

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

Salas de bate-papo virtuais

Fundamentos de Redes e Computadores

Feito em Grupo

ALUNOS:

Victor Buendia Cruz de Alvim	190020601
Yan Andrade de Sena	180145363
João Vitor de Souza Durso	180123459

Objetivos do Projeto de Pesquisa

Este trabalho final possui o intuito de permitir que o aluno compreenda a arquitetura de aplicações de rede (segundo na arquitetura TCP/IP) que envolvam gerência de diálogo. Dessa forma, os alunos irão construir uma aplicação que disponibiliza salas de bate-papo virtuais, nas quais os clientes possam ingressar e interagir. Além disso, os alunos deverão se atentar a utilizar a system call select(), a mesma será explicada nos tópicos a seguir.

Descrição do Problema

A criação de salas virtuais tem como principal problema saber gerenciar uma comunicação full-duplex. Uma estação full-duplex é onde ambas estações podem transmitir e receber simultaneamente. Deve ser ressaltado também a utilização do system call select(). O select() é uma função que permite que um programa monitore vários descritores de arquivo, esperando até que um ou mais descritores de arquivo fiquem “prontos” para alguma classe de operação. Um

arquivo descritor é considerado pronto se for possível realizar uma operação de I/O correspondente (por exemplo, read(2), ou um small write(2)) sem bloqueio.

A aplicação também deverá ser responsável em conseguir:

- Criar salas virtuais com nome da sala e limite de participantes;
- Listar participantes de uma determinada sala;
- Permitir ingresso de clientes, com um identificador, em uma sala existente, de acordo com o limite admitido para a sala;
- Saída de clientes de uma sala em que estava participando;
- Diálogo entre os clientes das salas

Metodologia Utilizada

Para conseguir solucionar os problemas propostos nesse trabalho, o nosso grupo se reuniu e decidiu que cada membro iria tentar realizar o mesmo através da consulta de sites e/ou cursos. Essa decisão era importante para trabalhar uma cooperação dos membros do grupo, no qual um apoiaria o outro em sua própria jornada para configurar a rede proposta, compartilhando conhecimento.

Em seguida, decidimos que o aluno que tivesse mais progresso no trabalho proposto explicaria aos outros como a solução havia sido concebida, com o intuito de compreender as competências a serem desenvolvidas nesta atividade.

Solução do Problema

No nosso grupo, iremos utilizar a linguagem de programação python e iremos criar dois arquivos, uma para o cliente e outro para o servidor. Para rodar o projeto deveremos compilar primeiro o arquivo chat_server.py utilizando o comando: python3.10 chat_server.py

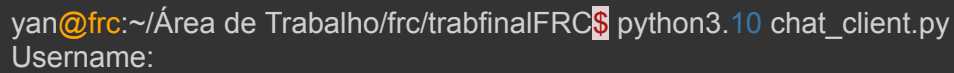
Quando compilado o arquivo ele irá mostrar as seguintes opções:

```
===== MENU =====  
/CRIAR ----> Cria uma nova sala.  
/LISTAR ----> Listar salas criadas.  
/PARTICIPANTES ----> Listar participantes de uma determinada sala.  
/SAIR ----> Encerra a aplicação.  
/LOGS ----> Exibir logs.  
Digite um comando:
```

Imagem 1: Menu para seleção de ações

Após ter selecionado a opção /CRIAR e inserir um nome para a sala e o tamanho dela a mesma será criada, em seguida devemos executar o arquivo de cliente, o arquivo se chama chat_client.py, executamos utilizando o mesmo comando para o server: python3.10 chat_client.py

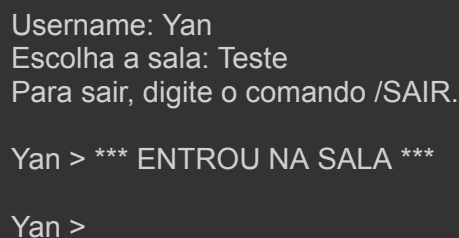
Quando o arquivo for executado ele pedirá para o usuário inserir um nome:



```
yan@frc:~/Área de Trabalho/frc/trabfinalFRC$ python3.10 chat_client.py
Username:
```

