네트워크 소켓 프로그래밍

경북대학교 컴퓨터학부

고석주

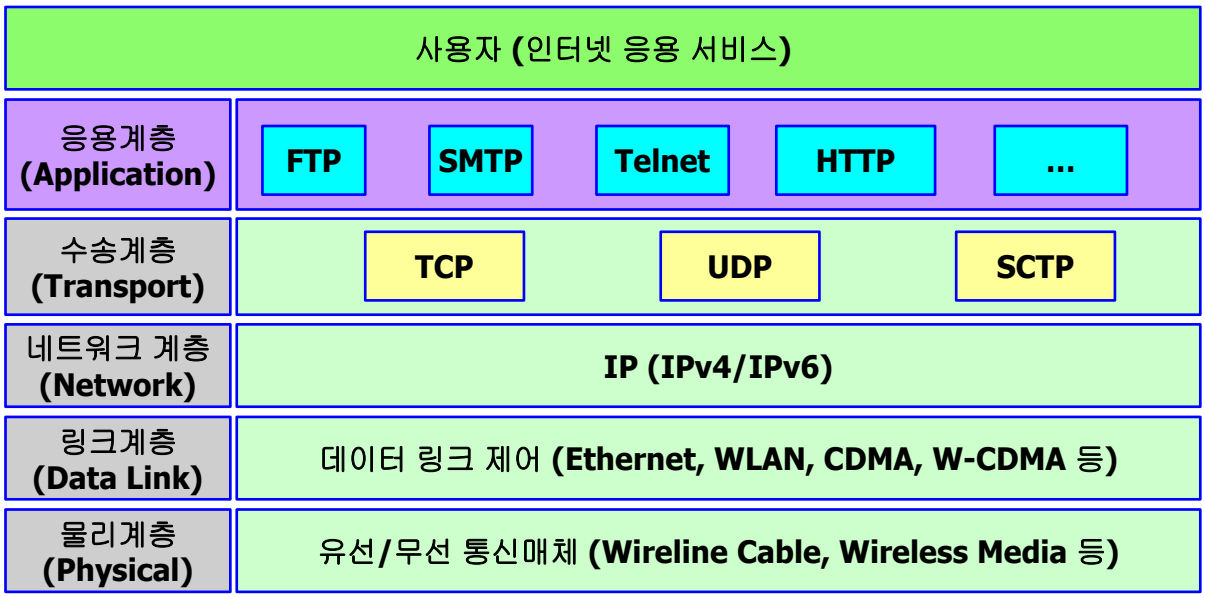
<목 차>

1. IoT 응용 개발을 위한 소켓 프로그래밍

(1) TCP/IP 응용 계층 프로토콜(프로그램)

- TCP/IP 패킷은 다양한 인터넷 응용(application) 서비스를 제공하기 위한 데이터를 포함한다. 이처럼 **"응용서비스별로 어떻게 데이터를 구성하고 전송해야 하는지"를 정의한 프로토콜이 응용 계층 프로토콜**이다.[1)]

- 다음 그림처럼 응용 계층 프로토콜은 상위에 있는 응용서비스 사용자와 하위에 있는 수송계층 프로토콜과의 중간 인터페이스 기능을 제공한다.



<TCP/IP 응용계층 프로토콜>

- HTTP, CoAP 등의 **응용 계층 프로토콜은 '소켓(socket) 프로그래밍' 기법을 사용하여 '응용 프로그램(application program)'으로 구현**되며, 프로토콜 동작에 필요한 데이터(HTTP 프로토콜의 payload 등)를 처리하는 기능이 소켓 프로그래밍 코드에 포함된다.

---

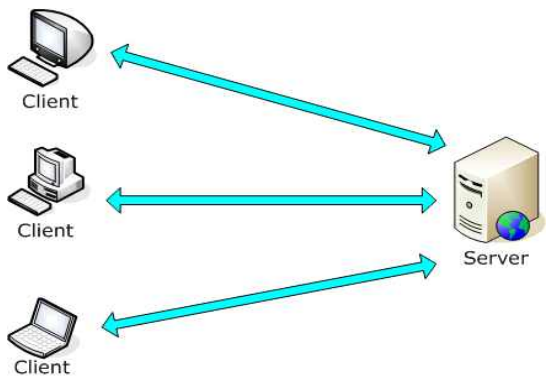
1) IoT 서비스를 위한 응용 계층 프로토콜에는 HTTP, MQTT, CoAP 등이 있다.

(2) 클라이언트-서버 모델

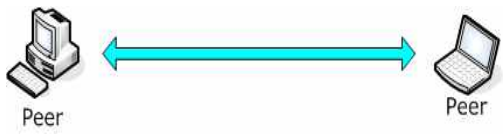
- **대부분의 인터넷 응용서비스들은 클라이언트-서버(Client-Server) 모델에 따라** 동작한다. 예를 들어, 웹서비스를 위해 웹클라이언트와 웹서버가 사용되며, 메일 서비스를 위해 메일 클라이언트와 메일 서버가 동작한다

- 웹서비스에서 알 수 있듯이, **client는 일반적인 서비스 사용자를 의미하고, server는 서비스를 제공하는 사업자**를[2] 의미한다. client는 필요할 때에 server에 접속하여(연결하여) 원하는 서비스를 받은 다음에 연결을 종료하는 반면에, **server는 1년 365일 동안 불특정 다수의 client 접속을 대기하고 client의 요청 서비스를 제공**한다. 대개 server는 다수의 client를 상대한다.

- 한편, Client-Server 모델에 대응되는 모델은 Peer-to-Peer (P-P) 모델이다. **P-P 모델에서는 두 단말이 동등한 입장에서 필요에 따라 client 역할과 server 역할을 모두 수행**한다. P-P 모델의 전형적인 예제는 VoIP(Voice over IP) 전화 서비스이다.



Client-Server 모델



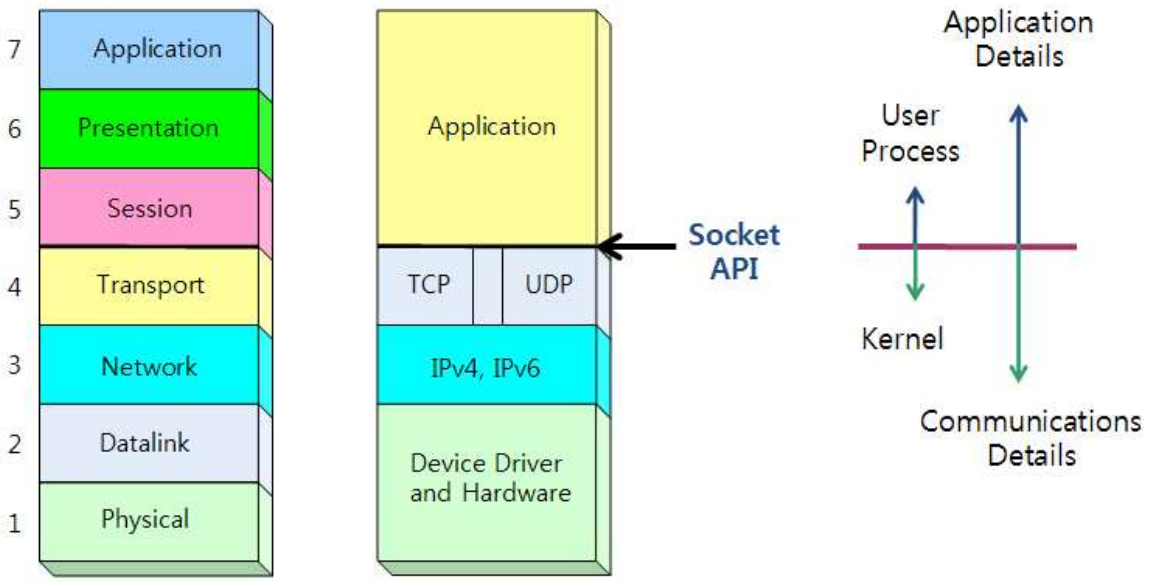
Peer-to-Peer 모델

2) 학교나 정부기관 등 각 기관에서 운영하는 홈페이지는 웹서버에 해당한다.

(3) TCP/IP 프로토콜과 소켓 프로그래밍

- TCP/IP 응용 개발을 위해 socket API(Application Programming Interface)가 사용된다. 응용 프로토콜(혹은 프로그램)은 응용 서비스 기능을 다루고 socket API를 통해 통신에 필요한 기능을(하위 4개 계층) 활용한다 (예: data 전송, 오류제어, 흐름제어 등).

- 소켓 프로그래밍을 통해 TCP/IP 프로토콜의 세부 동작에 대한 깊은 지식이 없이도 응용 프로그램 개발이 가능하다



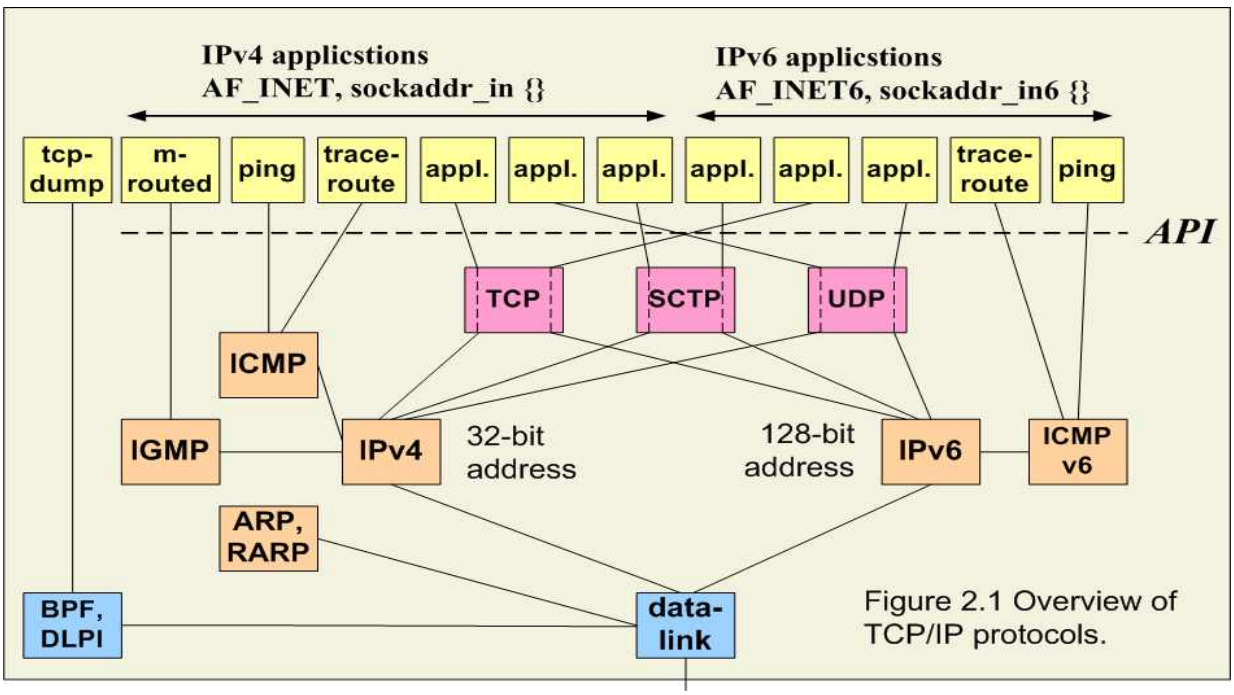
<TCP/IP 프로토콜과 소켓 API>

- 소켓 API의 상위 응용계층은 'user process'로 구현되고 하위 4개의 계층은 OS(operating system)의 '커널(kernel)'에서 제공된다. 이러한 의미에서 "socket API"라는 표현을 사용한다[3].

3) 소켓 API란 socket(), bind(), write(), read() 등의 소켓 프로그래밍 함수를 의미함

- 우리는 소켓 API 함수를 통해 OS의 커널에서 제공하는 다양한 TCP/IP 통신 기능을 활용할 수 있다.

