SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

삼성 청년 SW 아카데미

네트워크 프로그래밍

<알림>

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

4장. 멀티쓰레드 서버 구현

챕터의 포인트

- 멀티쓰레드
- 채팅 서버 분석
- 채팅 클라이언트 분석

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

멀티쓰레드

멀티쓰레드

- **쓰레드**
  - 하나의 프로세스 안에서 나뉘어 지는 실행 단위

- **쓰레드의 장점**
  - 빠른 속도
  - 멀티 프로세스보다 쉬운 구현

- **따라서, 멀티 쓰레드 기반의 채팅 프로그램을 구현할 것이다.**

멀티쓰레드의 규칙

1. 쓰레드만의 main 함수를 별도로 정의하고,
2. pthread_create 함수를 사용해 쓰레드 생성을 운영체제에 요청해준다.
3. phread_join 함수를 사용해 쓰레드의 실행 흐름을 조절한다.
4. gcc 로 컴파일 시엔, 쓰레드 컴파일 시 필요한 -lpthread 옵션을 추가한다.
5. 전역변수 사용 시 mutex 사용 (critical section)

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

채팅 서버 분석

서버 코드 분석

- 멀티 쓰레드를 활용한 서버의 코드를 분석해보자.

- 채팅 프로그램은 시스템 프로그래밍 시간에 배웠던 “멀티쓰레딩 " 을 연습하기에 아주 좋은 예제이다.

필요한 전역변수 지정

- 최대 몇 명까지 서비스할것인지 정함 = 500명
- 여러 명의 사용자에게 동시 서비스해야하기 때문에, 다음을 배열로 사용
  - client_sock, client_addr
  - tid - 쓰레드 아이디
- 쓰레드 종료가 되었는지 확실히 점검하는 플래그도 배열
- 뮤텍스 선언

```
#define MAX_CLIENT_CNT 500

const char *PORT = "12345";
int server_sock;

int client_sock[MAX_CLIENT_CNT];
struct sockaddr_in client_addr[MAX_CLIENT_CNT];

pthread_t tid[MAX_CLIENT_CNT];
int exitFlag[MAX_CLIENT_CNT];

// mutex 선언
pthread_mutex_t mutx;
```

멀티 채팅 서버 분석

interrupt

- **Ctrl + C 누르면 작동됨**
- **단, 작동중인 모든 쓰레드 및 소켓 종료**
  - pthread_cancel: 쓰레드 종료요청
  - pthread_join : 쓰레드 종료대기
- **이후 서버 소켓 종료**

```
void interrupt(int arg)
{
    printf("\nYou typed Ctrl + C\n");
    printf("Bye\n");

    for (int i = 0; i < MAX_CLIENT_CNT; i++)
    {
        if (client_sock[i] != 0)
        {
            pthread_cancel(tid[i]);
            pthread_join(tid[i], 0);
            close(client_sock[i]);
        }
    }
    close(server_sock);
    exit(1);
}
```

getClientID

- 접속한 클라이언트가 어느 소켓에 들어가야할지 적절한 위치 반환
- 총 500 개의 배열 중, 0 인 곳을 발견하면 그 곳으로 배정
- 만약 -1 이면? 500명의 사용자가 다 찼다는 뜻

```
int getClientID()
{
    for (int i = 0; i < MAX_CLIENT_CNT; i++)
    {
        if (client_sock[i] == 0)
            return i;
    }
    return -1;
}
```

이후, 다음의 과정을 거친다.

- socket()
- bind()
- listen()

전체 코드의 흐름은 다음을 제외하고 기존 echo 와 동일하다.

- mutex 초기화

```
// mutex init
pthread_mutex_init(&mutx, NULL);
```

id_table

- 이 배열의 역할은
  500 개의 클라이언트가 어떤 아이디에 배정되었는지 알기 위한 용도

```
// pthread argument bug fix
int id_table[MAX_CLIENT_CNT];
printf("Wait for next client...\n");
```

무한루프

- getClientID - client_sock 배열을 순회하면서, 빈 자리를 찾아 리턴
- 이 정보를 id_table 배열에 갱신한다.
- 만약 -1 반환이라면 500명의 사용자가 꽉 찼다는 뜻

- 새로운 사용자를 위해 주소 구조체 초기화

```
while (1)
{
    // get Client ID
    int id = getClientID();
    id_table[id] = id;

    if (id == -1)
    {
        printf("WARNING :: Client FULL\n");
        sleep(1);
    }

    // 새로운 클라이언트를 위해 초기화
    memset(&client_addr[id], 0, sizeof(struct sockaddr_in));
```

accept

- client_sock 은 배열임에 유의
- 나머지 사항은 기존 accept 와 같다.

```
// accpet
client_sock[id] = accept(server_sock, (struct sockaddr *)&client_addr[id], &client_addr_len);
if (client_sock[id] == -1)
{
    printf("ERROR :: 4_accept Error\n");
    break;
}
```

