Michał Byczko 160141
Łukasz Bartkowiak 160219

# Sprawozdanie sk2

# Radio Internetowe

## 1. Opis projektu

Celem projektu było stworzenie sieciowego systemu radiowego ("Radio Studenckie") działającego w architekturze klient-serwer. System umożliwia wielu użytkownikom jednoczesne słuchanie muzyki streamowanej przez serwer, prowadzenie czatu tekstowego oraz wspólne zarządzanie playlistą. Serwer pełni rolę automatycznego DJ-a, który dekoduje pliki MP3 i przesyła surowe dane audio do podłączonych klientów. Aplikacja kliencka posiada graficzny interfejs użytkownika (GUI), który pozwala na intuicyjną obsługę radia, wysyłanie wiadomości, dodawanie utworów do kolejki oraz głosowanie za pominięciem aktualnej piosenki (funkcja SKIP).

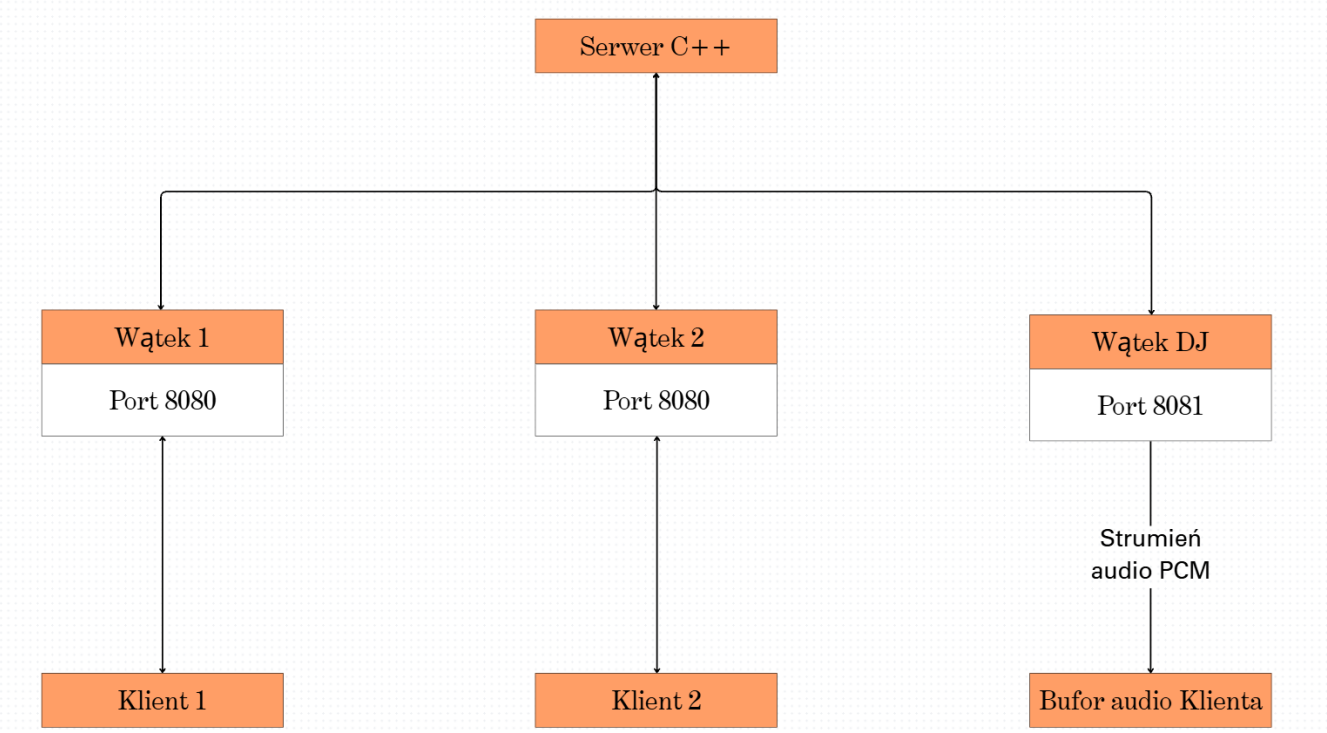
Użyta technologia: Projekt został zrealizowany w modelu hybrydowym, wykorzystującym dwa języki programowania:

- Serwer (Backend): Napisany w języku C++. Wykorzystuje niskopoziomowe gniazda sieciowe (BSD sockets) do komunikacji TCP. Do obsługi wielu klientów jednocześnie zastosowano mechanizm wielowątkowości (std::thread) oraz synchronizację wątków za pomocą muteksów (std::mutex), co zapewnia bezpieczeństwo danych przy modyfikacji playlisty. Dekodowanie plików audio (MP3 do PCM) realizowane jest poprzez potok (popen) do zewnętrznego narzędzia ffmpeg.

