**Отчет по лабораторной работе № 2**

Тарасов А. Т.

Группа ИУ9-31Б

**Цель:** Реализовать совместную работу Go ssh-сервера и Go ssh-клиента

Заявленной цели соответствуют следующие **задачи:**

**Реализовать ssh-клиент на языке GO, поддерживающий следующие функции:**

* создание директории go ssh-клиентом на ssh-сервере;
* удаление файла go ssh-клиентом с ssh-сервера;
* получение содержимого директории на ftp-сервере с помощью go ftp-клиента.

**Реализовать ssh-сервер на языке GO обладающий следующими функциями:**

* авторизация клиента на ssh-сервере;
* передача клиенту список содержимого заданной директории ssh-сервера по запросу;
* возможность создавать директории на ssh-сервере по запросу;
* возможность удалять директории на ssh-сервере по запросу.

**Реализовать клиент-серверное приложение на языке GO позволяющее автоматизировать выполнение вышеупомянутых команд на нескольких серверах**

**Исходный код ssh-клиента:**

package main

///Импорт библиотек

import (

    "bufio"

    "flag"

    "fmt"

    "log"

    "os"

    "strings"

    "golang.org/x/crypto/ssh"

)

///Конфигурация аргументов командной строки по умолчанию

var (

    user     = flag.String("u", "tarasov", "User name")

    password = flag.String("pwd", "\*\*\*\*\*\*\*\*\*", "Password")

    host     = flag.String("h", "lab2.posevin.com", "Host")

    port     = flag.String("p", "22", "Port")

)

var client \*ssh.Client

///Точка входа в программу

func main() {

    flag.Parse()

    ///Конфигурация ssh клиента

    config := &ssh.ClientConfig{

        User: \*user,

        Auth: []ssh.AuthMethod{

            ssh.Password(\*password),

        },

        HostKeyCallback: ssh.InsecureIgnoreHostKey(),

    }

    var err error

    client, err = ssh.Dial("tcp", \*host+":"+\*port, config)

    if err != nil {

        panic(err)

    }

    fmt.Printf("Connected to %s:%s ...", \*host, \*port)

    reader := bufio.NewReader(os.Stdin)

    ///Считываем ввод до ctrl+C или до exit

    for {

        fmt.Printf("> ")

        cmd, \_ := reader.ReadString('\n')

        execcmd(strings.TrimSpace(cmd))

        if cmd == "exit" {

            return

        }

    }

}

func execcmd(cmd string) {

    session, \_ := client.NewSession()

    resp, err := session.Output(cmd)

    if err != nil {

        log.Println(err)

    }

    if string(resp) != "" {

        log.Println(string(resp))

    }

    defer session.Close()

}

**Исходный код ssh-сервера:**

package main

///Импорт библиотек

import (

    "flag"

    "fmt"

    "log"

    "os/exec"

    "strings"

    "github.com/gliderlabs/ssh"

    "golang.org/x/crypto/ssh/terminal"

)

///Конфигурация аргументов командной строки по умолчанию

var (

    user     = flag.String("u", "tarasov", "User name")

    password = flag.String("pwd", "\*\*\*\*", "Password")

    host     = flag.String("h", "localhost", "Host")

    port     = flag.String("p", "9034", "Port")

)

/// Обработчик запросов

func handler(session ssh.Session) {

    var util, res string

    var rawres []byte

    var args []string

    log.Println("user connected")

    ///Создаем терминал

    term := terminal.NewTerminal(session, "> ")

    ///Считаем ввод до ctrl+C или exit

    for {

        cmd, err := term.ReadLine()

        log.Println(string(cmd))

        if err != nil {

            break

        }

        parts := strings.Fields(string(cmd))

        args = nil

        util = parts[0]

        if len(parts) > 1 {

            args = parts[1:]

        }

        log.Printf("exec %s + %s", util, args)

        if len(parts) > 0 {

            rawres, err = exec.Command(util, args...).Output()

        } else {

            exec.Command(util)

        }

        res = string(rawres)

        if res != "" {

            log.Println("res:", res)

            b, err := fmt.Fprintf(session, "%s", util)

            if err != nil {

                log.Println(b, err)

            }

        }

    }

    log.Println("terminal closed")

}

///Точка входа в программу

func main() {

    ssh.Handle(handler)

    log.Println("ssh server started on port 9034...")

    ///Запус сервера, логгирование в случае ошибки

    log.Fatal(ssh.ListenAndServe(

        \*host+":"+\*port,

        nil,

        ssh.PasswordAuth(func(ctx ssh.Context, pass string) bool {

            return pass == \*password

        }),

    ))

}

**Исходный код клиент серверного приложения:**

package main

///Импорт библиотек

import (

    "bufio"

    "encoding/json"

    "flag"

    "fmt"

    "log"

    "net"

    "os"

    "strings"

    "time"

    "golang.org/x/crypto/ssh"

)

type sshConfig struct {

    User, Password, Host string

}

type configuration struct {

    Hosts []sshConfig

}

type cmdTask struct {

    cmd  string

    host net.Addr

}

var (

    cfg     configuration

    clients []\*ssh.Client

    channel chan cmdTask

)

//Парсинг конфигурации из json

func parsecfg() {

    file, err := os.Open("./cfg.json")

    if err != nil {

        panic(err)

    }

    defer file.Close()

    decoder := json.NewDecoder(file)

    cfg = configuration{}

    err = decoder.Decode(&cfg)

    if err != nil {

        panic(err)

    }

}

///Точка входа в программу

func main() {

    fmt.Println("Welcome to multy ssh client")

    flag.Parse()

    channel = make(chan cmdTask)

    timeout := time.After(25 \* time.Second)

    parsecfg()

    fmt.Println(cfg)

    clients = make([]\*ssh.Client, len(cfg.Hosts))

    for i, h := range cfg.Hosts {

        client, e := ssh.Dial("tcp", h.Host, &ssh.ClientConfig{

            User: h.User,

            Auth: []ssh.AuthMethod{

                ssh.Password(h.Password),

            },

            HostKeyCallback: ssh.InsecureIgnoreHostKey(),

        })

        if e != nil {

            log.Println(e.Error())

        }

        clients[i] = client

    }

    reader := bufio.NewReader(os.Stdin)

    for {

        cmd, \_ := reader.ReadString('\n')

        if strings.TrimSpace(cmd) == "" {

            continue

        }

        for \_, c := range clients {

            go execcmd(strings.TrimSpace(cmd), c)

        }

        for i := len(clients); i != 0; {

            select {

            case res := <-channel:

                fmt.Printf("hostname: %s\n", res.host)

                fmt.Println(res.cmd)

                i--

            case <-timeout:

                fmt.Println("Execution timeout...")

                break

            }

        }

        if cmd == "exit" {

            return

        }

    }

}

func execcmd(cmd string, c \*ssh.Client) {

    start := time.Now()

    session, e := c.NewSession()

    if e != nil {

        panic(e)

    }

    o, e := session.Output(cmd)

    channel <- cmdTask{host: c.Conn.RemoteAddr(),

        cmd: string(o)}

    fmt.Printf("Estimated time %f\n", time.Now().Sub(start).Seconds())

    defer session.Close()

}