HW3

2021115744_권구태

1. **TCP** 데이터 송수신**(SEQ, ACK)**을 이용한 파일 전송 프로그램

[클라이언트**]**

1) 클라이언트는 사용자로부터 전송 받을 파일의 이름을 입력 받고, 서버에 해당

파일 이름을 전송함 (Packet 구조체의 buf[]에 파일 이름을 저장하여 전송)

- seq=0, ack=0으로 설정
- buf_len: 파일 이름의 길이

```
   // 파일 이름 받기
   printf("Input file name: ");
   fgets(pckt.buf, BUF_SIZE, stdin); // 공백이 없는 문자열을 입력할 땐
fgets이 유용
   size_t len = strlen(pckt.buf);
   if (len > 0 && pckt.buf[len - 1] == '\n') {
       pckt.buf[len - 1] = '\0';
   }
   char title[BUF_SIZE];
   strcpy(title, pckt.buf); // 입력받은 파일명 복사 해놓기. 나중에 써야 함.
   pckt.buf_len = strlen(pckt.buf);
   pckt.seq = 0;
   pckt.ack = 0;
   if (pckt.buf_len > 0 && pckt.buf[pckt.buf_len - 1] == '\n') {
       pckt.buf[pckt.buf_len - 1] = '\0';
   }

   printf("[Client] request %s\n\n", pckt.buf);
   // 서버 파일 이름 전송
   write(sock, pckt.buf, pckt.buf_len + 1);
   int recvLegth = 0; // 총 읽어들인 바이트 수를 저장하기 위함.
```

2) 클라이언트는 서버에서 전송한 파일 수신이 완료되면 소켓을 종료

```
if (pckt.buf_len < 100) {
            break;
        }
```

```
   printf("%s recieved (%d Bytes)\n", title, recvLegth);
```

```
    close(sock);
    printf("Exit client\n");
    return 0;
```

[서버**]**

1) 서버는 클라이언트에게 전송 받은 파일이름을 확인해서 파일이 있는 경우, 해당 파일을 전송함

```
// 파일 이름을 읽고 파일 이름이 있으면 열고, 없으면 에러 발생 표시
    read_len=read(clnt_sock, recv_msg, BUF_SIZE);
    if (read_len == -1) {
        error_handling("read() error!");
    } else {
        …
      }
    }

    while ((pckt.buf_len = read(fd, pckt.buf, BUF_SIZE)) > 0) { // 파일
내의 데이터를 읽음
        printf("[Server] Tx: SEQ: %d, %d byte data\n", pckt.seq,
pckt.buf_len);
        // 파일 내의 데이터를 client로 전송
        write(clnt_sock, pckt.buf, pckt.buf_len);
```

2) 파일이 서버에 없는 경우, buf[]에 "File Not Found"를 저장하여 클라이언트에게 전송하고 소켓 종료

- seq=0, ack=0으로 설정
- buf_len: “File Not Found”의 문자열 길이

```
fd = open(recv_msg, O_RDONLY);

        if (fd == -1) {
            printf("%s File Not Found\n", recv_msg);
            strcpy(pckt.buf, "File Not Found");
            pckt.seq = 0;
            pckt.ack = 0;
            pckt.buf_len = strlen(pckt.buf);
            //"File Not Found" client로 전송
            write(clnt_sock, pckt.buf, pckt.buf_len + 1);

            close(clnt_sock);
            close(serv_sock);
            printf("Exit server\n");
```

```
            return 0;  // 디렉토리를 열 수 없음
```

