

## Exercício 2.2

**Aluno:** Caio Augusto Alves Nolasco

**RA:** 195181

Instituto de Computação

Universidade Estadual de Campinas

Campinas, 10 de Novembro de 2020.

# Sumário

1	Instrução de compilação e execução . . . . .	2
2	Formato de implementação . . . . .	2
3	Questão 1 . . . . .	3
4	Questão 2 . . . . .	6
5	Questão 3 . . . . .	11
6	Questão 4 . . . . .	15
6.1	4-a - Comando "quit	15
6.2	4-b - Inverter string recebida pelo cliente . . . . .	16
7	Questão 5 . . . . .	17
8	Questão 6 . . . . .	17

## 1 Instrução de compilação e execução

Junto aos arquivos do servidor e cliente, são enviados arquivos para as função envelopadoras e um makefile. Os arquivos do servidor e do cliente são compilados separadamente: para compilar o servidor.c, chama-se "make servidor". Para compilar o cliente.c, chama-se "make cliente". Os executáveis têm nomes, respectivamente, "servidor" e "cliente".

Eles podem ser então executados normalmente pelo terminal, chamando-se `./servidor NUMPORT` para iniciar o servidor e `./cliente IPADDR NUMPORT` para conectar um cliente ao servidor.

É preciso que os arquivos "wrappers.c" e "wrappers.h" estejam no mesmo diretório dos arquivos do servidor e cliente.

## 2 Formato de implementação

Não consegui entender muito bem do enunciado o que precisava ser a saída do programa em cada questão, ou se as saídas precisavam ser cumulativas ou cada questão precisaria de uma saída totalmente diferente das demais questões. Então, optei por comentar trechos do código pertinentes para cada questão. O código como enviado é feito para as instruções da questão 2, em que o cliente executa os comandos enviados pelo servidor e envia de volta os resultados, junto com o endereço de IP e o número da porta do cliente.

Para realizar as instruções da Questão 3, é preciso comentar a versão da função da questão 2 `streach()` em `servidor.c` e `strcli()` em `cliente.c`, identificadas por um comentário em maiúsculo. Além disso, é preciso descomentar a segunda versão dessas funções em ambos os arquivos, também identificadas, adaptadas para a Questão 3. Por fim, é preciso descomentar

os comandos identificados por comentários da Questão 3 na função `main()` de `servidor.c`.

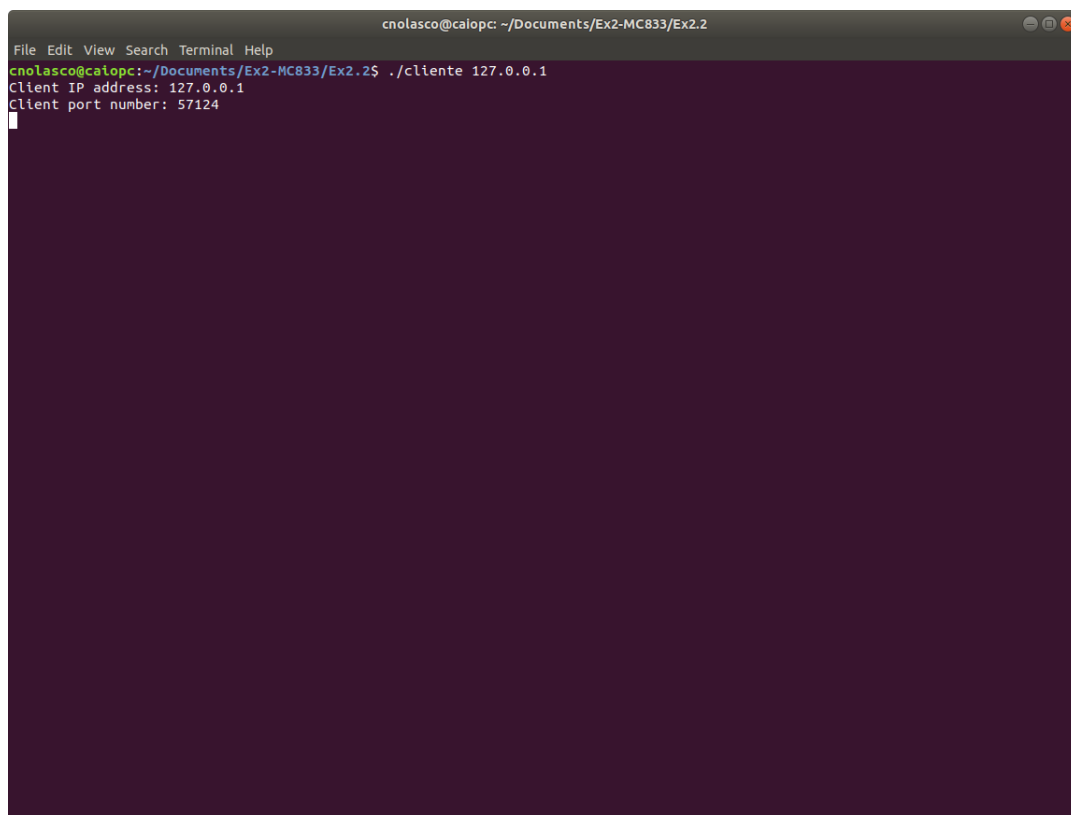
Para a questão 4, usa-se a mesma implementação das funções da questão 2, comentando-se novamente o que é pertinente a questão 3. Os comandos para a questão 4 são identificados por comentários dependendo do item que se deseja avaliar.

