



Redes de computadores A
Campinas, 26 de março de 2019

3ª Atividade

NOME:

Ettore Biazon Baccan

Mateus Henrique Zorzi

Matheus Martins Pupo

Murilo Martos Mendonça

Victor Hugo do Nascimento

RA:

16000465

16100661

16145559

16063497

16100588

Introdução

Nessa atividade foi implementada uma aplicação TCP mais próxima do uso real, com um servidor ativo e vários clientes se conectando de maneira simultânea.

Para cada cliente que se conectasse ao servidor, deveria ser criado um novo filho para atendê-lo, utilizando o comando “fork”, dessa maneira, o programa como um todo seria duplicado, estruturas e variáveis inclusive. Por outro lado, como é uma cópia, após o(s) filhos ser(em) executados, o que ocorre em um, não acontece no outro.

Passo a passo

Para resolver o problema de alterar as mensagens em um dos filhos e não ocorrer em outro, tratamos as mensagens salvas como uma região de memória compartilhada e o acesso foi controlado por semáforo, para garantir que apenas um cliente acessará as mensagens em um dado momento.

Código - Servidor

```
typedef struct
{
    char usuarios[10][20];
    char mensagens[10][80];
    int indice;
} dados_usuarios;
```

Para tal, criamos uma estrutura contendo os usuários e mensagens salvos.

```
if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0)
{
    perror("Socket()");
    exit(2);
}
```

Criamos o socket do servidor.

```
if (bind(s, (struct sockaddr *)&server, sizeof(server)) < 0)
{
    perror("Bind()");
    exit(3);
}
```

O conectamos a porta recebida pelo ARGV.

```
if (listen(s, 1) != 0)
{
    perror("Listen()");
    exit(4);
}
printf("\n...SERVIDOR INICIADO...\n");
```

Em seguida esperamos por conexões.

```
if ((ns = accept(s, (struct sockaddr *)&client, &namelen)) == -1)
{
    perror("Accept()");
    exit(5);
}

if ((pid = fork()) == 0)
{
```

Aceitamos a conexão e já usamos o fork para criar um filho que atenderá o cliente.

```
if (recv(ns, recvbuf, sizeof(recvbuf), 0) == -1)
{
    perror("Recvbuf()");
    exit(6);
}
printf("\n[%d] Mensagem recebida do cliente: %s\n", fid, recvbuf);
```

Recebemos do cliente a operação desejada:

“cad” - caso queira cadastrar uma nova mensagem.

“ler” - caso queira ler as mensagens cadastradas.

“apa” - caso queira apagar alguma mensagem cadastrada.

Cadastrar

```
if (strcmp(recvbuf, "cad") == 0)
{
    // Cadastrar mensagem
    if (shared_mem_address->indice == 10)
    {
        strcpy(sendbuf, "Numero maximo de mensagens atingido!");
        if (send(ns, sendbuf, strlen(sendbuf) + 1, 0) < 0)
        {
            perror("Send()");
            exit(7);
        }
    }
    else
    {
        if (recv(ns, mensagem_inteira, sizeof(mensagem_inteira), 0) == -1)
        {
            perror("Usuariobuf()");
            exit(6);
        }
        else
        {
            printf("\n[%d] Mensagem inteira: %s\n", fid, mensagem_inteira);
            lockSemaphore(semaphore_id_A);
            strcpy(shared_mem_address->usuarios[shared_mem_address->indice], strtok(mensagem_inteira, "#"));
            strcpy(shared_mem_address->mensagens[shared_mem_address->indice], strtok('\0', "$$"));

            printf("[%d] Usuario cadastrado: %s\n", fid, shared_mem_address->usuarios[shared_mem_address->indice]);
            //strcpy(mensagens[indice], mensagembuf);
            printf("[%d] Mensagem cadastrada: %s\n", fid, shared_mem_address->mensagens[shared_mem_address->indice]);
            printf("[%d] Indice: %d\n", fid, shared_mem_address->indice);
            shared_mem_address->indice++;
            unlockSemaphore(semaphore_id_A);

            /* Envia uma mensagem ao cliente através do socket conectado */
            strcpy(sendbuf, "Mensagem cadastrada com sucesso!");
            if (send(ns, sendbuf, strlen(sendbuf) + 1, 0) < 0)
            {
                perror("Send()");
                exit(7);
            }
        }
    }
    printf("[%d] Mensagem enviada ao cliente: %s\n", fid, sendbuf);
}
```

Caso o comando seja o cadastrar, primeiro verificamos (através da variável Índice, que se encontra na estrutura da memória compartilhada) se o número de mensagens cadastradas já atingiu o limite (10).

