CSE 3027: Computer Networks

Project 1

컴퓨터학부 20학번

2020076108 이희수

1. 서버 기본 구성요소

```
int main(int argc, char* argv[]) {
    int server_port = atoi(argv[1]);
```

서버를 실행할 때 클라이언트와 통신할 포트를 인자로 주기 때문에 인자를 형변환하여 포트를 나타내는 server_port 변수에 저장합니다.

```
int server_socket = socket(AF_INET, SOCK_STREAM, 0);
if (server_socket == -1) {
    perror("Socket creation failed");
    return 1;
}
```

서버의 소켓을 생성합니다. socket() 함수의 매개변수인 AF_INET은 IPv4를 의미하고 SOCK_STREAM은 TCP를 의미합니다.

```
struct sockaddr_in server_addr, client_addr;
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(server_port);
server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
if (bind(server_socket, (struct sockaddr*)&server_addr, sizeof(server_addr)) == -1) {
    perror("Binding failed");
    return 1;
}
```

소켓의 정보를 담는 구조체 server_addr, client_addr를 선언합니다.

sin_family 멤버는 주소 체계를 나타냅니다. AF_INET, 즉 IPv4로 설정합니다.

sin_port 멤버는 소켓이 바인딩될 포트입니다. 처음에 저장해두었던 server_port로 설정합니다.

sin_addr 구조체는 소켓이 바인딩될 IP 주소입니다. INADDR_ANY, 즉 가능한 모든 IP로 설정합니다.

bind() 함수로 소켓과 소켓의 정보를 가진 server_addr 구조체, 구조체의 크기를 매개변수로 주어 소켓 IP주소와 포트를 바인딩합니다.

```
if (listen(server_socket, BACKLOG) == -1) {
    perror("Listening failed");
    return 1;
}
```

listen() 함수를 이용하여 소켓의 백로그를 생성합니다.

```
while (1) {
    socklen_t client_addr_len = sizeof(client_addr);
    int client_socket = accept(server_socket, (struct sockaddr*)&client_addr, &client_addr_len);
    if (client_socket == -1) {
        perror("Accepting client failed");
        continue;
    }

    printf("Accepted connection from %s:%d\n", inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));
```

accept() 함수를 이용하여 백로그에 요청을 받습니다. 매개변수는 서버의 소켓, 요청이 온 client의 정보를 담을 구조체 client_addr의 주소, client_addr 구조체의 크기의 주소입니다.

accept() 함수의 리턴값은 client와 통신하기 위한 소켓입니다. 서버는 client_socket으로 client에게 메시지를 보낼 수 있습니다.

```
char request[1024];
recv(client_socket, request, sizeof(request), 0);
printf("Received request:\n%s\n", request);

requestHandle(client_socket, request);
```

client의 요청 메시지를 담을 request 변수를 선언하고, recv 함수를 이용하여 client에게서 온 요ㄴ청 메시지를 request에 담고 출력합니다.

따로 작성한 requestHandle 함수를 호출하여 요청 메시지를 처리합니다.

requestHandle 함수는 client_socket과 요청 메시지를 인자로 받아 요청에 따라 클라이언트에게 응답을 보내는 함수입니다.

2. requestHandle 함수

```
void requestHandle(int client_socket, char* request) {
    char method[10];    //요청 메소드
    char path[20];      //파일 경로
    char* filename;     //파일명
    char* extention = NULL; //파일 확장자
    FILE* file;     //파일에 대한 포인터
    long fsize;     //파일의 크기
    int option;     //현재 어떤 파일을 열었는지 표시
    char buffer[BUFFER_SIZE];   //파일 전송을 위한 버퍼
    char response[BUFFER_SIZE]; //응답
    size_t bytesRead;   //파일 전송 추적을 위한 변수

    sscanf(request, "%s %s", method, path); //start-line에서 필요한 정보를 읽어온다.

    /*
     * 요청 path의 맨 앞 '/' 제거
     */
    filename = strdup(path);
    filename += strspn(filename, "/");

    /*
     * 요청받은 파일을 연다.
     * 파일이 있는 경우 크기를 계산하고 파일 확장자를 저장한다.
     */
    file = fopen(filename, "rb");
    if (file == NULL) {
        perror("Opening file failed");
    }
    else {
        fseek(file, 0, SEEK_END);
        fsize = ftell(file);
        fseek(file, 0, SEEK_SET);

        extention = strchr(filename, '.') + 1;
    }
```

