Trabalho prático - Programação de sockets

Carlos Araújo

2023.2

1 Funções

1.1 Servidor

- **comando**

  Essa função é responsável por processar os comandos enviados pelos clientes. Ela recebe um objeto de soquete (client_socket) como argumento e, em seguida, entra em um loop para receber dados do cliente e processar os comandos.

- **comando_individual**

  Esta função é chamada pela função comando para processar comandos individuais. Ela recebe um comando como entrada e executa uma ação com base no comando. Os comandos possíveis incluem: CONSULTA, HORA, ARQUIVO, LISTAR e SAIR.

1.2 Cliente

- **enviar_comando**

  Esta função é responsável por enviar comandos para o servidor e receber respostas. Ela recebe um comando como argumento, cria um socket cliente, tenta estabelecer uma conexão com o servidor usando

o endereço e a porta especificados, envia o comando codificado em bytes, recebe a resposta do servidor (até 1024 bytes) e a decodifica para uma string. Em seguida, a função retorna a resposta.

- **menu_comandos**

  Esta função exibe um menu de opções de comandos para o usuário e solicita que o usuário escolha uma opção. Ela retorna a escolha do usuário como uma string.

2 Bibliotecas

- **socket**

  Usei para criar o socket do servidor e também os sockets dos clientes quando uma conexão é aceita.

- **os**

  Usei para listar os arquivos no diretório atual (os.listdir()) quando o comando LISTAR é escolhido. Também é usada para abrir e ler o conteúdo de arquivos especificados quando o comando ARQUIVO é escolhido.

- **threading**

  Usei para criar uma nova thread sempre que uma nova conexão de cliente é aceita, com isso, o servidor consegue fazer múltiplas conexões de clientes de forma concorrente, processando os comandos de cada cliente em uma thread separada.

- **datetime**

  Usei para pegar o horário atual quando o comando HORA fosse escolhido.

- **shutil**

  Usei para fazer uma copia de um arquivo no servidor para o endereço do cliente.

3 Funções do python

- **getpeername**

  Usada para obter informações como endereço IP e número da porta do cliente conectado ao servidor.

- **recv**

  Usada para receber os dados enviados pelo cliente para o servidor.

- **decode**

  Usado para decodificar os dados recebidos. Como os dados são recebidos em formato de bytes, a função converte esses bytes para formato de string.

- **send**

  Usada para enviar a resposta do servidor de volta para o cliente.

- **encode**

  Usada para codificar a resposta do servidor antes de enviá-la para o cliente. A resposta digitada pelo usuário é feita em formato de string e precisa chegar ao servidor em formato de bytes.

- **datetime.now**

  Usei para pegar a hora atual.

- **strftime**

Usei para formatar a saída do horário com o formato pedido no trabalho.

- **startswith**

  Usei para verificar se o comando estava relacionado a um arquivo.

- **bind**

  Usada para associar o endereço IP e a porta definida ao soquete do servidor.

- **listen**

  Usada no soquete do servidor após a associação com a função **bind**. Ela define o número máximo de conexões pendentes que o servidor pode tratar simultaneamente.

- **accept**

  Usada para aceitar uma nova conexão de cliente quando ela chega.

- **start**

  Usada para iniciar a execução da thread.

- **connect**

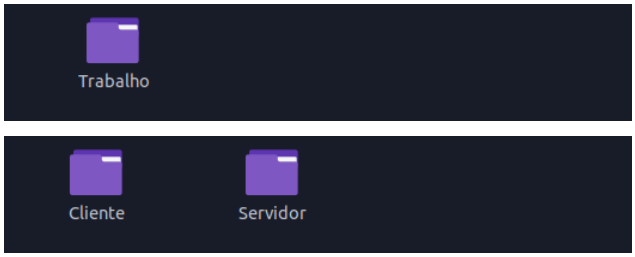
  Usada no socket do cliente para estabelecer uma conexão com o servidor.

- **sendall**

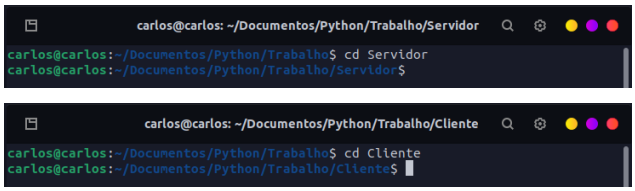
  Usada para enviar um comando codificado em bytes para o servidor. A função encode converte os dados para bytes e a função sendall envia.

4 Instruções

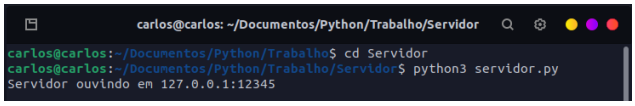
- Primeiro você deve entrar na pasta **Trabalho**



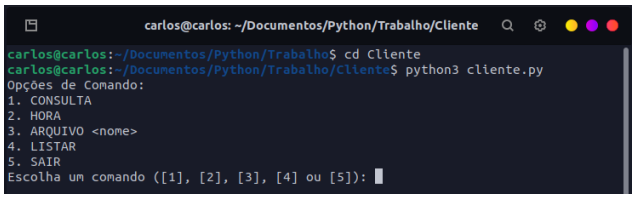
- Em seguida deve abrir um terminal para a pasta **Servidor** e outro terminal para a pasta **Cliente**



- Executar primeiro o arquivo do servidor em um terminal



- Em seguida, executar o arquivo do cliente em outro terminal



- São disponibilizadas cinco opções de consultas para o cliente, escolha a que preferir.

5 Dificuldades

- Não foram encontrados muitos problemas na implementação. Foi relativamente simples de usar a biblioteca socket do python. A maior

dificuldade foi lidar com múltiplas conexões de clientes de forma concorrente, mas encontrei alguns exemplos com o uso de threads que resolvem esse problema.