- 아래 그림처럼 IPv4/IPv6 응용 프로토콜(프로그램) 개발을 위해 socket API 함수를 사용하고, socket API 함수는 내부적으로 TCP, UDP, ICMP, IPv4/IPv6 등의 프로토콜을 사용하여 상대방 컴퓨터와의 통신 기능을 수행한다.



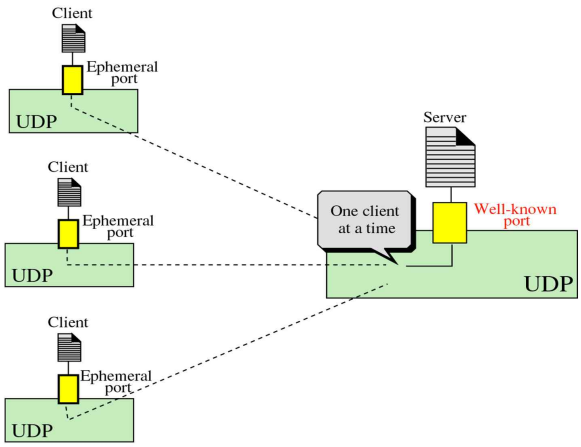
<소켓 API를 활용한 다양한 TCP/IP 통신 기능 활용>

- 소켓 프로그래밍 작업시에 하위 계층의 통신 프로토콜에 대한 전문적인 지식이 없어도 가능하나, 통신 프로토콜에 대한 이해도가 높을수록 더욱 정교한 응용 프로그램을 개발할 수 있다[4].

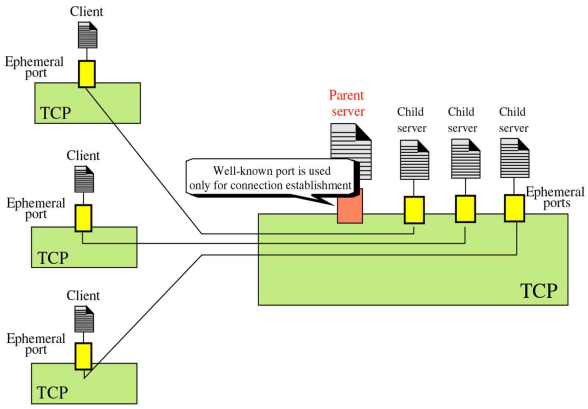
4) '소켓 프로그래밍 학습'이란 다양한 '소켓 API 함수의 사용법'을 배우는 과정이다. 즉, 소켓 API 함수의 입력(내부 arguments) 방법을 익히고, 함수 호출 시 이에 대한 출력(결과)에 대한 이해도를 높이는 과정이다.

(4) 서버 구현 모델

- TCP/IP 통신의 클라이언트-서버 모델에서 클라이언트에 비해 '서버'의 구현이 다소 복잡하다. 서버 구현 기법은 크게 'iterative' 서버와 'concurrent' 서버로 구분된다[5].



Iterative 서버 (UDP)



Concurrent 서버 (TCP)

- Iterative(반복적) 서버는 보통 UDP에서 많이 사용하는데, 한 순간에 하나의 클라이언트를 처리한다. 즉, 클라이언트들을 차례대로 하나씩 하나씩 처리하는 개념이다.

- 이에 비해 Concurrent(동시적) 서버는 동시에 여러 클라이언트를 상대한다. 이를 위해 '다중 프로세스(multi-processes)' 혹은 '다중 쓰레드(multi-threads) 기법이 사용된다. 즉, 동일한 프로세스 혹은 쓰레드를 생성하여 각각의 클라이언트를 상대한다. 이에 대한 세부 사항은 다음절에서 기술한다.

---

5) '클라이언트' 프로그램 구현은 상대적으로 간단하다. 서버는 다수의 클라이언트를 상대하는 반면에 클라이언트는 서버만 상대하기 때문이다.

2. UDP 소켓 함수

(1) 소켓 기본 API

- 먼저 기본적으로 사용하는(호출하는) 소켓 API 함수를 알아보자[6].

A. socket()

- 소켓을 생성하기 위해 맨 처음 호출하는 함수이다. socket() 함수는 시스템(운영체제) 폴더의 'sys/socket.h' 파일에 정의되어 있다.

```
#include <sys/socket.h>

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

Returns: non-negative descriptor if OK, -1 on error
```

- family, type 필드(argument)에는 다음 값들이 올 수 있다[7].

Family	Description
AF_INET	IPv4 protocol
AF_INET6	IPv6 protocol

Type	Description
SOCK_STREAM	Stream (TCP) socket
SOCK_DGRAM	Datagram (UDP) socket

6) 여기서는 '리눅스(linux)' 소켓을 기술한다. '윈도우즈(windows)' 기반 소켓도 리눅스와 유사하다.

7) 'protocol' 필드에는 프로토콜 번호(TCP:6, UDP:17)가 오는데 '0'으로 입력하면 자동으로 번호를 인식한다.

B. bind()

- **'서버'에서 사용하는 함수**이다. socket()에서 생성된 소켓 descriptor(정수값)에 서버의 소켓 주소(IP주소+포트번호[8])를 할당(bind)해 준다.

```
#include <sys/socket.h>

int bind (int sockfd, const struct sockaddr *myaddr, socklen_t addrlen);

Returns: 0 if OK,-1 on error
```

- 두 번째 필드의 IPv4 socket address 구조체는 <netinet/in.h>에 정의되어 있고, IP주소와 포트번호를 포함한다.

```
struct in_addr {
in_addr_t    s_addr;        /* 32-bit IPv4 address */
};

struct sockaddr_in {
  uint8_t        sin_len;    /* length of structure */
  sa_family_t     sin_family;  /* AF_INET */
  in_port_t      sin_port;   /* 16-bit TCP or UDP port number */
  struct in_addr   sin_addr;   /* 32-bit IPv4 address */
  char         sin_zero[8];  /* unused */
};
```

- IPv6 socket address 구조체도 유사한 방식으로 사용된다.
- 세 번째 필드는 주소 구조체의 길이를 의미한다.

8) 서버는 대개 'well-known port number(1~1024)'를 사용한다. 반면에 클라이트는 포트번호로서 큰 숫자를 사용하며 이를 ephemeral(혹은 dynamic, local) 포트번호라고 부른다.

C. sendto(), recvfrom()

- UDP 소켓에서는 **데이터 송수신**을 위해 sendto(), recvfrom()을 사용한다[9].

```
ssize_t recvfrom(int sockfd, void *buff, size_t nbytes, int
flags, struct sockaddr *from, socklen_t *addrlen);

ssize_t sendto(int sockfd, const void *buff, size_t nbytes, int
flags, const struct sockaddr *to, socklen_t addrlen);
```

- 두 번째 필드인 buff는 송수신할 문자열을 의미하고, 세 번째 필드는 문자열 길이를 의미한다[10].
- 마지막 5, 6번째 필드에는 상대방(송신자 혹은 수신자)의 소켓 주소를 의미한다[11].

D. close()

- 소켓 통신 완료 후 종료하기 위해 close() 함수를 호출한다.

```
#include <unistd.h>

int close (int sockfd);

Returns: 0 if OK, -1 on error
```

9) 반면에 TCP는 대개 write(), read()를 사용하며 더욱 간단하다. 이에 대해서는 나중에 설명한다.
10) 네 번째 필드에는 각종 옵션(option) 값이 올 수 있는데 여기서는 '0'으로 설정한다.
11) sendto의 마지막 argument는 정수값이지만, recvfrom의 마지막 argument는 포인터(pointer)임에 유의하자. recvfrom()의 경우 시스템에 값을 주는게 아니라 시스템에서 관련 정보를 읽어오기 때문이다.

(2) 예제: UDP echo 서버 코드

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#define BUF_SIZE 30

int main(int argc, char *argv[])
{
	int serv_sock;			/* 사용될 변수 선언 */
	char message[BUF_SIZE];
	int str_len;
	socklen_t clnt_adr_sz;
	struct sockaddr_in serv_adr, clnt_adr;

	serv_sock=socket(PF_INET, SOCK_DGRAM, 0);

	memset(&serv_adr, 0, sizeof(serv_adr)); /* 소켓 구조체 초기화 */
	serv_adr.sin_family=AF_INET;
	serv_adr.sin_addr.s_addr=htonl(INADDR_ANY);
	serv_adr.sin_port=htons(atoi(argv[1]));

	bind(serv_sock, (struct sockaddr*)&serv_adr, sizeof(serv_adr));

	while(1) {	/* 데이터 송수신: 클라이언트 데이터 수신 후 echo 전송 */
		clnt_adr_sz=sizeof(clnt_adr);
		str_len=recvfrom(serv_sock, message, BUF_SIZE, 0,
			(struct sockaddr*)&clnt_adr, &clnt_adr_sz);
		sendto(serv_sock, message, str_len, 0,
			(struct sockaddr*)&clnt_adr, clnt_adr_sz);
	}
	close(serv_sock);
	return 0;
}
```

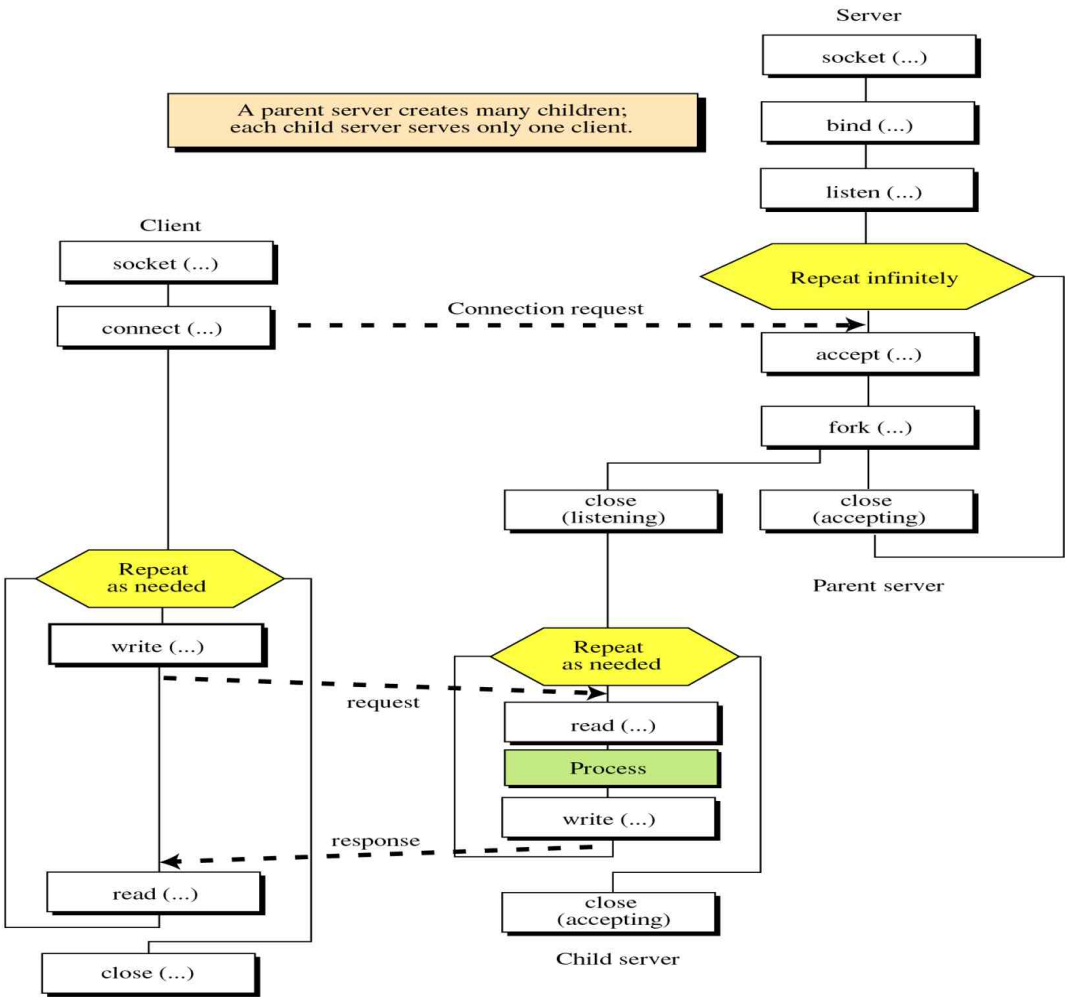
(3) 예제: UDP echo 클라이언트 코드

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#define BUF_SIZE 30

int main(int argc, char *argv[])
{
	int sock;                    /* 사용될 변수 선언 */
	char message[BUF_SIZE];
	int str_len;
	socklen_t adr_sz;
	struct sockaddr_in serv_adr, from_adr;

	sock=socket(PF_INET, SOCK_DGRAM, 0);

	memset(&serv_adr, 0, sizeof(serv_adr)); /* 소켓 구조체 초기화 */
	serv_adr.sin_family=AF_INET;
	serv_adr.sin_addr.s_addr=inet_addr(argv[1]);
	serv_adr.sin_port=htons(atoi(argv[2]));

	while(1)   /* 데이터 송수신: 서버에게 데이터 송신 후 echo 출력 */
	{
		fputs("Insert message(q to quit): ", stdout);
		fgets(message, sizeof(message), stdin);
		if(!strcmp(message,"q\n") || !strcmp(message,"Q\n"))
			break;

		sendto(sock, message, strlen(message), 0,
		     (struct sockaddr*)&serv_adr, sizeof(serv_adr));
		adr_sz=sizeof(from_adr);
		str_len=recvfrom(sock, message, BUF_SIZE, 0,
			(struct sockaddr*)&from_adr, &adr_sz);

		message[str_len]=0;
		printf("Message from server: %s", message);
	}
	close(sock);
	return 0;
}
```

