

**Universidade Federal de Minas Gerais**  
**UFMG**

**Departamento de Engenharia Eletrônica**



**ELT091**  
**Redes TCP/IP**

---

**TCPIP - TRABALHO PRÁTICO 1**  
**2023/1**

---

**Autores**

Breno Gomes de Oliveira Santos	2018020042
Gabriel de Oliveira Andrade	2019027520
Lucas de Almeida Martins	2018020328
Rodolfo de Albuquerque Lessa Villa Verde	2018020719

**Professor**

Luciano de Errico

**Belo Horizonte, 10 de maio de 2023**

## 1 Introdução

Este relatório tem como objetivo explorar os princípios básicos de comunicação em redes de computadores, com foco nos protocolos TCP e UDP. Foram realizados exercícios de códigos em C para demonstrar a implementação prática de soquetes (interface de comunicação) no uso do protocolo UDP, um protocolo de camada de transporte sem conexão, e do protocolo TCP, um protocolo de camada de transporte orientado a conexão.

A fim de rodar o servidor, basta digitar `./server_exX` e, para rodar o cliente, `./client_exX + numero_ip_local` (Onde X é o numero do exercício).

[Link do Repositório do Github](#)

## 2 Experimentos

### Experimento 1

#### Código Fonte

```
1 #include <netdb.h>
2 #include <netinet/in.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <sys/socket.h>
6 #include <sys/types.h>
7
8 #define SERVER_PORT 54321
9 #define MAX_PENDING 1
10 #define MAX_LINE 256
11
12 int main() {
13     struct sockaddr_in sin;
14     char buf[MAX_LINE];
15     int buf_len, addr_len;
16     int s, new_s;
17
18     /* build address data structure */
19     bzero((char *)&sin, sizeof(sin));
20     sin.sin_family = AF_INET;
21     sin.sin_addr.s_addr = INADDR_ANY;
22     sin.sin_port = htons(SERVER_PORT);
23
24     /* setup passive open */
25     if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
26         perror("simplex-talk: socket");
27         exit(1);
28     }
29     if ((bind(s, (struct sockaddr *)&sin, sizeof(sin))) < 0) {
30         perror("simplex-talk: bind");
31         exit(1);
32     }
33     listen(s, MAX_PENDING);
34     /* wait for connection, then receive and print text */
35     addr_len = sizeof(sin);
36     while (1) {
37         if ((new_s = accept(s, (struct sockaddr *)&sin, &addr_len)) < 0) {
38             perror("simplex-talk: accept");
39             exit(1);
40         }
41         while (buf_len = recv(new_s, buf, sizeof(buf), 0)){
42             fputs(buf, stdout);
43         }
44         close(new_s);
45     }
46 }
```

**Ex33:** Obtain and build the simplex-talk sample socket program shown in the text. Start one server and one client, in separate windows. While the first client is running, start 10 other clients that connect to the same server; these other clients should most likely be started in the background with their input redirected from a file. What happens to these 10 clients? Do their connect()s fail, time out, or succeed? Do any other calls block? Now let the first client exit. What happens? Try this with the server value MAX\_PENDING set to 1 as well.

Com o objetivo de obter o endereço IP da máquina, pesquisa-se o procedimento para o S.O. do usuário. Percebe-se que no início, após iniciar o servidor, seta-se uma variável no linux com o IP desejado. Depois disso, rodam-se o seguinte comando:

```
gabriel@gabriel-pc:~/Desktop$ ifconfig | grep inet -w
inet 127.0.0.1 netmask 255.0.0.0
inet 192.168.18.18 netmask 255.255.255.0 broadcast 192.168.18.255
```

Figura 1: Obtenção do IP

```
ip='192.168.18.18'
for i in {1..10}; do ./client_base $ip & done
```

**OBS:** Atenção, não pode haver espaços na atribuição de variável `ip='...'`

```
gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip/Ex33$ ip='192.168.18.18'
gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip/Ex33$ for i in {1..10}; do ./client_base $ip & done
[1] 21555
[2] 21556
[3] 21557
[4] 21558
[5] 21559
[6] 21560
[7] 21561
[8] 21562
[9] 21563
[10] 21564

[1] Stopped ./client_base $ip
[2] Stopped ./client_base $ip
[3] Stopped ./client_base $ip
[4] Stopped ./client_base $ip
[5]- Stopped ./client_base $ip
[6]+ Stopped ./client_base $ip
gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip/Ex33$ jobs
[1] Stopped ./client_base $ip
[2] Stopped ./client_base $ip
[3] Stopped ./client_base $ip
[4] Stopped ./client_base $ip
[5]- Stopped ./client_base $ip
[6]+ Stopped ./client_base $ip
[7] Running ./client_base $ip &
[8] Running ./client_base $ip &
[9] Running ./client_base $ip &
[10] Running ./client_base $ip &
gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip/Ex33$ simplex-talk: connect: Connection timed out
simplex-talk: connect: Connection timed out
simplex-talk: connect: Connection timed out
```

