CS 252 Assignment 3

Team

- B.Karthikeya 190050026
- D.Chandra Sekhar 190050031
- K.Vishwanth 190050131

Instructions for running the code

- Primary Note: This code is best run on a Ubuntu O.S.
- **Compile**
    - compile sender.cpp using g++ -o sender.out sender.cpp
    - compile receiver.cpp using g++ -o receiver.out receiver.cpp
- **Execute**
    - ./sender.out `<SenderPort>` `<ReceiverPort>` `<RetransmissionTimer>` `<NoOfPacketsToBeSent>`
    - ./receiver.out `<ReceiverPort>` `<SenderPort>` `<PacketDropProbability>`
- **Some Miscellaneous Notes**
    - Retransmission timer should be mentioned in seconds
    - It's best to start executing receiver.out before starting the execution of sender.out
    - receiver.out has to manually terminated after receiving all packets(This issue is discussed on Teams and it is told that this is fine).
    - On executing the programs two files named receiver.txt and sender.txt get created. Required outputs are printed to the console and also will be written to those text files

Description of the code

- ### Sender.cpp
    - This code is mainly divided into three function.
    - **int SetConnection(SenderPort,ReceiverPort)**
        - SenderPort is the port number from which you are going to send your messages and ReceiverPort is the port from which you receive messages.
        - The function creates a scoket and then binds the socket to the SenderPort(using standard bind fucntion) and connects to the ReceiverPort(using standard connect fucntion)
        - Finally the function returns the socketdescriptor of the socket that is created (which is binded to the SenderPort) and this socket descriptor is used when sending messages from the sender's end and receiving acknowledgements from the receiver's end.

- **bool ReceivePacket(socketdescriptor,expected_number,Retransmission time)**
  - Here we used time.h to implement the Retransmission Timer.
  - A `while loop` is run which terminates when the time for which the loop is executed exceeds Retransmission timer or right packet is received before timer expires.
  - In while loop we try to check if we can receive any packet
    - If number of bytes received is zero then we continue
    - If not zero then we check if the packet arrived is correct one.
      - If correct one then return true
      - else continue in the `while` loop untill the timer expires
  - If no correct packet received before timer expires return `false`;
  - receving packet is implemented using standard `recv` function.
  - standard `recv` function waits untill the packet arrives which sometimes maybe too long. So we use `ioctl()` to quickly check if there is something that the socket is receiving and if **yes** then we will use the `recv` function to get that data
  - **Note** The code for `ioctl()` part is taken from here - Link
- **void sendPacket(socketdescriptor,packetnumber)**
  - This function just uses the standard `send` function to send the messages.
  - This function doesn't return anything
- Integrating these functions is quite trivial and is done by `main()`

Receiver.cpp

- This code is mainly divided into five functions.
- **int SetConnection(SenderPort,ReceiverPort)**
  - This is exactly same as the SetConnection of Sender.cpp(), here the function returns the socketdescriptor of the socket which is created and binded to the ReceiverPort, this is used when receiving messages from sender's end and sending acknowledgements from the receiver's end
- **seed_uniform_generator**
  - This basically seeds a random variable generator
- **double get_random()**
  - This returns a random number between 0-1 from a uniform distribution.
  - This uses `unif()` function of c++ and the random number generator that is seeded above
  - **Note** The code for random number generation is taken from here How to Generate Uniform Random Numbers
- **void receive_packet(socketdescriptor,reqd_packetnumber,packetdrop_probability)**
  - A infinite `while loop` is run which sees if there is any incoming packet using `ioctl()`
    - If no such packet continue in the while loop
    - Else use standard `recv()` function to get the packet

        - If we receive a packet then check if its the required packet
            - If not ignore and continue in the while loop
            - Else pick a random number between 0-1.
              Check if this number < packetdrop_probability
                - If yes then drop the packet i.e just ignore and continue in the while loop
                - Otherwise send the new Acknowledgement using send_packet function and keep waiting for the new reqd numbered packet for which we have sent the acknowledgement
- **void send_packet(socketdescriptor,Acknumber)**
    - This function just uses the standard `send` function to send the message.
    - This function doesn't return anything
- Integrating these functions is quite trivial and is done by `main()` function