

## Trabalho 2 - Cliente e Servidor TCP (MC833)

José Ribeiro Neto - RA 176665

Utilize cliente.c e servidor.c disponibilizados para realizar este exercício.

1. Analise os códigos dos programas cliente.c e servidor.c e identifique as funções usadas para comunicação via socket. Procure nas páginas de manual do Linux, a descrição das funções que estão relacionadas ao uso de sockets. Procure também nos códigos a natureza dos parâmetros que cada programa deve receber, se for o caso.

R: Em suma, as seguintes chamadas de sistema são utilizadas:

- No Cliente: {socket(), connect(), read()}

- No Servidor: {socket(), bind(), listen(), laço infinito = {accept(), write(), close()}}

- (a) int socket(int domain, int type, int protocol): cria uma porta (endpoint) para comunicação e retorna um descritor de arquivo que referência a porta criada.

O argumento 'domain' especifica o domínio de comunicação. No caso da nossa aplicação cliente / servidor, utilizamos um 'domain = AF\_INET' (domínio = Internet); isto seleciona a família de protocolos nos quais serão utilizadas para a comunicação.

O argumento 'type', o qual especifica a semântica da comunicação. No caso da nossa aplicação cliente / servidor, utilizamos o 'SOCK\_STREAM', que indicará uma comunicação sequencial, confiável, de duplo sentido, e baseada em streams de bytes.

Já o argumento 'protocol' especifica um protocolo particular a ser utilizado com o socket. Como não fazemos uso de nenhum protocolo específico agregado ao socket, passamos o valor 0.

Em caso de sucesso, o descritor de arquivo para o novo socket será retornada pela syscall socket. Já em caso de erro, o valor -1 será retornado.

- (b) int connect(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen): essa chamada de sistema conecta o socket sendo referenciado por 'sockfd' ao endereço especificado pelo 'addr'. O campo 'addrlen' especifica o tamanho da estrutura 'addr'. O formato do endereço especificado pelo 'addr' será determinado pelo tipo de socket referenciado por 'sockfd'. Se a conexão é sucedida, então a syscall retorna 0, do contrário, -1 é retornado.
- (c) ssize\_t read(int fd, void \*buf, size\_t count): Tenta ler do descritor de arquivo 'fd' o número 'count' de bytes, os quais são escritos em 'buf'. Quando sucedido, o número de bytes lidos é retornado (sendo que 0 indica final de arquivo), a posição do arquivo avança essa mesma quantidade de bytes. Quando um erro acontece, o retorno será igual a -1.
- (d) int bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen): Quando o socket sendo referenciado por 'sockfd' é criado, o mesmo existe apenas no espaço de nomes (address family), contudo não possui endereço atribuído ao mesmo. A syscall bind() então atribui o endereço especificado por 'addr' ao socket referenciado por 'sockfd'. O

campo 'addrlen' especifica o tamanho, em bytes, da estrutura de endereço apontada por 'addr'. Em caso de sucesso, 0 é retornado, e em caso de erro, -1 é retornado.