Figura 2: Teste função *Jobs*

Feito isso, ao usar a função "jobs", nota-se pela Figura 2, que os 10 comandos foram iniciados. Sendo que os 6 primeiros estão parados e os 4 últimos, rodando.

Após um tempo, nota-se que os 3 últimos serviços sofreram de: *Connection timed out*, pois eles não se conectaram ao servidor no tempo estipulado.

```

gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip/Ex33$ simplex-talk: connect: Connection timed out
simplex-talk: connect: Connection timed out
simplex-talk: connect: Connection timed out
jobs
[1] Stopped                  ./client_base $ip
[2] Stopped                  ./client_base $ip
[3] Stopped                  ./client_base $ip
[4] Stopped                  ./client_base $ip
[5] Stopped                  ./client_base $ip
[6]- Stopped                 ./client_base $ip
[7]+ Stopped                 ./client_base $ip
[8] Exit 1                   ./client_base $ip
[9] Exit 1                   ./client_base $ip
[10] Exit 1                  ./client_base $ip
gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip/Ex33$ jobs
[1] Stopped                  ./client_base $ip
[2] Stopped                  ./client_base $ip
[3] Stopped                  ./client_base $ip
[4] Stopped                  ./client_base $ip
[5] Stopped                  ./client_base $ip
[6]- Stopped                 ./client_base $ip
[7]+ Stopped                 ./client_base $ip
gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip/Ex33$

```

Figura 3: Teste função *Jobs* timed out

Já na Figura 3, nota-se que consegue ver a saída dos comandos que resultaram em time out, ao mesmo tempo da confirmação de que os outros 7 processos estão rodando.

Os 7 servidores que se conectaram com sucesso deve-se a fila, denominada como MAX\_PENDING, ser setada como 5. Isso ocorre devido a contagem começar a partir do valor 0. Dessa forma, acredita-se que ao ter 7 clients não desativados, é devido ao fato de ter 1 ativo e 6 em espera.

Ao iniciar os testes, que não há entradas no servidor, entra-se individualmente nos serviços rodando, e tenta-se enviar mensagens a partir deles para o servidor.

Isto é feito através do comando "fg num\_job", que traz para o "foreground" o comando que está sendo rodado em "background".

```

gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip$ ./server_base
host: 0
[2] 21556
[3] 21557
[4] 21558
[5] 21559
[6] 21560
[7] 21561
[8] 21562
[9] 21563
[10] 21564
[1] Stopped                  ./client_base $ip
[2] Stopped                  ./client_base $ip
[3] Stopped                  ./client_base $ip
[4] Stopped                  ./client_base $ip
[5]- Stopped                 ./client_base $ip
[6]+ Stopped                 ./client_base $ip
gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip/Ex33$ jobs
[1] Stopped                  ./client_base $ip
[2] Stopped                  ./client_base $ip
[3] Stopped                  ./client_base $ip
[4] Stopped                  ./client_base $ip
[5]- Stopped                 ./client_base $ip
[6]+ Stopped                 ./client_base $ip
[7] Running                  ./client_base $ip &
[8] Running                  ./client_base $ip &
[9] Running                  ./client_base $ip &
[10] Running                 ./client_base $ip &
gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip/Ex33$ simplex-talk: connect: Connection timed out
simplex-talk: connect: Connection timed out
simplex-talk: connect: Connection timed out
jobs
[1] Stopped                  ./client_base $ip
[2] Stopped                  ./client_base $ip
[3] Stopped                  ./client_base $ip
[4] Stopped                  ./client_base $ip
[5]- Stopped                 ./client_base $ip
[6]+ Stopped                 ./client_base $ip
[7]+ Stopped                 ./client_base $ip
[8] Exit 1                   ./client_base $ip
[9] Exit 1                   ./client_base $ip
[10] Exit 1                  ./client_base $ip
gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip/Ex33$ jobs
[1] Stopped                  ./client_base $ip
[2] Stopped                  ./client_base $ip
[3] Stopped                  ./client_base $ip
[4] Stopped                  ./client_base $ip
[5] Stopped                  ./client_base $ip
[6]- Stopped                 ./client_base $ip
[7]+ Stopped                 ./client_base $ip
gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip/Ex33$

