MC833 A - Programação de redes de computadores
Relatório - Tarefa 06

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

9 de junho de 2016

Prof. Paulo Lício de Geus
IC – UNICAMP

Sumário

Introdução

Nesta tarefa aprenderemos sobre sockets UDP. Além disso, serão construídos um servidor de eco e um cliente UDP.

Parte A
Cliente-Servidor de eco com sockets UDP

1 Questão 1

Todas as funções e macros estão definidas em ⟨*sys/socket.h*⟩.

- `recvfrom`

  Assinatura:

```
ssize_t recvfrom(int socket, void *restrict buffer, size_t length,
      int flags, struct sockaddr *restrict address,
      socklen_t *restrict address_len);
```

  A função `recvfrom` é usada para receber mensagens de um socket, tanto orientados a conexão como não orientados a conexão. No entanto, ela é principalmente utilizada em sockets não orientados a conexão, pois permite à aplicação recuperar o endereço do remetente da mensagem.

  O tipo `ssize_t` representa o tamanho de blocos que podem ser lidos/escritos em uma única operação. É um inteiro com sinal de pelo menos 16 bits.

  Seus argumentos são:

  a) *socket*
  Especifica o socket a ser utilizado para a comunicação.

  b) *buffer*
  É um apontador para o buffer que receberá a mensagem.

  c) *length*
  Especifica o tamanho (em bytes) do buffer apontado por *buffer*. É o tamanho máximo de mensagem que o buffer consegue armazenar.

  d) *flags*
  Especifica o tipo de recebimento de mensagem. As opções são `MSG_PEEK`, `MSG_OOB` e `MSG_WAITALL`.

  e) *address*
  Pode conter um apontador nulo ou um ponteiro para uma estrutura `sockaddr` onde o endereço do remetente será armazenado.

  f) *address_len*
  Especifica o tamanho da estrutura `sockaddr` apontada pelo argumento *address*.

  O valor de retorno é o tamanho da mensagem (em bytes). Se nenhuma mensagem estiver disponível, mas a conexão for encerrada corretamente, o valor de retorno é zero. Caso contrário, a função retorna -1 e `errno` é setado para indicar o tipo de erro.

- `sendto`

  Assinatura:

```
ssize_t sendto(int socket, const void *message, size_t length,
      int flags, const struct sockaddr *dest_addr,
      socklen_t dest_len);
```

  A função `sendto` é usada para enviar mensagens por meio de um socket, tanto orientados a conexão como não orientados a conexão. Caso o socket não seja orientado a conexão, a mensagem será enviada ao endereço especificado em *dest_addr*; caso ele seja, esse argumento é ignorado.

  Seus argumentos são:

a) *socket*
Especifica o socket a ser utilizado para a comunicação.

b) *message*
É um apontador para o buffer contendo a mensagem a ser enviada.

c) *length*
Especifica o tamanho da mensagem.

d) *flags*
Especifica o tipo de recebimento de mensagem. As opções são `MSG_EOR` e `MSG_OOB`.

e) *dest_addr*
É um ponteiro para uma estrutura `sockaddr` onde o endereço de destino está armazenado.

f) *dest_len*
Especifica o tamanho da estrutura `sockaddr` apontada pelo argumento *dest_addr*.

O valor de retorno é o tamanho da mensagem enviada (em bytes). Caso contrário, a função retorna -1 e `errno` é setado para indicar o tipo de erro.

2 Questão 2

Foram implementados um cliente e um servidor de eco usando o protocolo UDP. Testes usando o binário do cliente e o `netcat` são mostrados abaixo:

- Cliente UDP 1

```
bash-4.3$ ./client_udp.out localhost
Oi do cliente 1
Oi do cliente 1
```

- Cliente UDP 2

```
bash-4.3$ ./client_udp.out localhost
Oi do cliente 2
Oi do cliente 2
```

- Cliente netcat

```
bash-4.3$ nc -u localhost 31472
Oi do netcat
Oi do netcat
```

- Servidor UDP

```
bash-4.3$ ./server_udp.out
Oi do cliente 1

Oi do cliente 2

Oi do netcat
```

Assim, verificamos que o servidor suporta atendimento a vários clientes "simultaneamente".

