<https://www.studytonight.com/computer-networks/stopandwait-protocol>

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

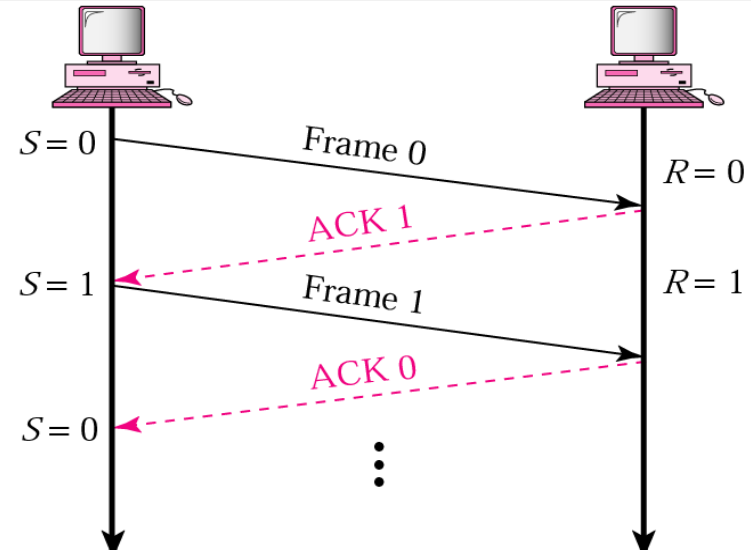
5. Transmit the next frame

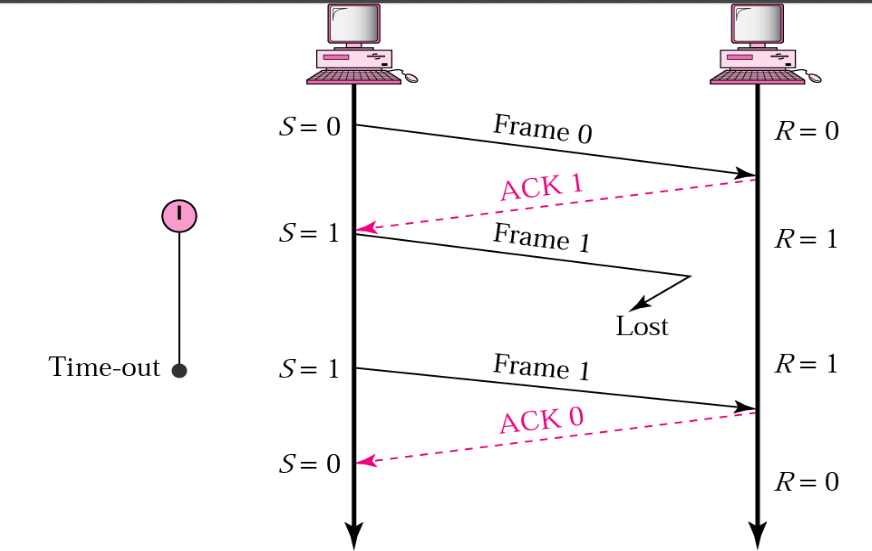
6. Find the remaining frames to be sent.

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();

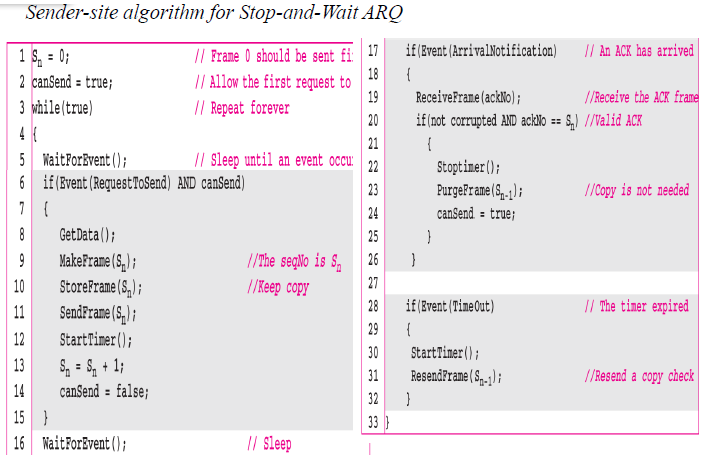
ExtractData();

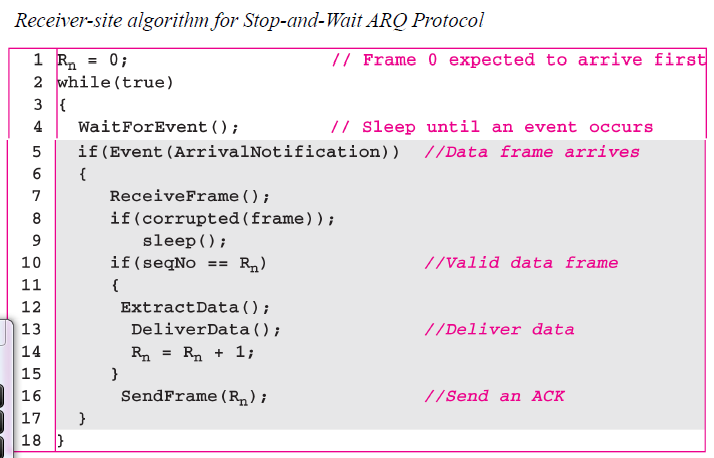
Deliver(data); //delivers the data to the network layer.

