

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
FLUMINENSE

FELIPE MUROS

**SISTEMAS DISTRIBUÍDOS: Estudo sobre a implementação de Sockets  
utilizando o protocolo UDP através da linguagem Python**

Relatório apresentado ao Instituto Federal de  
Educação, Ciência e Tecnologia Fluminense  
como parte das exigências para a conclusão da  
disciplina de Sistema Distribuídos do Curso  
Bacharelado em Sistemas de Informação.

Professora: Msc. [Maria Alciléia Alves Rocha](#)

Campos dos Goytacazes, RJ

Março, 2022

## SUMÁRIO

<b>1. INTRODUÇÃO</b>	<b>3</b>
1.1 Objetivo	5
1.2 Recursos utilizados	5
<b>2. IMPLEMENTAÇÃO</b>	<b>7</b>
2.1 Cliente	7
2.2 Servidor	8
<b>3. RESULTADOS</b>	<b>10</b>
<b>REFERÊNCIAS</b>	<b>12</b>

## ÍNDICE DE FIGURAS

Figura 1: Divisão lógica e física de comunicação dentro da pilha TCP/IP. Fonte (Panzuta, 2022) ..	3
Figura 2: Especificações do dispositivo.....	5
Figura 3: Especificações do Windows.....	6
Figura 4: Versão PyCharm.....	6
Figura 5: Terminal do cliente.....	10
Figura 6: Terminal do Servidor.....	11

## 1. INTRODUÇÃO

Segundo Panzuta (2022) “...os sockets foram criados na forma de uma API que possibilita aplicações/processos se comunicarem”. Considerando essa informação e o ambiente de desenvolvimento considerado ser Windows 10 é necessário conhecer que de acordo com Microsoft (2022) O uso de sockets no Windows permite que programadores criem aplicativos avançados de Internet, intranet e outros aplicativos com capacidade de rede para transmitir dados de aplicativos pela rede, independentemente do protocolo de rede usado.

O socket então é uma API disponível para diversos sistema operacionais e que opera na camada de rede para o envio e recebimento de dados através de um cliente e um servido. A Figura 1 mostra a localização do socket em uma pilha TCP/IP.

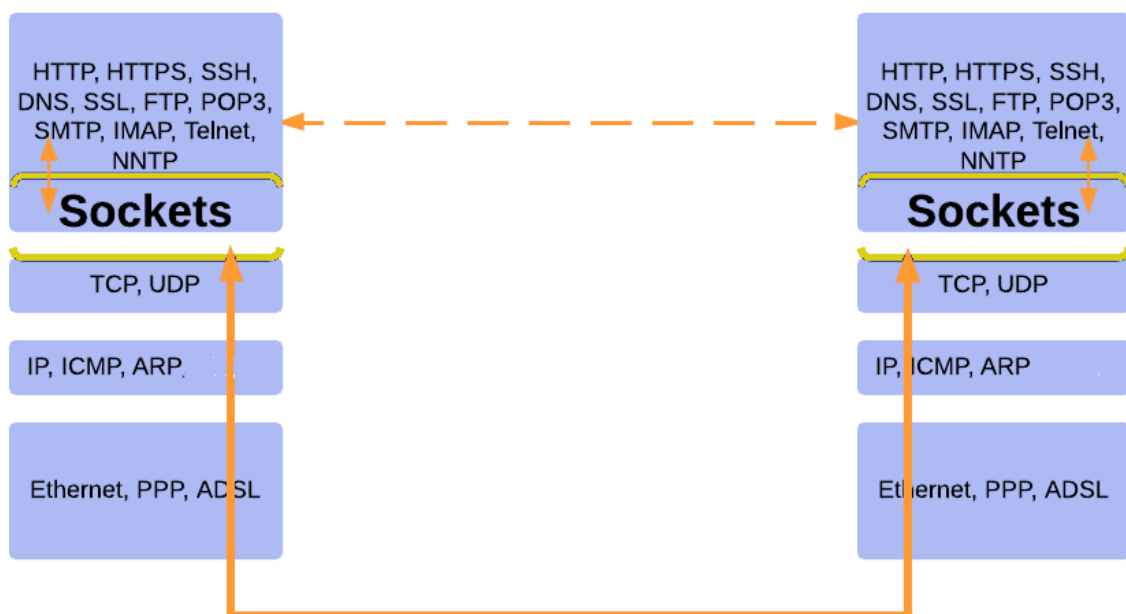


Figura 1: Divisão lógica e física de comunicação dentro da pilha TCP/IP. Fonte (Panzuta, 2022)