Imagem 2: Compilação de chat_client.py

Após inserido o nome o cliente irá se conectar a sala criada pelo arquivo de servidor inserindo um nome de sala já existente, se caso o cliente informar um nome de sala de forma errônea a aplicação irá aguardar até ser inserido um nome correto.



```
Username: Yan
Escolha a sala: Teste
Para sair, digite o comando /SAIR.

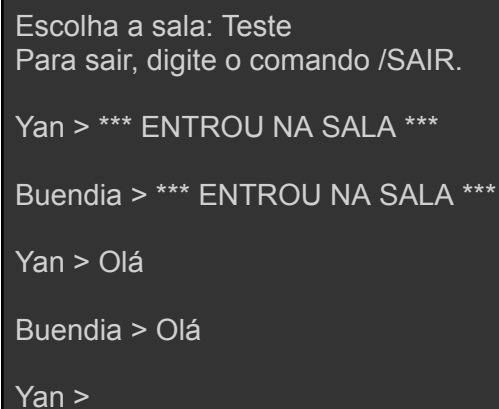
Yan > *** ENTROU NA SALA ***

Yan >
```

Imagem 3: Inserindo um nome de usuário e um nome de sala existente

A aplicação ficará disponível para os clientes enviarem mensagens entre si. Para o cliente sair da aplicação basta ele digitar a palavra /SAIR e sairá da aplicação.

A seguir mostraremos um diálogo entre dois participantes da sala virtual de nome Teste



```
Escolha a sala: Teste
Para sair, digite o comando /SAIR.

Yan > *** ENTROU NA SALA ***

Buendia > *** ENTROU NA SALA ***

Yan > Olá

Buendia > Olá

Yan >
```

Imagem 4: Sala Teste com Yan e Buendia conversando na sala (chat_client.py)

```
***** SALAS *****  
1: Teste - 2/2 usuários conectados.  
Pressione uma tecla para continuar...
```

Imagem 5: Exibindo sala com nome e tamanho (chat_server.py)

```
Username: Buendia  
Escolha a sala: Teste  
Para sair, digite o comando /SAIR.  
  
Buendia > *** ENTROU NA SALA ***  
  
Yan > Olá  
  
Buendia > Olá  
  
Buendia >
```

Imagem 6: O cliente Buendia envia a mensagem (chat_client.py)

```
Escolha a sala: Teste  
Para sair, digite o comando /SAIR.  
  
Buendia > *** ENTROU NA SALA ***  
  
Yan > Olá  
  
Buendia > Olá  
  
Buendia > *** SAIU DA SALA ***  
  
Yan >
```

Imagem 7: Buendia sai da aplicação (chat_client.py)

```
***** PARTICIPANTES DA SALA "Teste" *****  
  
1 - Yan  
Pressione uma tecla para continuar...
```

Imagem 8: Opção de listar participantes de uma sala (chat_server.py)

Agora iremos explicar o que realizamos no código, tanto do chat_server.py quanto no chat_client.py.

Chat_server.py

As primeiras linhas do código nós definimos informações básicas do nosso servidor, como o tamanho do HEADER, o IP, e a PORT:

```
HEADER_LENGTH = 10

IP = "127.0.0.1"
PORT = 1234
```

Depois definimos as opções do menu:

```
comandos = {
    "/CRIAR": "Cria uma nova sala.",
    "/LISTAR": "Listar salas criadas.",
    "/PARTICIPANTES": "Listar participantes de uma determinada sala.",
    "/ENCERRAR": "Encerrar uma sala.",
    "/SAIR": "Encerrar a aplicação.",
    "/LOGS": "Exibir logs"
}
```

Declaramos também um vetor de logs que irá armazenar os logs da aplicação

```
logs = [ ]
```

Criamos um socket com conexão TCP

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

Depois realizamos o Bind, para que o servidor informe ao sistema operacional que ele usará determinado IP e PORT que antes configuramos

```
server_socket.bind((IP, PORT))
```

O servidor agora estará ouvindo novas conexões

```
server_socket.listen()
```

Definimos uma lista de sockets para o select()

```
sockets_list = [server_socket]
```

Depois definimos uma lista de clientes conectados (socket como chave, header do usuário e nome)

```
clients = { }
```

Definimos também a listas de salas abertas contendo limite de usuários e usuários Conectados

```
rooms = [ ]
```