### **3 Questão 1**

Não. Mesmo chamando o comando `sleep()` antes da primeira conexão ser fechada, um segundo cliente que tente se conectar ao servidor não consegue se conectar.

```
File Edit View Search Terminal Help
cnolasco@calopc: ~/Documents/Ex2-MC833/Ex2.2
cnolasco@calopc:~/Documents/Ex2-MC833/Ex2.2$ ./servidor
Server port number: 33251
Client IP address: 127.0.0.1
Client port number: 57112
```

```
File Edit View Search Terminal Help
cnolasco@calopc: ~/Documents/Ex2-MC833/Ex2.2
cnolasco@calopc:~/Documents/Ex2-MC833/Ex2.2$ gcc cliente.c -o cliente
cnolasco@calopc:~/Documents/Ex2-MC833/Ex2.2$ ./cliente 127.0.0.1
Client IP address: 127.0.0.1
Client port number: 57112
Mon Nov 9 21:00:07 2020
```

A terminal window titled 'cno lasco@calopc: ~/Documents/Ex2-MC833/Ex2.2'. The window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal output shows the command './cliente 127.0.0.1' being executed, followed by two lines of output: 'Client IP address: 127.0.0.1' and 'Client port number: 57124'. The terminal background is dark purple, and the text is light green.

```
cno lasco@calopc: ~/Documents/Ex2-MC833/Ex2.2
cno lasco@calopc:~/Documents/Ex2-MC833/Ex2.2$ ./cliente 127.0.0.1
Client IP address: 127.0.0.1
Client port number: 57124
```

## 4 Questão 2

Para essa questão, meus programas fazem algumas suposições sobre a implementação: a sequência de caracteres são informadas em um só string, separados por espaços entre eles para cada comando (para meu caso, são invocados os comandos `pwd`, `ls` e `ifconfig`); são impressos no terminal do cliente: seu endereço de IP e o número da porta; no terminal do servidor são impressos os resultados dos comandos executados nos cliente, e o endereço de IP e o número da porta enviados por ele. A concorrência é feita pela chamada da função `fork()` no servidor, que então chama a função `strecho()` para tratar o cliente. Semelhantemente, o cliente chama a função `strcli()` para receber e executar os comando, e enviar os resultados para o

servidor. Os resultados enviados pelo cliente também são salvos em um arquivo de texto "output.txt", no mesmo diretório do servidor.

A seguir, imagens do código pertinente a essa implementação.

```
28 int main (int argc, char **argv) {
29     int listenfd, connfd; //Descritores de socket, para escutar e para quando conexão é aceita
30     struct sockaddr_in servaddr; //Struct sockaddr_in para armazenar endereço de domínio e número de porta
31     pid_t childpid;
32
33     if (argc != 2)
34     {
35         printf("Informe número da porta\n");
36         exit(1);
37     }
38
39     listenfd = Socket(AF_INET, SOCK_STREAM, 0); //Inicialização do socket
40
41     bzero(&servaddr, sizeof(servaddr));
42     servaddr.sin_family = AF_INET;
43     servaddr.sin_addr.s_addr = INADDR_ANY; //Adota como endereço de domínio o IP da máquina do servidor
44     servaddr.sin_port = htons(atoi(argv[1])); //Permite que o sistema atribua um número de porta temporário
45
46     Bind(listenfd, (struct sockaddr *)&servaddr, sizeof(servaddr)); //Unir o socket com a porta de número especificada
47     Listen(listenfd, LISTENQ); //Iniciar a escuta por conexões, usando o descritor de socket listenfd criado
48
49     for ( ; ; ) {
50         connfd = Accept(listenfd, (struct sockaddr *) NULL, NULL); //Aceita a conexão, e inicia outro descritor de socket para esta conexão
51
52         /*socklen_t len = sizeof(servaddr);
53         if (getpeername(connfd, (struct sockaddr *)&servaddr, &len) == -1)
54             perror("getpeername");
55         else
56         {
57             char clientIP[16];
58             inet_ntop(AF_INET, &servaddr.sin_addr, clientIP, sizeof(clientIP));
59             printf("Client IP address: %s\n", clientIP);
60             printf("Client port number: %d\n", ntohs(servaddr.sin_port));
61         } //Função antiga para pegar dados de IP/porta de cliente. Não é usada no ex 2.2, mas achei melhor preservar por precaução.
62
63         if ((childpid = Fork()) == 0)
64         {
65             close(listenfd);
66             str_echo(connfd);
67             exit(0);
68         }
69         close(connfd);
70     }
71     return(0);
72 }
73
74 }
```

```
1 #include "wrappers.h"
2
3 #define LISTENQ 10
4 #define MAXDATASIZE 4096
5
6 void str_echo(int sockfd)
7 {
8     char a[] = {"pwd ls ifconfig"}; //Comandos para enviar ao cliente, separados por espaço
9     char buf[MAXDATASIZE];
10    char output_name[16] = "output.txt";
11
12    char string_output [MAXDATASIZE];
13
14    FILE *fp = fopen(output_name, "ab");
15
16    Write(sockfd, a, strlen(a)); //Escrever os comandos no socket
17
18    while( Read(sockfd, buf, MAXDATASIZE) > 0)
19    {
20        printf("%s", buf); //Ler resultados enviados pelo cliente
21        strcat(string_output, buf);
22    }
23
24    fwrite(string_output, strlen(string_output), 1, fp);
25    fclose(fp);
26 }
```



```

1  #include "wrappers.h"
2
3  #define MAXLINE 4096
4
5  void str_cli(int sockfd, struct sockaddr_in servaddr){
6      char buf[MAXLINE]; //Buffer para comandos enviados pelo servidor
7      char command[20]; //string para comando
8      size_t buf_idx = 0; //Index para buffer de comandos
9      size_t cmd_idx = 0; //Index para string de comando
10     char output_buff[MAXLINE]; //Buffer para arquivo de saída do comando
11
12     if(Read(sockfd, buf, MAXLINE) < 0)
13     {
14         perror("read error");
15         exit(1);
16     } //Ler cadeia de caracteres para mensagem enviada pelo servidor.
17
18     while(buf_idx < strlen(buf) + 1)
19     {
20         if(buf[buf_idx] == ' ' || buf[buf_idx] == '\0')
21         {
22             FILE *fp = popen(command, "r"); //Executar o comando e abrir um descritor de arquivo com a sua saída.
23
24             fread(output_buff, MAXLINE, 1, fp);
25
26             Write(sockfd, output_buff, sizeof(output_buff)); //Escrever saída no fd do socket.
27
28             memset(command, 0, strlen(command)); //Zerar buffer de comando para ler próximo
29             cmd_idx = 0;
30
31             fclose(fp);
32         }
33
34         else
35         {
36             command[cmd_idx] = buf[buf_idx];
37             cmd_idx++; //Adiciona cada caracter da lista de comando até achar um comando
38         }
39
40         buf_idx++;
41     }
42
43     char clientIP[16];
44     char port_str[16]; //Strings para IP e porta do cliente
45     uint16_t port_num = 0;
46     char client_info[32];
47
48     socklen_t len = sizeof(servaddr);
49     if (getsockname(sockfd, (struct sockaddr *)&servaddr, &len) == -1)

