Programming Final Project

김형석 2019227626

Contents

1. Server.c
    1. Socket
    2. PThread
2. Client.c
    1. Socket
    2. PThread
3. Send msg handler
4. Add-ons

Socket - server.c

- Client Structure
  - 다수의 클라이언트를 관리하기 위해 각 클라이언트가 생성될 시 고유의 이름, id, sockfd 및 주소를 할당받는다.
- 전역 변수로 클라이언트를 지정하고 메시지를 주고 받는 쓰레드를 관리하기 위해 pthread_mutex를 초기화한다.
- 서버는 총 100명의 클라이언트를 수용할 수 있도록 하였다.

```
19  /* Client structure */
20  typedef struct{
21      struct sockaddr_in address;
22      int sockfd;
23      int uid;
24      char name[32];
25  } client_t;
26
27  client_t *clients[MAX_CLIENTS];
28
29  pthread_mutex_t clients_mutex = PTHREAD_MUTEX_INITIALIZER;
```

Socket - server.c

- Socket settings
  - server_sockfd = socket(AF_INET, SOCK_STREAM, 0)
  - server_address.sin_family = AF_INET
  - server_address.sin_addr.s_addr = inet_addr(ip) //ip=127.0.0.1
  - server_address.sin_port = htons(port) //port = 9000
  - setsockopt(server_sockfd, SOL_SOCKET, (SO_REUSEPORT | SO_REUSEADDR), (char*)&option, sizeof(option))

Socket - server.c

- Igonre pipe signals
  - signal(SIGPIPE, SIG_IGN);
- Bind
  - bind(server_sockfd, (struct sockaddr*)&server_address, sizeof(server_address))
- Listen
  - listen(server_sockfd, 10)

Socket - server.c

- 위의 Server Socket setting 과정이 에러 없이 진행된다면 while 반복문을 통해 accept()함수를 이용하여 client가 연결될 때까지 기다린다.

```
while(1){
    socklen_t clilent_len = sizeof(client_address);
    printf("Number of Clients: %d\n",cli_count);
    /*wait until client connect to server*/
    client_sockfd = accept(server_sockfd, (struct sockaddr*)&client_address, &clilent_len);
```

- client_sockfd = accept(server_sockfd, (struct sockaddr*)&client_address, & client_len);

```
kim@iasl-pc:~/programming/term_prj$ ./server 9000
===============Chatting Room's Server=================
Number of Clients: 0
```

Socket - server.c

- Accept함수를 통해 client와 연결이 되었다면 임시 cli변수를 malloc함수로 할당하고 해당 클라이언트의 주소, sockfd 및 id값을 복사한다.

- 복사한 클라이언트를 서버 생성시 초기화 했던 clients값에 지정하여 queue에 더하고 client.c 프로그램에서 생성한 쓰레드에 포크 한다. (1 thread per 1 client)

```c
/* Check Max*/
if((cli_count + 1) == MAX_CLIENTS){
    printf("ERROR: Max clients reached.");
    close(client_sockfd);
    continue;
}

/* Client settings */
client_t *cli = (client_t *)malloc(sizeof(client_t));
cli->address = client_address;
cli->sockfd = client_sockfd;
cli->uid = uid++;

/* Add client to the queue and fork thread */
queue_add(cli);
pthread_create(&tid, NULL, &handle_client, (void*)cli);
```