- (e) `int listen(int sockfd, int backlog)`: Marca o socket referenciado pelo descritor de arquivo 'sockfd' como sendo um socket passivo, isto é, como sendo um socket que irá apenas aceitar conexões de entrada (requisições) através da syscall `accept()`. O argumento 'backlog' define o tamanho máximo no qual a fila de requisições pendentes poderá ter. Caso uma nova requisição chegue quando a fila já estiver cheia, então o cliente realizando a requisição poderá receber um erro. Quando houver sucesso na chamada do `listen()`, a syscall retornará 0, já no caso de erro, retornará -1.
- (f) `int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen)`: esta chamada de sistema é utilizada para sockets que são baseados em conexões. A mesma extrai a primeira conexão pendente da fila de conexões pendentes criada pelo socket passivo referenciado por 'sockfd', cria um novo socket conectado, e retorna o novo descritor de arquivo que referencia o socket criado. Tal socket criado encontra-se em um estado diferente do passivo (listening), sendo que o socket original 'sockfd' é inalterado pela chamada de tal syscall. O campo 'addr' é um ponteiro para a estrutura 'sockaddr' que conterá o endereço do socket do cliente. Já o campo 'addrlen' é o argumento que contém o tamanho em bytes da estrutura 'sockaddr' (ou de acordo com o tamanho da estrutura de 'addr'). Quando bem sucedida, esta syscall retorna um inteiro não negativo representando o descritor de arquivo do novo socket criado. Já em caso de erro, -1 é retornado.
- (g) `ssize_t write(int fd, const void *buf, size_t count)`: escreve até 'count' bytes do buffer de origem 'buf' no arquivo referenciado pelo descritor de arquivo 'fd'. Em caso de sucesso, o número de bytes escritos no arquivo é retornado pela syscall, e em caso de erro o valor -1 é retornado.
- (h) `int close(int fd)`: Esta syscall fecha um descritor de arquivo 'fd', de forma que a mesma não mais referencia qualquer arquivo. Caso o fechamento do descritor ocorre de forma correta, o valor 0 é retornado. Em caso de erro o valor -1 é retornado.

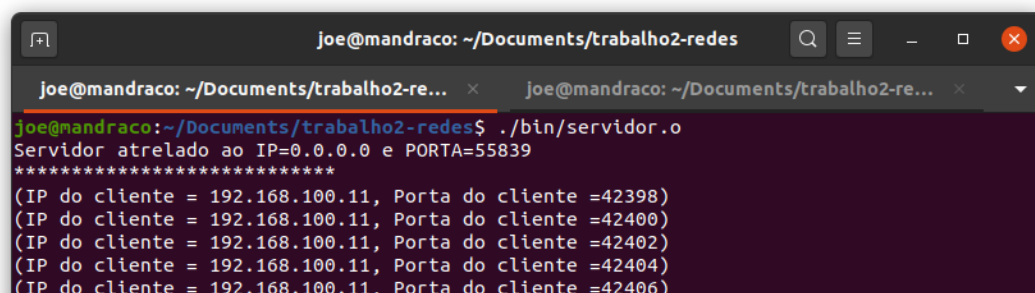
2. Compile e execute os programas cliente.c e servidor.c em uma mesma máquina. Houve algum erro? Em caso afirmativo, qual a sua causa? Se necessário, modifique os programas de forma que este erro seja corrigido e informe quais modificações foram realizadas. (Insira uma figura mostrando que o seu código executou sem erros)

R: Houve erro. Em verdade, o programa do servidor nem compilou. Isto pois, na linha `"servaddr.sin_addr.s_addr = htonl("192.168.0.16");"` é usado a função `htonl()` recebendo uma string ao invés de um `uint32_t`. Ou seja, o `htonl` converte inteiros (no caso, converte a ordem dos bytes do inteiro para estar em concordância com o utilizado na rede, que no caso é o big-endian) e não strings. Para resolver o problema, podemos substituir o `htonl` pela função `"inet_addr"` responsável por converter strings na notação de IPV4 para binários com ordem de bytes especificado pela rede (big-endian). Também, poderíamos substituir a string pelo campo `"INADDR_ANY"`, que faria com que o servidor atrelasse seu socket em todas as interfaces do host.

3. Altere o código do servidor para que seja automatizado a escolha da porta e utilize sempre o IP da máquina que está sendo executado.

R: Para que o servidor automatize a escolha do IP, podemos utilizar a seguinte linha: `"servaddr.sin_addr.s_addr = htonl(INADDR_ANY);"`. Desta forma, o servidor passará a escutar requisições em todas as interfaces de rede presente no host onde está executando, portanto

sempre utilizará o IP da máquina que está sendo executado. Agora para automatizar a escolha da porta, basta que setemos a linha: "servaddr.sin\_port = htons(0);". Assim, o servidor passará a escutar numa porta que esteja disponível na interface de rede. Para facilitar que o usuário saiba qual é esta porta, a mesma foi printada na saída padrão (stdout) do servidor. A imagem 1 abaixo mostra, na primeira linha, a porta que o servidor escolheu automaticamente, sendo igual a 55839. A figura 2 mostra a forma como o cliente faz a requisição ao servidor com base na porta que o mesmo printou no stdout (55839), de forma que o primeiro argumento passado para o programa do cliente é o IP e o segundo argumento é a porta onde o servidor está escutando requisições.

