report

서버 기능 분류 및 설명

이 서버에서 제공하는 기능은 `set_contentType`, `make_header`, `send_error`, `send_HTTP_response`로 분류할 수 있습니다. 그리고 각 기능은 최종적으로 `main 함수`에서 호출되어 실행됩니다.

- `set_contentType` 함수

```
// 참고: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types
// ? set_contentType 호출 후에 content-type 값이 content_type 변수에 저장된다.
void set_contentType(char* uri) {
    /* 확장자 추출 */
    char* extension = strrchr(uri, '.'); // strrchr: https://www.ibm.com/docs/ko/i/

    /* .html file */
    if (strcmp(extension, ".html") == 0) {
        strcpy(content_type, "text/html");
    }
    /* .png file */
    else if (strcmp(extension, ".png") == 0) {
        strcpy(content_type, "image/png");
    }
    /* .gif file */
    else if (strcmp(extension, ".gif") == 0) {
        strcpy(content_type, "image/gif");
    }
    /* .jpeg .jpg file */
    else if (strcmp(extension, ".jpeg") == 0 || strcmp(extension, ".jpg") == 0) {
        strcpy(content_type, "image/jpeg");
    }
    /* .pdf file */
    else if (strcmp(extension, ".pdf") == 0) {
        strcpy(content_type, "application/pdf");
    }
    /* .mp3 file */
    else if (strcmp(extension, ".mp3") == 0) {
        strcpy(content_type, "audio/mpeg");
    }
    /* other */
    else {
        strcpy(content_type, "text/plain");
    }
}
```

  이 함수는 클라이언트에게 입력받은 uri를 파싱하여 response header에 입력할 **content-type**을 결정하는 함수입니다. 함수 내부 코드를 살펴보면 `html`, `png`, `gif`, `jpeg/jpg`, `pdf`, `mp3` 파일을 지원하는 것을 확인할 수 있습니다.

  해당 함수에 `uri` 파라미터를 전달하여 호출하게 되면, *전역변수 content_type*에 content-type 값이 저장됩니다.

  여기서 **default type**은 *text/plain*으로 설정했습니다.

- `make_header` 함수

```
// 참고: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status
// ? make_header 호출 후에 헤더 내용이 header 변수에 저장된다.
void make_header(int status_code, long content_lenght) {
    /* status code에 알맞는 status 문구를 결정한다. */
    char status_text[50];

    switch (status_code)
    {
        case 200:
            strcpy(status_text, "OK");
            break;

        case 400:
            strcpy(status_text, "Bad Request");

        case 404:
            strcpy(status_text, "Not Found");
            break;

        case 500:
        default:
            strcpy(status_text, "Internal Server Error");
            break;
    }

    /* status code/text, content-type, content-length을 조합하여 헤더를 완성한다. */
    sprintf(header, HEADER_FORMAT, status_code, status_text, content_lenght, content_type);
}
```

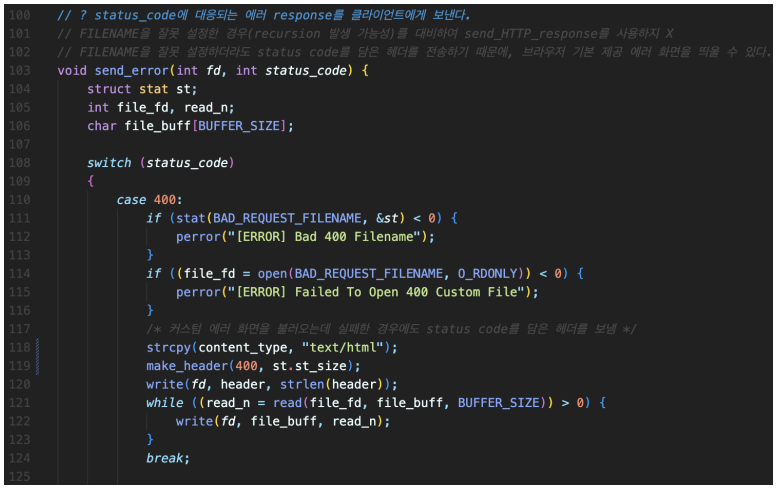
→ `make_header` 함수 내부

이 함수는 입력받은 *status_code*, *content_length*, *content_type*을 통해 **HTTP Response의 헤더** 내용을 채워주는 역할을 수행합니다.

