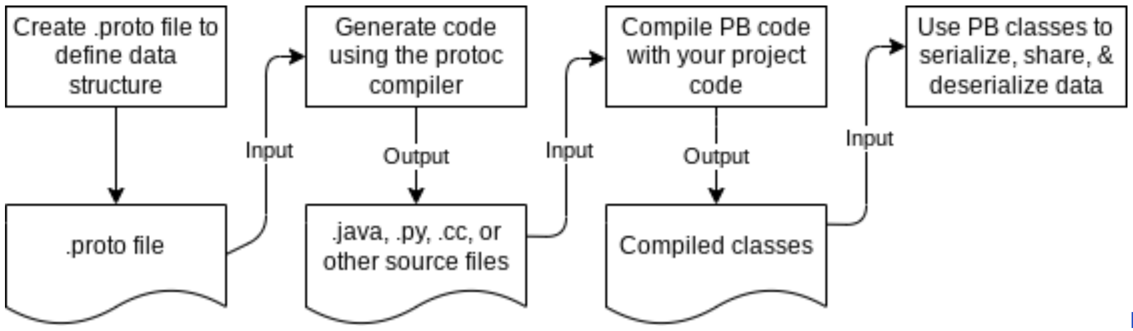
Client Server communication (2022 OS)

In this part of the project, we will implement the RPC interface on the server program to handle incoming client RPC requests. At the moment, you don’ t need to implement the actual server logic. You only need to print the messages to indicate the receipt of client requests and the sending of server responses.

1. Basic Idea

Google Protocol Buffer

We use Google’ s Protocol Buffers to serialize/deserialize the messages. Besides, we will use `proto2` (the second version of Protocol Buffer) in this FTP project.



Create .proto file to define data structure
Generate code using the protoc compiler
Compile PB code with your project code
Use PB classes to serialize, share, & deserialize data
Input
Output
Input
Output
Input
.proto file
.java, .py, .cc, or other source files
Compiled classes

How do Protocol Buffers Work?

gRPC

For the communication between the FTP server and the FTP clients, we will use Google gRPC. gRPC is a Remote Procedure Call (RPC) framework. Assume that program X and program Y run on two separate machines connected by the network. The RPC framework allows program X to invoke functions in program Y. This saves the troubles of network programming. A good starting point for learning gRPC is gRPC Basic Tutorial.

(We use the synchronous API in this part of the project)

Environment Setup (Recap)

Recommend OS distro: Ubuntu 20.04 LTS.

Please refer to this guide to install gRPC, or follow the instructions down below. **Besides, if you already installed gRPC in HW1, you don’t need to setup the environment again!**

```
sudo apt-get update && apt-get upgrade
sudo apt install -y git cmake build-essential autoconf libtool pkg-
config

cd ~
git clone --recurse-submodules -b v1.46.3 --depth 1 \
          --shallow-submodules https://github.com/grpc/grpc

cd grpc
mkdir -p cmake/build
pushd cmake/build
cmake -DgRPC_INSTALL=ON \
      -DgRPC_BUILD_TESTS=OFF \
      ../..
make -j 4
sudo make install
popd
```

2. Implementation

In this section, we start to build our FTP server. Since our FTP system uses gRPC for the server and client to communicate, defining the `.proto` file is the foundation of the system. We will first define the prototypes of all the RPC calls that server provided for users to call and also the message format for user request and server response.

We provide templates for this assignment, please download the supplementary file from E3 and do the assignment based on the templates (of course you can still implement the FTP system on your own as long as you fulfill the requirement and submission checklist).

We listed some FAQs about building on the E3 Forum. If you encounter any file or library missing issue while building, you can try to tackle the problem by looking up the discussion.

Defining `.proto` File

> *`.proto` file template is in /HW3/protos.*

Client Server communication APIs

> *The prototypes down below are all the APIs that the server provides for users to call. We need to define these APIs as the `service` in our .proto file. Besides, the function of each API is described in the following section.*

```
// A. Login
Login(User) returns (SessionID)
Logout(SessionID) returns (FtpStatus)

// B. List Directory
ListDirectory(SessionID) returns (Directory)

// C. Working Directory Operations
GetWorkingDirectory(SessionID) returns (Path)
ChangeWorkingDirectory(ChangeInfo) returns (FtpStatus)
```

A. Login

For the login process, you need to implement the following two RPC APIs

- Login(User) returns (SessionID)

  A client establishes a session with the FTP server by calling Login with the login credential.

  Return Value

  SessionID=0 : Unsuccessful login (invalid user or pwd)

  SessionID>0 : The login is successful. The return value is the session ID

  Parameters

  User.name: Name of the user (e.g., 'john')

  User.pwd: Password of the user (e.g., '1234567')

- Logout(SessionID) returns (FtpStatus)

  A client discontinues a session with the FTP server by calling Logout with the login credential.

