Experiment – 8: Write a program for congestion control using Leaky bucket algorithm.

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include <unistd.h>
#define NUM_PACKETS 10
// Custom random function to ensure non-zero result
int get_random(int max_value) {
    int result = (random() % 10) % max_value;
    return result == 0 ? 1 : result;
}
int main() {
    int packet_size[NUM_PACKETS];
    int i, clock_tick, bucket_size, output_rate;
    int remaining_data = 0, data_to_send, transmission_time, transmitted;
    // Generate random packet sizes
    for (i = 0; i < NUM_PACKETS; ++i)
        packet_size[i] = get_random(6) * 10;
    // Display the generated packet sizes
    for (i = 0; i < NUM_PACKETS; ++i)
        printf("\nPacket[%d]: %d bytes", i + 1, packet_size[i]);
    // Get output rate and bucket size from the user
    printf("\n\nEnter the Output Rate (bytes/unit time): ");
    scanf("%d", &output_rate);
    printf("Enter the Bucket Size (in bytes): ");
    scanf("%d", &bucket_size);
```

```
    // Process each incoming packet
    for (i = 0; i < NUM_PACKETS; ++i) {
        // Check if incoming packet + remaining data exceeds bucket size
        if ((packet_size[i] + remaining_data) > bucket_size) {
            if (packet_size[i] > bucket_size) {
                printf("\n\nIncoming Packet[%d] size (%d bytes) exceeds bucket capacity (%d bytes)
- PACKET REJECTED!", i + 1, packet_size[i], bucket_size);
            } else {
                printf("\n\nBucket capacity exceeded for Packet[%d] - PACKET REJECTED!", i + 1);
            }
        }
else {
            // Accept and add packet to bucket
            remaining_data += packet_size[i];
            printf("\n\nIncoming Packet[%d] size: %d bytes", i + 1, packet_size[i]);
            printf("\nTotal bytes to transmit (in bucket): %d", remaining_data);
            // Simulated transmission time (random)
            transmission_time = get_random(4) * 10;
            printf("\nEstimated transmission time: %d units", transmission_time);
            // Transmit data in units of time
            for (clock_tick = 10; clock_tick <= transmission_time; clock_tick += 10) {
                sleep(1);  // Simulate delay (1 second per 10 units)
                if (remaining_data > 0) {
                    if (remaining_data <= output_rate) {
                        transmitted = remaining_data;
                        remaining_data = 0;
```

```
            } else {
                transmitted = output_rate;
                remaining_data -= output_rate;
            }
            printf("\nTransmitted %d bytes ---- Remaining in bucket: %d bytes", transmitted, remaining_data);
        } else {
            printf("\nNo data to transmit at time unit %d!", clock_tick);
        }
      }
    }
    return 0;
}
```

Output:

Packet[1]: 30 bytes

Packet[2]: 10 bytes

Packet[3]: 10 bytes

Packet[4]: 50 bytes

Packet[5]: 30 bytes

Packet[6]: 50 bytes

Packet[7]: 10 bytes

Packet[8]: 20 bytes

Packet[9]: 30 bytes

Packet[10]: 10 bytes

Enter the Output Rate (bytes/unit time): 10

Enter the Bucket Size (in bytes): 30

Incoming Packet[1] size: 30 bytes

Total bytes to transmit (in bucket): 30

Estimated transmission time: 20 units

Transmitted 10 bytes ---- Remaining in bucket: 20 bytes

Transmitted 10 bytes ---- Remaining in bucket: 10 bytes

Incoming Packet[2] size: 10 bytes

Total bytes to transmit (in bucket): 20

Estimated transmission time: 30 units

Transmitted 10 bytes ---- Remaining in bucket: 10 bytes

Transmitted 10 bytes ---- Remaining in bucket: 0 bytes

No data to transmit at time unit 30!

Incoming Packet[3] size: 10 bytes

Total bytes to transmit (in bucket): 10

Estimated transmission time: 10 units

Transmitted 10 bytes ---- Remaining in bucket: 0 bytes

Incoming Packet[4] size (50 bytes) exceeds bucket capacity (30 bytes) - PACKET REJECTED!

Incoming Packet[5] size: 30 bytes

Total bytes to transmit (in bucket): 30

Estimated transmission time: 10 units

Transmitted 10 bytes ---- Remaining in bucket: 20 bytes

Incoming Packet[6] size (50 bytes) exceeds bucket capacity (30 bytes) - PACKET REJECTED!

Incoming Packet[7] size: 10 bytes

Total bytes to transmit (in bucket): 30

Estimated transmission time: 30 units

Transmitted 10 bytes ---- Remaining in bucket: 20 bytes

Transmitted 10 bytes ---- Remaining in bucket: 10 bytes

Transmitted 10 bytes ---- Remaining in bucket: 0 bytes

Incoming Packet[8] size: 20 bytes

Total bytes to transmit (in bucket): 20

Estimated transmission time: 20 units

Transmitted 10 bytes ---- Remaining in bucket: 10 bytes

Transmitted 10 bytes ---- Remaining in bucket: 0 bytes

Incoming Packet[9] size: 30 bytes

Total bytes to transmit (in bucket): 30

Estimated transmission time: 10 units

Transmitted 10 bytes ---- Remaining in bucket: 20 bytes

Incoming Packet[10] size: 10 bytes

Total bytes to transmit (in bucket): 30

Estimated transmission time: 20 units

Transmitted 10 bytes ---- Remaining in bucket: 20 bytes

Transmitted 10 bytes ---- Remaining in bucket: 10 bytes

Experiment – 9: Write a program for frame sorting techniques used in buffers.

Program:

