# 네트워크 프로그래밍

## - 기반 서버 & 클라이언트 통신 프로그램 -


*Korea University of Technology and Education*

| | |
|---|---|
| 과 목 명 | 네트워크 프로그래밍 |
| 교 수 명 | 서희석 교수님 |
| 분 반 | 01 |
| 조 원<br>(학 번) | 강인창(2011136004) |
| 제 출 일 | 2017.06.12 |

# 목 차

# Ⅰ. 서론

- 이번 과제의 목표는 채팅 프로그램을 기반으로 서버와 클라이언트 간의 통신을 실습하는 것이다. 조건은 다음과 같다.

1. 서버와 클라이언트 간의 통신 프로그램을 작성
2. 클라이언트는 최소 5개 이상 접속
3. 서버 프로세스가 'hello' 메시지를 받으면 모든 클라이언트에게 서버의 최초 접속시간을 전송한다. 클라이언트 화면에 시간 출력
4. 서버 화면에는 총 접속자 수(클라이언트의 총 접속자 수), 각 클라이언트의 접속 시간, 클라이언트의 IP를 출력

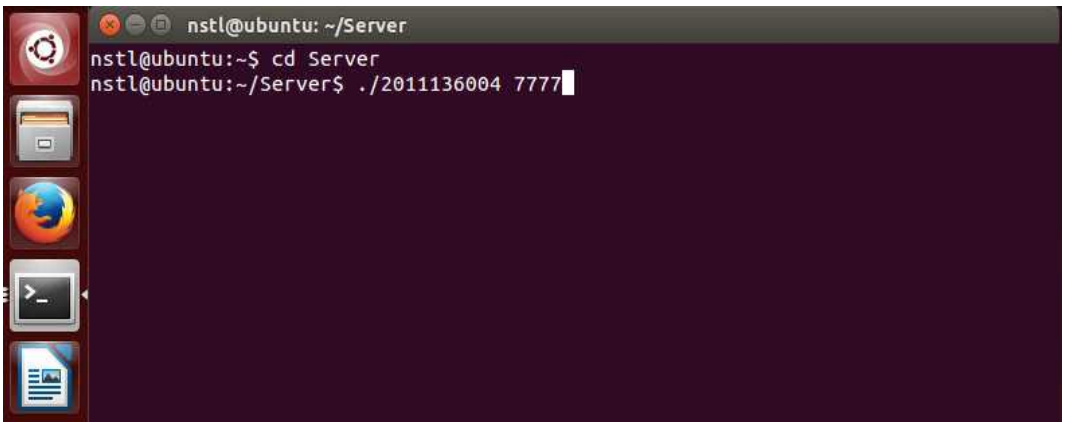
마지막으로, 각 클라이언트가 특정 메시지 'exit'을 입력하면, 서버로의 접속을 끊고 서버는 이를 공고하는 테스크를 추가하기로 계획하였다.

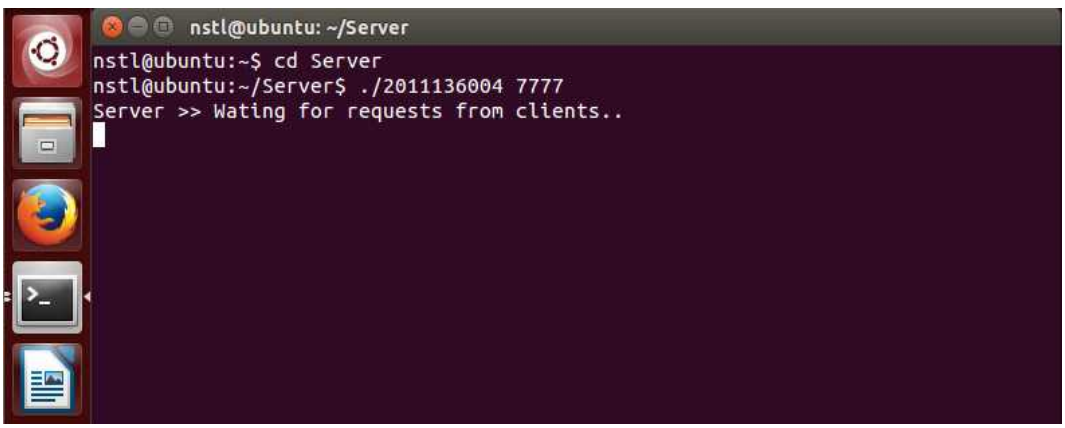
# Ⅱ. 본론

## 1. 통신 서버 테스트

### 1.1. 서버 실행

- 서버와 클라이언트 간 원활한 통신을 이루기 위해, 우선 서버를 실행한다. Server 폴더에 들어가 포트 번호를 입력하고 2011136004 파일을 실행한다.



### 1.1.1. 매개변수 검사

- 실행을 위한 매개변수가 적절히 입력되었는지 검사한다. 첫 번째 인자는 파일 이름이고 두 번째 인자는 포트 번호다.

```
//① 첫 번째 인자는 파일이름, 두 번째 인자는 포트 번호
if(argc != 2){
  printf("Usage : %s port₩n", argv[0]);
  exit(0);
}
```

