



Mateus Restier de Sousa Noronha - 20212103580

João Gabriel Fernandes Moniz de Aragão - 20211107700

João Victor Lira Menke Azarian – 20212104528

Alessandra Drummond Escocard Morisson – 20212104095

Léo Bastos Bellotti – 20212103688

Disciplina: Redes de Computadores

Rio de Janeiro

Setembro de 2022

UNIVERSIDADE VEIGA DE ALMEIDA

Introdução

1 - Explicação da opção pelo uso do TCP ou do UDP.

Nosso grupo optou pelo uso do protocolo TCP, por ser um meio mais confiável, e que garante que irá entregar todos os dados corretamente, mesmo que mais lentamente. Como o trabalho se tratava de uma transferência de arquivos, o mesmo não poderia chegar corrompido, ou incompleto. O protocolo UDP apesar de ser mais rápido em transferência, não é confiável a esse ponto.

2 - Explicação do funcionamento do código da aplicação, mostrando o(s) cabeçalho(s) utilizado(s)

Para começar essa explicação, iremos falar sobre o cabeçalho do código:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#include <time.h>
```

`#include <stdio.h>`

Essa biblioteca é responsável pela entrada e saída padrão, como `scanf` e `printf`, e as funções simples da linguagem, como `if`, `for`, `while` entre outras.

`#include <stdlib.h>`

Essa biblioteca é responsável pela manipulação de alocação e realocação de memória.

`#include <unistd.h>`

Essa biblioteca é responsável por fornecer acesso à API do sistema operacional POSIX.

`#include <string.h>`

Essa biblioteca é responsável por diversas funções para manipular strings.

`#include <arpa/inet.h>`

Essa biblioteca é responsável por algumas funções utilizadas para o funcionamento do código em relação a conexões e protocolos, como `"in_port_t"` e `"in_addr_t"`

`#include <time.h>`

Essa biblioteca é responsável por mexer com funções relacionadas a data do sistema

2.1 – Explicação do funcionamento do código

O código funcionará da seguinte forma, primeiro precisamos abrir o servidor, que irá receber o arquivo, e após isso o cliente deverá se conectar a ele.

Vamos começar falando pelo código do servidor:

O primeiro passo é a criação de variáveis, e a definição do IP e da porta, no caso estaremos utilizando o localhost 127.0.0.1 pois a transferência será realizada dentro no mesmo computador, mas para diretórios diferentes, porém poderia ser realizado entre máquinas alterando essas informações

```
int main(){
    char *ip = "127.0.0.1"; //ip utilizado
    int port = 8080; //porta utilizada
    int e;

    int sockfd, new_sock;
    struct sockaddr_in server_addr, new_addr; //informacoes do servidor
    socklen_t addr_size;
    char buffer[SIZE];
```

2.1.1 – Criação da conexão com servidor e socket TCP

O próximo passo é criar a conexão com o servidor, e para isso criaremos o socket TCP “SOCK_Stream”, e caso não seja possível criar o socket aparecerá uma mensagem de erro e o programa encerra, após isso será a construção do bind. Caso aconteça o mesmo erro, também será exibido uma mensagem de erro, e após isso, de fato a conexão será estabelecida.

```
//criacao do socket
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if(sockfd < 0) {
    perror("Erro no socket");
    exit(1);
}
printf("Socket do servidor foi criado com sucesso.\n");
//conectar ao servidor
server_addr.sin_family = AF_INET;
server_addr.sin_port = port; //configura a porta do servidor para conexao
server_addr.sin_addr.s_addr = inet_addr(ip);

//contrucao do bind
e = bind(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr));
if(e < 0) {
    perror("Erro no bind.");
    exit(1);
}
printf("Bind sucedido.\n");

//tratando conexoes
if(listen(sockfd, 10) == 0){
    printf("Esperando conexao...\n");
}else{
    perror("Erro ao conectar");
    exit(1);
}

addr_size = sizeof(new_addr);
new_sock = accept(sockfd, (struct sockaddr*)&new_addr, &addr_size);
```

Nesse momento o Servidor irá aguardar a conexão do cliente, que tem a princípio o código bem parecido com o server, com principalmente a diferença da criação da variável char *filename, responsável por definir o nome do arquivo a ser transferido (e deve estar na mesma pasta do arquivo cliente.c).

