

# Exercício 3 de MC833 — Programação em Redes de Computadores

Raul Rabelo Carvalho, 105607, turma A

19 de Março de 2014

## 1 htons

A função `htons` faz parte de um conjunto de funções usadas para converter um inteiro, tanto de 16 bits quanto de 32 bits, entre a *byte order* de rede (*big endian*) e a *byte order* do *host* local, que pode ser *big endian* ou *little endian* a depender da arquitetura. É óbvio que caso a arquitetura do *host* seja *big endian*, nada é feito.

Especificamente, a função `htons` converte inteiros de 16 bits da *byte order* do *host* para da rede.

## 2 Execução com código-fonte não-modificado

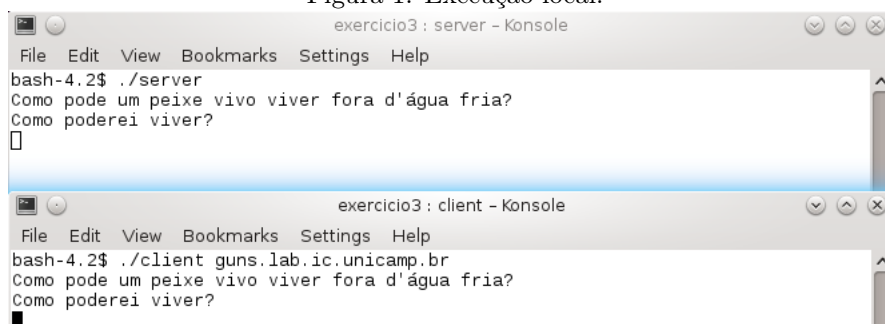
Não é possível executar o servidor sem alterações no *host* do laboratório do IC (`guns.ic.unicamp.br`), pois a função `bind` não tem permissão para utilizar a porta 10. Esta porta é uma das *well-known ports* e estão bloqueadas para usuários sem privilégios nas máquinas do IC.

## 3 Resolvendo o problema do bind

Alterando-se a porta empregada para 1234, tanto em `client.c` quanto em `server.c`, resolve-se o problema do servidor não conseguir reservar para si uma das *well-known ports*. No entanto, o cliente não consegue se conectar a esta porta, provavelmente devido à alguma política de bloqueio de portas no roteador dos laboratórios.

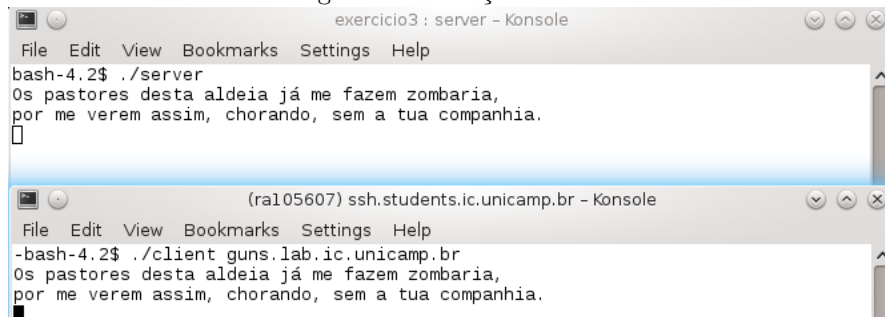
Fazendo uma segunda alteração para a porta 10000, o servidor consegue reservar a porta e executar normalmente, e o cliente consegue conectar-se a esta porta e comunicar-se com o servidor, como na figura 1.

Figura 1: Execução local.



Também foi possível executar o servidor no *host* local e o cliente na máquina *xaveco* via *ssh*, executado com o comando `./client guns.lab.ic.unicamp.br`.

