

2022년도

컴퓨터정보과 TCP/IP 기말 프로젝트

코드 리뷰

(분야 : 멀티 쓰레드 기반 다중 접속 서버)

이름 : 이교범

학번 : 201644056

2022. 6. 15



인하공업전문대학
INHA TECHNICAL COLLEGE
仁荷工業專門大學

본 과제(결과물)는 인하공업전문대학 컴퓨터정보과 TCP/IP 교과목의 기말고사
대체 코드리뷰 입니다.

CHAT_SERV.C 코드 (채팅 서버 코드)

<헤더파일>

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <string.h>
5  #include <arpa/inet.h>
6  #include <sys/socket.h>
7  #include <netinet/in.h>
8  #include <pthread.h>
9
```

1~8. 표준 입출력, 표준 라이브러리, 문자열 처리, 인터넷 프로토콜, 소켓 함수, 인터넷 주소 체계, 스레드 함수를 사용하기 위한 헤더파일들을 정의합니다.

<매크로 정의>

```
14 #define BUF_SIZE 100
15 #define MAX_CLNT 256
```

14~15. 채팅시 최대 메시지 길이와, 최대 동시 접속자 수를 지정한 매크로들입니다.

<서버에서 사용하는 함수 선언>

```
20 void * handle_clnt(void * arg);
21 void send_msg(char * msg, int len);
22 void error_handling(char * msg);
```

20~22. 소켓을 통해 클라이언트들과 통신하는 함수인 `handle_clnt`, 접속한 사용자들에게 메시지를 전달해주는 함수 `send_msg`, 예외 처리해주는 함수 `error_handling`을 선언합니다.

<변수 선언>

```
26 int clnt_cnt=0;
27 int clnt_socks[MAX_CLNT];
28 pthread_mutex_t mutex;
```

26~28. 현재 접속 중인 사용자 수인 clnt_cnt 변수, 정의된 매크로 MAX_CLNT값 256을 가진 배열 clnt_socks 선언(공유자원), 공유 자원에 한번에 하나의 스레드만 접근할 수 있도록 허용하는 MUTEX 객체인 mutex 선언합니다.

<메인 함수>

```
32 int main(int argc, char *argv[])
33 {
34     int serv_sock, clnt_sock;
35     struct sockaddr_in serv_adr, clnt_adr;
36     int clnt_adr_sz;
37     pthread_t t_id;
38
39
40     if(argc!=2)
41     {
42         printf("Usage : %s <port>\n", argv[0]);
43         exit(1);
44     }
45
46     pthread_mutex_init(&mutex, NULL);
47     serv_sock=socket(PF_INET, SOCK_STREAM, 0);
48
49     memset(&serv_adr, 0, sizeof(serv_adr));
50     serv_adr.sin_family=AF_INET;
51     serv_adr.sin_addr.s_addr=htonl(INADDR_ANY);
52     serv_adr.sin_port=htons(atoi(argv[1]));
53
54
55     if(bind(serv_sock, (struct sockaddr*)&serv_adr, &sizeof(serv_adr))==-1)
56         error_handling("bind() error");
57
58
59     if(listen(serv_sock, 5)==-1)
60         error_handling("listen() error");
61
62     while(1)
63     {
64         clnt_adr_sz=sizeof(clnt_adr);
65         clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_adr, &clnt_adr_sz);
66         pthread_mutex_lock(&mutex);
67         clnt_socks[clnt_cnt++]=clnt_sock;
68         pthread_mutex_unlock(&mutex);
69         pthread_create(&t_id, NULL, handle_clnt, (void*)&clnt_sock);
70         pthread_detach(t_id);
71         printf("Connected client IP: %s \n", inet_ntoa(clnt_adr.sin_addr));
72     }
73     close(serv_sock);
74     return 0;
75 }
```

<메인함수 - 변수선언>

```
34     int serv_sock, clnt_sock;
35     struct sockaddr_in serv_adr, clnt_adr;
36     int clnt_adr_sz;
37     pthread_t t_id;
```

34~37. 소켓 통신용 서버 소켓, 클라이언트 소켓 선언, 서버 주소와 클라이언트 주소의 구조체 선언, 클라이언트 구조체 주소를 담는 변수, 클라이언트 쓰레드용 ID를 선언합니다.

<메인함수 - 입력안내>

