RELATÓRIO IMPLEMENTAÇÃO SERVIDORES CONCORRENTES

1. Descrição de cada Implementação:

- **Servidor Forked:** Para cada nova conexão de cliente, o processo principal cria um novo processo filho usando `os.fork()`. O processo filho é uma cópia do pai e lida com toda a comunicação daquele cliente específico, enquanto o processo pai continua a aceitar novas conexões
- **Servidor Multithread:** Ao aceitar uma nova conexão, o servidor cria uma nova thread para gerenciar a comunicação com aquele cliente. Isso é feito usando o módulo `threading` do Python. Todas as threads rodam dentro do mesmo processo, compartilhando memória
- **Servidor Orientado a Eventos (Assíncrono):** Detalhe que esta abordagem usa um único processo e uma única thread para gerenciar múltiplas conexões de forma não-bloqueante. Utilizando a biblioteca `asyncio` do Python, o servidor espera por eventos (como a chegada de novos dados) e alterna a execução entre as diferentes conexões, otimizando o tempo ocioso de espera por I/O (entrada/saída).

2. Resultados do teste de Desempenho:

Cenário de Teste	Métrica	Servidor Forked	Servidor Multithread	ıidor Orientado a Eve
Carga Leve (100 clientes, 10 msgs)	RPS	1553.50	1693.69	2778.06
	Consumo Máx. CPU	~9%	~8%	~11%
	Consumo Máx. Memória	~9 MiB	~8 MiB	~10 MiB
Carga Média (1000 clientes, 10 msgs)	RPS	~750	~1250	~2600
	Consumo Máx. CPU	~85%	~70%	~45%
	Consumo Máx. Memória	~90 MiB	~30 MiB	~18 MiB

3. Conclusões (Vantagens e Desvantagens):

Com base nos seus resultados, discuta as características de cada abordagem:

- **Servidor Forked:**
  - **Vantagens:** Bom isolamento entre clientes (um erro em um processo filho não afeta os outros). Relativamente simples de implementar para tarefas básicas.

    - **Desvantagens:** Alto consumo de memória e CPU, pois criar um processo é "caro" para o sistema operacional. Não escala bem para um grande número de conexões simultâneas. No Windows, `os.fork()` não está disponível, o que limita a portabilidade.
- **Servidor Multithread:**
    - **Vantagens:** Menor consumo de recursos do que o modelo forked, pois threads são mais "leves" que processos. O compartilhamento de memória entre threads pode facilitar a comunicação entre clientes (embora exija cuidado com sincronização).
    - **Desvantagens:** A complexidade aumenta devido à necessidade de gerenciar condições de corrida e usar travas (`locks`) para sincronização. O "Custo de troca de contexto" entre muitas threads pode degradar o desempenho.
- **Servidor Orientado a Eventos:**
    - **Vantagens:** Extremamente eficiente em termos de recursos (CPU e memória), pois usa um único processo/thread. Altamente escalável, capaz de lidar com milhares de conexões simultâneas com baixa sobrecarga. É a abordagem mais moderna e performática para aplicações com muita E/S (I/O-bound).
    - **Desvantagens:** A lógica de programação pode ser mais complexa de entender no início (programação assíncrona). Não é ideal para tarefas que exigem muito processamento da CPU, pois uma tarefa longa pode bloquear todo o loop de eventos.