데이터 통신 보고서

- HDLC 프로토콜 구현

과목	데이터통신(02분반)
학번	201701975
이름	구건모

과제의 요구사항

- 헤더 구성 (출처: http://www.interfacebus.com/HDLC_Protocol_Description.html)
  - Flag: 0x7E(HEX값) 또는 임의 값
  - Address: sender(B 설정), receiver(A 설정) -> 1byte값
  - Control: 8bits
  - Data: Closing Flag 잘라내고 데이터 값 찾기
  - Closing Flag: Flag와 동일

Flag	Address	Control	Data	Flag

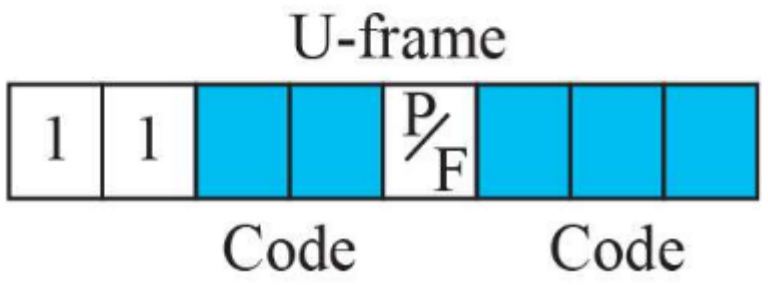
- receiver 프로그램 실행시 socket 열기
- sender 프로그램 실행시 1)connect [2) chat 3) disconnect] 메뉴 출력
- 1)connect 연결 수립할 경우에만 2),3) 기능이 가능함
- 연결설정(SABM), 수신준비(RR), 명령응답(UA),연결해제(DISC) 등 사용하여 프레임 별로 메뉴 선택에 따라 메시지 전송 진행
- 수신측 응답값은 프레임에 대한 응답과 메시지 전송시 소문자를 대문자로 변환 하여 응답 대문자를 소문자로 치환하여 응답함

HDLC 프레임 종류와 임의로 정의한 상수값

구현한 HDLC Frame 구조

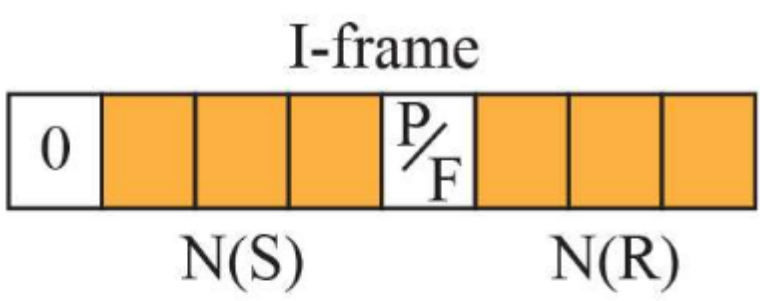
1byte 단위로 값을 저장하기 위해 char 배열을 선언한 후 각 필드의 크기와 순서에 맞게 값들을 초기화 하는 방식으로 HDLC Frame을 구성한 후, 만든 Byte array를 Socket을 통해 전송하는 방식으로 구현하였습니다. flag와 cflag은 값은 과제에 제시된 바와 동일하게 0x7E 값을 가지는 변수를 선언하여 초기화하거나 flag 값 확인에 사용하였고, control 필드는 U, S-Frame 은 고정된 상수값을 사용하였고 I-Frame의 경우 Seq, Ack 정보를 control 필드에 같이 담아서 보내기 위해서 별도의 구조체와 함수들을 구현하여 비트단위로 설정해주어 사용하였습니다. I-Frame은 Seq, Ack에 따라서 control 값이 계속 변하기 때문에 수신한 Frame이 I-Frame 인지 구분할수 있도록 하기 위해서 실제 HDLC 프레임의 control field 정의대로 첫번째 비트를 반드시 0으로 설정하도록 구현하였습니다.

Frame 별 Control Field의 Format




```
#define SABM 0x01
#define UA 0x02
#define DISC 0x03
```

U-Frame의 경우 control 비트가 변경될 일이 없기 때문에 위와 같이 임의의 고정된 값으로 선언하여 사용하였습니다.

