Aplicação Cliente/Servidor Concorrente

Exercício 1 - Programação com Socket

Grupo: Matheus Viana Coelho Albuquerque
Natália Souza Soares

Roteiro

- A aplicação
- Implementação com o protocolo TCP
- Implementação com o protocolo UDP
- Avaliação de desempenho
- Conclusão
- Demonstração da Execução

A aplicação

- Requisitos:
  - Arquitetura Cliente/Servidor
  - Suporte à 2 ou mais clientes
  - Suporte à requisições concorrentes
  - Utilizar API de socket (pacote net) - TCP e UDP

- Funcionalidade:
  - Armazenamento de dados
  - **Cliente** - envia dados para serem armazenados no servidor
  - **Servidor** - recebe os dados e armazena-os em um arquivo retornando o status da operação

Implementação TCP :: Servidor

```go
func main() {
    // Inicializa o servidor na porta 8080 e protocolo TCP
    port := ":8080"
    l, err1 := net.Listen("tcp", port)
    checkError(err1)

    // Fecha o socket no final da execução
    defer l.Close()

    for {
        // Aceita conexões e inicia a rotina de tratamento (goroutine)
        conn, err2 := l.Accept()
        checkError(err2)

        // Lida com a requisição do client em uma goroutine
        go handleConn(conn)
    }
}
```

Lida com a questão da concorrência de requisições

Implementação TCP :: Servidor

```
stp := true
for stp{
    commandName, err := user.reader.ReadString(' ')
    checkError(err)

    switch commandName {
    case "MSG ":
        // Abre arquivo de saida
        nameDataBase := "./data_bases/dataBase" + conn.RemoteAddr().String() + ".csv"
        dataBase, err := os.OpenFile(nameDataBase,os.O_APPEND|os.O_WRONLY, 0600)
        if err != nil {
            panic(err)
        }
        // Recebe os dados
        data := receiveMessage(&user)
        // Escreve no arquivo o dado recebido, verificando possíveis erros na escrita
        writeFile(&user,data,"Data Stored!\n",dataBase)
        // Fecha o arquivo
        dataBase.Close()
        if err != nil {
            panic(err)
        }
    case "STOP ":
        // Abre arquivo de saida
        nameDataBase := "./data_bases/dataBase" + conn.RemoteAddr().String() + ".csv"
        dataBase, err := os.OpenFile(nameDataBase,os.O_APPEND|os.O_WRONLY, 0600)
        if err != nil {
            panic(err)
        }
        // Recebe os dados
        data := receiveMessage(&user)
        // Escreve no arquivo o dado recebido, verificando possíveis erros na escrita
        writeFile(&user,data,"STOP\n",dataBase)
        // Fecha o arquivo
        dataBase.Close()
        if err != nil {
            panic(err)
        }
        // Fecha a conexão e encerra o hadle desse client
        conn.Close()
        stp = false
    default:
        errMsg := "Unknown command\n"
```

```
func receiveMessage(user *client) string {
    // Recebe o nome do client
    msg, err := (*user).reader.ReadString('\n')
    checkError(err)

    return msg
}

func sendMessage(user *client, msg string){
    (*user).conn.Write([]byte(string(msg)))
}

func writeFile(user *client, data string, logMsg string, dataBase *os.File){
    t := time.Now().UTC()
    if _, err := dataBase.Write([]byte(t.Format("2006,01,02") + "," + data)); err != nil {
        // Retorna uma mensagem de status negativo em caso de erro
        logMsg = err.Error()
        sendMessage(user,logMsg)
        panic(err)
    }

    // Retorna uma mensagem de status possitivo
    sendMessage(user,logMsg)
}
```