Agora trataremos quando o usuário é adicionado a uma sala

```
def add_user_to_room(room_name, user):
```

A sala selecionada inicializa com valor 'None', em seguida fazemos um laço for dentro do vetor de rooms que é onde todas as salas estão armazenadas

```
room_selected = None  
for room in rooms:
```

Fazemos a verificação do nome da sala inserida pelo cliente, se ela existir a sala selecionada recebe as informações da sala

```
if room['name'] == room_name:  
    room_selected = room
```

Se a sala não for existente é notificado para o cliente inserir uma sala com nome existente

```
else:  
    log_message(f'{user["username"]} tentou entrar em sala inexistente.')  
    data = {'username': 'Sistema', 'message': '404'}  
    encoded_data = json.dumps(data).encode('utf-8')  
    user['socket'].send(encoded_data)  
  
    user['socket'].shutdown(socket.SHUT_RDWR)  
    user['socket'].close()
```

Se a sala selecionada anteriormente for diferente de 'None' então o limite da sala escolhido é definido

```
if room_selected is not None:
    room_limit = room_selected["limit"]
```

Agora verificamos se um usuário pode ingressar em uma sala, se ela estiver cheia o cliente é notificado e não consegue entrar em uma sala. Senão o cliente consegue entrar na sala com êxito

```
if (len(room_selected["users"]) + 1) <= room_limit:
    room_selected["users"].append(user)
    user["room"] = room_selected
    clients[user["socket"]] = user

    notify_user_join_room(
        username_join=user["username"], room=room_selected)
    log_message(
        f'{user["username"]} entrou na sala {room_selected["name"]}.'.)

    return room_selected
else:
    log_message(
        f'{user["username"]} tentou entrar na sala \'{room_selected["name"]}\', mas seu
limite de participantes já foi atingido.')
    data = {'username': 'Sistema', 'message': '501'}
    encoded_data = json.dumps(data).encode('utf-8')
    user["socket"].send(encoded_data)

    user["socket"].shutdown(socket.SHUT_RDWR)
    user["socket"].close()
```

Agora falaremos sobre a criação de salas. Primeiro pedimos o nome da sala e o limite da mesma, em seguida geramos um id para a mesma e salvamos seus dados. Também fazemos uma validação desses dados se os dados estiverem corretos a sala é criada, senão espera o usuário inserir dados corretos.

```
def create_room():
    print("\nCriando sala...")

    try:
        room_name = input('Digite o nome da sala: ')
        room_limit = input('Digite o limite da sala: ')
        room_id = (rooms[len(rooms) - 1]["id"] + 1) if (len(rooms) > 0) else 1
        rooms.append({'id': room_id, 'name': room_name,
            'limit': int(room_limit), 'users': [ ]})
```

```

except ValueError as e:
    print("\nInsira dados válidos.")
    time.sleep(3)
    clear()
    create_room()

clear()
print(f'Sala {room_name} criada.')
log_message(
    f'Sala {room_name} criada. Atualmente, existem {len(rooms)} salas criadas.')

```

Em seguida mostraremos a parte de excluir salas virtuais. Primeiro exibimos as salas disponíveis com seu ID, depois pedimos pro usuário inserir o ID da sala a ser removida. Em seguida fazemos um for procurando a sala selecionada e removemos ela. Também verificamos se ela existe. Se o id inserido não for encontrado, notificamos ao usuário

```

def delete_room():
    show_rooms()
    room_to_be_removed = None

    id_room_to_be_removed = input('Selecione o id da sala a ser removida:')
    for room in rooms:
        if room['id'] == int(id_room_to_be_removed):
            room_to_be_removed = room
            continue

    if room_to_be_removed is not None:
        remove_users_from_room(room=room_to_be_removed)
        rooms.remove(room_to_be_removed)
        print("Sala encerrada")
        log_message(f'Sala {room_to_be_removed["name"]} foi encerrada.')
        threading.Thread(target=manage_sockets).start()

    else:
        print('Sala não encontrada.')