[TCP 데이터 송수신 과정 출력**]**

1. **hw3_server.c** 의 코드 일부

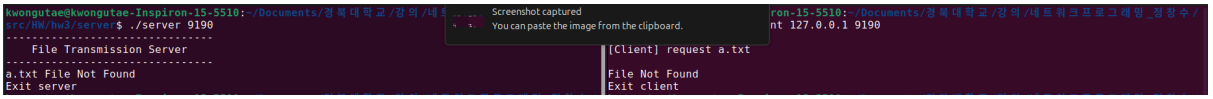
```
   pckt.seq = SEQ_START;
      (…)

      printf("[Server] sending %s\n\n", recv_msg);

   while ((pckt.buf_len = read(fd, pckt.buf, BUF_SIZE)) > 0) { // 파일
내의 데이터를 읽음
       printf("[Server] Tx: SEQ: %d, %d byte data\n", pckt.seq,
pckt.buf_len);
       // 파일 내의 데이터를 client로 전송
       write(clnt_sock, pckt.buf, pckt.buf_len);
       // 지금까지 읽은 바이트 수 업데이트
       sentLegth += pckt.buf_len;

       // 현재의 SEQ 값을 client로 전송
       write(clnt_sock, &pckt.seq, sizeof(pckt.seq));

       // 클라리언트가 ack 값 read
       read(clnt_sock, &pckt.ack, sizeof(int));

       if (pckt.buf_len < 100) {
           break;
       } else {
           printf("[Server] Rx ACK: %d\n\n", pckt.ack);
           // seq 값 업데이트
           pckt.seq = pckt.ack;
       }
   }

   printf("%s sent (%d Bytes)\n", recv_msg, sentLegth);
   printf("Exit server\n");
   close(clnt_sock);
   close(serv_sock);
   return 0;
```

2. hw3_client.c의 코드 일부

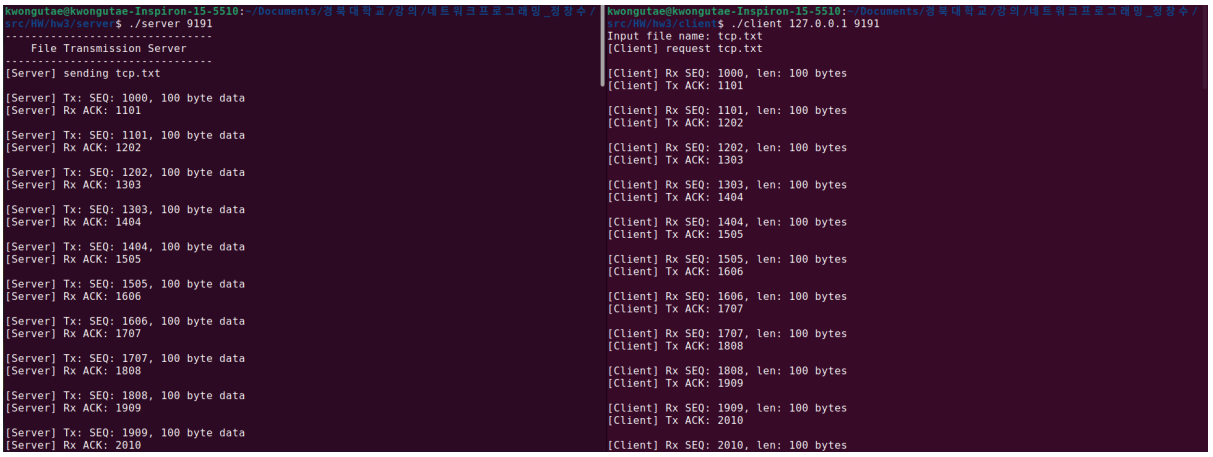
```
    printf("[Client] request %s\n\n", pckt.buf);
    // 서버 파일 이름 전송
    write(sock, pckt.buf, pckt.buf_len + 1);
    int recvLegth = 0; // 총 읽어들인 바이트 수를 저장하기 위함.

    while (1)
    {
        pckt.buf_len=read(sock, pckt.buf, BUF_SIZE); // 파일의 데이터를
읽는 read 함수
        if (pckt.buf_len == 0) {  // 서버가 연결을 종료한 경우
            break;
        } else if (pckt.buf_len == -1) {
            error_handling("read() error!");
            break;
        } else if (strcmp(pckt.buf, "File Not Found") == 0) {
            printf("File Not Found\n");
            printf("Exit client\n");
            return 0;
        }
        // SEQ의 넘버를 읽는 read 함수
        read(sock, &pckt.seq, sizeof(pckt.seq));  // &pckt.seq

        printf("[Client] Rx SEQ: %d, len: %d bytes\n", pckt.seq,
pckt.buf_len);
        // 지금까지 읽은 바이트 수 업데이트
        recvLegth += pckt.buf_len;

        if (pckt.buf_len < 100) {
            break;
        } else {
            // ack 값 업데이트
            pckt.ack = pckt.seq + pckt.buf_len + 1;
            // ack를 계산하고 서버로 보내는 write 함수
            write(sock, &pckt.ack, sizeof(pckt.ack)); // &pckt.ack
            printf("[Client] Tx ACK: %d\n\n", pckt.ack);
        }
    };
    printf("%s recieved (%d Bytes)\n", title, recvLegth);
    close(sock);
    printf("Exit client\n");
    return 0;
```

