**HW 5. Retransmission 알고리즘 구현 2**컴퓨터네트워크 02분반

컴퓨터공학과

2142851 김형준

**목차**

1. **Fast Retransmission**
   1. 코드
   2. 결과 출력
   3. 어려웠던 점
2. **Fast Retransmission**
3. **코드**

FR\_Sender.c

*// 2142851 컴퓨터공학과 김형준*

*// FR\_Sender.c*

*#include* "../Common.h"

*#include* <pthread.h>

*#define* SERVERPORT 9000

*#define* BUFSIZE    512

*#define* SIZE 5

bool isReceived[SIZE] = { false, false, false, false, false };

bool sender\_something\_wrong[SIZE] = { false, false, false, false, false };

int arrow = 0;

typedef struct {

    int index; *// 스레드 인덱스*

    SOCKET sock; *// 소켓 정보*

} ThreadArgs;

char packets[SIZE][BUFSIZE] = {

    "packet 0:I am a boy.",

    "packet 1:You are a girl.",

    "packet 2:There are many animals in the zoo.",

    "packet 3:철수와 영희는 서로 좋아합니다!",

    "packet 4:나는 점심을 맛있게 먹었습니다."

};

char correct\_ACK[SIZE + 1][BUFSIZE] = {

    "ACK = 0",

    "ACK = 11",

    "ACK = 26",

    "ACK = 60",

    "ACK = 103",

    "ACK = 146"

};

char last\_ACK[BUFSIZE];

int receive\_arrow = 0;

int pre\_receive = 0;

*// 서버로부터 메시지를 수신하는 함수*

void \*receive\_func(void \**arg*) {

    ThreadArgs \*args = (ThreadArgs \*)arg; *// thread arg*

    int index = args->index;

    SOCKET sock = args->sock;

    char buf[BUFSIZE + 1];

    int retval;

*while* (1) {

*// 데이터 받기*

        retval = recv(sock, buf, BUFSIZE, 0);

*if* (retval == SOCKET\_ERROR) {

            err\_display("recv()");

*break*;

        } *else* *if* (retval == 0) {

*break*;

        }

*// 받은 데이터 출력*

        buf[retval] = '\0';

*// printf("[TCP Sender] %d바이트를 받았습니다.\n", retval);*

        printf("(%s) is received.\n", buf);

*// break;*

*// if (strcmp(correct\_ACK, buf) == 0) {*

*//     break;*

*// }*

*// =======================================================================================*

        strcpy(last\_ACK, buf);

*if* (strcmp(correct\_ACK[receive\_arrow + 1], buf) == 0) {

            isReceived[receive\_arrow] = true;

            receive\_arrow++;

            pre\_receive = receive\_arrow + 1;

*break*;

        }

*else* {

            isReceived[pre\_receive] = true;

            sender\_something\_wrong[receive\_arrow] = true;

            pre\_receive++;

        }

*// printf("\n");*

*// printf("[isReceived]");*

*// printf("[%d %d %d %d %d]\n\n", isReceived[0], isReceived[1], isReceived[2], isReceived[3], isReceived[4]);*

    }

    int check\_i = 0;

*while* (1) {

*if* (!isReceived[check\_i]) {

            check\_i = 0;

*continue*;

        }

        check\_i++;

*if* (check\_i == SIZE) *break*;

    }

    close(sock);

    pthread\_exit(NULL);

}

*// Time out*

clock\_t startTime[SIZE], endTime[SIZE];

double elapsedTime[SIZE];

bool first\_packet1 = true;

*// 서버로 메시지를 보내는 함수*

void \*send\_func(void \**arg*) {

    ThreadArgs \*args = (ThreadArgs \*)arg; *// thread arg*

    int index = args->index;

    SOCKET sock = args->sock;

    bool is\_retransmittion = false;

*// 0.5 \* index second delay*

    usleep(500000 \* index);

    char buf[BUFSIZE + 1];

    int retval;

*while* (!isReceived[index]) {

*while* (arrow != index);

*// printf("%d is run\n", arrow);*

        arrow++;

        memset(buf, 0, sizeof(buf));

        strcpy(buf, packets[index]);

        int len = (int)strlen(buf);

*if* (buf[len - 1] == '\n')

            buf[len - 1] = '\0';

*if* (strlen(buf) == 0)

*break*;

*// first\_packet1 => no!!*

*// if (first\_packet1 && strcmp(packets[1], buf)) {*

*//     continue;*

*// }*

        retval = send(sock, buf, (int)strlen(buf), 0);

*if* (retval == SOCKET\_ERROR) {

            err\_display("send()");

*break*;

