Experiment No: 7 **Date:**

Stop Wait Protocol

AIM: Implement Stop Wait Protocol using Interprocess 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:

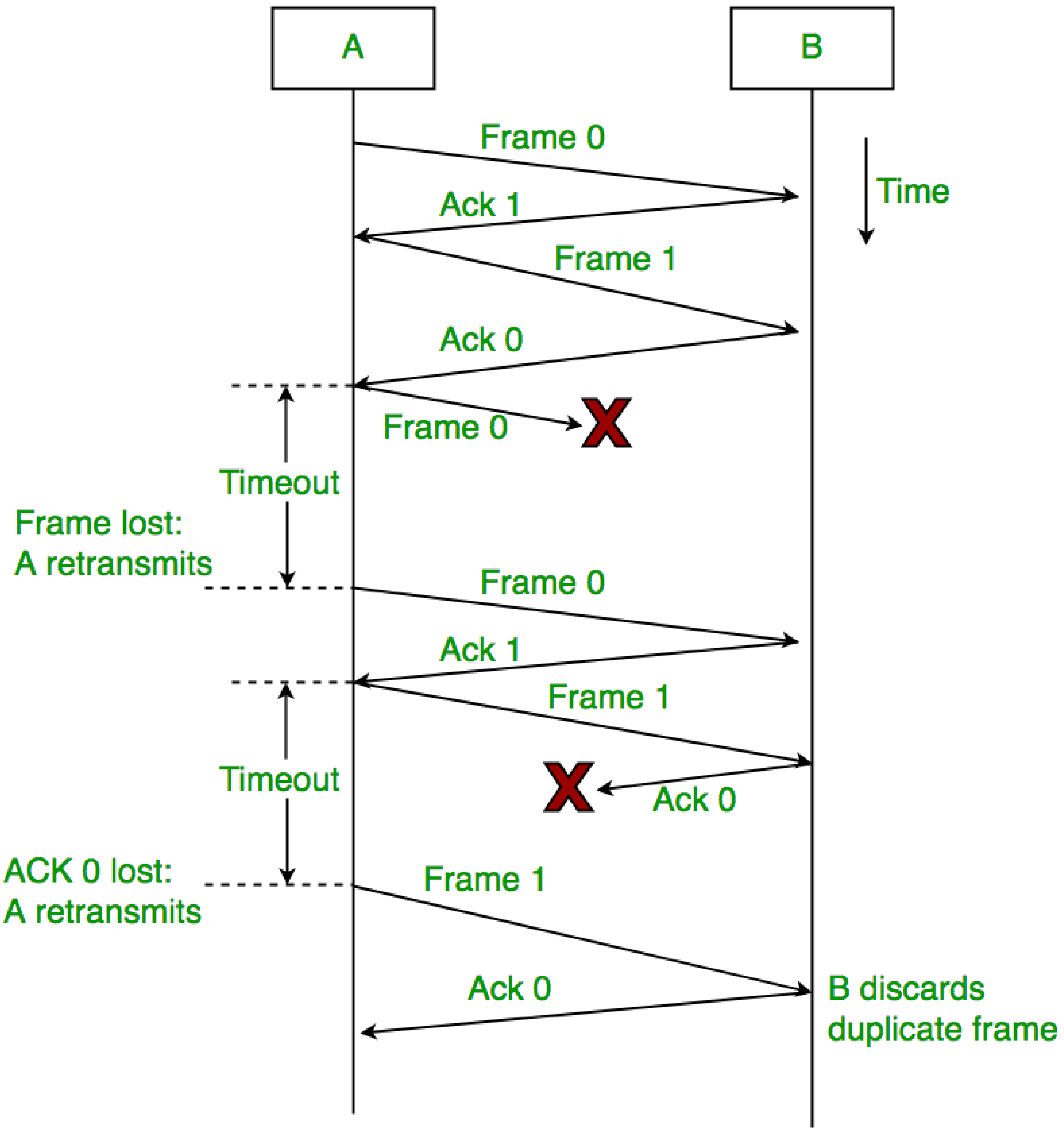
1. **Reliability**: Ensures reliable delivery of data by employing acknowledgment mechanisms.
2. **Simple Implementation**: Stop-and-wait protocol is relatively easy to implement compared to other sliding window protocols, making it suitable for simple communication systems.
3. **Flow Control**: Helps 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.

