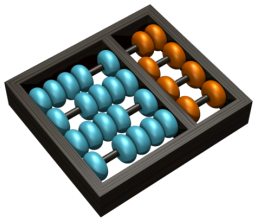

MC833 Relatório 3

Cliente e servidor TCP

Aluno: Fábio Camargo Ricci

RA: 170781

Instituto de Computação

Universidade Estadual de Campinas

Campinas, 19 de Setembro de 2021.

Sumário

1 Questões

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.

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)

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.

4. Liste as chamadas de sistema necessárias para um servidor escutar conexões futuras. Justifique.

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

6. 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.

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

2 Respostas

1. Funções usadas para comunicação via socket:

   (a) **socket**(int **domain**, int **type**, int **protocol**): Cria um endpoint de comunicação e retorna um file descriptor que referencia o mesmo.

      i. **domain**: Refere-se ao domínio de comunicação (a família de protocolos que será utilizada para comunicação).

      ii. **type**: Especifica as semânticas de comunicação.

      iii. **protocol**: Indica qual protocolo de comunicação será utilizado

   (b) **bind**(int **sockfd**, const struct sockaddr ***addr**, socklen_t **addrlen**): Relaciona um nome a um socket. Quando um socket é criado, o mesmo existe em uma família de endereços mas não possui um endereço atrelado. O método **bind()** atrela o endereço especificado por **addr** ao socket **sockfd**. **addrlen** indica o tamanho, em bytes, da estrutura apontada por **addr**.

   (c) **listen**(int **sockfd**, int **backlog**): Espera por conexões em um socket. O método marca o socket indicado por **sockfd** como um socket passivo, esperando conexões através de **accept()**. **backlog** indica quantas conexões pendentes para o determinado socket podem existir.

   (d) **accept**(int **sockfd**, struct sockaddr *restrict **addr**, socklen_t *restrict **addrlen**): Aceita uma conexão em um socket. **sockfd** se refere a um socket passivo que está escutando conexões, que foi criado com **socket()**, atrelado a um endereço com **bind()** e estava esperando por conxões com **listen()**. **addr** é um ponteiro

para um estrutura do tipo **sockaddr**, que é populada com o endereço do socket sendo conectado (par). **addrlen** é o tamanho, em bytes da estrutura **addr**. Esse método cria um novo socket conectado (baseado no socket passivo criado anteriormente) que estará atrelado à nova conexão e servirá como meio de comunicação.

(e) **connect**(int **sockfd**, const struct sockaddr ***addr**, socklen_t **addrlen**): Inicia uma conexão com um socket. Esse método conecta o socket descrito por **sockfd** ao endereço especificado por **addr**.

2. Não houve nenhum erro ao executar o cliente ou servidor. Compilou-se os programas com os comandos:
   gcc -o servidor servidor.c -Wall
   gcc -o cliente cliente.c -Wall

   As saídas obtidas foram:

```
[fabio@Fabios-MacBook-Pro Relatório 3 % ./cliente 127.0.0.1
Sun Sep 19 22:18:59 2021
fabio@Fabios-MacBook-Pro Relatório 3 %
```

```
[fabio@Fabios-MacBook-Pro Relatório 3 % gcc -o servidor servidor.c -Wall
[fabio@Fabios-MacBook-Pro Relatório 3 % ./servidor
```

3. Atribuindo a constante INADDR_ANY ao endereço do socket servidor (**servaddr.sin_addr.s_addr = htonl(INADDR_ANY)**), o mesmo passa a aceitar conexões em todos os endereços IPs disponíveis no host (após a chamada do método **bind()**)
   Para automatizar a escolha da porta, atribuiu-se o valor de **servaddr.sin_port**

= **0**, de modo que fica a cargo do sistema escolher uma porta disponível.

```
bzero(&servaddr, sizeof(servaddr));           // inicializa servaddr (pree
servaddr.sin_family = AF_INET;                // IPv4
servaddr.sin_addr.s_addr = htonl(INADDR_ANY); // aceita conxões em todos o
servaddr.sin_port = 0;                        // força o sistema a escolhe

if (bind(listenfd, (struct sockaddr *)&servaddr, sizeof(servaddr)) == -1)
{
    perror("bind");
    exit(1);
}
```

4. Para um servidor escutar conexões futuras, são necessárias quatro chamadas de sistema (descritas em detalhes na questão 1):

   (a) **socket**(int **domain**, int **type**, int **protocol**): Cria o socket que servirá como meio para a comunição.

   (b) **bind**(int **sockfd**, const struct sockaddr ***addr**, socklen_t **addrlen**): Atrela o socket acriado anteriormente a um endereço.

   (c) **listen**(int **sockfd**, int **backlog**): Espera por conexões futuras em um socket, marcando o mesmo como um socket passivo. Possui uma fila de conexões pendentes com tamanho de **backlog**.

   (d) **accept**(int **sockfd**, struct sockaddr *restrict **addr**, socklen_t *restrict **addrlen**): Aceita uma nova conexão no socket passivo descrito, criando um novo socket conectado que estará atrelado à nova conexão.

