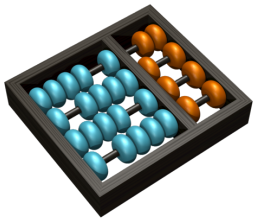

Relatório - Cliente e Servidor TCP

Alunos: Rian Radeck e Igor Brito

RAs: 187793 e 171929

Instituto de Computação

Universidade Estadual de Campinas

Campinas, 05 de Setembro de 2023.

Sumário

1 Sockets

1.1 Função socket()

A função socket é utilizada para criação de um file descriptor que representará o nosso socket. Ela recebe como parâmetros os protocolos especificados pelo programador, sendo eles IPV4 ou IPV6, TCP ou UDP e etc.

De acordo com o manual nosso construtor é definido da seguinte maneira **int socket(int domain, int type, int protocol);**

- int domain: Específica o domínio da comunicação, podendo ser local, IPV4, IPV6, etc. Em nosso código utilizamos AF_INET, que significa que nosso socket vai operar no domínio de IPV4.
- int type: O tipo (protocolo) de comunicação, podendo ser TCP (SOCK_STREAM) ou UDP (SOCK_DGRAM).
- int protocol: Geralmente definido como 0, que representa o protocolo IP.

1.2 Função bind()

Vincula um socket a um endereço e uma porta, para que ele possa receber conexões. Recebe o file descriptor do socket e endereço a ser vinculado. Tradicionalmente, esta operação é chamada de "atribuir um nome a um socket".

De acordo com o manual nosso construtor é definido da seguinte maneira **int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);**

- int sockfd: O socket a ser vinculado.
- const struct sockaddr *addr: O endereço a ser vinculado.
- socklen_t addrlen: O tamanho da estrutura do endereço.

1.3 Função connect()

Inicia a conexão em um socket. A chamada de sistema connect() conecta o socket referido pelo file descriptor para o endereço especificado no construtor.

De acordo com o manual nosso construtor é definido da seguinte maneira **int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);**

A definição dos parâmetros é a mesma da chamada bind().

1.4 Função read()

Lê uma quantidade de bytes da conexão estabelecida no socket e guarda essse bytes em um buffer, ou sucintamente recebe dados da conexão do socket. Resumidamente, read() tenta ler até uma certa quantidade de bytes do file descriptor em um buffer.

De acordo com o manual nosso construtor é definido da seguinte maneira **ssize_t read(int fd, void buf[.count], size_t count);**

- int fd: File descriptor a ser lido.
- void buf[.count]: Buffer que armazena os dados da leitura.
- size_t count: Número máximo de bytes a serem lidos.

1.5 Função write()

Escreve uma quantidade de bytes em um buffer na conexão estabelecida no socket, ou seja, envia dados para a outra ponta do socket. Em outras palavras, write() vai escrever uma certa quantidade de bytes de um buffer em um file descriptor.

De acordo com o manual nosso construtor é definido da seguinte maneira **ssize_t read(int fd, void buf[.count], size_t count);**

A definição dos parâmetros é a mesma da chamada read().

1.6 Função listen()

Marca o socket como passivo, isto é, um socket que será usado para aceitar conexões pela chamada de sistema accept().

De acordo com o manual nosso construtor é definido da seguinte maneira **int listen(int sockfd, int backlog);**

- int sockfd: O file descriptor do socket selecionado.
- int backlog: Tamanho máximo da fila de conexões pendentes.

1.7 Função accept()

A chamada de sistema accept() extrai a primeira solicitação de conexão na fila de conexões pendentes para um determinado socket passivo, cria um novo socket para essa conexão e retorna o file descriptor referente a esse novo socket.

De acordo com o manual nosso construtor é definido da seguinte maneira **int accept(int sockfd, struct sockaddr *_Nullable restrict addr, socklen_t *_Nullable restrict addrlen);**

- int sockfd: O socket passivo que recebeu a conexão.
- struct sockaddr *_Nullable restrict addr: O endereço que está fazendo aquela solicitação.
- socklen_t *_Nullable restrict addrlen: O tamanho da estrutura do endereço.

2 Handshake

O processo three-way handshake acontece efetivamente quando o servidor executa accept e o cliente usa connect. Quando o cliente inidcia a chamada do connect(), o handhsake se inicia. Quando o servidor devolve um SYNACK, o connect() retorna, e em seguida o accept() retorna depois do ACK. No momento que eles criam os sockets através de socket(), eles especificam qual protocolo usar, e isso define se o three-way handshake vai acontecer, no caso do TCP, por exemplo.

