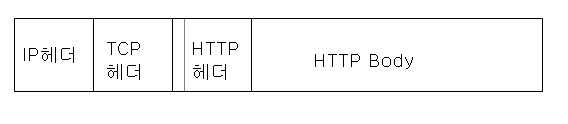
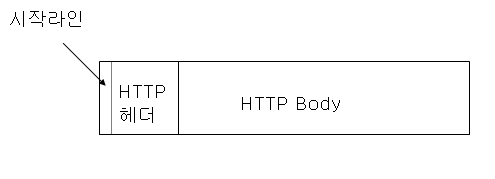
1. **프로토콜**
   1. 정의

* 컴퓨터간에 정보를 주고받을 때의 통신 방법에 대한 규칙과 약속(네이버 지식백과)
* 정보기기 사이 즉 컴퓨터끼리 또는 컴퓨터와 단말기 사이 등에서 정보교환이 필요한 경우, 이를 원활하게 하기 위하여 정한 여러 가지 통신 규칙과 방법에 대한 약속 즉, 통신의 규약을 의미한다.
* 통신규약이라 함은 상호간의 접속이나 전달방식, 통신방식, 주고받을 자료의 형식, 오류검출방식, 코드변환방식, 전송속도 등에 대하여 정하는 것을 말한다. 일반적으로 기종(機種)이 다른 컴퓨터는 통신규약도 다르기 때문에, 기종이 다른 컴퓨터간에 정보통신을 하려면 표준 프로토콜을 설정하여 각각 이를 채택하여 [통신망](https://terms.naver.com/entry.nhn?docId=1153390&ref=y)을 구축해야 한다.
  1. 자신이 사용해본 프로토콜  
     (컴퓨터 기반 아닌 소통하기 위한 절차가 있으면 모든 가능함)
  2. 우리가 배운 프로토콜 종류
* 2계층 프로토콜: Ethernet (CSMA/CD)
* 3계층 프로토콜: IP
* 4계층 프로토콜: TCP/UDP
* 차이점 (조별 활동, 5분)
  + 목적
  + 프레임/패킷/세그먼트 포맷
  + 그 외
* 공통점 (조별 활동, 5분)
  + 목적
  + 프레임/패킷/세그먼트 포맷
  + 데이터 표현 방식
  + 그 외
  1. 응용 계층 프로토콜
* 종류
  + HTTP (Hyper Text Transfer Protocol)
  + FTP (File Transfer Protocol)
  + Telnet (Tele network)
  + SMTP(E-mail) (Simple Mail Transfer Protocol)
  + POP (Post Office Protocol)
  1. HTTP (Hyper Transfer Transmit Protocol)
* 정의: TCP/IP 프로토콜 상에서 텍스트 형태의 메시지를 웹 서버와 클라이언트 사이에서 서로 주고받는 방식으로 동작한다.
* 프레임 포맷
  + TCP/IP 사용



* + 메시지 포맷 = 시작 라인 + 헤더 + 바디 로 구성

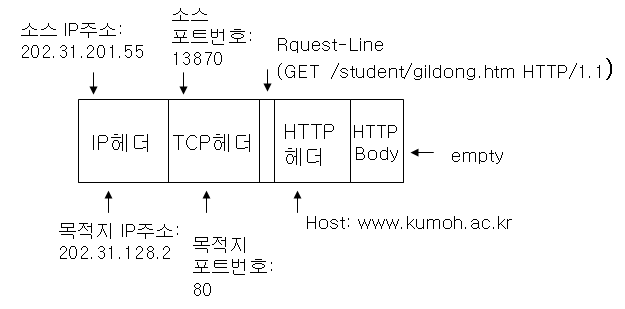


* + - 시작라인: 요청과 응답으로 나뉨 🡺 클라이언트/서버 통신

|  |
| --- |
| 요청 시(request line) 메소드 종류 |
|  |
| 응답 시(status line) 응답 상태 코드 종류 |
|  |

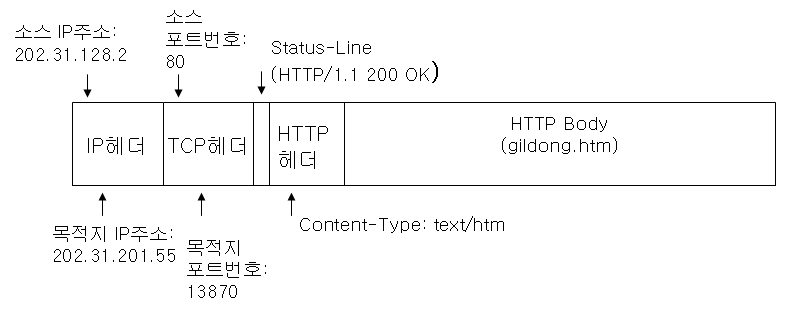
* + - 헤더:
      * + 헤더 필드명 + 값의 형태로,
        + 한 줄에 하나씩, 한 줄 끝나면 줄 바꿈
        + 헤더는 여러 개 올 수 있음
        + 헤더가 끝나면 빈 줄을 하나 넣고 바디 시작
        + 헤더 종류 조사
    - 바디
  1. 동작 과정
* HTTP 요청 ( 🡪 )

클라이언트 IP 주소인 소스 IP 주소는 202.31.201.55, 소스 포트번호는 13870이 들어있다. HTTP 바디(body) 부분은 비어있다

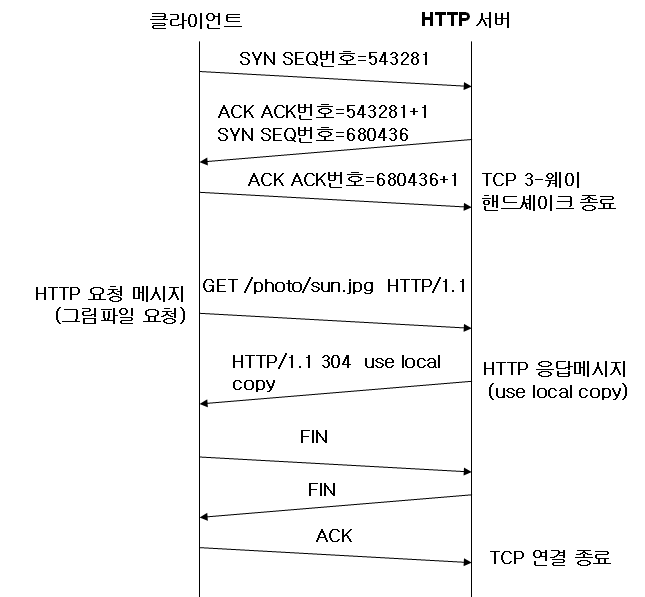


* HTTP 응답 ( 🡪 )

소스 IP 주소는 서버의 주소인 202.31.128.2, 소스 포트번호는 80, 목적지 IP 주소는 202.31.201.55, 목적지 포트번호는 13870이 각각 들어가고. HTTP 바디 부분에는 웹문서(gildong.htm)가 들어간다.



* 캐쉬 동작

다음 그림은 html 문서를 서버로부터 전송 받는 도중에 그림이 포함되어 있어 새로운 HTTP 연결을 설정한 경우, 그림 파일 요청에 대해 클라이언트 내부 캐시를 사용하라는 응답(use local copy 또는 Not Modified)을 서버로부터 받은 경우에 오가는 패킷의 동작을 보였다.

(html 문서에서 그림은

html <img> 태그를 사용해

<img src = /photo/sun.jpg>

식으로 파일 위치를 나타낸다.)