```

Figura 4: Print antes de iniciar os testes de conexão

Após digitar o comando, tem-se:

```
gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip/Ex33$ fg 1
./client_base $ip
```

Figura 5: Resultado do comando "fg num job"

Ao enviar uma mensagem para o servidor, nota-se o seguinte:

```
gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip$ ./server_base
0 host: 0
estou no job 1

gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip/Ex33$ ./client_base $ip
simplex-talk: connect: Connection timed out
simplex-talk: connect: Connection timed out
jobs
[1] Stopped ./client_base $ip
[2] Stopped ./client_base $ip
[3] Stopped ./client_base $ip
[4] Stopped ./client_base $ip
[5] Stopped ./client_base $ip
[6] Stopped ./client_base $ip
[7] Stopped ./client_base $ip
[8] Stopped ./client_base $ip
[9] Stopped ./client_base $ip
[10] Stopped ./client_base $ip
gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip/Ex33$ jobs
[1] Stopped ./client_base $ip
[2] Stopped ./client_base $ip
[3] Stopped ./client_base $ip
[4] Stopped ./client_base $ip
[5] Stopped ./client_base $ip
[6] Stopped ./client_base $ip
[7] Stopped ./client_base $ip
[8] Stopped ./client_base $ip
[9] Stopped ./client_base $ip
[10] Stopped ./client_base $ip
gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip/Ex33$ fg 1
./client_base $ip
estou no job 1
```

Figura 6: Conexão cliente e servidor bem sucedida *Job 1*

Nota-se pela Figura 6 que a mensagem digitada no *job 1* é enviada com sucesso para o servidor, que a reproduz em seguida.

Com objetivo de ir para o segundo job, sai do *job 1* sem derrubá-lo, através do atalho *Crtl+Z*.

```
gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip$ ./server_base
0 host: 0
estou no job 1

gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip/Ex33$ ./client_base $ip
[3] Stopped ./client_base $ip
[4] Stopped ./client_base $ip
[5] Stopped ./client_base $ip
[6] Stopped ./client_base $ip
[7] Stopped ./client_base $ip
[8] Stopped ./client_base $ip
[9] Stopped ./client_base $ip
[10] Stopped ./client_base $ip
gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip/Ex33$ jobs
[1] Stopped ./client_base $ip
[2] Stopped ./client_base $ip
[3] Stopped ./client_base $ip
[4] Stopped ./client_base $ip
[5] Stopped ./client_base $ip
[6] Stopped ./client_base $ip
[7] Stopped ./client_base $ip
[8] Stopped ./client_base $ip
[9] Stopped ./client_base $ip
[10] Stopped ./client_base $ip
gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip/Ex33$ fg 1
./client_base $ip
estou no job 1
^Z
[1] Stopped ./client_base $ip
gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip/Ex33$ fg 2
./client_base $ip
estou no job 2
```

Figura 7: Conexão cliente e servidor falha *Job 2*

Percebe-se pela Figura 7 que ao tentar enviar uma mensagem ao servidor, o cliente falha. Isso ocorre, pois o servidor não está em conexão ativa com esse cliente, o que faz com o servidor não receba ou reproduza a mensagem. O mesmo acontece para os demais jobs do 1 ao 7.

```
gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip$ ./server_base
0 host: 0
estou no job 1
estou no job 2

gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip/Ex33$ ./client_base $ip
[1] Stopped ./client_base $ip
[2] Stopped ./client_base $ip
[3] Stopped ./client_base $ip
[4] Stopped ./client_base $ip
[5] Stopped ./client_base $ip
[6] Stopped ./client_base $ip
[7] Stopped ./client_base $ip
gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip/Ex33$ kill %1
[1] Terminated ./client_base $ip
[2] Stopped ./client_base $ip
[3] Stopped ./client_base $ip
[4] Stopped ./client_base $ip
[5] Stopped ./client_base $ip
[6] Stopped ./client_base $ip
[7] Stopped ./client_base $ip
gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip/Ex33$ jobs
[2] Stopped ./client_base $ip
[3] Stopped ./client_base $ip
[4] Stopped ./client_base $ip
[5] Stopped ./client_base $ip
[6] Stopped ./client_base $ip
[7] Stopped ./client_base $ip
```