requestHandle 함수의 윗부분입니다.

method, path: 클라이언트의 요청 메시지에서 필요한 부분을 담습니다.

filename, extension: 각각 파일명, 파일의 확장자를 담습니다.

file: 요청에 따라 파일을 보내기 위해 열게될 파일입니다.

fsize: 파일의 크기입니다.

option: 확장자에 따라 응답 메시지를 다르게 작성하기 위한 변수입니다.

buffer: 파일을 보낼 때 사용하는 버퍼입니다.

response: 클라이언트에게 보낼 응답 메시지입니다.

bytesRead: 파일 전송을 추적하기 위한 변수입니다.

sscanf 함수를 이용하여 요청 메시지의 첫 두 부분을 읽습니다.

```
GET / HTTP/1.1
Host: 10.0.2.15:8080
Connection: keep-alive
```

이런 요청 메시지가 있다면 method에는 "GET", path에는 '/'가 들어갑니다.

```
filename = strdup(path);
filename += strspn(filename, "/");
```

path에 들어간 문자열은 '/'으로 시작하기 때문에 현재 디렉토리에서 파일을 찾기 위해 filename이 '/' 다음을 가리키게 합니다.

```
file = fopen(filename, "rb");
if (file == NULL) {
    perror("Opening file failed");
}
else {
    fseek(file, 0, SEEK_END);
    fsize = ftell(file);
    fseek(file, 0, SEEK_SET);

    extention = strchr(filename, '.') + 1;
}
```

요청받은 파일을 엽니다. 파일이 존재한다면 파일의 크기를 계산하고 extention이 파일명의 확장자 부분을 가리키게 합니다.

```
if (extention == NULL) {
    option = 0;
}
else if (!strcmp(extention, "html")) {
    option = 1;
}
else if (!strcmp(extention, "gif")) {
    option = 2;
}
else if (!strcmp(extention, "jpeg") || !strcmp(extention, "jpg")) {
    option = 3;
}
else if (!strcmp(extention, "mp3")) {
    option = 4;
}
else if (!strcmp(extention, "pdf")) {
    option = 5;
}
else {
    option = 0;
}
```

파일의 확장자에 따라 option을 설정합니다. 파일이 존재하지 않거나(extention이 NULL) html, gif, jpg, mp3, pdf 이외의 확장자면 0으로 설정합니다.

```
switch(option) {
    case 0:
        sprintf(response, "HTTP/1.1 404 Not Found\r\nContent-Type: text/html\r\n");
        break;
    case 1:
        sprintf(response, "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\nContent-Length: %ld\r\n\r\n", fsize);
        break;
    case 2:
        sprintf(response, "HTTP/1.1 200 OK\r\nContent-Type: image/gif\r\nContent-Length: %ld\r\n\r\n", fsize);
        break;
    case 3:
        sprintf(response, "HTTP/1.1 200 OK\r\nContent-Type: image/jpeg\r\nContent-Length: %ld\r\n\r\n", fsize);
        break;
    case 4:
        sprintf(response, "HTTP/1.1 200 OK\r\nContent-Type: audio/mpeg\r\nContent-Length: %ld\r\n\r\n", fsize);
        break;
    case 5:
        sprintf(response, "HTTP/1.1 200 OK\r\nContent-Type: application/pdf\r\nContent-Length: %ld\r\n\r\n", fsize);
        break;
}
```

