Aim:-

To write a program to simulate stop – and – wait protocol.

Description of Stop-and-Wait Protocol

Stop-and-wait Protocol is a flow control protocol used in the data link layer for transmission of data in noiseless channels. Sender keeps on sending messages to the Receiver. In order to prevent the receiver from overwhelming, there is a need to tell the sender to slow down the transmission of frames. We can make use of feedback from the receiver to the sender. Frames 0 sends to receiver, ACK 1 will be sentback to sender; frame 1 goes to receiver, ACK 0 will be back to sender, and so on.

Algorithm

1. Start the program

2. Generate a random number that gives the total number of frames to be transmitted.

3. Transmit the first frame

4. Receive the acknowledgement for the first frame

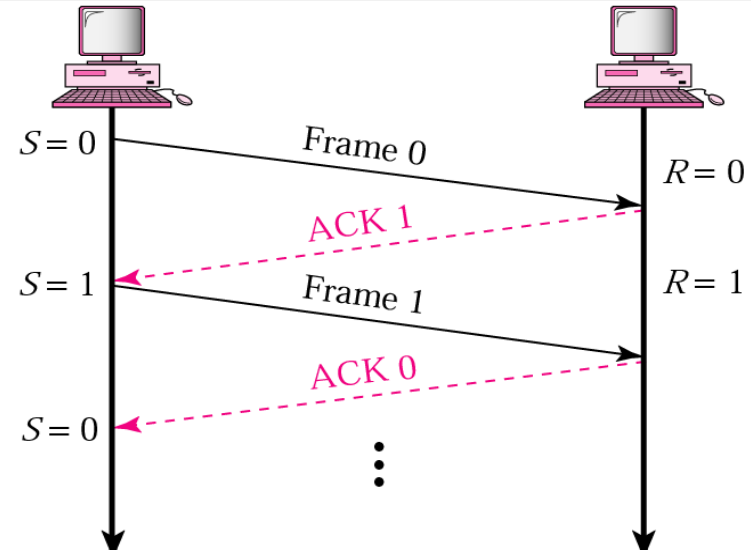
5. Transmit the next frame

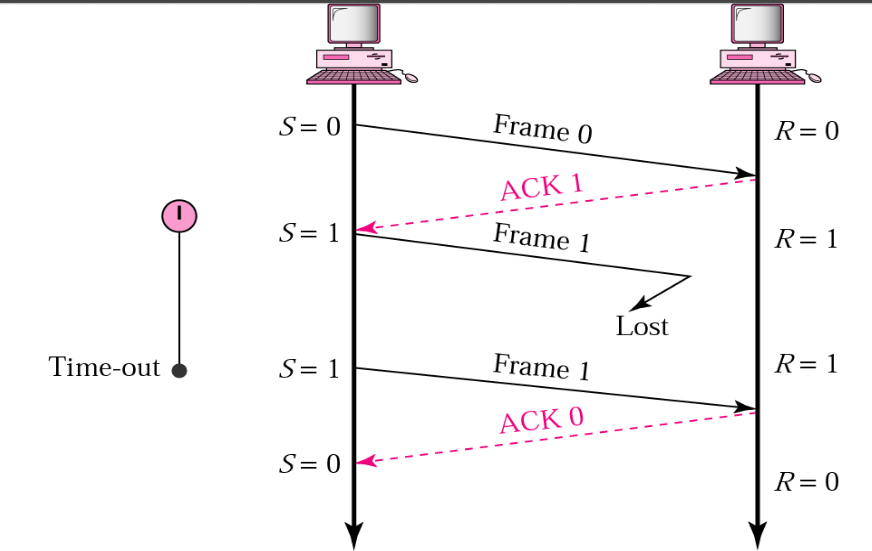
6. Find the remaining frames to be sent.

7. If an acknowledgement is not received for a particular frame, retransmit that frame alone again.

8. Repeat the steps 5 to 7 till the number of remaining frames to be sent becomes zero.

9. Stop the program.





**Code** –

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

int main()

{

int i,j,noframes,x,x1=10,x2;

for(i=0;i<200;i++)

rand();

noframes=rand()/200;

i=1;

j=1;

noframes = noframes / 8;

printf("\n number of frames is %d",noframes);

while(noframes>0)

{

printf("\nsending frame %d",i);

srand(x1++); //The srand() function sets the starting point for producing a series of pseudo-random integers

x = rand()%10;

if(x%2 == 0)