Figura 8: Conexão cliente e servidor *Job 1* e *Job 2*

Após o processo 1 anteriormente mencionado terminado, nota-se o seguinte: recebe-se, do *Job 2*, a mensagem previamente digitada, e pode-se usufruir o *Job 2* como cliente ativo, como evidenciado na Figura 8 e na Figura 9.

```

gabriel@gabriel-pc: ~/Desktop/Repos/tp1_tcp_ip/Ex33 $ ./server_base
0 host: 0
estou no job 1
estou no job 2
estou no job 2 pela segunda vez

gabriel@gabriel-pc: ~/Desktop/Repos/tp1_tcp_ip/Ex33 $ jobs
[1]-  Terminated                  ./client_base $ip
[2]+  Stopped                      ./client_base $ip
[3]   Stopped                      ./client_base $ip
[4]   Stopped                      ./client_base $ip
[5]   Stopped                      ./client_base $ip
[6]   Stopped                      ./client_base $ip
[7]   Stopped                      ./client_base $ip
gabriel@gabriel-pc: ~/Desktop/Repos/tp1_tcp_ip/Ex33 $ kill K1
gabriel@gabriel-pc: ~/Desktop/Repos/tp1_tcp_ip/Ex33 $ jobs
[1]-  Terminated                  ./client_base $ip
[2]+  Stopped                      ./client_base $ip
[3]   Stopped                      ./client_base $ip
[4]   Stopped                      ./client_base $ip
[5]   Stopped                      ./client_base $ip
[6]   Stopped                      ./client_base $ip
[7]   Stopped                      ./client_base $ip
gabriel@gabriel-pc: ~/Desktop/Repos/tp1_tcp_ip/Ex33 $ jobs
[1]-  Terminated                  ./client_base $ip
[2]+  Stopped                      ./client_base $ip
[3]   Stopped                      ./client_base $ip
[4]   Stopped                      ./client_base $ip
[5]   Stopped                      ./client_base $ip
[6]   Stopped                      ./client_base $ip
[7]   Stopped                      ./client_base $ip
gabriel@gabriel-pc: ~/Desktop/Repos/tp1_tcp_ip/Ex33 $ fg 2
./client_base $ip
estou no job 2 pela segunda vez

```

Figura 9: Conexão cliente e servidor *Job 1* e *Job 2*

Após esse processo, cria-se uma nova versão do código com a variável `MAX_PENDING=1` e repete o processo como evidenciado na Figura 10.

```

Ex33 > C server_base_ex33.c > MAX_PENDING
/
8  #define SERVER_PORT 54321
9  #define MAX_PENDING 1
10 #define MAX_LINE 256

```

Figura 10: Alteração da variável `MAX_PENDING`

Conforme era esperado, evidenciado pela Figura 11, a conexão com os 10 clientes resultou em um maior número de timed out's.

```

gabriel@gabriel-pc: ~/Desktop/Repos/tp1_tcp_ip/Ex33 $ ./server_ex33
[1] 26302
[2] 26303
[3] 26304
[4] 26305
[5] 26306
[6] 26307
[7] 26308
[8] 26309
[9] 26310
[10] 26311
[1] Stopped ./client_base $ip
[2]- Stopped ./client_base $ip
[3]+ Stopped ./client_base $ip
gabriel@gabriel-pc: ~/Desktop/Repos/tp1_tcp_ip/Ex33 $ jobs
[1]- Stopped ./client_base $ip
[2]- Stopped ./client_base $ip
[3]+ Stopped ./client_base $ip
[4] Running ./client_base $ip &
[5] Running ./client_base $ip &
[6] Running ./client_base $ip &
[7] Running ./client_base $ip &
[8] Running ./client_base $ip &
[9] Running ./client_base $ip &
[10] Running ./client_base $ip &
gabriel@gabriel-pc: ~/Desktop/Repos/tp1_tcp_ip/Ex33 $ simplex-talk: connect: Connection timed out
simplex-talk: connect: Connection timed out
simplex-talk: connect: Connection timed out
simplex-talk: connect: Connection timed out
simplex-talk: connect: Connection timed out
simplex-talk: connect: Connection timed out
simplex-talk: connect: Connection timed out
simplex-talk: connect: Connection timed out
simplex-talk: connect: Connection timed out
simplex-talk: connect: Connection timed out
[1] Stopped ./client_base $ip
[2]- Stopped ./client_base $ip
[3]+ Stopped ./client_base $ip
[4] Exit 1 ./client_base $ip
[5] Exit 1 ./client_base $ip
[6] Exit 1 ./client_base $ip
[7] Exit 1 ./client_base $ip
[8] Exit 1 ./client_base $ip
[9] Exit 1 ./client_base $ip
[10] Exit 1 ./client_base $ip
gabriel@gabriel-pc: ~/Desktop/Repos/tp1_tcp_ip/Ex33 $

```

Figura 11: Conexão entre 10 clientes e 1 servidor

A lógica dos hosts ativos e em espera ainda é a mesma.

