TCP/IP 기말고사 대체 과제

202044066 박지원

chat_serv.c

```
#include <stdio.h> // 입출력
#include <stdlib.h> // 문자열 변환 , 난수 생성
#include <unistd.h> // 표준 심볼 상수 및 자료형
#include <string.h> // 문자열 상수
#include <arpa/inet.h> // 주소 변환
#include <sys/socket.h> // 소켓 연결
#include <netinet/in.h> //IPv4 전용 기능
#include <pthread.h> // 쓰레드 사용

#define BUF_SIZE 100
#define MAX_CLNT 256

void * handle_clnt(void * arg); // 클라이언트로부터 받은 메시지 처리
void send_msg(char * msg, int len); // 클라이언트로부터 메시지 받아옴
void error_handling(char * msg); // 에러 처리

int clnt_cnt=0; // 서버에 접속한 클라이언트 수
int clnt_socks[MAX_CLNT]; // 클라이언트와의 송수신 위해 생성한 소켓의 파일 디스크립터 저장한 배열
pthread_mutex_t mutx; // mutex 통한 쓰레드의 동기화를 위한 변수

int main(int argc, char *argv[])
{
        Int serv_sock, clnt_sock; // 서버 소켓, 클라이언트 소켓 선언
        struct sockaddr_in serv_adr, clnt_adr; // 서버 주소 , 클라이언트 주소 구조체 선언
        socklen_t clnt_adr_sz; // 클라이언트 주소 크기 선언
        pthread_t t_id; // 쓰레드 ID 선언
        if(argc != 2) { // 실행파일의 경로 / port 번호 입력 받아야함
                printf("Usage : %s <port>\n", argv[0]); // 출력
                exit(1);
        }

        /* mutex 생성 */
        pthread_mutex_init(&mutx, NULL);
                                        // &mutx 는 뮤텍스의 참조 값 저장을 위한 변수의 주소 값을 전달한다.
                                        // NULL 은 생성하는 뮤텍스의 특성정보를 담고 있는 변수의 주소 값 전달, 별도의 특성 지정하지 않을 경우 NULL 전달
```

```
/* 서버 소켓 (리스닝 소켓) 생성 */
serv_sock=socket(PF_INET, SOCK_STREAM, 0);
                                        // PF_INET 는 IPv4 인터넷 프로토콜 체계 사용
                                        // SOCK_STREAM 는 연결지향형 소켓 ( 소켓의 데이터 전송 방식)
/* 주소 정보 초기화 */
memset(&serv_adr, 0, sizeof(serv_adr));
serv_adr.sin_family=AF_INET;
serv_adr.sin_addr.s_addr=htonl(INADDR_ANY);
serv_adr.sin_port=htons(atoi(argv[1]));

/* 주소 정보 할당 , 실패 시 -1 반환 */
if(bind(serv_sock, (struct sockaddr*) &serv_adr, sizeof(serv_adr))==-1)
        /* bind() 에러 생겼을 때 */
        error_handling("bind() error");

/* 연결 요청 대기 큐 생성 , 실패 시 -1 반환 */
if(listen(serv_sock, 5)==-1)
                        // serv_sock 는 연결 요청 대기상태에 두고자 하는 소켓의 파일 디스크립터
                        // 5는 연결 요청 대기 큐의 크기 정보
                        // 실패 시 –1 반환

        /* listen() 에러 생겼을 때 */
        error_handling("listen() error");

while(1)
{