Implementação TCP :: Cliente

```go
func main(){
    // Conectando ao servidor na máquina local na porta 8080 (default)
    server := "127.0.0.1:8080"
    conn, err := net.Dial("tcp", server)
    checkError(err)

    // Cria um objeto do tipo client
    user := client{"clientName",conn,bufio.NewReader(conn)}

    // Mandando o nome do client para o server e aguarda o retorno
    sendMessage(&user,"MSG " + user.name + "\n")
    receiveMessage(&user)

    x := 0.000000
    for i:=0; i <= 1E4; i++{
        if i == 1E4{
            sendMessage(&user,"STOP " + strconv.FormatFloat(x,'f',6,64) +"\n")
            receiveMessage(&user)
        }else{
            time1 := time.Now()
            sendMessage(&user,"MSG " + strconv.FormatFloat(x,'f',6,64) +"\n")
            receiveMessage(&user)
            time2 := time.Now()
            x = float64(time2.Sub(time1).Nanoseconds()) / 1E3
        }
    }
}
```

```go
func sendMessage(user *client, msg string){
        _,err := (*user).conn.Write([]byte(msg))
        checkError(err)
}

func receiveMessage(user *client){
    _, err := (*user).reader.ReadString('\n')
    checkError(err)

    // Escrevendo a resposta do servidor no term
    //fmt.Printf(menssage)
}
```

Implementação UDP :: Servidor

```go
func main() {
    // Inicializa o servidor na porta 8080 e protocolo UDP
    //fmt.Println("Server waiting for connections...")
    port := ":8080"

    service := "localhost" + port
    // Retorna um endereço de udp
    udpAddr, err := net.ResolveUDPAddr("udp", service)
    checkError(err)

    // Se conecta ao endereço na rede nomeada. (udpAddr)
    // Conecta o servidor
    l, err := net.ListenUDP("udp", udpAddr)
    checkError(err)

    // Fecha o socket na saida
    defer l.Close()

    for {
        // ao receber uma nova mensagem cria uma thread para ministrar o que foi recebido
        msgFromClient, addr := receiveMessage(l)
        go handleConn(l, msgFromClient, addr)
    }
}
```

```go
func receiveMessage(conn *net.UDPConn) (string, *net.UDPAddr) {
    // Recebe a mensagem do cliente
    buffer := make([]byte, 1024)
    n, addr, err := conn.ReadFromUDP(buffer)
    checkError(err)
    //fmt.Println("receiveMessage: ", string(buffer[:n]))

    return string(buffer[:n]), addr
}
```

Implementação UDP :: Servidor

```go
func handleConn(conn *net.UDPConn, msg string, addr *net.UDPAddr) {

    var user client
    var dataBase *os.File

    i := strings.IndexByte(msg, ' ')
    commandName := msg[:i]
    //fmt.Println("Comando separado", commandName)

    switch commandName {
    case "MSG":

        // Verifica se ta salvo
        if isNewUser(addr.String(), users) {
            user = client{msg[i+1:], conn, addr}
            users = append(users, user)

            // open output file
            nameDataBase := "./data_bases/dataBase" + addr.String() + ".csv"
            var err error
            dataBase, err = os.Create(nameDataBase)
            checkError(err)

        } else {
            var err error
            dataBase, err = os.OpenFile("./data_bases/dataBase"+addr.String()+".csv", os.O_APPEND|os.O_WRONLY, 0600
            checkError(err)
        }

        data := msg[i+1:]
        //fmt.Println("dado da mensagem:", data)
        // Escrevo no arquivo o que foi recebido junto com um formato de tempo
        t := time.Now().UTC()
        if _, err := dataBase.Write([]byte(t.Format("2006,01,02") + "," + data)); err != nil {
            panic(err)
        }

        // Fecho o arquivo depois de utilizá-lo
        if err := dataBase.Close(); err != nil {
            panic(err)
        }

        logMsg := "Data Stored!\n"
        sendMessage(addr, logMsg, conn)
```

```go
var users []client

type client struct {
    name string
    conn net.Conn
    addr *net.UDPAddr
}
```

```go
func sendMessage(addr *net.UDPAddr, msg string, conn *net.UDPConn) {
    msgToClient := []byte(msg)
    _, err := conn.WriteToUDP(msgToClient, addr)
    checkError(err)
}
```

Implementação UDP :: Servidor

```
104     case "STOP":
105
106         // Verficia se ta salvo
107 >       if isNewUser(addr.String(), users) {…
116 >       } else {…
120         }
121         data := msg[i+1:]
122         //fmt.Println("dado da mensagem:", data)
123
124         // Escrevendo no arquivo
125         t := time.Now().UTC()
126         if _, err := dataBase.Write([]byte(t.Format("2006,01,02") + "," + data)); err != nil {
127             panic(err)
128         }
129
130         if err := dataBase.Close(); err != nil {
131             panic(err)
132         }
133         stopSign := "STOP\n"
134         sendMessage(addr, stopSign, conn)
135
136     default:
137         errMsg := "Unknown command!\n"
138         sendMessage(addr, errMsg, conn)
139     }
140     return
141 }
```

No caso do UDP, não há conexão a ser fechada.

