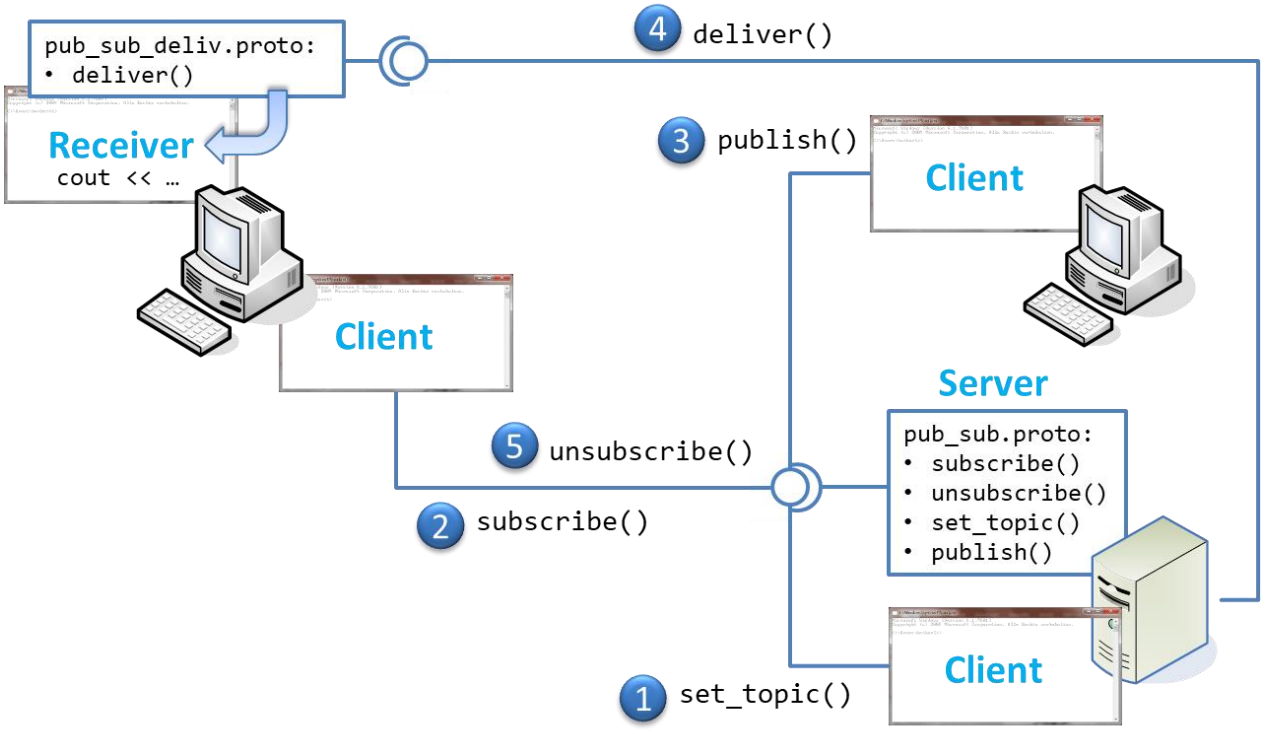
Aufgabenblatt 4

Publish-Subscribe-System mit RPCs

Auf diesem Aufgabenblatt soll ein System zur Nachrichtenverteilung per RPC realisiert werden.

Ganz Konkret soll ein RPC-basiertes System mit Google RPC (gRPC) in C++ implementiert werden, welches es einem oder mehreren Sendern (*Publisher*) erlaubt, Nachrichten an einen Server zu schicken. Diese werden von dort aus an eine Menge zuvor registrierter Empfänger (*Subscriber*) weitergeleitet. Weiterhin ist es möglich, ein Thema / Topic zu setzen, das als Präfix vor die ausgelieferte geschrieben wird. Auf diese Weise ist es möglich, unterschiedliche Themen-spezifische Server zu starten.