Em caso negativo, recebemos o nome e a mensagem do cliente através do "recv".

O modelo que escolhemos para a mensagem foi o seguinte:

“NomeUsuario#Mensagem\$\$”, para que pudéssemos receber ambos em uma única mensagem e evitar erros.

Em seguida, fechamos o semáforo, para que nenhum outro processo consiga alterar a memória compartilhada.

Separamos a mensagem em duas (nome e usuário) e os salvamos. Após salvos, incrementamos o índice.

Após alterarmos a região de memória compartilhada, abrimos o semáforo, dessa maneira, outros clientes podem ser atendidos.

Respondemos ao cliente com uma mensagem, tanto em caso de sucesso, quanto em caso de fracasso.

Ler

```
if (strcmp(recvbuf, "ler") == 0)
{
    // Ler mensagem
    char qtd_msg[2];

    sprintf(qtd_msg, "%d", shared_mem_address->indice);

    strcpy(sendbuf, qtd_msg);
    printf("[%d] SENDBUF: %s\n", fid, sendbuf);
    if (send(ns, sendbuf, strlen(sendbuf) + 1, 0) < 0)
    {
        perror("Send()");
        exit(7);
    }
    lockSemaphore(semaphore_id_A);

    for (int i = 0; i < shared_mem_address->indice; i++)
    {
        sleep(1);
        memset(sendbuf, 0, sizeof(sendbuf));
        strcpy(mensagem_inteira, shared_mem_address->usuarios[i]);
        strcat(mensagem_inteira, "#");
        strcat(mensagem_inteira, shared_mem_address->mensagens[i]);
        strcat(mensagem_inteira, "$$");
        strcpy(sendbuf, mensagem_inteira);

        if (send(ns, sendbuf, strlen(sendbuf) + 1, 0) < 0)
        {
            perror("Send()");
            exit(7);
        }

        //receber msg confirmação do cliente
    }
    unlockSemaphore(semaphore_id_A);
}
```

Caso o cliente opte por ler as mensagens cadastradas, o informamos o número de mensagens cadastradas e percorremos os vetores de mensagens e de usuários os enviando em uma única mensagem usando a mesma convenção já citada acima (que será tratada no cliente, na exibição).

Antes de percorrer os vetores o semáforo é travado, dessa forma, garantimos que a região de memória compartilhada (contendo as mensagens) não será alterada enquanto as lemos.

Após percorrer todas as mensagens, reabrimos o semáforo.

Apagar

```
if (strcmp(recvbuf, "apa") == 0)
{
    // Apaga mensagem
    char nome[20];
    if (recv(ns, recvbuf, sizeof(recvbuf), 0) == -1)
    {
        perror("Recvbuf()");
        exit(6);
    }
    strcpy(nome, recvbuf);
    strcpy(sendbuf, "Usuario nao encontrado!\n");

    // MOSTRAR MENSAGENS APAGADAS

    int msg_apagadas = 0;
    char k[2];

    lockSemaphore(semaphore_id_A);

    for (int i = 0; i < shared_mem_address->indice; i++)
    {
        printf("[%d] Nome: %d\n", fid, i);
        if (strcmp(nome, shared_mem_address->usuarios[i]) == 0)
        {
            msg_apagadas++;
        }
    }
    unlockSemaphore(semaphore_id_A);
    sprintf(k, "%d", msg_apagadas);
    strcpy(sendbuf, k);

    printf("[%d] SENDBUF: %s\n", fid, sendbuf);
    if (send(ns, sendbuf, strlen(sendbuf) + 1, 0) < 0)
    {
        perror("Send()");
        exit(7);
    }
    lockSemaphore(semaphore_id_A);
}
```

Optando por apagar, recebemos o nome do usuário dono da(s) mensagem(s), fechamos o semáforo e verificamos se existe mensagem cadastrada sob esse nome, caso negativo, avisamos ao cliente que o usuário não foi encontrado.

Caso exista mensagem cadastrada sob o nome de usuário enviado, contamos a quantidade e armazenamos na variável “msg_apagadas”.

