Introduction to Computer Networks
Lab 04: select 함수를 이용한 Client Implementation

1. 실습의 목표

Lab 1에서 구현한 클라이언트 프로그램은 서버로 보내고 서버의 응답을 기다리는 send → recv 형태로 구현 되어있다. 이러한 구조에서는 반드시 클라이언트가 서버로 메시지를 보내야만 (send) 서버의 메시지를 받을 수 있다 (recv). 마찬가지로 내가 한번 메시지를 보내고 나서는 서버로부터 메시지를 받아야 다음 메시지를 전송할 수 있다. 하지만 채팅 프로그램을 생각해보면 내가 서버로 메시지를 보내지 않아도 언제든지 서버로부터 메시지가 올 수 있고, 나도 언제든지 메시지를 보낼 수 있어야 한다. 이렇게 하기 위해서는 클라이언트에서도 select 함수를 사용해야 한다. 본 실습에서는 클라이언트도 select 함수를 사용하도록 수정한다.

2. 작성해 볼 프로그램

selectclient: socket 통신을 위한 클라이언트 프로그램. select 함수를 이용하여 구현한다.

3. 프로그램 예제 설명

아래의 예제를 따라 selectclient.cc 프로그램을 작성한다. 음영처리 된 부분이 client.cc에서 달라진 부분이다.

```
// selectclient.c -- socket client using select()

#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    // max number of bytes we can get at once

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

    fd_set master, read_fds;
```

```
    if (argc != 3) {
        fprintf(stderr,"usage: %s hostname portnum\n", argv[0]);
        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); // all done with this structure

    FD_ZERO(&master);
    FD_SET(0, &master);
    FD_SET(sockfd, &master);

    int quit = 0;
    while(1) {
        read_fds = master;
        if(select(sockfd+1, &read_fds, NULL, NULL, NULL) == -1) {
            perror("select");
            exit(1);
        }

        for(int i=0; i<=sockfd+1; i++) {
            if(FD_ISSET(i, &read_fds)) {
                if(i == 0) {
                    // message entered by user
                    gets(buf);
                    if(strcmp(buf, "quit") == 0) {
                        quit = 1;
                        break;
                    }
                    if(send(sockfd, buf, strlen(buf), 0) == -1) {
                        perror("send");
                        close(sockfd);
                        exit(1);
                    }
```

```
            }
            else if(i == sockfd) {
                // message received from server
                if((numbytes = recv(sockfd, buf, MAXDATASIZE-1, 0)) == -1) {
                    perror("recv");
                    close(sockfd);
                    exit(1);
                }

                if(numbytes == 0) {
                    printf("server disconnected.\n");
                    quit = 1;
                    break;
                }


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

    if(quit) break;
}

printf("connection closed.\n");
close(sockfd);

return 0;
}
```

4. 프로그램 예제 설명

아래에는 selectclient의 내용을 라인별로 설명하였다.

```
fd_set master, read_fds;
```

- fd_set에 관한 자세한 내용은 Lab 3의 설명을 참조한다. selectserver와 같이 selectclient에서도 소켓을 관찰하기 위해 master와 read_fds의 자료구조가 필요하다.

클라이언트에서는 중요한 소켓이 두 개이다. 사용자 입력이 들어오는 경우에는 0번 소켓이 켜지고, 서버로부터 메시지가 오는 경우에는 일반적으로 3번 소켓이 켜질 것이다. 서버처럼 여러 개의 노드와 동시에 통신하는 것이 아니기 때문에 fdmax 값을 관리할 필요는 없다.

```
FD_ZERO(&master);
FD_SET(0, &master);
FD_SET(sockfd, &master);
```

- master 자료구조에서 관찰하고자 하는 소켓을 표시한다. 처음에 FD_ZERO는 모든 소켓을 꺼주는 함수이고, FD_SET은 소켓을 켜주는 함수이다. 먼저 0번 (사용자입력)을 켜주고, 그 다음으로는

서버와 연결된 소켓 (sockfd) 을 켜준다. sockfd는 일반적으로 3번일 것이며, master는 다음과 같이 될 것이다.

0	1	2	3	4	5	6	7	8	...

```
    int quit = 0;
    while(1) {
        read_fds = master;
        if(select(sockfd+1, &read_fds, NULL, NULL, NULL) == -1) {
            perror("select");
            exit(1);
        }
```

- 클라이언트도 무한루프에 들어간다. 이 때 quit 변수는 루프를 나오기 위한 조건을 표시하기 위해 선언해 준 것이다.

무한루프에 들어가면 selectserver와 마찬가지로 read_fds로 master의 값을 복사해주고, read_fds를 이용하여 select 함수를 호출한다. select 함수는 read_fds의 소켓들 중 켜져 있는 소켓에 대해서 메시지가 왔는지 확인하고, 메시지가 있으면 켜진 채로 놔두고 그렇지 않으면 해당 번호를 꺼준다.

- select 함수 호출 전 read_fds

0	1	2	3	4	5	6	7	8	...

- select 함수 호출 후, 만약 사용자 입력만 들어온 경우

0	1	2	3	4	5	6	7	8	...

- select 함수 호출 후, 만약 서버로부터 메시지가 온 경우 (사용자 입력은 없음)

0	1	2	3	4	5	6	7	8	...

- select 함수 호출 후, 사용자 입력도 없고 서버로부터 메시지도 오지 않은 경우

0	1	2	3	4	5	6	7	8	...