```

Agora iremos explicar a inicialização do programa. Ele dá início a uma thread e em seguida exibe o menu que antes foi declarado no começo do código. Ele fica executando um loop infinito validando se o que o usuário digitar é uma opção válida ou não

```

def init():
    threading.Thread(target=manage_sockets).start()
    log_message(f'Esperando conexões em {IP}:{PORT}...')
    option = ""
    first_time = 0
    while True:

```



```

print(option)
match option.upper():
    case '/CRIAR':
        create_room()

    case '/LISTAR':
        show_rooms()
        input('Pressione uma tecla para continuar...')
        print("\n\n")

    case '/LOGS':
        show_logs()

    case '/ENCERRAR':
        delete_room()

    case '/PARTICIPANTES':
        show_users_in_room()
        input('Pressione uma tecla para continuar...')
        print("\n\n")

    case '/SAIR':
        print("\nEncerrando aplicação...")
        os._exit(1)

    case _:
        if first_time != 0:
            print("\nComando Inválido!")
            time.sleep(1)
            first_time = 1
clear()
option = menu()

```

No final da função de inicialização também imprimimos a barra de menu onde irá receber os comandos do usuário

```

def menu():
    print('===== MENU =====')
    for comando in comandos:
        print(f'{comando} -----> {comandos[comando]}')
    try:
        option = input('Digite um comando: ')
    except KeyboardInterrupt as e:
        print("\n\nO programa foi finalizado abruptamente.")
        os._exit(1)

    return option

```

Também temos uma função que salva a mensagem dos logs, a função recebe uma mensagem e salva a mesma.

```
def log_message(message):  
    logs.append(message)
```

Agora tratamos o recebimento das mensagens vindo de chat_client.py

```
def receive_message(client_socket):  
    try:
```

Recebemos o header contendo o tamanho da mensagem

```
message_header = client_socket.recv(HEADER_LENGTH)
```

Se não recebermos dados, o servidor irá encerrar a conexão, usando socket.close() ou socket.shutdown(socket.SHUT_RDWR)

```
if not len(message_header):  
    return False
```

Convertemos o header da mensagem, para inteiro

```
message_length = int(message_header.decode('utf-8').strip())
```

Retornamos o header da mensagem e os dados também

```
return {'header': message_header, 'data': client_socket.recv(message_length)}
```

Se caso o cliente encerrou a conexão abruptamente ou perdeu a conexão retornamos false

```
except:  
    return False
```

Em seguida, fazemos um laço onde checamos os sockets presentes dentro da lista de sockets. Lemos as informações dos sockets presentes e realizamos a operação select()

```
def manage_sockets():  
    while True:  
        check_sockets_list()  
        read_sockets, _, exception_sockets = select.select(  
            sockets_list, [ ], sockets_list)
```

Aqui na função de verificar se o socket está presente na lista de sockets se ele não estiver ele é excluído

```
def check_sockets_list():  
    for checking_socket in sockets_list:  
        if checking_socket.fileno() == -1:  
            sockets_list.remove(checking_socket)
```

Iteramos os sockets notificados

```
for notified_socket in read_sockets:
```

Se o socket notificado é um socket servidor aceitamos a conexão

```
if notified_socket == server_socket:
```

Aceitando uma nova conexão, damos um novo socket (socket agora do cliente), conectado apenas a esse determinado cliente. O outro objeto retornado é ip/port set

```
client_socket, client_address = server_socket.accept()
```

Mandamos o nome do cliente

```
user = receive_message(client_socket)
```

Se o nome do usuário for falso significa que o mesmo caiu antes de mandar o nome

```
if user is False:
```

continue

Adicionamos o socket aceito à lista do select()

```
sockets_list.append(client_socket)
```

Salvamos o nome de usuário e o header e também codificamos essas informações num objeto JSON

```
data = json.loads(user['data'].decode('utf-8'))  
username = data['username']  
room = data['room']
```

Também adicionamos o usuário a uma sala

```
add_user_to_room(room_name=room, user={  
    'socket': client_socket, 'username': username})
```

Se o socket já está na lista então o mesmo está enviando uma mensagem e recebemos a mensagem

```
else:  
    send_message(socket=notified_socket)
```

Agora verificamos se um socket foi notificado dentro de uma lista dos sockets que receberam algum tipo de exceção. Tentamos remover esse socket se não for possível excluir o mesmo uma exceção é lançada

```
for notified_socket in exception_sockets:  
    try:  
        sockets_list.remove(notified_socket)  
        del clients[notified_socket]  
    except Exception as e:  
        log_message(f'Erro ao remover socket que deu errado: {e}')
```