* 1. 과제: GET과 POST, PUT의 특징 및 차이 조사하기

1. **HTTPS**
   1. Hyper Text Transfer Protocol Secure

* Hyper Text Transfer Protocol over Secure Socket Layer, HTTP over TLS, HTTP over SLS
  + TLS나 SLS로 암호화(전송 계층 암호화): 공개키, 비밀키 기반 암호화
  + Port: 443
  + 속도: HTTP > HTTPS
  + 중간 변조 불가능, 네트워크 상에서 열람 수정 불가능
  + 서버 측: 인증서 유지 비용 추가, HTTP보다 트래픽 발생 많음
  1. SSL (Secure Socket Layer) 와 TLS (Transport Layer Security)
* SSL (표준화 이전 이름) = TLS (표준화 이름)
* 서버와 클라이언트 간에 인증(Certification)으로 RSA 방식과 X.509를 사용하고 암호화된 정보 암호화 소켓 채널을 통해 전송
* 신분 확인(Authentication), 암호화, 메시지의 무결성을 보장
* 절차

1) 지원 가능한 알고리즘 서로 교환 🡪 2)에서 사용할 암호화 방법, MAC 결정

2) 키 교환, 인증 🡪 공개키(RSA) 또는 TLS-PSK(pre-shared key cipher suites) 사용, RSA, ECDSA

3) 대칭키 암호로 암호화하고 메시지 인증 🡪 RC4, 3DES, AES, IDEA 등

% DTLS (Datagram TLS) DTLS 1.2: TLS 1.2 기반

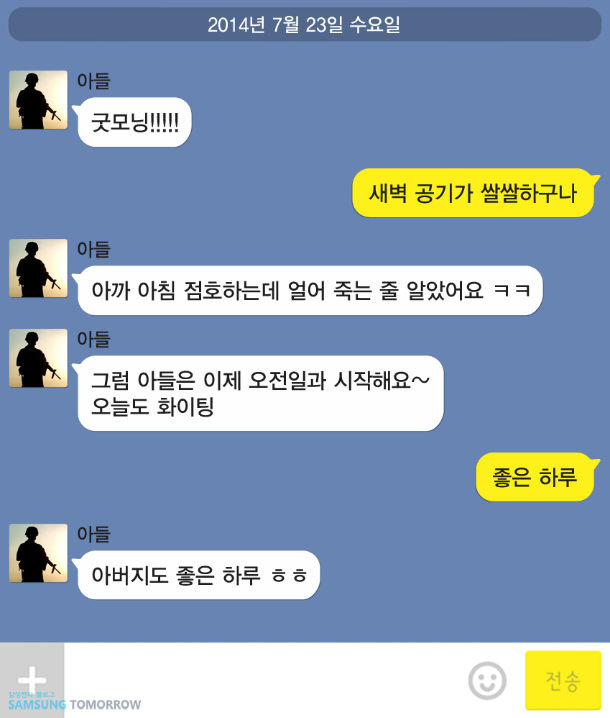
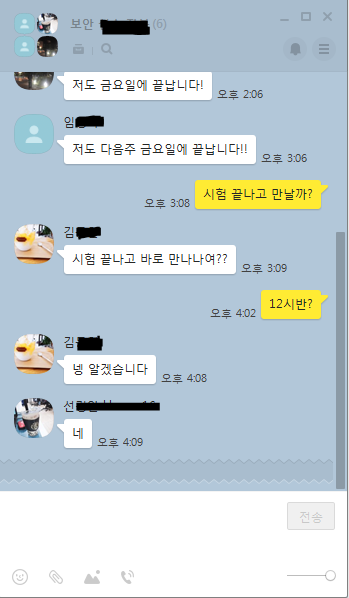
* 1. HTTP 와 VPN

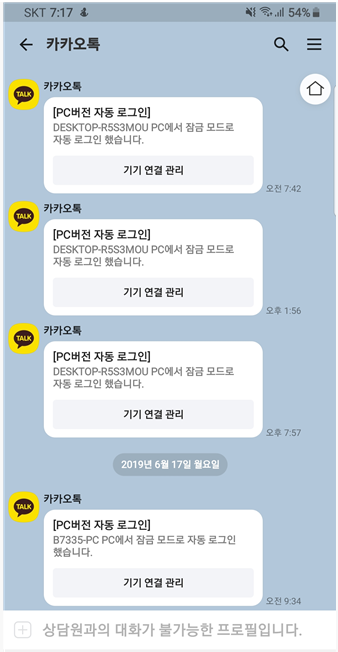
🡺 

HTTP HTTP over VPN

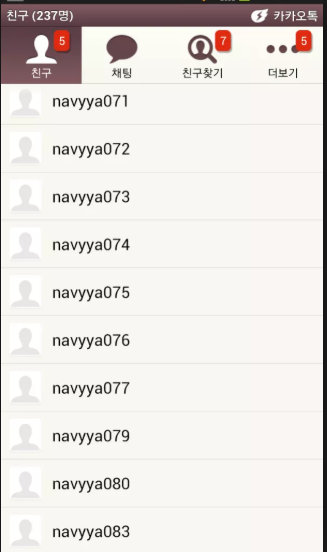
* HTTP over VPN:
  + 느린 속도: VPN을 경유하여 인터넷을 사용  
    (1) 내 컴퓨터에서 인터넷에 요청할 정보를 암호화하여 VPN에 보냅니다.  
    (2) VPN에서는 전달받은 내용을 복호화 하여 인터넷에 정보를 보냅니다.  
    (3) 인터넷에서의 결과를 VPN에 전달합니다.  
    (4) VPN에서 정보를 암호화하여 내 컴퓨터에 전달합니다.
  + 익명성 보장: 인터넷 서버에게 원 사용자는 숨길 수 있다.
  + VPN 서버는 내 정보를 모두 알 수 있다.

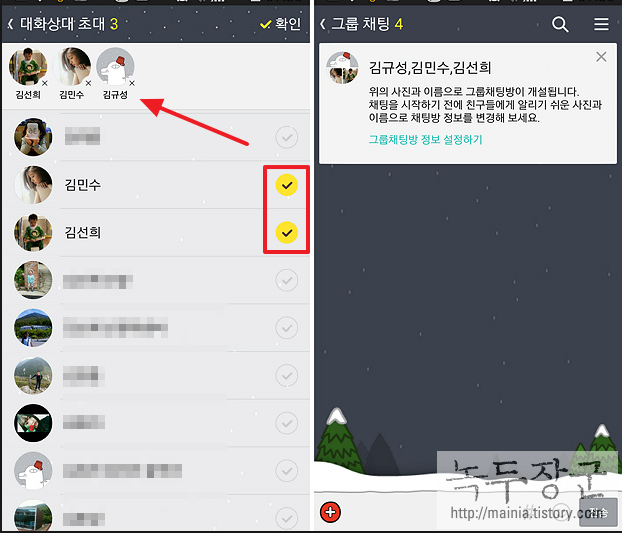
패스워드 읽힐 수 있음, 자동 로그인 사이트는 쿠키 통해 로그인 탈취 가능, 보안 사이트 접근도 읽힐 수 있음

* 1. 팀 활동: 프로토콜 만들어보기
* 일대일 채팅의 프로토콜: 다음과 같이 닉네임 ‘아들’과 ‘아버지’가 채팅한다고 가정할 때 필요한 기능과 프로토콜을 만들어보자.
  + - * + 아들이 아버지에게 보내는 메시지
        + 아버지가 아들에게 보내는 메시지
        + 일대일 프로토콜
* 일대다 채팅(그룹 채팅)의 프로토콜: 다음과 같이 6명의 구성원(C1~C6)이 채팅한다고 가정할 때 필요한 기능과 프로토콜을 만들어보자.
  + - * + 
* 서버 공지 프로토콜: 다음과 같이 서버가 모든 클라이언트에게 또는 한 클라이언트에게 공지를 보내는 프로토콜을 만들어보자.



* 친구 관리 프로토콜: 다음과 같이 서버가 클라이언트에게 친구목록을 보내는 프로토콜을 만들어보자.



* 가입 또는 로그인 프로토콜: 다음과 같이 서버가 클라이언트에게 친구목록을 보내는 프로토콜을 만들어보자.
  + - * + 가입에 필요한 정보는 무엇일까?
        + 클라이언트가 서버에게 어떤 메시지를 보낼까?
        + 서버는 클라이언트의 요청에 어떤 메시지를 보낼까?
* 그룹채팅을 위한 방 만들기 프로토콜: 다음과 같이 서버가 클라이언트에게 친구목록을 보내는 프로토콜을 만들어보자.
  + - * + 방 이름 정하기:   
          클라이언트가 서버에게 방 이름 알릴 메시지 보내기  
          서버는 이에 대한 확인 메시지 보내기
        + 친구초청  
          클라이언트가 서버에게 그룹채팅할 구성원 보내기  
          서버는 이에 대한 확인 메시지 보내고 구성원들에게 그룹 채팅할 수 있도록 초청한다.