쓰레드 생성

- **accept 된 클라이언트를 위해 쓰레드 생성**

&tid[id]	-	쓰레드 아이디
NULL	-	기본 쓰레드
client_handler	-	쓰레드 함수
(void *)&id_table[id]	-	쓰레드 파라미터

- **왜 파라미터로 그냥 id 를 넘기면 안되는걸까?**
  - 시스템프로그래밍 시간에 다뤘듯이, 쓰레드가 싹 다 생성되고나서 파라미터가 들어가기때문
  - 그래서, 원하는 id 값이 안들어가므로 배열로 확실히 구분된 id 값을 넘겨주는것
- **쓰레드 생성 이후, 무한루프의 가장 위쪽으로 간 이후, 다음 사용자 accept 준비**
- **쓰레드가 핵심이기 때문에, 잠시 후 깊게 알아보자.**

```
pthread_create(&tid[id], NULL, client_handler, (void *)&id_table[id]);
```

쓰레드 종료

- 하나의 쓰레드가 생길 때마다, exitFlag 배열을 전체 반복
- 만약 종료해야 할 쓰레드가 있다면,
  - 쓰레드 종료
  - 클라이언트 소켓 종료

```
// check ExitFlag
for (int i = 0; i < MAX_CLIENT_CNT; i++)
{
    if (exitFlag[i] == 1)
    {
        exitFlag[i] = 0;
        pthread_join(tid[i], 0);
        client_sock[i] = 0;
    }
}
```

client_handler

- 주인공인 쓰레드 함수를 분석해보자
- 파라미터를 int 형으로 변환 : 배정된 id 받아옴
- name 사용자 id 를 출력하기 위한 용도
- inet_ntoa 빅 엔디안 long int IP 를 문자열로 바꿈
  사용자의 IP 를 출력하기 위한 용도

```
void *client_handler(void *arg)
{
    int id = *(int *)arg;

    char name[100];
    // inet_ntoa 는 ,빅 엔디안 long int ip 를 문자열로 바꿈
    strcpy(name, inet_ntoa(client_addr[id].sin_addr));
    printf("INFO :: Connect new Client (ID : %d, IP : %s)\n", id, name);
```

read/write

- 만약 클라이언트가 종료된다면
- exitFlag[id] 를 올려서
- 안전한 종료 절차를 거치도록

- 접속한 클라이언트 모두에게 메세지 보내되,
- Critical Section 인 client_sock 배열은
- 변경되지 않은 상태여야하므로, mutex 잠금

- 이후 client_sock 배열의 참조가 끝나면
- mutex 해제

- 클라이언트와 연결이 끊기면,
- 클라이언트 소켓을 닫아줌

```
// wait & write
char buf[100];
while (1)
{
    memset(buf, 0, 100);
    int len = read(client_sock[id], buf, 99);
    if (len == 0)
    {
        printf("INFO :: Disconnect with client.. BYE\n");
        exitFlag[id] = 1;
        break;
    }

    if (!strcmp("exit", buf))
    {
        printf("INFO :: Client want close.. BYE\n");
        exitFlag[id] = 1;
        break;
    }

    // remove '\n'
    removeEnterChar(buf);

    // send new message
    // mutex
    pthread_mutex_lock(&mutx);
    for (int i = 0; i < MAX_CLIENT_CNT; i++)
    {
        if (client_sock[i] != 0)
        {
            write(client_sock[i], buf, strlen(buf));
        }
    }
    pthread_mutex_unlock(&mutx);
}
close(client_sock[id]);
```

서버 코드 빌드 및 실행 방법

1. $ gcc multi_echo_serv.c -o multi_echo_server -lpthread
2. $ ./multi_echo_server
3. 서버가 켜진 상태로 클라이언트 터미널을 "여러 개" **실행한다.**
   - $ nc 127.0.0.1 12345
   - $ nc 127.0.0.1 12345
   - $ nc 127.0.0.1 12345
   - 하나의 터미널에서 데이터 입력 시, 다른 모든 터미널에도 출력되어야한다.

