1. **Modifique o programa cliente de echo do exercício anterior para que este receba como entrada e envie ao servidor não mais linhas digitadas pelo usuário, mas linhas de um arquivo binário qualquer (O arquivo será passado utilizando o caracter de redirecionamento '`<`'). O cliente continuará recebendo o eco enviado pelo servidor, que deverá ser escrito em um arquivo (O arquivo será criado utilizando o caracter de redirecionamento '`>`'). Seu programa deverá necessariamente utilizar ou a função `select` ou a função `poll`. Cada linha deve ser enviada separadamente para o servidor e elas não necessariamente precisam constituir comandos do Unix, porque o servidor não deverá executá-las. O servidor só irá enviá-las de volta para o cliente.**

   O cliente deve finalizar sua execução assim que tiver recebido todo o arquivo ecoado pelo servidor.

   - **Dica 1: Dependendo da sua forma de implementação, poderá acontecer do cliente encerrar a conexão ao ler uma linha em branco ou o fim do arquivo, sem esperar a chegada de todos os dados transmitidos pelo servidor. Caso isso aconteça, é necessário tratar o trecho de captura dos dados da entrada padrão no seu código. Uma forma de verificar se isso está ocorrendo é rodar o `diff` no cliente entre o arquivo de entrada (passado pelo '`<`') e o arquivo de saída (gerado pelo '`>`')**
   - **Dica 2: Uma boa ajuda para desenvolver esta atividade poderá ser encontrada nos exemplos do livro-texto da disciplina (capítulo 6) e nos programas utilizados na quarta atividade.**
     - **Links úteis:**
       select
       poll

2. **Comparando o código implementado no passo 1 com o código original da atividade prática 5 (considerando a transferência de arquivo) em termos de melhor utilização da rede e, consequentemente, tempo de execução, há alguma vantagem em utilizar o código novo? Justifique.**

   - **Dica 1: Uma forma de verificar a diferença é comparar o tempo de execução dos dois clientes. Para isso você pode utilizar o programa `time` que já vem embutido no shell bash e verificar a saída da linha `real`. Para executá-lo basta chamá-lo na linha de comando antes do cliente. Por exemplo:**

     ```
     mc833@localhost:$ time ./cliente 192.168.1.13 1234 < in > out

     real 0m0.191s

     user 0m0.004s

     sys 0m0.052s
     ```

- **Dica neste link**

Cliente exercicio 5:

real 0m0.012s

user 0m0.007s

sys 0m0.009s

Cliente exercicio 6:

real 0m0.009s

user 0m0.000s

sys 0m0.003s

O multiplexador TCP permite que o mesmo que ocorre para uma aplicação baseada em TCP ocorra aqui, com a diferença do reuso. Ao invés de ser limitado para apenas um cliente, ele é reutilizado para vários clientes. Resultando assim numa eficiência maior dos servidores e de aplicações mais velozes em performance.

1. Melhora na performance:

Todas as aplicações podem se beneficiar de um crescimento na performance, devido ao fato do servidor não ter que abrir e fechar centenas de conexões TCP por segundo. Ao remover a necessidade de gerenciar as conexões constantemente, a aplicação se preocupa apenas com gerar e disponibilidade o conteúdo demandado, especialmente quando o numero de requisições é alto.

2. Melhora na capacidade dos servidores:

Em geral, servidores são capazes de lidar com um número maximo de usuários concorrentes, e este numero, no passado, quase sempre ligado com o numero de conexões TCP simultaneas abertas que o sistema operacional e a aplicação podem lidar.

Ao reduzir o numero de conexões TCP demandadas pelo mesmo número de usuarios e requisições, ele deixa o servidor com a habilidade de suportar mais conexões e assim servir mais usuários.

3. Torna a consolidação mais fácil:

Como a multiplexação TCP melhora a capacidade dos servidores, isso torna possível reduzir o número total de servidores que são necessários para uma mesma base. Menos servidores significa que você pode redistribui-los para outros usuários ou desligá-los o que diminui custos de operação e gerenciamento. Ou você pode adiar a aquisições de novos servidores numa empresa nova.