1. **UNIX C 소켓** 
   1. 정의

* 소켓이란: 응용프로그램을 개발할 때 TCP/UDP 또는 IP(raw socket)를 이용하여 프로그램을 개발 할 수 있도록 지원
* 유닉스/리눅스에서의 소켓 인터페이스: 파일 입출력(I/O)과 유사한 구조

유닉스/리눅스에서 다음과 같이 파일을 오픈하면

fd = open("sample.txt", O\_RDONLY)

양의 정수값을 리턴하는데 이를 파일 디스크립터(file descriptor, 일종의 포인터)라고 하고 프로그램에서 이 오픈된 파일을 액세스할 때 이 파일 디스크립터를 사용

read(fd, buf, 512)

* 소켓 디스트립터

sd = socket(AF\_INET, SOCK\_STREAM, 0)

* 소켓 함수 포맷

int socket(int family, int type, int protocol)

AF\_INET: AF\_INET (인터넷 주소 체계) 🡪 주로 사용

AF\_UNIX (유닉스 주소 체계)

AF\_INET6 (128비트 IPv6 주소 체계) 🡪 IPv6일 때 사용

Type: 서비스 타입(type of service)을 의미  
연결형(stream) 서비스를 위해서는 SOCK\_STREAM을,   
비연결형(datagram) 서비스를 위해서는 SOCK\_DGRAM을 사용

Protocol: 소켓 지원 프로그램 지정, 보통 0 사용

* 소켓 프로그램에서는 소켓주소를 담을 구조체인 sockaddr를 아래와 같이 정의

struct sockaddr {

u\_short sa\_family; /\* address family \*/

char sa\_data[14]; /\* address \*/

};

struct sockaddr\_in {

short sin\_family; /\* 주소 패밀리 \*/

u\_short sin\_port; /\* 16비트 포트번호 \*/

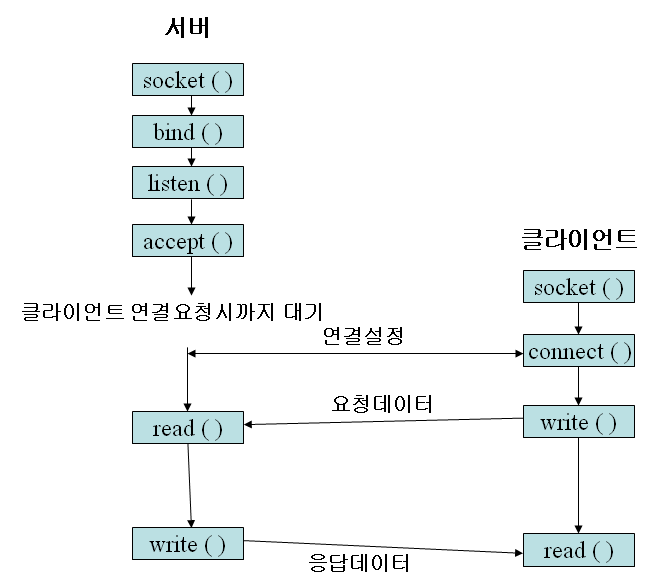
struct in\_addr sin\_addr; /\* 32비트 IP 주소\*/

char sin\_zero[8]; /\* 더미 문자열: 전체 크기를 sockaddr 구조체와 \*/

/\* 동일한 16바이트로 하기 위한 \*/

};

* 소켓을 이용한 네트워크 연결 절차



* + 클라이언트

1. 클라이언트는 다음의 socket() 함수를 호출하여 정수값인 소켓 디스크립터(소켓 번호)를 얻는다.

int socket(int family, int type, int protocol)

이때 함수 파라미터 family를 AF\_INET, type은 SOCK\_STREAM, protocol은 0으로 각각 설정한다. TCP/IP 네트워크 프로그램에서는 socket() 함수 호출시 함수 파라미터 type 부분만 SOCK\_STREAM(TCP 사용 시) 또는 SOCK\_DGRAM(UDP 사용 시)으로 하고 나머지는 같다.

2) 연결하고자 하는 서버의 IP 주소와 포트번호를 소켓주소 구조체(sockaddr\_in)에 지정한다.

3) 다음과 같이 connect() 함수를 호출한다. 이때 위에서 만든 소켓주소 구조체를 사용한다.

int connect (int s, const struct sockaddr \*addr, int addrlen)

여기서 s는 socket() 함수에서 얻은 소켓 디스크립터이고, addr은 2)번에서 만든 상대방 서버의 소켓주소 구조체의 포인터이며 addrlen은 이 구조체(\*addr)의 길이이다.

4) 앞의 connect() 함수를 통해 소켓주소에 지정된 서버의 IP 주소와 포트번호로 연결을 시도하여 연결에 성공하면 0이 리턴된다.

5) 클라이언트가 서버와 연결되면 send(), recv() 함수를 사용하여 서버와 데이터를 송수신할 수 있다. 또는 write(), read() 함수도 사용 가능하다. 다음에 send() 함수의 포맷을 보였다.

int send(int s, char\* buf, int length, int flags)

여기서 s는 최초의 socket() 함수에서 리턴된 소켓 디스크립터이고, buf는 클라이언트가 송신할 데이터가 저장된 버퍼이며, length는 buf의 길이이고 flags는 보통 0으로 둔다. 리턴되는 값은 실제로 송신된 데이터 크기(바이트 단위)이다.

6) 연결을 종료하려면 다음의 close() 함수를 호출한다. 이 함수는 송신 버퍼에서 아직 전송하지 못한 데이터나 전달중인 데이터들을 모두 처리한 후에 소켓을 닫는다.

close(int s)

여기서 s는 socket() 함수에서 리턴된 소켓 디스크립터이다.

* + 서버

1) 서버 프로그램도 클라이언트 프로그램과 동일하게 socket() 함수를 호출하여 리턴값으로 정수값인 소켓 디스크립터(소켓 번호)를 얻는다.

int socket(int family, int type, int protocol)

이때 파라미터 family는 AF\_INET, type은 SOCK\_STREAM(TCP 프로토콜을 사용하므로), protocol은 0을 각각 넣는다.

2) bind() 함수를 사용하여 위의 socket() 함수를 통해서 리턴된 소켓 디스크립터와 TCP/IP 시스템이 제공하는 소켓 주소(32비트 IP 주소, 16비트 포트번호)를 연결한다. bind() 함수의 포맷은 다음과 같다.

int bind (int s, struct sockaddr \*addr, int len)

여기서 s는 socket() 함수에서 리턴된 소켓 디스크립터이고 addr은 서버 소켓주소 구조체의 포인터이며 len은 \*addr 구조체의 길이이다. 이 함수는 성공하면 0을, 실패하면 -1을 리턴한다. \

3) 다음 서버는 클라이언트로부터의 연결 요청을 받아들이기 위하여 listen() 함수를 호출한다. 이 함수는 소켓을 단지 수동 대기모드로 바꾸는 것이므로 이 함수의 호출은 즉시 리턴되며 이때 리턴되는 값은 성공 시에는 0, 실패 시에는 -1이다. listen() 함수의 사용 포맷을 보였다.

int listen (int s, int log)

여기서 파라미터 s는 socket() 함수에서 리턴된 소켓 디스크립터(소켓 번호)이고 log는 연결을 기다리는 클라이언트의 최대수를 나타낸다.

4) 다음 서버는 accept() 함수를 호출한다. 이 함수는 클라이언트의 연결요청이 있을 때까지 리턴되지 않고 대기(wait)상태에 있다가 클라이언트의 연결요청이 있을 때 이를 처리하게 된다. accept() 함수의 호출이 성공한 경우에는 연결된 상대방 클라이언트와 서버사이의 통신에 사용할 새로운 소켓이 만들어지고 이 소켓 디스크립터 값을 리턴하며 실패 시에는 -1을 리턴한다.

int accept (int s, struct sockaddr \*addr, int \*addrlen)

여기서 s는 앞의 1)번에서 서버의 socket() 함수에서 리턴된 소켓 디스크립터이고 addr은 서버에 연결 요청을 한 상대방 클라이언트의 소켓주소 구조체를 가리키는 포인터이다.

5) 이와 같이 하여 서버가 클라이언트와 연결되면 send(), recv() 함수(또는 write(), read() 함수)를 사용하여 클라이언트와 데이터를 송수신할 수 있다.

이들 함수에 사용할 소켓 디스크립터는 1)번의 socket() 함수에서 얻은 소켓 디스크립터가 아니라 4)번의 accept() 함수에서 리턴된 새로운 소켓 디스크립터 이다.

6) 소켓을 닫을 때 클라이언트의 경우와 같이 close() 함수를 호출한다. 이때 송신 버퍼에서 아직 전송하지 못한 데이터나 전달중인 데이터들을 모두 처리한 후에 소켓을 닫는다.

**\*\*과제**

1. 다음 서버코드와 클라이언트 코드를 보고 소켓 통신에 관련된 코드 즉, 소켓 생성 socket(), bind(), listen(), accept() connect() 부분에 형광펜으로 표시하시오.
2. 생성할 소켓의 속성을 지정하는 부분을 박스로 표시하시오.
3. 부모 프로세스가 하는 부분과 자식 프로세스가 하는 부분을 구분하시오.
4. 부모 프로세스가 하는 일과 자식 프로세스가 하는 일을 쓰시오.