```
#include<stdio.h>
#include <stdlib.h>
#include <string.h>
#define FRAME_TEXT_SIZE 3      // Each frame holds 3 characters
#define MAX_NO_OF_FRAMES 127   // Maximum number of frames supported
char inputMessage[FRAME_TEXT_SIZE * MAX_NO_OF_FRAMES]; // Buffer to store the original message
// Frame structure to hold individual fragments of the message
struct Frame {
    char text[FRAME_TEXT_SIZE];
    int sequenceNumber;
} frames[MAX_NO_OF_FRAMES], shuffledFrames[MAX_NO_OF_FRAMES];
// Function to split the message into frames and assign sequence numbers
int assignSequenceNumbers() {
    int i = 0, j = 0, frameIndex = 0;
    while (i < strlen(inputMessage)) {
        frames[frameIndex].sequenceNumber = frameIndex;
        for (j = 0; j < FRAME_TEXT_SIZE && inputMessage[i] != '\0'; j++)
            frames[frameIndex].text[j] = inputMessage[i++];
        frameIndex++;
    }
    printf("\nAfter assigning sequence numbers:\n");
    for (i = 0; i < frameIndex; i++)
```

```
        printf("%d:%s ", frames[i].sequenceNumber, frames[i].text);
    return frameIndex;  // Return total number of frames created
}
// Utility to generate an array of unique random indices for shuffling
void generateRandomArray(int *randomArray, int limit) {
    int r, i = 0, j;
    while (i < limit) {
        r = rand() % limit;
        for (j = 0; j < i; j++)
            if (randomArray[j] == r)
                break;
        if (i == j)  // r is unique
            randomArray[i++] = r;
    }
}

// Shuffle the frames based on random sequence numbers
void shuffleFrames(int totalFrames) {
    int randomIndices[totalFrames];
    generateRandomArray(randomIndices, totalFrames);
    for (int i = 0; i < totalFrames; i++)
        shuffledFrames[i] = frames[randomIndices[i]];
    printf("\n\nAfter shuffling:\n");
    for (int i = 0; i < totalFrames; i++)
        printf("%d:%s ", shuffledFrames[i].sequenceNumber, shuffledFrames[i].text);
}
```

```
// Sort the shuffled frames based on their original sequence numbers
void sortFrames(int totalFrames) {
    int i, j, swapped;
    struct Frame temp;
    for (i = 0; i < totalFrames - 1; i++) {
        swapped = 0;
        for (j = 0; j < totalFrames - i - 1; j++) {
            if (shuffledFrames[j].sequenceNumber > shuffledFrames[j + 1].sequenceNumber) {
                temp = shuffledFrames[j];
                shuffledFrames[j] = shuffledFrames[j + 1];
                shuffledFrames[j + 1] = temp;
                swapped = 1;
            }
        }

        if (!swapped) break; // Optimization: Break if already sorted
    }
}

int main() {
    int totalFrames;
    printf("Enter the message: ");
    fgets(inputMessage, sizeof(inputMessage), stdin);
    // Remove newline character if present
    size_t len = strlen(inputMessage);
```

```
    if (len > 0 && inputMessage[len - 1] == '\n')
        inputMessage[len - 1] = '\0';
    // Split into frames, shuffle them, then sort to recover original message
    totalFrames = assignSequenceNumbers();
    shuffleFrames(totalFrames);
    sortFrames(totalFrames);
    // Display the final reconstructed message
    printf("\n\nAfter sorting (Reconstructed message):\n");
    for (int i = 0; i < totalFrames; i++)
        printf("%s", shuffledFrames[i].text);
    printf("\n\n");
    return 0;
}
```

Output:

Enter the message: i love computer networks

After assigning sequence numbers:

0:i l 1:ove 2: co 3:mpu 4:ter 5: ne 6:two 7:rks

After shuffling:

7:rks 6:two 1:ove 3:mpu 2: co 4:ter 5: ne 0:i l

After sorting (Reconstructed message):

i love computer networks