3. TCP 소켓 함수

(1) TCP 기반 Concurrent 서버

- UDP 서버와는 달리 TCP 서버는 여러 클라이언트를 동시에 상대한다. TCP는 대개 대용량 데이터를 장시간 전송하기 위해 사용되기 때문이다.
- TCP 서버는 concurrent 서버로 구현되며, 새로운 클라이언트가 접속할 때 마다 해당 클라이언트를 상대하기 위해 새로운 프로세스를 생성한다.

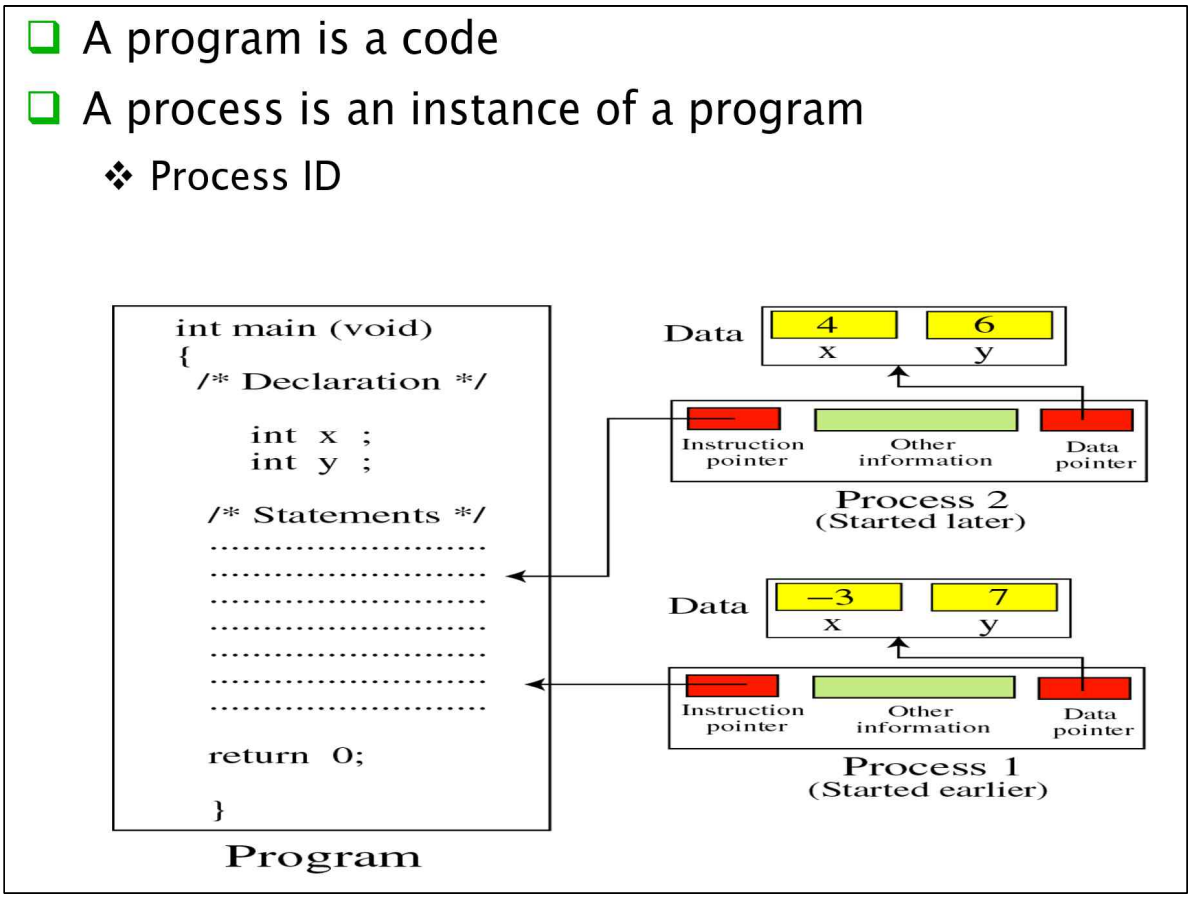


<TCP concurrent 서버의 소켓 API 호출 순서>

(2) 프로그램(program) vs. 프로세스(process)

- TCP concurrent 서버를 이해하기 위해서는 '프로세스'와 '프로그램'의 차이를 알아야 한다. 프로그램은 코드(code)를 의미하고, 프로세스는 '실행 중인 프로그램'을 의미한다.

- TCP 서버는 새로운 클라이언트가 접속하면, 새로운 프로세스를 생성시킨다. 즉, fork() 함수를 호출하여 동일한 프로그램 코드를 한번 더 실행시킨다. 이처럼 생성된 새로운 프로세스를 통해 해당 클라이언트와 데이터 송수신을 수행한다[12].



<프로세스 vs. 프로그램>

12) 이를 다중 프로세스 기반 서버라고 하며, 프로세스 대신에 쓰레드(thread)를 사용하기도 한다.

(3) TCP 소켓 API

- TCP에서 주로 사용하는 소켓 API 함수들은 다음과 같다.

A. connect()

- **TCP 클라이언트에서 호출하는 함수**이며, 서버와의 TCP 연결설정(3-way handshaking)을 위해 사용된다.
- 두 번째 필드에 서버의 소켓 주소가 들어간다.

```
#include <sys/socket.h>

int connect(int sockfd, const struct sockaddr *servaddr,

            socklen_t addrlen);

Returns: 0 if OK, -1 on error
```

B. listen()

- **TCP 서버에서 호출하는 함수**이며, 임의의 클라이언트가 접속하기를 기다리기 위해 호출하는 함수이다.
- 두 번째 필드에 동시접속이 가능한 클라이언트 개수를 기입한다.

```
#include <sys/socket.h>

#int listen (int sockfd, int backlog);

Returns: 0 if OK, -1 on error
```

C. accept()

- **TCP 서버는 listen 상태에서 새로운 클라이언트가 접속해 오면, 이를 처리(수용)하기 위해 accept()함수를 호출한다.**

- **클라이언트와 서버간에 TCP 연결과정이 완료되면** accept() 함수는 해당 '**클라이언트'의 소켓 주소**와(두번째 필드) 함께 대응되는 새로운 소켓 번호(descriptor)를 반환(return)한다[13].

```
#include <sys/socket.h>

int accept (int sockfd, struct sockaddr *cliaddr,

            socklen_t *addrlen);

Returns: non-negative descriptor if OK, -1 on error
```

D. write(), read()

- TCP 소켓에서는 **데이터 송수신**을 위해 write(), read()를 사용한다.

- UDP 소켓의 sendto(), recvfrom()과는 달리 3개의 필드만 포함된다. 상대방의 소켓 주소는 connect(), accept() 함수에서 이미 파악했기 때문에 필드에 추가할 필요가 없다.

```
ssize_t read(int sockfd, void *buff, size_t nbytes);

ssize_t write(int sockfd, void *buff, size_t nbytes);
```

13) 연결과정이 실패할 경우 accept() 함수는 -1을 반환한다.

E. fork()

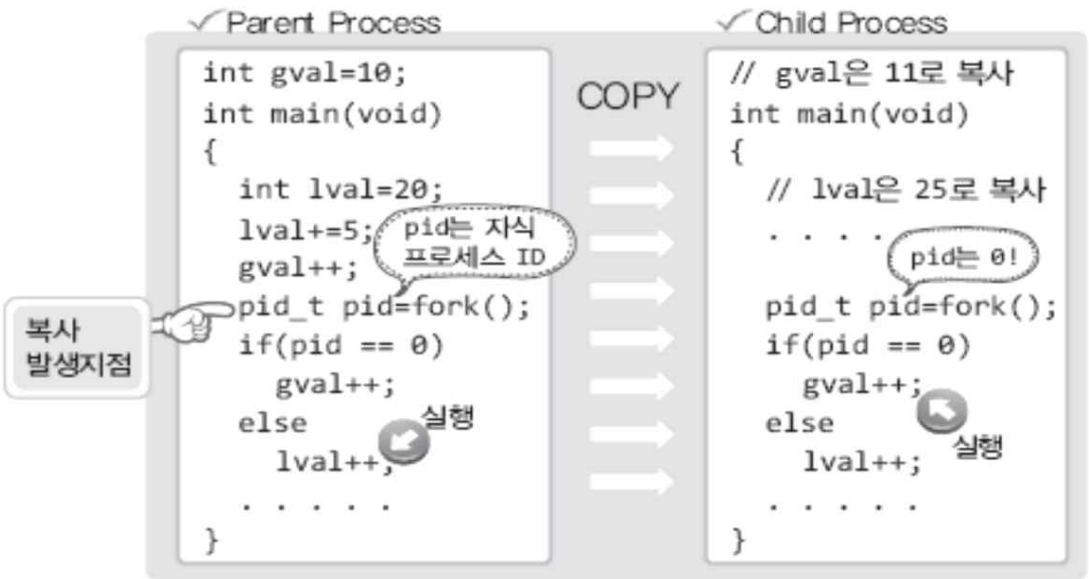
- TCP concurrent 서버는 새로운 클라이언트를 상대할 프로세스를 생성하기 위해 fork() 함수를 호출한다.

```
#include <unistd.h>

pid_t fork(void);

Returns: 0 in child, process ID of child in parent, -1 on error
```

.

- **fork() 함수가 호출되면 호출한 프로세스가 복사**되어, fork() **함수 호출 이후를 각각의 프로세스가 독립적으로 실행**하게 된다.