        }

        char send\_message[BUFSIZE];

        memcpy(send\_message, buf + 9, strlen(buf) - 9 + 1);

*if* (is\_retransmittion) {

            printf("packet %d is retransmitted. (%s)\n", index, send\_message);

        }

*else* {

            printf("packet %d is transmitted. (%s)\n", index, send\_message);

        }

        startTime[index] = clock();

*while* (!isReceived[index]) {

            endTime[index] = clock();

            elapsedTime[index] = (double)(endTime[index] - startTime[index]) / CLOCKS\_PER\_SEC;

*// ACK wrong*

*if* (sender\_something\_wrong[index]) {

*// printf("sender\_something\_wrong[%d]: true\n", index);*

*// 3 second delay*

                usleep(3000000);

                sender\_something\_wrong[index] = false;

                arrow = index;

*break*;

            }

*// 전송 후 10초가 지나면 time out*

*if* (elapsedTime[index] > 10) {

                printf("packet %d Time out\n", index);

                arrow = index;

*break*;

            }

        }

        is\_retransmittion = true;

    }

*// printf("[Thread %d is done]\n", index);*

    int check\_i = 0;

*while* (1) {

*if* (!isReceived[check\_i]) {

            check\_i = 0;

*continue*;

        }

        check\_i++;

*if* (check\_i == SIZE) *break*;

    }

    close(sock);

    pthread\_exit(NULL);

}

int main(int *argc*, char \**argv*[]) {

    int retval;

*// 명령행 인수가 있으면 IP 주소로 사용*

    char \*SERVERIP = (argc > 1) ? argv[1] : (char \*)"127.0.0.1";

*// 소켓 생성*

    SOCKET sock = socket(AF\_INET, SOCK\_STREAM, 0);

*if* (sock == INVALID\_SOCKET) err\_quit("socket()");

*// connect()*

    struct sockaddr\_in serveraddr;

    memset(&serveraddr, 0, sizeof(serveraddr));

    serveraddr.sin\_family = AF\_INET;

    inet\_pton(AF\_INET, SERVERIP, &serveraddr.sin\_addr);

    serveraddr.sin\_port = htons(SERVERPORT);

    retval = connect(sock, (struct sockaddr \*)&serveraddr, sizeof(serveraddr));

*if* (retval == SOCKET\_ERROR) err\_quit("connect()");

    pthread\_t r\_tid[SIZE], s\_tid[SIZE];

*// 스레드 인수 초기화*

    ThreadArgs r\_args[SIZE], s\_args[SIZE];

*for* (int i = 0; i < SIZE; i++) {

        r\_args[i].index = i;

        r\_args[i].sock = sock;

        s\_args[i].index = i;

        s\_args[i].sock = sock;

    }

*// 서버로부터 메시지를 수신하는 스레드 생성*

*for* (int i = 0; i < SIZE; i++) {

*if* (pthread\_create(&r\_tid[i], NULL, receive\_func, (void \*)&r\_args[i]) != 0) {

            printf("[TCP Sender] 수신 스레드 생성 실패\n");

            close(sock);

            exit(1);

        }

*// 0.1 second delay*

        usleep(100000);

    }

*// 서버로 메시지를 보내는 스레드 생성*

*for* (int i = 0; i < SIZE; i++) {

*if* (pthread\_create(&s\_tid[i], NULL, send\_func, (void \*)&s\_args[i]) != 0) {

            printf("[TCP Sender] 전송 스레드 생성 실패\n");

            close(sock);

            exit(1);

        }

    }

*for* (int i = 0; i < SIZE; i++) {

        pthread\_join(s\_tid[i], NULL);

    }

*// 소켓 닫기*

    close(sock);

*return* 0;

}

Receiver.c

*// 2142851 컴퓨터공학과 김형준*

*// FR\_Receiver.c*

*#include* "../Common.h"

*#include* <pthread.h>

*#include* <string.h>

*#define* BUFSIZE    512

*#define* SERVERPORT 9000

*#define* SIZE 5

typedef struct {

    int index; *// 스레드 인덱스*

    SOCKET sock; *// 소켓 정보*

} ThreadArgs;

bool send\_flag[SIZE] = { false, false, false, false, false };

bool is\_end[SIZE] = { false, false, false, false, false };

char ACK[BUFSIZE];

long int ACK\_num = 0;

int arrow = 0;

char packet[BUFSIZE];

char packets[SIZE][BUFSIZE] = { "packet 0", "packet 1", "packet 2", "packet 3", "packet 4" };

char message[BUFSIZE];

bool something\_wrong = false;

bool first\_packet\_loss = true;

*// return message length*

int message\_length(const char \**packet*) {

    const char \*a = strchr(packet, ':');

