PROGRAM 3

- ASHISH KUMAR
- 2K18/SE/041

Aim:- Write a C++ program to implement Stop and Wait Protocol.

Theory:- Stop-and-wait ARQ, also referred to as alternating bit protocol, is a method in telecommunications to send information between two connected devices. It ensures that information is not lost due to dropped packets and that packets are received in the correct order.

It is the simplest automatic repeat-request (ARQ) mechanism. A stop-and-wait ARQ sender sends one frame at a time; it is a special case of the general sliding window protocol with transmit and receive window sizes equal to one and greater than one respectively. After sending each frame, the sender doesn't send any further frames until it receives an acknowledgement (ACK) signal. After receiving a valid frame, the receiver sends an ACK. If the ACK does not reach the sender before a certain time, known as the timeout, the sender sends the same frame again. The timeout countdown is reset after each frame transmission. The above behavior is a basic example of Stop-and-Wait. However, real-life implementations vary to address certain issues of design.

CODE:-

```
#include<iostream>

#include<bits/stdc++.h>

#include<dos.h>

#include <unistd.h>

#define time 5

#define max_seq 1

#define tot_pack 5

using namespace std;
```

```
int randn(int n)

{

   return rand()%n + 1;

}

typedef struct

{

      int data;

}packet;

typedef struct

{

       int kind;

       int seq;

       int ack;

       packet info;

}frame;

typedef enum{ frame_arrival,error,time_out}event_type;

frame data1;

//creating prototype

void from_network_layer(packet *);

void to_physical_layer(frame *);

void to_network_layer(packet *);

void from_physical_layer(frame*);

void sender();

void receiver();

void wait_for_event_sender(event_type *);
void wait_for_event_receiver(event_type *);


#define inc(k) if(k<max_seq)k++;else k=0;


int i=1;

char turn;

int disc=0;

int main()

{

  while(!disc)

  {  sender();

     //delay(400);

     receiver();

  }

   getchar();

}

void sender()

{

    static int frame_to_send=0;

    static frame s;

    packet buffer;

    event_type event;

    static int flag=0;      //first place

    if (flag==0)

    {
```

```
   from_network_layer(&buffer);
   s.info=buffer;
   s.seq=frame_to_send;
   cout<<"\n\nSender information : "<<s.info.data<<"\n";

   turn='r';
   to_physical_layer(&s);
   flag=1;
     }
 wait_for_event_sender(&event);
 if(turn=='s')
 {
     if(event==frame_arrival)
      {
      from_network_layer(&buffer);
      inc(frame_to_send);
      s.info=buffer;
      s.seq=frame_to_send;
      cout<<"\n\nSender information : "<<s.info.data<<"\n";
      turn='r';
      to_physical_layer(&s);
      }
 }
}                    //end of sender function
```

```
void from_network_layer(packet *buffer)
{
     (*buffer).data=i;
     i++;
}                              //end of from network layer function

void to_physical_layer(frame *s)
{
     data1=*s;
}              //end of to physical layer function

void wait_for_event_sender(event_type *e)
{
      static int timer=0;
      if(turn=='s')
      {   timer++;
           return ;
  }
  else                            //event is frame arrival
    {
       timer=0;
       *e=frame_arrival;
    }
}               //end of wait for event function
```

