EE414 Embedded Systems
Lab 5. Network Input Output

***Due** Demo 4 – 6 PM, Nov 30, Thu.*
Report 6 PM, Dec 5, Tue.

1. Purpose

Understand how to use Ethernet and program a typical network program using sockets, and implement a remote user command input for the metronome.

2. Problem Statement

Problem 5 (Network Input)

Design and implement a metronome which can respond to the remote user command input from remote PC or notebook via network, specifically using Ethernet stream socket using TCP.

The "metroserver" program is a server program in the embedded board waiting for service request of metronome. The client program named "'metroclient" in the remote PC/notebook connects to the "metroserver" in the embedded board via wired/wireless Ehternet, and sends command strings to start/stop the metronome according to user inputs (with menus as in Lab 3).

Then "metroserver" interprets the user command, and outputs to the "metrocilent" using text string – the results of interpretation and also beat signal strings. The remote user can see the metronome playing via character string display on the display in the remote PC.

If the command is "start", the "metroserver" should output the following text string infinite times with timing given by "tempo" and "time-singature".

Character pattern

Time signature 2/4:	# !
Time signature 3/4:	# ! !
Time signature 4/4:	# ! + !
Time signature 6/8:	# ! ! + ! !

3. Technical Backgrounds

A. Hardware connection

Beaglebone board has one Ethernet port whose speed is 1 Gbps max. This Ethernet port can be connected to the Ethernet port in the PC using a Router as shown in Fig. 1.

Note that the remote PC can be replaced by a notebook computer. In this case, connection from the notebook to the Network can be wireless (WiFi).