```
40     if(argc!=2)
41     {
42         printf("Usage : %s <port>\n", argv[0]);
43         exit(1);
44     }
```

40~44. 서버포트 입력시 부정확한 정보가 입력되면, 정확한 정보를 입력할 수 있도록 안내 문구를 출력시킵니다.

<메인함수 - 소켓 설정 및 mutex 초기화>

```
46     pthread_mutex_init(&mutex, NULL);
47     serv_sock=socket(PF_INET, SOCK_STREAM, 0);
48
49     memset(&serv_adr, 0, sizeof(serv_adr));
50     serv_adr.sin_family=AF_INET;
51     serv_adr.sin_addr.s_addr=htonl(INADDR_ANY);
52     serv_adr.sin_port=htons(atoi(argv[1]));
```

46~52. mutex객체를 초기화, 서버 소켓을 IPv4, TCP로 이용하기 위하여 SOCK_STREAM을 이용해 TCP 소켓 생성, 서버 소켓의 정보를 담은 구조체를 0으로 초기화, 서버 주소를 Ipv4로 초기화, ip를 할당해주는데 long 자료형의 변수를 호스트 바이트순서에서 네트워크 바이트 순서로 변환 빅 엔디안으로 변경해주고 서버의 모든 아이피의 접근을 허용, port번호를 할당함 short형의 port번호를 호스트 바이트 순서에서 네트워크 바이트 순서로 변환합니다.

<메인함수 - 서버 소켓 bind>

```
55     if(bind(serv_sock, (struct sockaddr*) &serv_addr, &sizeof(serv_addr))==-1)
56         error_handling("bind() error");
```

55~56. bind함수를 통해 serv_sock을 통해 받은 파일디스크립터에 serv_addr 서버주소를 할당하고, 해당 주소 정보를 담은 변수를 통해 성공여부를 확인합니다. bind()는 성공시 0, 실패시 -1을 반환하며, -1일 경우 예외처리 함수 error_handling 호출하여 에러 전달합니다.

<메인함수 - 연결 요청 수락 대기상태>

```
59     if(listen(serv_sock, 5)==-1)
60         error_handling("listen() error");
```

59~60. 클라이언트가 서버소켓에 연결할 수 있도록 대기하는 상태를 만드는 함수로써 listen()함수가 호출되어야 클라이언트에서 connect함수를 호출할 수 있습니다. 연결 요청을 대기하는 큐 5개를 만들었으며, 최대 5개까지 가능합니다. 에러발생시 error_handling 에러값 전달하여 호출합니다.

<메인함수 - 연결 무한 루프 함수>

```
62     while(1)
63     {
64         clnt_addr_sz=sizeof(clnt_addr);
65
66         clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_addr, &clnt_addr_sz);
67         pthread_mutex_lock(&mutex);
68         clnt_socks[clnt_cnt++]=clnt_sock;
69         pthread_mutex_unlock(&mutex);
70         pthread_create(&t_id, NULL, handle_clnt, (void*)&clnt_sock);
71         pthread_detach(t_id);
72         printf("Connected client IP: %s \n", inet_ntoa(clnt_addr.sin_addr));
73     }
74     close(serv_sock);
75     return 0;
```

62~69. While 무한루프를 돌고, 클라이언트 구조체의 크기를 얻습니다. 클라이언트 소켓으로부터 연결 요청이 들어오면 연결 수락, clnt_socks[] 값은 공유자원으로 사용되며, 공유자원으로 사용될 때 해당 값을 변경하는 clnt_socks[clnt_cnt++]구문이 동작 되기전 다른 사용자가 서버에 접속하여 해당 값이 변경될 수 있기에 mutex를 이용하여 상호배제를 진행한다.

상호배제란 특정시점에 한 개의 프로세스만 사용할 수 있도록 하고, 다른 프로세스는 접근을 금지하는 것이다. mutex_lock을 이용하여 clnt_cnt 값 변경 전 lock을 걸고 값이 변경되어지면 unlock하여 다른 프로세스가 값을 변경할 수 있도록 합니다.

70~72. 쓰레드 생성하며 handle_clnt 함수 실행하고, 값으로는 클라이언트 구조체의 주소를 넘기고 구조체 주소에는 포트가 포함되어 있습니다. pthread_detach를 통해 실행된 쓰레드 종료시 사용했던 자원을 자동으로 반납한다. 접속한 클라이언트 IP를 서버 콘솔에 출력합니다.

74~75. 쓰레드가 끝나면 서버소켓을 종료한다.

<클라이언트와 통신하는 함수 handle_clnt>

