시스템 프로그래밍

2차 Warm-up 과제

pthread를 이용한
client-side socket programming

group 15

컴퓨터학과 2013210107 김정수

컴퓨터학과 2014210075 이한얼

제출 : 2016.11.24

사용 freeday – 0일

환경

가상머신 : Oracle VM VirtualBox

운영체제 : Ubuntu 16.04 LTS

커널 : linux-4.4.1

작성한 부분에 대한 설명

- **server.c**

서버에 대한 코드는 이미 제공해 주었지만, client에 대한 부분을 설명하기 위해서는 서버 측의 소스 구조를 먼저 파악해야 한다.

우선 서버 측은 5개의 소켓을 만들게 되는데, 각 소켓에 대하여 다음과 같은 과정이 진행된다.

socket() - 소켓 하나를 생성

bind() - 생성한 소켓을 server socket으로 등록

listen() - server socket을 통해 클라이언트의 접속 요청 확인

accept() - 클라이언트 접속 요청 대기 및 허락

이후 read()/write()

실제 다음과 같은 코드에서 각각의 port를 입력 받은 후

```
    printf("Type Server Ports(format:<Port1> <Port2> <Port3> <Port4> <Port5> : \n");
    scanf("%d %d %d %d %d", &servPorts[0], &servPorts[1], &servPorts[2], &servPorts[3], &servPorts[4
]);
```

socket 생성 후(IPv4, TCP 형식)

```
hServSock[i] = socket(PF_INET, SOCK_STREAM, 0);
```

입력된 port 주소대로 서버 소켓을 등록하여준다. (bind)

```
servAddr[i].sin_family = AF_INET;
servAddr[i].sin_addr.s_addr = htonl(INADDR_ANY);
servAddr[i].sin_port = htons(servPorts[i]);

if (bind(hServSock[i], (struct sockaddr*)&servAddr[i], sizeof(servAddr[i])) < 0 )
```

이후 listen을 통하여 소켓을 bind된 포트에 연결시킨다.

(backlog는 5이기 때문에 최대 5개까지)

```
if (listen(hServSock[i], 5)<0)  ErrorHandling("listen() error");
```

이제 accept를 통하여 클라이언트의 접속을 기다리게 된다. 이 때, 원래는 클라이언트가 접속을 할 때까지 기다려야 하지만, fcntl함수를 이용하여 non-block 모드로 바꿈으로써, block당하지 않고 계속해서 수행하게 두었다. Accept의 무한루프 문의 경우에는 지속적으로 연결된 소

켓이 총 5개인지를 확인하여 조건을 만족해야만 비로소 루프 문을 빠져나가 다음의 send함수를 실행할 수 있다.

```
fcntl( hServSock[i], F_SETFL, fcntl( hServSock[i], F_GETFL, 0 ) | O_NONBLOCK );
```

```
hClntSock[i] = accept(hServSock[i], (struct sockaddr*)&clntAddr[i], &szClntAddr);
```

이 후 random하게 클라이언트 쪽 소켓을 정하여 임의의 data를 전송한다.

무작위로 전송한다는 부분이 이후, pthread를 사용하는 이유가 된다.

```
//send
while (1) {
    i = rand() % 5;
    //send ◆ ◆
    r = rand() % 10;
    for (int k = 0; k < 1 + (rand() % 50 == 7 ? 10000: 0); k++) {
        //printf("send from port: %d!\n", servPorts[i]);
        for (int j = 0; j < presetMSize[i][r]; j++) message[j] = (rand() % 26) + 'A';
        send(hClntSock[i], message, presetMSize[i][r], 0);

        usleep(rand() % 10);
    }
}
```

- **client.c**

- **<header>**

  pthread.h - thread를 사용

  netinet/in.h - struct sockaddr_in 등을 사용

  sys/socket.h - socket(), bind(), connect() 등 소켓 관련 함수

  arpa/inet.h - inet_addr()

  time.h - 현재 시간 측정

```
1 #include <linux/kernel.h>
2 #include <pthread.h>    // use thread
3 #include <stdio.h>
4 #include <string.h>
5 #include <stdlib.h>
6 #include <unistd.h>
7 #include <netinet/in.h> // struct sockaddr_in
8 #include <sys/socket.h> // connect ... functions for socket
9 #include <sys/types.h>  // typedef
10 #include <arpa/inet.h>  // inet_addr()
11 #include <time.h>
12 #include <sys/time.h>
```