```

```

43     char clientIP[16];
44     char port_str[16]; //Strings para IP e porta do cliente
45     uint16_t port_num = 0;
46     char client_info[32];
47
48     socklen_t len = sizeof(servaddr);
49     if (getsockname(sockfd, (struct sockaddr *)&servaddr, &len) == -1)
50     {
51         perror("getsockname"); //Usa getsockname para pegar informações do cliente para enviar para o servidor.
52     }
53     else
54     {
55         inet_ntop(AF_INET, &servaddr.sin_addr, clientIP, sizeof(clientIP));
56         printf("Client IP address: %s\n", clientIP);
57         port_num = ntohs(servaddr.sin_port);
58         printf("Client port number: %d\n", port_num);
59     }
60
61     strcat(client_info, clientIP);
62     strcat(client_info, "\n");
63     sprintf(port_str, "%u", port_num);
64     strcat(client_info, port_str);
65     strcat(client_info, "\n"); //Junta strings de IP e porta em uma só e escreve no socket.
66
67     Write(sockfd, client_info, sizeof(client_info));
68 }

```

```

69 int main(int argc, char **argv) {
70     int sockfd; //descritores de socket, para se comunicar com o servidor
71     char error[MAXLINE + 1];
72     struct sockaddr_in servaddr; //Struct sockaddr_in para armazenar endereço de domínio e número de porta
73
74     if (argc != 3) {
75         strcpy(error, "uso: ");
76         strcat(error, argv[0]);
77         strcat(error, " <IPaddress>");
78         perror(error);
79         exit(1);
80     }
81
82     sockfd = Socket(AF_INET, SOCK_STREAM, 0); //Inicialização do socket
83
84     bzero(&servaddr, sizeof(servaddr));
85     servaddr.sin_family = AF_INET;
86     servaddr.sin_port = htons(atoi(argv[2])); //Número de porta do socket do servidor
87     inet_pton(AF_INET, argv[1], &servaddr.sin_addr);
88
89     Connect(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr)); //Faz a requisição de conexão com o servidor, e inicia o descritor de socket sockfd
90
91     str_cli(sockfd, servaddr);
92
93     exit(0);
94 }

```

O código produz os seguintes resultados:

A terminal window titled 'cnolasco@calopc: ~/Documents/Ex2-MC833/Ex2.2' displays the execution of a client program. The user enters the command './cliente 127.0.0.1 34455'. The program outputs 'Client IP address: 127.0.0.1' and 'Client port number: 38244'. The prompt returns to the user, indicating successful execution.

```

cnolasco@calopc: ~/Documents/Ex2-MC833/Ex2.2
cnolasco@calopc:~/Documents/Ex2-MC833/Ex2.2$ ./cliente 127.0.0.1 34455
Client IP address: 127.0.0.1
Client port number: 38244
cnolasco@calopc:~/Documents/Ex2-MC833/Ex2.2$





