[데이터통신]_텀프로젝트(HDLC Protocol)_202102699_정민경

1. HDLC Protocol 설명

HDLC Protocol 은 3개의 프레임이 사용된다.

1) I-Frame : flag, address, seq 번호와 ack 번호, 보낼 데이터, cflag 로 구성되어있어 데이터를 보낼 때 즉, 우리 실습에서 봤을 때 chat 모드에서 사용되는 frame.
2) S-Frame : error 나 flow control을 하기 위한 frame. (내가 받지 못하는 정보에 대한 요청)
3) U-Frame : flag, address, 미리 정해진 code, cflag 로 구성되어있고, HDLC 연결 시 사용되는 frame. 실습구현에서 봤을 때 connection, disconnection 기능에서 사용되는 frame.

실습과정을 설명하면 다음과 같다.

1. 처음 receiver 실행 후 sender 실행하면 socket으로 연결된다.
2. 실행하면 sender 화면에 connection 이라는 1번 기능만 있는 메뉴화면이 뜬다. 이때 1번기능을 실행하면 U-Frame으로 receiver와 통신하여 연결한다.
    - sender -> receiver 과정에서 sender는 SABM을 보내기 때문에 그에 맞는 코드를 frame 에 담아 전달한다.
    - receiver는 받은 Frame에서 flag, address를 확인한다. 이 때 출력은 printf문으로 terminal 화면에 출력하는 것으로 대체한다.
    - SABM을 잘 수신했다면 이번에는 receiver -> sender 로 UA 메시지를 전송한다. 이 때 frame에 담기는 control 값은 UA의 코드를 담아 전송하면 된다.
    - sender 측에서 UA를 잘 받았다면 sender 측에서도 받은 메시지에서 flag와 address를 확인하고 “연결성공” 메시지를 출력한다.
3. 1번 기능인 connect 기능을 수행해 연결에 성공했다면 2번 chat 기능과 3번 disconnect 기능이 있는 메뉴 화면을 출력한다.

4-1. 만약 2번 (chat) 을 선택했다면

I-Frame으로 sender와 receiver 가 통신한다.

- sender -> receiver 로 I-Frame을 보낼 때, seq_number 와 ack_number, receiver에게 보낼 data를 함께 보낸다.
- receiver는 본인이 원하는 데이터 즉, 본인의 ack번호와 일치하는 seq번호의 데이터가 수신되었는지 확인 후 본인의 seq번호와 ack번호, 전달받은 메시지를 출력한다. 이때 메시지는 대소문자를 바꿔 출력한다.
- 그 후 receiver -> sender 에게로 잘 받았다고 I-Frame으로 반응을 보낸다. 이때 담기는 정보는 본인의 seq, ack 번호와 출력한 메시지를 함께 보낸다.
- 만약 sender가 보낼 메시지가 “quit” 이거나 “exit” 이라면 통신 후에 3번과정으로 돌아간다.

4-2. 만약 3번 disconnect을 선택했다면

- sender -> receiver 로 DISC 전송한다.
- receiver에서 DISC 제대로 수신 후 receiver -> sender로 UA 응답 전송한다.
- sender 측에서 UA 응답을 받은 후 2번 과정으로 돌아간다.

2. 기능 별 코드구현 (공통)

- 정의된 Macro

```
#define PORT 8080
#define WINDOW_SIZE 4
#define SENDER_ADDRESS 'b'
#define RECEIVER_ADDRESS 'a'
#define I_FRAME 0b00000000
#define S_FRAME 0b10000000
// U-Frame 값 define
#define SABM 0b11110100
#define UA 0b11000110
#define DISC 0b11000010
#define FLAG 0x7E
```

각각의 frame의 control 비트를 미리 define 해두었다.
I, S-Frame은 초깃값을 정의해두었고, U-Frame 값은 각각의 code로 정의해두었다.