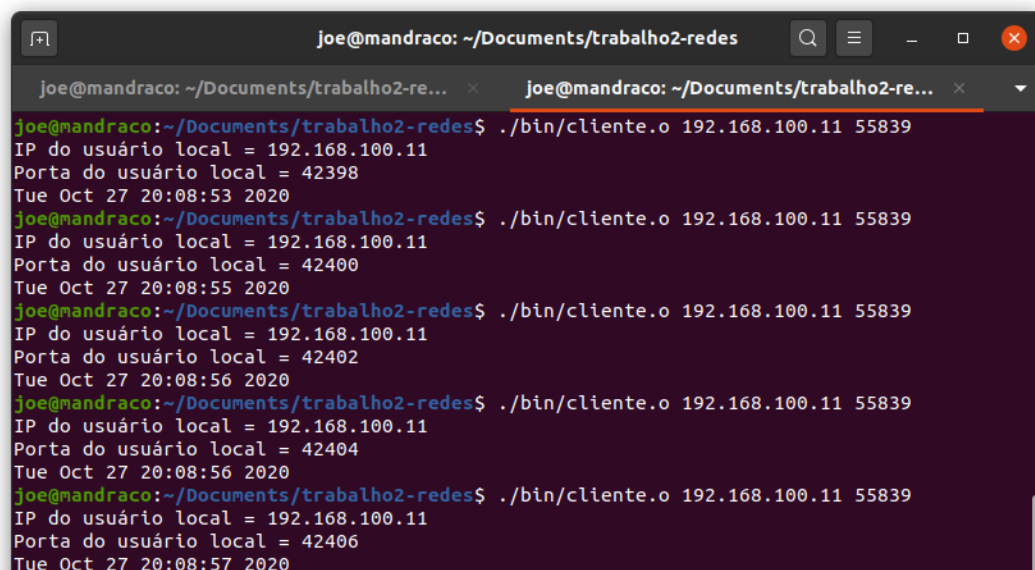


```

joe@mandraco: ~/Documents/trabalho2-redes
joe@mandraco: ~/Documents/trabalho2-re... x joe@mandraco: ~/Documents/trabalho2-re... x
joe@mandraco:~/Documents/trabalho2-redes$ ./bin/servidor.o
Servidor atrelado ao IP=0.0.0.0 e PORTA=55839
*****
(IP do cliente = 192.168.100.11, Porta do cliente =42398)
(IP do cliente = 192.168.100.11, Porta do cliente =42400)
(IP do cliente = 192.168.100.11, Porta do cliente =42402)
(IP do cliente = 192.168.100.11, Porta do cliente =42404)
(IP do cliente = 192.168.100.11, Porta do cliente =42406)

```

Figure 1: Resposta da questão 3



```

joe@mandraco: ~/Documents/trabalho2-redes
joe@mandraco: ~/Documents/trabalho2-re... x joe@mandraco: ~/Documents/trabalho2-re... x
joe@mandraco:~/Documents/trabalho2-redes$ ./bin/cliente.o 192.168.100.11 55839
IP do usuário local = 192.168.100.11
Porta do usuário local = 42398
Tue Oct 27 20:08:53 2020
joe@mandraco:~/Documents/trabalho2-redes$ ./bin/cliente.o 192.168.100.11 55839
IP do usuário local = 192.168.100.11
Porta do usuário local = 42400
Tue Oct 27 20:08:55 2020
joe@mandraco:~/Documents/trabalho2-redes$ ./bin/cliente.o 192.168.100.11 55839
IP do usuário local = 192.168.100.11
Porta do usuário local = 42402
Tue Oct 27 20:08:56 2020
joe@mandraco:~/Documents/trabalho2-redes$ ./bin/cliente.o 192.168.100.11 55839
IP do usuário local = 192.168.100.11
Porta do usuário local = 42404
Tue Oct 27 20:08:56 2020
joe@mandraco:~/Documents/trabalho2-redes$ ./bin/cliente.o 192.168.100.11 55839
IP do usuário local = 192.168.100.11
Porta do usuário local = 42406
Tue Oct 27 20:08:57 2020