Informamos ao cliente, em seguida, quantas mensagens foram encontradas.

```
for (int i = 0; i < shared_mem_address->indice; i++)
{
    if (strcmp(nome, shared_mem_address->usuarios[i]) == 0)
    {
        sleep(1);
        memset(sendbuf, 0, sizeof(sendbuf));
        strcpy(mensagem_inteira, shared_mem_address->usuarios[i]);
        strcat(mensagem_inteira, "#");
        strcat(mensagem_inteira, shared_mem_address->mensagens[i]);
        strcat(mensagem_inteira, "$$");
        strcpy(sendbuf, mensagem_inteira);

        if (send(ns, sendbuf, strlen(sendbuf) + 1, 0) < 0)
        {
            perror("Send()");
            exit(7);
        }
    }
    //receber msg confirmação do cliente
}
```

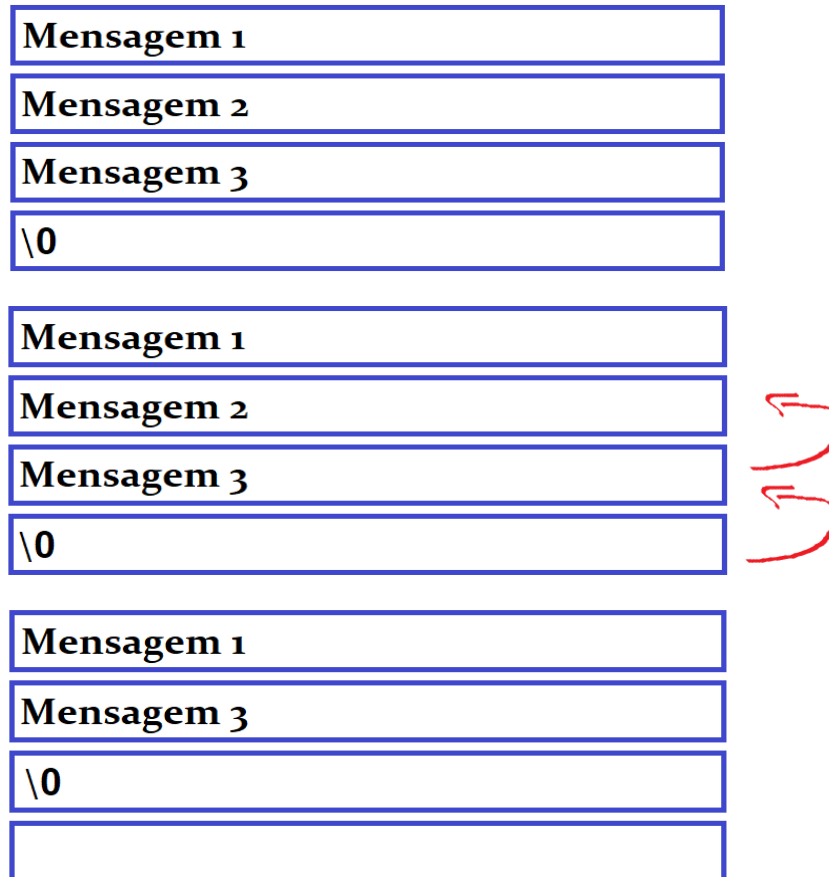
As mensagens são apagadas e enviadas ao cliente na mesma convenção já citada.


```

lockSemaphore(semaphore_id_A);
for (int i = 0; i < shared_mem_address->indice; i++)
{
    printf("[%d] Nome: %d\n", fid, i);
    if (strcmp(nome, shared_mem_address->usuarios[i]) == 0)
    {
        printf("[%d] Nome %d localizado\n", fid, i);
        for (int j = i; j < shared_mem_address->indice; j++)
        {
            printf("[%d] Usuario %d recebe usuario %d\n", fid, j, j + 1);
            strcpy(shared_mem_address->usuarios[j], shared_mem_address->usuarios[j + 1]);
            strcpy(shared_mem_address->mensagens[j], shared_mem_address->mensagens[j + 1]);
        }
        shared_mem_address->indice--;
        printf("[%d] Indice: %d\nI: %d\n", fid, shared_mem_address->indice, i);
    }
}
unlockSemaphore(semaphore_id_A);/*

```

Finalmente, reposicionamos as mensagens de maneira que os espaços vazios (sem mensagens) sempre fiquem nas últimas posições, fazendo um loop que move as mensagens da posição $i + 1$ para i a partir da mensagem que foi apagada.



