

# Implementação dos programas Servidor e Cliente TCP

## 1.1- Configurações iniciais do Servidor:

Figura 01 - Configuração inicial do servidor.

```
import socket
import os
import glob
import hashlib
import time

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
ip = "127.0.0.1"
porta = 1234
server_address = (ip, porta)
DIRETORIO = 'files/'

# Vincula o servidor ao endereço e porta
server.bind(server_address)
server.listen(5) # Configura o servidor para aceitar conexões

print(f'Servidor escutando em.... {ip}:{porta}')
```

Fonte: Autoria Própria, 2025

Configurações básicas para conseguir se conectar ao servidor e cliente.

## 1.2 Funções criadas no Servidor:

Figura 02 - Função calcular hash..

```
# Função para calcular o hash, apresentada nos comandos hash e eget
def calcular_hash(file_path, tamanho=None):
    with open(file_path, 'rb') as f:
        if tamanho:
            return hashlib.sha1(f.read(tamanho)).hexdigest()
        return hashlib.sha1(f.read()).hexdigest()
```

Fonte: Autoria Própria, 2025.

A função `calcular_hash` como o nome indica, é utilizada para calcular o hash SHA1 dos arquivos contidos no servidor e localmente no cliente, ele é utilizado nos comandos “hash” e “cget”.

Figura 03 - Função de verificar caminho.

```
# Função de segurança para verificar se o arquivo está dentro do diretório

def in_directory(base_directory, file_path):
    real_path = os.path.realpath(file_path) # Elimina qualquer arquivo com caminho contendo '..'
    if '..' in real_path.split(os.sep):
        client_socket(f"ERRO! O arquivo {file_path} contém '..', o que é proibido por questões de segurança.".encode('utf-8'))
        return False
    return real_path.startswith(os.path.realpath(base_directory)) # Verifica se o caminho real do arquivo está dentro do diretório base
```

Fonte: Autoria Própria, 2025

A função `In_directory()` serve para verificar se o caminho do arquivo solicitado pelo cliente está dentro de seu diretório base. Essa função, a partir do método `file_path` também elimina os “..” caso sejam enviados juntos da solicitação. Dessa forma, arquivos que estão antes da pasta files não serão servidos ao cliente, por questões de segurança.

### 1.3 Funções criadas no cliente:

Figura 04 - Função de pedir arquivo.

```
def pedir_arquivo(arquivo, tamArq):
    if os.path.exists(DIRETORIO + arquivo):
        resp = input(f"O arquivo '{arquivo}' já existe, deseja sobrescrever? S ou N: ")
        if resp != "s" and resp != "S":
            print("Operação cancelada.")
            return
        else:
            print(f"Substituindo o arquivo '{arquivo}'...")

    if tamArq > 0:
        print(f"Salvando o arquivo '{arquivo}' localmente...")
        with open(DIRETORIO + arquivo, "wb") as fd:
            recebido = 0
            while recebido < tamArq:
                data = tcpSock.recv(4096)
                fd.write(data)
                print("Lidos:", recebido, "bytes")
                recebido += len(data)
            print(f"O arquivo '{arquivo}' foi recebido com sucesso.\n")
            time.sleep(2)
            print("Voltando ao menu principal...\n")
    else:
        print("Erro: Arquivo não encontrado.")
```

Fonte: Autoria Própria, 2025.

A função `pedir_arquivo()` utilizado no comando “sget” serve para verificar se o arquivo existe na pasta files do client e se quer substituir para recebê-lo. utilizando `if os.path.exists(DIRETÓRIO+arquivo, "wb") as fd`, cria a condição de substituir o arquivo. A lógica de receber o arquivo funciona quando o servidor envia o nome e o tamanho do arquivo em bytes, assim que recebe, entra em um loop que vai ficar pedindo o

bytes do arquivo enquanto ele ser menor que o tamanho anteriormente recebido. Quando finaliza o recebimento, espera dois segundos e volta pro menu principal.

