实验课程：分布式系统

实验名称：第三次作业

专业名称：计算机科学与技术

学生姓名：吴臻

学生学号：21307371

班级：计科3班

问题描述

> 使用protobuf和gRPC等远程过程调用的方法实现消息订阅（publish-subscribe）系统
> 1.该订阅系统能够实现简单的消息传输
> 2.客户端可以查看服务器端主题并订阅，可以向服务器端发送消息；同时服务器端收到订阅请求后当收到新消息时转发给所有订阅的客户端
> 3.还可以控制消息在服务器端存储的时间。
> 编程语言不限，但是推荐使用python和C；

解决方案

搭建环境

- 配置python

```
wz@wz-VirtualBox:~$ python3 --version
Python 3.9.0
```

- 安装protobuf和gRPC

```
sudo pip3 install grpcio-tools
```

```
Installing collected packages: grpcio, protobuf, grpcio-tools
Successfully installed grpcio-1.59.0 grpcio-tools-1.59.0 protobuf-4.24.4
```

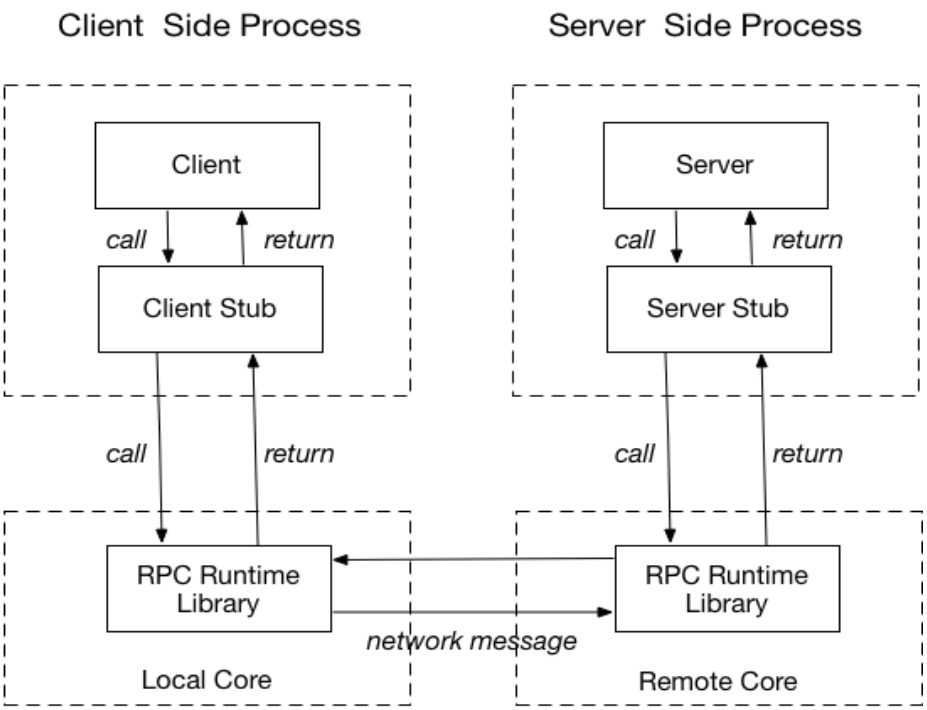
1. **protobuf**是Google公司提出的一种轻便高效的结构化数据存储格式，常用于结构化数据的序列化，具有语言无关、平台无关、可扩展性特性，常用于通讯协议、服务端数据交换场景
   protobuf的核心内容包括：

- 定义消息：消息的结构体，以message标识。
- 定义接口：接口路径和参数，以service标识。
  通过protobuf提供的机制，服务端与服务端之间只需要关注接口方法名（service）和参数（message）即可通信，而不需关注繁琐的链路协议和字段解析

2. **gRPC**一开始由 Google 开发，是一款语言中立、平台中立、开源的远程过程调用(RPC)系统

- 定义一个服务，指定其能够被远程调用的方法（包含参数和返回类型）。在服务端实现这个接口，并运行一个 gRPC 服务器来处理客户端调用。在客户端拥有一个存根能够像服务端一样的方法。

3. **远程过程调用**



Remote Procedure Call Flow

实验步骤

- proto文件
  i. `syntax = "proto3";`
  - 这是 Protocol Buffers（ProtoBuf）定义文件的语法版本，使用的是 proto3 版本。

  ii.

```
service Pubsub {
rpc publish(publishRequest) returns (reply) {}
rpc browse(browseRequest) returns (stream reply) {}
rpc subcribe(subRequest) returns (stream reply) {}
}
```