Ao fim, abrimos o semáforo para liberar o uso da região de memória por outros clientes.

```
if (strcmp(recvbuf, "out") == 0)
{
    close(ns);

    /* Processo filho termina sua execução */
    printf("[%d] Processo filho terminado com sucesso.\n", fid);
    exit(0);
}
if (strcmp(recvbuf, "stp") == 0)
{
    printf("Processo servidor encerrado por:[%d]\n", fid);
    removeSharedMemory(&shared_mem_id);
    removeSemaphore(semaphore_id_A);
    exit(0);
}
```

Caso o cliente opte por sair da aplicação, o servidor é informado e o socket é fechado, em seguida, o processo filho é encerrado.

Inserimos também um comando que não é informado ao usuário final, a fim de facilitar o encerramento do servidor e a remoção da região de memória compartilhada e o semáforo.

Código - Cliente

```
if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0)
{
    perror("Socket()");
    exit(3);
}

/* Estabelece conexão com o servidor */
if (connect(s, (struct sockaddr *)&server, sizeof(server)) < 0)
{
    perror("Connect()");
    exit(4);
}
```

No início, criamos o socket do cliente e o conectamos ao servidor.

Cadastrar

```
case 1:
    // Cadastrar mensagem
    strcpy(operacao, "cad");
    if (send(s, operacao, strlen(operacao) + 1, 0) < 0)
    {
        perror("Send()");
        exit(5);
    } //informa ao servidor qual operacao sera feita

    __fpurge(stdin);
    printf("\nDigite o nome: ");
    fgets(nome, sizeof(nome), stdin);
    strtok(nome, "\n"); //tira o \n inserido pelo fgets

    __fpurge(stdin);
    printf("Digite a mensagem: ");
    fgets(mensagem, sizeof(mensagem), stdin);
    strtok(mensagem, "\n"); //tira o \n inserido pelo fgets
    printf("\nNome recebido do cliente: %s\n", nome);
    printf("\nMensagem recebida do cliente: %s\n", mensagem);

    strcpy(envio, nome);
    strcat(envio, "#");
    strcat(envio, mensagem);
    strcat(envio, "$$");
    printf("\nEnvio: %s\n", envio);

    if (send(s, envio, strlen(envio) + 1, 0) < 0)
    {
        perror("Send()");
        exit(5);
    } //envia o nome ao servidor

    if (recv(s, recvbuf, sizeof(recvbuf), 0) < 0)
    {
        perror("Recv()");
        exit(6);
    } //recebe a resposta do servidor
    printf("Servidor: %s\n", recvbuf);
```

Recebemos a operação escolhida pelo cliente, caso seja “cadastrar”, informamos o servidor e, em seguida, enviamos a mensagem e o nome do usuário no modelo adotado:

“Nome#conteudo da mensagem\$”.

Em seguida, aguardamos a resposta do servidor, que informará se o cadastro foi realizado com sucesso ou não.

Ler

```

case 2:
    // Ler mensagens
    strcpy(operacao, "ler");

    if (send(s, operacao, strlen(operacao) + 1, 0) < 0)
    {
        perror("Send()");
        exit(5);
    } //informa ao servidor qual operacao sera feita

    //salva as mensagens e ja printa
    if (recv(s, recvbuf, sizeof(recvbuf), 0) < 0)
    {
        perror("Recv()");
        exit(6);
    } //recebe o numero de mensagens cadastradas
    printf("\n%s\n", recvbuf);

    char indice_recebido[2];
    strcpy(indice_recebido, recvbuf);
    quantidade = atoi(indice_recebido);
    printf("QUANTIDADE: %d\n", quantidade);

    for (int i = 0; i < quantidade; i++)
    {
        //recebe o nome e imprime
        if (recv(s, envio, sizeof(envio), 0) < 0)
        {
            perror("Nome()");
            exit(6);
        }

        strcpy(nome, strtok(envio, "#"));
        strcpy(mensagem, strtok('\0', "$$"));

        printf("Usuario: %s", nome);
        printf("\t\tMensagem: %s\n", mensagem);
    }

```

Caso o usuário deseje ler as mensagens cadastradas, informamos ao servidor o comando escolhido.

O servidor retorna ao cliente a quantidade de mensagens cadastradas, com essa informação, utilizamos uma iteração para receber todas as mensagens (no modelo adotado e citado anteriormente) e exibí-las na tela.

