Bruno de Sousa Almeida - NºUSP: 9911451

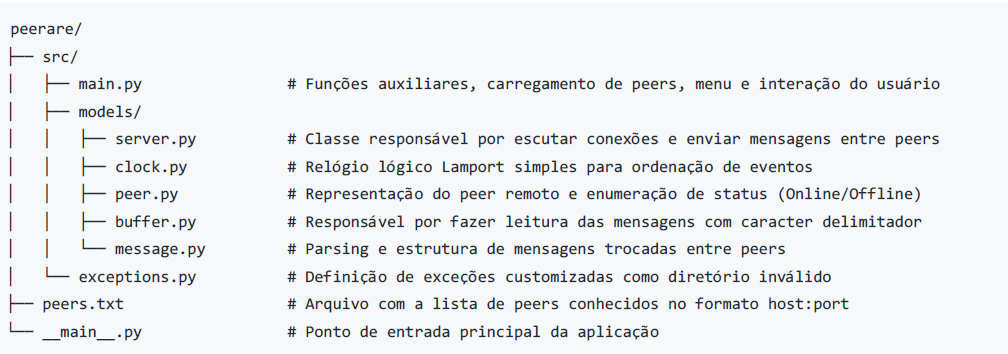
**Relatório do Exercício Programa Parte 3**

São Paulo

2025

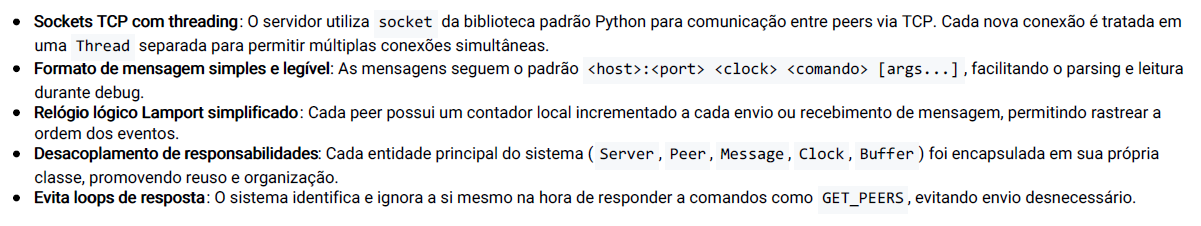
# Estrutura do Projeto

O projeto está organizado de forma modular para facilitar a leitura, manutenção e evolução do código. A estrutura segue a separação entre responsabilidades:

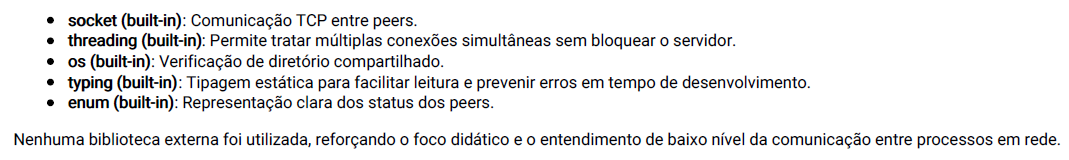


**Vale frisar que todo o conteúdo listado abaixo encontra-se no arquivo *readme.md*, anexado ao projeto no Python.**

# Decisões do Projeto



# Bibliotecas Utilizadas



1. **Como as escolhas feitas na implementação das partes anteriores influenciaram as alterações necessárias para a parte 3?**

Nas partes anteriores, a implementação estabelecia uma nova conexão para cada mensagem enviada, fechando-a logo em seguida. Esse modelo, porém, se mostrou inviável no envio de arquivos em chunks, pois para arquivos grandes — combinados com chunks pequenos — isso resultava na abertura e fechamento de um grande número de conexões em um curto espaço de tempo. Como consequência, o sistema operacional acabava bloqueando ou limitando o fluxo devido ao uso excessivo de sockets em estados TIME\_WAIT .

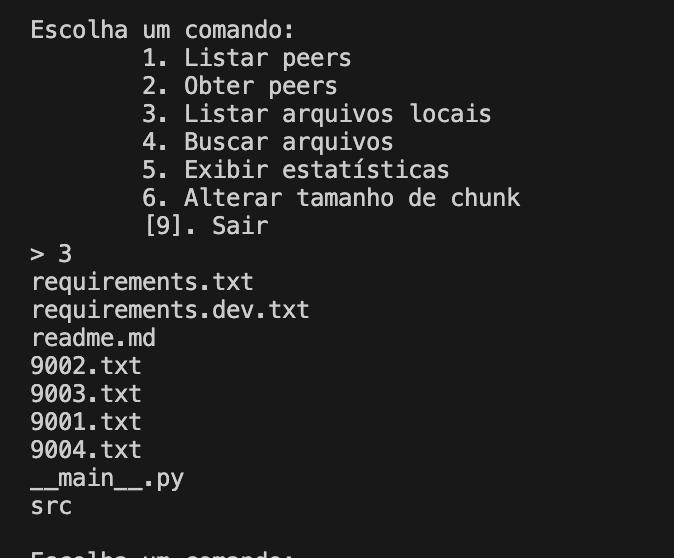
Para resolver isso, na parte 3, foi necessário revisar a arquitetura de comunicação, passando a manter conexões persistentes entre os peers. Isso exigiu também uma

