

4. UDP 소켓 프로그래밍

4.1 UDP 서버/클라이언트 구현

UDP의 특징 요약

항목	설명
전송 방식	비연결형, 단방향 메시지
신뢰성	없음 (손실 가능, 순서 뒤바뀔 수 있음)
헤더 크기	작음 (8바이트)
속도	빠름, 지연 시간 작음
사용 사례	DNS, 스트리밍, VoIP 등

1. UDP 서버 구조

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <unistd.h>
5  #include <arpa/inet.h>
6
7  #define PORT 8888
8  #define BUF_SIZE 1024
9
10 int main() {
11     int sock;
12     struct sockaddr_in server_addr, client_addr;
13     socklen_t client_len = sizeof(client_addr);
14     char buffer[BUF_SIZE];
15
16     // 1. 소켓 생성
17     sock = socket(AF_INET, SOCK_DGRAM, 0);
18
19     // 2. 주소 설정
20     memset(&server_addr, 0, sizeof(server_addr));
21     server_addr.sin_family = AF_INET;
22     server_addr.sin_addr.s_addr = INADDR_ANY;
23     server_addr.sin_port = htons(PORT);
24
25     // 3. 바인딩
26     bind(sock, (struct sockaddr*)&server_addr, sizeof(server_addr));
27
28     printf("UDP 서버 실행 중 (포트 %d)...\n", PORT);
29
30     // 4. 데이터 수신 및 응답
```

```

31     while (1) {
32         int n = recvfrom(sock, buffer, BUF_SIZE, 0,
33                         (struct sockaddr*)&client_addr, &client_len);
34         buffer[n] = '\0';
35
36         printf("받은 메시지: %s\n", buffer);
37         sendto(sock, buffer, strlen(buffer), 0,
38               (struct sockaddr*)&client_addr, client_len);
39     }
40
41     close(sock);
42     return 0;
43 }

```

✓ 2. UDP 클라이언트 구조

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <unistd.h>
5  #include <arpa/inet.h>
6
7  #define SERVER_IP "127.0.0.1"
8  #define PORT 8888
9  #define BUF_SIZE 1024
10
11 int main() {
12     int sock;
13     struct sockaddr_in server_addr;
14     char buffer[BUF_SIZE];
15
16     // 1. 소켓 생성
17     sock = socket(AF_INET, SOCK_DGRAM, 0);
18
19     // 2. 서버 주소 설정
20     memset(&server_addr, 0, sizeof(server_addr));
21     server_addr.sin_family = AF_INET;
22     server_addr.sin_port = htons(PORT);
23     inet_pton(AF_INET, SERVER_IP, &server_addr.sin_addr);
24
25     while (1) {
26         printf("보낼 메시지: ");
27         fgets(buffer, BUF_SIZE, stdin);
28
29         // 3. 서버에 데이터 전송
30         sendto(sock, buffer, strlen(buffer), 0,
31               (struct sockaddr*)&server_addr, sizeof(server_addr));
32
33         // 4. 서버로부터 응답 수신
34         int n = recvfrom(sock, buffer, BUF_SIZE, 0, NULL, NULL);
35         buffer[n] = '\0';

```

```

36     printf("서버 응답: %s\n", buffer);
37 }
38
39 close(sock);
40 return 0;
41 }

```

📌 3. 주의사항 및 팁

주제	설명
<code>recvfrom()</code>	보낸 클라이언트의 주소가 함께 전달됨
<code>sendto()</code>	매번 목적지 주소 명시
데이터 유실	손실 감지 및 재전송은 직접 구현해야 함
MTU 초과	1472바이트 이상이면 IP 조각화 발생 주의

🧪 테스트

서버 실행:

```

1 gcc -o udp_server udp_server.c
2 ./udp_server

```

클라이언트 실행:

```

1 gcc -o udp_client udp_client.c
2 ./udp_client

```

✅ 정리 요약

항목	TCP	UDP
연결 여부	연결 지향 (3-way handshake)	비연결형
API	<code>listen</code> , <code>accept</code> 등	없음 (<code>sendto</code> , <code>recvfrom</code>)
데이터 흐름	스트림	메시지 단위
사용 예	HTTP, SSH	DNS, VoIP

4.2 recvfrom, sendto 함수 이해

sendto() 함수

정의

```
1 ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,  
2               const struct sockaddr *dest_addr, socklen_t addrlen);
```

인자	설명
sockfd	UDP 소켓 파일 디스크립터
buf	보낼 데이터 버퍼
len	보낼 데이터 길이
flags	일반적으로 0
dest_addr	목적지 주소 (struct sockaddr_in*)
addrlen	주소 구조체의 크기 (sizeof(struct sockaddr_in))