```

Figure 2: Resposta da questão 3

4. Liste as chamadas de sistema necessárias para um servidor escutar conexões futuras. Justifique.  
R: Primeiro chama a syscall socket() para criar o meu socket TCP. Depois chama a syscall

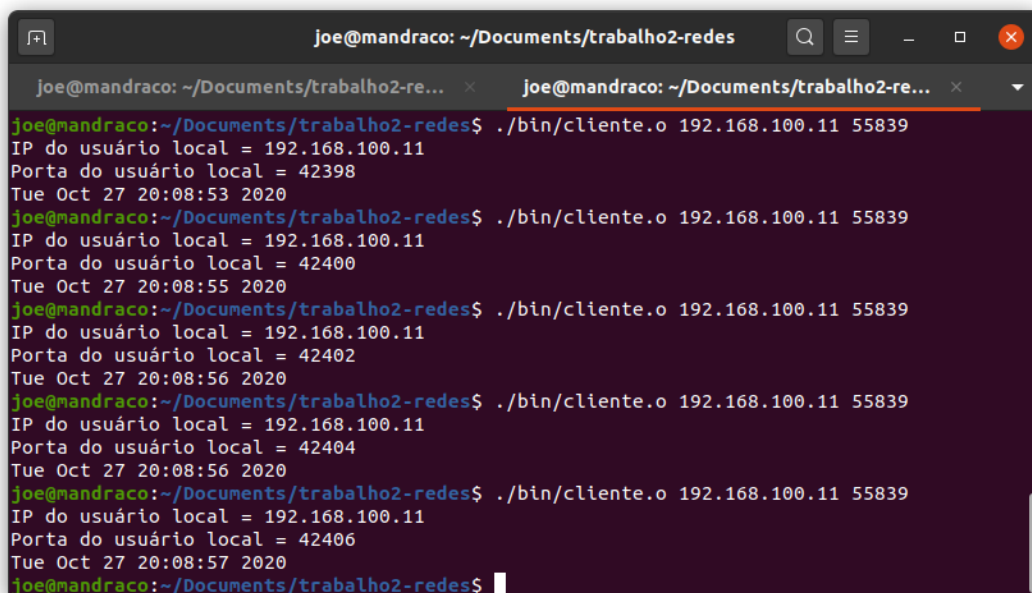
bind() para atrelar o socket criado acima a uma interface (IP) e a uma porta (caso especificada, do contrário, setada aleatória). Por fim, chama a syscall listen() para converter o socket criado acima em um socket que ficará monitorando requests na porta e interface ao qual está atrelado.

5. Adicione comentários necessários ao código. (Não precisa anexar esta questão no relatório. Os comentários serão analisados no código enviado).

R:

6. Modifique o programa cliente.c para que ele obtenha as informações do socket local ("#" IP, "#" porta local) através da função getsockname().

R: Como podemos perceber pela imagem 3 abaixo, quando o cliente é executado, utilizamos a função getsockname() para obter o IP do cliente e a porta no qual o socket sendo utilizado para fazer a requisição ao servidor está atrelado. No caso da imagem, temos que o IP do cliente é dado por "192.168.100.11" e temos que a porta no qual o socket foi atrelado sendo igual a {42398, 42400, 42402, 42404, 42406} (valores variando conforme a disponibilidade de portas).

