Instituto Federal de Educação Ciência e Tecnologia do RN
Diretoria Acadêmica de Tecnologia da Informação
Curso de Tecnologia em Redes de Computadores
Disciplina de Programação para Redes
Professor Galileu Batista

Lista de Exercícios Avaliativa sobre Sockets UDP e TCP.

Desenvolva os programas solicitados nas questões respeitando as seguintes regras:

a) use a linguagem de programação Python (versão 3.X);

b) **além de `socket`, não empregue bibliotecas ou funções prontas do Python**, exceto quando o enunciado da questão definir essa possibilidade;

c) **Comente o código;**

d) **Trate os erros de entrada inválida;**

e) desenvolva a atividade com até mais um colega; **apenas um dos membros da dupla entrega a atividade**; se mais de um entregar apenas o primeiro será considerado na avaliação;

Para a entrega siga as seguintes regras:

a) Crie um repositório no www.github.com com o nome `ProgRedes-2024.2`

b) Dentro do repositório deverá ter uma pasta com o nome `Unidade01-Avaliacao02` com os programas que respondem às questões;

c) Dentro da pasta crie os subpastas para cada uma das questões com os nomes: `Q1`, `Q2` e assim sucessivamente; o conteúdo a ser colocado em cada pasta depende da questão e será especificado no respectivo enunciado;

d) A resposta a essa atividade deve um pequeno texto contendo apenas:

- Os nomes, matrículas e usuário github dos membros grupo;
- O nome do usuário github que está entregando a atividade.

A avaliação será feita tendo por base o nome do usuário, com o nome do repositório e pasta acima citados.

e) A correção será realizada com o último *commit* anterior à data de entrega. Também pode ser objeto de avaliação a frequência em que cada membro do grupo fez *commits*;

Qualquer infração às regras acima implica na não correção da atividade e/ou questão, com a nota sendo zerada para o item/atividade.

Instruções gerais para essa atividade:

Conforme discutido em sala, é possível fazer dois pequenos programas que funcionam como cliente e servidor de arquivos UDP (por mais estranho que isso pareça).

Versões iniciais dos programas estão em apêndice. Nessas versões o cliente apresenta um problema: fica travado esperando dados do servidor quando este já terminou de enviar

todo o arquivo solicitado, mas o cliente não sabe. Existem, pelo menos, duas soluções básicas para esse problema:

- Programar um *timeout* no lado do cliente (usando o método **settimeout** da API Socket do Python). Assim, se **recvfrom** demorar mais do que o tempo de espera estabelecido, o cliente entende que não mais receberá dados e pode prosseguir;
- Programar o envio do tamanho do arquivo pelo servidor imediatamente antes de enviar os próprios dados. O cliente deve receber esse tamanho e parar de esperar por dados do servidor quando já recebeu dados suficientes.

Em todas as versões dos programas seguintes, tanto do lado do cliente quanto do lado do cliente quanto do servidor, use a pasta **files** como repositório dos arquivos. O usuário não deve mencionar esse nome quando solicitando operações sobre os arquivos, somente o nome destes.

1. (2 pontos) Implemente o uso do envio do tamanho do arquivo do servidor para o cliente e teste a solução. Trate eventuais exceções. Apresente na pasta de resposta a essa questão:
    - a pasta **`server`** com o arquivo **`udp-file-server-size.py`** e uma subpasta **`files`** com alguns arquivos para servir;
    - a pasta **`client`** com o arquivo **`udp-file-client-size.py`** e uma subpasta **`files`** vazia, para recebimento dos arquivos;
    - O arquivo **`Q1.pdf`**, que deve conter a explicação de como o envio/recebimento do tamanho foi implementado.

2. (1 ponto) Explique porque tanto os programas que usam *timeout* quanto tamanho (resposta à questão 1), mesmos sem apresentar erros aparentes não transferem arquivos grandes como deveriam. Gere um pequeno arquivo **Q2.pdf** na pasta de resposta a essa questão. Essa resposta deve ser detalhada (pelo menos 30 linhas e eventualmente figuras), sob pena de ser ignorada. O texto será submetido à apreciação de ferramentas que verificam a criação por meio de IA, situação em que será ignorada.

3. (7 pontos) Implemente o mesmo servidor em TCP. Mas agora trate a seguintes funcionalidades:
    a. O usuário do cliente pode solicitar a listagem dos arquivos no servidor (**list**). A resposta deve incluir o nome do arquivo e o tamanho de cada arquivo. Lembre que arquivos podem ter espaços no nome e que diretório podem ter milhares de arquivos;
    b. O usuário pode solicitar o *download* de um arquivo (**sget**); trate os casos em que o arquivo não existe, dando uma resposta adequada ao usuário;
    c. O usuário poderá solicitar *download* mais de um arquivo (**mget**), do servidor mediante o uso de máscaras (exemplo: `*.jpg`). Cada arquivo obtido no servidor deve ser salvo no cliente com o mesmo nome que está no servidor e se o arquivo já existe no lado cliente, pergunte ao usuário se quer sobrescrever. Dica: no servidor use a função `glob.glob()` para obter a lista de arquivos que atendem a uma máscara;

d. Implemente a funcionalidade de obter o **hash** SHA1 de um arquivo no servidor até uma posição específica. Ou seja, o usuário do cliente pode pedir para calcular o *hash* do arquivo `x.txt` no servidor até a posição 500;

e. Implemente a funcionalidade de continuar o *download* de um arquivo (**cget**) já presente no lado cliente. Antes de fazer o pedido ao servidor, confirme, por meio do hash de ambos os lados de a parte presente no cliente corresponde ao que está no servidor;

f. Em todas as operações, o servidor deve limitar-se aos arquivos na pasta `files`, que armazena os arquivos a servir. Em outras palavras, se o cliente solicitar o arquivo `..\x.txt`, ele não deve ser servido, pois não está uma pasta antes da pasta de **`files`**. Dica: use `os.path.realpath()` para eliminar os "`..`" no caminho de um arquivo;

Apresente na pasta de resposta a essa questão:

- a pasta **`server`** com o arquivo **`tcp-file-server.py`** e uma subpasta **`files`** com alguns arquivos para servir;
- a pasta **`client`** com o arquivo **`tcp-file-client.py`** e uma subpasta **`files`** vazia, para recebimento dos arquivos;
- O arquivo **`Q3.pdf`**, que deve conter a explicação de, pelo menos, a forma como os itens *a* até *f* foram implementados, incluindo o protocolo entre cliente e servidor.

Boa sorte.