

# 디렉토리 인덱싱 웹서버

컴퓨터과학과 201710895 김동규

유닉스 프로그래밍 프로젝트

프로젝트 기간: 2019.11.26 ~ 2019.12.09

# 목차

## 발표

- 프로젝트 소개
- 프로그램 구현 개요
- DEMO

## 산출물

- 요구사항
- 소스코드 설명
- 사용설명서
- 어려웠던 점





# 프로젝트 소개

# 프로젝트 소개

- Apache HTTP Server는 현재 세계에서 가장 인기있는 웹 서버이다. 리눅스, 윈도우 가리지 않고 잘 설치되며 잘 이용된다.
- 보통 APM 또는 XAMPP 를 설치해서 웹서버를 돌려보곤 했을 것이다.
- 이러한 웹 서버에는 다양한 기능들이 존재한다.
- 그 중 하나가 디렉토리 리스팅이라는 기능이다.
- 본인은 아파치 웹서버가 지원하는 디렉토리 리스팅 기능을 모방해서 구현해보고자 한다.

# 프로젝트 소개 — 아파치 웹서버 디렉토리 리스팅 예시

## Index of /img

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 <a href="#">Parent Directory</a>		-	
 <a href="#">module table bottom.png</a>	2013-11-20 21:48	751	
 <a href="#">module table top.png</a>	2013-11-20 21:48	337	
 <a href="#">rabbit.jpg</a>	2019-12-04 13:46	184K	

*Apache/2.4.16 (Win32) OpenSSL/1.0.1p PHP/5.6.12 Server at localhost Port 80*

# 프로젝트 구현 개요

# 프로젝트 구현 개요

서버와 클라이언트(웹 브라우저)와의 소켓 통신

# 프로젝트 구현 개요

- 서버

1. 요청이 들어올때까지 대기
3. URI 파싱 후 클라이언트가 접속하고자 하는 경로 확인
4. 요청 경로가 일반파일인지 디렉토리인지 확인
5. 각 상황에 맞게 처리
  - 일반파일: 다운로드 및 출력
  - 디렉토리: 목록 출력
6. 응답 헤더 및 본문 작성 후 클라이언트에게 전송

- 웹브라우저

2. 서버 접속 요청(URI입력함)  
(192.168.10.11:9090/)
7. 응답 받고, 웹페이지에서 확인



# 프로젝트 구현 개요(사용된 함수)

- ch01:유닉스 시스템 프로그래밍 개요
  - perror() : 에러 메시지 출력에 사용
- ch02:파일 입출력
  - open() : 클라이언트가 요청한 일반 파일 또는 로그 파일 열기에 사용
  - close() : 소켓 또는 파일 디스크립터 닫기
  - read() : 파일 또는 클라이언트 요청 읽기에 사용
  - write() : 파일 또는 클라이언트 응답 쓰기에 사용
  - printf() : 디버깅 시 값 출력에 많이 사용됨
  - openat() : 해당 파일 디스크립터에 파일 열 때 사용
- ch03:파일과 디렉토리
  - fstat() : 파일/디렉토리 정보 검색 시 사용
  - fdopendir() : 디렉토리 엔트리 포인터를 반환 받기 위해 사용
  - closedir() : 디렉토리 닫기에 사용
  - readdir() : opendir()로 열기한 디렉토리에 대해, 그 안에 있는 모든 파일과 디렉토리 정보를 구하는 데 사용
- ch04:시스템 정보
  - time() : 로그 현재 시간 입력에 사용
  - strftime() : 로그 및 파일 최근 수정된 날짜에 형식에 맞춰서 쓰게 하기 위해서 사용
  - localtime() : 위와 같이 사용됨
- ch05:프로세스 정보
  - getpid() : 로그 작성 시 프로세스 ID 작성하도록 함
- ch06:프로세스 생성과 실행
  - fork() : 동시 접속 및 다중 프로세스에 사용됨
  - exit() : 에러 발생 시 강제 종료에 사용됨

# 프로젝트 구현 개요(사용된 함수)

- ch07:시그널
  - signal() : 지정한 시그널을 받았을 때 어떻게(핸들러함수) 처리 (주로 브라우저에 관한 문제를 해결하기 위해서 사용, 시그널 무시에도 사용)
- ch08:메모리 매핑
  - mmap() : 파일 본문의 내용을 메모리 맵핑 후 그대로 클라이언트 응답 body에 write() 하는데 사용됨
  - munmap() : 메모리 맵핑 해제에 사용됨
- ch11:소켓 프로그래밍 기초
  - htonl() : 32비트 HBO를 32비트 NBO로 변환(IPv4주소 변환용에 사용됨)
  - htons() : 16비트 HBO를 16비트 NBO로 변환(Port주소 변환용에 사용됨)
  - socket() : 소켓 생성에 사용됨
  - setsockopt() : 소켓 옵션 지정 (주로 오류 처리 및 성능향상을 위해서 사용)
  - bind() : 소켓 바인딩
  - listen() : 소켓 리스닝
  - accept() : 클라이언트의 접속 허용
- 기타 문자열 관련 함수
  - atoi() : 문자열 정수로 변환
  - memset() : 메모리 값 지정
  - sprintf() : 데이터를 형식에 맞추어 쓴다
  - strcmp() : 문자열 비교
  - sscanf() : 문자열에서 형식화 된 데이터를 읽어옴
  - strlen() : 문자열 길이 구함
  - strcpy() : 문자열 복사
  - strchr() : 마지막 문자를 찾아서, 해당 포인터 반환

DEMO




발표 끝

산출물들

# 요구사항-과제 개요

- 아파치 웹서버의 디렉토리 인덱싱을 모방하는 것이 이번 과제의 목표다.
- 평소에 APM(Apache, PHP, Mysql)을 설치해서 웹서버를 돌리곤 했는데, 정작 이런 서버가 어떻게 만들어졌고 동작하는 지에 대해서는 궁금한데 알길이 없었다.
- 유닉스 프로그래밍 수업을 들으면서 감이 잡혔고, 구현해보고자 한다.