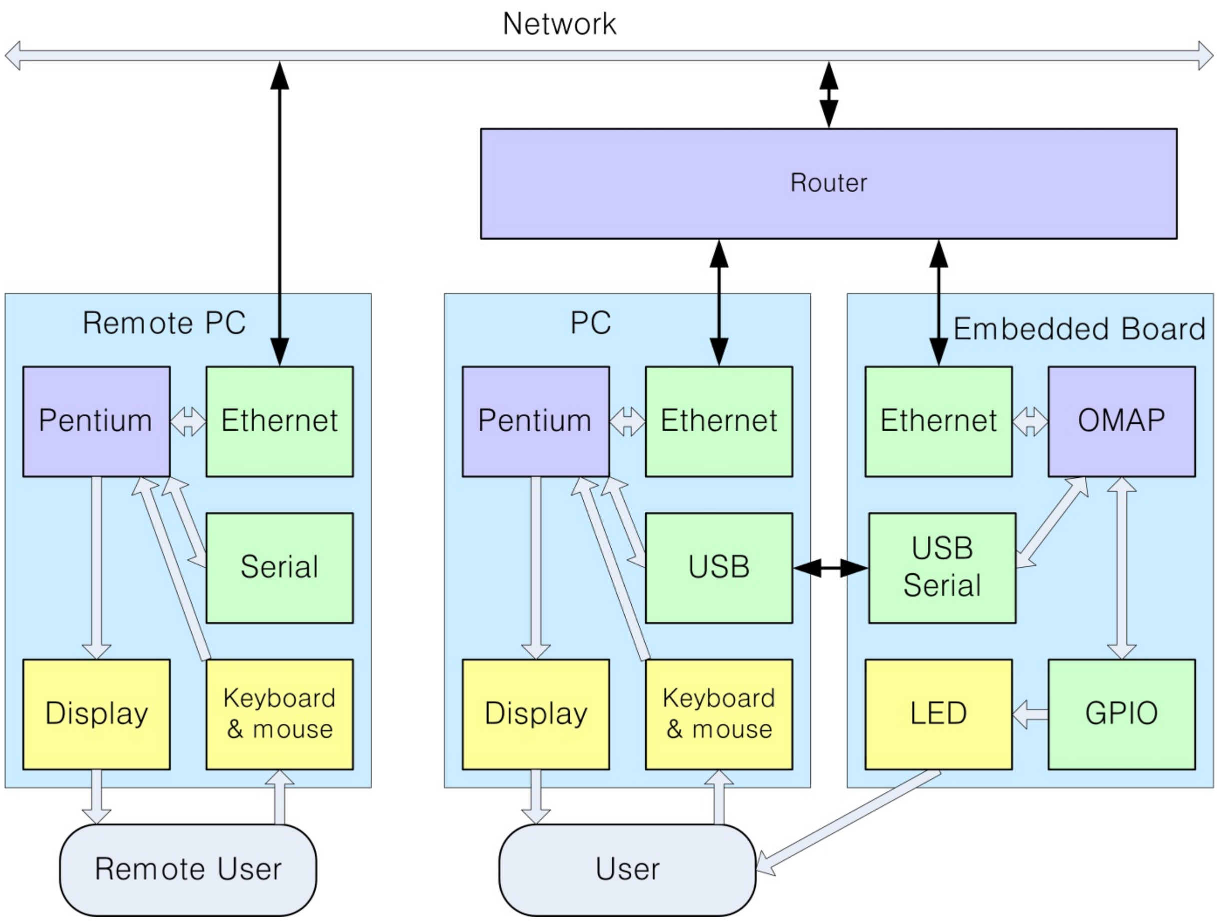

Fig. 1 Block Diagram for Lab 5

B. Server and client program

Use the stream (not datagram) socket for Ethernet communication between PC and embedded board. Refer to "Socket programming How To" and “Beej‵s Guide to Network Programming” in the web. Especially, check the flow of the programs named Server.c (a simple stream server) and Client.c (a simple stream client), and understand the socket functions called.

From “Beej‵s Guide to Network Programming”:

● server.c	A simple stream server
● client.c	A simple stream client

A simple stream server		**A simple stream client**
1. Open a stream socket		
2. Set socket option		
3. Bind socket_id to socket structure		
4. Listen		
5. Loop		
		1. Get the host information
		2. Open a stream socket
5A. Wait accept incoming connection request…		
5B. Accepted: Create a child process.	←	3. Connect to the server
5C. Parent still remains within the accept loop.		
Child process:		
		4. Recv() waiting…
1. Send a message to the socket	→	Receive a message from the socket.
		5. Print the received message.
2. Close the socket.		6. Close the socket

C. Datagram socket

From "Beej's Guide to Network Programming":

● listener.c	Datagram listener on port 4950
● talker.c	Datagram talker to port 4950

listener		**talker**
1. getaddrinfo()		
2. Open a datagram socket and bind address		
3. freeaddrinfo()		
4. recvfrom()		
		1. getaddrinfo()
		2. Open a datagram socket
	←	3. Sendto()
4'. Recvfrom() returned.		
		5. freeaddrinfo()
9. Close the socket.		9. Close the socket

D. Bidirectional stream socket

Can you implement two-way communication using stream socket?

If they have the same rate and synchronized, it is easy to implement:

Server child	Network	Client
Loop		Loop
recv()	←	send()
send()	→	recv()

Suppose two-way communications have different rate and unsynchronized, such as Metronome:

Client to server:	When user input happened asynchronously.
Server to client:	Periodic output depending on tempo and time-signature.

In this case, we use the thread as in Lab 2.

Main: Client sends a message and server receives the message.

Thread: Server sends messages and client receives messages in higher rate.

We are going to test with two programs:

ClientBD.c		**ServerBD.c**
		init_thread
		socket()
		accept()
init_thread		
socket()		
connect()		
		Child
Loop send	→ →	Loop recv
Thread		***Child_Thread***
Loop recv fast	← ← ← ←	Loop send fast

We test main communication 5 times with 1 Hz, and thread communication 10 times with 2 Hz.
Also we change main communication rate according to user input.

4. Design and Preparation

1. Stream socket

   Understand the server and client programs in "Beej's Guide to Network Programming" in the internet.

2. Find the meaning of the following IP addresses:

   143.248.1.177
   192.168.0.1
   127.0.0.1

3. Bidirectional stream socket

   Design the program serverBD.c and clientBD.c

Change main communication rate according to user input.

4. Design Metronome server and client

System design

Hardware	**PC**	**Router**	**Bone**
Software			
App	MetroClient		MetroServer
Comm	Stream socket		Stream socket
Network		Ethernet	

MetroClient		**MetroServer**
Input: User key input without enter key.	
Process user input to command packet.	
Output: Command packet to stream socket	→	Input: Command packet from stream socket
	Parse command packet and run

Metronome

Thread		***HRTimer signal_handler***
Loop		Play Metronome one-half note via LEDs
Input: Character for Metronome strength	← ←	Output: Character for Metronome strength.
Output: Display received characters		

Command packet to MetroServer

ASCII string

"TimeSig A/B, Tempo NNN, Start" or
"Stop" or
"Quit"

Metronome output packet for Metronome strength (in-time)

"#" or "+" or "!" Single character packet.

