UNIVERSIDADE FEDERAL DE RORAIMA
CENTRO DE CIÊNCIA E TECNOLOGIA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO
SISTEMAS DISTRIBUÍDOS
PROFESSOR DOUTOR LEANDRO NELINHO BALICO

LUCAS BESSA FACANHA PEREIRA
NATÁLIA RIBEIRO DE ALMADA

RELATÓRIO DE IMPLEMENTAÇÃO DE ALGORITMO CLIENTE SERVIDOR

Boa Vista
Abril de 2023

1. Introdução

Este relatório tem como objetivo elucidar acerca do desenvolvimento de um algoritmo de cliente-servidor, utilizando como base o algoritmo demonstrado durante as aulas da disciplina de Sistemas Distribuídos.

2. Implementação

2.1. Chat

Implementação de um laço de repetição While True para enviar e receber mensagem ao mesmo tempo.

```
25  while True:
26      mensagem = input().encode()
27      # realiza a criptografia da mensagem utilizando a chave publica carregada
28      mensagem_criptografada = rsa.encrypt(mensagem, publicKey)
29      cliente_socket.send(mensagem_criptografada)
```

2.2. Criptografia

Realização da importação do LIB, um arquivo python auxiliar, onde foram definidas duas funções.

```
1  import socket
2  import threading
3  from lib import *
```

A primeira função para **salvar** as chaves pública e privada da criptografia com RSA (biblioteca utilizada para criptografia) dentro de uma pasta chamada *KEYS*.

```
1  import rsa
2
3  def generateKeys():
4      (publicKey, privateKey) = rsa.newkeys(512)
5      with open('keys/publicKey.pem', 'wb') as p:
6          p.write(publicKey.save_pkcs1())
7      with open('keys/privateKey.pem', 'wb') as p:
8          p.write(privateKey.save_pkcs1())
```

E outra função implementada para ler estas chaves.

```python
def loadKeys():
    with open('keys/publicKey.pem', 'rb') as p:
        publicKey = rsa.PublicKey.load_pkcs1(p.read())
    with open('keys/privateKey.pem', 'rb') as p:
        privateKey = rsa.PrivateKey.load_pkcs1(p.read())
    return privateKey, publicKey
```

Então, sempre que for enviada uma mensagem, o código enviará a mensagem criptografada com a chave pública, e o servidor usará a chave privada para descriptografar(traduzir) a mensagem.

Para a criação das chaves pública e privada, foi utilizada da Lib o Generate key

```python
# Cria as chaves publica e privada para criptografia e as salva
generateKeys()
privateKey, publicKey = loadKeys()
print("Chaves geradas com sucesso!")
```

2.3. Threads

Sendo executado um método da funçao thread, que automaticamente cria threads e o método start as inicia. Na parte do servidor, inicia a execuçuão do código a partir do generatekeys(que gera as chaves pública e privada), carrega as chaves e inicia o server com capacidade máxima de 5 clientes simultâneos.

```python
generateKeys()
privateKey, publicKey = loadKeys()
print("Chaves geradas com sucesso!")

# inicia um servidor com capacidade de até 5 clientes
servidor_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
servidor_socket.bind((HOST, PORT))
servidor_socket.listen(5)

print(f"Servidor escutando na porta {PORT}")
print("Aperte 'q' para finalizar o programa")

# espera a tecla q ser pressionada para finalizar o programa
thread_finalizacao = threading.Thread(target=finaliza_servidor)
thread_finalizacao.start()
```

No loop While - True, o código aceita novos clientes se conectarem. Cada vez que um cliente se conhecta, é criada uma nova thread, e a thead executa a função LIDAR COM O CLIENTE.

```
while True:
    # o servidor começa a aceitar conexões
    cliente_socket, endereco = servidor_socket.accept()

    # quando uma nova conexão é realizada uma nova thread é iniciada com o intuito de receber
    # as mensagens do cliente e as repassar para todos os outros
    thread_cliente = threading.Thread(target=lidar_com_cliente, args=(cliente_socket, endereco))
    thread_cliente.start()
```

A função LIDAR COM O CLIENTE recebe mensagem e traduz essas mensagens, e transmite para os outros clientes através da função TRANSMITIR MENSAGEM (pega lista de clientes e manda a mensagem que recebeu pra todos os clientes, exceto o remetente)

```
# inicia o processo de leitura e tradução das mensagens enviadas pelos clientes
def lidar_com_cliente(cliente_socket, endereco):
    print(f"Conexão estabelecida com {endereco}")
```

```
# manda a mensagem passada para todos os clientes conectados
def transmitir_mensagem(mensagem, remetente):
    for cliente in clientes:
        if cliente != remetente:
            cliente.send(mensagem)

```

No lado Cliente, somente inicia o servidor e cria uma thread de recebimento de mensagem(lê as mensagens do servidor, traduz com o método decript e printa na tela).

```
def receber_mensagens(cliente_socket):
    while True:
        try:
            mensagem = cliente_socket.recv(1024)
            # realiza a tradução da mensagem utilizando a chave privada carregada
            mensagem_traduzida = rsa.decrypt(mensagem, privateKey).decode()
            if(mensagem_traduzida == "disconnect"):
                print("O servidor foi finalizado")
                finalizar_programa()

            print(mensagem_traduzida)
        except:
            break
```

Pede entrata de texto, criptografa usando chave publica e manda pro server, que tem o trabalho de encaminhar as mensagens.