```

```
File Edit View Search Terminal Help
cnolasco@calopc: ~/Documents/Ex2-MC833/Ex2.2
cnolasco@calopc:~/Documents/Ex2-MC833/Ex2.2$ ./servidor 34455
/home/cnolasco/Documents/Ex2-MC833/Ex2.2
cliente
cliente.c
cliente.o
makefile
servidor
servidor.c
servidor.o
wrappers.c
wrappers.h
wrappers.o
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 17391 bytes 2362100 (2.3 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 17391 bytes 2362100 (2.3 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

tun0: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST> mtu 1500
    inet 172.16.201.135 netmask 255.255.254.0 destination 172.16.201.135
    inet6 fe80::f75e:dfa:7b0d:7229 prefixlen 64 scopeid 0x20<link>
    unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 txqueuelen 100 (UNSPEC)
    RX packets 11341 bytes 2189205 (2.1 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 497 bytes 89492 (89.4 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.0.7 netmask 255.255.255.0 broadcast 192.168.0.255
    inet6 fe80::de34:647d:ffea:a6f2 prefixlen 64 scopeid 0x20<link>
    ether 00:e1:8c:6e:4a:29 txqueuelen 1000 (Ethernet)
    RX packets 5258884 bytes 7428737392 (7.4 GB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1744666 bytes 194888398 (194.8 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

127.0.0.1
38244
█
```

```
Open ▾  output.txt  
~/Documents/Ex2-MC833/Ex2.2 Save ≡   
```

```
/home/cnolasco/Documents/Ex2-MC833/Ex2.2  
cliente  
cliente.c  
cliente.o  
makefile  
output1.txt  
output.txt  
servidor  
servidor.c  
servidor.o  
wrappers.c  
wrappers.h  
wrappers.o  
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536  
    inet 127.0.0.1 netmask 255.0.0.0  
    inet6 ::1 prefixlen 128 scopeid 0x10<host>  
    loop txqueuelen 1000 (Local Loopback)  
    RX packets 20419 bytes 2991710 (2.9 MB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 20419 bytes 2991710 (2.9 MB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
tun0: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST> mtu 1500  
    inet 172.16.201.135 netmask 255.255.254.0 destination 172.16.201.135  
    inet6 fe80::f75e:dfa:7b0d:7229 prefixlen 64 scopeid 0x20<link>  
    unspec 00-00-00-00-00-00-00-00-00-00-00-00 txqueuelen 100 (UNSPEC)  
    RX packets 14545 bytes 2821660 (2.8 MB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 770 bytes 141976 (141.9 KB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
wlp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
    inet 192.168.0.7 netmask 255.255.255.0 broadcast 192.168.0.255  
    inet6 fe80::de34:647d:ffea:a6f2 prefixlen 64 scopeid 0x20<link>  
    ether 00:e1:8c:6e:4a:29 txqueuelen 1000 (Ethernet)  
    RX packets 5879863 bytes 8238903700 (8.2 GB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 1999101 bytes 228876491 (228.8 MB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
127.0.0.1  
46862
```

```
Plain Text ▾ Tab Width: 8 ▾ Ln 1, Col 41 ▾ INS
```

## 5 Questão 3

Para a questão 3, achei melhor comentar as funções antigas da questão 2 `strecho()` e `strcli()` para implementar outras versões que fazem somente o que é pedido para esta questão. Os tempos de conexão e desconexão de um cliente são escritos em um arquivo de texto chamado "serverlog.txt", criado pela `main()` do servidor para esta questão.

```

int main (int argc, char **argv) {
    int listenfd, connfd; //Descritores de socket, para escutar e para quando conexão é aceita
    struct sockaddr_in servaddr; //Struct sockaddr_in para armazenar endereço de domínio e número de porta
    pid_t childpid;

    //ABRIR ARQUIVO PARA QUESTÃO 3
    FILE *fp = fopen("server.log.txt", "ab+");

    if(argc != 2)
    {
        printf("Informe número da porta\n");
        exit(1);
    }

    listenfd = Socket(AF_INET, SOCK_STREAM, 0); //Inicialização do socket

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = INADDR_ANY; //Adota como endereço de domínio o IP da máquina do servidor
    servaddr.sin_port = htons(atoi(argv[1])); //Permite que o sistema atribua um número de porta temporário

    Bind(listenfd, (struct sockaddr *)&servaddr, sizeof(servaddr)); //Unir o socket com a porta de número especificada

    Listen(listenfd, LISTENQ); //Iniciar a escuta por conexões, usando o descritor de socket listenfd criado

    for ( ; ; ) {
        connfd = Accept(listenfd, (struct sockaddr *) NULL, NULL); //Aceita a conexão, e inicia outro descritor de socket para esta conexão

        // QUESTÃO 3 - CLIENTE CONECTOU
        char time_buf[MAXDATASIZE];
        time_t ticks = time(NULL);
        snprintf(time_buf, sizeof(time_buf), "%.24s\r\n", ctime(&ticks));
        fwrite(time_buf, strlen(time_buf), 1, fp);

        /*socklen_t len = sizeof(servaddr);
        if (getpeername(connfd, (struct sockaddr *)&servaddr, &len) == -1)
        {
            perror("getpeername");
        }
        else
        {
            char clientIP[16];
            inet_ntop(AF_INET, &servaddr.sin_addr, clientIP, sizeof(clientIP));
            printf("Client IP address: %s\n", clientIP);
            printf("Client port number: %d\n", ntohs(servaddr.sin_port));
        } */ //Função antiga para pegar dados de IP/porta de cliente. Não é usada no ex 2.2, mas achei melhor preservar por precaução.

        if ((childpid = Fork()) == 0)
        {
            Close(listenfd);