## 2 Funcionalidades (Comandos):

### 2.1 - Comando List

Cliente:

Figura 05 - Comando list cliente.

```
elif nomeArq == "list":
    pedido = "1"
    print("Enviando pedido de listagem dos arquivos.\n")
    tcpSock.send(pedido.encode('utf-8'))
    dataTam = tcpSock.recv(2048)
    resposta = str(dataTam.decode('utf-8'))
    print(f"{resposta}\n")
    time.sleep(5)
    print("Voltando ao menu principal...\n")
    time.sleep(2)
```

Fonte: Autoria Própria, 2025.

Caso o usuário solicite ao servidor usando o comando **list**, o cliente solicitará a listagem dos arquivos presentes na pasta files do servidor. O pedido é enviado ao servidor pelo código representado pelo **"1"**. E o servidor responde com a lista de arquivos que o cliente printa na tela, espera um pouco e volta pro menu.

Servidor:

Figura 06 - Comando list servidor.

```
# Caso o cliente solicite a listagem de arquivos (list) - Comando 1
if filename == "1":
    print(f'Recebida solicitação de listagem de arquivos do cliente {client_address}')
    list_arq = os.listdir(DIRETORIO)
    listagem = []
    for file in list_arq:
        file_path = os.path.join(DIRETORIO, file)
        if os.path.isfile(file_path):
            file_size = os.path.getsize(file_path)
            listagem.append(f"{file} ({file_size} bytes)")
    if listagem:
        client_socket.sendall("\n".join(listagem).encode())
    else:
        client_socket.sendall(b'Nenhum arquivo encontrado no servidor.')
```

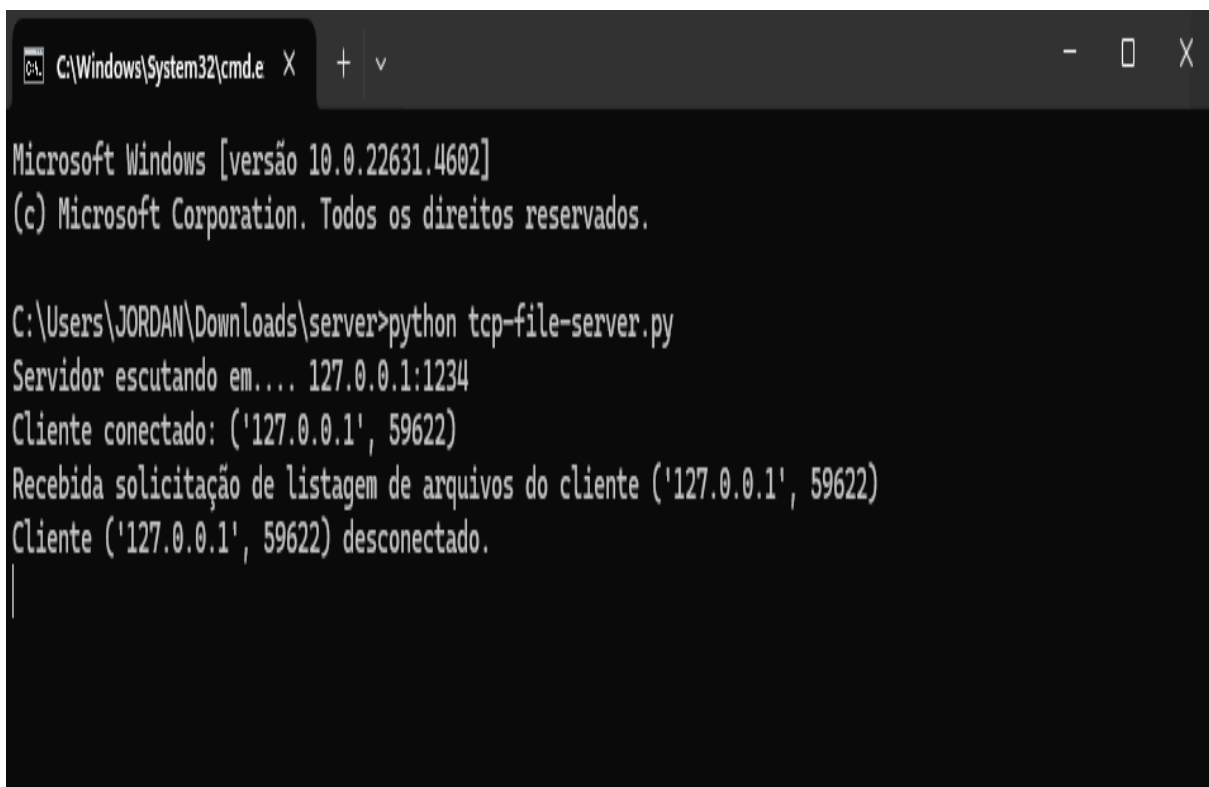
Fonte: Autoria Própria, 2025.