- **<define>**

  버퍼 사이즈(server 의 buffer 사이즈에 맞춤), 서버 IP 주소, 포트 번호 등을 미리 정의.

```
14 #define BUFFER_LEN 65536
15 #define BUFF_SIZE 65536
16 #define SERV_IP_ADDR "192.168.56.103" // SERVER IP address
17 #define PORT1 1111      // each SERVER PORTS
18 #define PORT2 2222
19 #define PORT3 3333
20 #define PORT4 4444
21 #define PORT5 5555
22 #define NUM_PORTS 5     // number of ports
```

- **<main>**

  pthread_t sock_th[] - 각각의 thread를 지시하는 id

  int err - 각 thread 관련 함수의 성공 여부 error 체크

  int port[] - 각각의 port 번호를 저장할 변수

  우선 기본적인 과정은 다음과 같다.

  - 5개의 thread를 pthread_create를 통해 생성 및 sock_th로 그 thread id를 받음.

  이 때, 각 thread에 대해 5개의 연결할 서버 port 번호를 각각 넘겨준다.

  각 thread는 f_socket을 사용하게 된다. 이 함수는 뒤에 설명.

```
// start thread, by function f_socket with throw value port number
for(i=0; i<NUM_PORTS; i++) {
    err = pthread_create(&sock_th[i], NULL, f_socket, (void*)&port[i]);

    // error check
    if (err != 0) {
        printf("port: %d thread create error\n", port[i]);
    }
}
```

  - pthread_join을 통해 모든 thread가 끝나야만 main 함수가 종료되게끔 설정.

  NULL로 표시된 부분은 thread 함수 내 return을 통해 값을 받아낼 수 있는데, 현재는 사용하지 않음.

```
// join each thread.
// : main function will not finished before jointed thread finished
for(i=0; i<NUM_PORTS; i++) {
    // return value is NULL, it can be &status.
    err = pthread_join(sock_th[i], NULL);

    if (err != 0) {
        printf("port: %d thread join error\n", port[i]);
    }
}
```

- **<f_socket>**

  기본적으로 pthread를 사용할 때 넘겨주는 함수는 void 포인터 함수이다.

  이 함수는 하나의 포트에 대한 처리과정이다.

  int port - pthread_create 시에 넘겨받은 port 번호를 저장

```
int port = *((int*) arg);        // receive server port number
```

  int client_socket - 만들어진 socket에 대한 식별자, 실패시 -1 return

  struct sockaddr_in server_addr - 서버 주소 정보

  char buff[] - 넘겨 받을 data를 저장할 버퍼

  함수의 진행과정은 다음과 같다.

  - socket()함수를 통하여, IPv4 버전, TCP를 사용하는 소켓 생성 및 client_socket에 id 반환.

```
client_socket = socket(AF_INET, SOCK_STREAM, 0);
// if fail, return -1
```

  - 서버 주소 설정. memset()을 통한 0으로 값 초기화 및 IPv4, 서버 IP, PORT 값 설정

```
// memset - reset server address 0
memset(&server_addr, 0, sizeof(server_addr));

server_addr.sin_family = AF_INET;    // IPv4
server_addr.sin_port = htons(port); // SERVER PORT
server_addr.sin_addr.s_addr = inet_addr(SERV_IP_ADDR); // SERVER IP
```

  - connect() 함수를 통한 서버에 연결.

  연결할 client쪽 소켓, 서버 주소 및 크기를 인자로 넘긴다. 실패 시 -1 return.

```
// connect to SERVER with client socket
if(connect(client_socket, (struct sockaddr*)&server_addr, sizeof(server_addr)) == -1) {
    printf("connect FAIL\n");
    pthread_exit(NULL);
}
```

  - 출력 파일 및 현재 시간을 기록할 변수 설정.

