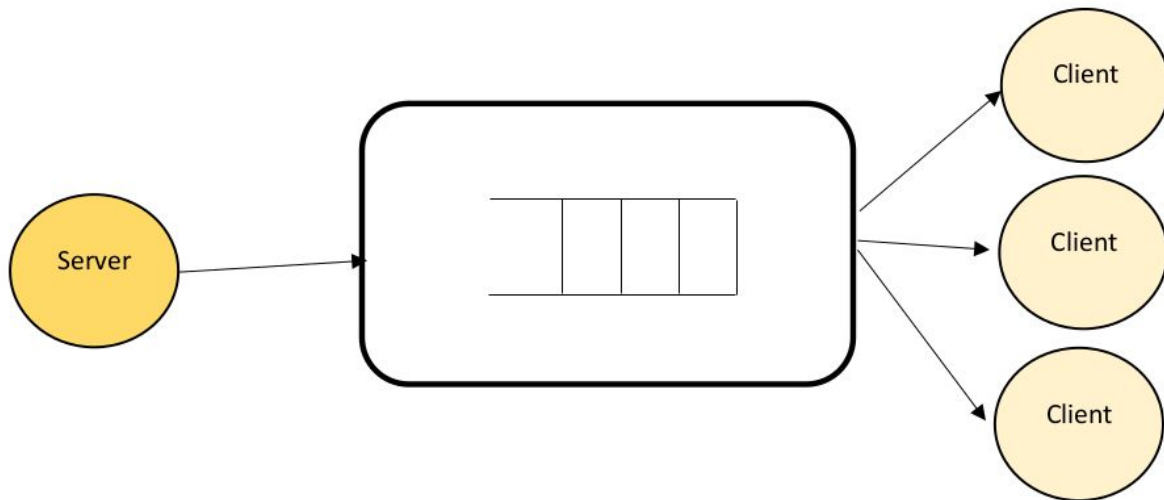


DISTRIBUTED SYSTEMS CA1 REPORT

Queuing using broker, server and multiple clients



محمد صابر ابراهيم نژاد 810196400

سيد پارسا حسيني نژاد 8100886604

معرفی

در این پروژه به پیاده سازی Broker که از یک صف تشکیل شده است پرداخته می‌شود. پروژه این گونه عمل می‌کند که در ابتدا server یک پیام به Broker ارسال می‌کند. حال این پیام در یک صف در Broker ذخیره می‌شود و هنگامی که به سر صف رسید، از صف خارج شده و به اولین client که قابلیت پذیرفتن پیام را دارد ارسال می‌کند.

پیاده سازی

این پروژه با استفاده از زبان Go پیاده سازی شده و کدهای مربوط نیز پیوست شده است. برای پیاده سازی این برنامه، سه بخش کلی پیاده سازی شده است:

1. Server : برای پیاده سازی سرور، ابتدا 2 ورودی گرفته می‌شود که ورودی اول آدرس Broker و ورودی دوم نوع پیاده سازی (sync یا async) است. سپس، یک اتصال بین سرور و Broker به وجود می‌آید.

```
if len(os.Args) != 3 {  
    fmt.Println("Wrong input format")  
    os.Exit(1)  
}  
  
brokerAddress := os.Args[1]  
serverType := os.Args[2]  
var input string  
  
broker, err := net.ResolveTCPAddr("tcp4", brokerAddress)  
checkError(err)  
  
conn, err := net.DialTCP("tcp", nil, broker)  
checkError(err)
```

حال سرور شروع به گرفتن ورودی از کاربر و ارسال آن به Broker می‌کند. اگر نوع پیاده سازی synchronous بود، سرور منتظر می‌ماند تا ack مربوط به بسته‌ی ارسالی دریافت شود و اگر از نوع asynchronous بود، تابع sendMessage به صورت موازی در یک thread دیگر اجرا می‌شود تا منتظر دریافت ack نماند و به ارسال بقیه‌ی بسته‌ها می‌پردازد.

```

for {
    fmt.Scanln(&input)
    if input == "exit" {
        break
    }

    if serverType == "sync" {
        sendMessage(conn)
    }
    if serverType == "async" {
        go sendMessage(conn)
    }
}

```

در تابع `sendMessage` نیز یک ورودی از کاربر گرفته شده و با Broker ارسال می‌شود و منتظر دریافت `ack` می‌ماند. وقتی `ack` دریافت شد، این ارسال تمام شده و به سراغ ارسال بعدی می‌رویم.

```

func sendMessage(conn net.Conn) {
    input := make([]byte, 512)
    ack := make([]byte, 512)

    fmt.Scanln(&input)

    _, err := conn.Write(input)
    if err != nil {
        fmt.Println("Connection error during sending message")
        return
    }

    _, err = conn.Read(ack)
    if (err != nil) {
        fmt.Println("Acknowledge not recived for this message")
        return
    }

    fmt.Println("Ack recieved for message", input)
}

