MC833 - 2s2017
Bruno Orsi Berton **RA 150573 - Turma B**
Fábio Takahashi Tanniguchi **RA 145980 - Turma A**

1) Explique o funcionamento das funções inet_pton, htons e htonl.

inet_pton: converte um endereço IP em formato legível (string) em uma estrutura que pode ser utilizada em funções referentes a rede (sockaddr_in, caso IPv4, ou sockaddr_in6, caso IPv6). O nome da função se refere a "**p**rintable **to n**etwork", que caracteriza sua utilidade.

htons e **htonl** são funções que convertem a ordem dos bytes da sua máquina (*big endian* ou *little endian*) para a ordenação de bytes que é usada na rede por padrão (*big endian*). No caso, **htons h**ost **to n**etwork **s**hort, converte para a byte order da rede os 16 bits que são byte order da máquina local e o **htonl h**ost **to n**etwork **l**ong, converte para a byte order da rede os 32 bits que são byte order da máquina local.

2) Compile e execute os programas cliente.c e servidor.c em uma mesma máquina. A função bind reportou 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.

A compilação dos programas ocorreu sem problemas.

Ao executar o servidor, o seguinte erro é exibido: **bind: Permission denied**. Este erro acontece porque o servidor está tentando fazer o bind da porta "13", o que pertence as **well-known ports** que são as portas com números de 0 a 1023, e um usuário comum não consegue fazer o bind com qualquer uma dessas portas. Substituindo esse valor por algum qualquer fora deixa faixa, o servidor passa a funcionar.

Ao executar o cliente, o seguinte erro é exibido: **inet_pton error: Success.** Este erro acontece porque o cliente original estava tentando se conectar a um servidor na porta "13". Substituindo essa porta pela mesma escolhida pelo servidor, foi possível executar o cliente com sucesso.

3) Com as modificações feitas, re-execute os programas cliente e servidor em uma mesma máquina e em máquinas diferentes. Quais as saídas dos programas? Foi possível fazer todas enviar/receber dados executando em máquinas distintas? Se sim, passe para o próximo exercício, se não corrija o código de modo que seja possível a comunicação entre máquinas diferentes.

O cliente se conecta com o servidor, e o servidor envia de volta a sua hora para o cliente. Em máquinas distintas inicialmente não estava funcionando.

```
bberton@bberton-Latitude-3440:~/Documents/mc833/exercicio_3$ ./cliente 127.0.0.1
Mon Sep 11 21:42:16 2017
```

Foi necessário alterar o servidor, alterando a linha **htonl(INADDR_LOOPBACK);** para **INADDR_ANY;** pois o INADDR_LOOPBACK faz com que apenas conexões da máquina que executa o servidor sejam aceitas, e o INADDR_ANY faz com o que o servidor aceite conexões de qualquer endereço IP. Referência: http://www.delorie.com/gnu/docs/glibc/libc_317.html

E esta linha foi adicionada ao cliente para conseguir se conectar ao servidor.

servaddr.sin_addr.s_addr = inet_addr(argv[1]);

Outra alteração foi a inclusão de mensagens de output para comprovar o recebimento e envio das mensagens.

```
printf("Recebeu mensagem\n");
ticks = time(NULL);
snprintf(buf, sizeof(buf), "%.24s\r\n", ctime(&ticks));
write(connfd, buf, strlen(buf));
printf("Enviou mensagem %s\n", buf);
```

```
bberton@bberton-Latitude-3440:~/Documents/mc833/exercicio_3$ ./cliente 177.220.84.25
Mon Sep 11 21:56:32 2017
bberton@bberton-Latitude-3440:~/Documents/mc833/exercicio_3$ ./cliente 177.220.84.25
Mon Sep 11 21:56:35 2017
bberton@bberton-Latitude-3440:~/Documents/mc833/exercicio_3$ ./cliente 177.220.84.25
Mon Sep 11 21:56:38 2017
```

```
ftt@FTT-ASUS-S46:~/UNICAMP/MC833/ex3$ ./servidor
Recebeu mensagem
Enviou mensagem Mon Sep 11 21:55:55 2017

Recebeu mensagem
Enviou mensagem Mon Sep 11 21:56:32 2017

Recebeu mensagem
Enviou mensagem Mon Sep 11 21:56:35 2017

Recebeu mensagem
Enviou mensagem Mon Sep 11 21:56:38 2017
```

4) Através de ferramentas existentes no sistema operacional, como você comprova, durante a execução em máquinas diferentes, que os códigos estão realizando uma comunicação via rede?

