UNIVERSIDADE DE BRASÍLIA - UNB

Faculdade UnB Gama - FGA
Curso de Engenharia de Software

PROJETO DE PESQUISA - CRIANDO AMBIENTES VIRTUAIS DE CONVERSAÇÃO COM O USO SYSTEM CALL SELECT()

Fundamentos de Redes de Computadores - Turma A

Professor: Fernando W. Cruz

Pedro Henrique Carvalho Campos. 190036435

Victor Hugo Carvalho Silva. 190038969

Brasília, DF
2021

SUMÁRIO

1. Introdução

O presente relatório tem como objetivo aprofundar os conhecimentos trazidos em sala de aula acerca de aplicações em redes de computadores, seguindo as premissas da arquitetura TCP/IP. Para que esse objetivo seja alcançado, foi construído uma aplicação de bate-papo que disponibiliza a criação de salas, com grupos virtuais, para que as pessoas possam entrar nesses grupos e interagir.

O protocolo TCP/IP significa protocolo de controle de transmissão/ protocolo da internet, sendo esse um conjunto de regras padronizadas para que os computadores possam se comunicar em uma rede. Ele foi desenvolvido como um modelo de arquitetura como solução real para transmissão de dados e foi amplamente divulgado e utilizado. O TCP/IP é dividido em quatro camadas, são elas:

- Camada de aplicação;
- Camada de transporte;
- Camada de internet;
- Camada de enlace;
- Camada de Interface de Redes;

O protocolo TCP/IP fornece utilidades importantes para a comunicação segura entre máquinas (sendo esse um dos principais motivos pela sua popularização), sendo bastante recomendado para ser usado em chats.

Para realização da aplicação foi utilizada a linguagem de programação python, utilizando-se de bibliotecas responsáveis por fazer a comunicação full-duplex entre cliente e servidor. Foi utilizado a System Call Select (biblioteca select em Python) para que fosse possível gerenciar o diálogo entre Cliente e Servidor.

As funcionalidades presentes na aplicação incluem a criação de salas (com possibilidade de escolher o nome), listagem dos membros participantes de uma sala, ingresso de clientes em uma sala (tendo a possibilidade de escolher seu identificador), saída de clientes de uma sala e diálogos entre clientes de uma mesma sala.

2. Metodologia Utilizada

A metodologia aplicada no projeto se baseou, principalmente, em uma pesquisa em livros, vídeos na Internet (mais especificamente o portal YouTube) e sites na Internet. Para a realização do projeto foram realizados encontros virtuais síncronos (utilizando a ferramenta discord), onde foi possível fazer programação em pares, e também se baseando em conversas por serviços de troca de mensagem como o WhatsApp para organizar os encontros e troca de mensagens rápidas.

O início do projeto se deu no dia 22/09/2022, onde a dupla se reuniu de maneira síncrona para definição de qual linguagem de programação seria utilizada na aplicação, além de dar início a programação, além de outras definições que seriam de extrema importância para a continuidade do projeto.

Com as definições básicas feitas foi possível dar início ao projeto de maneira mais apropriada. No dia seguinte se deu início ao desenvolvimento do projeto, utilizando-se da programação em pares para que fosse possível avançar mais rapidamente. Para isso foi utilizado o compartilhamento de tela da ferramenta Discord, onde um dos membros programava e o outro ajudava, fazendo pesquisas e dando ideias de como progredir.

A última reunião foi feita no dia 25/09/2022, onde a dupla se reuniu para finalização do projeto, fazendo a confecção do relatório e dos slides

3. Descrição da Solução

Como dito, o projeto foi construído com base na linguagem de programação Python e se baseia em dois arquivos, sendo eles o server.py e o client.py. Esses dois arquivos têm como objetivo fazer a comunicação entre Cliente e Servidor. O servidor permite a possibilidade de que vários clientes possam adentrar em uma mesma sala para que assim seja possível a comunicação entre um grupo de pessoas com uma comunicação Full-Duplex. Para que possa ser possível rodar a aplicação alguns pré-requisitos devem ser cumpridos, entre eles estão:

- Python: Versão 2.7 ou superior;

- TKinter: Ferramenta para facilitação de criação de interfaces GUI no Python;
- PySimpleGUI: Ferramenta para facilitação de criação de interfaces GUI no Python;

O link do projeto para que se possa ser acessado e executado é o seguinte: https://github.com/Peh099/Trabalho-Final-FRC. Lá é possível encontrar uma descrição básica de como rodar a aplicação. O processo para execução baseia-se em:

1) Instalação do PySimpleGUI: $ python3 -m pip install PySimpleGUI
2) Executar o servidor: $ python server.py
3) Executar o cliente: $ python client.py

3.1. Server.py

O arquivo server.py é responsável por gerir todos os clientes que se conectarem à aplicação, atuando como servidor na comunicação entre cliente-servidor. Pode-se dizer que o servidor atua como intermediário entre os clientes, transmitindo as mensagens para todos os clientes dentro de uma determinada sala. O arquivo pode ser visualizado na Figura 1, sendo inicializado com a declaração de qual será o endereço host da comunicação (sendo que nesse caso será utilizado o endereço de loopback para comunicação local (“127.0.0.1”)) e a porta que será utilizada na comunicação entre cliente-servidor (9001).

Logo após é declarada a variável servidor, que é responsável por criar o socket. Um socket é responsável por prover a comunicação entre duas pontas (fonte e destino) entre dois processos que estejam na mesma máquina ou na mesma rede. Logo após é utilizado o método setsockopt, sendo esse um fornecedor de meios para que se possa controlar o comportamento desse socket.

Após isso é feito um bind no socket recém criado. Esse método bind() atribui um endereço IP e número de porta a uma instância de socket. Com isso, o socket recém criado irá se tornar um socket servidor. Dessa maneira um socket do cliente poderá usar o método connect() para contatar o socket servidor.

A constante MAXCLI determina o máximo de clientes que poderão se conectar a uma determinada sala. Para facilitar a experimentação o número máximo de clientes em uma sala será dois, uma vez que se o número fosse suficientemente maior seria difícil demonstrar a funcionalidade.

As declarações seguintes guardam as listas de informações que são importantes para outras funções dentro do arquivo server.py.

```
2  import socket
3  import select
4
5  # ip host e porta
6  HOST = '127.0.0.1'
7  PORT = 9001
8
9  # conexão socket
10 servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)    # criar socket
11 servidor.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)  # fornece os meios para controlar o comportamento do socket
12 servidor.bind((HOST, PORT))  # bind
13
14 MAXCLI = 2 # máximo de clientes por sala
15
16 # servidor espera por conexões
17 servidor.listen(MAXCLI)
18
19 nomes = []          # lista dos nomes dos clientes
20 salas = []          # lista dos nomes das salas
21 paresCliSala = []   # lista de pares (socket do cliente, nome da sala)
22 paresSalaTam = []   # lista de pares (nome da sala, numero de pessoas na sala)
23 all_sockets=[]      # lista de todos os sockets
24 paresSalaNomes = [] # lista de pares (nome da sala, nome do cliente)
25
```

Figura 1: Algumas declarações de variáveis e constantes.

