

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

# CLIENT.PY

Conexão ao servidor  
via UDP

```
1  from socket import *
2  import socket
3
4  server_name = 'localhost'
5  server = socket.gethostname(socket.gethostname())
6  server_port = 12456
7  ADDR = (server, server_port)
8  FORMAT = 'utf-8'
9
10 client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
11 client_socket.connect(ADDR)
12
13 print("Conexão Sucedida!")
14
```

# CLIENT.PY

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

```
15 while 1:
16     number1 = input('Input number1: ')
17     number2 = input('Input number2: ')
18     operator = input('Input operator (+, -, *, /): ')
19
20     if number1.isnumeric() and number2.isnumeric():
21         client_socket.sendto(number1.encode(FORMAT), ADDR)
22         client_socket.sendto(number2.encode(FORMAT), ADDR)
23         client_socket.sendto(operator.encode(FORMAT), ADDR)
24
25         answer, clientAddress = client_socket.recvfrom(2048)
26         answer_decoded = answer.decode(FORMAT)
27         print('Result of Operation: ', answer_decoded)
28
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)
31
32         if int(again) == 0:
33             break
34         else:
35             print("Insert a number, please.")
36
37 client_socket.close()
```

# SERVER.PY

Conexão ao cliente via  
UDP

```
1  import socket
2  from calculator import calculate
3
4  server_name = 'Localhost'
5  server = socket.gethostbyname(socket.gethostname())
6  server_port = 12456
7  ADDR = (server, server_port)
8  FORMAT = 'utf-8'
9
10 server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
11 server_socket.bind(ADDR)
12
13 print('The server is connected')
14 print('Traget IP: ', server_name)
15 print('Target Port:', server_port)
16 print('\n')
```

# SERVER.PY

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

```
18 while 1:
19     number1, number2, operator = 0, 0, 0
20
21     number1, clientAddress = server_socket.recvfrom(2048)
22     number1_decoded = float(number1.decode(FORMAT))
23     print('Number1 received: ', number1_decoded)
24
25     number2, clientAddress = server_socket.recvfrom(2048)
26     number2_decoded = float(number2.decode(FORMAT))
27     print('Number2 received: ', number2_decoded)
28
29     operator, clientAddress = server_socket.recvfrom(2048)
30     operator_decoded = operator.decode(FORMAT)
31     print('Operator received: ', operator_decoded)
32
33     print('Calculating...')
34     result = calculate(number1_decoded, number2_decoded, operator_decoded)
35     if result != None:
36         answer = str(result).encode(FORMAT)
37     else:
38         msg = "Operação Inválida"
39         answer = msg.encode(FORMAT)
40
41     server_socket.sendto(answer, clientAddress)
42
43     again, clientAddress = server_socket.recvfrom(2048)
44     again_decoded = int(again.decode(FORMAT))
45     print('Again? ', again_decoded)
46
47     if again_decoded == 0:
48         break
49
50 server_socket.close()
```

# CALCULATOR.PY

Execução dos cálculos  
recebidos pelo servidor  
de forma desacoplada

```
1
2  def calculate(number1, number2, operator):
3      if operator == '+':
4          result = number1 + number2
5          return result
6
7      elif operator == '-':
8          result = number1 - number2
9          return result
10
11     elif operator == '*':
12         result = number1 * number2
13         return result
14
15     elif operator == '/':
16         result = number1 / number2
17         return result
18
19     else:
20         return None
```



# CHAT TCP

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



# CLIENT.PY

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

```
39 if connected == 0:
40     input_i = input('')
41     if input_i == '/ENTRAR':
42         print("Para realizar a conexão, solicitamos o IP, porta do servidor e o nickname.")
43         server = input("IP do Servidor: ")
44         port = input("Porta do Servidor: ")
45         nickname = input("Choose your nickname: ")
46
47         ADDR = (server, int(port))
48         try:
49             client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
50             client_socket.connect(ADDR)
51             print("Servidor Conectado!")
52
53             connected = 1
54             receive_thread = threading.Thread(target=receive, args=(client_socket, nickname))
55             receive_thread.start()
56
57             write_thread = threading.Thread(target=write, args=(client_socket, nickname))
58             write_thread.start()
59
60         except Exception as e:
61             print(e)
```

# CLIENT.PY

Implementação da  
função de recebimento  
de mensagens

```
1  import socket
2  import threading
3
4  FORMAT = 'utf-8'
5  connected = 0
6
7  def receive(client_socket, nickname):
8      global connected
9      while True:
10         try:
11             message = client_socket.recv(1024).decode(FORMAT)
12             if message == 'NICK':
13                 client_socket.send(nickname.encode(FORMAT))
14             elif message == 'Desconectado com sucesso!':
15                 print(message)
16                 connected = 0
17                 client_socket.close()
18                 break
19             else:
20                 print(message)
21         except Exception as e:
22             print(e)
```

