Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 6

по курсу «Компьютерные системы и сети»

«Разработка FTP- клиента»

Студент группы ИУ9-32Б Федуков А. А.

Преподаватель Посевин Д. П.

12 ноября 2024 г.

Цель работы

Целью данной работы является разработка FTP-клиента на языке GO.

Задание

Задача 1: Реализовать FTP-клиент на сервере.
Задача 2: Протестировать соединение GO FTP-клиента с установленным на сервере students.yss.su FTP-сервером со следующими параметрами доступа:

- ftp-host: students.yss.su
- login: ftpiu8
- passwd: 3Ru7yOTA

Задача 3: Реализовать следующую функциональность:

- загрузку файла;
- скачивание файла;
- создание директории;
- удаление файла;
- получение содержимого директории;
- переход в директорию;
- удаление пустой директории;
- удаление директории рекурсивно.

Реализация

Я создал программу, которая подключается по FTP

Код

Листинг 1: Файл main.go

```go
package main

import (
  "bufio"
  "fmt"
  "log"
  "os"
  "path/filepath"
  "strings"

  "github.com/secsy/goftp"
)

var globalRemotePath = "/"

func main() {
  // FTP connection configuration
  config := goftp.Config{
    User:     "ftpiu8",   // FTP username
    Password: "3Ru7yOTA", // FTP password
  }
  client, err := goftp.DialConfig(config, "students.yss.su:21")
  if err != nil {
    log.Fatalf("Failed to connect to FTP server: %v", err)
  }
  defer client.Close()

  // Main command loop
  scanner := bufio.NewScanner(os.Stdin)
  fmt.Printf("\n[%s] Enter command:\n", globalRemotePath)
  fmt.Println("upload <local_path> <remote_path>")
  fmt.Println("download <remote_path> <local_path>")
  fmt.Println("mkdir <remote_path>")
  fmt.Println("rm <remote_path>")
  fmt.Println("ls <remote_path>")
  fmt.Println("cd <remote_path>")
  fmt.Println("rmdir <remote_path>")
  fmt.Println("rmdirall <remote_path>")
  fmt.Println("exit")

  for {
    fmt.Printf("[%s] > ", globalRemotePath)

