UNIVERSIDADE FEDERAL DO CEARÁ
CIÊNCIA DA COMPUTAÇÃO
SISTEMAS DISTRIBUÍDOS

FRANCISCO DANIEL BEZERRA DE SOUZA PRACIANO - 366389
ISRAEL DE CASTRO VIDAL - 370019

- **TCP:**
  - O código desenvolvido segue as especificações definidas, isso é:
    1. Baseia-se na arquitetura Cliente-Servidor, utilizando o protocolo TCP para a comuniação entre os processos;
    2. O servidor aceita requisições utilizando uma Thread para cada cliente, para aceitar várias requisições concorrentemente;
    3. O processo servidor, após responder a uma requisição de um cliente e enviar a resposta de volta através do Socket, loga, utilizando a biblioteca logging do Python, em um arquivo a operação feita.
    4. O processo TCPServer deve ser iniciado através do terminal utilizando um comando como o seguinte:

       python3 TCPServer Porta*, onde porta é um número que indica onde o servidor aguardará novas requisições;*
    5. O processo TCPClient deve ser iniciado através do terminal utilizando um comando como o seguinte:

       python3 TCPClient ip porta num_1 num_2 op *, onde ip e porta tem significado intuitivo e num_1, num_2 e op são, respectivamente os números e a operação que deverá ser executada pelo servidor.*

    6. Por fim, o cliente exibe o resultado da operação que foi enviada pelo servidor.

  - Especificações:
    1. A classe Requisition foi criada para encapsular uma requisição que será enviada do Cliente para o Servidor. Um objeto dessa classe é serializado utilizando uma biblioteca do Python chamada Pickle. Quando o servidor recebe uma requisição, utiliza essa mesma biblioteca para deserializar o objeto novamente;
    2. O script TCPServer possui a função *execute*, que é a função que será executada concorrentemente através de Threads. Esse script abre um socket que fica aguardando(listening) requisições em uma porta arbitrária e, após receber uma requisição, abre uma conexão TCP em uma outra porta, para que possa continuar aguardando novas requisições;

3. O script TCPClient cria um socket, tenta abrir uma conexão com o servidor e, quando essa conexão é aberta, envia a requisição(um objeto da classe Requisition) serializada(através da biblioteca Pickle) para o servidor através do método socket.send(), em seguida, o cliente aguarda a resposta ser enviada pelo servidor para esse mesmo socket, através do método socket.recv(). Por último o resultado da operação é exibido no terminal.

- **UDP:**

  Foram desenvolvidos dois arquivos: UDPServer.py e UDPClient.py. Vale lembrar que os dois foram construídos seguindo a especificação dada na solicitação do trabalho. O UDPServer.py foi desenvolvido da seguinte forma:

  1) Criou-se uma classe Executor que é uma Thread e esta é responsável por gerar (inverter a string recebida) e enviar a resposta para o cliente. Para tal, a biblioteca socket do python foi utilizada, sendo o método **sendto()** utilizado para enviar as mensagens.
  2) Posteriormente, criou-se o socket responsável por ouvir as solicitações que chegam dos clientes e repassar estas para um novo Executor (Thread). Para receber essas solicitações, utilizou-se o método **recvfrom()**.

  Já para o UDPClient.py, fizemos:

  1) Criou-se o socket responsável por enviar a solicitação do cliente e também para receber a resposta do servidor. Os dois métodos citados anteriormente foram novamente utilizados aqui. No entanto, o método **setblocking()** se fez necessário porque o cliente não necessariamente irá receber a resposta do servidor, então o cliente não pode ficar bloqueado somente até chegar uma mensagem. Ao invés disso, o cliente fica bloqueado até receber a mensagem ou estourar o tempo de timeout. Quando ocorre o timeout, que foi definido em 0.7 segundo, é gerado uma exceção timeout que é tratada com um print na tela avisando do ocorrido. Caso contrário, a resposta recebida do servidor é mostrada na tela.