Programming Assignment #4

Due: November 17, 11:59:59 PM

1. Introduction

- 이번 과제를 통해 HTTP/1.0을 따르는 멀티 프로세스 웹 서버를 만들어본다.

2. Overview

- PA #3에서 작성했던 서버-클라이언트 프로그램의 서버를 확장해, Google Chrome과 같은 상용 웹 브라우저로 접속시켜 내용을 확인할 수 있는 웹 서버를 만든다. 해당 웹 서버는 HTTP 프로토콜을 준수하여 클라이언트로부터 들어오는 요청에 대한 적절한 응답을 하고, 멀티 프로세스를 이용해 동시에 여러 요청을 처리한다.

3. Background

- HTTP (hypertext transfer protocol)
  - ✓ HTTP는 인터넷에서 가장 널리 쓰이는 프로토콜의 하나로, WWW(world wide web)에서 HTML(hypertext markup language) 문서를 전달하기 위해 이용한다.
  - ✓ TCP와 UDP를 이용하며 80번 포트에서 데이터를 주고 받는다.
  - ✓ 웹 브라우저로 도메인 주소 "`http://csl.skku.edu/`" 에 접속한 것을 소켓 연결의 측면으로 간단히 보면 다음과 같다.
    1. 웹 브라우저 프로세스(이하 클라이언트)가 새로운 TCP 소켓을 만든다.
    2. "csl.skku.edu" 서버(이하 서버)의 TCP 80번 포트에 접속해 연결을 맺는다.
    3. 클라이언트는 "/" 에 위치한 웹 문서를 달라는 HTTP 요청 메시지를 보낸다.
    4. 서버는 요청에 대응하는 "/"에 해당하는 HTML 문서를 찾거나 만들고, 적절한 HTTP 응답 헤더와 HTML 문서를 클라이언트에게 전송한다.
    5. 소켓 연결을 종료하고, 새로운 사용자 요청이 발생하면 1로 되돌아간다.
  - ✓ HTTP에 관한 더 구체적으로 알고 싶다면, 다음 링크와 교재 11.5장을 참고한다.
    http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

- HTTP 통신

요청 (여기서 빨간색 /은 서버의 상대 주소 "/" 에 위치한 문서를 요청하는 것이다)

```
GET / HTTP/1.1
Host: csl.skku.edu
(EMPTY LINE)
```

응답

```
HTTP/1.1 200 OK
Date: Mon, 21 Oct 2013 15:02:18 GMT
Server: Apache/2.2.16 (Debian)
X-Powered-By: PHP/5.3.3-7
Expires: Tue, 01 Jan 2002 00:00:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Vary: Accept-Encoding
Content-Length: 7034
Connection: close
Content-Type: text/html; charset=UTF-8
(EMPTY LINE)
<!DOCTYPE html PUBLIC "……">
<html>
[[이하 생략]]
```

HTTP response header

HTTP body (html, png, ...)

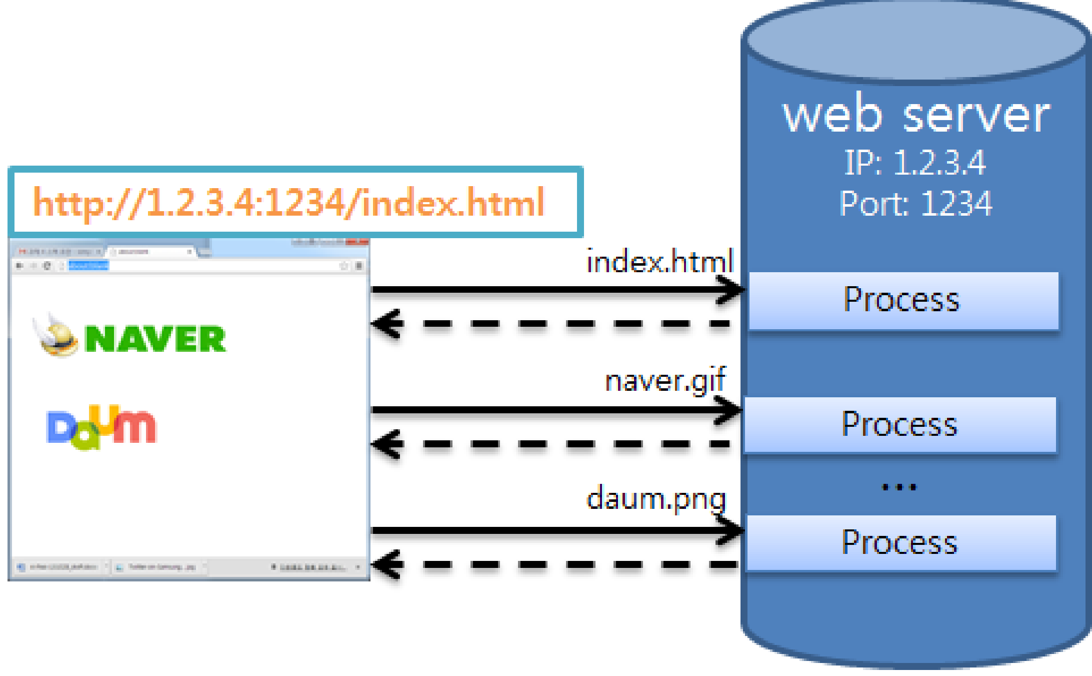
EMPTY LINE은 **HTTP의 개행 문자(<CR><LF>, "\r\n")**로만 이루어진 라인이다. 요청의 경우 헤더의 종료를 알리며, 응답에선 HTTP 응답 헤더와 HTML 문서 본문의 경계를 나눠주는 역할을 한다. HTTP 정의에 따르면, 헤더(응답과 요청 모두)에서 한 라인의 끝은 <LF>가 아닌 <CR><LF>으로 정의한다. (carriage return and line feed)