Dada a definição de API para o socket, é importante conhecer os principais funções na programação utilizando sockets, Panzuta (2022) lista algumas funções e suas breve descrições:

- `getaddrinfo()` // Traduz nomes para endereços sockets
- `socket()` // Cria um socket e retorna o descritor de arquivo
- `bind()` // Associa o socket a um endereço socket e uma porta
- `connect()` // Tenta estabelecer uma conexão com um socket
- `listen()` // Coloca o socket para aguardar conexões
- `accept()` // Aceita uma nova conexão e cria um socket
- `send()` // caso conectado, transmite mensagens ao socket
- `recv()` // recebe as mensagens através do socket
- `close()` // desaloca o descritor de arquivo
- `shutdown()` // desabilita a comunicação do socket

É importante ter acesso à documentação da biblioteca utilizada para facilitar a programação. Especificamente para a programação em Python a biblioteca é disponibilizada por Python (2022) que define a documentação como “Este módulo fornece acesso à interface do soquete BSD. Está disponível em todos os sistemas Unix modernos, Windows, MacOS e provavelmente em plataformas adicionais.”.

Ainda no site da linguagem Python, encontra-se McMillan (2022) que disponibilizou um artigo com os passos básicos para implementação de sockets em Python e define seu artigo da seguinte maneira: “não é essencialmente um tutorial - ainda terás o trabalho para tornar tudo operacional.”

Tendo conhecimento introdutório sobre socket e conhecimento de linguagem de programação Python, é possível criar um programa para testar a conectividade utilizando socket.

## 1.1 Objetivo

O objetivo desse trabalho é pesquisar sobre a comunicação entre processos e desenvolver um programa para estabelecer a comunicação entre dois processos utilizando os sockets com base na linguagem de programação Python.

Os resultados obtidos serão apresentados, bem como o código fonte desenvolvido com comentários para facilitar o entendimento do leitor.

Não faz parte do escopo deste trabalho a implementação de Orientação a objetos, e sim uma programação estruturada simples.

## 1.2 Recursos utilizados

Computador tipo desktop:

- Intel Core I7 12700F;
- 16GB RAM DDR 4 3200MHz;
- SSD Raid 0 1Tb,
- HDD Raid 10 500Gb 5400RPM;
- HDD 500Gb 5400RPM
- Geforce GTX 1080 8GB VRAM;

Conforme Figura 2.

Especificações do dispositivo	
Nome do dispositivo	DESKTOP-LHMP76G
Processador	11th Gen Intel(R) Core(TM) i7-11700F @ 2.50GHz 2.50 GHz
RAM instalada	16,0 GB (utilizável: 15,9 GB)
ID do dispositivo	BEA1ED94-3D17-4EB3-AED8-A1F8E6C7AAED
ID do Produto	00330-80000-00000-AA291
Tipo de sistema	Sistema operacional de 64 bits, processador baseado em x64
Caneta e toque	Nenhuma entrada à caneta ou por toque disponível para este vídeo

Figura 2: Especificações do dispositivo

Especificações da versão do Windows conforme Figura 3.

A screenshot of the Windows System Information window. The title bar is dark blue with the text 'Especificações do Windows' in white. The background is white. The table has two columns: 'Propriedade' and 'Valor'.

Edição	Windows 10 Pro
Versão	21H2
Instalado em	09/02/2022
Compilação do SO	19044.1586
Experiência	Windows Feature Experience Pack 120.2212.4170.0

Figura 3: Especificações do Windows

Programado na linguagem Python na versão: Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] utilizando a IDE PyCharm, versão conforme Figura 4