3 Questão 3

Tanto TCP quanto UDP são protocolos de transmissão de dados usados na Internet.

TCP é um protocolo orientado a conexão sobre uma rede IP. Ele garante que todos os pacotes enviados vão chegar ao destinatário, na ordem correta. Para isso, são utilizados pacotes de confirmação (ACK, de *acknowledgement*), que são enviados ao remetente para indicar o recebimento bem-sucedido de pacotes; e retransmissão automática de pacotes para os quais o remetente não recebeu confirmação de recebimento. Isso implica atrasos e uma transmissão de modo geral menos eficiente que a utilizada por UDP.

Por outro lado, UDP é um protocolo não orientado a conexão. A comunicação, nesse caso, é feita por meio de datagramas, que são entidades de dados que possuem informações suficientes para serem entregues ao destinatário, sem depender de comunicação prévia entre o remetente e o destinatário. O protocolo apenas garante integridade de dados contidos em um único datagrama; datagramas podem ser entregues fora de ordem ou nem sequer chegar ao seu destinatário. A comunicação por UDP é, de modo geral, mais eficiente que a por TCP por não utilizar pacotes ACK nem realizar outros tipos de controle.

As principais diferenças entre os dois protocolos estão sumarizadas na Tabela 1.

Tabela 1: principais diferenças entre TCP e UDP

	TCP	**UDP**
Acrônimo de	*Transmission Control Protocol*	*User Datagram Protocol*
Conexão	Orientado a conexão	Não orientado a conexão
Ordem de pacotes	Ordena os pacotes na ordem especificada	Não possui ordem inerente pois os datagramas são independentes entre si. Se a ordenação for necessária, ela deve ser implementada em nível de aplicação
Confiabilidade	Garante que os dados transferidos permaneçam intactos e cheguem na mesma ordem em que foram enviados	Não há garantia de que os datagramas cheguem em ordem ou de que eles sejam entregues com sucesso
Checagem de erros	Há checagem de erros	Há checagem de erros, mas sem opções de recuperação

A característica que permite ao servidor UDP atender mais de um cliente, mesmo sendo um servidor iterativo, é o fato de ele não ser orientado a conexão. Assim, um socket do servidor consegue atender vários clientes, sem necessidade de concorrência. Em aplicações como a utilizada nessa tarefa, o trabalho do servidor consiste apenas em ecoar a mensagem enviada pelo cliente; para isso, não é necessário que sejam criados sockets dedicados: o servidor recebe a mensagem, anota o endereço do remetente e envia a resposta para aquele endereço.

4 Questão 4

Após a inicialização tanto do servidor como de um processo cliente, a mensagem "Ola, eu sou o cliente" foi enviada para o servidor.

A saída do servidor pode ser verificada abaixo.

```
niko@ubuntu:~/Desktop/mc833/t6$ ./server_udp.out
Ola, eu sou o cliente
```

A saída do cliente está copiada a seguir.

```
niko@ubuntu:~/Desktop/mc833/t6$ ./client_udp.out localhost
Ola, eu sou o cliente
Ola, eu sou o cliente
```

Em seguida, utilizando a ferramenta `netstat`, pôde-se descobrir o número da porta efêmera utilizada pelo cliente. A saída do comando está impressa a seguir.

```
niko@ubuntu:~/Desktop/mc833$ netstat -lnpu
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address       Foreign Address       State       PID/Program name
udp        0      0 0.0.0.0:31472       0.0.0.0:*                         28529/server_udp.ou
udp        0      0 0.0.0.0:68          0.0.0.0:*                         -
udp        0      0 0.0.0.0:5353        0.0.0.0:*                         -
udp        0      0 0.0.0.0:51562       0.0.0.0:*                         -
udp        0      0 0.0.0.0:2427        0.0.0.0:*                         -
udp        0      0 0.0.0.0:631         0.0.0.0:*                         -
udp        0      0 0.0.0.0:58007       0.0.0.0:*                         28532/client_udp.ou
udp6       0      0 :::39826            :::*                              -
udp6       0      0 :::5353             :::*                              -
udp6       0      0 :::33188            :::*                              -
```

