CALCULADORA UDP

CHAT TCP

AMBIENTE INTELIGENTE

DISTRIBUÍDOS - TRABALHO 2

- Luís Gustavo 418210
- Iury Rosal 422067
- Vinícius Almeida 413129



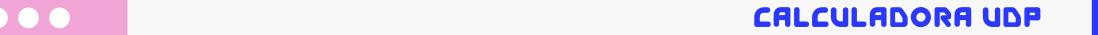


CALCULADORA UDP

- Linguagem utilizada: Python
- Comunicação UDP entre cliente e servidor
- Bibliotecas utilizadas:
 - Socket

DISTRIBUÍDOS - TRABALHO 2

Conexão ao servidor via UDP



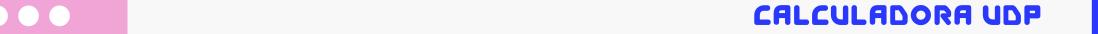
```
from socket import *
import socket
server_name = 'Localhost'
server = socket.gethostbyname(socket.gethostname())
server_port = 12456
ADDR = (server, server_port)
FORMAT = 'utf-8'
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
client_socket.connect(ADDR)
print("Conexão Sucedida!")
```

Input do cálculo
desejado pelo usuário
e envio desse calculo
para o servidor, além
do recebimento da
resposta

CALCULADORA UDP

```
while 1:
15
16
         number1 = input('Input number1: ')
         number2 = input('Input number2: ')
         operator = input('Input operator (+, -, *, /): ')
18
19
28
         if number1.isnumeric() and number2.isnumeric():
             client_socket.sendto(number1.encode(FORMAT), ADDR)
22
             client_socket.sendto(number2.encode(FORMAT), ADDR)
             client_socket.sendto(operator.encode(FORMAT), ADDR)
             answer, clientAddress = client_socket.recvfrom(2048)
             answer_decoded = answer.decode(FORMAT)
             print('Result of Operation: ', answer_decoded)
29
             again = input('Do you want to do another operation? \n 1 - Yes. \n 0 - No. \n')
30
             client_socket.sendto(again.encode(FORMAT), ADDR)
32
             if int(again) == 0:
33
                 break
34
         else:
35
             print("Insert a number, please.")
     client_socket.close()
```

Conexão ao cliente via UDP



```
import socket
     from calculator import calculate
     server_name = 'Localhost'
     server = socket.gethostbyname(socket.gethostname())
     server_port = 12456
     ADDR = (server, server_port)
    FORMAT = 'utf-8'
     server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
     server_socket.bind(ADDR)
    print('The server is connected')
     print('Traget IP: ', server_name)
14
     print('Target Port:', server_port)
    print('\n')
```

Recebimento da
mensagem do cliente,
execução do calculo
com auxilio da
Calculator.py e envio
do resultado para o
cliente

CALCULADORA UDP

```
while 1:
         number1, number2, operator = 0, 0, 0
19
21
         number1, clientAddress = server_socket.recvfrom(2048)
22
         number1_decoded = float(number1.decode(FORMAT))
         print('Number1 received: ', number1 decoded)
25
         number2, clientAddress = server_socket.recvfrom(2048)
         number2_decoded = float(number2.decode(FORMAT))
         print('Number2 received: ', number2_decoded)
28
29
         operator, clientAddress = server socket.recvfrom(2048)
30
         operator_decoded = operator.decode(FORMAT)
         print('Operator received: ', operator_decoded)
32
         print('Calculating...')
34
         result = calculate(number1_decoded, number2_decoded, operator_decoded)
         if result != None:
36
             answer = str(result).encode(FORMAT)
         else:
             msg = "Operação Inválida"
             answer = msg.encode(FORMAT)
41
         server_socket.sendto(answer, clientAddress)
42
         again, clientAddress = server_socket.recvfrom(2048)
         again_decoded = int(again.decode(FORMAT))
         print('Again? ', again_decoded)
         if again_decoded == 0:
             break
     server_socket.close()
```

CALCULATOR.PY

Execução dos cálculos recebidos pelo servidor de forma desacoplada



```
def calculate(number1, number2, operator):
         if operator == '+':
             result = number1 + number2
             return result
         elif operator == '-':
             result = number1 - number2
             return result
10
         elif operator == '*':
11
             result = number1 * number2
12
             return result
14
         elif operator == "/":
             result = number1 / number2
16
             return result
18
         else:
19
20
             return None
```





CHAT TCP

- Linguagem utilizada: Python
- Comunicação TCP entre cliente e servidor
- Suporte a comandos
- Bibliotecas utilizadas:
 - Socket
 - ∘ Threading

DISTRIBUÍDOS - TRABALHO 2

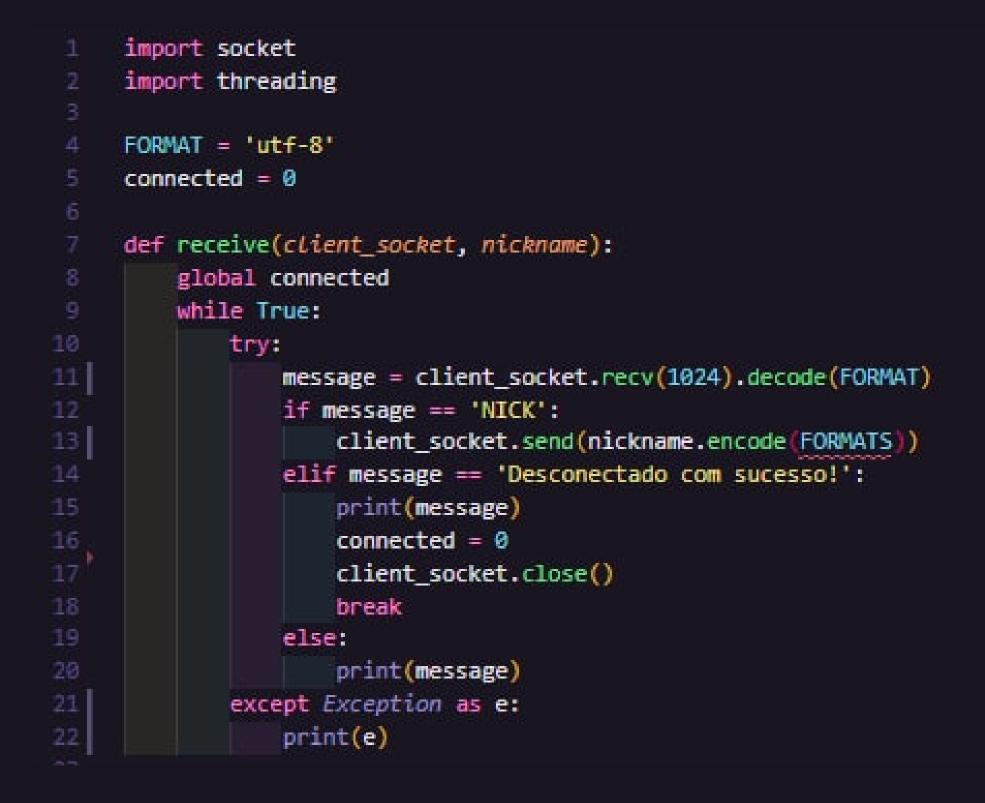
Conexão ao servidor via TCP e implementação do "/ENTRAR"



```
if connected == 0:
         input i = input('')
41
         if input_i == '/ENTRAR':
             print("Para realizar a conexão, solicitamos o IP, porta do servidor e o nickname.")
42
43
             server = input("IP do Servidor: ")
44
             port = input("Porta do Servidor: ")
             nickname = input("Choose your nickname: ")
             ADDR = (server, int(port))
             try:
                 client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                 client socket.connect(ADDR)
                 print("Servidor Conectado!")
                 connected = 1
                 receive_thread = threading.Thread(target=receive, args=(client_socket, nickname))
                 receive_thread.start()
                 write_thread = threading.Thread(target=write, args=(client_socket, nickname))
                 write_thread.start()
             except Exception as e:
61
                 print(e)
```

Implementação da função de recebimento de mensagens





Implementação da função de envio de mensagens



Inicialização do servidor



```
import socket
     import threading
     server_name = 'Localhost'
     server = socket.gethostbyname(socket.gethostname())
     server_port = 12456
     ADDR = (server, server_port)
     FORMAT = 'utf-8'
     server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
     server_socket.bind(ADDR)
     server_socket.listen()
13
     print("Servidor ligado!")
15
     print("IP DO SERVIDOR: ", ADDR)
16
     clients = []
     nicknames = []
```

Conexão aos clientes



```
59
     def start_connection():
60
         while True:
61
             client, address = server_socket.accept()
62
             print("Connected with {}".format(str(address)))
64
             client.send('NICK'.encode(FORMAT))
             nickname = client.recv(1024).decode(FORMAT)
             nicknames.append(nickname)
67
             clients.append(client)
             print("Nickname is {}".format(nickname))
             broadcast("{} joined!".format(nickname).encode(FORMAT))
             client.send('Connected to server!'.encode(FORMAT))
             thread = threading.Thread(target=handle, args=(client,))
             thread.start()
     start_connection()
```

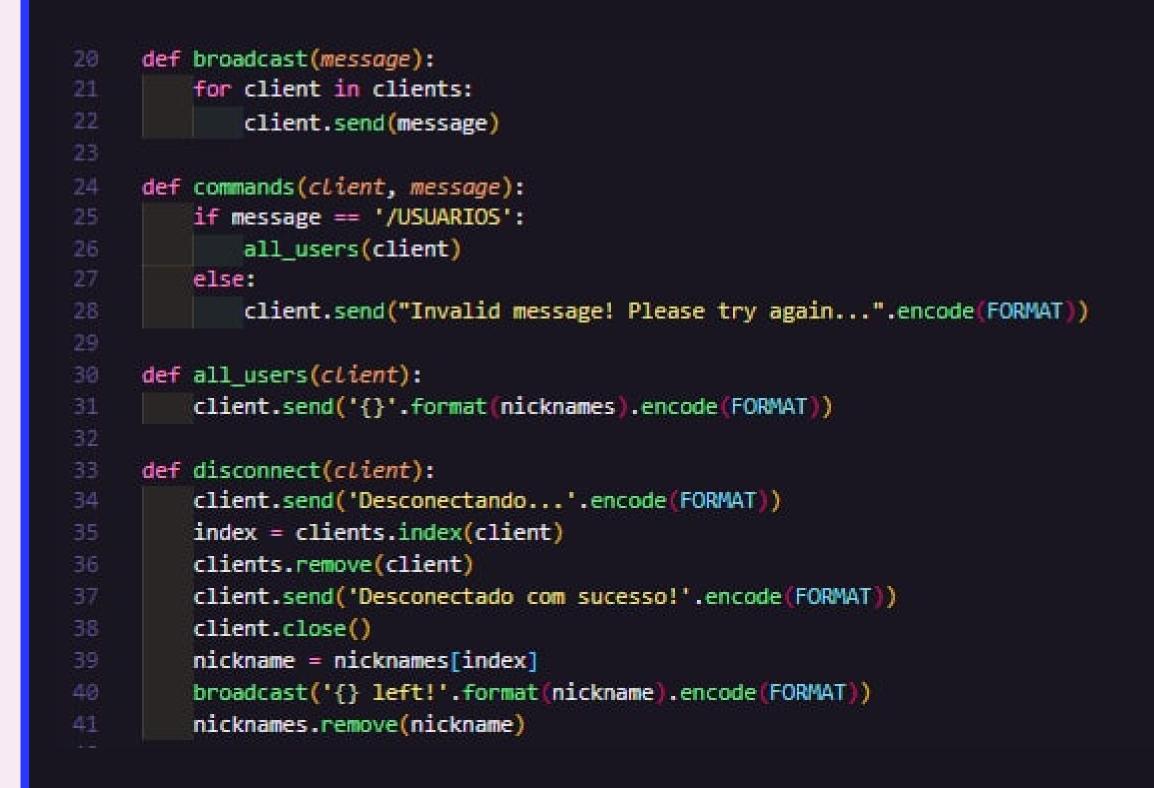
Recebimento de mensagens dos clientes e processamento da resposta



```
def handle(client):
43
         while True:
45
             try:
46
                 message = client.recv(1024)
47
                 message_decoded = message.decode(FORMAT)
48
                 if message_decoded[0] == '/':
                     if message_decoded == '/SAIR':
                         disconnect(client)
50
51
                         break
                     commands(client, message_decoded)
                 else:
54
                     broadcast(message)
55
             except:
56
                 disconnect(client)
57
                 break
```

Funções para processamento das mensagens e envio da resposta







- Linguagem utilizada: Python
- Uso de multicast para identificação de novos objetos
- Comunicação TCP entre objetos e gateway
- Comunicação TCP como uso de protocol buffers entre aplicação e gateway
- Suporte a comandos
- Bibliotecas utilizadas:
 - Socket
 - Threading
 - ∘ Time
 - Struct
 - Protocol Buffers

OBJETOS

- LÂMPADA (LAMP.PY)
 - STATUS: LIGADO / DESLIGADO
 - ENVIO DO STATUS QUANDO SOLICITADO
- BORRIFADOR (SPRINKLER.PY)
 - STATUS: LIGADO / DESLIGADO
 - ENVIO DO STATUS QUANDO SOLICITADO
- AR CONDICIONADO (AC.PY)
 - STATUS: LIGADO / DESLIGADO
 - TEMPERATURA
 - ENVIO PERIODICO DE SUA TEMPERATURA E STATUS

LÓGICA MULTICAST

FUNCIONALIDADE DE DESCOBERTA DE EQUIPAMENTOS INTELIGENTES, USANDO COMUNICAÇÃO EM GRUPO (MULTICAST).

- GATEWAY ENVIA O IP E PORTA PARA OS OBJETOS
- OBJETOS RECEBEM ADDR DO GATEWAY PARA REALIZAR CONEXÃO

TCP POSTERIORMENTE

RECEIVE_MULTICAST _GROUP.PY

RECEBIMENTO DE

MENSAGEM VIA MULTICAST

(FUNÇÃO QUE É HERDADA

PELOS CLIENTES,

OBJETOS)

```
import socket
     import struct
     import sys
     def receive_multicast():
         multicast group = '224.3.29.71'
         server_address = ('', 10000)
         sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
12
         group = socket.inet aton(multicast group)
         mreq = struct.pack('4sL', group, socket.INADDR_ANY)
         sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEPORT, 1)
         sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, mreq)
         sock.bind(server_address)
19
         while True:
             print('\nWaiting to receive message by multicast\n')
             data, address = sock.recvfrom(1024)
             print('Received %s bytes from %s\n' % (len(data), address))
24
             if data != None:
                 return data
```

```
Função dentro do
Client responsável
pelo recebimento do
endereço do gateway
(IP, PORT).
```

Os objetos herdam a classe Client.



```
from multicast.receive_multicast_group import receive_multicast
     class Client:
         FORMAT = "utf-8"
10
         def get_addr_by_mult(self):
11
             addr = receive_multicast().decode(Client.FORMAT)
12
             # Tratando o fortmato da mensagem
13
             addr = addr.split()
14
             addr[1] = int(addr[1])
15
             addr = tuple(addr)
16
             return addr
```

SEND_MULTICAST_GR OUP.PY

ENVIO DE MENSAGEM VIA MULTICAST (FUNÇÃO QUE É HERDADA PELO GATEWAY)

```
import socket
     import struct
     import sys
     def send_multicast(message):
         multicast_group = ('224.3.29.71', 10000)
         sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
         sock.settimeout(0.2)
10
11
12
13
         ttl = struct.pack('b', 1)
         sock.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL, ttl)
14
15
         try:
             print('Sending by multicast "%s"\n' %message)
             message = message.encode('UTF-8')
18
             sock.sendto(message, multicast_group)
19
20
         finally:
21
             print('Closing multicast socket\n')
22
23
             sock.close()
```

GATEWAY.PY

Montagem e envio da mensagem contendo o ADDR do próprio Gateway para o Client via Multicast



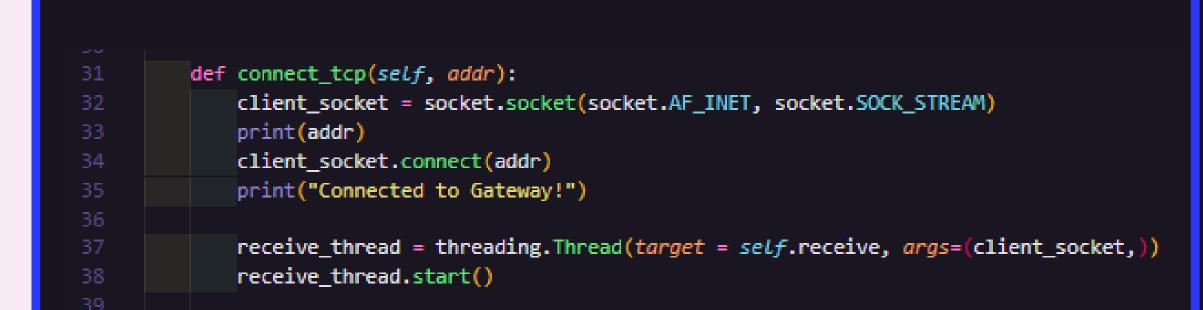
```
27  def send_gateway_address():
28  message = f"{IP} {PORT}"
29  send_multicast(message)
```

CONEXÃO TCP

REALIZADA APÓS TODOS OS OBJETOS TEREM SIDO DESCOBERTOS
PELO GATEWAY, ESTES TEREM RECEBIDO SEU ADDR. OS OBJETOS
ENVIAM O SEU TIPO, IP E PORTA AO GATEWAY DURANTE A
CONEXÃO TCP.

Função que ao inserir o ADDR do Gateway realiza a conexão com o mesmo via TCP.





return client_socket

GATEWAY.PY

Função que abre a conexão TCP do Gateway. Logo, em seguida, função que realiza a conexão TCP com os objetos e a aplicação

```
20
      def start_server():
            server_tcp_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
           server_tcp_socket.bind(ADDR)
22
           server_tcp_socket.listen()
           print("Gateway On!\n")
25
           return server_tcp_socket
     def connect_client_by_tcp(server_tcp_socket):
        print("Waiting TCP's connections...\n")
        while True:
            try:
                client, address = server_tcp_socket.accept()
                if address == ADDR_APP:
                   socket app.append(client)
                   application_thread = threading.Thread(target=application_handle, args=(client,))
                   application_thread.start()
                else:
                   client_type = client.recv(1024).decode(FORMAT)
                   clients_types.append(client_type)
                   clients.append(client)
                   print("Connected to {}\n".format(str(address)))
                   thread = threading.Thread(target=handle, args=(client,))
                   thread.start()
            except Exception as e:
                print(e)
```

APPLICATION.PY

Ao ligar a aplicação, esta já realiza a conexão com o gateway via TCP.



```
# conecta via tcp com o servidor
106
107
      client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
      client_socket.bind(ADDR_APP)
108
      client_socket.connect(GATEWAY_ADDR)
109
      print("### Connected with Gateway")
111
112
113
      receive_thread = threading.Thread(target=receive, args=(client_socket, ))
      receive_thread.start()
115
      print("### Listening gateway's answer")
116
117
      main(client_socket)
```

FORMATO DO PROTOCOL BUFFERS

Formato das mensagens que serão "trocadas" entre Application.py e Gateway.py

```
syntax = "proto3";
     message ApplicationMessage {
       enum MessageType {
           IDENTIFICATION = 0;
           COMMAND = 1;
       MessageType type = 1;
       string command = 2;
       string args = 3;
13
     message Object {
       string address = 1;
       string type = 2;
       string status = 3;
       int32 temp = 4;
20
     message GatewayMessage {
22
       enum MessageType {
           UPDATE = 0;
           GET = 1;
           LIST = 2;
       MessageType response_type = 1;
       repeated Object object = 2;
```

APPLICATION.PY

Main com a interface de usuário, onde digita os comandos que serão enviados ao gateway.



GATEWAY.PY

Função que realiza a "conversa" entre Application e Gateway.

```
119
      def application_handle(client):
120
          while True:
121
              try:
122
                  print("Waiting application's messages")
123
                  message = client.recv(1024)
                  message_decoded = messages_pb2.ApplicationMessage()
124
                  message_decoded.ParseFromString(message)
125
126
127
                  if message_decoded.type == 1:
128
                      if message_decoded.command == 'list_objects':
                          return_list_object(client)
129
                      elif message decoded.command == 'request status':
131
                           request_object_status(client, message_decoded.args)
                      elif message_decoded.command == 'set_status':
132
133
                           set_object_status(client, message_decoded.args)
                      elif message_decoded.command == 'set_attributes':
135
                          set_object_attributes(client, message_decoded.args)
136
                      else:
137
                          pass
              except Exception as e:
                  print(e)
139
140
                  client.close()
141
                  break
```

GATEWAY.PY

```
Função que realiza a
"conversa" entre
Gateway e Objetos.
Dependendo da mensagem
enviada pelo objeto,
essa função realiza o
envio para a
Application.py (linha
57)
```

```
def handle(client):
    while True:
        try:
            answer = messages_pb2.GatewayMessage()
            answer.response_type = messages_pb2.GatewayMessage.MessageType.GET
            message = client.recv(1024)
            message_decoded = message.decode(FORMAT)
            if message_decoded.split()[0] == 'acinfo':
                global ac info
                ac_info = f"AC {message_decoded.split()[1]} {message_decoded.split()[2]}"
                print(ac info)
            elif message decoded.split()[0] == 'lampinfo' or message decoded.split()[0] == 'sprinklerinfo':
                info = message decoded
                print(info)
                iobject = answer.object.add()
                iobject.type = message_decoded.split()[0]
                iobject.status = info.split()[1]
                answer_serialized = answer.SerializeToString()
                socket_app[0].send(answer_serialized)
            else:
                answer = 'Return of gateway: You are connected by TCP\n'
                client.send(answer.encode(FORMAT))
        except Exception as e:
            print(e)
            client.close()
            break
```

OBRIGADO PELA ATENÇÃO!!

DISTRIBUÍDOS - TRABALHO 2

• LUÍS GUSTAVO - 418210 EMAIL: gustavo.castro97@alu.ufc.br

• IURY ROSAL - 422067 EMAIL: iuryrosal@alu.ufc.br

• VINÍCIUS ALMEIDA - 413129 EMAIL: viniciusAC@alu.ufc.br