Quando o servidor recebe do cliente o código “1”, ele irá listar todos os arquivos da pasta e enviá-los ao cliente, a partir dos seguintes passos:

- 1- Lista os arquivos presentes na pasta files usando `os.listdir(DIRETORIO)` armazenando a listagem em `list_arq`;
- 2- É criada a lista `listagem` a fim de armazenar o nome e o tamanho de cada arquivo presente em files;
- 3- O tamanho do arquivo é capturado por meio da função `file_size = os.path.getsize(file_path)`;
- 4- O caminho para encontrar os arquivos está armazenado na variável `file_path`;
- 5- Por fim, após construir a lista listagem com o nome e o tamanho de todos os arquivos, a listagem é enviada ao cliente em:

```
if listagem:  
    client_socket.sendall("\n".join(listagem).encode())
```


Figura 07 - Recebido pedido de list no servidor.



```
C:\Windows\System32\cmd.e X + v  
Microsoft Windows [versão 10.0.22631.4602]  
(c) Microsoft Corporation. Todos os direitos reservados.  
  
C:\Users\JORDAN\Downloads\server>python tcp-file-server.py  
Servidor escutando em... 127.0.0.1:1234  
Cliente conectado: ('127.0.0.1', 59622)  
Recebida solicitação de listagem de arquivos do cliente ('127.0.0.1', 59622)  
Cliente ('127.0.0.1', 59622) desconectado.  
|
```

Fonte: Autoria Própria, 2025.

Figura 08 - Solicitando list no cliente.



```
C:\Windows\System32\cmd.e X + v
Microsoft Windows [versão 10.0.22631.4602]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\JORDAN\Downloads\client>python tcp-file-client.py
Digite o nome do arquivo desejado ou digite 0 para sair: list
Enviando pedido de listagem dos arquivos.

barco.jpg (7507955 bytes)
boneco.jpg (2739 bytes)
web.xml (803 bytes)
x.txt (40 bytes)

C:\Users\JORDAN\Downloads\client>
```

Fonte: Autoria Própria, 2025.

## 2.2 Comando sget

Cliente:

Figura 09 - Comando sget cliente.

```
elif nomeArq == "sget":
    pedido = "2"
    print("Enviando pedido de download de um único arquivo ao servidor...\n")
    tcpSock.send(pedido.encode('utf-8'))

    dataTam = tcpSock.recv(2048)
    resposta = (dataTam.decode('utf-8'))
    arquivo = input(resposta)

    laco = True
    while laco:
        if arquivo[0:2] == "..":
            print ("Enviando pedido a", (SERVER, PORT), "para", arquivo)
            tcpSock.send(arquivo.encode('utf-8'))
            dataTam = tcpSock.recv(2048)
            resposta = (dataTam.decode('utf-8'))
            print(resposta)

        else:
            print ("Enviando pedido a", (SERVER, PORT), "para", arquivo)
            tcpSock.send(arquivo.encode('utf-8'))

            dataTam = tcpSock.recv(2048)

            try:
                tamArq = int(dataTam.decode('utf-8'))
                print(f"O arquivo '{arquivo}' possui o tamanho de {tamArq} Bytes.")
            except ValueError:
                continue

            pedir_arquivo(arquivo, tamArq)
            laco = False
```

Fonte: Autoria Própria, 2025.

Caso o usuário solicite ao servidor usando o comando **sget**, o cliente solicitará ao servidor o pedido de download de um único arquivo, enviando o código representado pelo **"2"**. Após isso, o servidor envia de volta qual o arquivo o cliente deseja representado pelo `arquivo = input(resposta)` e `print(resposta)`. Caso o nome do arquivo desejado contenha `"../"`, é feita uma verificação e envia ao servidor esperando a resposta de que é proibido tentar essa ação. Caso não, a resposta do servidor é o tamanho do arquivo representado por `tamArq = int(dataTam.decode('utf-8'))`. Caso o valor seja 0 ou maior, a função `pedir_arquivo()` será chamada para fazer o download do arquivo ou informar que o arquivo não existe.