*if* (a == NULL) {

*return* 0;

    }

*return* strlen(a + 1);

}

*// checksum*

int checksum\_func(const char \**packet*) {

    int length = message\_length(packet);

*return* length;

}

*// 클라이언트로부터 메시지를 수신하는 함수*

void \*receive\_func(void \**arg*) {

    ThreadArgs \*args = (ThreadArgs \*)arg; *// thread arg*

    int index = args->index;

    SOCKET client\_sock = args->sock;

*// int client\_sock = \*((int \*)arg);*

    char buf[BUFSIZE + 1];

    int retval;

*while* (1) {

        bool checksum\_result;

*// 데이터 받기*

        memset(buf, 0, sizeof(buf));

        retval = recv(client\_sock, buf, BUFSIZE, 0);

*if* (retval == SOCKET\_ERROR) {

            err\_display("recv()");

*break*;

        } *else* *if* (retval == 0) {

            printf("[TCP Receiver %d] END\n", index);

*break*;

        }

*// checksum*

        int checksum = checksum\_func(buf);

*// printf("\nData Received.\n");*

*// printf("[DATA]: %s\n", buf);*

*// packet 부분을 추출하여 저장*

*for* (int i = 0; i < 8; ++i) {

            packet[i] = buf[i];

        }

        packet[8] = '\0';

*// message 부분을 추출하여 저장*

        int j = 0;

*for* (int i = 9; i < strlen(buf); ++i) {

            message[j++] = buf[i];

        }

        message[j] = '\0';

*// printf("[reciever %d]\n", index);*

*// printf("[packet: %s]\n", packet);*

*// printf("[message: %s]\n", message);*

*// 받은 데이터 출력*

        buf[retval] = '\0';

*// =================================== Data Check ===================================*

        int n;

        sscanf(packet, "packet %d", &n);

*// first packet loss*

*if* ((first\_packet\_loss) && (n == 1)) {

            first\_packet\_loss = false;

*continue*;

        }

*if* (strcmp(packet, packets[arrow]) != 0) {

*// printf("[Something Wrong!]\n");*

*// printf("[packet: %s]\n", packet);*

*// printf("[packet[%d]: %s]\n", arrow, packets[arrow]);*

            something\_wrong = true;

        }

*else* {

            something\_wrong = false;

            arrow++;

        }

*// checksum*

*if* (checksum == strlen(message)) {

            printf("%s is received and there is no error. (%s) ", packet, message);

        }

*else* {

            printf("%s is received and there is an error. (%s) ", packet, message);

        }

        send\_flag[n] = true;

        is\_end[n] = true;

*// printf("\nsend\_flag %d true\n", n);*

*// printf("[TCP 서버] %d바이트를 받았습니다.\n", retval);// 받은 데이터 출력*

*// printf("- Client: %s\n", buf);*

*// break;*

    }

    close(client\_sock);

    pthread\_exit(NULL);

}

*// 클라이언트로 메시지를 보내는 함수*

void \*send\_func(void \**arg*) {

    ThreadArgs \*args = (ThreadArgs \*)arg; *// thread arg*

    int index = args->index;

    SOCKET client\_sock = args->sock;

*// int client\_sock = \*((int \*)arg);*

    char buf[BUFSIZE + 1];

    int retval;

*while* (1) {

*// printf("\n\n[%d is wait..]\n\n", index);*

*while* (!send\_flag[index]);

*// printf("\n[Sender %d]: send\_flag is true\n", index);*

        char save\_message[BUFSIZE];

        strcpy(save\_message, message);

*// printf("\n\n====================================== CLEAR ======================================\n\n");*

        memset(packet, 0, sizeof(packet));

        memset(message, 0, sizeof(message));

*// printf("packet: %s\n", packet);*

*// printf("message: %s\n", message);*

        char send\_message[BUFSIZE] = "ACK = ";

*if* (!something\_wrong) {

            ACK\_num += strlen(save\_message);

        }

        something\_wrong = false;

*// printf("\n\n[%d is run!]\n\n", index);*

        send\_flag[index] = false;

*// printf("\nstrlen(save\_message) : %ld\n", strlen(save\_message));*

        sprintf(send\_message + strlen(send\_message), "%ld", ACK\_num);

        strcpy(ACK, send\_message);

        strcpy(buf, ACK);

        printf("(%s) is transmitted.\n", buf);

*// 5 second*

        usleep(5000000);

        int len = (int)strlen(buf);

*if* (buf[len - 1] == '\n')

            buf[len - 1] = '\0';

*if* (strlen(buf) == 0)

*break*;

        retval = send(client\_sock, buf, (int)strlen(buf), 0);