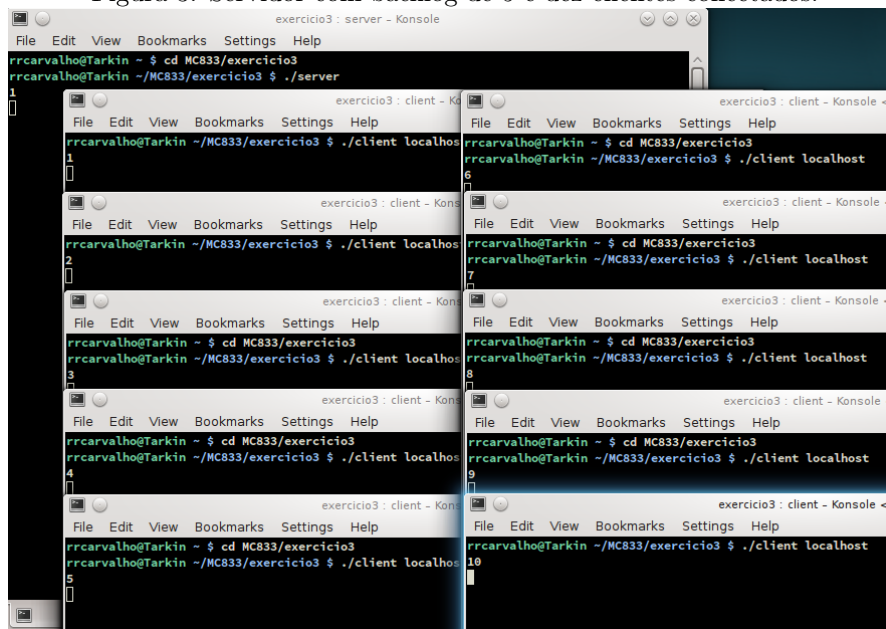
Figura 2: Execução remota.



## 4 Múltiplos clientes

O comportamento dos clientes e do servidor quando dez clientes tentam se conectar ao servidor, quando este está configurado com um *backlog* de cinco está mostrado nas figuras 3, 4 e 5. O que ocorre é que além do primeiro cliente cuja conexão foi aceita, mais seis conexões foram sendo aceitas a medida em que as anteriores eram encerradas. A sexta conexão além das cinco em que o servidor configurou o *socket* pode ser devido à implementação deste no sistema, pois a maioria dos sistemas operacionais tomam o parâmetro *backlog* como uma indicação e não como uma regra forte a ser seguida<sup>1</sup>

Figura 3: Servidor com backlog de 5 e dez clientes conectados.



<sup>1</sup>Stevens, Fenner, Rudoff, "Unix Network Programming: The Sockets Network API", Volume 1, Third Edition, pp. 106-108.

Figura 4: Servidor com backlog de 5 após o encerramento do primeiro dos dez clientes.