Sliding Window Using Stop & Wait Protocol.

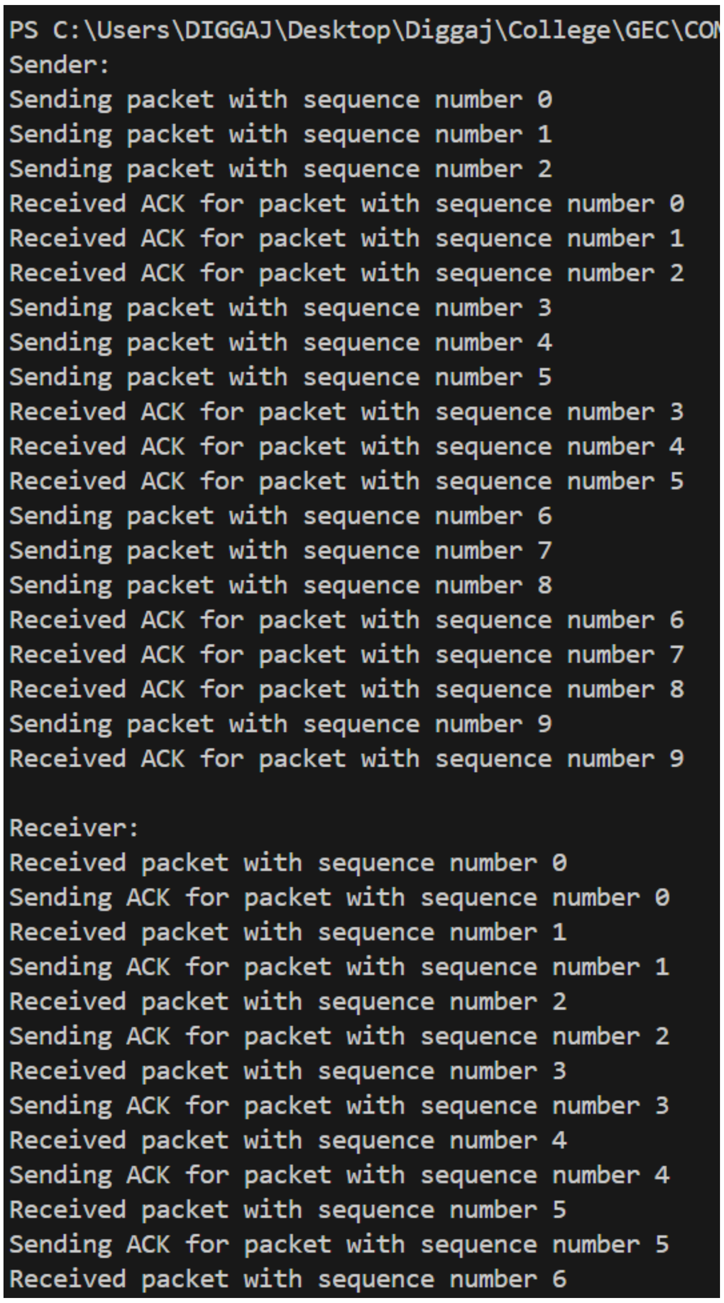


CODE

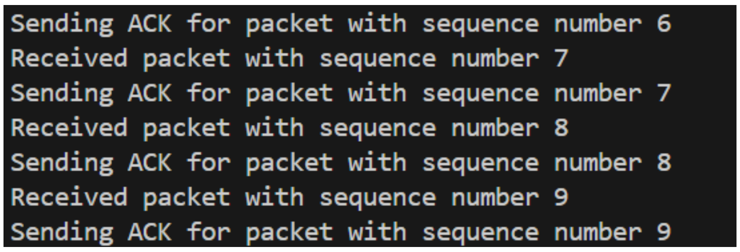
```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <string>
using namespace std;

// Simulated transmission delay
const int TRANSMISSION_DELAY = 1000; // in milliseconds

// Simulated ACK/NACK loss probability
const double ACK_LOSS_PROBABILITY = 0.1;

// Sliding window size
const int WINDOW_SIZE = 3;

// Timeout duration in milliseconds
const int TIMEOUT_DURATION = 2000;

// Packet structure
struct Packet {
    int sequenceNumber;
    bool isAcked;
    string data;
};

// Function to simulate transmission delay
void delay(int milliseconds) {
    clock_t start = clock();
    while ((clock() - start) * 1000 / CLOCKS_PER_SEC < milliseconds);
```

```
}

// Function to simulate ACK/NACK loss
bool isAckLost() {
    return (rand() / (RAND_MAX + 1.0)) < ACK_LOSS_PROBABILITY;
}

// Sender function
void sender(Packet packets[], int numPackets) {
    int base = 0;
    int nextSeqNum = 0;

    while (base < numPackets) {
        // Send packets in the window
        for (int i = base; i < min(base + WINDOW_SIZE, numPackets); ++i) {
            if (!packets[i].isAcked) {
                // Simulate transmission delay
                delay(TRANSMISSION_DELAY);
                cout << "Sending packet with sequence number " << packets[i].sequenceNumber
<< endl;
            }
        }

        // Receive ACKs
        for (int i = base; i < min(base + WINDOW_SIZE, numPackets); ++i) {
            if (!packets[i].isAcked) {
                // Simulate ACK loss
                if (!isAckLost()) {
                    cout << "Received ACK for packet with sequence number " <<
packets[i].sequenceNumber << endl;
                    packets[i].isAcked = true;
                nextSeqNum = max(nextSeqNum, packets[i].sequenceNumber + 1);
            }
        }
    }

    // Move the base
    while (base < numPackets && packets[base].isAcked) {
        ++base;
    }
  }
}

// Receiver function
void receiver(Packet packets[], int numPackets) {
    int expectedSeqNum = 0;

    for (int i = 0; i < numPackets; ++i) {
        // Simulate transmission delay
        delay(TRANSMISSION_DELAY);

        // Simulate packet loss
        if (!isAckLost()) {
            if (packets[i].sequenceNumber == expectedSeqNum) {
                cout << "Received packet with sequence number " << packets[i].sequenceNumber
<< endl;
                ++expectedSeqNum;
            }
            // Send ACK
            cout << "Sending ACK for packet with sequence number " <<
packets[i].sequenceNumber << endl;
