MC833 A - Programação de redes de computadores
Relatório - Tarefa 03
TCP - SIMPLEX-TALK

093125 - Tiago Martinho de Barros - *tiago.ec09@gmail.com*
093175 - Victor Fernando Pompêo Barbosa - *victorfpb@gmail.com*

28 de abril de 2016

Prof. Paulo Lício de Geus
IC – UNICAMP

Sumário

1 Introdução

Nesta tarefa estudaremos a abstração de `socket` na linguagem C e as chamadas de sistema associadas. A importância dessas chamadas reside na sua utilidade para construção de aplicações de rede.

2 Desenvolvimento

Este exercício consiste basicamente no estudo de funções usadas na programação de softwares que se utilizam de rede para transmitir e receber mensagens. E também na compreensão do programa *Simplex-Talk*, que é um programa simples de comunicação via redes, bem como em modificações dele.

3 Questão 1

Nesta questão, explicaremos o funcionamento de duas funções: `accept` e `htons`.

1. `accept`
   Esta função está definida em $\langle sys/socket.h \rangle$ e sua assinatura é:

   ```
   int accept(int socket, struct sockaddr *restrict address,
      socklen_t *restrict address_len);
   ```

   O que ela faz é pegar a primeira conexão da lista de conexões pendentes, criar um novo *socket* com o mesmo protocolo e família de endereço que o *socket* passado como parâmetro, e também alocar um novo descritor de arquivo para o *socket* criado.

   Os parâmetros são:

   (a) *socket*
   Um *socket* criado com a função `socket()`, que foi associado a um endereço com a função `bind()` e que foi usado na função `listen()`.

   (b) *address*
   Apontador para uma estrutura **sockaddr** em que o endereço do *socket* usado na conexão deve ser retornado. Pode ser NULL.

   (c) *address_len*
   Apontador para um objeto **socklen_t** que, na entrada, especifica o tamanho da estrutura **sockaddr** especificada e, na saída, especifica o tamanho do endereço armazenado. Pode ser NULL se *address* for NULL.

   O valor de retorno é um descritor de arquivo não negativo se a chamada foi completada com sucesso. Caso contrário, -1 é retornado.

2. `htons`
   Esta função está definida em $\langle arpa/inet.h \rangle$ (ou em $\langle netinet/in.h \rangle$) e sua assinatura é:

   ```
   uint16_t htons(uint16_t hostshort);
   ```

   Ela recebe um **unsigned short integer** (inteiro curto, sem sinal, de 16 bits), encodado na arquitetura de ordem de bytes do cliente (no caso do i386, a codificação é *Little Endian*), e o converte para a arquitetura de ordem de bytes da rede (na Internet, a codificação é *Big Endian*). Na arquitetura *Big Endian*, o byte mais significativo de uma palavra é armazenado em certo endereço de memória e os bytes subsequentes são guardados nos próximos endereços, de maior valor. A arquitetura *Little Endian*, por sua vez, armazena o byte menos significativo no endereço de memória de menor valor e o byte mais significativo no endereço de memória de maior valor.

4 Questão 2

Os programas fornecidos no site da disciplina foram compilados e executados numa mesma máquina. A entrada/saída do servidor pode ser verificada a seguir.

```
niko@ubuntu:~/Desktop/mc833/t3/code$ ./server
Testing
client2servermessage
Trying whitespaces now
Isso funciona com sinais gráficos?
E se colocarmos c cedilha? ç
```

A entrada/saída do cliente, por sua vez, pode ser verificada a seguir.

```
niko@ubuntu:~/Desktop/mc833/t3/code$ ./client localhost
Testing
client2servermessage
Trying whitespaces now
Isso funciona com sinais gráficos?
E se colocarmos c cedilha? ç
```

Como pode ser observado, a comunicação ocorreu perfeitamente, transmitindo inclusive sinais gráficos e cê-cedilha.