## Index of /img

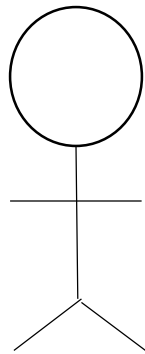
<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 <a href="#">Parent Directory</a>		-	
 <a href="#">module table bottom.png</a>	2013-11-20 21:48	751	
 <a href="#">module table top.png</a>	2013-11-20 21:48	337	
 <a href="#">rabbit.jpg</a>	2019-12-04 13:46	184K	

Apache/2.4.16 (Win32) OpenSSL/1.0.1p PHP/5.6.12 Server at localhost Port 80

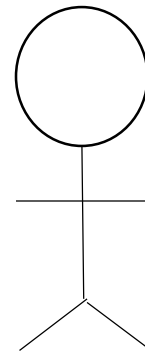
[아파치(Apache) 웹서버 디렉토리 인덱싱/리스트]

# 요구사항-사용자 분석

액터	설명
서버	디렉토리 리스팅 서비스를 구동한다.
클라이언트	웹 브라우저를 통해서 서버와 소켓 통신을 하고, 주로 HTTP 통신을 하게 된다.



서버



클라이언트

# 요구사항-기능 요구사항

구분	상세
통신	<ul style="list-style-type: none"><li>• 웹 통신을 할 수 있도록 소켓 통신을 통해 구현한다.</li></ul>
클라이언트 요청(Request)	<ul style="list-style-type: none"><li>• 웹 브라우저에서 보내는 HTTP Request 를 처리한다.</li><li>• URI을 기반으로 파일 또는 디렉토리 경로를 파싱한다.</li></ul>
클라이언트 응답(Response)	<ul style="list-style-type: none"><li>• 파일 확장자별로 다른 응답이 나오도록 한다.<ul style="list-style-type: none"><li>- 텍스트, 이미지, 웹페이지, 영상</li></ul></li><li>• 파일이면 다운로드가 될 수 있도록 한다.</li><li>• 디렉토리이면 해당 디렉토리 목록을 웹페이지에 보여줄 수 있도록 한다.</li></ul>
웹 로그	<ul style="list-style-type: none"><li>• 서버 시작, 클라이언트 접속, 에러 관련해서 로그를 남기고 파일로 실시간 저장하도록 한다.</li></ul>
인터페이스	<ul style="list-style-type: none"><li>• 최대한 Apache 디렉토리 리스팅과 유사하게 한다.</li></ul>



# 요구사항-비기능 요구사항

요구항목	설명
제약조건	<ul style="list-style-type: none"><li>교재 단원 별로 함수 1개 이상 사용하기 (단, 통신 관련은 소켓을 이용)</li></ul>
성능	<ul style="list-style-type: none"><li>다중 프로세스로 처리되어 클라이언트의 서버 접속 또는 요청/응답 처리 속도 향상</li></ul>
보안성	<ul style="list-style-type: none"><li>서버가 직접 종료하지 않는 이상 종료되면 안 된다.</li></ul>

# 소스코드 설명

함수	설명
main	소켓 통신 관련 처리
webprocess	웹 요청/응답 처리(총괄)
requestprocess	클라이언트 요청 처리(URI)
getfiletype	파일 확장자 처리
sizetranform	파일 크기 변환(B, KB, MB, GB)
directoryhandler	클라이언트 디렉토리 요청 처리
servestatic	클라이언트 파일 요청 처리
clienterror	클라이언트 에러 처리(400, 404)
weblog	웹 로그 저장(web.log)

# 소스코드 설명 전 명령 인자 관련 설명

webserver 9090 명령 실행, 명령인자는 포트번호 하나임.  
프로그램 실행 후 바로 백그라운드 프로세스로 실행됨

```
pintos@pintos-VirtualBox:~/hw2$ webServer 9090
pintos@pintos-VirtualBox:~/hw2$ ps
```

PID	TTY	TIME	CMD
5427	pts/4	00:00:00	bash
8016	pts/4	00:00:00	webServer
8019	pts/4	00:00:00	ps

선언된 헤더 파일	
1. #include <stdio.h> 2. #include <stdlib.h> 3. #include <string.h> 4. #include <fcntl.h> 5. #include <unistd.h> 6. #include <sys/types.h> 7. #include <sys/socket.h> 8. #include <netinet/in.h> 9. #include <netinet/tcp.h> 10. #include <signal.h> 11. #include <sys/stat.h> 12. #include <dirent.h> 13. #include <time.h> 14. #include <arpa/inet.h> 15. #include <errno.h> 16. #include <sys/mman.h>	17. 18. #define LOG 100 19. #define ERROR 200 20. #define MAXLINE 1024 21. 22. #define ROOT "hw2" 23. #define SERVERNAME "MyServer" 24. #define VERSION "1.0" 25.
1~16 : 해당 프로그램에 필요한 헤더 파일들 추가	17~18 : 로그 기록 함수에서 사용되는 매크로, 해당 로그가 에러인지 그냥 일반 기록 용인지 int 형으로 구분한다. 19 : 한 줄의 최대 길이를 정의하고 있다. 21 : 웹서버가 처음에 실행되는 현재 디렉토리 경로로써 ROOT로 명명한다. ~/hw2 가 프로그램이 실행되는 경로다. 22~23 : 이후 웹 페이지 하단에 출력될 정보이다.