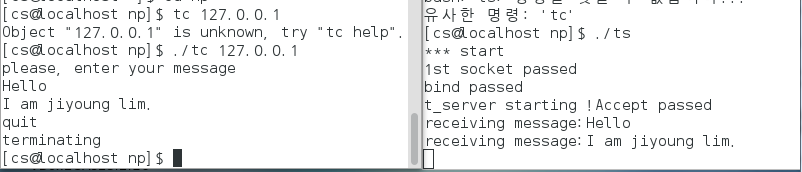
* tServer.c: 클라이언트가 전송한 문자열을 그대로 클라이언트로 재전송하는 서버

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <signal.h>  #include <string.h>  #define PORT 4000  #define END "quit"  int main(int argc, char \*argvi[]){  struct sockaddr\_in addr\_svr, addr\_cl;  int sock\_svr, sock\_cl;  int proc\_id, addr\_cl\_size, str\_len;  char msg\_send[256], msg\_recv[256];  printf("\*\*\* start\n");  if ((sock\_svr = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {  printf("TCP socket creation failure !");  return -1;  }  printf("1st socket passed\n");  bzero((char \*)&addr\_svr, sizeof(addr\_svr));  addr\_svr.sin\_family = AF\_INET;  addr\_svr.sin\_addr.s\_addr = htonl(INADDR\_ANY);  addr\_svr.sin\_port = htons(PORT);  if (bind(sock\_svr, (struct sockaddr \*)&addr\_svr, sizeof(addr\_svr))<0){  printf("binding error!");  return -1;  }  printf("bind passed\n");  listen(sock\_svr, 10);  printf("t\_server starting !\n");  addr\_cl\_size = sizeof(addr\_cl);  if ((sock\_cl = accept(sock\_svr, (struct sockaddr \*)&addr\_cl, &addr\_cl\_size))<0){  printf("accept failure!\n");  return -1;  }  printf("Accept passed \n");  if ((proc\_id = fork()) >0) {  //parent process  while (fgets(msg\_send, 256, stdin) != NULL) {  str\_len = strlen(msg\_send);  if (write(sock\_cl, msg\_send, str\_len) <0)  printf(" writing failure ");  if (strstr(msg\_send, END) != NULL) {  printf(" terminating ");  close(sock\_cl);  return -1;  }  }  } else if (proc\_id ==0) {  while(1){  if ((str\_len = read(sock\_cl, msg\_recv, 256)) <0) {  printf("reading error");  close(sock\_cl);  return -1;  }  msg\_recv[str\_len] = '\0';  if (strstr(msg\_recv, END) != NULL) break;  printf("receiving message:%s", msg\_recv);  }  }  close(sock\_svr);  close(sock\_cl);  } |

* tClient.c: 클라이언트에서 입력한 문자열을 서버에 전송하고 서버로부터 문자열을 받는 클라이언트

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <sys/types.h>  #include <signal.h>  #include <string.h>  #define PORT 4000  #define END "quit"  int main (int argc, char \*argv[]){  struct sockaddr\_in addr\_svr;    char msg\_send[256], msg\_recv[256];  int proc\_id, sock\_cl, str\_len;    if (argc != 2) {  printf("execution method: %s server ip address\n", argv[0]);  return -1;  }  if ((sock\_cl = socket(AF\_INET, SOCK\_STREAM, 0) ) < 0) {  printf("TCP socket creation failure \n");  return -1;  }  bzero((char \*)&addr\_svr, sizeof(addr\_svr));  addr\_svr.sin\_family = AF\_INET;  addr\_svr.sin\_addr.s\_addr = inet\_addr(argv[1]);  addr\_svr.sin\_port = htons(PORT);  if (connect(sock\_cl, (struct sockaddr \*)&addr\_svr, sizeof(addr\_svr))<0){  printf("connection failure \n");  return -1;  }  printf("please, enter your message \n");  if ((proc\_id = fork () ) > 0) {  //parent  while (fgets(msg\_send, 256, stdin) != NULL) {  str\_len = strlen(msg\_send);  if (write(sock\_cl, msg\_send, strlen(msg\_send)) < 0)  printf("writing failure\n");  if (strstr(msg\_send, END) != NULL ) {  printf("terminating\n");  close(sock\_cl);  return -1;  }  }  } else if (proc\_id ==0) {  while (1) {  if ((str\_len = read(sock\_cl, msg\_recv, 256)) < 0) {  printf("reading error \n");  close(sock\_cl);  return -1;  }  msg\_recv[str\_len] = '\0';  if (strstr(msg\_recv, END) != NULL ) break;  printf(" receiving message %s", msg\_recv);  }  }  close(sock\_cl);  } |

실행화면



1. **C# 스레드 델리게이트 이벤트** 
   1. 스레드 (C#으로 배우는 네트워크 프로그래밍 1장 참조)

* 스레드 사용 이유
  + C# 프로그램에서 동시 처리 작업 수행 가능
  + 응용 프로그램 리소스 공유
  + .Net Framework의 System.Threading 네임스페이스를 사용
* 스레드 생성과 생성
  + Thread WorkThread = new Thread(DoWork);
  + WorkThread.Start();
* 스레드 종료
  + WorkThread.Abort() // 강제 종료
  + WorkThread.Join(); // 완료되는 확인하고 리소스 정리한 후 종료
  1. 예제: ThreadLife 프로그램
  + Main() 함수가 수행하는 부분
  + LifeThread가 수행하는 부분
  + 실행결과:

스레드 종료 시점

* 1. 예제: 프로그램 간 데이터 전달과 수신
  + without Thread

using System.Diagnostics;

bool Runflags = true;

//버튼 클릭 시 스레드 없이 데이터 전달

if (porc.ProcessName.ToString() == “mook\_sendData”)

* + with Thread

using System.Diagnostics;

using System.Threading;

//버튼 클릭 시 스레드 생성

RaceThread = new Thread(Receivedata);

RaceThread.Start();

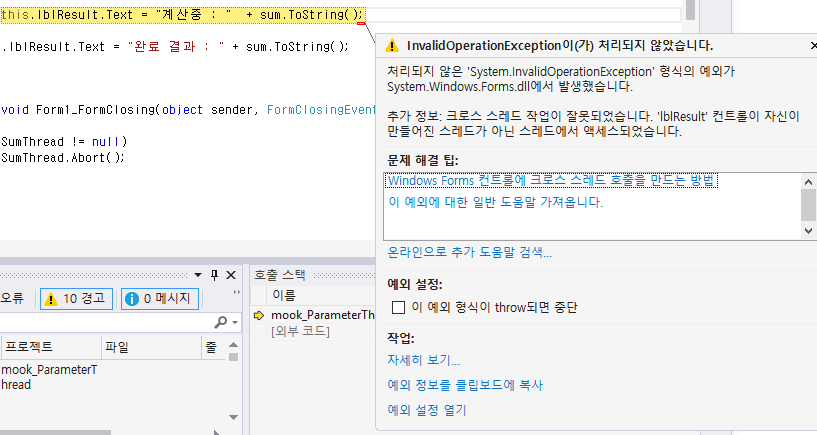
//스래드가 수행할 함수에서 데이터 전달

if (porc.ProcessName.ToString() == “mook\_sendData”)

* + 두 방식의 차이
  1. 대리자 (delegate)
* 메서드를 다른 메서드의 인수로 전달 가능
* 메소드를 사용하는 새로운 종류의 메서드 타입
* 주로 이벤트 처리할 때 유용
* 특징
  + C++의 함소 포인터와 유사, 안전한 형식
  + 대리자를 통해 메서드를 매개변수로 전달가능
  + 대리자를 사용하여 콜백 메서드 정의 가능
  + 여러 대리자 연결 가능
* 예제: mook\_ParameterThread: 대리자 없이 입력받은 숫자를 스레드에서 0부터 더하는 프로그램

|  |
| --- |
| Thread SumThread = null;  private void btnSum\_Click(object sender, EventArgs e)  {  SumThread = new Thread(new ParameterizedThreadStart(NumSum));  SumThread.Start(this.txtNum.Text);  }  private void NumSum(object n)  {  long sum = 0;  long k = Convert.ToInt64(n);  for (long i = 0; i <= k; i++)  {  Thread.Sleep(1);  sum += i;    this.lblResult.Text = "계산중 : " + sum.ToString();  }  this.lblResult.Text = "완료 결과 : " + sum.ToString();  } |

* + 실행 결과
    - 실행 시 발생하는 문제?