```

2. Broker : این قسمت وظیفه‌ی دریافت بسته از سرور و پخش آنها بین client ها را دارد. در ابتدا پورت‌های مربوط به خود Broker و client ها از ورودی دریافت می‌شود. سپس یک سوکت روی پورت مربوطه ساخته شده و به سرور متصل می‌شود.

```
func connectToServer(serverAddress string) net.Conn {
    fmt.Println("Broker is listening for a server...")
    service := serverAddress
    tcpAddr, err := net.ResolveTCPAddr("tcp4", service)
    checkError(err)
    listener, err := net.ListenTCP("tcp", tcpAddr)
    checkError(err)

    for {
        conn, err := listener.Accept()
        if err == nil {
            fmt.Println("Server connected!")
            return conn
        }
    }
}
```

حال این connection در یک متغیر گلوبال ذخیره می‌شود. حال برای دریافت پیام از سرور، یک thread جدید می‌سازیم تا از سرور پیام گرفته و روی صف که همان channel است بریزد. دقت شود که صف تعریف شده سائز 10 دارد و پیام‌ها در آن به صورت صف buffer می‌شوند. این thread همواره در حال اجراست.

```
func readMessage() {
    for {
        message := make([]byte, 512)
        serverCon.Read(message)
        channel <- string(message)
    }
}
```

حال شروع به اتصال به client ها می‌کنیم و هر client را که میتوان به آن متصل شد را پیدا کردیم، به آن عضو سر صف را ارسال میکنیم. دقت شود که وقتی یک پیام به یک client ارسال می‌شود، آن اتصال بسته می‌شود و هر وقت process مربوط به آن پیام در client تمام شد، درخواست اتصال به Broker می‌دهد. هنگامی هم که پیام از صف خارج شد و به client ارسال شد، ack مربوط به آن نیز به server ارسال می‌شود. این ارسال به client ها هم به صورت موازی و با استفاده از goroutine انجام می‌شود.

```
func connectToClient(clientAddress string) {
    fmt.Println("Broker is listening for new clients...")
    service := clientAddress
    tcpAddr, err := net.ResolveTCPAddr("tcp4", service)
    checkError(err)
    listener, err := net.ListenTCP("tcp", tcpAddr)
    checkError(err)

    for {
        conn, err := listener.Accept()
        if err == nil {
            fmt.Println("New client connected!")
            go runBroker(conn)
        }
    }
}
```

```
func runBroker(conn net.Conn) {
    message := <-channel
    conn.Write([]byte(message))
    serverCon.Write([]byte("ack"))
    conn.Close()
    fmt.Println("Message delivered")
}
```

3. Client : این بخش وظیفه‌ی دریافت پیام‌ها و process کردن آن‌ها را بر عهده دارد. در این قسمت نیز آدرس Broker از کاربر دریافت شده و به آن متصل می‌شویم. حال در صورت اتصال پیام worker ready به Broker ارسال می‌شود تا به این client پیام بفرستد. حال پیام دریافت شده را می‌خوانیم و thread مربوطه را به مقدار رندوم بین 1 تا 11 ثانیه sleep می‌کنیم تا نمایانگر انجام process روی پیام باشد. بعد این زمان، client دوباره درخواست دریافت پیام را به Broker ارسال می‌کند.

```

if len(os.Args) != 2 {
    fmt.Println("Wrong input format")
    os.Exit(1)
}
brokerAddress := os.Args[1]

var sleep_delay int
task := make([]byte, 512)

broker, err := net.ResolveTCPAddr("tcp4", brokerAddress)
checkError(err)
for true {
    conn, err := net.DialTCP("tcp", nil, broker)
    checkError(err)

    _, err = conn.Write([]byte("woker ready"))
    checkError(err)

    _, err = conn.Read(task)
    if err != nil {
        continue
    }

    if string(task) == "close" {
        fmt.Println("Client closed!")
        break
    }

    sleep_delay = rand.Intn(10) + 1
    fmt.Println("Message", string(task), "recieved and now sleeping for", sleep_delay, "seconds")
    time.Sleep(time.Duration(sleep_delay) * time.Second)
}
os.Exit(0)

```

نحوه‌ی اجرا

برای اجرای این برنامه ابتدا نیاز به دو پورت آزاد از دستگاه می باشد. برای اجرا ابتدا فایل broker.go را به همراه پورت ارتباط با سرور و پورت ارتباط کلاینت ها اجرا می کنیم.