Como pode ser verificado, a porta utilizada pelo servidor é 31472, enquanto a porta utilizada pelo cliente é 58007. Dessa maneira, utilizamos a ferramenta `netcat` para enviar mensagens para o cliente.

```
niko@ubuntu:~/Desktop/mc833/t6$ nc -u localhost 58007
Mensagem usando netcat
```

Considerando que o cliente está programado para escutar uma mensagem apenas após o envio de outra (pois supostamente o único processo que conhece seu número de porta é o servidor que ecoará sua mensagem), é necessário digitar uma mensagem para poder receber a comunicação feita via `netcat`. Pode-se observar isso na saída a seguir, em que se enviou o texto "Mensagem para o servidor".

```
niko@ubuntu:~/Desktop/mc833/t6$ ./client_udp.out localhost
Ola, eu sou o cliente
Ola, eu sou o cliente

Mensagem para o servidor
Mensagem usando netcat
```

Assim, a chamada feita a `recvfrom`, que deveria capturar o eco da mensagem enviada ao servidor, recebeu, em vez disso, a mensagem enviada por `netcat`. Como o protocolo UDP não é orientado a conexão, não há uma conexão única firmada entre o cliente e o servidor e uma chamada a `recvfrom` é capaz de obter mensagens de qualquer fonte, misturando as mensagens advindas do servidor com as mensagens enviadas por `netcat`.

5 Questão 5

A chamada a `recvfrom` do cliente foi alterada de modo a ignorar qualquer datagrama recebido de um endereço diferente do servidor para o qual o datagrama original foi enviado. A versão alterada do loop principal do cliente pode ser verificada a seguir.

```
    while(fgets(buf, sizeof(buf), stdin)) {
        buf[MAX_LINE-1] = '\0';
        if(sendto(s, buf, strlen(buf)+1, 0,
                  (struct sockaddr *)&sin, sizeof(sin)) < 0) {
            perror("client-udp: sendto");
        }

        while(recvfrom(s, buf, sizeof(buf), 0, (struct sockaddr *)&peer, &peerlen) > 0){

            if( (peer.sin_addr.s_addr == sin.sin_addr.s_addr) && (peer.sin_port == htons(SERVER_PORT)
                ) ) {
                fprintf(stdout, "%s\n", buf);
                break;
            }
        }


    }
```

Utilizando a nova versão do cliente, foi simulada uma comunicação entre cliente e servidor. Entre o par de mensagens enviadas a partir do cliente, houve uma tentativa mal sucedida de envio de mensagem, por meio de `netcat`, ao cliente. A saída referente ao cliente pode ser verificada a seguir.

```
niko@ubuntu:~/Desktop/mc833/t6$ ./client5.out localhost
mensagem inicial
mensagem inicial

cliente
cliente
```

A saída referente ao servidor está impressa a seguir.

```
niko@ubuntu:~/Desktop/mc833/t6$ ./server.out localhost
mensagem inicial
```

```
cliente
```

Foi utilizada a ferramenta `netstat` para capturar o número de porta utilizado pelo cliente.

```
nniko@ubuntu:~/Desktop/mc833$ netstat -lnpu
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address       Foreign Address       State      PID/Program name
udp        0      0 0.0.0.0:31472       0.0.0.0:*                        28905/server.out
udp        0      0 0.0.0.0:68          0.0.0.0:*                        -
udp        0      0 0.0.0.0:5353        0.0.0.0:*                        -
udp        0      0 0.0.0.0:51562       0.0.0.0:*                        -
udp        0      0 0.0.0.0:2427        0.0.0.0:*                        -
udp        0      0 0.0.0.0:53794       0.0.0.0:*                        28907/client5.out
udp        0      0 0.0.0.0:631         0.0.0.0:*                        -
udp6       0      0 :::39826            :::*                             -
udp6       0      0 :::5353             :::*                             -
udp6       0      0 :::33188            :::*                             -
```

Então, por meio de `netcat`, foi enviada uma mensagem ao cliente. Como pudemos verificar na saída do cliente, a mensagem foi descartada.

```
niko@ubuntu:~/Desktop/mc833/t6$ nc -u localhost 53794
tentativa usando nc
```