  출력파일의 이름은 넘겨받은 port에 대하여 같은 폴더에 port번호.txt 형태로 만든다.

```
// create output file
char outputfile[256];
char portnum[6];

// outputfile = "./port.txt"
strcpy(outputfile, "./");
sprintf(portnum, "%d", port);
strcat(outputfile, portnum);
strcat(outputfile, ".txt");

// open file
FILE* fp;
fp = fopen(outputfile, "wt+");

// declare time structure
struct timeval tv;
struct timezone tz;
struct tm *tm;
```

- 무한 루프.

read() 함수를 통해 서버에서 들어오는 데이터를 읽어 들이고, 동시에 그 데이터의 길이를 int read_size 변수로 받는다.

gettimeofday, localtime 함수를 통하여 현재 시간을 기록하여

[h:m:s.ms 메시지 길이 메시지] 형태로 출력파일에 출력한다.

이후 1초 휴식 후 같은 과정 반복.

```
// infinite LOOP !!
while(1) {
    // read Message from client_socket
    char buffer[BUFFER_LEN];
    int read_size = read(client_socket, buffer, BUFFER_LEN);
    buffer[read_size] = '\0';

    // get current time
    gettimeofday(&tv, &tz);
    tm = localtime(&tv.tv_sec);

    // print "h:m:s.ms msgLen contents"
    fprintf(fp, "%d:%02d:%02d.%ld  %d  %s\n",
            tm->tm_hour, tm->tm_min, tm->tm_sec, tv.tv_usec,
            read_size, buffer);

    // show finish one LOOP
    printf("port::%d::read\n", port);

    // wait 1 second
    sleep(1);

}
```

이 때, read함수는 서버 쪽에서 send된 데이터가 올 때까지 block 상태에 있다. 앞에서 보았듯이 서버는 random한 포트로 데이터를 보내게 된다. 만약에 thread를 쓰지 않는다면, 순차적으로 client 쪽의 특정 포트에 대한의 read에서 원하는 포트로 데이터를 전달받지 못한다면 다른 포트에 대한 처리를 못한 채 계속해서 멈춰있는 상황이 발생할 수 있다. 이를 해결하기 위해 어떤 포트에서 데이터가 오든 각각 독립, 병렬적으로 해당 과정을 수행하기 위하여 thread방식을 사용해야 한다.

결과 및 출력

서버, 클라이언트 ip주소를 설정.

서버 :
```
$ sudo ifconfig enp0s3 192.168.56.103
```

클라이언트 :
```
sudo ifconfig enp0s3 192.168.56.104
```

서버 쪽 포트 설정 및 실행:

```
syspro@oslab:~$ ./server
Type Server Ports(format:<Port1> <Port2> <Port3> <Port4> <Port5> :
1111 2222 3333 4444 5555
```

클라이언트 실행

```
osta@osta-VirtualBox:~/SP_02$ ./client
port :: 5555
port :: 4444
port :: 3333
port :: 2222
port :: 1111
```

출력파일 생성 및 기록

```
1111.txt  3333.txt  5555.txt
2222.txt  4444.txt  aa
```

```
  1 4:16:28.336861  3268  HQNNGCTGCDKBOQKHMUQIYRLYLXPURKYAAONJQJPUMBWAUIJIE
    DEYSBREZTHNJQGSHAGJYIDIRLNQDSJGVQGQRZUQSBGBTOVBOBMPJRXAEKRHECNASVQJUMCM
    LMONADWRDXVPQEUSTULOMWIYYXOINAGYOVLPAIIDHGTZKNRGJEUVBFTZCKKPMSPANBPQLYT
```

과제 수행 시 어려웠던 부분과 해결 방법

비교적 예전에 java로 소켓프로그래밍을 한 경험이 있었기 때문에 전체 개념은 어렵지 않았다. 다만, 클라이언트 측 프로그래밍이 과제였지만 우선적으로 서버 측 코드를 먼저 분석해야 이 과제가 더 수월하게 진행되지 않나 싶다.

처음 호스트 간에 연결문제가 약간 당황스러웠다. DHCP로 IP 자동할당 되게끔 설정되었었는데, 과제 설명서의 캡쳐화면과 달리 ifconfig, ip addr show를 해보아도 IPv4 주소는 없고 IPv6만 있었다. 따라서 임의로 ifconfig enp0s3 ~~.~~.~~.~~ 를 통하여 주소를 바꾼 다음 ping을 날려보았지만, 연결이 되지 않았다. 와이파이를 통하여서도 시도해보고, 포트포워딩도 시도하려 해보다가 그냥 과제 설명서의 192.168.52.103~104 로 두 게스트OS의 IP를 설정해주니 잘 연결이 되었다. 이후, 나머지 실행 역시 수월하였다.