여기서 주의해야 할 점은 `content_type` 값을 결정한 후에 make_header 함수를 호출해야한다는 점입니다. (`send_HTTP_response` *함수에서 실제 함수를 호출하기 전에 set_contentType 함수를 호출하는 것을 확인할 수 있습니다*) 또한 content_type이 이미 전역변수로 선언되어있기 때문에 content_type에 대한 파라미터는 전달하지 않습니다.

정리하면, 이 함수는 입력된 status_code에 따라 status_text를 결정하고, 앞서 결정된 헤더 정보들을 통해 **response 헤더**를 완성하여 이를 *전역변수 header*에 저장하는 기능을 수행합니다.

- `send_error` 함수



```
// ? status_code에 대응되는 에러 response를 클라이언트에게 보낸다.
// FILENAME을 잘못 설정한 경우(recursion 발생 가능성)를 대비하여 send_HTTP_response를 사용하지 X
// FILENAME을 잘못 설정하더라도 status code를 담은 헤더를 전송하기 때문에, 브라우저 기본 제공 에러 화면을 띄울 수 있다.
void send_error(int fd, int status_code) {
    struct stat st;
    int file_fd, read_n;
    char file_buff[BUFFER_SIZE];

    switch (status_code)
    {
        case 400:
            if (stat(BAD_REQUEST_FILENAME, &st) < 0) {
                perror("[ERROR] Bad 400 Filename");
            }
            if ((file_fd = open(BAD_REQUEST_FILENAME, O_RDONLY)) < 0) {
                perror("[ERROR] Failed To Open 400 Custom File");
            }
            /* 커스텀 에러 화면을 불러오는데 실패한 경우에도 status code를 담은 헤더를 보냄 */
            strcpy(content_type, "text/html");
            make_header(400, st.st_size);
            write(fd, header, strlen(header));
            while ((read_n = read(file_fd, file_buff, BUFFER_SIZE)) > 0) {
                write(fd, file_buff, read_n);
            }
            break;
```

→ `send_error` 함수 내부의 일부

전달받은 error status_code를 통해 *소켓으로 error response를 보내는 함수*입니다. 원래는 이 함수 내부에서 `send_HTTP_response` 함수를 호출하는 방식으로 작성하였으나, 혹시라도 파일명을 잘못 명시한 경우에 send_error 함수와 send_HTTP_response 함수가 *서로를 무한히 호출하게 되는 문제*가 발생할 수 있다는 것을 깨달았습니다. 따라서 이를 대비하기 위해 코드가 중복되더라도 send_HTTP_response 호출 없이 코드를 분리하여 작성하였습니다.

그리고 커스텀 html 파일이 제대로 열리지 않더라도, 에러 코드에 대한 헤더를 클라이언트에게 보내줌으로써 브라우저가 에러 코드에 알맞는 화면을 띄울 수 있도록 하였습니다.

- `send_HTTP_response` 함수

  send_HTTP_response 함수는 전달받은 *파일 내용을 소켓을 통해 클라이언트에게 전달*하는 함수입니다. 이 함수는 위의 `send_error` 함수와 거의 유사하게 작성되었습니다. (이미지 높이가 큰 관계로, 해당 함수에는 이미지 첨부를 생략했습니다)

- `main` 함수

  main 함수에서는 우선 클라이언트와 연결할 ***소켓***을 생성하고, 해당 소켓에 포트번호를 `bind`합니다. 이후 클라이언트의 연결 요청을 기다린 후 클라이언트의 connect 요청에 따라 연결이 되면, *무한 반복문을 돌면서 소켓 통신을 진행*합니다.