```

```

bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = INADDR_ANY; //Adota como endereço de domínio o IP da máquina do servidor
servaddr.sin_port = htons(atoi(argv[1])); //Permite que o sistema atribua um numero de porta temporário

Bind(listenfd, (struct sockaddr *)&servaddr, sizeof(servaddr)); //Unir o socket com a porta de numero especificada

Listen(listenfd, LISTENQ); //Iniciar a escuta por conexões, usando o descritor de socket listenfd criado

for ( ; ; ) {
    connfd = Accept(listenfd, (struct sockaddr *) NULL, NULL); //Aceita a conexão, e inicia outro descritor de socket para esta conexão

    // QUESTÃO 3 - CLIENTE CONECTOU
    char time_buf[MAXDATASIZE];
    time_t ticks = time(NULL);
    snprintf(time_buf, sizeof(time_buf), "%.24s\r\n", ctime(&ticks));
    fwrite(time_buf, strlen(time_buf), 1, fp);

    /*socklen_t len = sizeof(servaddr);
    if (getpeername(connfd, (struct sockaddr *)&servaddr, &len) == -1)
        perror("getpeername");
    else
    {
        char clientIP[16];
        inet_ntop(AF_INET, &servaddr.sin_addr, clientIP, sizeof(clientIP));
        printf("Client IP address: %s\n", clientIP);
        printf("Client port number: %d\n", ntohs(servaddr.sin_port));
    } */ //Funcao antiga para pegar dados de IP/porta de cliente. Não é usada no ex 2.2, mas achei melhor preservar por precaução.

    if ((childpid = Fork()) == 0)
    {
        Close(listenfd);
        str_echo(connfd, fp);

        memset(time_buf, 0, strlen(time_buf));
        ticks = time(NULL);
        snprintf(time_buf, sizeof(time_buf), "%.24s\r\n", ctime(&ticks));
        fwrite(time_buf, strlen(time_buf), 1, fp);
        exit(0);
    }

    Close(connfd);
}
fclose(fp);
return(0);

```

```

//FUNÇÃO ECHO PARA QUESTÃO 3
void str_echo(int sockfd, FILE *fp)
{
    char buf[MAXDATASIZE];
    char string_output [MAXDATASIZE];

    while( Read(sockfd, buf, MAXDATASIZE) > 0)
    {
        strcat(string_output, buf);
    }

    fwrite(string_output, strlen(string_output), 1, fp);
}

```





```
//FUNÇÃO CLIENTE PARA QUESTÃO 3
void str_cli(int sockfd, struct sockaddr_in servaddr){
    char clientIP[16];
    char port_str[16]; //Strings para IP e porta do cliente
    uint16_t port_num = 0;
    char client_info[32] = "";

    socklen_t len = sizeof(servaddr);
    if (getsockname(sockfd, (struct sockaddr *)&servaddr, &len) == -1)
        perror("getsockname"); //Usa getsocketname para pegar informações do cliente para enviar para o servidor.
    else
    {
        inet_ntop(AF_INET, &servaddr.sin_addr, clientIP, sizeof(clientIP));
        printf("Client IP address: %s\n", clientIP);
        port_num = ntohs(servaddr.sin_port);
        printf("Client port number: %d\n", port_num);
    }

    printf("%s", client_info);

    strcat(client_info, clientIP);
    strcat(client_info, "\n");
    sprintf(port_str, "%u", port_num);
    strcat(client_info, port_str);
    strcat(client_info, "\n"); //Junta strings de IP e porta em uma só e escreve no socket.

    Write(sockfd, client_info, strlen(client_info));
}
```

Open  server\_log.txt  
~/Documents/Ex2-MCB33/Ex2.2 Save   

```
Tue Nov 10 23:16:12 2020
127.0.0.1
48138
Tue Nov 10 23:16:12 2020
```