정의된 구조체	
<pre> 26. typedef struct { 27.     char *ext; // extension 28.     char *filetype; // mime_type 29. } extension_map; 30. 31. typedef struct { 32.     char filename[512]; 33.     off_t offset;           // for range 34.     size_t end; 35. } http_request; 36. </pre>	<pre> 37. extension_map file_types [ ] = { 38.     {".css", "text/css"}, 39.     {".js", "application/javascript"}, 40.     {".pdf", "application/pdf"}, 41.     {".mp4", "video/mp4"}, 42.     {".svg", "image/svg+xml"}, 43.     {".xml", "text/xml"}, 44.     {".gif", "image/gif" }, 45.     {".jpg", "image/jpeg"}, 46.     {".jpeg","image/jpeg"}, 47.     {".png", "image/png" }, 48.     {".zip", "image/zip" }, 49.     {".gz", "image/gz" }, 50.     {".tar", "image/tar" }, 51.     {".htm", "text/html" }, 52.     {".html","text/html" }, 53.     {0,0} }; //NULL, NULL </pre>
<p>26~29 : 파일 확장자 구조체이다. (37번째 줄 참고)</p> <p>31~35 : http request를 처리하기 위한 구조체이다. 클라이언트가 요청한 URI 및 http request를 파싱해서 이 구조체에 저장한다.</p>	<p>37~53 : 서버에서 지원하는 파일 확장자 모음이다.</p>

## 전역변수 및 함수 원형 선언

```
54.  
55.char *default_file_type = "text/plain"; // default mime type  
56.  
57.char *SERVERADDRESS = "127.0.0.1";  
58.int PORT = 9999;  
59.  
60.void webprocess(int c_sock, struct sockaddr_in *c_addr);  
61.void weblog(int type, char s1[ ], char s2[ ], int n);  
62.void requestprocess(int fd, http_request *req);  
63.static const char* getfiletype(char *filename);  
64.void sizetransform(char* buf, struct stat *stat);  
65.void directoryhandler(int out_fd, int dir_fd, char *filename);  
66.void servestatic(int outfd, int infd, http_request *req, size_t total_size);  
67.void clienterror(int fd, int code, char *msg, char *lmsg);  
68.
```

55 : extension\_map 구조체에 매칭되는 파일 확장자가 아니라면, 기본값으로 text/plain으로 mime type을 설정한다.  
57 : 서버 IP 주소를 나타낸다. 이후 서버 관련 정보 출력 시나 로그 기록 시에 사용된다.  
58 : 프로그램 실행 시 명령 인자로 포트번호를 주어지지 않았을 경우 기본값으로 9999 포트 번호를 사용하도록 한다.  
하지만 사실 명령인자를 무조건 받도록 검증하는 문장이 있기 때문에 기본값의 개념보다는 초기화의 개념이다.  
60~67 : 본 프로그램에서 구현할 함수의 원형을 선언

main 함수	
<pre> 69.int 70.main(int argc, char *argv[ ]) { 71.    struct sockaddr_in s_addr, c_addr; 72.    int      s_sock, c_sock; 73.    int      len; 74. 75.    unsigned short port; 76. 77.    int      pid; 78. 79.    if(argc != 2){ 80.        printf("usage: webServer port_number"); 81.        return -1; 82.    } 83. </pre>	<p>71 : 서버와 클라이언트 소켓주소를 정의한다.</p> <p>72 : 서버와 클라이언트 소켓 file descriptor를 정의한다.</p> <p>73 : accept 함수에서 필요할 클라이언트 주소의 길이를 넘겨줄 때 사용된다.</p> <p>75 : 포트번호를 저장하기 위한 변수다.</p> <p>76 : fork( )함수의 반환값을 저장하기 위한 변수다.</p> <p>79~82 : 명령인자가 프로그램명 외에 한 개가 아닐 경우 usage를 출력하고 종료한다.</p>

main 함수(continued)		
84.	if(fork( ) != 0)	//background process
85.	return 0;	// parent return to shell
86.		
87.	weblog(LOG, SERVERADDRESS, "web server start", getpid());	
88.		
89.	if((s_sock=socket(AF_INET, SOCK_STREAM, 0))<0){	
90.	weblog(ERROR, "SYSCALL", "web server listen socket open error", s_sock);	
91.	}	
92.		
84~85 : fork() 후 부모는 바로 종료, 자식은 작업을 수행하도록 함으로써, 백그라운드 프로세스로 전환한다.		
87 : 웹서버의 시작을 알리는 로그를 저장한다. 실제 로그는 아래와 같이 web.log 파일에 저장된다.		
[2019-12-09 02:18:10] STATUS 127.0.0.1 web server start 9049		
89~91 : 소켓을 생성하고, 실패 시 로그를 저장한다. 그리고 weblog 함수는 내부적으로 ERROR일 경우에는 강제 종료하도록 되어 있다. 일반 LOG일 경우엔 종료하지 않는다.		



## main 함수(continued)

```
93.     PORT = port=atoi(argv[1]);
94.     if(port > 60000)
95.         weblog(ERROR, "PORT", "invalid port number", port);
96.
97.     int optval = 1;
98.     /* Eliminates "Address already in use" error from bind. */
99.     if (setsockopt(s_sock, SOL_SOCKET, SO_REUSEADDR,
100.         (const void *)&optval , sizeof(int)) < 0)
101.         return -1;
102.
103.     // 6 is TCP's protocol number
104.     // enable this, much faster : 4000 req/s -> 17000 req/s
105.     if (setsockopt(s_sock, 6, TCP_CORK,
106.         (const void *)&optval , sizeof(int)) < 0)
107.         return -1;
108.
```

93 : 명령인자로 받은 포트번호를 atoi( ) 함수로 정수로 변환 후 port 변수와 전역변수 PORT에 저장한다.

94~95 : 만약 포트번호가 60000을 넘을 경우에는 에러 로그를 남기고 종료한다.

97 : setsockopt( ) 함수를 사용하기 위해서 변수를 초기화 한다.

98~101 : "Address already in use"라고 가끔 바인딩 시에 나오는 오류를 없애준다.

103~107 : TCP 프로토콜 사용 시에 생기는 지연 문제를 해결하는 알고리즘을 사용하도록 한다. 성능 향상을 위해 사용한다.