앞에서 설정한 option에 따라 Content-Type을 다르게 응답 헤더를 작성합니다. option이 0인 경우는 404 Not Found입니다.

```
send(client_socket, response, strlen(response), 0);
if (option > 0) {
    while ((bytesRead = fread(buffer, 1, BUFFER_SIZE, file)) > 0) {
        send(client_socket, buffer, bytesRead, 0);
    }
    fclose(file);
}
```

클라이언트에게 response를 보내고 option이 0보다 크면, 즉 보내야할 파일이 있다면 버퍼를 통해 파일을 읽고 클라이언트에게 보냅니다.

3. 결과에 대한 간략한 설명

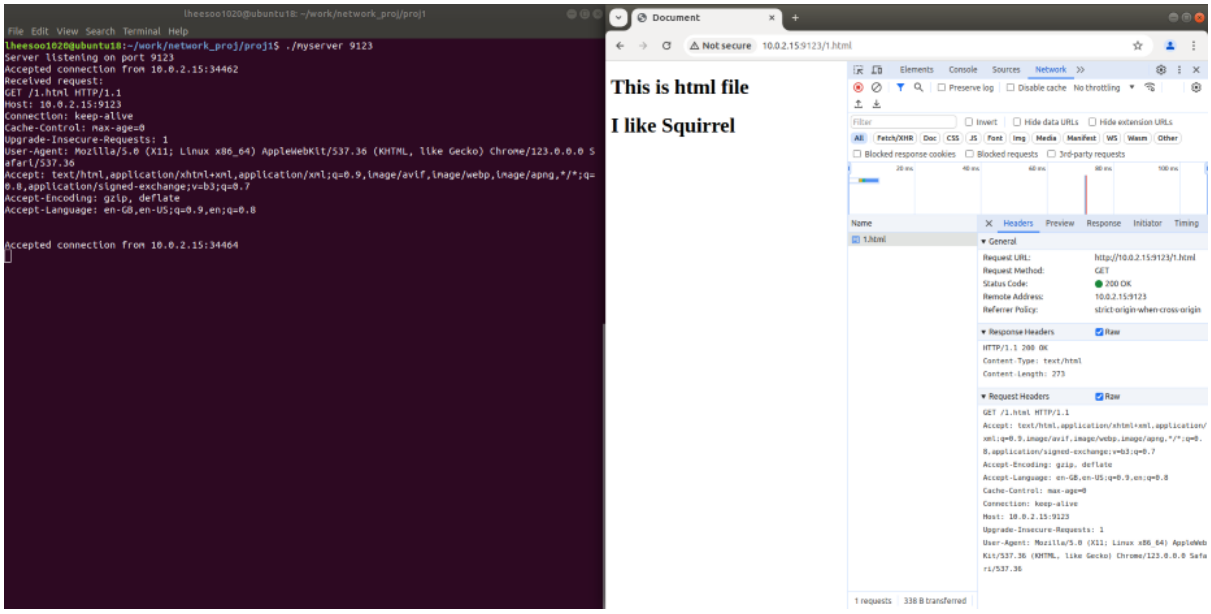
```
lheesoo1020@ubuntu18:~/work/network_proj/proj1$ ./myserver 9123
Server listening on port 9123
```

./myserver 9123을 실행하면 9123번 포트로 서버를 열게 됩니다. 서버는 소켓을 만들고, 소켓에 IP와 포트를 바인딩하고, 백로그를 만들고 메시지를 기다리는 중입니다.