  while문 안에서는 클라이언트의 request 메시지를 전달받고, 전달 받은 HTTP request에서 **method**와 **요청 파일명**을 파싱합니다. 최종적으로 이 정보를 `send_HTTP_response` 함수에 전달하여, 요청 파일에 대한 response를 클라이언트에게 보내는 작업을 진행합니다. (main 함수 또한 코드가 긴 관계로 이미지 첨부를 생략했습니다)

문제 해결

- **소켓 프로그래밍 개념 이해 부족**

  처음에 main 함수에서 클라이언트와 소켓을 연결할 때 사용되었던 `socket_fd`가 실제 클라이언트와 통신하는 과정에서 사용되는 소켓(new_socket_fd)과 별개로 존재한다는 점을 잘 이해하지 못한 상태에서 코드를 작성하다보니, 클라이언트의 데이터를 수신하는 과정에서부터 어려움을 겪었습니다.

  따라서 한글로 정리된 관련 자료를 읽으면 이를 이해하는 데 도움이 될 것 같아, 관련 자료를 찾으면서 해당 내용을 이해하여 코드를 작성할 수 있었습니다. (도움을 받은 포스팅: https://recipes4dev.tistory.com/153)

- **기능(함수) 분류**

  맨 처음 최종적으로 완성했던 코드는 **중복 사용되는 코드**를 따로 함수로 분류하지 않아, 코드의 가독성이 현재 코드보다 현저히 떨어졌습니다. 특히 시간이 지난 후에 코드를 이어서 작성하기 위해, 앞서 작성했던 코드 내용들을 다시 이해하는 데 많은 어려움을 느꼈습니다. 특히 중복되는 코드로 인해 코드를 이해하는 과정에서부터 피로도가 크게 느껴졌습니다.

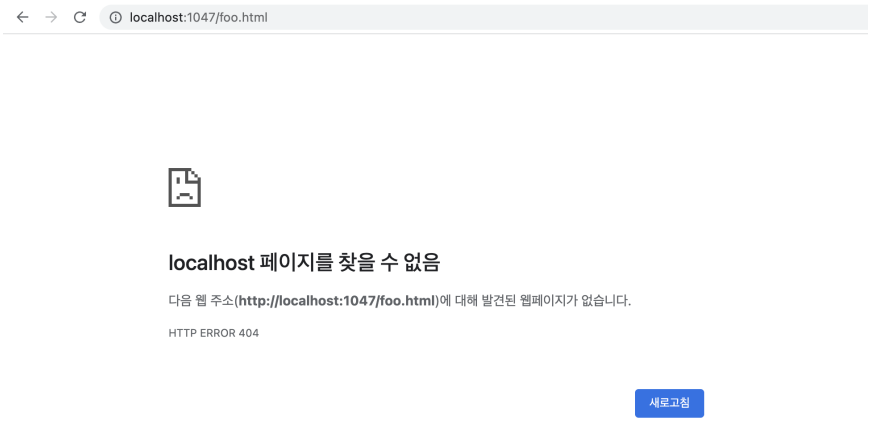
  이러한 문제점을 인지한 후, 주석을 더 꼼꼼하게 추가하고 중복되는 부분은 함수로 따로 분류하여 코드를 수정하였습니다. 그 결과 코드의 가독성이 이전보다 높아졌음을 느낄 수 있었습니다.

- **에러 처리 함수 작성 -** `send_error`

  원래는 `send_error` 함수에서 `send_HTTP_response` 함수를 호출하는 방식으로 작성하였으나, 검토 과정에서 한 가지 문제점을 발견할 수 있었습니다. 만일 서버 코드 내에서 에러 페이지 파일명(ex. `NOTFOUND_FILENAME`)을 잘못 기입하는 경우, *send_error에서* ***send_HTTP_response 호출*** *→ send_HTTP_response에서* ***send_error 호출*** *→ send_error에서* ***send_HTTP_response 호출*** *→ …* 와 같은 과정이 *무한히 반복되어 서버가 갑자기 종료되어 버린다는 것*입니다.

