

Trabalho de Redes de Computadores

Gabriel Fagá Monteiro

Universidade Estadual de Mato Grosso do Sul

Sumário

1 - Resumo	03
2 - Descrições Gerais	04
2.1 - Servidor	04
2.2 - Cliente	10
2.3 - Mensagens de Erro	12
2.4 - Makefile	13
3 - Retransmissão de Mensagens	13
4 - Testes	15
5 - Conclusão	16
6 - Referências Bibliográficas	16

1 - Resumo

Proposto como trabalho prático da disciplina de Redes de Computadores, este programa representa uma ferramenta que possibilita que usuários criem, editem e visualizem documentos de texto, podendo compartilhá-los com outros usuários. O foco principal tem base na criação de documentos colaborativos onde cada integrante pode adicionar ideias. Um documento possui níveis de interações como edição, comentários e visualização.

Palavras-chave: WebDocs, Documentos, Edição de Texto.

2 - Descrições Gerais

2.1 - Servidor

Para a construção do servidor responsável pela edição geral dos documentos foram utilizadas bibliotecas da linguagem C - além das padrões como “stdio.h” e “stdlib.h” - como a “pthread.h” dividindo o trabalho e permitindo que cada usuário tenha seu próprio “perfil” e também a “semaphore.h” com a função de impedir que as threads implementadas sobreponham umas às outras, resolvendo assim o problema de um documento ser alterado para um usuário de uma forma e não para outro.

```
/* Definição de constantes úteis */  
#define MAX_CLIENTE 500  
#define MAX_FILE 50  
#define BUFFER_SIZE 2048
```

Figura 1 - Constantes Globais - Fonte: Recursos próprios - Disponível no arquivo <servidor.c>

As constantes acima controlam, respectivamente, a quantidade máxima de clientes que podem ser registrados no servidor, a quantidade máxima de arquivos que o servidor é capaz de gerar e o tamanho do buffer que será responsável pela gravação de arquivos.

```
struct File  
{  
    char filename[50];  
    char owner[50];  
};  
  
struct Usuario  
{  
    char nome[50];  
};
```

Figura 2 - Estruturas File e Usuário - Fonte: Recursos Próprios - Disponível no arquivo <servidor.c>

A estrutura File tem como função guardar o nome do arquivo - que pode ser de até 50 caracteres - e o nome do criador do arquivo, já a estrutura Usuário é responsável por armazenar o nome dos usuários.

```
/***** Variáveis globais *****/
/* Contagem de clientes */
int clientCount = 0;

/* Semáforo para sincronizar as threads */
sem_t sinaleiro;

/* Protótipo da função de cada thread filha */
void *clientThread(void *param);

/* Vetor global para guardar os usuários */
struct Usuario allocatedUsers[MAX_CLIENTE];

/* Vetor global para guardar as informações dos arquivos do cliente */
struct File allocatedFiles[MAX_CLIENTE];
```

Figura 3 - Variáveis Globais - Fonte: Recursos Próprios - Disponível no arquivo <servidor.c>

As variáveis globais acima possuem funções descritas durante o código, foram declaradas como globais para facilitar a manipulação e garantir que os dados sejam atualizados para todas as funções, visando reduzir os erros.

Para adicionar um novo registro de cliente, foi criada uma função com o nome de “addRergister”, que tem como parâmetros o nome do usuário e o nome do arquivo, ela possui um loop principal que só executado enquanto a variável de controle de usuários cadastrados for menor que o máximo de clientes.

Para realizar um cadastro de um novo cliente e seu arquivo, após estabelecer a conexão entre o cliente e o servidor - detalhado no arquivo readme.md - deve-se digitar no terminal do cliente o comando N <NomeUsuario> <NomeArquivo>.

```
if ((strcmp(username, allocatedUsers[i].nome)) == 0)
{
    /* Verificar se existe algum arquivo com o nome passado */
    for (int j=0; j<MAX_CLIENTE; j++)
    {
        if ((strcmp(filename, allocatedFiles[j].filename)) == 0)
            return 2;
    }

    /* Criar arquivo caso não exista com o nome passado */
    memset(tempFilename, 0, sizeof(tempFilename));
    strcat(tempFilename, filename);
    strcat(tempFilename, ".txt");

    writeFile = fopen(tempFilename, "w");

    /* Verificar qual posição do vetor de arquivos está livre */
    for (int j=0; j<MAX_CLIENTE; j++)
    {
        if (allocatedFiles[j].filename[0] == '\0')
        {
            strcpy(allocatedFiles[j].filename, filename);
            strcpy(allocatedFiles[j].owner, username);

            fclose(writeFile);
            return 1;
        }
    }
}
```