SendFrame(); //Send the ACK frame

}

}





**CLIENT SIDE**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#define PORT 8080

void stop\_and\_wait(int client\_socket){

char frame[100];

int seqNo = 0;

char ack[6];

int i=0;

int next = 1;

char exit[4] = {'e', 'x', 'i' ,'t'};

while(1){

int noExit = 0;

if(next==1){

bzero(frame,100);

i=1;

printf("\nEnter message to server : ");

while((frame[i++]=getchar())!='\n');

frame[i-1]='\0';

frame[0] = seqNo;

}

sleep(1.5);

send(client\_socket,frame,sizeof(frame),0); //Send frame

bzero(ack,6); //Empty ack

recv(client\_socket,ack,sizeof(ack),0); //Receive ack

if(ack[0]=='A'){

//If frame acknowledged by server, proceed to next frame by setting next=1

printf("Acknowledgement received from server for frame %d - [",seqNo);

for(int j=1;frame[j]!='\0';j++) printf("%c",frame[j]);

printf("]\n");

seqNo = seqNo==0?1:0;

next = 1;

}

else{

//If frame not acknowledged, resend same frame by setting next =0

next = 0;

printf("Negative acknowledgement received from server for frame %d - [",seqNo);

for(int j=1;frame[j]!='\0';j++) printf("%c",frame[j]);

printf("]\nResending frame %d - [",seqNo);

for(int j=1;frame[j]!='\0';j++) printf("%c",frame[j]);

printf("]\n");

}

//If frame contains the message "exit", exit the network

for(int j=1;j<5;j++){

if(frame[j]!=exit[j-1]) {

noExit = 1;

break;

}

}

if(noExit==0){

printf("Client has successfully exited the network ... \n");

return;

}

}

}

**Server Side**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#define PORT 8080

#define n 4

//For demonstration of loss of frames, server sends negative ack for every nth frame

void stop\_and\_wait(int client\_socket){

char frame[100]; //Initializations

int seqNo = 0;

char ack[6];

int i=0;

int count = 1;

char exit[4] = {'e','x','i','t'};

while(1){

int noExit = 0;

bzero(frame,100); //Empty frame

recv(client\_socket,frame,sizeof(frame),0); //Receive frame from client

bzero(ack,6); //Empty ack

if(frame[0] == seqNo && count%n!=0){

//If expected sequence number received, send ack and change seqNo

ack[0] = 'A';

ack[1] = 'C';

ack[2] = 'K';

ack[3] = seqNo;

ack[4] = '\0';

printf("Frame received from client : ");

for(int j=1;frame[j]!='\0';j++) printf("%c",frame[j]);

printf("\n");

seqNo = seqNo==0?1:0;

}

else{

//If expected sequence number not received, send negative ack

ack[0] = 'N';

ack[1] = 'A';

ack[2] = 'C';

ack[3] = 'K';

ack[4] = seqNo;

ack[5] = '\0';

}

sleep(1);

send(client\_socket,ack,sizeof(ack),0);

//If frame contains the message "exit", exit the network

for(int j=1;j<5;j++){

if(frame[j]!=exit[j-1]) {

noExit = 1;

break;

}

}

if(noExit==0){

printf("Server has successfully exited the network ... \n");

return;

}

count++;

}

}

int main(){

//Initialize socket descriptor

int server\_socket;

//create a socket

server\_socket = socket(PF\_INET,SOCK\_STREAM,0);

if(server\_socket<0){

printf("Error creating socket ...\n");

exit(1);

}

else{

printf("Socket created successfully ...\n");

}

//Binding socket to port

//Initializing sockddr\_in structure before binding

struct sockaddr\_in sa;

bzero(&sa,sizeof(sa));

sa.sin\_family = PF\_INET; //Refers to anything in the protocol

sa.sin\_port = htons(PORT); //Port number 8080

sa.sin\_addr.s\_addr = inet\_addr("127.0.0.7");

if( bind(server\_socket,(struct sockaddr\*)&sa, sizeof(sa))== 0){

printf("Socket binded successfully ...\n");

}

else{

printf("Unable to bind server... An error has occurred \n");

exit(1);

}

//Listen

if (listen(server\_socket,10)==0){

printf("Server listening...\n");

}

else{

printf("Server listen failed\n");

}

//Accept connection

struct sockaddr\_in cli;

int len = sizeof(cli);

int client\_socket = accept(server\_socket,(struct sockaddr\*)&cli,&len);

if(client\_socket<0){

printf("Falied to accept client\n");

exit(1);

}

else{

printf("Server accepted client\n");

}

stop\_and\_wait(client\_socket);

close(server\_socket);

**FIRST EXECUTE SERVER AND THEN ONLY CLIENT**

**Output - Client side**

net@inlab:~$ gcc client.c -o c

net@inlab:~$ ./c

Socket creation successful

Connected to server

Enter message to server : hello

Acknowledgement received from server for frame 0 - [hello]

Enter message to server : welcome to lmcst

Acknowledgement received from server for frame 1 - [welcome to lmcst]

Enter message to server : God Bless you

Acknowledgement received from server for frame 0 - [God Bless you]

Enter message to server : Wow!

Negative acknowledgement received from server for frame 1 - [Wow!]

Resending frame 1 - [Wow!]

Acknowledgement received from server for frame 1 - [Wow!]

Enter message to server : Bye

Acknowledgement received from server for frame 0 - [Bye]

Enter message to server : exit

Negative acknowledgement received from server for frame 0 - [exit]

Resending frame 0 - [exit]

Client has successfully exited the network ...

**Output - Server side**

net@inlab:~$ gcc server.c -o s

net@inlab:~$ ./s

Socket created successfully ...

Socket binded successfully ...

Server listening...

Server accepted client

Frame received from client : hello

Frame received from client : welcome to lmcst

Frame received from client : God Bless you

Frame received from client : Wow!

Frame received from client : Bye

Frame received from client : EXIT

Server has successfully exited the network ...

net@inlab:~$