```
int main()
{
    char *ip = "127.0.0.1"; // ip utilizado
    int port = 8080;        // porta utilizada
    int e, n;

    int sockfd;
    struct sockaddr_in server_addr; // informacoes do servidor
    FILE *fp;
    char *filename = "arquivo.txt"; // nome do arquivo

    // criacao do socket
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
    {
        perror("Erro no socket");
        exit(1);
    }
    printf("Socket do servidor foi criado com sucesso.\n");

    // conectar ao servidor
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = port; // configura a porta do servidor para conexao
    server_addr.sin_addr.s_addr = inet_addr(ip);

    e = connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr));
    if (e == -1)
    {
        perror("Erro no socket.");
        exit(1);
    }
    printf("Conectado ao servidor.\n");
```

2.1.2 – Cálculo do tamanho do arquivo

Após isso, com a função “fopen”, iremos abrir o arquivo.txt para leitura, e logo após isso calcular seu tamanho com a função calcularTamanhoArquivo para mais tarde descobrir a taxa de transferência.

```
// abrir o arquivo
fp = fopen(filename, "r");
long tamanho = calcularTamanhoArquivo(fp);
if (fp == NULL)
{
    perror("Erro ao ler o arquivo.");
    exit(1);
}
```

```

long calcularTamanhoArquivo(FILE *arquivo)
{
    // guarda o estado ante de chamar a função fseek
    long posicaoAtual = ftell(arquivo);

    // guarda tamanho do arquivo
    long tamanho;

    // calcula o tamanho
    fseek(arquivo, 0, SEEK_END);
    tamanho = ftell(arquivo);

    // recupera o estado antigo do arquivo
    fseek(arquivo, posicaoAtual, SEEK_SET);

    return tamanho;
}

```

2.1.3 – Envio do arquivo

Após isso, será de fato realizado o envio do arquivo, juntamente com o calculo da taxa de transferência que será explicada no próximo tópico.

```

// enviar o arquivo
char data[SIZE] = {0};

struct timeval start, end;

gettimeofday(&start, NULL); // primeira captura do horário

while (fgets(data, SIZE, fp) != NULL)
{
    if (send(sockfd, data, sizeof(data), 0) == -1)
    {
        perror("Erro ao enviar o arquivo.");
        exit(1);
    }
    bzero(data, SIZE);
}

// segunda captura do horário
gettimeofday(&end, NULL);
double segundos = (double)((((end.tv_sec - start.tv_sec) * 1000000) + end.tv_usec) - (start.tv_usec))/1000000;

double mbps = (((double)tamanho * 8) / segundos) / 1000000;
printf("\nVelocidade: %lf\n", mbps);

printf("Arquivo enviado com sucesso.\n");

printf("Fechando a conexao.\n");
close(sockfd);

return 0;

```

2.1.4 – Arquivo reescrito em uma pasta do server.c

Com o arquivo enviado, ele será reescrito em um arquivo novo na pasta do server.c com a função `write_file`, dessa forma finalizando a transferência.

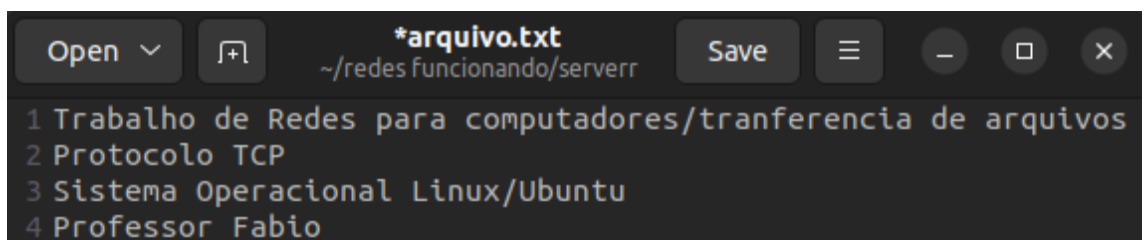
```
//reescrever arquivo do cliente no destino (servidor)
write_file(new_sock);