Implementação UDP :: Cliente

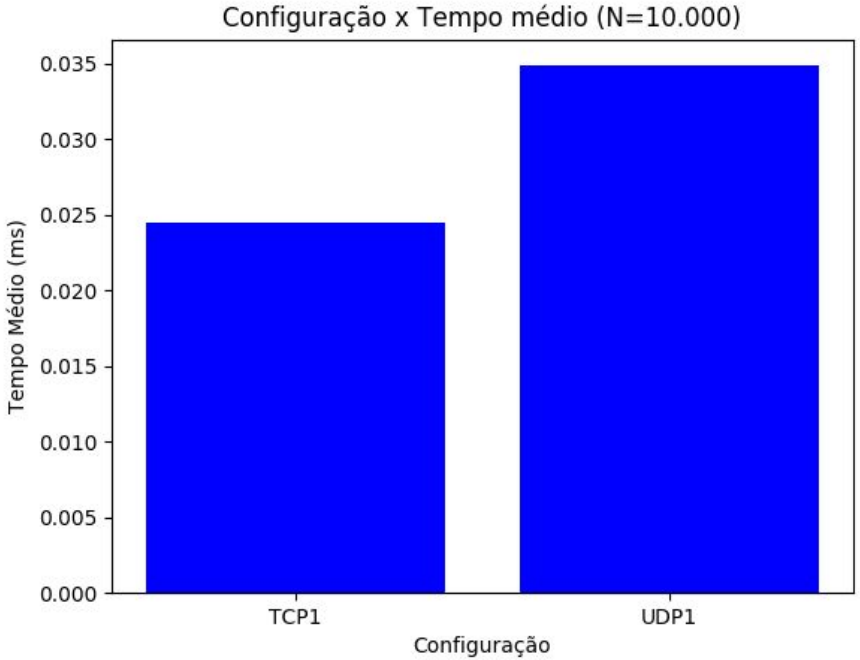
```go
func runClient(clientName string, server string) {
    RemoteAddr, err := net.ResolveUDPAddr("udp", server)
    checkError(err)

    // Conectando ao servidor
    conn, err := net.DialUDP("udp", nil, RemoteAddr)
    checkError(err)
    //fmt.Printf("Connected to server at %s!\n", server)

    user := client{clientName, conn, bufio.NewReader(conn)}

    // Mandando o nome
    sendMessage(&user, "MSG "+user.name+"\n")
    receiveMessage(&user)

    x := float64(NUMCLIENTS)
    for i := 0; i <= 1E4; i++ {
        if i == 1E4 {
            sendMessage(&user, "STOP "+strconv.FormatFloat(x, 'f', 6, 64)+"\n")
            receiveMessage(&user)
        } else {
            time1 := time.Now()
            sendMessage(&user, "MSG "+strconv.FormatFloat(x, 'f', 6, 64)+"\n")
            receiveMessage(&user)
            time2 := time.Now()
            x = float64(time2.Sub(time1).Nanoseconds()) / 1E3
        }
    }
    wg.Done()
}
```

```go
type client struct {
    name   string
    conn   *net.UDPConn
    reader *bufio.Reader
}
```

- Diferentemente do TCP, o UDP não estabelece uma pré conexão com o servidor.