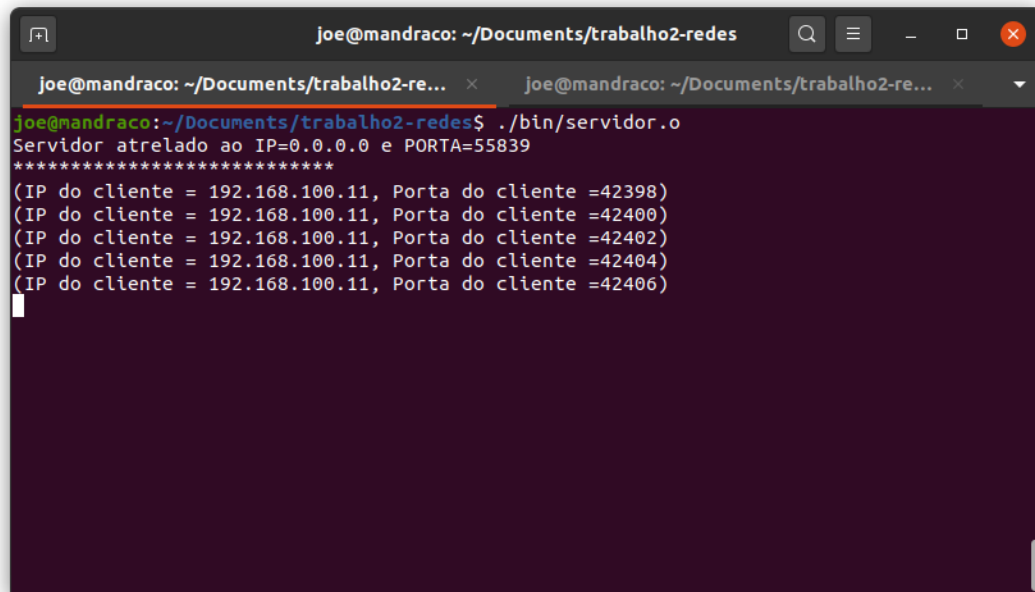


```
joe@mandraco: ~/Documents/trabalho2-redes
joe@mandraco: ~/Documents/trabalho2-redes$ ./bin/cliente.o 192.168.100.11 55839
IP do usuário local = 192.168.100.11
Porta do usuário local = 42398
Tue Oct 27 20:08:53 2020
joe@mandraco: ~/Documents/trabalho2-redes$ ./bin/cliente.o 192.168.100.11 55839
IP do usuário local = 192.168.100.11
Porta do usuário local = 42400
Tue Oct 27 20:08:55 2020
joe@mandraco: ~/Documents/trabalho2-redes$ ./bin/cliente.o 192.168.100.11 55839
IP do usuário local = 192.168.100.11
Porta do usuário local = 42402
Tue Oct 27 20:08:56 2020
joe@mandraco: ~/Documents/trabalho2-redes$ ./bin/cliente.o 192.168.100.11 55839
IP do usuário local = 192.168.100.11
Porta do usuário local = 42404
Tue Oct 27 20:08:56 2020
joe@mandraco: ~/Documents/trabalho2-redes$ ./bin/cliente.o 192.168.100.11 55839
IP do usuário local = 192.168.100.11
Porta do usuário local = 42406
Tue Oct 27 20:08:57 2020
joe@mandraco: ~/Documents/trabalho2-redes$
```

Figure 3: Resposta da questão 6

7. Modifique o programa servidor.c para que este obtenha as informações do socket remoto do cliente ("#" IP remoto, "#" porta remota), utilizando a função getpeername(). Imprima esses valores na saída padrão.

R: No código do servidor, utilizamos o getpeername após a syscall accept(), objetivando obter o IP do cliente e a porta do socket na qual seu socket está atrelado. Assim, podemos ver na imagem 4 abaixo, que o servidor printa no stdout o IP do cliente "192.168.100.11" assim como a porta no qual o mesmo está utilizando para fazer as requisições ao servidor (no caso, "42398", "42400", "42406").

