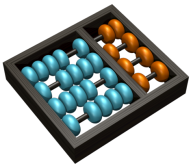
UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Nome: Mateus Shiguetochi Kimura
RA:174383



1. Como executar

Primeiramente, para compilar os arquivos foi fornecido um Makefile.
Portanto, apenas execute:

```
make compile-all
```

Para executar o servidor:

```
./myserver
```

Para executar o cliente:

```
./myclient
```

2. Introdução

O Projeto 1 da matéria MC833 do 1º semestre de 2021 teve como objetivo implementar uma arquitetura de servidor concorrente sobre o protocolo de comunicação TCP/IP.

O serviço a ser replicado é de uma plataforma de gerenciamento de usuários de uma universidade. Os perfis armazenam as seguinte informações : Email (chave identificadora), nome, sobrenome, cidade de residência, formação acadêmica, ano de formatura, habilidades e experiência profissional.

Apenas a pessoa administradora, que já está configurada e não pode ser mudada, pode realizar mudanças das informações dos perfis, ou seja, fazer operações de edição, inserção e deleção de usuários. Outras pessoas que já possuem perfil cadastrado podem ingressar na plataforma através de seus emails e realizar pesquisas de busca.

Sobre a implementação, foi utilizado uma arquitetura cliente-servidor utilizando-se protocolo TCP-IP para comunicação. A função *fork()* foi utilizada para se extrair a concorrência do servidor e assim, possibilitando a utilização do mesmo por diversos usuários simultaneamente.

3. Metodologia

Neste tópico será abordado a metodologia utilizada no projeto, exemplificando com partes do código, e caso seja necessário, justificando a implementação.

3.1. Casos de Uso

O sistema possui 4 casos de usos: Criação, alteração, busca e deleção de usuários.

3.1.1. Criação

Quem pode: Apenas o administrador.
Ação: Cadastrar um novo perfil utilizando o email como identificador;
Inputs necessários: email, nome, sobrenome, cidade de residência, formação acadêmica, ano de formatura, número de habilidades, habilidades, número de experiências profissionais, experiências profissionais.

3.1.2. Alteração

Quem pode: Apenas o administrador.
Ação: Acrescentar uma nova experiência profissional em um perfil;
Inputs necessários: email e nova experiência profissional.

3.1.3. Busca

Quem pode: Todos usuários cadastrados na plataforma
Ação: Dado uma chave de busca, o servidor irá retornar todos os usuários que correspondam a busca ou então, todas as informações de todos os usuários.
Inputs necessários: Chave de busca (Curso, habilidade, ano de formação, email).

3.1.4. Deleção

Quem pode: Apenas o administrador.
Ação: Remover um perfil a partir de seu identificador (email)
Inputs necessários: email

3.2. Armazenamento

As informações dos usuários são armazenadas em arquivos na pasta *users* com o padrão de nomenclatura *userX.txt* , onde *X* representa o Id do usuário, que mesmo que seja uma informação não requisitada pelo projeto, foi utilizada para fins de organização do armazenamento. Outra informação adicional, está na última linha de cada arquivo de usuário, que possui um número inteiro informando se o usuário é administrador (1) ou não (0).

Um exemplo de arquivo de armazenamento:

```
2                                                       # id
maria_silva@gmail.com                                   # email
Maria                                                   # First Name
Silva                                                   # Last Name
Campinas                                                # City
Ciência da Computação                                   # Graduation course
2015                                                    # Year of graduation
3                                                       # Number of skills
Análise de Dados                                        # Skill 1
Internet das Coisas                                     # Skill 2
Computação em Nuvem                                     # Skill 3
2                                                       #Number of experiences
Estágio de 1 ano na Empresa X, onde trabalhei como analista de dados # Ex 1
Trabalhei com IoT e Computação em Nuvem por 5 anos na Empresa Y      # Ex 2
0                                                       # Root flag
```

O número de usuários cadastrados fica armazenado no arquivo *numberofusers.txt*, que possui apenas um inteiro na primeira linha que representa o número de usuários cadastrados.