간단히 테스트해볼 수 있는 코드는 다음과 같다.

```
char request[] = "GET / HTTP/1.1\r\nHost: csl.skku.edu\r\n\r\n";
char response[5000];
write(sockfd, request, strlen(request));
int read_bytes = read(sockefd, response, 4999);
response[read_bytes] = 0;
fgets(response, stdout);
```

- HTTP Version
  - ✓ HTTP는 1991년에 V0.9, 1996년에 V1.0, 1999년에 V1.1이 발표되었고, 현재 거의 모든 상용 프로그램은 V1.1을 따르고 있다. (통신 예제의 "HTTP/1.1" 이란 문구)
  - ✓ V1.0에서 V1.1으로 발전하면서 개선된 사항엔, 한 번 TCP 연결을 맺으면 해당 연결에서 하나 이상의 요청/응답을 처리하도록 하는 **Persistent connections**, 여러 요청/응답을 하나로 묶어서 한꺼번에 전송하는 **Pipelining** 등이 있다.
  - ✓ 하지만 이런 기능들을 구현하는 것은 복잡하니, 본 과제에선 HTTP/1.0 을 따르는 서버를 구현한다. (`http.c::make_http_header()` 내의 "HTTP/1.0" 참고)
- 브라우저의 시각에서 본 웹 서버

웹 서버를 그림으로 그리면 다음처럼 동작한다.



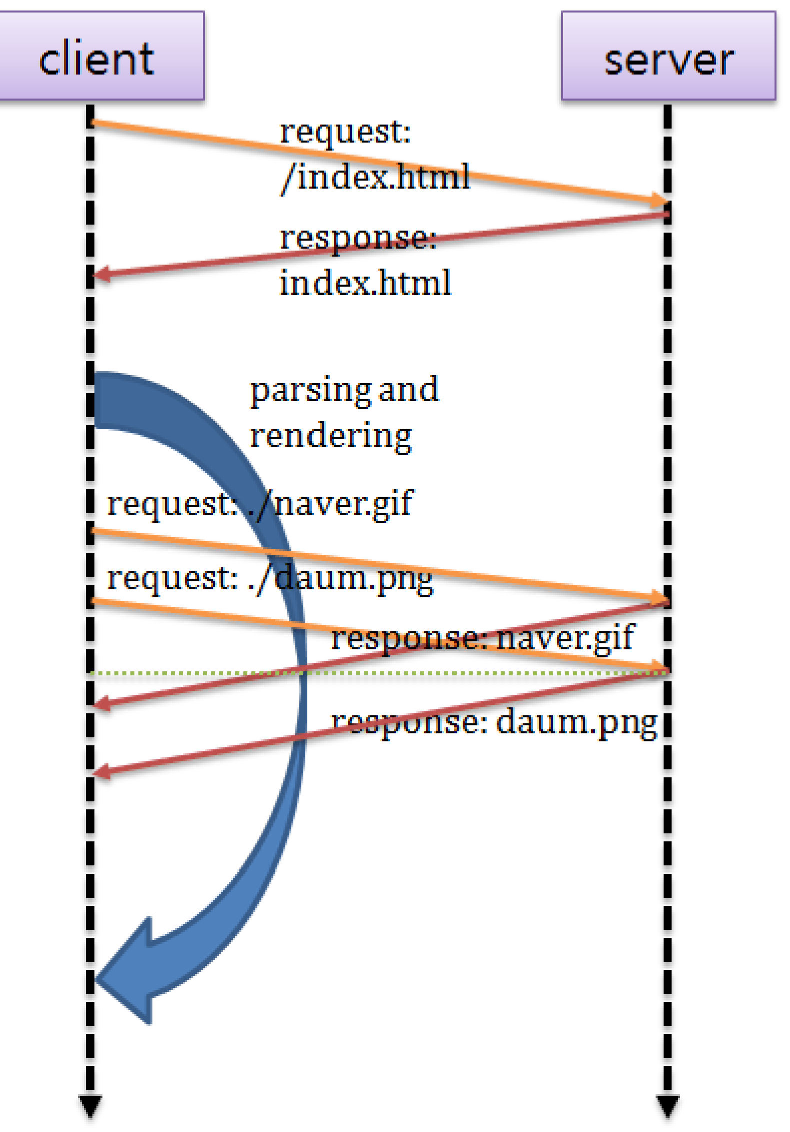
위 그림의 브라우저처럼 출력되는 html 문서의 코드는 다음 상자에 있다.

```
<html>
<body>
<img src="./naver.gif"><br>
<img src="./daum.png">
</body>
</html>
```