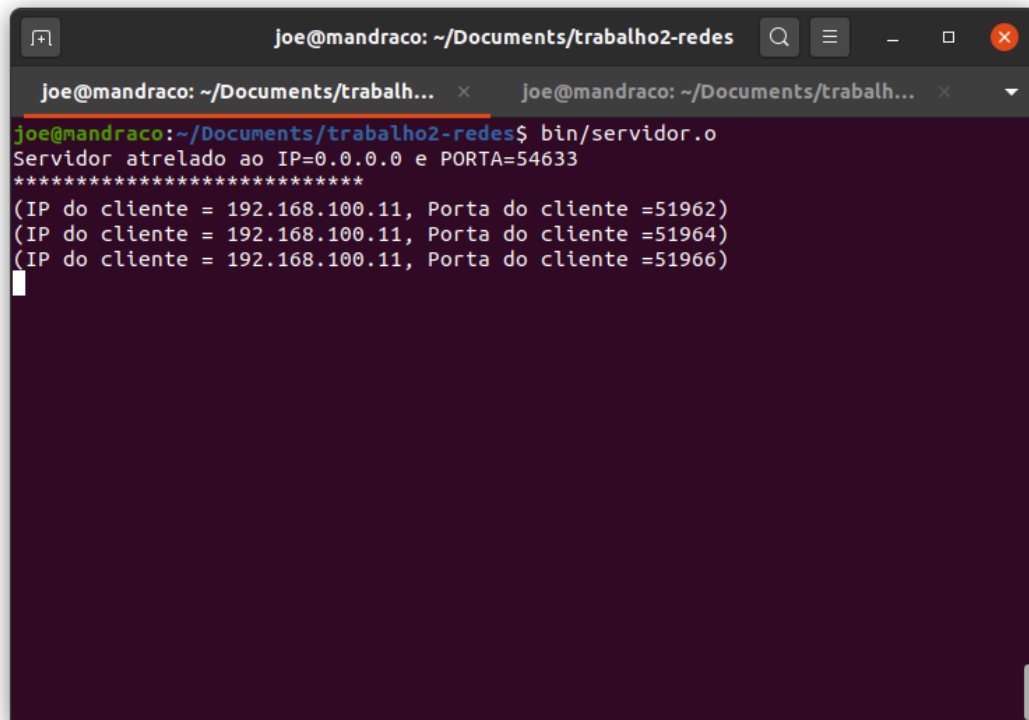
A terminal window with a dark purple background. The title bar shows 'joe@mandraco: ~/Documents/trabalho2-redes'. The terminal content shows a command prompt where the user has run './bin/servidor.o'. The output indicates the server is listening on IP 0.0.0.0 and port 55839. It then lists five incoming connections from IP 192.168.100.11 with ports 42398, 42400, 42402, 42404, and 42406. A cursor is visible on the line following the last connection.

```
joe@mandraco: ~/Documents/trabalho2-redes
joe@mandraco:~/Documents/trabalho2-redes$ ./bin/servidor.o
Servidor atrelado ao IP=0.0.0.0 e PORTA=55839
*****
(IP do cliente = 192.168.100.11, Porta do cliente =42398)
(IP do cliente = 192.168.100.11, Porta do cliente =42400)
(IP do cliente = 192.168.100.11, Porta do cliente =42402)
(IP do cliente = 192.168.100.11, Porta do cliente =42404)
(IP do cliente = 192.168.100.11, Porta do cliente =42406)
█
```

