ASSIGNMENT – 4 (miniProject)

SOCKET PROGRAMMING

School of Computing, National University of Singapore

This is a Group Project with group size of 4 members per group. As informed early, pls register your groups @ *Canvas->**People->A4-ProjectGroup***. We strongly recommend teams to **start early**.

You are allowed to use any tool/opensource codes to support you in this project, just acknowledge them in the report. We will be looking only for correct functionality, features and efficiency (eg. Low latency).

A parallel **research study** is conducted by TAs 'how ChatGPT/LLMS support science/engineering education'. Please keep track of all your interaction with LLMS, especially we need to know the complete chain of prompts/questions.

Learning Objectives

This assignment/project provides you a chance to learn about details of SMTP and HTTPS Protocols, OpenSSL API, Websockets API, WebRTC API, TCP/UDP Stream/Datagram sockets API and develop services using socket programming.

[Option 1] Smart Mailer Program

The smart mailer program sends mails to a list of emails provided in a ***csv*** file using any SMTP services of your choice (eg. smtp.gmail.com). The mailer should introduce some delay between emails to reduce spam mails. You mailer should establish an SMTP connection and exchange SMTP messages to send mail. You can use any programming language to implement the smart mailer (c, c++, java, python and use any high-level API (such as *openssl API*) to establish connection with the SMTP server.

Requirements

- The mailer should accept an input file (maildata.csv) with list of email ids, names, department codes
- The mailer should accept a department code and send mails only to those departments that are in the list. If the department code is "all" then the mails should be send to all email ids in the list.
- The mailer should accept subject and body information (body in HTML format) from a text file. The body should have placeholders which are marked with special characters (such as, #name#, #department#). The placeholders should be replaced by actual names and department names before sending the mail.
- The mailer sends URL of a transparent 1x1 pixel image (usually a .png or .gif file) in the HTML email body. This single image should be hosted in a publicly accessible

HTTP/HTTPS server of your choice. Provide some method for the user who is sending email to view a counter value in this server which increments each time the image is accessed. [Note: In this way the user can track number of recipients opened the mail].
- The program print a report showing number of emails sent grouped by department code.
- UI can be GUI or CUI.
- Your codes should be well written and well commented.

[Option 2] Smart Push-to-Talk (PTT) Web Application for Real-Time Class Discussions using Websockets.

The goal of this assignment is to design and implement a Push-to-Talk (PTT) web application using WebSockets. The application will allow multiple clients (students) to press a button to stream live audio (their voice) to a server (instructor's machine). This is a one-way streaming system, where the audio is streamed from the client to the server for verbal discussion in a large class. The assignment will focus on WebSockets for real-time communication, audio streaming, and handling multiple client connections in a simple, interactive way.

Requirements:

1. Client-Side (Student's Device):
    - A web-based client where the student can press a button (PTT) to start streaming their voice to the server.
    - Audio is captured from the device's microphone.
    - The audio stream is sent to the server using WebSockets.
    - Only one student can speak at a time (First-Come-First-Served basis).
2. Server-Side (Professor's Device):
    - A web-based server that receives the audio stream from connected clients.
    - The server processes and plays the audio live through the professor's computer speakers.
    - The server handles only one client connection at a time and streams audio from the client (based on who pressed the button first). For all other clients the connection can be rejected.
    - The audio may be stored (optional), and streaming stops when the button is released.
3. Concurrency Control:
    - Only one client (student) can speak at a time.
    - If multiple clients press the PTT button at the same time, the server should reject the requests until the current client which is streaming stops the stream/connection.
4. WebSocket Integration:
    - WebSockets must be used for real-time communication between clients and the server.

- Implement the connection setup, message passing, and stream handling over WebSocket protocol.

Deliverables:

- A working web-based Push-to-Talk application with both client and server components.
- A report explaining the architecture, design choices, implementation, and testing.
- A short demonstration video showcasing the working application.

Features:

- Push-to-Talk Button: Pressing and holding the button streams the audio to the server.
- Audio Streaming: Voice is captured from the client's microphone and transmitted via WebSockets.
- Concurrency Control: Only one client can stream at a time.
- Error Handling: The system should handle potential errors like multiple clients attempting to speak simultaneously.
- Optional: Implement a basic UI/UX design for the web application to make it intuitive for students to use. The UI can ask for the students name at the beginning. If the audio is stored at the server side, the student name with some additional serial number should be used as filename.

Suggested Architecture [You are not required to use the same architecture or sample codes]:

1. Client-Side Architecture:
   - Use HTML5 Web Audio API to capture microphone input.
   - Use WebSockets for streaming audio data to the server.
   - Implement a PTT button in the client interface to control the audio stream.