Utilizando o comando tcpdump -i wlan0 host 177.220.84.90 é possível visualizar todos os pacotes que as duas máquinas trocam. Nos prints abaixo, é possível ver claramente o *three-way handshake*, a troca de mensagem e o encerramento da conexão TCP.

O IP da máquina cliente é: **177.220.84.90**

```
wlp6s0    Link encap:Ethernet  HWaddr 9c:ad:97:fd:fe:35
          inet addr:177.220.84.90  Bcast:177.220.85.255  Mask:255.255.254.0
          inet6 addr: fe80::4025:c291:5ac9:df33/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:68025 errors:0 dropped:0 overruns:0 frame:0
          TX packets:47175 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:45185069 (45.1 MB)  TX bytes:9733231 (9.7 MB)

bberton@bberton-Latitude-3440:~/Documents/mc833/exercicio_3$
```

E o IP da máquina servidor é: **177.220.84.25**

```
ftt@FTT-ASUS-S46:~/UNICAMP/MC833/ex3$ sudo tcpdump -i wlan0 host 177.220.84.90
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on wlan0, link-type EN10MB (Ethernet), capture size 262144 bytes
22:10:40.313355 IP wifi-177-220-84-90.wifi.ic.unicamp.br.45678 > FTT-ASUS-S46.13000: Flags [S],
seq 2018321854, win 29200, options [mss 1386,sackOK,TS val 1498284303 ecr 0,nop,wscale 7], lengt
h 0
22:10:40.313470 IP FTT-ASUS-S46.13000 > wifi-177-220-84-90.wifi.ic.unicamp.br.45678: Flags [S.],
 seq 164576015, ack 2018321855, win 28960, options [mss 1460,sackOK,TS val 3224404059 ecr 149828
4303,nop,wscale 7], length 0
22:10:40.317343 IP wifi-177-220-84-90.wifi.ic.unicamp.br.45678 > FTT-ASUS-S46.13000: Flags [.],
ack 1, win 229, options [nop,nop,TS val 1498284306 ecr 3224404059], length 0
22:10:40.318550 IP FTT-ASUS-S46.13000 > wifi-177-220-84-90.wifi.ic.unicamp.br.45678: Flags [P.],
 seq 1:27, ack 1, win 227, options [nop,nop,TS val 3224404061 ecr 1498284306], length 26
22:10:40.318647 IP FTT-ASUS-S46.13000 > wifi-177-220-84-90.wifi.ic.unicamp.br.45678: Flags [F.],
 seq 27, ack 1, win 227, options [nop,nop,TS val 3224404061 ecr 1498284306], length 0
22:10:40.321406 IP wifi-177-220-84-90.wifi.ic.unicamp.br.45678 > FTT-ASUS-S46.13000: Flags [.],
ack 27, win 229, options [nop,nop,TS val 1498284307 ecr 3224404061], length 0
22:10:40.370716 IP wifi-177-220-84-90.wifi.ic.unicamp.br.45678 > FTT-ASUS-S46.13000: Flags [F.],
 seq 1, ack 28, win 229, options [nop,nop,TS val 1498284307 ecr 3224404061], length 0
22:10:40.370830 IP FTT-ASUS-S46.13000 > wifi-177-220-84-90.wifi.ic.unicamp.br.45678: Flags [.],
ack 2, win 227, options [nop,nop,TS val 3224404074 ecr 1498284307], length 0
^C
8 packets captured
8 packets received by filter
0 packets dropped by kernel
```

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

```
bberton@bberton-Latitude-3440:~/Documents/mc833/exercicio_3$ ./servidor
Recebi uma conexão
IP address do cliente: 177.220.84.25
Porta do cliente: 41410

Recebi uma conexão
IP address do cliente: 127.0.0.1
Porta do cliente: 39378
Recebi uma conexão
IP address do cliente: 177.220.84.25
Porta do cliente: 41414


Recebi uma conexão
IP address do cliente: 177.220.84.25
Porta do cliente: 41416
```

Foi adicionado ao servidor o seguinte trecho de código:

```
peeraddr_len = sizeof(struct sockaddr);
if (getpeername(connfd, (struct sockaddr *) &peeraddr, &peeraddr_len) == -1) {
  perror("getpeername() failed");
  return -1;
}
printf("IP address do cliente: %s\n", inet_ntoa(peeraddr.sin_addr));
printf("Porta do cliente: %d\n", (int) ntohs(peeraddr.sin_port));
```

Que captura o endereço IP e a porta utilizada pelo cliente.