A função de notificar quando um usuário sai da sala é simples. Realizamos um laço for para ver se o usuário se encontra na sala, em seguida enviamos os dados como nome de usuário e a mensagem que ele saiu da sala, codificamos essa mensagem em json e em seguida tentamos enviar a mensagem. Se não obtivermos êxito uma exceção é lançada

```
def notify_user_left_room(username_left, room):
    for user in room['users']:
        data = {'username': username_left, 'message': '*** SAIU DA SALA ***'}
        encoded_data = json.dumps(data).encode('utf-8')
        try:
            user['socket'].send(encoded_data)
        except Exception as e:
            log_message(f'Não foi possível enviar mensagem: {e}')
```

A função de notificar quando um usuário entra na sala funciona da seguinte maneira. Realizamos um laço for para ver se o usuário se encontra na sala, em seguida salvamos os dados como nome de usuário e a mensagem que ele entrou na sala, a informação é codificada em formato json e em seguida enviamos essa informação

```
def notify_user_join_room(username_join, room):
    for user in room['users']:
        data = {'username': username_join, 'message': '*** ENTROU NA SALA ***'}
        encoded_data = json.dumps(data).encode('utf-8')
        user['socket'].send(encoded_data)
```

A função de remover usuários de uma sala executa um laço for para ver se o usuário se encontra na sala, em seguida enviamos os dados como nome de usuário e a mensagem que a sala foi encerrada, a informação é codificada em formato json e em seguida enviamos essa informação. Em seguida as conexões dos sockets são finalizadas.

```
def remove_users_from_room(room):
    for user in room['users']:
        data = {'username': 'Sistema',
                'message': '*** ESTA SALA FOI ENCERRADA ***'}
        encoded_data = json.dumps(data).encode('utf-8')
        user['socket'].send(encoded_data)

    user['socket'].shutdown(socket.SHUT_RDWR)
    user['socket'].close()
```

A função de remoção dos sockets funciona da seguinte maneira. Pegamos os dados do usuário que saiu, na sala que o usuário saiu e em seguida removemos o usuário da lista de usuários. Notificamos a sala que o usuário saiu e em seguida excluimos o seu socket. Atualizamos a lista de sockets e em seguida emitimos uma mensagem de log afirmando que a conexão do usuário foi encerrada.

```
def remove_socket(socket_closed):
    user_left = clients[socket_closed]
    room_user_left = user_left['room']
    room_user_left['users'].remove(user_left)
```

```

notify_user_left_room(
    username_left=user_left['username'], room=room_user_left)

del clients[socket_closed]
sockets_list.remove(socket_closed)
log_message(f'Conexão encerrada de: {user_left["username"]}')

```

A função de envio de mensagem começa recebendo uma mensagem do socket, verifica se essa mensagem é falsa (se a mensagem for falsa então o cliente desconectou, então podemos removê-lo da lista de sockets). Se não pegamos o usuário do socket notificado, para saber quem enviou a mensagem e em seguida gravamos o log desta mensagem. Iteramos sobre os clientes conectados e as mensagens Broadcast. Verificamos para não enviarmos a mensagem para quem enviou a mesma, codificamos a mensagem para formato json e em seguida enviamos essa mensagem.

```

def send_message(socket):
    message = receive_message(socket)

    if message is False:
        remove_socket(socket)
        return

    sender_user = clients[socket]
    username = sender_user['username']

    log_message(
        f'{username} [{sender_user["room"]}["name"]]: {message["data"].decode("utf-8")}')

    for user in sender_user['room']['users']:

        if user['socket'] != socket:
            data = {'username': username,
                    'message': message['data'].decode('utf-8')}
            encoded_data = json.dumps(data).encode('utf-8')

            user['socket'].send(encoded_data)

```

A função de mostrar o log é muito simples. Ela apenas realiza um laço for, pega as mensagens e outras informações de log e exibe para o usuário.

```

def show_logs():
    clear()

```

```

print('***** LOGS *****')

for log in logs:
    print(f'- {log}')

input('Pressione uma tecla para continuar...')
print("\n\n")

```

A função de mostrar salas também é muito simples. Ela apenas realiza um laço for, pega o id das salas, o nome das salas, o número de participantes da sala que estão conectados e também o número limite de participantes da sala e exibe para o usuário.

```

def show_rooms():
    clear()
    print('***** SALAS *****')
    for room in rooms:
        print(
            f'{room["id"]}: {room["name"]} - {len(room["users"])} / {room["limit"]} usuários conectados.')