## main 함수(continued)

```
109.    memset(&s_addr, 0, sizeof(s_addr));
110.    s_addr.sin_family = AF_INET;
111.    s_addr.sin_addr.s_addr = htonl(INADDR_ANY);
112.    s_addr.sin_port = htons(port);
113.
114.    if(bind(s_sock, (struct sockaddr *) &s_addr, sizeof(s_addr)) < 0){
115.        perror("bind");
116.        weblog(ERROR, "BIND", "server cannot bind", 0);
117.    }
118.
119.    if(listen(s_sock, 10) < 0) // Max_Client = 10
120.        return -1;
121.
```

109~112 : 서버 소켓 구조체에 family 타입, IP주소, Port 주소를 설정한다.

113~116 : 소켓을 바인딩 한다. 가끔 이미 사용하고 있는 포트번호에 바인딩할 수도 있는데, 이때에는 바로 에러를 stdout에 출력하고 동시에 로그도 남기고 종료하도록 한다.

117~118 : 서버 리스닝을 한다. 최대 연결 가능한 클라이언트 수는 10개로 한다.

```
122.    signal(SIGCHLD, SIG_IGN);           // ignore child death(prevent zombies)
123.    signal(SIGPIPE, SIG_IGN);          // ignore browser interrupts
124.
```

119 : 클라이언트가 계속해서 http request를 보내오고, 이것을 즉시 바로바로 처리해야되기 때문에 waitpid( ) 함수를 이 때 사용하게 되면 자칫 fork( ) 된 자식들이 일을 끝낼 때까지 기다리게 되어 무한루프가 도는 현상이 일어난다. 그래서 waitpid( )함수를 사용할 수가 없기 때문에 자식보다 부모 프로세스가 먼저 끝나게 됨으로써, 자식이 좀비프로세스가 될 수밖에 없는 구조이다. 이 때 signal 함수를 이용하여 부모가 자식의 종료 시그널을 무시하도록 함으로써, 시스템 프로세스가 주기적으로 좀비 프로세스를 종료하도록 한다. 이로써 좀비 프로세스(defunct 문제)를 해결하였다.

120 : 브라우저에서 인터럽트 신호를 서버에 가끔 전달하게 되는데, 이때 이 인터럽트 신호를 허용하게 되면, 서버의 프로세스들이 모두 종료가 되는 상황이 발생하게 된다. 이를 해결하고자 브라우저에서 보내는 인터럽트 신호를 무시하도록 한다.

main 함수(continued)	
<pre> 125.     while(1){ // Accept 126.         len = sizeof(c_addr); 127.         c_sock = accept(s_sock, (struct sockaddr *) &amp;c_addr, &amp;len); 128.         if(c_sock &lt; 0){ 129.             perror("accept"); 130.             exit(-1); 131.         } 132.         if((pid = fork( )) &lt; 0) { 133.             perror("fork"); 134.             exit(-1); 135.         } else if(pid == 0) { // child 136.             //printf("child forked pid = %d\n", getpid()); 137.             close(s_sock); 138.             webprocess(c_sock, &amp;c_addr); 139.             close(c_sock); 140.             return 0; </pre>	<pre> 141.         } else { 142.             close(c_sock); 143.         } 144.     } 145. 146.     return 0; 147. } 148. </pre>
<p>125 : 무한 루프를 돌게 해서 클라이언트로부터의 요청을 계속 받는다.</p> <p>126 : 클라이언트 주소의 크기를 구한다.</p> <p>127 : accept( ) 함수를 호출해서 클라이언트의 file descriptor을 얻는다.</p> <p>128~131 : accept( ) 함수 오류를 예외처리한다.</p> <p>132~134 : 자식 프로세스를 생성한다.</p> <p>135~140 : 자식이라면, 클라이언트의 요청을 처리한다.</p>	<p>141~143 : 부모 프로세스는 그대로 소켓을 닫고, 다시 accept( ) 함수를 기다린다.</p>

웹 요청/응답 처리(총괄) 함수 – webprocess( )	
<pre> 149.void 150.webprocess(int c_sock, struct sockaddr_in *c_addr) 151.{    // status code (default: 200, success) 152.    int        code = 200; 153.    int         fd; 154. 155.    struct stat sbuf; 156.    http_request req; 157. 158.    // request packet parsing 159.    requestprocess(c_sock, &amp;req); // get req 160. </pre>	<pre> 161.    fd = open(req.filename, O_RDONLY, 0); 162.    if(fd &lt;= 0){ 163.        code = 404; 164.        char *msg = "File not found"; 165.        clienterror(c_sock, code, "Not found", 166.                    msg); 167.    } else { </pre>
<p>152 : 기본 상태 코드를 200으로 초기화한다.</p> <p>153 : file descriptor 변수이다.</p> <p>155 : 파일의 정보를 담기 위한 구조체이다.</p> <p>156 : http request정보를 담기 위한 구조체이다.</p> <p>159 : 클라이언트의 요청을 처리하기 위한 함수이다.</p>	<p>161 : requestprocess( ) 함수가 실행되면 req 구조체에 정보가 담긴다. 이를 통해 얻은 파일이름을 통해서 파일을 열고 file descriptor을 fd에 대입한다.</p> <p>162~167 : 만약 파일이 없다면 상태코드를 404를 대입하고, clienterror 함수를 호출해서 클라이언트에게 404 File Not Found 에러를 웹페이지에 출력하도록 전송한다.</p>

웹 요청/응답 처리(총괄) 함수 – webprocess( ) (continued)	
168.	fstat(fd, &sbuf); // find file stat
169.	if(S_ISREG(sbuf.st_mode)){ // if reg file
170.	if(req.end == 0){
171.	req.end = sbuf.st_size;
172.	}
173.	if(req.offset > 0){
174.	code = 206
175.	}
176.	// read file and send to client
177.	servestatic(c_sock, fd, &req, sbuf.st_size);
178.	} else if(S_ISDIR(sbuf.st_mode)){ // if directory
179.	code = 200; // OK
180.	directoryhandler(c_sock, fd, req.filename);
<p>169 : fd를 통해서 파일의 정보를 검색한다.</p> <p>170~178 : 만약 클라이언트가 요청한 파일이 일반 파일이라면 파일의 크기를 http_request구조체에 저장하고, code를 206 (Partial Content)로 설정한 다음, servestatic( )함수를 호출하여 일반파일일 때의 상황을 처리한다.</p> <p>179~180 : 만약 디렉토리파일이라면 상태코드를 200으로 하고, directoryhandler( )함수를 호출하여 디렉토리일 때의 상황을 처리한다.</p>	