MetroServer.c

Files

metro_server.c	with signal_handler
gpio_led_fu[_sim].c	GPIO_LED routines (for Bone/PC)
serve.c	Socket server routines

Algorithm

```
1. Init GPIO LED
    Init HR timer
        Create signal handler for HR Timer - metronome processing
2. Init stream socket server

5. Loop
  Wait accept and connection.
  In Child process
        A. Get the command packet
        B. Enable HRTimer: Play Metronome via sig_handler
        D. Print single line message: Input & Status (without linefeed)
```

sig_handler (HR Timer)

Play one-half note (on or off pattern)
Send reply packet of pattern.

Make two versions: PC and Bone

MetroClient.c

Files

metro_client.c
key_input_fu.c
clien.c

Algorithm

```
1.  Init_clien()    Init socket client
    Init Thread (for socket input to display output2)
2. Init key processing
       Set termios
       Print title & menu.
3. Set default values to parameters (TimeSig 3 (3/4), Tempo 90, Stop)
       Print default values.

5.  Loop
        A. Get user input key without enter in blocking mode.
        B. If 'q' break
        C. Interpret the key
              If 'z' (Time-signature)
                    inc TimeSig
                    If TimeSig >= 4 TimeSig = 1         (Rotating)
              If 'c' (Dec Tempo)
                    Tempo = Tempo – 5
                    If Tempo < 30 Tempo = 30
              If 'b' (Inc Tempo)
                    Tempo = Tempo + 5
                    If Tempo > 200 Tempo = 200
              If 'm' (Start/Stop)
                    Start = 1 if stop, 0 else
                    start/stop HR timer
        D. Print single line message: Input & Status (without linefeed)

7. Print quit message
8. reset termios
```

Thread

```
Loop
        recv metronome packet
        Display metronome packer character.
```

5. Experiment Procedures

Step 1. Test stream socket

Native-compile and run the server and the client using two windows in PC Linux.

Server on PC ←→ Client on PC

Cross-compile the server and native-compile the client and run on the embedded board and a PC respectively, with Ethernet connection.

Server on Bone ←→ Client on PC

Step 2. Test datagram socket

Native-compile and run the listener and the talker using two windows in PC Linux.

Listener on PC ←→ Talker on PC

Cross-compile the listener and native-compile the talker and run on the embedded board and a PC respectively, with Ethernet connection.

Listener on Bone ←→ Talker on PC

Step 3. Test bidirectional stream socket

3.1 Test serverBD and clientBD

Test serverBD.c and clientBD.c both on PC
Does bidirectional communications works as expected?

Test serverBD.c on Bone and clientBD.c on PC
Does bidirectional communications works as expected?

Note:
Time sync required.
Set activated after server recv the first cmd packet.
Start send reply in thread only after activated.

Allow time to send reply after the last recv.

3.2 Test serverBD and clientBD with user input

Add single key input to clientBD.c to make main communication asynchronous.

Note
accept()
accept(): Blocking system call.
Returns errno of EINTR: The system call was interrupted by a signal that was caught
before a valid connection arrived; see signal(7).
Need to modify suitably.

recv()
recv() is also a blocking call.
Need to modify suitably.

Step 4. Metronome server and client

Since Metronome server and client are fairly complex program, step-by-step debugging is necessary:

Test both on PC

A. metro_client_pc1		metro_server_sim2	TUI, Print command packet
B. metro_client_pc2	>	metro_server_sim2	TxRx cmd, Action
C. metro_client_pc2	>	metro_server_sim3	Gen reply & Print reply packet.
D. metro_client_pc4	><	metro_server_sim4	TxRx reply, Display reply.

Test server on Bone and client on PC

E. metro_client_pc4	><	metro_server_bone	TxRx reply, Display reply.

6. Demonstration

Demonstrate the result of the Step 4E.

7. Report

Each student should prepare his own report containing:
Purpose
Experiment sequence
Experimental results
Discussion: should be different even for each member of the same team.
References.

Include in the Discussion:

1. Suppose multiple MetroClients wants to connect to the Metronome server. We have only one resource of Metronome service. How can you handle this situation?

2. Question: Suppose you are going to replace the notebook computer to an Android smartphone. Is this possible? Discuss the required functionality of Remote MetroClient App.

8. References

[1] Beej's Guide to Network Programming, Internet.