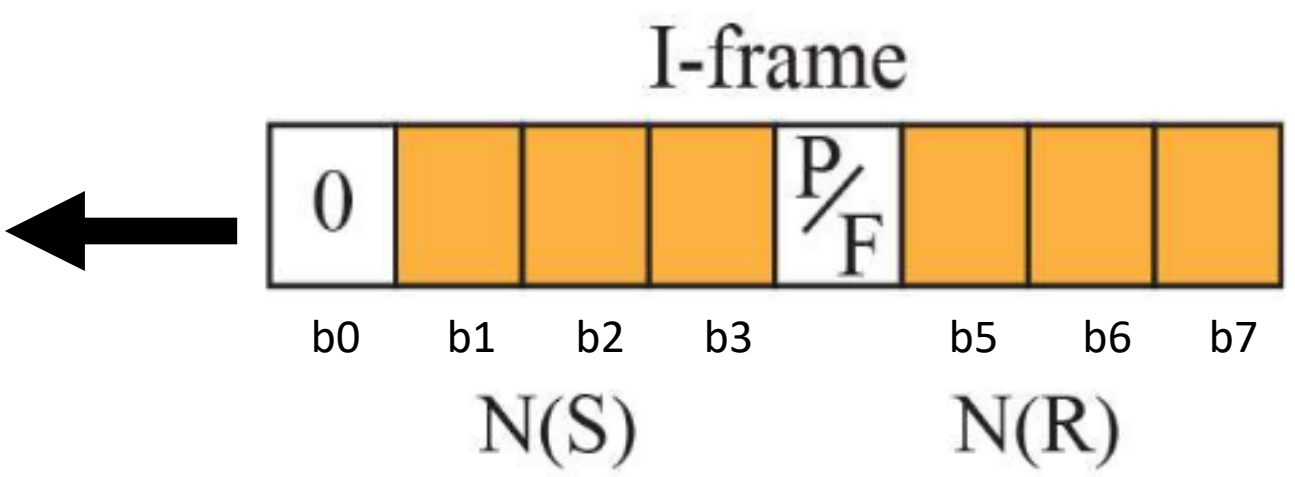


I-Frame의 경우 채팅과정에서 I-Frame의 control 내에 N(S) 와 N(R) 비트값에 Sequence와 Acknowledge 값을 넣어주기 위해 정적인 값으로 선언하지 않고 control 필드를 비트 단위로 다룰 수 있도록 구조체와 별도의 함수들을 구현하여 사용하였습니다.

control 를 bit 단위로 설정해주기 위한 구조체와 함수들

I-frame을 구성시 control 를 bit 단위로 설정해주기 위한 구조체와 함수들

```
struct control{
    unsigned b0 : 1;
    unsigned b1 : 1;
    unsigned b2 : 1;
    unsigned b3 : 1;
    unsigned b4 : 1;
    unsigned b5 : 1;
    unsigned b6 : 1;
    unsigned b7 : 1;
};
```



I-Frame의 Control 필드 내부 구조

각각 setter와 getter를 구현하여 control을 비트단위로 초기화할 때 사용하였습니다.

set_iframe_sequence_number와 set_iframe_acknowledge_number 함수는 각각 control 필드의 Sequence와 Acknowledge 가 저장되는 특정위치 비트값을 설정하는 함수로 seq 또는 ack 값을 integer 타입으로 받은 후 bit 연산자를 통해서 인자값에 대한 binary 값을 control 필드내의 seq와 ack가 저장되는 위치의 비트 값들로 할당해주도록 구현하였습니다.

```
void set_iframe_sequence_number(struct control* c, unsigned seq_num) {
    if (seq_num < 8) {
        // 각 자리수에 해당하는 값으로 &를 취해서 해당자리 값만 남긴 후, 한자리 비트값으로 만들기 위해 자리수 만큼 right shift
        c->b1 = (seq_num & 4) >> 2;
        c->b2 = (seq_num & 2) >> 1;
        c->b3 = seq_num & 1;
    }
}

void set_iframe_acknowledge_number(struct control* c, unsigned ack_num){
    if (ack_num < 8) {
        // 각 자리수에 해당하는 값으로 &를 취해서 해당자리 값만 남긴 후, 한자리 비트값으로 만들기 위해 자리수 만큼 right shift
        c->b5 = (ack_num & 4) >> 2;
        c->b6 = (ack_num & 2) >> 1;
        c->b7 = ack_num & 1;
    }
}
```

getter의 경우 control 필드내의 seq, ack 위치의 비트를 읽어 10진수 변환된 값을 리턴하도록 구현하였습니다. 각 비트값의 자리수만큼 left shift를 수행한 후, 수행된 결과를 전부 or 하여 10진수로 변환하도록 구현하였습니다.

```
int get_iframe_sequence_number(struct control* c){
    unsigned int res = (c->b1 << 2) | (c->b2 << 1) | c->b3;
    return res;
}

int get_iframe_acknowledge_number(struct control* c){
    unsigned int res = (c->b5 << 2) | (c->b6 << 1) | c->b7;
    return res;
}
```

Connection/Disconnection 관련 주요 구현

Connection을 위한 send_sabm 함수

```
void send_sabm(){
    // Receiver로 SABM 전송하는 함수
    char hdlc_frame[MAX_SIZE];
    int total_length = 0;

    /* Frame의 각 필드값 할당 */
    hdlc_frame[0] = DEFAULT_FLAG;   // flag 설정
    hdlc_frame[1] = RECEIVER_ADDR; // Receiver의 주소('B')로 목적지 주소를 설정
    hdlc_frame[2] = SABM; // control을 SABM로 설정
    // SABM(uframe)은 별도의 데이터를 필요로 하지 않으므로 생략
    hdlc_frame[3] = DEFAULT_FLAG; // cflag 설정

    // 별도의 데이터를 포함하지 않기 때문에 전송하는 프레임 크기는 헤더 크기인 MIN_HDLC_SIZE(=4)로 할당
    total_length = MIN_HDLC_SIZE;

    // 생성한 SABM 프레임을 send socket을 이용하여 Receiver로 전송
    sendto(sndsock, &hdlc_frame, total_length, 0, (struct sockaddr*)&s_addr, sizeof(s_addr));

    // 연결 요청했다는 메시지와 방금 보낸 프레임 내용 출력
    print_current_time();
    printf("연결을 요청합니다.(SABM)\n");
    printf("  └———> 보낸 프레임(%d bytes): \n", total_length);
    print_frame((unsigned char *)hdlc_frame, total_length);

}
```

Connection 과정 Sender는 SABM을 Receiver에게 전송하여 연결을 요청합니다.

이 과정에서 에서 Sender가 전송할 SABM 프레임을 만들고 Receiver로 전송하는 함수로 send_sabm이라는 함수를 구현하였습니다. 우선 전송에 앞서 전송할 HDLC 프레임을 초기화 합니다. HDLC 프레임의 첫번째 바이트 부분에 위치한 Flag는 사전에 정의한 DEFAULT_FLAG 상수 값인 0x7E로 설정해주었습니다.

Sender는 Receiver의 주소('B')를 목적지 주소로 설정하고, Control 필드는 사전에 define한 SABM 값으로 설정하여 Receiver 측에서 수신시 SABM Frame임을 구별할 수 있도록 하였습니다. SABM(U-Frame)의 경우 별도로 전달할 데이터가 없기 때문에 Data 필드는 초기화 하지 않았고, 마지막에 위치한 cflag 또한 flag와 동일하게 사전에 정의한 DEFAULT_FLAG 상수 값인 0x7E로 설정해주어 SABM에 대한 HDLC 프레임을 구성을 완료한 후, 전송 내용을 화면에 바이트 단위로 출력해주도록 하였습니다. socket을 앞서 구성된 SABM Frame이 Receiver에게 전송됩니다.

Connection/Disconnection 관련 주요 구현

Disconnection을 위한 send_disc 함수

```
void send_disc(){
    char hdlc_frame[MAX_SIZE];
    int total_length = 0;

    hdlc_frame[0] = DEFAULT_FLAG;
    hdlc_frame[1] = 'B';
    hdlc_frame[2] = DISC;
    hdlc_frame[3] = DEFAULT_FLAG;

    total_length = 4;

    //send hdlc packet
    sendto(sndsock, &hdlc_frame, total_length, 0, (struct sockaddr*)&s_addr, sizeof(s_addr));

    print_current_time();
    printf("연결 해제를 요청합니다.(DISC)\n");
    printf("  └─────> 보낸 프레임(bytes): \n");
    print_frame((unsigned char*)hdlc_frame, total_length);

}
```

Disconnection 과정에서 Sender는 DISC 프레임을 Receiver로 전송하여 연결해제를 시작합니다.
해당 send_disc 함수는 Sender에서 Disconnection 수행시 DISC 프레임을 만들고 전송하는 역할을 수행하는 함수입니다. Sender는 Receiver의 주소('B')를 목적지로 설정하고, Control 필드는 사전에 define한 DISC 값으로 설정하여 Receiver 측에서 수신시 DISC Frame임을 구별할 수 있도록 하였습니다. SABM(U-Frame)의 경우 별도로 전달할 데이터가 없기 때문에 Data 필드는 초기화 하지 않았고, HDLC 프레임의 첫번째 바이트 부분에 위치한 Flag와, 마지막에 위치한 Cflag는 사전에 정의한 DEFAULT_FLAG 상수 값인 0x7E로 설정해주어 DISC에 대한 HDLC 프레임을 구성한 후, 전송 내용을 화면에 바이트 단위로 출력해주도록 하였습니다. socket을 앞서 구성된 Frame이 Receiver에게 전송됩니다.

Connection/Disconnection 관련 주요 구현

Receiver의 send_ua 함수

```
void send_ua(){

    char hdlc_frame[MAX_SIZE];
    int total_length = MIN_HDLC_SIZE; // MIN_HDLC_SIZE=4

    hdlc_frame[0] = DEFAULT_FLAG; // flag 설정, DEFAULT_FLAG(=0x7E)
    hdlc_frame[1] = SENDER_ADDER; // sender의 주소인 SENDER_ADDER('A')로 설정
    hdlc_frame[2] = UA; // 사전 정의한 UA에 대한 비트값 UA(=0x02)로 설정
    hdlc_frame[3] = DEFAULT_FLAG; // cflag 설정, DEFAULT_FLAG(=0x7E)

    // 전송 내용 출력
    print_current_time();
    printf("Sender에게 UA를 전송합니다.\n");
    printf("  └─────> 전송한 프레임(%d bytes): ", total_length);
    for(int i=0; i<total_length; i++){
        printf("[%d]%#0.2x ", i, hdlc_frame[i]);
    } printf("\n");
    print_frame((unsigned char *)hdlc_frame,total_length);

    // socket으로 생성한 UA Frame 전송
    sendto(sndsock, &hdlc_frame, total_length, 0, (struct sockaddr*)&s_addr, sizeof(s_addr));

}
```

Sender로부터 Connect 요청(SABM)이나 Disconnect 요청 (DISC)를 수신한 경우 그에 대한 응답으로 UA를 전송하게 됩니다. Receiver는 Sender의 주소('A')를 목적지로 설정하고, Control 필드는 사전에 define한 UA 값으로 설정하여 UA Frame임을 알 수 있도록 합니다. U-frame의 경우 별도로 전달할 데이터가 없기 때문에 Data 필드는 따로 할당하지 않았습니다. HDLC 프레임의 첫번째 바이트 부분에 위치한 Flag와, 마지막에 위치한 Cflag는 사전에 정의한 DEFAULT_FLAG 상수 값인 0x7E로 설정해주어 UA에 대한 HDLC 프레임을 구성한 후, 전송 내용을 화면에 바이트 단위로 출력해주도록 하였습니다. socket을 통해 구성된 HDLC 프레임이 Sender에게 전송됩니다.

