UNIVERSIDADE FEDERAL DE PELOTAS – UFPEL

CENTRO DE DESENVOLVIMENTO TECNOLÓGICO – CDTEC

CIÊNCIA DA COMPUTAÇÃO (BACHARELADO)

REDES DE COMPUTADORES

Leonardo dos Santos Güths - 19100392

PELOTAS – RS

2021

Relatório sobre a Atividade 1 da disciplina de Redes de Computadores, regida pelo professor Guilherme Corrêa.

1. Exercício 1

Para iniciar a atividade, peguei os códigos disponibilizados pelo professor e configurei para ajustar com minhas configurações de IP e porta, como fiz neste trecho de código:

```
from socket import *

serverIP = '192.168.1.106'
serverPort = 4567        # u
```

Obtive o endereço de IP necessário executando o comando “ipconfig” no terminal do Windows, e a porta escolhi de forma aleatória respeitando as limitações recomendadas (número acima de 1023).

Após este processo de configuração inicial, fui testar o programa para entender seu funcionamento e rodei primeiramente o servidor em um terminal:

```
C:\Users\leo_g\Desktop\leo ufpel\oitavo semestre\Redes de Computadores\Atividade 1\ex1>python server.py
Server is on!
```

E o cliente em outro terminal, enviando algumas frases de teste:

```
leo_g@LAPTOP-FPBROQ1P MINGW64 ~/Desktop/leo ufpel/oitavo semestre/Redes de Computadores/Atividade 1/ex1 (main)
$ python client.py
Enter your lower-case word: teste exercicio um
TESTE EXERCICIO UM

leo_g@LAPTOP-FPBROQ1P MINGW64 ~/Desktop/leo ufpel/oitavo semestre/Redes de Computadores/Atividade 1/ex1 (main)
$ python client.py
Enter your lower-case word: server e client usando protocolo udp
SERVER E CLIENT USANDO PROTOCOLO UDP
```

Após verificar que tudo correu bem, fui realizar os próximos exercícios, onde eu teria que mexer no código proporcionado pelo professor.

2. Exercício 2

Para realização do exercício 2, utilizei a mesma preparação do primeiro com a mesma organização de terminais para cliente e servidor.

Para a realização da conexão utilizando o protocolo TCP, modifiquei a criação do socket que antes utilizava SOCK_DGRAM, como apresentado no exemplo abaixo (trecho do código do exercício 1, que utiliza o protocolo UDP):

```
serverSocket = socket(AF_INET,SOCK_DGRAM)
serverSocket.bind((serverIP, serverPort))
```

e alterei para que o socket utilizasse SOCK_STREAM, que é o padrão utilizado para o protocolo TCP:

```
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind((serverIP, serverPort))
serverSocket.listen(1)
```

adicionando também a execução da função “listen(1)” que faz com que o servidor fique aberto a novas solicitações de conexão.

Para a utilização do protocolo TCP foram necessárias mais algumas mudanças breves no código, que foram todas obtidas através dos slides disponibilizados (Unidade 2 - Camada de aplicação, slides 74-82), e podem ser vistas neste trecho:

```
while 1:
    connectionSocket, clientIP = serverSocket.accept()      # server
    message = connectionSocket.recv(1500)
    decodedMessage = message.decode()
    modifiedMessage = decodedMessage.upper()
    encodedMessage = modifiedMessage.encode()
    connectionSocket.send(encodedMessage)                   # sends c
    print ("Message \"" + modifiedMessage + "\" sent successfully.")
```

Algumas mudanças são sutis, como por exemplo o socket utilizado nesta conexão não é direto o do servidor tendo que sempre informar o IP do cliente, mas sim o socket da conexão, que já sabe qual o IP do cliente e apenas envia direto através da conexão já previamente estabelecida.