```
79 void * handle_clnt(void * arg)
80 {
81     int clnt_sock=*((int*)arg);
82     int str_len=0, i;
83     char msg[BUF_SIZE];
84
85     while((str_len=read(clnt_sock, msg, sizeof(msg)))!=0)
86         send_msg(msg, str_len);
87     pthread_mutex_lock(&mutx);
88     for(i=0; i<clnt_cnt; i++)
89     {
90         if(clnt_sock==clnt_socks[i])
91         {
92             while(i++<clnt_cnt-1)
93                 clnt_socks[i]=clnt_socks[i+1];
94             break;
95         }
96     }
97     clnt_cnt--;
98     pthread_mutex_unlock(&mutx);
99     close(clnt_sock);
100     return NULL;
101 }
```

79~83. 매개변수로 넘어온 클라이언트 주소값을 소켓 변수에 선언하여 넣어주고, 클라이언트 메시지 전송 여부 확인하는 변수, 메시지 담을 문자형 배열 선언합니다.

85~86. while반복문으로 클라이언트와 통신하는 동안은 계속 돌아가며, 메시지 수신후, 받은 메시지를 send_msg 함수에 매개변수로 전달합니다.

87~96. 클라이언트가 종료되었을 때 총 접속자 수인 clnt_socks[(공유자원) 값을 변경해야되기 때문에 mutex를 이용하여 lock을 걸어 변경중에 다른 프로세스가 접근할 수 없도록 한다. for 반복문을 통해 현재 종료되어지는 clnt_sock(매개변수로 받은 클라이언트 소켓 값)을, clnt_socks[(전역변수로 총 클라이언트 소켓 값을 가지고 있는 배열)에서 찾은 후, while반복문을 통해 현재 종료된

클라이언트 값 자리부터 그 다음에 있던 값들을 하나씩 반복문에서 돌아 추가해준다.

97~101. 클라이언트 한 명이 종료되었기 때문에 총 클라이언트 수를 줄이고, 더 이상 clnt_cnt 값을 변경하지 않기에, unlock을 통해 mutex를 해제해준다. 클라이언트 소켓을 닫고, null 값을 리턴합니다.

<접속한 클라이언트들에게 메시지를 전달하는 함수 send_msg>

```
103 void send_msg(char * msg, int len)
104 {
105     int i;
106     pthread_mutex_lock(&mutex);
107     for(i=0; i<clnt_cnt; i++)
108         write(clnt_socks[i], msg, len);
109     pthread_mutex_unlock(&mutex);
110 }
```

103~110. 전체 접속한 클라이언트에게 메시지를 전송하기 위해 clnt_sock[]소켓 정보를 이용해야 하는데 참조하는 동안 값이 변경되지 않기 위하여 mutex_lock을 건다. 이후 for 반복문을 통해 전체 사용자들만큼 반복되어지고 clnt_socks[i]에 저장된 클라이언트 소켓으로 전달받은 메시지 값을 전송하고 전송이 완료되면 mutex_unlock으로 해제합니다.

<예외 처리를 담당하는 함수 void error_handling>

```
113 void error_handling(char * msg)
114 {
115     fputs(msg, stderr);
116     fputc('\n', stderr);
117     exit(1);
118 }
```

113~118. 에러함수 호출시에 전달받은 에러메세지 값을 출력합니다.

CHAT_CLINT.C 코드 (채팅 클라이언트 코드)

<헤더파일>

```
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <unistd.h>
5  #include <string.h>
6  #include <arpa/inet.h>
7  #include <sys/socket.h>
8  #include <pthread.h>
9
10 #define BUF_SIZE 100
11 #define NAME_SIZE 20
```

2~8. 표준 입출력, 표준 라이브러리, 문자열 처리, 인터넷 프로토콜, 소켓 함수, 쓰레드 함수입니다.
10~11. 매크로 메시지 최대길이를 100으로 사이즈 지정, 닉네임 사이즈를 최대 20으로 사이즈 지정합니다.

<클라이언트에서 사용되는 함수 및 변수 선언>

