Relatório final ICC

SCC 0124

Instituto de Ciências Matemáticas e de Computação

São Carlos, 18 de junho de 2024

Biblioteca

Paho-MQTT

Eduardo Magaldi Magno - 15448780
Luiz Eduardo Cardoso Garcia - 15473080
Pedro da Silva Panini - 15483543

1 - Objetivos:

O objetivo deste trabalho foi de utilizar a biblioteca paho.MQTT para criação de um programa que realizasse a comunicação entre dois computadores. A biblioteca faz uso do protocolo MQTT (Message Queueing Telemetry Transport) que é um protocolo de comunicação máquina-máquina que funciona com o sistema cliente-servidor, no modelo publish-subscribe, em que uma máquina se conecta ao broker como o "publisher" (que vai enviar a mensagem) e outra máquina se conecta como "subscriber" (que vai receber a mensagem), dessa forma é feita a comunicação através da rede TCP/IP.

No programa projetado é possível enviar mensagens através do terminal de uma máquina inscrita como "publisher" e receber no terminal de outra inscrita como "subscriber", havendo um sistema que armazena as mensagens em um histórico (arquivo .txt).

2 - Introdução:

2.1 - Biblioteca Utilizada:

Após realizar pesquisas, a biblioteca escolhida foi a paho.MQTT, uma biblioteca de código aberto em python criada pela Eclipse Paho. Essa biblioteca permite que aplicativos de conectem a broker MQTT para publicar e receber mensagens dentro de tópicos específicos.

2.2 - Aplicações:

Por utilizar o protocolo MQTT, a biblioteca se mostra leve e confiável em situações que demandam a comunicação M2M (Máquina e Máquina), possibilitando uma variedade de aplicações tais quais:

- Monitoramento de sensores;
- Automatização residencial (IoT →Internet das Coisas);
- Rastreamento e monitoramento de ativos;
- Telemetria Veicular (informações sobre as condições do automóvel);
- Sistemas de Alerta;
- Sistemas de mensagens instantâneas (escolhido para o projeto).

2.3 - Principais Funções da biblioteca:

Dentre as funções presentes na biblioteca as principais e mais utilizadas durante o projeto foram:

- Conexão → usada para conectar uma máquina a um servidor, permitindo assim, a comunicação com outra máquina.
    - **connect():** Estabelece uma conexão com um broker MQTT. Parâmetros como endereço do servidor, porta, nome de usuário e senha podem ser especificados.

    - **disconnect():** Encerra a conexão com um servidor MQTT.
- Assinatura → usada para definir tópico de mensagens em que deseja-se entrar (usada como usuário no nosso código)
    - **subscribe():** Assina um tópico para receber mensagens publicadas por outros clientes.
    - **on_message():** Define uma função de callback que será chamada sempre que uma mensagem for recebida em um tópico assinado.

- Publicação → usada para enviar as mensagens dentro de um tópico selecionada (no nosso caso enviar mensagens no nome de um usuário)
    - **publish():** Publica uma mensagem em um tópico específico.
    - **loop_start():** Inicia um loop de eventos que processa mensagens recebidas e publica novas mensagens.
    - **loop_stop():** Para o loop de eventos iniciado pela função loop_start().

- Outras funções:
    - **create_client():** Cria um novo objeto cliente MQTT.
    - **get_client():** Obtém um objeto cliente MQTT existente.
    - **destroy_client()**: Destrói um objeto cliente MQTT.
    - **loop_forever():** Mantém o client em execução contínua, recebendo mensagens e chamando callbacks caso sejam definidos.

3- Problemas encontrados e soluções:

3.1 - Caminhos dos arquivos:

A primeira dificuldade encontrada no projeto era relacionada aos caminhos dos arquivos. Quando inseridos da maneira convencional, o diretório através do qual os arquivos eram acessados acabava impactando na funcionalidade do código. Para resolver isso, por indicação do monitor Victor Klann, utilizamos a biblioteca “os”, que funcionou muito bem e foi implementada no código.

3.2 - Pip no Windows:

A princípio, o projeto não funcionava corretamente no Windows, pois a instalação da biblioteca playsound sempre falhava. Pesquisando o erro, a Inteligência Artificial do Google, Gemini, nos recomendou atualizar os pacotes functools e wheels, através do comando “pip install upgrade functools wheel”. Funcionou, o que faz sentido, dado que o pacote wheels é crucial para a instalação de bibliotecas de terceiros.

4 - Código Final:

4.1 - Config:

Nessa parte foram feitas as configurações iniciais do código, que seriam usadas em todos os processos.

```python
# Importa a biblioteca paho.mqtt.client para utilizar o MQTT

import paho.mqtt.client as mqtt

# Define o endereço do servidor MQTT

host = "mqtt.eclipseprojects.io"

# Cria uma instância do cliente MQTT e conecta ao servidor MQTT

mqttc = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
mqttc.connect(host, 1883, 60)

# Define a lista dos tópicos MQTT

topicos = ["MAGNO", "LEQUE", "PERIGOSO", "GBR", "ANDRE"]
```

Imagem 4.1.1- Código das configurações. Fonte: Elaborado pelo Grupo..

