

Bernardo de Castro Cerqueira - 217031134
Gustavo Luppi Siloto - 217031126

Link do github:

<https://github.com/Rissko/redes.git>

Introdução:

Nessa parte do trabalho fizemos a função de ligação utilizando o socket UDP e a biblioteca pyaudio.

OBS: Conseguimos consertar os problemas declarados pelo monitor nas duas observações colocadas.

Implementação:

```
class BusyManager:
    def __init__(self):
        self.flag = {'online': False}
```

Para começo de conversa, criamos uma classe denominada *BusyManager.py*, a qual possui um flag que é iniciado como falso. Esse flag é visto por todo o programa. Realizamos dessa forma, porque ao realizarmos uma conexão com outro usuário, o flag é setado para True e assim podemos impedir que outra pessoa se conecte a chamada.

método `receive()` de `client.py`

```
def receive():
    global client
    client = socket(AF_INET, SOCK_STREAM)
    ip_info = ip.get()
    HOST = ip_info
    PORT = 5000
    client.connect((HOST, PORT))
    client.send(name.get().encode())
    message = client.recv(1024).decode()
    print("passei aqui " + message)
    if message == "Ja existe um usuario com o mesmo nome":
        write(message)
    elif "Conectado ao servidor" in message:
        split = message.split("/")
        write(split[0])
        registrybtn['state'] = "disabled"
        registryentry['state'] = "disabled"
        connectbtn['state'] = "active"
        connectentry['state'] = "normal"
        endbtn['state'] = "active"
        ipentry['state'] = "disabled"
        screen.protocol("WM_DELETE_WINDOW", closeConnThread)
        thread = threading.Thread(target=listen)
        thread.start()
        thread = threading.Thread(target=serverLigacao.iniciarServidorLigacao, args=(str(split[1]), popup, endcallbtn, busy, console, connectbtn))
        thread.start()
```

Ao iniciarmos a conexão, agora iniciamos outra thread da função `serverLigacao.iniciarServidorLigacao()`, passando como argumentos o ip do servidor, o método `popup`

`popup()` de `client.py`

```
def popup(origem, origem_nome):
    global popupwindow
    global resp
    #TODO: Adicionar botoes de aceito/rejeito
    #TODO: Criar métodos no server ligação para enviar resposta a origem
    popupwindow = Tk()
    popupwindow.geometry("300x300")
    popupwindow.title("Convite de chamada")
    connectionText = Label(popupwindow, text=origem_nome + str(origem) + "Quer se conectar")
    connectionText.pack()
    acceptbtn = Button(popupwindow, text="Aceitar", height="2", width="30", command=ligacaoAceita)
    acceptbtn.pack()
    declinebtn = Button(popupwindow, text="Rejeitar", height="2", width="30", command=ligacaoRejeitada)
    declinebtn.pack()
    popupwindow.mainloop()
    return resp
```

que será a nossa janela de convite tendo dois botões, o `acceptbtn` e o `declinebtn` que chamam, respectivamente, as funções `ligacaoAceita()` e `ligacaoRejeitada()`, portanto se o botão `acceptbtn` for apertado a variável `resp` terá o valor 's', se o `declinebtn` for pressionado, ela terá o valor 'n'.

```
def ligacaoAceita():
    global popupwindow
    global resp
    resp = 's'
    popupwindow.destroy()

def ligacaoRejeitada():
    global popupwindow
    global resp
    resp = "n"
    popupwindow.destroy()
```

Passaremos como parâmetro para a função `serverLigacao.iniciarServidorLigacao()` também o botão `endcallbtn` que é o botão para fechar a ligação, `busy`, o nosso objeto da classe `BusyManager`, o `console`, que é o nosso log de mensagens e, por fim, o botão de conexão.

