Introduction to Computer Networks

Lab 01: Basic Socket Communication

1. 실습의 목표

socket 통신의 client 예제를 따라 해보고, 코드를 이해한다.

2. 작성해야 할 프로그램

client: socket 통신을 위한 클라이언트 쪽 프로그램

3. 프로그램 예제 설명

아래의 예제를 따라 client.cc 프로그램을 작성한다. 절대 copy & paste 하지 말고 직접 손으로 타이핑할 것! 그래야 자기 것으로 만들 수 있다.

```
// client.cc -- a simple socket client

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>

#define MAXDATASIZE 1000

int main(int argc, char *argv[])
{
    int sockfd, numbytes;
    char buf[MAXDATASIZE];
    struct addrinfo hints, *servinfo;
    int rv;
    char s[INET_ADDRSTRLEN];

    if(argc != 3) {
        fprintf(stderr, "usage: client hostname portnum\n");
        exit(1);
    }

    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;

    if((rv = getaddrinfo(argv[1], argv[2], &hints, &servinfo)) != 0) {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
        return 1;
    }
```

```
    if((sockfd = socket(servinfo->ai_family, servinfo->ai_socktype,
                            servinfo->ai_protocol)) == -1) {
        perror("client: socket");
        return 2;
    }

    if(connect(sockfd, servinfo->ai_addr, servinfo->ai_addrlen) == -1) {
        close(sockfd);
        perror("client: connect");
        return 2;
    }

    inet_ntop(servinfo->ai_family, &((struct sockaddr_in*)servinfo->ai_addr)-
>sin_addr, s, sizeof s);
    printf("client: connecting to %s\n", s);

    freeaddrinfo(servinfo);

    strcpy(buf, "hello");
    if(send(sockfd, buf, strlen(buf), 0) == -1) {
        perror("send");
        close(sockfd);
        exit(1);
    }

    if((numbytes = recv(sockfd, buf, sizeof buf, 0)) == -1) {
        perror("recv");
        close(sockfd);
        exit(1);
    }

    buf[numbytes] = '\0';
    printf("client: received '%s'\n", buf);

    close(sockfd);
    return 0;
}
```

4. 프로그램 예제 설명

아래에는 코딩한 내용을 라인별로 설명하였다.

```
// client.cc -- a simple socket client

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>
```

- client 프로그램을 만들 때 사용되는 헤더 파일들. Socket programming을 위해서 필요한 헤더들이므로 늘 모두 include 시켜준다.

```
#define MAXDATASIZE 1000
```

- buf의 크기로 사용할 MAXDATASIZE를 define문으로 결정.

```
int main(int argc, char *argv[])
{
```

- main 함수. argc는 command line argument의 개수를, argv는 command line string 자체를 담게 될 변수들이다. argc에는 프로그램의 이름 자체가 포함된다. 예를 들어 프로그램 실행 시,

$ ./client 127.0.0.1 55555

이와 같이 실행시키면
argc = 3
argv[0] = "./client"
argv[1] = "127.0.0.1"
argv[2] = "55555"
가 된다.

```
    int sockfd, numbytes;
    char buf[MAXDATASIZE];
    struct addrinfo hints, *servinfo;
    int rv;
    char s[INET_ADDRSTRLEN];
```

- 사용될 변수들. INET_ADDRSTRLEN은 헤더에 미리 지정 되어있는 변수로, IP address를 담을 수 있는 크기이다. 소켓 프로그래밍에서 매우 중요하게 사용되는 구조체인 struct addrinfo는 뒤에 설명하였다.

```
    if(argc != 3) {
        fprintf(stderr, "usage: client hostname portnum\n");
        exit(1);
    }
```

- client 프로그램은 접속할 목적지의 주소와 포트 번호를 인자로 받아야 한다. 따라서 인자가 2개(argc==3)가 아닌 경우에는 안내문을 내보내고 프로그램을 종료한다.

```
    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;

    if((rv = getaddrinfo(argv[1], argv[2], &hints, &servinfo)) != 0) {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
        return 1;
    }
```

- getaddrinfo 함수의 원형은 아래와 같다.

```
int getaddrinfo(const char *node, const char *service,
             const struct addrinfo *hints, struct addrinfo **res);
```

이 함수는 node, service, hints 이 세 가지의 정보를 토대로 res 변수에 결과값을 채우는 함수이다.

struct addrinfo 구조체는 어떤 host의 주소 및 접속할 때 사용하는 프로토콜을 규정하는 구조체로서, 다음과 같은 형태이다.

```
struct addrinfo {
       int                 ai_flags;
       int                 ai_family;
       int                 ai_socktype;
       int                 ai_protocol;
       size_t              ai_addrlen;
       struct sockaddr     *ai_addr;
       char                *ai_canonname;
       struct addrinfo     *ai_next;
};
```

