Experiment 5: ARQ Mechanisms in DLL

Aim: To implement receiver algorithms for the different ARQ mechanisms at the Data Link Layer

Objective: After carrying out this experiment, students will be able to:

- implement receiver algorithms for the different ARQ mechanisms at the Data Link Layer
- Analyze the differences between the ARQ mechanisms

Problem statement: You are required to write a program that can receive frames at the data link layer. Assume that the user is entering the frames as the transmitter. You are required to implement stop and wait, go back N and selective repeat ARQ mechanisms. Consider that you have to transmit and receive a total of 20 frames using $W_T=W_R=1$, $W_T=5$ and $W_R=1$ and $W_T=W_R=5$ for stop and wait, go back N and selective repeat respectively

Analysis: While analyzing your program, you are required to address the following points:

- Difference between stop and wait, go back N and selective repeat.
- Comparison of the disadvantages of the different ARQ mechanisms.

MARKS DISTRIBUTION

Component	Maximum Marks	Marks Obtained
Preparation of Document	7	
Results	7	
Viva	6	
Total	**20**	

Submitted by: Subhendu Maji

Register No: 18ETCS002121

1. Algorithm/Flowchart

The algorithm for Go-Back-N is as follows:

```
STEP 1: Start

STEP 2: initialize variables

STEP 3: numOfFrames, windowSize ← from user

STEP 4: while count is less than numOfFrames, do

        4.1: numOfTransInCurrentWindow ← 0

        4.2: for j from count to numOfFrames and count + windowSize,
        do

             4.2.1: display transmission message

             4.2.1: totalTransmissions++

        4.3: for j from count to numOfFrames and count + windowSize,
        do

             4.3.1: flag ← random number % 2

             4.3.1: if flag is 0, do

                  4.3.1.1: display success message

                  4.3.1.2: numOfTransInCurrentWindow++

             4.3.2: else, display failure message, break

STEP 5: display results

STEP 6: Stop
```

Algorithm for selective repeat is as follows:

```
STEP 1: Start
```


```
STEP 2: initialize variables

STEP 3: numOfFrames, windowSize ← from user

STEP 4: while count is less than numOfFrames, do

        4.1: numOfTransInCurrentWindow ← 0

        4.2: for j from count to numOfFrames and count + windowSize,
        do

             4.2.1: display transmission message

             4.2.1: totalTransmissions++

        4.3: for j from count to numOfFrames and count + windowSize,
        do

             4.3.1: flag ← random number % 2

             4.3.1: if flag is 0, do

                  4.3.1.1: display success message

                  4.3.1.2: numOfTransInCurrentWindow++

             4.3.2: else, display failure message, retransmit message

STEP 5: display results

STEP 6: Stop
```

Algorithm for stop and wait is as follows:

```
STEP 1: Start

STEP 2: initialize vars

STEP 3: numOfFrames ← from user

STEP 4: while count is less than numOfFrames, do

        4.1: display sending message
```


4.2: flag = random number % 2

4.3: if flag is 0, display success, count++

4.4: else, display failure, retransmitting message

STEP 5: Stop

2. Program

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

/**
 * Go Back N protocol.
 */
int main()
{
    // declaring variables
    int numOfFrames, windowSize;
    int count = 0, totalTransmissions = 0;
    int numOfTransInCurrentWindow = 0;
    //seeding random number generator
    srand(time(NULL));

    // gathering user input
    printf("Enter the number of frames: ");
    scanf("%d", &numOfFrames);
    printf("Enter the window size: ");
    scanf("%d", &windowSize);

    while (count <= numOfFrames)
    {
        numOfTransInCurrentWindow = 0;

        // loop for transmitting frames
        for (int j = count; j < count + windowSize && j <= numOfFrames; j++)
        {
            printf("Frame %d sent.\n", j);
            totalTransmissions++;
        }

        // loop for receiving acknowledgement
        for (int j = count; j < count + windowSize && j <= numOfFrames; j++)
        {
            // using flag to simulate whether frame is lost
            int flag = rand() % 2;

            // if flag is 0, implies acknowledgement received.
            if (!flag)
            {
                printf("Acknowledgement recieved for frame %d\n", j);
                numOfTransInCurrentWindow++;
            }
            else
            {
                printf("Frame %d not recevied. Retransmitting window.\n", j);
                break;
            }
        }
        printf("\n");
        count += numOfTransInCurrentWindow;
    }

    printf("total number of transmissions: %d\n", totalTransmissions);
}
```

Figure 1 Go Back N Source Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>

/**
 * Selective Repeat protocol.
 */
int main()
{
    // declaring variables
    int numOfFrames, windowSize, flag;
    int count = 0, empty = 0, numOfAcks = 0;
    int *buffer;

    // seeding rand
    srand(time(NULL));

    // gathering input from user
    printf("Enter the number of frames: ");
    scanf("%d", &numOfFrames);
    printf("Enter the window size: ");
    scanf("%d", &windowSize);

    // mallocing buffer
    buffer = malloc(windowSize * sizeof(*buffer));
    memset(buffer, -1, windowSize * sizeof(*buffer));

    empty = windowSize;

    while (numOfAcks < numOfFrames)
    {
        // add frames to buffer
        for (int i = count + 1; i <= count + empty && i <= numOfFrames; i++)
        {
            for (int j = 0; j < windowSize; j++)
            {
                if (buffer[j] == -1)
                {
                    buffer[j] = i;
                    break;
                }
            }
        }

        // after filling buffer, number of empty slots is 0
        empty = 0;

        // transmitting frames
        for (int i = 0; i < windowSize; i++)
        {
            if (buffer[i] == -1)
                continue;
            printf("sent frame %d.\n", buffer[i]);
            if (buffer[i] > count)
                count = buffer[i];
        }
        printf("\n");

        // receiving acknowledgements
        for (int i = 0; i < windowSize; i++)
        {
            if (buffer[i] == -1)
                continue;
            flag = rand() % 2;

            // if the flag is 0, it means that acknowledgement has successfully been
            // received.
            if (!flag)
            {
                printf("Acknowledgement for frame %d received.\n", buffer[i]);
                buffer[i] = -1;
                empty++;
                numOfAcks++;
            }
            else
            {
                printf("Acknowledgement for frame %d not received.\n", buffer[i]);
            }
        }
    }
}
```