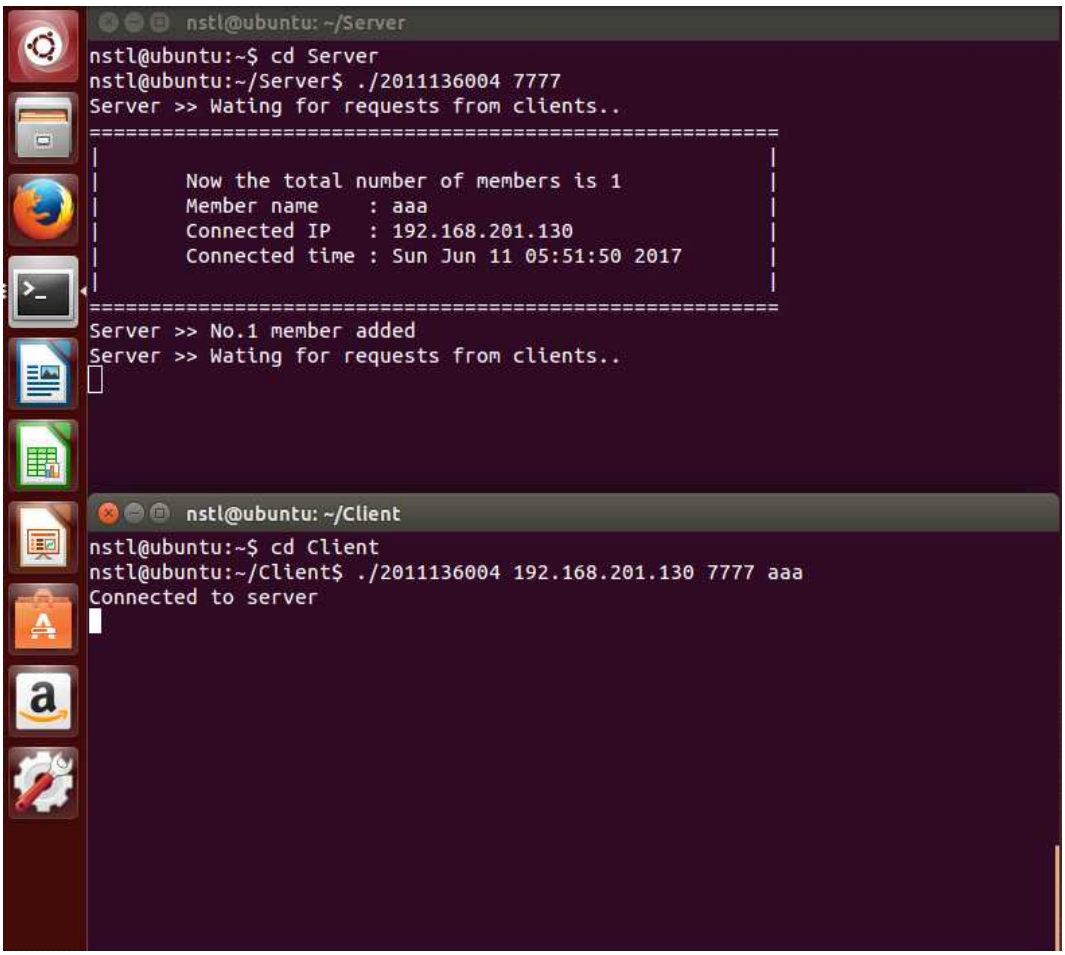
### 1.1.2. 연결 요청 수신

- 서버는 tcp_listen() 메소드를 활용하여 클라이언트의 연결 요청을 기다린다.

```
//① 클라이언트의 연결 요청을 받는다.
listen_sock = tcp_listen(INADDR_ANY, atoi(argv[1]), 5);
int tcp_listen(int host, int port, int backlog) {
  int sd;
  struct sockaddr_in servaddr;
  //② 소켓 생성
  sd = socket(AF_INET, SOCK_STREAM, 0);
  if(sd == -1) {
    perror("Socket Failed");
    exit(1);
  }
  //③ servaddr 구조체의 내용을 세팅한다.
  bzero((char*)&servaddr, sizeof(servaddr));
  servaddr.sin_family = AF_INET;
  servaddr.sin_addr.s_addr = htonl(host);
  servaddr.sin_port = htons(port);
  // ④요청이 오면 알 수 있도록 바인드한다.
  if(bind(sd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
    perror("Bind Failed");
    exit(1);
  }
  //⑤ 클라이언트로부터 연결 요청을 기다린다.
  listen(sd, backlog);
  return sd;
}
```

## 1.2. 클라이언트 수용 및 정보 출력

- 클라이언트로부터 채팅 참가 요청이 오면 이를 받아들이고, 서버 화면에 현재 접속 중인 총 클라이언트 수와 각 클라이언트의 이름, IP, 최초 접속 시간 정보를 출력한다.




### 1.2.1. 모든 소켓의 I/O 변화 감지

- 비동기 모드로 select() 시스템 콜을 사용하여 모든 소켓의 I/O 변화를 감지한다. 이 때, select()를 호출하기 위해 fd_set 구조체를 만들고 읽기 변화 감지용 read_fds를 초기화한 후 I/O 변화에 관심이 있는 소켓번호를 선택한다.

```
//① select()를 호출하기 위해 fd_set 구조체를 만든다.
while(1) {
  //② read_fds의 모든 비트를 0으로 초기화한다.
  FD_ZERO(&read_fds);
  //③ I/O 변화에 관심이 있는 모든 연결용 소켓을 선택한다.
  FD_SET(listen_sock, &read_fds);
  for(i=0; i<num_members; i++) FD_SET(clisock_list[i], &read_fds);
  maxfdp1 = getmax() + 1;
  //④ 읽기 변화 감지용 fd_set으로 select() 시스템 콜을 호출한다.
  if(select(maxfdp1, &read_fds, NULL, NULL, NULL) < 0) errquit("Select Failed");
}
```

### 1.2.2. 클라이언트 소켓 요청 수용

- 서버가 while 무한 루프를 계속 돌면서, 클라이언트 프로세스로부터 채팅 참가 요청을 확인하고 이를 받아들인다.

```
while(1){
                    ⋮
                    ⋮
  //① 연결용 소켓에서 입력이 발생할 경우
  if(FD_ISSET(listen_sock, &read_fds)) {
    //② 요청을 받아들인다.
    accp_sock = accept(listen_sock, (struct sockaddr *)&cliaddr, &addrlen);
    if(accp_sock == -1) errquit("Accept Failed");
    //③ 클라이언트 리스트에 새로운 클라이언트를 추가한다.
    addClient(accp_sock, &cliaddr);
    //④ 정보 출력
    display();
    printf("Server >> No.%d member added₩n", num_member);
  }
}
```

### 1.2.3. 총 클라이언트 수, 각 클라이언트의 이름, IP, 최초 접속시간 기록

- 서버가 클라이언트의 채팅 참가 요청을 받아들일 경우, 총 클라이언트의 수를 증가시키고 클라이언트 각각의 이름과 IP, 최초 접속 시간을 myClient 구조체 변수에 기록한다. 최대 클라이언트 수를 10으로 제한하였다.

```
typedef struct {
        char client_id[50];
        char client_ip[50];
        char client_ftime[50];
}clientinfo;
clientinfo myClient[10];
```

### 1.2.4. 클라이언트 추가

- addClient() 메소드를 통해 클라이언트 리스트에 새로운 클라이언트를 추가하고, display() 메소드로 정보(총 클라이언트 수, 각 클라이언트의 이름, IP, 최초 접속 시간)를 출력한다.

```
void addClient(int s, struct sockaddr_in *newcliaddr) {
  char buf[50], client_name_temp[50];
  char *client_name;
  char exception[] = " []:", exception2[] = "₩n";
  int nbyte = 0;
  //① 클라이언트의 IP를 받아 저장한다.
  inet_ntop(AF_INET, &newcliaddr->sin_addr, buf, sizeof(buf));
  strcpy(myClient[num_members].client_ip, buf);
  //② 클라이언트가 접속한 시간을 받아 저장한다.
  time_t seconds = time(NULL);
```

```
            char *time = strtok(ctime(&seconds), exception2);
            strcpy(myClient[num_members].client_ftime, time);
            //③ 클라이언트의 이름을 받아 저장한다.
            clisock_list[num_members] = s;
            nbyte = recv(clisock_list[num_members], client_name_temp, 50, 0);
            client_name_temp[nbyte] = 0;
            client_name = strtok(client_name_temp, exception);
            strcpy(myClient[num_members].client_id, client_name);
            //④ 총 클라이언트 수를 증가시킨다.
            num_members++;
        }
```