```

```
44      // Get user input
45      if !scanner.Scan() {
46        break
47      }
48
49      input := strings.Fields(scanner.Text())
50      if len(input) == 0 {
51        continue
52      }
53      command := input[0]
54
55      // Handle each command
56      switch command {
57      case "help":
58        fmt.Println("upload <local_path> <remote_path>")
59        fmt.Println("download <remote_path> <local_path>")
60        fmt.Println("mkdir <remote_path>")
61        fmt.Println("rm <remote_path>")
62        fmt.Println("ls <remote_path>")
63        fmt.Println("cd <remote_path>")
64        fmt.Println("rmdir <remote_path>")
65        fmt.Println("rmdirall <remote_path>")
66        fmt.Println("exit")
67      case "upload":
68        if len(input) != 3 && len(input) != 2 {
69          fmt.Println("Usage: upload <local_path> <remote_path>")
70          continue
71        }
72        localPath := input[1]
73        remotePath := ""
74        if len(input) == 2 {
75          remotePath = globalRemotePath + filepath.Base(input[1])
76        } else {
77          remotePath = resolvePath(input[2])
78        }
79
80        err := uploadFile(client, localPath, remotePath)
81        if err != nil {
82          fmt.Printf("Failed to upload file: %v\n", err)
83        } else {
84          fmt.Println("File uploaded successfully.")
85        }
86
87      case "download":
88        if len(input) != 3 && len(input) != 2 {
89          fmt.Println("Usage: download <remote_path> <local_path>")
```

```
90          continue
91        }
92        remotePath := input[1]
93        localPath := ""
94        if len(input) == 2 {
95          localPath = filepath.Base(input[1])
96        } else {
97          localPath = resolvePath(input[2])
98        }
99
100       err := downloadFile(client, remotePath, localPath)
101       if err != nil {
102         fmt.Printf("Failed to download file: %v\n", err)
103       } else {
104         fmt.Println("File downloaded successfully.")
105       }
106
107     case "mkdir":
108       if len(input) != 2 {
109         fmt.Println("Usage: mkdir <remote_path>")
110         continue
111       }
112       remotePath := resolvePath(input[1])
113       _, err := client.Mkdir(remotePath)
114       if err != nil {
115         fmt.Printf("Failed to create directory: %v\n", err)
116       } else {
117         fmt.Println("Directory created successfully.")
118       }
119
120     case "rm":
121       if len(input) != 2 && len(input) != 1 {
122         fmt.Println("Usage: rm <remote_path>")
123         continue
124       }
125
126       remotePath := ""
127       if len(input) == 1 {
128         remotePath = resolvePath("")
129       } else {
130         remotePath = resolvePath(input[1])
131       }
132
133       err := client.Delete(remotePath)
134       if err != nil {
135         fmt.Printf("Failed to delete file: %v\n", err)
```

```
136         } else {
137           fmt.Println("File deleted successfully.")
138         }
139
140       case "ls":
141         fmt.Println(input, len(input))
142         if len(input) != 2 && len(input) != 1 {
143           fmt.Println("Usage: ls <remote_path>")
144           continue
145         }
146
147         remotePath := ""
148         if len(input) == 1 {
149           remotePath = resolvePath("")
150         } else {
151           remotePath = resolvePath(input[1])
152         }
153
154         entries, err := client.ReadDir(remotePath)
155         if err != nil {
156           fmt.Printf("Failed to list directory: %v\n", err)
157         } else {
158           fmt.Println("Directory contents:")
159           for _, entry := range entries {
160             fmt.Printf(" - %s\n", entry.Name())
161           }
162         }
163
164       case "cd":
165         if len(input) != 2 {
166           fmt.Println("Usage: cd <remote_path>")
167           continue
168         }
169         newPath := input[1]
170         if err := changeDirectory(client, newPath); err != nil {
171           fmt.Printf("Failed to change directory: %v\n", err)
172         } else {
173           fmt.Printf("Changed directory to: %s\n", globalRemotePath)
174         }
175
176       case "rmdir":
177         if len(input) != 2 && len(input) != 1 {
178           fmt.Println("Usage: rmdir <remote_path>")
179           continue
180         }
181
```

```
182         remotePath := ""
183         if len(input) == 1 {
184           remotePath = resolvePath("")
185         } else {
186           remotePath = resolvePath(input[1])
187         }
188 
189         err := client.Rmdir(remotePath)
190         if err != nil {
191           fmt.Printf("Failed to remove directory: %v\n", err)
192         } else {
193           fmt.Println("Directory removed successfully.")
194         }
195 
196       case "rmdirall":
197         if len(input) != 2 && len(input) != 1 {
198           fmt.Println("Usage: rmdirall <remote_path>")
199           continue
200         }
201 
202         remotePath := ""
203         if len(input) == 1 {
204           remotePath = resolvePath("")
205         } else {
206           remotePath = resolvePath(input[1])
207         }
208 
209         err := removeDirectoryRecursive(client, remotePath)
210         if err != nil {
211           fmt.Printf("Failed to recursively delete directory: %v\n", err)
212         } else {
213           fmt.Println("Directory recursively deleted successfully.")
214         }
215 
216       case "exit":
217         fmt.Println("Exiting...")
218         return
219 
220       default:
221         fmt.Println("Unknown command.")
222       }
223     }
224 }
225 
226 // Change directory function that updates globalRemotePath
227 func changeDirectory(client *goftp.Client, newPath string) error {
```

```
228     // Resolve the path relative to current directory
229     newFullPath := resolvePath(newPath)
230 
231     // Check if the directory exists on the server by attempting to read
          it
232     _, err := client.ReadDir(newFullPath)
233     if err != nil {
234       return fmt.Errorf("directory does not exist or cannot be accessed")
235     }
236 
237     // Update the global path
238     globalRemotePath = newFullPath
239 
240     if globalRemotePath[len(globalRemotePath)-1] != '/' {
241       globalRemotePath += "/"
242     }
243 
244     return nil
245 }
246 
247 // Resolve path based on the current globalRemotePath
248 func resolvePath(path string) string {
249 
250     if path == "" {
251       return globalRemotePath
252     }
253 
254     if filepath.IsAbs(path) {
255       return path
256     }
257     return filepath.Join(globalRemotePath, path)
258 }
259 
260 // Other helper functions for FTP operations
261 func uploadFile(client *goftp.Client, localPath, remotePath string)
        error {
262     file, err := os.Open(localPath)
263     if err != nil {
264       return err
265     }
266     defer file.Close()
267     return client.Store(remotePath, file)
268 }
269 
270 func downloadFile(client *goftp.Client, remotePath, localPath string)
        error {
```

```
271     file, err := os.Create(localPath)
272     if err != nil {
273         return err
274     }
275     defer file.Close()
276     return client.Retrieve(remotePath, file)
277 }
278 
279 func removeDirectoryRecursive(client *goftp.Client, remotePath string)
        error {
280     entries, err := client.ReadDir(remotePath)
281     if err != nil {
282         return err
283     }
284     for _, entry := range entries {
285         fullPath := filepath.Join(remotePath, entry.Name())
286         if entry.IsDir() {
287             err := removeDirectoryRecursive(client, fullPath)
288             if err != nil {
289                 return err
290             }
291 
292         } else {
293             err := client.Delete(fullPath)
294             if err != nil {
295                 return err
296             }
297         }
298     }
299     return client.Rmdir(remotePath)
300 }
```

Вывод программы

Программа запрашивает команды у пользователя и выполняет их удаленно по протоколу FTP.

Вывод

В этот лабораторной я научился использовать протокол FTP на языке GO. Я реализовал основные функции удаленного файлового менеджера, в том числе загрузка и скачивание.