- 这定义了一个 gRPC 服务，名为 Pubsub，该服务包含以下三个RPC方法：publish，browse，和 subcribe。
- **publish**:接受一个 publishRequest 消息作为输入参数，用于发布消息,返回一个 reply 消息，表示发布操作的响应.
- **browse**:接受一个 browseRequest 消息作为输入参数，用于浏览消息。返回一个流（stream）的 reply 消息，允许客户端持续接收消息，用于浏览操作的响应。
- **subcribe**:接受一个 subRequest 消息作为输入参数，用于订阅消息。返回一个流的 reply 消息，允许客户端持续接收消息，用于订阅操作的响应。

iii.

```
message publishRequest {
    string topic = 1;
    string context = 2;
    string id = 3;
}
```

- 这是一个消息类型的定义，用于发布消息的请求。
  它包含三字段：topic，context和id，分别表示消息的主题、内容和发布用户。

iv.

```
message reply {
string message = 1;
}
```

- 用于表示通用的响应消息。
  它包含一个字段：message，表示响应消息的内容。

v.

```
message browseRequest {
string topic = 1;
}
```

- 用于浏览消息的请求。包含一个字段：topic，表示要浏览的消息主题。

vi.

```
message subRequest {
string topic = 1;  // 主题
string clientId = 2;  // 客户端ID
int32 TTL = 3;  // 订阅有效期
}
```

- 用于订阅消息的请求。
  它包含三个字段：
  topic 表示要订阅的消息主题。
  clientId 表示客户端的唯一标识。
  TTL 表示订阅的有效期（Time To Live）

client

i. **导入必要的模块**：

```
import grpc
import time
import threading
import grpc_pb2
import grpc_pb2_grpc
```

- 这些导入语句用于导入必要的 Python 模块，包括 gRPC 库以进行通信，时间模块用于添加延时，以及线程模块用于创建独立线程以进行消息订阅。

ii. **设置客户端**：

```
clientId = input("Input Id: ")
channel = grpc.insecure_channel('localhost:50051')
stub = grpc_pb2_grpc.PubsubStub(channel)
```

- clientId 用于存储用户提供的客户端标识。
- channel 建立了到运行在本地主机的端口50051上的gRPC服务器的连接。
- stub 创建了一个 Pubsub 服务的 gRPC 存根（stub），允许客户端调用该服务的 RPC 方法。

iii. **定义 publish 函数**:

```
def publish(topic, context):
    print("Publishing message in {}:{}".format(topic, context))
    res = stub.publish(grpc_pb2.publishRequest(topic=topic, context=context, id = cli
    print(res.message)
    print('')
```

- publish 函数接受 topic 和 context 作为参数，分别表示主题和消息内容。
- 它调用 publish RPC 方法，使用提供的主题和内容。
- 它打印一条消息，指示消息已经发布，并显示从服务器接收到的响应消息。

iv. **定义 browse 函数**:

```
def browse(topic):
    print("Browsing topic {} at {}".format(topic,time.time()))
    res = stub.browse(grpc_pb2.browseRequest(topic=topic))
    for msg in res:
        print(msg.message)
    print('')
```

- browse 函数接受 topic 作为参数，表示要浏览的主题。
- 它调用 browse RPC 方法，使用提供的主题。
- 它打印一条消息，指示正在浏览指定主题，并迭代处理从服务器返回的消息。

v. **定义 sub_rec 函数和 subcribe 函数**:

```
def sub_rec(topic, TTL):
    for msg in stub.subcribe(grpc_pb2.subRequest(topic=topic, clientId=clientId, TTL=
        print("Receive message from {}:{}".format(topic, msg.message))

def subcribe(topic, TTL=20):
    print("Subscribed {} successfully at {}.".format(topic, time.time()))
    thrd = threading.Thread(target=sub_rec, args=(topic, TTL))
    thrd.start()
```

- **sub_rec** 函数用于实际订阅消息。它接受 topic 和 TTL 作为参数，表示要订阅的主题和订阅的最大持续时间。它在订阅期间迭代处理从服务器返回的消息，并打印消息内容。
- **subcribe** 函数用于启动消息订阅线程。它接受 topic 和可选的 TTL 参数，表示要订阅的主题和订阅的最大持续时间，默认为20秒。在开始订阅时，它打印一条消息表示成功订阅。

server

i. **导入必要的模块：**

```
from threading import Event
from concurrent import futures
import time
import grpc
import grpc_pb2
import grpc_pb2_grpc
```