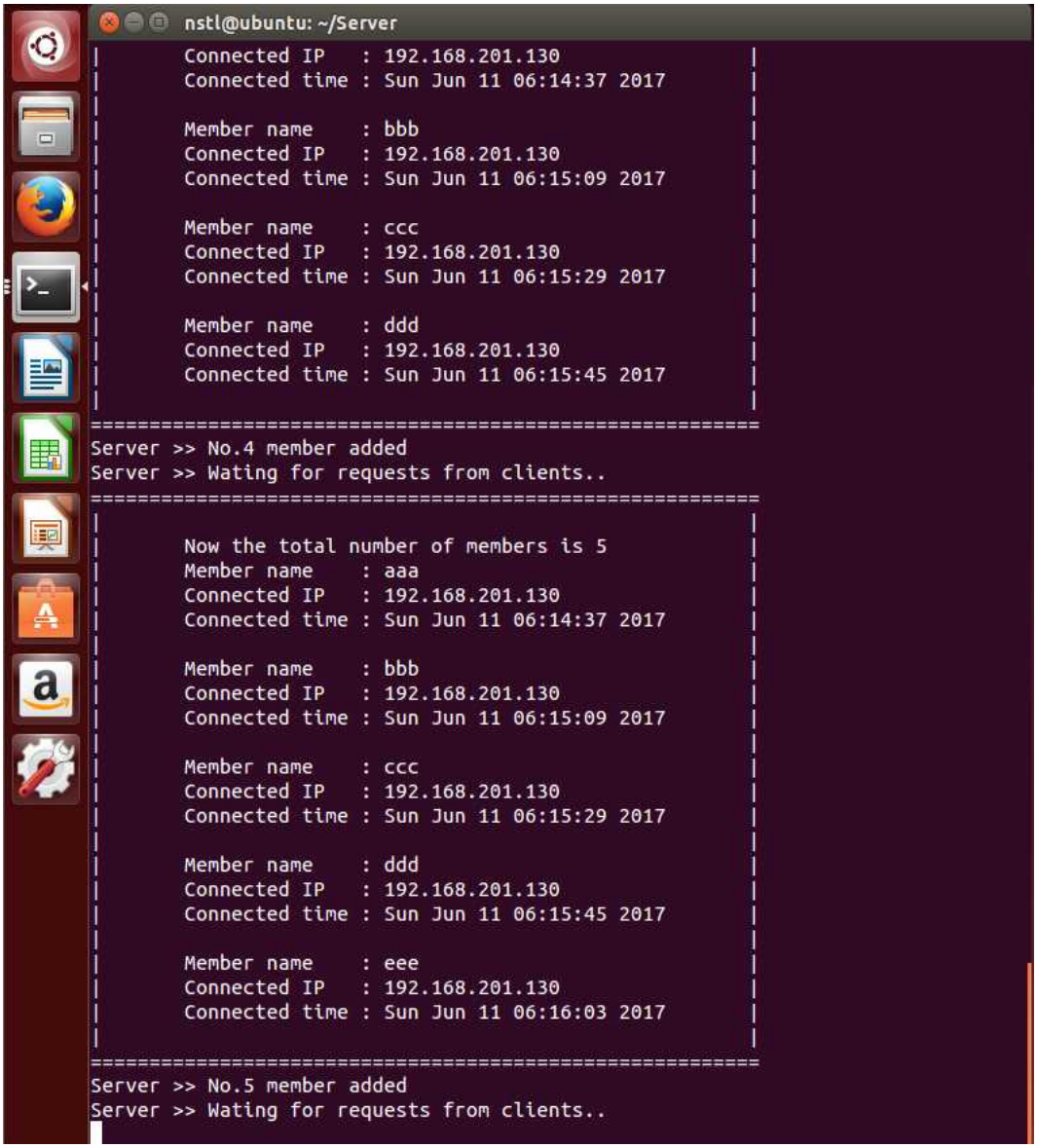
### 1.2.5. 정보 출력

- 클라이언트 접속 시, display() 메소드에서 모든 클라이언트의 myClient 구조체 변수에 접근하여 클라이언트의 이름과 IP, 최초 접속 시간을 서버 화면에 출력한다.

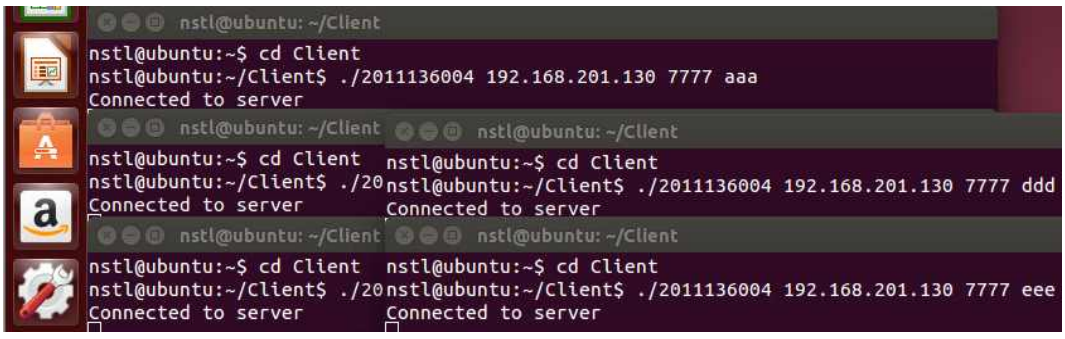
```
        void display() {
            int i;
            printf("================================================
                ========₩n|₩t₩t₩t₩t₩t₩t₩t|₩n");
            printf("|₩tNow the total number of members is %d₩t₩t|₩n", num_members);
            for(i=0; i<num_members; i++) {
                printf("|₩tMember name      : %s₩t₩t₩t₩t|₩n|₩tConnected IP    : %s₩t₩t|₩n|
                    ₩tConnected time : %s₩t|₩n|₩t₩t₩t₩t₩t₩t₩t|₩n", myClient[i].client_id,
                    myClient[i].client_ip, myClient[i].client_ftime);
            }
            printf("============================================
                ============₩n");
        }
```

## 1.3. 다중 클라이언트 정보 출력

- 다중 클라이언트 접속 요청을 받아들이고, 서버 화면에 현재 접속 중인 총 클라이언트 수와 각 클라이언트의 이름, IP, 최초 접속 시간 정보를 각각 출력한다.