Apagar

```
case 3:
    // Apagar mensagem
    strcpy(operacao, "apa");

    if (send(s, operacao, strlen(operacao) + 1, 0) < 0)
    {
        perror("Send()");
        exit(5);
    } //informa ao servidor qual operacao sera feita

    printf("Digite o nome do usuario que tera a mensagem apagada: ");

    __fpurge(stdin);
    fgets(nome, sizeof(nome), stdin);
    strtok(nome, "\n"); //tira o \n inserido pelo fgets

    if (send(s, nome, strlen(nome) + 1, 0) < 0)
    {
        perror("Send()");
        exit(5);
    } //envia o nome associado a mensagem que sera apagada

    if (recv(s, recvbuf, sizeof(recvbuf), 0) < 0)
    {
        perror("Recv()");
        exit(6);
    } //recebe a resposta do servidor
```

No caso do comando “apagar”, novamente informamos ao servidor o comando selecionado. Então, informamos o nome do usuário dono da(s) mensagem(s) que deseja-se apagar. Em sequência aguardamos a resposta do servidor.

```

char msg_apagadas[2];

strcpy(msg_apagadas, recvbuf);

printf("\nMensagens apagadas: %s\n", recvbuf);
int z;
z = atoi(msg_apagadas);
for (int i = 0; i < z; i++)
{
    //recebe o nome e imprime
    if (recv(s, envio, sizeof(envio), 0) < 0)
    {
        perror("Nome()");
        exit(6);
    }

    strcpy(nome, strtok(envio, "#"));
    strcpy(mensagem, strtok('\0', "$$"));

    printf("Usuario: %s", nome);
    printf("\t\tMensagem: %s\n", mensagem);
}
break;

```

O servidor retorna ao cliente o número de mensagens associadas ao nome de usuário enviado, esse valor é salvo em “msg_apagadas” e convertido para inteiro na variável auxiliar “z”.

Enfim recebemos o conteúdo das mensagens que foram excluídas e as exibimos na tela.

```

case 4: //sair
    printf("Obrigado por utilizar a aplicacao\n");

    strcpy(operacao, "out");

    if (send(s, operacao, strlen(operacao) + 1, 0) < 0)
    {
        perror("Send()");
        exit(5);
    } //informa ao servidor qual operacao sera feita

    break;

case 9:
    printf("Comando secreto para encerrar servidor\n");

    strcpy(operacao, "stp");

    if (send(s, operacao, strlen(operacao) + 1, 0) < 0)
    {
        perror("Send()");
        exit(5);
    } //informa ao servidor qual operacao sera feita
    break;
default:
    printf("Opcao invalida!\n");
    break;
}

```

Se o cliente optar por sair da aplicação, informamos o servidor para que o socket seja fechado.

Adicionamos, também, um comando para que o servidor seja fechado, esse comando jamais existiria em uma aplicação real. Nossa intenção aqui foi apenas facilitar o encerramento da aplicação servidor, principalmente por conta da região de memória compartilhada e semáforo.

```

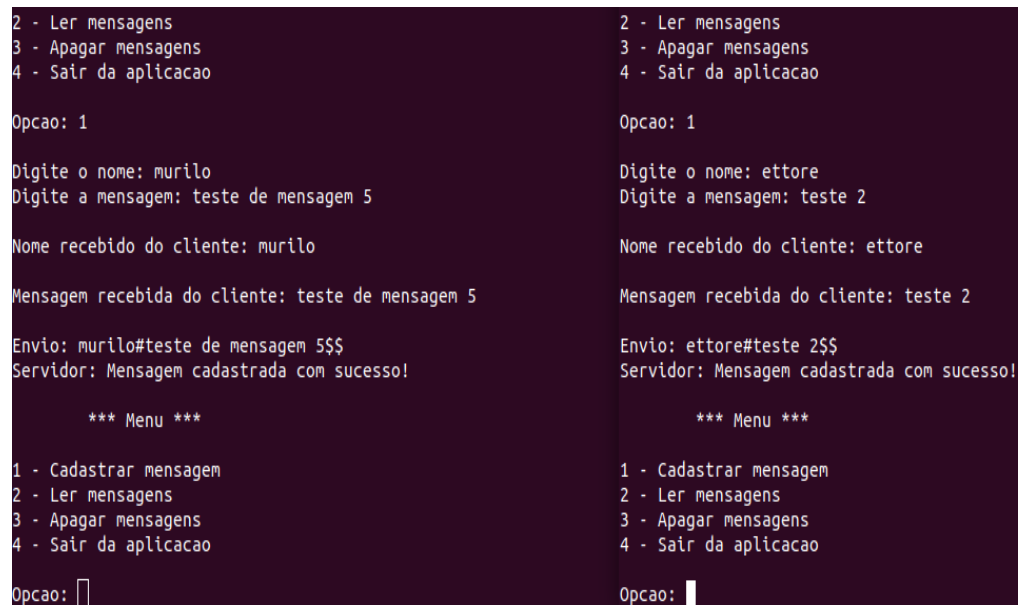
/* Fecha o socket */
close(s);

printf("Cliente terminou com sucesso.\n");
exit(0);

