SISTEMAS DISTRIBUÍDOS: LAB 3 - 06/08/2021, UFRJ

Nome: Breno Trindade Tostes; DRE: 117099396

1ª Atividade:

Objetivo: Reprojetar a implementação do servidor.

1. Altere o projeto de implementação do servidor para que ele **seja capaz de receber comandos básicos da entrada padrão** (inclua no mínimo um comando para permitir finalizar o servidor quando não houver clientes ativos). **Use a função *select*.**
2. Altere o projeto de implementação do servidor para que ele **se torne um servidor concorrente**, isto é, trate **cada nova conexão de cliente** como um novo fluxo de execução e atenda as requisições desse cliente dentro do novo fluxo de execução.

Crie threads ou processos filhos.

A interação com os clientes (incluindo o formato das mensagens trocadas) não deve ser alterada.

Projeto da atividade anterior modificado:

Proposta de arquitetura de sistema:

1. *Lado cliente*: implementa a camada de interface com o usuário. O usuário poderá solicitar o processamento de uma ou mais buscas em uma única execução da aplicação: o programa espera pelo nome do arquivo e da palavra de busca, faz o processamento, retorna o resultado, e então aguarda um novo pedido de arquivo e palavra ou o comando de finalização.
2. *Lado servidor*: implementa a camada de processamento e a camada de acesso aos dados. Projete um servidor iterativo, isto é, que trata as requisições de um cliente de cada vez, em um único fluxo de execução (estudaremos essa classificação depois). Terminada a interação com um cliente, ele poderá voltar a esperar por nova conexão. Dessa forma, o programa do servidor fica em loop infinito (depois veremos como lidar com isso).

NOVIDADES:

1. Agora o servidor poderá receber os comandos de forma não bloqueante devido a multiplexação com select:
   - ➢ $exit - Bloqueia o recebimento de novas conexões e encerra o servidor quando as conexões ativas finalizarem;
   - ➢ $list connections (ou apenas $lc) - Exibe a lista de endereços de clientes conectados;
   - ➢ $list threads (ou apenas $lt) - Exibe a lista de threads ativas;
   - ➢ $help - Exibe a lista de comandos disponíveis e suas funcionalidades.

```python
while True:
    r, w, x = s.select(entry_points, [], [])

    for ready in r:
        if ready == sckt:
            # Aceita a conexao
            client_sckt, client_addr = acceptConnection(sckt)

            # Cria a nova thread que ira lidar com a conexao
            client = threading.Thread(target=requestHandler, args=(client_sckt, client_addr))
            client.start()

            # Adiciona a nova thread na lista de threads ativas
            client_threads.append(client)

            # Protecao contra alteracoes problematicas por multithreading
            lock.acquire()
            connections[client_sckt] = client_addr  # Adiciona a conexao nova ao mapa de conexoes
            lock.release()
        elif ready == sys.stdin:
            # Permite interacao com o servidor
            cmd = input()
            internalCommandHandler(cmd, sckt, client_threads)
```

Figura 1: Trecho do código responsável pela multiplexação de entradas com o select.

2. O servidor utiliza multithreading para lidar com as requisições dos clientes de forma simultânea (como é mostrado na figura 1, acima). Sempre que uma nova conexão for aceita pelo servidor, uma thread será designada para lidar com a requisição através da função *requestHandler( )* .

Devido ao aspecto multithread do código, é necessário que a estrutura (dicionário) responsável por armazenar as conexões e seus respectivos endereços seja protegida com um thread lock tanto para inserções quanto remoções (figuras 1 e 2) - a estrutura é usada pelo comando *$lc*.

```python
def file_unavailable(newSckt, address):
    """
    Modularizacao do codigo que lida com encerramento da conexao
    """
    print(f'{str(address)}> Encerrou a conexao')

    # Protecao contra eventuais alteracoes na estrutura por outras threads
    lock.acquire()
    del connections[newSckt]
    lock.release()

    # Encerra a conexao
    newSckt.close()
    pass
```

Figura 2: Trecho do código com a proteção via thread lock para a operação de deleção no dicionário .