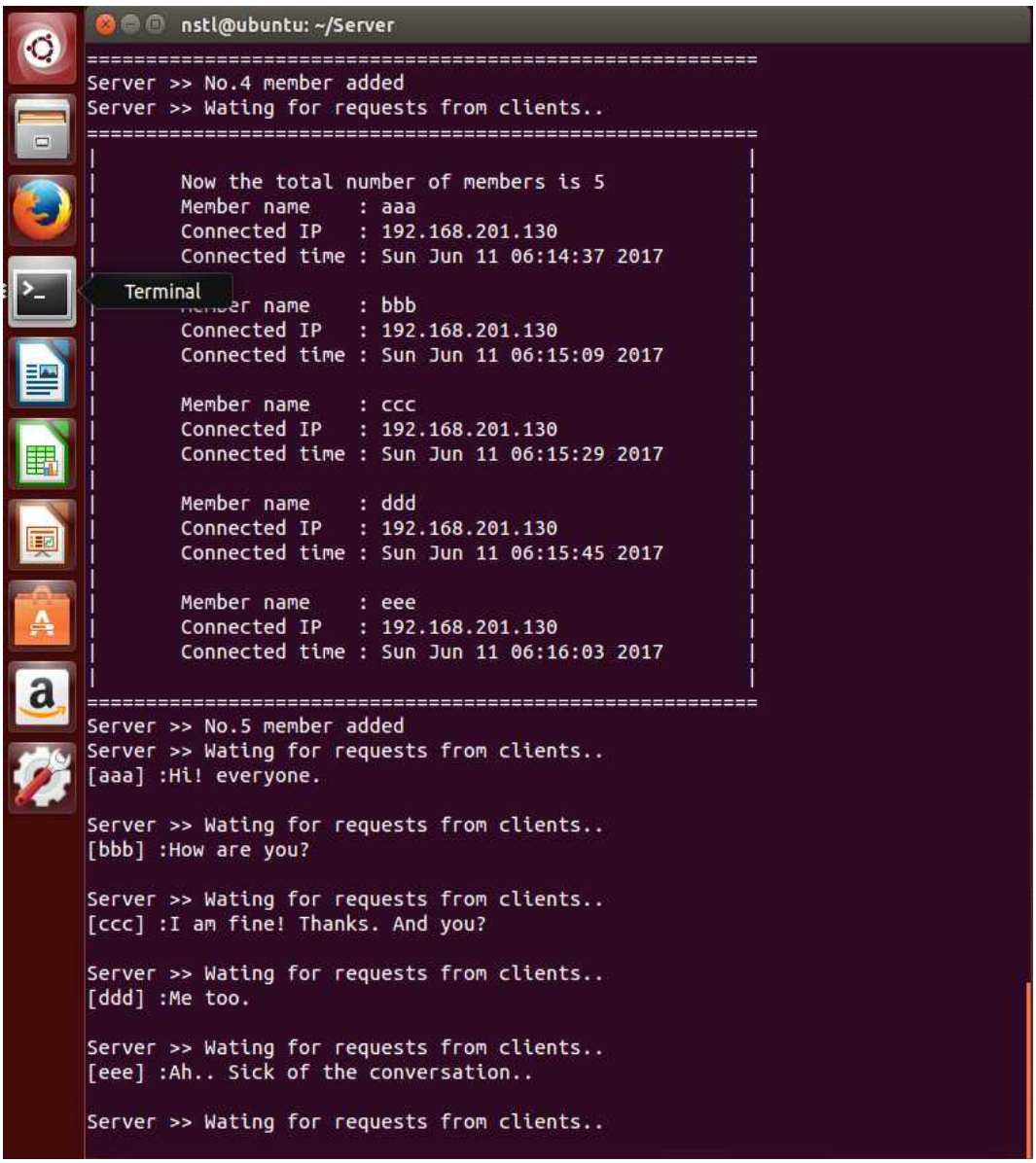

**< 다섯 클라이언트를 유지하는 서버 >**



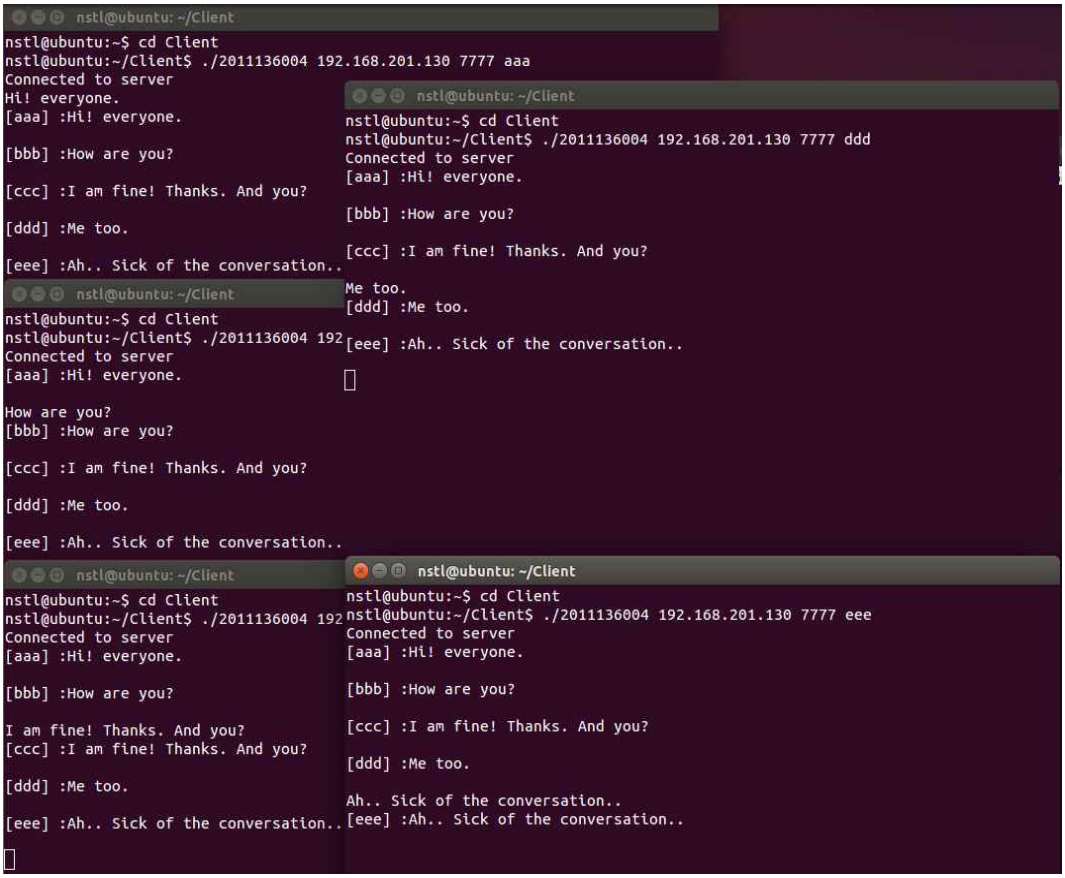

**< 서버에 접속한 다섯 클라이언트 >**

## 1.4. 일반 채팅 기능

- 서버 내에서 다중 클라이언트를 유지하며, 클라이언트 프로세스로부터 받은 메시지를 서버 화면에 출력하고, 모든 클라이언트 프로세스로 브로드캐스팅하여 TCP 기반의 채팅 환경을 이루도록 한다.




