[EE414] Embedded System
Lab5 Report

20180480 Woojin, Lee

1. Purpose
This lab is about the networking paradigm with socke, endpoint of network communication. You are going to run the metroServer file at Beaglebone board, and serve user input at PC with metroClient file. We have to construct bidirectional communication between metroServer and metroClient files with sockets.
Also, you are going to use the timer, thread and process concept again for this lab.

2. Experiment sequence
First of all, to enhance the knowledge of socket programming, we are going to analyze skeleton code from "Beej's guide to network programming". Starting from the provided code server.c and client.c, you can learn the concept of stream socket, and then learn the concept of datagram socket with talker.c and listner.c. In this process, you can learn various APIs of sockets such as making a socket, binding, connecting, listening, etc...

Then, you are going to implement a bidirectional communication program, which is the combination of socket network and process/thread. Provided server.c and client.c codes are for single-direction communication(server sends, and client receives.) You can also check whether this scheme also works for beaglebone-PC networks.

After you get familiar with network programming, you are going to implement metronome with network! Regard your beaglebone as server, and PC as client. PC will get the user's input and send it to the beaglebone board. Beaglebone will generate periodical beat signals, and show it to GPIO_LED and PC.

3. Experimental results
<Stream socket and Datagram socket>
Stream socket is for TCP connection, and Datagram socket is for UDP connection. I implemented bidirectional code using pthread.

<Bidirectional code>
I implemented bidirectional code using stream socket. I compressed the file because there are lots of files(bidirectional_20180480_WoojinLee.tar.gz). Instead, I made a Makefile so that you can easily run my file. Note that I compiled in a way that the server file runs on a beaglebone board. For user input in client.c file, type ./client (hostname) (tempo). In my case:

```
./client 192.168.0.39 10
```

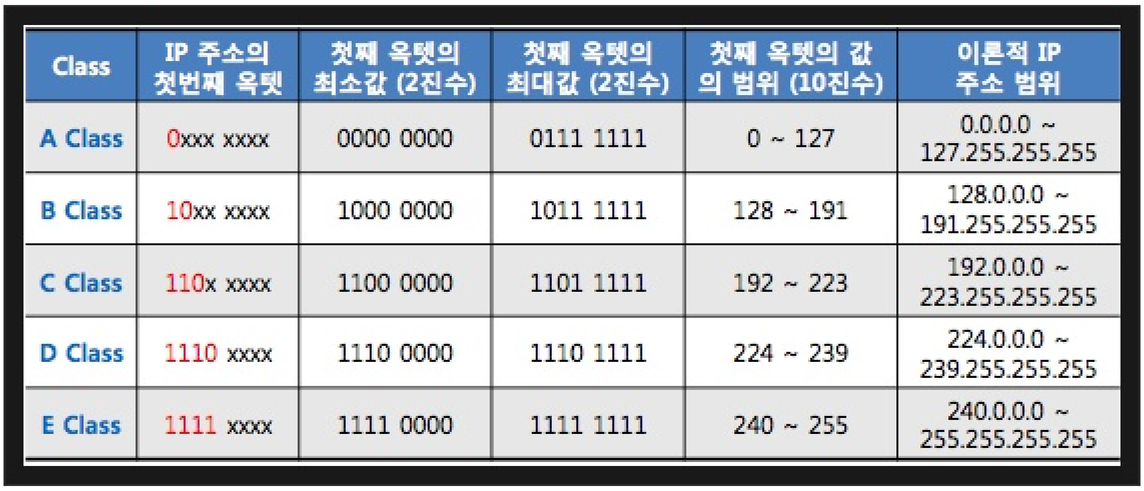
<Main code>
I attached a demo video. I had to compress the file because there are lots of files. Instead, I made a Makefile so that you can easily run my file. To compile, unzip metronome_20180480_WoojinLee.tar.gz and type:

```
make clean && make
```

Client calculates signal information and sends it to the server. This message contains: command(Quit, Stop), timesignal, and beat.
Server will interpret this message and operate(timer setting, turning on LED, sending back #!+ to the client) properly. First, you make a socket, bind, and listen. Also, you set SIGALRM so that you can use the timer properly. When a new client tries to make a connection, the server fork and handle that condition in the child process. Child will interpret the packet, and set a timer. Using the handler, you can control the GPIO LED and send back information to the client.

