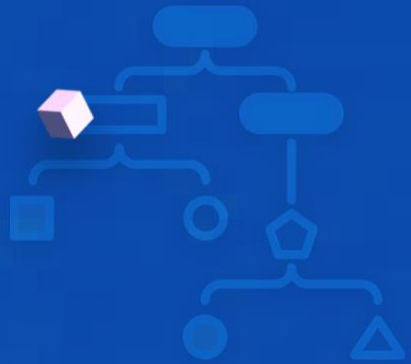
강원혁신플랫폼

리눅스프로그래밍

UDP 네트워크 프로그래밍



Linux 네트워크 프로그래밍에서 빠르고
지연 시간이 짧은 통신에 일반적으로 사용되며,
연결성이 없고 신뢰할 수 없는 전송 프로토콜은 무엇인가요?

UDP
(User Datagram Protocol)

학습목표

학습 내용

1. UDP 서버와 클라이언트의 통신 설정
2. socket(), bind(), htonl(), sendto(), recvfrom(), memset()
3. 빅 엔디안과 리틀 엔디안

학습 목표

- UDP 서버와 클라이언트의 통신 설정에 대해 설명할 수 있다.
- socket(), bind() 등의 함수에 대해 설명할 수 있다.
- 빅 엔디안과 리틀 엔디안에 대해 설명할 수 있다.

UDP 서버와 클라이언트의 통신 설정

UDP(User Datagram Protocol)

네트워크 프로그래밍에서 빠르고 지연 시간이 짧은 통신에 일반적으로 사용

연결이 없고 신뢰할 수 없는 전송 프로토콜

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

UDP 네트워크 프로그래밍에 필요한 함수

01 02 03 04

소켓 생성

- UDP 연결을 설정하려면 socket() 시스템 호출을 사용하여 UDP 소켓을 생성해야 함

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

UDP 네트워크 프로그래밍에 필요한 함수

01 02 03 04

주소 바인딩(선택 사항)

