

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего
образования
«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Кафедра комплексной информационной безопасности электронно-вычислительных
систем
(КИБЭВС)

Сокеты

Отчёт по лабораторной работе №7
по дисциплине «Системное программирование»

Выполнил: Студент 717-1 гр.

Калжан К.Ж

«__»_____ 2021 г.

Проверил: Преподаватель

кафедры КИБЭВС

Полюга В.А

«__»_____ 2021 г.

1 Введение

Цель работы познакомиться с основными аспектами работы с сокетами и познакомиться с соответствующими функциями WinAPI и POSIX API.

Задания:

1. Изучить краткие теоретические сведения и лекционный материал по теме практического задания.
2. Реализовать примеры клиентских программ для обмена сообщениями с серверами TCP и UDP для Unix/Linux.
3. Написать отчет и защитить у преподавателя.

2 Ход работы

2.1 Изучение теоретического материала

В ходе изучения теоретических материалов ознакомился с работой сокетами, а после выполняю индивидуальную программу на высоком языке программирования.

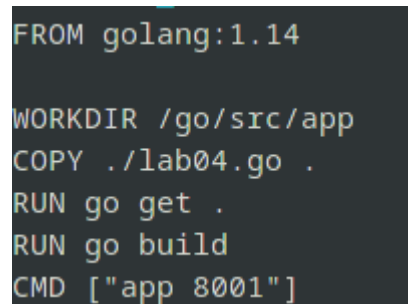
2.2 Подготовка образа

Подготовим образ операционной системы Linux для Docker для запуска программ:

- 1) Напишем Dockerfile;
- 2) Напишем программы по заданию.

Распишем составляющий Dockerfile:

- 1) Строиться образ будет на Golang;
- 2) Пакеты которые установим для работы с GO(рисунок 2.1)



```
FROM golang:1.14
WORKDIR /go/src/app
COPY ./lab04.go .
RUN go get .
RUN go build
CMD ["app 8001"]
```

Рисунок 2.1 - установочные пакеты

2.3 Сокеты

2.3.1 TCP

TCP — Transfer Control Protocol. Протокол управления передачей. Он служит для обеспечения и установление надежного соединения между двумя устройствами и надежную передачу данных. При этом протокол TCP контролирует оптимальный размер передаваемого пакета данных, осуществляя новую посылку при сбое передачи

Для показа работы протокола реализуем сервер и клиент tcp.

2.3.1.1 Сервер

Реализация программы будет на языке программирования GO

При написания программы в первую очередь мы инициализируем работу server , который будет ждать ответов(рисунок 2.2)

```
func main(){
    arguments := os.Args
    if len(arguments) == 1{
        fmt.Println("Please provide host:port")
        return
    }
    PORT := ":" + arguments[1]
    l, err := net.Listen("tcp", PORT)
    if err != nil{
        fmt.Println(err)
        return
    }
    defer l.Close()
    c, err := l.Accept()
    if err != nil{
        fmt.Println(err)
        return
    }
}
```

Рисунок 2.2 - Инициализация сервера

После того как мы сделали инициализацию сервера запускаем его выполнение в бесконечном цикле(рисунок 2.3).

```
for {
    netData, err := bufio.NewReader(c).ReadString('\n')
    fmt.Println("start")
    if err != nil{
        fmt.Println(err)
        return
    }
    if strings.TrimSpace(string(netData)) == "STOP"{
        fmt.Println("Exiting TCP server")
        return
    }
    fmt.Print("->", string(netData))
    t := time.Now()
    myTime := t.Format(time.RFC3339) + "\n"
    c.Write([]byte(myTime))
}
```

Рисунок 2.3 - Бесконечный цикл

2.3.1.2 Клиент

Для общения с написанным сервером tcp реализуем клиента.

Для работы клиента мы передаем ему ip-address и port на котором работает сервер.

Клиент как только определит сервер он с ним соединиться(рисунок 2.4)

```
arguments := os.Args
if len(arguments) == 1{
    fmt.Println("Please provide host:port")
    return
}
CONNECT := arguments[1]
c, err := net.Dial("tcp", CONNECT)
if err != nil{
    fmt.Println(err)
    return
}
```

Рисунок 2.4 - Соединение клиента с сервером

После соединения мы также запускаем клиента в бесконечном циклу, для того чтобы выполнение выполнялось до момента нашего прерывания (рисунки 2.5).

```
for {
    reader := bufio.NewReader(os.Stdin)
    fmt.Print(">> ")
    text, _ := reader.ReadString('\n')
    fmt.Fprintf(c, text+"\n")
    message, _ := bufio.NewReader(c).ReadString('\n')
    fmt.Print("->: "+message)
    if strings.TrimSpace(string(text)) == "STOP" {
        fmt.Println("TCP client exiting....")
        return
    }
}
```