        clnt_adr_sz=sizeof(clnt_adr);

        /* 연결 요청 수락하기 */
        clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_adr, &clnt_adr_sz);
                                        // serv_sock 은 서버 소켓의 파일 디스크립터
                                        // &clnt_adr 은 연결 요청한 클라이언트의 주소 정보를 담을 변수의 주소 값 전달
                                        // &clnt_adr_sz 은 &clnt_adr에 전달된 주소의 변수 크기를 바이트 단위로 전달

        /* 임계영역 들어가기 전 호출 . 다른 쓰레드가 임계영역 실행 중이면 그 쓰레드가 unlock() 호출할 때까지 블로킹 */
        pthread_mutex_lock(&mutx);

        /* 클라이언트 수와 파일 디스크립터 등록 */
        clnt_socks[clnt_cnt++]=clnt_sock;

        /* 임계영역 나오고 나서 호출 */
        pthread_mutex_unlock(&mutx);
```

```
                /* 쓰레드 생성과 실행 흐름의 구성 */
                pthread_create(&t_id, NULL, handle_clnt, (void*)&clnt_sock);
                                                        // t_id 은 생성할 쓰레드의 ID를 저장할 변수의 포인터 전달
                                                        // NULL 은 생성하고자 하는 쓰레드의 속성 설정, 일반적으로 NULL
                                                        // handle_clnt 은 쓰레드 생성 후 실행할 함수 루틴 설정
                                                        // &clnt_sock 은 쓰레드에 의해 호출되는 함수에 전달한 인자값

                /* 쓰레드 소멸 */
                pthread_detach(t_id);
                                // t_id 은 종료와 동시에 소멸시킬 쓰레드의 ID 정보 전달

                printf("Connected client IP : %s \n", inet_ntoa(clnt_adr.sin_addr));
                                                // inet_ntoa() 는 네트워크 바이트 순서로 정렬된 정수형 IP 주소 정보를 우리가 눈으로 쉽게 인식할 수 있는 문자열의 형태로 변환

        }
        close(serv_sock); // 서버 소켓 종료

        return 0;
}

void * handle_clnt(void * arg) // 수신된 메시지를 모든 클라이언트에게 전송하는 함수
{
        int clnt_sock=*((int*)arg); // 클라이언트 소켓 할당

        int str_len = 0, i; // 문자열 길이 선언

        char msg[BUF_SIZE];

        while((str_len=read(clnt_sock, msg, sizeof(msg)))!=0) // 클라이언트로부터 EOF 를 수신할 때까지 msg 읽음

                send_msg(msg, str_len); // msg 읽는 함수 실행

        /* 임계영역 들어가기 전 호출 */
        pthread_mutex_lock(&mutx);

        for(i=0; i<clnt_cnt; i++)
        {
                if(clnt_sock==clnt_socks[i]) // 현재 해당하는 파일 디스크립터 찾을 시
                {
                        while(i++<clnt_cnt-1) // 클라이언트가 연결 요청했으므로 해당 정보 덮여
```