Zur Verdeutlichung der Abläufe kann das folgende Diagramm mit den Komponenten und deren Schnittstellen dienen:



- Der eigentliche Server (im folgenden auch *Dispatcher* genannt) sorgt für die Nachrichten-Verteilung und implementiert das Interface **`pub_sub.proto`**. Es enthält Funktionen zur Registration / Deregistration von Empfängern, die in einer geeigneten Datenstruktur verwaltet werden, sowie zum Veröffentlichen von Nachrichten. Per **`set_topic()`** kann ein Topic gesetzt werden, welches in der ausgelieferten Nachricht jeweils mitgeschickt wird. Nachrichten können per Aufruf von **`publish()`** verschickt werden. Das Setzen eines Topics soll nur möglich sein, wenn zuvor ein korrekter Passcode eingegeben wurde.

- Der Empfang von Nachrichten erfolgt pro Client durch einen separaten Server (*Receiver*), der in einem eigenen Prozess läuft und die Schnittstelle **`pub_sub_deliv.proto`** implementiert. Die über die Funktion **`deliver()`** empfangenen Nachrichten werden der Einfachheit halber

zusammen mit einem bezogen auf den Client lokalen Zeitstempel auf der Console ausgegeben.

- Die Steuerung des Nachrichtenempfangs erfolgt über einen *Client*, der zusammen mit dem Receiver auf demselben Host läuft. Dieser ruft das Interface des Dispatchers auf, indem er
    - nach dem Start des Receivers, den lokalen Rechnern per **`subscribe()`** für den Nachrichtenempfang registriert
    - und diesen nach Beendigung des Receivers per **`unsubscribe()`** deregistriert.
    - Daneben initiiert der Client die Auslieferung von Nachrichten, sowie das Setzen der Topics.

Testen Sie Ihre Lösungen auch untereinander, in dem Ihr Client Kontakt mit dem Server e*iner anderen Gruppe* aufnimmt und umgekehrt. Dokumentieren Sie die Testfälle im Protokoll.

Beantworten Sie im Protokoll darüber hinaus die folgenden **Fragen**:

1. Receiver und Client sind voneinander getrennte Prozesse. Warum ist dies so?
2. Handelt es sich um ein synchron oder asynchron arbeitendes System? Woran machen Sie das fest?
3. Die Registrierung / De-Registrierung erfolgt über IP-Adressen. gRPC arbeitet mit http als Transport-Protokoll und kann auch zum Aufrufen von Diensten im Internet (evtl. Cloud) genutzt werden. Welche Probleme können dabei auftreten?

Hinweise:

- Die oben beschriebenen Schnittstellen (**`.proto`**-Dateien) werden vorgegeben. Sie finden diese im Lernraum zur Veranstaltung. Sie sind frei in der sonstigen Umsetzung des Projektes. Sie können für das Einrichten der notwendigen Projekte auf die Bespiele aus der Vorlesung zurückgreifen. Achten Sie aber darauf, dass Sie die **Bezeichner sinnvoll abändern**!
- Es ist möglich, die mitgelieferte Projekt-Schablone (Makefile-Projekt) zu verwenden. Die Stellen, an denen Ihre Ergänzungen integriert werden müssen, sind gekennzeichnet (‚**`TODO`**'). Beachten Sie, dass der Lerneffekt dabei geringer ausfällt. In der Schablone ist das automatische Starten und Beenden des Receivers schon umgesetzt.
- Bei der Implementierung Verteilter Systeme sollte man generell auf den sparsamen Umgang mit Netzwerkressourcen achten. In der vorliegenden Aufgabe wird dem dadurch Rechnung getragen, dass die Rückgabewerte der Funktionen des Interfaces **`pub_sub.proto`** in einen Aufzählungstyp kodiert werden. Übergeordnete Definitionen finden Sie in **`pub_sub_common.proto`**.
- Als **freiwillige Zusatzaufgabe** können Sie sich überlegen, wie man den Receiver als Kindprozess des Clients vor der Registration automatisch starten (**`fork`**) und vor Deregistration (**`kill`**) beenden kann.

- Speicherfehlern kommen Sie mit dem Tool **`valgrind`** auf die Spur.
  Aufruf: **`valgrind –v <executable>`**

Für das Testat ist ein Protokoll bekannter Formatierung (siehe Blatt 1) inkl. der durchgeführten Tests vorzulegen.

Testierung: 25./26.4.2023