iniciarServidorLigacao() em serverLigacao.py

```
def iniciarServidorLigacao(meuIp, callback, endcallbtn, busy, console, connectbtn):
    global servidorUdp
    global origem_call
    global output_stream
    busy.flag['online'] = False
    HOST = meuIp
    PORT = 6000
    servidorUdp = socket(AF_INET, SOCK_DGRAM)
    origem = (HOST, PORT)
    servidorUdp.bind(origem)
    write(console, "Iniciando servidor de ligação")
    py_audio = pyaudio.PyAudio()
    buffer = 4096 # 127.0.0.1
    output_stream = py_audio.open(format=pyaudio.paInt16, output=True, rate=44100, channels=2, frames_per_buffer=buffer)
    while True:
        msg, origem_call = servidorUdp.recvfrom(4096)
        if "convite" in str(msg):
            if busy.flag['online']:
                servidorUdp.sendto("resposta_ao_convite/ocupado".encode(), origem_call)
            else:
                split = str(msg).split('/')
                resp = callback(origem_call, split[1])
                if "s" in resp:
                    write(console, "Convite Aceito")
                    write(console, "Começando ligação...")
                    output_stream = py_audio.open(format=pyaudio.paInt16, output=True, rate=44100, channels=2, frames_per_buffer=buffer)
                    servidorUdp.sendto("resposta_ao_convite/aceito".encode(), origem_call)
                    busy.flag['online'] = True
                    endcallbtn['state'] = "active"
                    connectbtn['state'] = "disabled"
                    thread2 = threading.Thread(target=enviarAudio, args=(busy, endcallbtn, console, connectbtn))
                    thread2.start()
                elif "n" in resp:
                    servidorUdp.sendto("resposta_ao_convite/rejeitado".encode(), origem_call)
        elif "encerrar_ligacao" in str(msg):
            busy.flag['online'] = False
            endcallbtn['state'] = "disabled"
            connectbtn['state'] = "active"
        elif busy.flag['online']:
            output_stream.write(msg)
```

É aqui que o servidor UDP é inicializado. Primeiro setamos o valor de nosso flag para False, depois fazemos o bind de nosso servidor e iniciamos o pyaudio e um stream para escutar o áudio. Depois disso iniciamos um while True para escutarmos as mensagens de *clientLigacao.py*. Se a mensagem for “convite” veremos se a flag está com o estado True, se estiver, quer dizer que a ligação está online então enviaremos para o cliente a resposta de ocupado. Se o flag for falso então poderemos iniciar uma nova ligação. Primeiro realizamos a chamada da função callback que é o parâmetro que a função popup é passada na função. Realizamos essa função e um convite irá aparecer na tela para o usuário destino da ligação. Se o botão de aceitar for apertado resp receberá a string ‘s’ será enviada uma mensagem de convite “aceito” para o usuário de origem da chamada a flag será setada para True, o botão de finalizar a ligação se tornará ativo enquanto o de se conectar estará desativado e a thread de *enviarAudio()* se iniciará. Caso seja apertado o botão de rejeitar a ligação será enviado para o cliente origem a resposta “rejeitado”.

enviarAudio()

```
def enviarAudio(busy, endcallbtn, console, connectbtn):
    global py_audio
    global input_stream
    py_audio = pyaudio.PyAudio()
    buffer = 1024
    input_stream = py_audio.open(format=pyaudio.paInt16, input=True, rate=44100, channels=2, frames_per_buffer=buffer)
    while busy.flag['online']:
        data = input_stream.read(buffer)
        servidorUdp.sendto(data, origem_call)
    busy.flag['online'] = False
    endcallbtn['state'] = "disabled"
    connectbtn['state'] = "active"
    write(console, "Ligacao finalizada")
    input_stream.stop_stream()
    input_stream.close()
    output_stream.stop_stream()
    output_stream.close()
    py_audio.terminate()
```

A função *enviarAudio()* abrirá um stream de input de áudio, enquanto o flag for True ele continuará enviando áudio para o cliente de origem, ao ser setado para False, saímos do while e setamos o flag para False pois a ligação já estará encerrada, desabilitamos o botão de finalizar ligação, habilitamos o de conexão e logo depois encerramos os streams e o próprio pyaudio.

