13. 실제 프로젝트 예제

13.1 채팅 서버/클라이언트 구현

C 언어 기반 TCP 소켓 프로그래밍의 대표적인 예제 중 하나는 **다중 클라이언트 채팅 서버**다.
이 예제를 통해 다음을 실전 구현하며 학습할 수 있다:

- `socket()`, `bind()`, `listen()`, `accept()` 함수의 연동
- 클라이언트 동시 접속 처리 (`select`, `pthread`, `fork` 방식 가능)
- 메시지 브로드캐스트 로직
- 연결 종료 및 에러 처리

✅ 1. 기본 구조 요약

```
[Client 1]---\         /---[Client N]
              \       /
           [ Chat Server ]
```

서버는 모든 클라이언트의 소켓을 관리하며,
→ 하나의 클라이언트로부터 수신된 메시지를
→ **전체 클라이언트에게 브로드캐스트**한다.

✅ 2. 요구사항 정리

- 클라이언트는 서버에 접속 후 메시지를 입력해 전송
- 서버는 해당 메시지를 모든 다른 클라이언트에게 전송
- `exit` 입력 시 클라이언트 종료
- 서버는 클라이언트 연결 해제 시 리소스 회수

✅ 3. 구현 방식

- 소켓은 `TCP (SOCK_STREAM)` 사용
- 다중 접속은 `select()` 함수 기반 (간단하고 직관적)

✅ 4. 서버 코드 (`chat_server.c`)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

```c
#include <sys/select.h>

#define PORT 12345
#define MAX_CLIENTS 10
#define BUF_SIZE 1024

int main() {
    int server_fd, client_fd, max_fd;
    int client_socks[MAX_CLIENTS] = {0};
    struct sockaddr_in serv_addr, cli_addr;
    socklen_t addr_len = sizeof(cli_addr);
    char buf[BUF_SIZE];

    fd_set read_fds;

    server_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (server_fd < 0) { perror("socket"); exit(1); }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(PORT);

    bind(server_fd, (struct sockaddr*)&serv_addr, sizeof(serv_addr));
    listen(server_fd, 5);

    printf("Chat server started on port %d\n", PORT);

    while (1) {
        FD_ZERO(&read_fds);
        FD_SET(server_fd, &read_fds);
        max_fd = server_fd;

        for (int i = 0; i < MAX_CLIENTS; i++) {
            if (client_socks[i] > 0) {
                FD_SET(client_socks[i], &read_fds);
                if (client_socks[i] > max_fd)
                    max_fd = client_socks[i];
            }
        }

        select(max_fd + 1, &read_fds, NULL, NULL, NULL);

        // 새 연결 수락
        if (FD_ISSET(server_fd, &read_fds)) {
            client_fd = accept(server_fd, (struct sockaddr*)&cli_addr, &addr_len);
            for (int i = 0; i < MAX_CLIENTS; i++) {
                if (client_socks[i] == 0) {
                    client_socks[i] = client_fd;
                    printf("Client connected: FD %d\n", client_fd);
                    break;
                }
            }
        }
```

```
        // 클라이언트 메시지 수신
        for (int i = 0; i < MAX_CLIENTS; i++) {
            int sd = client_socks[i];
            if (FD_ISSET(sd, &read_fds)) {
                int n = read(sd, buf, BUF_SIZE);
                if (n <= 0) {
                    close(sd);
                    client_socks[i] = 0;
                    printf("Client disconnected: FD %d\n", sd);
                } else {
                    buf[n] = '\0';
                    printf("Message from FD %d: %s", sd, buf);

                    // 다른 모든 클라이언트에게 브로드캐스트
                    for (int j = 0; j < MAX_CLIENTS; j++) {
                        if (client_socks[j] > 0 && client_socks[j] != sd)
                            send(client_socks[j], buf, n, 0);
                    }
                }
            }
        }
    }

    return 0;
}
```

✅ 5. 클라이언트 코드 (`chat_client.c`)

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

#define PORT 12345
#define BUF_SIZE 1024

int sockfd;

void* recv_handler(void* arg) {
    char buf[BUF_SIZE];
    int n;
    while ((n = read(sockfd, buf, BUF_SIZE)) > 0) {
        buf[n] = '\0';
        printf(">> %s", buf);
    }
    return NULL;
}

```

