Nome: João Paulo Vieira de Carvalho **RA:** 11004415

Link do vídeo do funcionamento (screencast):

Não foi possível produzir o vídeo. Apesar disso, consta abaixo a lista de funcionalidades que deveria ser exibida. Ela será usada para elencar as funcionalidades que foram implementadas com sucesso

a. Join do peer e aceitação pelo Servidor.
b. Busca por um arquivo que não existe no sistema.
c. Busca por um arquivo que existe no sistema.
d. Download de um arquivo de vídeo. Mostre que pesa mais de 1 GB, mostre que na pasta não existe nenhum arquivo antes do download.
e. Mostre a visualização do vídeo por um reprodutor de vídeo após o download.

Explicação das funcionalidades do servidor:

Requisição JOIN (linhas 47-60):

```
47      #Seção 4.b Requisição JOIN
48      def resolve_JOIN(self, peer_info):
49          # Processar informações do par e armazená-las
50          # Exemplo: peer_info = ['PeerX', 'file1.mp4', 'file2.mp4']
51          peer_nome = peer_info[0]
52          peer_arquivos = peer_info[1:]
53
54          peer_endereco = peer_nome.split(":")
55          peer_ip = peer_endereco[0]
56          peer_porta = int(peer_endereco[1])
57
58          self.peers[(peer_ip, peer_porta)] = peer_arquivos
59
60          return "JOIN_OK"
```

O servidor recebe uma requisição JOIN do peer, processa as informações do peer e armazena em um dicionário.

Requisição SEARCH (linhas 62-73):

```
61
62      #Seção 4.c Requisição SEARCH
63      def resolve_SEARCH(self, nome_arquivo):
64          # Procura peers que possuem o arquivo solicitado
65          peers_com_arquivo = []
66          for peer, arquivos in self.peers.items():
67              if nome_arquivo in arquivos:
68                  peers_com_arquivo.append(peer)
69
70          # Formata os pares de IP e Porta
71          peers_formatados = [f"{ip}:{porta}" for ip, porta in peers_com_arquivo]
72
73          return " ".join(peers_formatados)
74
```

O servidor recebe uma requisição SEARCH do peer, verifica quais peers possuem o arquivo solicitado e retorna os endereços formatados dos peers.

Requisição UPDATE (linhas 75-82):

```
74
75      #Seção 4.d Requisição UPDATE
76      def resolve_UPDATE(self, nome_arquivo):
77          # Atualiza informações do peer após o download de um arquivo
78          for peer, files in self.peers.items():
79              if nome_arquivo not in files:
80                  self.peers[peer].append(nome_arquivo)
81
82          return "UPDATE_OK"
83
```

O servidor recebe uma requisição UPDATE do peer após o download de um arquivo e atualiza as informações do peer.

Explicação das funcionalidades do peer:

Requisição JOIN (linhas 34-56):

```
34      #Seção 5.b Requisição JOIN
35      def envia_JOIN(self):
36          servidor_ip = input("Servidor IP: ")
37          servidor_porta = int(input("Servidor Porta: "))
38
39          try:
40              peer_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
41              peer_socket.connect((servidor_ip, servidor_porta))
42
43              requisicao_JOIN = f"JOIN {self.ip}:{self.porta} {self.get_nome_arquivo()}"
44              peer_socket.sendall(requisicao_JOIN.encode())
45
46              resposta = peer_socket.recv(1024).decode()
47              if resposta == "JOIN_OK":
48                  print(f"Sou peer {self.ip}:{self.porta} com arquivos {self.get_nome_arquivo()}")
49                  self.envia_UPDATE(servidor_ip, servidor_porta)
50              else:
51                  print("Falha no Registro com o Servidor")
52
53              peer_socket.close()
54          except Exception as e:
55              print("Erro enquanto conectava com o servidor:", str(e))
56
```

O peer envia uma requisição JOIN ao servidor, informando seu endereço IP, porta e a lista de arquivos que possui.

Requisição UPDATE (linhas 56-75):

```
56
57      #Seção 5.c Requisição UPDATE
58      def envia_UPDATE(self, servidor_ip, servidor_porta):
59          try:
60              peer_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
61              peer_socket.connect((servidor_ip, servidor_porta))
62
63              requisicao_UPDATE = f"UPDATE {self.ip}:{self.porta} {self.get_nome_arquivo()}"
64              peer_socket.sendall(requisicao_UPDATE.encode())
65
66              # resposta = peer_socket.recv(1024).decode()
67              # if resposta == "UPDATE_OK":
68              #     print("Atualização enviada com sucesso para o servidor")
69              # else:
70              #     print("Falha ao enviar atualização para o servidor")
71
72              peer_socket.close()
73          except Exception as e:
74              print("Erro ao se conectar com o servidor:", str(e))
75
```

O peer envia uma requisição UPDATE ao servidor após o download de um arquivo, atualizando as informações do peer no servidor.