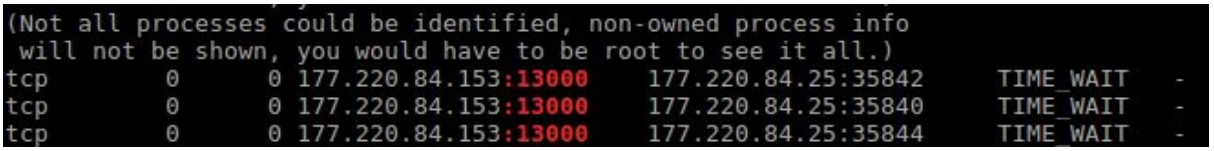
```
bberton@bberton-Latitude-3440:~/Documents/mc833/exercicio_3$ ./cliente 127.0.0.1
Mon Sep 11 22:45:42 2017
IP address do socket: 127.0.0.1
Client port do socket: 39378
```

Foi adicionado ao cliente o seguinte trecho de código:

```
servaddr_len = sizeof(struct sockaddr_in);
if (getsockname(sockfd, (struct sockaddr *) &servaddr, &servaddr_len) == -1) {
  perror("getsockname() failed");
  return -1;
}
printf("IP address do socket: %s\n", inet_ntoa(servaddr.sin_addr));
printf("Client port do socket: %d\n", (int) ntohs(servaddr.sin_port));
```

Que imprime os dados do socket.

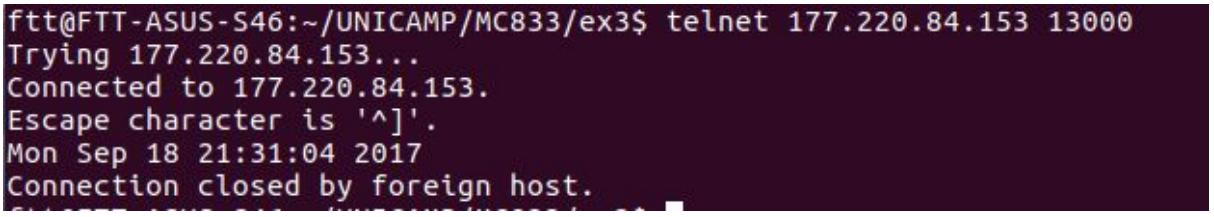
7) Mantenha o binário do servidor.c executando em uma máquina A e execute três vezes seguidas o binário do cliente.c em uma máquina B. Observando a saída do comando netstat, qual das duas máquinas (A ou B) ficam no estado TIME_WAIT? Explique porque a outra máquina não fica nesse estado.



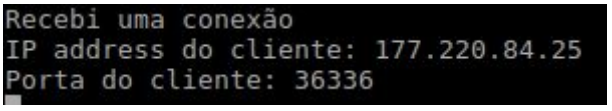
O cliente, a máquina B, fica no estado TIME_WAIT, pois um dos motivos para que o cliente fique nesse estado é evitar que pacotes atrasados da uma conexão possam ser mal interpretados em conexões futuras, e para que a conexão possa ser encerrada com sucesso mesmo que algum ACK do encerramento da conexão se perca.

8) O programa telnet pode ser usado no lugar do binário do client.c? Por que? Caso o telnet possa ser usado, copie a saída do servidor e da execução do telnet. Cite uma modificação no servidor.c que impediria a utilização do telnet caso a afirmação anterior seja verdadeira.

cliente.c (em 177.220.84.25)



servidor.c (em 177.220.84.153)



O telnet, como pode ser visto nas imagens acima, pode funcionar no lugar do *cliente.c* implementado, pois o cliente se resume ao estabelecimento de conexão com o servidor e recebimento de mensagem.

Uma possível mudança no servidor que impediria essa alternativa é o uso de protocolo UDP.