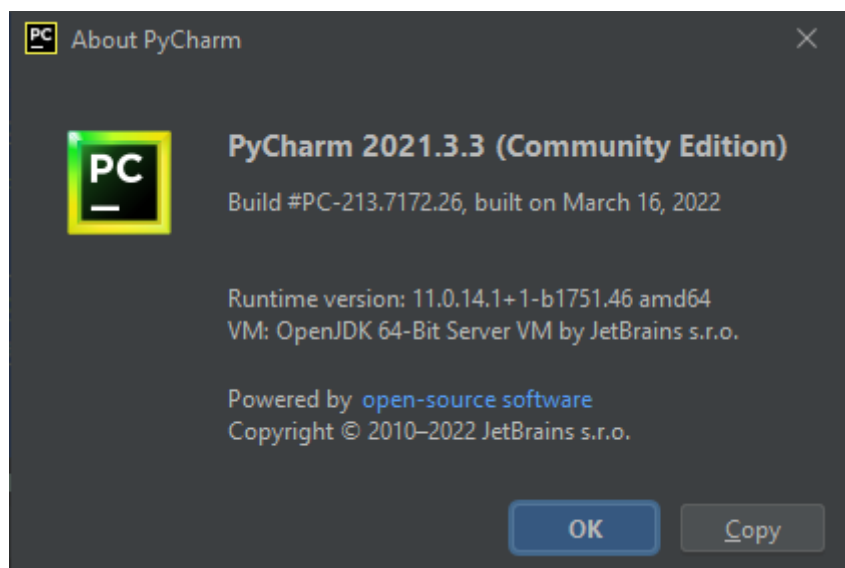


Figura 4: Versão PyCharm

## 2. IMPLEMENTAÇÃO

O código foi comentado para facilitar o entendimento, desta maneira foi dispensado um texto explicativo específico. O que se repete em cliente e servidor, foi comentado apenas no código do programa do cliente. No entanto é importante destacar o uso de duas portas no cliente e no servidor para testar a conversa entre as instâncias utilizando uma porta para receber e uma porta para enviar dados.

### 2.1 Cliente

```
import socket #biblioteca para trabalhar com socket
import datetime #biblioteca para trabalhar com data e hora

UDP_IP = "127.0.0.1" #constante para o endereço de ip do servidor de destino
UDP_PORT_SEND = 5005 #constante para a porta de envio de dados
UDP_PORT_REC = 5010 #constante para a porta de recebimento de dados

#instancia das variáveis de socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #socket.AF_INET
identifica que estamos trabalhando Internet Protocol (IP)
sock.bind((UDP_IP, UDP_PORT_REC)) # Associa o socket a um endereço socket e uma
porta
sock.settimeout(10) #tempo limite para receber a resposta do fornecedor = 10
segundos

#O servidor roda um programa que retorna a data e hora na máquina em que está
rodando
print ("\nSolicitando programa ao servidor...")
#envia ao servidor a string de autenticação a letra b no inicio identifica que
a string foi codificada
sock.sendto(b"login", (UDP_IP, UDP_PORT_SEND))

try: #bloco try para tratar exceção caso não receba resposta do servidor no
tempo determinado
    while True:
        data, addr = sock.recvfrom(1024) #ativa o listener aguardando o servidor
enviar as opções do menu
        if data: #caso tenha recebido algo do servidor o programa segue
            menu = input(data.decode()) #exibido o menu recebido do servidor
após decodificar e guarda a resposta do usuário na variável menu
            sock.sendto(menu.encode(), (UDP_IP, UDP_PORT_SEND)) #envia ao
servidor a resposta, novamente codificada
            while True:
                data, addr = sock.recvfrom(1024) #ativa o listener aguardando o
servidor enviar a resposta
                if data.decode().find("Conexao encerrada") != -1: #se o retorno
conter Conexão encerrada, o programa finaliza do lado do cliente
                    now = datetime.datetime.now()
                    print(data.decode(), "às ", now.strftime("%H:%M:%S"))
#Decodifica e imprime a mensagem final do servidor
```

```

        sock.close() #encerra o socket
        quit() #encerra o programa
    else: #Se a solicitação foi da hora, o servidor envia novamente
as opções ao usuário
        menu = input(data.decode()) #exibido o menu recebido do
servidor após decodificar e guarda a resposta do usuário na variável menu
        sock.sendto(menu.encode(), (UDP_IP, UDP_PORT_SEND)) #envia
ao servidor a resposta, novamente codificada
    else: #se não recebeu conteúdo em data, o programa encerra
        print ("\nFalha na conexao")
        quit()
except TimeoutError: #se não houver resposta do fornecedor no tempo estipulado o
programa encerra
    print ("\nConexão encerrada por inatividade do servidor.")
    sock.close() #encerra o socket
    quit()