Na função receber ocorre a organização do diálogo com a system call select para fazer a comunicação full-duplex entre servidor e cliente. Então, ele passa por todos os sockets que podem ser lidos através do select, caso haja uma nova conexão, ele recebe alguns dados do servidor como os nomes dos clientes, as salas e a lista de integrantes por sala que foram cadastradas. Após isso, é realizada a conexão, o preenchimento de alguns valores iniciais como nome do cliente que está fazendo a conexão e a sala que deseja entrar, é realizada algumas validações quanto ao limite da sala, algumas listas são atualizadas e a mensagem de novo integrante da sala é enviado aos participantes da mesma. Por fim, caso não for uma nova conexão e sim mensagem de algum cliente, a função chat é chamada passando o socket da mensagem em questão como parâmetro.

```python
def receber():
    all_sockets.append(servidor) # adiciona primeiro socket
    print(f'servidor online na porta {PORT}...')
    while True:
        # pega a lista de todos os sockets que podem ser lidos através do select
        readable_sock,writable_sock,error_sock=select.select(all_sockets,[],[],0)

        for sock in readable_sock:
            if sock==servidor: # se for uma nova conexão
                confirmacao=0       # variável pra validar se ainda tem vaga na sala

                # lista os dados que tem no servidor
                print('Clientes:')
                print(nomes)
                print('Salas:')
                print(salas)
                for sala in salas:
                    print(f'Integrantes da sala {sala}:')
                    print(f'\t Nomes:')
                    for par in paresSalaNomes:
                        if par[0]==sala:
                            print(f'\t\t{par[1]}')

                sockdd, adress = servidor.accept()  # aceita conexão
                all_sockets.append(sockdd)          # adicionan novo socket
                print(f'Conectado em {str(adress)}')

                sockdd.send("NOME".encode('utf-8'))
                nome = sockdd.recv(1024).decode('utf-8') # recebe o nome do integrante da sala
                print(f'Nome do cliente: {nome}')

                sockdd.send("SALA".encode('utf-8'))
                sala = sockdd.recv(1024).decode('utf-8') # recebe o nome da sala

                for par in paresSalaTam:
                        # verifica se o máximo de clientes numa sala foi atingido
                        # se sim, retorna a esperar por nome do cliente e da sala
                        if par[0] == sala and par[1] > MAXCLI-1:
                            sockdd.send("salaMAX".encode('utf-8'))
                            confirmacao=1
                            break
                if confirmacao==1:
                    continue

                nomes.append(nome) # adiciona nome a lista
                paresSalaNomes.append((sala,nome)) # adiciona par sala/nome a lista

                print(f'Nome da sala: {sala}')
                if sala not in salas:
                    salas.append(sala) # adiciona a sala a lista, se não tiver já

                if (sockdd, sala) not in paresCliSala:
                    paresCliSala.append((sockdd, sala)) # adiciona par sockdde/sala a lista de pares

                check = True
                for par in paresSalaTam: # incrementa a quantidade de clientes no par sala/quantidade de clientes
                    if par[0] == sala:
                       check = False
                       i = par[1] + 1
                       paresSalaTam.remove(par)
                       paresSalaTam.append((sala, i))
                if check:
                    paresSalaTam.append((sala, 1)) # sala nova -> adiciona novo par sala/1

                global salaAtual
                salaAtual = sala # variável da sala do cliente que acabou de ser criado

                print(f'{nome} foi adicionado a lista')
                # enviar mensagem de entrada para todos os integrantes da sala
                enviaMensagens(sock,f'{nome} entrou no chat!'.encode('utf-8'))
            else: # lida com as mensagens, não uma nova conexão
                chat(sock)

    servidor.close()
```

Figura 2: Função receber.

A função chat cuida das mensagens recebidas pelo chat do cliente. Para isso, primeiro ele verifica se tem alguma mensagem no socket, depois é identificado

qual grupo do cliente que está enviando a mensagem e guardado em uma variável global, ao final é chamada a função enviaMensagens com o cliente e sua mensagem como parâmetros. Caso não haja nada no socket provavelmente a conexão foi quebrada, então, esse socket é removido da lista de sockets.