{

for (x2=1; x2<2; x2++)

{

printf("waiting for %d seconds\n", x2);

sleep(x2);

printf("Missing Acknowledgement %d",i);

}

printf("\nsending frame %d",i);

srand(x1++);

x = rand()%10;

}

printf("\nack received for frame %d",j);

noframes-=1;

i++;

j++;

}

printf("\n end of stop and wait protocol");

}

**Output**

number of frames is 1314574

sending frame 1

ack received for frame 1

sending frame 2

ack received for frame 2

sending frame 3waiting for 1 seconds

Missing Acknowledgement 3

sending frame 3

ack received for frame 3

sending frame 4

ack received for frame 4

sending frame 5

ack received for frame 5

sending frame 6waiting for 1 seconds

Missing Acknowledgement 6

sending frame 6

ack received for frame 6

sending frame 7waiting for 1 seconds

Missing Acknowledgement 7

sending frame 7

ack received for frame 7

sending frame 8

ack received for frame 8

sending frame 9waiting for 1 seconds

Missing Acknowledgement 9

sending frame 9

ack received for frame 9

sending frame 10waiting for 1 seconds

Missing Acknowledgement 10

sending frame 10

ack received for frame 10

**Algorithms for both Sender and Receiver sides**

The algorithm used at the sender site for the stop-and-wait protocol

This is an algorithm used at the sender site for the stop-and-wait protocol. Applications can have its implementation in its own programming language.

while(true) //Repeat forever

canSend=true //It will allow the first frame to go.

{

WaitForEvent(); //sleep until the occurrence of an event

if(Event(RequestToSend) AND canSend) {

GetData();

MakeFrame();

SendFrame(); //Send the data frame

canSend=false; //cannot send until the acknowledgement arrives.

}

WaitForEvent(); //sleep until the occurrence of an event

if(Event(ArrivalNotification)) //indicates the arrival of the acknowledgement

{

ReceiveFrame(); //Means the ACK frame received

canSend=true;

}

}

Algorithm At the Receiver Side

This is an algorithm used at the receiver side for the **stop-and-wait protocol**. Applications can have their implementation in their own programming language.

while(true) //means Repeat forever

{

WaitForEvent(); //sleep until the occurrence of an event

if(Event(ArrivalNotification)) //indicates arrival of the data frame

{

ReceiveFrame();

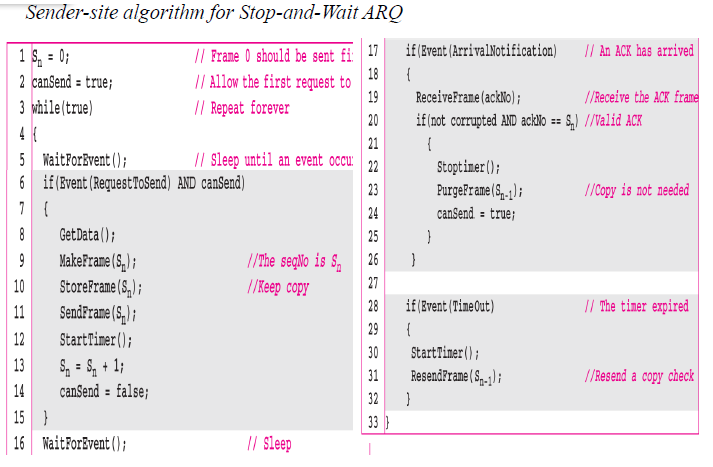
ExtractData();

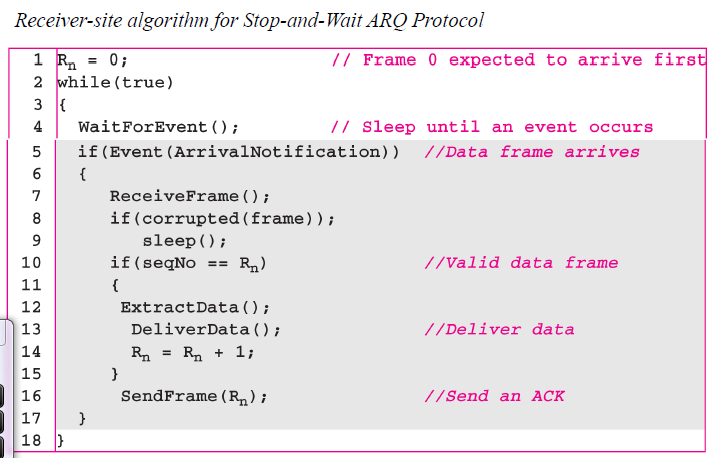
Deliver(data); //delivers the data to the network layer.

