Ex No: 5

Date:

Simulation of Sliding – Window Protocol

Aim:

To write a C program for simulation of sliding- window protocol.

Algorithm:-

Step-1:- Start the program.
Step-2:- Initialize the sender & receiver buffers.
Step-3:-Print "Simulation of sliding-window protocol"
Step-4-While next_frame to send <WINDOW_ SIZE

a) Send frames in the window.
b) Receive acknowledgement.

Step-5:-Print All frames have been successfully transmitted.
Step-6:-End the program.

Program :-

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <unistd.h> // For sleep function
#define WINDOW_SIZE 4 // Sliding window size
#define TOTAL_FRAMES 10 // Total frames to send
typedef struct
{
int frame_id;
bool sent;
bool acked;
}
Frame;
Frame buffer[WINDOW_SIZE]; // Buffer for the sliding window
void send_frame(Frame *frame);
bool receive_ack (Frame frame);
int main()
{
```

```
int next_frame = 0; // Frame ID to be sent next
int base = 0; // Base of the sliding window
int acked = 0;                    // Number of acknowledged frames
printf("Sliding Window Protocol with Buffer Simulation\n");
printf("Window Size: %d\n", WINDOW_SIZE);
printf("Total Frames: %d\n\n", TOTAL_FRAMES);
     while (acked < TOTAL_FRAMES)
     {
       // Load frames into the buffer
       for (int i = 0; i < WINDOW_SIZE; i++)
      {
          if (base + i < TOTAL_FRAMES &&!buffer[i].sent)
             {
              buffer[i].frame_id = base + i;
              buffer[i].sent = false;
              buffer[i].acked = false;
              }

         }
// Send frames within the window
for (int i = 0; i < WINDOW_SIZE; i++)
{
   if (!buffer[i].sent && base + i <TOTAL_FRAMES)
  {
    printf("Sending frame %d\n", buffer[i].frame_id);
    send_frame(&buffer[i]); buffer[i].sent = true;
   }
}

// Receive acknowledgments
for (int i = 0; i < WINDOW_SIZE; i++)
{
  if (buffer[i].sent && !buffer[i].acked)
 {
    if (receive_ack(&buffer[i]))
       {
         printf("Acknowledged frame %d\n", buffer[i].frame_id);
```

```
            buffer[i].acked = true; acked++;
        }
    else
       {
        printf("Frame %d lost! Retransmitting from frame %d\n", buffer[i].frame_id, base);
        // Retransmit all frames from the base
        for (int j = 0; j < WINDOW_SIZE; j++)
                {
                  buffer[j].sent = false; buffer[j].acked = false;
                }
                break;
                }
        }
    }
// Slide the window
while (buffer[0].acked)
{
  for (int i = 1; i < WINDOW_SIZE; i++) { buffer[i - 1] = buffer[i];
}
buffer[WINDOW_SIZE - 1].sent = false; // Clear the last slot
base++;
}
printf("\n");
}
printf("All frames sent and acknowledged successfully!\n");
return 0;
}
void  send_frame(Frame  *frame)
{
  // Simulate sending a frame sleep(1);
}
bool receive_ack(Frame *frame)
{
// Simulate random acknowledgment loss (10% chance of loss)
return rand() % 10 >= 1;
}
```

Output:-

Sliding Window Protocol with Buffer Simulation

Window Size: 4

Total Frames: 10

Sending frame 0

Sending frame 1

Sending frame 2

Sending frame 3

Acknowledged frame O Acknowledged frame 1

Acknowledged frame 2

Acknowledged frame 3

RESULT:

Thus, the C program has been executed & output is verified successfully.

Ex No: 6

Date:

Simulation Of The BGF/OSPF Routing Protocols

Aim:

To write a c program for the simulation. of BGP/OSPF routings protocol.

Algorithm:-

Step-1:- Start the program.

Step-2:- Declare variables routers (MAX-NODES] to store router information & network [MAX_NODES] [MAX_NODES] to store network links & costs.

Step-3:- Initialize routers & network links.

Step-4:- Simulate OSPF routers to cydate routing tables.

Step-5:- Cyndate routing table function:

- Iterate through all nodes in the networks.
- If there is a direct link between the current router & another node. (network [i][j] = INFINITY)

Step-7:-Update the routing table of the current router based on the received information using the 'update routing Table' function.

Step-8:- Update routing table:

- ✓ Iterate through all nodes in the network.
- ✓ If the receiver costs of a node is less than the current cost in the routing table to the received cost.
- ✓ print the message indicating the update cost.

step-9:- Print the routing tables of all routers.

Step-10: End the program.

Program:-