```

## 2.2 Servidor

```

import socket
import datetime

UDP_IP = "127.0.0.1"
UDP_PORT_REC = 5005
UDP_PORT_SEND = 5010

#instancia das variáveis de socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((UDP_IP, UDP_PORT_REC))
sock.settimeout(10)

print("\nAguardando cliente conectar através da porta UDP: ", UDP_PORT_REC)

try:
    while True:
        data, addr = sock.recvfrom(1024) # buffer size is 1024 bytes
        if data.decode() == "login": #prossegue se receber do cliente o
identificador de login
            now = datetime.datetime.now()
            print ("\nHost logado atraves do IP: ", addr, "às",
now.strftime("%H:%M:%S")) #imprime o endereço do host conectado ao servidor
            strLogado = "\n \n ****ESCOLHA UMA OPCAO ABAIXO**** \n1 - Verificar
a hora no servidor \n2 - Encerrar a conexao. \n"
            sock.sendto(strLogado.encode(), (UDP_IP, UDP_PORT_SEND)) #envia o
menu de opções ao cliente
            while True:
                data, addr = sock.recvfrom(1024) # buffer size is 1024 bytes
                opcao = int(data.decode()) #switch case para tratar a resposta
do cliente ao menu enviado
                match opcao:
                    case 1: #Neste caso o cliente pediu para exibir a hora no
servidor. que envia uma string codificada com essa informação e novamente o menu
                        now = datetime.datetime.now()
                        print("\ndata e hora enviada ao cliente: \n", addr)
                        horaString = "\nData e hora no servidor: " +
now.strftime("%d/%m/%Y, %H:%M:%S") + "\n \n****ESCOLHA UMA OPCAO ABAIXO**** \n1

```



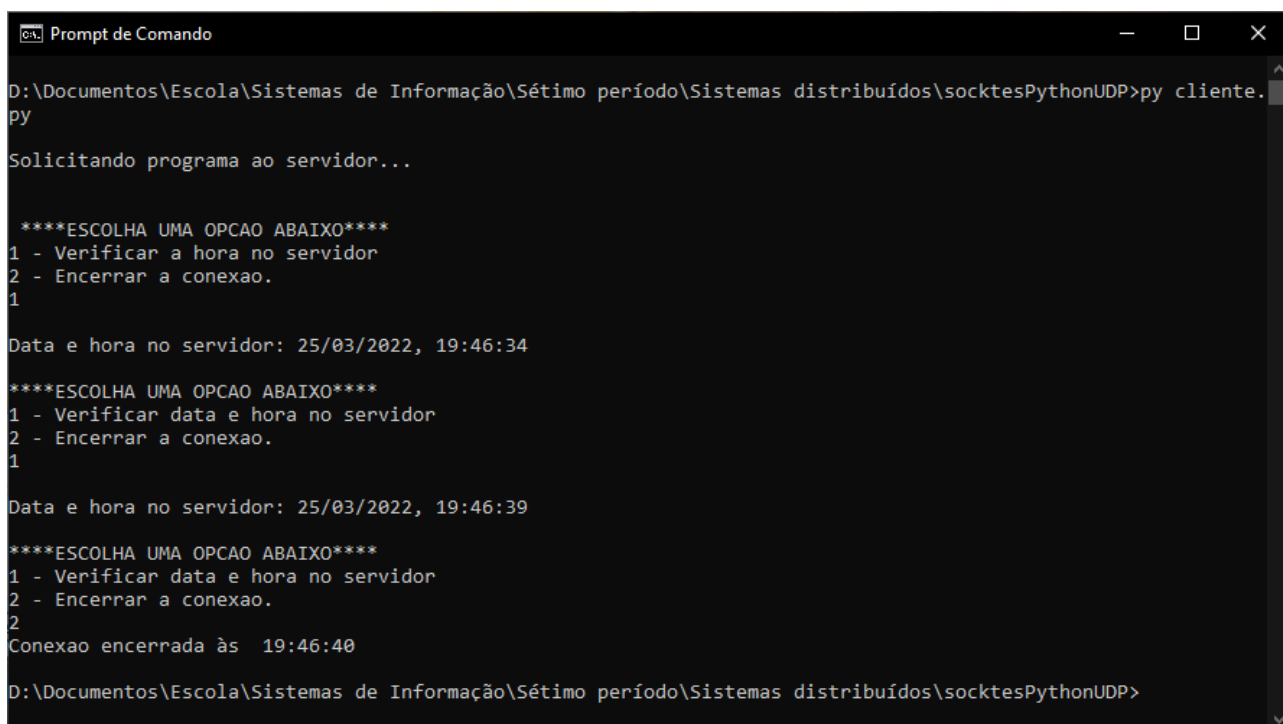
```

- Verificar data e hora no servidor \n2 - Encerrar a conexao. \n"
    sock.sendto(horaString.encode(), (UDP_IP,
UDP_PORT_SEND))
    case 2: #Neste caso o cliente escolheu encerrar a conexão, o
servidor envia a resposta e encerra a conexão
    sock.sendto(b"Conexao encerrada", (UDP_IP,
UDP_PORT_SEND))
    sock.close()
    now = datetime.datetime.now()
    print("\nConexao encerrada", "às ",
now.strftime("%H:%M:%S"))
    quit()
    case _: #Se a opção recebida for inválida o servidor
comunica e encerra a conexão
    sock.sendto(b"Opcao invalida. Conexao encerrada",
(UDP_IP, UDP_PORT_SEND))
    sock.close()
    now = datetime.datetime.now()
    print("\nConexao encerrada", "às ",
now.strftime("%H:%M:%S"))
    quit()
    else: #Encerra o programa caso não receba o pedido de login do cliente
    sock.close()
    print("Conexao encerrada")
    quit()
except TimeoutError:
    print ("Conexao encerrada por inatividade do cliente")
    sock.close()
    quit()