SendFrame(); //Send the ACK frame

}

}





**CLIENT SIDE**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

typedef struct packet{

char data[1024];

}Packet;

typedef struct frame{

int frame\_kind; //ACK:0, SEQ:1 FIN:2

int sq\_no;

int ack;

Packet packet;

}Frame;

int main(int argc, char \*\*argv[]){

if (argc != 2){

printf("Usage: %s <port>", argv[0]);

exit(0);

}

int port = atoi(argv[1]);

int sockfd;

struct sockaddr\_in serverAddr;

char buffer[1024];

socklen\_t addr\_size;

int frame\_id = 0;

Frame frame\_send;

Frame frame\_recv;

int ack\_recv = 1;

sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);

memset(&serverAddr, '\0', sizeof(serverAddr));

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(port);

serverAddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

while(1){

if(ack\_recv == 1){

frame\_send.sq\_no = frame\_id;

frame\_send.frame\_kind = 1;

frame\_send.ack = 0;

printf("Enter Data: ");

scanf("%s", buffer);

strcpy(frame\_send.packet.data, buffer);

sendto(sockfd, &frame\_send, sizeof(Frame), 0, (struct sockaddr\*)&serverAddr, sizeof(serverAddr));

printf("[+]Frame Send\n");

}

int addr\_size = sizeof(serverAddr);

int f\_recv\_size = recvfrom(sockfd, &frame\_recv, sizeof(frame\_recv), 0 ,(struct sockaddr\*)&serverAddr, &addr\_size);

if( f\_recv\_size > 0 && frame\_recv.sq\_no == 0 && frame\_recv.ack == frame\_id+1){

printf("[+]Ack Received\n");

ack\_recv = 1;

}else{

printf("[-]Ack Not Received\n");

ack\_recv = 0;

}

frame\_id++;

}

close(sockfd);

return 0;

}

**Server Side**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/socket.h>

#include <unistd.h>

#include <arpa/inet.h>

typedef struct packet{

char data[1024];

}Packet;

typedef struct frame{

int frame\_kind; //ACK:0, SEQ:1 FIN:2

int sq\_no;

int ack;

Packet packet;

}Frame;

int main(int argc, char\*\* argv){

if (argc != 2){

printf("Usage: %s <port>", argv[0]);

exit(0);

}

int port = atoi(argv[1]);

int sockfd;

struct sockaddr\_in serverAddr, newAddr;

char buffer[1024];

socklen\_t addr\_size;

int frame\_id=0;

Frame frame\_recv;

Frame frame\_send;

sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);

memset(&serverAddr, '\0', sizeof(serverAddr));

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(port);

serverAddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

bind(sockfd, (struct sockaddr\*)&serverAddr, sizeof(serverAddr));

addr\_size = sizeof(newAddr);

while(1){

int f\_recv\_size = recvfrom(sockfd, &frame\_recv, sizeof(Frame), 0, (struct sockaddr\*)&newAddr, &addr\_size);

if (f\_recv\_size > 0 && frame\_recv.frame\_kind == 1 && frame\_recv.sq\_no == frame\_id){

printf("[+]Frame Received: %s\n", frame\_recv.packet.data);

frame\_send.sq\_no = 0;

frame\_send.frame\_kind = 0;

frame\_send.ack = frame\_recv.sq\_no + 1;

sendto(sockfd, &frame\_send, sizeof(frame\_send), 0, (struct sockaddr\*)&newAddr, addr\_size);

printf("[+]Ack Send\n");

}else{

printf("[+]Frame Not Received\n");

}

frame\_id++;

}

close(sockfd);

return 0;

}

**Output**

**Client side**

gcc client.c -o c

./c 4000

Enter Data: 1234

[+]Frame Send

[+]Ack Received

Enter Data: 0100

[+]Frame Send

[+]Ack Received

Enter Data: -5

[+]Frame Send

[+]Ack Received

Enter Data: abc

[+]Frame Send

[+]Ack Received

**Server side**

gcc server.c -o s

net@inlab:~/Desktop$ ./s 4000

[+]Frame Received: 1234

[+]Ack Send

[+]Frame Received: 0100

[+]Ack Send

[+]Frame Received: -5

[+]Ack Send

[+]Frame Received: abc

[+]Ack Send