Em um servidor em modo de escuta (com um socket passivo) o processo é descrito da seguinte maneira:

1. O cliente envia uma mensagem TCP com a bandeira Synchronize. Indicando ao servidor que o cliente deseja estabelecer uma conexão.
2. O servidor recebe a solicitação do cliente e devolve uma mensagem com as bandeiras Synchronize e Acknowledge ativadas, indicando ao cliente que ele aceita a solicitação e está pronto para receber a conexão.
3. Finalmente, o cliente envia uma confirmação para o servidor, estabelecendo a conexão e ambos os lados podem começar a trocar dados.

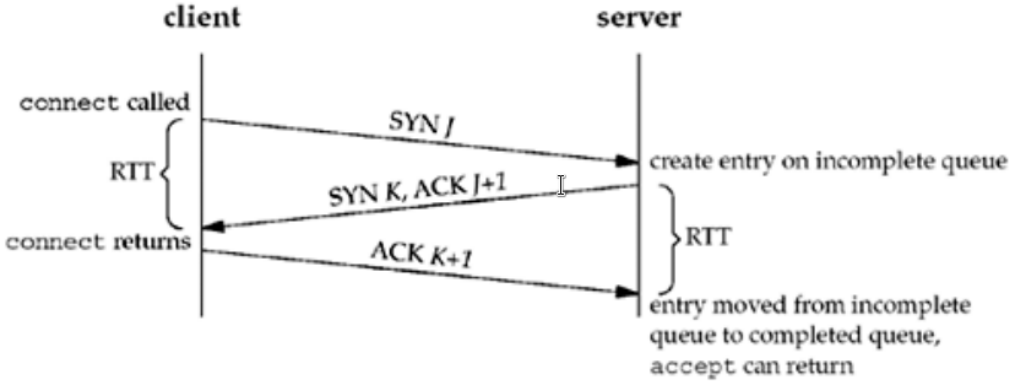

Figura 1: Diagrama de um Three-Way Handshake

3 Compilação e Execução

Foram utilizados os seguintes comandos para execução:

- `gcc -Wall cliente.c -o cliente`
- `gcc -Wall servidor.c -o servidor`

Após isso, o servidor foi executado e em seguida o cliente foi executado.

A função bind() funcionou normalmente, porém após finalizar o processo do servidor, temos que esperar alguns segundos para executar bind() novamente na mesma porta. Também é valido notar que não conseguimos executar simultaneamente duas ou mais instancias do servidor na mesma máquina, pois a porta só pode ser vinculada a um dos servidores.

4 Porta automática

Para que o SO selecione automaticamente uma porta disponível no momento do bind(), basta que façamos a atribuição a 0 no campo **sin_port**.

Note que como o servidor escolhe uma porta em tempo de execução, também precisamos modificar o código do cliente para ter como argumento a porta do servidor remoto.

```
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family      = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
  //Escolhemos porta 0 para que o SO atribua automaticamente uma porta
  //disponivel para o socket (Questao 4)
    servaddr.sin_port        = htons(0);
```

Além disso, também precisamos imprimir qual porta o SO selecionou para aquele servidor, e fazemos isso através da função getsockname().

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

printf("Bound to %d\n", (int)ntohs(servaddr.sin_port));
```

Dessa maneira, podemos informar essa porta ao nosso programa cliente, indicando que ela é a porta que está vinculada ao socket passivo.

5 Obtendo informações sobre um socket

Para obter algumas informações a respeito de um determinado socket utilizamos a função getsockname(), de maneira similar ao item anterior, passando o file descriptor do socket (já conectado) e uma estrutura do tipo **sockaddr** para ser populada com essas informações (como porta local e ip

local)

```
/* Codigo para obter (#IP, #porta local) da questao 5*/
socklen_t sz = sizeof(servaddr);
if (getsockname(sockfd, (struct sockaddr*)&servaddr, &sz) == -1)
{
  perror("getsockname");
  exit(1);
}

printf("Local: %s %d\n", inet_ntoa(servaddr.sin_addr), (int)ntohs(servaddr.sin_port)
  );
```

6 Obtendo informações do cliente

Para o servidor obter as informações do cliente requisitante chamamos a função getpeername() após o retorno da função accept(), onde o `connfd` foi populado com o socket que representa a conexão com o cliente recém-conectado.

Para chamar a função getpeername() passamos o socket e a estrutura de um endereço que será populada com as informações que buscamos (como IP remoto e porta remota)

```

  for ( ; ; ) {
    if ((connfd = accept(listenfd, (struct sockaddr *) NULL, NULL)) == -1 ) {
      perror("accept");
      exit(1);
      }

    //Computa e imprime o ip e a porta do cliente: (Questao 6)
    socklen_t peersz = sizeof(peeraddr);
```

