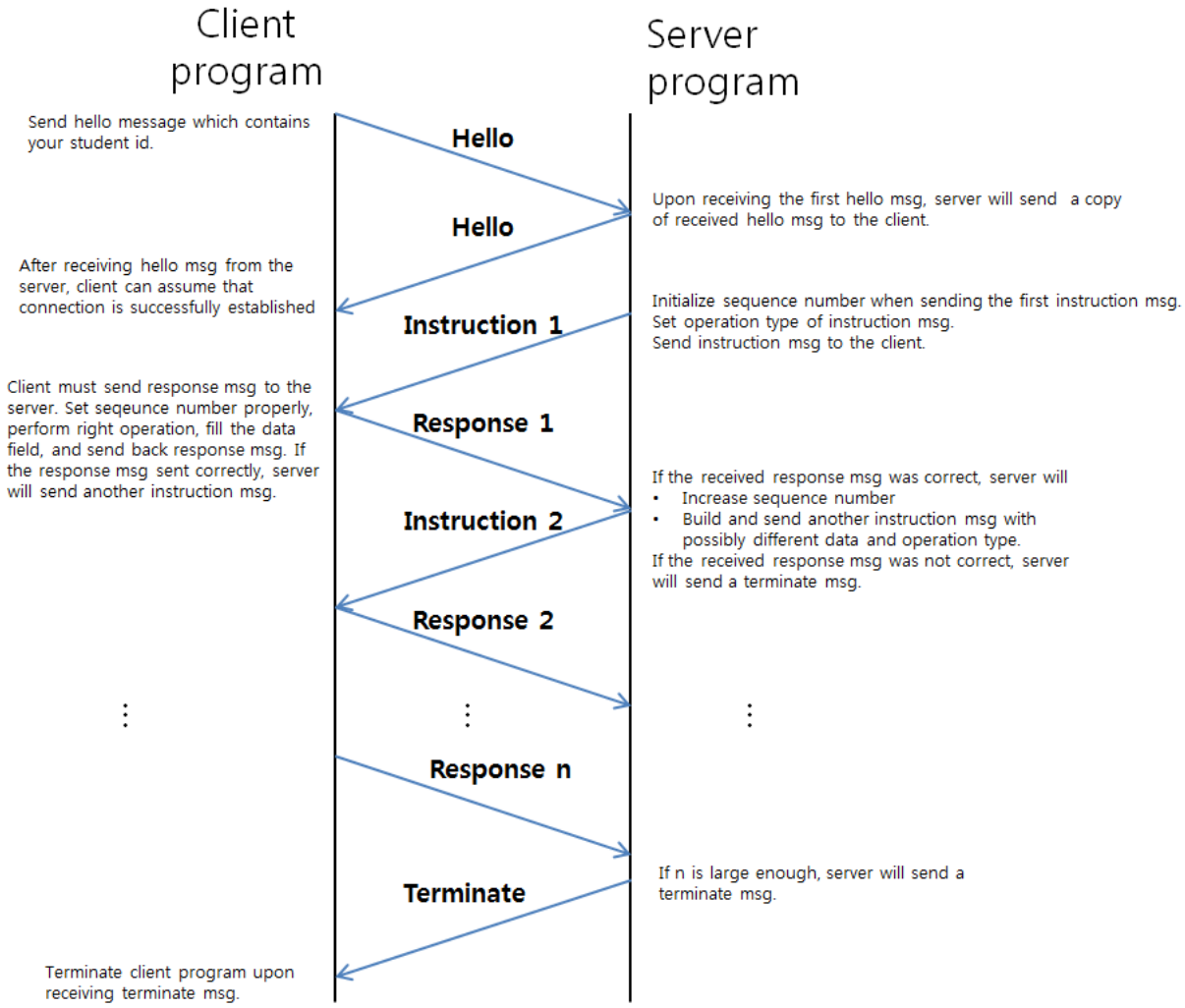
Socket Programming HW2

We will upload a server executable file on the BlackBoard. You can check whether your code is correct or not in the Ubuntu terminal using the server executable file.

0) Overview:

The goal of Socket programming HW2 is to implement below simple application layer protocol. We will name it **IR (Instruction-Response) protocol**.



