MC833 A - Programação de redes de computadores
Relatório - Tarefa 07

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

16 de junho de 2016

Prof. Paulo Lício de Geus
IC – UNICAMP

Sumário

Introdução

Nesta tarefa construiremos um servidor que escuta chamadas utilizando tanto o protocolo UDP quanto o protocolo TCP, com o auxílio do comando `select`. Além disso, será feito um cliente que poderá se conectar ao servidor também utilizando os dois tipos de conexão, a depender da opção do usuário.

1 Questão 1

A função desejada está definida em ⟨*sys*/*socket.h*⟩.

- `setsockopt`

  Assinatura:

  ```
  int setsockopt(int sockfd, int level, int optname,
                      const void *optval, socklen_t optlen);
  ```

  A função `setsockopt` é usada para manipular opções para o socket referenciado por `sockfd`. As opções podem existir em múltiplos níveis de protocolo, e estão sempre presentes no nível mais superior de protocolos.

  Dessa maneira, é necessário definir o nível em que a opção desejada reside, o que é feito por meio do parâmetro `level`. Para manipular opções no nível da API de sockets, a variável deve ser setada como `SOL_SOCKET`. Para manipular opções de outro nível, deve-se alterar o valor de `level` para o número de protocolo correspondente.

  Uma lista de opções que podem ser utilizadas em `optname` estão listadas a seguir. A página de manual do Linux sobre sockets pode ser acessada para uma lista completa, por meio do comando `man 7 socket`.

  1. `SO_BINDTODEVICE`: Utilizado para ligar o socket a um nome simbólico de dispositivo, como `eth0`, em vez de utilizar um endereço de IP.
  2. `SO_REUSEADDR`: Permite que outros sockets se liguem à mesma porta, a não ser que já haja um socket ativamente escutando conexões nessa porta. Isso funciona como uma forma de circundar mensagens de erro como "Address already in use." obtidas após uma tentativa de reiniciar o servidor.
  3. `SO_BROADCAST`: Permite que datagramas UDP enviem e recebam pacotes utilizando o endereço de transmissão.

  O argumento `optval` é usualmente um ponteiro para um inteiro que indica o valor em questão, enquanto `optlen` contém o tamanho de `optval` (usualmente, é setado para `sizeof(int)`).

  A função retorna zero quando bem sucedida e -1 caso encontre erros, setando `errno` com informações referentes ao tipo de erro.

2 Questão 2

Foi implementado um servidor de eco utilizando a função `select` para multiplexar entre um socket UDP e um socket TCP, utilizando o mesmo número de porta.

O socket TCP foi programado para permitir que outros sockets possam utilizar a mesma porta, por meio da opção `SO_REUSEADDR`.

```
    /* configure the TCP socket to reuse address */
    if(setsockopt(st, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(optval)) < 0) {
        perror("server: setsockopt");
        exit(1);
    }
```

O laço principal do programa está copiado a seguir.

```
for (;;) {

    /* select to get TCP or UDP incoming requests */
    if((nready = select(su+1, &rset, NULL, NULL, NULL)) < 0) {
```

```
        perror("server: select");
        close(st);
        close(su);
        exit(1);
    }

    /* TCP connection */
    if(FD_ISSET(st, &rset)) {
        ulen = sizeof(sin);

        if((sconn = accept(st, (struct sockaddr *)&sin, &ulen)) < 0) {
            perror("server: accept");
            close(st);
            close(su);
            exit(1);
        }

        /* fork a process for the child */
        if((pid = fork()) == 0) {
            while((slen = recv(sconn, buf, sizeof(buf), 0))) {
                if(slen < 0) {
                    perror("server: recv");
                }
                else {
                    fprintf(stdout, "%s\n", buf);
                    if(send(sconn, buf, strlen(buf)+1, 0) < 0) {
                        perror("server: send");
                    }
                }
            }

            close(sconn);
        }
        else {
            close(sconn);
        }
    }

    /* UDP request */
    else if(FD_ISSET(su, &rset)) {
        if((slen = recvfrom(su, buf, sizeof(buf), 0,
                    (struct sockaddr *)&peer, &peerlen)) < 0) {
            perror("server: recvfrom");
        }
        else {
            fprintf(stdout, "%s\n", buf);
            if(sendto(su, buf, strlen(buf)+1, 0,
                        (struct sockaddr *)&peer, peerlen) < 0) {
                perror("server: sendto");
            }
        }
    }

    FD_SET(st, &rset);
    FD_SET(su, &rset);
}
```

O comando `select` multiplexa entre os dois sockets, e, utilizando `FD_ISSET`, que testa se um descritor de arquivo faz parte do conjunto de descritores apontados por `rset`, o programa lida com o tipo específico de cliente. O tratamento dado a cada tipo de cliente é semelhante ao verificado nas tarefas anteriores.

