Documentation

This application implements a Distributed Averaging System (DAS) that operates using UDP communication.

The application is designed to run in two modes: Master and Slave. The purpose of the system is to collect numbers from multiple processes (Slaves), compute an average, and broadcast the result across the network. The system works by receiving numbers from Slaves and calculating the average, then broadcasting it.

Description of modes:

Master

If the UDP port provided as a parameter is available, the Master mode is chosen. It stores its number provided as a parameter and starts listening to Slaves till the termination.

New number received from the Slave:

- Integer (not 0 or -1): stores it for further calculations.
- 0:
    1. calculates the average of all numbers sent before (the average is an integer)
    2. displays *"Average: ..."*
    3. broadcasts the average
    4. displays *"Broadcasted average: ..."*
- -1:
    1. displays -1
    2. broadcasts -1 to all machines in local network which use the same port
    3. displays the messages *"Broadcasting -1"* and *"Terminating"*
    4. terminates

Slave

When the UDP port is in use, the Slave is created. It sends its provided number to the port, which is used by the Master and provided as an argument, then terminates.

Protocol Description

The communication protocol used by the DAS system over UDP is as follows:

- **Message Format** (integer)
    - Integers (not 0 or -1): These represent data sent by Slaves to the Master for calculating the average

    - 0: This signals the Master to compute the average of all received numbers
    - -1: This indicates termination, prompting the Master to broadcast a shutdown signal and close the connection

- **Broadcasting**:
    - When the Master computes an average or receives a -1, it sends a broadcast message representing the average or *"-1"* to all machines in the local network on the same port. The message isn't broadcasted to itself.

Requirements Met

1. *Master Mode*:
    - correctly processes integers
    - does not process the data received from itself after broadcasting
    - computes averages when 0 is received
    - terminates after receiving -1
2. *Slave Mode*:
    - sends the number to the Master
    - terminates after transmission
3. *Retrieving the broadcast address*

Difficulties and errors present:

- The Master process *does not currently handle timeouts*, so if no messages -1 are received, it continues indefinitely.
- I faced challenges in *retrieving the broadcast address*. At first, I manually computed it using bitwise operations. Later, I discovered a built-in function that directly provides the broadcast address, simplifying the process. Additionally, I experimented with iterating over all network interfaces to find a suitable one. However, I realized this approach might not always select the correct interface, as it returns the first matching interface rather than the one required. Therefore, I determined it is more reliable to identify the network interface associated with the local address explicitly.

Description of program methods:

1. *Main Method:*

    Entry point for the program.

Functionality:

- Parses arguments to determine the port and number.
- Decides between Master mode and Slave mode based on the success of creating a DatagramSocket.
- If the socket is created successfully, it runs as a Master.
- If the socket fails due to a SocketException, it runs as a Slave.

2. *runMaster(DatagramSocket socket, int masterNumber)*
   Handles the logic for the Master mode.

   Functionality:

   - calls *getBroadcastAddress()* to retrieve a broadcast address. If it is null, terminates.
   - Continuously listens for incoming UDP packets.
   - Validates the sender's IP address and ignores packets from itself.
   - Processes the number received:
     - Adds it to the list if it's valid and logs the message that it received a number
     - If the number is 0, calls *broadcastAverage*.
     - If the number is -1, logs that -1 received, calls *sendSignalAndTerminaty* and terminates.
3. *runSlave(int port, int number)*

   Handles the logic for the Slave mode.

   Functionality:

   - Creates a UDP socket.
   - Sends the given number as a UDP packet to the master on the specified port.
   - Logs the sent message.
   - Closes the socket after sending the message.
4. *broadcastAverage(DatagramSocket socket, List<Integer> nums)*
   Calculates the average of the received numbers and broadcasts it.

   Functionality:

   - Calls *calculateAverage* to compute the average of the numbers in the list.
   - Prepares the message as a string and sends it using *broadcastMessage*.
   - Logs the broadcasted average.

5. *broadcastMessage(DatagramSocket socket, String message)*
   Sends a broadcast message to all devices in the local network.

   Functionality:

   - Creates a DatagramPacket with the message, broadcast address, and port.
   - Sends the packet via the provided DatagramSocket.

6. *sendSignalAndTerminate(DatagramSocket socket)*

   Broadcasts a termination signal (-1) and ends the master's operation.

   Functionality:

   - Calls *broadcastMessage* with the termination signal (-1).
   - Logs the termination message.

7. *calculateAverage(List<Integer> nums)*

   Computes the average of numbers in the list.

   Functionality:

   - Sums up all the numbers in the list.
   - Divides the sum by the number of elements in the list to calculate the average.
   - Returns the calculated average.

8. *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.