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

채팅 클라이언트 분석

필요한 전역변수 설정

- 총 두 개의 쓰레드 필요
  - 수신 쓰레드, 송신 쓰레드
- exitFlag
  - 수신 쓰레드, 송신 쓰레드 중
  - 한 쪽이 종료되면 다른 쓰레드 종료하기 위한 플래그
- name, msg 문자열 선언
  - name 은 채팅에 사용될 닉네임

```
#define NAME_SIZE 20
#define MSG_SIZE 100

// AWS IP
const char *IP = "127.0.0.1";
const char *PORT = "12345";

pthread_t send_tid;
pthread_t receive_tid;
int exitFlag;
int sock;

// 채팅창에 보여질 이름의 형태
char name[NAME_SIZE] = "[DEFAULT]";
char msg[MSG_SIZE];
```

멀티 채팅 클라이언트 분석

interrupt

- Ctrl + C 입력 시 인터럽트
- pthread_cancel 이후 pthread_join 사용

```
void interrupt(int arg)
{
    printf("\nYou typped Ctrl + C\n");
    printf("Bye\n");

    pthread_cancel(send_tid);
    pthread_cancel(receive_tid);
    pthread_join(send_tid, 0);
    pthread_join(receive_tid, 0);

    close(sock);
    exit(1);
}
```

커멘드 아규먼트

./chat_client ssafy

- ssafy 라는 닉네임으로 채팅 시작

- 잘못 사용했을 경우, 프로그램 사용법 안내

- 받아온 이름을 sprintf 를 사용해 name 전역변수에 저장

```
// 커멘드 아규먼트 필요로 함 : 사용할 닉네임
int main(int argc, char *argv[])
{
    signal(SIGINT, interrupt);

  // 잘못 사용했을 경우, 프로그램 사용법 안내
    if (argc != 2)
    {
        printf("Usage : %s <name>\n", argv[0]);
        exit(1);
    }

    // sprintf 사용해서, [닉네임] 형태로 name 저장
    sprintf(name, "[%s]", argv[1]);
```

멀티 채팅 클라이언트 분석

pthread_create

- 송신 쓰레드, 수신 쓰레드 각각 선언

- 클라이언트에서 쓰레드가 왜 필요할까?
  - 쓰레드로 구분하지 않으면, 다른 사람이 보낸 채팅을 받을 방법이 없다!

```
pthread_create(&send_tid, NULL, sendMsg, NULL);
pthread_create(&receive_tid, NULL, receiveMsg, NULL);

pthread_join(send_tid, 0);
pthread_join(receive_tid, 0);
```

sendMsg

- 송신 쓰레드
- 닉네임과 메세지를 합쳐서, 서버로 보낼 예정
- 무한루프인데, exitFlag 를 종료조건으로 설정
  - sendMsg 와, receiveMsg 두 개의 쓰레드로 나뉨
  - exitFlag 는 전역변수이므로, 둘 중 어느 한 쪽의 쓰레드가 끝나면
  - 다른 한 쪽의 쓰레드도 종료하기 위함

```
void *sendMsg()
{
    char buf[NAME_SIZE + MSG_SIZE];

    // exitFlag 는, 다른 한 쪽의 쓰레드를 종료하는 용도
    while (!exitFlag)
    {
        memset(buf, 0, NAME_SIZE + MSG_SIZE);
```

멀티 채팅 클라이언트 분석

write

- scanf 대신 fgets 사용
  - scanf 는 띄어쓰기 인식 못함
  - 채팅 프로그램이므로, 띄어쓰기까지 받아야
  - 대신, 엔터도 입력 들어오므로
    strcmp 사용 시 주의할 것
- exitFlag
  - exitFlag 올리면 receiveMsg 쓰레드도 종료됨
  - if (exitFlag) 사용해, receiveMsg 쓰레드에서
    종료가 먼저 일어났으면, sendMsg 쓰레드 종료
- sprint 사용해 서버로 보낼 형태 만들어줌
  - [ssafy] 안녕 얘들아

```
fgets(msg, MSG_SIZE, stdin);
if (!strcmp(msg, "exit\n"))
{
    exitFlag = 1;
    write(sock, msg, strlen(msg));
    break;
}
if (exitFlag)
    break;
// 실제로 서버에 보낼 메세지 형태
// buf = "[ssafy] 안녕 얘들아"
sprintf(buf, "%s %s", name, msg);
write(sock, buf, strlen(buf));
```

멀티 채팅 클라이언트 분석

receiveMsg

- 수신 쓰레드
- 역시, 종료조건으로 exitFlag

- 서버로부터 받은 메세지 출력

- kill(0, SIGINT)
  - Ctrl + C 를 입력한 효과
  - interrupt 함수 실행
  - 쓰레드, 소켓, 프로그램 안전종료

```
void *receiveMsg()
{
    char buf[NAME_SIZE + MSG_SIZE];
    while (!exitFlag)
    {
        memset(buf, 0, MSG_SIZE);
        int len = read(sock, buf, NAME_SIZE + MSG_SIZE - 1);
        if (len == 0)
        {
            printf("INFO :: Server Disconnected\n");
            kill(0, SIGINT);
            exitFlag = 1;
            break;
        }
        printf("%s\n", buf);
    }
}
```

