Chat Cliente / Servidor

Neste exercício, para ser realizado em duplas, vocês irão implementar um sistema de bate-papo. O sistema irá incluir vários clientes que se comunicam através de um servidor central.

Especificação: O trabalho consiste na concepção e a implementação de ambos os lados servidor e cliente, e do protocolo de comunicação entre eles.

Parte 1: o servidor

O programa chamado "server_chat" receberá um único argumento - um número de porta.

Uso: server_chat <PORT>

O servidor irá escutar as conexões de novos clientes na porta especificada e atuará como um mediador entre os clientes.

Ao receber uma nova conexão, ele irá obter o nome do cliente (única palavra) e imprimir na tela (stdout = saída padrão) a seguinte mensagem:

<HOUR>:<MINUTES> \t <CLIENT_NAME> \t Conectado

Um cliente fica conectado ao servidor até que ele (o cliente) sai do chat, ou que o servidor termina. Cada vez que o servidor recebe um comando de um cliente, a seguinte mensagem será impressa na tela:

<TIME> \t <CLIENT_NAME> \t <COMMAND> \t Executado:<Sim\Não>

O servidor irá suportar os seguintes comandos a partir dos clientes:

Comando	Argumentos	Descrição
SEND	<MESSAGE>	Envia <CLIENTS_NAME>: <MESSAGE> para todos os clientes conectados (menos o cliente emissor).
SENDTO	<CLIENT_NAME> <MESSAGE>	Idêntico com SEND, porém envia a mensagem apenas para o cliente especificado pelo <CLIENT_NAME>
WHO		Retorna a lista dos clientes conectados ao servidor
HELP		Retorna a lista de comandos suportados e seu uso

Quando um cliente se desconecta do servidor, o servidor imprime a seguinte mensagem para a tela <Time> \t <CLIENT_NAME> \t Desconectado.

A qualquer momento o usuário pode digitar CTRL-C no stdin (terminal) do servidor e o servidor irá desconectar todos os clientes e sair.

O servidor dever usar dois threads T1 e T2. T1 espera as conexões com accept(). T2 gerencia as conexões abertas usando a função 'select()' (http://linux.die.net/man/2/select_tut). Um pipe() p entre T1 e T2 permite informar T2 quando T1 aceitou uma nova conexão. Em tal evento, T2 acessa uma variável global (protegida por mutex) contendo o conjunto de conexões abertas e atualiza seu conjunto local.

Nota que este conjunto global deverá conter o descritor p[0] do pipe() de comunicação entre T1 e T2.

Parte 2: o cliente

O programa chamado " client_chat " irá receber 3 argumentos - nome do cliente, endereço do servidor e porta.

Uso: client_chat <CLIENT_NAME> <SERVER_ADDRESS> <SERVER_PORT>

Sob execução do cliente, ele tentará se conectar ao servidor especificado. Após o sucesso na conexão, o cliente deve imprimir "Conectado com sucesso" na tela do seu terminal e ficar aguardando comandos do stdin (teclado). Se um cliente tenta se conectar e CLIENT_NAME já for em uso, o cliente deve falhar a se conectar ao servidor.

O cliente irá suportar todos os comandos especificados acima. Um comando é definida como uma linha, ou seja, todo o texto digitado até ENTER for pressionado é considerado um único comando. Cada comando digitado é enviado para o servidor. Além disso, o cliente também deve reconhecer o comando CTRL-C, o que fará com que o cliente se desconecta do servidor e termina.

Quando o cliente recebe uma mensagem do servidor, ele a imprime imediatamente na tela. Assim como acontece com o servidor, o cliente deve gerenciar a conexão com o servidor e stdio usando 'select()'. No caso do termino do servidor, o comportamento do cliente é indefinido, mas deve se comportar razoavelmente.

Exemplo:

Server Input	Server Output	Client1 Input	Client1 Output	Client2 Input	Client2 Output
server_chat 12345					
		client_chat client1 localhost 12345			
	5:38 client1 Conectado		Conectado com sucesso		
				client_chat client2 localhost 12345	
	5:40 client2 Conectado				Conectado com sucesso
				SEND Ola pessoal!	
	5:45 client2 SEND Executado:Sim				
			client2: Ola pessoal!		

Um excelente tutorial: Network Programming - http://beej.us/guide/bgnet/

Dicas:

- Quando um socket está pronto para a leitura, pode usar um loop while para ler a mensagem inteira até que a leitura seja concluída.
- O tamanho de uma mensagem deve ser menor que 2000 bytes. Mas o sistema não deve falhar caso seja maior.

Note que o protocolo entre os clientes e o servidor é com você. Em outras palavras, vocês decidem exatamente o formato para usar quando passar a informação de um para o outro. Em particular, vocês devem decidir o que fazer se algum comando falhar. Diretrizes básicas são as seguintes:

- Qualquer componente que reconhece uma condição de erro deve imprimir uma mensagem de erro informativo para a sua tela. Isso deve começar com a palavra "ERRO". Se um erro do servidor também diz respeito ao cliente (por exemplo, quando um comando do cliente falhar), o cliente deve imprimi-lo também.
- Os programas não devem travar ou falhar, mesmo quando um dos lados não usa o mesmo protocolo.
- Não se esqueça de verificar o valor de retorno de todas as chamadas de sistema!

A entrega do trabalho acontecerá fisicamente no laboratório. Os seguintes arquivos deverão ser apresentados:

- README;
- O arquivos .c, com os devidos comentários;
- Makefile – Depois da execução do comando 'make', os programas 'server_chat' e 'client_chat' serão criados.

O arquivo README deverá incluir os seguintes detalhes:

- Nomes e endereços email.
- Comentários sobre o programa e o desenho do protocolo.

Última data de entrega: **20 de maio de 2016**