Network Programming Assignment #12

2017116186
임정민

Program Objective

리눅스의 epoll 기능을 이용한 다수의 클라이언트가 참여하는 채팅 프로그램 구현

Server

- 클라이언트로부터 메세지를 입력 받아 채팅에 참여중인 다른 클라이언트들에게 전송
- 전송 받은 메세지를 출력하여 전송받은 메세지와 전송된 메세지가 같은지 확인
- 전송 받은 메세지에 메세지를 전송한 클라이언트 번호를 추가하여 누가 보낸 메세지인지 확인

Client

- 다른 클라이언트들로부터 메세지 수신
- 다른 클라이언트들에게 메세지 전송

Motivation

epoll 함수를 이해하고 사용해보기 위해 epoll 함수의 file descriptor 관리를 적절히 사용해 볼 수 있는 다수의 클라이언트가 참여할 수 있는 채팅 프로그램 선정

Solution Code

Server

```
#include <stdio.h>
…
void setnonblockingmode(int fd);
void error_handling(char *buf);

int main(int argc, char *argv[]){
        …
        int str_len, count = 0, i, j;
        // 메세지를 전송한 클라이언트 이외의 다른 클라리언트의 file descriptor 확인을 위한 count
        …

        while(1){
        // CTRL+C를 통해 종료될 때까지 계속 실행을 위한 while문
                event_cnt=epoll_wait(epfd, ep_events, EPOLL_SIZE, -1);
                // 이벤트가 발생된 갯수 event_cnt
                if(event_cnt==-1){
                        puts("epoll_wait() error");
                        break;
                }

                puts("return epoll_wait");

                for(i=0; i<event_cnt; i++){
                // even가 발생된 곳을 모두 확인
                        if(ep_events[i].data.fd==serv_sock){
                        // 새로운 클라이언트 연결
```

```
                adr_sz=sizeof(clnt_adr);
                …
                epoll_ctl(epfd, EPOLL_CTL_ADD, clnt_sock, &event);

                printf("connected client: %d \n", clnt_sock);
                count++;
        }

        else{
                client_fd = ep_events[i].data.fd;
                // 메세지를 전송한 클라이언트 정보 저장
                str_len=read(ep_events[i].data.fd, buf, BUF_SIZE);

                if(str_len == 0){
                // 클라이언트 연결 종료
                        epoll_ctl(epfd, EPOLL_CTL_DEL, ep_events[i].data.fd,
NULL);
                        close(ep_events[i].data.fd);
                        printf("closed client: %d \n", ep_events[i].data.fd);
                        count--;
                        break;
                }
                else{
                // 입력받은 메세지 처리 후 연결된 클라이언트에 전송
                        message[0] = '\0';
                        buf[str_len] = '\0';
                        sprintf(message, "client %d: %s", client_fd, buf);
                        buf[0] = '\0';
                        strcat(buf, message);
                        str_len = strlen(buf);

                        for(j = 5; j < 5+count; ++j){
                        if(j != client_fd) write(j, buf, str_len);
```

```
                                        }
                                    }
                                }
                            }
                        }
        close(serv_sock);
        close(epfd);
        return 0;
}

void setnonblockingmode(int fd){
        int flag=fcntl(fd, F_GETFL, 0);
        fcntl(fd, F_SETFL, flag|O_NONBLOCK);
}

void error_handling(char *message){
        fputs(message, stderr);
        fputc('\n', stderr);
        exit(1);
}
```

Client

```
// 11주차의 echo_mpclient.c와 동일
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>

#define BUF_SIZE 100
```

```
#define NAME_SIZE 20

void error_handling(char *message);
void read_routine(int sock, char *buf);
// 메세지 수신을 위한 쓰레드 함수
void write_routine(int sock, char *buf);
 // 메세지 전송을 위한 쓰레드 함수

char buf[BUF_SIZE];

int main(int argc, char *argv[]) {
    int sock;
    pid_t pid;
    char buf[BUF_SIZE];
    struct sockaddr_in serv_adr;

    if(argc!=3) {
        printf("Usage : %s <IP> <port> \n", argv[0]);
        exit(1);
    }

    sock=socket(PF_INET, SOCK_STREAM, 0);
    memset(&serv_adr, 0, sizeof(serv_adr));
    serv_adr.sin_family=AF_INET;
    serv_adr.sin_addr.s_addr=inet_addr(argv[1]);
    serv_adr.sin_port=htons(atoi(argv[2]));

    if(connect(sock, (struct sockaddr*)&serv_adr, sizeof(serv_adr))==-1)
        error_handling("connect() error!");

    pid=fork();
    if(pid==0)
        write_routine(sock, buf);
```

```
    else
        read_routine(sock, buf);

    close(sock);
    return 0;
}

void read_routine(int sock, char *buf) {
    char message[BUF_SIZE];
    while(1) {
        int str_len=read(sock, message, BUF_SIZE);

        if(str_len==0)
            return;

        message[str_len]=0;
        fputs(message, stdout);
    }
}

void write_routine(int sock, char *buf) {
    char message[BUF_SIZE];
    while(1) {
        fgets(buf, BUF_SIZE, stdin);

        if(!strcmp(buf,"q\n") || !strcmp(buf,"Q\n")) {
            shutdown(sock, SHUT_WR);
            return;
        }
                else if(!strcmp(buf, "\n")){
                        continue;
                }
```

```
        sprintf(message, "%s", buf);

        write(sock, message, strlen(message));
    }
}

void error_handling(char *message) {
    fputs(message, stderr);
    fputc('\n', stderr);
    exit(1);
}
```

Result