일단 getaddrinfo 함수를 부르기 위해서는, 첫 번째 인자인 node에 IP address가 담긴 스트링을, 두 번째 인자인 service에 포트 번호가 담긴 스트링을, 그리고 세 번째 인자인 *hints*에 ai_family와 ai_socktype을 채운 후에 호출하면 된다. 그러면, getaddrinfo가 ai_protocol, ai_addrlen, ai_addr 등을 채워서 그 결과를 *res*에 채운 다음 리턴한다.

```
    if((sockfd = socket(servinfo->ai_family, servinfo->ai_socktype,
                        servinfo->ai_protocol)) == -1) {
        perror("client: socket");
        return 2;
    }
```

- servinfo 구조체를 이용하여 socket을 만든다. 만약 socket 함수의 리턴 값이 -1일 경우 socket 만들기에 실패한 것이므로 에러로 처리해준다.

```
    if(connect(sockfd, servinfo->ai_addr, servinfo->ai_addrlen) == -1) {
        close(sockfd);
        perror("client: connect");
        return 2;
    }
```

- socket을 만들었으면 서버로 연결을 시도한다. connect 함수의 리턴 값이 -1인 경우 연결에 실패한 것이므로 에러로 처리해준다.

```
    inet_ntop(servinfo->ai_family, &((struct sockaddr_in*)servinfo->ai_addr)-
>sin_addr, s, sizeof s);
    printf("client: connecting to %s\n", s);
```

- 서버로 연결했다는 메시지를 출력하기 위한 부분으로 연결 자체와는 관련이 없다. 여기서 알아야 할 것은, struct addrinfo에 들어있는 ai_addr이 ip address를 담고 있다는 것과, 이 ip address를 스트링으로 만들기 위해서는 위와 같이 inet_ntop 함수를 호출해주면 된다는 것이다. 그러면 ip address가 스트링의 형태로 *s*에 담기게 된다.

```
    freeaddrinfo(servinfo);
```

- 연결에 성공했다면 서버의 정보를 담고 있는 servinfo 구조체는 더 이상 필요 없으므로 free 시켜준다.

```
    strcpy(buf, "hello");
    if(send(sockfd, buf, strlen(buf), 0) == -1) {
        perror("send");
        close(sockfd);
        exit(1);
    }
```

```
    if((numbytes = recv(sockfd, buf, sizeof buf, 0)) == -1) {
        perror("recv");
        close(sockfd);
        exit(1);
    }
```

- socket이 연결된 후에 메시지를 주고받기 위한 함수는 send와 recv이다. send와 recv가 항상 if 문과 함께 쓰인 이유는 에러처리 때문이며 항상 이런 식으로 사용하는 것이 좋다.

send 함수의 원형은 아래와 같다.
ssize_t send(int *sockfd*, const void **buf*, size_t *len*, int *flags*);

첫 번째 인자는 통신하는 소켓을 의미하고, 두 번째가 전송할 스트링이다. 세 번째 인자는 전송하는 메시지의 사이즈를 의미하는 것으로 내가 크기를 특정 크기로 제한하는 경우가 아니라면 일반적으로 strlen(buf) 와 같이 써주면 된다. 마지막 인자는 특별한 일이 없는 한 0으로 해주면 된다.

send 함수가 성공하면 리턴 값은 전송한 스트링의 길이이고, 실패할 경우 -1이 리턴된다.

recv 함수의 원형은 아래와 같다.
ssize_t recv(int *sockfd*, void **buf,* size_t *len*, int *flags*);

여기서의 인자들은 send와 동일하지만 의미는 다르다. 두 번째 인자인 buf에는 수신된 스트링이 담긴다. 그리고 세번째 인자인 len은 수신된 메시지의 크기가 얼마일지 모르기 때문에 buf의 크기인 sizeof buf와 같이 지정해준다. 만약 수신된 메시지의 크기가 buf보다 크다면 buffer overflow가 발생할 수 있다. 따라서 일반적으로 buf의 크기는 크게 잡아주는 것이 좋다. (여기 예제에서는 MAXDATASIZE, 즉 1000으로 잡았다.)

만약 어떤 메시지가 수신되었는데 스트링의 크기가 1500이었다고 하자. 이 때 recv 함수의 세 번째 인자를 1000으로 하면 메시지가 분할되어 먼저 앞의 1000개의 문자만 수신되고, 그 다음에 500개의 문자는 다음 번 recv 함수가 불렸을 때 수신된다. 이와 같은 동작을 잘 이해해야 나중에 이에 따라 프로그램을 작성할 수 있으므로 유의해야 한다.

recv 함수의 리턴 값도 중요하다. recv가 성공할 경우 수신된 메시지의 바이트 수가 리턴된다. recv가 실패할 경우에는 -1이 리턴된다. 만약 recv의 리턴 값이 0이라면, 통신을 하고 있는 상대방이 연결을 끊었다는 뜻이다. 이 사실은 서버를 구현하는데 있어서 매우 중요하다.

```
    buf[numbytes] = '\0';
    printf("client: received '%s'\n", buf);
```