```c
int main() {
    struct sockaddr_in serv_addr;
    char buf[BUF_SIZE];
    pthread_t recv_thread;

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) { perror("socket"); exit(1); }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);
    serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    if (connect(sockfd, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) < 0) {
        perror("connect"); exit(1);
    }

    pthread_create(&recv_thread, NULL, recv_handler, NULL);

    while (1) {
        fgets(buf, BUF_SIZE, stdin);
        if (strncmp(buf, "exit", 4) == 0)
            break;
        send(sockfd, buf, strlen(buf), 0);
    }

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

✅ 6. 실행 방법

1. 서버 실행:

```
gcc -o chat_server chat_server.c
./chat_server
```

2. 클라이언트 실행:

```
gcc -o chat_client chat_client.c -pthread
./chat_client
```

💡 여러 터미널에서 클라이언트를 동시에 실행하여 서로 채팅 가능.

✅ 확장 아이디어

- 사용자 ID 별도 관리 (예: 닉네임)
- 메시지 형식 지정 (JSON 구조 등)
- 채팅방 기능 (room 분리)

- 멀티스레드 서버 전환 (`pthread` 방식)
- 보안 추가 (TLS/SSL)

13.2 멀티 클라이언트 Echo 서버

📌 목적

여러 클라이언트가 동시에 접속해 서버에 메시지를 보내면, 서버는 **각 클라이언트에게 해당 메시지를 그대로 돌려주는** Echo 기능을 수행한다.
다중 접속 처리와 **클라이언트 구분**을 위한 실습용 예제로 적합하며, `select()` 기반 I/O multiplexing 구조를 체득하는 데 효과적이다.

✅ 1. 핵심 기술 요약

- `socket()`, `bind()`, `listen()`, `accept()`
- `select()`로 소켓 상태 모니터링
- 접속된 클라이언트 각각에게 **자신이 보낸 메시지만** Echo 전송
- 서버는 다중 클라이언트의 접속/종료를 관리

✅ 2. 구현 구조

```
[Client 1] → Hello → [ Server ] → Hello → [Client 1]
[Client 2] → Test  → [ Server ] → Test  → [Client 2]
```

브로드캐스트 없이, **개별 클라이언트와 서버 간의 1:1 Echo 처리**

✅ 3. 코드: `echo_server.c`

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/select.h>

#define PORT 12345
#define MAX_CLIENTS 10
#define BUF_SIZE 1024

int main() {
    int server_fd, new_socket, max_sd, sd;
    int client_socket[MAX_CLIENTS] = {0};
    struct sockaddr_in address;
    socklen_t addrlen = sizeof(address);
    char buffer[BUF_SIZE];
```

```
19
20      fd_set readfds;
21
22      server_fd = socket(AF_INET, SOCK_STREAM, 0);
23      if (server_fd == 0) { perror("socket failed"); exit(EXIT_FAILURE); }
24
25      address.sin_family = AF_INET;
26      address.sin_addr.s_addr = INADDR_ANY;
27      address.sin_port = htons(PORT);
28
29      bind(server_fd, (struct sockaddr*)&address, sizeof(address));
30      listen(server_fd, 5);
31      printf("Echo server listening on port %d\n", PORT);
32
33      while (1) {
34          FD_ZERO(&readfds);
35          FD_SET(server_fd, &readfds);
36          max_sd = server_fd;
37
38          for (int i = 0; i < MAX_CLIENTS; i++) {
39              sd = client_socket[i];
40              if (sd > 0) FD_SET(sd, &readfds);
41              if (sd > max_sd) max_sd = sd;
42          }
43
44          select(max_sd + 1, &readfds, NULL, NULL, NULL);
45
46          // 새 클라이언트 연결
47          if (FD_ISSET(server_fd, &readfds)) {
48              new_socket = accept(server_fd, (struct sockaddr*)&address, &addrlen);
49              printf("New connection, socket fd is %d\n", new_socket);
50
51              for (int i = 0; i < MAX_CLIENTS; i++) {
52                  if (client_socket[i] == 0) {
53                      client_socket[i] = new_socket;
54                      break;
55                  }
56              }
57          }
58
59          // 기존 클라이언트 메시지 처리
60          for (int i = 0; i < MAX_CLIENTS; i++) {
61              sd = client_socket[i];
62              if (FD_ISSET(sd, &readfds)) {
63                  int valread = read(sd, buffer, BUF_SIZE);
64                  if (valread <= 0) {
65                      close(sd);
66                      client_socket[i] = 0;
67                      printf("Client disconnected: fd %d\n", sd);
68                  } else {
69                      buffer[valread] = '\0';
70                      send(sd, buffer, valread, 0); // Echo
71                  }
```