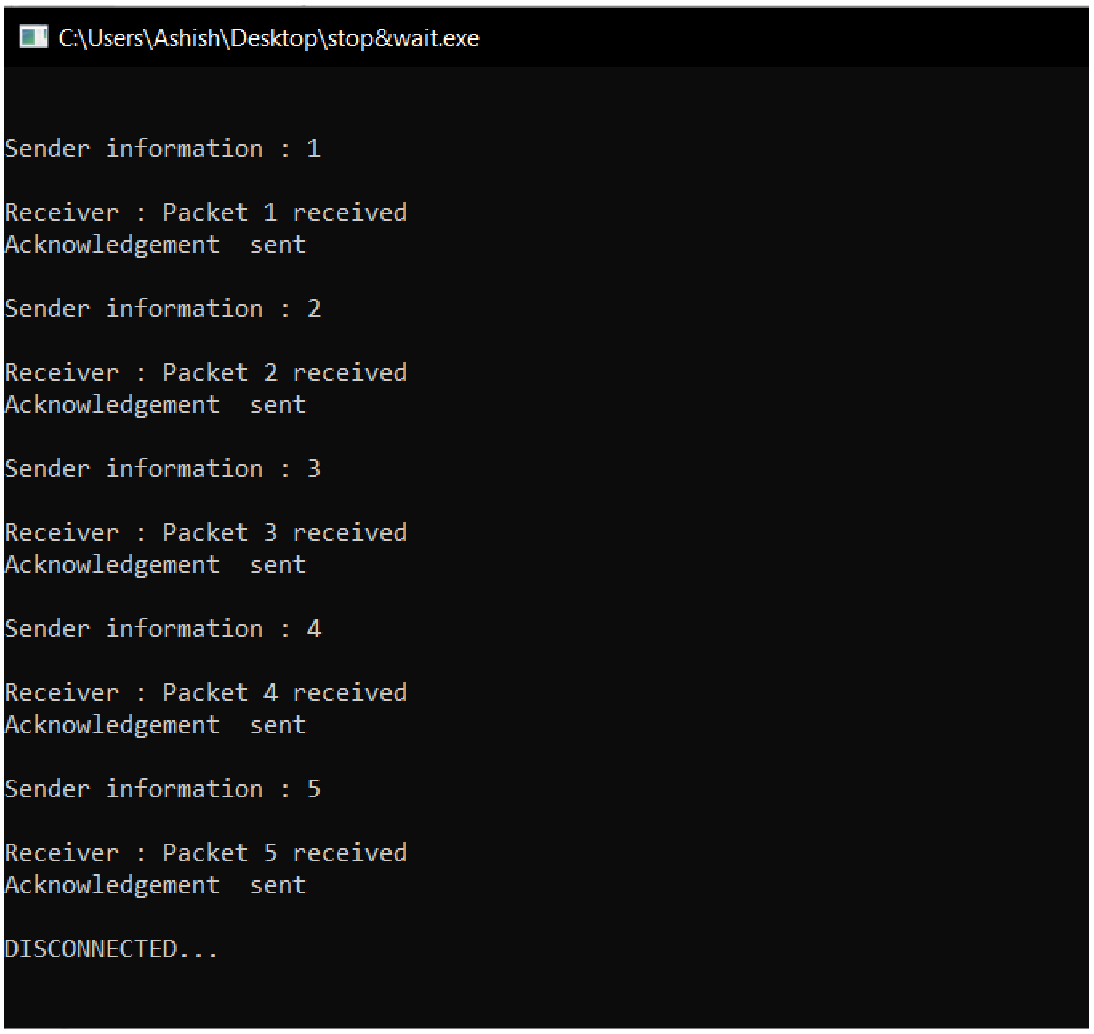
```
void receiver()
{
     static int frame_expected=0;
     frame s,r;
     event_type event;
     wait_for_event_receiver(&event);
     if(turn=='r')
     {  if(event==frame_arrival)
    {
          from_physical_layer(&r);
          if(r.seq==frame_expected)
    {
      to_network_layer(&r.info);
      inc (frame_expected);
    }
    else
    cout<<"\nReceiver :Acknowledgement resent \n";
    turn='s';
    to_physical_layer(&s);
      }
     }
}                        //end of receiver function
```

```
void wait_for_event_receiver(event_type *e)
{
    if(turn=='r')
    {
                *e=frame_arrival;
    }
}

void from_physical_layer(frame *buffer)
{
   *buffer=data1;
}

void to_network_layer(packet *buffer)
{
    cout<<"\nReceiver :"<<" Packet "<<i-1<<" received \t";
    cout<<"\nAcknowledgement  sent ";
    if(i>tot_pack)
     {          disc=1;
                cout<<"\n\nDISCONNECTED...\n";
     }
}
```

OUTPUT:-



```
C:\Users\Ashish\Desktop\stop&wait.exe


Sender information : 1

Receiver : Packet 1 received
Acknowledgement  sent

Sender information : 2

Receiver : Packet 2 received
Acknowledgement  sent

Sender information : 3

Receiver : Packet 3 received
Acknowledgement  sent

Sender information : 4

Receiver : Packet 4 received
Acknowledgement  sent

Sender information : 5

Receiver : Packet 5 received
Acknowledgement  sent

DISCONNECTED...
```

Learning Outcome:- We have successfully implemented stop and wait protocol and we learnt its application in computer network. The Stop and Wait ARQ solves main three problems, but may cause big performance issues as sender always waits for acknowledgement even if it has next packet ready to send. Consider a situation where you have a high bandwidth connection and propagation delay is also high (you are connected to some server in some other country though a high speed connection). To solve this problem, we can send more than one packet at a time with a larger sequence numbers. So Stop and Wait ARQ may work fine where propagation delay is very less for example LAN connections, but performs badly for distant connections like satellite connection.