```
13 void * send_msg(void * arg);
14 void * recv_msg(void * arg);
15 void error_handling(char * msg);
16
17 char name[NAME_SIZE]="[DEFAULT]";
18 char msg[BUF_SIZE];
```

13~15. 송신을 담당하는 함수 send_msg, 수신을 담당하는 함수 recv_msg, 에러 발생시 에러 출력하는 error_handling 함수를 선언합니다.

17~18. 문자열로 채팅방에서 사용할 name을 DEFAULT 값으로 초기화 시키고, 채팅방에서 이용될 msg 값도 문자열 배열로 초기화 진행합니다.

<메인함수>

```
21 int main(int argc, char *argv[])
22 {
23     int sock;
24     struct sockaddr_in serv_addr;
25     pthread_t snd_thread, rcv_thread;
26     void * thread_return;
27     if(argc!=4)
28     {
29         printf("Usage : %s <IP> <port> <name>\n", argv[0]);
30         exit(1);
31     }
32
33     sprintf(name, "[%s]", argv[3]);
34     sock=socket(PF_INET, SOCK_STREAM, 0);
35
36     memset(&serv_addr, 0, sizeof(serv_addr));
37     serv_addr.sin_family=AF_INET;
38     serv_addr.sin_addr.s_addr=inet_addr(argv[1]);
39     serv_addr.sin_port=htons(atoi(argv[2]));
40
41
42     if(connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr))==-1)
43         error_handling("connect() error");
44
45     pthread_create(&snd_thread, NULL, send_msg, (void*)&sock);
46     pthread_create(&rcv_thread, NULL, rcv_msg, (void*)&sock);
47     pthread_join(snd_thread, &thread_return);
48     pthread_join(rcv_thread, &thread_return);
49     close(sock);
50     return 0;
51 }
```

<메인함수-변수선언>

```
21 int main(int argc, char *argv[])
22 {
23     int sock;
24     struct sockaddr_in serv_addr;
25     pthread_t snd_thread, rcv_thread;
26     void * thread_return;
```

21~26. 소켓 생성, 서버 주소를 담은 구조체 변수, 송, 수신에 이용할 쓰레드 생성, 쓰레드 종료시 리턴 값을 받는 변수를 선언합니다.

<메인함수 - 입력안내>

```
27     if(argc!=4)
28     {
29         printf("Usage : %s <IP> <port> <name>\n", argv[0]);
30         exit(1);
31     }
```

27~31. 사용자가 초기 IP, port, name 입력시 잘못 입력했을 때 정확한 정보를 입력할 수 있도록 안내 문구를 출력시킵니다.

<메인함수-소켓설정>

```
33     sprintf(name, "[%s]", argv[3]);
34     sock=socket(PF_INET, SOCK_STREAM, 0);
35
36     memset(&serv_addr, 0, sizeof(serv_addr));
37     serv_addr.sin_family=AF_INET;
38     serv_addr.sin_addr.s_addr=inet_addr(argv[1]);
39     serv_addr.sin_port=htons(atoi(argv[2]));
```

33~39. 클라이언트 파일 실행시 입력받았던 name(argv[3])를 name 변수에 저장. TCP로 이용하기 위하여 SOCK_STREAM으로 TCP 소켓 생성, 서버 소켓의 정보를 담을 구조체를 0으로 초기화, 서버 주소를 ipv4로 해당하는 주소체계로 채워줌, 초기 파일 실행시 입력받았던 IP주소(argv[1])를 수치화하고 빅엔디언 방식을 이용함, 입력 받았던 port(argv[2])번호를 할당함 htons형의 port번호를 호스트 바이트 순서에서 네트워크 바이트 순서로 변환합니다.

<메인함수-서버접속>

```
42     if(connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr))==-1)
43         error_handling("connect() error");
```

42~43. 서버에 연결시도, sock을 통해 받은 파일디스크립트에 서버주소를 할당하고 해당 서버와 연결시도, 연결이 정상적으로 진행되었다면 0을 반환, 연결되지 않았다면 -1을 반환하고 error_handling함수의 에러구문을 매개변수로 주고 호출한다.

<메인함수-쓰레드실행>