- Frame header 구현

```
struct Frame {
    int Flag;
    char Address; // sender : b | receiver : a
    int Control; // I-frame : 0 = 0 | S-frame : 10 = 2 | U-frame : 11 = 3
    char Data[1024]; //Closing Falg 잘라내고 데이터 값 찾기;
    int Closing_Flag;
};
```

Frame의 구조는 위와 같이 flag, address, control, data, closing_flag 이렇게 구현이 되어있다. 각각의 frame에 따라 알맞은 값을 저장해준 후 send 하면 되고, 각각의 frame을 구별하는 값은 control 로 c언어의 macro 로 미리 지정해주었다. (I, S-frame 은 초깃값을 매크로로 지정, U-Frame은 각각 미리 정해져있는 값 지정)

- 각 메뉴 선택 시 화면에 보이는 메뉴화면에 없는 번호를 입력받으면 다시 입력받도록 구현하였다.
    - 만약 연결이 되지 않은 상황이라면 1번만 선택이 가능한데 1이 아닌 다른 값을 입력한다면 다시 메뉴 입력받도록 구현.
    - 연결 후 채팅기능 (2번) 또는 disconnect (3번) 이 아닌 다른 값을 입력한다면 다시 메뉴 입력받도록 구현.

2-1. 기능별 코드구현 (sender)

- 1번 기능 선택 시 (connect 가 아직 되지 않았을 경우)

```
// 사용자에게 1을 입력받을때까지 반복
while(1) {
    if(!is_connect) {
        print_connect_menu();
        int select = 0;
        scanf("%d", &select);
        if(select != 1) { // 1번을 선택하지 않았다면 다시 선택하도록
            printf("retry\n");
            continue;
        }

        printf("\n================================\n");
        printf("request connection. (SABM)\n");
        struct Frame sabm = {FLAG, SENDER_ADDRESS, SABM, "", FLAG};
        send(sock, &sabm, sizeof(sabm), 0);

        /* 통신 결과 수신까지 기다리기 */
        /* 수신받은 소켓(frame)의 정보를 우리가 따로 생성한 frame 에 저장 */
        /* valread : read 함수가 실제로 읽은 byte 수  */
        /* 우리가 수신받은 데이터는 f 에 저장되어있음. */
        valread = read(sock, &f, sizeof(struct Frame));

        printf("Received UA message.\n");
        check_flag(f);
        check_address(f);

        printf("connection success\n");
        is_connect = 1;
        printf("================================\n");
```

메뉴 출력 후 번호를 입력받아 SABM의 값으로 frame 생성 후 receiver에 보낸다.
그 후 receiver가 보낸 정보를 기다린 후 잘 받았는지 flag와 address 확인 후 연결을 성공시킨다.
연결이 되었는지 확인 여부는 is_connect라는 전역변수로 확인하므로 연결 성공 후 is_connect값을 1로 변경해준다.

- 2번 기능 (chat) 선택 시

```
} else { // 연결되었다면
    print_menu();
    int select = 0;
    scanf("%d", &select);

    // 입력 버퍼 비우기
    int c;
    while ((c = getchar()) != '\n' && c != EOF) {}

    switch(select){
        case 2: // chat mode
            while(1){
                // 보낼 메세지 입력받기
                printf("Send[SEQ: %d]: ", seq_num);

                fgets(f.Data, sizeof(f.Data), stdin);
                printf("\n");

                // if(check_str(f.Data, "quit\n") || check_str(f.Data, "exit\n")){
                //     printf("break chat\n");
                //     seq_num = 0;
                //     break;
                // }

                // frame 만들기
                f.Flag = FLAG;
                f.Address = SENDER_ADDRESS;
                f.Control = I_FRAME + ((seq_num % 8) << 4);
                f.Closing_Flag = FLAG;
                // struct Frame f2 = {FLAG, SENDER_ADDRESS, I_FRAME, d, FLAG};
                // struct Frame sabm = {FLAG, SENDER_ADDRESS, SABM, "", FLAG};
                seq_num++;

                // 보내기
                send(sock, &f, sizeof(f), 0);
```

