MC823 Atividade 2

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

1 Introdução

Nesta atividade abordamos vários temas, dentre eles **Fronteiras de mensagens no TCP**, **Conseguindo dados da conexão com getpeername() e getsockname()**, **Transmitindo dados binários** e **Servidor Concorrente**.

2 Fronteiras de mensagens no TCP

Nesta parte noticiamos que `send()` e `recv()` não são necessariamente correspondentes. Para tal foi implementado um servidor *discard* que pode receber 32Kb+1byte de uma só vez (com um só `recv()`) e um cliente que envia 32Kb+1byte para o servidor.

Foi possível notar que o `send()` enviava todos os dados, independente da plataforma e da rede, enquanto que o `recv()` recebia somente parte dos dados na maioria das vezes, necessitando de vários para receber todos os dados.

Os dados experimentais estão listados na Tabela 2 e os códigos estão em Listagem 17 (servidor) e Listagem 18 (cliente). O resultado dos comandos são encontrados nas Listagens: 2 (mesmo hospedeiro, linux), 3 (hospedeiros diferentes, linux), 4 (mesmo hospedeiro, linux), 5 (solaris para linux), 6 (linux para solaris).

	Linux 1	Linux 2	Solaris 1	Solaris 2
Linux 1	1..3	23	20	—
Linux 2	—	—	—	—
Solaris 1	12	—	2..3	—
Solaris 2	—	—	—	—

Tabela 1: Número de `recv()`s necessários para conseguir 32Kb+1b

3 `getsockname()`/`getpeername()` e números de portas

Nesta parte modificamos o código do *"hello world"* da atividade 1 para que o cliente e servidor imprimissem os dados referentes ao endereço e porta do cliente. Para isso foram utilizadas as funções `getsockname()` (cliente) e `getpeername()` (servidor).

Depois modificamos o cliente para utilizar `bind()` e então escolher uma porta local para estabelecer a conexão com o servidor.

Dos testes requisitados, os resultados foram:

- Foi possível rodar o cliente com a porta 2222, a qual não estava em uso no momento.
- Ao tentar utilizar uma porta a qual já estava sendo utilizada, o erro foi: `bind: Address already in use`.
- Ao tentar utilizar uma porta baixa (< 1024), o erro foi: `bind: Permission denied`.

O código necessário para permitir ao cliente escolher a porta, em resumo, é evidenciado na Listagem 1. O código completo se encontram em Listagem 20 (cliente) e Listagem 19 (servidor).

Listing 1: Código necessário para o cliente escolher a porta.

```
my_addr.sin_family      = AF_INET;
my_addr.sin_port        = htons( localport );
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 );
}
```

4 Transmitindo dados binários

Nesta parte experimentamos a transmissão de dados binários sem usar a conversão dos dados entre o modo do hospedeiro (*host*) e da rede (*network*) com as funções `htonl()`, `ntohl()` e afins.

Noticiamos que transmissões entre a mesma plataforma não aparentam problemas, porém a transmissão entre diferentes pode ocasionar interpretar *big-endian* como *little-endian* e vice versa. Vide as Listagens 7 (Intel para Intel), 8 (Sparc para Sparc), 9 (Sparc para Intel), e 10 (Intel para Sparc) para os resultados obtidos.

O código do cliente está em Listagem 21 e o do servidor está em Listagem 22.

5 Servidor Concorrente

Nesta parte verificamos que o servidor de *echo* na atividade 1 não aceita várias conexões simulatâneas, então este foi mudado para trabalhar com um novo processo para cada nova conexão.

Antes de usar `fork()` para tratar cada nova conexão, apos a primeira, as demais ficavam esperando ate que esta termine para que sejam tratadas. O comando `netstat -ant` as lista como estabelecidas, porem ficam travadas. Com o novo codigo elas são tratadas simulataneamente, cada uma por um processo.

O código do novo servidor se encontra na Listagem 23.

O estado do sistema foi anotado para confirmar o funcionamento correto do programa. Os resultados do `netstat -ant` estão nas Listagens 11 (antes), 12 (durante) e 13 (depois). Os resultados do `ps aux` estão nas Listagens 14 (antes), 15 (durante) e 14 (depois).