Figura 4- Cadastro do Usuário - Fonte: Recursos Próprios - Disponível no arquivo <servidor.c>

Este trecho de código verifica se um usuário com este nome já está cadastrado e também analisa se já existe um arquivo com o nome já registrado, após a verificação cria um arquivo que recebe o nome do arquivo digitando durante o uso da operação N, e.g: AulaRedes.txt/Notas2023.txt. O restante da função é apenas um tratamento de cadastro do usuário caso este seja seu primeiro registro, o nome do arquivo segue a mesma nomenclatura.

Para que um cliente possa editar um arquivo existente foi criada uma função chamada “messageReceive”, que recebe como parâmetros o nome do usuário, o nome do arquivo, o conteúdo a ser inserido no arquivo e também o número do parágrafo desejado, caso seja de interesse do usuário.

Para funcionamento correto do servidor, primeiramente verifica-se a existência do usuário no vetor de usuários, caso o usuário exista é realizada uma busca do nome do arquivo no vetor de arquivos, tendo todas as condições satisfeitas é feita a contagem de parágrafos que o arquivo possui, por fim avança para a inserção da mensagem no arquivo, onde foram implementados tratamentos para inserção no início, fim e parágrafo específico.

Para realizar a edição de um documento já criado anteriormente deve-se digitar no terminal do cliente: S <NomeUsuario> <NomeArquivo> [CONTEÚDO A SER GRAVADO NO ARQUIVO] [pos]. Os argumentos envoltos pelo símbolo de colchete devem ser digitados seguindo os exemplos abaixo para que o código funcione corretamente.

Linha de código	Local de Inserção
S Gabriel MeusDados [Nome: Gabriel Fagá Monteiro] [inicio]	Início
S Gabriel MeusDados [Curso: Ciência da Computação] P N	Parágrafo N
S Gabriel MeusDados [RGM: 43560] [fim]	Fim

Tabela 1 - Exemplos de execução para a instrução ‘S’ - Fonte: Recursos Próprios

```
/* Verificar qual posição inserir no arquivo */
/* Caso 1: inserir no início */
if ((strcmp(paragraphPosition, "[início]")) == 0)
{
    writeFile = fopen(tempFilename, "w");
    fprintf(writeFile, "[%s] %s\n", allocatedUsers[i].nome, paragraphContent);

    for (int a=0; a<paragraphCount; ++a)
        fprintf(writeFile, "%s", tempString[a]);

    fclose(writeFile);
}
else if ((strcmp(paragraphPosition, "[fim]")) == 0)
{
    /* Caso 2: inserir no fim */
    writeFile = fopen(tempFilename, "w");

    for (int a=0; a<paragraphCount; ++a)
        fprintf(writeFile, "%s", tempString[a]);

    fprintf(writeFile, "[%s] %s\n", allocatedUsers[i].nome, paragraphContent);

    fclose(writeFile);
}
else if (paragraphPosition[0] == 'P')
{
    /* Caso 3: inserir no parágrafo desejado */
    if (paragraphNumber > paragraphCount || paragraphNumber <= 0)
        return -3;

    writeFile = fopen(tempFilename, "w");
    for (int a=0; a<paragraphCount; ++a)
    {
        if ((a+1) == paragraphNumber)
            fprintf(writeFile, "[%s] %s\n", allocatedUsers[i].nome, paragraphContent);

        fprintf(writeFile, "%s", tempString[a]);
    }

    fclose(writeFile);
}
```

Figura 5 - Tratamento de Inserção - Fonte: Recursos Próprios - Disponível em: <servidor.c>