Servidor:

Figura 10 - Comando sget servidor.

```
# Caso o cliente solicite o envio de apenas um arquivo (sget) - Comando 2
elif filename == "2":
    print(f'Recebida a solicitação de apenas um arquivo do cliente {client_address}')
    client_socket.sendall(b'Envie o nome do arquivo desejado: ')
    arquivo = client_socket.recv(4096).decode()

    path = os.path.join(DIRETORIO, arquivo)
    if os.path.isfile(path):
        tam = os.path.getsize(path)
        print(f'Enviando o tamanho do arquivo {arquivo} ({tam} bytes) ao cliente {client_address}')
        client_socket.sendall(str(tam).encode())
        # Envia o arquivo em blocos de 2048 bytes
        with open(path, 'rb') as file:
            while chunk := file.read(4096):
                client_socket.sendall(chunk)
        print(f'Arquivo {arquivo} enviado com sucesso para {client_address}')

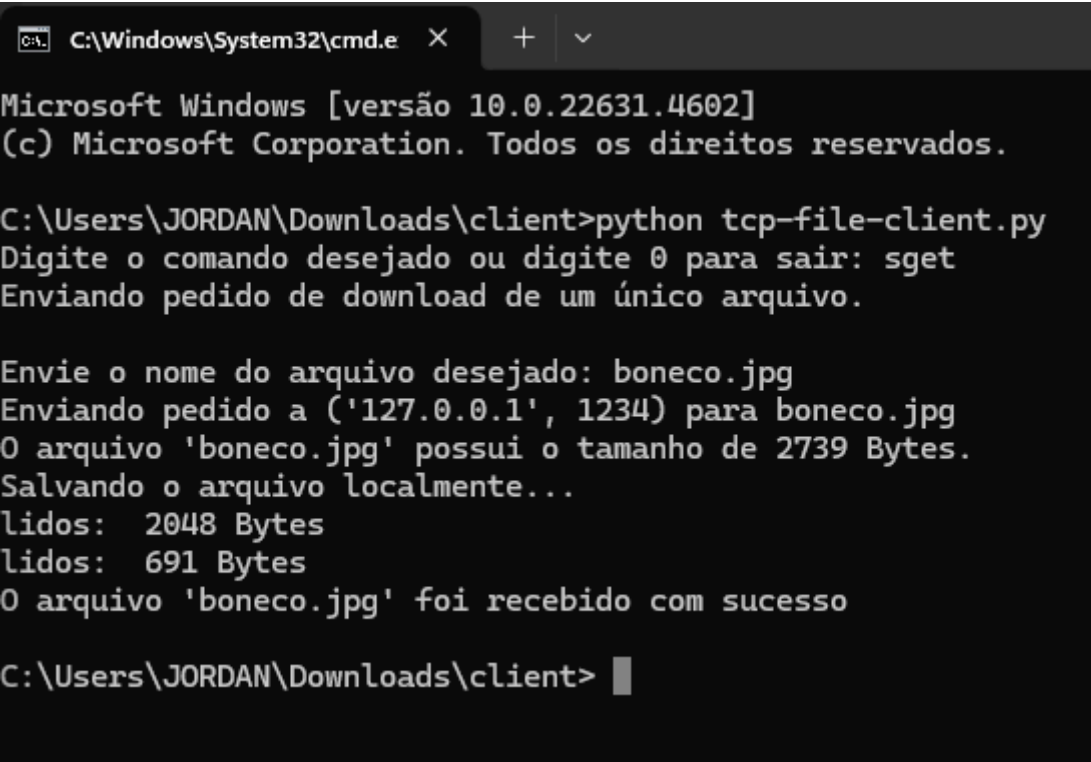
    elif '..' in filename:
        in_directory(DIRETORIO, file_path)
```

Fonte: Autoria Própria, 2025.