Outra informação que fica armazenada em arquivos são os menus de cada tipo de usuário. O administrador irá utilizar o *menuRoot.txt* localizado na pasta *menu* e os outros usuários irão utilizar o *menuUser.txt*.

3.3. Comunicação

As mensagens trocadas entre cliente e servidor são feitas nas funções :

Servidor:

```
void receiveFromClient(int connfd, char* buffer);
void sendIntToClient(int connfd, int n);
void sendMessageToClient(int connfd, char msg[MAXLINE]);
```

Cliente:

```
void sendToServer(int clientSocket, char* buffer);
void sendIntToServer(int clientSocket, int n);
void getAndSendToServer(int clientSocket);
void readFromServer(int clientSocket, char* buffer);
```

Onde o buffer é a mensagem que será enviada para o cliente/servidor na forma de char*. Caso haja dúvidas sobre determinada função, todas as funções possuem descrição no código em seu cabeçalho.

As mensagens são enviadas de forma particionada para que o tamanho máximo não seja excedido. Portanto, julguei que não seria necessário implementar a forma de leituras sequenciais do lado do cliente proposta pelo docentes. O caso de limitação do programa seria para experiências do usuário muito extensas, o envio seria limitado apenas para 1024 caracteres. Para os casos de busca, onde é possível o retorno de múltiplos usuários, cada informação de cada usuário encontrado (*email* e *nome*) é enviado em mensagens separadas.

```
// exemplo de envio para usuários encontrados para determinado curso
for(i = 0; i < numberOfUsers; i++)
   if(strcmp(users[i]->course, course) == 0){
      snprintf(buffer, MAXLINE, "\n--User %d--\nEmail: %s",i+1, \
                                                      users[i]->email);

      sendMessageToClient(connfd, buffer);
      snprintf(buffer, MAXLINE, "Name: %s %s\n", users[i]->firstName, \
                                                   users[i]->lastName);
      sendMessageToClient(connfd, buffer);
    }
```

3.4. Implementação

Todas as funções escritas possuem um comentário de cabeçário que resume sua funcionalidade e descreve suas entradas e, caso seja necessário, seu retorno.

Users

Foi implementado um módulo *users.h* que realiza procedimentos relacionados aos usuários e seus arquivos. Nele está presente a estrutura de usuários que será utilizada ao decorrer do sistema.

```
typedef struct _User {
	char *id;
	char *email;
	char *firstName;
	char *lastName;
	char *city;
	char *course;
	char *year;
	int nSkills;
	char **skills;
	int nExperiences;
	char **experiences;
	int root;
} User;
```

Cliente

O cliente cria localmente um socket TCP no padrão IPv4 e testa a criação:

```
	// socket create and verification
	clientSocket = socket(PF_INET, SOCK_STREAM, 0);
	if(clientSocket<0){
		printf("[-] Fail to create the Socket\n");
		return;
	}	printf("[+] Client Socket Created Sucessfully.\n");
```

Atribui o IP e a porta de conexão com o servidor ao socket. No caso foi utilizado a porta 8080 e o ip local da máquina, porém caso o servidor esteja em outra máquina, é preciso substituir o valor de `serverAddr.sin_addr.s_addr` para o ip da máquina onde o servidor está rodando. O cliente irá tentar a conexão com o servidor e validar se a conexão foi realizada com sucesso.

```
	memset(&serverAddr, '\0', sizeof(serverAddr));
	serverAddr.sin_family = AF_INET;
	serverAddr.sin_port = htons(PORT);
	// Change here to the server host ip
	serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
	 // connect the client socket to server socket
	if(connect(clientSocket, (struct sockaddr *) &serverAddr, \
		    sizeof(serverAddr))<0){
		perror("[-] Connection Error");
		// close(clientSocket);
		exit(EXIT_FAILURE);
	}
	printf("[+] Connected to Server.\n");
```

Caso a conexão com o servidor seja estabelecida com sucesso, o cliente terá que logar dentro do sistema informando seu email. O login é feito pela função :

```
int loginIntoServer(int clientSocket);
```

Caso o usuário exista, o server irá retornar “1” caso o usuário seja administrador, “0” caso não seja administrador e “-1” caso contrário.

