*Verteilte Systeme im Sommersemester 2021*

Steffen Herweg, Matr. Nr. 873475

Luca Fabio Kock, Matr. Nr. 879534 Osnabrück, 13.05.2021

# Aufgabenblatt 4

Receiver und Client sind voneinander getrennte Prozesse. Warum ist dies so?

In der hier vorgegebenen Struktur sind Client und Receiver getrennt. Deshalb erzeugt der Client einen neuen Prozess für den Receiver.

-Handelt es sich um ein synchron oder asynchron arbeitendes System? Bei asynchronen Systemen hatten über Umsetzungsalternativen gesprochen. Ggf.: welche Variante liegt hier vor?

Synchron

-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?

IP-Adresse eines Receivers kann sich ändern

Wir haben die gegebene Vorlage genutzt und die einzelnen To-Do’s umgesetzt:

Pub\_sub\_Client.cc:

    static std::string stringify(pubsub::ReturnCode\_Values value)

    {

        switch (value)

        {

        case pubsub::ReturnCode\_Values\_OK:

            return "OK";

        case pubsub::ReturnCode\_Values\_CANNOT\_REGISTER:

            return "Cannot Register";

        case pubsub::ReturnCode\_Values\_CLIENT\_ALREADY\_REGISTERED:

            return "Client Already Registered";

        case pubsub::ReturnCode\_Values\_CANNOT\_UNREGISTER:

            return "Cannot Unregister";

        case pubsub::ReturnCode\_Values\_CANNOT\_SET\_TOPIC:

            return "Cannot Set Topic";

        case pubsub::ReturnCode\_Values\_UNKNOWN\_ERROR:

            return "Unknown Error";

        default:

            return "";

        }

    }

 /\* TODO: Hier den Request verschicken und Ergebnis auswerten! \*/

                // Platzhalter fuer Request, Kontext & Reply.

                // Muss hier lokal definiert werden,

                // da es sonst Probleme mit der Speicherfreigabe gibt.

                Topic request;

                ReturnCode reply;

                // Kontext kann die barbeitung der RPCs beeinflusst werden. Wird nicht genutzt.

                ClientContext context;

                // TODO: Topic fuer Server vorbereiten ...

                request.set\_passcode(passcode.c\_str());

                request.set\_topic(topic.c\_str());

                // TODO: RPC abschicken ...

                Status status = stub\_->set\_topic(&context, request, &reply);

                // Status / Reply behandeln

                this->handle\_status("set\_topic()", status, reply);

 /\* TODO: Hier den Request verschicken und Ergebnis auswerten! \*/

                    /\* Platzhalter wie oben lokal erstellen ... \*/

                    ClientContext clientContext;

                    SubscriberAddress request;

                    ReturnCode response;

                    // TODO: Receiver Adresse setzen ...

                    request.set\_ip\_address(get\_receiver\_ip());

                    request.set\_port(40041);

                    // TODO: RPC abschicken ...

                    Status status = stub\_->subscribe(&clientContext, request, &response);

                    // TODO: Status / Reply behandeln ...

                    this->handle\_status("subscribe()", status, response);

/\* TODO: Hier den Request verschicken und Ergebnis auswerten! \*/

                /\* Platzhalter wie oben lokal erstellen ... \*/

                ClientContext clientContext;

                SubscriberAddress request;

                ReturnCode response;

                // TODO: Receiver Adresse setzen ...

                request.set\_ip\_address(get\_receiver\_ip());

                request.set\_port(40041);

                // TODO: RPC abschicken ...

                Status status = stub\_->unsubscribe(&clientContext, request, &response);

                // TODO: Status / Reply behandeln ...

                this->handle\_status("unsubscribe()", status, response);

/\* TODO: Hier den Request verschicken und Ergebnis auswerten! \*/

                /\* Platzhalter wie oben lokal erstellen ... \*/

                ClientContext clientContext;

                Message request;

                ReturnCode response;

                // TODO: Message setzen ...

                request.set\_message(cmd.c\_str());

                // TODO: RPC abschicken ...

                Status status = stub\_->publish(&clientContext, request, &response);

                // TODO: Status / Reply behandeln ...

                this->handle\_status("publish()", status, response);

Pub\_Sub\_Receiver.cc:

std::string create\_timestamp(){

  time\_t rawtime;

  struct tm \* timeinfo;

  char buffer[80];

  time (&rawtime);

  timeinfo = localtime(&rawtime);

  strftime(buffer,sizeof(buffer),"%d-%m-%Y %H:%M:%S",timeinfo);

  return std::string(buffer);

}

// Implementierung des Service

class PubSubDelivServiceImpl final : public PubSubDelivService::Service {

  Status deliver(ServerContext\* context, const Message\* request,

                EmptyMessage\* reply) override {

    // TODO: Zeitstempel erzeugen und zusammen mit Nachricht ausgeben.