웹 요청/응답 처리(총괄) 함수 – webprocess( ) (continued)	
181.	} else { // else files
182.	code = 400;
183.	char *msg = "Unknown Error";
184.	clienterror(c_sock, code, "Error", msg);
185.	}
186.	close(fd);
187.	}
188.	weblog(LOG, inet_ntoa(c_addr->sin_addr), req.filename, code);
189.	}
190.	
181~185 : 만약 일반파일도, 디렉토리도 아니라면 본 서버에서 처리할 수 없는 파일임을 나타내기 위해서, 상태코드를 400으로, 클라이언트에게 전송되는 메시지는 Unknown Error이 출력되도록 한다.	
186 : 마지막으로 파일 디스크립터를 닫는다.	
188 : 클라이언트의 IP주소와 요청한 파일 또는 디렉토리와 상태코드를 로그에 남긴다.	

## 클라이언트 디렉토리 요청 처리 함수 – directoryhandler( )

```
191. // if request parameter is directory
192. void
193. directoryhandler(int out_fd, int dir_fd, char *filename){
194.     char buf[MAXLINE], m_time[32], size[16];
195.     struct stat statbuf;
196.     sprintf(buf, "HTTP/1.1 200 OK\r\n%s%s%s%s%s",
197.             "Content-Type: text/html\r\n\r\n",
198.             "<html> <head> <style> ",
199.             "td {padding: 1.5px 6px; text-align: center;}",
200.             ".name {text-align: left;}",
201.             "</style> <title>Unix Project</title> </head> <body>\r\n");
202.
203.     DIR *d = fdopendir(dir_fd); // return dirent
204.     write(out_fd, buf, strlen(buf));
205.
```

194 : 클라이언트 요청의 한 줄 단위로 읽을 수 있는 버퍼와 디렉토리 안에 들어있는 파일들의 최근 수정된 날짜와 파일 크기를 저장하기 위한 변수이다.

195 : 파일의 상태를 검색하기 위한 구조체이다.

196~201 : 버퍼에 클라이언트에 요청에 응답할 헤더를 작성한다.

203 : 열고자 하는 디렉토리의 엔트리를 저장한다. fdopendir( )는 fd를 기반으로 엔트리를 반환한다.

204 : 클라이언트 file descriptor(브라우저)에 버퍼의 내용을 쓰기 한다. go로 전송한다.



## 클라이언트 디렉토리 요청 처리 함수 – directoryhandler() (continued)

```
206.    // index of directory
207.    sprintf(buf, "<h1>Index of /%s</h1>", filename[0] == '.' ? ROOT:filename);
208.    write(out_fd, buf, strlen(buf));
209.
210.    // name, last modified size(dir)
211.    sprintf(buf, "<table> <tr> <th> <font color='blue'>Name</font> </th> <th> <font color='blue'>Last
    Modified</font> </th> <th> <font color='blue'>Size</font> </th> </tr> <tr> <th
    colspan='3'> <hr> </th> </tr> \n");
212.    write(out_fd, buf, strlen(buf));
213.
214.    // goto parent directory
215.    if(strcmp(filename, ".")){
216.        sprintf(buf,
217.            "<tr> <td> <a href='javascript:history.back()'>Parent Directory</a> </td> </tr> \n");
218.        write(out_fd, buf, strlen(buf));
219.    }
220.
```

207~208 : 현재 클라이언트가 조회하고 있는 디렉토리의 경로를 출력하기 위한 버퍼를 작성하고 보낸다.

211~212 : 파일이름, 최근 수정된 날짜, 파일 크기를 출력하기 위한 버퍼를 작성하고 보낸다.

215~219 : 파일이름을 비교해서 만약 "." 로써 루트 디렉토리(hw2)가 아니라면 Parent Directory를 출력도록 보낸다.

사실 Parent Directory를 누르게 되면 그냥 javascript:history.back()이 실행된다.

루트 디렉토리에서 이전으로 돌아가면 안되는 이유가 일단 보안문제가 생긴다.

클라이언트 디렉토리 요청 처리 함수 – directoryhandler() (continued)	
<pre> 221. struct dirent *dp; //ino,off,reclen,name 222. int ffd; 223. while ((dp = readdir(d)) != NULL){ // read dir stat 224.     if(!strcmp(dp-&gt;d_name, ".")    !strcmp(dp-&gt;d_name, "..")){ 225.         continue; // block . or .. 226.     } 227.     //openat: open file at dir_fd 228.     if ((ffd = openat(dir_fd, dp-&gt;d_name, O_RDONLY)) == -1){ 229.         perror(dp-&gt;d_name); 230.         continue; 231.     } 232.     fstat(ffd, &amp;statbuf); // read file stat 233.     strftime(m_time, sizeof(m_time), // time transform 234.         "%Y-%m-%d %H:%M", localtime(&amp;statbuf.st_mtime)); 235.     sizetransform(size, &amp;statbuf); //[DIR] or size(KMG) </pre>	
<p>221 : 디렉토리 엔트리 구조체 포인터이다.</p> <p>222 : 디렉토리 목록 안의 파일들의 정보를 검색하기 위한 변수이다.</p> <p>223 : 디렉토리의 목록을 조회하기 위한 반복문이다.</p> <p>224~226 : 디렉토리 목록의 "." 과 ".."은 취급하지 않도록 한다.</p> <p>228~231 : 디렉토리 목록의 파일을 읽어서 파일 디스크립터를 반환한다.</p> <p>232~235 : 파일의 정보를 읽어서, 시간을 형식에 맞게 저장하고, sizetransform( ) 함수를 실행해서 파일크기를 구한다.</p>	

