

Exercício 2 – Laboratório Cliente Servidor TCP

Aluna: Naomi Takemoto

RA: 184849

Instituto de Computação
Universidade Estadual de Campinas

Outubro de 2020

Exercício 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.

Para o caso do Servidor

Uso

- **listenfd = socket(AF_INET, SOCK_STREAM, 0)**

Descrição da função

- Cria um endpoint de comunicação e retorna um descriptor. Nesse caso, a comunicação será baseada no versão 4 dos protocolos de internet (argumento 1) usando-se byte streams. Essa função cria um socket.

Descrição dos parâmetros

- int domain: seleciona a família de protocolos que deve ser utilizada. Os valores possíveis são definidos em <sys/socket.h> como se segue (transcritos diretamente da *man page* do comando *socket*):
 - PF_LOCAL Host-internal protocols, formerly called PF_UNIX,
 - PF_UNIX Host-internal protocols, deprecated, use PF_LOCAL,
 - PF_INET Internet version 4 protocols,
 - PF_ROUTE Internal Routing protocol,
 - PF_KEY Internal key-management function,
 - PF_INET6 Internet version 6 protocols,
 - PF_SYSTEM System domain,
 - PF_NDRV Raw access to network device
- int type: especifica a semântica da comunicação. Os valores possíveis (transcritos da *man page*) são:
 - SOCK_STREAM: comunicação baseada em byte streams.
 - SOCK_DGRAM: suporte para datagramas (não orientado a conexão, com troca de mensagens não confiável e de tamanho máximo fixo (geralmente pequeno)).
 - SOCK_RAW: possibilita acesso a protocolos e interfaces internos estando disponível apenas para o *super-user*.
- int protocol: especifica um protocolo em particular a ser usado pelo socket. Está ligado ao domínio, geralmente um domínio possui somente um protocolo disponível para uso, no entanto em alguns casos podem haver mais.

Descrição dos argumentos

Uso

- **bzero(&servaddr, sizeof(servaddr));**

Descrição da função

- Escreve zeros em um byte string, no caso de uso está sendo utilizada para zerar o endereço IP.

Descrição dos parâmetros

- void * s: apontador genérico para uma sequência de bytes
- size_t n: número de bytes a serem modificados

Descrição dos argumentos

- &serveraddr: apontador/referência para a string guarda o endereço IP do servidor.
 - sizeof(serveraddr): tamanho em bytes da string serveraddr
-

Uso

- **bind(listenfd, (struct sockaddr *)&servaddr, sizeof(servaddr)) == -1**

Descrição da função

- Associa um nome a um socket sem nome. A função bind faz com que address seja atribuído ao socket passado.

Descrição dos parâmetros

- int socket
- const struct sockaddr *address
- socklen_t address_len

Descrição dos argumentos

- listenfd: identificador do socket
- (struct sockaddr *)&servaddr: endereço a ser atribuído
- sizeof(servaddr): tamanho do endereço.

Retorno

- Em caso de sucesso retorna-se 0, caso contrário -1 e a variável global errno é setada para indicar erro.
-

Uso

- **listen(listenfd, LISTENQ) == -1**

Descrição da função

- Escuta conexões em um socket. Essa função especifica a susceptibilidade/regras em se aceitar conexões e limite para o número de conexões na fila. Se aplica somente para sockets do tipo SOCK_STREAM.

Descrição dos parâmetros

- int socket
- int backlog: define o maior valor para o tamanho a fila de conexões pendentes.

Descrição dos argumentos

- listenfd: socket
- LISTENQ: número máximo de item pendentes na fila.

Retorno

- 0 se a operação for bem sucedida e -1 caso contrário. Se um request de conexão for recusado
-

Uso

- `connfd = accept(listenfd, (struct sockaddr *) NULL, NULL) == -1`

Descrição da função

- Aceita conexões em um determinado socket, extraíndo para tando a primeira requisição de conexão da fila de conexões pendentes. Cria-se um novo socket com as mesmas propriedades que aquele passado com argumento e aloca-se um novo descriptor.

Descrição dos parâmetros

- int socket: socket que foi criado com o comando socket(2)
- struct sockaddr *restrict address: é um parâmetro usado para receber um resultado, no caso é o endereço da entidade que se comunica. O seu tipo depende do domínio especificado para o socket.
- socklen_t *restrict address_len: é também utilizado para receber um resultado, no caso o tamanho do endereço.