Para que os clientes possam ser listados, foi implementada a função “verifyRegister” que tem a função de verificar se o cliente está registrado e listar os arquivos que ele já registrou, juntamente com a quantidade de parágrafos que este arquivo possui. Para executá-lo basta digitar: L <NomeUsuário>

```
/* Verificar se o usuário existe */
for(int i=0; i < MAX_CLIENTE; ++i)
{
    /* Caso o usuário não exista */
    if((strcmp(username, allocatedUsers[i].nome)) != 0)
    {
        if(i+1 == MAX_CLIENTE)
        {
            strcpy(tempStr, "L - 1\n");
            return tempStr;
        }
    }
}

/* Caso o usuário exista */
if(strcmp(username, allocatedUsers[i].nome) == 0)
{
    /* Colocar na tempString o L 0 */
    strcpy(tempStr, "L 0\n");
    /* Verificar o vetor de arquivos procurando os arquivos criados pelo mesmo */
    for(int j=0; j < MAX_CLIENTE; ++j)
    {
        paragraphCount = 0;
        /* Caso o usuário seja o dono, adiciona na tempStr */
        if(strcmp(username, allocatedFiles[j].owner) == 0)
        {
            /* Colocar na variável temporária de arquivo, o nome do mesmo conforme foi gravado */
            memset(tempFilename, 0, sizeof(tempFilename));
            strcpy(tempFilename, allocatedFiles[j].filename);
            strcat(tempFilename, ".txt");

            writeFile = fopen(tempFilename, "r");

            /* Contagem dos parágrafos do arquivo */
            while((fgets(tempParagraph, BUFFER_SIZE, writeFile)) != NULL)
                ++paragraphCount;

            fclose(writeFile);
            char count[4];
            sprintf(count, "%d", paragraphCount);
            /* Inicializar tempParagraph e concatenar as informações de cada linha */
            memset(tempParagraph, 0, sizeof(tempParagraph));
            strcat(tempParagraph, tempFilename);
            strcat(tempParagraph, " ");
            strcat(tempParagraph, count);
            strcat(tempParagraph, " ");
            strcat(tempParagraph, username);
            strcat(tempParagraph, "\n");
            strcat(tempStr, tempParagraph);
        }
    }
    return tempStr;
}
```

Figura 6 - Loop para percorrer todos os clientes cadastrados e imprimir os arquivos do cliente informado na requisição do serviço L - Fonte: Recursos Próprios - Disponível em: <servidor.c>

No trecho de código abaixo são declaradas algumas variáveis utilizadas durante o código, assim como o semáforo responsável por manter a ordem das alterações realizadas nos arquivos, e também os descritores que são responsáveis por transmitir informações entre o cliente e o servidor.


```
int main(int argc, char **argv)
{
    /* Iniciando semáforo para sincronização das threads */
    sem_init(&sinaleiro, 0, 1);

    /* Criando pthread e mutex iniciado */
    pthread_t clients;

    /* Estruturas para o servidor e cliente */
    struct sockaddr_in serverAddress;
    struct sockaddr_in clientAddress;

    /* Descritores para o servidor e cliente */
    int serverSocket, clientSocket, *newSocket;

    char buffer[BUFFER_SIZE];
```

Figura 7 - Função Principal - Fonte: Recursos Próprios - Disponível no arquivo <servidor.c>

Para evitar erros durante a execução de testes e funcionamento do servidor, foi utilizada a função “setsockopt”, permitindo com que o servidor reutilize portas utilizadas anteriormente.

```
while (1)
{
    /* Obtendo o tamanho do endereço do cliente */
    socklen_t clientAddrLength = sizeof(clientAddress);

    /* Aceita a conexão do cliente */
    clientSocket = accept(serverSocket, (struct sockaddr*)&clientAddress, &clientAddrLength);
    /* Verifica se está na capacidade máxima */
    if(++clientCount == MAX_CLIENTE)
    {
        /* Mensagem de erro para o servidor */
        fprintf(stderr, "ERRO ES06\nVerifique o arquivo DocumentaçãoWebdocs.pdf\n\n");
        /* Mensagem de erro para o cliente */
        sprintf(buffer, "ERRO EC05\nVerifique o arquivo DocumentaçãoWebdocs.pdf\n\n");
        /* Envia a mensagem e encerra a conexão */
        send(clientSocket, buffer, strlen(buffer), 0);
        shutdown(clientSocket, SHUT_RDWR);
        close(clientSocket);
        continue;
    }

    /* Cria thread para o cliente */
    newSocket = malloc(1);
    *newSocket = clientSocket;
    clientCount++;
    if(pthread_create(&clients, NULL, clientThread, (void *)newSocket) < 0)
    {
        fprintf(stderr, "ES07\nVerifique o arquivo DocumentaçãoWebdocs.pdf\n\n");
        exit(-1);
    }

    memset(buffer, 0, BUFFER_SIZE);
}
```

Figura 8 - Loop Principal - Fonte: Recursos Próprios - Disponível no arquivo <servidor.c>

Função responsável por admitir as operações realizadas no servidor, criando uma thread para cada cliente, a não ser que haja mais clientes que o definido anteriormente.