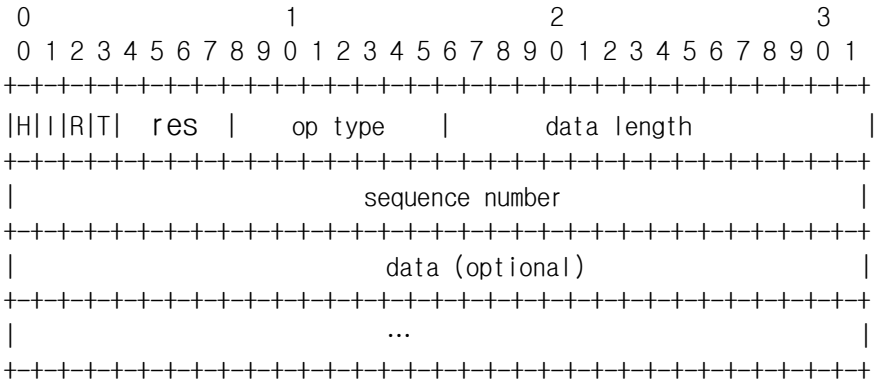
In **IR protocol**, four message types are defined : Hello, Instruction, Response, and Terminate. When executed, the server program will wait for a hello message from client side. If the server receives a hello message, server will send a hello message to the client program.

After sending a message to the server, client must wait for a message from server. When the client received instruction message, client should parse the message, extract operation type,

operand and data value, do calculation according to the received instruction message's operation type, and send result with response message.

After receiving a response message from client, the server may send another instruction message or a terminate message. If the client receives a terminate message, the client program should terminate.

1) Packet structure of IR protocol.



In IR protocol, the data is exchanged following the above packet format.

- Each tick mark represents one bit position.
- The number above represents bit index.

- The header is 64 bits (8 bytes), up to 'sequence number' field.
- There may exist data which follows 'sequence number' field. The length of this data is arbitrary, which is specified by 'data length' field.

- First 4 bits (H, I, R, and T) represents message types of IR protocol, HELLO, INSTRUCTION, RESPONSE and TERMINATION. If one of these 4 bits is set to 1, the others must be set to 0, in other words, the first 4 bits cannot have values other than 0b0001, 0b0010, 0b0100 and 0b1000.
- Second 4 bits are reserved. Set these bits to 0.
  - Next 8-bit field is op type field. When the server sends an instruction message to the client program, the value of op type field designates type of operation the client program must perform.
    ① Op type 0x00 : This represents ECHO request. The server will send a null-terminated string to the client. Client must build a send the same string in the instruction message. Set message type to response and set op type field to 0x00 whenever you response to the server.
    ② Op type 0x01 : This represents INCREMENT message. The sever will send a 4-byte random integer number to the client with this message. Client

must add 1 to received 4-byte binary number and send back the resulting 4-byte number to the server using response message.

③ Op type 0x02 : This represents DECREMENT message. Subtract 1 from the received 4-byte binary number and send it back to the server using response message.

When a client program sends a packet, this field needs to be always set to 0.

- 16-bit data length field represents **the length of data in bytes**. This field represents the length of data in 'data(optional)' field. Both client and server program must fill this field with the appropriate field. If there is no data in the packet, this field must be set to 0.
- Sequence number is 32-bit field used by IR protocol.
  - For client's hello messages, set this field to 0.
  - When the server sends an instruction message, the server will provide a certain sequence number in this field. When a client program receives an instruction message whose sequence number is **x**, client should send **response message with the same sequence number x** used in the instruction message.

2) Protocol details

2-1) The server will await the connection. The protocol is always initiated by the client.

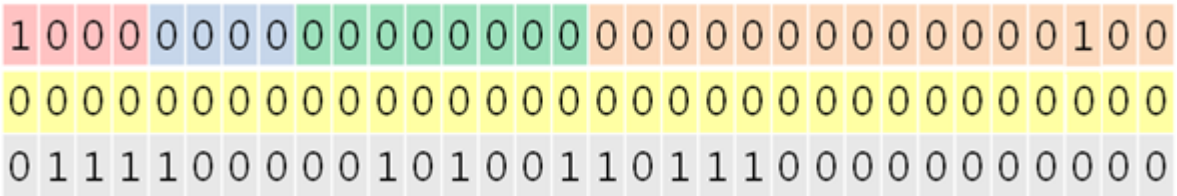
2-2) The client initiates protocol by sending a HELLO message. The server will respond with HELLO message. In other words, the following handshake occurs:

(1) C --> HELLO --> S
(2) C <-- HELLO <-- S

In the client's HELLO message, the client packet must have the following information:

- in the header, H bit is set to 1
- in the header, 'op type' and 'sequence number' fields are set to 0
- in the header, the data length must be set to 4
- in the data field, 4-byte student ID should be set
- HELLO message examples

Assuming a student whose student id is 2016000000 (= 0b0111 1000 0010 1001 1011 1000 0000 0000) sending a hello message to the server program, the IR protocol message should look like :



Where the red bits represent message type field, blue the reserved field, green the op type field, orange the data length field, yellow the sequence number field and gray datagram.

If the server receives the message, the server will respond with the exactly same packet to the client. If successful, the server will print out the messages such as the received student ID. If there is error, the server will output an error message, and the session is terminated.