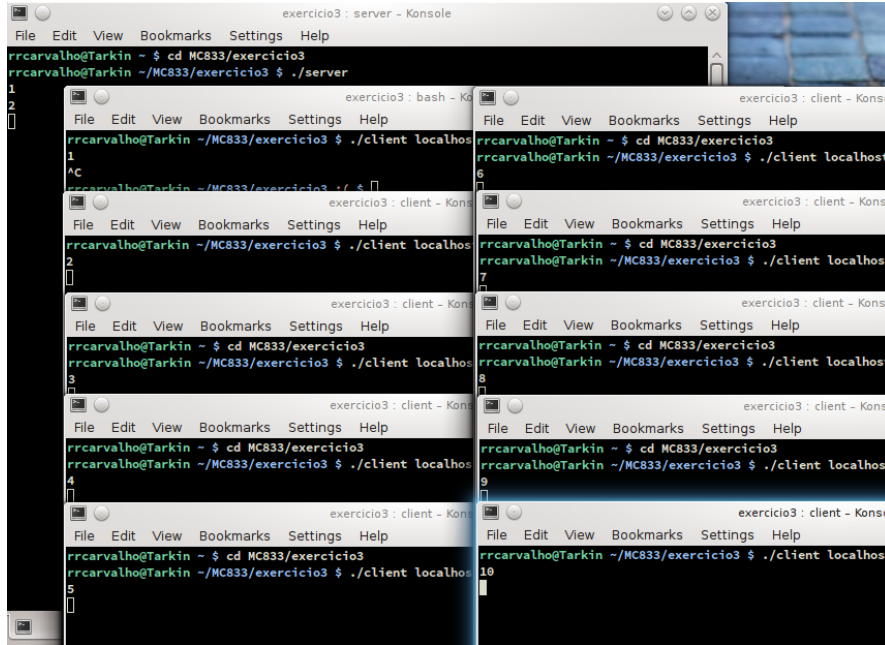
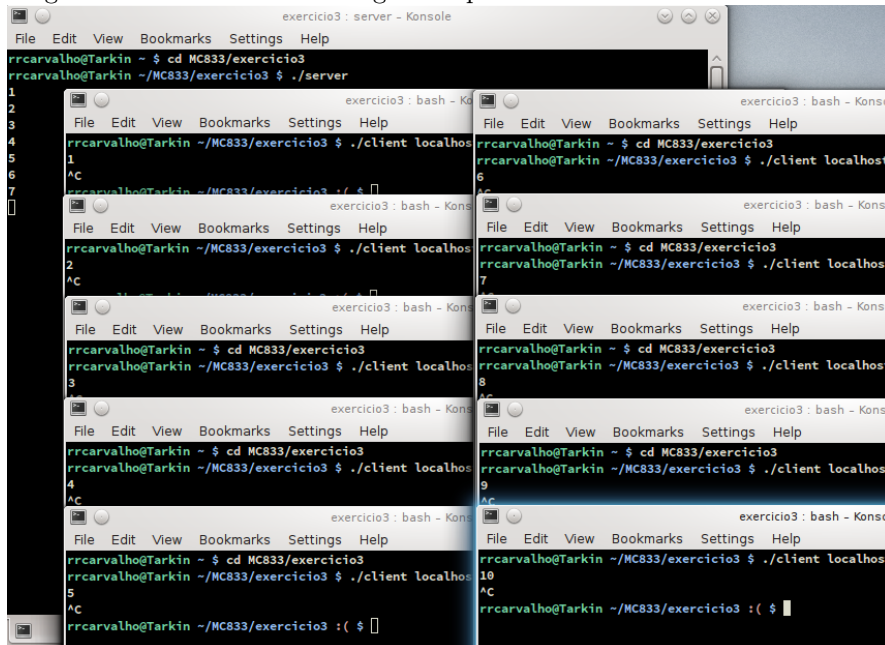


Figura 5: Servidor com backlog de 5 após o encerramento dos dez clientes.



O *backlog* do servidor foi mudado para um e o resultado de dez conexões é mostrado nas figuras 6, 7 e 8. Ocorre novamente que um cliente extra além do

cliente 2 que estava na fila de tamanho um agora conseguiu se conectar; e, além disso, o cliente 10 também se conectou logo após o cliente 3 ser encerrado. Ocorre que o tempo gasto na obtenção das imagens não foi longo o suficiente para o *timeout* do cliente 10 ter se esgotado quando do encerramento do cliente 2, assim, ele foi capaz de entrar na fila tão logo o cliente 1 foi encerrado. Aguardando um tempo maior antes de encerrar as conexões (figura 9, o cliente 10 não consegue uma conexão.

Figura 6: Servidor com backlog de 1 e dez clientes conectados.