  Return Value

  Status.code=0 : Successful log out

  Status.code=1 : Unsuccessful log out (invalid user or pwd)

  Parameters

  SessionID: The ID of the session to be discontinued.

B. List Directory

After a successful login, the client may ask for a list of the files under the current working directory on the server.

- ListDirectory(SessionID) returns (Directory)

  A client calls `ListDirectory` with its SessionID. The server will enumerate the files and subdirectories under the current working directory and return them in the list `Directory` back to the client.

  Return Value

  Directory: a list of the files and the subdirectories under the current working directory on the server.

  In case of errors, the server should returne an empty Directory.

  Parameters

  SessionID: The ID of the session established through a successful login.

C. Directory Traversal

- GetWorkingDirectory(SessionID) returns (Path)

  A client calls `GetWorkingDirectory` to get the current working directory on the server.

  Return Value

  Path: The full path pointing to the current working directory.

  In the case of errors, the returned Path will be an empty string.

  Parameters

  SessionID: The ID of the session established through a successful login.

- ChangeWorkingDirectory(ChangeInfo) returns (FtpStatus)

  A client calls `ChangeWorkingDirectory` to change the current working directory on the server.

  Return Value

  Status=0: The operatoin is successful.

  Status=1: The operatoin is unsuccessful.

  Parameters

  ChangeInfo.SessionID: The ID of the session established through a successful login.

  ChangeInfo.Path: The full path of the new working directory on the server.

Client Server communication messages

```
message User {
    required string name = 1;
    required string pwd = 2;
}
```

```
message FtpStatus {
  required int32 code = 1;
}
```

```
message SessionID {
  required bytes id  = 1;
}
```

```
message DEntry {
  required string name  = 1;

  enum PathType {
    FILE = 0;
    DIRECTORY = 1;
  }

  required PathType type = 2;
  required uint64 size = 3;
}
```

```
message Directory {
  repeated DEntry dentries  = 1;
}
```

```
message Path {
  required string path  = 1;
}
```

```
message ChangeInfo {
  required SessionID sessionid = 1;
  required Path path = 2;
}
```

Compiling Protocol Buffers

> *Reference:* *Protocol Buffer Basics: C++*

After defining our proto specification, we can use `protoc` compiler to generate the classes to read and write `tiny_ftp` messages. You can use the following command to generate `*.pb.cc` and `*pb.h` files at once. Also, we recommend you read through the `Makefile` to get a brief understanding of how to use `proroc`.

```
cd protos
make
```

- `*.pb.h`: the header file that declares your generated classes, the server and the client needs to include this file later.
- `*.pb.cc`: the main logic code that protoc generated based on your proto file and contains the implementation of your classes

https://docs.microsoft.com/en-us/aspnet/core/grpc/performance?view=aspnetcore-7.0

Building a dummy server

Now we start to build our FTP server without implementing the full features, it’ s the reason we call it a “dummy ”. You only need to make sure your server can respond to the client’ s RPC call correctly and output the log message.

The server template is included in the supplementary file, your main job in this section is finishing the class `FtpServerServiceImpl()` in `server/server.cc`. You can use the following command to compile the server. This will generate an executable file calls `server` in the working directory.

```
cd server
make

## If you successfully build your server, start the server with the
following command.
./server
```

Log message format for server

API	Server Output Message (stdout)
Login()	User: Bob login with pwd: BobPassword
Logout()	Logout session: 54321
ListDirectory()	List the directory of the session: 54321
GetWorkingDirectory()	Get the working directory of the session: 54321
ChangeWorkingDirectory()	The working directory of the session: 54321 has changed to home

