I demonstrate here a simple server and client side of a Project in language **“go”**

The problem is:

There is a simple implementation of file server based on http.FileServer handle ( [https://pkg.go.dev/net/http#example-FileServer](https://pkg.go.dev/net/http" \l "example-FileServer" \t "_blank) ).

The server instance is running on top of simple file folder which doesn’t have nested subfolders.

Please implement client which downloads files using this server.

You should download a file containing char 'A' on earlier position than other files.

In case several files have the 'A' char on the same the earliest position you should download all of them.

Each TCP connection is limited by speed. The total bandwidth is unlimited.

You can use any disk space for temporary files.

The goal is to minimize execution time and data size to be transferred.

======================================================

Example

If the folder contains the following files on server:

'file1' with contents: "---A---"

'file2' with contents: "--A------"

'file3' with contents: "------------"

'file4' with contents: "==A=========="

then 'file2' and 'file4' should be downloaded

For the solution:

We should consider the following conditions:

* The directory is empty
* None of the filenames include the letter “A”
* “a” letter is also considered
* Some filenames length is few or very long (filename with one letter or 50 letters)

After you run the server side code:

You can access the related pages as:

Localhost:8090

Localhost:8090/hello

Localhost:8090/headers

Localhost:8090/docs

Localhost:8090/root

/hello and /headers call the function hello and headers respectively.

/docs and /root list the files under /users/share/doc and / respectively.

**Server Side**

package main  
  
import (  
 "errors"  
 "fmt"  
 "io"  
 "net/http"  
)  
  
func main() {  
 fmt.Println("welcome")  
 http.HandleFunc("/hello", hello)  
 http.HandleFunc("/headers", headers)  
 http.Handle("/docs/", http.StripPrefix("/docs", http.FileServer(http.Dir("/users/share/doc"))))  
 http.Handle("/root/", http.StripPrefix("/root", http.FileServer(http.Dir("/"))))  
  
 var err error  
 err = http.ListenAndServe(":8090", nil)  
 if errors.Is(err, http.ErrServerClosed) {  
 fmt.Printf("server one closed\n")  
 } else if err != nil {  
 fmt.Printf("error listening for server one : %s\n", err)  
 } else {  
 fmt.Printf("no error : %s catched\n", err)  
 }  
}  
  
func hello(w http.ResponseWriter, req \*http.Request) {  
 io.WriteString(w, "Hello World")  
}  
  
func headers(w http.ResponseWriter, req \*http.Request) {  
 for name, headers := range req.Header {  
 for \_, h := range headers {  
 fmt.Fprintf(w, "%v: %v\n", name, h)  
 }  
 }  
}

On the client side: Reads the content through <http://localhost:8090/docs/>

Append all the filenames to the sfile[] list.

The first and last lines which consist of <pre> and </pre> won’t be appended.

With the use of index() method the leftmost indexes of letter “A” in each filenames are appended to winner[]. -1 is appended if there is no letter “A” in a filename.

The filenames who has the lowest indexes (and also >-1) are appended to dlfile[]. Finally filenames in dlfile[] are downloaded to the default directory.

**Client Side**

package main  
  
import (  
 "bufio"  
 "fmt"  
 "io"  
 "net/http"  
 "os"  
 "strings"  
 "sync"  
)  
  
func main() {  
 urlMain := "http://localhost:8090/docs/"  
 resp, err := http.Get(urlMain)  
 if err != nil {  
 panic(err)  
 }  
 defer resp.Body.Close()  
 var sfile []string // List of all files shown on the site  
 var dlfile []string // List of files having the letter "A" at left most  
 scanner := bufio.NewScanner(resp.Body)  
 for i := 0; scanner.Scan(); i++ {  
 rf := scanner.Text()  
 /\*  
 The scanner.Text() returns as following  
  
 <pre>  
 <a href="Estimation.pdf">Estimation.pdf</a>  
 <a href="ForcAsting.pdf">ForcAsting.pdf</a>  
 . . .  
 <a href="Households%20pro-environment.pdf">Households pro-environment.pdf</a>  
 <a href="treeApplications.pdf">treeApplications.pdf</a>  
 </pre>  
  
 To extract just the filenames the following codes are processed  
 \*/  
  
 if len(rf) > 4 && rf[len(rf)-4:] == "pre>" {  
 fmt.Printf("head/tail\n")  
 } else {  
 s := scanner.Text()  
 readfile := s[strings.Index(s, ">")+1 : strings.Index(s, "</a>")]  
 sfile = append(sfile, readfile)  
 fmt.Println("..read filename: ", readfile)  
 }  
 }  
 // Just the filenames are shown  
 for i := 0; i < len(sfile); i++ {  
 fmt.Println(sfile[i])  
 }  
 var winner []int // Array, holds the left most index of letter "A" of all filenames. If not -1 is set  
 var n = len(sfile) // Holds the number of filenames  
 var leftmost = 1000 // The global minimum index of the letter "A" found in filenames.  
 var curLeft = -1 // Gets the index of left most letter "A".  
 if n > 0 { // Checks whether the server has at least one file  
  
 for j := 0; j < n; j++ {  
 curLeft = strings.Index(strings.ToUpper(sfile[j]), "A")  
 if curLeft > -1 && curLeft < leftmost {  
 leftmost = curLeft  
 }  
 winner = append(winner, curLeft)  
 }  
 if leftmost < 1000 { // If there is at least one letter "A"  
 for j := 0; j < n; j++ {  
 if winner[j] == leftmost {  
 dlfile = append(dlfile, sfile[j])  
 fmt.Println("Downloadable file: ", sfile[j])  
 }  
 }  
  
 //\*\*\*\* download files begin  
 var w sync.WaitGroup  
 for i := 0; i < len(dlfile); i++ {  
 w.Add(1)  
 filename := dlfile[i]  
 url := urlMain + filename  
 if err := DownloadFile(filename, url, w); err != nil {  
 panic(err)  
 }  
 fmt.Println(url)  
 }  
 w.Wait()  
 //\*\*\*\* download files end  
 } else {  
 fmt.Println("\nNone of the filenames involve the letter <A>")  
 }  
 } else {  
 fmt.Println("\n No file to process!")  
 }  
}  
  
func DownloadFile(filepath string, url string, wg sync.WaitGroup) error {  
  
 resp, err := http.Get(url)  
 defer wg.Done()  
 if err != nil {  
 return err  
 }  
 defer resp.Body.Close()  
  
 out, err := os.Create(filepath)  
 if err != nil {  
 return err  
 }  
 defer out.Close()  
  
 \_, err = io.Copy(out, resp.Body)  
 return err  
}