6 Apêncice

6.1 Registro dos resultados

Listing 2: Diferença entre `send()` e `recv()` em um mesmo hospedeiro linux

```
linux01$ ./serv-32kb
server: got connection from 127.0.0.1
         | received 32769 bytes from 32769 requested (100.00%).
         `-> got 32769 bytes with 1 recv()s
server: closed connection to 127.0.0.1

...

server: got connection from 127.0.0.1
         | received 16383 bytes from 32769 requested (50.00%).
         | received 16383 bytes from 16386 requested (99.98%).
         | received 3 bytes from 3 requested (100.00%).
         `-> got 32769 bytes with 3 recv()s
server: closed connection to 127.0.0.1


linux01$ ./cli-32kb 127.0.0.1
sent 32769 bytes from 32769.

...

sent 32769 bytes from 32769.
```

Listing 3: Diferença entre `send()` e `recv()` em diferentes hospedeiros linux

```
linux01$ ./serv-32kb
server: got connection from 200.174.211.28
         | received 1448 bytes from 32769 requested (4.42%).
         | received 1448 bytes from 31321 requested (4.62%).
         | received 1448 bytes from 29873 requested (4.85%).
         | received 1448 bytes from 28425 requested (5.09%).
         | received 1448 bytes from 26977 requested (5.37%).
```

```
         | received 1448 bytes from 25529 requested (5.67%).
         | received 1448 bytes from 24081 requested (6.01%).
         | received 1448 bytes from 22633 requested (6.40%).
         | received 1448 bytes from 21185 requested (6.84%).
         | received 1448 bytes from 19737 requested (7.34%).
         | received 1448 bytes from 18289 requested (7.92%).
         | received 1448 bytes from 16841 requested (8.60%).
         | received 1448 bytes from 15393 requested (9.41%).
         | received 1448 bytes from 13945 requested (10.38%).
         | received 1448 bytes from 12497 requested (11.59%).
         | received 1448 bytes from 11049 requested (13.11%).
         | received 1448 bytes from 9601 requested (15.08%).
         | received 1448 bytes from 8153 requested (17.76%).
         | received 1448 bytes from 6705 requested (21.60%).
         | received 1448 bytes from 5257 requested (27.54%).
         | received 1448 bytes from 3809 requested (38.02%).
         | received 1448 bytes from 2361 requested (61.33%).
         | received 913 bytes from 913 requested (100.00%).
         `-> got 32769 bytes with 23 recv()s
server: closed connection to 200.174.211.28



linux02$ ./cli-32kb 143.106.24.52
sent 32769 bytes from 32769.
```

Listing 4: Diferença entre `send()` e `recv()` em um mesmo hospedeiro Solaris

```
solaris01$ ./serv-32kb
server: got connection from 127.0.0.1
         | received 8192 bytes from 32769 requested (25.00%).
         | received 24577 bytes from 24577 requested (100.00%).
         `-> got 32769 bytes with 2 recv()s
server: closed connection to 127.0.0.1
server: got connection from 127.0.0.1
         | received 8192 bytes from 32769 requested (25.00%).
         | received 16384 bytes from 24577 requested (66.66%).
         | received 8193 bytes from 8193 requested (100.00%).
         `-> got 32769 bytes with 3 recv()s
server: closed connection to 127.0.0.1


solaris01$ $ ./cli-32kb 127.0.0.1
sent 32769 bytes from 32769.

solaris01$ $ ./cli-32kb 127.0.0.1
sent 32769 bytes from 32769.
```