2. Server-Side Architecture (Option A: Python):

   - Use the `websockets` library to create a WebSocket server in Python that listens for client connections.
   - The server will maintain a queue (FCFS model) to ensure only one client can talk at a time.
   - When a client sends an audio stream, the server will play it live using `PyAudio`.

3. Server-Side Architecture (Option B: Node.JS):
   - Use Node.js with the ws WebSocket library to handle WebSocket connections.

- Use Web Audio API or native system functions to play received audio on the server side (professor's computer).
- Implement logic to ensure that only one client's audio is streamed at a time.

[Option 3] Smart Push-to-Talk (PTT) Web Application for Real-Time Class Discussions using TCP Sockets or UDP Sockets

This is same as option 2, but used TCP Sockets or UDP Sockets and Streaming protocols. Given the use case, UDP will be more efficient than TCP. However, there are pros and cons. You can choose the one which you feel easy to implement.

TCP/UDP Client (Student's Device):

- The client will capture the microphone input (using the **Web Audio API** or any platform-specific microphone input library) and send the audio data over a TCP connection to the server.
- Audio data can be encoded using a simple codec (like **PCM**, **Opus**, or **MP3**) to reduce bandwidth and then streamed over the TCP connection.

TCP/UDP Server (Professor's Device):

- The server will receive the audio stream, decode it if necessary, and play it on the system's speakers in real-time.
- The server should be capable of managing multiple connections and enforcing the rule that only one client can speak at a time (First-Come-First-Served basis). [Single threaded/process simple TCP server is sufficient] [For UDP: You can implement a **First-Come-First-Served** mechanism by maintaining a simple flag or queue on the server, ensuring only one client is allowed to stream at a time.

Recommended Codecs/Libraries.

- **GStreamer (Open Source):**
  - **GStreamer** is a powerful open-source multimedia framework that supports handling of audio and video streaming over networks using **TCP** or **UDP**. It can be integrated into both the client and server to handle audio encoding, decoding, and transmission.

  - You can create a pipeline where the audio is captured on the client-side, encoded, and streamed to the server via TCP.
  - Example usage:
  - **Client**: Captures audio, encodes it (e.g., Opus), and sends it over TCP.
  - **Server**: Receives audio stream, decodes it, and plays it back.
- **2. FFmpeg (Open Source):**
  - **FFmpeg** is another open-source library that can handle audio encoding and streaming over TCP. It supports real-time multimedia streaming and can easily be integrated into the client-server architecture for a push-to-talk application.
  - Example usage: You can use FFmpeg to capture audio input, compress it using Opus, and stream it via TCP.
  -
- **3. Opus Codec:**
  - **Opus** is a highly efficient audio codec designed for interactive audio streaming. It supports low-latency audio and works well for voice transmission. You can use Opus in combination with either FFmpeg or GStreamer to compress audio before sending it over TCP.
  - Opus is especially useful for minimizing bandwidth while maintaining good audio quality for live streaming.
  - Opus Codec Documentation

 TCP/UDP Comparison for PTT Application

- **Faster**: UDP is faster because it avoids the connection setup and teardown that TCP requires.
- **Lower Latency**: UDP is more suitable for real-time applications where some packet loss is acceptable.
- **No Retransmission**: UDP does not guarantee delivery or order, which can be beneficial in time-sensitive applications like live audio streaming.

Sample codes <Python version Client and Server using TCP and UDP> just for your reference to show the steps. <Pls refer to canvas->Files->A4 Sample Codes>

Submission:

Submission Due: **Thursday 07-Nov-2024**

- Submit **one ZIP/RAR file** (change filename to your *Group Number*) to Canvas containing **your source code, short** demonstration video showcasing the working application,

one PDF report describing the architecture, features, design choices, acknowledgements of using opensource codes and other code generation tools instructions for compiling and running your application source code. (ref. **Appendix A)**

- Peer-Review will be conducted after the due date to access individual contributions.
- Late submission penalty: **10% per day.**

If you have any question/clarification, discuss through **discord channel 'assignments**.

Bhojan Anand /NUS

'Students who approach education from a **deep-learning perspective** make significant improvements, remember what they learn, and feel empowered to make a difference. Those who take a surface approach to learning may get good grades, but they rarely benefit much in the long term'. *– from several research findings.*

Learning in depth is the key focus of CS3103. Try more beyond the questions above.