## 6 Questão 4

### 6.1 4-a - Comando "quit"

Para a questão 4-a, a função `strcli()` agora checa se o comando é "quit". Caso sim, a função retorna e o processo filho é terminado. O comando "quit" foi adicionado no final da cadeia de caracteres na corrente enviada pelo servidor em relação a questão 2.

```
//QUESTÃO 2 - FUNÇÃO CLIENTE PARA QUESTÃO 2
//QUESTÃO 4) A - FECHAR CONEXÃO AO RECEBER O COMANDO "QUIT" DO SERVIDOR
void str_cli(int sockfd, struct sockaddr_in servaddr){
    char buf[MAXLINE]; //Buffer para comandos enviados pelo servidor
    char command[20]; //string para comando
    size_t buf_idx = 0; //Index para buffer de comandos
    size_t cmd_idx = 0; //Index para string de comando
    char output_buff[MAXLINE]; //Buffer para arquivo de saída do comando

    if(Read(sockfd, buf, MAXLINE) < 0)
    {
        perror("read error");
        exit(1);
    } //Ler cadeia de caracteres para mensagem enviada pelo servidor.

    while(buf_idx < strlen(buf) + 1)
    {
        if(buf[buf_idx] == ' ' || buf[buf_idx] == '\0')
        {
            //QUESTÃO 4 - COMANDO QUIT
            if(!strcmp(command, "quit"))
            {
                printf("Comando quit encontrado. Encerrando cliente\n");
                return;
            }

            FILE *fp = popen(command, "r"); //Executar o comando e abrir um descritor de arquivo com a sua saída.

            fread(output_buff, MAXLINE, 1, fp);

            Write(sockfd, output_buff, sizeof(output_buff)); //Escrever saída no fd do socket.

            memset(command, 0, strlen(command)); //Zerar buffer de comando para ler próximo
            cmd_idx = 0;

            fclose(fp);
        }
        else
        {
            command[cmd_idx] = buf[buf_idx];
            cmd_idx++; //Adiciona cada caracter da lista de comando até achar um comando.
        }
    }
}
```



```

//FUNÇÃO ECHO PARA QUESTÃO 2
//QUESTÃO 4) A - FECHAR CONEXÃO AO RECEBER O COMANDO "QUIT" DO SERVIDOR
void str_echo(int sockfd)
{
    //QUESTÃO 2
    //char a[] = {"pwd ls ifconfig"}; //Comandos para enviar ao cliente, separados por espaço

    //QUESTÃO 4)A
    char a[] = {"pwd ls ifconfig quit"};

    char buf[MAXDATASIZE];
    char output_name[16] = "output.txt";

    char string_output [MAXDATASIZE];

    FILE *fp = fopen(output_name, "ab");

    Write(sockfd, a , strlen(a)); //Escrever os comandos no socket

    while( Read(sockfd, buf, MAXDATASIZE) > 0)
    {
        printf("%s", buf); //Ler resultados enviados pelo cliente
        strcat(string_output, buf);
    }

    fwrite(string_output, strlen(string_output), 1, fp);
    fclose(fp);
}

```

## 6.2 4-b - Inverter string recebida pelo cliente

```

//QUESTÃO 4-b) Inverter cada comando recebido pelo cliente
char cpy_cmd[20];
strcpy(cpy_cmd, command);
printf("%s\n", reverse_str(cpy_cmd, strlen(cpy_cmd)));

```

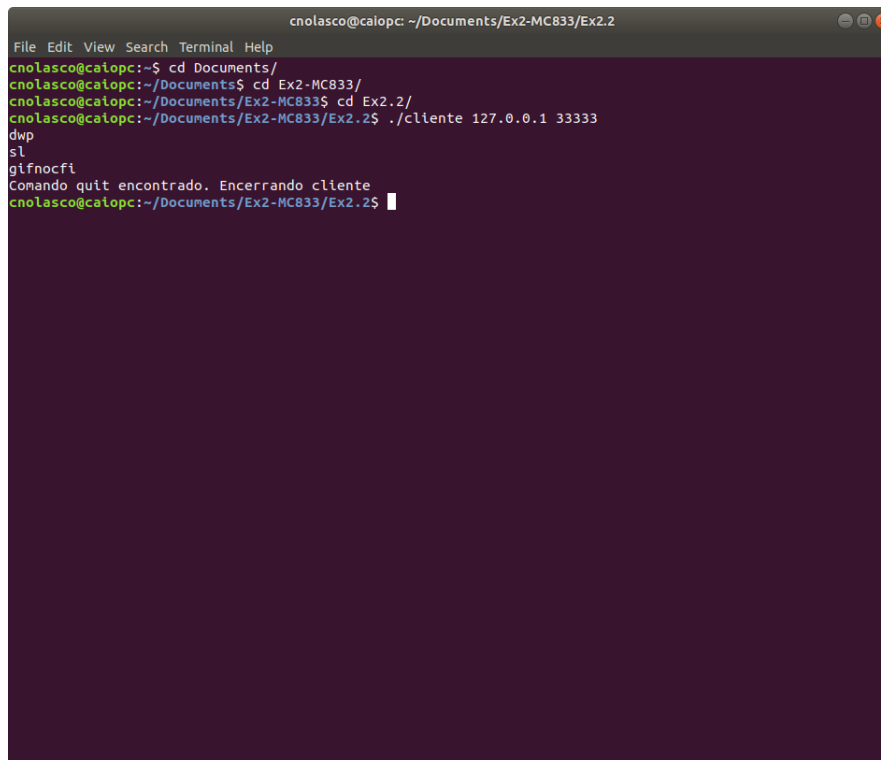
```

char *reverse_str(char *str, int len)
{
    char *p1 = str;
    char *p2 = str + len - 1;

    while (p1 < p2) {
        char tmp = *p1;
        *p1++ = *p2;
        *p2-- = tmp;
    }

    return str;
}

