Documentation

The project involves creating a Centralized Computing System (CCS) server and a client application for testing. The CCS server supports service discovery, client communication, and periodic reporting of statistics.

Detailed Functionalities

Service Discovery

Purpose:

Allows clients to discover the CCS server in a local network using UDP protocol.

Implementation:

- A broadcast mechanism is employed, allowing any client in the local network to send discovery messages.
- The server listens on a specified UDP port for incoming discovery messages (CCS DISCOVER).
- Upon receiving a valid discovery message, it responds with CCS FOUND.
- Handles invalid discovery messages gracefully by ignoring them.

Client Communication

Purpose:

Facilitates computation requests and responses between the client and server using the TCP protocol.

Implementation:

- The server listens on the same port for TCP connections.
- Each client connection is handled in a separate thread, enabling multiple clients to communicate with the server concurrently.
- The client sends requests in the format <OPER> <ARG1> <ARG2>, where:
  1. OPER is the operation type (ADD, SUB, MUL, DIV).
  2. ARG1 and ARG2 are integer operands.
- The server validates the request and performs the operation or returns ERROR for invalid requests.
- Supports any number of concurrent clients using a thread pool (*Thread Pool*).
- Handles edge cases such as division by zero, invalid input, and disconnected clients.

Statistics Reporting

Purpose:

Maintains and periodically displays every 10 seconds statistics about server activity for last 10 seconds and for all time of working.

Implementation:

- Enum *Stats* created to categorize statistics. Categories include:
    1. *CONNECTED_CLIENTS:* Number of connected clients
    2. *COMPUTED_REQESTS:* Total computed requests.
        - Count of specific operations (*ADD, SUB, MUL, DIV*).
        - *INCORRECT_OPERATIONS*: Number of invalid operations.
        - *SUM_OF_RESULTS*: Sum of results from all computations.
- Every 10 seconds, the server prints:
    1. **Global statistics**: Metrics since the server started.
    2. **Last 10 seconds statistics.**
- A dedicated thread scheduled using ScheduledExecutorService manages periodic reporting.

Error Handling

The solution incorporates error handling mechanisms to ensure server stability:

1. Service Discovery:
    - Ignores malformed or invalid discovery messages.
    - Logs errors related to network communication.
2. Client Communication:
    - Handles client disconnections gracefully by closing sockets and releasing resources (connected clients).
    - Validates client requests for format and content before processing.
    - Catches and logs exceptions during computation, such as ArithmeticException for division by zero.

Difficulties:

1. Problem: If the scheduler thread is blocked or delayed due to system load or resource contention, the statistics may not be reported exactly every 10 seconds.

    Solution: Use a dedicated thread pool for the scheduler to avoid contention with client handlers.

Known Issues:

- None observed during testing in CCS.

- Running on different machines, there is an issue with finding the broadcast address in Client.

Method Descriptions:

Class: CCS

1. *CCS(int port)*

   Constructor to initialize the CCS server with the specified port and set up data structures for statistics.

   Functionality:

   - Initializes the globalStats and lastStats for tracking statistics.
   - ScheduledExecutorService for periodic statistics reporting.

2. *void start()*

   Starts the CCS server by launching threads for UDP listener, TCP server, and statistics reporting.

   Functionality:

   - Creates a thread for UDP discovery.
   - Creates a thread for handling TCP connections.
   - Schedules periodic statistics reporting every 10 seconds.

3. *void startUDPListener()*

   Listens for service discovery requests on a UDP socket.

   Functionality:

   - Receives discovery messages (*CCS DISCOVER*).
   - Sends a response message (*CCS FOUND*) to the sender's address and port.
   - Handles invalid messages.

4. *void startTCPServer()*

   Listens for client connections on a TCP socket.

   Functionality:

   - Accepts incoming client connections.
   - Creates a thread for each connected client using the thread pool.
   - Tracks the number of connected clients as part of statistics.

5. *void handleClient(Socket clientSocket)*

   Handles communication with a connected client.

   Functionality:

   - Receives computation requests from the client.
   - Validates and processes the requests.
   - Sends the computation results or error messages back to the client.
   - Updates statistics for valid and invalid requests.
   - Closes the client socket when communication ends and change stats for connected users.

6. *int performOperation(String operation, int arg1, int arg2)*

   Performs the requested arithmetic operation on the given arguments.

   Functionality:

   - Performs the requested operation (addition, subtraction, multiplication, division).
   - Throws an exception for invalid operations or division by zero.
   - Return the result of the operation (integer)
   - Updates operation-specific statistics.

7. *void reportStatistics()*

   Reports global and last 10-second statistics to the console.

   Functionality:

   - Prints global statistics since server's start.
   - Prints statistics for the last 10-second interval.
   - Resets last-interval statistics after reporting.

8. *void changeStats(Stats stat, int value)*

   Modifies a specific statistic by the given value.

   Functionality:

   - Updates the global and last-interval statistics safely.
   - Prevents negative values for *CONNECTED_CLIENTS*.

9. *void incrementStats(Stats stat)*

Increments a specific statistic by 1.

Functionality: Calls *changeStats()* with a value of 1 for the given statistic.

Class: Client

1. *Client(int port)*

   Constructor to initialize the client with the specified port.

2. *void discoverService()*

   Sends a UDP broadcast to discover the CCS server in the local network.

   Functionality:

   - Gets the broadcast address
   - Sends a *CCS DISCOVER* broadcast message.
   - Waits for a response (*CCS FOUND*) from the server.
   - Extracts the server's IP address from the response.
   - Initiates a TCP connection to the server.

3. *void connectToServer(InetAddress serverAddress)*

   Establishes a TCP connection with the CCS server and sends computation requests.

   Functionality:

   - Connects to the server's TCP port.
   - Sends 50 random computation requests (valid and invalid)
   - Prints the server's responses to the console.
   - Handles random intervals between requests

4. *InetAddress getBroadcastAddress()*

   Finds and returns the broadcast address for the local network interface.

   Functionality:

   - Retrieves the local IP address using Inet4Address.getLocalHost.
   - Finds the network interface associated with the local IP address.
   - Iterates through the interface’s addresses to find an IPv4 broadcast address.
   - Returns the broadcast address if found or null otherwise