```

A função de listar participantes em uma sala inicia atribuindo o valor da variável da sala selecionada em 'None' em seguida mostra as salas disponíveis e pede para um usuário digitar o nome da sala que deseja ver os participantes. Com o nome da sala fazemos um laço for para procurar se a sala inserida está entre a lista de salas existentes. Se o nome da sala for existente, mostramos o nome da sala e o nome dos participantes. Se a sala não existir, notificamos para o usuário que a sala não existe e pedimos pro mesmo tentar inserir outro nome de uma sala existente.

```

def show_users_in_room():
    clear()
    room_selected = None
    show_rooms()
    room_name = input(
        "\nDigite o nome da sala que deseja ver os participantes: ")
    for room in rooms:
        if room['name'] == room_name:
            room_selected = room

    users_in_room = len(room["users"])

    if room_selected is not None:
        clear()
        print(f'***** PARTICIPANTES DA SALA "{room_name}" *****')
        i = 1
        print("")

```

```

for user in room_selected["users"]:
    print(f'{i} - {user["username"]}')
    i+=1
else:
    print("\nSala não encontrada. Digite outra sala.")
    time.sleep(3)
    clear()
    show_users_in_room()

```

Por fim temos a execução do início do programa onde chamamos a função `init()` previamente definida e também tratamos a exceção de quando apertamos `Ctrl +C`

```

try:
    init()
except KeyboardInterrupt as e:
    print("\n\nO programa foi finalizado abruptamente.")
    os._exit(1)

```

Chat_client.py

As primeiras linhas do código nós definimos informações básicas do nosso servidor, como o tamanho do `HEADER`, o `IP`, e a `PORT`:

```

HEADER_LENGTH = 10

IP = "127.0.0.1"
PORT = 1234

```

Em seguida limpamos o `buffer`

```

def limparBuffer(my_username):
    print(f'\n{my_username} > ', end="")
    sys.stdout.flush()

```

Em seguida criamos uma função que recebe a mensagem e decodifica a mesma, fazemos também a verificação se o programa não foi encerrado de forma forçada (Apertando `Ctrl +C`). Se o programa não foi finalizado a função retorna a mensagem decodificada e em formato `json`.

```

def decode_message(receiver):
    message = ""
    has_message = True

```



```

while has_message:
    try:
        char = receiver.recv(1).decode('utf-8')
        message += char

        if char == '}':
            has_message = False
    except KeyboardInterrupt as e:
        print("\n\nO programa foi finalizado abruptamente.")
        os._exit(1)

return json.loads(message)

```

Na função de inicialização pedimos para o cliente inserir um nome de sala

```

def init():
    my_room = input("Escolha a sala: ")

```

Criamos um socket com conexão TCP

```

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

```

Conectamos o socket em um ip e em uma porta

```

client_socket.connect((IP, PORT))

```

Definimos a conexão para não-blocante, para que o recv() não bloqueie, só retorne algum erro

```

client_socket.setblocking(False)

```

Preparamos o nome de usuário e também o nome da sala para enviar em seguida. Codificamos os dados para json e também codificamos para bytes, depois contamos o número de bytes e preparamos o header com tamanho fixo, que também codificamos para bytes

```

data = {
    'username': my_username,
    'room': my_room,
}

```

```

encoded_data = json.dumps(data).encode('utf-8')
username_header = f'{{len(encoded_data):<{HEADER_LENGTH}}}'.encode('utf-8')
client_socket.send(username_header + encoded_data)

```

Em seguida exibimos a opção de sair e criamos uma thread para limpar o buffer

```

print('Para sair, digite o comando /SAIR.')
threading.Thread(target=limparBuffer(my_username)).start()

```

Realizamos um laço while que faz as operações select(), e separamos quais readers estão presentes no client_socket

```

while True:
    readers, _, _ = select.select([sys.stdin, client_socket], [], [])
    for reader in readers:
        if reader is client_socket:
            try:

```

Fazemos a decodificação da mensagem que foi recebida pelo client_socket e analisamos o conteúdo dessa mensagem verificando se o usuário que enviou ela se chama 'Sistema'. Se caso as mensagens recebidas for do usuário 'Sistema' significa que são mensagens de erros e exceções que em seguida são tratadas e exibidas. Além disso, a thread faz a limpeza do buffer.

```

message = decode_message(receiver=client_socket)
if message["username"] == 'Sistema':
    match message["message"]:
        case '404':
            print("\n\nA sala escolhida não existe. Tente novamente.")
            time.sleep(3)
            clear()
            init()
        case '501':
            print("\n\nO limite da sala escolhida foi atingido. Escolha outra sala.")
            time.sleep(3)
            clear()
            init()
    sys.stdout.flush()
    print("\033[A \033[A")
    print(f'\n{message["username"]} > {message["message"]}')
    threading.Thread(target=limparBuffer(my_username)).start()

```

Se caso não vier dados, tentamos novamente. Caso dê errado novamente executamos sys.exit()

```
except IOError as e:
    if e.errno != errno.EAGAIN and e.errno != errno.EWOULDBLOCK:
        print('Reading error: {}'.format(str(e)))
        sys.exit()
    continue

except Exception as e:
    print('Reading error: {}'.format(str(e)))
    sys.exit()
```

Se não ocorreu nenhum erro esperamos o usuário mandar a mensagem

```
else:
    threading.Thread(target=limparBuffer(my_username)).start()
```

Verificamos tanto se o usuário encerrou abruptamente a conexão quanto se ele digitou o comando de sair da sala.

```
try:
    message = input()
except KeyboardInterrupt as e:
    print("\n\nO programa foi finalizado abruptamente.")
    os._exit(1)

if message.upper() == "/SAIR":
    print('Saindo da sala...')
    client_socket.shutdown(socket.SHUT_RDWR)
    client_socket.close()
    time.sleep(1)
    clear()
    init()
```

Se a mensagem não estiver vazia ele envia ela, porém antes codifica a mensagem para bytes, prepara o header e converte para bytes

```
if message:
    message = message.encode('utf-8')
    message_header = f"{len(message):<{HEADER_LENGTH}}".encode('utf-8')
    client_socket.send(message_header + message)
```

Por fim temos a execução do início do programa onde pedimos para que o usuário se identifique, e logo em seguida chamamos a função `init()` previamente definida e também tratamos a exceção de quando apertamos `Ctrl +C`

```
try:
    my_username = input("Username: ")
    init()
except KeyboardInterrupt as e:
    print("\n\nO programa foi finalizado abruptamente.")
    os._exit(1)
```

Aprendizados

João Vitor de Souza Durso

Yan Andrade de Sena

Neste projeto foi muito satisfatório e enriquecedor, tinha mexido pouquíssimas vezes com a linguagem de programação python mas com esse projeto pude me aprofundar um pouco mais. Usando a função `select()` pude ver como ela torna a gerência mais fácil entre sockets e também pude ver como um servidor e um cliente se diferenciam.

Além disso, achei bacana a parte de criptografar a mensagem para ela ser enviada para um servidor e também achei muito produtivo ver como uma conexão TCP funciona.

Já teve atividades passadas que o professor passou que pude ver as diferenças entre TCP e UDP mas sinto que nesse projeto consegui me aprofundar mais sobre o protocolo TCP/IP.

Também gostei da forma que o servidor gerencia as funções quando elas são inseridas no terminal pelo usuário, gostei bastante da apresentação dos menus.

Tive muita dificuldade em conseguir desenvolver algo no código já que consegui me reunir poucas vezes com meu grupo. Tentei fazer a autenticação com OAuth2.0 mas não consegui rodar o código de exemplo disponibilizado na documentação do OAuth2.0 com python. Mas consegui aprender os conceitos de OAuth2.0 e também como ele seria aplicado no projeto.

Tive bastante dificuldade também em conseguir desenvolver a função de criação de salas privadas e públicas, eu sabia o que fazer em teoria mas limitações como conhecimento da linguagem me impediram de implementar essa funcionalidade.

Em resumo, gostei de entender sobre o projeto, explicar cada linha de código neste relatório e aprender mais sobre o protocolo TCP/IP além de aplicar o mesmo.

Victor Buendia Cruz de Alvim