# STOP AND WAIT PROTOCOL

**Experiment No : 7 DATE:25-03-2024**

**Aim**: To implement stop and wait protocol using inter process communication.

# Theory:

**Sliding Window Protocol using Stop-and-Wait:**

* The sliding window protocol is a method used in computer networks to ensure reliable and efficient data transmission between two devices.
* Stop-and-wait is a variant of the sliding window protocol where the sender sends one packet at a time and waits for an acknowledgment from the receiver before sending the next packet.

**Real-World Use Case:**

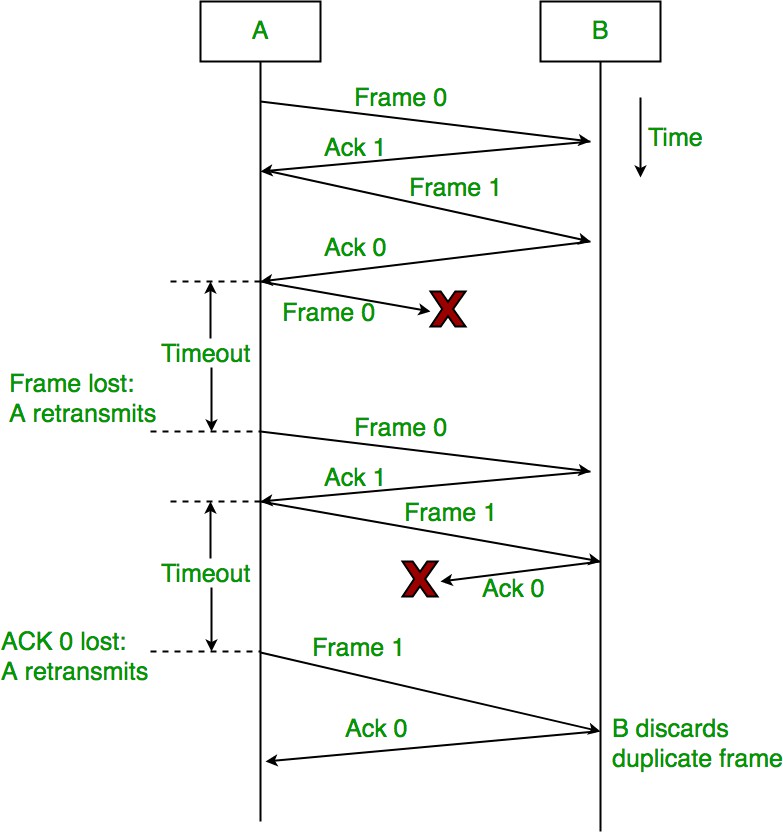
* Stop-and-wait sliding window protocol is commonly used in scenarios where the reliability of data transmission is crucial, such as in wireless communication systems, satellite communication, and point-to-point connections with high error rates.
* It is also employed in scenarios where the network bandwidth is limited or where the network conditions are prone to fluctuations, as it helps in controlling the flow of data to avoid congestion and packet loss.

**Advantages:**

1. **Reliability**: Ensures reliable delivery of data by employing acknowledgment mechanisms.
2. **Simple Implementation**: Stop-and-wait protocol is relatively easy to implement compared to other sliding window protocols, making it suitable for simple communication systems.
3. **Flow Control**: Helps in regulating the flow of data to prevent congestion and optimize network performance.
4. **Error Detection**: Facilitates the detection of transmission errors through acknowledgment and timeout mechanisms.

**Disadvantages:**

1. **Low Efficiency**: Since the sender can only transmit one packet at a time, the protocol may not fully utilize the available bandwidth, leading to lower efficiency, especially in high-speed networks.
2. **Increased Latency**: The transmission of each packet is followed by a waiting period for acknowledgment, which can introduce additional latency, particularly in networks with long propagation delays.
3. **Limited Throughput**: Due to its stop-and-wait nature, the protocol may not achieve high throughput, especially in scenarios where the round-trip time is high or the bandwidth-delay product is large.
4. **Inefficiency in High Error Environments**: In networks with high error rates, the protocol may experience frequent retransmissions, leading to decreased overall efficiency.



**Code:**

// Sender.java

import java.io.\*;

import java.net.\*;

public class Sender {

static int d\_size = 5;

static int[] data = {5, 4, 3, 2, 1};

public static void main(String[] args) {

try {

ServerSocket server\_socket = new ServerSocket(6666);

System.out.println("Waiting for the receiver to connect.");

Socket socket = server\_socket.accept();

System.out.println("Receiver connected.");

ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());

ObjectInputStream in = new ObjectInputStream(socket.getInputStream());

int i = d\_size - 1;

System.out.print("Data: ");

for(int j=0; j<d\_size; j++)

System.out.print(data[j]+" | ");

System.out.println();

while (i >= 0)

{

//Send data

out.writeObject(data[i]);

System.out.println("Sent: " + data[i]);

System.out.print("Data: ");

for(int j=0; j<i; j++)

System.out.print(data[j]+" | ");

System.out.println();

//Receive and check ACK

String ack = (String) in.readObject();

int frame\_no = Character.getNumericValue(ack.charAt(3));

if (frame\_no != d\_size - 1 - i)

continue;

i--;

Thread.sleep(2000);

}

in.close();

out.close();

socket.close();

server\_socket.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}

//Receiver.java

import java.io.\*;

import java.net.\*;

import java.util.ArrayList;

public class Receiver {

static ArrayList<Integer> data = new ArrayList<>();

public static void main(String[] args) {

try {

Socket socket = new Socket("localhost", 6666);

ObjectInputStream in = new ObjectInputStream(socket.getInputStream());

ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());

System.out.println("Receiver:");

for (int i = 0; i < 5; i++) {

int received\_data = (int) in.readObject();

System.out.println("Received: " + received\_data);

data.add(received\_data);

System.out.print("Data: ");

for (int j = data.size() - 1; j >= 0; j--)

{

System.out.print(data.get(j)+" | ");

}

System.out.println();

out.writeObject("ACK" + i);

}

in.close();

out.close();

socket.close();

} catch (Exception e) {

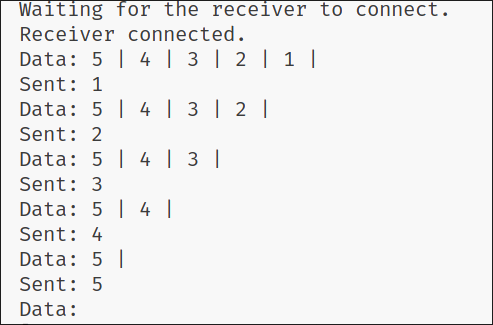
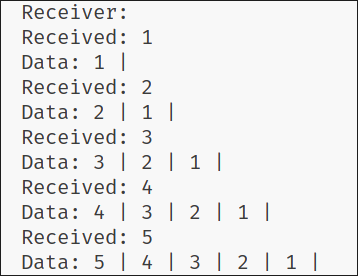
e.printStackTrace();

}

}

}

**Output:**

**Conclusion:**

The Stop and Wait technique was studied and implemented successfully.