Universidade de Brasília Faculdade UnB Gama

Projeto de Pesquisa

Criando ambientes virtuais de conversação com uso system call select()

Fundamentos de Redes de Computadores
Professor :Fernando Willian Cruz

Alunos:

Jonathan Jorge Barbosa Oliveira – 18/0103580
Júlia Farias Sousa – 18/0103792
Lucas Lima Ferraz – 18/0105345

1. Introdução

A comunicação hoje é feita majoritariamente de forma virtual, como visto em (ASCOM, 2021). Seja por *Whatsapp, Telegram* ou SMS o mundo inteiro está ligado por intermédio de conexões. Uma das formas de realizar essas conversas em uma Rede de computadores é pelo *System Call Select*.

A chamada *select()* monitora a atividade em um conjunto de sockets procurando por sockets prontos para leitura, gravação ou com uma condição de exceção pendente.
Exemplo:

```
#include <manifest.h>
#include <socket.h>
#include <bsdtypes.h>
#include <bsdtime.h>

int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds,
struct timeval *timeout)
```

Fonte: IBM

Este trabalho tem como objetivo exemplificar de forma funcional a implementação desse sistema utilizando um protocolo supracitados com um servidor para gerenciar conversas em um conjunto de salas.

2. Metodologia

Foi feita uma pesquisa na base de códigos disponibilizados pelo professor e na internet para entender como se comportam as funções. Depois foi feita uma programação em pares durante 3 reuniões entre os integrantes da equipe. Após essas 3 reuniões a equipe se reuniu uma última vez para confeccionar a apresentação do trabalho e o relatório dele.

3. Solução

A solução está disponível no GitHub no seguinte endereço:

https://github.com/mibasFerraz/redes-system-call-select

Primeiramente, no arquivo de servidor, utilizamos duas *structs*:

```c
typedef struct
{
    int client_sock;
    char name[256];
    int active;
} Client;
```

Fonte: Autor

A figura acima corresponde a *struct* de clientes, onde encontramos os dados referentes a identificação dele, tendo seu *descriptor* definido como *cliente_sock*, o name, para guardar seu nome e a variável active para definir se ele estará ativo ou não dentro da sala.

```c
typedef struct
{
    fd_set room_fd;
    int limit;
    int quantity;
    int active;
    char name[100];
    Client *clients;
} Room;
```