```
            }
        }
    }

    return 0;
}
```

✅ 4. 클라이언트: `echo_client.c`

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

#define PORT 12345
#define BUF_SIZE 1024

int main() {
    int sock;
    struct sockaddr_in serv_addr;
    char buffer[BUF_SIZE], recv_buf[BUF_SIZE];

    sock = socket(AF_INET, SOCK_STREAM, 0);
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);
    serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr));

    while (1) {
        printf("Message: ");
        fgets(buffer, BUF_SIZE, stdin);

        if (strncmp(buffer, "exit", 4) == 0)
            break;

        send(sock, buffer, strlen(buffer), 0);
        int n = read(sock, recv_buf, BUF_SIZE);
        recv_buf[n] = '\0';
        printf("Echo: %s", recv_buf);
    }

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

✅ 5. 실행 예

서버 실행

```
gcc -o echo_server echo_server.c
./echo_server
```

클라이언트 실행

```
gcc -o echo_client echo_client.c
./echo_client
```

💡 여러 터미널에서 클라이언트 실행 가능. 각자의 입력에 대해 자기만 응답받음.

✅ 확장 포인트

- `select()` → `epoll()`으로 확장 (성능 향상)
- `pthread` 또는 `fork`로 클라이언트 처리 병렬화
- Echo 외에도 명령어 기반 입력/응답 구조 도입
- 메시지 로그 저장 및 기록 기능 추가
- TCP Keep-Alive 설정 (`SO_KEEPALIVE`)

13.3 파일 업로드/다운로드 서버

파일 전송 기능은 네트워크 프로그래밍에서 매우 중요한 응용 분야다.
이 항목에서는 **TCP 기반으로 파일을 업로드하거나 다운로드하는 서버/클라이언트 구조**를 직접 구현하며, 실전에서 사용하는:

- 파일 입출력(`fopen`, `fread`, `fwrite`)
- 전송 프로토콜 설계 (간단한 명령어 기반)
- 버퍼 크기 설정, 전송 완료 처리

등을 모두 다룬다.

✅ 1. 전체 설계 개요

```
[Client] --- upload/download 명령어 --> [Server]

upload:  client → server 전송 → 서버 파일 저장
download: server → client 전송 → 클라이언트 저장
```

✅ 2. 명령어 프로토콜 설계 (간단한 구조)

클라이언트가 문자열을 먼저 전송:

- `UPLOAD filename\n` : 서버에게 업로드 요청
- `DOWNLOAD filename\n` : 서버에게 다운로드 요청

그 다음 서버가 OK 응답 후 전송이 시작된다.

✅ 3. 서버 코드 (`file_server.c`)

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

#define PORT 12345
#define BUF_SIZE 1024

void handle_client(int client_fd) {
    char command[BUF_SIZE], filename[BUF_SIZE];
    FILE* fp;
    int bytes;
    char buffer[BUF_SIZE];

    // 명령어 수신
    bytes = read(client_fd, command, sizeof(command) - 1);
    command[bytes] = '\0';
    sscanf(command, "%s %s", buffer, filename);

    if (strcmp(buffer, "UPLOAD") == 0) {
        fp = fopen(filename, "wb");
        if (!fp) {
            perror("fopen");
            return;
        }
        write(client_fd, "OK", 2);
        while ((bytes = read(client_fd, buffer, BUF_SIZE)) > 0)
            fwrite(buffer, 1, bytes, fp);
        fclose(fp);
        printf("Uploaded file: %s\n", filename);

    } else if (strcmp(buffer, "DOWNLOAD") == 0) {
        fp = fopen(filename, "rb");
        if (!fp) {
            write(client_fd, "NOFILE", 6);
            return;
        }
        write(client_fd, "OK", 2);
        while ((bytes = fread(buffer, 1, BUF_SIZE, fp)) > 0)
```

```
            write(client_fd, buffer, bytes);
        fclose(fp);
        printf("Sent file: %s\n", filename);
    }

    close(client_fd);
}

int main() {
    int server_fd, client_fd;
    struct sockaddr_in serv_addr, cli_addr;
    socklen_t cli_len = sizeof(cli_addr);

    server_fd = socket(AF_INET, SOCK_STREAM, 0);
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(PORT);

    bind(server_fd, (struct sockaddr*)&serv_addr, sizeof(serv_addr));
    listen(server_fd, 5);
    printf("File server listening on port %d\n", PORT);

    while (1) {
        client_fd = accept(server_fd, (struct sockaddr*)&cli_addr, &cli_len);
        if (fork() == 0) {
            close(server_fd);
            handle_client(client_fd);
            exit(0);
        }
        close(client_fd);
    }

    return 0;
}
```