5. A função **getsockname()** recupera as informações do endereço do socket local (atribuído ao chamar **bind()** anteriormente). Com isso, utilizou-se a função **inet_ntop()** para obtermos a representação em string IPv4 do endereço obtido.

```c
// informação do socket servidor
struct sockaddr_in addr;
socklen_t len = sizeof(addr);
char name[128];

if (getsockname(listenfd, (struct sockaddr *)&addr, &len) < 0) //
{
    perror("getsocketname");
    exit(1);
}

if (inet_ntop(AF_INET, &(addr.sin_addr), name, sizeof(name)) < 0)
{
    perror("inet_ntop");
    exit(1);
}

printf("Listening on %s port %d\n", name, ntohs(addr.sin_port));
```

No cliente, alterou-se o código para aceitar o endereço IP e a porta que o servidor está escutando (previamente so aceitava o endereço IP, com a porta fixa).

```c
bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(atoi(argv[2]));
if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0)
{
    perror("inet_pton error");
    exit(1);
}
```

Além disso, adicionou-se um trecho de código (utilizando a função **getsockname()**) para mostrar em qual porta local o cliente está conectado.

```c
// informações do socket local
if (getsockname(sockfd, (struct sockaddr *)&addr, &len) < 0) // rec
{
   perror("getsocketname");
   exit(1);
}

if (inet_ntop(AF_INET, &(addr.sin_addr), name, sizeof(name)) <
{
   perror("inet_ntop");
   exit(1);
}

printf("Connected on %s port %d\n\n", name, ntohs(addr.sin_port));
```

Servidor:

```
fabio@Fabios-MacBook-Pro Relatório 3 % gcc -o servidor servidor.c -Wall
fabio@Fabios-MacBook-Pro Relatório 3 % ./servidor
Listening on 0.0.0.0 port 51420
```

Cliente:

```
fabio@Fabios-MacBook-Pro Relatório 3 % gcc -o cliente cliente.c -Wall
fabio@Fabios-MacBook-Pro Relatório 3 % ./cliente 0.0.0.0 51420
Connected on 127.0.0.1 port 51426

Mon Sep 27 22:31:40 2021
fabio@Fabios-MacBook-Pro Relatório 3 %
```

6. A função **getpeername()** recupera as informações do endereço do socket do cliente conectado ao servidor (conectado ao socket connfd). Com isso, utilizou-se a função **inet_ntop()** para obtermos a representação em string IPv4 do endereço obtido.

```c
// informações do socket cliente
if (getpeername(connfd, (struct sockaddr *)&addr, &len) < 0) // recupera
{
    perror("getpeername");
    exit(1);
}

if (inet_ntop(AF_INET, &(addr.sin_addr), name, sizeof(name)) < 0) // rec
{
    perror("inet_ntop error");
    exit(1);
}

printf("Client connected on %s port %d\n", name, ntohs(addr.sin_port));
```

Servidor:

```
fabio@Fabios-MacBook-Pro Relatório 3 % gcc -o servidor servidor.c -Wall
fabio@Fabios-MacBook-Pro Relatório 3 % ./servidor
Listening on 0.0.0.0 port 51472

Client connected on 127.0.0.1 port 51488
```

Cliente:

```
fabio@Fabios-MacBook-Pro Relatório 3 % gcc -o cliente cliente.c -Wall
fabio@Fabios-MacBook-Pro Relatório 3 % ./cliente 0.0.0.0 51472
Connected on 127.0.0.1 port 51488

Mon Sep 27 22:36:59 2021
fabio@Fabios-MacBook-Pro Relatório 3 %
```

Nota-se que, nesse exemplo, o servidor está escutando conxões na porta local 51472 e o cliente está conectado na sua porta local 51488.

7. Sim, é possível, uma vez que o comando telnet recebe como parâmetro o endereço IP e a porta em que o servidor está escutando e conecta-se ao mesmo, funcionando da mesma forma que o binário do cliente discutido anteriormente, imprimindo o retorno recebido. A única diferença na execução é que o telnet não imprime qual porta local está conectado com o servidor

Servidor:

```
fabio@Fabios-MacBook-Pro Relatório 3 % gcc -o servidor servidor.c -Wall
fabio@Fabios-MacBook-Pro Relatório 3 % ./servidor
Listening on 0.0.0.0 port 51575

Client connected on 127.0.0.1 port 51577
```

Cliente (telnet):

```
fabio@Fabios-MacBook-Pro Relatório 3 % telnet 0.0.0.0 51575
Trying 0.0.0.0...
Connected to 0.0.0.0.
Escape character is '^]'.
Mon Sep 27 22:42:50 2021
Connection closed by foreign host.
fabio@Fabios-MacBook-Pro Relatório 3 %
```

Nota-se que, nesse exemplo, o servidor está escutando conxões na porta local 51575 e o cliente (telnet) está conectado na sua porta local 51577.