```
#include <stdio.h>
#include <limits.h> // For INT_MAX
#include <stdbool.h>
#define MAX_NODES 10  // Maximum number of nodes (routers)
#define INFINITY INT_MAX // Representation of no direct link
// Structure to store router information typedef struct
{
    int routing_table[MAX_NODES][MAX_NODES];
    // Routing table for each router
    } Router;
// Function declarations

void initialize_router_and_network(int network[MAX_NODES][MAX_NODES], int nodes);
void simulate_OSPF(Router *routers, int network[MAX_NODES][MAX_NODES], int nodes);
void update_routing_table(Router *router, int network[MAX_NODES][MAX_NODES], int nodes);
void print_routing_table(Router *routers, int nodes);
int main() {
int network[MAX_NODES][MAX_NODES];
int nodes;
printf("Enter the number of routers (max %d): ", MAX_NODES);
scanf("%d",  &nodes);
if (nodes > MAX_NODES || nodes <= 0)
{
printf("Invalid number of routers!\n"); return 1;
}
Router routers[MAX_NODES]; // Array of routers

// Step 3: Initialize the network and routers initialize_router_and_network(network, nodes);

// Step 4: Simulate OSPF to update routing tables
simulate_OSPF(routers, network, nodes);

// Print final routing tables printf("\nFinal Routing Tables:\n");
print_routing_table(routers, nodes);
return 0;
}

// Step 3: Initialize router and network links
void initialize_router_and_network(int network[MAX_NODES][MAX_NODES], int nodes)
{
```

```
printf("Enter the network cost matrix (use %d for no direct link):\n", INFINITY);
for (int i = 0; i < nodes; i++) {
for (int j = 0; j < nodes; j++) {
scanf("%d", &network[i][j]);
if (i == j) {
network[i][j] = 0; // Cost to itself is always 0
}
}
}
}
// Step 4 & 5: Simulate OSPF and update routing tables
void simulate_OSPF(Router *routers, int network[MAX_NODES][MAX_NODES], int nodes) {
for (int i = 0; i < nodes; i++) {
// Initialize each router's routing table update_routing_table(&routers[i], network, nodes);
}
}
// Step 6: Update routing table for a specific router
void update_routing_table(Router *router, int network[MAX_NODES][MAX_NODES],
int nodes) {
    for (int i = 0; i < nodes; i++) {
        for (int j = 0; j < nodes; j++) {
            if (network[i][j] == INFINITY) {
                router->routing_table[i][j] = INFINITY;
// No direct link
            } else {
                router->routing_table[i][j] = network[i][j]; // Direct link cost
            }
        }
    }
}
```

```
// Step 7: Print routing tables for all routers
void print_routing_table(Router *routers, int nodes) {
    for (int i = 0; i < nodes; i++) { printf("\nRouting Table for Router %d:\n", i);
        printf("Destination\tCost\n");

        for (int j = 0; j < nodes; j++) {

            if (routers[i].routing_table[i][j] == INFINITY) {

                printf("%d\t\t%s\n", j, "INFINITY");

            } else {
                printf("%d\t\t%d\n", j, routers[i].routing_table[i][j]);

            }

        }
    }
}
```

Output:

Enter the number of routers (max 10): 4

Enter the network cost matrix (use 2147483G47 for no direct link):

1 4 2147483G47 3

2147483G47 3 4 5

1 2147483G47 5 4

1 2 3 2147583G47

Final Routing Tables:

Routing Table for Router 0: Destination Cost

0	0
1	4
2	Infinity
3	3

Routing Table for Router 1: Destination Cost

0	Infinity
1	0
2	4
3	5

Routing Table for Router 2:

Destination Cost

0	1
1	Infinity
2	0
3	4

Routing Table for Router 3:

Destination	Cost
0	1
1	2
2	3
3	4

RESULT:

Thus, the program has been executed and Output is verified successfully.