Descrição dos argumentos

- listenfd: socket criado anteriormente com o comando socket, que foi associado a um endereço com o comando bind e espera por conexões conforme configurado pelo comando listen(2).

Retorno

- -1 caso ocorra algum erro, caso contrário retorna um inteiro não negativo, no caso será guardado na variável connfd que é o descritor do socket da conexão aceita.
-

Para o caso do Cliente

Uso

- `inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0`

Descrição da função

- Converte o formato de um endereço de printável para o formato da rede, geralmente um struct in_addr ou algum outro formato .

Descrição dos parâmetros

- int af
- const char * restrict src
- void * restrict dst)

Descrição dos argumentos

- AF_INET: especifica to tipo de endereço neste caso compatível com o protocolo IPV4.
- argv[1]:
- &servaddr.sin_addr:

Retorno

- 1 caso o endereço passado seja válido, 0 se o endereço não é parseável e -1 se um erro de sistema ocorreu.

Exercício 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)

Na compilação do código servidor.c houve a geração de um warning como mostra a figura:

```
Naomi$ gcc -Wall servidor.c
servidor.c:29:37: warning: incompatible pointer to integer conversion passing 'char [13]' to parameter of type
      'uint32_t' (aka 'unsigned int') [-Wint-conversion]
      servaddr.sin_addr.s_addr = htonl("192.168.0.16");
                                   ^
/Library/Developer/CommandLineTools/SDKs/MacOSX10.15.sdk/usr/include/sys/_endian.h:136:46: note: expanded from
      macro 'htonl'
#define htonl(x)      __DARWIN_OSSwapInt32(x)
                                   ^
/Library/Developer/CommandLineTools/SDKs/MacOSX10.15.sdk/usr/include/Libkern/_OSByteOrder.h:75:76: note:
      expanded from macro '__DARWIN_OSSwapInt32'
      (__builtin_constant_p(x) ? __DARWIN_OSSwapConstInt32(x) : _OSSwapInt32(x))
                                   ^
/Library/Developer/CommandLineTools/SDKs/MacOSX10.15.sdk/usr/include/Libkern/_i386/_OSByteOrder.h:56:20: note:
      passing argument to parameter '_data' here
      _data
      ^
1 warning generated.
Naomi$
```

Isso ocorre pois htonl() espera um inteiro não uma string. Além disso ao executar o servidor com par ip/porta especificado entrou-se o erro "bind: Can't assign requested

address".

Substituindo as linhas do servidor:

```
servaddr.sin_addr.s_addr = htonl("192.168.0.16");
```

```
servaddr.sin_port = htons(13);
```

Por, onde 127.0.0.1 é o localhost:

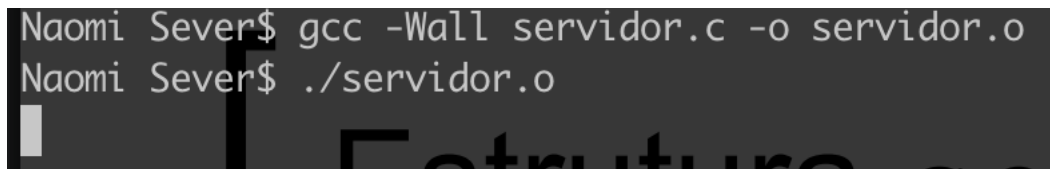
```
servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
```

```
servaddr.sin_port = htons(1024);
```

No código do cliente também foi necessário fazer uma modificação:

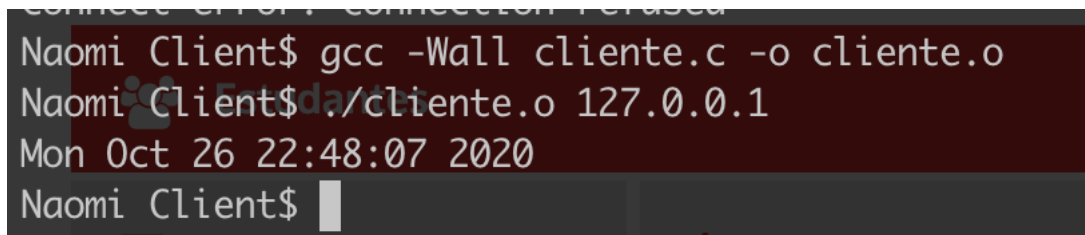
```
servaddr.sin_port = htons(1024);
```

Resolveu-se o warning e o erro de execução. Conforme mostram as figuras a seguir:



```
Naomi Sever$ gcc -Wall servidor.c -o servidor.o
Naomi Sever$ ./servidor.o
```

O cliente compilou e executou com sucesso:



```
Naomi Client$ gcc -Wall cliente.c -o cliente.o
Naomi Client$ ./cliente.o 127.0.0.1
Mon Oct 26 22:48:07 2020
Naomi Client$
```

Exercício 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.