보낼 메시지를 입력받아서 I-Frame을 만든다. 이 때 I-Frame에는 seq, ack 번호가 들어가야한다. 그래서 이 부분은 비트연산으로 값을 집어넣어주었다. I-Frame control에서 5~7번째 비트가 seq 번호가 들어가는 자리이다. 그렇기 때문에 int type 으로 1씩 더해주고 있는 seq_num을 4bit left shift 연산한 후 I-Frame 값에 더해준다. 그리고 이 때 값을 더해줄 때 그냥 seq_num을 더해버리면 3bit로 표현할수 있는 0~7의 범위를 넘어가게 된다. 그렇기 때문에 %8 연산을 해줘 0~7을 순환하도록 구현하였다.
이렇게 I-Frame을 다 만들었다면 receiver측으로 송신한다.

```
        if(check_str(f.Data, "quit\n") || check_str(f.Data, "exit\n")){
            printf("break chat\n");
            seq_num = 0;
            break;
        }

        // 수신
        valread = read(sock, &f, sizeof(f));
        for (int i = 0; f.Data[i] != '\0'; i++) {
            if (islower(f.Data[i])) {
                f.Data[i] = toupper(f.Data[i]);  // 소문자를 대문자로 변경
            } else if (isupper(f.Data[i])) {
                f.Data[i] = tolower(f.Data[i]);  // 대문자를 소문자로 변경
            }
        }

        if((seq_num % 8) != (f.Control % 8)){
            printf("send NAK\n");
            printf("seq_num : %d, f.Control: %d\n", (seq_num % 8), (f.Control % 8));
        } else {
            printf("correct!!\n");
            printf("seq_num : %d, f.Control: %d\n", (seq_num % 8), (f.Control % 8));
        }
        // printf("receive from client : %s\n", f.Data);
        // break;
    }
    break;
```

이렇게 보냈다면, receiver의 수신을 기다린다. 수신이 되었다면 수신받은 frame의 값을 확인한다.
먼저, 현재 seq 번호와 전달받은 control 의 ack 번호를 비교해 같지 않다면 중간에 한번 lost 된 것이므로 nak이 있음을 확인할 수 있다.
그렇지 않으면 올바르게 통신이 되고 있는 것을 확인할 수 있다.

그리고 chatting 과정에서 보내는 data가 “quit” 또는 “exit” 일때는 문자 송신 후 2, 3번 메뉴를 고르는 화면으로 이동한다.

- 3번 기능 (disconnect) 선택 시

```
case 3: // disconnect
    // f = {FLAG, SENDER_ADDRESS, DISC, "", FLAG};
    f.Flag = FLAG;
    f.Address = SENDER_ADDRESS;
    f.Control = DISC;
    strcpy(f.Data, "");
    f.Closing_Flag = FLAG;
    send(sock, &f, sizeof(f), 0);

    valread = read(sock, &f, sizeof(f));
    printf("\n================================\n");
    printf("Received UA message from receiver.\n");
    printf("Disconnect.\n");
    printf("================================\n");
    is_connect = 0; // 연결 해제
    seq_num = 0;
    continue; // 다시 연결하려는 코드 실행
```

3번 disconnect 기능을 수행하면 sender -> receiver로 U-Frame 중 DISC control을 저장해 보낸다. frame을 보낸 후 receiver 가 보내는 UA 메시지를 기다리다가 받으면 잘받았다는 것을 출력하고 연결을 해제 후 1번 connect 기능이 있는 메뉴화면을 출력하는 부분을 실행하도록 한다.

2-2. 기능 별 코드구현 (receiver)

- connect 기능 구현

```
// 사용자에게 1을 입력받을때까지 반복
while(1) {
    if(!is_connect) {
        /* 통신 결과 수신까지 기다리기 */
        /* 수신받은 소켓(frame)의 정보를 우리가 따로 생성한 frame 에 저장 */
        /* valread : read 함수가 실제로 읽은 byte 수  */
        /* 우리가 수신받은 데이터는 f 에 저장되어있음. */
        valread = read(new_socket, &f, sizeof(struct Frame));
        printf("\n================================\n");

        check_flag(f);
        check_address(f);

        printf("received message is SABM.\n");
        printf("Send UA message.\n");

        struct Frame ua = {FLAG, RECEIVER_ADDRESS, UA, "", FLAG};
        send(new_socket, &ua, sizeof(ua), 0);
        is_connect = 1;
        printf("================================\n\n");
```

receiver는 sender 가 보낸 frame을 받는 것부터 시작한다. 그래서 sender 가 보낸 U-Frame (SABM)을 수신 후 flag와 address를 체크하고 잘 받았다는 print문 출력한다. 그 후 잘 받았다는 response 로 UA 가 담긴 U-Frame을 sender에 전송 후 연결되었다는 is_connect의 값을 1로 설정한다.