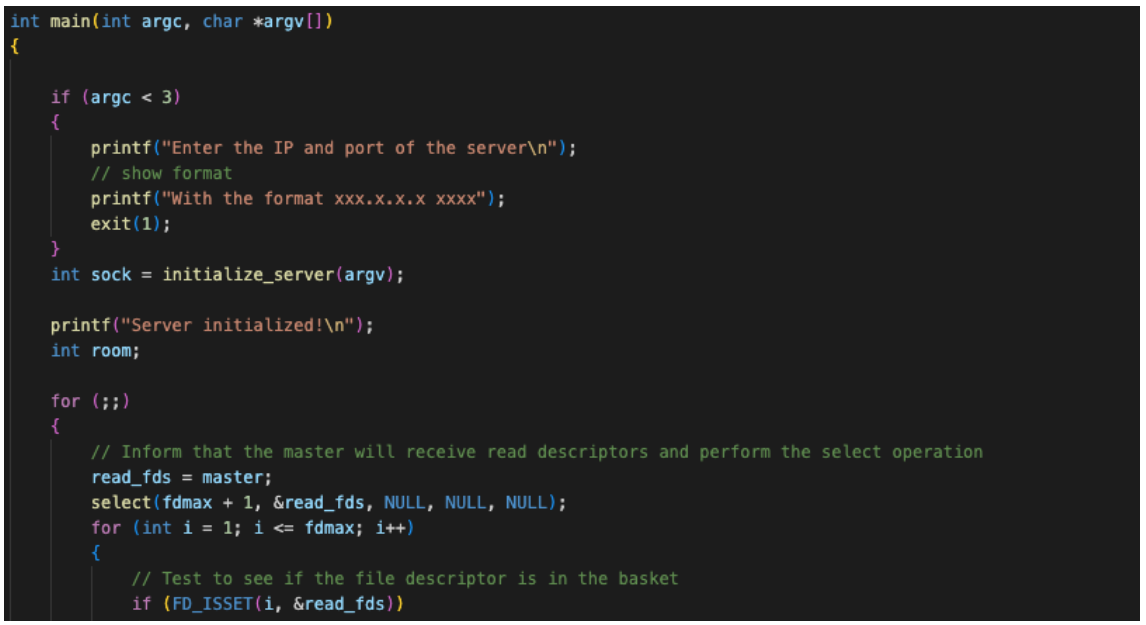
Fonte: Autor

A figura acima corresponde a *struct* de sala, sendo o *room_fd* responsável pela identificação da sala e as limit, *quantity* e *active* representam a quantidade máxima de pessoas da sala, a quantidade atual e se ela está ou não ativa. No *name* temos o " apelido " da sala.

```c
// Socket configuration
int sock = socket(AF_INET, SOCK_STREAM, 0);
setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int));

myaddr.sin_family = AF_INET;
myaddr.sin_addr.s_addr = inet_addr(argv[1]);
myaddr.sin_port = htons(atoi(argv[2]));
memset(&(myaddr.sin_zero), 0, 8);
bind(sock, (struct sockaddr *)&myaddr, sizeof(myaddr));
listen(sock, 10);

// Adding descriptors files
FD_SET(sock, &master);
FD_SET(STDIN, &master);
fdmax = sock;
addrlen = sizeof(remoteaddr);
return sock;
```

Fonte: Autor

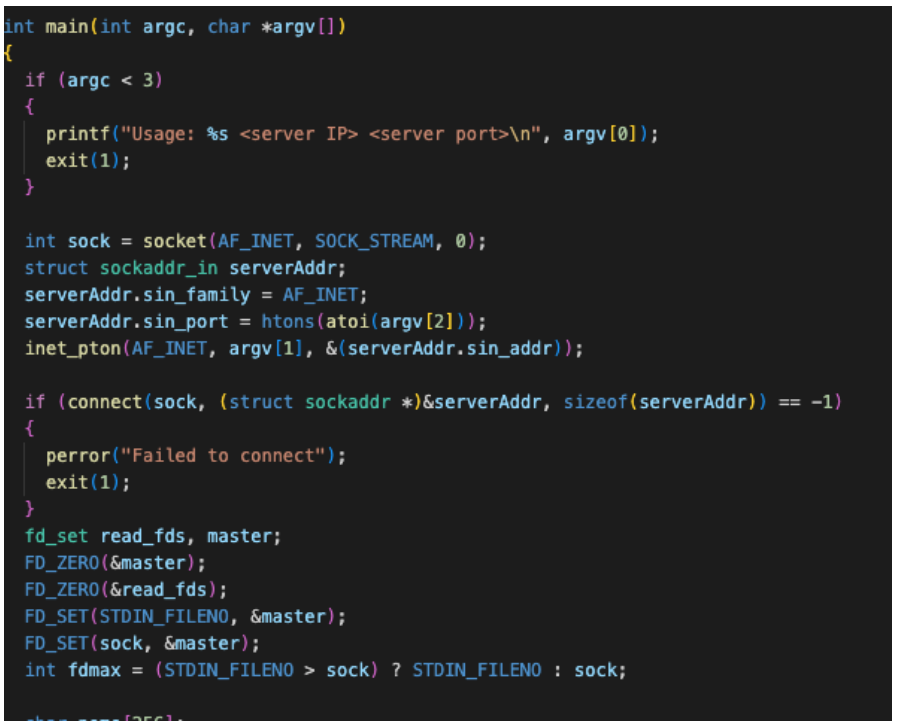
Antes de criarmos a *main*, já configuramos o socket e os métodos de *select* para direcionar os fluxos de criação de grupos. Como podemos ver na imagem acima.



```
int main(int argc, char *argv[])
{

    if (argc < 3)
    {
        printf("Enter the IP and port of the server\n");
        // show format
        printf("With the format xxx.x.x.x xxxx");
        exit(1);
    }
    int sock = initialize_server(argv);

    printf("Server initialized!\n");
    int room;

    for (;;)
    {
        // Inform that the master will receive read descriptors and perform the select operation
        read_fds = master;
        select(fdmax + 1, &read_fds, NULL, NULL, NULL);
        for (int i = 1; i <= fdmax; i++)
        {
            // Test to see if the file descriptor is in the basket
            if (FD_ISSET(i, &read_fds))
```

Fonte: Autor

Na imagem acima temos a *main* do nosso servidor, onde fazemos as verificações e criamos loopings para que os fluxos desejados ocorram da forma que queremos.



```
int main(int argc, char *argv[])
{
  if (argc < 3)
  {
    printf("Usage: %s <server IP> <server port>\n", argv[0]);
    exit(1);
  }

  int sock = socket(AF_INET, SOCK_STREAM, 0);
  struct sockaddr_in serverAddr;
  serverAddr.sin_family = AF_INET;
  serverAddr.sin_port = htons(atoi(argv[2]));
  inet_pton(AF_INET, argv[1], &(serverAddr.sin_addr));

  if (connect(sock, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) == -1)
  {
    perror("Failed to connect");
    exit(1);
  }
  fd_set read_fds, master;
  FD_ZERO(&master);
  FD_ZERO(&read_fds);
  FD_SET(STDIN_FILENO, &master);
  FD_SET(sock, &master);
  int fdmax = (STDIN_FILENO > sock) ? STDIN_FILENO : sock;
```