5 Questão 3

A primeira ferramenta utilizada para comprovação de que os códigos estão realizando uma comunicação via rede é o comando `tcpdump`. Invocando o comando com instruções explícitas para que escutasse a interface de loopback, pudemos identificar os pacotes transmitidos por meio dos códigos executados. A entrada/saída do comando `tcpdump` está exibida a seguir.

```
niko@ubuntu:~/Desktop/mc833/t3/code$ sudo tcpdump -i lo
    tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
    listening on lo, link-type EN10MB (Ethernet), capture size 65535 bytes
    16:16:02.844376 IP localhost.52844 > localhost.31472: Flags [P.], seq 2140592837:2140592869,
        ack 1074105590, win 342, options [nop,nop,TS val 21866426 ecr 21863306], length 32
    16:16:02.844402 IP localhost.31472 > localhost.52844: Flags [.], ack 32, win 342, options
        [nop,nop,TS val 21866426 ecr 21866426], length 0
    16:16:06.592760 IP localhost.50765 > ubuntu.domain: 53303+ A? www.papeeria.com. (34)
    16:16:06.592928 IP localhost.50765 > ubuntu.domain: 39377+ AAAA? www.papeeria.com. (34)
    16:16:06.611305 IP ubuntu.domain > localhost.50765: 53303 2/4/8 CNAME papeeria.com., A
        104.155.84.196 (358)
    16:16:06.624588 IP ubuntu.domain > localhost.50765: 39377 1/1/0 CNAME papeeria.com. (138)
    16:16:07.257027 IP localhost.52844 > localhost.31472: Flags [P.], seq 32:72, ack 1, win 342,
        options [nop,nop,TS val 21867529 ecr 21866426], length 40
    16:16:07.257054 IP localhost.31472 > localhost.52844: Flags [.], ack 72, win 342, options
        [nop,nop,TS val 21867529 ecr 21867529], length 0
    16:16:11.630504 IP localhost.52844 > localhost.31472: Flags [P.], seq 72:133, ack 1, win 342,
        options [nop,nop,TS val 21868622 ecr 21867529], length 61
    16:16:11.630529 IP localhost.31472 > localhost.52844: Flags [.], ack 133, win 342, options
        [nop,nop,TS val 21868622 ecr 21868622], length 0
    16:16:12.992878 IP localhost.40878 > ubuntu.domain: 31618+ A? ads.papeeria.com. (34)
    16:16:12.993188 IP localhost.40878 > ubuntu.domain: 45969+ AAAA? ads.papeeria.com. (34)
    16:16:12.997273 IP localhost.41561 > ubuntu.domain: 17325+ A? www.papeeria.com. (34)
    16:16:12.997298 IP localhost.41561 > ubuntu.domain: 3480+ AAAA? www.papeeria.com. (34)
    16:16:13.015900 IP ubuntu.domain > localhost.41561: 3480 1/1/0 CNAME papeeria.com. (138)
    16:16:13.016243 IP ubuntu.domain > localhost.40878: 45969 0/1/0 (124)
    16:16:13.017286 IP ubuntu.domain > localhost.41561: 17325 2/4/8 CNAME papeeria.com., A
        104.155.84.196 (358)
    16:16:13.144504 IP ubuntu.domain > localhost.40878: 31618 1/4/8 A 104.155.12.58 (344)
```

A entrada/saída do programa que funciona como servidor pode ser verificada a seguir.

```
niko@ubuntu:~/Desktop/mc833/t3/code$ ./server
This message has 30 characters
And now we're writing some other stuff
Just to see if tcpdump can correctly identify these packets
```

E, por fim, a seguir podem ser encontradas a entrada/saída do programa que funciona como cliente.

```
niko@ubuntu:~/Desktop/mc833/t3/code$ ./client localhost
This message has 30 characters
And now we're writing some other stuff
Just to see if tcpdump can correctly identify these packets
```