Listing 5: Diferença entre `send()` e `recv()` com Solaris enviando para Linux

```
linux01$ ./serv-32kb
server: got connection from 143.106.7.16
         | received 1460 bytes from 32769 requested (4.46%).
         | received 1460 bytes from 31309 requested (4.66%).
         | received 1460 bytes from 29849 requested (4.89%).
         | received 1460 bytes from 28389 requested (5.14%).
         | received 1460 bytes from 26929 requested (5.42%).
         | received 2920 bytes from 25469 requested (11.46%).
         | received 2920 bytes from 22549 requested (12.95%).
         | received 1460 bytes from 19629 requested (7.44%).
         | received 1460 bytes from 18169 requested (8.04%).
         | received 1460 bytes from 16709 requested (8.74%).
         | received 1460 bytes from 15249 requested (9.57%).
         | received 1460 bytes from 13789 requested (10.59%).
         | received 1460 bytes from 12329 requested (11.84%).
         | received 1460 bytes from 10869 requested (13.43%).
         | received 1460 bytes from 9409 requested (15.52%).
         | received 1460 bytes from 7949 requested (18.37%).
         | received 1460 bytes from 6489 requested (22.50%).
         | received 1460 bytes from 5029 requested (29.03%).
         | received 1460 bytes from 3569 requested (40.91%).
         | received 2109 bytes from 2109 requested (100.00%).
         `-> got 32769 bytes with 20 recv()s
server: closed connection to 143.106.7.16



solaris01$ $ ./cli-32kb 143.106.24.52
sent 32769 bytes from 32769.
```

Listing 6: Diferença entre `send()` e `recv()` com Linux enviando para Solaris

```
solaris01$ ./serv-32kb
server: got connection from 143.106.24.52
         | received 1448 bytes from 32769 requested (4.42%).
         | received 1448 bytes from 31321 requested (4.62%).
         | received 2896 bytes from 29873 requested (9.69%).
         | received 4344 bytes from 26977 requested (16.10%).
         | received 1448 bytes from 22633 requested (6.40%).
         | received 1448 bytes from 21185 requested (6.84%).
         | received 4344 bytes from 19737 requested (22.01%).
         | received 1448 bytes from 15393 requested (9.41%).
         | received 4344 bytes from 13945 requested (31.15%).
         | received 4344 bytes from 9601 requested (45.25%).
         | received 1448 bytes from 5257 requested (27.54%).
         | received 3809 bytes from 3809 requested (100.00%).
         `-> got 32769 bytes with 12 recv()s

linux01$ ./cli-32kb 143.106.7.16
sent 32769 bytes from 32769.
```

Listing 7: Transmissão binária de Intel para Intel, sem `htonl()`

```
linux-intel01 $ ./cli 127.0.0.1 12345
Sent '12345'.

linux-intel01 $ ./serv
server: got connection from 127.0.0.1
Received '12345'
```

Listing 8: Transmissão binária de Sparc para Sparc, sem `htonl()`

```
solaris-sparc01 $ ./cli 127.0.0.1 12345
Sent '12345'.

solaris-sparc01 $ ./serv
server: got connection from 127.0.0.1
Received '12345'
```

Listing 9: Transmissão binária de Sparc para Intel, sem `htonl()`

```
solaris-sparc01 $ ./cli 143.106.24.52 12345
Sent '12345'.

linux-intel01   $ ./serv
server: got connection from 143.106.7.16
Received '959447040'
```

Listing 10: Transmissão binária de Intel para Sparc, sem `htonl()`

```
linux-intel01   $ ./cli 143.106.7.16 12345
Sent '12345'.

solaris-sparc01 $ ./serv
server: got connection from 143.106.24.52
Received '959447040'
```

Listing 11: Servidor Concorrente: netstat antes de conexões

```
gustavo@ltc08 gustavo $ netstat  -ant | grep 3490
tcp        0      0 0.0.0.0:3490            0.0.0.0:*               LISTEN
```

Listing 12: Servidor Concorrente: netstat durante conexões

```
gustavo@ltc08 gustavo $ netstat  -ant | grep 3490
tcp        0      0 0.0.0.0:3490            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:3490          127.0.0.1:32772         ESTABLISHED
tcp        0      0 127.0.0.1:3490          127.0.0.1:32771         ESTABLISHED
tcp        0      0 127.0.0.1:32772         127.0.0.1:3490          ESTABLISHED
tcp        0      0 127.0.0.1:32771         127.0.0.1:3490          ESTABLISHED
```

Listing 13: Servidor Concorrente: netstat depois das conexões

```
gustavo@ltc08 gustavo $ ps aux | grep "./serv"
gustavo   4907  0.0  0.0  1288  320 pts/1    S    17:26   0:00 ./serv
```