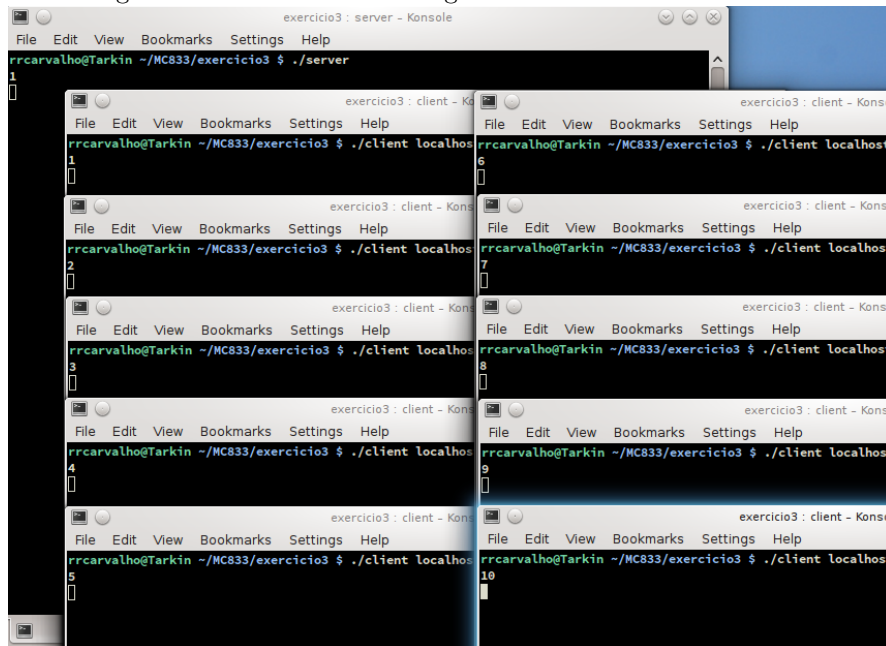


Figura 7: Servidor com backlog de 1 após o encerramento do primeiro dos dez clientes.

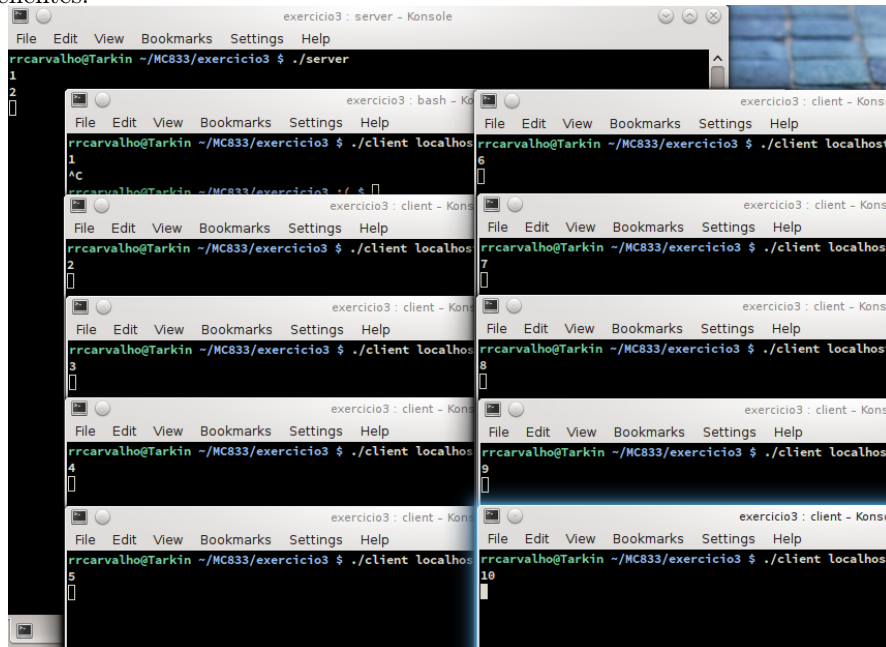


Figura 8: Servidor com backlog de 1 após o encerramento dos dez clientes.