Agora que falamos sobre o cliente destinatário, começaremos a falar sobre o cliente de origem e como ele vai enviar o convite, receber áudio e enviar áudio também. Tudo começa com o botão de conexão, o connectbtn.

```
connectbtn = Button(text="Procurar usuário e conectar", height="2", width="30", state="disabled", command=sendNameThread)
```

```
def sendNameThread():
    thread = threading.Thread(target=sendName)
    thread.start()
```

```
def sendName():
    dest_info = dest.get()
    print("Chamando consulta")
    client.send("consulta".encode())
    client.send(dest_info.encode())
```

Ao pressionar o botão connectbtn, a thread será chamada e enviaremos uma mensagem de “consulta” para o servidor

em server.py

```
elif estado == "consulta":
    requested_name = client.recv(1024).decode()
    if requested_name == names[clients.index(client)]:
        client.send("Nao e possivel conectar a si mesmo".encode())
    elif requested_name not in names:
        client.send("Nome nao encontrado".encode())
    else:
        i = names.index(requested_name)
        host, port = clients[i].getpeername()
        client.send(("endereco/" + str(host) + "/" + str(port) + "/" + names[clients.index(client)]).encode())
```

A mensagem é recebida na função *handle()* do servidor. Os nomes são devidamente tratados e por fim se o nome for válido, é enviado para o cliente atual uma mensagem "endereco" que irá enviar o ip, a porta do cliente destinatário e o nome do cliente atual.

```
def listen():
    print("Iniciei o listen")
    while True:
        message = client.recv(1024).decode()
        if "endereco" in message:
            print("RECEBI DO SERVER ESSES DADOS: " + str(message))
            dados = message.split('/')
            clientLigacao.iniciaConexaoUDP(dados[1], dados[3], endcallbtn, busy, console, connectbtn)
```

De volta à cliente.py essa mensagem é recebida na função *listen()* e logo depois é chamado o método *clientLigacao.iniciaConexaoUDP()* enviando como parâmetros o ip do cliente destinatário, o nome do cliente atual, o botão endcallbtn, o objeto busy, o log console e o connectbtn.

iniciaConexaoUDP() em *clientLigacao.py*

```
def iniciaConexaoUDP(dest_ip, origem_name, endcallbtn, busy, console, connectbtn):
    global conexaoUdp
    global dest
    write(console, " Destino: " + str(dest_ip))
    HOST = dest_ip
    PORT = 6000
    conexaoUdp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    dest = (HOST, PORT)
    write(console, "Enviando convite")
    conexaoUdp.sendto(("convite/" + origem_name).encode(), dest)
    thread = threading.Thread(target=ouvirResposta, args=(conexaoUdp, endcallbtn, busy, console, connectbtn))
    thread.start()
```

Nesse método é criado o socket UDP e é enviado o convite para o destinatário, depois disso, iniciamos uma thread da função *ouvirResposta()*.

ouvirResposta()

```
def ouvirResposta(conexaoUdp, endcallbtn, busy, console, connectbtn):
    global output_stream
    global py_audio
    py_audio = pyaudio.PyAudio()
    buffer = 4096
    output_stream = py_audio.open(format=pyaudio.paInt16, output=True, rate=44100, channels=2,
                                   frames_per_buffer=buffer)

    while True:
        msg, endereco = conexaoUdp.recvfrom(4096)
        #TODO: Os pacotes de audio devem chegar aqui tbm, tratar isso no futuro
        if "aceito" in str(msg):
            write(console, "Iniciando chamada")
            busy.flag['online'] = True
            endcallbtn['state'] = "active"
            connectbtn['state'] = "disabled"
            thread = threading.Thread(target=send_audio, args=(endereco, busy, console, endcallbtn))
            thread.start()
        elif "rejeitado" in str(msg):
            write(console, "Convite rejeitado")
        elif "ocupado" in str(msg):
            write(console, "Usuário está ocupado")
        elif "encerrar_ligacao" in str(msg):
            busy.flag['online'] = False
            endcallbtn['state'] = "disabled"
            connectbtn['state'] = "active"
        elif busy.flag['online']:
            output_stream.write(msg)
```