3. GUI Client for Testing

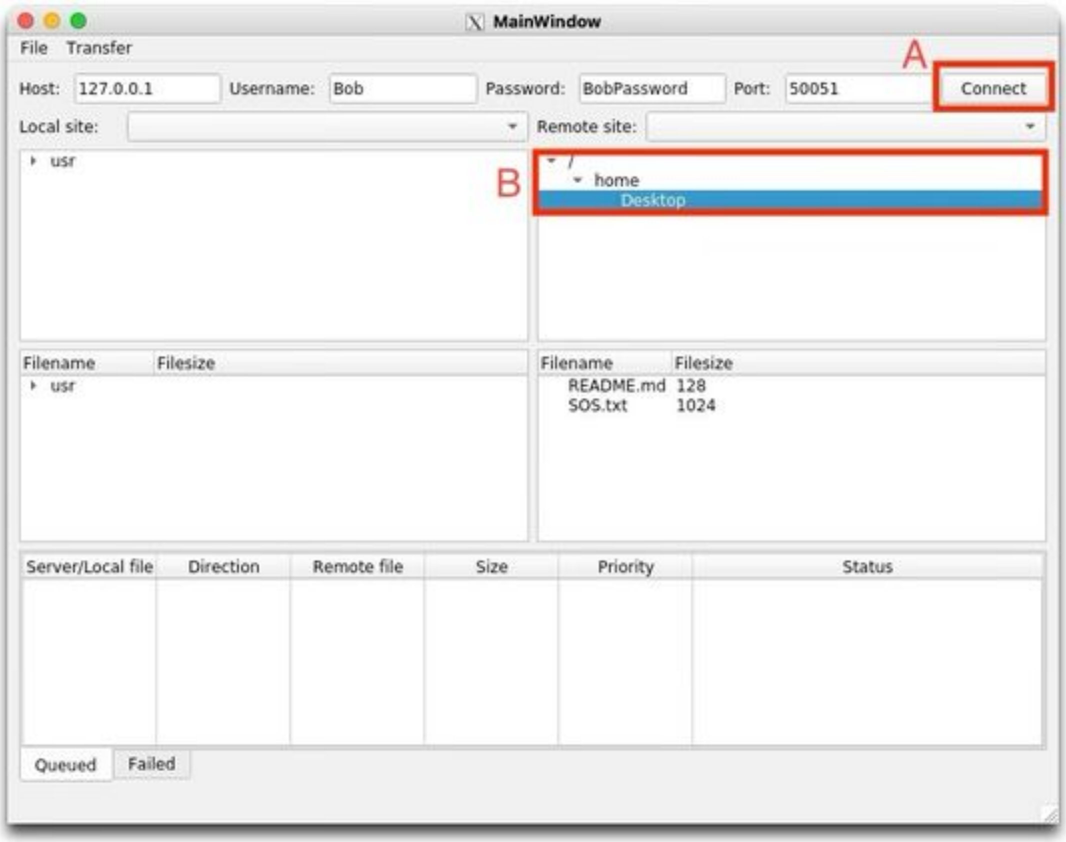
⚠ You should at least finish and compile your `.proto` file before building the client because it depends on the proto files you defined!

After building your server, we provide a sample GUI client for you to test your server, run the following command to generate the client’ s binary.

```
cd ftp_gui_im
qmake
make

## Run the client (make sure you start the server first)
./ftp_gui_im
```

This GUI client will do following things:



Host: 127.0.0.1 Username: Bob Password: BobPassword Port: 50051

- usr

- /
  - home
    - Desktop

Filename	Filesize
usr	

Filename	Filesize
README.md	128
SOS.txt	1024

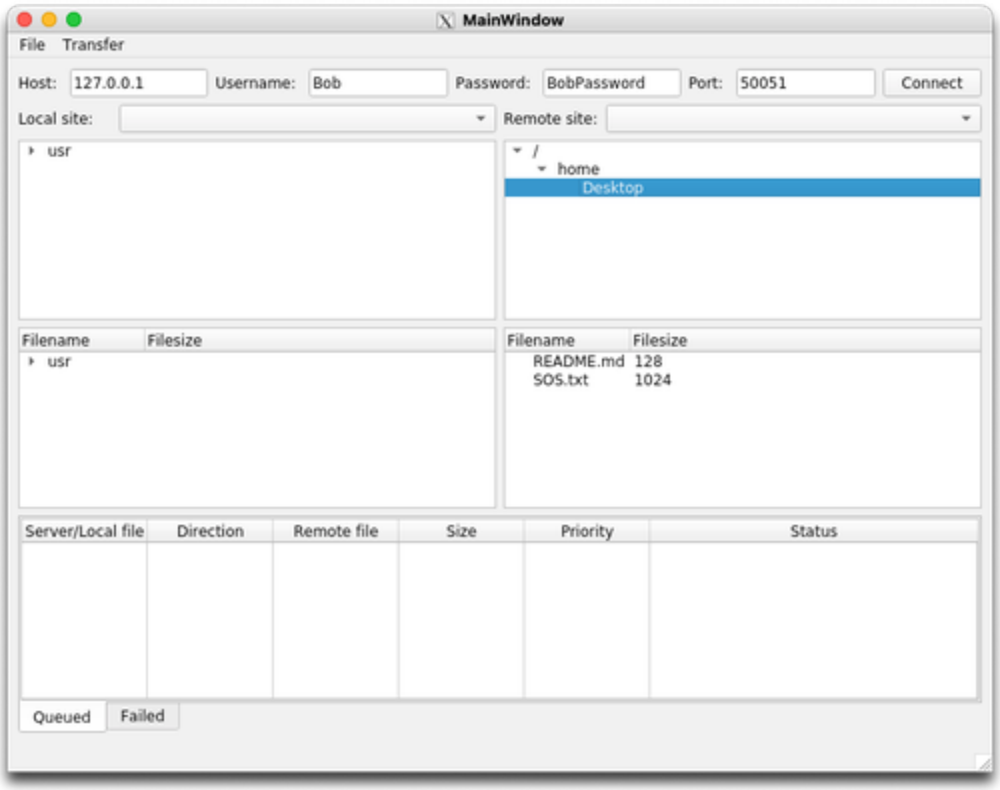
Server/Local file	Direction	Remote file	Size	Priority	Status