**< 채팅 시 서버 화면 >**




**< 채팅 시 클라이언트 화면 >**

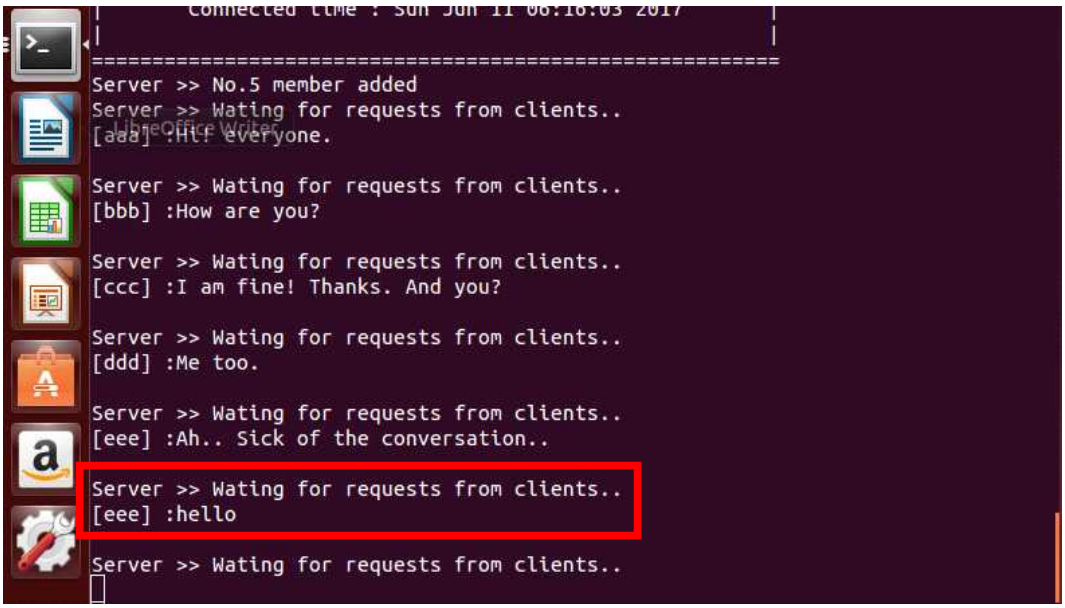
### 1.4.1. 메시지 수신 및 송신

- 모든 클라이언트 I/O 변화를 감지하면서, 메시지 수신 후 송신한다.

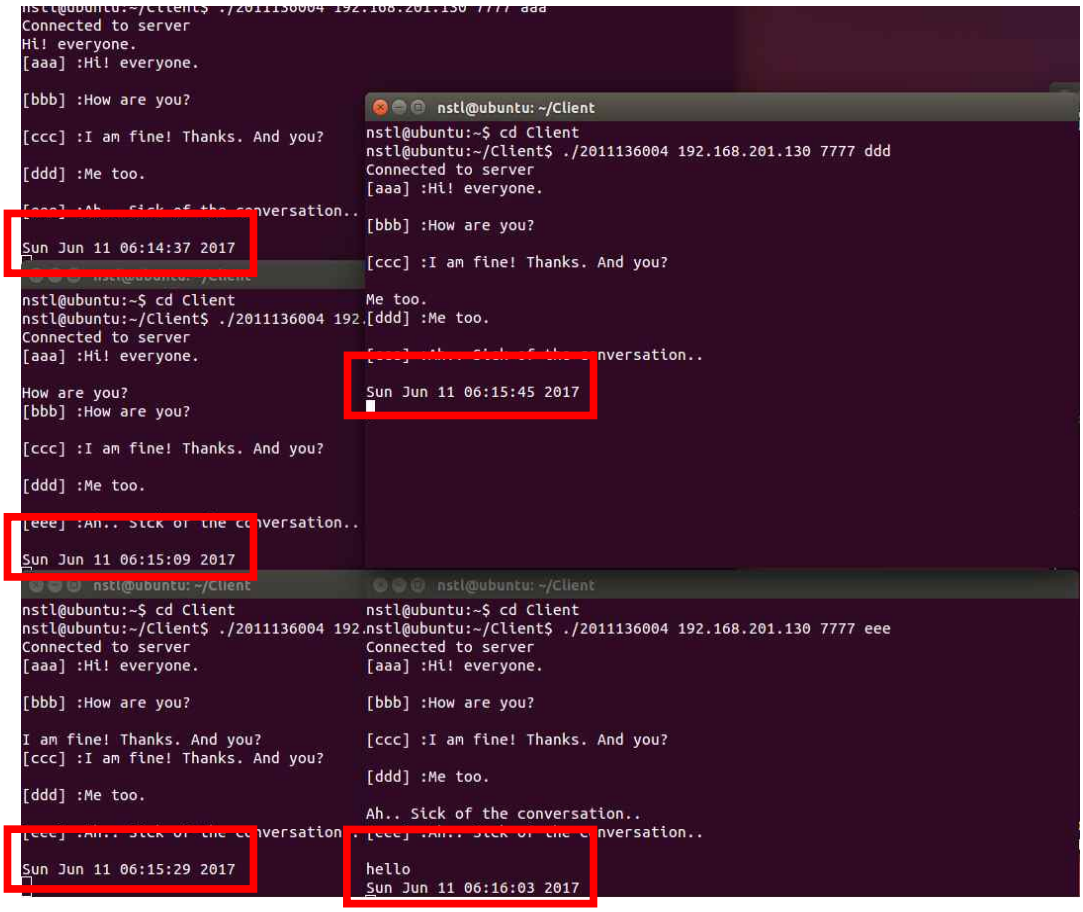
```
while(1) {
  for(i=0; i<num_members; i++) {
    //① 모든 클라이언트로부터 메시지를 수신한다.
    if(FD_ISSET(clisock_list[i], &read_fds)) {
      nbyte = recv(clisock_list[i], buf, MAXLINE, 0);
                ⋮
                ⋮
      //② 수신한 메시지를 모든 클라이언트에게 송신한다.
      for(j=0; j<num_members; j++) {
        send(clisock_list[j], buf, nbyte, 0);
      }
      //③ 서버 내 화면에 출력한다.
      printf("%s₩n", buf);
    }
  }
}
```

## 1.5. 'hello' 메시지 수신 시, 클라이언트로 최초 접속 시간 송신

- 서버 프로세스는 클라이언트 프로세스의 I/O 변화를 감지하면서, 'hello'라는 특정 메시지를 수신 시, 모든 클라이언트 프로세스로 서버 최초 접속 시간을 각각 송신 한다.




