CMSC 417 Computer Networks **Fall 2012**

Programming Assignment 2

Assigned: September 20th *Due: September 30th, 11:59:59 PM.*

1 Description

In distributed file systems, large files could be stored in the form of multiple fixed-size chunks that are distributed and replicated among multiple servers. This approach aims at enabling the clients to retrieve different parts of a file in parallel or in an out-of-order manner. It also enables the clients to download specific parts of the file according to their need without having to download all the file content.

In this assignment, your goal is to develop a chunk server that handles chunk requests from clients. The server should be able to handle multiple clients concurrently. You will also develop a client to be able to test your server. The server and the client will communicate using the TCP protocol. Figure 1 provides an overview of the communication between the server and the clients in this assignment.

You will use Google protocol buffers to serialize and deserialize the messages and data exchanged between the server and the clients. Google Protocol buffers are an extensible, language-neutral and platform-neutral way for serializing structured data[1]. This method employs an interface description language through which the structure of data can be defined, then a compiler is invoked to produce the necessary source files to be used by programmers.

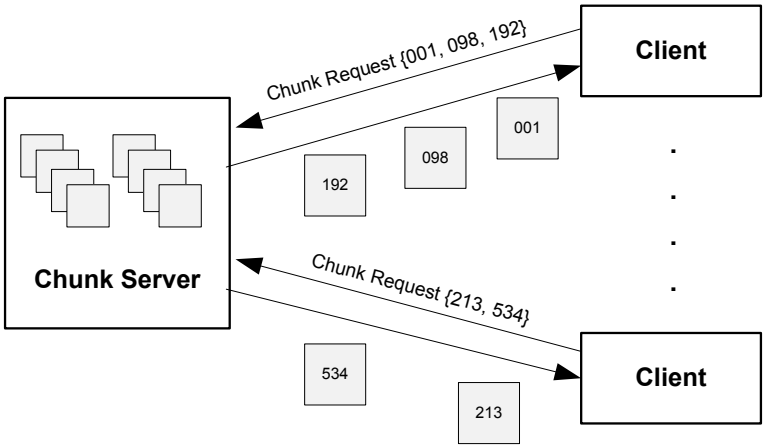

Figure 1: Interaction between the chunk server and the clients.

2 Protocol

The protocol between the server and the client should be as follows:

[1]https://developers.google.com/protocol-buffers/

1. A client sends a chunk request to the server. This request message includes: a client id string (e.g. a username), the message type (i.e. Request) and the list of chunks requested.
2. The server then sends a response message to the client that contains its id string, the message type (i.e. Response) and the list of chunks found at the server side.
3. The client sends an acknowledgment message to the server.
4. The server starts sending chunks consecutively in the order of the of the list of found chunks that was sent earlier within the message at step 2.

3 Specifications

Server Side

1. The server program receives two arguments: the port number and a path to the directory that includes the chunks.
2. The name of each chunk has three numeral characters. It will be in the format: 000, 001, 002, .. etc. We will provide the chunks folder to you.
3. You can pick any arbitrary id string for the server, e.g. "Server".
4. The server should use threading to handle the multiple clients (see pthread). You can check the example for using pthread in the TCP/IP Sockets in C book in page 123.
5. When the server receives a list of requested chunks from a client, it will search in the specified directory, and determines the chunks it has. Then, it will respond accordingly informing the client with the chunks it found.

Client Side

1. The client program receives four arguments: the server IP address, the server port, a username to identify the client with, and a path to the directory in which chunks downloaded from the server will be stored.
2. The client should generate 20 random chunk names to be requested in the range 000 to 999.
3. The client program should create a folder under the provided path argument, in which it will store the received chunks. The name of this folder can be the same as the username.
4. Note: in order to test concurrency, you will need to run multiple clients concurrently. You will find a guiding script included in the package we give you.

Communication Messages

As mentioned earlier, protocol buffers will be used for serializing the messages exchanged between the server and the client. There are two types of messages here: messages that are used for communication (e.g. request/response/acknowledgment), and messages used for carrying chunks data and content. Both types are defined in the protocol.proto buffer as follows:

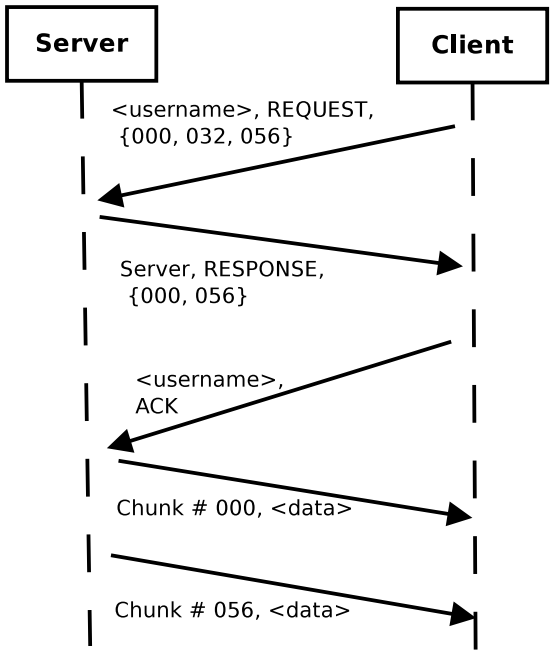

Figure 2: An example communication scenario.

```
message Message{
    required string id = 1;
    enum MessageType{
        REQUEST = 1;
        RESPONSE = 2;
        ACK = 3;
    }
    required MessageType messageType = 2;
    repeated string chunksList = 3;
}

message Chunk{
    required string chunkId = 1;
    required bytes chunkContent = 2;
}
```

When this file is compiled using the protobuf-c compiler, it produces header and source files that you can use in your program. You will find these files included in the provided package. You **must** check these files to know what the expected data types and names are. For how to encode and decode the messages, examples are available on the internet[2].

Notes:

- Figure 2 provides an example scenario for the communication between a server and a client.

[2]http://code.google.com/p/protobuf-c/wiki/Examples

- In the Message object, the client uses the chunksList to tell the server about the chunks it requests in the initial communication, whereas the server uses it to tell the client about the chunks it found in the first response.
- Since all the chunks are the same size, the size of each serialized Chunk object will be constant and it will be equal to 16393 bytes, if the chunk object is filled appropriately. You will need to be aware of that number.

Additional Remarks

- The package included with this assignment contains all what you need, including the chunks folder and the necessary include and library files.
- The chunkprotocol.proto file has been already compiled, and it produced the files: chunkprotocol.pb-c.h and chunkprotocol.pb-c.c, which you will use directly in your program.
- Write your code in chunkserver.c and chunkclient.c. Don't edit any other existing files or folders.
- Use the provided makefile to compile your code.
- Use the provided script test.sh to test the concurrency aspect in your implementation.

4 Policies

- No late submissions will be accepted.
- Your code must be -Wall clean on gcc. Do not ask the TA for help on (or post to the forum) code that is not -Wall clean.
- The TA will post information on Piazza about how to submit your assignment.