Requisição SEARCH (linhas 91-114):

```
90
91      #Seção 5.d Requisição SEARCH
92      def envia_SEARCH(self):
93          nome_arquivo = input("Insira o nome do arquivo para buscar: ")
94
95          try:
96              servidor_ip = input("Servidor IP: ")
97              servidor_porta = int(input("Servidor Porta: "))
98
99              peer_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
100             peer_socket.connect((servidor_ip, servidor_porta))
101
102             requisicao_SEARCH = f"SEARCH {self.ip}:{self.porta} {nome_arquivo}"
103             peer_socket.sendall(requisicao_SEARCH.encode())
104
105             resposta = peer_socket.recv(1024).decode()
106             if resposta:
107                 peers_com_arquivo = resposta.split()
108                 print("peers com arquivo solicitado:", " ".join(peers_com_arquivo))
109             else:
110                 print("Não há nenhum peer com este arquivo.")
111
112             peer_socket.close()
113         except Exception as e:
114             print("Erro ao se conectar com o servidor:", str(e))
115
```

O peer envia uma requisição SEARCH ao servidor, informando o nome do arquivo que deseja buscar. O peer recebe a resposta do servidor contendo os endereços dos peers que possuem o arquivo.

Requisição enviar, realizar DOWNLOAD e receber arquivo(linhas 116-124, 126-149 e 151-164):

```
115
116     #Seção 5.f Ainda precisa arrumar.
117     def envia_DOWNLOAD(self):
118         peer_ip = input("Peer IP: ")
119         peer_porta = int(input("Peer Porta: "))
120         nome_arquivo = input("Nome do arquivo para baixar: ")
121
122
123         download_thread = threading.Thread(target=self.realizar_download, args=(peer_ip, peer_porta, nome_arquivo))
124         download_thread.start()
125
```

```python

    #Seção 5.g Ainda precisa arrumar.
    def realizar_download(self, peer_ip, peer_porta, nome_arquivo):
        try:

            peer_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

            peer_socket.connect((peer_ip, peer_porta))

            requisicao_DOWNLOAD = f"DOWNLOAD {nome_arquivo}"
            peer_socket.sendall(requisicao_DOWNLOAD.encode())

            resposta = peer_socket.recv(1024).decode()
            if resposta.startswith("Arquivo Encontrado!"):
                self.receber_arquivo(peer_socket, nome_arquivo)
            elif resposta == "Arquivo não encontrado":
                print("Arquivo não encontrado no peer")
            else:
                print("Erro durante a transferência do arquivo")

            peer_socket.close()
        except ConnectionRefusedError:
            print("Conexão recusada. Verifique se o peer está em execução e a porta está correta.")
        except Exception as e:
            print("Erro ao conectar com o Peer:", str(e))

```

```python
    #Seção 5.h
    def receber_arquivo(self, peer_socket, nome_arquivo):
        try:
            caminho_arquivo = os.path.join(self.diretorio, nome_arquivo)
            with open(caminho_arquivo, "wb") as arquivo:
                while True:
                    data = peer_socket.recv(1024)
                    if len(data) == 0:
                        break
                    arquivo.write(data)

            print(f"Arquivo {nome_arquivo} baixado com sucesso na pasta {self.diretorio}")
        except Exception as e:
            print("Erro enquanto recebia o download:", str(e))

```

O peer envia uma requisição DOWNLOAD a outro peer para baixar um arquivo específico. O peer estabelece uma conexão com o peer que possui o arquivo e realiza o download, que chama a função receber_arquivo.

Uso de threads:

No método iniciar do peer.py, uma thread é criada para executar o método iniciar_peer_servidor. A função threading.Thread é usada para criar uma nova thread, passando o método iniciar_peer_servidor como alvo, e em seguida, a thread é iniciada com o método start.

```python
    def iniciar(self):
        server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server_socket.bind((self.ip, self.porta))
        server_socket.listen(5)
        print(f"Servidor iniciado em {self.ip}:{self.porta}")

        while True:
            peer_socket, client_address = server_socket.accept()
            self.resolve_requisicao_PEER(peer_socket, client_address)

```

Dessa forma, o peer é iniciado em uma thread separada, permitindo que o programa principal continue a execução e fique disponível para receber interações do usuário por meio do menu interativo.

Além disso, no método envia_DOWNLOAD, uma nova thread é criada para realizar o download de um arquivo. A função realizar_download é executada nessa nova thread, passando os parâmetros necessários. Isso permite que o programa continue executando outras operações enquanto o download é realizado em segundo plano.

Portanto, o código utiliza threads para permitir a execução concorrente do servidor e do processo de download, melhorando a capacidade de resposta e permitindo que o programa execute várias operações simultaneamente.

Implementação de transferência de arquivos gigantes:

A implementação de transferência de arquivos gigantes não foi possível de ser realizada. Desta forma, as linhas de código relevantes não estão presentes.

Links dos lugares de onde baseou seu código:

https://docs.python.org/3/library/os.html
https://docs.python.org/3/library/socket.html
https://docs.python.org/3/library/threading.html