- recv 함수를 이용해 받은 스트링의 특징은, 마지막에 널문자가 붙지 않은 상태로 받는다. 따라서 스트링으로 사용하기 위해서는 반드시 buf[numbytes] = '₩0' 과 같이 처리해주어야 한다. numbytes는 수신된 바이트 수를 의미하므로, 이렇게 하면 수신된 문자열의 맨 마지막에 널문자를 넣어주는 역할을 하게 된다.

```
    close(sockfd);
    return 0;
}
```

- 프로그램의 마지막이다. 파일처리에서도 파일을 마지막에 닫아주듯이, 소켓도 close함수를 이용하여 닫아준다. 그리고는 프로그램을 끝낸다.

5. 컴파일 및 실행

- 프로그램을 컴파일 하여 실행파일을 얻는다.
$ g++ -o client client.cc

- 서버를 실행시킨다. (서버와 클라이언트를 위해 두 개의 창을 띄운다.)
$ ./server 55555

- 클라이언트를 실행시킨다.
$ ./client 127.0.0.1 55555

주의!) 포트번호는 각자 다른 번호로 해야 한다. 포트번호는 20000-60000 사이 중에서 아무 번호나 선택한다. 만약 server 프로그램을 실행시켰을 때 Address already in use 에러가 난다면 다른 사람의 포트번호와 충돌한 것이므로 다른 번호를 선택한다.

server가 어떤 역할을 하는지 확인해본다.

6. recv 함수에 대하여

recv 함수의 사용법은 위에서 설명하였다. 여기서 알아야 할 것은, recv가 blocking 함수라는 것이다. blocking 함수란, 함수가 수행하는 역할이 끝날 때까지 리턴 되지 않는 함수를 말한다. 다시 말해, 어떤 메시지가 수신되기 전까지는 recv 함수에서 걸려서 넘어가지 않는다.

예를 들어, 위의 예제에서 send와 recv의 순서를 바꿔서 다음과 같이 작성했다고 하자.

```
    if((numbytes = recv(sockfd, buf, sizeof buf, 0)) == -1) {
        perror("recv");
        close(sockfd);
        exit(1);
    }
```

```
    strcpy(buf, "hello");
    if(send(sockfd, buf, strlen(buf), 0) == -1) {
        perror("recv");
        close(sockfd);
        exit(1);
    }
```

만약 이렇게 구현한다면, 무언가 서버로부터 메시지가 오지 않는 한 recv 함수는 리턴되지 않고 "hello" 메시지는 전송되지 않은 채 계속 기다리게 된다.

7. 과제

client.cc를 다음과 같이 수정하여 lab1_client.cc 파일로 만든다.

- 현재는 "hello"를 보내고 서버로부터 메시지를 수신한다. 실행해보면 서버는 받은 메시지를 그대로 돌려주는 echo server 라는 것을 알 수 있다.

- 작성할 프로그램에서는 코드 상에서 "hello"를 보내는 대신, 사용자 입력을 받아서 서버 쪽으로 전송하는 것으로 한다. 아래와 같은 조건을 만족시켜야 한다.

1) 프로그램이 실행되면, 아래와 같은 프롬프트(prompt)가 표시된다.

```
command>
```

여기에 사용자가 무언가를 입력하면 그 내용을 서버 쪽으로 보낸다. 메시지를 받은 서버는 클라이언트 쪽으로 메시지를 보내줄 것이고, 그것을 받아서 화면에 출력하면 된다.

2) 본 랩에서 구현한 예제에서는, 한번 "hello"를 전송하고 서버로부터 메시지를 받아서 출력한 다음 프로그램을 끝낸다. 하지만 과제에서는 프로그램을 끝내지 않고, 서버에서 받은 내용을 화면에 출력한 후 다시 프롬프트를 띄워서 사용자의 입력을 받게 해야 한다.

3) 만약 사용자가 "quit"이라고 입력하면, 그 때 소켓을 닫고 프로그램을 종료한다.

- 아래와 같이 컴파일 하여 서버를 켜 놓고 테스트한다.

```
$ g++ -o lab1_client lab1_client.cc
$ ./lab1_client 127.0.0.1 55555
```

Tip) 사용자 입력을 받기 위해서는, gets 또는 fgets 함수를 주로 사용하는데, gets는 보안 문제 발생의 위험이 있기 때문에 대부분 fgets를 사용한다. 아래와 같이 사용하면 된다.

```
char str[100];
fgets(str, 100, stdin);
```

이렇게 하면 사용자로부터 입력을 기다리고, 사용자가 무언가를 입력한 후 엔터 키를 치면 fgets가 함수가 리턴 되면서 str에 사용자가 입력한 내용이 들어간다.