MC823 Atividade 1: Cliente/Servidor TCP Iterativo

Gustavo Sverzut Barbieri <ra008849@ic.unicamp.br>

1 Introdução

Nessa atividade nós modificamos cliente e servidor simples que utilizam o protocolo TCP/IP.

2 Erro em `TIME_WAIT`

Ao testar várias instâncias do servidor concurrentemente ou enquanto o *kernel* ainda não liberou o *socket* previamente alocado, isto é, o **netstat** acusa `TIME_WAIT`, percebemos que a função `bind` retorna com falha `EADDRINUSE` (*"Address already in use"*).

Isto pode ser evitado configurando o *socket* para reaproveitar o endereço, como exemplificado na Listagem 1.

Listing 1: Configurando *socket* para reaproveitar endereço

```
int yes = 1;
if ( setsockopt( sockfd , SOL_SOCKET, SO_REUSEADDR,
                 &yes, sizeof( int ) ) == -1 )
{
  perror( "setsockopt" );
  exit( 1 );
}
```

3 Modificando o servidor para retornar data e hora

Para que o servidor retorne a data e a hora, utilizamos as funções `time` para conseguir o tempo decorrido desde a época Unix (00:00:00, 1-Janeiro-1970) e então utilizamos `ctime` para transformá-lo em uma representação textual da data e hora atuais, a qual é enviada ao servidor.

O código necessário para o servidor, sem conferência de limites para a *string*, está na Listagem 2. O código completo se encontra na Listagem 7.

Listing 2: Enviando data e hora para o cliente

```
time( &tm );
strncpy( msg, ctime( &tm ), MAX_MSG_LEN );
msg[ MAX_MSG_LEN -1 ] = 0;
len_msg = strlen( msg );
```

```
if ( send( new_fd, msg, len_msg, 0 ) == -1 )
   /* procedimento convencional de conferência
      de erro de transmissão ...*/
```

4 Implementando um servidor *echo*

Um servidor *echo* deve retornar uma cópia do que lhe foi enviado. Isto foi obtido utilizando-se de um laço que recebe um caractere do *socket* e logo o envia ao cliente. Isto pode ser obtido com o código na Listagem 3. O código completo do servidor *echo* se econtra na Listagem 8.

Listing 3: Implementando servidor *echo*

```
while (1)
{
  int recvd = recv( new_fd, &input, 1, 0 );
  if ( recvd == -1 )
  {
    perror( "recv" );
    break;
  }
  else if ( recvd == 0 )
  {
    printf( "closing connection!\n" );
    break;
  }

  if ( send( new_fd, &input, 1, 0 ) == -1 )
  {
    perror( "send" );
    break;
  }
}
```

Já o cliente deve ler da entrada padrão e enviar o caractere ao servidor. A implementação está na Listagem 4 e utiliza `select` para não bloquear na entrada padrão ou no *socket*.

Listing 4: Implementando cliente *echo*

```
FD_ZERO( &readfds );
FD_SET( 0, &readfds );
FD_SET( sockfd, &readfds );

while ( 1 )
{
  rds = readfds;
```

```
  sr = select( sockfd + 1, &rds, NULL, NULL, NULL );

  if ( sr == -1 )
     perror( "select" );
  else if ( sr )
  {
    /* Data from standard input ? */
    if ( FD_ISSET( 0, &rds ) )
    {
      char input;
      int r;
      if ( ( r = read( 0, &input, 1 ) ) == 1 )
      {
        if ( send( sockfd, &input, 1, 0 ) == -1 )
        {
          perror( "send" );
          exit( -1 );
        }
      }
      else if ( r == 0 )
         break;
    }

    /* Data from socket ? */
    else if ( FD_ISSET( sockfd, &rds ) )
    {
      char input;
      if ( ( numbytes = recv( sockfd, &input, 1, 0 ) )== -1 )
      {
        perror( "recv" );
        exit( -1 );
      }
      else if ( numbytes == 0 )
      {
        puts( "Connection closed by peer.\n" );
        break;
      }
      else
        putchar( input );
    }
  }
}
```

5 Resolvendo IPs a partir de um nome

Para possibilitar que um aplicativo utilize nomes (*"hostnames"*) no lugar de números IP, é necessário poder resolver os mesmos, conseguindo o número IP correspondente.