반환값

- 성공: 전송한 바이트 수
- 실패: -1 반환

예제

```
1 sendto(sock, "PING", 4, 0, (struct sockaddr*)&server_addr, sizeof(server_addr));
```

recvfrom() 함수

정의

```
1 ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,  
2               struct sockaddr *src_addr, socklen_t *addrlen);
```

인자	설명
sockfd	수신 대기 중인 소켓 FD
buf	수신 데이터를 저장할 버퍼
len	버퍼 크기
flags	일반적으로 0

인자	설명
<code>src_addr</code>	데이터를 보낸 클라이언트의 주소 (출처 저장)
<code>addrlen</code>	주소 구조체의 크기 (입출력)

✓ 반환값

- 성공: 수신한 바이트 수
- 실패: `-1` 반환

🔧 예제

```

1 char buffer[1024];
2 struct sockaddr_in client;
3 socklen_t clen = sizeof(client);
4
5 int n = recvfrom(sock, buffer, sizeof(buffer), 0,
6                 (struct sockaddr*)&client, &clen);

```

- 클라이언트 주소는 `client` 에 저장됨
- 응답 보낼 때는 이 주소를 그대로 `sendto()` 에 재사용

✓ 주요 사용 시나리오

상황	<code>recvfrom()</code>	<code>sendto()</code>
UDP 서버	클라이언트 주소를 알아야 하므로 사용	응답을 보내기 위해 사용
UDP 클라이언트	보통 상대 주소 필요 없음 → NULL 가능	항상 목적지 주소 지정해야 함

```

1 | recvfrom(sock, buffer, sizeof(buffer), 0, NULL, NULL); // 주소 무시 가능

```

🚩 flags 옵션

- `MSG_DONTWAIT`: 블로킹 없이 호출
- `MSG_PEEK`: 데이터를 소비하지 않고 미리 보기

```

1 | recvfrom(sock, buffer, sizeof(buffer), MSG_PEEK, NULL, NULL); // 버퍼 유지됨

```

✓ 정리 요약

항목	sendto()	recvfrom()
동작	데이터를 전송	데이터를 수신
주소 필요	목적지 주소 필수	출처 주소 저장용
UDP 서버	클라이언트 응답 시 사용	클라이언트 주소 추출용
TCP에서	보통 사용 안 함 (stream 기반)	사용 안 함

4.3 클라이언트 주소 식별 및 처리

📌 왜 식별이 필요한가?

UDP는 상태를 유지하지 않아, 서버가 수신한 데이터가 누가 보냈는지 알 수 없으면 **응답을 보낼 수 없다**.

따라서 `recvfrom()` 호출 시 **발신자 주소를 함께 추출**해서 저장해두고, `sendto()` 호출 시 그 주소를 이용해서 다시 응답을 보내는 방식으로 작동해야 함.

■ 기본 예제 코드 (UDP 서버)

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <arpa/inet.h>
4  #include <sys/socket.h>
5  #include <unistd.h>
6
7  int main() {
8      int sock;
9      struct sockaddr_in server_addr, client_addr;
10     socklen_t client_len = sizeof(client_addr);
11     char buffer[1024];
12
13     sock = socket(AF_INET, SOCK_DGRAM, 0);
14
15     memset(&server_addr, 0, sizeof(server_addr));
16     server_addr.sin_family = AF_INET;
17     server_addr.sin_port = htons(8888);
18     server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
19
20     bind(sock, (struct sockaddr*)&server_addr, sizeof(server_addr));
21
22     while (1) {
23         int n = recvfrom(sock, buffer, sizeof(buffer), 0,
24                         (struct sockaddr*)&client_addr, &client_len);
25
26         buffer[n] = '\0';
27
28         printf("👤 Client IP: %s, Port: %d\n",
```

```

29         inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));
30         printf("📡 Received: %s\n", buffer);
31
32         // 응답 전송
33         sendto(sock, buffer, n, 0,
34               (struct sockaddr*)&client_addr, client_len);
35     }
36
37     close(sock);
38     return 0;
39 }

```

✅ 핵심 함수: `inet_ntoa()`, `ntohs()`

- `inet_ntoa(client_addr.sin_addr)` → IPv4 주소 문자열 반환
- `ntohs(client_addr.sin_port)` → 포트 번호를 호스트 바이트 순서로 변환

🧠 클라이언트별 응답 분기 처리 예시

```

1  if (strcmp(inet_ntoa(client_addr.sin_addr), "192.168.0.100") == 0) {
2      printf("특정 클라이언트 처리 로직 실행\n");
3  }

```

📦 응용: 클라이언트 주소를 문자열로 저장