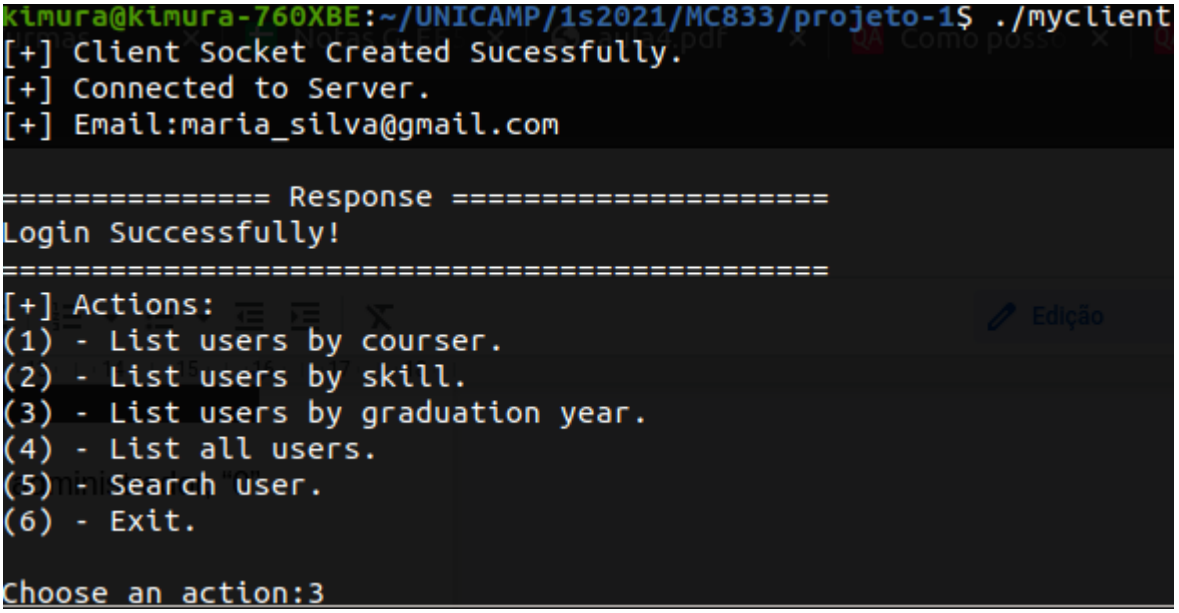
A função :

```
void startService( int clientSocket, int user);
```

será a função de interação do usuário com o servidor. O fluxo de execução será um loop infinito que possui o fluxo de execução: O cliente recebe do servidor o menu de comandos possíveis a serem executados, envia o comando escolhido pelo usuário e entra no fluxo de execução de cada comando.

```
	// loop to client iterate through commands
	while(1){
		gettingMenuFromServer(clientSocket);
		// Get action to be executed [1-8] and send to server
		// 9 - exit
		fgets(buffer, MAXLINE, stdin);
		sendToServer(clientSocket, buffer);
		command = atoi(buffer);
		executeCommand(clientSocket, user, command);
	}
```

Exemplo da tela inicial vista por um cliente não administrador que logou na plataforma com sucesso:



Comandos de busca no cliente: As buscas podem ser feitas utilizando diferentes chaves (curso, habilidade …). Quando realizada a busca, o cliente irá seguir um fluxo padrão de : primeiro ler o inteiro retornado pelo servidor, que representa a quantidade de usuários encontrados na busca, e posteriormente irá realizar a leitura dos valores de cada perfil retornado pelo servidor.