4.2 - Publish:

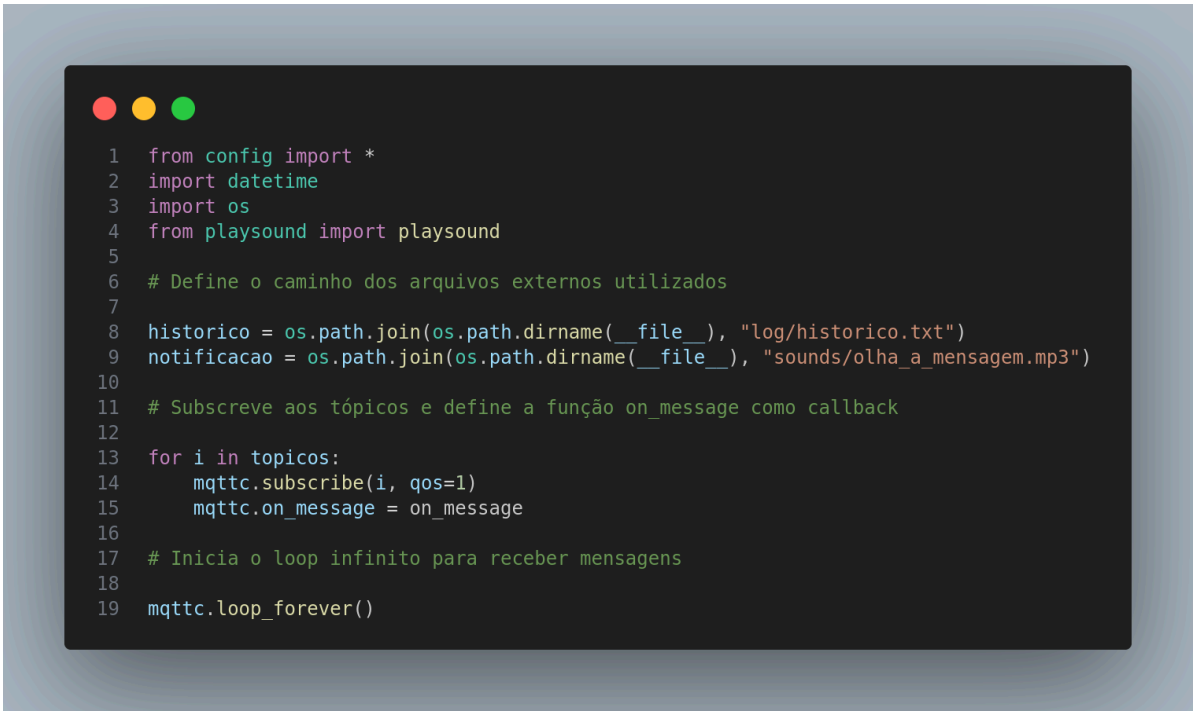
Nessa etapa foram definidas as configurações para a publicação de mensagens em tópicos determinados.

```python
from config import *

# Fornece a lista de usuários cadastrados

print(f"Lista de usuários: {topicos}")

# Solicita ao usuário que digite o seu nome de usuário

topico = input("Digite o seu usuário: ").upper()

# Verifica se o nome fornecido está presente na lista de usuários cadastrados

if topico not in topicos:
    print("[ERRO]: Usuário não encontrado")
    exit()

# Solicita ao usuário que digite a mensagem a ser publicada

mensagem = input("Digite a mensagem: ")

# Publica a mensagem no tópico correspondente ao usuário usando o protocolo MQTT com qualidade de serviço 1

mqttc.publish(topico, mensagem, qos=1)

mqttc.disconnect()
```

Imagem 3.2.1 - Código das configurações do publish. Fonte: Elaborado pelo Grupo.

