#DOCUMENTATION:

File pair description (Client.py and Server.py):

\*client.py (Key Functionalities and Features):

#Connect to Server:

-The client script initiates a connection to the server by utilizing a pre-defined server IP address (127.0.0.1) and port (7777).

-It instantiates a client socket and establishes a connection to the server.

#File Transfer Loop:

-The script traverses through each file in the client's "client\_files" directory.

-For each file, it transmits the file's name to the server.

-If the server responds with a "file not found" message, an error message is displayed.

#Send File Data:

-The client script reads binary files from the client side and transfers them to the server in 1024-byte chunks.

-The client script transmits the file data continuously to the server until the entire file is transferred successfully.

#Close Connection:

-Once all the files are sent, the client script terminates the connection with the server.

\*server.py (Key Functionalities and Features):

#Set Up Server:

-The server script initializes a server socket, associates it with the IP address (127.0.0.1) and port (7777), and begins listening for incoming connections.

#Accept Client Connections:

-The server continuously loops, accepting incoming client connections.

-For every accepted connection, a new thread named client\_handler is spawned to independently manage the client.

#Handle Client:

-The handle\_client function is responsible for all client-related operations.

-It receives the file name from the client and constructs the corresponding file path on the server side.

#Send File Data or Error Message:

-If the file exists on the server, the server reads the file in binary mode and sends it to the client in chunks of 1024 bytes.

-If the file does not exist, the server sends an error message to the client.

#Close Client Connection:

-The server script terminates the client socket following the conclusion of all client communication.

#Main Server Loop:

-The server persistently listens for incoming connections, spawning threads to manage each client in a distinct thread.

\*\*Client.py and Server.py pair description:

These code scripts establish a basic client-server file transfer system architecture using sockets in Python, with the server storing received files in a designated directory.

\*\*

File pair description (autoClient.py and autoServer.py):

\*autoClient.py (Key Functionalities and Features):

-Incorporates sampling techniques with a probability of less than 40%, excluding the utilization of advanced features.

#Distributed Tracing Configuration:

-The script configures distributed tracing through the utilization of OpenTelemetry.

-Implements a probability-based sampler (40%) for the client configuration.

-Uses both Jaeger and Console exporters for tracing.

#File Transfer Function (receive\_file):

-Establishes a connection to the server utilizing a predefined IP address (127.0.0.1) and port (7777).

-Transmits the file name to the server.

-Defines the client output directory and generates it if it does not already exist.

-Receives file data from the server and writes it to a new file within the client directory.

-The autoClient script terminates the client socket after all file transfer has been completed.

#Main Function (main):

-Specifies the directory containing files to be transferred (client\_files\_dir).

-Iterates through each file in the directory, invoking the "receive\_file" function for each file.

\*autoServer.py (Key Functionalities and Features):

-OpenTelemetry integration with the "alwaysOn" feature.

#Distributed Tracing Setup:

-The script sets up distributed tracing using OpenTelemetry.

-Configures an always-on sampler for the server (100% sampling).

-Uses Jaeger and Console exporters for tracing.

#Client Handling Function (handle\_client):

-Accepts a client connection.

-Receives the file name from the client.

-Specifies the server directory for file storage.

-Check if the file exists and send the file data to the client if it does.

-Sends an error message if the file is not found.

-Closes the client socket after handling the client.

#Main Function (main):

-Creates a server socket and listens for incoming connections.

-Accepts client connections in a loop, creating a new thread (client\_handler) to handle each client.

-Each thread is associated with a span for distributed tracing (more illustrative info. in Jaeger Output Folder).

-The main function continuously listens for incoming connections.

\*\*autoClient.py and autoServer.py pair description:

These code scripts implement a basic client-server file transfer system architecture with integrated distributed tracing using OpenTelemetry. The system facilitates the transmission of files from a client to a server, with the server storing the received files in a designated directory. Notably, both scripts incorporate distributed tracing functionality through OpenTelemetry, enabling comprehensive monitoring and analysis of the execution flow across both the client and server components.

\*\*

File pair description (AdvancedClient.py and AdvancedServer.py):

\*AdvancedClient.py (Key Functionalities and Features):

-Incorporates the same functionalities as the "autoClient" script, enhanced with compression and chunking features.

\*AdvancedServer.py (Key Functionalities and Features):

- Incorporates the same functionalities as the "autoServer" script, enhanced with all the "Advanced Features" described later in the document.

\*\*AdvancedClient.py and AdvancedServer.py pair description:

These scripts offer flexibility for customization through the toggling of feature flags and adjustment of configuration parameters. This allows users to tailor the functionality to meet specific requirements related to file transfer and distributed tracing.