✅ 4. 클라이언트 코드 (`file_client.c`)

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

#define PORT 12345
#define BUF_SIZE 1024

void upload_file(int sock, const char* filename) {
    char buffer[BUF_SIZE];
    FILE* fp = fopen(filename, "rb");
    if (!fp) { perror("fopen"); return; }

```

```
    sprintf(buffer, "UPLOAD %s\n", filename);
    send(sock, buffer, strlen(buffer), 0);

    read(sock, buffer, 2); // OK 응답 대기

    int bytes;
    while ((bytes = fread(buffer, 1, BUF_SIZE, fp)) > 0)
        send(sock, buffer, bytes, 0);
    fclose(fp);
}

void download_file(int sock, const char* filename) {
    char buffer[BUF_SIZE];
    sprintf(buffer, "DOWNLOAD %s\n", filename);
    send(sock, buffer, strlen(buffer), 0);

    read(sock, buffer, 2); // OK or NOFILE
    if (strncmp(buffer, "OK", 2) != 0) {
        printf("File not found on server.\n");
        return;
    }

    FILE* fp = fopen(filename, "wb");
    int bytes;
    while ((bytes = read(sock, buffer, BUF_SIZE)) > 0)
        fwrite(buffer, 1, bytes, fp);
    fclose(fp);
}

int main() {
    int sock;
    struct sockaddr_in serv_addr;
    char cmd[BUF_SIZE], filename[BUF_SIZE];

    sock = socket(AF_INET, SOCK_STREAM, 0);
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);
    serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr));

    printf("Command (upload/download): ");
    scanf("%s %s", cmd, filename);

    if (strcmp(cmd, "upload") == 0) {
        upload_file(sock, filename);
    } else if (strcmp(cmd, "download") == 0) {
        download_file(sock, filename);
    }

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

✅ 5. 실행 순서

서버 실행

```
gcc -o file_server file_server.c
./file_server
```

클라이언트 실행

```
gcc -o file_client file_client.c
./file_client
```

입력 예:

```
upload data.txt
```

또는

```
download image.jpg
```

✅ 향후 확장 아이디어

- 파일 크기 먼저 전송하여 정확한 수신 종료 조건 설정
- 전송 상태 표시 (진행률 %)
- 대용량 파일을 위한 `sendfile()` 활용
- 파일 충돌/중복 처리 로직 추가
- TLS 암호화, 인증 기능 추가

13.4 RESTful API 프록시 서버

🎯 목적

RESTful API 프록시 서버는 클라이언트의 HTTP 요청을 받아, **백엔드 서버(API)**로 요청을 중계하고 응답을 다시 클라이언트에게 전달한다.
이는 다음과 같은 상황에서 유용하다:

- 보안, 인증을 프록시에서 처리
- 로깅 및 필터링 적용
- 트래픽 로드밸런싱
- API 엔드포인트 추상화

✅ 1. 구조 개요

```
[Client] --> [C 프록시 서버] --> [REST API 서버 (ex. httpbin.org)]
   ⇅               ⇅
REQ/RES          REQ/RES
```