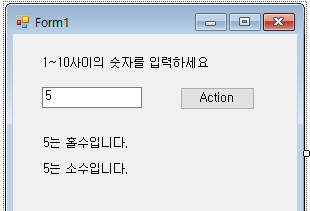
* + - 작업자 스레드에서 주 스레드에서 정의된 윈도우 컨트롤 직접 참조하는 경우 발생하는 에러

lblResult를 만든 스레드:

lblResult.Text에 문자열을 수정하는 스레드:

* 스레드 생성 비교
  + RaceThread = new **Thread**(Receivedata);
  + SumThread = new Thread(new **ParameterizedThreadStart**(**NumSum**));  
    SumThread.Start(**this.txtNum.Text**);
* 예제: mook\_DelegateParameterThread: 대리자 사용하여 입력받은 숫자를 스레드에서 0부터 더하는 프로그램

|  |
| --- |
| Thread SumThread = null;  **private delegate void OnResultDelegate(string strText); //델리게이트 선언**  **private OnResultDelegate ResultView = null; //델리게이트 개체 생성**  private void btnSum\_Click(object sender, EventArgs e)  {  SumThread = new Thread(new ParameterizedThreadStart(NumSum));  SumThread.Start(this.txtNum.Text);  }  private void NumSum(object n)  {  long sum = 0;  long k = Convert.ToInt64(n);  for (long i = 0; i <= k; i++)  {  Thread.Sleep(1);  sum += i;  **Invoke(ResultView, "계산중 : " + sum.ToString());**  }  **Invoke(ResultView, "완료 결과 : " + sum.ToString());**  SumThread.Abort();  }  private void Form1\_Load(object sender, EventArgs e)  {  **ResultView = new OnResultDelegate(ResultSum);**  }  private void **ResultSum**(string NumSum)  {  this.lblResult.Text = NumSum;  } |

% 과제: 1~10의 수를 입력받아 Action 버튼을 누르면 그림과 같이 결과를 홀수인지 짝수인지 출력하는 레이블과 소수인지 아닌지를 출력하는 레이블에 출력하시오.

* 1. 이벤트 (Event)
* 이벤트:
* 게시자: 이벤트를 보내거나 발생시킴
* 구독자: 이벤트를 받거나 처리하는 일
* 대리자 또는 인보크와 비슷
* 예제: mook\_EventNumCatch: 콘솔프로그램으로 1~10까지의 숫자에 대해 짝수(2, 4)와 3의 배수(6, 9)를 판단하는 프로그램

|  |
| --- |
| **// 이벤트가 발생하면 호출되는 메서드**  static void Catch1(object sender, EventArgs e)  {  Console.Write("짝수 : ");  }  static void Catch2(object sender, EventArgs e)  {  Console.Write("3의 배수 : ");  } |

|  |
| --- |
| **// 게시자 클래스 정의**  **class CatchClass**  {  **//이벤트 발생하면 이벤트를 처리할 메소드 타입정의**  **public delegate void OnEventHandler(object sender, EventArgs e);**  **// 이벤트 정의**  **public event OnEventHandler NumCatch**;  public void GoEvent()  {  if (NumCatch != null) //이벤트 구독자 있는지 검사  {  EventArgs e = new EventArgs();  NumCatch(this, e); //NumCatch 실행  }  }  } |

|  |
| --- |
| static void Main(string[] args)  {  // 게시자 클래스 객체 생성  **CatchClass Ctc = new CatchClass();**  **// 이벤트가 발생하면 Catch1을 호출**  Ctc.NumCatch += new CatchClass.OnEventHandler(Catch1);  for (int i = 1; i < 10; ++i)  {  if (i % 2 == 0 && i < 5)  {  **Ctc.GoEvent();** //이벤트 발생, NumCatch에 등록한 이벤트핸들러 수행  }  if (i == 5)  {  **Ctc.NumCatch -= new CatchClass.OnEventHandler(Catch1);**  **Ctc.NumCatch += new CatchClass.OnEventHandler(Catch2);**  }  if (i % 3 == 0 && i > 5)  {  **Ctc.GoEvent();** //이벤트 발생, NumCatch에 등록한 이벤트핸들러 수행  }  Console.WriteLine("{0}", i);  }  }  } |

* + 실행결과
* 예제: mook\_EventNumCatch: 콘솔프로그램으로 1~10까지의 숫자에 대해 짝수(2, 4)와 3의 배수(6, 9)를 판단하는 프로그램

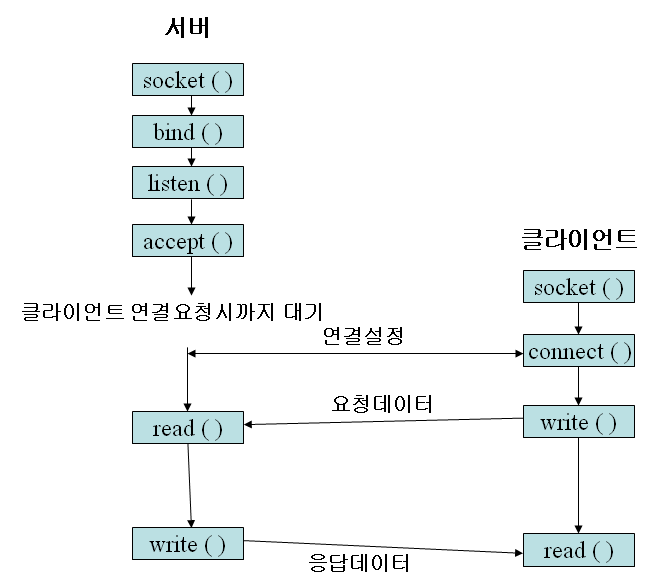
|  |
| --- |
| **private delegate void OnDelegateHeight(int Flag);**  **private OnDelegateHeight OnHeight = null;**  private void Form1\_Load(object sender, EventArgs e)  {  **OnHeight = new OnDelegateHeight(MsgView**);  this.Size = new System.Drawing.Size(170, 0);  this.Location =  new System.Drawing.Point(Screen.PrimaryScreen.WorkingArea.Width  - this.Width - 20, Screen.PrimaryScreen.WorkingArea.Height - this.Height);  TimerEvent = new System.Timers.Timer(2); //millisecond  **TimerEvent.Elapsed += new ElapsedEventHandler(OnPopUp);**  TimerEvent.Start();  } |

|  |
| --- |
| private void **OnPopUp**(object sender, ElapsedEventArgs e)  {  if (Height < 120)  {  Invoke(OnHeight, 0);  }  else  {  TimerEvent.Stop();  TimerEvent.Elapsed -= new ElapsedEventHandler(OnPopUp);  TimerEvent.Elapsed += new ElapsedEventHandler(OnPopOut);  TimerEvent.Interval = 3000;  TimerEvent.Start();  }  Application.DoEvents();  } |

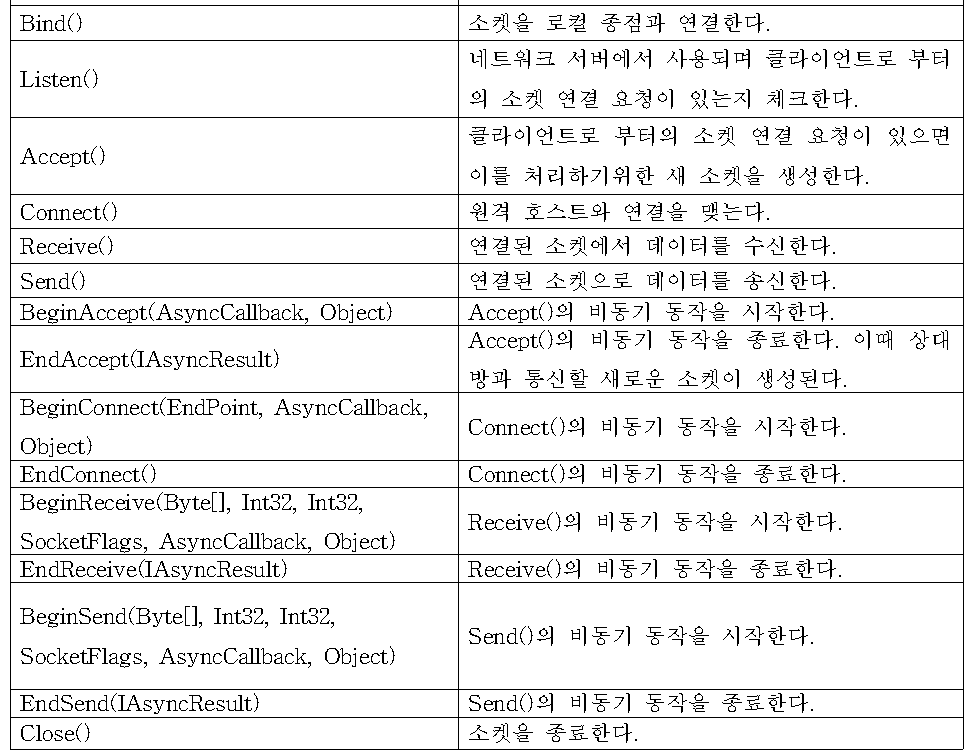
|  |
| --- |
| private void **OnPopOut**(object sender, ElapsedEventArgs e)  {  while (Height > 2)  {  Invoke(OnHeight, 1);  }  TimerEvent.Stop();  Application.DoEvents();  Invoke(OnHeight, 2);  } |

1. **C# 네트워크 통신** 
   1. System.Net.