씌여 삭제

```
                                        clnt_socks[i]=clnt_socks[i+1];
                                break;
                        }
        }
        clnt_cnt--; // 클라이언트 수 감소

        /* 임계영역 나오고 나서 호출 */
        pthread_mutex_unlock(&mutx);

        close(clnt_sock); // 클라이언트 소켓 종료

        return NULL;
}

void send_msg(char * msg, int len) // send to all
{
        int i;

        /* 임계영역 들어가기 전 호출 */
        pthread_mutex_lock(&mutx);

        for(i=0; i<clnt_cnt; i++) // 현재 연결된 모든 클라이언트에게 메시지 전송

                write(clnt_socks[i], msg, len); // 데이터 송신
                        // clnt_socks[i] 는 데이터 전송대상을 나타내는 파일 디스크립터 전달
                        // msg 는 전송할 데이터가 저장된 버퍼의 주소값 전달
                        // len 은 전송할 데이터의 바이트 수 전달

        /* 임계영역 나오고 나서 호출 */
        pthread_mutex_unlock(&mutx);
}

void error_handling(char * msg)
{
        fputs(msg, stderr); // 에러 메시지(문자열) 출력
        fputc('\n', stderr); // 문자 출력
        exit(1);
}
```

chat_client.c

```
#include <stdio.h> // 입출력
#include <stdlib.h> // 문자열 변환 , 난수 생성
#include <unistd.h> // 표준 심볼 상수 및 자료형
#include <string.h> // 문자열 상수
#include <arpa/inet.h>
#include <sys/socket.h> // 소켓 연결
#include <pthread.h> // 쓰레드 사용

#define BUF_SIZE 100
#define NAME_SIZE 20

void * send_msg(void *arg); // 메시지 보냄
void * recv_msg(void * arg); // 메시지 받음
void error_handling(char * msg); // 에러 처리

char name[NAME_SIZE]="[DEFAULT]"; // 이름 선언 및 초기화
char msg[BUF_SIZE];

int main(int argc, char *argv[])
{
        int sock; // 소켓 선언

        struct sockaddr_in serv_addr; // 소켓 주소 구조체 선언

        pthread_t snd_thread, rcv_thread; // 보내고, 받는 쓰레드 선언

        void * thread_return;
        if(argc!=4) { // 실행파일 경로 / port 번호 / name 입력받아야 함
                printf("Usage : %s <IP> <port> <name>\n", argv[0]);
                exit(1);
        }

        sprintf(name, "[%s]", argv[3]); // 이름 입력 받음

        /* 서버 소켓 (리스닝 소켓) 생성 */
        sock=socket(PF_INET, SOCK_STREAM, 0);

        /* 주소 정보 초기화 */
        memset(&serv_addr, 0, sizeof(serv_addr));
        serv_addr.sin_family=AF_INET;
        serv_addr.sin_addr.s_addr=inet_addr(argv[1]);
        serv_addr.sin_port=htons(atoi(argv[2]));

        /* 서버 주소 정보를 기반으로 연결 요청 , 실패 시 -1 반환 */
        if(connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr))==-1)
                                                        // sock 은 클라이언트 소켓의 파일
```

디스크립터

// &serv_addr 은 연결 요청할 서버의 주소 정보를 담은 변수의 주소 값

// sizeof(serv_addr) 은 serv_addr에 전달된 주소의 변수 크기를 바이트 단위로 전달

```
		/* connect() 에러 생겼을 때 */
		error_handling("connect() error");

	/* 쓰레드 생성과 실행 흐름의 구성 */
	pthread_create(&snd_thread, NULL, send_msg, (void*)&sock);
	pthread_create(&rcv_thread, NULL, recv_msg, (void*)&sock);

	/* 실행 시간 예측하지 않고 해결 , 쓰레드 종료까지 대기 */
	pthread_join(snd_thread, &thread_return);
```

// snd_thread 은 인자로 들어오는 ID의 쓰레드가 종료할 때까지 실행 지연 시킴

// &thread_return 은 쓰레드 종료 시 반환하는 값에 접근할 수 있는 포인터

```
	pthread_join(rcv_thread, &thread_return);

	close(sock); // 클라이언트 소켓 연결 종료

	return 0;
}

void * send_msg(void * arg)
{
	int sock=*((int*)arg); // 클라이언트의 파일 디스크립터
	char name_msg[NAME_SIZE+BUF_SIZE];
	while(1)
	{
		fgets(msg, BUF_SIZE, stdin); // 문자열 입력
		if(!strcmp(msg, "q\n")||!strcmp(msg, "Q\n")) // 'q' 또는 'Q' 입력 시 종료
		{
			close(sock); // 클라이언트 소켓 연결 종료 후

			exit(0); // 프로그램 종료
		}
		sprintf(name_msg, "%s %s", name , msg); // 클라이언트 이름과 메시지 합침

		write(sock, name_msg, strlen(name_msg)); // 널문자 제외하고 서버로 문자열 송신
	}
	return NULL;
}

void * recv_msg(void * arg)
```

```
{
	int sock=*((int*)arg);   // 클라이언트의 파일 디스크립터

	char name_msg[NAME_SIZE+BUF_SIZE];
	int str_len;
	while(1)
	{
		str_len=read(sock, name_msg, NAME_SIZE+BUF_SIZE-1); // 데이터 수신
			// sock 은  데이터 수신 대상을 나타내는 파일 디스크립터 전달
			// name_msg 는 수신 데이터가 저장된 버퍼의 주소값 전달
			// NAME_SIZE+BUF_SIZE-1 은 수신 데이터의 바이트 수 전달

		if(str_len==-1) // read() 실패 시 -1 반환
			return (void*)-1;
		name_msg[str_len]=0;
		fputs(name_msg, stdout); // 문자열 출력
	}
	return NULL;
}

void error_handling(char * msg)
{
	fputs(msg, stderr); // 에러 메시지(문자열) 출력
	fputc('\n', stderr); // 문자 출력
	exit(1);
}
```