- 따라서 **fork() 함수의 반환값의 차이를 토대로 부모(parent) 프로세스와 자식(child) 프로세스의 프로그램 흐름을 구분**한다. (**부모 프로세스: fork() 반환값 = 자식 프로세스의 ID, 자식 프로세스: fork() 반환값 = 0)**



<fork() 함수 반환값을 통한 부모-자식 프로세스의 구별>

F. 기타 소켓 API 함수들

- 이 외에도 다양한 소켓 API 함수가 존재한다.
- 먼저 소켓 생성 후에 **소켓의 상태 정보를 파악하기 위해 getsockopt()**를 사용하며, **상태 정보를 변경할 때에는 setsockopt()를 사용**한다.

```
#include <sys/socket.h>
int getsockopt(int sockfd, int level, int optname,
                  void *optval, socklen_t *optlen);
int setsockopt(int sockfd, int level, int optname,
                  const void *optval, socklen_t optlen);
```

- 도메인 이름을 IP 주소로 변환하기 위해서는 gethostbyname() 함수를 사용한다(예: www.naver.com에 대한 IP 주소 파악).

```
#include <netdb.h>
struct hostent * gethostbyname(const char *hostname)
```

- 나의(local) 소켓 혹은 상대방(peer) 소켓 정보를 알고 싶을 경우에는 다음 소켓 API 함수를 사용한다.

```
#include <sys/socket.h>
int getsockname(sockfd, *localaddr, *addrlen);
int getpeername(sockfd, *peeraddr, *addrlen);
```

(4) 예제: TCP echo 서버 코드

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>

#define BUF_SIZE 30

int main(int argc, char *argv[])
{
	int serv_sock, clnt_sock;
	struct sockaddr_in serv_adr, clnt_adr;

	pid_t pid;
	struct sigaction act;
	socklen_t adr_sz;
	int str_len, state;
	char buf[BUF_SIZE];
	if(argc!=2) {
		printf("Usage : %s <port>\n", argv[0]);
		exit(1);
	}

	act.sa_handler=read_childproc;
	sigemptyset(&act.sa_mask);
	act.sa_flags=0;
	state=sigaction(SIGCHLD, &act, 0);
	serv_sock=socket(PF_INET, SOCK_STREAM, 0);
	memset(&serv_adr, 0, sizeof(serv_adr));
	serv_adr.sin_family=AF_INET;
	serv_adr.sin_addr.s_addr=htonl(INADDR_ANY);
	serv_adr.sin_port=htons(atoi(argv[1]));
```

```
        bind(serv_sock, (struct sockaddr*) &serv_adr,
             sizeof(serv_adr));
        listen(serv_sock, 5);

        /* 다중 프로세스를 사용하여 여러 클라이언트 상대하기 */
        while(1)
        {
              adr_sz=sizeof(clnt_adr);
              clnt_sock=accept(serv_sock,
                        (struct sockaddr*)&clnt_adr, &adr_sz);
             /* 클라이언트 소켓 생성 */
              puts("new client connected...");

              pid=fork();
              if(pid==0)    /* 자식 프로세스 */
              {
                    close(serv_sock);  /* 서버 소켓 닫기 */

                   /* client의 데이터를 수신하고 echo 하기 */
                    while((str_len=read(clnt_sock, buf,
                         BUF_SIZE))!=0)
                          write(clnt_sock, buf, str_len);

                    close(clnt_sock);
                    puts("client disconnected...");
                    return 0;
              }
              else       /* 부모 프로세스 */
                    close(clnt_sock);  /* 클라이언트 소켓 닫기 */
        }
        close(serv_sock);
        return 0;
}
```

(5) 예제: TCP echo 클라이언트 코드

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#define BUF_SIZE 1024

int main(int argc, char *argv[])
{
	int sock;
	char message[BUF_SIZE];
	int str_len;
	struct sockaddr_in serv_adr;

	sock=socket(PF_INET, SOCK_STREAM, 0);

	memset(&serv_adr, 0, sizeof(serv_adr));
	serv_adr.sin_family=AF_INET;
	serv_adr.sin_addr.s_addr=inet_addr(argv[1]);
	serv_adr.sin_port=htons(atoi(argv[2]));
	connect(sock, (struct sockaddr*)&serv_adr, sizeof(serv_adr));

	while(1) {
		fputs("Input message(Q to quit): ", stdout);
		fgets(message, BUF_SIZE, stdin);

		if(!strcmp(message,"q\n") || !strcmp(message,"Q\n"))
			break;

		write(sock, message, strlen(message));
		str_len=read(sock, message, BUF_SIZE-1);
		message[str_len]=0;
		printf("Message from server: %s", message);
	}
	close(sock);
	return 0;
}
```

4. (실습1) WSL 설치하기

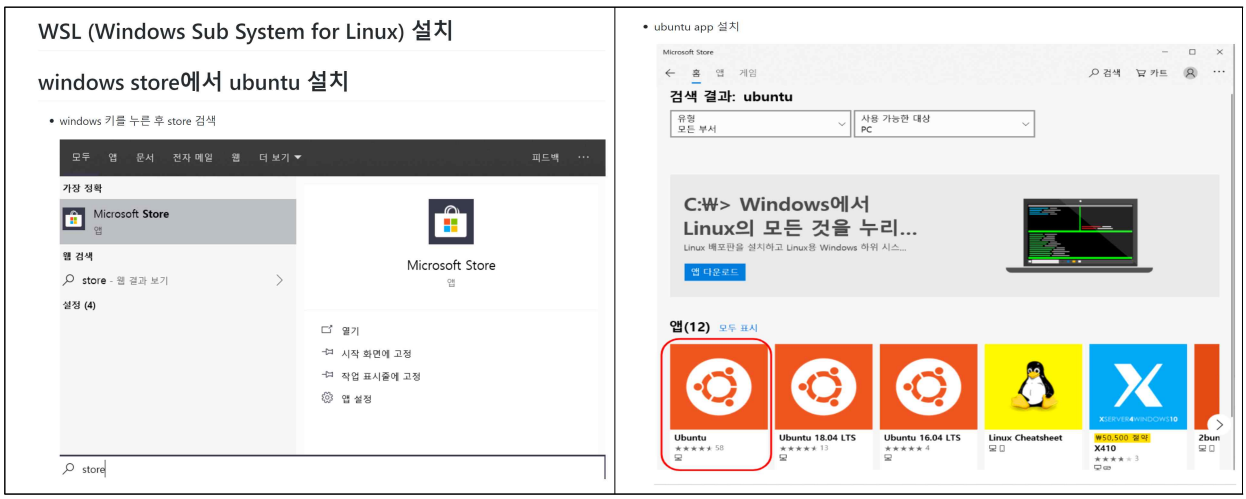
(1) 윈도우즈 환경에서 Linux 사용을 위해 WSL 설치[14].

* WSL 설치 방법은 다음 사이트(URL)를 참조해도 된다.

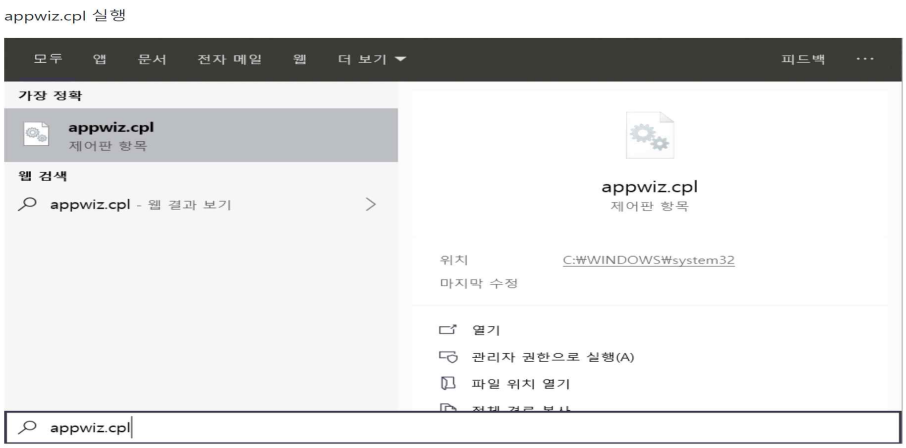
https://github.com/iot-standards-laboratory/Lecture.netProgramming

https://youtu.be/vtSzxkDM7rw

- windows store에서 ubuntu 설치 (windows 키를 누른 후 store 검색)