Já a função enviaMensagens, como o próprio nome já revela, é responsável por enviar a mensagem recebida para todos os integrantes da sala atual, que foi definida posteriormente.

```
101  def chat(cliente): # cuida das mensagens do chat
102      try:
103          msg = cliente.recv(1024) # recebe mensagem no socket
104          print(msg)
105          if msg: # tem alguma mensagem no socket
106              for par in paresCliSala:
107                  if par[0] == cliente: # encontra o grupo do cliente que esta enviando a msg
108                      global salaAtual
109                      salaAtual = par[1]
110                      break
111
112              enviaMensagens(cliente, msg)
113          else:   # socket quebrado
114              if cliente in all_sockets:
115                  all_sockets.remove(cliente)
116      except: # socket está desconectado
117          cliente.send("salaMAX".encode('utf-8'))
118
119  def enviaMensagens(sock, msg): # envia mensagem para todos clientes da sala atual
120      print(f'Enviando mensagens, sala {salaAtual}')
121      for socketA in all_sockets:
122          if socketA != servidor: # envia somente pro peer
123              try:
124                  if (socketA, salaAtual) in paresCliSala:
125                      socketA.send(msg)
126              except:
127                  print('Conexão quebrada!!')
128                  socketA.close()
129                  if socketA in all_sockets:
130                      all_sockets.remove(socketA)
131
132
133  receber()# inicia servidor
134
```

Figura 3: Função chat e função enviaMensagens.

3.2. Client.py

No arquivo client.py é feita toda a parte do cliente na comunicação cliente-servidor. Ele é responsável por apresentar a interface do usuário, a qual o cliente utilizará para acessar a aplicação, inserir seu nome, a sala que deseja entrar e enviar e visualizar as mensagens do chat. Na Figura 4, é possível visualizar as

primeiras linhas desse arquivo, onde há os devidos imports e as atribuições do endereço de host e a porta utilizada sendo as mesmas do servidor.

A função init da classe, que nada mais é que a função realizada em sua instanciação, é a primeira a ser executada quando usamos o comando para iniciar o cliente. Nela ocorre a criação do socket e a sua conexão ao host, depois ocorre a criação da interface gráfica com a ferramenta PySimpleGUI em que o usuário informará seu nome e a sala que quer entrar. Por fim, é feita a thread na função front e ocorre a chamada da função receber.

```python
import socket
import threading
import select
import sys
from tkinter import *
import tkinter.scrolledtext
from tkinter import simpledialog
from tkinter import messagebox

import PySimpleGUI as sg

sg.theme('SandyBeach')

# ip host e porta
HOST = '127.0.0.1'
PORT = 9001

def rgb_hack(rgb):
    return "#%02x%02x%02x" % rgb

class Cliente:

    def __init__(self, host, port):
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # cria socket
        self.socket.connect((host, port))       # conecta ao host

        # cria layout da caixa que pergunta o nome e sala
        layout = [
        [sg.Text('Digite seu nome:', size =(15, 1),font=(50), pad=(15,15)), sg.InputText(size =(20),font=(50))],
        [sg.Text('Digite a sua sala:', size =(15, 1),font=(50), pad=(15,15)), sg.InputText(size =(20),font=(50))],
        [sg.Submit('Confirmar',size =(15, 1),font=(50),pad=(15,20)), sg.Cancel('Cancelar',size =(15, 1),font=(50),pad=(15,20))]
        ]

        title='Seja Bem-Vindo!'
        window = sg.Window(title, layout,size=(400,220))
        event, values = window.read()
        window.close()

        self.nome = values [0] # Guarda o nome do cliente numa variável

        self.sala = values [1] # Guarda o nome da sala numa variavel

        self.mensagem = Label(text="")
        self.mensagem.pack()

        self.front_pronto = False
        self.running = True

        # thread que recebe mensagens no front
        front_thread = threading.Thread(target=self.front)
        front_thread.start()

        self.receber() # função que recebe as mensagens
```

Figura 4: Imports, declarações de constantes e função __init__.