- 프록시 서버는 HTTP 요청을 파싱하고, 백엔드에 동일 요청을 보내고, 응답을 받아 다시 클라이언트에게 전달함

✅ 2. 요구사항 정리

- HTTP `GET`, `POST` 요청을 받아 다른 서버로 중계
- 클라이언트 요청 헤더와 바디를 그대로 전달
- 서버 응답 헤더와 바디를 다시 클라이언트에 반환
- `select()` 또는 단일 스레드 기반으로 구현

✅ 3. 라이브러리 선택

- **libcurl**: REST API 호출에 사용 (`curl_easy_perform`)
- **기본 C 소켓**: 클라이언트 요청 수신

설치:

```
sudo apt install libcurl4-openssl-dev
```

✅ 4. 코드 구현 (`proxy_server.c`)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <curl/curl.h>

#define PORT 8080
#define BUF_SIZE 4096

struct MemoryStruct {
    char* memory;
    size_t size;
};

size_t write_callback(void* contents, size_t size, size_t nmemb, void* userp) {
    size_t realsize = size * nmemb;
    struct MemoryStruct* mem = (struct MemoryStruct*) userp;

```

```
    mem->memory = realloc(mem->memory, mem->size + realsize + 1);
    memcpy(&(mem->memory[mem->size]), contents, realsize);
    mem->size += realsize;
    mem->memory[mem->size] = 0;

    return realsize;
}

void handle_http_proxy(int client_sock) {
    char buffer[BUF_SIZE];
    int len = read(client_sock, buffer, BUF_SIZE - 1);
    buffer[len] = '\0';

    printf("Client request:\n%s\n", buffer);

    // libcurl로 백엔드 REST API 요청 전송
    CURL* curl = curl_easy_init();
    if (curl) {
        struct MemoryStruct chunk = { malloc(1), 0 };

        curl_easy_setopt(curl, CURLOPT_URL, "https://httpbin.org/get");
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, write_callback);
        curl_easy_setopt(curl, CURLOPT_WRITEDATA, (void*)&chunk);

        CURLcode res = curl_easy_perform(curl);
        if (res != CURLE_OK) {
            char* err = (char*) curl_easy_strerror(res);
            write(client_sock, err, strlen(err));
        } else {
            char response[BUF_SIZE];
            snprintf(response, sizeof(response),
                "HTTP/1.1 200 OK\r\nContent-Length: %ld\r\nContent-Type:
application/json\r\n\r\n%s",
                chunk.size, chunk.memory);
            write(client_sock, response, strlen(response));
        }

        free(chunk.memory);
        curl_easy_cleanup(curl);
    }

    close(client_sock);
}
```

✅ 5. 메인 서버 루프

```
int main() {
    int server_fd, client_fd;
    struct sockaddr_in addr;
    socklen_t addrlen = sizeof(addr);

```

```
    curl_global_init(CURL_GLOBAL_ALL);
    server_fd = socket(AF_INET, SOCK_STREAM, 0);

    addr.sin_family = AF_INET;
    addr.sin_port = htons(PORT);
    addr.sin_addr.s_addr = INADDR_ANY;

    bind(server_fd, (struct sockaddr*)&addr, sizeof(addr));
    listen(server_fd, 5);

    printf("Proxy server running on port %d\n", PORT);

    while (1) {
        client_fd = accept(server_fd, (struct sockaddr*)&addr, &addrlen);
        if (fork() == 0) {
            close(server_fd);
            handle_http_proxy(client_fd);
            exit(0);
        }
        close(client_fd);
    }

    curl_global_cleanup();
    return 0;
}
```

✅ 6. 실행 방법

빌드

```
gcc -o proxy_server proxy_server.c -lcurl
```

실행

```
./proxy_server
```

테스트 (다른 터미널에서)

```
curl http://localhost:8080
```

응답:

```
{
  "args": {},
  "headers": {
    ...
  },
  "url": "https://httpbin.org/get"
}
```

✅ 확장 아이디어

- HTTP `POST`, `PUT`, `DELETE` 지원
- 프록시 요청 로그 남기기
- 인증 헤더 추가 (Bearer, API key)
- TLS 인증서 검증 옵션 조정
- 멀티 클라이언트 지원 (`select`, `pthread`)

13.5 부하 테스트용 네트워크 트래픽 생성기

🎯 목적

부하 테스트(load testing)는 네트워크 서버나 시스템이 얼마나 많은 요청을 처리할 수 있는지를 평가하는 데 필수적인 절차다.
이 항목에서는 **C 언어로 다수의 클라이언트 요청을 빠르게 생성해 네트워크 트래픽을 의도적으로 발생시키는 테스트 도구**를 작성한다.

✅ 1. 주요 기능 요약

- TCP 연결을 수백 개 이상 생성 (멀티스레딩 또는 루프 기반)
- 일정 주기로 요청 전송
- 서버의 응답 여부와 처리 시간 측정
- 초당 요청 수(RPS) 및 응답 속도 측정 가능

✅ 2. 트래픽 생성기 구조

```
[Load Generator]
 ├─ Thread 1 → connect + send → recv → close
 ├─ Thread 2 → connect + send → recv → close
 └─ ... × N

↓ 수천 개 요청 생성 후 응답 시간, 실패율 측정 가능
```

✅ 3. 간단한 트래픽 생성기 예제 (`loadgen.c`)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <arpa/inet.h>
#include <sys/time.h>

```

```
#define THREAD_COUNT 100
#define REQ_PER_THREAD 50
#define SERVER_IP "127.0.0.1"
#define SERVER_PORT 12345
#define MESSAGE "ping\n"

void* thread_func(void* arg) {
    for (int i = 0; i < REQ_PER_THREAD; i++) {
        int sock = socket(AF_INET, SOCK_STREAM, 0);
        if (sock < 0) continue;

        struct sockaddr_in serv_addr = {
            .sin_family = AF_INET,
            .sin_port = htons(SERVER_PORT)
        };
        inet_pton(AF_INET, SERVER_IP, &serv_addr.sin_addr);

        if (connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) < 0) {
            close(sock);
            continue;
        }

        send(sock, MESSAGE, strlen(MESSAGE), 0);

        char buf[1024];
        int n = read(sock, buf, sizeof(buf) - 1);
        if (n > 0) {
            buf[n] = '\0';
            printf("[Thread %ld] Response: %s", (long)arg, buf);
        }

        close(sock);
        usleep(50000); // 50ms delay between requests
    }
    return NULL;
}

int main() {
    pthread_t threads[THREAD_COUNT];
    struct timeval start, end;
    gettimeofday(&start, NULL);

    for (long i = 0; i < THREAD_COUNT; i++)
        pthread_create(&threads[i], NULL, thread_func, (void*)i);

    for (int i = 0; i < THREAD_COUNT; i++)
        pthread_join(threads[i], NULL);

    gettimeofday(&end, NULL);
    double elapsed = (end.tv_sec - start.tv_sec) + (end.tv_usec - start.tv_usec) / 1e6;
    printf("\nTotal time: %.3f sec\n", elapsed);
    printf("Total requests: %d\n", THREAD_COUNT * REQ_PER_THREAD);
```

```
    printf("Requests per second: %.2f RPS\n", (THREAD_COUNT * REQ_PER_THREAD) / elapsed);
    return 0;
}
```

✅ 4. 실행 방법

서버 준비

먼저, `echo_server` 혹은 테스트 대상 TCP 서버를 12345 포트에서 실행 중이어야 함

빌드 및 실행

```
gcc -o loadgen loadgen.c -pthread
./loadgen
```

✅ 5. 출력 예시

```
[Thread 0] Response: pong
[Thread 1] Response: pong
...
Total time: 3.258 sec
Total requests: 5000
Requests per second: 1534.77 RPS
```

✅ 6. 확장 아이디어

- 요청 메시지를 파일 또는 인자로부터 동적으로 설정
- 실패율, 평균 RTT 측정
- 응답 로그 파일 저장
- `epoll` 기반으로 트래픽 증가
- UDP 트래픽도 지원 가능하게 수정
- JSON 기반 요청 + HTTP API 부하 테스트로 전환 가능