* IPAddress class
  + 32비트 IP 주소 저장: byte[4]
  + 201.203.122.134을 저장하려면
  + IPAddress adAddr = IPAddress.Parse(“201.203.122.134“);   
    Console.WriteLine(ipAddr);
  + IPAddress adAddr2 = new IPAddress(new byte[] {201, 203, 122, 134});  
    Console.WriteLine(ipAddr2);
  + IPv4, IPv6 모두 제공
  + 보통 서버는 IPv4 사용, 일반 클라이언트는 사설 IP 주소 사용하거나 IPv6 주소 배부  
    사설 주소 범위: 10.0.0.0~10.255.255.255,   
     172.16.0.0~172.32.255.255,   
     192.168.0.0~192.168.255.255
* IPEndPoint class
  + IP 주소와 포트번호
  + IPEndPoint endpoint = new IPEndPoint(IPAddress.Parse(“201.203.122.134“), 9000);
* Dns class
  + 도메인 이름에 할당된 IP 주소 찾기  
    IPHostEntry entry = Dns.GetHostEntry(“[www.microsoft.com](http://www.microsoft.com)”);  
    foreach (IPAddress ipAddress in entry.AddressList){  
     Console.WriteLine(ipAddress);  
    }
  + 현재 컴퓨터에 할당된 IP 주소 찾기  
    string myComputer = Dns,GetHostName();  
    Console.WriteLine(“컴퓨터 이름: ” + myComputer);  
    IPHostEntry entry = Dns.GetHostEntry(myComputer);  
    foreach (IPAddress ipAddress in entry.AddressList){  
     Console.WriteLine(ipAddress);  
    }
  + IP주소를 사용할 때와 DNS주소를 사용할 때의 장단점  
    장점:  
    단점:
  + 서버의 경우 여러 IP 주소가 할당된 이유는 무엇인가
  1. System.Net.Sockets.Socket
* TCP/IP 통신 제공  
  public Socket {  
   AddressFamily addressFamily, // 주소 패밀리, AddressFamily.InterNetwork  
   SocketType socketType, // 소켓 타입, SocketType.Stream/Dgram   
   ProtocolType protocolType // 프로토콜 타입, ProtocolType.Tcp/Udp  
  }
* 생성자  
  socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp); //IPv4, TCP  
  socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp); //IPv4, UDP  
  socket(AddressFamily.InterNetworkV6, SocketType.Stream, ProtocolType.Tcp); //IPv6, TCP
* 소켓 통신 절차



* C# Socket 클래스의 주요 메서드

****

* 1. Synchronous Server and Client
* Server

|  |
| --- |
| using System;  using System.Net;  using System.Net.Sockets;  using System.Text;  using System.Threading;  class Program  {  static void Main(string[] args)  {  // 서버 소켓이 동작하는 스레드  Thread serverThread = new Thread(serverFunc);  serverThread.IsBackground = true;  serverThread.Start();  Thread.Sleep(500); // 소켓 서버용 스레드가 실행될 시간을 주기 위해  Console.WriteLine("종료하려면 아무 키나 누르세요...");  Console.ReadLine();  }  private static void serverFunc(object obj)  {  using (Socket srvSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp))  {  IPEndPoint endPoint = new IPEndPoint(IPAddress.Any, 11200);  srvSocket.Bind(endPoint);  srvSocket.Listen(10);  while (true)  {  Socket clntSocket = srvSocket.Accept();  byte[] recvBytes = new byte[1024];  int nRecv = clntSocket.Receive(recvBytes);  string txt = Encoding.UTF8.GetString(recvBytes, 0, nRecv);  byte[] sendBytes = Encoding.UTF8.GetBytes("Hello: " + txt);  clntSocket.Send(sendBytes);  clntSocket.Close();  }  }  }  } |

* Client

|  |
| --- |
| using System;  using System.Net;  using System.Net.Sockets;  using System.Text;  using System.Threading;  class Program  {  static void Main(string[] args)  {  // 클라이언트 소켓이 동작하는 스레드  Thread clientThread = new Thread(clientFunc);  clientThread.IsBackground = true;  clientThread.Start();  Console.WriteLine("종료하려면 아무 키나 누르세요...");  Console.ReadLine();  }  private static void clientFunc(object obj)  {  Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);  EndPoint serverEP = new IPEndPoint(IPAddress.Loopback, 11200);  socket.Connect(serverEP);  byte[] buf = Encoding.UTF8.GetBytes(DateTime.Now.ToString());  socket.Send(buf);  byte[] recvBytes = new byte[1024];  int nRecv = socket.Receive(recvBytes);  string txt = Encoding.UTF8.GetString(recvBytes, 0, nRecv);  Console.WriteLine(txt);  socket.Close();  Console.WriteLine("TCP Client socket: Closed");  }  } |

* 코드 파헤치기

- try/catch 문장은 왜 있는가? 어떨 때 catch 문장이 사용되는가?

- 위의 본 예제는 서버의 경우 클라이언트의 요청을 기다리는 동안 다른 일을 할 수 있는가?

- 클라이언트와 서버가 마음대로 데이터를 보낼 수 있는가?

- 서버가 클라이언트가 보내는 데이터를 받기 위해서는 무슨 일하는가?

- 여러 명의 클라이언트가 동시에 접속할 수 있는가?

- 우리가 흔히 쓰는 채팅 프로그램이 되려면 어떤 기능이 추가되어야 할까?

* 1. Asynchronous Server and Client
* 채팅 프로그램에서 Thread를 사용하는 경우 문제점

- 32 비트 서버에서 사용자 프로그램 사용가능 메모리 크기: 2GB

- 1MB/Thread 🡪 2000개 Thread 생성 가능 🡪 2000명 동시 접속 가능

**- 추가 오버헤드:**

**- 해결방안:**

* 사용 메소드

|  |  |
| --- | --- |
| 서버 | 클라이언트 |
| - BeginAccept🡪EndAccept | - BeginConnect🡪EndConnect |
| - BeginSend🡪EndSend | - BeginReceive 🡪 EndReceive |
| - BeginReceive 🡪 EndReceive | - BeginSend🡪EndSend |

* 서버 코드

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Net;  using System.Net.Sockets;  using System.Threading;  namespace consoleAsyncTCPServer  {  class Program  {  public static ManualResetEvent Accept\_Evnt = new ManualResetEvent(false);  public static ManualResetEvent Sendt\_Evnt = new ManualResetEvent(false);  public static ManualResetEvent Recieve\_Evnt = new ManualResetEvent(false);  public const int bufSize = 512;  public static byte[] rcv\_buff = new byte[bufSize];  public static string rcvMsg = null;  public static Socket client = null;  public static void Accept\_Cbk(IAsyncResult ar)  {  Thread thr = Thread.CurrentThread;  Console.WriteLine("status of main Thread : " + thr.ThreadState);  Socket srvr = (Socket)ar.AsyncState;  client = srvr.EndAccept(ar);  Accept\_Evnt.Set();  }  public static void Send\_Cbk(IAsyncResult ar)  {  Socket ct = (Socket)ar.AsyncState;  int recvNum = ct.EndSend(ar);  Sendt\_Evnt.Set();  }  public static void Receive\_Cbk(IAsyncResult ar)  {  Socket ct = (Socket)ar.AsyncState;  int recvNum = ct.EndReceive(ar);  if (recvNum > 0)  {  rcvMsg = Encoding.UTF8.GetString(rcv\_buff, 0, recvNum);  Console.WriteLine("Client: " + rcvMsg);  Recieve\_Evnt.Set();  }  }  static void Main(string[] args)  {  int portNum = 4000;  Thread thr = Thread.CurrentThread;  Console.WriteLine("status of main Thread : " + thr.ThreadState);  IPAddress locIP = IPAddress.Parse("127.0.0.1");  IPEndPoint locEnd = new IPEndPoint(locIP, portNum);  try  {  Socket svrSock = new Socket(AddressFamily.InterNetwork,  SocketType.Stream, ProtocolType.Tcp);  svrSock.Bind(locEnd);  svrSock.Listen(5);  Console.WriteLine("Server: start " );  svrSock.BeginAccept(new AsyncCallback(Accept\_Cbk), svrSock);  Accept\_Evnt.WaitOne();  Console.WriteLine("Server:connected a connect");  client.BeginReceive(rcv\_buff, 0, bufSize, 0,  new AsyncCallback(Receive\_Cbk), client);  Recieve\_Evnt.WaitOne();  byte[] buff = Encoding.UTF8.GetBytes(rcvMsg);  client.BeginSend(buff, 0, buff.Length, 0,  new AsyncCallback(Send\_Cbk), client);  Sendt\_Evnt.WaitOne();  }  catch (Exception excp)  {  Console.WriteLine(excp.ToString());  }  }  }  } |

