Programming Assignment #4

Introduction to Computer Networks

The Assignment

Having experience programming the file upload client, you are ready to program the file upload server. Your `PA4.go` should allow your `PA3.go` to upload a file. More specifically, the server:

(1) listens at <your port#> until there's an upload request
(2) reads from the socket first the file size (just the number in a single line)
(3) reads from the socket one line at a time
(4) prepend the line count to each line and store the new line into a new file: `whatever.txt`
(5) repeats (3) and (4) until all lines in the file is processed
(6) sends a message back that tells the client the original file and the new file size
(7) closes the connections and terminates the program

Find out <your port#> from: http://homepage.ntu.edu.tw/~pollyhuang/teach/intro-cn-pa/port_assignment.pdf. You will see each team is assigned a unique port number. Use strictly only your port number, please. This is important because two server programs cannot be running on the same port.

What you learn in PA3 is sufficient to carry you through PA4. However, going through the following examples might make the programming of `PA4.go` easier. The two examples are to introduce the `Reader` object and associated APIs in package `bufio`, as an alternative to the `Scanner` object/APIs.

Now follow through the examples below and practice the socket APIs of Go.

1. Another Server-Client Pair

This example implements a server that receives a string from the socket, prints it on the screen, and sends back the size of the string. Start a file `server-102.go` and type up the following code.

```
package main

import "fmt"
import "bufio"
import "net"


func check(e error) {
   if e != nil {
      panic(e)
   }
}

func main() {
   fmt.Println("Launching server...")
   ln, _ := net.Listen("tcp", ":<your port#>")
   conn, _ := ln.Accept()
   defer ln.Close()
   defer conn.Close()

   reader := bufio.NewReader(conn)
   message, errr := reader.ReadString('\n')
   check(errr)
   fmt.Printf("%s", message)

   writer := bufio.NewWriter(conn)
   newline := fmt.Sprintf("%d bytes received\n", len(message))
   _, errw := writer.WriteString(newline)
   check(errw)
   writer.Flush()
}
```

Start a file `client-102.go` and type up the following code.

```
package main

import "fmt"
import "bufio"
import "net"


func check(e error) {
   if e != nil {
      panic(e)
   }
}

func main() {
   conn, errc := net.Dial("tcp", "127.0.0.1:<your port#>")
   check(errc)
   defer conn.Close()

   writer := bufio.NewWriter(conn)
   len, errw := writer.WriteString("Hello World!\n")
   check(errw)
   fmt.Printf("Send a string of %d bytes\n", len)
   writer.Flush()

   reader := bufio.NewReader(conn)
   message, errr := reader.ReadString('\n')
   check(errr)
   fmt.Printf("Server replies: %s", message)
}
```

Replace `<your port#>` with the port number assigned to your team. Start the server code first.

```
$ go run server-102.go
Launching server...
```

Then, the client code.

```
$ go run client-102.go
Send a string of 13 bytes
Server replies: 13 bytes received
$
```

The terminal running the server code should continue with:

```
Hello World!
$
```

Code walk-through:

- `bufio.NewReader(conn)` works just like `bufio.NewScanner()`. In that, `NewReader()` creates a `Reader` object that wraps around `conn` to allow the socket connection access to APIs available to the `Reader` type.
- `reader.ReadString(\n)` reads a string delimited by the character in the `()`. In effect, this call reads a line, including the `\n`. Giving it, i.e., the delimiter, an empty space, the call will read a word. Text extraction/processing is even easier with `Reader` than with `Scanner`.

2. PA4.go

To test your `PA4.go`, do modify your `PA3.go` to dial to the IP address of the machine your `PA4.go` is running on and the port number you are assigned.

To help you verify your implementation, polly has made the compiled byte code of her `PA4.go` available here: http://homepage.ntu.edu.tw/~pollyhuang/teach/intro-cn-pa/PA4/PA4. Login to the workstation and download the byte code to your account:

```
    $ curl homepage.ntu.edu.tw/~pollyhuang/teach/intro-cn-pa/PA4/PA4
> pollys-PA4
```

The `curl` command downloads the byte code and saves it as a file, filename `pollys-PA4`. You'll need to change the permission to allow user to execute the file by the following before trying it out:

```
$ chmod u+x pollys-PA4
$ ./pollys-PA4
```

To not cause any port conflict with you guys' servers, polly's PA4 server is running on port# 11999. To see how polly's `PA4` behaves, you'll need to modify your `PA3.go` to dial to the workstation and port# 11999. Cross compare execution result of your `PA4.go` to the outcome of executing `PA4`. If they work the same, you will be done and safe.

Chance is low but there is a chance that two or more teams try to start polly's `PA4` byte code on the workstation at the same time. There'll still be port conflict. Two ways to resolve the issue. First, give it a bit time and then run polly's PA4 again. Hope the other team has completed their test when you start polly's `PA4` again. Second, in case there's a process running non-stop on 11999. It is likely some other team running polly's PA4 byte code and forget to turn it off. Just pair that process with your PA3.go and observe how it behaves. It could well be working and you do not need to start polly's PA4 yourself. The issue with this case is that you don't see the output from polly's PA4 server, so you can confirm whether your `PA4.go` is generating output as expected.

3. Go Documentation

For details and other APIs in the `bufio` package, visit this page:
https://golang.org/pkg/bufio/

You will see 3 major object types, `Reader`, `Scanner`, and `Writer`. `Reader`'s method set is significantly richer than the `Scanner`'s. One can `ReadByte()` to read a single byte, `ReadBytes()` to read a byte segment, `ReadLine()` to read a line, `ReadString()` to read a string, or simply `Read()` such as the native socket I/O (only) does. You might see a bit why many developers appreciate Go – a rich and coder-friendly collection of packages. Imagine how much faster we will be able to code, and build on that, how much faster the next generation of coders code. Hope you get the idea and the power of open source.

4. Submit your PA4

`ssh` to the `140.112.42.221` workstation. At the team account's home directory, create a directory `PA4`. Upload your `PA4.go` to directory `PA4`. Test your `PA4.go` again on the workstation just to make sure it's working as expected.