2.2 - Cliente

Para a construção do cliente foram utilizadas praticamente as mesmas bibliotecas utilizadas na construção do servidor, com exceção da “semaphore.h”. Também foi definida a variável BUFFER_SIZE responsável por armazenar o tamanho do buffer responsável pela gravação de arquivos.

```
/* Ponteiro para a tabela de entrada do host */
struct hostent *hostTable;

/* Estrutura para o servidor */
int serverSocket;
struct sockaddr_in serverAddress;

/* Variáveis para salvar o nome do host e o numero da porta */
char *host;
short port;

/* Buffer de envio e recebimento */
char buffer[BUFFER_SIZE];
char receiveBuffer[BUFFER_SIZE];

/* Variáveis para controle da quantidade de bytes recebidos e enviados */
int received;
```

Figura 9 - Variáveis utilizadas para o código Cliente - Fonte: Recursos Próprios - Disponível no arquivo <cliente.c>

O código cliente é evidentemente mais simples que o código do servidor, visto que sua função se baseia somente em enviar mensagens para que o servidor as processe e execute da maneira esperada, retornando o resultado para o cliente que mostra o resultado da operação para o usuário. Logo, todas as operações - de controle, envio e tratamentos de erro - foram executadas na função principal.

```
/* Mapeando valores da estrutura do servidor */
serverAddress.sin_family = AF_INET;
serverAddress.sin_port = htons(port);
serverAddress.sin_addr.s_addr = *((unsigned long *)hostTable->h_addr);

memset(&serverAddress.sin_zero, 0, sizeof(serverAddress.sin_zero));

if(connect(serverSocket, (struct sockaddr*)&serverAddress, sizeof(serverAddress)) == -1)
{
    perror("\nCÓDIGO: EC05\nVerifique o arquivo DocumentaçãoWebDocs.pdf\n\n");
    close(serverSocket);
    exit(-1);
}

printf("> ");
fflush(stdout);
fgets(buffer, BUFFER_SIZE, stdin);
fflush(stdin);
buffer[strlen(buffer)-1] = '\0';

if(strcmp(buffer, "exit") == 0)
{
    close(serverSocket);
    exit(-1);
}

send(serverSocket, buffer, strlen(buffer), 0);

/* Parte do Servidor */
received = recv(serverSocket, receiveBuffer, sizeof(buffer), 0);
receiveBuffer[received] = '\0';
printf("%s\n", receiveBuffer);

close(serverSocket);
```

Figura 10 - Conexão e Transmissão de Dados entre Cliente e Servidor - Fonte: Recursos Próprios - Disponível em: <cliente.c>

É possível observar que existe um tratamento para a palavra exit, quando acionada no código do cliente encerra o socket e finaliza a conexão de forma mais intuitiva.

2.3 - Mensagens de Erro

Durante a execução do código, alguns erros podem ocorrer, abaixo estão listados os códigos de erros esperados durante o funcionamento.

Código	Significado
ES01	Erro no código de execução: Deve-se utilizar: “./servidor <numeroPorta>”.
ES02	Erro no número de porta, deve ser maior que 5000
ES03	O Socket não foi criado corretamente.
ES04	Função bind não funcionou como esperado.
ES05	Função listen não funcionou como esperado.
ES06	Capacidade máxima de clientes atingida.
ES07	Erro na thread do cliente.
ES08	Os dados não foram recebidos corretamente.

Tabela 2 - Código de erros do Servidor - Fonte: Recursos Próprios

Código	Significado
EC01	Erro no código de execução: Deve-se utilizar: “./cliente <ip> <numeroPorta>”.
EC02	Erro no número de porta, deve ser maior que 5000.
EC03	O ip inserido é inválido. Use: “localhost” ou “127.0.0.1”
EC04	O socket não foi criado da maneira correta.
EC05	Função connect não funcionou da maneira correta.
EC06	Capacidade máxima de clientes atingida.