3 Questão 3

O cliente foi implementado com a opção de se comunicar com o servidor tanto utilizando o protocolo TCP quanto o protocolo UDP. Para isso, ele deve ser invocado por meio da seguinte sequência de argumentos:

`client <mode> <host>`.

Dependendo da opção escolhida pelo usuário, o cliente se comporta de maneira semelhante a um dos clientes TCP e UDP das tarefas anteriores.

Para testar o funcionamento do servidor, foi utilizado o comando `tcpdump`. Após inicializar o servidor, enviamos a mensagem "Testing TCP" utilizando a versão TCP do cliente programado para esta tarefa. A saída do cliente está listada a seguir.

```
niko@ubuntu:~/Desktop/mc833/t7$ ./client.out -t localhost
Testing TCP
Testing TCP
```

Em seguida, enviamos a mensagem "Testing UDP" utilizando a versão UDP do cliente programado para esta tarefa. A saída do cliente está listada a seguir.

```
niko@ubuntu:~/Desktop/mc833/t7$ ./client.out -u localhost
Testing UDP
Testing UDP
```

Depois disso, utilizamos a ferramenta `netcat` para enviar a mensagem "Testing netcat" ao servidor. A saída do terminal está listada a seguir.

```
niko@ubuntu:~/Desktop/mc833/t7$ nc -u localhost 31472
Testing netcat
Testing netcat
```

Por fim, foi utilizado o cliente da tarefa 4 para enviar a mensagem "Testing client4". A saída está mostrada a seguir.

```
niko@ubuntu:~/Desktop/mc833/t4$ ./client.out localhost
--------------------
IP local: 127.0.0.1
Porta local: 51118
--------------------

Testing client4
Testing client4
```

A saída do servidor está copiada a seguir.

```
niko@ubuntu:~/Desktop/mc833/t7$ ./server.out
Testing TCP

Testing UDP

Testing netcat

Testing client4
```

A saída do comando `tcpdump`, que monitorou toda a comunicação, está apresentada a seguir.

```
niko@ubuntu:~/Desktop/mc833$ 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 262144 bytes
05:14:50.457991 IP localhost.51116 > localhost.31472: Flags [S], seq 2765961544, win 43690, options
    [mss 65495,sackOK,TS val 39537746 ecr 0,nop,wscale 7], length 0
05:14:50.458002 IP localhost.31472 > localhost.51116: Flags [S.], seq 2142835947, ack 2765961545, win
    43690, options [mss 65495,sackOK,TS val 39537746 ecr 39537746,nop,wscale 7], length 0
05:14:50.458013 IP localhost.51116 > localhost.31472: Flags [.], ack 1, win 342, options [nop,nop,TS
    val 39537746 ecr 39537746], length 0
05:14:53.503043 IP localhost.51116 > localhost.31472: Flags [P.], seq 1:14, ack 1, win 342, options
    [nop,nop,TS val 39538508 ecr 39537746], length 13
05:14:53.503067 IP localhost.31472 > localhost.51116: Flags [.], ack 14, win 342, options [nop,nop,TS
    val 39538508 ecr 39538508], length 0
05:14:53.503141 IP localhost.31472 > localhost.51116: Flags [P.], seq 1:14, ack 14, win 342, options
```

```
    [nop,nop,TS val 39538508 ecr 39538508], length 13
05:14:53.503150 IP localhost.51116 > localhost.31472: Flags [.], ack 14, win 342, options [nop,nop,TS
    val 39538508 ecr 39538508], length 0
05:15:03.690487 IP localhost.34681 > localhost.31472: UDP, length 13
05:15:03.690560 IP localhost.31472 > localhost.34681: UDP, length 13
05:15:15.673442 IP localhost.60981 > localhost.31472: UDP, length 15
05:15:15.673494 IP localhost.31472 > localhost.60981: UDP, length 16
05:15:22.954370 IP localhost.51118 > localhost.31472: Flags [S], seq 327063604, win 43690, options
    [mss 65495,sackOK,TS val 39545871 ecr 0,nop,wscale 7], length 0
05:15:22.954382 IP localhost.31472 > localhost.51118: Flags [S.], seq 4170852344, ack 327063605, win
    43690, options [mss 65495,sackOK,TS val 39545871 ecr 39545871,nop,wscale 7], length 0
05:15:22.954393 IP localhost.51118 > localhost.31472: Flags [.], ack 1, win 342, options [nop,nop,TS
    val 39545871 ecr 39545871], length 0
05:15:32.027797 IP localhost.51118 > localhost.31472: Flags [P.], seq 1:18, ack 1, win 342, options
    [nop,nop,TS val 39548139 ecr 39545871], length 17
05:15:32.027820 IP localhost.31472 > localhost.51118: Flags [.], ack 18, win 342, options [nop,nop,TS
    val 39548139 ecr 39548139], length 0
05:15:32.027879 IP localhost.31472 > localhost.51118: Flags [P.], seq 1:18, ack 18, win 342, options
    [nop,nop,TS val 39548139 ecr 39548139], length 17
05:15:32.027888 IP localhost.51118 > localhost.31472: Flags [.], ack 18, win 342, options [nop,nop,TS
    val 39548139 ecr 39548139], length 0
```