- 서버 응용 프로그램을 개발하는 경우 bind() 시스템 호출을 사용하여 소켓을 특정 IP 주소 및 포트에 바인딩함

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

UDP 네트워크 프로그래밍에 필요한 함수

01 02 03 04

데이터 보내기

- UDP를 통해 데이터를 보내려면 sendto() 또는 sendmsg() 시스템 호출을 사용함

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

UDP 네트워크 프로그래밍에 필요한 함수

01 02 03 04

데이터 수신

- UDP 소켓에서 데이터를 수신하려면 recvfrom() 또는 recvmsg() 시스템 호출을 사용함

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

UDP 네트워크 프로그래밍의 특징

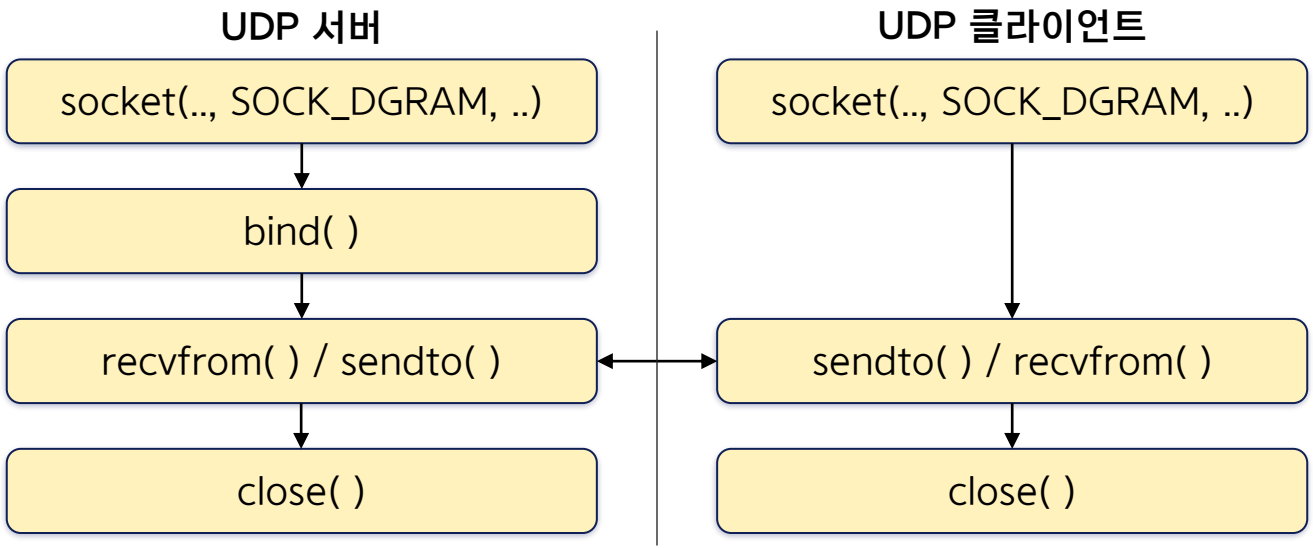
01 • 비연결형 프로토콜이므로 패킷 전달이나 순서가 보장되지 않음

02 • 짧은 대기 시간이 필요하고 간헐적인 데이터 손실을 허용할 수 있는 애플리케이션에 적합

03 • 필요한 경우 응용 프로그램의 안정성을 보장하려면 적절한 오류 검사 및 처리 메커니즘을 구현

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

UDP 서버와 클라이언트의 통신 설정



출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

socket(), bind(), htonl(), sendto(), recvfrom(), memset()

socket(), bind()

socket(): UDP 소켓을 생성하고 소켓 설명자를 반환

```
int socket_fd = socket(AF_INET, SOCK_DGRAM, 0);
```

bind(): UDP 소켓을 특정 로컬 주소 및 포트에 바인딩

```
struct sockaddr_in server_addr;
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = INADDR_ANY; // 또는 특정 IP 주소 지정
server_addr.sin_port = htons(1234); // 포트 번호 지정

int bind_result = bind(socket_fd, (struct sockaddr*)&server_addr,
sizeof(server_addr));
```

htonl(), sendto(), recvfrom(), memset()

htonl()(Host to Network Long): 32비트 부호 없는 정수를
호스트 바이트 순서에서 네트워크 바이트 순서(big-endian)로 변환

- 일반적으로 IP 주소와 포트 번호를 변환하는 데 사용
- uint32_t ip_address = htonl(0xC0A80101); // 192.168.1.1을 네트워크 바이트 순서로 변환

htonl(), sendto(), recvfrom(), memset()

sendto(): UDP 패킷을 특정 대상으로 전송

```
int bytes_sent = sendto(socket_fd, data_buffer, data_length, 0, (struct sockaddr*)&dest_addr, sizeof(dest_addr));
```

htonl(), sendto(), recvfrom(), memset()

recvfrom(): 네트워크에서 UDP 패킷을 수신

```
int bytes_received = recvfrom(socket_fd, recv_buffer, recv_buffer_size, 0,
(struct sockaddr*)&src_addr, &addr_len);
```

memset(): 메모리 블록을 특정 값으로 설정하는 데 사용

```
memset(data_buffer, 0, sizeof(data_buffer)); // 모든 바이트를 0으로 설정하여 data_buffer를 지움
```

소켓의 도메인

도메인	내용	비고
PF_INET, AF_INET	IPv4 인터넷 프로토콜을 사용함	
PF_INET6, AF_INET6	IPv6 인터넷 프로토콜을 사용함	
PF_LOCAL, AF_LOCAL	같은 유닉스 시스템 내에서 프로세스끼리 통신함	PF_UNIX, AF_UNIX
PF_PACKET, AF_PACKET	저수준 소켓 인터페이스를 이용함	SOCK_PACKET
PF_NS, AF_NS	제록스 네트워크 시스템 프로토콜을 사용함	
PF_IPX, AF_IPX	노벨의 IPX 프로토콜을 사용함	
PF_APPLETALK, AF_APPLETALK	애플의 AppleTalk DDS 프로토콜을 사용함	

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

소켓의 타입

타입	상수	내용	비고
스트림 소켓 (stream socket)	SOCK_STREAM	연결 지향형의 TCP를 기반 소켓 간의 연결 후 데이터 전송	1
데이터그램 소켓 (datagram socket)	SOCK_DGRAM	비연결형의 UDP를 기반 송수신 시 도착지 주소 필수	2
Raw 소켓 (raw socket)	SOCK_RAW	저수준 프로토콜 접근 IP 계층을 이용하며 ICMP, OSPF 등이 사용함	3

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

소켓에서 사용하는 주소

주소	내용
sockaddr	일반 소켓 주소 구조
sockaddr_un (유닉스 도메인 소켓)	동일한 시스템에서 Unix 도메인 소켓을 주소 지정하는 데 사용
sockaddr_in (IPv4 소켓)	인터넷 통신에 사용되는 IPv4 소켓 주소 지정에 사용