A título de curiosidade, com o *max pending* igual a 0, obtém o resultado da Figura 12.

```

[1]- Stopped                               ./client_base $ip
gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip/Ex33$ simplex-talk: connect: Connection timed out
simplex-talk: connect: Connection timed out
simplex-talk: connect: Connection timed out
simplex-talk: connect: Connection timed out
simplex-talk: connect: Connection timed out
simplex-talk: connect: Connection timed out
simplex-talk: connect: Connection timed out
simplex-talk: connect: Connection timed out
jobs
[1]- Stopped                               ./client_base $ip
[2]+ Stopped                               ./client_base $ip
[3] Exit 1                                 ./client_base $ip
[4] Exit 1                                 ./client_base $ip
[5] Exit 1                                 ./client_base $ip
[6] Exit 1                                 ./client_base $ip
[7] Exit 1                                 ./client_base $ip
[8] Exit 1                                 ./client_base $ip
[9] Exit 1                                 ./client_base $ip
[10] Exit 1                                ./client_base $ip
gabriel@gabriel-pc:~/Desktop/Repos/tp1_tcp_ip/Ex33$

```

Figura 12: Conexão com *max pending* igual a 0

Conclui-se que o *MAX\_PENDING* começa a contar do zero. Nos exemplos anteriores, quando *MAX\_PENDING* é igual a 5, há 1 conexão ativa, e 6 conexões em espera (do 0 ao 5). Ao *MAX\_PENDING* mudar de valor, a mesma logica ocorreu, fazendo haver 3 ligações para *MAX\_PENDING* = 1 (1 ativa, 2 e espera) e duas para *MAX\_PENDING* = 0 (1 ativa, 1 em espera).

## Experimento 2

**Ex34:** Modify the simplex-talk socket program so that each time the client sends a line to the server, the server sends the line back to the client. The client (and server) will now have to make alternating calls to *recv()* and *send()*.

Os códigos desenvolvidos para esse experimento pode ser observado abaixo. Primeiro o servidor:

```

1 #include <netdb.h>
2 #include <netinet/in.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <sys/socket.h>
6 #include <sys/types.h>
7
8 #define SERVER_PORT 54321
9 #define MAX_PENDING 5
10 #define MAX_LINE 256
11
12 /* Rode esse em um terminal e o client em outro.
13
14 para rodar esse, digitar:
15
16     ./server_ex34
17
18
19 p/ compilar e rodar:
20
21     gcc server_ex34.c -o server_ex34 && ./server_ex34
22
23 */
24
25 int main() {
26     struct sockaddr_in sin;
27     char buf[MAX_LINE];
28     int buf_len, addr_len, client_len;
29     int s, new_s;
30
31     /* build address data structure */

```

```

32     bzero((char *)&sin, sizeof(sin));
33     sin.sin_family = AF_INET;
34     sin.sin_addr.s_addr = INADDR_ANY;
35     sin.sin_port = htons(SERVER_PORT);
36
37     /* setup passive open */
38     if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
39         perror("simplex-talk: socket");
40         exit(1);
41     }
42     if ((bind(s, (struct sockaddr *)&sin, sizeof(sin))) < 0) {
43         perror("simplex-talk: bind");
44         exit(1);
45     }
46     listen(s, MAX_PENDING);
47     /* wait for connection, then receive and print text */
48     addr_len = sizeof(sin);
49     while (1) {
50         if ((new_s = accept(s, (struct sockaddr *)&sin, &addr_len)) < 0) {
51             perror("simplex-talk: accept");
52             exit(1);
53         }
54         while (buf_len = recv(new_s, buf, sizeof(buf), 0)){
55             fputs(buf, stdout);
56             // server sending back to the client
57             buf[MAX_LINE - 1] = '\0';
58             client_len = strlen(buf) + 1;
59             send(new_s, buf, client_len, 0);
60         }
61         close(new_s);
62     }
63 }

