

Atividade 2: Cliente e Servidor

TCP

Aluno: Guilherme Luis Domingues

RA: 155619

Instituto de Computação
Universidade Estadual de Campinas

Campinas, 22 de Outubro de 2020.

Sumário

1	Análise dos códigos	2
1.1	Cliente	2
1.2	Servidor	3
2	Compilação e execução	4
3	Generalizando as portas	5
4	Chamadas de sistema necessárias	5
5	Comentando o código	6
6	Obtendo as informações do socket local	6
7	Obtendo as informações do socket remoto	6
8	Utilizando o telnet como cliente	6

1 Análise dos códigos

1.1 Cliente

Dentro do código cliente.c, temos as seguintes funções utilizadas para a comunicação com os sockets:

- `socket(int domain, int type, int protocol)`
 - Cria um endpoint para comunicação e retornar sua descrição.
 - `domain`: Responsável para definir a comunicação que será utilizada. Por Exemplo: `PF_INET` para IPV4 e `PF_INET6` para IPV6.
 - `type`: Define o tipo da conexão. : `SOCK_STREAM` para TCP e `SOCK_DGRAM` para UDP.
 - `protocol`: Utilizado para definir um protocolo particular. Normalmente utiliza-se como 0.
- `inet_pton(int af, const char * restrict src, void * restrict dst)`
 - Esta função converte a representação de endereço para o formato de rede, utilizando a struct `inadd`. Retorna 1 se o endereço fornecido é valido, 0 caso não foi possível realizar o parse e -1 caso tenha ocorrido algum erro.
 - `af`: tipo de conexão: TCP ou UDP.
 - `src`: o endpoint criado anteriormente que será convertido para o formato de rede.
 - `dst`: endereço do servidor com que a conexão deverá ser estabelecida.
- `connect(int socket, const struct sockaddr *address, socklen_t address_len)`

- Inicializa a conexão do socket.
- socket: O socket que deverá ser inicializado.
- address: o endereço de conexão do socket destino.
- address_len: tamanho do address do socket destino.
- read(int fd, void *buf, size_t nbytes)
 - Lê as mensagens recebidas através do socket aberto
 - fd: origem das mensagens.
 - buf: buffer que será utilizado para receber as mensagens.
 - nbytes: número de bytes por mensagem.

1.2 Servidor

O código do servidor.c compartilha algumas funções com o cliente.c, então aqui estão as existentes apenas na implementação do servidor.

- bzero(void *s, size_t n)
 - Essa função apaga os dados nos n bytes da memória começando no local apontado por s, escrevendo zeros em toda a memória até o limite determinado pelo parâmetro n.
 - af: tipo de conexão: TCP ou UDP.
 - src: o endpoint criado anteriormente que será convertido para o formato de rede.
 - dst: endereço do servidor com que a conexão deverá ser estabelecida.
- bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)

- Inicializa a conexão do socket, traduzindo-a de "texto" para binário.
- socket: O socket que deverá ser inicializado.
- address: o endereço de conexão do socket destino.
- address_len: tamanho do address do socket destino.
- listen(int sockfd, int backlog)
 - Essa função marca o socket referido por sockfd como um socket passivo, ou seja, como um socket que será usado para aceitar a conexão de entrada.
 - sockfd: socket a escutar.
 - backlog: comprimento máximo de conexões pendentes que o socket pode crescer.
- accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen)
 - Essa função pega a primeira solicitação das fila pendente, e cria um novo socket, retornando o valor dele.
 - sockfd: socket a escutar.
 - addr: endereço do socket.
 - addrlen: tamanho do endereço do socket.

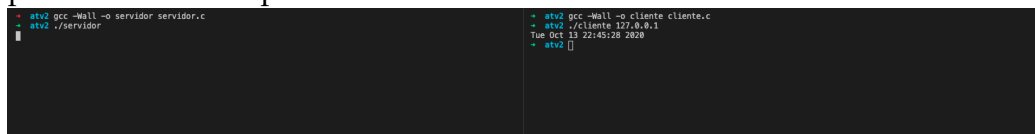
2 Compilação e execução

Ao tentar executar o código disponibilizado, houve um erro para instanciar a porta. O endereço passado é "192.168.0.16", o qual não é possível

se conectar, podendo já estar sendo utilizado por outra aplicação ou então não deve existir.

Para que o código funcionasse, tivemos que modificar o endereço do servidor para um conhecido pelo sistema e disponível na máquina, isto é o endereço padrão INADDR_ANY.

A Figura 1 mostra a execução do código sem erros, bem como a resposta do servidor para o cliente.



```
stiv2 gcc -Wall -o servidor servidor.c
stiv2 ./servidor

stiv2 gcc -Wall -o cliente cliente.c
stiv2 ./cliente 127.0.0.1
Tue Oct 13 22:45:28 2020
stiv2 []
```

3 Generalizando as portas

Para deixar a decisão de escolha das portas para o programa servidor.c, precisamos passar o valor 0 como sendo a porta que o socket address recebe. Traduzindo para o código, precisamos fazer:

```
servaddr.sin_port = 0;
```

4 Chamadas de sistema necessárias

- socket()

Responsável por escrever no arquivo o protocolo com que a conexão deverá ser estabelecida.

- bind()

Assinar um endereço e porta para esta conexão.

- listen()

Aguardar se alguma chamada efetuada está sendo feita para o mesmo endereço e porta assinada para ele.

- `accept()`

Extraí a primeira requisição de conexão na lista de pendentes, cria um novo socket com as mesmas propriedades e aloca um novo arquivo contendo sua descrição para o socket.

5 Comentando o código

6 Obtendo as informações do socket local

Para obter as informações do socket local, precisamos utilizar a função `getsockname`, passando como parâmetro o socket criado localmente.

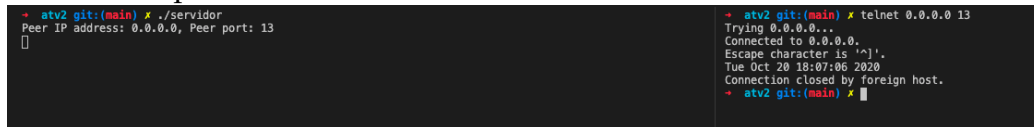
7 Obtendo as informações do socket remoto

Para obter as informações do socket remoto, precisamos utilizar a função `getpeername`, passando como parâmetro o socket do cliente que solicitou a conexão.

8 Utilizando o telnet como cliente

Sim, é possível utilizar o programa `telnet` no lugar do `cliente.c`. Como o programa `telnet` é uma forma de conexão com um servidor, basta apontar o endereço IP e porta que está sendo utilizada pelo servidor que a conexão é estabelecida.

Como Podemos exemplificar com a seguinte imagem, onde podemos ver que a conexão com o endereço 0.0.0.0:13 foi estabelecida com sucesso, obteve a resposta do servidor e a conexão foi encerrada com sucesso.



```
→ atv2 git:(main) ✗ ./servidor
Peer IP address: 0.0.0.0, Peer port: 13
█

→ atv2 git:(main) ✗ telnet 0.0.0.0 13
Trying 0.0.0.0...
Connected to 0.0.0.0.
Escape character is '^]'.
Tue Oct 20 18:07:06 2020
Connection closed by foreign host.
→ atv2 git:(main) ✗ █
```