A comunicação foi observada com o auxílio da ferramenta `tcpdump`. Sua saída pode ser observada a seguir.

```
niko@ubuntu:~/Desktop/mc833$ sudo tcpdump -i lo udp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on lo, link-type EN10MB (Ethernet), capture size 262144 bytes
23:41:06.821619 IP localhost.53794 > localhost.31472: UDP, length 18
23:41:06.821710 IP localhost.31472 > localhost.53794: UDP, length 18
23:41:47.102606 IP localhost.40238 > localhost.53794: UDP, length 20
23:41:58.693425 IP localhost.53794 > localhost.31472: UDP, length 9
23:41:58.693475 IP localhost.31472 > localhost.53794: UDP, length 9
```

O primeiro par de datagramas se refere à mensagem "mensagem inicial" enviada pelo cliente e ecoada pelo servidor; o datagrama seguinte é referente à mensagem "tentativa usando nc" enviada por `netcat` e descartada pelo cliente; e, por fim, o último par de datagramas são referentes à mensagem "cliente", enviada pelo cliente e ecoada pelo servidor.

Parte B

Erros Assíncronos e Sockets UDP Conectados

Antes de iniciar as modificações desta parte B, executamos o cliente sem que o servidor estivesse rodando e capturamos o tráfego com o `tcpdump` quando o cliente tenta enviar uma mensagem:

```
bash-4.3$ ./client_udp.out localhost
oi
```

```
bash-4.3$ 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
19:12:01.186146 IP localhost.46276 > localhost.31472: UDP, length 4
19:12:01.186188 IP localhost > localhost: ICMP localhost udp port 31472 unreachable, length 40
^C
2 packets captured
4 packets received by filter
0 packets dropped by kernel
```

Vemos o erro ICMP de *port unreachable*, porém não vemos erro algum na aplicação do cliente. Através

do `strace`, notamos que ele fica indefinidamente aguardando a resposta do servidor, bloqueado na chamada à `recvfrom`.

6 Questão 6

As seguintes modificações foram feitas no código do cliente, após remoção da funcionalidade implementada na questão 5:

- Inclusão da chamada à função connect:

```
if(connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
    perror("client-udp: connect");
    close(s);
    exit(1);
}
```

- Alteração de `sendto` para `send`:

```
if(send(s, buf, strlen(buf)+1, 0) < 0) {
    perror("client-udp: sendto");
}
```

- Alteração de `recvfrom` para `recv`:

```
if(recv(s, buf, sizeof(buf), 0) > 0) {
    fprintf(stdout, "%s\n", buf);
}
```

7 Questão 7

7.1 Questão 7.1

Novamente executamos o cliente, agora usando `connect`, sem que o servidor estivesse rodando. Abaixo, temos os resultados do cliente e do `tcpdump`:

```
bash-4.3$ ./client_udp_connected.out localhost
oi
client-udp: recv: Connection refused
```

```
bash-4.3$ 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
19:18:48.670947 IP localhost.37606 > localhost.31472: UDP, length 4
19:18:48.670999 IP localhost > localhost: ICMP localhost udp port 31472 unreachable, length 40
^C
2 packets captured
4 packets received by filter
0 packets dropped by kernel
```

O resultado do `tcpdump` permanece o mesmo, porém a saída do cliente apresenta um erro agora. Rodando com o `strace`, vemos que o `send` retorna com sucesso, mas o `recv` retorna o erro *ECONNREFUSED (Connection refused)*. Isso acontece devido à conexão que o cliente estabeleceu com o host e a porta em que o servidor deveria estar escutando.

7.2 Questão 7.2

As modificações realizadas na questão 6 resolveram o problema da "resposta falsa", apresentado na questão 4.

Montando um cenário similar ao da questão 4, temos:

```
bash-4.3$ ./server_udp.out
Ola, eu sou o cliente
```

A saída do cliente está copiada a seguir.

```
bash-4.3$ ./client_udp_connected.out localhost
Ola, eu sou o cliente
Ola, eu sou o cliente
```

Desta vez, executando `netstat -lnpu` não gera o resultado para o cliente:

```
bash-4.3$ netstat -lnpu
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address       Foreign Address       State      PID/Program name
udp       0      0 0.0.0.0:31472       0.0.0.0:*                        29341/server_udp.ou
udp       0      0 0.0.0.0:68          0.0.0.0:*                        -
udp       0      0 0.0.0.0:5353        0.0.0.0:*                        -
udp       0      0 0.0.0.0:51562       0.0.0.0:*                        -
udp       0      0 0.0.0.0:2427        0.0.0.0:*                        -
udp       0      0 0.0.0.0:631         0.0.0.0:*                        -
udp6      0      0 :::39826            :::*                             -
udp6      0      0 :::5353             :::*                             -
udp6      0      0 :::33188            :::*                             -
```