Listing 14: Servidor Concorrente: ps antes de conexões

```
gustavo@ltc08 gustavo $ ps aux | grep "./serv"
gustavo   4907  0.0  0.0  1284  240 pts/1    S    17:26   0:00 ./serv
```

Listing 15: Servidor Concorrente: ps durante conexões

```
gustavo@ltc08 gustavo $ ps aux | grep "./serv"
gustavo   4907  0.0  0.0  1288  320 pts/1    S    17:26   0:00 ./serv
gustavo   4965  0.0  0.0  1288  320 pts/1    S    17:29   0:00 ./serv
gustavo   4967  0.0  0.0  1288  320 pts/1    S    17:29   0:00 ./serv
```

Listing 16: Servidor Concorrente: ps depois das conexões

```
gustavo@ltc08 p4 $ ps aux | grep "./serv"
gustavo   4907  0.0  0.0  1288  320 pts/1    S    17:26   0:00 ./serv
```

6.2 Códigos Usados

Listing 17: Servidor discard 32kb+1b

```
#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
#define DATASIZE (32 * 1024 + 1)

int main( void )
{
  int sockfd, new_fd;
  struct sockaddr_in my_addr;
  struct sockaddr_in their_addr;
  int sin_size, i, total, n;
  char buf[ DATASIZE + 1 ];
```

```
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 )
  {
      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 ) );

      total = DATASIZE;
      i     = 0;
      n     = 0;
      while ( total > 0 )
        {
          if ( ( i = recv( new_fd, buf, total, 0 ) ) == -1 )
            {
              perror( "recv" );
              exit( 1 );
            }
          else if ( i == 0 )
```

```
            {
              printf( "peer closed connection\n" );
              break;
            }
          else
            {
              printf( "\t | received %d bytes from %d " \
                      "requested (%0.2f%%).\n",
                      i, total, 100.0f * i / total );
              total -= i;
              n ++;
            }
        }

      printf( "\t `-> got %d bytes with %d recv()s\n",
              DATASIZE, n );

      printf( "server: closed connection to %s\n",
              inet_ntoa( their_addr.sin_addr ) );

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

Listing 18: Cliente 32kb+1b

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

#define PORT     3490
#define DATASIZE (32 * 1024 + 1)

int main( int argc, char *argv[] )
{
  int sockfd, i;
  char buf[ DATASIZE + 1 ];
  struct sockaddr_in their_addr;

  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 );
    }

  for ( i = 0; i < DATASIZE; i ++ )
    {
      buf[ i ] = i % 26 + 'a';
    }

    buf[ DATASIZE ] = 0;


  if ( ( i = send( sockfd, buf, DATASIZE, 0 ) ) == -1 )
    {
      perror( "send" );
      exit( -1 );
    }

  printf( "sent %d bytes from %d.\n", i, DATASIZE );

  close(sockfd);

  return 0;
}
```

Listing 19: Servidor com `getpeername()`

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
```

```c
#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;

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

```
      {
        struct sockaddr_in name;
        sin_size = sizeof( struct sockaddr_in );
        if ( getpeername( new_fd, (struct sockaddr *) &name,
                          &sin_size ) == -1 )
          {
            perror( "getpeername" );
            exit( 1 );
          }

        printf( "server: got connection from %s:%d\n",
                inet_ntoa( name.sin_addr ),
                ntohs( name.sin_port ) );
      }

      /*
        printf( "server: got connection from %s:%d\n",
              inet_ntoa( their_addr.sin_addr ),
              ntohs( their_addr.sin_port ) );
      */

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

Listing 20: Cliente com `getsockname()`

```
#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>

#define PORT        3490
#define MAXDATASIZE 100

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

```
  if ( argc != 3 ) {
    fprintf( stderr, "usage: client hostip localport\n" );
    exit( 1 );
  }

  if ( ( localport = atoi( argv[ 2 ] ) ) == 0 )
    {
      fprintf( stderr, "ERROR: localport should be an "\
               "integer, '%s' was given.\n", argv[ 2 ] );
      exit( 1 );
    }

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

  my_addr.sin_family      = AF_INET;
  my_addr.sin_port        = htons( localport );
  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 );
    }



  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 );
    }

  {
```

```
    struct sockaddr_in name;
    int sin_size = sizeof( struct sockaddr_in );
    if ( getsockname( sockfd, (struct sockaddr *) &name,
                      &sin_size ) == -1 )
      {
        perror( "getsockname" );
        exit( 1 );
      }

    printf( "client: connected using local %s:%d\n",
            inet_ntoa( name.sin_addr ), ntohs( name.sin_port));

  }


  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 21: Cliente que transmite um inteiro sem `htonl()`