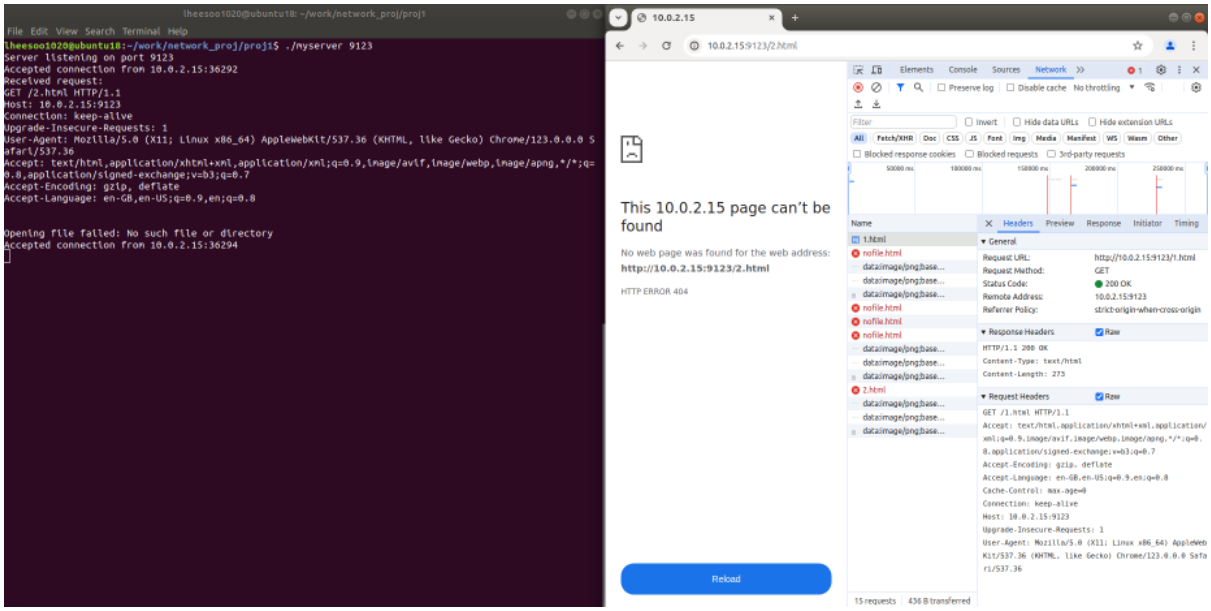
```
lheesoo1020@ubuntu18:~/work/network_proj/proj1$ ./myserver 9123
Server listening on port 9123
Accepted connection from 10.0.2.15:53848
Received request:
GET / HTTP/1.1
Host: 10.0.2.15:9123
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 S
afari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=
0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
```

웹 브라우저에서 서버에 접속하면 서버는 웹 브라우저의 요청을 받아 연결을 성립시킨 뒤, 메시지를 받습니다. 출력된 요청 메시지를 보면 첫 줄인 GET / HTTP/1.1는 request line입니다. 다음 줄부터 끝까지는 header lines입니다.