- 2번 기능 (chat) 선택 시

```
} else { // 연결되었다면
    valread = read(new_socket, &f, sizeof(struct Frame));

    if((f.Control >> 7) == 0b0) {
        for (int i = 0; f.Data[i] != '\0'; i++) {
            if (islower(f.Data[i])) {
                f.Data[i] = toupper(f.Data[i]);  // 소문자를 대문자로 변경
            } else if (isupper(f.Data[i])) {
                f.Data[i] = tolower(f.Data[i]);  // 대문자를 소문자로 변경
            }
        }

        printf("Receiver[SEQ: %d, ACK: %d]: %s\n", seq_num, (count + 1), f.Data);

        if((count % 8) != (f.Control >> 4)) {
            printf("conut : %d, f.Control ; %d", count, f.Control);
            printf("NAK\n"); // 이론상 unreachable
            continue;
        }

        count++;

        // printf("receive from server : %s\n", f.Data);


        f.Flag = FLAG;
        f.Address = RECEIVER_ADDRESS;
        f.Control = I_FRAME + (ack_num % 8);
        f.Closing_Flag = FLAG;

        ack_num++;

        if(check_str(f.Data, "QUIT\n") || check_str(f.Data, "EXIT\n")){
            printf("break chat\n");
            count = 0;
            ack_num = 1;
            seq_num = 0;
            continue;
        }
        send(new_socket, &f, sizeof(f), 0);
```

sender 가 보낸 Frame을 받아서 control 의 값을 비교한다.
이 때 control 의 가장 처음 값이 I-Frame의 값 즉, 0이면 chat 기능이다라는 것으로 판단해주었다.
만약 chat 기능에서의 I-Frame이면 Frame 안의 data를 읽어와 대소문자를 변경해주고, seq, ack 번호와 함께 출력해주었다. 이때 seq_num 은 본인의 seq_num 즉, 0이고, ack 번호는 내가 다음에 받고자 하는 번호이다. 이 번호는 receive할 때 1씩 증가시켜 따로 저장하는 count를 출력해주었다.
그리고 우리는 실습에서는 정상적으로 통신되는 경우밖에 없으므로 중간에 lost 되는 경우가 없지만, 그런 경우를 구현해줘야 했다. 그래서 나는 현재 내가 받고자 했던 번호와, sender에서 넘어온 즉, sender 의 seq 번호를 비교해 같지 않다면 중간에 lost 된 경우가 있다라고

판단해 sender 에 nak을 보내줄 수 있도록 하였다. (이 부분은 정상적인 통신만 이뤄진다는 가정하에 진행된다면 unreachable 하므로 재전송 또는 nak를 보내지는 않고 print문으로 출력만 해주도록 구현했다.)
그 후에는 I-Frame에 receiver의 seq, ack 번호를 저장해 sender에게 보내주었다.
그리고 만약 sender 가 보내려고 했던 data가 “quit” 또는 “exit” 이었다면 사용했던 ack, seq, count 값을 모두 초기화시켜주었다.