printf("Arquivo transferido com sucesso.\n");
```

```
//funcao para reescrever o arquivo txt no destino
void write_file(int sockfd){
    int n;
    FILE *fp;
    char *filename = "arquivo.txt";
    char buffer[SIZE];

    fp = fopen(filename, "w");
    while (1) {
        n = recv(sockfd, buffer, SIZE, 0);
        if (n <= 0){
            break;
            return;
        }
        fprintf(fp, "%s", buffer);
        bzero(buffer, SIZE);
    }
    return;
}
```

Dessa forma foi concluído o funcionamento do código, e o arquivo foi entregue com sucesso



The screenshot shows a terminal window with a dark background. The title bar at the top indicates the file is `*arquivo.txt` located at `~/redes funcionando/serverr`. The window has standard Linux window controls (Open, Save, and window management icons). The terminal content displays the text of the file, which is a list of four items:

```
1 Trabalho de Redes para computadores/tranferencia de arquivos
2 Protocolo TCP
3 Sistema Operacional Linux/Ubuntu
4 Professor Fabio
```


3 - Explicação de como foi realizado o cálculo da taxa de transferência.

Primeiramente, pegamos o tamanho do arquivo usando a função “calcularTamanhoArquivo”, depois disso, usamos a função “gettimeofday” salvamos no programa o horário do sistema, antes de realizar a transferência, e depois de realizar também. Com esses dois tempos, podemos subtrair-los e obter o tempo da transferência. Para obter o resultado em Megabits por segundo, multipliquei o tempo por 8, para obter o resultado em bits.

Após isso, dividimos pelos segundos obtendo a taxa em bits por segundo, e depois de todos esses passos divide-se o número obtido por 1000000, para obter os megabits por segundo.

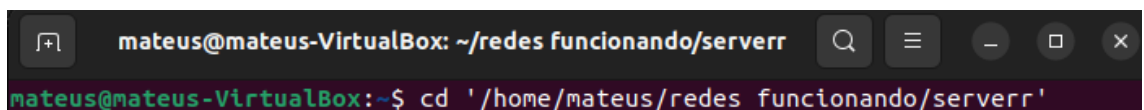
```
double mbps = (((double)tamanho * 8) / segundos) / 1000000;  
printf("\nVelocidade: %lf\n", mbps);
```

4 - Explicação de como o programa deve ser compilado e executado.

Nosso programa funciona independente de qualquer IDE, portanto explicaremos como compila-lo pelo terminal do Linux.

O primeiro passo é executar o servidor. Sendo assim, devemos acessar a pasta onde se encontra o arquivo do programa. Para fazer isso, basta digitar: “CD (diretório da pasta) ”.

Assim como no print abaixo:

A screenshot of a Linux terminal window. The title bar shows 'mateus@mateus-VirtualBox: ~/redes funcionando/serverr'. The terminal text shows the prompt 'mateus@mateus-VirtualBox:~\$' followed by the command 'cd '/home/mateus/redes funcionando/serverr'' which has been executed, changing the directory to the server code folder.

```
mateus@mateus-VirtualBox:~$ cd '/home/mateus/redes funcionando/serverr'
```

Após isso, devemos compilar o arquivo server.c, para fazer isso digitamos “gcc server.c -o server”

```
mateus@mateus-VirtualBox:~/redes funcionando/serverr$ gcc server.c -o server
```

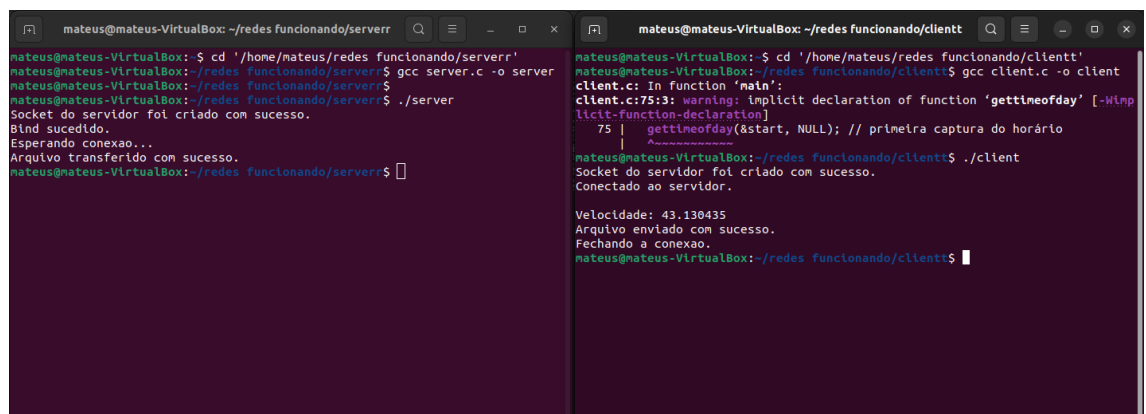
Após isso, o arquivo estará compilado sem nenhum problema, agora para executá-lo, digitamos “./server”