```

Por fim, o código do cliente:

```

1  #include <stdio.h>
2  // #include <stdlib.h>
3  #include <string.h>
4  #include <sys/socket.h>
5  #include <sys/types.h>
6  #include <arpa/inet.h> // to use inet_addr
7  #include <netinet/in.h>
8  #include <netdb.h>
9
10 #define SERVER_PORT 54321
11 #define MAX_LINE 256
12
13 /*
14 Coloque o seu ip ao rodar esse comando, da seguinte forma:
15 gcc client_ex34.c -o client_ex34 && ./client_ex34 host_name_of_server
16
17 Exemplo de host: 192.168.18.18
18
19 Exemplo:
20 P/ compilar e rodar:
21     gcc client_ex34.c -o client_ex34 &&
22
23 Somente para rodar:
24     ./client_ex34 192.168.18.18
25
26 */
27 int main(int argc, char *argv[]) {
28     FILE *fp;
29     struct hostent *hp;
30     struct sockaddr_in sin;
31     char *host;
32     char buf[MAX_LINE], buf_receive[MAX_LINE];
33     int s;
34     int len, buf_len;
35

```



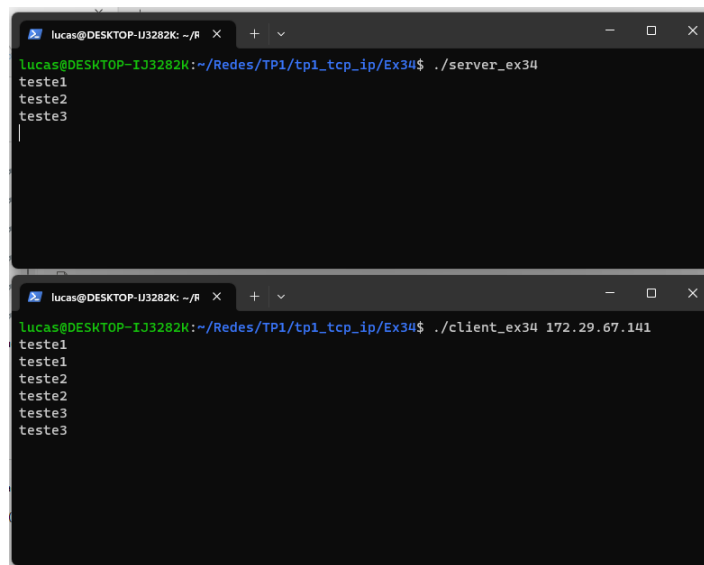
```

36     len = sizeof(sin);
37
38     in_addr_t binary_address;
39     if (argc == 2) {
40         host = argv[1];
41     } else {
42         fprintf(stderr, "usage: simplex-talk host\n");
43         exit(1);
44     }
45
46     /* translate host ASCII notation to Binary */
47     //printf("%s", host);
48     binary_address = inet_addr(host);
49     //printf("%u", binary_address);
50     if (!binary_address) {
51         fprintf(stderr, "simplex-talk: unknown host: %s\n", host);
52         exit(1);
53     }
54
55     /* build address data structure */
56
57     bzero((char *)&sin, sizeof(sin)); // which means that sin has received zeros
58     sin.sin_family = AF_INET;
59
60     bcopy(&binary_address, (char *)&sin.sin_addr, sizeof(binary_address));
61     sin.sin_port = htons(SERVER_PORT);
62     /* active open */
63     if (
64         (s = socket(PF_INET, SOCK_STREAM, 0)) < 0
65     ) {
66         perror("simplex-talk: socket");
67         exit(1);
68     }
69     if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
70         perror("simplex-talk: connect");
71         close(s);
72         exit(1);
73     }
74     /* main loop: get and send lines of text */
75     while (fgets(buf, sizeof(buf), stdin)) {
76         buf[MAX_LINE - 1] = '\0';
77         len = strlen(buf) + 1;
78         send(s, buf, len, 0);
79
80         //receives from the server
81         buf_len = recv(s, buf_receive, sizeof(buf), 0);
82         fputs(buf_receive, stdout);
83     }
84 }

```

As alterações necessárias no código encontram-se na seção do 'loop', tanto no cliente quanto no servidor. No código do cliente, dentro do 'loop', depois do envio da mensagem pelo 'send' é adicionado o 'recv' que recebe a resposta do server. No código do server, dentro do 'loop', é adicionado uma seção que envia a mensagem recebida devolta para o cliente.

O resultado desse procedimento pode ser observado na figura a seguir, na qual a mensagem enviada pelo cliente é recebida pelo servidor, que envia devolta a mensagem para o cliente que a recebe com sucesso.