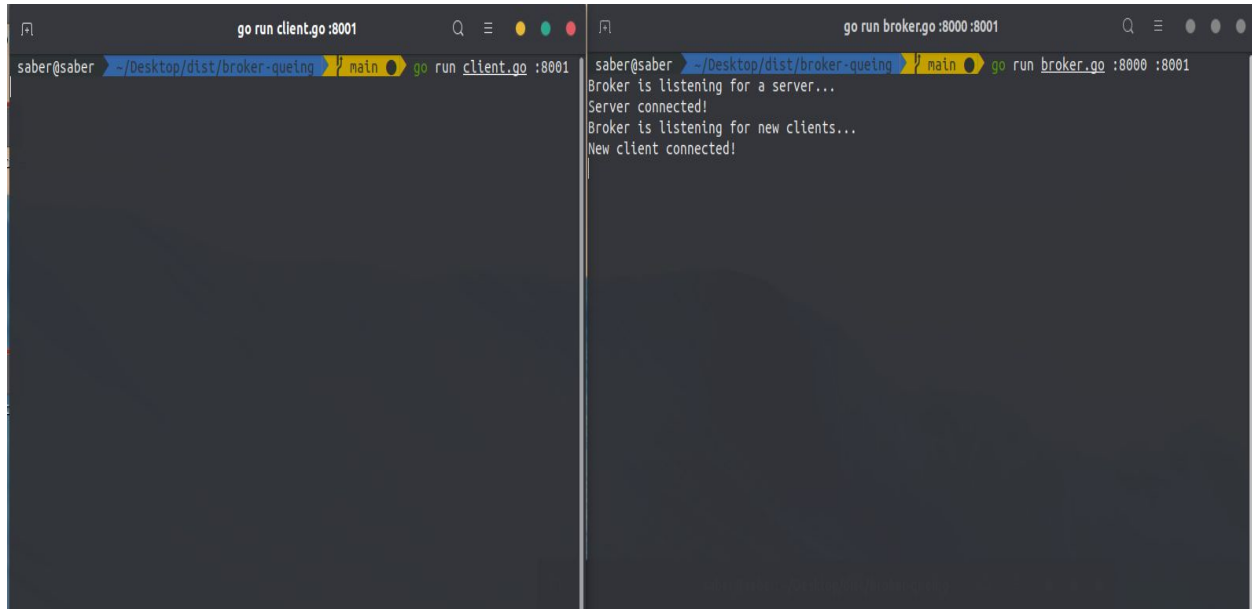
```
go run broker.go :8000 :8001
saber@saber ~/Desktop/dist/broker-queing: main go run broker.go :8000 :8001
Broker is listening for a server...
```

پس از آن فایلی که شمال کد های سرور می باشد را به همراه پورت broker و نوع ارتباط با broker با استفاده از go اجرا می کنیم. در این مرحله broker ارتباط با سرور را تایید می کند و از این پس منتظر ارتباط کلاینت ها می شود تا پیام هایی که از سرور دریافت می کند را منتقل نماید.

```
go run server.go :8000 async
saber@saber ~/Desktop/dist/broker-queing: main go run server.go :8000
async

go run broker.go :8000 :8001
saber@saber ~/Desktop/dist/broker-queing: main go run broker.go :8000 :8001
Broker is listening for a server...
Server connected!
Broker is listening for new clients...
```