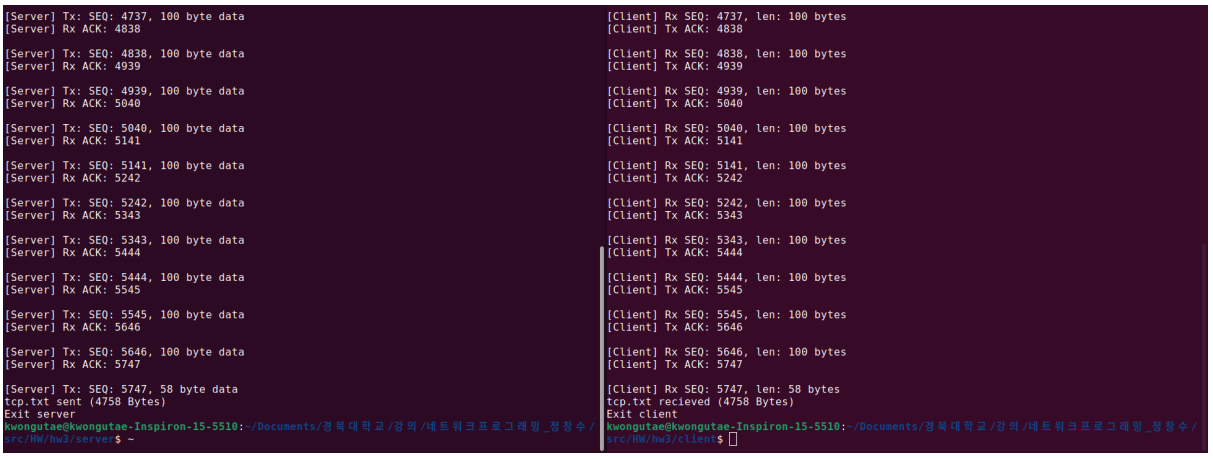
[실행 결과]

1. 존재하지 않는 파일 입력시



2. 존재하는 파일 입력시





[새롭게 알게된 점 및 주의사항]

1. 하나의 while문 안에 read-write 문을 여러개 사용하며 과제를 풀다보니, read와 write의 작동 순서 및 방식을 완벽히 익힘. 둘 중 하나라도 read가 나오면 통신 소켓에 데이터가 들어올때까지 기다리고, 다른 쪽에서 write를 해줘야만 다음 코드가 실행이 된다.
2. read 함수의 두번째 인자에는 주소값을 넘겨줘야 한다는 점을 간과.
3. read 함수가 읽어들인 바이트 수만큼을 int 값으로 반환하지만, write 자체에서 100바이트를 보내게 되면, 실제 유효 데이터가 58 바이트더라도 100바이트를 모두 읽음. strlen(pckt.buf) -> 100이 출력됨. 따라서 write

함수를 작성할 때부터 write(clnt_sock, pckt.buf, strlen(pckt.buf)); 대신, 처음부터 파일을 읽었을 때 저장한 write(clnt_sock, pckt.buf, pckt.buf_len); 해당 값을 넣고 전송해야 client read 함수도 58 바이트 수만큼 읽어들임.