The image shows two terminal windows. The top window is titled 'lucas@DESKTOP-IJ3282K: ~/R' and shows the command `./server_ex34` being executed. The output is `teste1`, `teste2`, and `teste3` on separate lines. The bottom window is titled 'lucas@DESKTOP-IJ3282K: ~/R' and shows the command `./client_ex34 172.29.67.141` being executed. The output is `teste1`, `teste1`, `teste2`, `teste2`, `teste3`, and `teste3` on separate lines.

```
lucas@DESKTOP-IJ3282K: ~/R x + v - □ x
lucas@DESKTOP-IJ3282K:~/Redes/TP1/tp1_tcp_ip/Ex34$ ./server_ex34
teste1
teste2
teste3

lucas@DESKTOP-IJ3282K: ~/R x + v - □ x
lucas@DESKTOP-IJ3282K:~/Redes/TP1/tp1_tcp_ip/Ex34$ ./client_ex34 172.29.67.141
teste1
teste1
teste2
teste2
teste3
teste3
```

Figura 13: Comunicação entre cliente e servidor ex34

Portanto, pode-se concluir que esse exercício pode ter implementações práticas, pois essa implementação evidencia o sucesso da comunicação, já que a mensagem enviada pelo cliente deve ser a mesma que ele recebe, confirmando o envio correto dos dados.

## Experimento 3

**Ex35:** Modify the simplex-talk socket program so that it uses UDP as the transport protocol rather than TCP. You will have to change `SOCK_STREAM` to `SOCK_DGRAM` in both client and server. Then, in the server, remove the calls to `listen()` and `accept()`, and replace the two nested loops at the end with a single loop that calls `recv()` with socket `s`. Finally, see what happens when two such UDP clients simultaneously connect to the same UDP server, and compare this to the TCP behavior.

A alteração do código do **servidor** pode ser vista a seguir:

```

1  #include <netdb.h>
2  #include <netinet/in.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <sys/socket.h>
6  #include <sys/types.h>
7
8  #define SERVER_PORT 54321
9  #define MAX_PENDING 5
10 #define MAX_LINE 256
11
12 /* Rode esse em um terminal e o client em outro.
13
14 para rodar esse,      s   digitar:
15
16     ./server_ex35
17
18 p/ compilar e rodar:
19
20     gcc server_ex35.c -o server_ex35 && ./server_ex35
21
22 */
23
24
25 int main() {
26     struct sockaddr_in sin;
27     char buf[MAX_LINE];
28     int buf_len, addr_len;
29     int s, new_s;
30
31     /* build address data structure */
32     bzero((char *)&sin, sizeof(sin));
33     sin.sin_family = AF_INET;
34     sin.sin_addr.s_addr = INADDR_ANY;
35     sin.sin_port = htons(SERVER_PORT);
36
37     /* setup passive open */
38     if ((s = socket(PF_INET, SOCK_DGRAM, 0)) < 0) {
39         perror("simplex-talk: socket");
40         exit(1);
41     }
42     if ((bind(s, (struct sockaddr *)&sin, sizeof(sin))) < 0) {
43         perror("simplex-talk: bind");
44         exit(1);
45     }
46     /* receive and print text */
47     addr_len = sizeof(sin);
48
49     while (buf_len = recv(s, buf, sizeof(buf), 0)) {
50         fputs(buf, stdout);
51     }
52     close(s);
53 }