* 클라이언트 코드

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Net;  using System.Net.Sockets;  using System.Threading;  namespace consoleAsyncTCPClient  {  class Program  {  public static ManualResetEvent Connect\_Evnt = new ManualResetEvent(false);  public static ManualResetEvent Sendt\_Evnt = new ManualResetEvent(false);  public static ManualResetEvent Recieve\_Evnt = new ManualResetEvent(false);  public const int bufSize = 512;  public static byte[] rcv\_buff = new byte[bufSize];  public static string rcvMsg = null;  public static void Connect\_Cbk(IAsyncResult ar)  {  Thread thr = Thread.CurrentThread;  Console.WriteLine("status of main Thread : " + thr.ThreadState);  Socket ct = (Socket)ar.AsyncState;  ct.EndConnect(ar);  Connect\_Evnt.Set();  }  public static void Send\_Cbk(IAsyncResult ar)  {  Socket ct = (Socket)ar.AsyncState;  int recvNum = ct.EndSend(ar);  Sendt\_Evnt.Set();  }  public static void Receive\_Cbk(IAsyncResult ar)  {  Socket ct = (Socket)ar.AsyncState;  int recvNum = ct.EndReceive(ar);  if (recvNum > 0)  {  rcvMsg = Encoding.UTF8.GetString(rcv\_buff, 0, recvNum);  Console.WriteLine("서버 응답: " + rcvMsg);  Recieve\_Evnt.Set();  }  }  static void Main(string[] args)  {  int portNum = 4000;  //Thread thr = Thread.CurrentThread;  //Console.WriteLine("status of main Thread : " + thr.ThreadState);  //IPHostEntry remote = Dns.GetHostEntry(hostName);  // IPAddress remoteIP = remote.AddressList[0];  IPAddress remoteIP = IPAddress.Parse("127.0.0.1");  IPEndPoint remoteEnd = new IPEndPoint(remoteIP, portNum);  try  {  Socket ctSock = new Socket(AddressFamily.InterNetwork,  SocketType.Stream, ProtocolType.Tcp);  ctSock.BeginConnect(remoteEnd,  new AsyncCallback(Connect\_Cbk), ctSock);  Connect\_Evnt.WaitOne();  Console.WriteLine("client: connected ");  string tmp = "From Client : Hello this is a test message";  byte[] buff = Encoding.UTF8.GetBytes(tmp);  ctSock.BeginSend(buff, 0, buff.Length, 0,  new AsyncCallback(Send\_Cbk), ctSock);  Sendt\_Evnt.WaitOne();  Console.WriteLine("client: sent a message: "+ tmp);  ctSock.BeginReceive(rcv\_buff, 0, bufSize, 0,  new AsyncCallback(Receive\_Cbk), ctSock);  Recieve\_Evnt.WaitOne();  ctSock.Close();  }  catch (Exception excp)  {  Console.WriteLine(excp.ToString());  }  }  }  } |

* 코드 파헤치기

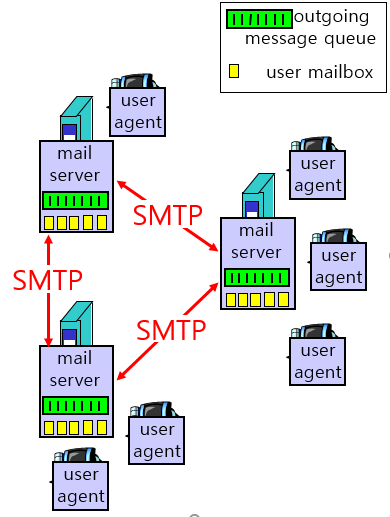
- 동기화 통신과 비동기 통신이 어떻게 다른가?

사용하는 함수

결과

1. **SMTP (Simple Mail Transfer Protocol)**
   1. 정의

SMTP 프로토콜은 인터넷에서 메일 전달하기 위해 사용되는 통신 방식이면 TCP에서 25번 포트를 사용하여 동작한다. 이 프로토콜은 아래 그림처럼 메일 서버와 클라이언트, 메일 서버와 메일 서버사이에서 동작하여 메일을 전달한다.



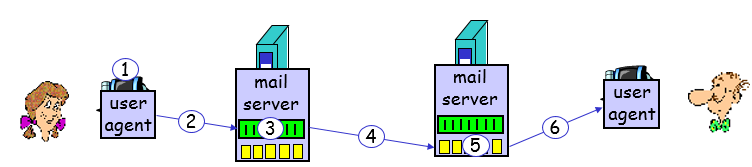
* 1. 구성요소

E-mail의 구성요소는 user agent, mail server, SMTP이다.

* User Agent
* 메일 작성, 편집, 메일 읽기
* 서버에 저장된 메일을 내보내거나 들여오기
* Mail Servers
* 사용자를 위한 받은 메일을 담고 있는 메일박스
* 보낸 메일의 메시지 큐
* 메일 전송을 위한 메일 서버 간의 SMTP protocol 동작

“client”: sending mail server ↔ “server”: receiving mail server

* SMTP
  + Command/response
    - Command: ASCII text
    - Response: status code와 status phrase
  + SMTP 동작



1. 앨리스가 user agent를 사용하여 밥에게 메일을 작성한다. ([bob@bible.edu](mailto:bob@bible.edu))
2. 앨리스의 UA가 메일을 메일 서버로 보낸다. 이 메일은 메시지 큐에 저장된다.
3. SMTP의 클라이언트(앨리스의 메일 서버)는 밥의 메일 서버에게 TCP 연결을 연다.
4. SMTP의 클라이언트는 밥의 메일 서버에게 밥의 메일을 전송한다.
5. 밥의 메일 서버가 밥의 메일 박스에 메일을 저장한다.
6. 밥은 메일을 읽으려고 자신의 UA를 실행시킨다.
   * SMTP Commands/Response

**S: 220 bible.com**

**C: HELO bible.ac.kr**

**S: 250 Hello bible.ac.kr, pleased to meet you**

**C: MAIL FROM: <alice@bible.ac.kr>**

**S: 250 alice@bible.ac.kr... Sender ok**

**C: RCPT TO: <bob@bible.com>**

**S: 250 bob@bible.edu... Recipient ok**

**C: DATA**

**S: 354 Enter mail, end with "." on a line by itself**

**C: Do you like ketchup?**

**C: How about pickles?**

**C: .**

**S: 250 Message accepted for delivery**

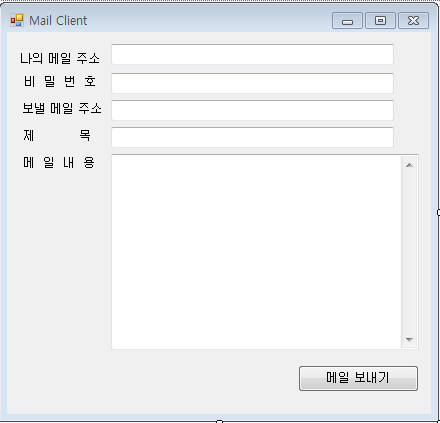
**C: QUIT**

**S: 221 bible.com closing connection**

* + SMTP 클라이언트 만들기

구글의 SMTP 서버를 이용하여 SMTP 클라이언트를 작성  
보안 문제로 인하여 다음 링크에서 보안수준을 낮추어야 오류 없이 실행가능

https://myaccount.google.com/u/1/lesssecureapps?pageId=none



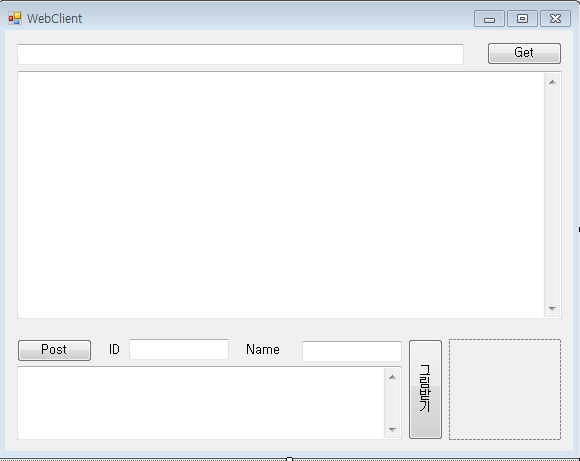
|  |
| --- |
| string mailServer = "smtp.gmail.com";  string fromPassword = txtPassword.Text;  MailAddress from = new MailAddress(textFromAddr.Text);  MailAddress to = new MailAddress(textToAddr.Text);  SmtpClient client = new SmtpClient(mailServer, 587);  client.UseDefaultCredentials = false;  client.EnableSsl = true;  client.DeliveryMethod = SmtpDeliveryMethod.Network;  client.Credentials = new NetworkCredential(from.Address, fromPassword);  client.Timeout = 20000;  MailMessage message = new MailMessage(from, to)  {  Subject = txtSubject.Text,  SubjectEncoding = System.Text.Encoding.UTF8,  Body = txtContent.Text,  BodyEncoding = System.Text.Encoding.UTF8  };  try  {  client.Send(message);  }  catch (Exception ex) {  MessageBox.Show(ex.ToString());  }  finally  {  message.Dispose();  } |

