

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>";
    }
};

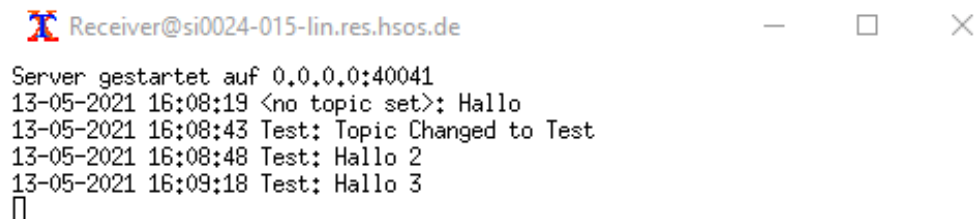
```

Tests

Ausgabe auf dem Client-Terminal:

```
lucakock@id.hsos.de@si0024-015-lin:~/Verteilte_Systeme/Blatt764$ ./pub_sub_client
Pub / sub server is: 0.0.0.0:40040
Client usage:
  'quit' to exit;
  'set_topic' to set new topic;
  'subscribe' subscribe to server & register / start receiver;
  'unsubscribe' from this server & terminate receiver.
> subscribe()
publish() -> OK
> asd
publish() -> OK
> jnjnj
publish() -> OK
> unsubscribe
unsubscribe() -> Cannot Unregister
> subscribe
subscribe() -> OK
> Hallo
publish() -> OK
> set_topic
enter topic> Test
enter passcode> 0815
set_topic() -> OK
> Hallo 2
publish() -> OK
> set_topic
enter topic> Test 2
enter passcode> nbfcnf
set_topic() -> Cannot Set Topic
> Hallo 3
publish() -> OK
> unsubscribe
unsubscribe() -> OK
> █
```

Ausgabe auf dem Receiver:



```
Receiver@si0024-015-lin.res.hsos.de
Server gestartet auf 0.0.0.0:40041
13-05-2021 16:08:19 <no topic set>: Hallo
13-05-2021 16:08:43 Test: Topic Changed to Test
13-05-2021 16:08:48 Test: Hallo 2
13-05-2021 16:09:18 Test: Hallo 3
█
```