브라우저는 서버로부터 위와 같은 html 문서를 받으면, 위에서 아래로 해석하며 우리가 보는 화면을 그린다. 거의 모든 html 문서는 독자적으로 완성되지 않고 다른 외부 파일을

같이 이용하는데, 예를 들면 javascript, css, 그림 등이 있고 위 코드에선 붉은 색으로 강조된 그림 파일에 해당한다. 이런 외부 자원을 만나면 다시 서버로 요청을 보내 해당 파일을 받는다. 모든 외부 자원을 받은 다음, 이 모든 파일을 조합하여 하나의 페이지를 완성하여 유저에게 보여준다.

아래 그림은 위 html 코드를 그리기 위하여 브라우저와 서버 사이에 일어나는 통신의 순서를 나타낸 그림이다. 어떤 요청에 대한 응답 도중에 새로운 요청에 대한 응답을 시작할 수 있어야 함을 참고 (**Multi-processing**)



4. Specification

- (참고) 과제를 간단히 하기 위해, 웹 브라우저로 접속하면 브라우저에 DEFAULT란 화면을 보여주도록 하는 서버 코드가 스켈레톤 코드로 제공된다.

<서버 생성>

- 서버의 기본 포트는 1234이고, 프로그램 인자가 1개 존재하면 그것을 포트로 쓴다.
- 소켓에 접속 요청이 들어오면, 연결을 맺고 fork()로 자식 프로세스를 생성하여 해당 연결을 전담하여 처리하게 하고, 바로 다음 요청을 처리하기 위해 대기한다.

<요청의 처리>

- 소켓에서부터 클라이언트의 요청을 전부 읽어낸다.
  - ✓ 요청의 마지막은 위 예시처럼 <EMPTY LINE>만으로 이뤄진 라인으로 구성된다.
  - ✓ (참고) 상용 브라우저에서 들어오는 요청은 3장의 HTTP 요청 예시처럼 세 줄이 아니라 더 길다.
  - ✓ (참고) 채점시 서버에서 들어오는 한 요청의 크기는 1024바이트를 넘지 않는다.
- 클라이언트의 요청의 첫째 줄을 분석해 적절한 문서를 찾는다.
  - ✓ 요청의 첫째 줄은 "GET /test.txt HTTP/1.1" 같은 형식으로 시작하는데, 여기서 상대 주소 "/test.txt" 를 구하고, 이를 "$(HOME_DIR)/test.txt" 으로 조합한 뒤 해당 파일을 열어 파일의 내용을 클라이언트에게 전송한다.
  - ✓ 기존 과제와 마찬가지로, $(HOME_DIR)은 환경 변수 HOME_DIR의 값이다.
  - ✓ 만약 상대 주소가 "/" 라면, "/index.html" 이라는 상대 주소를 받았다고 본다.
- 파일 확장자는 html, png, gif, jpg만 허용한다. (다른 확장자는 이용할 권한이 없다)