```


Por fim, encerramos o cliente e fechamos o socket utilizado.

Testes



```
2 - Ler mensagens
3 - Apagar mensagens
4 - Sair da aplicacao

Opcao: 1

Digite o nome: murilo
Digite a mensagem: teste de mensagem 5

Nome recebido do cliente: murilo

Mensagem recebida do cliente: teste de mensagem 5

Envio: murilo#teste de mensagem 5$$
Servidor: Mensagem cadastrada com sucesso!

*** Menu ***

1 - Cadastrar mensagem
2 - Ler mensagens
3 - Apagar mensagens
4 - Sair da aplicacao

Opcao: █

2 - Ler mensagens
3 - Apagar mensagens
4 - Sair da aplicacao

Opcao: 1

Digite o nome: etторе
Digite a mensagem: teste 2

Nome recebido do cliente: etторе

Mensagem recebida do cliente: teste 2

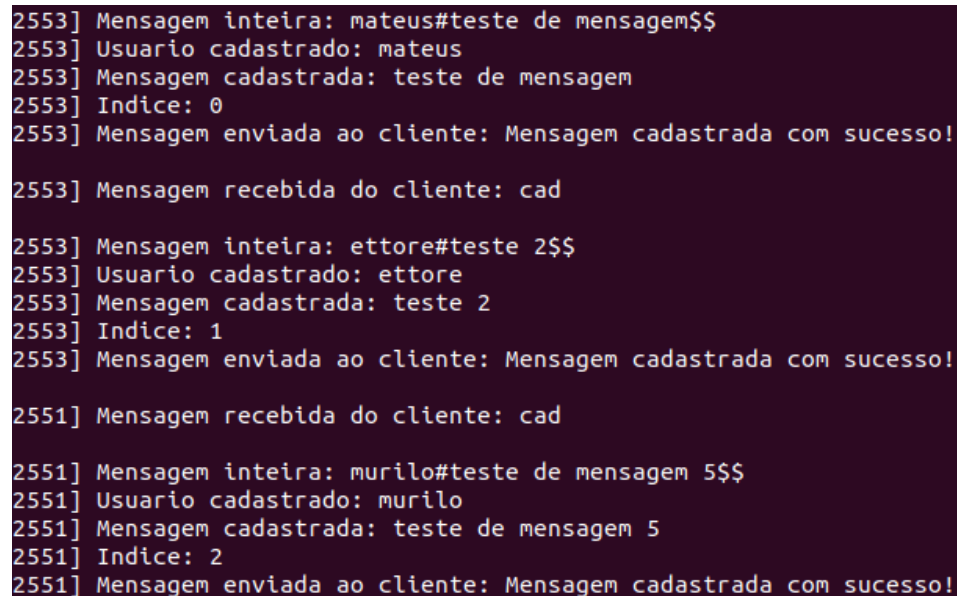
Envio: etторе#teste 2$$
Servidor: Mensagem cadastrada com sucesso!

*** Menu ***

1 - Cadastrar mensagem
2 - Ler mensagens
3 - Apagar mensagens
4 - Sair da aplicacao

Opcao: █
```

Imagem 1: Cadastramos os usuários através de dois clientes concorrentes.



```
2553] Mensagem inteira: mateus#teste de mensagem$$
2553] Usuario cadastrado: mateus
2553] Mensagem cadastrada: teste de mensagem
2553] Indice: 0
2553] Mensagem enviada ao cliente: Mensagem cadastrada com sucesso!

2553] Mensagem recebida do cliente: cad

2553] Mensagem inteira: etторе#teste 2$$
2553] Usuario cadastrado: etторе
2553] Mensagem cadastrada: teste 2
2553] Indice: 1
2553] Mensagem enviada ao cliente: Mensagem cadastrada com sucesso!

2551] Mensagem recebida do cliente: cad

2551] Mensagem inteira: murilo#teste de mensagem 5$$
2551] Usuario cadastrado: murilo
2551] Mensagem cadastrada: teste de mensagem 5
2551] Indice: 2
2551] Mensagem enviada ao cliente: Mensagem cadastrada com sucesso!
```

Imagem 2: Mensagem que o servidor exibe ao cadastrar novos usuários.



Imagem 3: As mensagens cadastradas são lidas através de dois clientes concorrentes.

```
2551] Mensagem inteira: murilo#teste de mensagem 5$$
2551] Usuario cadastrado: murilo
2551] Mensagem cadastrada: teste de mensagem 5
2551] Indice: 2
2551] Mensagem enviada ao cliente: Mensagem cadastrada com sucesso!

2551] Mensagem recebida do cliente: ler
2551] SENDBUF: 3

2553] Mensagem recebida do cliente: ler
2553] SENDBUF: 3
```

Imagem 4: Mensagem exibida pelo servidor ao receber o comando “ler”, mostrando quantas mensagens existem cadastradas.

```
*** Menu ***
1 - Cadastrar mensagem
2 - Ler mensagens
3 - Apagar mensagens
4 - Sair da aplicacao

Opcao: 1

Digite o nome: ettoe
Digite a mensagem: teste 4

Nome recebido do cliente: ettoe

Mensagem recebida do cliente: teste 4

Envio: ettoe#teste 4$$
Servidor: Mensagem cadastrada com sucesso!

*** Menu ***
1 - Cadastrar mensagem
2 - Ler mensagens
3 - Apagar mensagens
4 - Sair da aplicacao

Opcao: 2

4
QUANTIDADE: 4
Usuario: mateus      Mensagem: teste de mensagem
Usuario: ettoe      Mensagem: teste 2
Usuario: murilo      Mensagem: teste de mensagem 5
Usuario: ettoe      Mensagem: teste 4
```

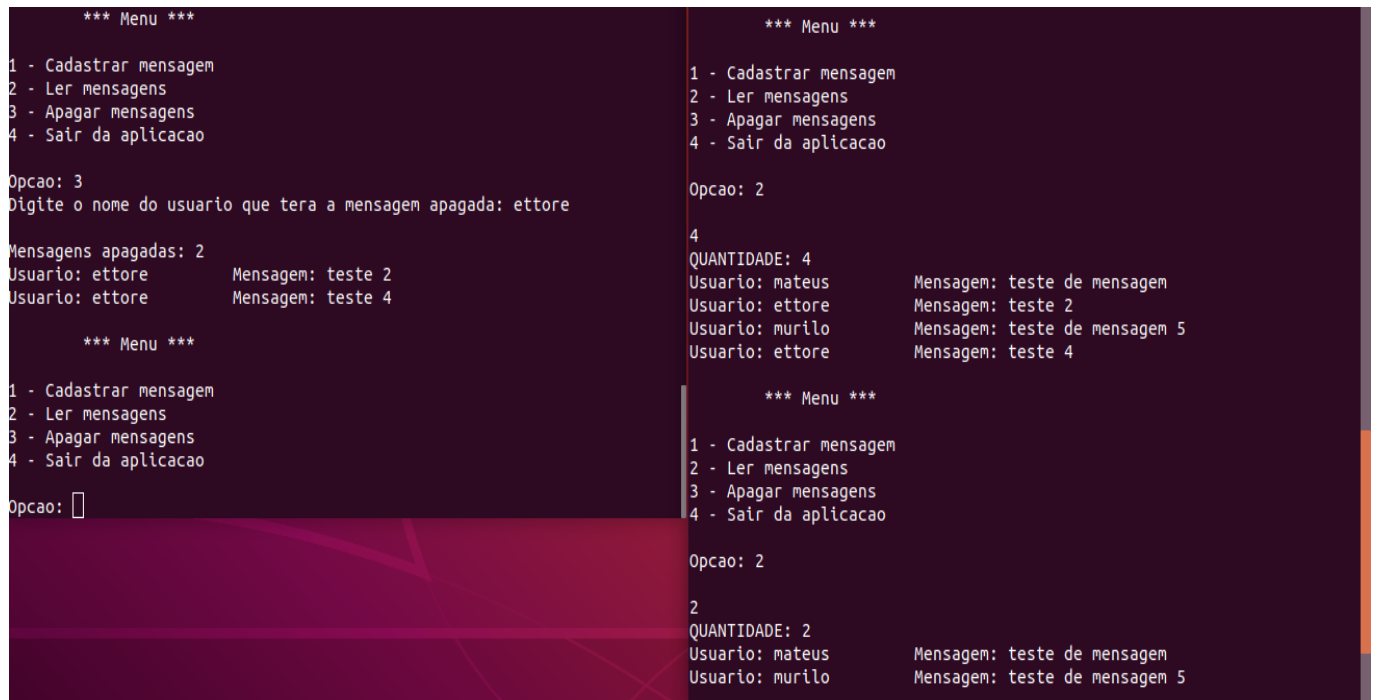
Imagem 5: Cadastramos mais um usuário com o nome “Ettore” e lemos as mensagens cadastradas no momento para realizarmos o teste de apagar com eficácia.

```
*** Menu ***
1 - Cadastrar mensagem
2 - Ler mensagens
3 - Apagar mensagens
4 - Sair da aplicacao

Opcao: 3
Digite o nome do usuario que tera a mensagem apagada: ettoe

Mensagens apagadas: 2
Usuario: ettoe      Mensagem: teste 2
Usuario: ettoe      Mensagem: teste 4
```