멀티 채팅 테스트

빌드

```
$ gcc ./chat_client.c -o chat_client -lpthread
```

서버 동작

```
$ ./chat_server
```

클라이언트 동작

세 개 정도 터미널을 켜 테스트

```
$ ./chat_client david
$ ./chat_client sujin
$ ./chat_client minho
```

```
안녕 내 이름은 민호야
[minho] 안녕 내 이름은 민호야
다들 잘 지내?
[minho] 다들 잘 지내?
[sujin] 안녕 민호
[sujin] 나는 수진이야
[david] 하이
[david] 나는 데이비드!
```

socket 프로그래밍은 정확한 사용법을 익혀야한다

- **오늘 배운 내용 철저히 복습**
  - echo 서버, 클라이언트
  - chat 서버, 클라이언트
  - 네 개의 코드를 직접 작성해보고, 완벽히 익히기

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

TCP/IP 응용

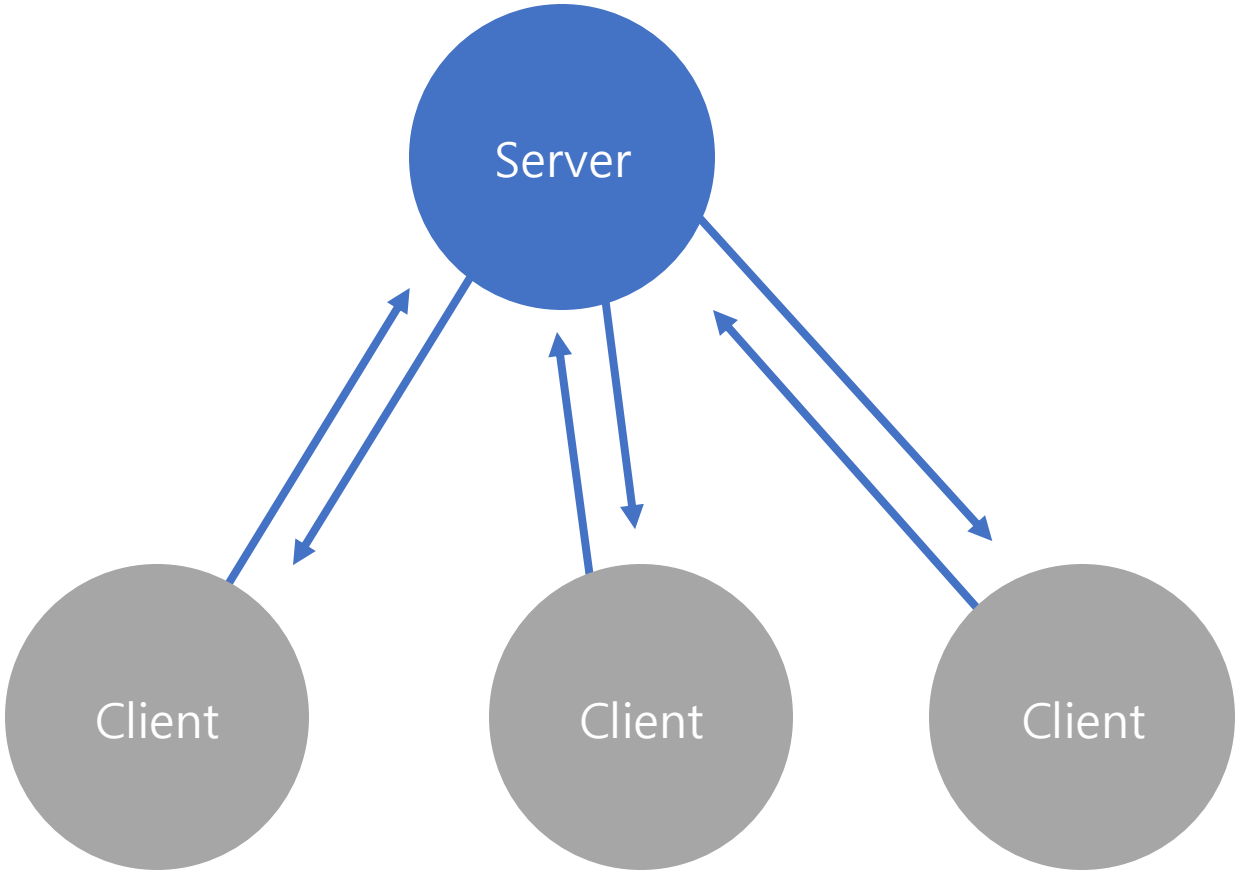
1: 多통신구현

채팅이 아니다.

여러 Client의 요청을

개별적으로 처리

1. Client가 소문자로 전송하면,
2. Server가 그 메세지를 대문자로 바꾼다.
3. 다시 Client 가 전달한다.
4. Client 출력한다.



SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

내일 방송에서 만나요!

삼성 청년 SW 아카데미