Fonte: Autor

Estabelecendo as conexões acima, cliente-servidor, e utilizando a *main,* já podemos fazer uso das demais funções da aplicação que se encontram no arquivo do cliente. Inserindo as informações do cliente como nome e número da sala conseguimos identificá-lo, depois digitamos “-1” para que o programa peça o limite de pessoas na sala e logo em seguida ela será criada, como podemos ver abaixo.

```
char nome[256];
printf("Enter your name: ");
scanf("%s^\n", nome);
nome[strcspn(nome, "\n")] = '\0';
send(sock, nome, strlen(nome), 0);

int sala_id;
printf("Enter the room ID (-1 to create a new room): ");
scanf("%d", &sala_id);
snprintf(buff, sizeof(buff), "%d", sala_id);
send(sock, buff, strlen(buff), 0);
clear_input_buffer();

if (sala_id == -1)
{

  int limit;
  char room_name[100];
  printf("Enter the room`s name: ");
  scanf("%s^\n", room_name);

  room_name[strcspn(room_name, "\n")] = '\0';
  send(sock, room_name, strlen(room_name), 0);
  clear_input_buffer();

  printf("Enter the room limit: ");
  scanf("%d", &limit);
  clear_input_buffer();

  printf("limit %d\n", limit);
```

Fonte: Autor

Para fluxo de mensagens foi utilizada uma vertente broadcast em que o cliente envia a mensagem para todos do chat. Temos que a distribuição da mensagem acontece para todos os *descriptors* dentro do mesmo cesto ou grupo de *select,* por isso deve-se saber antes o id do cliente que está transmitindo a mensagem através do socket.

3.1 Rodando a aplicação

- Compilar o código:
$ gcc server.c -o servidor
$ gcc client.c -o cliente

- Executar o código (precisa-se do endereço de IP e da porta desejada):
$ ./servidor 127.0.0.1 8000

- Conectar o cliente
$ ./cliente 127.0.0.1 8000

- Comandos do Menu (/help)
$./exit
$./show_users
$./show_rooms
$./change_room
$./clear
$./bat

4. Conclusão

4.1 Resultados

O programa final torna possível a comunicação entre os usuários nas salas de bate papo, com suas devidas identificações e possui as funções pedidas no enunciado, como listar participantes, realizar a saída de clientes, assim como o diálogo entre eles.

4.2 OAuth2.0

Durante o processo de estudo do protocolo OAuth2.0 e sua aplicação em nossa aplicação de salas de bate-papo virtuais, enfrentamos alguns desafios que nos impediram de realizar a implementação completa do código. No entanto, dedicamos um tempo significativo à compreensão dos conceitos e funcionamento do OAuth2.0, o que nos proporcionou valiosos conhecimentos sobre como esse protocolo poderia ser aplicado para melhorar a segurança e usabilidade da nossa aplicação.

Embora não tenhamos conseguido concluir a implementação do OAuth2.0 devido a outras atividades e compromissos dos membros do grupo, pudemos estudar a documentação e explorar exemplos práticos para entender os diferentes fluxos de autenticação e autorização oferecidos pelo protocolo.

Durante o estudo, compreendemos que o OAuth2.0 é amplamente utilizado na indústria para permitir que os usuários autentiquem-se em aplicativos por meio de provedores de identidade conhecidos, como Google, Facebook, Twitter, entre outros. Isso

elimina a necessidade de os usuários criarem novas credenciais para cada aplicação e simplifica o processo de login.

Além disso, aprendemos sobre os diferentes tipos de fluxos de autenticação do OAuth2.0, como o fluxo de concessão de autorização (Authorization Code Grant), o fluxo de senha do proprietário (Resource Owner Password Credentials Grant) e o fluxo implícito (Implicit Grant), entre outros. Cada fluxo possui suas próprias características e é adequado para diferentes cenários de uso.

Embora não tenhamos implementado o código de integração do OAuth2.0 em nossa aplicação, a compreensão desses conceitos nos permitiu visualizar como poderíamos melhorar a segurança e a experiência do usuário. Por exemplo, poderíamos permitir que os usuários se autenticarem utilizando suas contas de provedores de identidade conhecidos, proporcionando uma experiência de login mais rápida e segura.

Apesar de não termos atingido o objetivo de implementar o OAuth2.0 em nossa aplicação, reconhecemos o valor do conhecimento adquirido durante o estudo deste protocolo. Compreendemos como o OAuth2.0 pode ser aplicado em outros projetos futuros, fornecendo uma camada adicional de segurança e facilitando o processo de autenticação para os usuários.