- threading：用于多线程操作。
  concurrent.futures：用于创建线程池。
  time：用于处理时间相关的操作。
  grpc：用于实现 gRPC 通信。
  grpc_pb2 和 grpc_pb2_grpc：这两个模块是根据 Protocol Buffers 文件生成的 gRPC 代码，用于定义消息和服务。

ii. **定义 Pubsub 类**(一个本地的消息发布与订阅系统,它包含了一个 storage 字典，用于存储不同主题的消息，以及一个 subscribe_list 字典，用于处理订阅事件)

a. **定义publish函数**

```
def publish(self, topic, message, cid):
    msg = ""
    if topic not in self.storage:
        self.storage[topic] = [{'createTime': time.time(), 'message': message, 'i
        msg += "create topic: {}\n".format(topic)
    else:
        self.storage[topic].append({'createTime': time.time(), 'message': message
    if topic in self.subscribe_list:
        for client in self.subscribe_list[topic]:
            self.subscribe_list[topic][client].set()
    msg += "publish successful"
    return msg
```

- 用于发布消息到指定主题，将新发布的消息添加到特定主题的消息存储中，记录了消息的创建时间和内容，如果主题不存在则创建主题。它还通知订阅该主题的客户端

b. **定义browse函数**

```
def browse(self, topic):
    if topic not in self.storage:
        return ["topic not created"]
    for msg in self.storage[topic]:
        yield self.gen_msg(msg)
```

- 允许用户浏览特定主题中的所有消息，并将它们逐个返回，以便查看或处理

c. **定义subcribe函数**

```
def subcribe(self, topic, clientId, TTL=20):
    if topic not in self.subscribe_list:
        self.subscribe_list[topic] = {}
    self.subscribe_list[topic][clientId] = Event()
    createTime = time.time()
    remainTime = TTL
    while True:
        self.subscribe_list[topic][clientId].wait(remainTime)
        remainTime = TTL - (time.time() - createTime)
        if remainTime <= 0:
            break
        yield self.gen_msg(self.storage[topic][-1])
        self.subscribe_list[topic][clientId].clear()
```

- 让客户端订阅特定主题，并等待新消息的到达，然后通过 generator 返回新消息。如果 TTL 到期或客户端取消订阅，订阅会结束。

d. **定义refresh函数**

```
def refresh(self, TTL=10):
    ddl = time.time() - TTL
    for topic in self.storage:
        while len(self.storage[topic]) and self.storage[topic][0]['createTime'] <
            del self.storage[topic][0]
```

- 定期刷新存储的消息，删除超过指定存储时间的消息，以控制消息在服务器上的存储时间

e. **main函数**

```
if __name__ == '__main__':
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    pubsubServe = PubsubService()
    grpc_pb2_grpc.add_PubsubServicer_to_server(pubsubServe, server)
    server.add_insecure_port('[::]:50051')
    server.start()
    try:
        while True:
            time.sleep(1)
            pubsubServe.pubsub.refresh()
    except KeyboardInterrupt:
        server.stop(0)
```

- 在 main 函数中，创建 gRPC 服务器，并将 PubsubService 类的实例添加到服务器中。然后，服务器以不安全的方式绑定到端口 50051 并启动。
- 服务器在一个无限循环中运行，每秒调用一次 refresh 方法，以清理过期的消息。如果按下键盘上的 Ctrl+C，服务器将在执行完当前任务后停止。

测试程序（客户端）

```
publish('test_topic', 'message1')
browse('test_topic')
time.sleep(5)
print("5 seconds passed")
print('')

subcribe('test_topic', 10)
publish('test_topic', 'message2')
time.sleep(7)
print("7 seconds passed")
print('')
browse('test_topic')

time.sleep(5)
print("5 seconds passed")
print('')
publish('test_topic', 'message3')
```

- 运行
  i. 利用proto文件生成py文件

  ```
  python3 -m grpc_tools.protoc -I./ --python_out=. --grpc_python_out=. ./grpc.proto
  ```