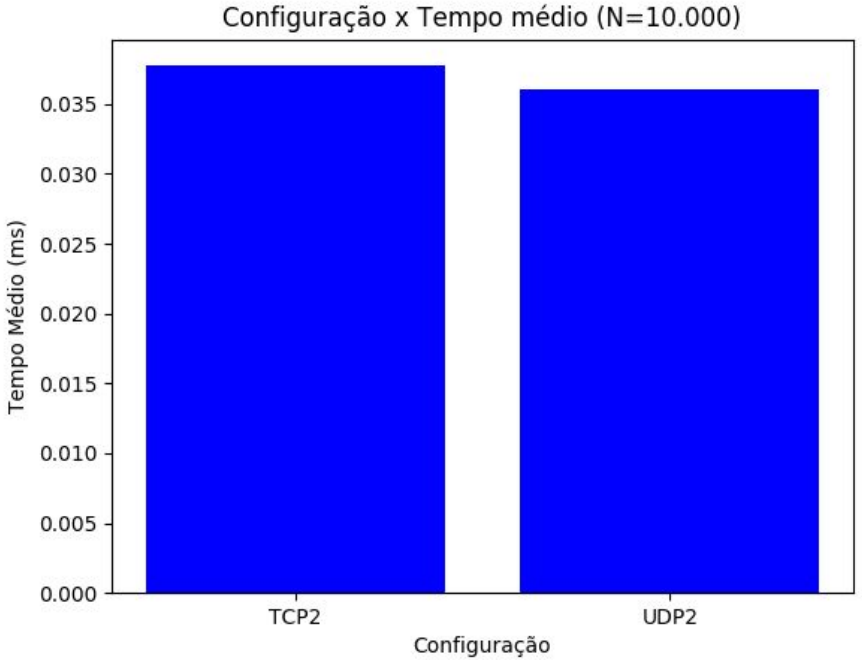
Avaliação de Desempenho :: 1 cliente

- TCP
  - Média = 0.024468 ms
  - Desvio Padrão = 0.009978 ms
- UDP
  - Média = 0.034851 ms
  - Desvio Padrão = 0.023668 ms



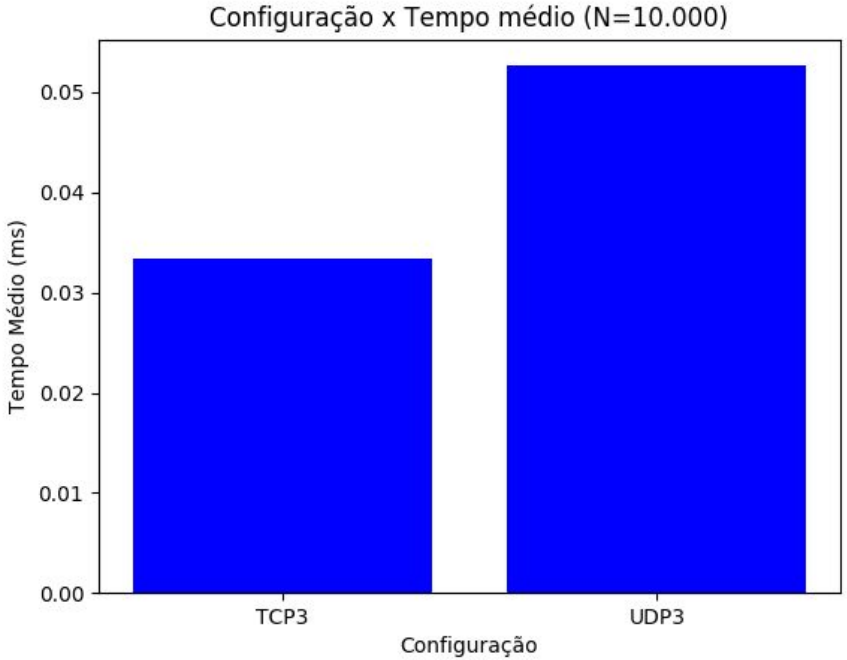
Avaliação de Desempenho :: 2 clientes

- TCP
  - Média = 0.037742 ms
  - Desvio Padrão = 0.026240 ms
- UDP
  - Média = 0.036032 ms
  - Desvio Padrão = 0.011811 ms