Na Figura 5 temos as funções front, entrada e fecha_janela. Na primeira delas, ocorre a criação da interface da sala de bate-papo com a ferramenta TKinter. Na segunda, que é executada quando o cliente aperta no botão enviar da interface, envia a mensagem que o usuário escreveu para o servidor. A última, só fecha a janela do bate-papo, para a execução e fecha o socket.

```python
    def front(self): # front-end
        # janela
        self.win = tkinter.Tk()
        self.win.configure(bg=rgb_hack((178, 50, 126)))
        self.win.title("Bate Papo")
        self.win.option_add('*Font', '22')
        self.win.geometry("800x600")

        # label nome da sala
        self.chat_label = tkinter.Label(
        self.win, text='Nome da sala: '+self.sala, bg=rgb_hack((178, 50, 126)),height=2)
        self.chat_label.configure(font=("Arial", 15))
        self.chat_label.pack(padx=20, pady=3)

        # label membros
        self.chat_members = tkinter.Label(
        self.win, text="Membros:", bg=rgb_hack((178, 50, 126)),height=2)
        self.chat_members.configure(font=("Arial", 15))
        self.chat_members.pack(padx=20, pady=0)

        #text box dos membros
        self.chat_text_members = tkinter.scrolledtext.ScrolledText(
            self.win, width=40, height=5)
        self.chat_text_members.pack(padx=20, pady=3)

        # text box do chat das mensagens
        self.chat_text = tkinter.scrolledtext.ScrolledText(
            self.win, width=40, height=10)
        self.chat_text.pack(padx=20, pady=5)

        # label da mensagem
        self.input_label = tkinter.Label(
            self.win, text="Mensagem", bg=rgb_hack((178, 50, 126)))
        self.input_label.pack(padx=20, pady=5)

        # input da mensagem
        self.input_text = tkinter.Entry(self.win, width=40)
        self.input_text.configure(font=("Courier", 12))
        self.input_text.pack(padx=20, pady=5)

        # botao enviar
        self.send_button = tkinter.Button(
            self.win, text="Enviar", command=self.entrada)
        self.send_button.pack(padx=20, pady=5)

        self.front_pronto = True

        # fechar janela
        self.win.protocol("WM_DELETE_WINDOW", self.fecha_janela)
        self.win.mainloop()

    def entrada(self):
        mensagem = f"{self.nome}: {self.input_text.get()}"   # msg do input

        self.socket.send(mensagem.encode('utf-8'))           # envia msg

        self.input_text.delete(first=0, last='end')         # limpa input

    def fecha_janela(self):
        self.running = False    # para o loop
        self.win.destroy()      # fecha janela
        self.socket.close()     # fecha socket
        sys.exit()
```

Figura 5: Funções front, entrada e fecha_janela.

A função receber, que pode ser visualizada na Figura 6, é responsável por receber as mensagens do servidor e similarmente ao servidor, utiliza a system call

select. Essa função envia ao servidor o nome do cliente e da sala, se a sala estiver cheia a janela de bate-papo é fechada e o socket também. E por fim, mostra as mensagens e membros da sala na interface.

```
    def receber(self):
        listaDeMembros=[]
        while self.running:
            all_sockets=[sys.stdin,self.socket] # adiciona o socket a lista
            # pega a lista de todos os sockets que podem ser lidos através do select
            readable,writable,error_s=select.select(all_sockets,[],[])

            for each_sock in readable:
                if each_sock==self.socket:
                    try:
                        # recebe mensagem do servidor
                        mensagem = each_sock.recv(1024).decode('utf-8')
                        if mensagem == 'NOME': # envia nome do cliente
                            self.socket.send(self.nome.encode('utf-8'))
                        elif mensagem == 'SALA': # envia nome da sala
                            self.socket.send(self.sala.encode('utf-8'))
                        elif mensagem == 'salaMAX': # se a sala tiver cheia fecha janela
                            print("A sala está lotada. Tente outra.")
                            self.running= False
                            self.win.destroy()
                            self.socket.close()
                            sys.exit()

                        else:
                            if self.front_pronto:  # chat foi renderizado
                                # adiciona cliente que enviou mensagem ao quadro de membros,
                                # se ainda não estiver
                                membro=mensagem.split()
                                if membro[0][:-1] not in listaDeMembros and mensagem.find(":")!=-1:
                                    listaDeMembros.append(membro[0][:-1])
                                    self.chat_text_members.config(state='normal')
                                    self.chat_text_members.insert('end', membro[0][:-1] + '\n')
                                    self.chat_text_members.yview('end')
                                    self.chat_text_members.config(state='disabled')
                                # adiciona mensagem ao chat
                                self.chat_text.config(state='normal')
                                self.chat_text.insert('end', mensagem + '\n')
                                self.chat_text.yview('end')
                                self.chat_text.config(state='disabled')

                    except ConnectionAbortedError():
                        sys.exit()
                        break
                    except:
                        print("Error")
                        self.sock.close()
                        break


def inicia():
    client = Cliente(HOST, PORT)

inicia() # inicia cliente

```

Figura 6: Funções receber e inicia.