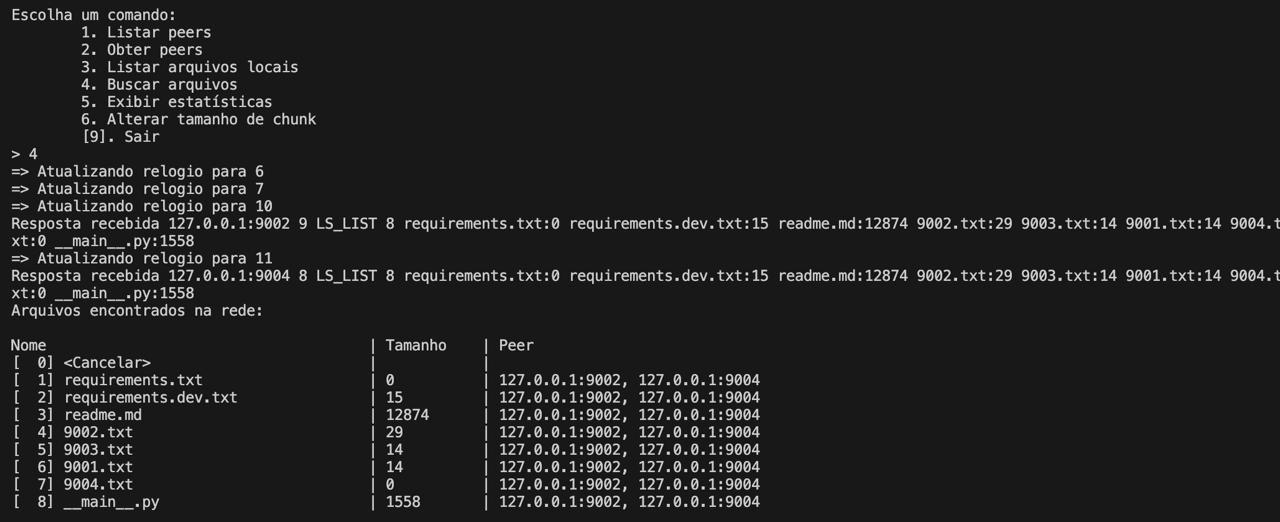
alteração na forma de interpretar os dados recebidos, já que uma conexão agora poderia transportar múltiplas mensagens e até mensagens parciais. Para isso,

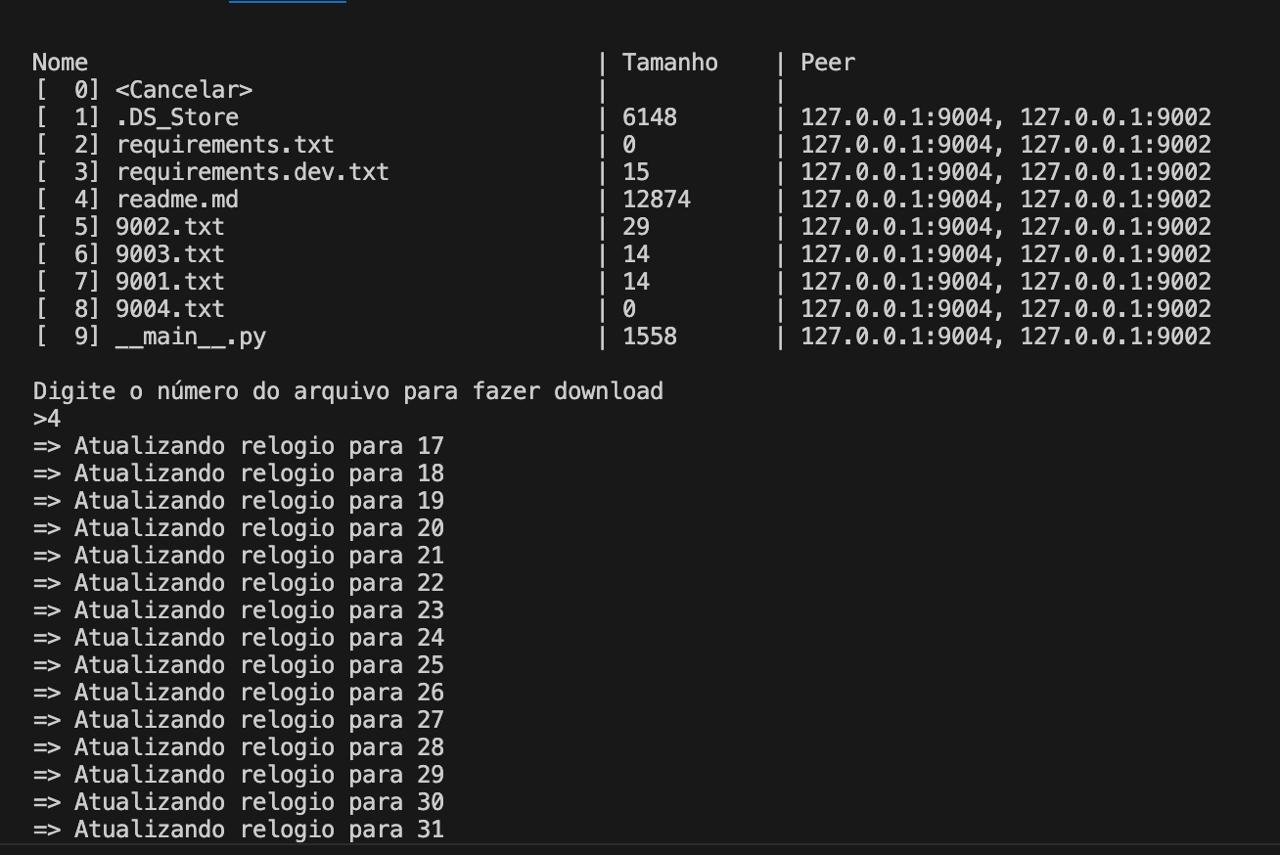
introduzi o uso de um marcador de fim de mensagem, permitindo identificar corretamente os limites entre as mensagens e reconstruí-las mesmo em streams