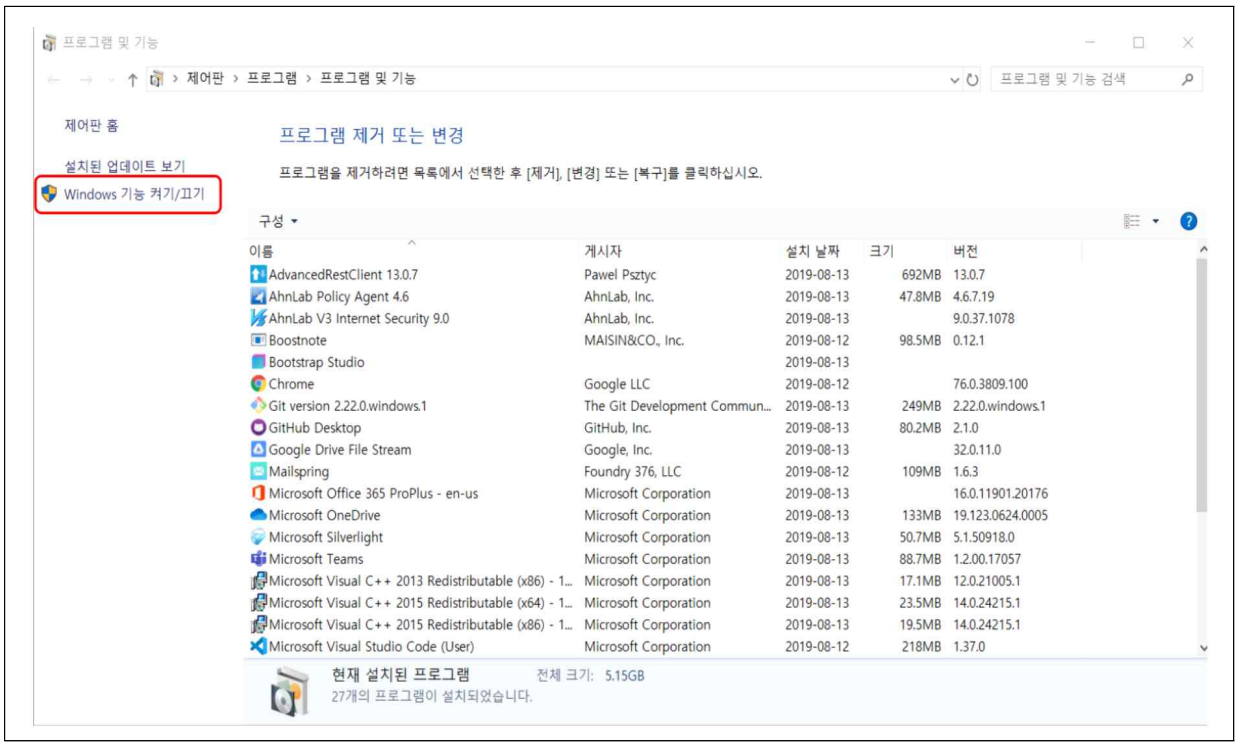


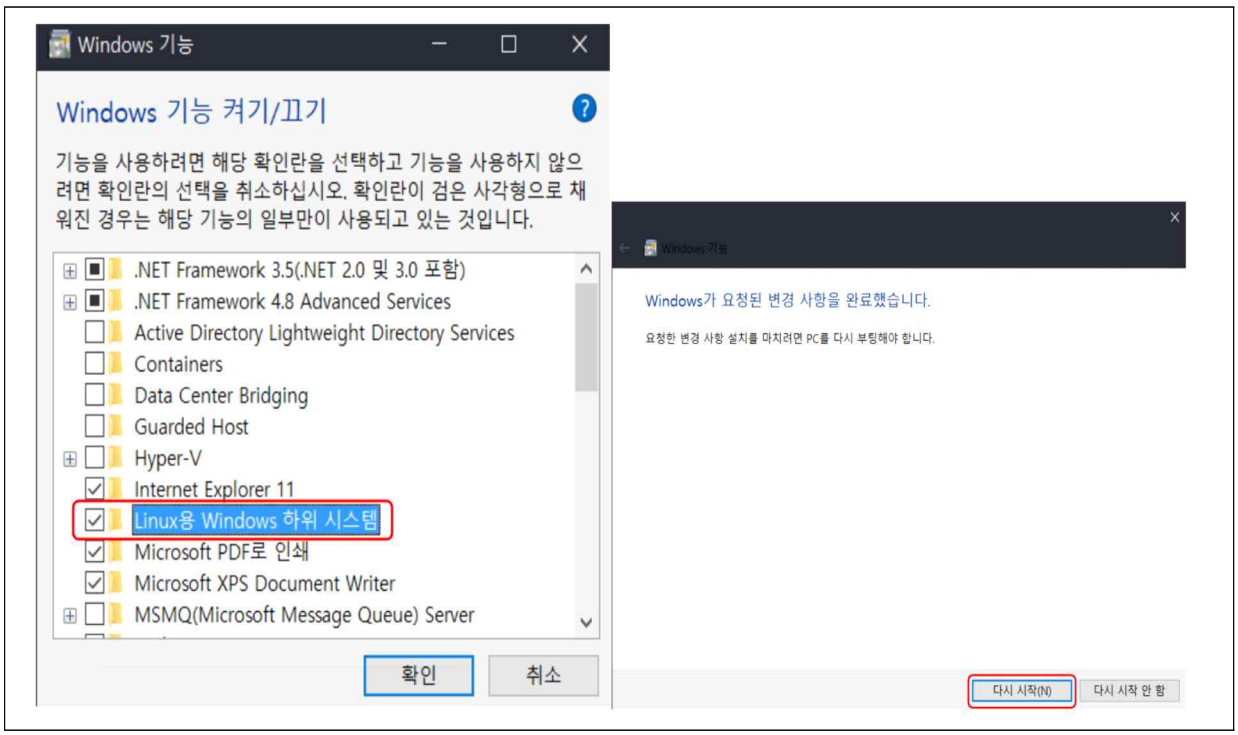
- appwiz.cpl 실행



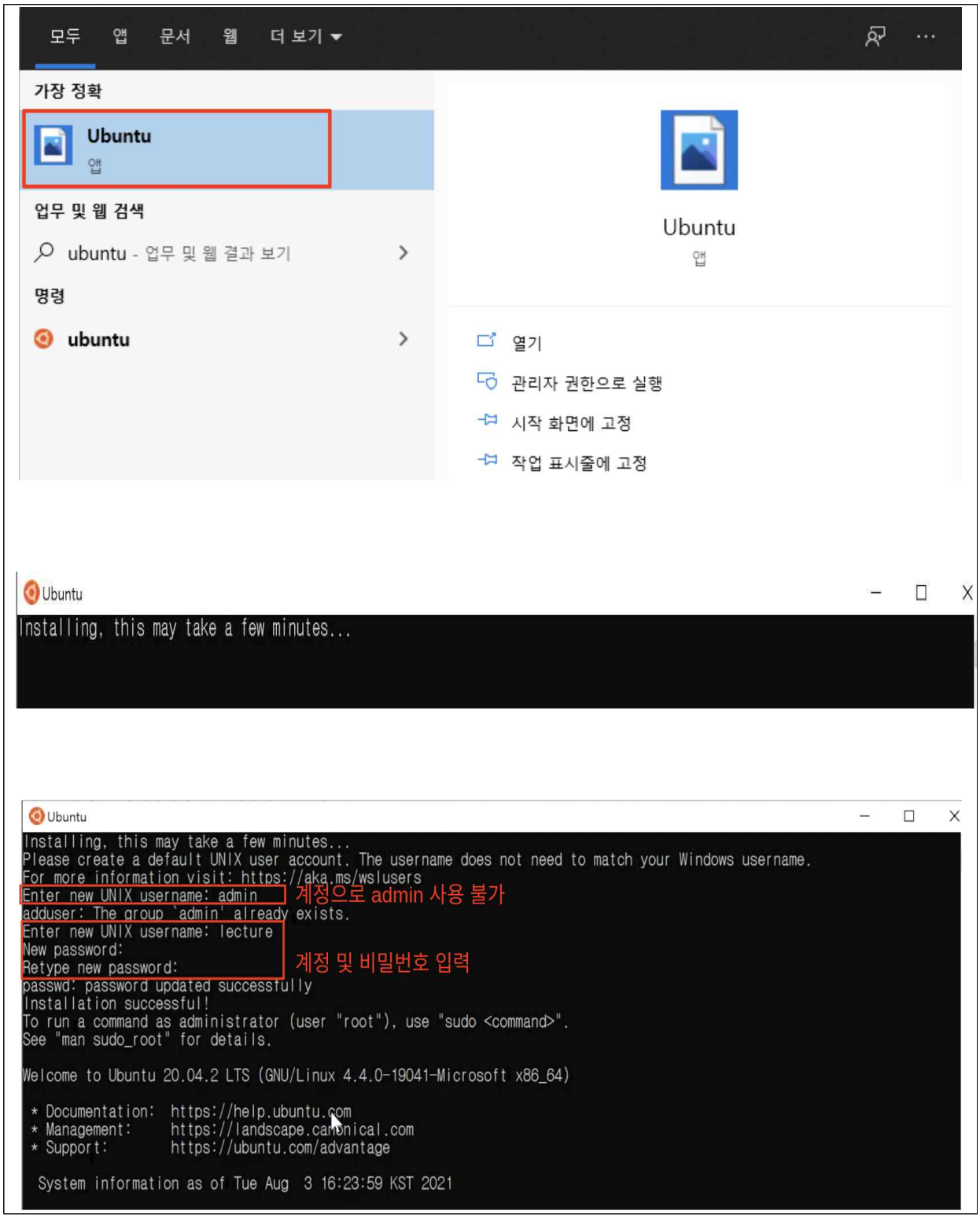
14) 이미 Linux 혹은 VMware 등이 설치되어 있다면, WSL 설치 없이 바로 실습(실행) 가능하다.

- 프로그램 및 기능에서 Windows 기능 켜기/끄기 -> WSL 설치 (재부팅 필요)





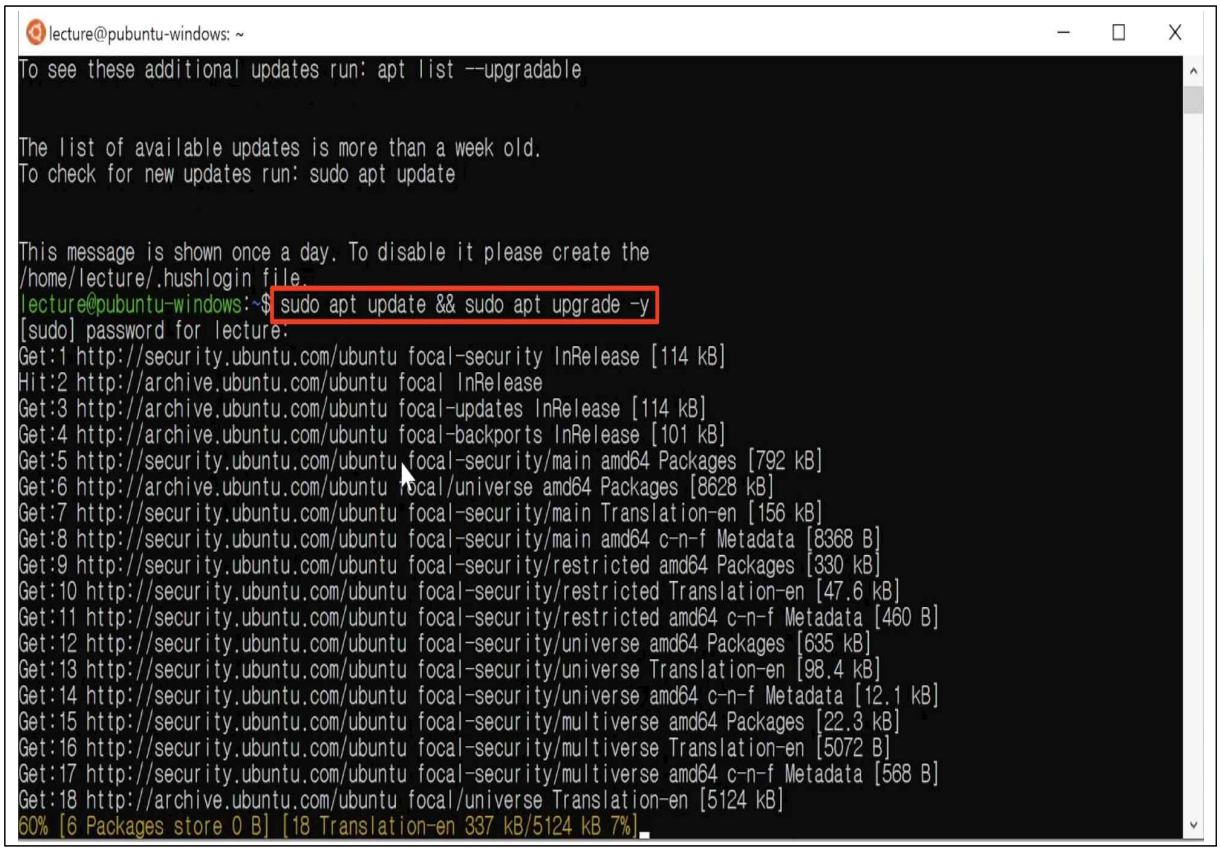
- ubuntu app 실행 및 ID와 Password 설정



- ubuntu update

  * 다음 명령어를 통해 ubuntu를 업데이트 한다.

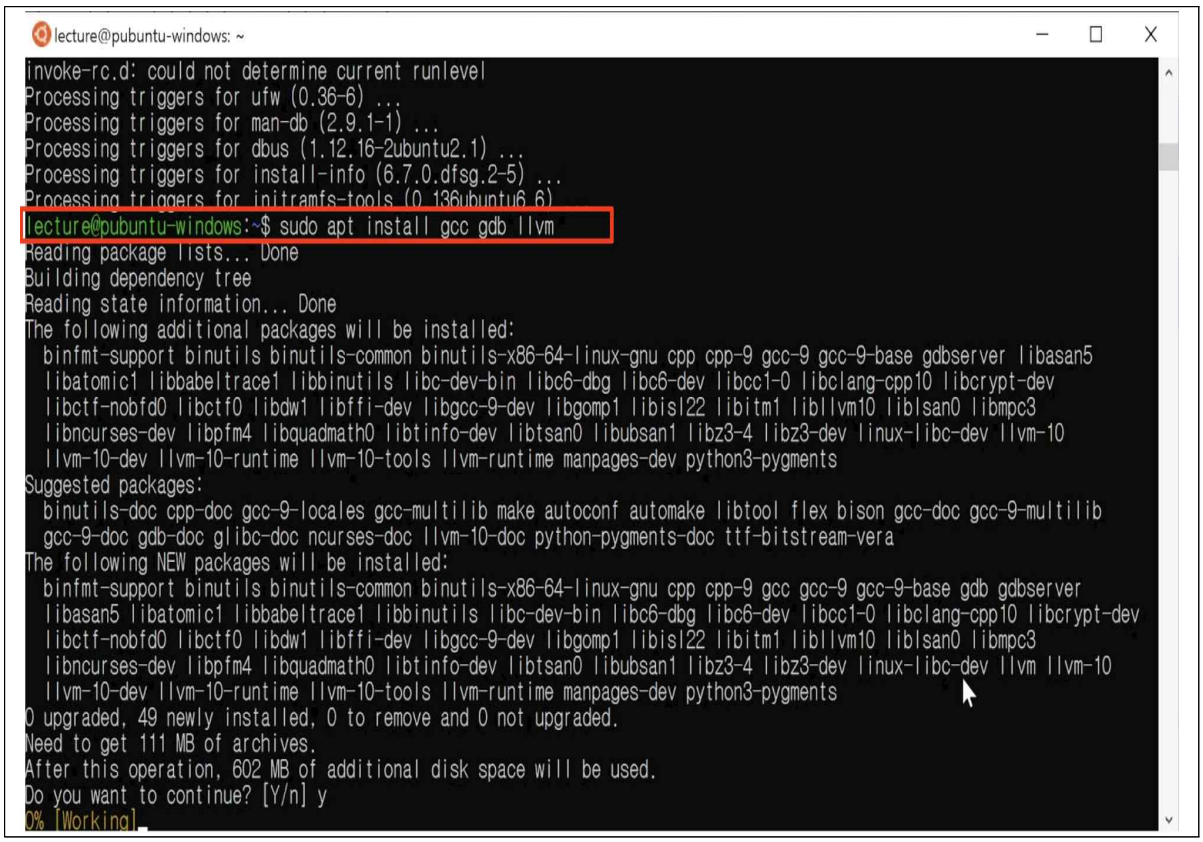
```
sudo apt-get update && sudo apt-get upgrade -y
```



(2) 컴파일러 설치

- 이제 소켓 프로그래밍 코드의 컴파일을 위해서 ubuntu app에서 gcc 컴파일러를 설치한다.