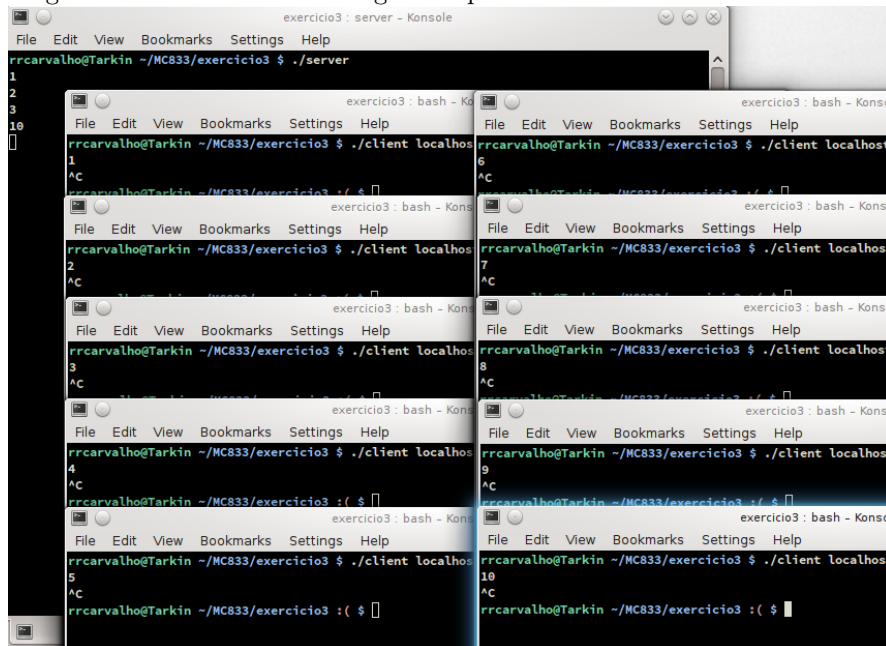
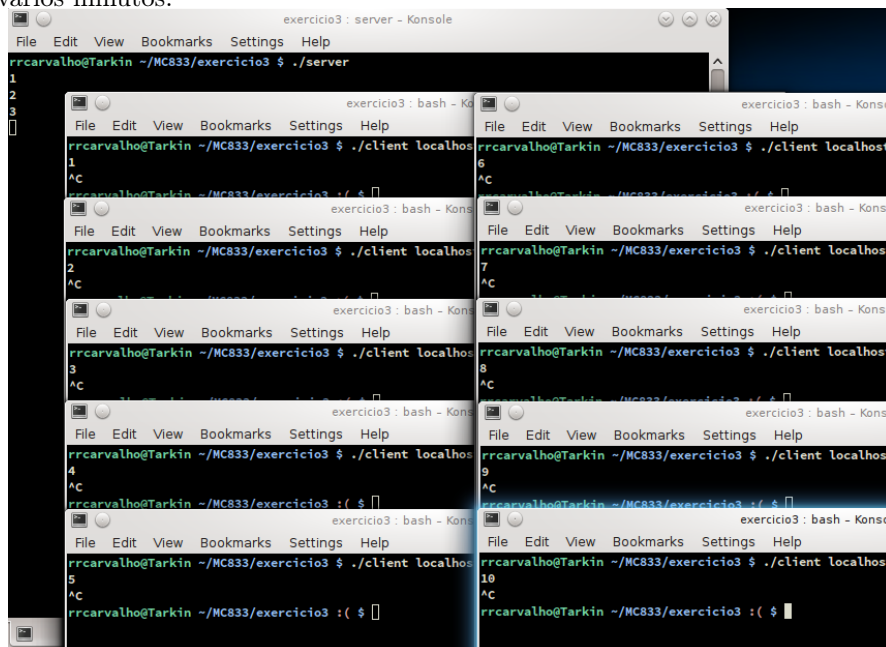


Figura 9: Servidor com backlog de 1 com os 10 clientes encerrados depois de vários minutos.