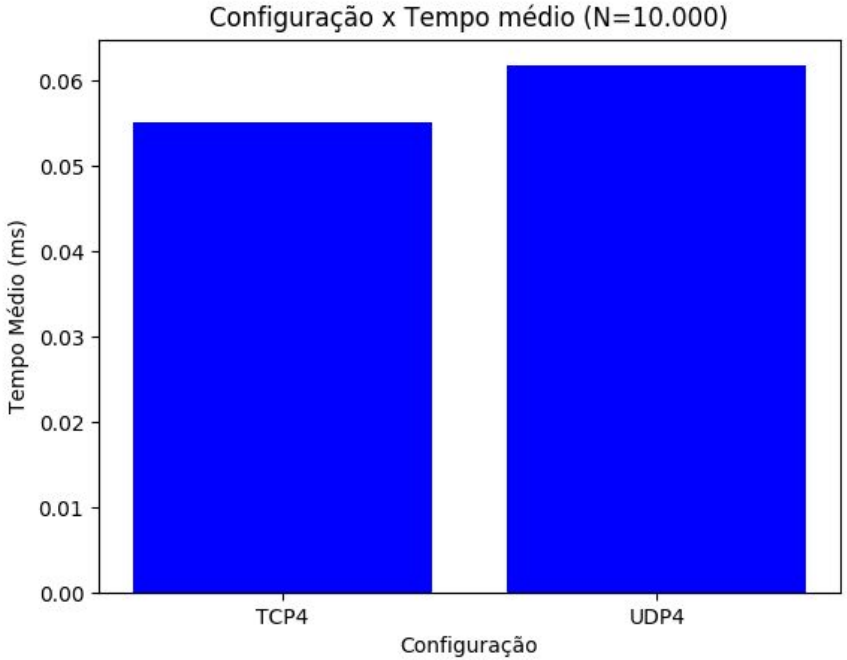
Avaliação de Desempenho :: 3 clientes

- TCP
  - Média = 0.033421 ms
  - Desvio Padrão = 0.056705 ms
- UDP
  - Média = 0.052662 ms
  - Desvio Padrão = 0.050205 ms



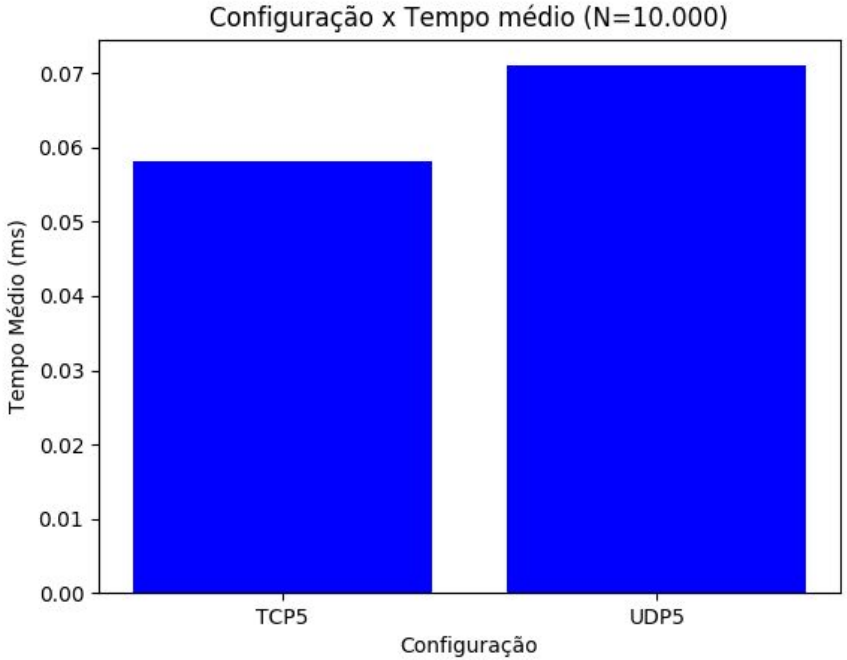
Avaliação de Desempenho :: 4 clientes

- TCP
  - Média = 0.055196 ms
  - Desvio Padrão = 0.058626 ms
- UDP
  - Média = 0.061749 ms
  - Desvio Padrão = 0.046845 ms



Avaliação de Desempenho :: 5 clientes

- TCP
  - Média = 0.058146 ms
  - Desvio Padrão = 0.103675 ms
- UDP
  - Média = 0.071031 ms
  - Desvio Padrão = 0.081084



Conclusões

- TCP mais rápido que UDP nos casos analisados devido às diferenças de implementação