- Section A: When you click the "Connect" button, the GUI will fetch the connection info you typed and create a channel with your server. Once the channel is created, the GUI will fire several test RPC calls to your server (listed down below). Section B will display the response it got from the server only if you implement the server correctly.
  - 1. `Login()`
  - 2. `GetWorkingDirectory()`
  - 3. `ListDirectory()`
- Section B: This section will display the file structure it got from the server just like Filezilla. When you click on each item (directory) in section B, the GUI will make the following RPC calls to your server in order to list the files under the directory you clicked.
  - 1. `ChangeWorkingDirectory()`
  - 2. `ListDirectory()`

---

4. Submission Checklist

1. Complete the `.proto` file in `HW3/protos/tiny_ftp.proto`
2. Implement the 5 RPC call APIs in `HW3/server/server.cc`
3. Run the server and the GUI client. On the client, perform the following steps in sequence:
   a. Fill in the connection info
   b. Click on "Connect" button
   c. Click on the root ("/") directory
   d. Click on "home" directory
   e. Click on "Desktop" directory
   f. **Take a screenshot of the GUI client!**
   g. Click "Logout" in the File drop-down menu.
   h. **Take a screenshot of the server's stdout!**
4. After finishing step 3, name the screenshots you captured as follows:
   a. GUI: gui_client.jpg
   b. server: server_stdout.jpg



Host: 127.0.0.1 Username: Bob Password: BobPassword Port: 50051

Local site:

Remote site:

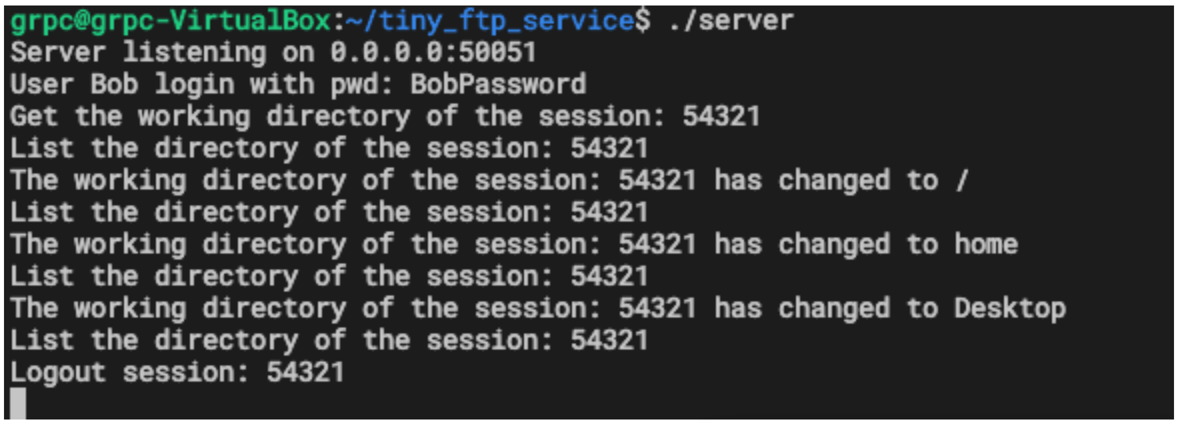
```
▸ usr
```

```
▾ /
  ▾ home
      Desktop
```

Filename	Filesize
usr	

Filename	Filesize
README.md	128
SOS.txt	1024

Server/Local file	Direction	Remote file	Size	Priority	Status



```
grpc@grpc-VirtualBox:~/tiny_ftp_service$ ./server
Server listening on 0.0.0.0:50051
User Bob login with pwd: BobPassword
Get the working directory of the session: 54321
List the directory of the session: 54321
The working directory of the session: 54321 has changed to /
List the directory of the session: 54321
The working directory of the session: 54321 has changed to home
List the directory of the session: 54321
The working directory of the session: 54321 has changed to Desktop
List the directory of the session: 54321
Logout session: 54321
```

5. Zip all the files and screenshots in the following directory structure. Name the zip file **HW3_<student id>.zip.** (Please run `make clean` to clean up the object files before you zip the folder)

```
HW3
├── ftp_gui_im
├── protos
├── server
├── gui_client.jpg
└── server_stdout.jpg
```

If you have any questions, please post them on the E3 forum for the course project. If needed, you can reach the TA for the project at Ping-Yang <pychang.cs10@nycu.edu.tw>.