1. **HTTP (Hyper Transfer Text Protocol)**
   1. System.Net.Web 클래스

* WebRequest 클래스의 하위 클래스가 HttpWebRequest임
  + HTTP 관련 구현 클래스 제공
  + HttpWebRequest myReq =

(HttpWebRequest)WebRequest.Create("http://www.contoso.com/");

* + URI는 http:// 또는 https:// 모두 가능
  + 서버로부터 데이터를 받을 때는 [GetResponse](https://docs.microsoft.com/ko-kr/dotnet/api/system.net.httpwebrequest.getrequeststream?view=netframework-4.7.2)()로 응답을 받아옴
  + 서버로부터 데이터를 받을 때는 [GetResponse](https://docs.microsoft.com/ko-kr/dotnet/api/system.net.httpwebrequest.getrequeststream?view=netframework-4.7.2)Stream()로 응답 데이터를 받아올 수 있는 스트림을 받아 데이터를 읽어들임.
  + 데이터를 서버에 전송할 때는 [GetRequestStream](https://docs.microsoft.com/ko-kr/dotnet/api/system.net.httpwebrequest.getrequeststream?view=netframework-4.7.2)()으로 스트림 개체를 가져와서 데이터를 전송
* 스트림이란
  + 메모리나 파일, 소켓 등에 데이터를 내보내거나 가져올 때 사용
  + 특히 소켓에서 send, receive 대신 read, write로 데이터 전송가능 해지면서 메모리, 파일, 소켓 통신에서 동일한 이름의 메소드 사용으로 프로그래밍이 쉬워짐



TEXTBOX: textURL Button: buttonGet

TEXTBOX: textBoxWeb

Button: buttonPost TEXTBOX: textBoxID , TEXTBOX: textBoxName

TEXTBOX: textBoxPost Button: buttonImageDown PictureBOX: imagePB

* 이벤트 코드

using System.Net; // webrequest

using System.IO; // file 처리

|  |
| --- |
| private void buttonGet\_Click(object sender, EventArgs e)  {  string url = textURL.Text; // "https://httpbin.org/get"; //테스트 사이트  if (url == "")  {  url = "https://httpbin.org/get";  }  string responseText = string.Empty;  HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);  request.Method = "GET";  request.Timeout = 30 \* 1000; // 30초  request.Headers.Add("Authorization", "BASIC SGVsbG8="); // 헤더 추가 방법  using (HttpWebResponse resp = (HttpWebResponse)request.GetResponse())  {  HttpStatusCode status = resp.StatusCode;  Console.WriteLine(status); // 정상이면 "OK"  textURL.Text = status.ToString();  Stream respStream = resp.GetResponseStream();  using (StreamReader sr = new StreamReader(respStream))  {  responseText = sr.ReadToEnd();  }  }  Console.WriteLine(responseText);  textBoxWeb.Text = responseText ;  } |

|  |
| --- |
| private void buttonPost\_Click(object sender, EventArgs e)  {  string data = "{ \"id\": \""+ textBoxID.Text + "\", \"name\" : \""+  textBoxName.Text + "\" }";  HttpWebRequest request = (HttpWebRequest)WebRequest.Create  ("https://httpbin.org/post");  request.Method = "POST";  request.ContentType = "application/json";  request.Timeout = 30 \* 1000;  //request.Headers.Add("Authorization", "BASIC SGVsbG8=");  // POST할 데이타를 Request Stream에 쓴다  byte[] bytes = Encoding.ASCII.GetBytes(data);  request.ContentLength = bytes.Length; // 바이트수 지정  using (Stream reqStream = request.GetRequestStream())  {  reqStream.Write(bytes, 0, bytes.Length);  }  // Response 처리  string responseText = string.Empty;  using (WebResponse resp = request.GetResponse())  {  Stream respStream = resp.GetResponseStream();  using (StreamReader sr = new StreamReader(respStream))  {  responseText = sr.ReadToEnd();  }  }  Console.WriteLine(responseText);  textBoxPost.Text = responseText;  } |

|  |
| --- |
| private void buttonImageDown\_Click(object sender, EventArgs e)  {  HttpWebRequest request =  (HttpWebRequest)WebRequest.Create("http://httpbin.org/image/png");  request.Method = "GET";  // Response 바이너리 데이타 처리. 이미지 파일로 저장.  using (WebResponse resp = request.GetResponse())  {  var buff = new byte[1024];  int pos = 0;  int count;  string filename = "image.png";  using (Stream stream = resp.GetResponseStream())  {  using (var fs = new FileStream(filename, FileMode.Create))  {  do  {  count = stream.Read(buff, pos, buff.Length);  fs.Write(buff, 0, count);  } while (count > 0);  }  }  imagePB.Image = new Bitmap(filename);  }  } |

1. **UDP 통신**
   1. UDP 클래스

* System.Net.Sockets
* 소켓 생성   
  Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp)
* 바인딩: Bind(endpoint)
* 데이터를 받을 때: ReceiveFrom(recvBytes, ref clientEP)
* 데이터를 보낼 때: SendTo(sendBytes, clientEP);
* UDPServer.cs

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Net;  using System.Net.Sockets;  using System.Threading;  namespace UDP\_Comm\_Console  {  class Program  {  static Socket serverS;  static void serverFunc()  {  try  {  using (serverS = new Socket(AddressFamily.InterNetwork,  SocketType.Dgram, ProtocolType.Udp))  {  IPAddress ipAddress = IPAddress.Parse("127.0.0.1");  // 0.0.0.0은 소켓을 모든 IP에 바인딩할 때 사용  //IPEndPoint endPoint = new IPEndPoint(ipAddress, 10200);  IPEndPoint endPoint = new IPEndPoint(ipAddress, 10200);  // 위 두 줄을 IPAddress.Any를 이용하여 한 줄로 처리.  serverS.Bind(endPoint);  byte[] recvBytes = new byte[1024];  EndPoint clientEP = new IPEndPoint(IPAddress.None, 0);  while (true)  {  int nRecv = serverS.ReceiveFrom(recvBytes, ref clientEP);  string str = Encoding.UTF8.GetString(recvBytes, 0, nRecv);  Console.WriteLine("From {0} : {1}", clientEP.ToString(), str);  byte[] sendBytes = Encoding.UTF8.GetBytes("Resend : " + str);  serverS.SendTo(sendBytes, clientEP);  }  }  }  catch (SocketException e)  {  Console.WriteLine(e.StackTrace);  }  }  static void Main(string[] args)  {  Thread th = new Thread(serverFunc);  th.Start();  while (true)  {  string line = Console.ReadLine();  if (line == "QUIT")  {  break;  }  }  serverS.Close();  th.Join();  }  }  } |

* UDPClient.cs

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Net;  using System.Net.Sockets;  using System.Threading;  namespace UDPClient  {  class Program  {  static IPAddress GetCurrentIPAddress()  {  IPAddress[] addrs = Dns.GetHostEntry(Dns.GetHostName()).AddressList;  foreach (IPAddress ipAddr in addrs)  {  if (ipAddr.AddressFamily == AddressFamily.InterNetwork)  {  return ipAddr;  }  }  return null;  }  static void clientFunc()  {  Socket clientS = new Socket(AddressFamily.InterNetwork,  SocketType.Dgram, ProtocolType.Udp);  //EndPoint serverEP = new IPEndPoint(GetCurrentIPAddress(), 10200);  EndPoint serverEP = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 10200);  EndPoint senderEP = new IPEndPoint(IPAddress.None, 0);  byte[] buf = Encoding.UTF8.GetBytes(DateTime.Now.ToString());  clientS.SendTo(buf, serverEP);  byte[] recvBuf = new byte[1024];  int nRecv = clientS.ReceiveFrom(recvBuf, ref senderEP);  string str = Encoding.UTF8.GetString(recvBuf, 0, nRecv);  Console.WriteLine(str);  clientS.Close();  }  static void Main(string[] args)  {  clientFunc();  }  }  } |

* UDP 프로그램을 서버에게 키보드로 입력한 데이터를 보내도록 변경해보자

|  |
| --- |
| while (true) {  str = Console.ReadLine();  if (str.Equals("quit")) break;  buf = Encoding.UTF8.GetBytes(str);  clientS.SendTo(buf, serverEP);  nRecv = clientS.ReceiveFrom(recvBuf, ref senderEP);  str = Encoding.UTF8.GetString(recvBuf, 0, nRecv);  Console.WriteLine(str);  } |