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

소켓에서 사용하는 주소

sockaddr 구조체(16바이트)

sa_len	sa_family	sa_data
2	2	14

sockaddr_in 구조체(16바이트)

sin_len	sin_family	sin_port	sin_addr	sin_zero
2	2	2	4	8

sockaddr_un 구조체(가변 길이)

sun_len	sun_family	sun_path
2	2	104

주소체계 **주소**

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

빅 엔디안과 리틀 엔디안

빅 엔디안과 리틀 엔디안

01 02 03

엔디안

- 멀티바이트 데이터 유형(예: 정수)이 메모리에 저장되는 바이트 순서

빅 엔디안과 리틀 엔디안

01 02 03

빅 엔디안

- 바이트 순서에서 최상위 바이트(MSB)는 최하위 메모리 주소에 저장되고 최하위 바이트(LSB)는 최상위 메모리 주소에 저장 (Network Byte Order)
- 주소: 0x100 0x101 0x102 0x103
- 값: 0x12 0x34 0x56 0x78

빅 엔디안과 리틀 엔디안

01 02 03

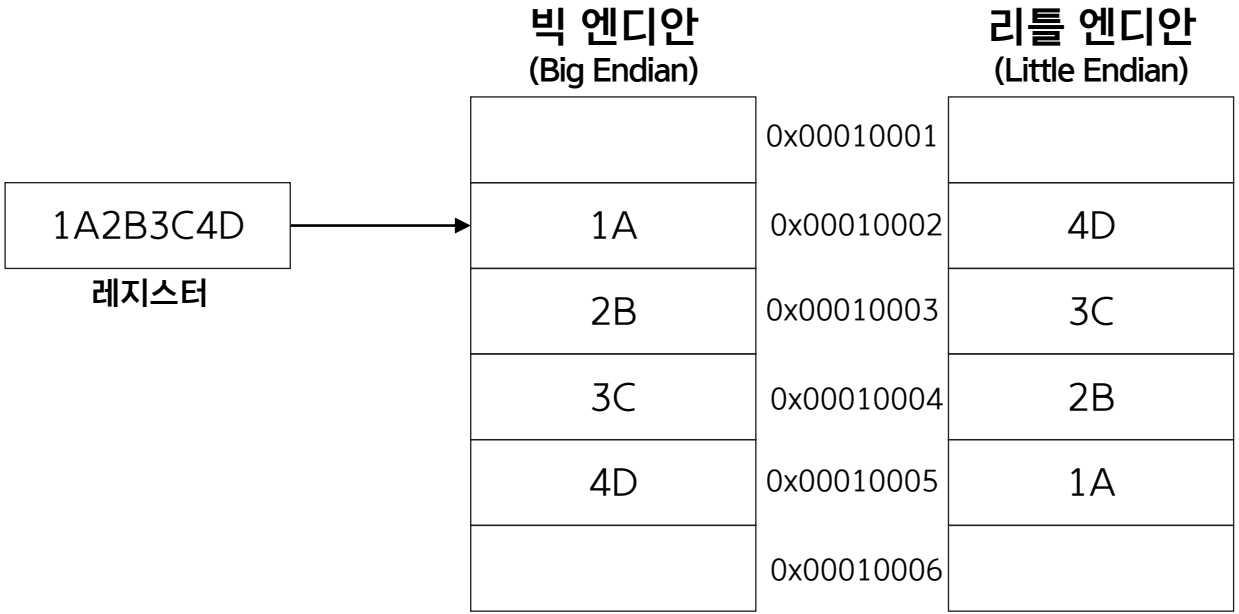
리틀 엔디안

- 바이트 순서에서 최하위 바이트(LSB)는 가장 낮은 메모리 주소에 저장되고 최상위 바이트(MSB)는 가장 높은 메모리 주소에 저장
- 주소: 0x100 0x101 0x102 0x103
- 값: 0x78 0x56 0x34 0x12

빅 엔디안과 리틀 엔디안

- <arpa/inet.h> 헤더의 htonl()(host to network long) 및 ntohl()(network to host long) 함수 사용
- x86 기반 프로세서(Linux 시스템에서 일반적으로 사용됨)를 포함한 **대부분의 최신 시스템은 리틀 엔디안 바이트 순서를 따름**

