A:

SERVIDOR:

```
if comando == 1:  # Pedido para listar arquivos
    print('Listando arquivos')

    lista_arquivos = os.listdir(DIRBASE)
    lista_arquivos = [f"{arq} ({os.path.getsize(DIRBASE + arq)} bytes)" for arq in lista_arquivos]

    lista_junta = '\n'.join(lista_arquivos).encode('utf-8')
    con.send(len(lista_junta).to_bytes(4, 'big'))
    con.send(lista_junta)
```

Recebe o comando informando que o cliente quer uma listagem de arquivos, com a função **os.listdir(DIRBASE)** será listado todos os arquivos dentro da pasta 'files'.
Para cada arquivo listado se obtem o tamanho usando **os.path.getsize,** Em seguida enviado ao cliente o tamanho e o nome.

CLIENTE:

```
def listagem():
    sock.send((1).to_bytes(2,'big')) #envia um codigo para solicitar a listagem
    print('Solicitando a listagem de arquivos...\n')
    tamanho = int.from_bytes(sock.recv(4), 'big')
    dados = b''

    while len(dados) < tamanho:
        dados += sock.recv(4096)

    print("Arquivos disponíveis:")
    print(dados.decode('utf-8'))
```

.

Além de enviar para o servidor a solicitação do cliente a função recebe o tamanho e o nome dos arquivos e exibe para o usuário

Protocolo:

```
# Criação do socket e conexão com o servidor
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

O protocolo usado é TCP e IPV4 em ambos os lados

B e C :

SERVIDOR:

```
elif comando == 2:  # Pedido de download de arquivos
    mensagem = con.recv(2)  # Recebe 2 bytes do tamanho
    mensagem = int.from_bytes(mensagem, 'big')  # Converte para

    fileName = con.recv(mensagem).decode('utf-8')  # Próximos by
    print("Recebi pedido para o arquivo ", fileName)
```

recebe o tamanho da string do nome e o nome ou mascara enviado pelo servidor.

```python
files = glob.glob(DIRBASE + fileName)
print(f"Arquivos encontrados: {files}")

if files:
    # Envia lista de arquivos encontrados
    con.send(b'\x00\x00')  # Sucesso
    con.send(len(files).to_bytes(4, 'big'))  # Envia o número de arquivos encontrados

    # Envia os nomes dos arquivos para o cliente
    for file in files:
        file_name = os.path.basename(file)  # Filtra apenas o nome do arquivo
        len_a = len(file_name.encode('utf-8')).to_bytes(2, 'big')
        msg = len_a + file_name.encode()  # Concatena o tamanho e o nome
        con.send(msg)

        print(f"nome do arquivo: {DIRBASE + file_name}")

        file_size = os.path.getsize(DIRBASE + file_name)  # Tamanho em bytes
        con.send(file_size.to_bytes(4, 'big'))  # Envia o tamanho do arquivo
        print(f"tamanho no server é : {file_size}")

        with open(file, 'rb') as f:
            print(f"Enviando arquivo {file}")
            file_data = f.read(4096)
            while file_data != b'':
                con.send(file_data)
                file_data = f.read(4096)
else:
    print("Nenhum arquivo encontrado com a máscara fornecida.")
    con.send(b'\x00\x01')  # Nenhum arquivo encontrado
```

usa **glob.glob()** para obter a lista de arquivos, se o arquivo existir é enviado um byte 0 e o numero de arquivos existentes dentro da pasta files.
Em seguida um **for** para obter o nome,tamanho do nome e tamanho em bytes de cada arquivo e enviar para o cliente. Após isso enviar o arquivo

CLIENTE:

```python
arq = input('Digite o nome do arquivo: ')

# Converte o nome do arquivo para bytes e envia o comprimento e o n
lenNameArq = len(arq.encode('utf-8')).to_bytes(2, 'big') #2 bytes d
msg = lenNameArq + arq.encode()  # Concatena o com  primento e o no
sock.send(msg)
```

Pega o tamanho da string do nome do arquivo e o nome do arquivo e envia parar o servidor

---

```python
fileIsOk = int.from_bytes(sock.recv(2), 'big')

if fileIsOk == 0:
    tam = int.from_bytes(sock.recv(4), 'big') #recebe 4 bytes do tamanho do arq e converte para int
    print(f"O número de arquivos encotrado é: {tam}")
    #tratamento para caso o arquivo já exista no cliente
    for c in range(tam):
        tamanho_nome = sock.recv(2)
        tamanho_nome=  int.from_bytes(tamanho_nome, 'big')
        fileName = sock.recv(tamanho_nome).decode('utf-8')

        bytes_arq = int.from_bytes(sock.recv(4),'big') #tamanho em bytes do arquivo
        print(f"arquivo: {fileName}")

        if os.path.exists(DIRBASE + fileName):
            resposta = input(f"Arquivo {fileName} já existe Deseja subistituir? [S/N] ").lower()

            while resposta != 's' and resposta != 'n':
                print("Resposta inválida tente novamente usando 's' ou 'n'. ")
                resposta = input("Deseja subistituir? [S/N] ").lower()
            if resposta == 's':
                None
            elif resposta == 'n':
                print("Download cancelado ")
                return

    with open(DIRBASE + fileName, 'wb') as fd:
        recebido = 0
        while recebido < bytes_arq:
            recBytes = sock.recv(min(4096, bytes_arq - recebido)) #pega o tam
            fd.write(recBytes)
            recebido += len(recBytes)
    print("Arquivo recebido com sucesso! ")
```

ao receber um byte 0(confirmação que o arquivo existe, o cliente receberá o tamanho do nome,nome e tamanho em bytes e será feito uma verificação para saber se o arquivo já existe ou não, e o arquivo será “escrito”

D:
HASH
Cliente:

```python
def calcular_hash():
    try:
        sock.send((3).to_bytes(2, 'big'))  # Solicita o cálculo do hash

        # Solicita o nome do arquivo e a posição
        arq = input('Digite o nome do arquivo: ')
        posicao = int(input('Digite a posição até onde calcular o hash: '))

        # Converte o nome do arquivo e a posição para bytes e envia
        lenNameArq = len(arq.encode('utf-8')).to_bytes(2, 'big')
        msg = lenNameArq + arq.encode() + posicao.to_bytes(4, 'big')  # Concatena o nome do arquivo e a posição
        sock.send(msg)

        # Recebe a resposta do hash calculado
        hash_recebido = sock.recv(1024).decode('utf-8')
        print(f"Hash SHA1 até a posição especificada: {hash_recebido}")
```

A função **calcular_hash** solicita ao usuário o nome de um arquivo e a posição até onde calcular o hash. Envia essas informações, junto com um pedido para calcular o hash, para o servidor. Depois, recebe e exibe o hash SHA1 calculado até a posição especificada. O nome do arquivo e a posição são convertidos em bytes antes de serem enviados ao servidor, que retorna o hash, que é então impresso na tela.

Servidor

```python
def calcular_sha1_ate_posicao(caminho_arquivo, posicao):
    # Função que calcula o hash SHA1 até a posição especificada
    sha1 = hashlib.sha1()
    try:
        with open(caminho_arquivo, 'rb') as f:
            dados = f.read(posicao)  # Lê o arquivo até a posição especificada
            sha1.update(dados)  # Atualiza o hash com os dados lidos
        return sha1.hexdigest()

    except FileNotFoundError:
        return None  # Arquivo não encontrado
```

A função **calcular_sha1_ate_posicao** calcula o hash SHA1 de um arquivo até uma posição específica. Ela lê os dados do arquivo até a posição indicada, atualiza o hash com esses dados e retorna o valor do hash em formato hexadecimal. Se o arquivo não for encontrado, retorna None.

```python
elif comando == 3:  # Pedido de calcular o hash SHA1 (posição específica)

    # Recebe o nome do arquivo e a posição
    mensagem = con.recv(2)
    mensagem = int.from_bytes(mensagem, 'big')

    fileName = con.recv(mensagem).decode('utf-8')  # Recebe o nome do arquivo
    posicao = int.from_bytes(con.recv(4), 'big')  # Recebe a posição até onde calcular o hash

    print(f"Calculando hash SHA1 do arquivo {fileName} até a posição {posicao}...")

    caminho_arquivo = os.path.join(DIRBASE, fileName)
    hash_resultado = calcular_sha1_ate_posicao(caminho_arquivo, posicao)

    if hash_resultado:
        con.send(hash_resultado.encode('utf-8'))  # Envia o hash para o cliente
        print(f"Hash SHA1 até a posição {posicao}: {hash_resultado}")
    else:
        con.send("Arquivo não encontrado.")  # Envia erro se o arquivo não for encontrado

    con.close()
```

O servidor recebe o tamanho e nome do arquivo, junto com a posição até onde calcular o hash. O caminho completo do arquivo é formado utilizando o diretório base (DIRBASE) combinado com

o nome do arquivo. A função calcular_sha1_ate_posicao é chamada para calcular o hash, e o resultado (ou uma mensagem de erro, se o arquivo não for encontrado) é enviado ao cliente

F:
SERVIDOR:

```
caminho_dir = os.path.realpath(DIRBASE)  # Caminho inteiro até "files"

def pasta_valida(pasta_solicitada):
    solicitacao = os.path.join(caminho_dir, pasta_solicitada)
    caminho_usuario = os.path.realpath(solicitacao)  # Caminho inteiro

    comparação = os.path.commonpath([caminho_dir, caminho_usuario])  #
    return comparação

pasta_valida(fileName)
if pasta_valida(fileName) == caminho_dir: # se
    con.send(b'\x00\x02')
else:
    con.send(b'\x00\x03')
```

Usa **os.path.realpath(DIRBASE)** para obter o caminho inteiro até files e na função **pasta_valida** verifica se o arquivo solicitado pelo cliente esta dentro da pasta “files” caso não esteja rejeita a solicitação e envia um byte 3, se estiver envia um byte 2

CLIENTE:

```
pasta_valida = int.from_bytes(sock.recv(2), 'big')
if pasta_valida == 3:
    print("Acesso negado, você não pode acessar arquivos fora da pasta files.")
    return
```

verifica se o byte recebido foi 3, caso seja informa ao cliente.