# CLIENT.PY

Implementação da  
função de envio de  
mensagens

```
def write(client_socket, nickname):  
    global connected  
    while True:  
        try:  
            if connected == 1:  
                text = input('')  
                if text[0] == '/':  
                    client_socket.send(text.encode(FORMAT))  
                else:  
                    message = '{}: {}'.format(nickname, text)  
                    client_socket.send(message.encode(FORMAT))  
            except Exception as e:  
                print(e)
```

# SERVER.PY

Inicialização do  
servidor

```
1  import socket
2  import threading
3
4  server_name = 'Localhost'
5  server = socket.gethostbyname(socket.gethostname())
6  server_port = 12456
7  ADDR = (server, server_port)
8  FORMAT = 'utf-8'
9
10 server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
11 server_socket.bind(ADDR)
12 server_socket.listen()
13
14 print("Servidor ligado!")
15 print("IP DO SERVIDOR: ", ADDR)
16
17 clients = []
18 nicknames = []
```

# SERVER.PY

Conexão aos clientes

```
--
59 | def start_connection():
60 |     while True:
61 |         client, address = server_socket.accept()
62 |         print("Connected with {}".format(str(address)))
63 |
64 |         client.send('NICK'.encode(FORMAT))
65 |         nickname = client.recv(1024).decode(FORMAT)
66 |         nicknames.append(nickname)
67 |         clients.append(client)
68 |
69 |         print("Nickname is {}".format(nickname))
70 |         broadcast("{} joined!".format(nickname).encode(FORMAT))
71 |         client.send('Connected to server!'.encode(FORMAT))
72 |
73 |         thread = threading.Thread(target=handle, args=(client,))
74 |         thread.start()
75 |
76 |     start_connection()
--
```

# SERVER.PY

Recebimento de  
mensagens dos clientes  
e processamento da  
resposta

```
43 def handle(client):
44     while True:
45         try:
46             message = client.recv(1024)
47             message_decoded = message.decode(FORMAT)
48             if message_decoded[0] == '/':
49                 if message_decoded == '/SAIR':
50                     disconnect(client)
51                     break
52                 commands(client, message_decoded)
53             else:
54                 broadcast(message)
55         except:
56             disconnect(client)
57         break
58
```



# SERVER.PY

Funções para  
processamento das  
mensagens e envio da  
resposta

```
20 def broadcast(message):
21     for client in clients:
22         client.send(message)
23
24 def commands(client, message):
25     if message == '/USUARIOS':
26         all_users(client)
27     else:
28         client.send("Invalid message! Please try again...".encode(FORMAT))
29
30 def all_users(client):
31     client.send('{}'.format(nicknames).encode(FORMAT))
32
33 def disconnect(client):
34     client.send('Desconectando...'.encode(FORMAT))
35     index = clients.index(client)
36     clients.remove(client)
37     client.send('Desconectado com sucesso!'.encode(FORMAT))
38     client.close()
39     nickname = nicknames[index]
40     broadcast('{} left!'.format(nickname).encode(FORMAT))
41     nicknames.remove(nickname)
```



# AMBIENTE INTELIGENTE

- 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)

```
1  import socket
2  import struct
3  import sys
4
5  def receive_multicast():
6      multicast_group = '224.3.29.71'
7      server_address = ('', 10000)
8
9      sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
10
11     # Solicita ao sistema operacional para adicionar o socket ao grupo multicast em todas as interfaces
12     group = socket.inet_aton(multicast_group)
13     mreq = struct.pack('4sL', group, socket.INADDR_ANY)
14     sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEPORT, 1)
15     sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, mreq)
16
17     sock.bind(server_address)
18
19     while True:
20         print('\nWaiting to receive message by multicast\n')
21         data, address = sock.recvfrom(1024)
22
23         print('Received %s bytes from %s\n' % (len(data), address))
24
25         if data != None:
26             return data
```

## CLIENT.PY

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

Os objetos herdam a classe Client.

```
5  from multicast.receive_multicast_group import receive_multicast
6
7  class Client:
8      FORMAT = "utf-8"
9
10     def get_addr_by_mult(self):
11         addr = receive_multicast().decode(Client.FORMAT)
12         # Tratando o formato da mensagem
13         addr = addr.split()
14         addr[1] = int(addr[1])
15         addr = tuple(addr)
16     return addr
17
```



## SEND\_MULTICAST\_GROUP.PY

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

```
1  import socket
2  import struct
3  import sys
4
5  def send_multicast(message):
6      multicast_group = ('224.3.29.71', 10000)
7
8      sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
9
10     sock.settimeout(0.2)
11
12     # Seta o time-to-live da mensagem para 1
13     ttl = struct.pack('b', 1)
14     sock.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL, ttl)
15
16     try:
17         print('Sending by multicast "%s"\n' % message)
18         message = message.encode('UTF-8')
19         sock.sendto(message, multicast_group)
20
21     finally:
22         print('Closing multicast socket\n')
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)
30
```