Para automatizar a escolha de porta no servidor, basta definir:

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

Para isso foi necessário alterar o código do cliente para que receba como argumento também o número da porta e assim realize a requisição de conexão corretamente.

Para compilar e executar o código do cliente:

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

```
./cliente.o 127.0.0.1 <#Port>
```

Exemplo de execução:

```
./cliente.o 127.0.0.1 19404
```

Exercício 4

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

Justifique.

A resposta para este exercício foi baseada no artigo: "Know your TCP system call sequences" [2].

A lista de chamadas de sistema necessárias para que um servidor escute conexões é:

- `socket()`: responsável por criar um novo socket associado a determinado protocolo (por exemplo IPV4 TCP como no caso deste trabalho) retornando um descriptor para o processo que faz esta *syscall*.
- `bind()`: associa o socket a um endereço (IP e número de Porta).
- `listen()`: faz com que o socket escute uma porta, isto é indica que o protocolo que serve o processo está pronto para aceitar novas conexões.
- `accept()`: é uma chamada bloqueante que espera por uma nova conexão, uma vez que ela é processada, um *socket descriptor* é retornado. Este socket está conectado ao cliente, enquanto o outro permanece no estado de LISTEN para aceitar novas conexões.

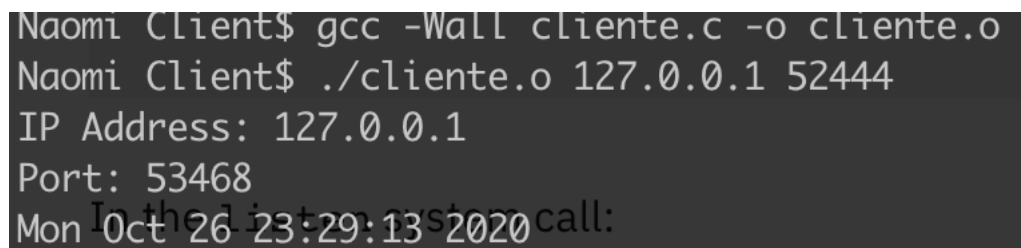
Exercício 5

Adicione comentários necessários ao código. (Não precisa anexar esta questão no relatório. Os comentários serão analisados no código enviado).

Exercício 6

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

As modificações no código estão indicadas no próprio arquivo de cliente.c. A figura a seguir mostra o funcionamento:



```
Naomi Client$ gcc -Wall cliente.c -o cliente.o
Naomi Client$ ./cliente.o 127.0.0.1 52444
IP Address: 127.0.0.1
Port: 53468
Mon Oct 26 23:29:13 2020
```

Exercício 7

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.

A função `getpeername(2)` retorna o endereço do peer conectado a um socket em específico.

```

Naomi Server$ gcc -Wall servidor.c -o servidor.o
Naomi Server$ ./servidor.o
Port: 19933
IP Address: 127.0.0.1
Address len: 16
Port: 56658
-----
IP Address: 127.0.0.1
Address len: 16
Port: 56659
-----
IP Address: 127.0.0.1
Address len: 16
Port: 56664
-----

```

Exercício 8

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

Foi possível fazer a conexão conforme mostra a figura:

Cliente

```

Naomi Client$ telnet localhost 58090
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Tue Oct 27 19:27:56 2020
Connection closed by foreign host.
Naomi Client$

```


Sevidor

```
Naomi Server$ ./servidor.o
Port: 58090
IP Address: 127.0.0.1
Address len: 16
Port: 58096
```

Referências

[1] What is the difference between 127.0.0.1 and 0.0.0.0. Disponível em:

<https://www.howtogeek.com/225487/what-is-the-difference-between-127.0.0.1-and-0.0.0.0/#:~:text=0.1%3F-,127.0.,used%20by%20the%20end%2Duser.>

[2] Know your TCP system call sequences. Disponível em:

[https://developer.ibm.com/technologies/systems/articles/au-tcpsystemcalls/.](https://developer.ibm.com/technologies/systems/articles/au-tcpsystemcalls/)