**< eee 클라이언트에서 'hello' 입력 시, 서버의 화면 >**




**< eee 클라이언트에서 'hello' 입력 시, 각 클라이언트의 화면 >**

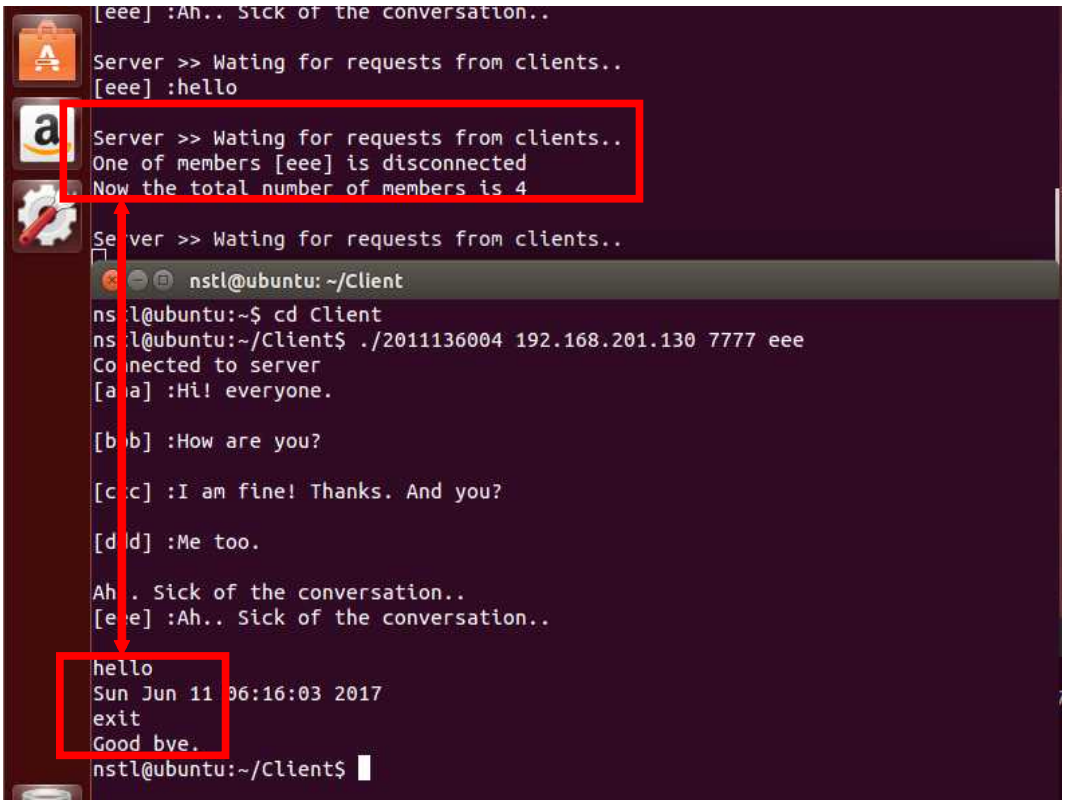
### 1.5.1. 'hello' 메시지 수신 및 최초 접속 시간 송신

- 모든 클라이언트 I/O 변화를 감지하면서, 'hello' 메시지 수신 후 모든 클라이언트로 최초 접속 시간을 송신한다.

```
char *FTIME_STRING = "hello";
while(1) {
  for(i=0; i<num_members; i++) {
    //① 모든 클라이언트로부터 메시지를 수신한다.
    if(FD_ISSET(clisock_list[i], &read_fds)) {
                ⋮
                ⋮
      //② 메시지가 'hello'일 경우 모든 클라이언트로 최초 접속 시간을 송신한다.
      if(strstr(buf, FTIME_STRING) != NULL) {
        for(j=0; j<num_members; j++) {
          send(clisock_list[j], myClient[j].client_ftime), 0);
        }
      }
    }
  }
}
```

## 1.6. 'exit' 메시지 수신 시, 해당 클라이언트 삭제 및 공고

- 서버 프로세스는 클라이언트 프로세스의 I/O 변화를 감지하면서 특정 메시지 'exit' 수신 시, 클라이언트 리스트에서 해당 클라이언트를 삭제 후 이를 서버 화면에 공고한다.



### 1.6.1. 'exit' 메시지 수신

- 모든 클라이언트 I/O 변화를 감지하면서, 'exit' 메시지 수신 후 모든 클라이언트로 최초 접속 시간을 송신한다.

```
char *EXIT_STRING = "exit";
while(1) {
  for(i=0; i<num_members; i++) {
    //① 모든 클라이언트로부터 메시지를 수신한다.
    if(FD_ISSET(clisock_list[i], &read_fds)) {
                    ⋮
                    ⋮
      //② 메시지가 'exit'일 경우 해당 클라이언트를 삭제한다.
      if(strstr(buf, EXIT_STRING) != NULL) {
        removeClient(i);
        continue;
      }
    }
  }
}
```

### 1.6.2. 클라이언트 삭제

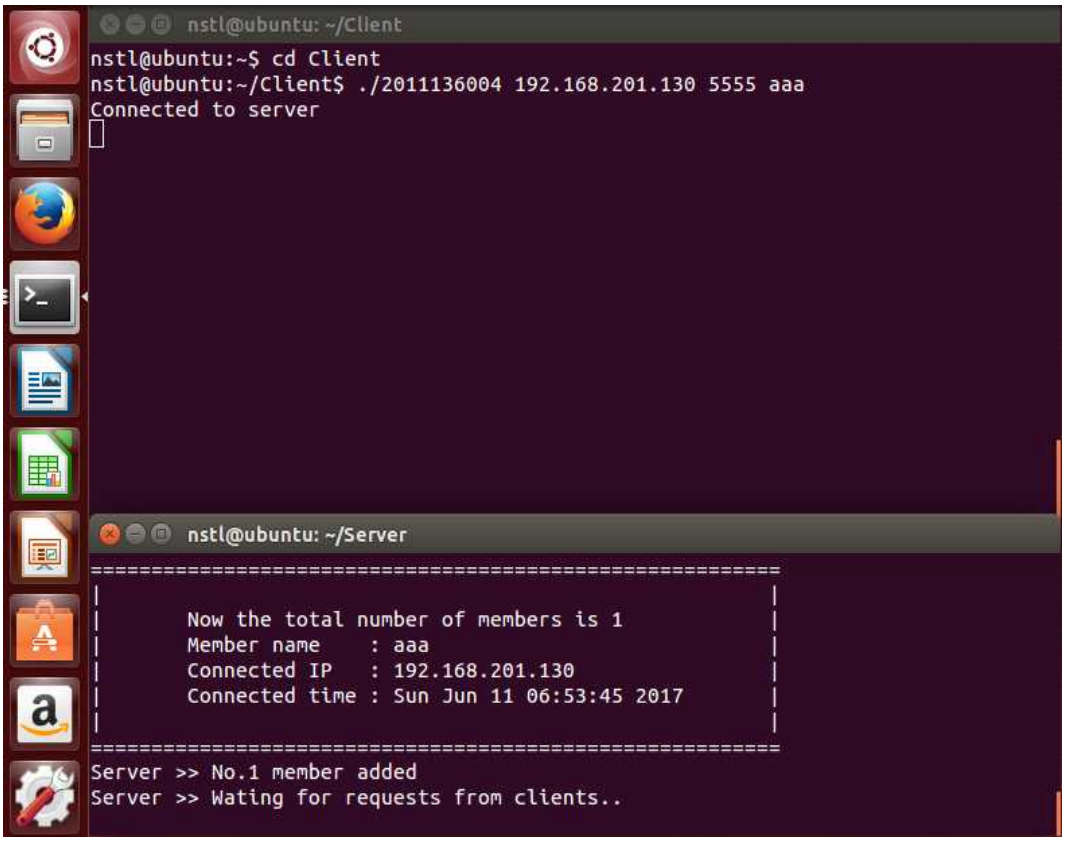
- 모든 클라이언트 I/O 변화를 감지하면서, 'exit' 메시지 수신 시, removeClient() 메소드를 활용하여 해당 클라이언트를 삭제한다. 또한 myClient 구조체 내에 있는 정보(클라이언트 이름, IP, 최초 접속 시간)를 삭제 한다.

```
void removeClient(int s) {
  printf("One of members [%s] is disconnected₩n", myClient[s].client_id);
  //① 연결을 끊는다.
  close(clisock_list[s]);
  int i;
  //② 해당 클라이언트의 이름과 IP, 최초 접속 시간을 삭제한다.
  for(i=s+1; i<num_members; i++) {
    strcpy(myClient[i-1].client_id, myClient[i].client_id);
    strcpy(myClient[i-1].client_ip, myClient[i].client_ip);
    strcpy(myClient[i-1].client_ftime, myClient[i].client_ftime);
  }
  //③ 해당 클라이언트의 소켓을 삭제한다.
  if(s != num_members-1)
    clisock_list[s] = clisock_list[num_members-1];
  //④ 총 클라이언트 수를 감소시킨다.
  num_members--;
  printf("Now the total number of members is %d₩n₩n", num_members);
}
```