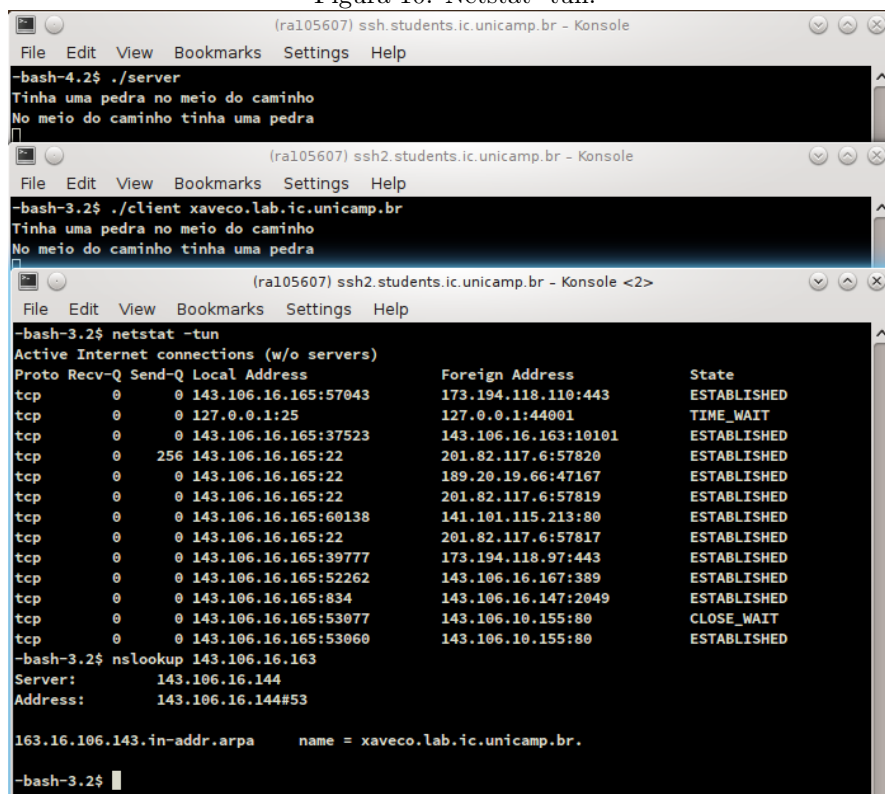


## 5 Verificando o uso da rede

Pode-se configurar a porta usada no servidor para uma porta disponível no *host* em que ele vá ser executado. Com conhecimento desta porta, se estabelece a conexão entre cliente e servidor normalmente. Em um terceiro **terminal**, executa-se o comando **netstat -tu** e, se a comunicação entre cliente e servidor se faz por rede, uma conexão estabelecida com endereço IP do *host* do servidor e a porta TCP empregada estará listada na saída desta ferramenta.

Como mostrado na figura 10, existe uma conexão estabelecida entre o servidor em `xaveco.lab.ic.unicamp.br`, usando a porta 10101. Foi usado o comando **netstat -tun**, pois a porta 10101, apesar de estar acima de 1024, tem uma aplicação conhecida associada a ela. A conexão em questão está listada na terceira linha da saída do comando **netstat** e um *reverse nslookup* foi feito para se confirmar que o endereço IP 143.106.16.163 corresponde ao nome `xaveco.lab.ic.unicamp.br`.

Figura 10: Netstat -tun.



```
(ra105607) ssh.students.ic.unicamp.br - Konsole
File Edit View Bookmarks Settings Help
-bash-4.2$ ./server
Tinha uma pedra no meio do caminho
No meio do caminho tinha uma pedra

(ra105607) ssh2.students.ic.unicamp.br - Konsole
File Edit View Bookmarks Settings Help
-bash-3.2$ ./client xaveco.lab.ic.unicamp.br
Tinha uma pedra no meio do caminho
No meio do caminho tinha uma pedra

(ra105607) ssh2.students.ic.unicamp.br - Konsole <2>
File Edit View Bookmarks Settings Help
-bash-3.2$ netstat -tun
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 143.106.16.165:57043    173.194.118.110:443     ESTABLISHED
tcp        0      0 127.0.0.1:25          127.0.0.1:44001        TIME_WAIT
tcp        0      0 143.106.16.165:37523    143.106.16.163:10101    ESTABLISHED
tcp        0 256 143.106.16.165:22      201.82.117.6:57820      ESTABLISHED
tcp        0      0 143.106.16.165:22      189.20.19.66:47167      ESTABLISHED
tcp        0      0 143.106.16.165:22      201.82.117.6:57819      ESTABLISHED
tcp        0      0 143.106.16.165:60138    141.101.115.213:80      ESTABLISHED
tcp        0      0 143.106.16.165:22      201.82.117.6:57817      ESTABLISHED
tcp        0      0 143.106.16.165:39777    173.194.118.97:443      ESTABLISHED
tcp        0      0 143.106.16.165:52262    143.106.16.167:389      ESTABLISHED
tcp        0      0 143.106.16.165:834      143.106.16.147:2049     ESTABLISHED
tcp        0      0 143.106.16.165:53077    143.106.10.155:80       CLOSE_WAIT
tcp        0      0 143.106.16.165:53060    143.106.10.155:80       ESTABLISHED
-bash-3.2$ nslookup 143.106.16.163
Server:      143.106.16.144
Address:     143.106.16.144#53

163.16.106.143.in-addr.arpa    name = xaveco.lab.ic.unicamp.br.
-bash-3.2$
```

