CN Project Phase #2 Report

B08902143 謝文傑

National Taiwan University — January 16, 2022

1 Team Members

- B08902143 謝文傑

2 Implementation Details

2.1 Architecture

The server/client architecture adopted is shown in below diagram. The server binds to a port and accept tcp connection from clients. Each connection is handled by unique go routine for coding simplity and support for concurrency. Once the client connect to the server, it begins the authentication phase, in which it requires to whether sign in or sign up by providing necessary credentials (username, password). Once authenticated, the client binds a local port which accepts tcp connection from browser. Each connection to client is also handled by unique go routine. A self-implemented http parser is used to parse the http request from the tcp connection and write generated http response back to tcp connection.

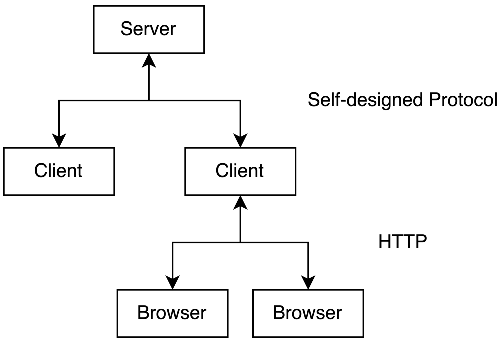

Figure 1: The Server/Client Architecture

2.2 HTTP Request / Response

A self-implemented HTTP Request Parser is used to parse http request from tcp connection, relevant source code can be found in `src/client/http/http.go`. To parse http request from tcp connection, the parser reads the top/header lines, then reads request body if `Content-Length` presented in headers. The type definition of HttpRequest and HttpResponse is shown below.

src/client/http/http.go

```
...
type HttpRequest struct {
  Method  string
  Target  *url.URL
  Headers map[string]string
  Data    []byte
}
type HttpResponse struct {
  StatusCode int
  Headers    map[string]string
  Data       []byte
}
...
```

To write http response back to the tcp connnection, the status code / header lines / data payload is sequentially send back to the tcp connection. The relevant function is shown below.

src/client/http/http.go

```
...
func SendResponse(res HttpResponse, conn net.Conn) {
  fmt.Fprintln(conn, "HTTP/1.1", res.StatusCode)
  for key, value := range res.Headers {
    fmt.Fprintf(conn, "%s: %s\n", key, value)
  }
  fmt.Fprintln(conn, "Content-Length:", len(res.Data))
  fmt.Fprintln(conn)
  conn.Write(res.Data)
}
...
```

2.3 Chat History

The user and message are stored as two data models defined as follows.

```
src/server/models/user.go, message.go

type User struct {
  ID           int          ‘json:"id"‘
  Username     string       ‘json:"username"‘
  PasswordHash string       ‘json:"password_hash"‘
  Friends      map[int]bool ‘json:"friends"‘
}
type Message struct {
  ID        int       ‘json:"id"‘
  From      string    ‘json:"from"‘
  To        string    ‘json:"to"‘
  Content   []byte    ‘json:"content"‘
  Type      string    ‘json:"type"‘
  Filename  string    ‘json:"filename"‘
  Timestamp time.Time ‘json:"timestamp"‘
}
```

Hare, a third party embedded key-value database golang package is used to persistently storing/querying the data about users and messages. When serving messages between two users, all the messages send from A to B or from B to A is collected and sorted by the timestamp and sent to the client.

2.4 Routes & Serving Content

When parsing the http reqeust, the target is being parsed as url.URL object, which allows us to check the path and queries options. When checking which route is requested, regex is used to match the path. Once the path is match for certain regex, the correspond hander will generate the response and serve the response.

To serve HTML content, the standard package `html/template` is used to generate html pages with templates in `views/` and fill in data. To serve file content, the `"Content-Disposition"` is set to `"attachment; filename={filename}"` so that the browser saves the file. When serving image / file content, the client reads the file from "client_dir" and then generate the response with file content as body.

2.5 Bonus

Lazy File Transfering

When trasnfering non-text messages (images/files), the server asks if client has already got the file or not to decide the need for high cost file transition. The client checks if the relevant file exists and report if it needs the file. Implement such lazy file transfering function can speed up the program by alot and saves lots of network transmission.

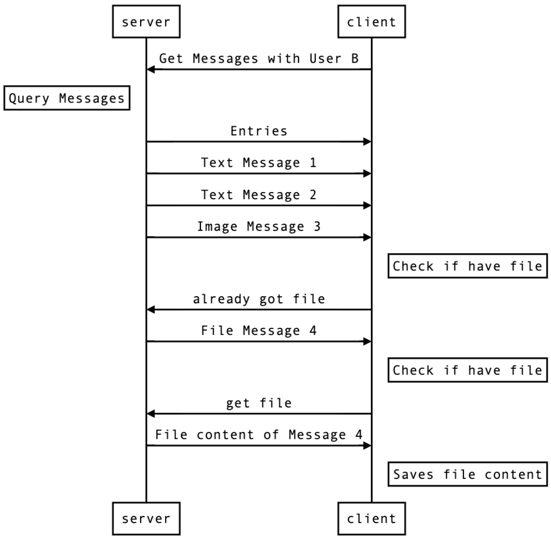

Figure 2: Lasy File Transfering

Safe Password Storage

To improve security, the password isn't stored as plaintext. Bcrypt is used to generated salted digest of password and stored in the database.

src/server/server.go

```
passwordHash, _ := bcrypt.GenerateFromPassword(
  []byte(password), 10,
)
user = models.User{
  Username:     username,
  PasswordHash: string(passwordHash),
  Friends:      map[int]bool{},
}
db.Insert("users", &user)
```

3 Demo Link

https://youtu.be/Zv1iGi3cwlI

4 Work Division

- B08902143: All the works