빅 엔디안과 리틀 엔디안



출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

리눅스의 네트워크 주소 변환 함수

함수	내용
inet_aton( ) 또는 inet_addr( )	문자열 형태의 IP 주소를 32비트 IP 주소로 변환함
inet_ntoa( )	32비트 IP 주소를 문자열 형태의 IP 주소로 변환
inet_network( )	IP 주소를 호스트 순서에서 네트워크 순서로 변환
inet_lnaof( )	IP 주소에서 호스트 주소 부분 추출
inet_netof( )	IP 주소에서 네트워크 주소 부분 추출

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

리눅스의 네트워크 주소 변환 함수

함수	내용
inet_makeaddr( )	분리된 호스트 주소와 네트워크 주소를 결합해서 IP 주소를 만듦
gethostbyname( )	도메인 주소로부터 IP 주소를 구함
gethostname( )	현재 프로세스가 실행되고 있는 호스트의 이름을 구함
sethostname( )	현재 프로세스가 실행되고 있는 호스트의 이름 설정

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

바이트 조작 함수

함수	내용	BSD 함수
void *memset(void *s, int c, size_t n);	바이트 영역을 특정 값으로 설정	두 번째 인자(c)가 00이면 bzero( )와 동일
void *memcpy(void *dest, const void *src, size_t n);	메모리를 복사함	bcopy( )
int memcmp(const void *s1, const void *s2, size_t n);	메모리를 비교함	bcmp( )

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

따라하기 udp_server.c


```
… 상부생략
#define UDP_PORT 5100

int main(int argc, char **argv) {
    int sockfd,n;    struct sockaddr_in servaddr, cliaddr;
    socklen_t len;    char mesg[1000];

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);    /* UDP를 위한 소켓 생성 */

    /* 접속되는 클라이언트를 위한 주소 설정 후 운영체제에 서비스 등록 */
    memset(&servaddr, 0, sizeof(servaddr));              servaddr.sin_family = AF_INET;
```

따라하기 udp_server.c


```
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port =    htons(UDP_PORT);
bind(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr));

/* 클라이언트로부터 메시지를 받아서 다시 클라이언트로 전송 */
do {
    len = sizeof(cliaddr);
    n = recvfrom(sockfd, mesg, 1000, 0, (struct sockaddr *)&cliaddr, &len);
    sendto(sockfd, mesg, n, 0, (struct sockaddr *)&cliaddr, sizeof(cliaddr));
    mesg[n] = '\0';
    printf("Received data : %s\n", mesg);
} while(strncmp(mesg, "q", 1));
```

```
    close(sockfd);                    /* 사용이 끝난 후 소켓 닫기 */
    return 0;
}
```

따라하기 udp_client.c


```
… 상부생략
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);    /* UDP를 위한 소켓 생성 */

    /* 서버의 주소와 포트 번호를 이용해서 주소 설정 */
    memset(&servaddr, 0, sizeof(servaddr));    servaddr.sin_family = AF_INET;

    /* 문자열을 네트워크 주소로 변경 */
    inet_pton(AF_INET, argv[1], &(servaddr.sin_addr.s_addr));    servaddr.sin_port =
htons(UDP_PORT);
```

따라하기 udp_client.c


```
    /* 키보드로부터 문자열을 입력받아 서버로 전송 */
    do {
        fgets(mesg, BUFSIZ, stdin);
        sendto(sockfd, mesg, strlen(mesg), 0, (struct sockaddr *)&servaddr,
        sizeof(servaddr));        clisize = sizeof(cliaddr);

        /* 서버로부터 데이터를 받아서 화면에 출력 */
        n = recvfrom(sockfd, mesg, BUFSIZ, 0, (struct sockaddr*) &cliaddr, &clisize);
        mesg[n] = '\0';        fputs(mesg, stdout);
    } while(strncmp(mesg, "q", 1));

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

따라하기 udp 실행결과

실행결과

```
$ gcc -o udp_server udp_server.c
$ gcc -o udp_client udp_client.c
$ ./udp_server
Received data : Hello World

Received data : Hello New World
```

따라하기 udp 실행결과

실행결과

새로운 터미털 창에서

$ ./udp_client

usage : ./udp_client <IP address>

$ ./udp_client localhost

Hello World

Hello World

Hello New World

Hello New World

요약정리

01 • UDP 서버와 클라이언트의 통신 설정

02 • socket(), bind(), htonl(), sendto(), recvfrom(), memset()

03 • 빅 엔디안과 리틀 엔디안