CLIENT-SERVER Application

Implemented for the Network programming course in FMI, lectured during the summer semester.

SERVER:

I.Purpose

The server file (server.py) is designed to implement a server that listens for client connections, receives arrays from clients, sorts them using parallel quicksort, and returns the sorted arrays to the clients.

II.Key Components

Libraries Used

- “socket”: Provides the necessary functions for creating sockets and establishing network connections.
- “threading”: Enables the server to handle multiple client connections concurrently by creating threads.
- “json”: Facilitates the serialization and deserialization of data in JSON format.
- “parallel_quicksort”: This module contains the implementation of the parallel quicksort algorithm.

Constants

- “PORT”: Specifies the port number on which the server listens for incoming connections.
- “SERVER”: Stores the IP address of the server.
- “ADDRESS”: Represents the tuple containing the server's IP address and port number.
- “DISCONNECT_MESSAGE”: Defines the message used by clients to disconnect from the server.
- “MAXIMUM_BYTES”: Indicates the maximum number of bytes to be received from the clients.

Functions

1. “respondToClient(connection, address)”: Handles client requests. It receives arrays from clients, sorts them using the implemented parallel quicksort, and sends the sorted data back to the clients.
2. “start()”: Initializes the server by setting up the socket, binding it to the specified address, and listening for incoming connections. It creates a new thread for each client connection to handle requests concurrently.
3. “getCurrTime()”: Returns the current time in the specified format.

Execution

- To start the server, execute the “start()” function.

Client:

I.Purpose

The client file (“client.py”) is intended to establish a connection with the server, send data in the form of arrays to the server for sorting, and display the sorted arrays received from the server. Additionally, it provides an option for the user to disconnect from the server.

II.Key Components

Libraries Used

- “socket”: Utilized for creating sockets and establishing network connections.

- “json”: Enables the serialization and deserialization of data in JSON format.

Constants

- “PORT”: Specifies the port number on which the client connects to the server.

- “SERVER”: Stores the IP address of the server.

- “ADDRESS”: Represents the tuple containing the server's IP address and port number.

- “DISCONNECT_MESSAGE”: Defines the message used to disconnect from the server.

- “MAXIMUM_BYTES”: Indicates the maximum number of bytes to be received from the server.

Functions

1. “sortArrayOnServer(array)”: Sends the input array to the server for sorting and displays the sorted array received from the server.

2. “connectToServer()”: Establishes a connection with the server, prompts the user to input commands (e.g., sending arrays for sorting or disconnecting), and handles the interaction with the server accordingly.

3. readArrayFromInput()”: Prompts the user to input an array and returns it after parsing the input.

4. serverIntroduction()”: Displays an introduction message to guide the user on how to interact with the server.

Execution

- To run the client, execute the “connectToServer()” function.

Parallel Quicksort:

I.Overview

Parallel Quicksort is an algorithm that leverages the power of parallelism to efficiently sort an array of elements. Quicksort is a well-known sorting algorithm that follows the divide-and-conquer approach, which makes it amenable to parallelization.

This paragraph covers the implementation of Parallel Quicksort in Python using threads and queues to achieve parallelism.

II.Implementation Details

parallelQuicksort (arr, queue)

- This function implements the parallel Quicksort algorithm.

Parameters:

- "arr" (list): The array to be sorted.
- "queue" (Queue): A queue object used to pass sorted sub-arrays back to the main thread.

Returns:

- The sorted sub-array is put into the queue.

Description:

- The function recursively sorts the input array "arr" in parallel using threads.
- It partitions the array into three parts: elements less than the pivot, elements equal to the pivot, and elements greater than the pivot.
- Threads are created to sort the left and right sub-arrays.
- The sorted sub-arrays are put into their respective queues.

Example Usage:

- Live test

Notes:

- This implementation uses threads and queues for parallelism. Threads are created to sort the left and right sub-arrays, and queues are used to pass the sorted sub-arrays back to the main thread.
- This implementation can improve performance on multi-core systems.
- Care should be taken when using this implementation with large arrays, as the number of threads created may impact performance and resource utilization.