```

O código do **cliente** UDP a seguir:

```

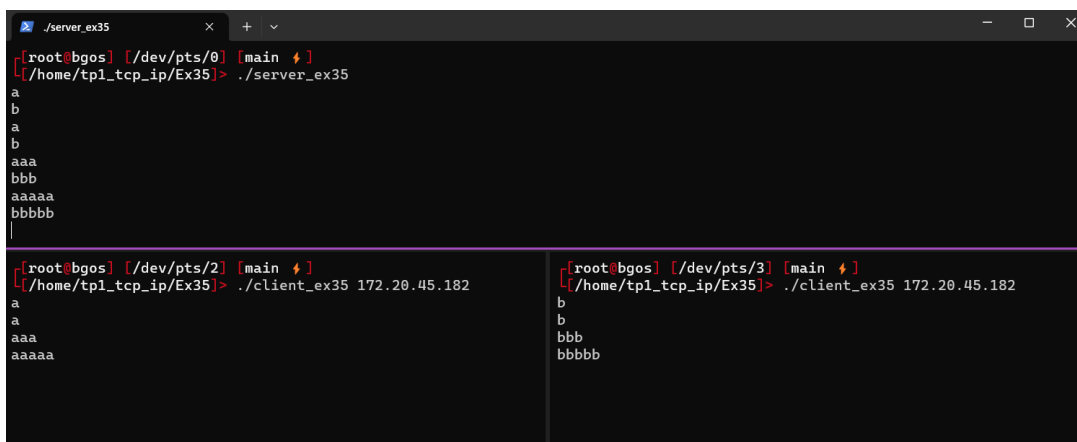
1 #include <stdio.h>
2 #include <arpa/inet.h> // to use inet_addr
3 #include <netdb.h>
4 #include <netinet/in.h>
5 #include <string.h>
6 #include <sys/socket.h>
7 #include <sys/types.h>
8
9 #define SERVER_PORT 54321
10 #define MAX_LINE 256
11
12 /*
13 Coloque o seu ip ao rodar esse comando, da seguinte forma:
14 gcc client_ex35.c -o client_ex35 && ./client_ex35 host_name_of_server
15
16 Exemplo de host: 192.168.18.18
17
18 Exemplo:
19 P/ compilar e rodar:
20     gcc client_ex35.c -o client_ex35
21
22 Somente para rodar:
23     ./client_ex35 192.168.18.18
24
25 */
26 int main(int argc, char *argv[]) {
27     FILE *fp;
28     struct hostent *hp;
29     struct sockaddr_in sin;
30     char *host;
31     char buf[MAX_LINE];
32     int s;
33     int len;
34
35     len = sizeof(sin);
36
37     in_addr_t binary_address;
38     if (argc == 2) {
39         host = argv[1];
40     } else {
41         fprintf(stderr, "usage: simplex-talk host\n");
42         exit(1);
43     }
44
45     /* translate host ASCII notation to Binary */
46     binary_address = inet_addr(host);
47     if (!binary_address) {
48         fprintf(stderr, "simplex-talk: unknown host: %s\n", host);
49         exit(1);
50     }
51
52     /* build address data structure */
53
54     bzero((char *)&sin,
55          sizeof(sin)); // which means that sin has received zeros here
56
57     sin.sin_family = AF_INET;
58
59     bcopy(&binary_address, (char *)&sin.sin_addr, sizeof(binary_address));
60     sin.sin_port = htons(SERVER_PORT);
61     /* active open */
62     if ((s = socket(PF_INET, SOCK_DGRAM, 0)) < 0) {
63         perror("simplex-talk: socket");
64         exit(1);
65     }
66     if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
67         perror("simplex-talk: connect");
68         close(s);

```

```
69     exit(1);
70 }
71 /* main loop: get and send lines of text */
72 while (fgets(buf, sizeof(buf), stdin)) {
73     buf[MAX_LINE - 1] = '\0';
74     len = strlen(buf) + 1;
75     send(s, buf, len, 0);
76 }
77 }
```

O UDP não requer uma conexão fixa como o TCP, permitindo que o servidor UDP processe simultaneamente as mensagens recebidas de vários clientes sem se fixar a uma conexão atual, o que pode resultar em filas de espera. Consequentemente, as mensagens são gerenciadas assim que são enviadas pelos clientes, como ilustrado na imagem abaixo.

A fim de testar o servidor UDP, 2 clientes UDP foram criados. A seguir, o resultado do teste:



The image shows three terminal windows. The top window is titled `./server_ex35` and shows the server running on `/dev/pts/0`. It receives a series of input lines: `a`, `b`, `a`, `b`, `aaa`, `bbb`, `aaaaa`, and `bbbbbb`. The bottom-left window is titled `./client_ex35 172.20.45.182` and shows the client running on `/dev/pts/2` sending the same sequence of lines. The bottom-right window is titled `./client_ex35 172.20.45.182` and shows the client running on `/dev/pts/3` sending the same sequence of lines. This demonstrates that the server can handle multiple clients simultaneously.

Figura 14: Saída ex35

Ao enviar mensagens alternadas de cada cliente, foi possível verificar se o servidor é capaz de processar e responder às mensagens de cada cliente de forma independente e correta. Isso pode ser particularmente importante em sistemas com alto tráfego de rede, onde múltiplos clientes podem estar enviando mensagens ao mesmo tempo.

## Bibliografia

- [1] Computer Networks: A System Approach; Larry Peterson, Bruce Davie

Acesso em 03 de Maio de 2023.