- select 함수 호출 후, 사용자 입력도 있고 서버로부터 메시지도 온 경우

0	1	2	3	4	5	6	7	8	...

```
    for(int i=0; i<=sockfd+1; i++) {
        if(FD_ISSET(i, &read_fds)) {
            if(i == 0) {
                // message entered by user
                gets(buf);
                if(strcmp(buf, "quit") == 0) {
                    quit = 1;
                    break;
                }
                if(send(sockfd, buf, strlen(buf), 0) == -1) {
                    perror("send");
                    close(sockfd);
                    exit(1);
                }
            }
```

- select 함수 호출 후에는 for loop을 이용하여 켜져 있는 소켓이 무엇인지 검색한다.
만약 0번이 켜져 있다면, 사용자 입력이 들어온 것을 말한다. 이러한 경우 다음과 같이 동작시킨다.

- gets 함수를 이용하여 사용자 입력을 읽는다.
- 만약 사용자 입력이 "quit"라면 무한루프를 나가기 위해 quit 변수를 1로 만들고 break로 for loop을 빠져나간다.
- 만약 사용자 입력이 "quit"이 아니라면, 서버로 전송한다.

```
            else if(i == sockfd) {
                // message received from server
                if((numbytes = recv(sockfd, buf, MAXDATASIZE-1, 0)) == -1) {
                    perror("recv");
                    close(sockfd);
                    exit(1);
                }

                if(numbytes == 0) {
                    printf("server disconnected.\n");
                    quit = 1;
                    break;
                }

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

- 만약 서버로부터 메시지가 온 경우에는 sockfd 에 해당하는 소켓 번호가 켜져 있을 것이다. 이런 경우에는 다음과 같이 동작시킨다.

- recv 함수를 이용하여 서버로부터 온 메시지를 읽어 들인다.
- 만약 numbytes 가 0이라면 서버가 connection을 종료시킨 것이므로 클라이언트도 종료시켜준다. (quit 변수를 1로 만들고 for loop을 나간다.)
- 만약 numbytes 가 0이 아닌 경우에는 모니터에 출력한다.

```
}
        }

        if(quit) break;
    }

    printf("connection closed.\n");
    close(sockfd);

    return 0;
}
```

- for loop이 끝났다는 것은, select 결과로 나온 read_fds에 대한 처리가 끝났다는 의미이다. 그러면 다시 while 루프의 처음으로 돌아가 select 함수를 호출한다.

이 때 quit 변수를 체크하여 만약에 quit 변수가 1이라면 무한루프를 빠져나간다.

무한루프를 빠져나간다는 것은 클라이언트를 종료시킨다는 뜻이므로, 소켓을 닫은 후 main 함수를 리턴시켜 프로그램을 끝낸다.

5. 컴파일 및 실행

이제 클라이언트까지 select 함수를 사용하게 되면 채팅 프로그램에 근접하게 된다. 바이너리 파일로 주어진 서버를 이용하여 selectclient의 테스트를 수행해보자.

- 프로그램을 컴파일 하여 실행파일을 얻는다.

```
$ g++ -o selectclient selectclient.cc
```

- **세 개**의 ssh 창을 띄운다. 먼저 바이너리로 주어진 **chatserver**를 실행시킨다.

```
$ ./chatserver 55555
```

- **selectclient 두 개를 실행시켜서 서버에 접속한다**. 클라이언트를 실행시킨다.

```
$ ./selectclient 127.0.0.1 55555
```

- 또 하나의 클라이언트를 실행시킨다.

```
$ ./selectclient 127.0.0.1 55555
```

각각의 클라이언트에서 메시지를 입력해보고 어떤 식으로 동작이 되는지 확인한다.

6. 과제

selectclient를 다음과 같이 수정하여 lab4_client.cc 파일로 만든다.

- 프로그램을 실행할 때 받는 인자를 하나 추가시킨다. 현재까지는 인자로 서버 주소와 포트 넘버 두 개를 받았는데, 세 번째 인자로 사용자 id를 받는다. 따라서 lab4_client는 다음과 같이 실행시켜야 할 것이다.

$ ./selectclient 127.0.0.1 55555 dragon

- 서버로 메시지를 보낼 때 사용자 id를 붙여서 보낸다. 예를 들어 사용자가 "hello"라고 입력하면, 서버로는 "dragon: hello" 와 같이 보내주면 된다.

- 테스트할 때는 바이너리로 제공된 chatserver를 이용해서 서버 1개와 클라이언트 2개를 띄워놓고 테스트해본다.

$ ./chatserver 55555 (1번 창)

$ ./lab4_client 127.0.0.1 55555 alice (2번 창)

$ ./lab4_client 127.0.0.1 55555 bob (3번 창)

- 채팅이 잘 되는지 확인한다.

주의: 콘솔에서 채팅 프로그램을 만들면, 사용자가 입력하는 부분과 서버의 메시지를 보여주는 부분이 같기 때문에, 내가 무언가 문장을 입력하고 있을 때 서버로부터 메시지가 와서 출력되면 겹칠 수가 있다. 이는 프로그램이 잘못된 것이 아니라 동일한 창에서 하기 때문에 발생하는 것이다. (채팅 프로그램들은 문자열을 입력하는 창이 따로 있다는 것을 생각하면 된다.)