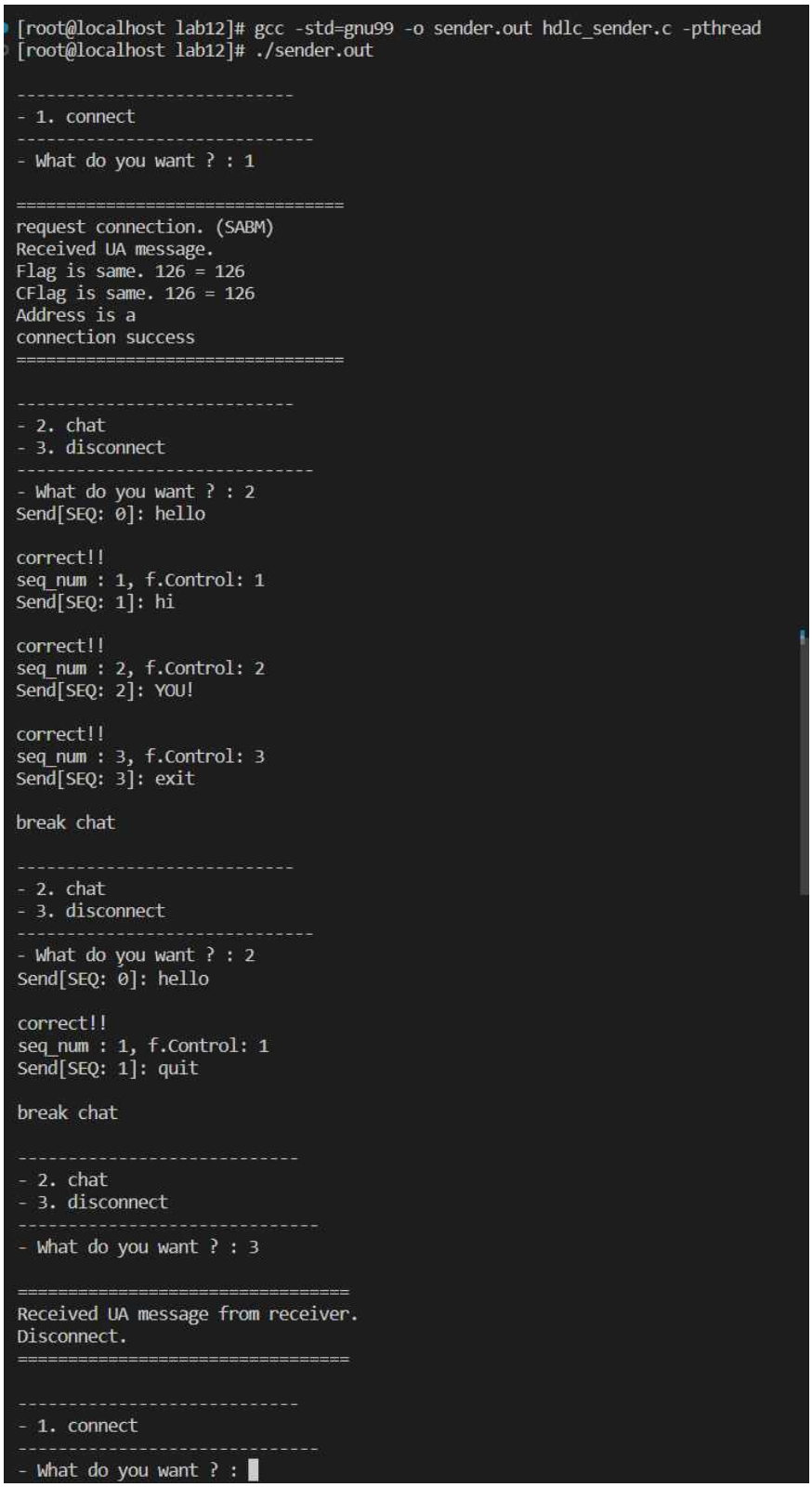
- 3번 기능 (disconnect) 선택 시

```
} else if((f.Control >> 6) == 0b11) {
    struct Frame ua = {FLAG, RECEIVER_ADDRESS, UA, "", FLAG};
    send(new_socket, &ua, sizeof(ua), 0);
    printf("\n================================\n");
    printf("send UA to sender\n");
    printf("Disconnect.\n");
    printf("================================\n");
    is_connect = 0;
    seq_num = 0;
    ack_num = 1;
    count = 0;
}
```

receiver에서는 sender에게 전달받은 frame이 어떤 frame인지 확인해 chatting 모드인지, disconnect 모드인지 판별해 수행하는 기능이 다르도록 구현했다.
그래서 3번 즉, disconnect 모드이면 sender에게 잘 받았다는 UA U-Frame을 보내고 is_connect 의 값을 연결 해제인 0으로 바꿔주고 모든 값들을 다시 초기화시켜주도록 하였다.
그리고 연결 해제되었다는 문구를 print 로 출력해 3번 기능까지 구현하였다.

3. 실행결과

[sender.c 실행]



[receiver.c 실행]

```
[root@localhost lab12]#  gcc -std=gnu99 -o receiver.out hdlc_receiver.c -pthr
ead
[root@localhost lab12]# ./receiver.out
Waiting for incoming connection...
Connection accepted

================================
Flag is same. 126 = 126
CFlag is same. 126 = 126
Address is b
received message is SABM.
Send UA message.
================================

Receiver[SEQ: 0, ACK: 1]: HELLO

Receiver[SEQ: 0, ACK: 2]: HI

Receiver[SEQ: 0, ACK: 3]: you!

Receiver[SEQ: 0, ACK: 4]: EXIT

break chat
Receiver[SEQ: 0, ACK: 1]: HELLO

Receiver[SEQ: 0, ACK: 2]: QUIT

break chat

================================
send UA to sender
Disconnect.
================================
```