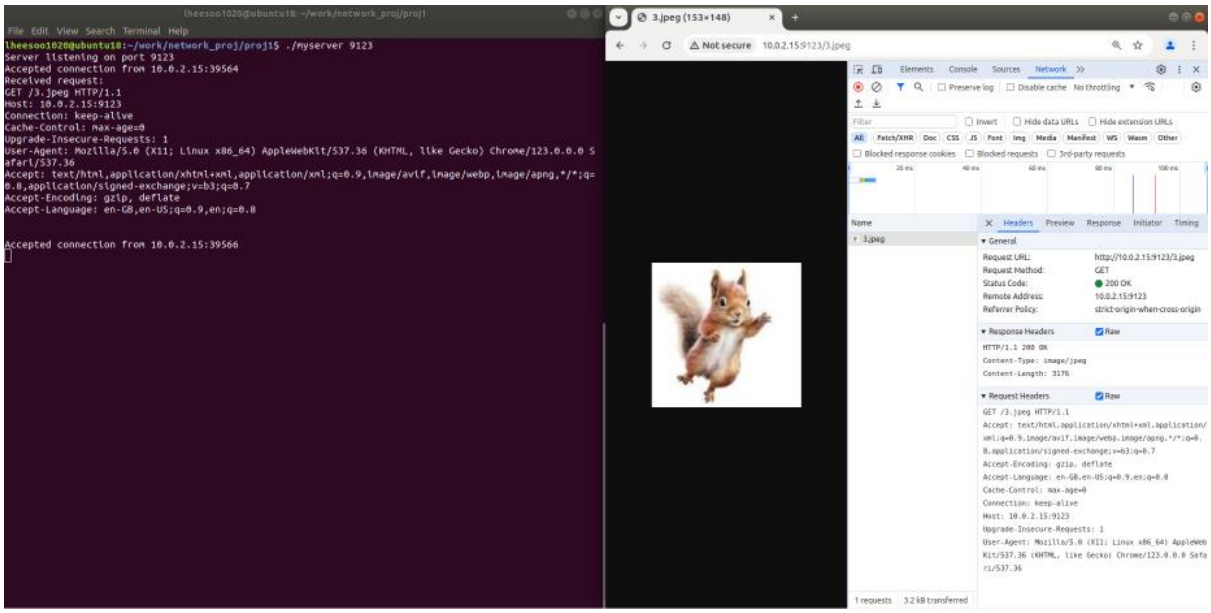
제가 작성한 서버는 요청 메시지를 requestHandle 함수로 처리하게 됩니다. "/"가 요청으로 들어왔기 때문에 이에 맞게 내부에서 문자열을 처리하여 응답 헤더와 함께 파일을 보내게 됩니다.



크롬 브라우저에서 서버에 접속한 모습입니다. 웹 브라우저는 GET /1.html 요청을 보냈고 서버로부터 응답 메시지를 받았습니다. 응답을 보면 status line에는 요청이 성공적이었음을 나타내는 HTTP/1.1 200 OK, header lines에는 content-type과 content-length가 있습니다. 그리고 브라우저에는 서버가 응답으로 보낸 html 파일이 표시된 것을 볼 수가 있습니다.



서버는 기존에 설정해두었던 요청이 아니면 404 Not Found 메시지를 보내도록 작성했기 때문에 존재하지 않는 파일인 2.html을 요청하면 사진과 같이 페이지를 찾을 수 없다고 표시되게 됩니다.



Jpeg와 같은 다른 형식의 파일 요청도 잘 처리되는 것을 볼 수 있습니다.

4. 과제 수행 중 발생한 문제점

서버 자체를 구현하는 것은 수업자료에 코드가 나와있기 때문에 참고하여 어렵지 않게 했습니다.

문제는 서버를 만들고 이후에 응답으로 헤더와 함께 파일을 보내도록 작성하였는데, 웹 브라우저에서 서버에 접속했을 때 서버가 보낸 파일을 표시하지 못하는 것이었습니다. 코드 곳곳에 프린트문을 넣어가면서 어디서 잘못됐는지 찾아보았는데 코드는 파일을 보내는 코드까지 쭉 문제없이 실행되었습니다. 웹 브라우저에서도 개발자 도구를 통해 응답 헤더를 보면 응답 헤더가 잘 도착하고 content-type도 잘 표시된 것을 볼 수 있었습니다. 여러 번 시도 끝에 찾은 결론은 다음과 같습니다.

서버에서 응답 헤더를 나타내는 변수는 char response[BUFFER_SIZE]로 선언되어 있습니다. 서버는 응답을 보낼 때 send()를 사용하는데 처음에는 매개변수로 send(client_socket, response, sizeof(response))를 주었습니다. 문제가 되었던 점은 sizeof(response)가 실제 response의 길이가 아닌 BUFFER_SIZE만큼 잡히기 때문에 응답 헤더의 크기가 클라이언트에게 잘못 전달되어 뒤에 오는 파일을 표시하지 못한 것이었습니다. 이 부분을 sizeof(response)를 strlen(response)로 바꾸니 문제가 해결되어 웹 브라우저에서 파일을 표시할 수 있었습니다.