Connection/Disconnection 관련 주요 구현

Connection / Disconnection 과정에서 flag, cflag, address 확인

```
print_current_time();
printf("Receiver로부터 UA를 수신하였습니다. (크기: %d bytes)\n", length);
printf("  └──> 수신한 프레임(bytes): ");
for(int i=0; i<length; i++){
    printf("[%d]%#0.2x ", i, received[i]);
}
printf("\n");
print_frame((unsigned char*)received, length);

/*──────────────── flag, cflag 값 확인 ────────────────*/
printf("\t[*] flag, cflag 값을 확인합니다...\n");
if(received[0] == DEFAULT_FLAG && received[length-1] == DEFAULT_FLAG)
{
    printf("\t  └──> flag:%#02x\n\t  └──> cflag:%#02x\n\t\t\t(확인완료)\n\n", received[0], received[length-1]);}
else{
    printf("\n\t\t[!] flag 또는 cflag 값이 유효하지 않습니다.\n");
}

/*──────────────── Address 값 확인 ────────────────*/
printf("\t[*] Address 값을 확인합니다...\n");
if(get_hdlc_addr(received) == SENDER_ADDR)
{printf("\t  └──> address:%c\n\t\t\t(확인완료)\n\n", get_hdlc_addr(received));}
else{
    printf("\n\t[!] address 값이 유효하지 않습니다.\n");
}

sleep(1);
print_current_time();
printf("연결 해제 완료\n\n");
printf("──────────────────────────────────────────────────\n");
```

Connection 또는 Disconnection 과정에서 상대방이 전달한 u-frame의 필드값중 flag, clfag가 유효한 flag 값인지 확인하고, Destination Address를 확인하여 본인에게 온 프레임이 맞는지 확인하는 코드 중 일부입니다.
수신한 Frame의 가장 처음과 끝부분에 flag, cflag 값이 위치하므로 해당 위치를 기존에 정의한 DEFAULT_FLAG(=0x7E) 상수값과 비교하여 수신한 Frame의 flag, cflag가 유효한지 확인합니다.
다음으로는 수신한 프레임의 목적지 주소가 수신자 본인의 주소와 동일한지 확인하는 부분입니다.
해당 코드는 sender의 main 코드 중 수신한 프레임에서 필드값의 유효성을 확인하는 부분중 일부이기 때문에 전달 받은 Frame에서 Address 필드 값이 sender 자신의 주소인 SENDER_ADDR(='A')와 같은지 비교하도록 구현하였습니다. Receiver 측에서도 위와 동일하게 구현하여 확인이 이루어지며, Receiver의 경우에는 해당 코드와 동일하게 수행하나 주소 비교 과정만 자신의 주소인 RECEIVER_ADDR(='B') 와 같은지 확인하도록 구현하였습니다.

Chatting 기능을 위한 make chat frame 함수

```
unsigned char* make_chat_frame(char *chat_data, int chat_length) {
    unsigned char* hdlc_frame = (unsigned char*)malloc((chat_length + MIN_HDLC_SIZE + 1) * sizeof(unsigned char));
    if (hdlc_frame == NULL) {
        return NULL;  // 메모리 할당 실패 시 NULL 반환
    }

    struct control ctr;
    set_hdlc_iframe(&ctr);
    set_iframe_sequence_number(&ctr, seq);
    set_iframe_acknowledge_number(&ctr, 0);

    hdlc_frame[0] = DEFAULT_FLAG;
    hdlc_frame[1] = 'B';
    hdlc_frame[2] = *(unsigned char *)&ctr;

    // /* DEBUG */
    // printf("[DEBUG] %s\n" , __func__);
    // for(int i=0; i< chat_length; i++){
    //     printf("chat_data[%d]: %#02x\n" , i ,chat_data[i]);
    // }

    for(int i=0; i< chat_length; i++){
        hdlc_frame[3+i] = chat_data[i];
    }

    hdlc_frame[3 + chat_length] = DEFAULT_FLAG;

    return hdlc_frame;
}
```