```

### 3. RESULTADOS

Como resultado podemos observar o funcionamento correto do programa. Como pode ser visto na Figura 5, o cliente solicitou a hora duas vezes seguidas para mostrar que o loop no servidor está funcionando, a Figura 6 mostra que o servidor identificou o cliente e enviou as respostas conforme as solicitações. Ao fim a opção de encerrar a conexão mostra o encerramento no mesmo tempo em ambos os lados (cliente e servidor).



```
Prompt de Comando

D:\Documentos\Escola\Sistemas de Informação\Sétimo período\Sistemas distribuídos\socketsPythonUDP>py cliente.py

Solicitando programa ao servidor...

****ESCOLHA UMA OPCAO ABAIXO****
1 - Verificar a hora no servidor
2 - Encerrar a conexao.
1

Data e hora no servidor: 25/03/2022, 19:46:34

****ESCOLHA UMA OPCAO ABAIXO****
1 - Verificar data e hora no servidor
2 - Encerrar a conexao.
1

Data e hora no servidor: 25/03/2022, 19:46:39

****ESCOLHA UMA OPCAO ABAIXO****
1 - Verificar data e hora no servidor
2 - Encerrar a conexao.
2

Conexao encerrada às 19:46:40

D:\Documentos\Escola\Sistemas de Informação\Sétimo período\Sistemas distribuídos\socketsPythonUDP>
```

Figura 5: Terminal do cliente

```
Prompt de Comando

D:\Documentos\Escola\Sistemas de Informação\Sétimo período\Sistemas distribuídos\socketsPythonUDP>py servidor
.py

Aguardando cliente conectar através da porta UDP: 5005

Host logado através do IP: ('127.0.0.1', 5010) às 19:46:32

data e hora enviada ao cliente:
('127.0.0.1', 5010)

data e hora enviada ao cliente:
('127.0.0.1', 5010)

Conexao encerrada às 19:46:40

D:\Documentos\Escola\Sistemas de Informação\Sétimo período\Sistemas distribuídos\socketsPythonUDP>
```

Figura 6: Terminal do Servidor

## REFERÊNCIAS

MCMILLAN G. **HOWTO sobre a Programação de Soquetes**. Disponível em: <https://docs.python.org/pt-br/3.8/howto/sockets.html>. Acesso em 24/03/2022

PANTUZA, G. **O que são e como funcionam os sockets**. Disponível em: <https://blog.pantuza.com/artigos/o-que-sao-e-como-funcionam-os-sockets>. Acesso em 25/03/2022

PHYTON. **socket — Low-level networking interface**. Disponível em: <https://docs.python.org/3/library/socket.html>. Acesso em 24/03/2022

MICROSOFT. **Windows Sockets 2**. Disponível em: <https://docs.microsoft.com/en-us/windows/win32/winsock/windows-sockets-start-page-2>. Acesso em em 25/03/2022