        } else {
            cout << "Packet with sequence number " << packets[i].sequenceNumber << " lost."
<< endl;
        }
    }
}

int main() {
    srand(time(0)); // Initialize random seed

    // Number of packets to be sent
    const int numPackets = 10;

    // Generate packets with sequence numbers and random data
    Packet packets[numPackets];
    for (int i = 0; i < numPackets; ++i) {
        packets[i].sequenceNumber = i;
        packets[i].isAcked = false;
        packets[i].data = "Packet " + to_string(i);
    }
    // Simulate sender and receiver
    cout << "Sender:" << endl;
    sender(packets, numPackets);

    cout << "\nReceiver:" << endl;
    receiver(packets, numPackets);

    return 0;
}
```

OUTPUT



```
PS C:\Users\DIGGAJ\Desktop\Diggaj\College\GEC\CO
Sender:
Sending packet with sequence number 0
Sending packet with sequence number 1
Sending packet with sequence number 2
Received ACK for packet with sequence number 0
Received ACK for packet with sequence number 1
Received ACK for packet with sequence number 2
Sending packet with sequence number 3
Sending packet with sequence number 4
Sending packet with sequence number 5
Received ACK for packet with sequence number 3
Received ACK for packet with sequence number 4
Received ACK for packet with sequence number 5
Sending packet with sequence number 6
Sending packet with sequence number 7
Sending packet with sequence number 8
Received ACK for packet with sequence number 6
Received ACK for packet with sequence number 7
Received ACK for packet with sequence number 8
Sending packet with sequence number 9
Received ACK for packet with sequence number 9

Receiver:
Received packet with sequence number 0
Sending ACK for packet with sequence number 0
Received packet with sequence number 1
Sending ACK for packet with sequence number 1
Received packet with sequence number 2
Sending ACK for packet with sequence number 2
Received packet with sequence number 3
Sending ACK for packet with sequence number 3
Received packet with sequence number 4
Sending ACK for packet with sequence number 4
Received packet with sequence number 5
Sending ACK for packet with sequence number 5
Received packet with sequence number 6
```



```
Sending ACK for packet with sequence number 6
Received packet with sequence number 7
Sending ACK for packet with sequence number 7
Received packet with sequence number 8
Sending ACK for packet with sequence number 8
Received packet with sequence number 9
Sending ACK for packet with sequence number 9
```

CONCLUSION:

The Stop Wait Protocol using interprocess communication was studied and implemented successfully.