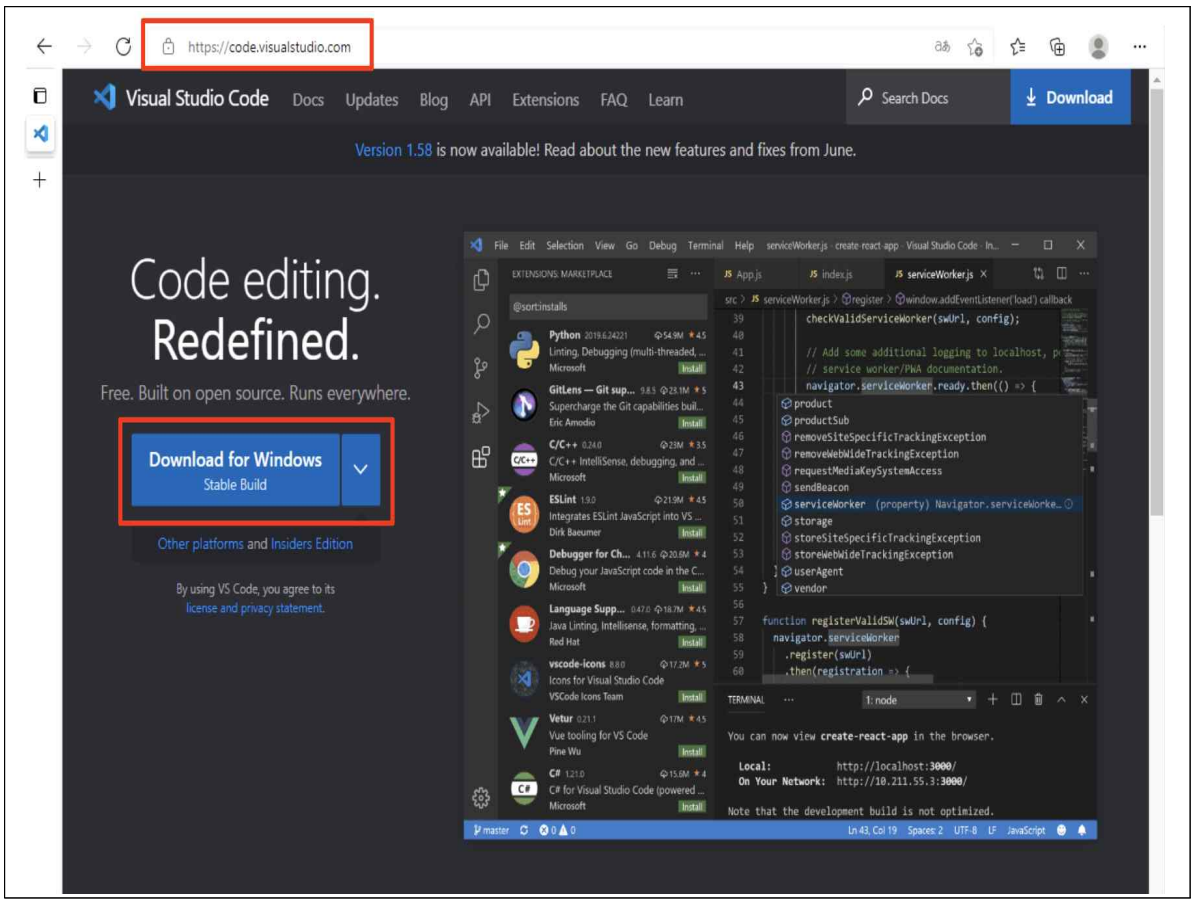
```
sudo apt-get install gcc gdb llvm
```



(3) vscode 설치 및 설정

- vscode는 마이크로소프트에서 제작한 code editor이자 개발도구로 아래의 사이트에서 자세한 사항을 확인할 수 있으며, 다운로드 받을 수도 있다.

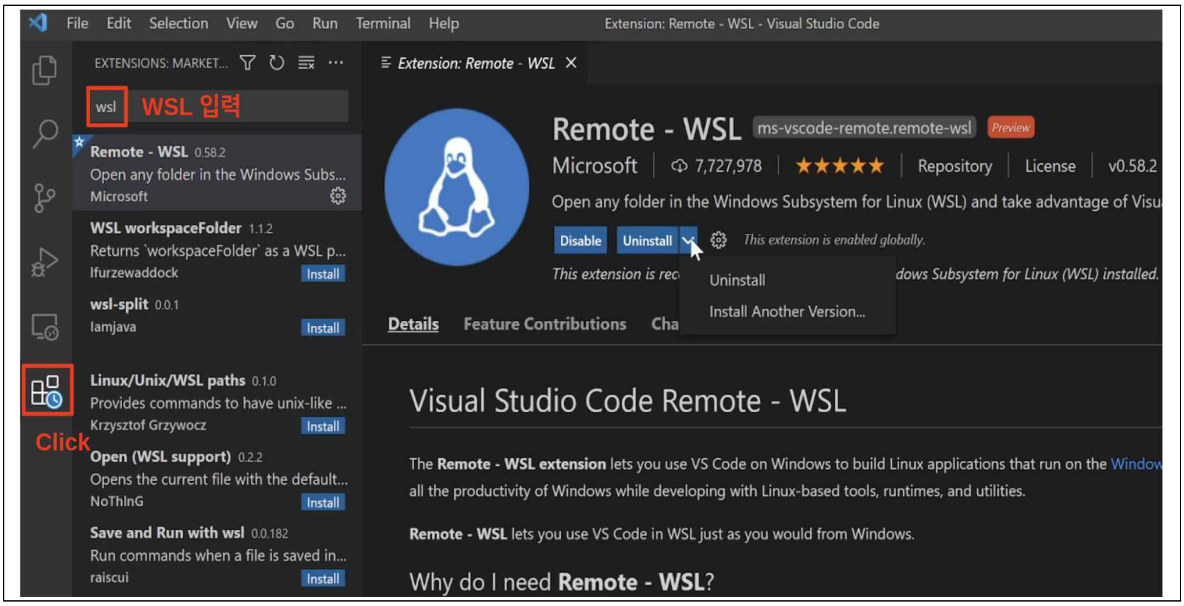
https://code.visualstudio.com



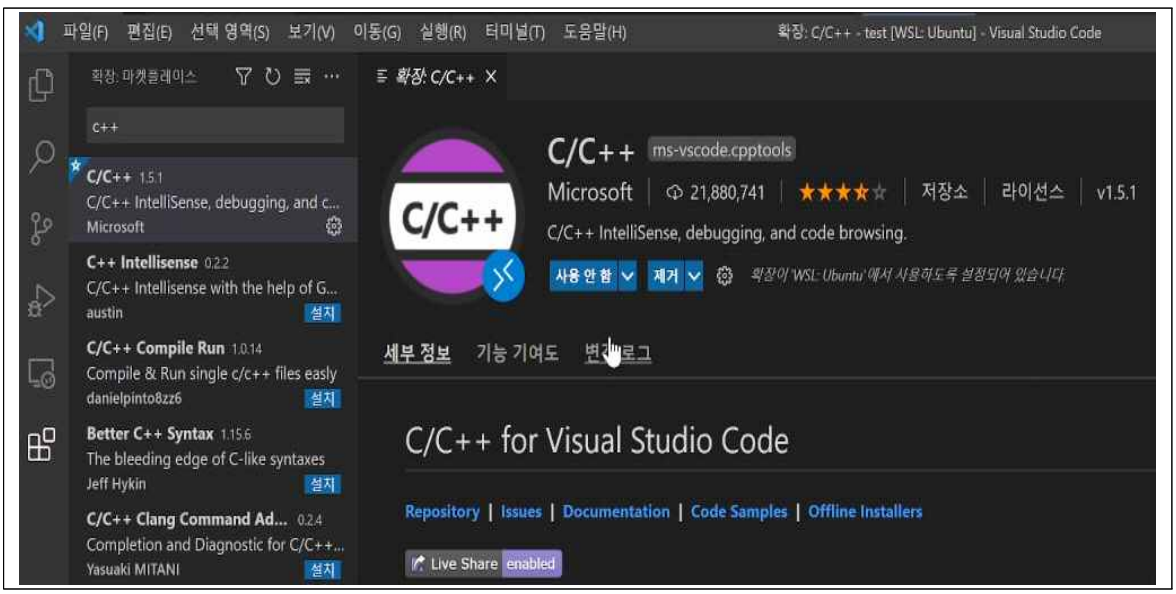
- vscode 설치가 완료되면 c/c++ 개발을 위한 설정을 수행한다.

- plugin 설치 - 다음 2개의 plugin을 설치한다.

  - Remote-WSL



  - c/c++ (WSL 실행 후 plugin 설치)



5. (실습2) 클라이언트-서버 실행하기

- WSL과 vscode를 설치하였으면 이제 앞에서 학습한 echo 클라이언트-서버 프로그램을 컴파일하고 실행시켜 보자.

(1) vscode를 사용하여 예제 코드 확인하기

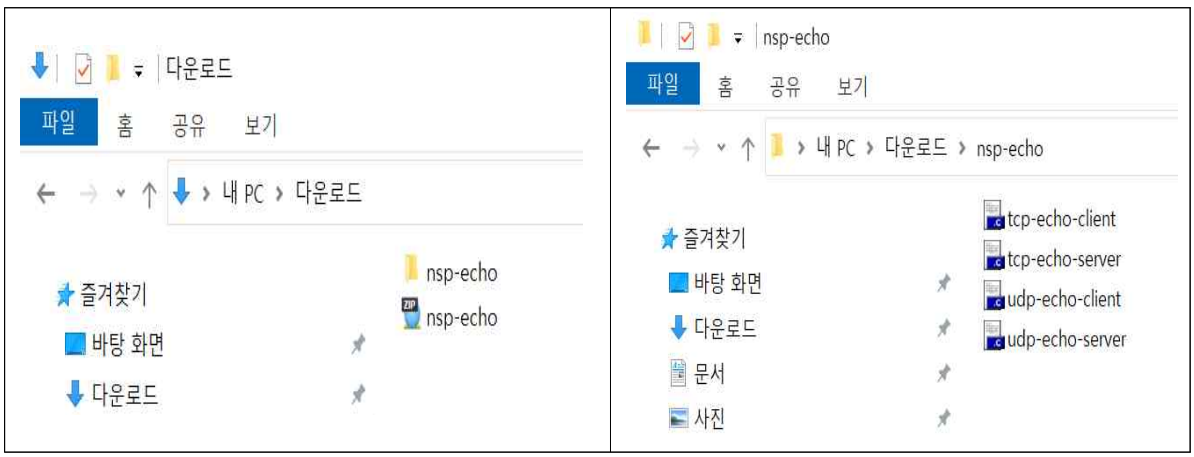
- 먼저 앞 절에서 보여준 UDP 및 TCP echo 클라이언트-서버 코드를 vscode를 실행시켜 확인해 보자

  - UDP 서버 및 클라이언트 코드: 18.2절의 (2), (3)

  - TCP 서버 및 클라이언트 코드: 18.3절의 (4), (5)