4. Conclusão

Neste relatório foi apresentado a aplicação do protocolo TCP/IP no desenvolvimento de salas virtuais de bate-papo com o uso da system call select() e sockets. O projeto aqui apresentado conseguiu atender os seguintes requisitos: Permitir o usuário entrar/criar/sair de uma sala com o nome dela com limite participantes, possuir identificação através de nome, enviar e mensagens dos participantes da sala; Aplicação construída em Python; O diálogo entre cliente e servidor deve ser feito usando a System call select() (lib do Python). Também foi desenvolvida uma interface gráfica para melhorar a interação dos clientes com a aplicação. Uma limitação é que os membros só se tornam visíveis na caixa de membros após enviarem mensagens.

5. Experiência da equipe

- **Pedro Henrique:** O trabalho final da disciplina de Fundamentos de Redes de Computadores foi importante para aprimorar os conceitos imprescindíveis pela qual a disciplina é proposta. O projeto foi importante para mim no sentido de conseguir ter um entendimento melhor do protocolo TCP/IP, tendo uma melhor compreensão de como funciona a camada de redes e de aplicação. Foi possível também ter um melhor entendimento de conceitos relativos à linguagem de programação Python, além de aprimorar os conceitos relativos ao processo de implementação de um TCP Server-Client. Como melhoria para o projeto, pode-se citar o fato de que os membros de uma sala só se mostram visíveis (por meio da interface) aos outros membros da sala quando enviam alguma mensagem no chat e não ao entrarem nessa sala. Uma melhoria que sugiro para a aplicação dessa atividade em semestres posteriores é que ela seja acordada com os alunos com maior antecedência, tendo assim mais tempo para a sua realização, uma vez que atividade foi colocada no mesmo período de realização de outras atividades importantes da disciplina (prova e entrega de laboratórios). A participação foi distribuída de maneira igualitária entre os dois integrantes do grupo, sendo que os dois participaram ativamente em todas as etapas de realização da atividade.
- **Victor Hugo:** É sempre uma ótima experiência colocar em prática o conteúdo da disciplina, pois é possível aprender bastante. E nesse projeto não foi diferente, consegui entender melhor como funciona o protocolo TCP/IP, sockets e a system call select, além de trabalhar com a linguagem Python, com a qual tive pouco contato anteriormente. Como melhoria ao projeto aqui apresentado seria interessante melhorar a forma que os membros são adicionados à interface e adicionar uma forma de autenticação do usuário para criação de chats privados. Agora, uma melhoria para a disciplina seria ter um prazo maior para realizar o projeto final, pois foi dado apenas uma semana e no mesmo período que ocorreu a prova 2, o que dificultou bastante. Quanto à participação, foi igual, visto que sempre que possível foi realizada reuniões virtuais síncronas, onde ocorria programação em pares,

além de sempre utilizar o whatsapp para troca de mensagens sobre o trabalho, então a nota deveria ser distribuída igualmente, 10/10 para os dois integrantes .

6. Referências

NETWORK PROGRAMMING II - CHAT SERVER & CLIENT. BogoToBogo. Disponível em: <https://www.bogotobogo.com/python/python_network_programming_tcp_server_client_chat_server_chat_client_select.php> Acessado em: 24 de set de 2022.

NEURALNINE. Simple GUI Chat in Python. YouTube, 2020. Disponível em: <https://www.youtube.com/watch?v=sopNW98CRag>. Acessado em: 23 de set de 2022.