Individualmente, cada membro do grupo dedicou tempo e esforço ao estudo do OAuth2.0, compartilhando conhecimentos e explorando recursos online para obter uma compreensão mais aprofundada do protocolo. Embora não tenhamos conseguido implementar o código, consideramos essa experiência valiosa em termos de aprendizado e reconhecemos a importância de equilibrar as demandas de diferentes atividades.

Em conclusão, embora não tenhamos conseguido implementar o OAuth2.0 em nossa aplicação de salas de bate-papo virtuais devido a outros compromissos, o estudo deste protocolo nos proporcionou uma compreensão sólida de seus conceitos e aplicabilidade. Estamos satisfeitos com o conhecimento adquirido e nos comprometemos a aplicar esse aprendizado em futuros projetos, considerando a integração do OAuth2.0 como uma opção para melhorar a segurança e a experiência do usuário.

4.3 Considerações Finais

Jonathan Jorge Barbosa Oliveira: durante o desenvolvimento do projeto, pude aprofundar meu conhecimento em arquitetura de aplicações de rede e compreender a importância da gestão de diálogos nesse contexto. Apesar de não termos conseguido implementar o código devido às demandas de outras atividades do grupo, dediquei-me a estudar o protocolo OAuth2.0 e suas possíveis integrações com o serviço de chat proposto. Além disso, minha contribuição foi significativa na construção do código fonte, identificando e implementando diversas funcionalidades essenciais. Também participei ativamente na elaboração do relatório e na pesquisa sobre o tema, buscando referências relevantes na web. Sinto que meu aprendizado foi enriquecido e considero uma nota justa para minha participação e envolvimento no projeto seria 9.0.

Júlia Farias Sousa: o projeto serviu para aplicar na prática tudo que vimos desde o começo do semestre de forma teórica e entender um pouco mais além de Redes de Computadores através de pesquisas na internet sobre o tema. Consegui aprender a como manipular os sockets e entender seus comportamentos e a importância de saber gerenciá-los em um projeto. A pesquisa sobre OAuth2.0 foi muito interessante, apesar de não termos conseguido implementar, vimos que criar essas conexões sem nenhum tipo de autenticação torna tudo que é falado no chat muito vulnerável. Participei tanto da criação do código fonte, principalmente no que se diz respeito ao servidor, como na produção do relatório e dos slides de apresentação, por isso creio que minha nota deveria ser 9.

Lucas Lima Ferraz: a matéria de redes é de extrema importância, tendo em vista que toda a internet é uma grande rede conectada. Entender os conceitos da matéria é entender os conceitos da vida prática e real e utilizar isso para melhorar os projetos que virem no futuro. A partir desse trabalho, aprendi a mexer com o protocolo TCP/IP, Sockets e a Select(). Participei ativamente de todo o processo de criação da aplicação no github, do relatório e dos slides. O estudo acerca do protocolo OAuth2.0 foi muito bem feito e proveitoso, já que entendi os conceitos e aplicações assim como seus benefícios e complicações de implementação. Não foi possível realizar a implementação correta, já que estamos em final de semestre e temos diversas atividades de outras disciplinas e tcc. Uma nota justa é 9.

5. Referências

- ASCOM, 2021, https://www.gov.br/mcom/pt-br/noticias/2021/abril/pesquisa-mostra-que-82-7-dos-domicilios-brasileiros-tem-acesso-a-internet

- IBM, 2021, Https://www.ibm.com/docs/en/zos/2.5.0?topic=calls-select

- https://man7.org/linux/man-pages/man2/select.2.html

- https://linuxhint.com/use-select-system-call-c/

-Oauth2
https://www.youtube.com/watch?v=YqGXbrVGUaU&list=PL1Nml43UBm6dOj4UuH-7a9e3wO6eL2SCi
https://github.com/babelouest/iddawc/tree/v1.1.8
https://oauth.net/2/

- What is the OAuth 2.0 Authorization Code Grant Type? -> link
https://developer.okta.com/blog/2018/04/10/oauth-authorization-code-grant-type

-Resource Owner Password Credentials Flow
https://cloudentity.com/developers/basics/oauth-grant-types/resource-owner-password-credentials/#:~:text=The%20Resource%20Owner%20Password%20Credentials%20grant%20(also%20known%20as%20ROPC,authorization%20server%20by%20the%20client.

- Is the OAuth 2.0 Implicit Flow Dead?
https://developer.okta.com/blog/2019/05/01/is-the-oauth-implicit-flow-dead mais em https://oauth.net/2/grant-types/implicit/
exemplos https://fusionauth.io/articles/oauth/modern-guide-to-oauth