    // ...

    std::cout << create\_timestamp() << " " << request->message() << std::endl;

    return Status::OK;

  }

};

Pub\_Sub\_Server.cc:

// TODO: Channel topic und Subscribers für diesen Server merken

  // ...

  std::string topic;

  std::map<std::string,std::unique\_ptr<PubSubDelivService::Stub>> subscribers;

Status subscribe(ServerContext \*context, const SubscriberAddress \*request,

                   ReturnCode \*reply) override

  {

    std::string receiver = stringify(\*request);

    bool created = subscribers.emplace(receiver, PubSubDelivService::NewStub(grpc::CreateChannel(receiver, grpc::InsecureChannelCredentials()))).second;

    if(created){

      reply->set\_value(pubsub::ReturnCode\_Values\_OK);

    }else{

      reply->set\_value(pubsub::ReturnCode\_Values\_CANNOT\_REGISTER);

    }

    return Status::OK;

  }

  Status unsubscribe(ServerContext \*context, const SubscriberAddress \*request,

                     ReturnCode \*reply) override

  {

    std::string receiver = stringify(\*request);

    int removed = subscribers.erase(receiver);

    if(removed > 0){

       reply->set\_value(pubsub::ReturnCode\_Values\_OK);

    }else{

      reply->set\_value(pubsub::ReturnCode\_Values\_CANNOT\_UNREGISTER);

    }

    return Status::OK;

  }

Status publish(ServerContext \*context, const Message \*request,

                 ReturnCode \*reply) override

  {

    // TODO: Nachricht an alle Subscriber verteilen

    ClientContext clientContext;

    EmptyMessage empty;

    Message requestOut;

    requestOut.set\_message((topic + ": " + request->message()));

    for (auto& subscriberPair : subscribers) {

       Status status = subscriberPair.second->deliver(&clientContext, requestOut, &empty);

       handle\_status("deliver()",status);

    }

    reply->set\_value(pubsub::ReturnCode\_Values\_OK);

    return Status::OK;

  }

  Status set\_topic(ServerContext \*context, const Topic \*request,

                     ReturnCode \*reply) override

  {

    if(request->passcode().compare(PASSCODE) == 0){

      // TODO: Topic setzen und Info ausgeben

      topic = request->topic();

      reply->set\_value(pubsub::ReturnCode\_Values\_OK);

      Message message;

      ReturnCode publishReply;

      message.set\_message(std::string("Topic Changed to ") + topic);

      publish(context,&message,&publishReply);

    }

    else{

      reply->set\_value(pubsub::ReturnCode\_Values\_CANNOT\_SET\_TOPIC);

    }

    return Status::OK;

  }

public:

  PubSubServiceImpl()

  {

    // TODO: Topic initialisieren

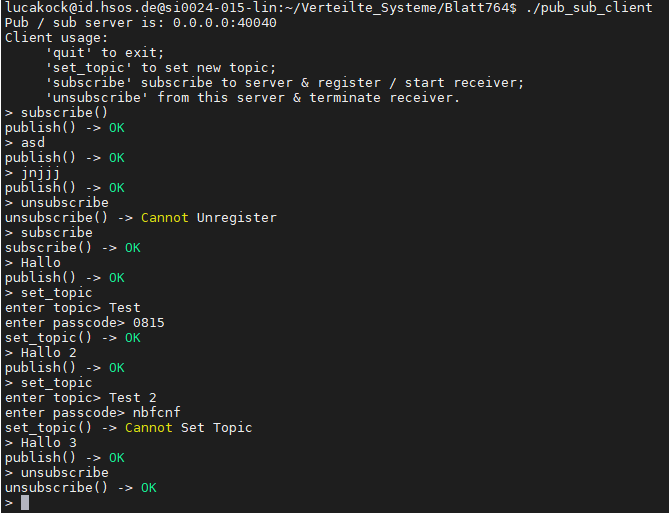
        topic = "<no topic set>";

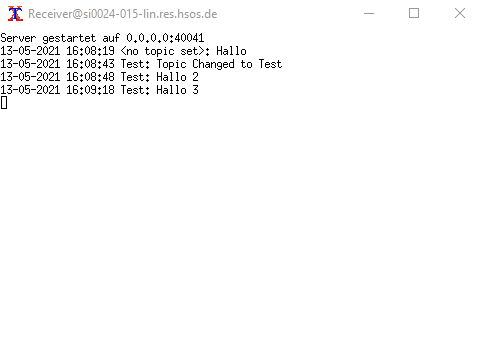
  }

};

**Tests**

Ausgabe auf dem Client-Terminal:



Ausgabe auf dem Receiver: