

Projeto de Pesquisa
Fundamentos de Redes de Computadores
Prof. : Fernando William Cruz

Alunos

Brenno Oliveira Silva - 19/0025379
João Pedro Moura Oliveira - 19/0030879

1. Introdução

É incomum hoje que pessoas não utilizem mensageiros, sejam pelos SMS ou por chats utilizando a internet. A partir da utilização do protocolo TCP/IP é possível criar uma aplicação de chat por meio de um servidor e clientes, dessa forma permitindo que usuários possam se comunicar.

Neste trabalho será apresentado uma solução para a comunicação entre clientes a partir da utilização do protocolo supracitado com um servidor para a gerência das conversas e o comando telnet, permitindo a criação de salas, o repasse de mensagens e a listagem de clientes na sala atual do cliente.

2. Metodologia utilizada

Para a realização do trabalho, a dupla revisou os códigos disponibilizados pelo professor a fim de entender melhor o funcionamento das funções necessárias para o programa de chat funcionar. Posteriormente a dupla se reuniu em duas seções de programação por pares, com duração de aproximadamente três horas cada, para assim poder realizar o desenvolvimento do chat. Por último a dupla se reuniu em três chamadas de voz para a confecção do relatório final e apresentação.

3. Descrição da solução

A solução proposta pela dupla consistiu na utilização de duas *structs* e de sete funções, incluindo a main, para dessa forma trazer maior organização do código. A primeira delas (Figura 1), a cliente, define os dados necessários para a identificação do próprio, tendo assim um inteiro para representar o *descriptor*, uma string para armazenar o nome escolhido e por último uma variável inteira para identificar se um cliente está ou não ativo dentro de uma sala.

FIGURAS 1 e 2: *Struct* dos clientes e das salas



Fonte: Autor.

A segunda *struct* (Figura 2) delimita informações referentes a uma sala, tendo um *fd_set* para identificação da sala, as variáveis inteiras: limite, quantidade e ativo, cada uma representando respectivamente a quantidade máxima de pessoas, a quantidade atual de pessoas e se a sala está ou não ativada. Por último temos um vetor de cliente para armazenar os usuários de uma sala.

Partindo para as funções temos a função *main*, responsável inicialmente por configurar informações do servidor, como por exemplo definir o *descriptor* e realizar o *bind* e o *listen*. Em seguida é necessário inicializar os métodos de *select* para direcionamentos de fluxos e criação de grupos na aplicação, conforme demonstrado na figura 3.

FIGURA 3: Inicializações realizadas no programa.



Fonte: Autor.

Com isso iniciado e já estabelecida a conexão entre cliente e servidor, são utilizadas diversas funções `recv` para receber dados sobre o nome do cliente, o número da sala em que deseja entrar, caso o número digitado for “-1” será requisitado um limite de pessoas e em seguida a sala será criada, já inserindo-o. Toda essa programação pode ser vista na figura 4 a seguir.

FIGURA 4: Fluxo de conexão de um novo `socket` no servidor.

```
1 newfd = accept(sd, (struct sockaddr *)&remoteaddr, &addrlen);
2 FD_SET(newfd, &master);
3
4 // Recebe nome do usuario e sala que quer entrar
5 int limite, tam_nome;
6 char nome[256];
7 tam_nome = recv(newfd, nome, 256, 0);
8 tam_nome -= 2;
9 recv(newfd, buf, 256, 0);
10 sala = atoi(buf);
11
12 // Se a sala for -1, cria uma nova com o limite informado
13 if (sala == -1) {
14     recv(newfd, buf, 256, 0);
15     limite = atoi(buf);
16     sala = cria_sala(limite);
17 }
18
19 // De qualquer forma insere ele na sala nova ou existente
20 inserir_na_sala(newfd, sala, nome, tam_nome);
```

Fonte: Autor.

Com os clientes conectados, já se torna possível a utilização do *chat* via terminal. Para tanto, o cliente precisa apenas digitar sua mensagem ou comando, que a mesma será processada e divulgada para o grupo correto em que o descritor se encontra conectado. Nesse segundo fluxo, também é necessário identificar em que sala o cliente se encontra para processar suas mensagens, comandos e tratar saídas inesperadas da aplicação, sendo esse código demonstrado na figura 5, 6 e 7.

FIGURA 5: Comandos de recebimento de mensagens.

```
1 // Se não for o descritor do socket, cria um buffer, recebe a mensagem
2 // e a retransmite por todos os sockets conectados
3 memset(&buf, 0, sizeof(buf));
4 nbytes = recv(i, buf, sizeof(buf), 0);
```

Fonte: Autor.

FIGURA 6: Comandos de localização da sala e tratamento de desconexões.

```
1 // Encontra a sala que o descritor do socket se encontra
2 int sala_id;
3 for (sala_id = 0; sala_id < MAX_SALAS; sala_id++)
4     if (FD_ISSET(i, &salas[sala_id].sala_fd))
5         break;
6
7 // Desconexão forçada
8 if (nbytes == 0) {
9     printf("Desconectando forçadamente o descritor %d\n", i);
10    sair_da_sala(i, sala_id);
11 }
```

Fonte: Autor.

FIGURA 7: Transmissão da mensagem ou execução de comandos.

```
1 // Caso o primeiro caracter da mensagem seja uma / executa comando
2 if (buf[0] == '/')
3     executa_comando(i, sala_id);
4 // Caso não, encaminha a mensagem na sala
5 else
6     envia_msg(i, sd, sala_id);
```

Fonte: Autor.

Para o envio de mensagens foi utilizado a mesma ideia de uma conexão *broadcast*, onde um cliente envia uma mensagem para todos da sua mesma rede. Entretanto, a sua diferença com o código apresentado na figura 8 é que nesse a distribuição da mensagem ocorre para todos os descritores dentro do mesmo cesto ou grupo de um *select*. Para tanto, previamente deve-se descobrir o id do cliente que está transmitindo a mensagem através de seu *socket descriptor*.

FIGURA 8: Transmissão da mensagem ou execução de comandos.

```

1 void
2 envia_msg (int sd, int server_sd, int sala_id) {
3     int cliente_id;
4     for (cliente_id = 0; cliente_id < salas[sala_id].limite; cliente_id++)
5         if (salas[sala_id].clientes[cliente_id].cliente_sd == sd)
6             break;
7
8     printf("Enviando mensagem do file descriptor %d na sala %d\n", sd, sala_id);
9     // Para cada file descriptor
10    for (int j = 0; j <= fdmax; j++)
11        // checa se ele esta no cesto do master
12        if (FD_ISSET(j, &salas[sala_id].sala_fd))
13            // e checa se o valor nao e o descritor de si mesmo
14            if (j != sd && j != server_sd) {
15                // por fim envia a mensagem para aquele socket descriptor
16                char mensagem[500] = "[";
17                strcat(mensagem, salas[sala_id].clientes[cliente_id].nome);
18                strcat(mensagem, "] => ");
19                strcat(mensagem, buf);
20                send(j, mensagem, 500, 0);
21            }
22 }

```

Fonte: Autor.

Caso o primeiro caracter da frase for uma barra ("/"), o servidor automaticamente redirecionará para o fluxo de execução de comandos. A API implementada possui três comandos disponíveis, sendo eles: o de sair da sala (/sair), o de listar participantes da sala (/listar) e o de trocar de sala (/trocar_sala). É importante ressaltar que todas as ações de um cliente na sala são mostradas no terminal do servidor como uma forma de *log*.

Após compilar o código, é preciso rodá-lo informando um ip e porta desejados, exemplificado abaixo:

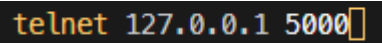
FIGURA 9: Execução do código.

```
./server 127.0.0.1 5000
```

Fonte: Autor.

Para conectar um cliente é preciso utilizar o comando telnet (ou netcat) da seguinte forma, sendo o ip e a porta informados apenas de exemplo:

FIGURA 10: Conexão ao servidor.



```
telnet 127.0.0.1 5000
```

Fonte: Autor.

Todo o código da aplicação desenvolvida pode ser acessado na plataforma do Github no seguinte link: <https://github.com/Joao-Moura/FRC-FGA>.

4. Conclusão

4.1 - Resultado do projeto

Com a solução desenvolvida se torna possível que usuários consigam se conectar a salas de *chat* e realizar conversas, com a identificação de cada um utilizando *sockets*. Também é possível utilizar diversos comandos, que facilitam e torna a solução bem mais interessante para uso. É interessante ressaltar que, a implementação de *chats* de voz e vídeos não foi feita, bem como, as validações das informações passadas também não foram concluídas, sendo essas as principais limitações da aplicação.

4.2 - Considerações dos membros

Brenno Oliveira Silva: O trabalho proporcionou um aprendizado no funcionamento de servidores para assim gerenciar a troca de informações. Acredito que uma nota 8 seria condizente com minha participação

João Pedro Moura Oliveira: O projeto serviu como uma boa fonte de conhecimentos sobre como os chats, querendo ou não, são implementados. Além disso, foi possível adquirir conhecimentos sobre a API do *select* muito interessante e versátil para esses tipos de aplicações. Assim como ressaltado anteriormente na metodologia, participei ativamente em todas as etapas do projeto em conjunto com o Brenno e acredito que minha nota deva ser 9.