```

1  char ip_str[INET_ADDRSTRLEN];
2  inet_ntop(AF_INET, &(client_addr.sin_addr), ip_str, sizeof(ip_str));
3
4  printf("클라이언트 IP 문자열: %s\n", ip_str);

```

IPv6의 경우 `AF_INET6` 와 `struct sockaddr_in6` 를 사용하면 됨.

🔒 주의할 점

- UDP는 상태가 없으므로, **매번 주소를 저장하고 참조**해야 한다.
- 클라이언트와의 상태 유지가 필요하다면, 주소별로 context를 **별도 구조체나 해시맵에 저장**하는 방식으로 구성해야 함.
- 동일 클라이언트라도 **포트가 바뀌면 다른 엔티티**로 간주된다.

✓ 요약

항목	내용
주소 추출	<code>recvfrom()</code> 의 <code>src_addr</code> 인자 사용
문자열 변환	<code>inet_ntoa()</code> , <code>inet_ntop()</code> 사용
포트 추출	<code>ntohs(client_addr.sin_port)</code>
응답 전송	<code>sendto()</code> 에서 추출된 주소 재사용
확장 처리	클라이언트별 구조체 관리로 상태 추적

4.4 UDP 패킷 손실/순서 문제 대응

🚨 UDP의 주요 한계

문제	설명
✗ 패킷 손실	중간에 패킷이 유실되어도 재전송하지 않음
✗ 순서 보장 없음	수신 순서가 송신 순서와 다를 수 있음
✗ 흐름 제어 없음	송신자가 과도하게 보내면 수신자가 버퍼 오버플로우 가능
✗ 중복 수신 가능성	같은 패킷이 두 번 도착할 수도 있음

✓ 일반적인 대응 전략

1. 시퀀스 번호 부여

```
1 struct packet {  
2     uint32_t seq_num;    // 시퀀스 번호  
3     char data[1024];    // 실제 데이터  
4 };
```

- 수신자는 `seq_num` 기준으로 정렬하거나 중복 제거 가능
- 송신자는 송신한 패킷을 `seq_num` 기준으로 재전송 관리 가능

2. ACK/NACK (응답) 기반 재전송 구현

- 클라이언트 → 서버로 전송 후, 서버가 `ACK(seq_num)` 응답
- 응답이 일정 시간 내 도착하지 않으면 클라이언트가 재전송


```

1 // sendto() 후 select()로 응답 대기
2 fd_set readfds;
3 struct timeval timeout = {2, 0}; // 2초 대기
4
5 FD_ZERO(&readfds);
6 FD_SET(sockfd, &readfds);
7
8 int ready = select(sockfd + 1, &readfds, NULL, NULL, &timeout);
9 if (ready == 0) {
10     // 타임아웃: 재전송 로직
11 }

```

3. 순서 재조립 버퍼 사용

- 도착한 패킷을 임시 버퍼에 저장
- `seq_num`이 누락된 경우 다음 패킷은 일단 보류
- 누락된 패킷 도착 시 재조립

```

1 // Pseudo-buffer: 버퍼[seq_num] = 데이터
2 char* reassembly_buffer[MAX_SEQ];

```

4. Sliding Window (슬라이딩 윈도우) 기법

- 여러 패킷을 연속해서 보내되, 일정 범위(window) 내에서만 ACK 확인
- TCP와 유사한 흐름 제어 가능
- 구현은 복잡하지만 성능/신뢰성 개선 효과 큼

5. 응용 계층에서 순서 보장 로직 작성

- 보낸 순서대로 큐 처리하고,
- 이전 시퀀스보다 낮은 번호는 무시하거나 버퍼에 보관
- 데이터베이스나 로그 처리 시 유용

예시 시퀀스 번호 포함 송수신 구조

송신 측

```

1 struct packet pkt;
2 pkt.seq_num = htonl(1);
3 strcpy(pkt.data, "Hello");
4
5 sendto(sock, &pkt, sizeof(pkt), 0,
6        (struct sockaddr*)&addr, sizeof(addr));

```

수신 측

```
1 struct packet recv_pkt;
2 recvfrom(sock, &recv_pkt, sizeof(recv_pkt), 0,
3           (struct sockaddr*)&client, &len);
4
5 uint32_t seq = ntohl(recv_pkt.seq_num);
6 printf("받은 시퀀스 번호: %u\n", seq);
```

✅ 실제 시스템에서의 예

시스템	해결 전략
VoIP	실시간성 중요 → 일부 손실 허용, 순서 정렬 사용
TFTP	단순 ACK 재전송 기반 신뢰성 확보
DNS	손실 시 클라이언트가 재요청
QUIC (UDP 기반)	자체 시퀀스/ACK/재전송 기능 구현 → TCP 수준 신뢰성 제공

🧠 요약

문제	대응 전략
패킷 손실	ACK/NACK + 재전송
순서 뒤바뀜	시퀀스 번호 + 재조립 버퍼
중복 수신	seq_num 중복 체크
흐름 제어	슬라이딩 윈도우 기법 도입
지연/무응답	타임아웃 + select/poll 사용

4.5 DNS 클라이언트 샘플 구현

📌 기본 개요


- 프로토콜: UDP 53번 포트
- 목적지: DNS 서버 (예: 8.8.8.8)
- 구조: DNS Query 패킷 생성 → UDP 전송 → 응답 수신 및 파싱

DNS 패킷 구조 (간략화)

필드	설명
Header (12 bytes)	ID, flags, question count 등
Question	질의 이름, 타입(A), 클래스(IN)
Answer, Authority, Additional	응답 세부 정보 (생략 가능)

주요 헤더 구조체