<응답의 처리>

- 클라이언트의 요청에 대응하는 응답을 소켓으로 보낸다.
- HTTP 응답은 <HTTP 헤더> + <EMPTY LINE> + <파일의 내용> 으로 구성된다.
- HTTP 헤더는 다음 정보를 이용해 만든다.
  - ✓ 과제를 간략히 하기 위해 3장 HTTP 응답 예시에서 굵게 처리된 부분인 <Status-code, Content-type, Content-length>만 만들어 보내준다.
  - ✓ http.c 에 구현된 make_http_header() 함수를 이용하면, HTTP 헤더로 사용할 수 있는 문자열을 동적으로 생성해준다. **이 헤더를 사용한 후 반드시 free()로 할당된 자원을 해제하여야 한다.**
  - ✓ Status-code는 다음 3 경우만 고려한다.

정상일 때	NO_ERR
해당 파일에 접근할 권한이 없을 때	ERR_NOT_ALLOWED
파일이 존재하지 않을 때	ERR_NO_FILE

- ✓ Content-length는 해당 파일의 크기이다. <파일의 내용>의 크기는 Content-length와 일치하여야 한다.
- ✓ Content-type은 해당 파일의 확장자에 따라서 `http.h`에 선언된 적절한 `enum` 값을 이용한다.

HTML 파일	TYPE_HTML
JPEG 파일	TYPE_JPEG
PNG 파일	TYPE_PNG
GIF 파일	TYPE_GIF
에러 발생	TYPE_UNKNOWN

- 파일의 내용은 기존에 하던 것처럼 해당 파일을 열어 읽고 소켓에 쓴다.
  - ✓ 만약 에러가 발생한 경우, "loading fail"이란 문자열을 <파일의 내용>으로 간주하고 소켓에 쓴다. 이 경우 Content-length는 12바이트이다. (문자열 길이)
- 전송을 마치고 소켓을 닫는다.

<연결 종료 이후>

- 부모 프로세스는 스스로 종료하지 않고 무한히 연결을 기다린다.
- 자식 프로세스는 클라이언트에게 전송을 마치면 종료한다.

<멀티 프로세스 처리>

- 부모는 자식 프로세스가 종료되면 `SIGCHLD` 시그널을 전달받는데, 이 시그널을 처리하는 핸들러를 등록하여 `wait*()` 를 수행한다.
  - ✓ 해당 시그널을 수신하면 좀비 상태에 있는 모든 자식 프로세스를 종료하여야 한다.
  - ✓ (참고) `wait()`의 경우 자식이 종료될 때까지 기다리지만, `waitpid()` 함수를 쓰고 옵션으로 `WNOHANG` 을 주면 종료한 자식이 없을 경우 바로 리턴한다. 구체적인 정보는 `$ man 2 wait`를 참고.
- 부모는 자식 프로세스를 성공적으로 종료하면, `print_pid()` 함수를 호출하고 함수의 인자로 종료한 프로세스 번호를 넘긴다.
  - ✓ 예를 들어, `wait*()` 의 리턴 값이 5012일 경우 다음과 같이 호출한다.
  - ✓ `print_pid(5012);`

5. Additional Information

- **소켓 `write()` 시엔, 내가 요청한 크기만큼 실제로 송신된다는 보장이 없다.**
  - ✓ 예를 들어, 받는 쪽과 보내는 쪽의 read/write 처리 속도가 다를 경우, 소켓 버퍼에 처리되지 못한 데이터가 쌓이다 어느 순간 가득 차게 된다. 이 때 write를 수행하면 요청한 크기보다 작은 양만큼 수행될 수 있다.
  - ✓ 어떤 파일을 읽어서 소켓에 쓰는 예시이다.

```
int length = read(filefd, file_buf, BUF_SIZE);
int sent = 0;
do {
  sent += write(socketfd, file_buf + sent, length - sent);
} while (sent < length); // until write is done
```

  - ✓ 내가 100 바이트를 보내고 싶다면, loop를 돌며 `write()` 하여야 하고, 시도한 모든 `write()` 리턴 값의 합이 정확히 100 바이트가 될 때까지 수행하는지 확인하여야 한다.

