**HW 6. 버퍼 overflow에 의한  
Packet loss 상황에서의 통신 구현**컴퓨터네트워크 02분반

컴퓨터공학과

2142851 김형준

**목차**

1. **PL\_Sender.c**
   1. 코드
   2. 결과 출력
2. **PL\_Receiver.c**
   1. 코드
   2. 결과 출력
3. **코드 간략한 설명**
4. **PL\_Sender.c**
5. **코드**

*// 2142851 컴퓨터공학과 김형준*

*// Sender*

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

*#include* <pthread.h>

*#define* SERVERPORT      9000

*#define* SIZE            32

*#define* BUFSIZE         50

*#define* PACKET\_SIZE     9

*#define* PAYLOAD\_SIZE    6

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

double elapsedTime[SIZE];

bool receive\_flag[SIZE];

typedef struct {

    int index; *// 스레드 인덱스*

    SOCKET sock; *// 소켓 정보*

} ThreadArgs;

typedef struct {

    char payload[PAYLOAD\_SIZE];

    short int checksum;

    char packet\_number;

} Packet;

Packet packet[32];

int ACK = 0;

int last\_ACK = 0;

*// ================================================================================= OUTPUT TEXT FILE =================================================================================*

FILE \*output;

int output\_setting() {

    output = fopen("output.txt", "w");

*if* (output == NULL) {

        perror("Write Failed.\n");

*return* 1;

    }

*if* (dup2(fileno(output), STDOUT\_FILENO) == -1) {

        perror("Error redirecting stdout.");

*return* 1;

    }

*return* 0;

}

*// ================================================================================= FILE PROCESSING =================================================================================*

void process\_file(FILE \**fp*) {

    char payload[PAYLOAD\_SIZE + 1];

    int packet\_number = 0;

*while* (fread(payload, 1, PAYLOAD\_SIZE, fp) == PAYLOAD\_SIZE) {

        payload[PAYLOAD\_SIZE] = '\0';

*// 띄어쓰기 부분 처리*

*for* (int i = 0; i < PAYLOAD\_SIZE; i++) {

*if* (payload[i] == ' ') {

                fseek(fp, -2, SEEK\_CUR);

                payload[PAYLOAD\_SIZE - 2] = '\0';

            }

        }

*// printf("Packet %2d - payload: %s\n", packet\_number, payload);*

        memcpy(packet[packet\_number].payload, payload, 6);

*// checksum error..*

*// packet[packet\_number].checksum = 1;*

        packet[packet\_number].packet\_number = (char)packet\_number;

        packet\_number++;

    }

*// 출력*

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

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

*//     printf("[Packet %2d] payload: %s\n", i, packet[i].payload);*

*//     printf("[Packet %2d] checksum: %c\n", i, packet[i].checksum);*

*//     printf("[Packet %2d] packet\_number: %d\n", i, packet[i].packet\_number);*

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

*// }*

}

*// ================================================================================= RECEIVE FUNC =================================================================================*

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*;

        }

        buf[retval] = '\0';

        int ACK\_number;

        sscanf(buf, "(ACK = %d)", &ACK\_number);

        ACK = ACK\_number;

        receive\_flag[ACK\_number / PACKET\_SIZE - 1] = true;

*if* (last\_ACK == ACK) {

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

        }

*else* {

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

        }

        last\_ACK = ACK;

*break*;

    }

*// close(sock);*

    pthread\_exit(NULL);

}

*// ================================================================================= SEND FUNC =================================================================================*

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

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

    int index = args->index;

    SOCKET sock = args->sock;

    int retval;

*// printf("send\_func %d is run\n", index);*

*while* (1) {

        Packet buf;

        memcpy(&buf, &packet[index], sizeof(Packet)); *// packet[index]의 내용을 buf로 복사*

*// printf("buf: %s\n", buf.payload);*

        retval = send(sock, &buf, sizeof(Packet), 0); *// Packet 구조체의 크기만큼 전송*

*// printf("retval: %d\n", retval);*

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

            printf("send() failed with error: %d\n", errno);

            perror("send() error");

            err\_display("send()");

*break*;

        }

        printf("packet %2d is transmitted. (%s)\n", index, buf.payload);

        startTime[index] = clock();

*while* ((!receive\_flag[index]) && (elapsedTime[index] < 0.5)) {

            endTime[index] = clock();

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

*// printf("elapsedTime[%d]: %lf\n", index, elapsedTime[index]);*

        }

*if* (receive\_flag[index]) {

*// printf("is received and ");*

*break*;

        }

*else* {

            printf("packet %2d is timeout.\n", index);

*break*;

        }

    }

*// close(sock);*

    pthread\_exit(NULL);

}

*// ================================================================================= MAIN =================================================================================*

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

*// fopen*

    FILE \*fp = fopen("text.txt", "r");

*if* (fp == NULL) {

        perror("Read Failed.\n");

*return* 1;

    }

    output\_setting();

    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];

*// FILE PROCESSING*

    process\_file(fp);

*// 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);

        }

    }

*// 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);

        }

        usleep(50000); *// 0.05초 지연*

    }

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

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

    }

    printf("END\n");

    fclose(output);

    fclose(fp);

*// 소켓 닫기*

    close(sock);

*return* 0;

}

1. **결과 출력**

텍스트, 스크린샷, 메뉴이(가) 표시된 사진

자동 생성된 설명

…

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. **PL\_Receiver.c**
2. **코드**