# 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.

## CLIENT.PY

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

```
30
31     def connect_tcp(self, addr):
32         client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
33         print(addr)
34         client_socket.connect(addr)
35         print("Connected to Gateway!")
36
37         receive_thread = threading.Thread(target = self.receive, args=(client_socket,))
38         receive_thread.start()
39
40     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():
21     server_tcp_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
22     server_tcp_socket.bind(ADDR)
23     server_tcp_socket.listen()
24     print("Gateway On!\n")
25     return server_tcp_socket
--

143 def connect_client_by_tcp(server_tcp_socket):
144     print("Waiting TCP's connections...\n")
145
146     while True:
147         try:
148             client, address = server_tcp_socket.accept()
149
150             # encaminhamento para a thread que irá lidar com as requisições da aplicação ou para a thread dos objetos
151             if address == ADDR_APP:
152                 socket_app.append(client)
153                 application_thread = threading.Thread(target=application_handle, args=(client,))
154                 application_thread.start()
155             else:
156                 # registra os clientes antes de iniciar a thread
157                 client_type = client.recv(1024).decode(FORMAT)
158
159                 clients_types.append(client_type)
160                 clients.append(client)
161
162                 print("Connected to {}\n".format(str(address)))
163
164                 thread = threading.Thread(target=handle, args=(client,))
165                 thread.start()
166         except Exception as e:
167             print(e)
168
```

## APPLICATION.PY

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

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



## FORMATO DO PROTOCOL BUFFERS

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

```
1  syntax = "proto3";
2
3  message ApplicationMessage {
4      enum MessageType {
5          IDENTIFICATION = 0;
6          COMMAND = 1;
7      }
8
9      MessageType type = 1;
10     string command = 2;
11     string args = 3;
12 }
13
14 message Object {
15     string address = 1;
16     string type = 2;
17     string status = 3;
18     int32 temp = 4;
19 }
20
21 message GatewayMessage {
22     enum MessageType {
23         UPDATE = 0;
24         GET = 1;
25         LIST = 2;
26     }
27
28     MessageType response_type = 1;
29     repeated Object object = 2;
30 }
```

## APPLICATION.PY

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

```
87 def main(client_socket):
88     while True:
89         time.sleep(1)
90         command = input('\nWrite a command:')
91         command_split = command.split()
92         try:
93             if command_split[0] == 'request_list':
94                 request_list_objects(client_socket)
95             elif command_split[0] == 'request_status':
96                 request_object_status(client_socket, command_split[1])
97             elif command_split[0] == 'set_status':
98                 set_object_status(client_socket, f"{command_split[1]} {command_split[2]}")
99             elif command_split[0] == 'set_attribute':
100                 set_object_attributes(client_socket, f"{command_split[1]} {command_split[2]} {command_split[3]}")
101             else:
102                 print('Invalid Command!')
103         except:
104             print('Invalid Command!')
105
```

## 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)
124             message_decoded = messages_pb2.ApplicationMessage()
125             message_decoded.ParseFromString(message)
126
127             if message_decoded.type == 1:
128                 if message_decoded.command == 'list_objects':
129                     return_list_object(client)
130                 elif message_decoded.command == 'request_status':
131                     request_object_status(client, message_decoded.args)
132                 elif message_decoded.command == 'set_status':
133                     set_object_status(client, message_decoded.args)
134                 elif message_decoded.command == 'set_attributes':
135                     set_object_attributes(client, message_decoded.args)
136                 else:
137                     pass
138             except Exception as e:
139                 print(e)
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)

```
--
34 def handle(client):
35     while True:
36         try:
37             answer = messages_pb2.GatewayMessage()
38             answer.response_type = messages_pb2.GatewayMessage.MessageType.GET
39
40             message = client.recv(1024)
41             message_decoded = message.decode(FORMAT)
42
43             if message_decoded.split()[0] == 'acinfo':
44                 global ac_info
45                 ac_info = f"AC {message_decoded.split()[1]} {message_decoded.split()[2]}"
46                 print(ac_info)
47
48             elif message_decoded.split()[0] == 'lampinfo' or message_decoded.split()[0] == 'sprinklerinfo':
49                 info = message_decoded
50                 print(info)
51
52                 iobject = answer.object.add()
53                 iobject.type = message_decoded.split()[0]
54                 iobject.status = info.split()[1]
55
56                 answer_serialized = answer.SerializeToString()
57                 socket_app[0].send(answer_serialized)
58
59             else:
60                 answer = 'Return of gateway: You are connected by TCP\n'
61                 client.send(answer.encode(FORMAT))
62
63         except Exception as e:
64             print(e)
65             client.close()
66             break
```

**OBRIGADO  
PELA  
ATENÇÃO!!**

• LUÍS GUSTAVO - 418210  
EMAIL: [gustavo.castro97@alu.ufc.br](mailto:gustavo.castro97@alu.ufc.br)

• IURY ROSAL - 422067  
EMAIL: [iuryrosal@alu.ufc.br](mailto:iuryrosal@alu.ufc.br)

• VINÍCIUS ALMEIDA - 413129  
EMAIL: [viniciusAC@alu.ufc.br](mailto:viniciusAC@alu.ufc.br)