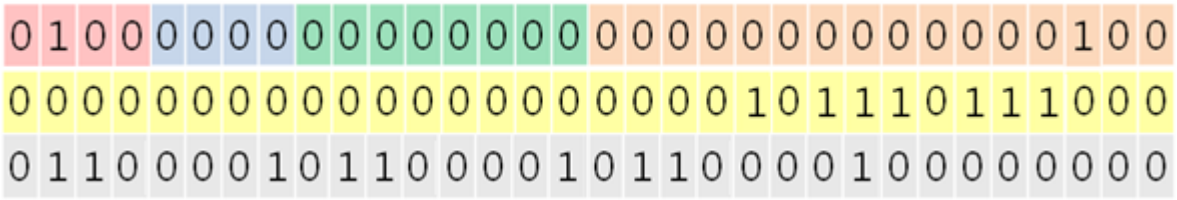
2-3) After a successful exchange of HELLO messages, the server will give instructions to client. The client must respond properly to each instruction. Each instruction-response process obeys the following handshake:

(1) C <-- INSTRUCTION <-- S
(2) C --> RESPONSE --> S

There are three types of instructions:

2-3-1) ECHO request

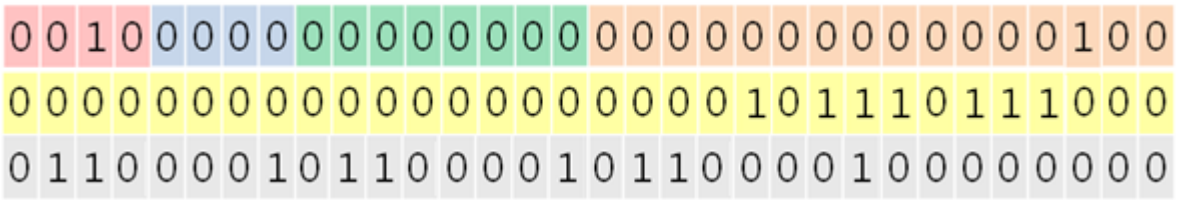
This instruction commands the client to simply echo the null-terminated string data sent by the server. The following packet is an example of ECHO request instruction sent by the server.



Observe that

- I bit is set to 1.
- Op type field is 0x00, which represents ‘ECHO request’.
- sequence number is set to 3000 (= 0b1011 1011 1000)
- data field contains the string sent by the server “aaa”. Note that the last 1 byte of data is null(ASCII code : a is 97 (0b0110 0001)).
- the data length field has value 4.

After receiving the above packet, the client side must send back a response message to the server. The correct response message would look like:



Observe that

- R bit is set to 1.
- sequence number is set to 3000 (= 0b1011 1011 1000) which is the same as the sequence number in the instruction packet.
- Data field contains the string sent by the server which is “aaa” and NULL.
- Data length field have value 4.

The message type is set to response and sequence number is the same as the received instruction message.

2-3-2) INCREMENT

The server will send a 4-byte unsigned integer data to the client. The client must increment that number by 1, and respond to the server with that data. The packet sent by the server should have the following field set:

- I bit is set to 1.
- Op type field is 0x01, which represents ‘INCREMENT’.
- sequence number is set to, for example, 3004 (= 0b1011 1011 1100)

- data field contains the 4-byte number sent by the server: for example, 10 (=0x0000000a).
- the data length field has value 4.

The client should respond as follows, for example:

- R bit is set to 1.
- sequence number is set to 3004 (= 0b1011 1011 1100), same as the one sent by the server.
- data field contains the 4-byte number which increments the number sent by the server by 1: in this example, 11 (=0x0000000b).
- the data length field has value 4.

2-3-3) DECREMENT

The server will send a 4-byte unsigned integer data to the client. The client must decrement that number by 1, and respond to the server with that data. The procedure is similar to INCREMENT.

2-4) After a certain number of instruction-response is completed, the server will send a TERMINATE message to the client. The process looks like

(1) C <-- TERMINATE <-- S

Note that this is one-way process. When the client receives the TERMINATE message, it should terminate the program and exit.
Note that until TERMINATE is received, the client must wait for incoming instruction. Only after TERMINATE is received, the client can end the program.

2) Test your code with the uploaded server executable file as follows :

1. Download and run the server executable file on the Ubuntu terminal.
2. The server program will await connection from 127.0.0.1:47500.
3. Compile and run your code. Your client program should send hello message, whose data field is filled with your student id. (DO NOT use string, use unsigned int type variable which is 4 bytes.)
4. If the server receives hello message successfully, server will send the client program an instruction message. If there is some error in the hello packet, the server will print out error message and terminate.
5. If your client program receives an instruction message, client should read operation type, and data field to perform instructed operation. Write the result of the operation in data field of your response message and send it back to the server. The server will send another instruction message if the result was correct.
6. After a certain number of correct responses are received, the server will send the client a terminate message and terminate. Client program should terminate after receiving a terminate message.