Na função *ouvirResposta()* iniciamos o pyaudio e uma stream de áudio que servirá para escutar o cliente destinatário. Dentro do while recebemos as mensagens do destinatário pelo método *iniciarServidorLigacao()* em *serverLigacao.py*. Se 'aceito' estiver na mensagem, o flag será setado como True, o endcallbtn será ativado e o connectbtn desativado no tkinter e iniciaremos a thread de *send_audio()* (Será explicado em seguida), se a mensagem for igual 'rejeitado', será escrito no console 'Convite rejeitado' e a conexão

não será inicializada, se a mensagem for “ocupado”, será escrito no console “Usuário está ocupado”, no caso em que a mensagem seja igual a ‘encerrar_ligação’, iremos setar o flag para False, dessa forma, a conexão será encerrada, iremos desabilitar o endcallbtn e habilitar o connectbtn. Se nada citado anteriormente ocorrer então entraremos no último elif, se o flag for True devemos receber a mensagem com os bytes de áudio.

send_audio():

```
def send_audio(dest, busy, console, endcallbtn, connectbtn):
    global input_stream
    global py_audio
    write(console, "Destino: " + str(dest))
    buffer = 1024

    input_stream = py_audio.open(format=pyaudio.paInt16, input=True, rate=44100, channels=2, frames_per_buffer=buffer)

    while busy.flag['online']:
        data = input_stream.read(buffer)
        conexaoUdp.sendto(data, dest)
    busy.flag['online'] = False
    endcallbtn['state'] = "disabled"
    connectbtn['state'] = "active"
    write(console, "Ligacao finalizada")
    input_stream.stop_stream()
    input_stream.close()
    output_stream.stop_stream()
    output_stream.close()
    py_audio.terminate()
```

Assim como na função *enviarAudio()* do *serverLigacao.py*, a função *send_audio()* do *clienteLigacao.py* abrirá um stream de input de áudio, enquanto o flag for True ele continuará enviando áudio para o cliente de origem, ao ser setado para False, saímos do while e setamos o flag para False pois a ligação já estará encerrada, desabilitamos o botão de finalizar ligação e habilitamos o de conexão e logo depois encerramos os streams e o próprio pyaudio.

Por fim, daremos uma olhada nas funções *finalizaConexao()* de *clientLigacao.py* e *servidorLigacao.py*.

finalizaConexao() de *clientLigacao.py*:

```
def finalizaConexao(busy):
    try:
        if busy.flag['online']:
            conexaoUdp.sendto("encerrar_ligacao".encode(), dest)
            busy.flag['online'] = False
    except:
        pass
```

finalizaConexao() de *serverLigacao.py*:

```
def finalizaConexao(busy):
    try:
        if busy.flag['online']:
            servidorUdp.sendto("encerrar_ligacao".encode(), origem_call)
            busy.flag['online'] = False
    except:
        pass
```

Na função *finalizaConexao()* , temos o flag como parâmetro. Vemos se o flag é True e mandamos uma requisição para encerrar a ligação, então o flag é setado para False.

Ambas as funções são chamadas em *client.py* quando é clicado no botão *endcallbtn*.

```
endcallbtn = Button(screen, text="Encerrar Ligação", height="2", state="disabled", width="30", command=finalizarLigacao)
```

```
def finalizarLigacao():
    endcallbtn['state'] = "disabled"
    clientLigacao.finalizaConexao(busy)
    serverLigacao.finalizaConexao(busy)
```

E também são chamadas na função *listen()* de cliente para que a ligação seja finalizada se um usuário conectado nela decidir encerrar a conexão com o servidor.

```
elif "finish" in message:
    if busy.flag['online']:
        clientLigacao.finalizaConexao(busy)
        serverLigacao.finalizaConexao(busy)
        write("Conexão encerrada")
        write("Fechando em 2 segundos...")
        time.sleep(2)
        screen.destroy()
```