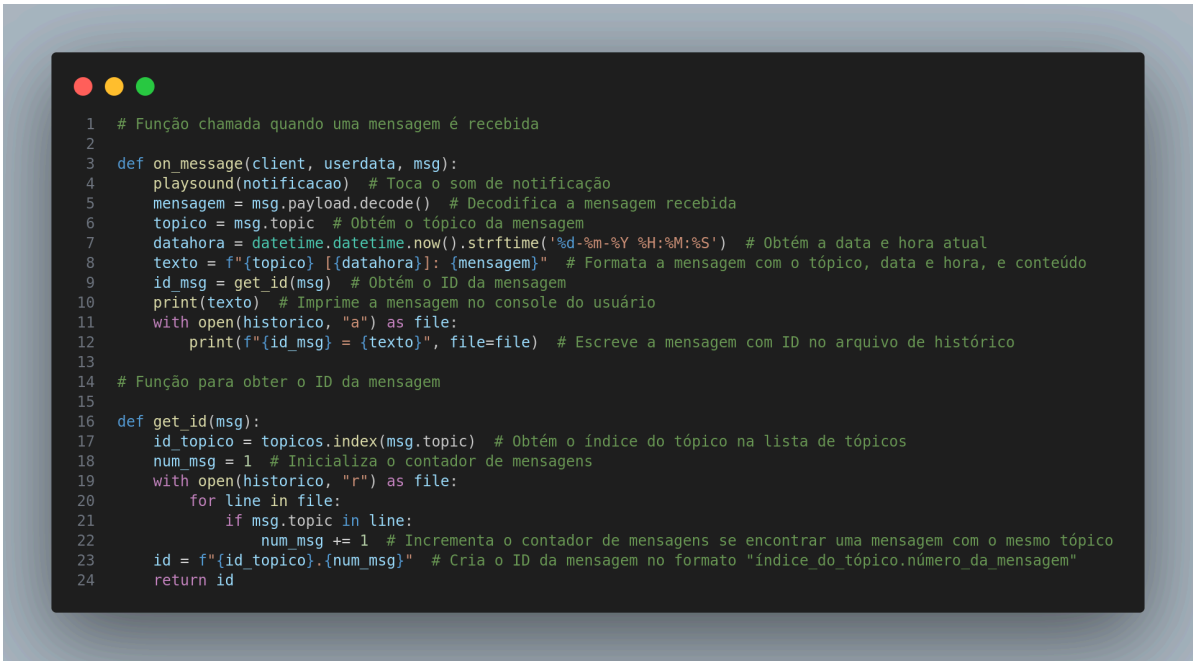
4.3 - Subscribe:

Por fim, foi configurado a recepção das mensagens enviadas no publish, contando com efeitos sonoros (biblioteca playsound) e com o uso da biblioteca datetime.



```python
from config import *
import datetime
import os
from playsound import playsound

# Define o caminho dos arquivos externos utilizados

historico = os.path.join(os.path.dirname(__file__), "log/historico.txt")
notificacao = os.path.join(os.path.dirname(__file__), "sounds/olha_a_mensagem.mp3")

# Subscreve aos tópicos e define a função on_message como callback

for i in topicos:
    mqttc.subscribe(i, qos=1)
    mqttc.on_message = on_message

# Inicia o loop infinito para receber mensagens

mqttc.loop_forever()
```

Imagem 4.3.1 - Parte 1 do código do subscribe. Fonte: Elaborado pelo Grupo.



```python
# Função chamada quando uma mensagem é recebida

def on_message(client, userdata, msg):
    playsound(notificacao)  # Toca o som de notificação
    mensagem = msg.payload.decode()  # Decodifica a mensagem recebida
    topico = msg.topic  # Obtém o tópico da mensagem
    datahora = datetime.datetime.now().strftime('%d-%m-%Y %H:%M:%S')  # Obtém a data e hora atual
    texto = f"{topico} [{datahora}]: {mensagem}"  # Formata a mensagem com o tópico, data e hora, e conteúdo
    id_msg = get_id(msg)  # Obtém o ID da mensagem
    print(texto)  # Imprime a mensagem no console do usuário
    with open(historico, "a") as file:
        print(f"{id_msg} = {texto}", file=file)  # Escreve a mensagem com ID no arquivo de histórico

# Função para obter o ID da mensagem

def get_id(msg):
    id_topico = topicos.index(msg.topic)  # Obtém o índice do tópico na lista de tópicos
    num_msg = 1  # Inicializa o contador de mensagens
    with open(historico, "r") as file:
        for line in file:
            if msg.topic in line:
                num_msg += 1  # Incrementa o contador de mensagens se encontrar uma mensagem com o mesmo tópico
    id = f"{id_topico}.{num_msg}"  # Cria o ID da mensagem no formato "índice_do_tópico.número_da_mensagem"
    return id
```

Imagem 4.3.2 - Parte 2 do código do subscribe. Fonte: Elaborado pelo Grupo.

5 - Conclusões:

Por meio desse projeto foi possível entender mais como o sistema de trocas de mensagens utilizando bibliotecas podem funcionar, além de mostrar o poder do uso de bibliotecas para facilitar o trabalho na realização de códigos.

O programa final funcionou como o esperado, recebendo mensagens e enviando-as sem demais problemas, com as funcionalidades de mostrar o tempo real em que a mensagem é recebida e ao som de notificação agindo como esperado. Muitas melhorias podem ser feitas como criar sistemas de cadastro de usuário ou chats privativos de mensagens, mas com o auxílio das funções da biblioteca paho.MQTT e de outras bibliotecas essas implementações com certeza poderão ser possíveis.

6 - Referências:

- https://pypi.org/project/paho-mqtt/
- https://eclipse.dev/paho/files/paho.mqtt.python/html/index.html
- https://docs.python.org/pt-br/3/library/os.html
- https://pypi.org/project/playsound/