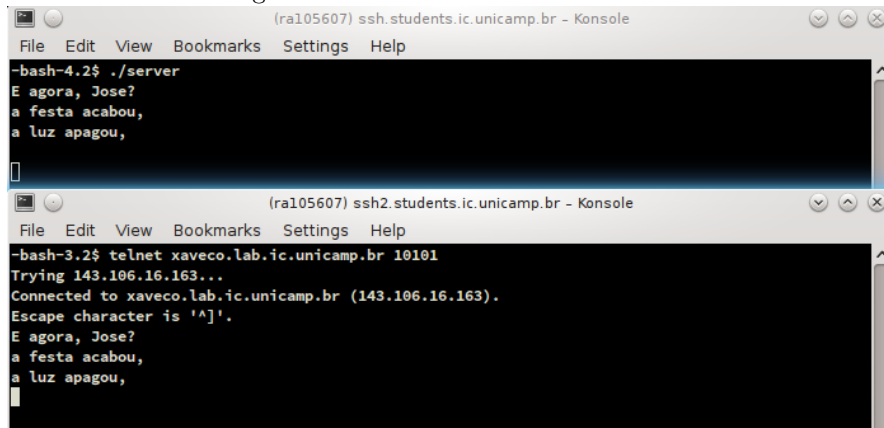
## 6 telnet

A figura 11 mostra o programa **telnet** conectando-se e enviando mensagens ao servidor. Como o **telnet** usa o protocolo TCP e pode conectar-se a qualquer



porta, não só a porta 23 do protocolo Telnet, um modo de impedir o acesso ao servidor é modificá-lo para receber pacotes UDP, o que implicaria em uma mudança no cliente também.

Figura 11: Usando telnet como cliente.



## 7 echo

Para a implementação da funcionalidade de *echo* no cliente e no servidor, foram modificadas algumas linhas no *loop* principal dos programas. Em *client.c*:

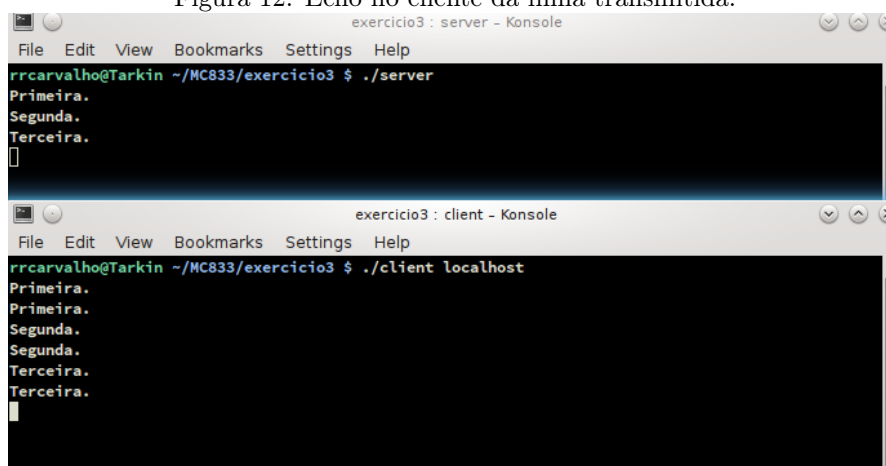
```
while (fgets(buf, sizeof(buf), stdin)) {
    buf[MAX_LINE-1] = '\0';
    len = strlen(buf) + 1;
    send(s, buf, len, 0);
    while (len = recv(s, buf, sizeof(buf), 0)) {
        fputs(buf, stdout);
        if (len) break;
    }
}
```

E em *server.c*:

```
while(1) {
    if ((new_s = accept(s, (struct sockaddr *)&sin, &len)) < 0) {
        perror("simplex-talk: accept");
        exit(1);
    }
    while (len = recv(new_s, buf, sizeof(buf), 0)) {
        fputs(buf, stdout);
        send(new_s, buf, len, 0);
    }
    close(new_s);
}
```

A figura 12 mostra a saída dos programas modificados.

Figura 12: Echo no cliente da linha transmitida.



```
exercicio3 : server - Konsole
File Edit View Bookmarks Settings Help
rrcarvalho@Tarkin ~/MC833/exercicio3 $ ./server
Primeira.
Segunda.
Terceira.
█

exercicio3 : client - Konsole
File Edit View Bookmarks Settings Help
rrcarvalho@Tarkin ~/MC833/exercicio3 $ ./client localhost
Primeira.
Primeira.
Segunda.
Segunda.
Terceira.
Terceira.
█
```