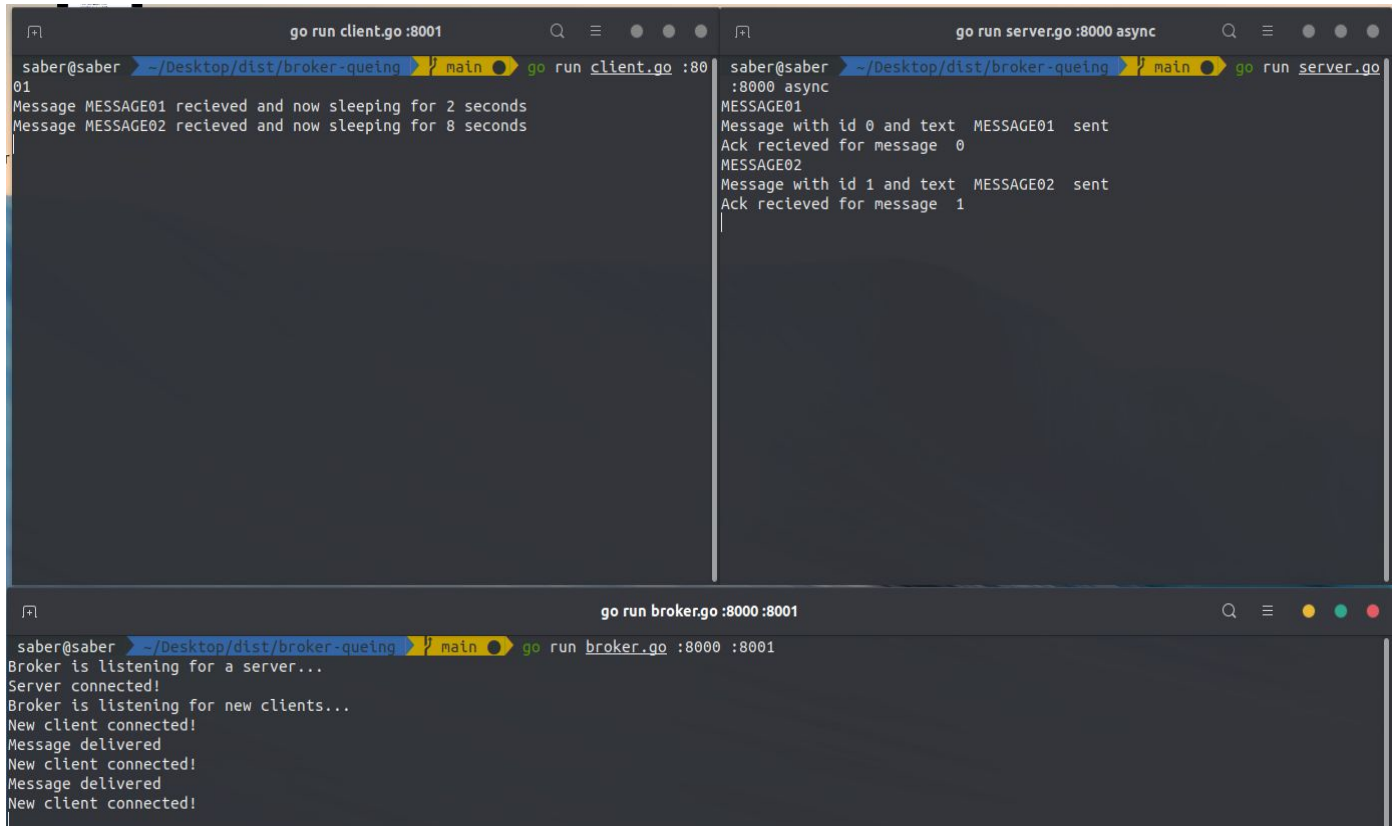
در این مرحله broker در هنگام وصل شدن هر کلاینت حضورش را تایید می کند و در صورت وجود پیام به آن ارسال می نماید.



```
go run client.go :8001
saber@saber ~/Desktop/dist/broker-queing: main go run client.go :8001

go run broker.go :8000:8001
saber@saber ~/Desktop/dist/broker-queing: main go run broker.go :8000 :8001
Broker is listening for a server...
Server connected!
Broker is listening for new clients...
New client connected!
```

در ادامه هر پیامی که در ترمینال server نوشته شود از طریق broker به کلاینت می رود و پس از آن client برای زمانی محدودی از دسترس خارج شده و پس از آن دوباره آماده دریافت پیام می شود.



```
go run client.go :8001
saber@saber ~/Desktop/dist/broker-queing: main go run client.go :8001
Message MESSAGE01 recieved and now sleeping for 2 seconds
Message MESSAGE02 recieved and now sleeping for 8 seconds

go run server.go :8000 async
saber@saber ~/Desktop/dist/broker-queing: main go run server.go :8000 async
MESSAGE01
Message with id 0 and text MESSAGE01 sent
Ack recieved for message 0
MESSAGE02
Message with id 1 and text MESSAGE02 sent
Ack recieved for message 1

go run broker.go :8000 :8001
saber@saber ~/Desktop/dist/broker-queing: main go run broker.go :8000 :8001
Broker is listening for a server...
Server connected!
Broker is listening for new clients...
New client connected!
Message delivered
New client connected!
Message delivered
New client connected!
```


نکات تکمیلی

در هنگامی که broker با Buffer Overflow مواجه شود با ارسال پیام "Broker queue is full!" از دریافت سایر پیام ها تا زمان رفع Buffer overflow خودداری می کند تا به شرایط عادی بازگردد.

```
func readMessage() {  
    for {  
        message := make([]byte, 512)  
        serverCon.Read(message)  
        if len(channel) == queueSize {  
            serverCon.Write([]byte("Broker queue is full!"))  
        } else {  
            channel <- string(message)  
        }  
    }  
}
```