As primeiras linhas correspodem à comunicação entre o cliente TCP da tarefa 7 e o servidor:

```
05:14:50.457991 IP localhost.51116 > localhost.31472: Flags [S], seq 2765961544, win 43690, options
    [mss 65495,sackOK,TS val 39537746 ecr 0,nop,wscale 7], length 0
05:14:50.458002 IP localhost.31472 > localhost.51116: Flags [S.], seq 2142835947, ack 2765961545, win
    43690, options [mss 65495,sackOK,TS val 39537746 ecr 39537746,nop,wscale 7], length 0
05:14:50.458013 IP localhost.51116 > localhost.31472: Flags [.], ack 1, win 342, options [nop,nop,TS
    val 39537746 ecr 39537746], length 0
05:14:53.503043 IP localhost.51116 > localhost.31472: Flags [P.], seq 1:14, ack 1, win 342, options
    [nop,nop,TS val 39538508 ecr 39537746], length 13
05:14:53.503067 IP localhost.31472 > localhost.51116: Flags [.], ack 14, win 342, options [nop,nop,TS
    val 39538508 ecr 39538508], length 0
05:14:53.503141 IP localhost.31472 > localhost.51116: Flags [P.], seq 1:14, ack 14, win 342, options
    [nop,nop,TS val 39538508 ecr 39538508], length 13
05:14:53.503150 IP localhost.51116 > localhost.31472: Flags [.], ack 14, win 342, options [nop,nop,TS
    val 39538508 ecr 39538508], length 0
```

As 2 linhas seguintes são referentes ao cliente UDP da tarefa 7.

```
05:15:03.690487 IP localhost.34681 > localhost.31472: UDP, length 13
05:15:03.690560 IP localhost.31472 > localhost.34681: UDP, length 13
```

As próximas 2 linhas são referentes ao uso de `netcat`.

```
05:15:15.673442 IP localhost.60981 > localhost.31472: UDP, length 15
05:15:15.673494 IP localhost.31472 > localhost.60981: UDP, length 16
```

Por fim, as últimas linhas se referem ao uso do cliente da tarefa 4.

```
05:15:22.954370 IP localhost.51118 > localhost.31472: Flags [S], seq 327063604, win 43690, options
    [mss 65495,sackOK,TS val 39545871 ecr 0,nop,wscale 7], length 0
05:15:22.954382 IP localhost.31472 > localhost.51118: Flags [S.], seq 4170852344, ack 327063605, win
    43690, options [mss 65495,sackOK,TS val 39545871 ecr 39545871,nop,wscale 7], length 0
05:15:22.954393 IP localhost.51118 > localhost.31472: Flags [.], ack 1, win 342, options [nop,nop,TS
    val 39545871 ecr 39545871], length 0
05:15:32.027797 IP localhost.51118 > localhost.31472: Flags [P.], seq 1:18, ack 1, win 342, options
    [nop,nop,TS val 39548139 ecr 39545871], length 17
05:15:32.027820 IP localhost.31472 > localhost.51118: Flags [.], ack 18, win 342, options [nop,nop,TS
    val 39548139 ecr 39548139], length 0
05:15:32.027879 IP localhost.31472 > localhost.51118: Flags [P.], seq 1:18, ack 18, win 342, options
    [nop,nop,TS val 39548139 ecr 39548139], length 17
05:15:32.027888 IP localhost.51118 > localhost.31472: Flags [.], ack 18, win 342, options [nop,nop,TS
    val 39548139 ecr 39548139], length 0
```