make_chat_frame 함수는 사용자가 입력한 채팅 내용과 Seq, Ack 정보를 반영하여 i-frame을 생성하는 함수입니다. 채팅 데이터와 sequence, acknowledge를 HDLC 프레임으로 만든 후 반환해주기 위해 값들을 담아 리턴할 hdlc_frame 배열을 malloc을 통해 동적 할당 해주었습니다. I-Frame의 경우 seq와 ack가 control 필드의 8bit 중 각각 3bit, 3bit씩 할당되어 들어가게 됩니다. set_iframe_sequence_number와 set_iframe_acknowledge_number 함수는 각각 control 필드의 Sequence와 Acknowledge 가 저장되는 특정위치 비트값을 설정하는 함수로 seq 또는 ack 값을 integer 타입으로 받은 후 bit 연산자를 통해서 control 필드내의 seq와 ack가 저장되는 위치의 비트 값들을 설정해줍니다. 따라서 Receiver 측이 해당 프레임을 수신했을 때 Control 필드 내에 binary로 저장된 seq와 ack 값을 확인하여 Sender가 보낸 프레임의 seq, ack 값을 확인할 수 있습니다. 사용자가 입력한 채팅에 대한 데이터는 hdlc 프레임의 data 필드에 들어가게 됩니다. 인자로 받은 chat_data를 hdlc 프레임의 data 필드 시작위치인 index 3번부터 순차적으로 할당해주었습니다. 모두 할당된 후에는 hdlc_frame의 마지막 값인 clfag 값을 설정해 준 후 생성한 hdlc_frame을 리턴해주었습니다. main에서 채팅 기능이 시작되어 채팅을 전송하는 과정에서 사용자가 입력한 채팅 내용은 위 함수를 통해 i-frame으로 변환되어 처리되게 됩니다. Sender 측에서 채팅을 보낼때 채팅 데이터를 바로 보내는 것이아니라 채팅 데이터와 seq, ack를 담은 i frame을 만들고 GBN 적용을 위해 sending 버퍼에 담고 sliding window의 상황을 고려하여 전송하는등, 프레임 생성 이외에도 채팅전송에 추가적인 과정이 필요하기 때문에 위 함수에서는 채팅 데이터에 대한 i-frame만 동적으로 생성하여 리턴하도록 구현하였습니다.

Chatting에 대한 response를 생성하는 make_response_str 함수

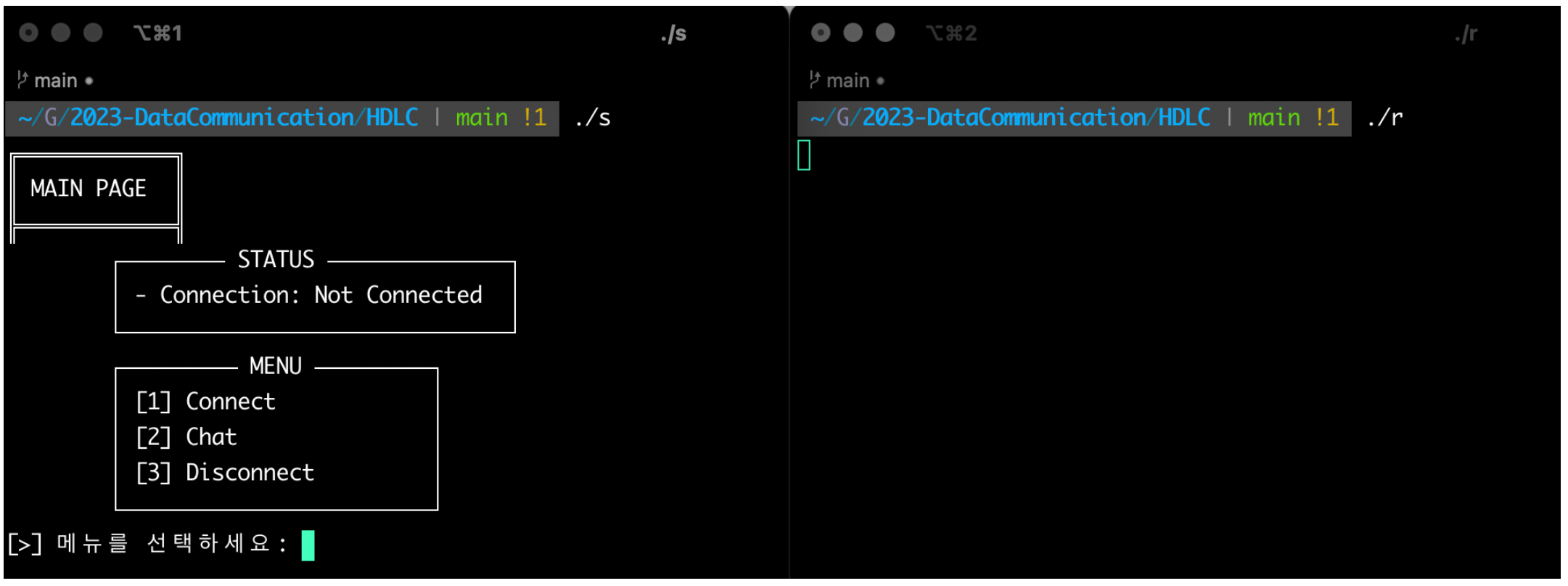
```
char* make_response_str(const char* received_data) {
    int i = 0;
    char* res = malloc(strlen(received_data) + 1);  // 결과 문자열을 저장할 메모리를 동적으로 할당

    while (received_data[i]) {
        if (islower(received_data[i])) {
            res[i] = toupper(received_data[i]);  // 소문자인 경우 대문자로 변환
        } else if (isupper(received_data[i])) {
            res[i] = tolower(received_data[i]);  // 대문자인 경우 소문자로 변환
        } else {
            res[i] = received_data[i];  // 알파벳이 아닌 경우 그대로 저장
        }
        i++;
    }

    res[i] = '\0';
    return res;
}
```

Receiver 측은 sender가 보낸 메시지에 대해서 ack 값과 sender과 보낸 채팅 데이터에서 대문자는 소문자로, 소문자는 대문자로 변환해야한다는 과제의 요구사항이 존재합니다. 이때 ack 값은 받은 프레임에서 seq 번호를 가져온 후 1 증가시킨 값을 그대로 이용하면 되지만 채팅 메시지에 대한 response는 대문자를 소문자로, 소문자를 대문자로 변환하는 과정이 필요합니다. 해당 과정을 별도의 make_response_str 라는 이름의 함수로 구현하였습니다. make_response_str 함수는 특정 문자열에 대해 대소문자가 반전된 결과를 리턴하도록 구현하였습니다. Receiver는 Sender로부터 채팅을 수신한 경우, Sender가 보낸 채팅 내용를 함수의 인자로 넘겨서 그에 대한 Response 데이터를 만들고, 이 데이터와 sender로 부터 수신한 seq에 대한 ack값을 합쳐 i-frame을 구성해서 Sender에게 응답하게 됩니다.

