// SELECTIVE REPEAT

#include <iostream>

#include <vector>

#include <ctime>

#include <cstdlib>

using namespace std;

int main() {

int windowSize, totalFrames;

cout << "Enter window size: ";

cin >> windowSize;

cout << "Enter number of frames to transmit: ";

cin >> totalFrames;

vector<int> sentFrames(totalFrames, 0);

vector<int> ackReceived(totalFrames, 0);

int nextFrameToSend = 0, nextAckExpected = 0;

srand(time(0)); // Seed for random number generation

while (nextAckExpected < totalFrames) {

// Send frames within the window

while (nextFrameToSend < totalFrames && nextFrameToSend < nextAckExpected + windowSize) {

if (sentFrames[nextFrameToSend] == 0) {

cout << "Sending frame " << nextFrameToSend << endl;

sentFrames[nextFrameToSend] = 1;

}

nextFrameToSend++;

}

// Simulate ACK reception

for (int i = nextAckExpected; i < nextFrameToSend; i++) {

if (sentFrames[i] == 1 && ackReceived[i] == 0) {

// Simulate ACK loss

if (rand() % 2 == 0) {

cout << "ACK received for frame " << i << endl;

ackReceived[i] = 1;

} else {

cout << "ACK lost for frame " << i << endl;

}

}

}

// Move the window forward by advancing nextAckExpected

while (nextAckExpected < totalFrames && ackReceived[nextAckExpected] == 1) {

nextAckExpected++;

}

// Retransmit unacknowledged frames in the current window

for (int i = nextAckExpected; i < nextFrameToSend; i++) {

if (sentFrames[i] == 1 && ackReceived[i] == 0) {

cout << "Retransmitting frame " << i << endl;

}

}

}

cout << "All frames transmitted successfully." << endl;

return 0;

}

// GO BACK N

#include <iostream>

#include <vector>

#include <ctime>

#include <cstdlib>

using namespace std;

int main() {

int windowSize, totalFrames, nextFrameToSend = 0, nextAckExpected = 0;

bool frameLost;

cout << "Enter the window size: ";

cin >> windowSize;

cout << "Enter the total number of frames to send: ";

cin >> totalFrames;

vector<bool> ackReceived(totalFrames, false);

srand(time(0)); // Seed for random number generation

while (nextAckExpected < totalFrames) {

// Send frames within the window

while (nextFrameToSend < nextAckExpected + windowSize && nextFrameToSend < totalFrames) {

cout << "Sending frame " << nextFrameToSend << endl;

nextFrameToSend++;

}

// Simulate frame loss

frameLost = (rand() % 2 == 0);

if (frameLost) {

cout << "Frame " << nextAckExpected << " lost!" << endl;

} else {

cout << "ACK received for frame " << nextAckExpected << endl;

ackReceived[nextAckExpected] = true;

nextAckExpected++;

}

// Go back N if frame lost

if (!frameLost && nextAckExpected < totalFrames && !ackReceived[nextAckExpected]) {

cout << "Timeout, retransmitting from frame " << nextAckExpected << endl;

nextFrameToSend = nextAckExpected;

}

}

cout << "All frames successfully sent!" << endl;

return 0;

}

//CNL: EXP 11: Leaky Bucket Algorithm (Congestion Control Algorithm)

#include <iostream>

using namespace std;

class LeakyBucket {

private:

int bucketSize;

int outputRate;

int currentSize;

public:

// Constructor to initialize bucket size and output rate

LeakyBucket(int bucketSize, int outputRate) {

this->bucketSize = bucketSize;

this->outputRate = outputRate;

this->currentSize = 0;

}

// Function to add packets to the bucket

void addPacket(int packetSize) {

if (packetSize > bucketSize) {

cout << "Packet of size " << packetSize << " discarded. Too large!" << endl;

} else if (currentSize + packetSize > bucketSize) {

cout << "Packet of size " << packetSize << " discarded. Bucket overflow!" << endl;

} else {

currentSize += packetSize;

cout << "Packet of size " << packetSize << " added to the bucket. Current size: " << currentSize << endl;

}

}

// Function to transmit packets from the bucket

void transmit() {

if (currentSize <= outputRate) {

cout << "Transmitting " << currentSize << " packets." << endl;

currentSize = 0;

} else {

cout << "Transmitting " << outputRate << " packets. " << currentSize - outputRate << " left in bucket." << endl;

currentSize -= outputRate;

}

}

};

int main() {

int bucketSize, outputRate, packetSize;

// Taking user input for bucket size and output rate

cout << "Enter bucket size: ";

cin >> bucketSize;

cout << "Enter output rate: ";

cin >> outputRate;

LeakyBucket bucket(bucketSize, outputRate);

char choice;

do {

cout << "\nEnter packet size to add: ";

cin >> packetSize;

bucket.addPacket(packetSize);

bucket.transmit(); // Transmit packets after each addition

cout << "\nDo you want to add another packet? (y/n): ";

cin >> choice;

} while (choice == 'y');

return 0;

}

//EXP 12: URL Splitting Experiment

#include <iostream>

#include <string>

using namespace std;

int main() {

string url;

cout << "Enter URL: ";

getline(cin, url);

string protocol, domain, path, query;

size\_t pos = 0, found;

// Find protocol

found = url.find("://");

if (found != string::npos) {

protocol = url.substr(0, found);

pos = found + 3; // Move past "://"

}

// Find domain

found = url.find("/", pos);

if (found != string::npos) {

domain = url.substr(pos, found - pos);

pos = found; // Position to start of path

} else {

domain = url.substr(pos);

pos = url.length();

}

// Find path

found = url.find("?", pos);

if (found != string::npos) {

path = url.substr(pos, found - pos);

pos = found + 1; // Move past "?"

} else {

path = url.substr(pos);

pos = url.length();

}

// Remaining part is query

query = url.substr(pos);

// Display the components

cout << "Protocol: " << (protocol.empty() ? "None" : protocol) << endl;

cout << "Domain: " << (domain.empty() ? "None" : domain) << endl;

cout << "Path: " << (path.empty() ? "/" : path) << endl;

cout << "Query: " << (query.empty() ? "None" : query) << endl;

return 0;

}