Figure 2 Selective Protocol Source Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

/**
 * Stop and Wait Protocol.
 */
int main()
{
    // declaring variables
    int numOfFrames, flag;
    int count = 0;
    //seeding random number generator
    srand(time(NULL));

    // gathering user input
    printf("Enter the number of frames: ");
    scanf("%d", &numOfFrames);

    printf("Simulating Stop and Wait Protocol:\n");
    while (count < numOfFrames)
    {
        printf("Sending frame %d\n", count + 1);

        // using flag to simulate whether frame is lost
        flag = rand() % 2;

        // if flag is 0, transmission is successful
        if (!flag)
        {
            printf("Acknowledgement for frame %d recieved.\n\n", count + 1);
            count++;
        }
        else
        {
            printf("Acknowledgement for frame %d not recieved. "
                   "Retransmitting.\n",
                   count + 1);
        }
    }
}
```

Figure 3 Stop and Wait Source Code

makefile

```
go_back_n:
	@gcc go_back_n.c -lm
	@./a.out

selective_repeat:
	@gcc selective_repeat.c -lm
	@./a.out

stop_and_wait:
	@gcc stop_and_wait.c -lm
	@./a.out

clean:
	@rm *.out

.PHONY: clean
```

Figure 4 Makefile

3. Results

```
/mnt/d/RUAS-sem-05/CN/lab05 main*
❯ make go_back_n
Enter the number of frames: 3
Enter the window size: 2
Frame 0 sent.
Frame 1 sent.
Acknowledgement recieved for frame 0
Frame 1 not recevied. Retransmitting window.

Frame 1 sent.
Frame 2 sent.
Acknowledgement recieved for frame 1
Frame 2 not recevied. Retransmitting window.

Frame 2 sent.
Frame 3 sent.
Frame 2 not recevied. Retransmitting window.

Frame 2 sent.
Frame 3 sent.
Acknowledgement recieved for frame 2
Acknowledgement recieved for frame 3

total number of transmissions: 8

/mnt/d/RUAS-sem-05/CN/lab05 main* 8s
❯
```

Figure 5 Execution of Go Back N

```
/mnt/d/RUAS-sem-05/CN/lab05 main*
❯ make selective_repeat
Enter the number of frames: 3
Enter the window size: 2
sent frame 1.
sent frame 2.

Acknowledgement for frame 1 received.
Acknowledgement for frame 2 not received.
sent frame 3.
sent frame 2.

Acknowledgement for frame 3 not received.
Acknowledgement for frame 2 not received.
sent frame 3.
sent frame 2.

Acknowledgement for frame 3 not received.
Acknowledgement for frame 2 not received.
sent frame 3.
sent frame 2.

Acknowledgement for frame 3 received.
Acknowledgement for frame 2 received.

/mnt/d/RUAS-sem-05/CN/lab05 main*
❯
```

Figure 6 Execution of Selective Protocol

```
/mnt/d/RUAS-sem-05/CN/lab05 main*
❯ make stop_and_wait
Enter the number of frames: 3
Simulating Stop and Wait Protocol:
Sending frame 1
Acknowledgement for frame 1 not recieved. Retransmitting.
Sending frame 1
Acknowledgement for frame 1 recieved.

Sending frame 2
Acknowledgement for frame 2 recieved.

Sending frame 3
Acknowledgement for frame 3 not recieved. Retransmitting.
Sending frame 3
Acknowledgement for frame 3 not recieved. Retransmitting.
Sending frame 3
Acknowledgement for frame 3 recieved.


/mnt/d/RUAS-sem-05/CN/lab05 main*
❯
```

Figure 7 Execution of Stop and Wait

4. Analysis and Discussions

Stop and Wait ARQ mechanism sends frames one by one and waits for a certain time for the receiver to receive those frames. Requires a lot of bandwidth and is very inefficient. If a frame doesn't reach the receiver in time, or is lost in the process or is corrupted, Stop and Wait mechanism, retransmits that frame. Both frames and acknowledgment (ACK) are numbered alternately 0 and 1. The sender keeps a copy of the sent frame transmitted until it receives an acknowledgment for reference.

Go Back-N ARQ mechanism uses sliding window protocol which makes it so that we send multiple frames in one go till the size of the sliding window. The sliding window is of the size on transmitter side is 2^n-1. In Go Back-N, we send multiple frames in the sliding window at once and the receiver sends an acknowledgement of the next expected frame. If a frame is damaged or out of order, the receiver is silent and will discard all subsequent frames. In this protocol, send frames are numbered sequentially. The window size of the receiver is 1. Go Back-N suffers in a noisy channel. It also requires more bandwidth when we discard frames because of faults in between.

Selective Repeat ARQ mechanism also uses sliding window protocol. It even works in noisy channel and resends only the damaged frame. It also has the concept of negative acknowledgement (NACK) that reports sequence of damaged frames before timer expires. The sliding window's size for both transmitter and receiver are $2^m/2$. Rest of the working is same as Go Back-N.

5. Conclusions

The Go Back-N, Selective Repeat and Stop and Wait ARQ mechanisms were successfully learned and simulated via program written in C.