Para isso utilizamos `getaddrinfo`, a qual recebe um nome como parâmetro e retorna o número associado. As modificações necessárias estão exemplificadas na Listagem 5, e o código completo de um cliente que consegue resolver nomes está na Listagem 10.

Listing 5: Resolvendo IPs a partir de nomes

```
/* Set hits */
memset( &hints, 0, sizeof( hints ) );
hints.ai_family     = PF_INET;
hints.ai_socktype   = SOCK_STREAM;
hints.ai_flags     |= AI_CANONNAME;

/* Look up address */
if ( ( err = getaddrinfo( argv[ 1 ], NULL,
                          &hints, &res ) ) != 0 )
   printf( "getaddrinfo: %s\n", gai_strerror( err ) );
else
{
  char host[ NI_MAXHOST ];
  struct addrinfo *i = res;
  memcpy( &their_addr.sin_addr,
          &((struct sockaddr_in *) res->ai_addr)->sin_addr,
          sizeof( struct in_addr ) );
  freeaddrinfo( res );
}
```

6 Resolvendo nomes a partir de um IP

É comum querermos saber o nome associado a um IP para mais facilmente podermos indentificá-lo, para isso podemos usar a função `getnameinfo`, como exemplifica-se na Listagem 6.

O código para um servidor que imprime o nome do cliente que conectou-se está descrito na Listagem 11.

Listing 6: Resolvendo nomes a partir de um IP.

```
/* Find out the client host name */
if ( getnameinfo( ( struct sockaddr * ) &their_addr,
                  sizeof( their_addr ),
                  host, NI_MAXHOST, NULL, 0, 0 ) != 0 )
{
  perror( "getnameinfo" );
  strncpy( host, "<UNKNOW_ERROR>", NI_MAXHOST - 1 );
}

printf( "server: got connection from %s (%s)\n",
```

```
        inet_ntoa( their_addr.sin_addr ),
        host );
```

7 Apêndice

Listing 7: Servidor de data e hora

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <time.h>

#define MYPORT 3490
#define BACKLOG 10

int main(void)
{
    int sockfd, new_fd;
    struct sockaddr_in my_addr;
    struct sockaddr_in their_addr;
    int sin_size;

#define MAX_MSG_LEN 30
    char msg[ MAX_MSG_LEN + 1 ];
    int len_msg = 0;
    time_t tm;


    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    my_addr.sin_family = AF_INET;
    my_addr.sin_port = htons(MYPORT);
    my_addr.sin_addr.s_addr = INADDR_ANY;
    memset(&(my_addr.sin_zero), '\0', 8);

    if (bind(sockfd, (struct sockaddr *)&my_addr,
             sizeof(struct sockaddr)) == -1) {
        perror("bind");
        exit(1);
```

```
    }

    if (listen(sockfd, BACKLOG) == -1) {
        perror("listen");
        exit(1);
    }

    while(1) {  // main accept() loop
        sin_size = sizeof(struct sockaddr_in);
        if ((new_fd = accept(sockfd,
                             (struct sockaddr *)&their_addr,
                             &sin_size)) == -1) {
            perror("accept");
            continue;
        }
        printf("server: got connection from %s\n",
               inet_ntoa(their_addr.sin_addr));


        time( &tm );
        strncpy( msg, ctime(&tm), MAX_MSG_LEN );
        msg[ MAX_MSG_LEN -1 ] = 0;
        len_msg = strlen( msg );
        if ( len_msg >= MAX_MSG_LEN )
          len_msg --;
        msg[ len_msg++ ] = '\n';
        msg[ len_msg ] = 0;