```
10    if (getpeername(connfd, (struct sockaddr*)&peeraddr, &peersz) == -1)
11    {
12      perror("getpeername");
13      exit(1);
14    }
15    printf("Remote client connected: %s: %d\n", inet_ntoa(peeraddr.sin_addr), ntohs(
      peeraddr.sin_port));
16
```

7 Cliente enviando uma mensagem para o servidor

Vamos assumir que o cliente só deseja mandar uma linha para o servidor e então encerrar a conexão.

No código do servidor adicionamos um loop depois de mandar os dados para o cliente em que ele fica lendo os bytes até encontrar um n. Utilizamos a função read() para popular um buffer e imediatamente escrevemos o buffer na saída do terminal.

```
1     int n;
2     while((n = read(connfd, buf, MAXDATASIZE)) > 0)
3     {
4       buf[n] = 0;
5       if(fputs(buf, stdout) == EOF)
6       {
7         perror("fputs");
8         exit(1);
9       }
10      if(buf[n - 1] == '\n')
11        break;
```

```
12      }
```

No código do cliente apenas adicionamos um buffer e uma função para ler do terminal e popular esse buffer. Depois da leitura no terminal, mandamos para o servidor imediatamente utilizando um write().

```
1  char buff[MAXLINE + 1];
2  fgets(buff, MAXLINE, stdin);
3  write(sockfd, buff, strlen(buff));
```

8 Utilizando telnet

Sim, é possível utilizarmos a ferramenta telnet para fazer o papel do nosso código cliente. Para fazer isso vamos executar o servidor e estabelecer uma conexão com o telnet, observe os passos necessários:

1. Compilar e executar o servidor (como na seção 3). Veja a imagem 2.
   (a) Observe a porta que o SO determinou para ser vinculada ao socket passivo do servidor, ela será utilizada posteriormente, vamos chamá-la de **server_port**.
2. Executar o seguinte comando em outro terminal:
   `$ telnet localhost` **server_port**. Veja imagem 3
   (a) Note que se faz necessário a substituição do nome **server_port**, em nosso caso 39139.

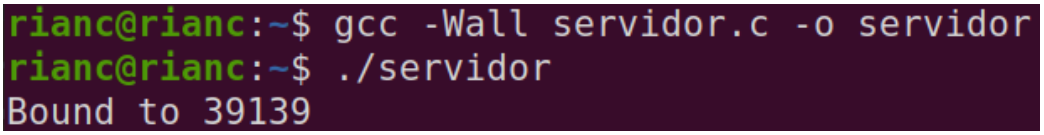

Figura 2: Log do servidor após o passo 1

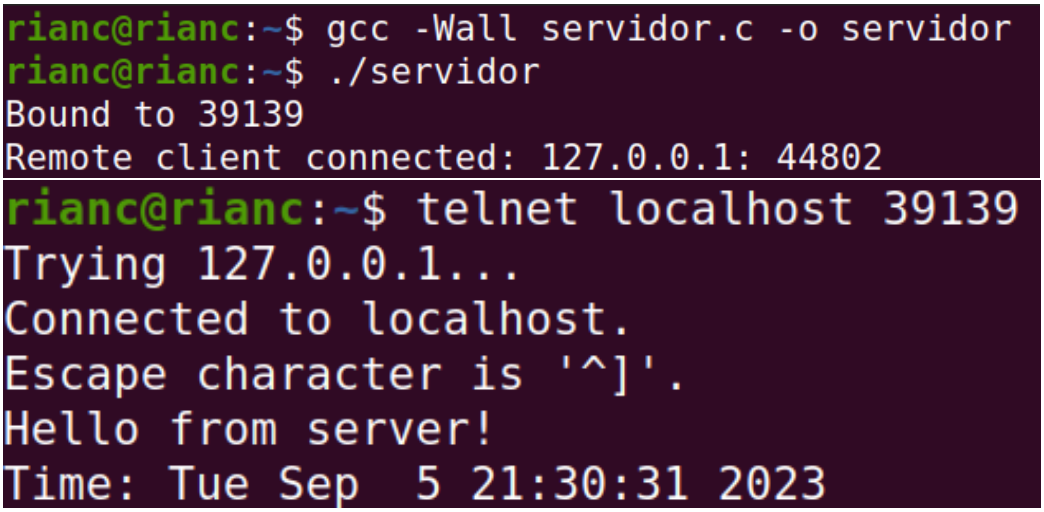

Figura 3: Logs do servidor e do cliente após o passo 2

No estado atual vemos que nosso servidor está conectado com um cliente que foi invocado pela ferramenta telnet.