## 클라이언트 디렉토리 요청 처리 함수 – directoryhandler() (continued)

```
236.         // common
237.         if(S_ISREG(statbuf.st_mode) || S_ISDIR(statbuf.st_mode)){
238.             char *d = S_ISDIR(statbuf.st_mode) ? "/" : "";
239.             // dir name/, reg name
240.             sprintf(buf,
241.                 "<tr> <td class='name'> <a
href=\"%s%s\">%s</a> <td>%s</td> <td>%s</td> </tr>\\n",
242.                 dp->d_name, d, dp->d_name, d, m_time, size);
243.             write(out_fd, buf, strlen(buf));
244.         }
245.         close(ffd);
246.     }
247.     //send to client the footer
248.     sprintf(buf, "<tr> <th colspan='3'> <hr> </th> </tr> "
249.         "<tr> <th colspan='3'>%s/%s Server at %s Port %d</th> </tr> "
250.         "</table> </body> </html>", SERVERNAME, VERSION, SERVERADDRESS, PORT);
251.     write(out_fd, buf, strlen(buf));
252.     closedir(d);
253. }
```

237~246 : 일반 파일이거나 디렉토리라면 해당 정보를 Name, Last Modified, Size 순으로 출력하도록 전송한다.  
248~253 : 최하단에는 서버의 정보를 출력하도록 전송한다. 마지막으로 디렉토리를 닫는다.

## 클라이언트 요청 처리(URI) 함수 – requestprocess()

```
255. // parse request
256. void
257. requestprocess(int fd, http_request *req){ // fd : c_sock
258.     char buf[MAXLINE], method[MAXLINE], uri[MAXLINE];
259.     int c; // c and ch is for read request line
260.     char ch;
261.     req->offset = 0;
262.     req->end = 0;
263.
264.     // read request line
265.     char *bufp = buf;
266.     for(int i = 1; i < MAXLINE; i++){
267.         if(c = read(fd, &ch, 1) == 1){
268.             *bufp++ = ch;
269.             if(ch == '\n')
270.                 break;
271.         } else break;
272.     }
273.     *bufp = 0; //'#0'
274.     sscanf(buf, "%s %s", method, uri); // version is not required
```

### [참고 이미지]

```
GET / HTTP/1.1\r\n
Host: 192.168.56.101:9090\r\n
Connection: keep-alive\r\n
Cache-Control: max-age=0\r\n
Upgrade-Insecure-Requests: 1\r\n
```

이 이미지에서는 GET과 /를 가져오게 된다.

265~274 : 클라이언트가 보낸 http request를 한줄 (\n 개행문자를 만날때까지) 읽어들이고, METHOD와 URI를 가져온다.

## 클라이언트 요청 처리(URI) 함수 – requestprocess() (continued)

```
275.  
276.    // 206 Partial Content : Request: Ranges, Response: Content-Range  
277.    while(buf[0] != '\n' && buf[1] != '\n') { /* \n || \r\n */  
278.        // read request line  
279.        char *bufp = buf;  
280.        for(int i = 1; i < MAXLINE; i++){  
281.            if(c = read(fd, &ch, 1) == 1){  
282.                *bufp++ = ch;  
283.                if(ch == '\n')  
284.                    break;  
285.            } else break;  
286.        }  
287.        *bufp = 0; /*'\0'*/  
288.        // if Range exists,  
289.        if(buf[0] == 'R' && buf[1] == 'a' && buf[2] == 'n'){  
290.            sscanf(buf, "Range: bytes=%lu-%lu",  
291.                &req->offset, (long unsigned int*)&req->end);  
292.            // Range: [start, end]  
293.            if( req->end != 0) req->end ++;  
294.        }  
295.    }
```

277~295 : 206 Partial Content(파일 다운로드 시)를 사용하기 위해 필요한 Range 값을 파싱 후 http request에 content의 시작지점과 끝지점을 저장한다. 텍스트, 이미지, 웹페이지, 특히 동영상을 지원하려면 Accept-Range 필요.

## 클라이언트 요청 처리(URI) 함수 – requestprocess() (continued)

```
296.  
297.     char* filename = uri;  
298.     if(uri[0] == '/'){  
299.         filename = uri + 1; // remove slash  
300.         int length = strlen(filename);  
301.         if(length == 0){ //localhost:9999/ connected  
302.             filename = "."; //cur dir  
303.         } else {  
304.             for(int i=0; i<length; i++){  
305.                 if(filename[i] == '?'){ //delete parameter  
306.                     filename[i] = '\0';  
307.                     break;  
308.                 }  
309.             }  
310.         }  
311.     }  
312.     //printf("filename = %s\n", filename); //.  
313.     strcpy(req->filename, filename);  
314.}
```

296~313 : 클라이언트가 요청한 URI가 / 라면 filename을 "."(루트 디렉토리=~/\hw2)로 설정하고, 혹시 만약에라도 파라미터값(?)이 있을 경우 널문자로 치환하여 없앤다. (현재버전은 파라미터는 지원하지 않는다.) 결국 클라이언트가 요청한 URI을 토대로 filename을 설정한다.

## 파일 확장자 처리 함수 – getfiletype()

```
315.  
316.// get file type  
317.static const char*  
318.getfiletype(char *filename){  
319.    char* dot = strrchr(filename, '.');  
320.    if(dot){ // strrchr : find last character('.') return pointer of this  
321.        //printf("dot = %s\n", dot);  
322.        extension_map *map = file_types;  
323.        while(map->ext){  
324.            if(strcmp(map->ext, dot) == 0){  
325.                //printf("map->ext = %s\n", map->ext);  
326.                return map->filetype;  
327.            }  
328.            map++;  
329.        }  
330.    }  
331.    return default_file_type;  
332.}  
333.
```

### [참고]

파일 확장자 처리는 servestatic() 함수에서 동작한다. 파일의 확장자를 얻어와서 웹페이지면 웹페이지, 텍스트면 텍스트, 다운로드면 다운로드 처리하게 되기 때문이다.