contínuos.

# Quais testes foram feitos?









1. **Como foi feita a distribuição de chunks entre os peers disponíveis?**

A distribuição dos chunks entre os peers disponíveis foi implementada utilizando a estratégia de round-robin (fila circular). Para cada chunk a ser baixado, o

sistema selecionava sequencialmente o próximo peer da lista de peers que possuíam o arquivo. Ao atingir o final da lista, o processo reiniciava a partir do primeiro peer, garantindo assim uma distribuição equilibrada e uniforme dos chunks entre todos os peers disponíveis.

Essa abordagem, apesar de simples, mostrou-se eficaz para o cenário da aplicação, pois:

* Permite um balanceamento natural da carga entre os peers.
* Evita que um único peer seja sobrecarregado.
* Reduz o tempo total de download ao explorar o paralelismo da rede.

1. **Como foi medido o tempo de download?**

O tempo de download foi medido da seguinte forma:

No momento em que foi iniciada a solicitação de download do arquivo (ou seja, quando os chunks começaram a ser requisitados aos peers), foi salvo no estado do servidor um timestamp de início, obtido com a função time.time() da biblioteca padrão do Python.

Quando foi detectado que todos os chunks haviam sido recebidos (ou seja, o download estava completo), foi feita a subtração entre o timestamp atual

(time.time()) e o timestamp salvo no início da requisição.

O resultado dessa subtração corresponde ao tempo total de download do arquivo, em segundos, com precisão de casas decimais.

1. **Resultados de experimentos**

Os testes realizados buscaram avaliar o impacto de diferentes escolhas de implementação na performance do sistema de envio de arquivos em chunks. A partir dos resultados obtidos, foi possível perceber os seguintes comportamentos:

**Chunks pequenos (128, 256):**

Aumentaram significativamente o tempo total de download, devido ao maior overhead de controle e número de mensagens trocadas.

Apresentaram maior desvio padrão, com tempos mais instáveis e sujeitos à variação do sistema operacional.

**Chunks maiores (1024, 2048, 5096):**

Reduziram substancialmente o tempo de download.

Proporcionaram transmissões mais estáveis, com desvios padrão menores.

A partir de um certo tamanho (~2048), o ganho adicional se estabilizou, indicando um “ponto ótimo”.

**Número de peers:**

* O uso de 3 peers foi mais vantajoso em arquivos grandes, pois permitiu maior paralelismo na transferência dos chunks.
* Em arquivos pequenos, o benefício de adicionar mais peers foi menos perceptível, pois o tempo de coordenação passa a ser relevante em relação ao tempo de transmissão.

**Estabilidade do protocolo:**

* A arquitetura com conexões persistentes e uso de marcador de fim de mensagem permitiu transmitir múltiplos chunks por conexão de forma confiável.
* Não foram observados problemas de perda ou corrompimento de mensagens, mesmo com variação no tamanho do chunk e no número de peers.

Em resumo, os testes demonstraram que a escolha do tamanho de chunk tem grande impacto na performance do sistema. Também ficou claro que o uso de

conexões persistentes foi essencial para garantir a estabilidade e a viabilidade da transmissão em chunks, evitando problemas que seriam causados por excesso de

aberturas/fechamentos de conexões. O sistema se comportou de forma robusta, escalando bem com o número de peers e com diferentes tamanhos de arquivos.