*if* (retval == SOCKET\_ERROR) {

            err\_display("send()");

*break*;

        }

*// break;*

    }

    close(client\_sock);

    pthread\_exit(NULL);

}

int main(int *argc*, char \**argv*[]) {

*// 소켓 생성*

    SOCKET listen\_sock = socket(AF\_INET, SOCK\_STREAM, 0);

*if* (listen\_sock == INVALID\_SOCKET) err\_quit("socket()");

    struct sockaddr\_in serveraddr;

    memset(&serveraddr, 0, sizeof(serveraddr));

    serveraddr.sin\_family = AF\_INET;

    serveraddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    serveraddr.sin\_port = htons(SERVERPORT);

*if* (bind(listen\_sock, (struct sockaddr \*)&serveraddr, sizeof(serveraddr)) == SOCKET\_ERROR)

        err\_quit("bind()");

*if* (listen(listen\_sock, SOMAXCONN) == SOCKET\_ERROR)

        err\_quit("listen()");

    struct sockaddr\_in clientaddr;

    socklen\_t addrlen = sizeof(clientaddr);

    SOCKET client\_sock = accept(listen\_sock, (struct sockaddr \*)&clientaddr, &addrlen);

*if* (client\_sock == INVALID\_SOCKET) {

        err\_display("accept()");

    }

    pthread\_t r\_tid[SIZE], s\_tid[SIZE];

*// 스레드 인수 초기화*

    ThreadArgs r\_args[SIZE], s\_args[SIZE];

*for* (int i = 0; i < SIZE; i++) {

        r\_args[i].index = i;

        r\_args[i].sock = client\_sock;

        s\_args[i].index = i;

        s\_args[i].sock = client\_sock;

    }

*// 클라이언트로부터 메시지를 수신하는 스레드 생성*

*for* (int i = 0; i < SIZE; i++) {

*if* (pthread\_create(&r\_tid[i], NULL, receive\_func, (void \*)&r\_args[i]) != 0) {

            printf("[TCP Receiver] 수신 스레드 생성 실패\n");

            close(client\_sock);

        }

*// printf("[Receiver Thread Create %d]\n", i);*

    }

*// 클라이언트로 메시지를 보내는 스레드 생성*

*for* (int i = 0; i < SIZE; i++) {

*if* (pthread\_create(&s\_tid[i], NULL, send\_func, (void \*)&r\_args[i]) != 0) {

            printf("[TCP Receiver] 전송 스레드 생성 실패\n");

            close(client\_sock);

        }

*// printf("[Sender Thread Create %d]\n", i);*

    }

*for* (int i = 0; i < SIZE; i++) {

        pthread\_join(s\_tid[i], NULL);

    }

*// 소켓 닫기*

    close(listen\_sock);

*return* 0;

}

1. **결과 출력**

Sender

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Receiver

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. **어려웠던 점**

**<(지연) 시간 제어>**

usleep 함수를 이용하여 시간 제어를 하였습니다. 하지만, 출력에서 순서가 바뀌고, packet 1, 2, 3의 내용이 합쳐져 한 번에 받아지는 등 문제점이 많았습니다. 이를 해결하기 위해 멀티 스레드를 사용하였습니다. 멀티 스레드를 사용하며 각각의 스레드에서 어떤 순서로 코드가 실행될 지 생각하는 부분이 어려웠습니다.

특히 receiver에서 packet을 받음을 알리는 메시지와 함께 ACK를 보내는 메시지가 출력되어야 하는데, 받음을 알리는 메시지만 한 번에 뜨고, 보냄을 알리는 메시지가 이후에 뜨는 문제점이 발생하였습니다. usleep 함수 사용의 논리적 오류가 있었기 때문에, 직접 그리고 설계하며 코딩하였습니다.

**<멀티 스레드 사용>**

위에서 언급했듯이, 멀티 스레드 각각의 기능을 병렬로 처리하려다 보니 어려움이 생겼습니다. 하지만 이번 과제를 통해 멀티 스레드 진행 과정에 대해 더 깊고 자세하게 이해할 수 있었습니다.

**<Check Sum>**

교재 내용의 Check Sum을 완벽하게 구현하지 못하였지만, message의 길이를 통해 잘 도착하였는지 확인하였습니다. 스레드를 사용하며 함수가 많이 생성된 상태에서 check sum 함수를 또 구현하려고 하니, 논리적인 어려움이 있었습니다. 혹여, 두 개 이상의 스레드가 check sum 함수에 동시에 들어가버리거나 변수의 값을 갑자기 바꿔버리는 등의 문제를 생각하며 지연 시간 제어를 활용했습니다.