Figure 4: Resposta da questão 7

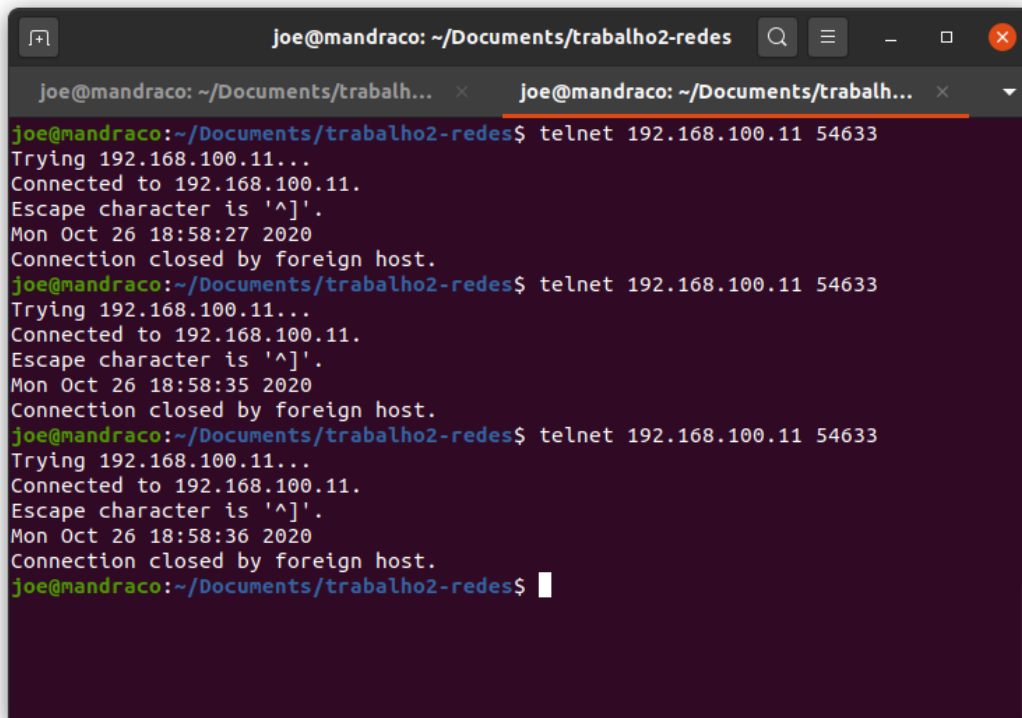
8. É possível usar o programa telnet no lugar do binário do cliente.c? Justifique e comprove sua resposta.

R: Podemos. Isto pois, o telnet (representando cliente) é um programa utilizado para realizar comunicação interativa com outro host (representando o servidor) usando para tanto o protocolo TELNET. Para realização de tal conexão, basta que especifiquemos o endereço do host (servidor) e a porta que o mesmo estará escutando (porta do servidor).



```
joe@mandraco: ~/Documents/trabalho2-redes
joe@mandraco: ~/Documents/trabalho2-redes$ bin/servidor.o
Servidor atrelado ao IP=0.0.0.0 e PORTA=54633
*****
(IP do cliente = 192.168.100.11, Porta do cliente =51962)
(IP do cliente = 192.168.100.11, Porta do cliente =51964)
(IP do cliente = 192.168.100.11, Porta do cliente =51966)
█
```

Figure 5: Servidor Telnet - Resposta da questão 7



```
joe@mandraco: ~/Documents/trabalho2-redes
joe@mandraco: ~/Documents/trabalho2-redes$ telnet 192.168.100.11 54633
Trying 192.168.100.11...
Connected to 192.168.100.11.
Escape character is '^]'.
Mon Oct 26 18:58:27 2020
Connection closed by foreign host.
joe@mandraco:~/Documents/trabalho2-redes$ telnet 192.168.100.11 54633
Trying 192.168.100.11...
Connected to 192.168.100.11.
Escape character is '^]'.
Mon Oct 26 18:58:35 2020
Connection closed by foreign host.
joe@mandraco:~/Documents/trabalho2-redes$ telnet 192.168.100.11 54633
Trying 192.168.100.11...
Connected to 192.168.100.11.
Escape character is '^]'.
Mon Oct 26 18:58:36 2020
Connection closed by foreign host.
joe@mandraco:~/Documents/trabalho2-redes$
```

Figure 6: Cliente Telnet - Resposta da questão 7

#### Instruções:

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'. Então o servidor printará na primeira linha a porta no qual ficará escutando requisições (PORTA=X). O IP do servidor também será printado nesta linha (IP=Y).
  - (b) Depois de executar o servidor, executar o comando './bin/cliente.o Y X' pra rodar o programa do cliente. Aqui, 'X' representa a porta que o servidor está escutando (printada na primeira linha do stdout do programa servidor) e 'X' representa o IP que o servidor estará escutando requisições (também printado na primeira linha do stdout do servidor).