- Klient (Frontend): Napisany w języku Python. Interfejs graficzny zbudowano w oparciu o bibliotekę Tkinter. Odtwarzanie dźwięku realizowane jest przez bibliotekę PyAudio, a manipulacja głośnością przez moduł audioop

- Komunikacja: Protokół TCP/IP.

2. Opis komunikacji pomiędzy serwerem i klientem




1) Kanał sterowania (Port 8080): Jest to połączenie dwukierunkowe, tekstowe.
   a) Klient -> Serwer: Wysyła komendy takie jak LIST (pobranie playlisty), ADD <nazwa> (dodanie utworu), MSG <tekst> (wiadomość na czat), SKIP (głosowanie za pominięciem) oraz inicjuje przesył plików (UPLOAD).
   b) Serwer - > Klient: Odsyła potwierdzenia (OK, READY), aktualizuje tytuł utworu (CURRENT ...) oraz rozsyła wiadomości czatu do wszystkich podłączonych użytkowników (broadcast).

2) Kanał audio (port 8081): Jest to połączenie jednokierunkowe (Serwer -> Klient).
   a) Serwer dekoduje plik MP3 do formatu PCM (16-bit, 44.1kHz, stereo) i wysyła surowe bajty w sposób ciągły.
   b) Klient odbiera pakiety danych (chunks) i po krótkim buforowaniu przekazuje je bezpośrednio do karty dźwiękowej.

3. Podsumowanie

Najważniejsze informacje o implementacji:
Kluczowym elementem implementacji serwera jest wykorzystanie zewnętrznego procesu ffmpeg uruchamianego funkcją popen. Dzięki temu serwer nie musi implementować skomplikowanego dekodowania MP3 – czyta gotowe próbki dźwiękowe ze standardowego wyjścia procesu potomnego i przekazuje je do gniazd sieciowych. Ważnym aspektem logicznym jest system demokratycznego głosowania (SKIP). Serwer przechowuje unikalne identyfikatory głosujących w zbiorze (std::set). Dopiero gdy liczba głosów przekroczy 50% liczby aktywnych użytkowników, serwer bezpiecznie zamyka potok ffmpeg i ładuje kolejny utwór. Implementacja przesyłania plików (UPLOAD) wymagała stworzenia prostego protokołu typu "handshake": klient najpierw wysyła nagłówek z rozmiarem pliku, a dopiero po otrzymaniu sygnału READY od serwera rozpoczyna binarny przesył danych. Zapobiega to mieszaniu się komend tekstowych z zawartością pliku MP3.

Co sprawiło trudność:

1) Synchronizacja wątków (Race Conditions): Największym wyzwaniem było zapewnienie bezpieczeństwa wątkowego w C++. Dostęp do wektora playlist oraz listy listeners musiał być rygorystycznie chroniony przez std::mutex. Początkowo brak blokad prowadził do błędów "Segmentation Fault", gdy jeden wątek usuwał klienta, a inny próbował wysłać do niego dane.

2) Płynność odtwarzania audio: Synchronizacja prędkości wysyłania danych przez serwer z prędkością odtwarzania przez klienta była problematyczna. Zbyt szybkie wysyłanie zapychało bufor sieciowy, a zbyt wolne powodowało przerywanie dźwięku (stuttering). Rozwiązaniem było wprowadzenie krótkiego opóźnienia (sleep) w pętli serwera oraz wstępnego buforowania (pre-buffering) po stronie klienta w Pythonie.

3) Obsługa uploadu: Implementacja przesyłania plików binarnych przez socket tekstowy wymagała precyzyjnego odczytywania określonej liczby bajtów. Błędy w obliczaniu rozmiaru bufora powodowały, że fragmenty pliku MP3 były interpretowane jako błędne komendy tekstowe przez serwer.