4) Requirements

1. Use c language.
2. Your client program code should not receive any command line arguments.
3. Do not use any function that takes input from terminal. (scanf(), fgets(), etc.)

4. Submit your c source code by uploading it to BlackBoard by due date. The name of your file must be your student id.
   ex) 2016001111.c
5. Use port number 47500.

5) Hints

5.1. You can use the following packet structure:

```
struct hw_packet {
        unsigned char flag; // HIRT-4bits, reserved-4bits
        unsigned char operation; // 8-bits operation
        unsigned short data_len;// 16 bits (2 bytes) data length
        unsigned int seq_num; // 32 bits (4 bytes) sequence number
        char data[1024]; // optional data
};
```

and flag macros as follows :

```
#define FLAG_HELLO              ((unsigned char)(0x01 << 7))
#define FLAG_INSTRUCTION        ((unsigned char)(0x01 << 6))
#define FLAG_RESPONSE           ((unsigned char)(0x01 << 5))
#define FLAG_TERMINATE          ((unsigned char)(0x01 << 4))

#define OP_ECHO                 ((unsigned char)(0x00))
#define OP_INCREMENT            ((unsigned char)(0x01))
#define OP_DECREMENT            ((unsigned char)(0x02))
```

5.2 Example of using the packet structure

```
unsigned int value;
struct hw_packet buf_struct;
buf_struct.flag = FLAG_HELLO;
buf_struct.operation = OP_ECHO;
buf_struct.data_len = 4;
buf_struct.seq_num = 0;
value = 2017010587;
memcpy(buf_struct.data, &value, sizeof(unsigned int));
```

5.3 Reading received data

We can receive the packet data and read the sequence number as follows:

```
recv(client_sockfd, &buf_struct_rcv, sizeof(struct hw_packet), 0);

buf_struct.seq_num = buf_struct_rcv.seq_num;
```

5.4 – convert data type using memcpy function.

Example 1) convert unsigned integer type to 4-bytes character type data(32 bit-stream) :

```
struct hw_packet buf_struct;
unsigned int value = 2017010587;
memcpy(buf_struct.data, &value, sizeof(unsigned int));
```

Example 2) convert 4-bytes character type data to unsigned integer type :

```
unsigned int value;
memcpy(&value, buf_struct_rcv.data, sizeof(unsigned int));
printf("%u\n", value);
```

Client testing results example

```
hw@hw:~/Downloads$ ./client2
*** starting ***

sending first hello msg...
received hello message from the server!
waiting for the first instruction message...
received instruction message! received data_len : 4 bytes
operation type is increment.
increment : 7998
sent response msg with seq.num. 5710 to server.

received instruction message! received data_len : 4 bytes
operation type is increment.
increment : 9173
sent response msg with seq.num. 5711 to server.

received instruction message! received data_len : 4 bytes
operation type is decrement.
decrement : 8897
sent response msg with seq.num. 5712 to server.

received instruction message! received data_len : 4 bytes
operation type is decrement.
decrement : 3128
sent response msg with seq.num. 5713 to server.

received instruction message! received data_len : 4 bytes
operation type is increment.
increment : 2481
sent response msg with seq.num. 5714 to server.

received instruction message! received data_len : 31 bytes
operation type is echo.
echo : LKJXLBSTGEGXOFAPQLNVRRDDTMNUFR
sent response msg with seq.num. 5715 to server.

received instruction message! received data_len : 4 bytes
operation type is decrement.
decrement : 2507
sent response msg with seq.num. 5716 to server.

received terminate msg! terminating...
hw@hw:~/Downloads$
```

Server results example

```
hw@hw:~/Downloads$ ./server2
Server : wating connection request.
Server : 127.0.0.1 client connected.
successfully received a hello msg from 2017010587.
sending hello msg to the client...
generated op type 0x01 instruction msg. data : 7997
sending first instruction message to client...


received a response message!
generated op type 0x01 instruction msg. data : 9172
sending instruction message to client...


received a response message!
generated op type 0x02 instruction msg. data : 8898
sending instruction message to client...


received a response message!
generated op type 0x02 instruction msg. data : 3129
sending instruction message to client...


received a response message!
generated op type 0x01 instruction msg. data : 2480
sending instruction message to client...


received a response message!
generated op type 0x00 instruction msg. data : LKJXLBSTGEGXOFAPQLNVRRDDTMNUFR
sending instruction message to client...


received a response message!
generated op type 0x02 instruction msg. data : 2508
sending instruction message to client...


received a response message!
*** Client program passed the test***
Sending terminate message...
hw@hw:~/Downloads$
```