Caso o servidor receba o comando “2”. O servidor entregará o arquivo solicitado pelo cliente, da seguinte forma:

- 1- Encontrar o arquivo a partir do caminho completo presente na variável `path`.
- 2- Caso o arquivo exista na pasta files, ele obterá o tamanho do arquivo a partir da função `os.path.getsize(path)`. Após isso, ele enviará o tamanho ao cliente.
- 3- Para enviar os bytes que compõem o arquivo, o servidor enviará os bytes em blocos de 4096 bytes ao cliente.
- 4- Caso o arquivo solicitado possuir '..', a função `in_directory` que remove esses dois pontos será aplicada.

Figura 11 - Solicitando arquivo no cliente.



```
C:\Windows\System32\cmd.e X + v
Microsoft Windows [versão 10.0.22631.4602]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\JORDAN\Downloads\client>python tcp-file-client.py
Digite o comando desejado ou digite 0 para sair: sget
Enviando pedido de download de um único arquivo.

Envie o nome do arquivo desejado: boneco.jpg
Enviando pedido a ('127.0.0.1', 1234) para boneco.jpg
O arquivo 'boneco.jpg' possui o tamanho de 2739 Bytes.
Salvando o arquivo localmente...
lidos: 2048 Bytes
lidos: 691 Bytes
O arquivo 'boneco.jpg' foi recebido com sucesso

C:\Users\JORDAN\Downloads\client> █
```

Fonte: Autoria Própria, 2025.



## 2.3 Comando mget:

Cliente:

Figura 12 - Comando mget cliente.

```
elif nomeArq == "mget":
    pedido = "3"
    print("Enviando pedido de download de múltiplos arquivos por máscara ao servidor...\n")
    tcpSock.send(pedido.encode('utf-8'))
    resposta = tcpSock.recv(4096).decode('utf-8')

    print(resposta)
    arquivo_mask = input("Digite a máscara de arquivos desejada (ex.: *.jpg): ")
    tcpSock.send(arquivo_mask.encode('utf-8'))

    qtd_arquivos = int(tcpSock.recv(4096).decode('utf-8'))
    if qtd_arquivos == 0:
        print("Nenhum arquivo encontrado com a máscara fornecida.")

    print(f"{qtd_arquivos} arquivos encontrados. Recebendo...")
    # Verifique se o formato dos metadados é válido
    while True:
        # Receber metadados do arquivo
        metadados = tcpSock.recv(4096).decode('utf-8').strip()

        if metadados == 'Encerrado':
            print("Todos os arquivos foram recebidos com sucesso!")
            break

        elif ':' not in metadados:
            print(f"Erro nos metadados recebidos: {metadados}")
            continue

        nome_arquivo, tamanho_arquivo = metadados.split(':')
        tamanho_arquivo = int(tamanho_arquivo)
        print(f"Recebendo arquivo '{nome_arquivo}' ({tamanho_arquivo} bytes)")
        caminho_arquivo = os.path.join(DIRETORIO, nome_arquivo)

        if os.path.exists(caminho_arquivo):
            resposta = input(f"O arquivo '{nome_arquivo}' já existe. Deseja sobrescrevê-lo? (s/n): ").strip().lower()
            if resposta != 's':
                print(f"Arquivo '{nome_arquivo}' não será sobrescrito. Pulando...")
                continue # Não sobrescreve o arquivo e passa para o próximo
```

Fonte: Autoria Própria, 2025.

Caso o usuário solicite fazer o download de múltiplos arquivos, deverá utilizar o comando `mget`, representado pelo número `"3"`. Ao enviar o comando, será enviado ao servidor que deverá responder com o pedido de uma máscara (por ex.: `*.jpg`) que o cliente deverá digitar. Após isso, será recebido em `qtd_arquivos` a quantidade de arquivos que o servidor possui e caso seja igual a zero, encerrará o `mget`. Caso contrário, entrará no loop para ficar recebendo os dados que estarão no formato `NomeDoArquivo:TamanhoDoArquivo` separado por `":"` na primeira vez e será baixado utilizando uma versão modificada da função `pedir_arquivo()`.