### [참고]

```
extension_map file_types [ ] = {  
    {".css", "text/css"},  
    {".js", "application/javascript"},  
    {".pdf", "application/pdf"},  
    {".mp4", "video/mp4"},  
    {".svg", "image/svg+xml"},  
    {".xml", "text/xml"},  
    {".gif", "image/gif"},  
    {".jpg", "image/jpeg"},  
    {".jpeg", "image/jpeg"},  
    {".png", "image/png"},  
    {".zip", "image/zip"},  
    {".gz", "image/gz"},  
    {".tar", "image/tar"},  
    {".htm", "text/html"},  
    {".html", "text/html"},  
    {0,0} }; //NULL, NULL
```

319 : strrchr(filename, '.')함수로 파일이름에서 가장 마지막 '.'을 찾아서 포인터를 반환한다. test.txt이면 포인터는 .txt의 .을 가르킨다.

323~330 : while문이 돌아가면서 extension\_map구조체 배열에 존재하는 확장자를 검색하고 존재하면 반환한다.

331 : 만일 목록에 존재하지 않는 확장자라면, text/plain으로 무조건 반환한다.

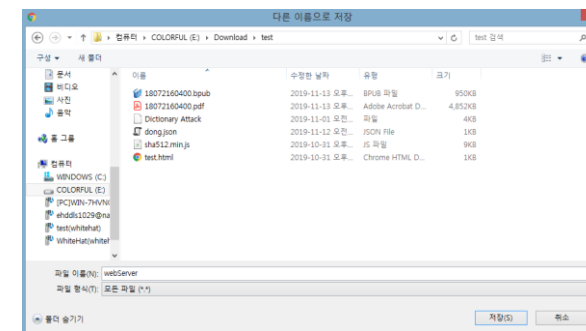
파일 크기 변환(B, KB, MB, GB) 함수 – sizetransform()	
<pre> 334. // file size formatting 335. void 336. sizetransform(char* buf, struct stat *stat){ 337.     if(S_ISDIR(stat-&gt;st_mode)){ 338.         sprintf(buf, "%s", "[DIRECTORY]"); // print in [DIR] 339.     } else { 340.         off_t size = stat-&gt;st_size; 341.         if(size &lt; 1024){ 342.             sprintf(buf, "%luB", size); 343.         } else if(size &lt; 1024 * 1024){ 344.             sprintf(buf, "%.1fKB", (double)size / 1024); 345.         } else if(size &lt; 1024 * 1024 * 1024){ 346.             sprintf(buf, "%.1fMB", (double)size / 1024 / 1024); 347.         } else { 348.             sprintf(buf, "%.1fGB", (double)size / 1024 / 1024 / 1024); 349.         } 350.     } 351. }</pre>	<p>[참고]</p> <p>directoryhandler( ) 함수에서 파일/디렉토리 목록을 출력할 때 파일의 크기를 출력하는 곳에서 사용되는 함수이다.</p>
<p>337~338 : 만약 파일이 디렉토리(S_ISDIR) 이면 "[DIRECTORY]"가 되도록 한다.</p> <p>339~351 : 디렉토리가 아니라면, 파일의 크기(stat-&gt;st_size)를 검사해서, 1024 보다 작다면, B로 표기, 1024 * 1024보다 작다면 KB로 표기, 이후 MB, GB로 표기하도록 한다.</p>	



## 클라이언트 파일 요청 처리 함수 – servestatic()

```
353. // serve-static
354. void
355. servestatic(int outfd, int infd, http_request *req, size_t total_size){
356.     char buf[256];
357.     if(req->offset > 0){
358.         sprintf(buf, "HTTP/1.1 206 PartialWrWn");
359.         // append string with sprintf : buf + strlen(buf)
360.         sprintf(buf + strlen(buf), "Content-Range: bytes %lu-%lu/%luWrWn",
361.                 (long unsigned int)req->offset,
362.                 (long unsigned int)req->end, (long unsigned int)total_size);
363.     } else {
364.         sprintf(buf, "HTTP/1.1 200 OKWrWnAccept-Ranges: bytesWrWn");
365.     }
366.     sprintf(buf + strlen(buf), "Cache-Control: no-cacheWrWn");
367.
368.     // Content-Length
369.     sprintf(buf + strlen(buf), "Content-Length: %luWrWn",
370.             (long unsigned int)(req->end - req->offset));
371.
```

[참고]  
다운로드 받거나 plaintext  
로 출력되는 페이지를 볼 수  
있다.



```
[2019-12-09 00:44:37] STATUS 127.0.0.1 web server start 8244
[2019-12-09 00:45:22] STATUS 192.168.56.1 . 200
[2019-12-09 00:45:34] STATUS 192.168.56.1 webServer 200
```

356~375 : 클라이언트에게 응답 헤더를 전송하는데 필요한 헤더들을 버퍼에 모두 넣고 write( ) 함수를 통해서 클라이언트에게 응답을 전송한다. Content-Range 또는 Accept-Ranges(동영상MP4 용도), Cache-Control, Content-Length, Content-Type 을 처리한다.

클라이언트 파일 요청 처리 함수 – servestatic()		
372.	// Content-Type	
373.	sprintf(buf + strlen(buf), "Content-Type: %s\r\n\r\n", getfiletype(req->filename));	
374.		
375.	write(outfd, buf, strlen(buf));	
376.	// memory mapping to write body of document(file)	
377.	int filesize = req->end - req->offset;	
378.	char* p = mmap(0, filesize, PROT_READ, MAP_PRIVATE, infd, 0);	
379.	write(outfd, p, filesize);	
380.	munmap(p, filesize);	
381.	}	
377~381 : 헤더들을 이전에 모두 전송하였으니, 이제는 파일의 내용 그대로를 body로 전송하면 된다. 메모리 매핑을 이용해서 파일 내용 그대로 메모리 매핑을 하고 클라이언트의 응답의 바디에 작성하고 전송한다. 이후 메모리 매핑해제를 하였다.		