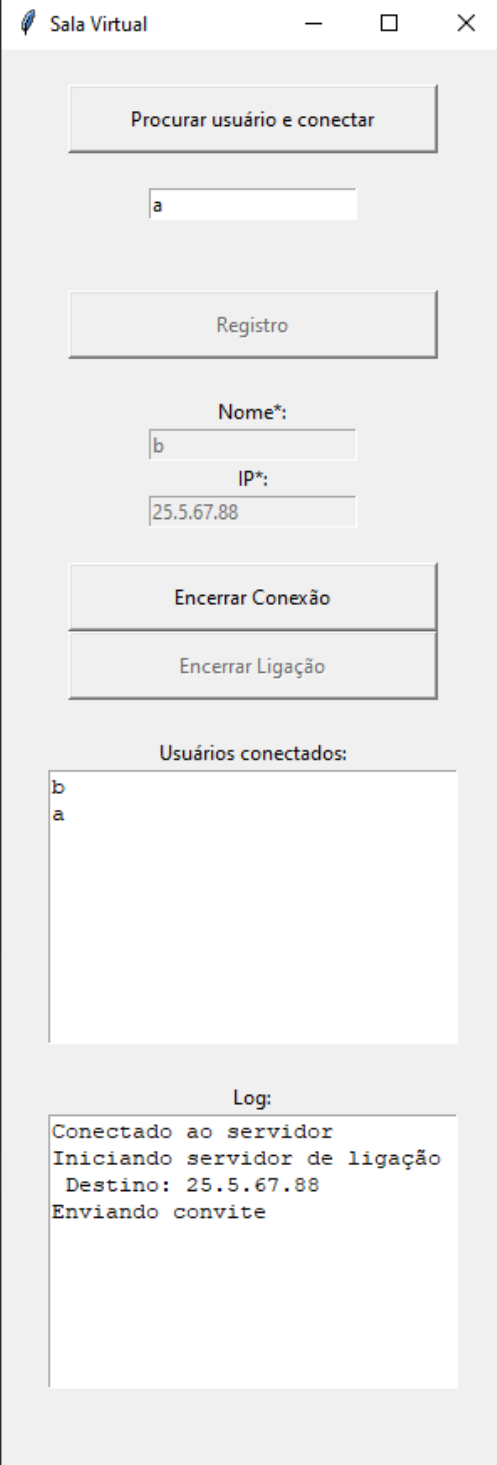

Funcionalidade no tkinter:

Após conectarmos ao servidor, aparecerá assim no tkinter:

The screenshot shows a Tkinter application window titled "Sala Virtual". The interface includes the following elements:

- A button labeled "Procurar usuário e conectar".
- An empty text input field.
- A button labeled "Registro".
- A label "Nome*:" followed by a text input field containing the letter "b".
- A label "IP*:" followed by a text input field containing the IP address "25.5.67.88".
- A button labeled "Encerrar Conexão".
- A button labeled "Encerrar Ligação".
- A label "Usuários conectados:" above a text area containing the text "b" and "a" on separate lines.
- A label "Log:" above a text area containing the text "Conectado ao servidor" and "Iniciando servidor de ligação" on separate lines, with a cursor at the end of the second line.

Então, a seguir, tentaremos nos conectar a outro usuário, clicando no botão *‘Procurar usuário e conectar’*, e no log será registrado que foi enviado o convite para o destino do outro usuário.

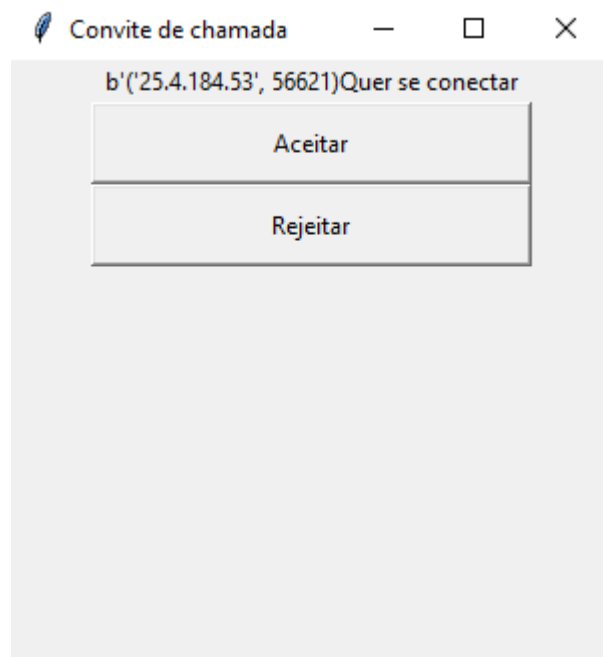


The image shows a window titled "Sala Virtual" with standard window controls (minimize, maximize, close). The interface is divided into several sections:

- Top Section:** A button labeled "Procurar usuário e conectar". Below it is a text input field containing the letter "a".
- Second Section:** A button labeled "Registro".
- Third Section:** Two input fields. The first is labeled "Nome*" and contains the letter "b". The second is labeled "IP*" and contains the address "25.5.67.88".
- Fourth Section:** Two buttons stacked vertically: "Encerrar Conexão" and "Encerrar Ligação".
- Fifth Section:** A label "Usuários conectados:" followed by a text area containing the letters "b" and "a" on separate lines.
- Sixth Section:** A label "Log:" followed by a text area containing the following text:

```
Conectado ao servidor
Iniciando servidor de ligação
Destino: 25.5.67.88
Enviando convite
```

Será recebido então para o outro usuário um popup do convite, podendo ele aceitar ou rejeitar.



Esta tela será apresentada se o convite pelo outro usuário for rejeitado , como pode ser visto no log do console.

Sala Virtual

— □ ×

Procurar usuário e conectar

a

Registro

Nome*:
b

IP*:
25.5.67.88

Encerrar Conexão

Encerrar Ligação

Usuários conectados:

b
a
c

Log:

Conectado ao servidor
Iniciando servidor de ligação
Destino: 25.5.67.88
Enviando convite
Convite rejeitado


Esta tela será apresentada se o convite for aceito pelo outro usuário, como podemos ver no log do console.

The screenshot shows a window titled "Sala Virtual" with standard window controls. The interface is organized into several sections:

- Top Section:** A button labeled "Procurar usuário e conectar".
- User Input Section:** A text input field containing the letter "a".
- Registration Section:** A button labeled "Registro".
- Form Section:** Three input fields labeled "Nome*", "IP*", and "25.5.67.88".
- Action Buttons:** Two buttons labeled "Encerrar Conexão" and "Encerrar Ligação".
- Connected Users Section:** A list titled "Usuários conectados:" containing the letters "b", "a", and "c".
- Log Section:** A text area titled "Log:" displaying the following text:

```
Conectado ao servidor
Iniciando servidor de ligação
Destino: 25.5.67.88
Enviando convite
Convite rejeitado
Destino: 25.5.67.88
Enviando convite
Iniciando chamada
Destino: ('25.5.67.88', 6301)
```

Esta tela aparecerá após finalizarmos a ligação, clicando no botão *‘Encerrar Ligação’*, será apresentado no log do console a frase *‘Ligação finalizada’*.

 Sala Virtual—□×

Procurar usuário e conectar

Registro

Nome*:

IP*:

Encerrar Conexão

Encerrar Ligação

Usuários conectados:

b


a

c

Log:

```
Iniciando servidor de ligação
Destino: 25.5.67.88
Enviando convite
Convite rejeitado
Destino: 25.5.67.88
Enviando convite
Iniciando chamada
Destino: ('25.5.67.88', 6301)
Ligacao finalizada
```

Esta tela irá aparecer se um outro usuário tentar se conectar a alguém que já está em uma conexão estabelecida. Irá aparecer no log a seguinte frase: *'Usuário ocupado'*.

 Sala Virtual—□×

Procurar usuário e conectar

b

Registro

Nome*:
c

IP*:
25.5.67.88

Encerrar Conexão


Encerrar Ligação

Usuários conectados:

b
a
c

Log:
Conectado ao servidor
Iniciando servidor de ligação
Destino: 25.4.184.53
Enviando convite
Usuário está ocupado

Esta tela ocorrerá quando o botão *'Encerrar Conexão'* for clicado. Na parte *'Usuários conectados'* o usuário será retirado e no log será escrito que a conexão foi encerrada.

 Sala Virtual

—

□

×

Procurar usuário e conectar

Registro

Nome*:

b

IP*:

25.5.67.88

Encerrar Conexão

Encerrar Ligação

Usuários conectados:

a

c

Log:

Enviando convite
Convite rejeitado
Destino: 25.5.67.88
Enviando convite
Iniciando chamada
Destino: ('25.5.67.88', 6301)
Ligacao finalizada
Conexão encerrada
Fechando em 2 segundos...