  - _pb2.py文件会对每一个message进行对应的信息存储，在处理service和client文件的时候会使用
  - _pb2_grpc.py用来存储每个服务的server与client以及注册server的工具

  ii. 运行服务器

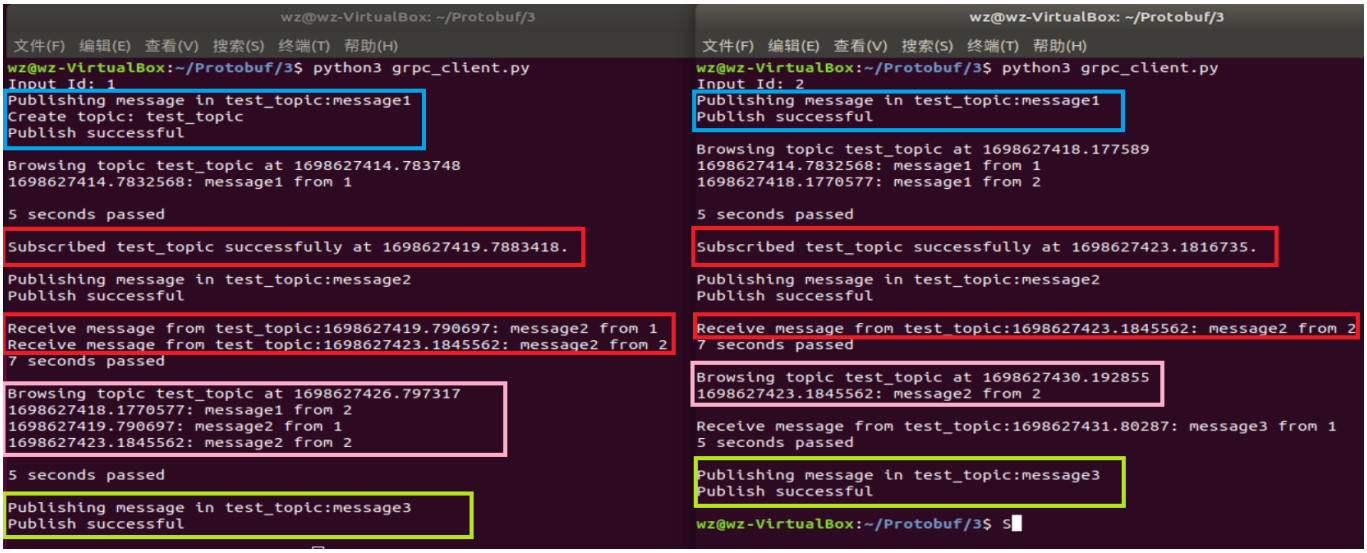
  ```
  python3 grpc_server.py
  ```

  iii. 分别运行两个客户端

  ```
  python3 grpc_client.py
  ```

实验结果

- 参数设定/提示：
  - 默认消息存放10秒
  - 订阅（设置为10秒），输出内容为创建时间+内容+发布用户
  - 测试代码只发布关于 `test_topic` 的主题



> 分析：两个用户分别和服务器建立联系（间隔大概是4秒），两个用户运行同一套代码
> 1.在蓝色方框内，两个用户都发送message1，因为是首次发送关于这个主题的内容，所以用户1需要先创建这个主题，而用户2则不需要。
> 2.之后，两个用户分别浏览该主题，因为用户1比较早运行，所以只能看到自己发布的。

> 3.红色方框内为两个用户订阅该主题，用户1订阅时间为419，用户2为423，订阅之后，用户1收到自己和用户2发送的message2，用户2则只收到自己发送的message2。
> 4.在粉色方框内，两个用户再次浏览该主题，到目前为止，两个用户一共发送了4条message，可以发现，用户1还能看到三条信息（**消息存放10秒**），用户1发送的message1已经过期且被删除了；而用户2只能看到自己发的最后一条message，其他都过期了。
> 5.在绿色方框内，用户1发送message3，因为此时用户2的订阅时间还没结束（订阅10秒），所以用户2能收到用户1的message3，但是用户1的订阅时间已经超过10秒，所以用户1不会收到该消息
> 6.最后，用户2自己发送message3，但是订阅时间已经超过10秒，所以用户2不会收到该消息

遇到的问题及解决方法

1. **python安装**:一开始运行python文件时，报错显示：没有ctypes，后来参考相关资料才解决，运行 `sudo apt-get install libffi-dev`
2. **代码编写**：因为涉及服务器和客户端交互式运行，所以在编写时难以调试，经常遇到一些难以理解的问题，后来在网上参考了许多代码，成功复现这些代码，并加上了自己设计的一些小功能

相关资料

1. python3.9版本安装：https://blog.csdn.net/u012080686/article/details/112600252
   （需要自行运行 `sudo apt-get install libffi-dev` 来解决没有ctypes的问题）
2. protobuf和gRPC：https://blog.csdn.net/maylcc/article/details/103558367
3. 远程调用：https://blog.csdn.net/newlw/article/details/124842412
4. 代码参考：https://github.com/He-Ze/Distributed-System-SYSU/blob/main/作业/作业3/README.md