Após as trocas realizei os mesmos procedimentos do exercício anterior em dois terminais separados para cliente e servidor e os resultados podem ser observados pelos prints abaixo:

```
C:\Users\leo_g\Desktop\leo ufpel\oitavo semestre\Redes de Computadores\Atividade 1\ex2>python server.py
Server is on!
Message "TESTE UM DOIS TRES" sent successfully.
Message "QUATRO CINCO SEIS" sent successfully.
```

```
leo_g@LAPTOP-FPBROQ1P MINGW64 ~/Desktop/leo ufpel/oitavo semestre/Redes de Computadores/Atividade 1/ex2 (main)
$ python client.py
Enter your lower-case word: teste um dois tres
TESTE UM DOIS TRES

leo_g@LAPTOP-FPBROQ1P MINGW64 ~/Desktop/leo ufpel/oitavo semestre/Redes de Computadores/Atividade 1/ex2 (main)
$ python client.py
Enter your lower-case word: quatro cinco seis
QUATRO CINCO SEIS
```

3. Exercício 3

Já para o exercício 3, mantive o protocolo TCP sendo utilizado da mesma forma que no exercício 2, porém algumas mudanças maiores foram implementadas, como serão apresentadas nos prints abaixo:

```
7   clientSocket.connect((serverIP, serverPort))
8
9   option = input ('Select server function:\n1. Lower to Upper\n2. Upper to Lower\n')
0   if (option == '1'):
1       message = input('Enter your lower-case word: ')
2       message = "CB+" + message
3   else:
4       message = input('Enter your upper-case word: ')
5       message = "CA+" + message
6
7   encodedMessage = message.encode()
8   clientSocket.send(encodedMessage)       # sends the message to server
```

No lado do cliente, fiz com que o programa peça para que o cliente informe qual a função que ele quer que o servidor execute, e implementei uma função que

concatena qual o tipo de mensagem foi digitada pelo cliente para realizar o envio para o servidor, concatenando “CB+” no início da string quando a mensagem for caixa baixa, e “CA+” quando ela for caixa alta.

```
while 1:
    connectionSocket, clientIP = serverSocket.accept()  # server waits in .accept() to requisitions
    message = connectionSocket.recv(1500)
    decodedMessage = message.decode()
    splitMessage = decodedMessage.split('+')
    if (splitMessage[0] == "CB"):
        modifiedMessage = splitMessage[1].upper()
    else:
        modifiedMessage = splitMessage[1].lower()

    encodedMessage = modifiedMessage.encode()
    connectionSocket.send(encodedMessage)       # sends converted (upper-case) sentence
    print ("Message \"" + modifiedMessage + "\" sent successfully.")
```

Já no lado do servidor, implementei uma função onde ele utiliza a função “.split(‘+’)” para separar a string no caractere ‘+’, e testa para ver se o cabeçalho da string (parte anterior ao ‘+’) é CB ou CA, a fim de saber se a mensagem recebida está em caixa baixa e precisa ser convertida para caixa alta ou vice-versa.

Após a implementação destas novidades no código, realizei basicamente o mesmo processo de testes que nos outros exercícios e o resultado pode ser observado pelas imagens abaixo:

```
C:\Users\leo_g\Desktop\leo ufpel\oitavo semestre\Redes de Computadores\Atividade 1\ex3>python server.py
Server is on!
Message "TESTE DO EXERCICIO 3" sent successfully.
Message "de caixa alta pra caixa baixa" sent successfully.
```

```
leo_g@LAPTOP-FPBROQ1P MINGW64 ~/Desktop/leo ufpel/oitavo semestre/Redes de Computadores/Atividade 1/ex3 (main)
$ python client.py
Select server function:
1. Lower to Upper
2. Upper to Lower
1
Enter your lower-case word: teste do exercicio 3
TESTE DO EXERCICIO 3

leo_g@LAPTOP-FPBROQ1P MINGW64 ~/Desktop/leo ufpel/oitavo semestre/Redes de Computadores/Atividade 1/ex3 (main)
$ python client.py
Select server function:
1. Lower to Upper
2. Upper to Lower
2
Enter your upper-case word: DE CAIXA ALTA PRA CAIXA BAIXA
de caixa alta pra caixa baixa
```