4. Discussion

- **Find the meaning of the following IP addresses:**
  143.248.1.177, 192.168.0.1, 127.0.0.1
  > IP address is the 32 bit identifier for host, router interface. Here, interface means a connection between host/router and physical link. Routers typically have multiple interfaces. Also, host typically has one or two interfaces. IP addresses are associated with each interface.
  Address format is a.b.c.d./x, where x is number of bits in the subnet portion of the address (called network prefix). Inside the subnet, devices have the the same subnet part of IP address, so they can physically reach each other without intervening router. Each IP address have network ID and host ID. Network ID is there to easily manage every hosts. Host ID is for manipulating hosts individually.
  There are some classes of IP address, A, B, C, D, and E. Common things are A, B, and C.



Class	IP 주소의 첫번째 옥텟	첫째 옥텟의 최소값 (2진수)	첫째 옥텟의 최대값 (2진수)	첫째 옥텟의 값의 범위 (10진수)	이론적 IP 주소 범위
A Class	0xxx xxxx	0000 0000	0111 1111	0 ~ 127	0.0.0.0 ~ 127.255.255.255
B Class	10xx xxxx	1000 0000	1011 1111	128 ~ 191	128.0.0.0 ~ 191.255.255.255
C Class	110x xxxx	1100 0000	1101 1111	192 ~ 223	192.0.0.0 ~ 223.255.255.255
D Class	1110 xxxx	1110 0000	1110 1111	224 ~ 239	224.0.0.0 ~ 239.255.255.255
E Class	1111 xxxx	1111 0000	1111 1111	240 ~ 255	240.0.0.0 ~ 255.255.255.255

  So, we can know:
  143.248.1.177 => Type B. Network ID: 143.248, Host ID: 1.177
  192.168.0.1 => Type C. Network ID: 192.168.0, Host ID: 1
  127.0.0.1 => Type A. Network ID: 127, Host ID: 0.0.1

- **Suppose multiple MetroClients want to connect to the Metronome server. We have only one source of Metronome service. How can you handle this situation?**
  > The breakthrough for this situation is threading. Although I didn't consider this situation in my lab, you can solve this issue by creating a thread whenever a new client arrives. Also, note that most kernels don't provide multiple bind() to an already

allocated port. You have to elaborately designate a proper port number to each client so that you can communicate as desired.
However, in this lab, physical resources(e.g., LED) are restricted. So, in this case, regarding everything in a concurrency manner may not be a good idea. Alternative way is pending incoming new clients while handling another client. You can think of the way of saving their sockfd in the buffer and starting them right after current metronome task is done.

- **Suppose you are going to replace the notebook computer with an Android smartphone. Is this possible? Discuss the required functionality of Remote MetroClient App.**
  > Since I have no experience with Android application programming, these may be incorrect.
  I think it would be possible if some requirements are fulfilled:
  First, we need a proper compiler so that our code can work for Android. Perhaps writing code with JAVA or Kotlin will help this issue.
  Second, we have to resolve the issue of routing. I think we can do it by controlling the router.
  Also, there was Android's property that it forbids placing any networking operation in the main UI since it has to interface with the user. You have to place networking code within a \`Thread\` or an \`AsyncTask\`. Also, note that you have to transmit through a timer because you can't receive data if you use an infinite loop.

5. References

Beej's Guide to Network Programming, Internet.
TCP Socket Android Client
https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=wndrlf2003&logNo=220110992861