# 2. 통신 클라이언트 테스트

## 2.1. 클라이언트 실행

- 서버 실행 후, Client 폴더로 들어가 서버의 IP와 포트 번호, 자신의 이름을 입력하고 2011136004 파일을 실행하여 해당 서버에 접속한다.

### 2.1.1. 매개변수 검사

- 실행을 위한 매개변수가 적절히 입력되었는지 검사한다. 첫 번째 인자는 파일 이름이고 두 번째 인자는 서버의 IP다. 다음으로 포트 번호와 클라이언트의 이름이다.

```
//① 파일 이름, 포트 번호, 클라이언트이름 순으로 인자가 들어간다.
if(argc != 4){
  printf("Usage : %s server_ip server_port client_name₩n", argv[0]);
  exit(0);
}
```

### 2.1.2. 연결 요청 송신

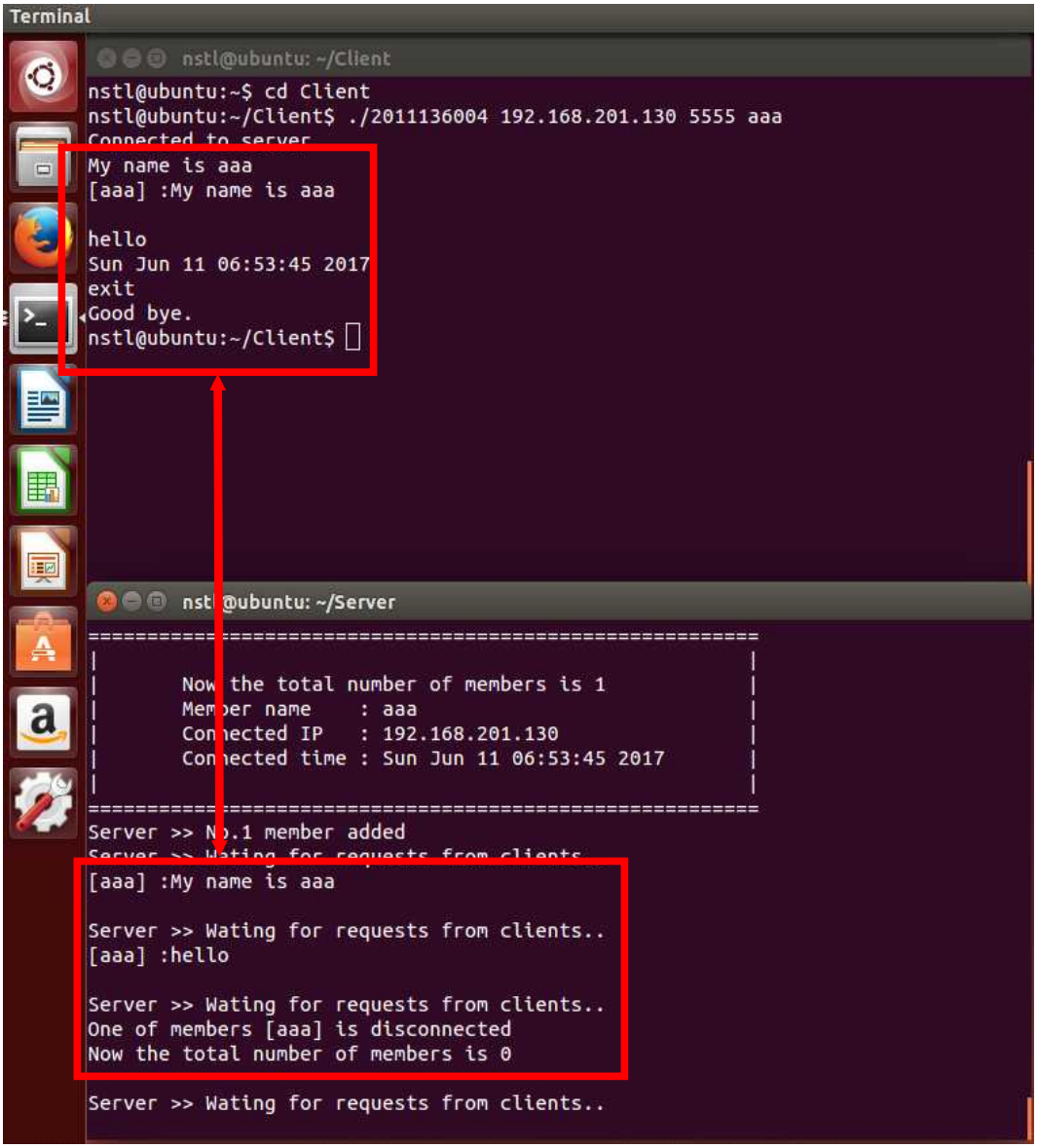
- 클라이언트는 tcp_connect() 메소드를 활용하여 서버로 연결 요청을 송신한다.

```
//① TCP 연결 설정
s = tcp_connect(AF_INET, argv[1], atoi(argv[2]));

int tcp_connect(int af, char *servip, unsigned short port) {
  struct sockaddr_in servaddr;
  int s;
  //② 소켓 생성
  if((s = socket(af, SOCK_STREAM, 0)) < 0)
  return -1;
  //③ 채팅 서버의 소켓 주소 구조체 servaddr 초기화
  bzero((char*)&servaddr, sizeof(servaddr));
  servaddr.sin_family = af;
  inet_pton(AF_INET, servip, &servaddr.sin_addr);
  servaddr.sin_port = htons(port);
  //④ 연결 요청
  if(connect(s, (struct sockaddr*)&servaddr, sizeof(servaddr)) < 0)
    return -1;
  return s;
}
```

## 2.2. 서버 메시지 수신 및 출력

- 키보드로부터 문자열을 입력 받아 'hello'를 입력할 경우, 서버로부터 최초 접속 시간을 수신 및 출력한다. 그리고 'exit'를 입력할 경우, 서버로부터 접속을 끊는다. 또한, 다른 내용을 입력할 경우엔, 일반 채팅처럼 해당 내용을 출력한다.