Tabela 3 - Código de erros do Cliente - Fonte: Recursos Próprios

2.4 - Makefile

Visando facilitar a compilação do código foi criado um Makefile, que consiste em apenas poucas palavras executar o mesmo comando mas de forma mais simplificada.

```
$(CC) = gcc

build:
    $(CC) -o servidor servidor.c -Wall -Wextra -pedantic -lpthread
    $(CC) -o cliente cliente.c -Wall -Wextra -pedantic -lpthread

server:
    $(CC) -o servidor servidor.c -Wall -Wextra -pedantic -lpthread

cliente:
    $(CC) -o cliente cliente.c -Wall -Wextra -pedantic -lpthread

clean:
    rm -rf *.txt servidor cliente
```

Figura 11 - Makefile - Fonte: Recursos Próprios - Disponível em: <makefile>

3 - Retransmissão de Mensagens

Elaborar sobre o envio das funções N, S, L e retorno das mesmas

Durante a conexão entre o servidor e seus clientes algumas mensagens podem ser enviadas, como abordado anteriormente no item 2.1.

O Cliente é responsável por transmitir estas mensagens para o servidor, como visto na imagem abaixo

```
printf("> ");
fflush(stdout);
fgets(buffer, BUFFER_SIZE, stdin);
fflush(stdin);
buffer[strlen(buffer)-1] = '\0';

if(strcmp(buffer, "exit") == 0)
{
    close(serverSocket);
    exit(-1);
}

send(serverSocket, buffer, strlen(buffer), 0);
```

Figura 12 - Envio de mensagens pelo Cliente - Fonte: Recursos Próprios - Disponível em: <cliente.c>

As mensagens são enviadas no uso da função “send” através do parâmetro “serverSocket”, que recebe o tratamento através das funções “main” e “clientThread” do código “servidor.c”, onde cada tipo de mensagem - N, S, L - recebe um tratamento específico devido a sua funcionalidade e também suas respectivas impressões. É enviado de volta para o cliente um código pré-estabelecido nas requisições do trabalho disponibilizado pelo professor, utilizando novamente a função “send”, porém com o uso do parâmetro “clientSocket”.

```
int operationResult;
char *tempResult = malloc(sizeof(char)*BUFFER_SIZE);
switch (messageType)
{
case 'N':
    operationResult = addRegister(username, filename);
    snprintf(sendBuffer, BUFFER_SIZE, "%c %d\n", messageType, operationResult);
    send(clientSocket, sendBuffer, strlen(sendBuffer), 0);
    break;

case 'L':
    tempResult = verifyRegister(username);
    snprintf(sendBuffer, BUFFER_SIZE, "%s\n", tempResult);
    send(clientSocket, sendBuffer, strlen(sendBuffer), 0);
    break;

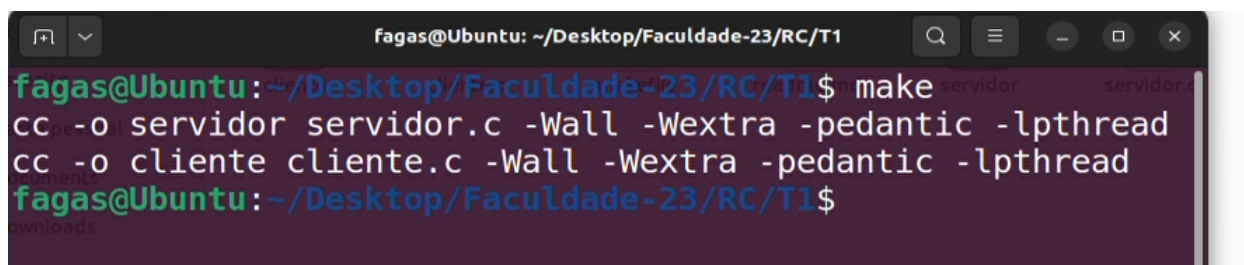
case 'S':
    operationResult = messageReceive(username, filename, paragraphContent, paragraphPosition, paragraphNumber);
    snprintf(sendBuffer, BUFFER_SIZE, "%c %d", messageType, operationResult);
    send(clientSocket, sendBuffer, strlen(sendBuffer), 0);
    break;

default:
    break;
}
```