Usando então `netstat -pu` temos (filtrando os resultados irrelevantes):

```
bash-4.3$ netstat -pu
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address       Foreign Address       State      PID/Program name
udp       0      0 localhost.localdo:60358 localhost.localdo:31472 ESTABLISHED 29346/client_udp_co
```

Como pode ser verificado, a porta utilizada pelo servidor é 31472, enquanto a porta utilizada pelo cliente é 60358. Dessa maneira, tentamos utilizar a ferramenta `netcat` para enviar mensagens para o cliente novamente:

```
bash-4.3$ nc -u localhost 60358
Mensagem usando netcat
Ncat: Connection refused.
```

O erro obtido foi "connection refused" como pode ser observado pela saída do netcat. Isso ocorre porque esse cliente modificado estabeleceu uma conexão com o servidor usando sua porta 60358 e, por isso, não pode receber mensagens de outros hosts através dessa porta.

```
bash-4.3$ ./client_udp_connected.out localhost
Ola, eu sou o cliente
Ola, eu sou o cliente

Mensagem para o servidor
Mensagem para o servidor
```

8 Questão 8

No caso de *socket* UDP não conectado, o `sendto` é chamado passando toda vez o endereço de destino, como mostra a execução usando `strace`:

```
bash-4.3$ strace ./client_udp.out localhost
read(0, teste
"teste\n", 1024)            = 6
sendto(3, "teste\n\0", 7, 0, {sa_family=AF_INET, sin_port=htons(31472),
    sin_addr=inet_addr("127.0.0.1")}, 16) = 7
```

```
recvfrom(3, "teste\n\0", 256, 0, NULL, NULL) = 7
read(0, teste
"teste\n", 1024)            = 6
sendto(3, "teste\n\0", 7, 0, {sa_family=AF_INET, sin_port=htons(31472),
    sin_addr=inet_addr("127.0.0.1")}, 16) = 7
recvfrom(3, "teste\n\0", 256, 0, NULL, NULL) = 7
read(0,
```

em que os *outputs* iniciais foram omitidos.

Já no caso de *socket* UDP conectado, o `sendto` é chamado sem o endereço de destino, pois o `connect` fixou o destinatário:

```
bash-4.3$ strace ./client_udp_connected.out localhost
read(0, teste
"teste\n", 1024)            = 6
sendto(3, "teste\n\0", 7, 0, NULL, 0) = 7
recvfrom(3, "teste\n\0", 256, 0, NULL, NULL) = 7
read(0, teste
"teste\n", 1024)            = 6
sendto(3, "teste\n\0", 7, 0, NULL, 0) = 7
recvfrom(3, "teste\n\0", 256, 0, NULL, NULL) = 7
read(0,
```

Porém, fazendo-se um teste com um arquivo de entrada 1026000 linhas, não se obteve diferenças significativas entre os tempos de execução das duas versões. Para três execuções de cada, temos:

- client_udp.out

```
bash-4.3$ time ./client_udp.out 127.0.0.1 < input

real  0m38.542s
user  0m0.770s
sys   0m5.783s

bash-4.3$ time ./client_udp.out 127.0.0.1 < input

real  0m38.457s
user  0m0.758s
sys   0m5.761s

bash-4.3$ time ./client_udp.out 127.0.0.1 < input

real  0m38.385s
user  0m0.797s
sys   0m5.653s
```

- client_udp_connected.out

```
bash-4.3$ time ./client_udp_connected.out 127.0.0.1 < input

real  0m38.458s
user  0m0.836s
sys   0m5.349s

bash-4.3$ time ./client_udp_connected.out 127.0.0.1 < input

real  0m38.269s
user  0m0.694s
sys   0m5.547s

bash-4.3$ time ./client_udp_connected.out 127.0.0.1 < input

real  0m38.483s
user  0m0.784s
sys   0m5.510s
```