        if (send(new_fd, msg, len_msg, 0) == -1)
          perror("send");
        close(new_fd);
    }
    return 0;
}
```

Listing 8: Servidor de *echo*

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

#define MYPORT 3490
#define BACKLOG 10
```

```
int main(void)
{
    int sockfd, new_fd;
    struct sockaddr_in my_addr;
    struct sockaddr_in their_addr;
    int sin_size, sr;
    char input;


    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    my_addr.sin_family = AF_INET;
    my_addr.sin_port = htons(MYPORT);
    my_addr.sin_addr.s_addr = INADDR_ANY;
    memset(&(my_addr.sin_zero), '\0', 8);

    if (bind(sockfd, (struct sockaddr *)&my_addr,
             sizeof(struct sockaddr)) == -1) {
        perror("bind");
        exit(1);
    }

    if (listen(sockfd, BACKLOG) == -1) {
        perror("listen");
        exit(1);
    }


    while(1) {  // main accept() loop
        sin_size = sizeof(struct sockaddr_in);
        if ((new_fd = accept(sockfd,
                             (struct sockaddr *)&their_addr,
                             &sin_size)) == -1) {
            perror("accept");
            continue;
        }
        printf("server: got connection from %s\n",
               inet_ntoa(their_addr.sin_addr));

        while (1) {

          int recvd = recv( new_fd, &input, 1, 0 );
          if ( recvd == -1 )
            {
              perror( "recv" );
              break;
            }
```

```
            else if ( recvd == 0 )
              {
                printf( "closing connection!\n" );
                break;
              }

            if ( send( new_fd, &input, 1, 0 ) == -1 )
              {
                perror( "send" );
                break;
              }
          }

          close(new_fd);
      }
      return 0;
}
```

Listing 9: Cliente de *echo*

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>

#define PORT 3490
#define MAXDATASIZE 100

int main(int argc, char *argv[])
{
    int sockfd, numbytes, sr;
    char buf[MAXDATASIZE];
    struct sockaddr_in their_addr;
    fd_set readfds, rds;
    char sbuf[MAXDATASIZE]; int sbuf_pos=0;

    if (argc != 2) {
        fprintf(stderr,"usage: client hostip\n");
        exit(1);
    }

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
```

```
    }

    their_addr.sin_family = AF_INET;
    their_addr.sin_port = htons(PORT);
    if (inet_pton(AF_INET, argv[1],
                  &their_addr.sin_addr) <= 0) {
      perror("inet_pton");
      exit(1);
    }
    memset(&(their_addr.sin_zero), '\0', 8);

    if (connect(sockfd,
                (struct sockaddr *)&their_addr,
                sizeof(struct sockaddr)) == -1) {
        perror("connect");
        exit(1);
    }

    FD_ZERO( &readfds );
    FD_SET( 0, &readfds );
    FD_SET( sockfd, &readfds );

    while ( 1 )
      {
        rds = readfds;
        sr = select( sockfd + 1, &rds, NULL, NULL, NULL );

        if ( sr == -1 )
          perror( "select" );
        else if ( sr )
          {

            /* Data from standard input ? */
            if ( FD_ISSET( 0, &rds ) )
              {
                char input;
                int r;
                if ( ( r = read( 0, &input, 1 ) ) == 1 )
                  {
                    if ( send( sockfd, &input, 1, 0 ) == -1 )
                      {
                        perror( "send" );
                        exit( -1 );
                      }
                  }
                else if ( r == 0 )
                  break;
              }

            /* Data from socket ? */
```

```
            else if ( FD_ISSET( sockfd, &rds ) )
              {
                char input;
                if ( ( numbytes = recv( sockfd,
                                        &input,
                                        1, 0 ) ) == -1 )
                  {
                    perror( "recv" );
                    exit( -1 );
                  }
                else if ( numbytes == 0 )
                  {
                    puts( "Connection closed by peer.\n" );
                    break;
                  }
                else
                  {
                    putchar( input );
                  }
              }
          }