As seções da saída do `tcpdump` que permitem identificar que a comunicação se dá via rede estão exibidas a seguir.

```
16:16:02.844376 IP localhost.52844 > localhost.31472: Flags [P.], seq 2140592837:2140592869,
    ack 1074105590, win 342, options [nop,nop,TS val 21866426 ecr 21863306], length 32
16:16:02.844402 IP localhost.31472 > localhost.52844: Flags [.], ack 32, win 342, options
    [nop,nop,TS val 21866426 ecr 21866426], length 0
```

```
16:16:07.257027 IP localhost.52844 > localhost.31472: Flags [P.], seq 32:72, ack 1, win 342,
    options [nop,nop,TS val 21867529 ecr 21866426], length 40
16:16:07.257054 IP localhost.31472 > localhost.52844: Flags [.], ack 72, win 342, options
    [nop,nop,TS val 21867529 ecr 21867529], length 0
```

```
16:16:11.630504 IP localhost.52844 > localhost.31472: Flags [P.], seq 72:133, ack 1, win
    342, options [nop,nop,TS val 21868622 ecr 21867529], length 61
16:16:11.630529 IP localhost.31472 > localhost.52844: Flags [.], ack 133, win 342, options
    [nop,nop,TS val 21868622 ecr 21868622], length 0
```

Como pode ser verificado, a comunicação se dá entre as mesmas portas do `localhost`, além de a contagem de bytes de cada uma das mensagens equivaler exatamente ao tamanho das mensagens transmitidas utilizando o par de programas.

Outra ferramenta que podemos utilizar para comprovar que os códigos estão realizando uma comunicação via rede é o `netstat`, que fornece informações sobre as conexões ativas. Executando o servidor e o cliente numa mesma máquina, temos:

```
bash-4.3$ netstat -tuW
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address       Foreign Address       State
tcp        0      0 taz.lab.ic.unicamp.br:56978 40.86.88.211:https ESTABLISHED
tcp        0      0 taz.lab.ic.unicamp.br:36146 196.84.155.104.bc.googleusercontent.com:https
    ESTABLISHED
tcp        0      0 taz.lab.ic.unicamp.br:40436 cb-in-f189.1e100.net:https ESTABLISHED
tcp        0      0 taz.lab.ic.unicamp.br:44660 msnbot-65-52-108-76.search.msn.com:https
    ESTABLISHED
tcp        0      0 localhost.localdomain:56562 localhost.localdomain:31472 ESTABLISHED
tcp        0      0 taz.lab.ic.unicamp.br:40378 58.12.155.104.bc.googleusercontent.com:https
    ESTABLISHED
tcp        0      0 taz.lab.ic.unicamp.br:36068 74.125.107.203:https ESTABLISHED
tcp        0      0 taz.lab.ic.unicamp.br:35574 196.84.155.104.bc.googleusercontent.com:https
    ESTABLISHED
tcp        0      0 taz.lab.ic.unicamp.br:33668 rio01s16-in-f5.1e100.net:https ESTABLISHED
tcp        0      0 localhost.localdomain:31472 localhost.localdomain:56562 ESTABLISHED
tcp        0      0 taz.lab.ic.unicamp.br:wpgs franjinha.lab.ic.unicamp.br:nfs ESTABLISHED
tcp        0      0 taz.lab.ic.unicamp.br:33700 rio01s15-in-f1.1e100.net:https ESTABLISHED
tcp        0      0 taz.lab.ic.unicamp.br:35128 edge-star-shv-04-prn2.facebook.com:https
    ESTABLISHED
tcp        0      0 taz.lab.ic.unicamp.br:36206 gru06s26-in-f14.1e100.net:https ESTABLISHED
tcp        0      0 taz.lab.ic.unicamp.br:59990 franjinha.lab.ic.unicamp.br:sunrpc TIME_WAIT
tcp        0      0 taz.lab.ic.unicamp.br:48726 cb-in-f93.1e100.net:https ESTABLISHED
tcp        0      0 taz.lab.ic.unicamp.br:49292 mingau.lab.ic.unicamp.br:ldap ESTABLISHED
tcp        0      0 taz.lab.ic.unicamp.br:34626 72.21.81.200:https ESTABLISHED
tcp        0      0 taz.lab.ic.unicamp.br:34024 cb-in-f138.1e100.net:https ESTABLISHED
tcp        0      0 taz.lab.ic.unicamp.br:928 franjinha.lab.ic.unicamp.br:47518 ESTABLISHED
```

Duas linhas mostram a conexão via rede na máquina:

```
tcp        0      0 localhost.localdomain:56562 localhost.localdomain:31472 ESTABLISHED
```

```
tcp       0      0 localhost.localdomain:31472 localhost.localdomain:56562 ESTABLISHED
```

Como sabemos, pelo código do servidor e do cliente, o servidor escuta na porta 31472, que aparece nas saídas acima. Também notamos que a porta efêmera usada pelo cliente neste caso foi 56562.