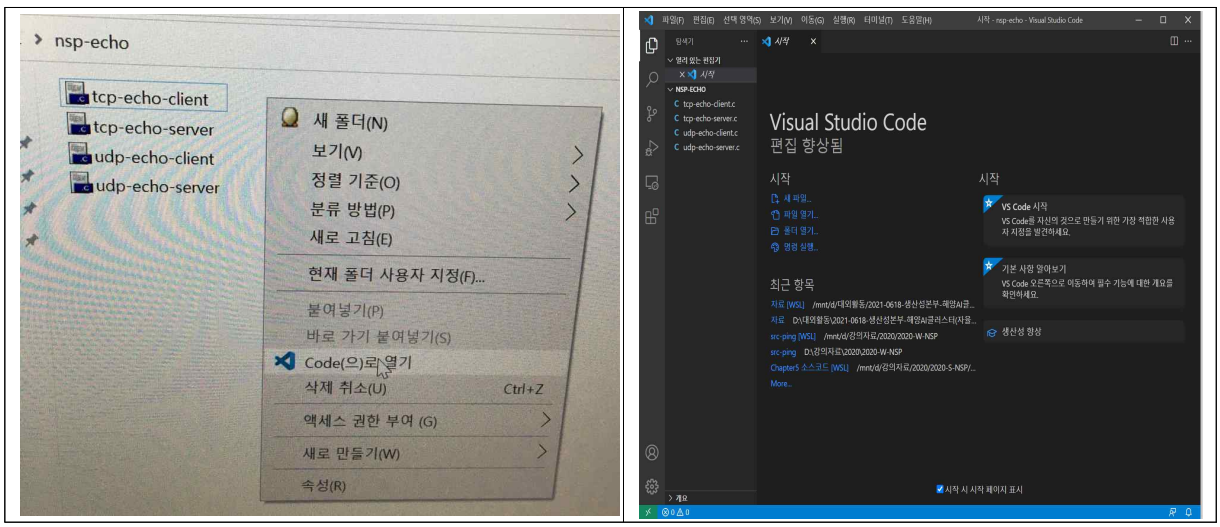
- 교재의 코드를 직접 입력하여 소소 코드(*.c)를 만들거나, 혹은 아래 사이트에서 해당 코드를 다운받는다. (4개의 소소 코드를 압축 파일 형태로 제공)

http://iot.knu.ac.kr/working/nsp-echo.zip



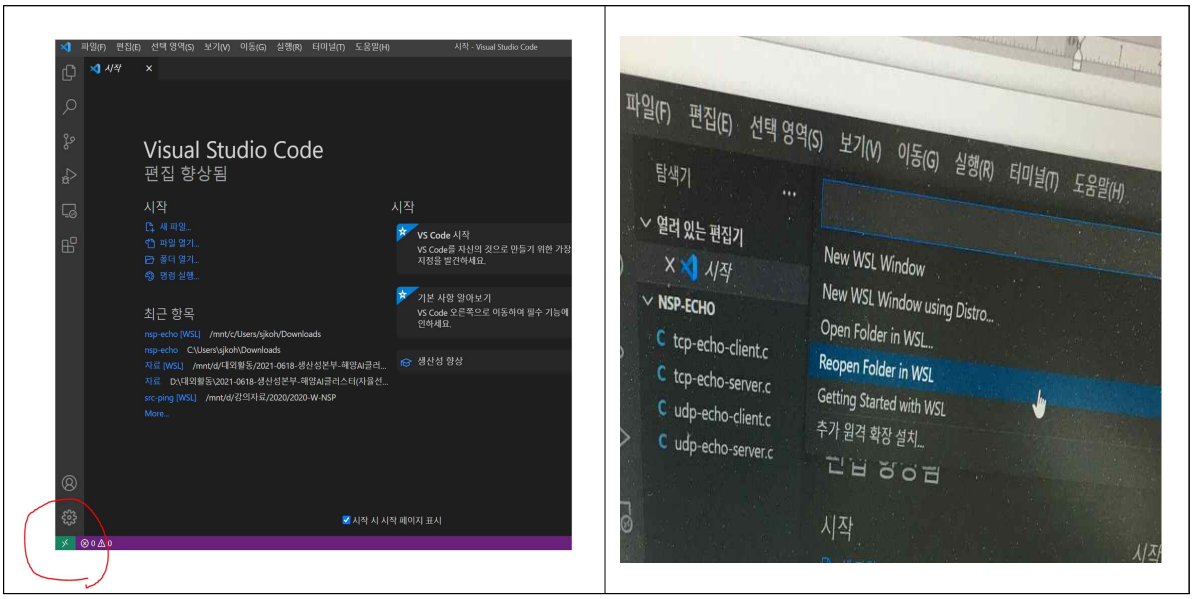
- 아래 그림처럼 소스코드 파일이 있는 폴더에서 vscode를 실행한다

  - 오른쪽 마우스 버튼 클릭 -> “Code로 열기” 클릭 (아래 그림)

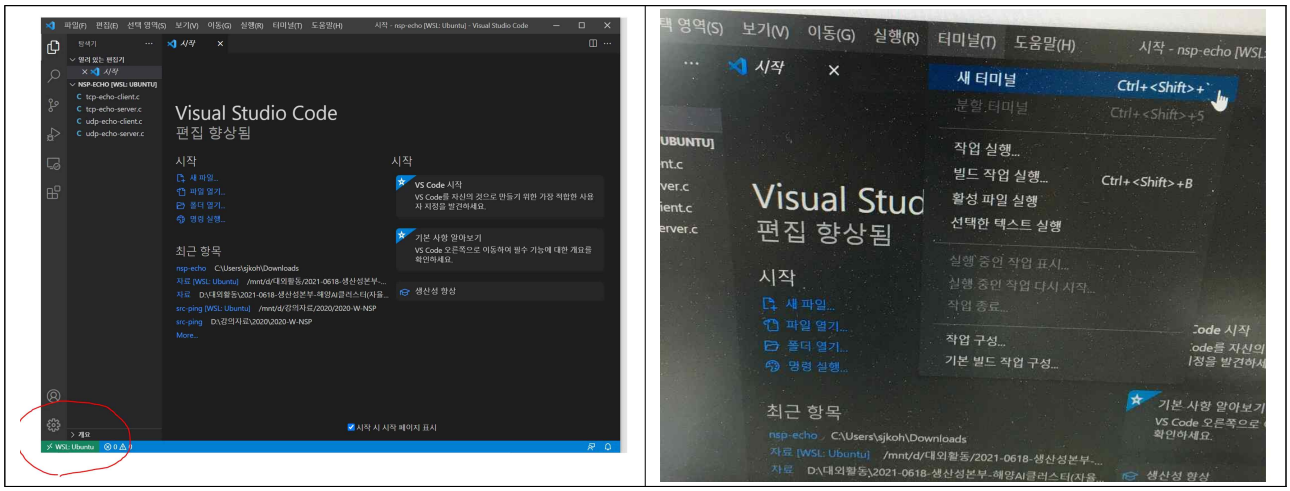


- 모니터 화면 맨 아래 왼쪽의 >< 버튼 클릭 (아래 그림 왼쪽 참조)

- “Reopen Folder in WSL” 클릭 (아래 그림 오른쪽 참조)

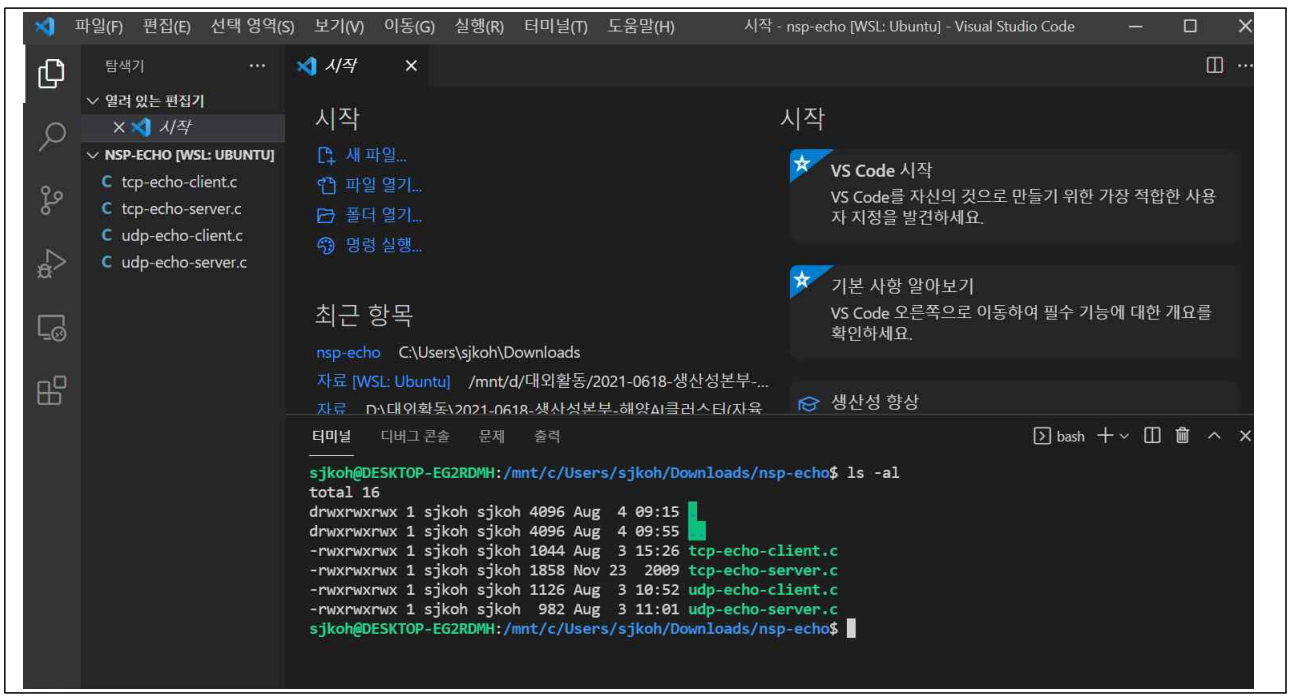


- 이제 작업 폴더에서 WSL이 동작함을 알수 있다. (아래 그림 왼쪽 참조)
- 화면의 상단 메뉴 중에서 "터미널"을 클릭하여 추가 작업을 위한 새로운 터미널 창을 오픈한다. (아래 그림 오른쪽 참조)



- 새로운 터미널 창이 열리면 ls -al 명령어를 입력한다

  - 폴더 안에 4개의 소스코드 파일이 있음을 확인한다.

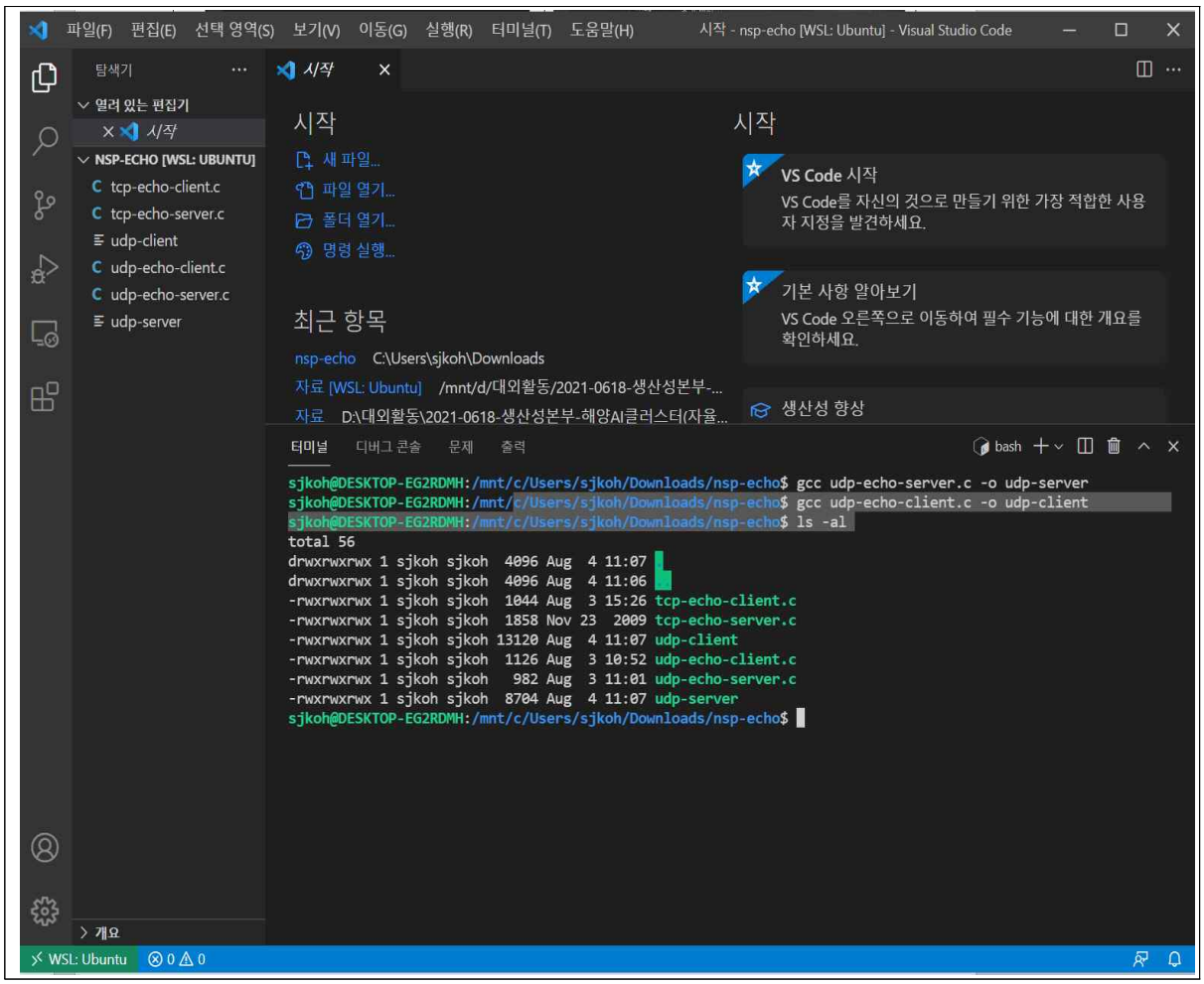


(2) UDP 서버 및 클라이언트 코드 실행하기

- UDP 서버와 클라이언트 코드를 각각 컴파일 한다.

```
nsp-echo$ gcc udp-echo-server.c -o udp-server
nsp-echo$ gcc udp-echo-client.c -o udp-client
nsp-echo$ ls -al
```