```
mateus@mateus-VirtualBox:~/redes funcionando/serverr$ ./server
Socket do servidor foi criado com sucesso.
Bind sucedido.
Esperando conexao...
```

Agora que o servidor está aberto, precisamos conectar o cliente. Para fazer isso, seguimos todos os passos feitos com o servidor, mas dessa vez com o cliente.c, após isso ambos os programas foram compilados com sucesso.

5 - Prints de tela mostrando o funcionamento do programa.

A seguir seguem os prints do programa funcionando, a esquerda o servidor.c e a direita cliente.c, com velocidade de 43.13 megabits por segundo.



```
mateus@mateus-VirtualBox: ~/redes funcionando/serverr
mateus@mateus-VirtualBox: $ cd '/home/mateus/redes funcionando/serverr'
mateus@mateus-VirtualBox: ~/redes funcionando/serverr$ gcc server.c -o server
mateus@mateus-VirtualBox: ~/redes funcionando/serverr$
mateus@mateus-VirtualBox: ~/redes funcionando/serverr$ ./server
Socket do servidor foi criado com sucesso.
Bind sucedido.
Esperando conexao...
Arquivo transferido com sucesso.
mateus@mateus-VirtualBox: ~/redes funcionando/serverr$

mateus@mateus-VirtualBox: ~/redes funcionando/clientt
mateus@mateus-VirtualBox: $ cd '/home/mateus/redes funcionando/clientt'
mateus@mateus-VirtualBox: ~/redes funcionando/clientt$ gcc client.c -o client
client.c: In function 'main':
client.c:75:3: warning: implicit declaration of function 'gettimeofday' [-Wimplicit-function-declaration]
   75 |     gettimeofday(&start, NULL); // primeira captura do horário
      |     ^~~~~~
mateus@mateus-VirtualBox: ~/redes funcionando/clientt$ ./client
Socket do servidor foi criado com sucesso.
Conectado ao servidor.