```



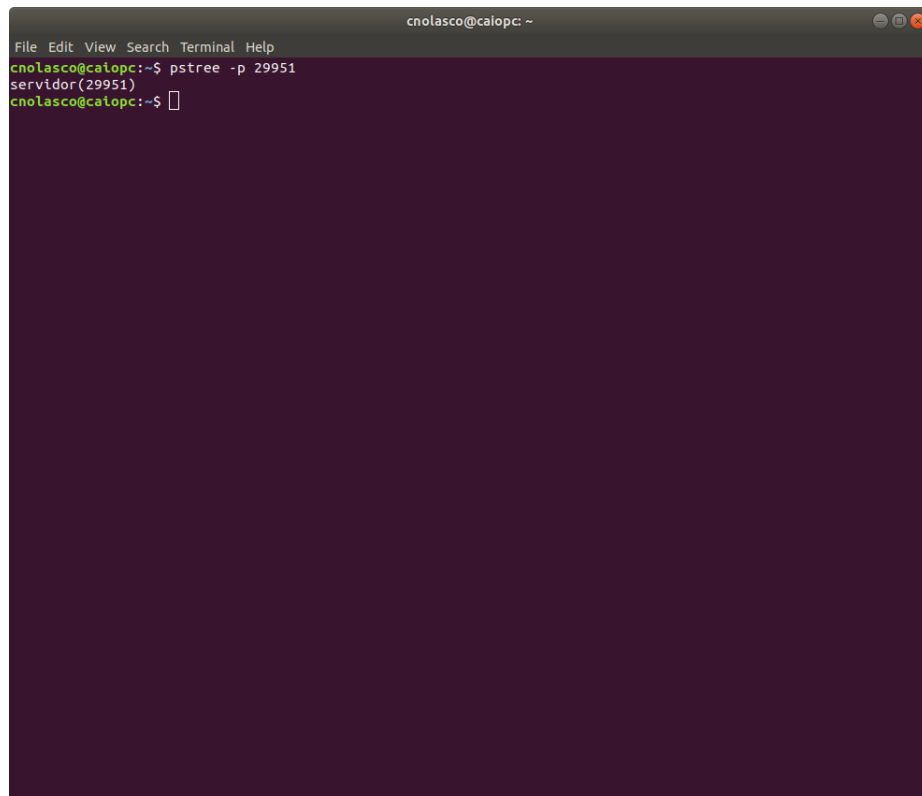
```
cnolasco@calopc: ~/Documents/Ex2-MC833/Ex2.2
File Edit View Search Terminal Help
cnolasco@calopc:~$ cd Documents/
cnolasco@calopc:~/Documents$ cd Ex2-MC833/
cnolasco@calopc:~/Documents/Ex2-MC833$ cd Ex2.2/
cnolasco@calopc:~/Documents/Ex2-MC833/Ex2.2$ ./cliente 127.0.0.1 33333
dwp
sl
gifnocfi
Comando quit encontrado. Encerrando cliente
cnolasco@calopc:~/Documents/Ex2-MC833/Ex2.2$
```

## 7 Questão 5

Não. Mesmo que servidor continue havendo, o processo filho criado pela chamada `fork()` fecha o socket criado pela chamada `listen()`. Após as funções especializadas para o tratamento do cliente são executadas, o processo filho é terminado pela chamada `exit()`, e a conexão com o cliente é fechada.

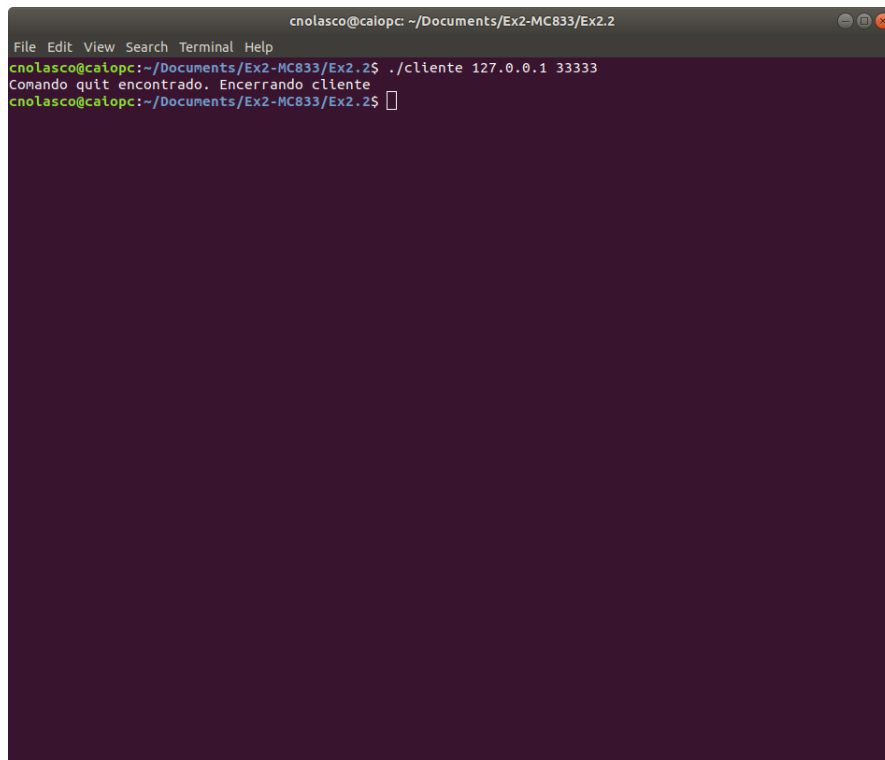
## 8 Questão 6

Com o PID do servidor em execução, usei o comando `ps tree -p PID`, para exibir a árvore do processo pai. Como nenhum cliente conectou no momento, o processo não tem filhos

A terminal window titled 'cnolasco@calopc: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the command 'cnolasco@calopc:~\$ pstree -p 29951' and its output 'servidor(29951)' followed by a blank line and a cursor. The terminal background is dark purple.