      }

    close(sockfd);

    return 0;
}
```

Listing 10: Cliente com resolução de nomes para IPs

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

#define PORT 3490
#define MAXDATASIZE 100

int main(int argc, char *argv[])
{
    int sockfd, numbytes, err;
    char buf[MAXDATASIZE];
    struct sockaddr_in their_addr;
```

```
    struct addrinfo hints, *res;

    if (argc != 2) {
        fprintf(stderr,"usage: client host\n");
        exit(1);
    }

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    their_addr.sin_family = AF_INET;
    their_addr.sin_port = htons(PORT);


    /* Set hits */
    memset( &hints, 0, sizeof( hints ) );
    hints.ai_family    = PF_INET;
    hints.ai_socktype  = SOCK_STREAM;
    hints.ai_flags    |= AI_CANONNAME;

    /* Look up address */
    if ( ( err = getaddrinfo( argv[ 1 ],
                              NULL, &hints, &res ) ) != 0 )
      printf( "getaddrinfo: %s\n", gai_strerror( err ) );
    else
      {
        char host[ NI_MAXHOST ];
        struct addrinfo *i = res;

#ifdef DEBUG
        printf( "'%s' has the following IP " \
                "addresses translations:\n",
                argv[ 1 ] );
        while ( i )
          {
            inet_ntop( i->ai_family,
              &((struct sockaddr_in *) i->ai_addr)->sin_addr,
                       host, NI_MAXHOST );
              printf( " >%s (%s)\n", host, i->ai_canonname );
              i = i->ai_next;
            }
          inet_ntop( res->ai_family,
            &((struct sockaddr_in *) res->ai_addr)->sin_addr,
                     host, NI_MAXHOST );
          printf( "Setting IP to %s\n", host );
#endif
          memcpy( &their_addr.sin_addr,
            &((struct sockaddr_in *) res->ai_addr)->sin_addr,
```

```
                    sizeof( struct in_addr ) );
            freeaddrinfo( res );
        }

    memset(&(their_addr.sin_zero), '\0', 8);

    if (connect(sockfd,
                (struct sockaddr *)&their_addr,
                sizeof(struct sockaddr)) == -1) {
        perror("connect");
        exit(1);
    }

    if ((numbytes=recv(sockfd, buf,
                       MAXDATASIZE-1, 0)) == -1) {
        perror("recv");
        exit(1);
    }

    buf[numbytes] = '\0';

    printf("Received: %s",buf);

    close(sockfd);

    return 0;
}
```

Listing 11: Servidor com resolução de IPs para nomes

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

#define MYPORT 3490
#define BACKLOG 10

int main(void)
{
    int sockfd, new_fd;
    struct sockaddr_in my_addr;
    struct sockaddr_in their_addr;
    int sin_size;
```

```
    char host[ NI_MAXHOST ];

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    my_addr.sin_family = AF_INET;
    my_addr.sin_port = htons(MYPORT);
    my_addr.sin_addr.s_addr = INADDR_ANY;
    memset(&(my_addr.sin_zero), '\0', 8);

    // lose the pesky "address already in use" error message
    {
      int yes=1;
      if (setsockopt(sockfd , SOL_SOCKET, SO_REUSEADDR, &yes,
                     sizeof(int)) == -1) {
        perror("setsockopt");
        exit(1);
      }
    }



    if (bind(sockfd, (struct sockaddr *)&my_addr,
             sizeof(struct sockaddr)) == -1) {
        perror("bind");
        exit(1);
    }

    if (listen(sockfd, BACKLOG) == -1) {
        perror("listen");
        exit(1);
    }

    while(1) {  // main accept() loop
        sin_size = sizeof(struct sockaddr_in);
        if ((new_fd = accept(sockfd,
                             (struct sockaddr *)&their_addr,
                             &sin_size)) == -1) {
            perror("accept");
            continue;
        }

        /* Find out the client host name */
        if ( getnameinfo( ( struct sockaddr * ) &their_addr,
                          sizeof( their_addr ),
                          host, NI_MAXHOST, NULL, 0, 0) != 0)
          {
            perror( "getnameinfo" );
```

```
            strncpy( host, "<UNKNOW_ERROR>", NI_MAXHOST - 1 );
          }

        printf( "server: got connection from %s (%s)\n",
                inet_ntoa( their_addr.sin_addr ),
                host );


        if (send(new_fd, "Hello, world!\n", 14, 0) == -1)
          perror("send");
        close(new_fd);
    }
    return 0;
}
```