- **소켓 read() 시엔, 전송 규약이 정해지지 않으면 보내는 쪽이 몇 바이트를 write 했는지 알 수 있는 방법이 없다.**
  - ✓ HTTP 규약에 따르면, HTTP 요청의 마지막 4문자는 <CR><LF><CR><LF>로 구성되니 그것을 확인하면 클라이언트의 요청 전송이 끝났는지 확인할 수 있다. (교재 11.4장과 스켈레톤 코드, 본 문서 3장 예제 참고)
- 시그널 핸들러가 수행 중일 땐 새로운 시그널을 수신하지 못 한다. (교재 12.1장)
- 교재 11.5장과 11.6장에 HTTP에 대한 설명과 예제 웹 서버가 있다.

6. Restriction

- 과제는 본인이 직접 설치한 리눅스 환경에서 구현한다.
  - ✓ 테스트 서버에서 컴파일, 실행시킬 수 없는 코드는 과제 제출물로 인정하지 않는다.
  - ✓ 테스트 서버는 Linux kernel 3.0-32bit, gcc 4.6.1 를 이용한다.
- 시스템 콜을 사용하여 프로그램을 작성하며, Standard C library의 `malloc()`, `free()`, `getenv()` 이외 함수는 사용하지 않는다.
- 파일 입출력의 경우 `open()`, `read()`, `write()`, `close()`, `lseek()` 시스템 콜 함수 중 필요한 것을 선택하여 사용한다.

- 프로세스 생성의 경우 fork(), wait*() 계열 POSIX 함수와 관련된 매크로 함수를 선택하여 사용한다.
- 소켓의 경우 강의 교안에 제시된 함수와 관련 함수를 사용한다.
- 시그널 처리의 경우 signal() 시스템 콜을 이용한다.
- 추가적으로 stat(), fstat() 시스템 콜을 사용할 수 있다.
- 필요한 경우 함수를 직접 구현하여 사용한다.
- 본 과제에서는 webserver.c 만 수정할 수 있다.
- **어떤 자원을 할당 받으면, 프로그램 종료 전에 반드시 해당 자원을 직접 해제한다.**
- **컴파일 시 발생하는 모든 warning을 잡아야 한다.**

7. Skeleton Codes

- 본 과제 수행을 위하여 아래와 같이 7개의 파일이 주어진다.
  - ✓ Makefile GNU make utility를 위해 사용되는 파일.
  - ✓ webserver.c webserver를 구현해 넣을 파일. 기본 코드가 구현되었다.
  - ✓ http.c make_http_header() 함수가 구현된 파일
  - ✓ http.h 프로그램 작성에 필요한 정보가 있는 헤더 파일
  - ✓ index.html 기본 제공 webserver 프로그램을 테스트할 수 있는 페이지
  - ✓ daum.png index.html을 그리기 위해 포함된 그림 (다음 로고)
  - ✓ naver.gif index.html을 그리기 위해 포함된 그림 (네이버 로고)

8. Hand in instruction

- 작성한 프로그램 코드 상단에 이름과 학번을 주석으로 표기한다.
- 작성한 과제 코드는 "**학번.tar.gz**" 형태로 압축해 http://sys.skku.edu에 제출한다.
  - ✓ 작업 디렉토리에서 "$ make tar 학번" 명령을 이용해 파일을 "학번.tar.gz" 으로 압축할 수 있다.
- 작성한 과제 코드의 디자인와 별도로 구현에 관한 내용을 담은 보고서를 "**학번.pdf**" 파일로 채점 서버에 제출한다.

9. Logistics

- 과제 제출 결과는 http://csl.skku.edu/SSE2030F13/Assignments 에서 확인할 수 있다.

- 과제 제출 기한은 http://sys.skku.edu 서버시간을 기준으로 하며, 기한 이후 24시간 내에 제출할 경우 30%, 48시간 내에 제출할 경우 60% 감점한다. 그 이후 제출이 불가능하며, 0점 처리한다.
- **본 과제에선 문서를 잘 작성하는 것이 중요하다.**
- 과제 점수는 컴파일/실행 가능 여부, 작성한 함수의 완성도, 출력 결과 및 작성한 문서에 의하여 평가된다.
- 과제에 대한 의논은 함께 할 수 있으나, 프로그램 소스코드 작성은 스스로 해야 한다.
    - ✓ 다른 사람의 과제를 copy 한 경우, 두 사람 모두 0점 처리하며 **학점상의 불이익이 있다**. 인터넷 등에서 찾은 소스 코드를 그대로 copy 한 경우에도 0점처리한다. **두 번 이상 적발되면 F 학점**을 받는다.