\*\*

File pair description (BuggyClient.py and BuggyServer.py):

\*BuggyClient.py (Key Functionalities and Features):

- The script builds upon the AdvancedClient, introducing a bug to simulate delays in the file transfer process.

#CSV Logging Initialization:

-Initializes a CSV file named 'execution\_data.csv' with headers to log execution data.

\*BuggyServer.py (Key Functionalities and Features):

- Incorporates the same functionalities as the "AdvancedServer" script, enhanced with all the "Advanced Features" described later in the document.

\*\*BuggyClient.py and BuggyServer.py pair description:

These scripts simulate a client-server file transfer system with distributed tracing, introducing a deliberate delay on the client side for testing purposes, and logging relevant data for analysis.

\*\*

File pair description (BugFixedClient.py and BugFixedServer.py):

\*BugFixedClient.py (Key Functionalities and Features):

- This script is similar to "BuggyClient" but with the modification that the deliberate delay percentage has been adjusted from 90% to 0%.

\*BugFixedServer.py (Key Functionalities and Features):

- Incorporates the same functionalities as the "AdvancedServer" script, enhanced with all the "Advanced Features" described later in the document.

\*\*BugFixedClient.py and BugFixedServer.py pair description:

These scripts simulate a client-server file transfer system with distributed tracing, without intentional delays on the client side. The modification from the previous script resolves the deliberate delay.

\*\*

\*\*Storage Folders

-Server\_files: The designated directory where files received from the client are stored.

-client\_files: The folder from which the client sends files to the server.

\*\*Features:

-Always On: A sampler that unconditionally provides the sampling result, irrespective of any conditions. This means that all traces are collected and sent to the tracing backend for analysis, providing a comprehensive view of the system's behaviour without the randomness introduced by other sampling methods.

-Probability Sampling: A sampler in which decisions are randomly made, influenced by a pre-configured sampling rate.

-Spans: Logical unit of work in the distributed system.

-: More information provided (in the Manual Instrumentation Server Terminal Output.txt file)

\*\*Advanced Features:

-Compression: This is a process of reducing the size of data by encoding it more efficiently. In the context of data transfer and storage, compression minimizes the amount of space needed, resulting in faster transmission and reduced storage requirements. We utilized gzip, a common compressing library.

-Chunking: This involves breaking down a large piece of data into smaller, more manageable units or chunks. In the context of data transfer, it allows for the incremental transmission of data in pieces rather than as a single continuous stream.

-Rate Limiting: This is a technique used to control the rate of incoming or outgoing requests for the server. It sets predefined limits on the number of requests that can be processed within a specific time frame.

\*\*Tools and Languages Utilized:

-Python 3.4

-Docker: //To containerize client-server applications using Docker, Dockerfiles were crafted for both the client and server components.

-Jaeger UI

-Open-Telemetry

IMPORTANT NOTICES!

- The Server\_Files directory is intentionally left empty so that, upon execution, it can be dynamically populated with contents from the client\_files directory.

- Always initiate the execution of the server scripts before running the client. A separate terminal window may be required to execute the client script. This sequence ensures that the server is ready to receive connections before the client attempts to establish communication.

- Utilized a relative path to ensure cross-device compatibility. Kindly refrain from modifying the file or folder structure, as it directly influences the overall network and file hierarchy, potentially compromising the program's effectiveness.

\*\*Docker command for Jaeger UI (Run in Terminal):-

"docker run -d --name jaeger \

-e COLLECTOR\_ZIPKIN\_HTTP\_PORT=9411 \

-p 5775:5775/udp \

-p 6831:6831/udp \

-p 6832:6832/udp \

-p 5778:5778 \

-p 16686:16686 \

-p 14268:14268 \

-p 14250:14250 \

-p 9411:9411 \

jaegertracing/all-in-one:latest"

\*web: http://localhost:16686

-Operates successfully when the Docker container, advancedClient, and advancedServer are executed concurrently.

-screenshots provided (Jaeger Output)

\*\*Docker command for Jaeger UI (Delay Bug):-

"docker run -d --name jaeger \

-e COLLECTOR\_ZIPKIN\_HTTP\_PORT=9411 \

-p 5775:5775/udp \

-p 6831:6831/udp \

-p 6832:6832/udp \

-p 5778:5778 \

-p 16686:16686 \

-p 14268:14268 \

-p 14250:14250 \

-p 9411:9411 \

jaegertracing/all-in-one:1.26"

#Bug Observational report On Word Document under Bug Documentation Folder.