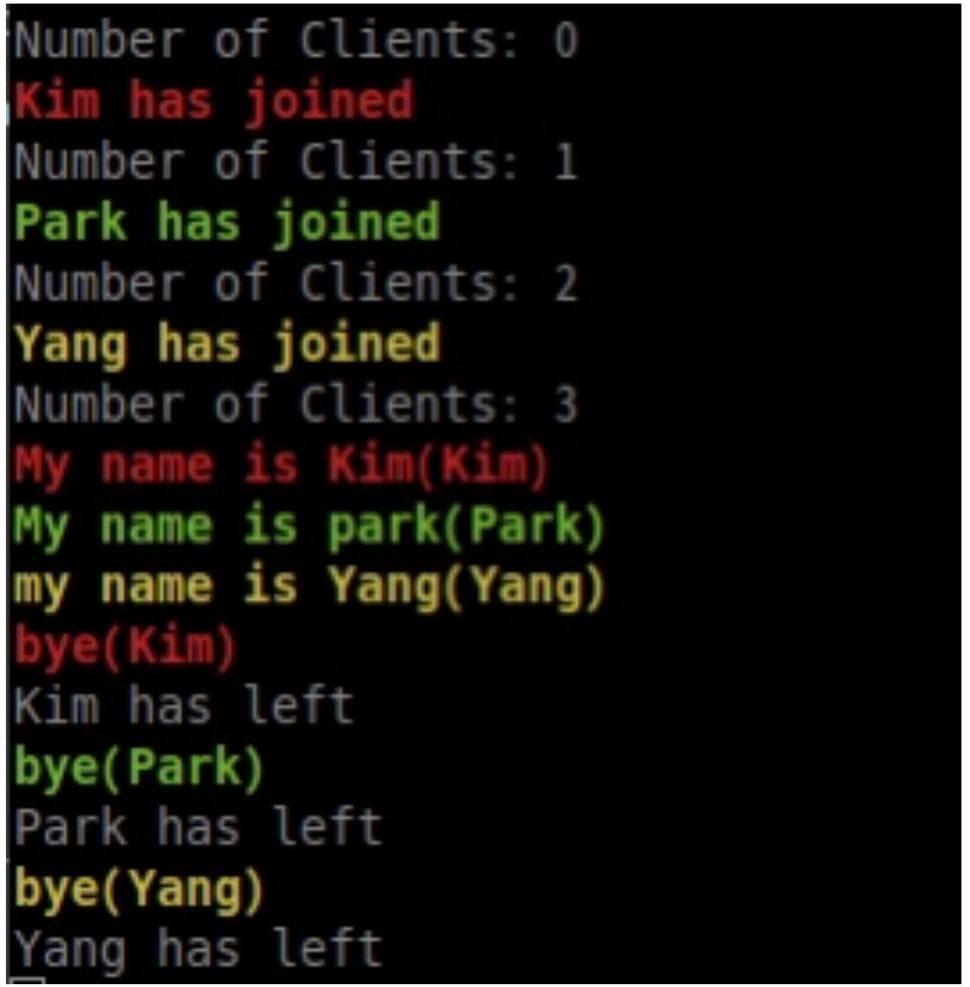
Pthread - server.c

- queue
  - 서버 생성시 만든 clients[100] 데이터를 큐 데이터 형식으로 관리한다.
  - 해당 q데이터는 메세지 수신 및 전송하는데 클라이언트의 id와 이름를 식별하는데 사용한다.
- queue_add(client_t *cli)
- queue_remove(int uid)
- pthread_mutex_lock, unlock함수를 통해 각 함수 실행시 데이터 교란 없이 안전하게 실행할 수 있도록 한다.

```c
/* Add clients to queue */
void queue_add(client_t *cl){
    pthread_mutex_lock(&clients_mutex);
    for(int i=0; i < MAX_CLIENTS; ++i){
        if(!clients[i]){
            clients[i] = cl;
            break;
        }
    }
    pthread_mutex_unlock(&clients_mutex);
}
/* Remove clients to queue */
void queue_remove(int uid){
    pthread_mutex_lock(&clients_mutex);
    for(int i=0; i < MAX_CLIENTS; ++i){
        if(clients[i]){
            if(clients[i]->uid == uid){
                clients[i] = NULL;
                break;
            }
        }
    }
    pthread_mutex_unlock(&clients_mutex);
}
```

Pthread - server.c

- pthread_create(&tid, NULL, &**handle_client**, (void*) cli)
- Void *handle_client(void *arg)
- Client.c 단에서 생성한 스레드를 포크한다. handle_client의 주요 기능은 다음과 같다.
  - Client 이름 중복 방지
  - Client 시작 및 종료시 쓰레드 및 큐데이터 관리
  - 귓속말 기능 flag 확인
- Pthread_mutex_lock, unlock함수를 통해서 데이터 교란 없이 안전하게 수행할 수 있도록하였다.



Socket - client.c

- Socket settings
  - server_sockfd = socket(AF_INET, SOCK_STREAM, 0)
  - server_address.sin_family = AF_INET
  - server_address.sin_addr.s_addr = inet_addr(ip) //ip=127.0.0.1
  - server_address.sin_port = htons(port) //port = 9000
  - **connect(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr))**

Socket - client.c

- server의 ip와 port에 맞게 설정하고 해당 서버와 연결하는 connect() 함수가 에러없이 진행 되었다면 메세지를 수신 및 전송하는 쓰레드를 생성한다.

- pthread_create(&send_msg_thread, NULL, (void*) **send_msg_handler**, NULL)

- pthread_create(&recv_msg_thread, NULL, (void*) **recv_msg_handler**, NULL)

```
pthread_t send_msg_thread;
if(pthread_create(&send_msg_thread, NULL, (void *) send_msg_handler, NULL) != 0){
    printf("ERROR: pthread\n");
    return EXIT_FAILURE;
}

pthread_t recv_msg_thread;
if(pthread_create(&recv_msg_thread, NULL, (void *) recv_msg_handler, NULL) != 0){
    printf("ERROR: pthread\n");
    return EXIT_FAILURE;
}
```

Pthread - client.c

- Void send_msg_handler(){}
  - fgets()함수로 메세지를 저장하고 메세지를 보내는 client의 name과 함께 send()함수를 통해 server로 전송한다.

```c
void send_msg_handler() {
    char message[LENGTH] = {};
    char buffer[LENGTH + 32] = {};

    while(1) {
        str_overwrite_stdout();
        fgets(message, LENGTH, stdin);
        str_trim_lf(message, LENGTH);

        if (strcmp(message, "exit") == 0) {
            break;
        } else {
            sprintf(buffer, "%s(%s)\n", message, name);
            send(sockfd, buffer, strlen(buffer), 0);
        }
        bzero(message, LENGTH);
        bzero(buffer, LENGTH + 32);
    }
    catch_ctrl_c_and_exit(2);
}
```

