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 de Computadores
Prof: Galileu Batista

Um fileserver baseado em TCP

Conforme discutido em sala, é possível fazer dois pequenos programas que funcionam como cliente e servidor de arquivos TCP. A função primária é que o cliente possa baixar arquivos do servidor, mas algumas outras operações correlatadas são suportadas.

Sua atividade consiste em desenvolver:

- Um programa servidor de arquivos, escutando em todos os IPs da sua máquina, na porta UDP 20000;
- Um programa cliente do servidor de arquivos anterior.

O cliente vai interagir um usuário, que solicitará alguma das operações disponíveis. O cliente vai solicitar esta operação junto ao servidor e realizar as operações associadas.

O protocolo de comunicação para as operações é:

Usuário	Cliente (envia)	Servidor (envia)

Solicita o *download* de um arquivo, informando o nome.	▪ 1 *byte* de operação (em binário);
	  ▪ Valor: 10
	▪ 4 bytes (em binário – big *endian*) com o tamanho do nome do arquivo;
	▪ Múltiplos *bytes* contendo o nome do arquivo. A quantidade corresponde ao tamanho informado antes.
		▪ 1 *byte* de status (em binário);
		  ▪ Valor: 0 (arquivo existe e será enviado a seguir);
		  ▪ Valor: 1 (arquivo não existe e nada mais será enviado);
		▪ 4 bytes (em binário – big *endian*) com o tamanho do arquivo que será enviado;
		▪ Múltiplos blocos de 1024 *bytes* contendo os dados do arquivo, até que todos os bytes (até o tamanho informado antes) sejam enviados.

Solicita a listagem dos arquivos disponíveis no servidor	▪ 1 *byte* de operação (em binário);
	  ▪ Valor: 20
		▪ 1 *byte* de status (em binário);
		  ▪ Valor: 0 (a listagem será enviada a seguir);
		  ▪ Valor: 1 (erro não identificado, nada mais será enviado);
		▪ 4 bytes (em binário – big *endian*) com o tamanho a seguir;
		▪ A listagem em formato JSON, com o seguinte formato:
		[{ ‘nome’: ‘arquivo1.txt’,
		‘tamanho’: ‘12345’ },
		{ ‘nome’: ‘arquivo2.jpg’,
		‘tamanho’: ‘45678’ },
		….]

<table>
<tr><td rowspan="2">Solicita o upload de um arquivo para o servidor, informando o nome.</td><td>▪ 1 byte de operação (em binário);
▪ Valor: 30
▪ 4 bytes (em binário – big endian) com o tamanho do nome do arquivo;
▪ Múltiplos bytes contendo o nome do arquivo. A quantidade corresponde ao tamanho informado antes.</td><td>▪ 1 byte de status (em binário);
▪ Valor: 0 (o upload foi aceito, os bytes serão esperados);
▪ Valor: 1 (erro não identificado, os bytes não serão esperados);</td></tr>
<tr><td>Só enviar se o servidor aceitou o upload.
▪ 4 bytes (em binário – big endian) com o tamanho do arquivo;
▪ Múltiplos blocos de 1024 bytes contendo os dados do arquivo, até que todo os bytes (até o tamanho informado antes) sejam enviados.</td><td>▪ 1 byte de status (em binário);
▪ Valor: 0 (o upload foi bem-sucedido);
▪ Valor: 1 (erro não identificado);</td></tr>
<tr><td rowspan="2">Solicita o download de arquivo a partir de determinada posição, informando o nome, a posição a partir de onde deseja e o MD5 da parte que possui.</td><td rowspan="2">▪ 1 byte de operação (em binário);
▪ Valor: 40
▪ 4 bytes (em binário – big endian) com o tamanho do nome do arquivo;
▪ Múltiplos bytes contendo o nome do arquivo. A quantidade corresponde ao tamanho informado antes;
▪ 4 bytes (em binário – big endian) contendo a posição inicial a partir de onde quer baixar o arquivo;
▪ 16 bytes (em binário – big endian) contendo o hash MD5 da parte que possui.</td><td>▪ 1 byte de status (em binário);
▪ Valor: 0 (o download foi aceito, os bytes serão esperados);
▪ Valor: 1 (erro não identificado, os bytes não serão esperados);</td></tr>
<tr><td>▪ 1 byte de status (em binário);
▪ Valor: 0 (o upload foi bem-sucedido);
▪ Valor: 10 (o arquivo não existe, nada mais será enviado);
▪ Valor: 20 (o hash não confere, nada mais será enviado);
▪ 4 bytes (em binário – big endian) com o tamanho remanescente do arquivo que será enviado;
▪ Múltiplos blocos de 1024 bytes contendo os dados do arquivo, até que todos os bytes (até o tamanho informado antes) sejam enviados.</td></tr>
<tr><td>Solicita o download de múltiplos arquivos, informando uma máscara. Ex: *.jpg</td><td>▪ 1 byte de operação (em binário);
▪ Valor: 50
▪ 4 bytes (em binário – big endian) com o tamanho do da máscara a enviar;
▪ Múltiplos bytes contendo o nome do arquivo. A quantidade corresponde ao tamanho informado antes;</td><td>▪ Faça uma proposta de como será a resposta do servidor.</td></tr>
</table>

Considere, em adição, os seguintes aspectos:

- Defina uma pasta no cliente e no servidor em que os arquivos serão armazenados; todos os pedidos respostas serão relativos a essas pastas. Não permita escape dessas pastas, ou seja, arquivos fora delas jamais deverão atendidos;
- Trata erros de conexão e congêneres;
- Pada operação se dá em uma conexão;
- Estabeleça um timeout para as operações;
- O cliente deve receber na linha de comando o nome (ou ip) do servidor e a porta à qual conectar, no formato ip:porta. Ex:

  ```
  python cliente.py 10.25.2.120:20000
  ```
- O servidor deve escutar em todas os IPs e receber na linha de comando em que porta. Ex:

  ```
  python servidor.py 20000
  ```
- O servidor deve informar os IPs e as portas em que está escutando;
- O servidor deve ter suporte a múltiplos *threads*, ou seja, poderá atender múltiplos clientes concorrentemente.

Não serão permitidos o uso de quaisquer bibliotecas prontas que realizem atividades similares ao que que se solicita. Portanto, limite-se às bibliotecas: **`socket`**, **`os`**, **`json`**, **`time`**. NÃO use **`requests`**, por exemplo.

A estrutura de pastas a colocar os arquivos já está criada no exercício.