Imagem 6: Utilizamos o comando de apagar mensagens escolhendo o usuário “Ettore”.



```
*** Menu ***
1 - Cadastrar mensagem
2 - Ler mensagens
3 - Apagar mensagens
4 - Sair da aplicacao

Opcao: 3
Digite o nome do usuario que tera a mensagem apagada: etторе

Mensagens apagadas: 2
Usuario: etторе      Mensagem: teste 2
Usuario: etторе      Mensagem: teste 4

*** Menu ***
1 - Cadastrar mensagem
2 - Ler mensagens
3 - Apagar mensagens
4 - Sair da aplicacao

Opcao: 
```

```
*** Menu ***
1 - Cadastrar mensagem
2 - Ler mensagens
3 - Apagar mensagens
4 - Sair da aplicacao

Opcao: 2

4
QUANTIDADE: 4
Usuario: mateus      Mensagem: teste de mensagem
Usuario: etторе      Mensagem: teste 2
Usuario: murilo      Mensagem: teste de mensagem 5
Usuario: etторе      Mensagem: teste 4

*** Menu ***
1 - Cadastrar mensagem
2 - Ler mensagens
3 - Apagar mensagens
4 - Sair da aplicacao

Opcao: 2

2
QUANTIDADE: 2
Usuario: mateus      Mensagem: teste de mensagem
Usuario: murilo      Mensagem: teste de mensagem 5
```

Imagem 7: Após apagar as mensagens do usuário “ettore” em um dos servidores, utilizamos o comando ler mensagens em outro cliente, para provar que a operação foi realizada com sucesso.

```
2551] Mensagem recebida do cliente: apa
2551] Nome: 0
2551] Nome: 1
2551] Nome: 2
2551] Nome: 3
2551] SENDBUF: 2
2551] Nome: 0
2551] Nome: 1
2551] Nome 1 localizado
2551] Usuario 1 recebe usuario 2
2551] Usuario 2 recebe usuario 3
2551] Usuario 3 recebe usuario 4
2551] Indice: 3
: 1
2551] Nome: 2
2551] Nome 2 localizado
2551] Usuario 2 recebe usuario 3
2551] Indice: 2
: 2
```

Imagem 8: Mensagens mostradas pelo servidor durante a operação de apagar mensagens, mostrando o passo a passo.

Conclusão

Para a compreensão e aplicação do TCP como teste, a solução encontrada foi muito boa, já que garante que todos os cliente acessem as informações e vejam as alterações nas mensagens assim que elas ocorrem.

Porém, numa aplicação real, acreditamos que o uso do semáforo não seja o ideal, já que pode causar muita lentidão, caso um grande número de clientes se conecte ao mesmo tempo, o que causaria insatisfação ao usuário.

Além disso, durante os testes, percebemos que cada socket ativo ocupa espaço em memória, então é importante a preocupação com o fechamento dos sockets após o uso, para evitar o desperdício dos recursos do servidor e até que o servidor caia, caso a memória seja completamente usada.