Рисунок 2.5 - Цикл клиента протокола tcp

Работу сервера и клиента представлена на рисунке 2.6

```
~/CN/lab_6 >>> go run ./tcp_server.go 8001
start
->Hello!
start
->Чт ты работаешь
start
Exiting TCP server
~/CN/lab_6 >>>

~/CN/lab_6 >>> go run ./clients_tcp.go localhost:8001
-> Hello!
->: 2021-03-25T12:59:24+07:00
-> А ты работаешь
->: 2021-03-25T12:59:38+07:00
-> STOP
->: TCP client exiting....
~/CN/lab_6 >>> |
```

Рисунок 2.6 - Работы сервера и клиента tcp

2.3.2 UDP

UDP — это транспортный протокол пользовательских датаграмм из набора правил TCP/IP. Позволяет отправлять информацию (датаграммы) по IP-сети без предварительного установления соединения и создания специального виртуального канала или путей данных. Официально был разработан в 1980 году человеком по имени Дэвид П. Рид. Полностью расшифровывается как — User Datagram Protocol.

2.3.2.1 Сервер

При написании программы в первую очередь мы инициализируем работу server, который будет ждать ответов (рисунки 2.2)

```
arguments := os.Args
if len(arguments) == 1 {
    fmt.Println("Please provide port number!")
    return
}
PORT := ":" + arguments[1]
l, err := net.ResolveUDPAddr("udp4", PORT)
if err != nil {
    fmt.Println(err)
    return
}
c, err := net.ListenUDP("udp4", l)
if err != nil {
    fmt.Println(err)
    return
}
defer c.Close()
buffer := make([]byte, 1024)
rand.Seed(time.Now().Unix())
```

Рисунок 2.2 - Инициализация сервера

После того как мы сделали инициализацию сервера запускаем его выполнение в бесконечном цикле(рисунок 2.3).

```
for {
    n,addr, err := c.ReadFromUDP(buffer)
    fmt.Print("-> ", string(buffer[0:n-1]))

    if strings.TrimSpace(string(buffer[0:n])) == "STOP"{
        fmt.Println("Exiting UDP server")
        return
    }
    data := []byte(strconv.Itoa(    random(1,1001)))
    fmt.Printf("data: %s\n",string(data))
    _, err = c.WriteToUDP(data,addr)
    if err != nil{
        fmt.Println(err)
        return
    }
}
```

Рисунок 2.3 - Бесконечный цикл

2.3.2.2 Клиент

Для общения с написанным сервером udo реализуем клиента.

Для работы клиента мы передаем ему ip-address и port на котором работает сервер.

Клиент как только определит сервер он с ним соединиться(рисунок 2.4)

```
arguments := os.Args
if len(arguments) == 1{
    fmt.Println("Please provide host:port strings")
    return
}
CONNECT := arguments[1]
s,err := net.ResolveUDPAddr("udp4",CONNECT)
c, err := net.DialUDP("udp4",nil,s)
if err != nil{
    fmt.Println(err)
    return
}
fmt.Printf("The Udp server is %s\n",c.RemoteAddr().String())
defer c.Close()
```

Рисунок 2.4 - Соединение клиента с сервером

После соединения мы также запускаем клиента в бесконечном циклу,для того чтобы выполнение выполнялось до момента наша прерывание(рисунок 2.5).

```

for {
    reader := bufio.NewReader(os.Stdin)
    fmt.Print(">> ")
    text, _ := reader.ReadString('\n')
    data := []byte(text + "\n")
    _, err = c.Write(data)
    if strings.TrimSpace(string(data)) == "STOP" {
        fmt.Println("TCP client exiting...")
        return
    }
    if err != nil {
        fmt.Println(err)
        return
    }
    buffer := make([]byte, 1024)
    n, _, err := c.ReadFromUDP(buffer)
    if err != nil {
        fmt.Println(err)
        return
    }
    fmt.Printf("Reply: %S\n", string(buffer[0:n]))
}

```

Рисунок 2.5 - Цикл клиента протокола udp

Работу сервера и клиента представлена на рисунке 2.6

```

~/CN/lab_6 >>> go run ./udp_server.go 8002
-> Hello!
data: 200
-> Start
data: 787
-> STOP
Exiting UDP server
~/CN/lab_6 >>>

~/CN/lab_6 >>> go run ./clients_udp.go localhost:8002
The Udp server is 127.0.0.1:8002
>> Hello!
Reply: 200
>> Start
Reply: 787
>> STOP
TCP client exiting...
~/CN/lab_6 >>>

```

Рисунок 2.6 - Работы сервера и клиента udp

3 Заключение

В ходе лабораторной работы ознакомились с основными аспектами работы с сокетами и ознакомились с соответствующими функциями WinAPI и POSIX API.

Задания:

1. Изучили краткие теоретические сведения и лекционный материал по теме практического задания.
2. Реализовать примеры клиентских программ для обмена сообщениями с серверами TCP и UDP для Unix/Linux.
3. Написать отчет и защитить у преподавателя.

Список источников

- 1) Github [Электронный ресурс] - Режим доступа:
https://github.com/kasymhan/sp_lab_7 (дата обращения: 25.03.2021).