Velocidade: 43.130435
Arquivo enviado com sucesso.
Fechando a conexao.
mateus@mateus-VirtualBox: ~/redes funcionando/clientt$
```

Foi usada uma máquina virtual com o sistema operacional Ubuntu instalado para o funcionamento do sistema.

Código do programa client.c

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/time.h>

#include <arpa/inet.h>


#include <time.h> // biblioteca para mexer com data

#define SIZE 1024


long calcularTamanhoArquivo(FILE *arquivo)
{

    // guarda o estado ante de chamar a função fseek

    long posicaoAtual = ftell(arquivo);


    // guarda tamanho do arquivo

    long tamanho;


    // calcula o tamanho

    fseek(arquivo, 0, SEEK_END);

    tamanho = ftell(arquivo);
```

```
// recupera o estado antigo do arquivo

fseek(arquivo, posicaoAtual, SEEK_SET);

return tamanho;
}

int main()
{
    char *ip = "127.0.0.1"; // ip utilizado
    int port = 8080;        // porta utilizada
    int e, n;

    int sockfd;

    struct sockaddr_in server_addr; // informacoes do servidor

    FILE *fp;

    char *filename = "arquivo.txt"; // nome do arquivo

    // criacao do socket

    sockfd = socket(AF_INET, SOCK_STREAM, 0);

    if (sockfd < 0)
    {
        perror("Erro no socket");
        exit(1);
    }
}
```

```
}

printf("Socket do servidor foi criado com sucesso.\n");

// conectar ao servidor

server_addr.sin_family = AF_INET;

server_addr.sin_port = port; // configura a porta do servidor para conexao

server_addr.sin_addr.s_addr = inet_addr(ip);


e = connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr));

if (e == -1)

{

    perror("Erro no socket.");

    exit(1);

}

printf("Conectado ao servidor.\n");


// abrir o arquivo

fp = fopen(filename, "r");

long tamanho = calcularTamanhoArquivo(fp);

if (fp == NULL)

{

    perror("Erro ao ler o arquivo.");

    exit(1);

}
```

```
// enviar o arquivo
```

```
char data[SIZE] = {0};
```

```
struct timeval start, end;
```

```
gettimeofday(&start, NULL); // primeira captura do horário
```

```
while (fgets(data, SIZE, fp) != NULL)
```

```
{
```

```
    if (send(sockfd, data, sizeof(data), 0) == -1)
```

```
    {
```

```
        perror("Erro ao enviar o arquivo.");
```

```
        exit(1);
```

```
    }
```

```
    bzero(data, SIZE);
```

```
}
```

```
// segunda captura do horário
```

```
gettimeofday(&end, NULL);
```

```
double segundos = (double)((((end.tv_sec - start.tv_sec) * 1000000) +  
end.tv_usec) - (start.tv_usec))/1000000;
```

```
double mbps = (((double)tamanho * 8) / segundos) / 1000000;
```

```
printf("\nVelocidade: %lf\n", mbps);
```

```
printf("Arquivo enviado com sucesso.\n");
```

```
printf("Fechando a conexao.\n");
```

```
close(sockfd);
```

```
return 0;
```

```
}
```

Código do programa server.c

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <arpa/inet.h>

#include <sys/time.h>


#define SIZE 1024


//funcao para reescrever o arquivo txt no destino

void write_file(int sockfd){

    int n;

    FILE *fp;

    char *filename = "arquivo.txt";

    char buffer[SIZE];


    fp = fopen(filename, "w");

    while (1) {

        n = recv(sockfd, buffer, SIZE, 0);

        if (n <= 0){

            break;

            return;

        }

    }

}
```



```
    }  
  
    fprintf(fp, "%s", buffer);  
  
    bzero(buffer, SIZE);  
  
    }  
  
    return;  
  
}
```

```
int main(){  
  
    char *ip = "127.0.0.1"; //ip utilizado  
  
    int port = 8080; //porta utilizada  
  
    int e;  
  
  
    int sockfd, new_sock;  
  
    struct sockaddr_in server_addr, new_addr; //informacoes do servidor  
  
    socklen_t addr_size;  
  
    char buffer[SIZE];  
  
  
    //criacao do socket  
  
    sockfd = socket(AF_INET, SOCK_STREAM, 0);  
  
    if(sockfd < 0) {  
  
        perror("Erro no socket");  
  
        exit(1);  
  
    }  
  
    printf("Socket do servidor foi criado com sucesso.\n");
```

```
//conectar ao servidor
```

```
server_addr.sin_family = AF_INET;
```

```
server_addr.sin_port = port; //configura a porta do servidor para conexao
```

```
server_addr.sin_addr.s_addr = inet_addr(ip);
```

```
//contrucao do bind
```

```
e = bind(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr));
```

```
if(e < 0) {
```

```
    perror("Erro no bind.");
```

```
    exit(1);
```

```
}
```

```
printf("Bind sucedido.\n");
```

```
//tratando conexoes
```

```
if(listen(sockfd, 10) == 0){
```

```
printf("Esperando conexao...\n");
```

```
}else{
```

```
perror("Erro ao conectar");
```

```
    exit(1);
```

```
}
```

```
addr_size = sizeof(new_addr);
```

```
new_sock = accept(sockfd, (struct sockaddr*)&new_addr, &addr_size);
```

```
//reescrever arquivo do cliente no destino (servidor)

write_file(new_sock);

printf("Arquivo transferido com sucesso.\n");

return 0;

}
```

Referências:

<https://idiotdeveloper.com/file-transfer-using-tcp-socket-in-c/>