[We have a Zero Tolerance for Plagiarism Policy. If you are here only for marks/grades, just let me the grade/mark you prefer to have!]

More on deep learning approach …

Students Who Take a Deep Approach--

- *Attempt to understand material for themselves*
- *Seek rigorous and critical interaction with knowledge content*
- *Relate ideas to previous knowledge and experience*
- *Discover and use organizing principles to integrate ideas*
- *Relate evidence to conclusions*
- *Examine the logic of arguments*

Appendix A: Report Format

These formats are just general guidelines, you can include the sections which are appropriate for your project.

Mass eMail:

1. Introduction

- Brief overview of the application (purpose and key functionalities)
- Summary of the report's sections

2. Application Architecture

- **Overview**: Diagram and description of the architecture, detailing main components.
  - Client Interface: UI elements for email composition, recipient list, and submission.
  - Backend Processing: Email scheduling, sending mechanism, number of emails sent and status tracking.
- **Email Sending Process**: Explain the flow of email creation to delivery.
- **Queue/Batching Mechanism**: Describe how emails are queued/batched if applicable, to avoid server overload.

3. Features

- **Core Features**:
  - Sending bulk emails to multiple recipients
  - Support for personalized email content (using placeholders)
  - Handling unsubscribe requests (optional, not required for this assignment)

4. Design Choices

- **Technology Stack**:
  - Languages, frameworks, and libraries used (e.g., Node.js, Python, SMTP server)
- **Database Choices (is used)**:
  - Describe any database choice (e.g., MySQL for storing email logs, Redis for queueing)
- **Email Sending Strategy**:
  - Justify the choice of using an ESP or custom SMTP, batching method, and personalization strategy
- **Error Handling and Logging**:
  - Describe any error handling, retry policies, and logging mechanisms implemented.

5. Acknowledgments

- **Open-source Libraries and Tools**:
  - Mention libraries or frameworks used (e.g., Nodemailer, Flask-Mail, SMTP libraries).
- **Code Generation Tools**:
  - List any code generators or tools (e.g., OpenAI's GPT for boilerplate code).
- **Contributions**:

- Recognize external resources, code snippets, or tools that helped in development.

Push-to-Talk:

Introduction

1.1 Overview

Provide an overview of the push-to-talk web application. Briefly explain the purpose and the main functionality, such as enabling one-way, real-time audio streaming from students to the lecturer's PC using WebSockets.

1.2 Objectives

Summarize the assignment's objectives, such as implementing a WebSocket-based application to capture and transmit audio from multiple clients to a single server while enforcing a First-Come-First-Served (FCFS) rule.

2. Architecture

2.1 System Architecture Diagram

Include a high-level architecture diagram showing:

- The components (Client and Server).
- Communication flow between the client and server.
- The queue mechanism for managing multiple clients.

2.2 Architecture Explanation

Describe the role of each component:

- **Client**: Captures audio from the microphone and streams it to the server over WebSocket.
- **Server**: Receives audio, enforces the FCFS rule, and plays the audio on the lecturer's PC using PyAudio.

3. Application Features

- **Real-Time Audio Streaming**: Explain how clients can press a button to speak, streaming their voice to the server in real-time.
- **First-Come-First-Served (FCFS) Queue**: Describe the FCFS mechanism that allows only one client to speak at a time.
- **Audio Playback on Server**: Describe how the server plays the received audio immediately without storing it.

4. Design Choices

4.1 Technology Choices

- **WebSocket**: For low-latency, full-duplex communication.
- **HTML5 Audio API**: For client-side audio capture.
- **Python WebSocket Library (websockets)**: For server-side WebSocket handling.
- **PyAudio**: For real-time audio playback on the server.

4.2 Format and Compression

- **Audio Format**: Explain why you chose 16-bit PCM for audio data, balancing quality and bandwidth.

4.3 FCFS Implementation

Explain the queue structure used on the server side and the logic behind allowing only the first client in the queue to transmit audio.

5. Code and Open-Source Acknowledgements

List any open-source code, tools, or resources used, along with brief attributions, including:

- **WebSocket** library for Python.
- **PyAudio** for audio handling.
- **Any tutorial** or **sample code** for WebRTC or WebSocket examples that contributed to the development.

6. Instructions for Compiling and Running the Application

Give commands for running server and client

Testing Instructions

Explain briefly how to test the application, including:

- **Testing the FCFS mechanism by connecting multiple clients.**
- **Verifying real-time audio playback on the server.**

7. Conclusion

Summarize the outcomes of the assignment, including how well the application met the objectives and any notable challenges or solutions found in implementing the push-to-talk functionality.