Figura 13 - Envio de mensagens pelo Servidor - Fonte: Recursos Próprios - Disponível em <servidor.c>

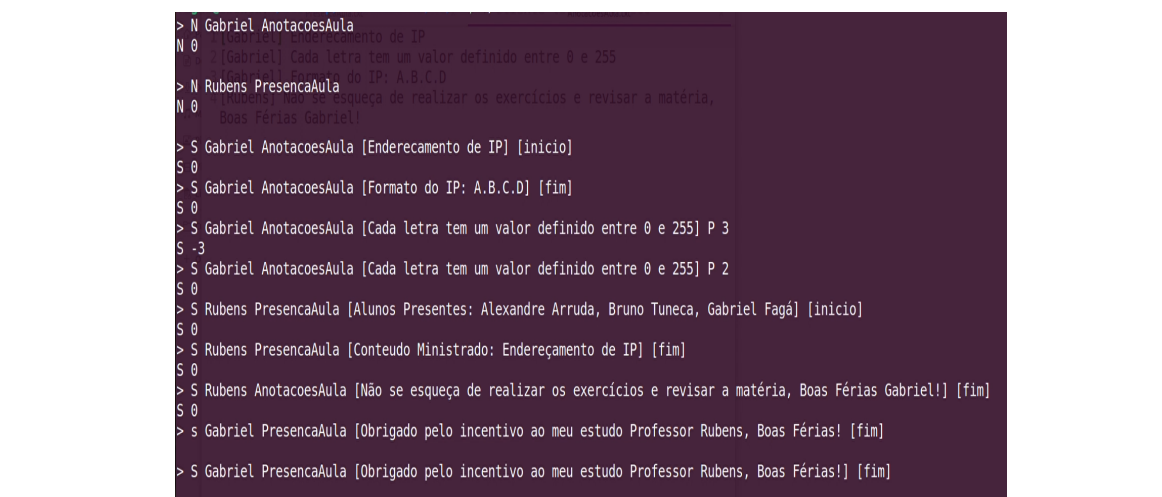
4 - Testes

Como alternativa para mostrar o funcionamento dos códigos de modo dinâmico foi gravado um vídeo mostrando as principais funcionalidades disponíveis e publicado no site “YouTube” que pode ser acessado somente através do link: <<https://youtu.be/pvZFEQZcuFY>>, porém abaixo também estarão relacionadas as capturas de tela mostrando os instantes de testes.



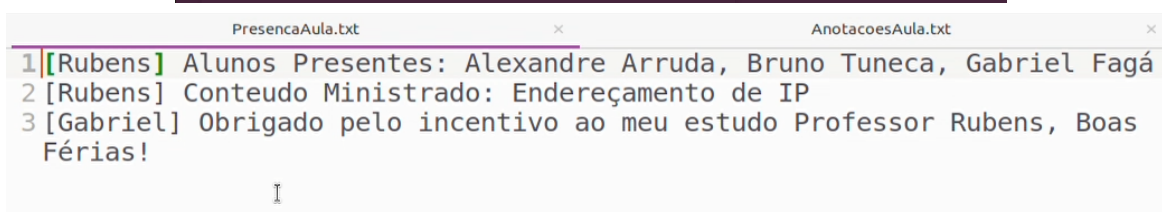
```
fagas@Ubuntu: ~/Desktop/Faculdade-23/RC/T1$ make servidor
cc -o servidor servidor.c -Wall -Wextra -pedantic -lpthread
fagas@Ubuntu: ~/Desktop/Faculdade-23/RC/T1$
```