```
1 struct DNS_HEADER {
2     uint16_t id;        // ID
3     uint16_t flags;     // Query/Response flags
4     uint16_t q_count;   // 질문 수
5     uint16_t ans_count;
6     uint16_t auth_count;
7     uint16_t add_count;
8 };
9
10 struct QUESTION {
11     uint16_t qtype;
12     uint16_t qclass;
13 };
```

 도메인 이름 변환 함수 (예: "www.google.com" →
\3www\6goog1e\3com\0)

```
1 void ChangetoDnsNameFormat(unsigned char* dns, const char* host) {
2     int lock = 0, i;
3     strcat((char*)host, ".");
4     for (i = 0; i < strlen((char*)host); i++) {
5         if (host[i] == '.') {
6             *dns++ = i - lock;
7             for (; lock < i; lock++) {
8                 *dns++ = host[lock];
9             }
10            lock++;
11        }
12    }
13    *dns++ = 0; // 끝 표시
14 }
```

✓ 전체 예제 코드 (기본 DNS A 레코드 질의)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <unistd.h>
5  #include <arpa/inet.h>
6
7  #define DNS_PORT 53
8  #define BUF_SIZE 512
9
10 struct DNS_HEADER {
11     uint16_t id, flags, q_count, ans_count, auth_count, add_count;
12 };
13
14 struct QUESTION {
15     uint16_t qtype, qclass;
16 };
17
18 void ChangetoDnsNameFormat(unsigned char* dns, const char* host) {
19     int lock = 0, i;
20     strcat((char*)host, ".");
21     for (i = 0; i < strlen((char*)host); i++) {
22         if (host[i] == '.') {
23             *dns++ = i - lock;
24             for (; lock < i; lock++) *dns++ = host[lock];
25             lock++;
26         }
27     }
28     *dns++ = 0;
29 }
30
31 int main() {
32     unsigned char buf[BUF_SIZE], *qname;
33     struct DNS_HEADER* dns = NULL;
34     struct QUESTION* qinfo = NULL;
35
36     struct sockaddr_in dest;
37     int sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
38     if (sock < 0) { perror("socket"); exit(1); }
39
40     dest.sin_family = AF_INET;
41     dest.sin_port = htons(DNS_PORT);
42     dest.sin_addr.s_addr = inet_addr("8.8.8.8"); // Google DNS
43
44     dns = (struct DNS_HEADER*)&buf;
45
46     dns->id = (uint16_t) htons(getpid());
47     dns->flags = htons(0x0100); // 표준 쿼리
48     dns->q_count = htons(1);
49     dns->ans_count = dns->auth_count = dns->add_count = 0;
50 }
```

```

51     qname = &buf[sizeof(struct DNS_HEADER)];
52     ChangetoDnsNameFormat(qname, "example.com");
53
54     qinfo = (struct QUESTION*)&buf[sizeof(struct DNS_HEADER) + strlen((const
char*)qname) + 1];
55     qinfo->qtype = htons(1);    // A 레코드
56     qinfo->qclass = htons(1);  // IN
57
58     int len = sizeof(struct DNS_HEADER) + strlen((const char*)qname) + 1 +
sizeof(struct QUESTION);
59     sendto(sock, buf, len, 0, (struct sockaddr*)&dest, sizeof(dest));
60
61     socklen_t slen = sizeof(dest);
62     int rlen = recvfrom(sock, buf, BUF_SIZE, 0, (struct sockaddr*)&dest, &slen);
63
64     printf(" 📧 응답 수신됨 (%d 바이트)\n", rlen);
65     close(sock);
66     return 0;
67 }

```

🧠 해석 결과 보기

위 코드는 응답까지 받지만, 아직 응답 파싱은 안 함.

다음 단계에서는 `buf` 내 응답을 해석해서 A 레코드(IP) 추출 가능. 이건 바이너리 디코딩이 필요해서 따로 다뤄야 한다.

✂ 응용 팁

- `qtype` 을 28로 설정하면 IPv6 주소(AAAA 레코드) 요청 가능
- `recvfrom()` 응답은 DNS 응답 헤더 + 응답 리스트 → 추출 필요
- `tcpdump -i lo udp port 53` 으로 실제 전송 확인 가능