Servidor:

Figura 13 - Comando mget servidor.

```
# Caso o cliente solicite mais de um arquivo que contenham máscara (mget) - Comando 3
elif filename == "3":
    print(f'Recebida a solicitação de arquivos com máscara do cliente {client_address}')
    client_socket.sendall(b'Envie a mascara de arquivos desejada (exemplo: *.jpg): ')
    mask = client_socket.recv(4096).decode()
    print(f'Recebida máscara de arquivos '{mask}' do cliente {client_address}')

    # Obtém a lista dos arquivos pertencentes a máscara
    try:
        files = glob.glob(os.path.join(DIRETORIO, mask))
        qtd_files = len(files) # Quantidade de arquivos que atendem a máscara
        client_socket.sendall(f'{qtd_files}'.encode())
        if files:
            client_socket.sendall(b'Arquivos encontrados. Iniciando envio...')
            for file_path in files:
                if in_directory(DIRETORIO, file_path):
                    filename = os.path.basename(file_path)
                    file_size = os.path.getsize(file_path)
                    print(f'Enviando arquivo '{filename}' ({file_size} bytes) ao cliente {client_address}')
                    client_socket.sendall(f'{filename}:{file_size}\n'.encode()) # Enviando o nome e o tamanho

            # Enviar conteúdo do arquivo
            with open(file_path, 'rb') as file:
                while chunk := file.read(4096):
                    client_socket.sendall(chunk)
            client_socket.sendall(b'FINAL\n') # Delimita o fim do arquivo
            print(f'Arquivo {filename} enviado com sucesso ao cliente {client_address}')
            time.sleep(0.5)
```

Fonte: Autoria Própria, 2025.

Caso o servidor receba o comando “3” do cliente, ele receberá a solicitação para enviar mais de um arquivo a partir de uma máscara, da seguinte forma:

- 1- Envia ao cliente uma solicitação para retornar ao servidor a máscara desejada pelo usuário;
- 2- Após receber a máscara, a partir da função `files = glob.glob(os.path.join(DIRETORIO, mask))` ele procurará os arquivos que atendem a ela. Após encontrar esses arquivos, a variável `qtd_files` irá armazenar a quantidade de arquivos a serem enviados, enviando-os ao cliente.
- 3- Com o cliente sabendo quantos arquivos serão enviados, o servidor enviará os arquivos sucessivamente ao cliente. Enviando o nome do arquivo e seu tamanho. Posteriormente, envia os bytes que compõem cada arquivo.

## 2.4 Comando hash:

Cliente:

Figura 14 - Comando hash cliente.

```
elif nomeArq== "hash":
    pedido = "4"
    print("Enviando pedido de Cálculo de Hash...\n")
    tcpSock.send(pedido.encode('utf-8'))

    dataTam = tcpSock.recv(2048)
    resposta = dataTam.decode('utf-8')

    arquivo = input(resposta)
    laco = True
    while laco:
        if arquivo[0:2] == "..":
            tcpSock.send(arquivo.encode('utf-8'))
            dataTam = tcpSock.recv(2048)
            resposta = (dataTam.decode('utf-8'))
            print(resposta)
            laco = False

        else:
            hashCalc = tcpSock.recv(2048).decode('utf-8')
            print(hashCalc)

            dataTam = tcpSock.recv(2048)
            resposta2 = dataTam.decode('utf-8')
            print(resposta2)
            laco = False
```

Fonte: Autoria Própria, 2025.

Caso o usuário solicite o hash de um arquivo, deverá utilizar o comando **hash** representado pelo número **“4”**. Quando enviado ao servidor, receberá uma resposta solicitando que envie o nome do arquivo. Por fim, receberá o hash calculado pelo servidor que será printado na tela.

Servidor:

Figura 15 - Comando hash servidor.

```
# Caso o cliente solicite o hash SHA 1 de um arquivo (hash) - Comando 4
elif filename == "4":
    print(f'Recebida a solicitação do hash SHA 1 do cliente {client_address}')
    client_socket.sendall(b'Envie o nome do arquivo e o posicionamento desejado (exemplo: barco.jpg:500): ')
    hash = client_socket.recv(4096).decode()
    name_arq, pos = hash.split(':') # Separação do nome do arquivo e da posição enviada pelo cliente
    pos = int(pos)
    if not ":" in hash:
        erro = client_socket.sendall(f'ERRO! O formato de escrita para o hash {hash} está incorreto!'.encode())
        print(erro)

    elif name_arq not in DIRETORIO:
        erro = client_socket.sendall(f'ERRO! o arquivo {name_arq} não existe no diretório {DIRETORIO}'.encode())
        print(erro)

    elif ":" in hash:
        path = os.path.join(DIRETORIO, name_arq)
        if os.path.isfile(path):
            # Cálculo do hash
            with open(path, 'rb') as file: # Lendo o arquivo em bytes até a posição solicitada pelo cliente
                client_socket.sendall(f'Obtendo o hash SHA1 do arquivo {name_arq} até a posição {pos}'.encode())
                calc = calcular_hash(path)
                client_socket.sendall(f'O hash SHA1 obtido do arquivo {name_arq} até a posição {pos} corresponde a : \n{calc}'.encode())

    elif '..' in filename:
        in_directory(DIRETORIO, file_path)
```

Fonte: Autoria Própria, 2025.

Caso o servidor receba o comando “4”, o servidor recebe o nome do arquivo e a posição solicitada pelo usuário. Após isso, ele entregará o nome do arquivo e seu respectivo hash, da seguinte forma:

- 1- Ao receber o nome e o hash no formato nome:posição, o programa vai separar o nome do arquivo da posição com o `split`, armazenando na variável `hash`;
- 2- Caso tenha “:” na variável hash, e o nome do arquivo esteja na pasta files do servidor, o cálculo do hash será realizado;
- 3- O resultado do hash está presente na variável `calc` que chamará a função `calcular_hash(path)`.

Figura 16 - Função calcular hash.

```
# Função para calcular o hash, apresentada nos comandos hash e eget
def calcular_hash(file_path, tamanho=None):
    with open(file_path, 'rb') as f:
        if tamanho:
            return hashlib.sha1(f.read(tamanho)).hexdigest()
        return hashlib.sha1(f.read()).hexdigest()
```

Fonte: Autoria Própria, 2025.

- 4- Após isso o hash é enviado.

## 2.5 Comando cget:

Servidor:

Figura 17 - Comando cget servidor.

```
# Caso o cliente desejar continuar o download de um arquivo do servidor a partir de onde foi interrompido
elif filename == "5":
    print(f"Solicitação para continuar download do cliente {client_address}")
    client_socket.sendall(b'Envie o nome do arquivo e o hash da parte baixada (exemplo: barco.jpg:hash): ')
    novo_hash = client_socket.recv(4096).decode()
    print(novo_hash)
    name_arq, hash_cliente = novo_hash.split(':') # Separa nome e o hash para serem tratados separadamente
    print(name_arq, hash_cliente)
    path = os.path.join(DIRETORIO, name_arq)

    if os.path.isfile(path):
        hash_servidor = calcular_hash(path)
        print(hash_servidor)

        if hash_cliente == hash_servidor:
            client_socket.sendall(f'Os hashes dos arquivos são iguais, logo o arquivo na pasta files do cliente está completo'.encode('utf-8'))
        else:
            client_socket.sendall(f'Os hashes dos arquivos são diferentes, logo o arquivo na pasta files do cliente está incompleto'.encode('utf-8'))
            tam_atual = int(client_socket.recv(4096).decode()) # Tamanho dos bytes do arquivo incompleto em files do cliente
            print(tam_atual)

            with open(path, 'rb') as file: # Seleciona o último byte onde o cliente interrompeu o download e armazena na variável resto
                file.seek(tam_atual)
                resto = file.read()
                tam_arq = len(resto) # tam_arq armazena o tamanho dos bytes que faltam para completar o arquivo
                client_socket.sendall(f'{int(tam_arq)}'.encode('utf-8')) # Envia o tamanho dos bytes que faltam para completar o arquivo
                for i in range(0, tam_arq, 4096): # Completando os bytes do arquivo interrompido para formar o arquivo completo
                    chunk = resto[i:i+4096]
                    client_socket.sendall(chunk)
                client_socket.sendall(f'Envio do restante do arquivo concluído.'.encode('utf-8'))

    elif '..' in filename:
        in_directory(DIRETORIO, file_path)
```

Fonte: Autoria Própria, 2025.