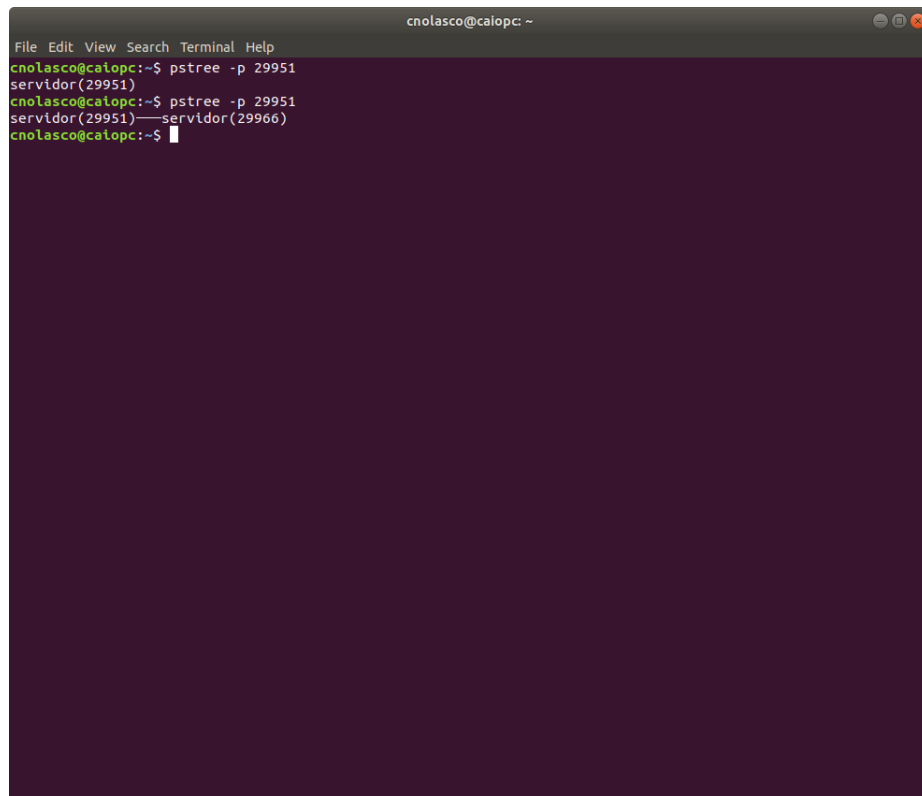
```
File Edit View Search Terminal Help
cnolasco@calopc:~$ pstree -p 29951
servidor(29951)
cnolasco@calopc:~$
```

Faço, então, o pedido conectar com um cliente.

A terminal window with a dark background and light-colored text. The window title is 'cnolasco@calopc: ~/Documents/Ex2-MC833/Ex2.2'. The menu bar includes 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal shows the following commands and output:

```
cnolasco@calopc:~/Documents/Ex2-MC833/Ex2.2$ ./cliente 127.0.0.1 33333
Comando quit encontrado. Encerrando cliente
cnolasco@calopc:~/Documents/Ex2-MC833/Ex2.2$
```

Chamando o comando pstree de novo, pode-se ver que agora o processo pai criou um filho, que lidou com o cliente.

A terminal window titled 'cnoiasco@calopc ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and output:

```
cnoiasco@calopc:~$ pstree -p 29951
servidor(29951)
cnoiasco@calopc:~$ pstree -p 29951
servidor(29951)---servidor(29966)
cnoiasco@calopc:~$
```

Conectando-se outro cliente, pode-se ver que o processo pai cria outro filho.

```
cnolasco@calopc: ~/Documents/Ex2-MC833/Ex2.2
File Edit View Search Terminal Help
cnolasco@calopc:~/Documents/Ex2-MC833/Ex2.2$ ./cliente 127.0.0.1 33333
Comando quit encontrado. Encerrando cliente
cnolasco@calopc:~/Documents/Ex2-MC833/Ex2.2$ ./cliente 127.0.0.1 33333
Comando quit encontrado. Encerrando cliente
cnolasco@calopc:~/Documents/Ex2-MC833/Ex2.2$
```

```
cnolasco@calopc: ~
File Edit View Search Terminal Help
cnolasco@calopc:~$ pstree -p 29951
servidor(29951)
cnolasco@calopc:~$ pstree -p 29951
servidor(29951)---servidor(29966)
cnolasco@calopc:~$ pstree -p 29951
servidor(29951)---servidor(29966)
                    |
                    |---servidor(29980)
cnolasco@calopc:~$
```