Figura 14 - Make funcionando sem Erros/Warnings

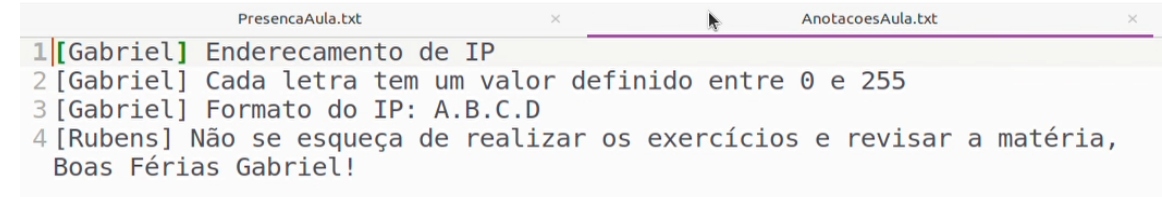


```
> N Gabriel AnotacoesAula
N 0 [Gabriel] Endereçamento de IP
> N Rubens PresencaAula
N 0 [Rubens] Não se esqueça de realizar os exercícios e revisar a matéria,
Boas Férias Gabriel!

> S Gabriel AnotacoesAula [Endereçamento de IP] [inicio]
S 0
> S Gabriel AnotacoesAula [Formato do IP: A.B.C.D] [fim]
S 0
> S Gabriel AnotacoesAula [Cada letra tem um valor definido entre 0 e 255] P 3
S -3
> S Gabriel AnotacoesAula [Cada letra tem um valor definido entre 0 e 255] P 2
S 0
> S Rubens PresencaAula [Alunos Presentes: Alexandre Arruda, Bruno Tuneca, Gabriel Fagá] [inicio]
S 0
> S Rubens PresencaAula [Conteúdo Ministrado: Endereçamento de IP] [fim]
S 0
> S Rubens AnotacoesAula [Não se esqueça de realizar os exercícios e revisar a matéria, Boas Férias Gabriel!] [fim]
S 0
> S Gabriel PresencaAula [Obrigado pelo incentivo ao meu estudo Professor Rubens, Boas Férias!] [fim]
> S Gabriel PresencaAula [Obrigado pelo incentivo ao meu estudo Professor Rubens, Boas Férias!] [fim]
```



```
1|[Rubens] Alunos Presentes: Alexandre Arruda, Bruno Tuneca, Gabriel Fagá
2|[Rubens] Conteúdo Ministrado: Endereçamento de IP
3|[Gabriel] Obrigado pelo incentivo ao meu estudo Professor Rubens, Boas Férias!
```



```
1|[Gabriel] Endereçamento de IP
2|[Gabriel] Cada letra tem um valor definido entre 0 e 255
3|[Gabriel] Formato do IP: A.B.C.D
4|[Rubens] Não se esqueça de realizar os exercícios e revisar a matéria, Boas Férias Gabriel!
```

Figuras 15, 16 e 17 - Criando Usuários e Populando os Arquivos

```
fagas@Ubuntu: ~/Desktop/Faculdade-23/RC/T1$ ./cliente localhost 9000
> S Rubens AnotacoesAula [Quase que você esquece de implementar o S né?] [fim]
S 0
> S Gabriel AnotacoesAula [Que isso professor, foi só pra ver se o senhor estava atento ao vídeo e aos prints] [fim]
S 0
> S Rubens AnotacoesAula [Aham, com certeza] [fim]
S 0
> S Rubens AnotacoesAula [Tanta preocupação na minha falta de atenção que não notou que o S foi mostrado, mas não o L] [fim]
S 0
> S Gabriel AnotacoesAula [Que nada, foi só pra mostrar a quantidade de parágrafos maior que poucas linhas] [fim]
S 0
> L Gabriel
L 0
AnotacoesAula.txt 5 Gabriel

> L Rubens
L 0
PresencaAula.txt 0 Rubens
```

Figura 18 - Listando os Arquivos de Cada Usuário

5 - Conclusão

Esse trabalho realizou uma implementação parcial de um aplicativo WebDocs, ficando indisponível somente a funcionalidade de visualização do arquivo, representada pela “chave R”. Com isso foi possível a obtenção de conhecimentos sobre sockets, conexão entre cliente e servidor, tratamento de arquivos, envio e recebimento de mensagens em uma rede, além da criação de arquivo makefile.

6 - Referências

Criando um Makefile, IBM, 2021, Disponível em:

<<https://www.ibm.com/docs/pt-br/developer-for-zos/9.1.1?topic=started-creating-makefile>>

Acesso em: 13 de julho de 2023.

COMER, D. E. Interligação em Redes TCP/IP: Princípios, Protocolos e Arquitetura. Campus, 2006. Vols. 1 e 2.

KUROSE, J.; ROSS, K. W. Redes de Computadores e a Internet: Uma abordagem Top-Down. Pearson, 2012. 6ª Edição.