*// 2142851 컴퓨터공학과 김형준*

*// FR\_Receiver.c*

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

*#include* <pthread.h>

*#include* <string.h>

*#define* SERVERPORT      9000

*#define* SIZE            32

*#define* BUFSIZE         50

*#define* PACKET\_SIZE     9

*#define* PAYLOAD\_SIZE    6

typedef struct {

    int index; *// 스레드 인덱스*

    SOCKET sock; *// 소켓 정보*

} ThreadArgs;

*// 패킷에 저장될 데이터들*

typedef struct {

    char payload[PAYLOAD\_SIZE];

    short int checksum;

    char packet\_number;

} Packet;

Packet packet[32];

*// "받았으니 보내라"를 의미하는 변수*

bool send\_flag[SIZE];

*// ACK 설정 (데이터의 크기만큼 증가)*

*// 버퍼에 더 못 넣을 경우 -> ACK 증가 X, buf\_overflow true로 변경*

int ACK = 0;

bool buf\_overflow;

*// 버퍼에 저장 (최대 크기 : 50)*

char buf\_save[BUFSIZE];

*// receive된 버퍼의 크기*

int buf\_size;

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;

}

*// ================================================================================= RECEIVE FUNC =================================================================================*

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

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

    int index = args->index;

    SOCKET client\_sock = args->sock;

    char buf[BUFSIZE + 1];

    int retval;

    int receive\_index;

*while* (1) {

*// 데이터 받기*

*// printf("Receive Thread %d is ready.\n", index);*

        retval = recv(client\_sock, &packet[index], sizeof(Packet), 0);

*// 인덱스 변경 필요!!*

*// 받은 packet\_number를 server receive\_func의 index에 넣기*

        receive\_index = packet[index].packet\_number;

*// CHECK retval*

*// printf("retval: %d\n", retval);*

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

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

            err\_display("recv()");

*break*;

        } *else* *if* (retval == 0) {

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

*break*;

        }

*// 받은 데이터 출력*

        buf[retval] = '\0';

        usleep(100000); *// 0.1초 지연*

*// printf("(int)strlen(buf\_save) + (int)strlen(packet[%d].payload) : %d\n", index, (int)strlen(buf\_save) + (int)strlen(packet[index].payload));*

*if* ((int)strlen(buf\_save) + (int)strlen(packet[index].payload) >= BUFSIZE) {

            buf\_overflow = true;

        }

*else* {

            strcat(buf\_save, packet[index].payload);

        }

        buf\_size = 0;

        buf\_size += (int)sizeof(packet[index].payload);

        buf\_size += (int)sizeof(packet[index].checksum);

        buf\_size += (int)sizeof(packet[index].packet\_number);

        printf("packet %2d is received and there is no error. (%s)\n", packet[index].packet\_number, packet[index].payload);

        send\_flag[receive\_index] = true;

*break*;

    }

*// close(client\_sock);*

    pthread\_exit(NULL);

}

*// ================================================================================= SEND FUNC =================================================================================*

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) {

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

*if* (!buf\_overflow) {

            ACK += buf\_size;

*// ACK += (int)sizeof(Packet);*

*// printf("sizeof(Packet.payload) = %d\n", (int)sizeof(packet[index].payload));*

*// printf("sizeof(Packet.checksum) = %d\n", (int)sizeof(packet[index].checksum));*

*// printf("sizeof(Packet.packet\_number) = %d\n", (int)sizeof(packet[index].packet\_number));*

        }

        sprintf(buf, "(ACK = %d)", ACK);

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

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

*break*;

    }

*// close(client\_sock);*

    pthread\_exit(NULL);

}

*// ================================================================================= MAIN =================================================================================*

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);

    }

    printf("\n[Buf Save]\n%s\n", buf\_save);

*// 소켓 닫기*

    close(listen\_sock);

    close(client\_sock);

*return* 0;

}

1. **결과 출력**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명

1. **코드 간략한 설명**

먼저, 각 코드마다 receive thread와 send thread를 보낼 개수만큼 할당하였습니다. 이후 각 스레드에 time out을 체크하고, ACK가 오거나, time out이 될 시, thread를 종료하게 하였습니다.

한글의 바이트는 2로 할 경우, 깨지는 문제가 발생하여 3으로 하였고, 그에 따라 ACK는 3\*2 + 2[short int] + 1[char] = 9를 가집니다.

Receiver에서 payload를 save\_buf에 저장하고, 다음 받을 payload를 save\_buf에 저장할 때, 50바이트를 넘기게 될 경우, ACK는 증가하지 않았습니다. 이를 통해 버퍼 overflow 처리와 Packet loss를 구현하였습니다. 따라서, ACK는 save\_buf에 payload 크기를 미리 넣어보고, 50을 넘는 순간에 증가하지 않습니다. (ACK: 최대 9번까지 증가. 따라서 ACK는 81에서 증가하지 않음)

ACK가 9번까지 증가하는 이유는 띄어쓰기가 포함될 경우 payload의 크기가 6이 아닌, 4가 되기 때문입니다.

다만, 실행 환경과 컴퓨터에 따라 ACK 증가 값에 변동(오류)이 있는 문제점을 확인하였습니다. 이 부분은 Sender의 지연시간을 늦출 시 안정적이게 작동하게 됩니다.

제 컴퓨터의 경우 Sender의 지연시간을 최소 0.08초로 설정할 시 안정적인 구동이 가능합니다.