Caso o servidor receba o comando **"5"**, o servidor deverá continuar o download de um arquivo que não foi concluído. Assim, o servidor receberá do cliente o nome do arquivo seguido por "." e o comando hash. A continuação do download será feita da seguinte forma:

1- Separa o nome do arquivo do hash e armazena cada um nas suas respectivas variáveis em:

```
name_arq, hash_cliente = novo_hash.split(':')
```

2- Possuindo o hash do arquivo presente na pasta files do cliente, ele irá comparar esse hash com o do mesmo arquivo na pasta files do servidor, caso ambos forem iguais, logo a imagem na pasta do cliente está completa. Caso sejam diferentes, ele entrará em uma condição para completar os bytes restantes.

3- Para completar os bytes restantes, primeiro ele recebe o tamanho do arquivo incompleto na pasta files do cliente para descobrir quantos bytes faltam para completar o arquivo.

4- A partir das linhas abaixo o processo que adicionam os bytes no arquivo completo é realizado:

Figura 18 - Código para adicionar os bytes restantes.

```
with open(path, 'rb') as file: # Seleciona o último byte onde o cliente interrompeu o download e armazena na variável resto
    file.seek(tam_atual)
    resto = file.read()
tam_arq= len(resto) # tam_arq armazena o tamanho dos bytes que faltam para completar o arquivo
client_socket.sendall(f'{int(tam_arq)}.encode('utf-8')) # Envia o tamanho dos bytes que faltam para completar o arquivo
for i in range(0,tam_arq, 4096): # Completando os bytes do arquivo interrompido para formar o arquivo completo
    chunk = resto[i:i+4096]
    client_socket.sendall(chunk)
client_socket.sendall(f'Envio do restante do arquivo concluído.'.encode('utf-8'))
```

Fonte: Autoria Própria, 2025.

Cliente:

Figura 19 - Comando cget cliente.

```
else:
    arquivo, hashPedido = arquivo.split(":")
    if hashPedido == "hash":
        print("Calculando hash do arquivo local...")
        path= os.path.join(DIRETORIO, arquivo)
        # Calculo do hash do arquivo na pasta files do CLIENTE
        if os.path.isfile(path):
            with open (path, 'rb') as file:
                leitura = file.read()
                calc= hashlib.sha1(leitura).hexdigest()
                client_hash= calc
                print(client_hash)
                tcpSock.send(f"{arquivo}:{client_hash}".encode())

        dataTam = tcpSock.recv(2048)
        resposta = dataTam.decode('utf-8')
        print(resposta)

        if resposta == "Os hashes dos arquivos são diferentes, logo o arquivo na pasta files do cliente está incompleto":
            print(resposta)

            with open (path, 'rb') as file:
                leitura = file.read()
                arqLocal = len(leitura)
                tcpSock.send(f"{int(arqLocal)}.encode()")

            dataTam = tcpSock.recv(2048)
            tamArq = int(dataTam.decode('utf-8'))
            print(f"0 restante do arquivo '{arquivo}' possui {tamArq} bytes.\n")
            print("Baixando o restante do arquivo, aguarde...\n")
            time.sleep(3)

            if tamArq > 0:
                with open(DIRETORIO + arquivo, "ab") as fd:
                    recebido = 0
                    while recebido < tamArq:
                        data = tcpSock.recv(4096)
                        fd.write(data)
                        print("Lidos: ", recebido, "Bytes")
                        recebido += len(data)
                dataTam = tcpSock.recv(2048)
                resposta = dataTam.decode('utf-8')
                print(resposta)
```

Fonte: Autoria Própria, 2025.

No input, o usuário digitará “NomeDoArquivo:hash” que “:” será o separador para a função `arquivo.split(":",)`, e o conteúdo vai para as variáveis `arquivo` e `hashPedido`. Se `hashPedido` for diferente da string “hash”, informará um erro. Caso contrário, será calculado o hash SHA1 do arquivo parcial que existe na pasta files do cliente que será aberto e utilizando a biblioteca `hashlib` e será enviado tanto o nome quanto o hash calculado ao servidor. Caso a resposta do servidor seja que o hashes são iguais, significa que

o arquivo está completo, mas se for diferente, o cliente receberá o tamanho restante do arquivo e começará a adicionar o restante dos bytes ao arquivo parcial do cliente.