### 2.2.1. 소켓의 I/O 변화 감지 및 메시지 출력

- 비동기 모드로 select() 시스템 콜을 사용하여 모든 소켓의 I/O 변화를 감지한다. 이 때, select()를 호출하기 위해 fd_set 구조체를 만들고 읽기 변화 감지용 read_fds를 초기화한 후 I/O 변황 관심이 있는 소켓번호를 선택한다. while 루프를 무한으로 실행하면서 I/O 변화를 감지할 경우, 서버로부터 메시지를 수신하여 이를 출력한다.

```
        //① select()를 호출하기 위해 fd_set 구조체를 만든다.
        //② read_fds의 모든 비트를 0으로 초기화한다.
        FD_ZERO(&read_fds);
```

```
        while(1) {
          //③ I/O 변화에 관심이 있는 모든 연결용 소켓을 선택한다.
          FD_SET(0, &read_fds); // 클라이언트
          FD_SET(s, &read_fds); // 서버
          if(select(maxfdp1, &read_fds, NULL, NULL, NULL) < 0)
            errquit("Select Failed");
          if(FD_ISSET(s, &read_fds)) {
            int nbyte;
            //① 서버로부터 수신한 내용을 출력한다.
            if((nbyte = recv(s, bufmsg, MAXLINE, 0)) > 0) {
              bufmsg[nbyte] = 0;
              printf("Time: %s", bufmsg);
            }
          }
          if(FD_ISSET(0, &read_fds)) {
            //② 키보드로부터 문자열을 입력 받는다.
            if(fgets(bufmsg, MAXLINE, stdin)) {
              if(send(s, bufmsg, strlen(bufmsg), 0) < 0)
                puts("Error : Write error on socket");
              //③ 'exit' 입력 시, 서버 접속을 종료한다.
              if(strstr(bufmsg, EXIT_STRING) != NULL) {
                puts("Good bye.");
                close(s);
                exit(0);
              }
            }
          }
        }
```

# Ⅲ. 결론

- C에서 제공해주는 소켓 라이브러리 함수를 활용하여 TCP 통신의 채팅 기반 서버와 클라이언트를 작성하였다. 서론에서 언급했던 조건을 모두 충족시킬 수 있도록 프로그래밍하였고, 모두 정상 작동하는 것을 확인하였다. 조건 충족 내용은 다음과 같다.

1. 서버와 클라이언트 간의 TCP 통신의 채팅 기반 프로그램을 작성하였다.
2. 클라이언트가 서버로 접속하였을 때, 곧바로 현재의 총 클라이언트 수, 각 클라이언트의 이름, IP, 최초 접속시간을 서버 내에 출력하도록 하였다.
3. 클라이언트를 최소 5개 이상 접속하도록 하였다. 클라이언트를 유지하는 구조체 변수의 크기를 10으로 하였기 때문에 최대 10개의 클라이언트를 유지할 수 있다.
4. 서버 프로세스가 'hello' 메시지를 받으면 모든 클라이언트에게 서버의 최초 접속시간을 전송하도록 하였다. 그리고 각각의 클라이언트 화면에 시간을 출력하였다.
5. 클라이언트가 'exit' 메시지를 송신하면 해당 클라이언트는 서버로의 접속을 끊고, 서버는 클라이언트 목록에서 해당 클라이언트를 삭제하며 이를 공고한다.

# IV. 부록

## 1. 고찰

- 이번 중간고사 대체 과제의 목표는 TCP 통신의 채팅 서비스를 기반으로 서버와 클라이언트 간의 통신을 실습하는 것이었다. 따라서 채팅에 참여한 클라이언트가 글을 입력했을 때, 내용을 서버로 송신하고 모든 클라이언트로 브로드캐스팅 하도록 하였다. 뿐만 아니라, 서버가 'hello'라는 특정 메시지를 받을 경우 각 클라이언트가 접속한 시간을 전송하도록 하였고, 클라이언트가 'exit'라는 특정 메시지를 입력할 경우 서버로의 접속을 끊도록 처리 하였다. 그 결과, 간단한 채팅 프로그램을 작성할 수 있었다.

과제를 수행하면서 가장 어려웠던 점은, 고려해야 할 예외가 많았다는 점이다. 예를 들어, 한 클라이언트가 FTP 서버에 접속하여 'get' 명령어를 입력하고 파일을 다운 받을 경우, 먼저 해당 파일의 존재 여부를 고려하고 해당 파일이 존재하면 다운을 받고 그렇지 않으면, 서버는 예외 메시지를 송신하고 클라이언트는 예외 메시지를 수신해야 한다. 이렇듯, 여러 예외를 처리해주는 데 많은 시간을 쏟아 부었다.

하지만, 이러한 문제에 대하여 전공 서적을 참고하고 검색을 통해 방법을 찾아 적용하여 해결할 수 있었다. 한 학기 동안 네트워크 프로그래밍을 수강하면서 많은 것을 배웠는데, 매번 과제 때마다 문자열 함수를 굉장히 많이 사용한 덕분에 C언어가 가지고 있는 문자열 함수들을 다루는데 능숙해질 수 있었다. 특히, 서버와 클라이언트 간 TCP 통신을 이루기 위해 사용하는 bind(), listen(), accept(), socket(), connect() 등 다양한 C 언어의 소켓 라이브러리 함수에 대해 완벽히 이해할 수 있었다. 또한, TCP에 국한되지 않고 UDP도 사용해보고 시그널, 프로세스 등 다양하게 학습하고 실습할 수 있어서 좋은 기회가 되었다.

컴퓨터공학 학생으로서 이 경험은 학교 밖으로 나가 취업 후에도 분명 큰 힘이 될 것이라고 생각한다.

## 2. 참고 자료


| 참고한 내용 | 사이트 주소 또는 서적 |
|---|---|
| C 소켓 라이브러리 함수 | http://forum.falinux.com/zbxe/index.php?mid=C_LIB&page=2 |
| | http://rotapple.tistory.com/6 |
| 소켓 프로그래밍 이해 | 김화종(2004).컴퓨터 네트워크 프로그래밍. 홍릉과학 출판사 |