Pthread - client.c

- Void recv_msg_handler(){}
  - while문 안에 recv()함수를 통해 server로 부터 전송되는 메시지를 계속 확인하고 메시지를 출력하는 기능을 한다.

```c
void recv_msg_handler() {
    char message[LENGTH] = {};
    while (1) {
        int receive = recv(sockfd, message, LENGTH, 0);
        if (receive > 0) {
            if(strcmp(message, "__exit__") == 0){
                printf("Existed name, try use another name\n");
                catch_ctrl_c_and_exit(2);
                break;
            }
            int len = strlen(message);
            color_switch(message[len-2]-'0');
            message[len-2]='\n';
            message[len-1]='\0';
            printf("%s", message);
            reset();
            str_overwrite_stdout();
        } else if (receive == 0) {
            break;
        }
        memset(message, 0, sizeof(message));
    }
}
```

Send message (self, all, whisper)

- Server - handle_client
  - Client 쓰레드가 메시지를 서버로 전송하면 server의 handle_client는 recv()함수로 부터 받은 메시지가 귓속말이지 먼저 확인하고 아니라면 send_message()함수를 통해 해당 메시지를 생성된 클라이언트 모두에게 전송한다.
  - send_message_self() 함수는 에러메시지를 해당 클라이언트에게만 전송할 수 있도록 따로 구현하였다.

```
/send_message except self
void send_message(char *s, int uid){
  char tmp;
  int len;
  pthread_mutex_lock(&clients_mutex);
  tmp = (uid%10)+'0';
  len = strlen(s);
  s[len-1] = tmp;
  strcat(s,"\n");
  for(int i=0; i<MAX_CLIENTS; ++i){
      if(clients[i]){
          if(clients[i]->uid != uid){
              if(write(clients[i]->sockfd, s, len+1) < 0){
                  perror("ERROR: write to descriptor failed");
                  break;
              }
          }
      }
  }
  pthread_mutex_unlock(&clients_mutex);


/send_message_self
```

Add-ons 1. Catch_ctrl_c_and exit()

- 전역 변수로 지정한 volatile sig_atomic_t flag 변수를 1로 지정하는 함수.
- 사용자가 ctrl_c 누르거나 또는 exit를 메세지 내용으로 전송하면 프로그램 종료.
- 이미 사용하고 있는 이름으로 채팅방이 들어오면 같은 이름이 있다는 걸 메시지로 전송하고 프로그램 종료.

```
volatile sig_atomic_t flag = 0;
int sockfd = 0;
char name[32];

void catch_ctrl_c_and_exit(int sig) {
    flag = 1;
}
```

Add-ons 2. Color_switch(int i)

- 각 Client는 자신만의 고유한 id를 가진다 이를 이용하여 메세지 전송이 각각 다른 텍스트 color를 가질 수 있도록 프로그램 하였다.


```
void color_switch (int i) {
    switch(i){
        case 0: printf("\033[1;31m");break;
        case 1: printf("\033[1;32m");break;
        case 2: printf("\033[1;33m");break;
        case 3: printf("\033[1;34m");break;
        case 4: printf("\033[1;35m");break;
        case 5: printf("\033[1;36m");break;
        case 6: printf("\033[7;31m");break;
        case 7: printf("\033[7;32m");break;
        case 8: printf("\033[7;33m");break;
        case 9: printf("\033[7;34m");break;
        default : reset();
    }
}
```

Add-ons 3. Send_whisper_message(char *s, int uid)

- 귓속말 기능
  - /w “name” msg 형태로 메세지를 보내면 서버에서 /w를 인식하고 귓속말 함수 send_whisper_message를 실행한다. 문자“ ”을 기점으로 for문을 사용해서 이름을 추출하고 해당 이름을 가지고 있는 클라이언트에게만 메세지를 전송한다. 만약 이름이 없거나 귓속말 포맷을 재대로 사용하지 않으면 다음과 같은 에러를 출력한다.

시 연

1. 1thread per 1 client

2. 클라이언트 색깔 구분

3. exit or ctrl-c signal catch

4. 이름 중복 방지

5. 귓속말 기능

결 론

- 쓰레드를 이용하여 채팅 서버와 클라이언트들을 생성하고 각 클라이언트는 서버를 통해 메시지를 주고 받을 수 있다.
- 각 클라이언트는 고유의 아이디 와 이름을 가지고 있으며 이를 이용하여 클라이언트를 색깔로 구별할 수 있게 했으며 귓속말 기능을 추가하였다.
- signal 함수를 사용하여 프로그램을 종료 할 수 있게 하였다.