```
45 pthread_create(&snd_thread, NULL, send_msg, (void*)&sock);
46 pthread_create(&rcv_thread, NULL, rcv_msg, (void*)&sock);
47 pthread_join(snd_thread, &thread_return);
48 pthread_join(rcv_thread, &thread_return);
49 close(sock);
50 return 0;
```

45~50. send_msg, rcv_msg 각각 쓰레드를 동작시킵니다. send_thread는 연결된 소켓에 메시지 입력 후 메시지 전송, rcv_thread는 소켓으로부터 메시지 수신등의 동작을 진행합니다. pthread_join을 통해 쓰레드 반환 값이 전달되기 전까지 블로킹 상태에 있게 합니다.

<서버에 메시지를 송신하는 함수 send_msg>

```
53 void * send_msg(void * arg)
54 {
55     int sock=*((int*)arg);
56     char name_msg[NAME_SIZE+BUF_SIZE];
57     while(1)
58     {
59         fgets(msg, BUF_SIZE, stdin);
60         if(!strcmp(msg, "q\n") || !strcmp(msg, "Q\n"))
61         {
62             close(sock);
63             exit(0);
64         }
65         sprintf(name_msg,"%s %s", name, msg);
66         write(sock, name_msg, strlen(name_msg));
67     }
68     return NULL;
69 }
70 }
```

53~70. 쓰레드 실행시 인자로 넘어온 소켓을 받고, 메시지를 담을 배열 name_msg를 선언합니다. while무한 루프를 돌면서 사용자에게서 키보드 입력을 받고, q, Q를 입력 받을 시에는 소켓을 종료합니다. 그외의 메시지를 입력했다면, 초기에 입력했었던 사용자명, 메시지를 name_msg배열에 담고 연결된 서버 소켓에 전송합니다.

<서버에서 메시지를 수신하는 함수 recv_msg>

```
73 void * recv_msg(void * arg)
74 {
75     int sock=*((int*)arg);
76     char name_msg[NAME_SIZE+BUF_SIZE];
77     int str_len;
78     while(1)
79     {
80         str_len=read(sock, name_msg, NAME_SIZE+BUF_SIZE-1);
81         if(str_len==-1)
82             return (void*)-1;
83         name_msg[str_len]=0;
84         fputs(name_msg, stdout);
85     }
86     return NULL;
87 }
```

73~87. 쓰레드 실행시 인자로 넘어온 소켓을 받고, 화면에 출력할 메시지를 담을 변수 name_msg 배열을 선언합니다. while무한 루프를 돌면서 연결된 서버로부터 메시지를 수신하고 해당 수신한 메시지를 name_msg에 담는다. 만약 str_len이 -1일 경우 통신이 끊겼기 때문에 쓰레드를 종료합니다. name_msg[str_len] = 0을 통해 수신받은 메시지 길이 다음 배열에 0을 넣어 메시지의 끝을 설정합니다. 이렇게 하는 이유 fputs의 경우 0을 만날 때까지 출력하기 때문입니다. 이렇게 저장된 name_msg배열을 출력을 진행합니다.

<예외처리를 담당하는 함수 error_handling>

```
90 void error_handling(char *msg)
91 {
92     fputs(msg, stderr);
93     fputc('\n', stderr);
94     exit(1);
95 }
```

90~95. 에러함수 호출시에 전달받은 에러메세지 값을 출력합니다.

<코드 실행 결과물>

서버 실행 결과

```
~/201644056
AzureAD+이 교 범 @DESKTOP-85TEC5I ~/201644056
$ ./tr1 9190
Connected client IP: 127.0.0.1
Connected client IP: 127.0.0.1
Connected client IP: 127.0.0.1
```

클라이언트1 접속 결과

```
~/201644056
AzureAD+이 교 범 @DESKTOP-85TEC5I ~/201644056
$ ./tr2 127.0.0.1 9190 cherry
안녕하세요
[cherry] 안녕하세요
[brian] 안녕하세요
[macCoco] 안녕하세요
```

클라이언트2 접속 결과

```
~/201644056
AzureAD+이 교 범 @DESKTOP-85TEC5I ~/201644056
$ ./tr2 127.0.0.1 9190 brian
[cherry] 안녕하세요
안녕하세요
[brian] 안녕하세요
[macCoco] 안녕하세요
```

클라이언트3 접속 결과

```
~/201644056
AzureAD+이 교 범 @DESKTOP-85TEC5I ~/201644056
$ ./tr2 127.0.0.1 9190 macCoco
[cherry] 안녕하세요
[brian] 안녕하세요
안녕하세요
[macCoco] 안녕하세요
```