```
#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>

#define PORT        3490

int main( int argc, char *argv[] )
{
  int sockfd;
  int n;
```

```
struct sockaddr_in their_addr;

if ( argc != 3 ) {
  fprintf( stderr, "usage: client hostip number\n" );
  exit( 1 );
}

if ( ( n = atoi( argv[ 2 ] ) ) == 0 )
  {
    fprintf( stderr, "ERROR: number should be > 0\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 );
  }


if ( send( sockfd, &n, sizeof( n ), 0 ) == -1 )
  {
    perror( "send" );
    exit( 1 );
  }
else
  {
    printf( "Sent '%d'.\n", n );
  }

close( sockfd );
```

```
  return 0;
}
```

Listing 22: Servidor que recebe um inteiro sem `ntohl()`

```
#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, n, i;

  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 )
```

```
    {
      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 ) );

      if ( ( i = recv( new_fd, &n, sizeof( n ), 0 ) ) == -1 )
        {
          perror( "recv" );
        }
      else if ( i == 0 )
        {
          fprintf( stderr, "connection closed by peer.\n" );
        }
      else
        {
          printf( "Received '%d'\n", n );
        }

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

Listing 23: Servidor de *Echo* Concorrente

```
#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 <sys/wait.h>

#ifndef NO_SIGCHLD_HANDLER
#include <signal.h>
void sigchld_handler( int a )
{
  while ( waitpid( -1, NULL, WNOHANG ) > 0 );
}
#endif /* NO_SIGCHLD_HANDLER */
```

```
#define MYPORT  3490
#define BACKLOG 1

inline void doit( int connfd )
{
  char input;
  int recvd;
  while ( 1 )
    {
      recvd = recv( connfd, &input, 1, 0 );
      if (  recvd == -1  )
        {
          perror( "recv" );
          break;
        }
      else if (  recvd == 0  )
        {
          struct sockaddr_in name;
          unsigned int sin_size = sizeof( struct sockaddr_in );
          if ( getpeername( connfd, (struct sockaddr *) &name,
                            &sin_size ) == -1 )
            {
              perror( "getpeername" );
              exit( 1 );
            }

          printf( "server: closing connection  %s:%d\n",
                  inet_ntoa( name.sin_addr),
                  ntohs( name.sin_port ) );
          break;
        }
      if (  send(  connfd, &input, 1, 0  ) == -1  )
        {
          perror( "send" );
          break;
        }
    }
}

inline void forkit( int listenfd, int connfd )
{
  int pid;
  if ( ( pid = fork() ) == 0 )
    {
      close( listenfd );
      doit( connfd );
      close( connfd );
      exit( 0 );
    }
  else if ( pid == -1 )
```

```
    {
      perror( "fork" );
    }
  close( connfd );
}

int main( void )
{
    int sockfd, new_fd;
    struct sockaddr_in my_addr;
    struct sockaddr_in their_addr;
    unsigned int sin_size;
#ifndef NO_SIGCHLD_HANDLER
    void (*sigchld_bkp)(int) ;
    if ( ( sigchld_bkp =
            signal( SIGCHLD, sigchld_handler ) ) == SIG_ERR )
      {
        fprintf( stderr, "ERROR: could not change "\
                 "signal SIGCHLD handler!\n" );
        exit( 1 );
      }
#endif /* NO_SIGCHLD_HANDLER */
    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 )
      {
        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:%d\n",
                inet_ntoa( their_addr.sin_addr ),
                ntohs( their_addr.sin_port) );
        forkit( sockfd, new_fd );
      }
#ifndef NO_SIGCHLD_HANDLER
    signal( SIGCHLD, sigchld_bkp );
#endif /* NO_SIGCHLD_HANDLER */
    return 0;
}
```