결과화면 – 초기 시작화면

Sender

Receiver



Sender와 Receiver 각각의 시작 화면입니다. Sender의 경우 메뉴를 출력하고, Receiver는 최초 실행시 별도의 출력없이 실행되며 Socket을 통해 Sender로부터 수신을 대기합니다. 추후 Sender 측에서 특정 메뉴를 선택한 후 그에 따른 Frame을 Receiver로 전송하게되면 Receiver는 Socket을 열어 대기하고 있다가 Sender로부터 받은 Frame에 따라 적절히 응답하도록 구현하였습니다.

각 과정에서 전달되는 프레임의 내용들은 모두 바이트 단위로 깔끔하게 출력되도록하여 실제 프레임을 구성하는 각 필드의 값들이 상황에 따라 올바른 값들이 들어가고 있는지 확인할 수 있도록 하였습니다.

결과화면 – Connection

Sender

main •

Connect

(07:08:45) 연결을 요청합니다.(SABM)
└——> 보낸 프레임(4 bytes):

Index	00	01	02	03
Value	0x7e	0x42	0x01	0x7e

UA 수신 대기 중...
(07:08:46) Receiver로부터 UA를 수신하였습니다. (크기: 4 bytes)
└——> 수신한 프레임(bytes): [0]0x7e [1]0x41 [2]0x02 [3]0x7e

Index	00	01	02	03
Value	0x7e	0x41	0x02	0x7e

[*] flag, cflag 값을 확인합니다...
└——> flag:0x7e
└——> cflag:0x7e
(확인완료)

[*] Address 값을 확인합니다...
└——> address:A
(확인완료)

(07:08:47) 연결 완료

MAIN PAGE

STATUS
- Connection: Connected

MENU
[1] Connect
[2] Chat
[3] Disconnect

[>] 메뉴를 선택하세요 :

Receiver

main •

~/G/2023-DataCommunication/HDLC | main !1 ./r
(07:08:45) Sender로부터 SABM을 수신하였습니다.
└——> 수신한 프레임(4 bytes): [0]0x7e [1]0x42 [2]0x01 [3]0x7e

Index	00	01	02	03
Value	0x7e	0x42	0x01	0x7e

[*] flag, cflag 값을 확인합니다...
└——> flag:0x7e
└——> cflag:0x7e
(확인완료)

[*] Address 값을 확인합니다...
└——> address:B
(확인완료)

(07:08:45) Sender에게 UA를 전송합니다.
└——> 전송한 프레임(4 bytes): [0]0x7e [1]0x41 [2]0x02 [3]0x7e

Index	00	01	02	03
Value	0x7e	0x41	0x02	0x7e

(07:08:45) 연결 완료

Sender에서 메뉴 1번(Disconnect)을 선택하면 Connection 과정이 시작됩니다.
Sender는 SABM 프레임을 Receiver로 전송하고 Receiver로부터 UA 프레임을 수신하기까지 대기합니디.
Receiver 측에서는 Sender로부터 수신한 SABM를 확인하고 해당 프레임 내의 필드값을 확인하여 flag, address, cflag의 유효성을 우선 확인합니다. 유효한 프레임일 경우 그에 대한 응답으로 UA 프레임을 다시 Sender에게 전송합니다. Sender는 Receiver의 UA를 수신하면
해당 프레임의 필드값(flag, cflag, address)의 유효성을 확인한 이후 유효한 경우 연결 완료를 출력하며 연결이 완료됩니다.연결 과정이 완료된 이후 메뉴 출력 부분에 STATUS를 확인해보면 connection이 Connected 로 바뀐 것을 확인해볼 수 있습니다.

connection 과정에서 주고받는 프레임의 정보들도 byte 단위로 확인해보면, flag, clfag는 모두 0x7E로 잘 설정되어 주고받는 것을 확인해 볼 수 있고, Address의 경우도 Sender가 Receiver로 보낼때는 Receiver의 주소인 ‘B’에 대한 아스키값 0x42가 들어가고 Receiver가 Sender로 보낼때는 Sender의 주소인 ‘A’에 대한 아스키값인 0x41이 들어가고 있는 것을 볼 수 있습니다.
control 필드의 경우 SABM 프레임은 사전에 SABM을 위해 정의한 0x01이, UA는 사전에 UA임을 나타내기 위해 정의한 값인 0x02 값이 할당되어 전송되는 것을 확인해 볼 수 있습니다.