Servidor

O servidor irá criar um socket TCP no padrão IPv4 utilizando o protocolo TCP/IP e validar se o socket foi criado com sucesso.

```
   // socket create and verification
    int sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (sockfd < 0){
        printf("[-] Socket creation error!\n");
        return 1;
    }else{
        printf("[+] Socket creation success!\n");
    }
```

Atribui o endereço local do protocolo. No caso foi utilizado do tipo *INADDR_ANY* que irá receber mensagens de qualquer endereço, realiza o bind e verifica se ocorreu com sucesso.

```
    // assign IP, PORT
    address.sin_addr.s_addr = htonl(INADDR_ANY);
    address.sin_port = htons(PORT);
    address.sin_family = AF_INET;

    int address_len = sizeof(address);
    // bind
    int server_bind = bind(sockfd, (struct sockaddr *)&address,
sizeof(address));
    if (server_bind) {
        printf("[-] Server binding failed!\n");
        return 1;
    }else{
        printf("[+] Server binding success!\n");
    }
```

Caso o bind tenha sido executado com sucesso, será executado um listen no socket:

```
// listen()
    int listening = listen(sockfd, 5);
    if (listening < 0){
        printf("[-] Failed to listen!\n");
        return 1;
    }
```

Nesta etapa o servidor já estará ouvindo a porta, caso a 8080, para verificar as requisições de conexão. Aceitando quaisquer requisições.

Assim que uma requisição de conexão chega ao servidor pela porta 8080, o servidor cria um novo socket para se comunicar com o cliente e retorna seu descritor para que seja possível a conexão com o cliente pelo novo socket.

```
// accept()
        connfd = accept(sockfd, (struct sockaddr *)&address, (socklen_t
*)&address_len);
        if (connfd < 0){
            printf("[-] Error in accept!\n");
            continue;
        }
        printf("[+] New connection established\n");
```

Após a criação do descritor do novo socket, o servidor irá realizar um *fork* do processo que está a nova conexão, e fechar a conexão do processo filho, assim novas requisições de novos usuários poderão ser realizadas. No processo pai, o fluxo de navegação do usuário será iniciado.

```
// copy process and execute the parent
        if ( (pid = fork()) == 0 ) {

            close(listening); /* child closes listening socket */

            user_login = login(connfd);
            sendIntToClient(connfd, user_login);

            if(user_login >= 0)
                startService(connfd, user_login);

            close(connfd);
            exit(0); //child terminates
        }
```

Primeiramente é feito o login no servidor, que retorna a flag de usuário (1-administrador / 0- não administrador/ -1- não existe) e envia para o cliente. Caso o usuário exista, o fluxo entrará na função de interação com o usuário

```
void startService( int connfd, int user);
```

Daqui em diante, o fluxo seguirá o padrão já explicado na parte do usuário, o servidor irá enviar o menu de comandos para o cliente, irá receber do cliente qual comando executar, e entrar no fluxo de execução de cada comando.

4. Conclusão

O projeto 1, que teve como objetivo criar um sistema de gerenciamento de usuários utilizando-se de uma arquitetura cliente-servidor com servidor concorrente e protocolo de comunicação TCP/IP, abordou de forma objetiva e eficaz a forma de comunicação utilizada nesse tipo de protocolo.

A comunicação feita pelo protocolo TCP/IP necessita de um socket de conexão entre o cliente e o servidor, que configura a primeira porta de comunicação, mas faz o gerenciamento dos endereços após a conexão ser estabelecida. O cliente irá enviar as mensagens através do socket que irá ser traduzido pelo servidor e executar os comandos e retornar os resultados encontrados.

A utilização da função *fork()* também foi necessária para que o servidor consiga suportar diversas conexões simultâneas de diferentes clientes. O servidor irá criar um processo filho que irá seguir com o fluxo para aceitar novas conexões, enquanto que o processo pai irá seguir com o fluxo de cada cliente.