Trabalho 4: Multiplexação de Entrada/Saída (MC833)

José Ribeiro Neto - RA 176665

1. Modifique o programa cliente da atividade 2 para que este receba como entrada e envie ao servidor linhas de um arquivo texto qualquer (O arquivo será passado utilizando o caracter de redirecionamento '<').

   (a) 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.

   (b) Cada linha deve ser enviada separadamente para o servidor. O servidor irá enviá-las de volta para o cliente, então cuidado com os $\backslash n$ e $\backslash t$.

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

   R: O programa foi modificado. O cliente está enviando de uma em uma linha lida na entrada padrão (stdin) para o servidor. O cliente está usando apenas a função 'select' para lidar com a multiplexão de dois descritores de arquivo: o descritor de arquivo da entrada (stdin, para leitura de um arquivo), e o descritor de arquivo do socket conectado com o servidor (para leitura do conteúdo ecoado). Para que fosse possível que o cliente envia-se de uma em uma linha de caracters pro servidor, optei por não utilizar o 'getline' ou o 'fgets' (dado que pra mim perderia um pouco o propósito em utilizar a multiplexação). Para isso, utilizei um buffer com tamanho máximo de 100000 elementos do tipo char. Assim, cada vez que o 'stdin' estava pronto para ser lido, lia somente um caracter, jogava no meu buffer, e verificava se o caracter era um $\backslash n$, um $\backslash t$ ou o arquivo havia chegado ao fim. Neste caso, quando um desses três casos ocorria, eu enviava todo o buffer para o servidor. Todo este processo está comentado no código que foi submetido em anexo. Também, utilizei multiplexação no servidor através do select. Contudo, diferentemente do cliente, não utilizei um buffer por socket aberto com cada cliente para armazenar uma linha completa e então ser enviada para o cliente. Simplesmente utilizei a função READ. E todo o conteúdo lido já era no mesmo instante enviado pelo servidor para o cliente. Todo este processo está comentado no código.

2. Cite um exemplo para uso de multiplexação de entrada e saída.

   R: Como exemplo de multiplexação, podemos citar o cenário onde um servidor precisa lidar tanto com um socket de escuta (para receber novas conexões com clientes) quanto com os sockets dos clientes (para realizar a comunicação com clientes). Uma solução que tínhamos visto nos últimos labs seria utilizar multi-threads ou multi-processos filho (através de fork) para resolver esse problema de multi-sockets no servidor. Contudo, utilizar multi-threads ou multi-processos requer muito mais recursos (tanto de memória quanto de execução na criação dos mesmos). E também, o número de threads / processos que produz uma performance aceitável de execução, frequentemente está associado com o número de cores da CPU do servidor. E

mais que isso, se o trabalho realizado por cada thread for pequeno (por exemplo, o cliente e o servidor trocam poucas informações), estaremos gastando muito recurso para manter as threads (e fazer as trocas das mesmas na CPU, dado que estamos assumindo que o número de threads é muito maior que o número de núcleos do processador) pra realizar pouco trabalho (execução por thread). Assim sendo, nem sempre é vantajoso utilizar este esquema de multi-threads / multi-processos. Contudo, podemos utilizar multiplexação para resolver este problema. Neste caso, teremos um servidor fazendo uso do select. No select, especificamos quais sockets estarão ativos para leitura / escuta. Assim, quando algum socket tiver algum conteúdo pra ser lido, o servidor sai da função select e realiza o trabalho necessário (lê uma stream de bytes do socket, por exemplo). Com isso, caso o trabalho realizado seja pequeno (ao ponto que o overhead de troca de threads na CPU seja significativo), é muito mais vantajoso utilizar a multiplexação (com o select) ao invés de multi-thread / multi-processo, dado que o maior custo de recursos fica a cargo de um único vetor (que no caso do meu sistema possui 16 elementos do tipo long int, totalizando 128 bytes). Ou seja, multiplexação é um mecanismo leve. Portanto, justifica seu uso neste exemplo.

Instruções de compilação e Execução:

1. **Instruções de Compilação: apenas rodar o comando 'make' no terminal a partir da pasta raiz. O comando 'make' utilizará o arquivo 'Makefile' para chamar o g++ e assim compilar o código.**
2. **Instruções de Execução:**
   (a) **Primeiro rodar o servidor com o comando './bin/servidor.o X', sendo X aqui a porta onde o servidor ficará escutando requisições.**
   (b) **Depois de executar o servidor, executar o comando './bin/cliente.o X Y < input.txt > output.txt' pra rodar o programa do cliente. Aqui, 'Y' representa a porta que o servidor está escutando (passada como argumento para o servidor) e 'X' representa o IP que o servidor estará escutando requisições. Também, 'input.txt' representa o arquivo txt a ser lido da entrada padrão pelo cliente, o qual será enviado para o servidor e então ecoado. Já o arquivo 'output.txt' será o arquivo que será gerado pelo cliente, contendo o conteúdo ecoado pelo servidor.**