Executando o comando `lsof` com parâmetro a porta 56562, temos:

```
bash-4.3$ lsof -i :56562
COMMAND  PID  USER  FD  TYPE DEVICE SIZE/OFF NODE NAME
server 13517 ra093125  4u  IPv4 334005    0t0  TCP
    localhost.localdomain:31472->localhost.localdomain:56562 (ESTABLISHED)
client 13654 ra093125  3u  IPv4 334205    0t0  TCP
    localhost.localdomain:56562->localhost.localdomain:31472 (ESTABLISHED)
```

Vemos que realmente se trata do Simplex-Talk.

6 Questão 4

Sim, o programa `telnet` pode ser usado no lugar do binário do `client.c`, pois podemos usar o programa `telnet` para conectar a um determinado host numa determinada porta, como abaixo:

- Saída da execução do `telnet`

```
bash-4.3$ telnet localhost 31472
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
testando telnet
```

- Saída do servidor

```
bash-4.3$ ./server
testando telnet
```

7 Questão 5

Os programas modificados são incluídos abaixo:

- `server.c`

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>

#define SERVER_PORT 31472
#define MAX_PENDING 5
#define MAX_LINE 256

int main()
{
  struct sockaddr_in sin;
  char buf[MAX_LINE];
  int len;
  int s, new_s;

  /* build address data structure */
  bzero((char *)&sin, sizeof(sin));
  sin.sin_family = AF_INET;
```

```
  sin.sin_addr.s_addr = INADDR_ANY;
  sin.sin_port = htons(SERVER_PORT);

  /* setup passive open */
  if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
     perror("simplex-talk: socket");
     exit(1);
  }
  if ((bind(s, (struct sockaddr *)&sin, sizeof(sin))) < 0) {
     perror("simplex-talk: bind");
     exit(1);
  }
  listen(s, MAX_PENDING);

  /* wait for connection, then receive and print text */
  while(1) {
     if ((new_s = accept(s, (struct sockaddr *)&sin, &len)) < 0) {
        perror("simplex-talk: accept");
        exit(1);
     }
     while ((len = recv(new_s, buf, sizeof(buf), 0))) {
        fputs(buf, stdout);
     if((send(new_s, buf, strlen(buf)+1, 0)) < 0) {
        perror("simplex-talk: send");
        exit(1);
     }
  }
     close(new_s);
  }
  return EXIT_SUCCESS;
}
```

- client.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>

#define SERVER_PORT 31472
#define MAX_LINE 256

int main(int argc, char * argv[])
{
  struct hostent *hp;
  struct sockaddr_in sin;
  char *host;
  char buf[MAX_LINE];
  int s;
  int len;
  if (argc==2) {
     host = argv[1];
  }
  else {
     fprintf(stderr, "usage: ./client host\n");
  exit(1);
}

  /* translate host name into peer's IP address */
  hp = gethostbyname(host);
  if (!hp) {
```

```
    fprintf(stderr, "simplex-talk: unknown host: %s\n", host);
    exit(1);
  }

  /* build address data structure */
  bzero((char *)&sin, sizeof(sin));
  sin.sin_family = AF_INET;
  bcopy(hp->h_addr, (char *)&sin.sin_addr, hp->h_length);
  sin.sin_port = htons(SERVER_PORT);

  /* active open */
  if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
    perror("simplex-talk: socket");
    exit(1);
  }
  if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
    perror("simplex-talk: connect");
    close(s);
    exit(1);
  }

  /* main loop: get and send lines of text */
  while (fgets(buf, sizeof(buf), stdin)) {
    buf[MAX_LINE-1] = '\0';
    len = strlen(buf) + 1;
    send(s, buf, len, 0);
    if(recv(s, buf, sizeof(buf), 0) > 0) {
      fputs(buf, stdout);
      fputs("\n", stdout);
    }
  }

  return EXIT_SUCCESS;
}
```