결과화면 – Chatting

Sender

Chatting

```
////////////////////////////////////////////////////////////
// [>] 채팅이 시작되었습니다.                              //
//      * 메시지를 입력하신 후 Enter를 누르시면 전송됩니다.  //
//      ("exit" 또는 "quit"을 입력하시면 채팅이 종료됩니다.) //
////////////////////////////////////////////////////////////
hello
       (07:13:05) Send[SEQ:0] hello
```

Index	00	01	02	03	04	05	06	07	08	09
Value	0x7e	0x42	0x00	0x68	0x65	0x6c	0x6c	0x6f	0x0a	0x7e

(07:13:06) Received[ACK:1]: HELLO

Index	00	01	02	03	04	05	06	07	08	09
Value	0x7e	0x41	0x80	0x48	0x45	0x4c	0x4c	0x4f	0x0a	0x7e

aBaBaB

(07:14:28) Send[SEQ:1] aBaBaB

Index	00	01	02	03	04	05	06	07	08	09	10
Value	0x7e	0x42	0x08	0x61	0x42	0x61	0x42	0x61	0x42	0x0a	0x7e

(07:14:29) Received[ACK:2]: AbAbAb

Index	00	01	02	03	04	05	06	07	08	09	10
Value	0x7e	0x41	0x40	0x41	0x62	0x41	0x62	0x41	0x62	0x0a	0x7e

연결이 완료된 이후에는
채팅기능을 이용할 수 있습니다.
Sender에서 2번 메뉴를 선택하면
채팅이 시작됩니다.
Sender가 임의의 문자열을
입력하면 해당 값과 sequence를
담은 i-frame을 만들어
Receiver로 전송합니다.

Sender에서 보낸 첫번째
메시지(seq=0) “hello” 에 대한
답변으로 Receiver로부터
ack=1과 “HELLO”를 받은것을
볼 수 있습니다.

Sender에서 보낸 두번째
메시지(seq=1) ”aBaBaB”의
경우에도 Receiver로부터
ack=2와 “AbAbAb”를 받은것을
확인할 수 있습니다.

Receiver

(07:13:05) Received [SEQ:0] hello

Index	00	01	02	03	04	05	06	07	08	09
Value	0x7e	0x42	0x00	0x68	0x65	0x6c	0x6c	0x6f	0x0a	0x7e

[*] Sender에게 다음과 같이 응답을 전송합니다:
  └——> 전송 내용 > [ACK:1] HELLO
(07:13:05) Send [ACK:1] HELLO

Index	00	01	02	03	04	05	06	07	08	09
Value	0x7e	0x41	0x80	0x48	0x45	0x4c	0x4c	0x4f	0x0a	0x7e

(07:14:28) Received [SEQ:1] aBaBaB

Index	00	01	02	03	04	05	06	07	08	09	10
Value	0x7e	0x42	0x08	0x61	0x42	0x61	0x42	0x61	0x42	0x0a	0x7e

[*] Sender에게 다음과 같이 응답을 전송합니다:
  └——> 전송 내용 > [ACK:2] AbAbAb
(07:14:28) Send [ACK:2] AbAbAb

Index	00	01	02	03	04	05	06	07	08	09	10
Value	0x7e	0x41	0x40	0x41	0x62	0x41	0x62	0x41	0x62	0x0a	0x7e

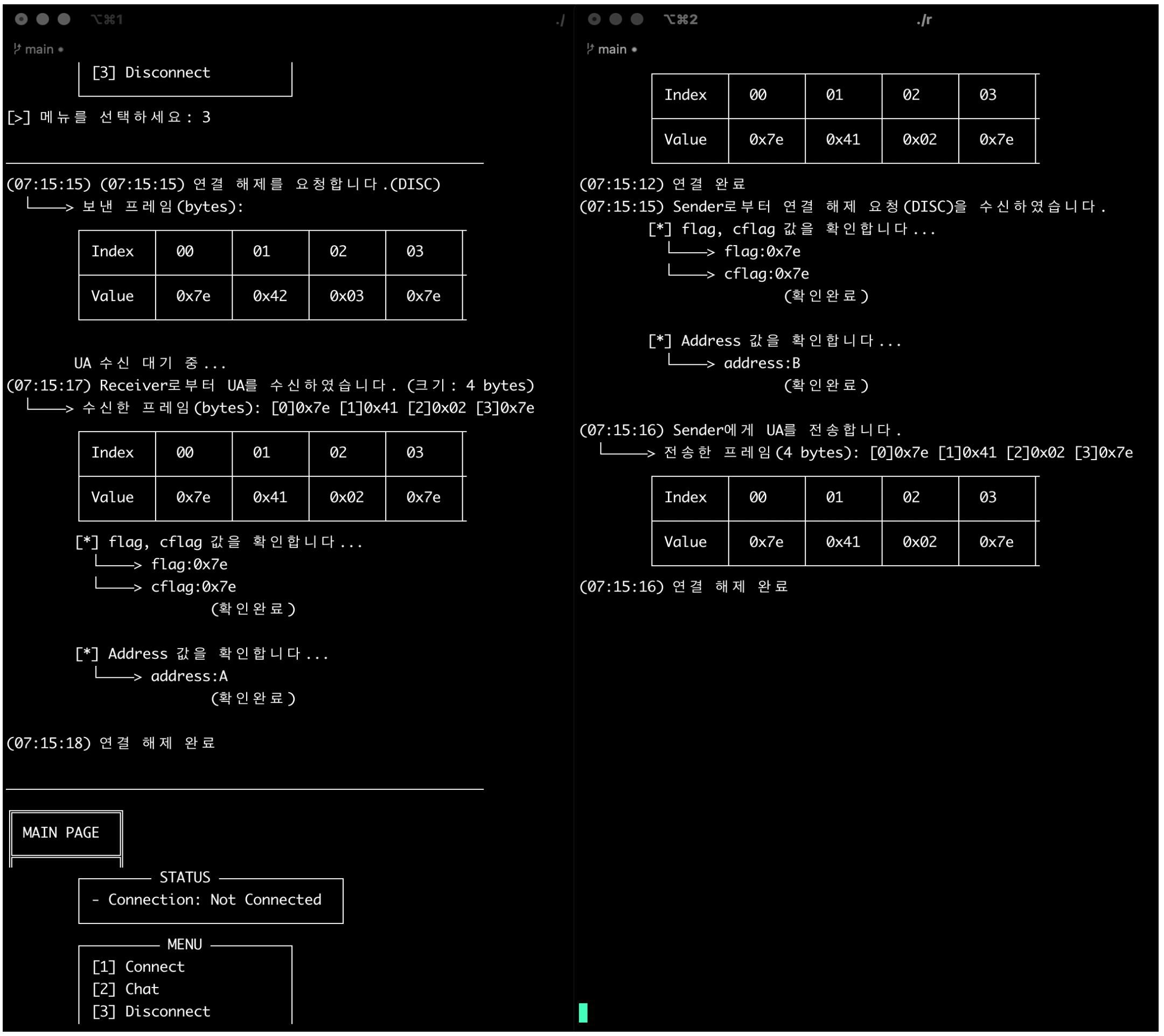
Receiver는 Sender가 보낸
seq 번호에 1을 더한 값을
ack로 보내어 해당 seq에 대한
프레임을 정상적으로
수신했음을 알리고 받은 채팅
내용에서 대문자는 소문자로,
소문자는 대문자로 변환한
값을 응답합니다.