  이러한 예외 상황에서 서버가 아무 예고 없이 종료되는 것보다는, 클라이언트에게 커스텀 html 파일은 전송하진 못하더라도 브라우저가 서버의 응답 코드에 대해서는 알고 있는 것이 더 좋을 것이라 생각했습니다.

  따라서 비슷한 코드가 중복되더라도 `send_error` 함수에서 `send_HTTP_response` 함수 호출 없이 직접 파일을 읽는 과정을 추가하였습니다. 단, *파일 처리 과정에서 에러가 발생*할 시 코드를 종료하지 않고 에러 로그만 띄우고 *이후 과정은 그대로 진행*하는 것을 채택하였습니다*(error status code를 클라이언트에게 전송하기 위해서 입니다)*. 이렇게 된다면 위와 같은 예외 상황에서 커스텀 html 화면은 띄우진 못하지만, 아래와 같이 브라우저가 기본적으로 제공하는 에러 화면을 띄울 수 있게 됩니다.

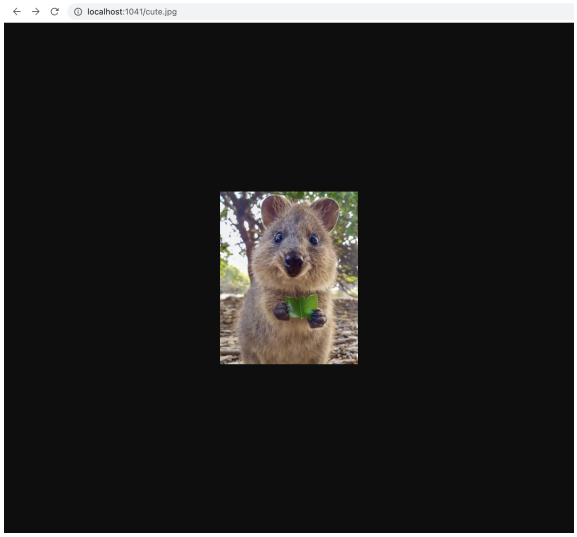


→ 서버 코드에서 `NOTFOUND_FILENAME` 값을 잘못 기입한 경우

출력 예제

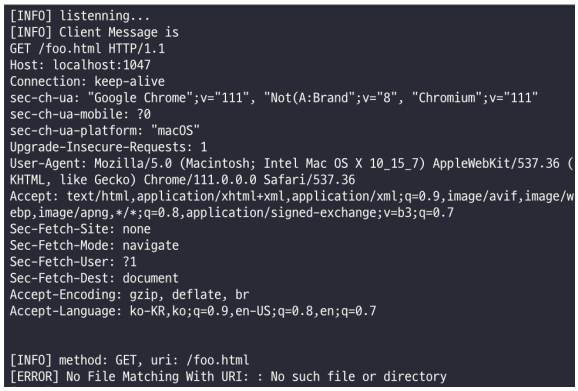
```
[INFO] listenning...
[INFO] Client Message is
GET /cute.jpg HTTP/1.1
Host: localhost:1041
Connection: keep-alive
sec-ch-ua: "Google Chrome";v="111", "Not(A:Brand";v="8", "Chromium";v="111"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (
KHTML, like Gecko) Chrome/111.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:1041/
Accept-Encoding: gzip, deflate, br
Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7
```

→ client 요청 출력1 (part A)



→ client의 출력 화면1 (jpg 파일 요청)

위 두 사진은 각각 서버 | 클라이언트의 출력 결과 예제입니다. `./myserver 1041` 명령어로 서버를 실행하고, 브라우저에서 1041 포트로 접속하여 jpg 확장자 파일을 요청한 결과입니다.



→ client 요청 출력2

→ client의 출력 화면2 (잘못된 파일 요청)

위 두 사진은 클라이언트 측에서 서버에 존재하지 않는 파일을 요청했을 시에 뜨는 결과화면을 캡처한 것입니다.