The issue in your code appears to be related to the integration between the C++ code and the Python script used for timekeeping, which is likely causing the timestamp-related problems you're encountering.

**Audio Data Handling (onOneWayAudioRawDataReceived)**:

* The method onOneWayAudioRawDataReceived is called whenever one-way audio data is received.
* The audio data is then written to a file with the filename based on the node\_id.
* The current timestamp is captured using std::chrono::steady\_clock::now() and is printed along with debug information.
* The method then checks if the received data is from a new speaker (i.e., a different node\_id or user). If so, it logs the speaker change and updates the current speaker.

**Python Timer Integration (StartPythonTimer and GetPythonElapsedTime)**:

* StartPythonTimer() is called to start the Python timer. It runs a Python command using system() to invoke a script that starts a timer.
* GetPythonElapsedTime() is called to retrieve the elapsed time from the Python script. It uses popen() to execute a Python command that prints the elapsed time and captures the output.
* void StartPythonTimer() {
* std::string command = "python3 -c 'import timer; timer.start\_timer()'";
* int result = system(command.c\_str());
* if (result != 0) {
* std::cerr << "Error starting Python timer" << std::endl;
* }
* }
* double GetPythonElapsedTime() {
* std::string command = "python3 -c 'import timer; print(timer.get\_elapsed\_time())'";
* FILE\* pipe = popen(command.c\_str(), "r");
* if (!pipe) {
* std::cerr << "Error getting elapsed time" << std::endl;
* return 0.0;
* }
* char buffer[128];
* std::string result = "";
* while (fgets(buffer, sizeof(buffer), pipe) != nullptr) {
* result += buffer;
* }
* pclose(pipe);
* return std::stod(result);

**File Writing (writeToFile)**:

* The audio data is written to a file.
* After writing the data, the elapsed time from the Python timer is retrieved and printed for debugging.

**Issue recognized by code**

1. **Asynchronous Execution**:
   * The Python timer is started using a separate system call (system()), and the elapsed time is retrieved using another system call (popen()). These calls are independent of the C++ code and can introduce synchronization issues. There is no guarantee that the Python timer is running or that it will return accurate elapsed time when requested.
2. **Overhead of System Calls**:
   * Each time you call StartPythonTimer() and GetPythonElapsedTime(), you're invoking a new process to run Python. This adds significant overhead and delays, especially when done repeatedly in a loop, as in the writeToFile method.

(Each time you use StartPythonTimer() and GetPythonElapsedTime(), you're basically starting a new instance of Python to do a small task, like starting or checking a timer. This is like turning on a whole computer just to look at the clock—it takes extra time and resources, which slows everything down, especially when you do it over and over again in a short time.)

* + This delay could be affecting the accuracy of the timestamps and causing inconsistencies in the timing data.

1. **Timing Mismatch**:
   * The Python timer's elapsed time is likely not synchronized with the timestamps generated by std::chrono::steady\_clock::now(). This can lead to discrepancies between the times printed by the C++ code and the times retrieved from the Python timer.
   * If the Python script is not working correctly or is slow to start, the timing information will be inaccurate or outdated.
2. **Potential Deadlocks and Delays**:
   * popen() can block if the command output is large or if there are issues with the Python script. This could cause the C++ code to hang or delay, leading to further timing issues.

'[{''name'': ''Aviraj Singh'', ''avatar'': ''null'', ''username'': ''Aviraj Singh'', ''start'': [7882.74, 7882.88, 7882.9, 7882.95, 7882.96, 7883.03, 7883.14, 7883.15, 7883.16, 7883.23, 7883.25, 7883.27, 7883.29, 7883.34, 7883.41, 7883.5, 7883.51, 7883.54, 7883.57, 7883.59, 7883.6, 7883.65, 7883.68, 7883.71, 7883.75, 7883.83, 7883.88, 7883.95, 7883.97, 7884.04, 7884.18, 7884.23, 7884.27, 7884.3, 7884.46, 7884.52, 7884.64, 7884.65, 7884.66, 7884.67, 7884.68, 7884.73, 7884.86, 7884.9, 7885.02, 7885.08, 7885.15, 7885.18, 7885.22, 7885.41, 7885.44, 7885.48, 7885.49, 7885.52, 7885.62, 7885.66], ''end'': [7882.75, 7882.9, 7882.91, 7882.96, 7882.97, 7883.1, 7883.15, 7883.16, 7883.17, 7883.25, 7883.27, 7883.28, 7883.34, 7883.4, 7883.5, 7883.51, 7883.53, 7883.55, 7883.59, 7883.6, 7883.64, 7883.68, 7883.71, 7883.75, 7883.81, 7883.86, 7883.91, 7883.96, 7883.99, 7884.17, 7884.2, 7884.24, 7884.3, 7884.42, 7884.52, 7884.64, 7884.65, 7884.66, 7884.67, 7884.68, 7884.69, 7884.82, 7884.88, 7884.98, 7885.03, 7885.11, 7885.18, 7885.19, 7885.4, 7885.43, 7885.48, 7885.49, 7885.52, 7885.53, 7885.65, 7885.67], ''talk\_time'': 1.91}, {''name'': ''onkar.mahapatra@intellai.ai'', ''avatar'': ''null'', ''username'': ''onkar.mahapatra@intellai.ai'', ''start'': [7882.75, 7882.91, 7882.97, 7883.1, 7883.17, 7883.28, 7883.4, 7883.53, 7883.55, 7883.64, 7883.81, 7883.86, 7883.91, 7883.96, 7883.99, 7884.17, 7884.2, 7884.24, 7884.42, 7884.69, 7884.82, 7884.88, 7884.98, 7885.03, 7885.11, 7885.19, 7885.4, 7885.43, 7885.53, 7885.65], ''end'': [7882.88, 7882.95, 7883.03, 7883.14, 7883.23, 7883.29, 7883.41, 7883.54, 7883.57, 7883.65, 7883.83, 7883.88, 7883.95, 7883.97, 7884.04, 7884.18, 7884.23, 7884.27, 7884.46, 7884.73, 7884.86, 7884.9, 7885.02, 7885.08, 7885.15, 7885.22, 7885.41, 7885.44, 7885.62, 7885.66], ''talk\_time'': 1.02}]'

**What to do to solve this issue??**

**Remove python timer**

**Instead of python time use C++ timer.**

**Remove unnecessary system call like system(), popen()**

void ZoomSDKAudioRawDataDelegate::writeToFile(const std::string& path, AudioRawData\* data) {

    // Start the Python timer once

    StartPythonTimer();

    std::ofstream file(path, std::ios::out | std::ios::binary | std::ios::app);

    if (!file.is\_open()) {

        Log::error("Failed to open audio file path: " + path);

        return;

    }

    file.write(data->GetBuffer(), data->GetBufferLen());

    file.flush();

    file.close();

    std::stringstream ss;

    ss << "Writing " << data->GetBufferLen() << " bytes to " << path << " at " << data->GetSampleRate() << " Hz";

    Log::info(ss.str());

    // Continuously print the elapsed time

    for (int i = 0; i < 10; ++i) {// Example loop, adjust as needed

        // std::this\_thread::sleep\_for(std::chrono::seconds(1)); // Wait for 1 second

        myseconds = GetPythonElapsedTime();

        std::cout << "[DEBUG] Elapsed Time: " << myseconds << " seconds" << std::endl;

    }

}