앞서 설명한 바와 같이
Sender측에서 보낸
두개의 프레임
”hello”와 ”aBaBaB”에 대해
그에 해당하는 ack 값과
대소문자가 반전된 값으로
i-frame을 구성하여
정상적으로 응답을 전송하는
것을 확인해 볼 수 있습니다.

결과화면 Disconnect

Sender

Receiver



Sender에서 메뉴 3번(Disconnect)을 선택하면 Disconnection 과정이 시작됩니다.

우선 Sender는 DISC 프레임을 Receiver로 전송하고 UA를 수신하기를 대기합니디.

Receiver 측에서는 Sender로부터 수신한 DISC를 확인하고 UA를 다시 Sender에게 전송합니다.

Sender는 Receiver의 UA를 대기하고 있다가 UA를 수신하면 연결 해제 완료를 출력하며 연결이 해제됩니다.

연결 해제가 완료된 이후 메뉴 출력 부분에 STATUS를 확인해보면 connection이 Not Connected로 바뀐 것을 확인해볼 수 있습니다.

Disconnection 과정에서 주고받는 프레임의 정보들도 byte 단위로 확인해보면, flag, clfag는 모두 0x7E로 잘 설정되어 주고받는 것을 확인해 볼 수 있고, Address의 경우도 Sender가 Receiver로 보낼때는 Receiver의 주소인 'B'에 대한 아스키값 0x42가 들어가고 Receiver가 Sender로 보낼때는 Sender의 주소인 'A'에 대한 아스키값인 0x41이 들어가고 있는 것을 볼 수 있습니다. control 필드의 경우 DISC 프레임은 사전에 DISC을 위해 정의한 0x03이, UA 프레임은 사전에 UA임을 나타내기 위해 정의한 값인 0x02 값이 할당되어 전송되는 것을 확인해 볼 수 있습니다.

기타 결과화면

chatting 에서 “exit ” 또는 ”quit” 입력시 종료

Chatting

//
// [>] 채팅이 시작되었습니다.   //
//   * 메시지를 입력하신 후 Enter를 누르시면 전송됩니다. //
//   ("exit" 또는 "quit"을 입력하시면 채팅이 종료됩니다.) //
//
abcd

(20:05:24) Send[SEQ:4] abcd

Index	00	01	02	03	04	05	06	07	08
Value	0x7e	0x42	0x02	0x61	0x62	0x63	0x64	0x0a	0x7e

(20:05:25) Received[ACK:5]: ABCD

Index	00	01	02	03	04	05	06	07	08
Value	0x7e	0x41	0xa0	0x41	0x42	0x43	0x44	0x0a	0x7e

exit

[>] 채팅을 종료합니다.

---

connection이 되지 않은 상태에서 chatting과 disconnection 메뉴 선택시 오류메시지 출력

MAIN PAGE

STATUS

- Connection: Not Connected

MENU

[1] Connect
[2] Chat
[3] Disconnect

[>] 메뉴를 선택하세요 : 2

[!] 연결이 필요합니다.

MAIN PAGE

STATUS

- Connection: Not Connected

MENU

[1] Connect
[2] Chat
[3] Disconnect

[>] 메뉴를 선택하세요 : 3

[!] 연결되어 있지 않습니다.
