Lab 3 - Socket Programming

Description:

This lab covers the basics of socket programming.

Two files are included, namely `sender.c` and `receiver.c`. This lab simulates communication between the server and receiver programs over a network. Since both programs run on the same machine, they communicate via the loopback interface. We emulate network delays and packet losses between the two of them using delays in the loopback device and randomly dropping some packets respectively.

The sender transmits a packet, and starts a wait timer to receive the acknowledgement. If the right acknowledgement is received within that time, the sequentially next packet is transmitted. If an incorrect acknowledgement is received, it is ignored. If the timer expires, the packet is retransmitted and the timer is restarted.

The receiver listens for packets at its port. If a packet with an unexpected sequence number is received, then an acknowledgement with the expected number is sent out. If the sequence number is correct, then the packet is dropped with some predecided probability (given by the user), and otherwise the packet is accepted and an acknowledgement with the next expected sequence number is sent out.

This process is repeated until all packets (number of packets decided by user) have been transferred from sender to receiver.

In `sender.c`, the main function uses calls to the function `sendMessage` to prepare data packets with appropriate sequence numbers and transmit them to the receiver program, and the function `receiveACK` to process the received acknowledgements and perform necessary operations. The `ms_to_wait_for` variable is used to handle the acknowledgement-wait timer, and is dynamically set and reset in all possible cases, i.e., correct/incorrect/no acknowledgements. `SEQ_NO` is used to maintain the current sequence number of the packet to be transmitted. It is incremented after each successful round of transmission and acknowledgement. The sender program also uses `poll()` system calls (reference) to listen in on the socket for acknowledgements.

In `receiver.c`, the main function uses calls to `receivePacket` to process the received packets from the sender program and print corresponding output, and the `sendACK` function to send acknowledgements to the sender program. The main function uses a random number generator and the given probability threshold to decide whether to drop the packet or accept it. `EXPECTED_SEQ_NO` is used to maintain the next expected packet number. This is the number sent out on acknowledgments, and is incremented every time a packet is accepted.

All outputs along with timestamps (in terms of local device time) are printed to the terminal windows as well as to two files named `sender.txt` and `receiver.txt`, using appropriate `printf` and `fprintf` macros respectively (reference).

Running the Code:

First, we emulate network delays using:

```
$ sudo tc qdisc add dev lo root netem delay <delay in milliseconds>
```

For example, you may replace `<delay in milliseconds>` with `100ms` in the above line.

Next, compile the two C programs, corresponding to sender and receiver respectively, as follows:

```
$ gcc sender.c -o sender
$ gcc receiver.c -o receiver
```

Finally, use a separate terminal window to run each of the output files. In one terminal window, run:

```
$ sudo ./sender <SenderPort> <ReceiverPort> <RetransmissionTimer> <NoOfPacketsToBeSent>
```

where:

SenderPort
: the port number to be used by the sender program

ReceiverPort
: the port number to be used by the receiver program

RetransmissionTimer
: the number of seconds that the sender should wait for an acknowledgement before retransmitting (should be at least two times the emulated network delay)

NoOfPacketsToBeSent
: the total number of packets to be transmitted by sender and received by receiver

As soon as this is done, the sender starts transmitting its first packet. This packet is transmitted repeatedly for now, since there is no active receiver to send back acknowledgements. Now, in a separate terminal window, run:

```
$ sudo ./receiver <ReceiverPort> <SenderPort> <PacketDropProbability>
```

where:

ReceiverPort
: the port number to be used by the receiver program

SenderPort
: the port number to be used by the sender program

PacketDropProbability

the probability (between 0 and 1) with which each packet received by the receiver gets dropped (here, this is used simply to simulate the loss of packets in real network communication, which may occur due to various real world factors)

Once both the sender and the receiver are up and running, packets and acknowledgements are transmitted and received, and all program outputs are printed to the terminal windows as well as to two files named `sender.txt` and `receiver.txt` respectively, with appropriate time stamps too.

We have included sample output files for one of our test runs in this directory. This run was conducted with an emulated delay of 100ms, retransmission timer of 300ms, 15 total packets, and a packet drop probability of 0.3.