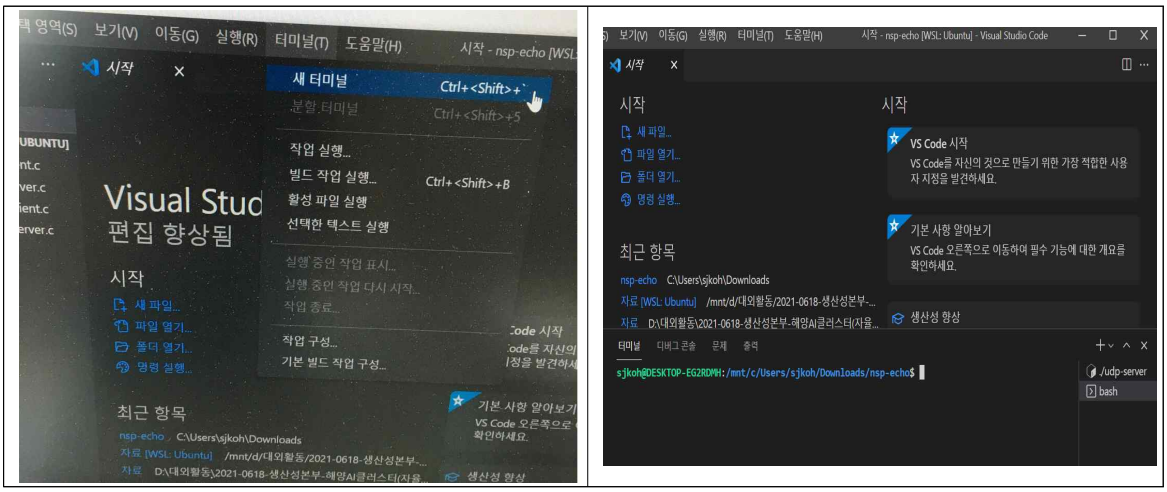
- 해당 폴더에 실행파일이 생성되었다. (udp-client, udp-server)



- 먼저 서버를 실행한다. (포트번호는 임의로 지정 가능)
  - 실행시에 ./를 추가해야 함 (현재 폴더에서 실행하라는 의미)

```
nsp-echo$ ./udp-server 9090
```

- 이제 클라이언트 실행을 위해 새로운 터미널 창을 연다.
  - 서버 프로세스(프로그램)과 클라이언트 프로세스가 다르기 때문이다.



- 새로운 터미널 창에서 클라이언트 프로그램을 실행한다.

  - 서버의 IP 주소와 포트번호를 입력한다.

```
nsp-echo$ ./udp-client 127.0.0.1 9090
```

- **IP 주소 "127.0.0.1"은 loopback 주소**라고 하며, 클라이언트와 서버가 같은 컴퓨터에 있는 경우에 사용한다.

- 이후 데이터를 입력하면 서버로부터 echo 메시지가 돌아와 출력된다.

  - 클라이언트를 종료하기 위해서는 q 혹은 CTRL-C를 누른다.

```
sjkoh@DESKTOP-EG2RDMH:/mnt/c/Users/sjkoh/Downloads/nsp-echo$ ./udp-client 127.0.0.1 9090
Insert message(q to quit): hello
Message from server: hello
Insert message(q to quit): nice to meet you!
Message from server: nice to meet you!
Insert message(q to quit): q
sjkoh@DESKTOP-EG2RDMH:/mnt/c/Users/sjkoh/Downloads/nsp-echo$
```

- 현재 실행 중인 프로세스를 확인하기 위해 "ps x" 입력(새로운 창에서)

```
sjkoh@DESKTOP-EG2RDMH:/mnt/c/Users/sjkoh/Downloads/nsp-echo$ ps x
  PID TTY      STAT   TIME COMMAND
   13 tty1     S      0:00 sh -c "$VSCODE_WSL_EXT_LOCATION/scripts/wslServer.sh" c3f126316
   14 tty1     S      0:00 sh /mnt/c/Users/sjkoh/.vscode/extensions/ms-vscode-remote.remot
   20 tty1     S      0:00 sh /home/sjkoh/.vscode-server/bin/c3f126316369cd610563c75b1b172
   22 tty1     Rl     0:02 /home/sjkoh/.vscode-server/bin/c3f126316369cd610563c75b1b1725e0
   33 tty1     Rl     0:00 /home/sjkoh/.vscode-server/bin/c3f126316369cd610563c75b1b1725e0
   44 tty1     Sl     0:00 /home/sjkoh/.vscode-server/bin/c3f126316369cd610563c75b1b1725e0
   86 tty1     Sl     0:05 /home/sjkoh/.vscode-server/bin/c3f126316369cd610563c75b1b1725e0
   97 pts/0    Ss     0:00 /bin/bash
  127 pts/0    S      0:00 ./udp-server 9090
  128 pts/1    Ss     0:00 /bin/bash
  142 pts/1    S      0:00 ./udp-client 127.0.0.1 9090
  143 pts/2    Ss     0:00 /bin/bash
  152 pts/2    R      0:00 ps x
sjkoh@DESKTOP-EG2RDMH:/mnt/c/Users/sjkoh/Downloads/nsp-echo$
```

(3) TCP 서버 및 클라이언트 코드 실행하기

- TCP 서버와 클라이언트 코드를 각각 컴파일 한다.

```
nsp-echo$ gcc tcp-echo-server.c -o tcp-server
nsp-echo$ gcc tcp-echo-client.c -o tcp-client
```

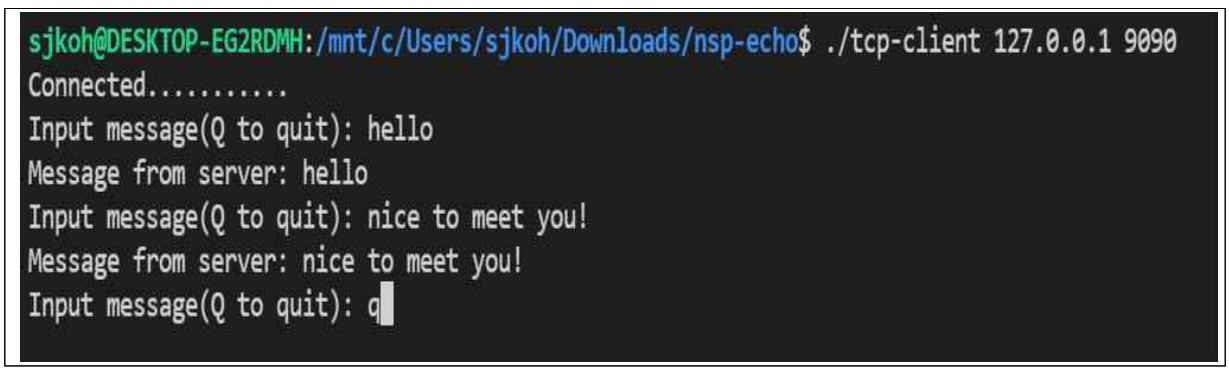
- 먼저 서버를 실행한다. (포트번호는 임의로 지정 가능)

```
nsp-echo$ ./tcp-server 9090
```

- 클라이언트 실행을 위해 **새로운 터미널 창을 열고**, client 프로그램을 실행한다.

```
nsp-echo$ ./tcp-client 127.0.0.1 9090
```

- 이후 데이터를 입력하면 서버로부터 echo 메시지가 돌아와 출력된다.



- **TCP 다중 프로세스 서버 동작을 확인**하기 위해 여러 개의 클라이언트 프로그램을 실행시킨다

  - **각 클라이언트 프로그램을 '새로운 터미널 창'에서 실행**한다.

```
sjkoh@DESKTOP-EG2RDMH:/mnt/c/Users/sjkoh/Downloads/nsp-echo$ ./tcp-client 127.0.0.1 9090
Connected..........
Input message(Q to quit):
```

```
sjkoh@DESKTOP-EG2RDMH:/mnt/c/Users/sjkoh/Downloads/nsp-echo$ ./tcp-client 127.0.0.1 9090
Connected..........
Input message(Q to quit):
```

```
sjkoh@DESKTOP-EG2RDMH:/mnt/c/Users/sjkoh/Downloads/nsp-echo$ ./tcp-client 127.0.0.1 9090
Connected..........
Input message(Q to quit):
```

```
sjkoh@DESKTOP-EG2RDMH:/mnt/c/Users/sjkoh/Downloads/nsp-echo$ ./tcp-client 127.0.0.1 9090
Connected..........
Input message(Q to quit):
```

- 여러 클라이언트 프로세스를 확인하기 위해 "ps x" 입력(새로운 창에서)

```
sjkoh@DESKTOP-EG2RDMH:/mnt/c/Users/sjkoh/Downloads/nsp-echo$ ps x
  PID TTY      STAT   TIME COMMAND
   13 tty1     S      0:00 sh -c "$VSCODE_WSL_EXT_LOCATION/scripts/wslServer.sh" c3f126316
   14 tty1     S      0:00 sh /mnt/c/Users/sjkoh/.vscode/extensions/ms-vscode-remote.remot
   20 tty1     S      0:00 sh /home/sjkoh/.vscode-server/bin/c3f126316369cd610563c75b1b172
   22 tty1     Rl     0:03 /home/sjkoh/.vscode-server/bin/c3f126316369cd610563c75b1b1725e0
   33 tty1     Rl     0:01 /home/sjkoh/.vscode-server/bin/c3f126316369cd610563c75b1b1725e0
   44 tty1     Sl     0:00 /home/sjkoh/.vscode-server/bin/c3f126316369cd610563c75b1b1725e0
   86 tty1     Sl     0:05 /home/sjkoh/.vscode-server/bin/c3f126316369cd610563c75b1b1725e0
   97 pts/0    Ss     0:00 /bin/bash
  128 pts/1    Ss     0:00 /bin/bash
  143 pts/2    Ss     0:00 /bin/bash
  167 pts/3    Ss     0:00 /bin/bash
  176 pts/3    S      0:00 ./tcp-client 127.0.0.1 9090
  178 pts/4    Ss     0:00 /bin/bash
  187 pts/4    S      0:00 ./tcp-client 127.0.0.1 9090
  188 pts/2    S      0:00 ./tcp-server 9090
  189 pts/5    Ss     0:00 /bin/bash
  204 pts/5    S      0:00 ./tcp-client 127.0.0.1 9090
  205 pts/6    Ss     0:00 /bin/bash
  214 pts/6    R      0:00 ps x
sjkoh@DESKTOP-EG2RDMH:/mnt/c/Users/sjkoh/Downloads/nsp-echo$
```