클라이언트 에러 처리(400, 404) 함수 – clienterror()	
<pre>382. // send error to client 383. void 384. clienterror(int fd, int code, char *msg, char *lmsg){ 385.     char buf[MAXLINE]; 386.     sprintf(buf, "HTTP/1.1 %d %s\r\n", code, msg); 387.     sprintf(buf + strlen(buf), 388.         "Content-Length: %lu\r\n\r\n", (long unsigned int)strlen(lmsg)); 389.     sprintf(buf + strlen(buf), "%s", lmsg); 390.     write(fd, buf, strlen(buf)); 391. } 392.</pre>	
<p>이 함수는 만약 클라이언트가 URI에 맞는 존재하지 않는 파일을 요청하거나(File not found, 404), 일반 파일이나 디렉토리 파일이 아닌 경우(Unknown Error, 400), 클라이언트에게 해당 상황에 맞는 상태코드 및 화면을 출력하도록 전송하는 함수이다.</p>	

<div data-bbox="17 14 1116 64" data-label="Text"> <p>클라이언트 에러 처리(400, 404) 함수 – clienterror()</p> </div> <div data-bbox="17 92 1865 1225" data-label="Code-Block"> <pre> 393. void 394. weblog(int type, char s1[ ], char s2[ ], int n) 395. { 396.     int      log_fd; 397.     char      buf[BUFSIZ], m_time[32]; 398.     time_t    t; 399.     time(&amp;t); 400.     strftime(m_time, sizeof(m_time), // time transform 401.              "%Y-%m-%d %H:%M:%S", localtime(&amp;t)); 402.     if(type == LOG) { 403.         sprintf(buf, "[%s] STATUS %s %s %d\n", m_time, s1, s2, n); 404.     } else if(type == ERROR) { 405.         sprintf(buf, "[%s] ERROR %s %s %d\n", m_time, s1, s2, n); 406.     } 407.     if((log_fd = open("web.log", O_CREAT O_WRONLY O_APPEND, 0644)) &gt;= 0) { 408.         write(log_fd, buf, strlen(buf)); 409.         close(log_fd); 410.     } 411.     if(type == ERROR) exit(-1); 412. }</pre> </div>	
<div data-bbox="17 1275 2522 1382" data-label="Text"> <p>서버 시작과 동시에 관련 모든 로그를 처리하여, 서버 프로그램이 존재하는 디렉토리에 web.log라는 파일을 생성하여 기록하는 함수이다.</p> </div>	

# 사용설명서

# 서버 시작

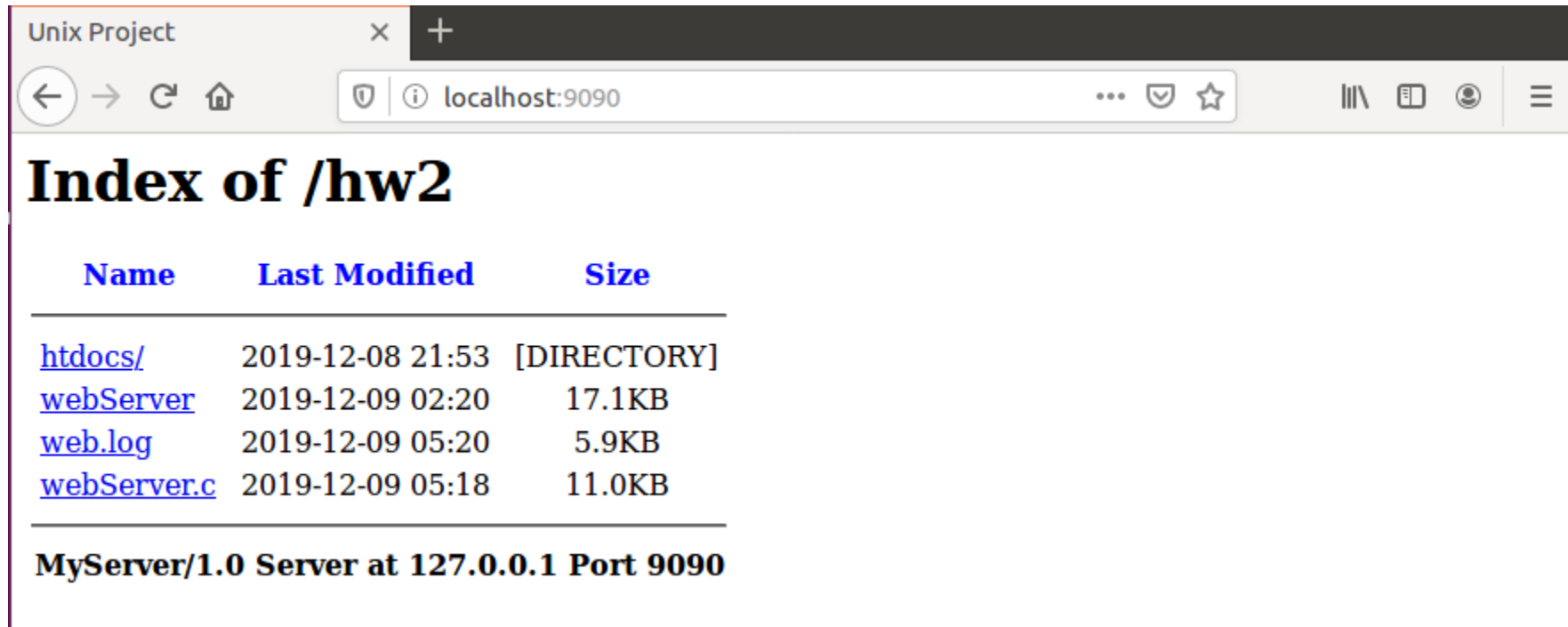
```
pintos@pintos-VirtualBox:~/hw2$ webServer 9090
pintos@pintos-VirtualBox:~/hw2$ ps
  PID TTY          TIME CMD
 8904 pts/4        00:00:00 bash
 9800 pts/4        00:00:00 webServer
 9801 pts/4        00:00:00 ps
pintos@pintos-VirtualBox:~/hw2$
```

- 서버 프로그램명과 포트번호와 함께 실행한다.

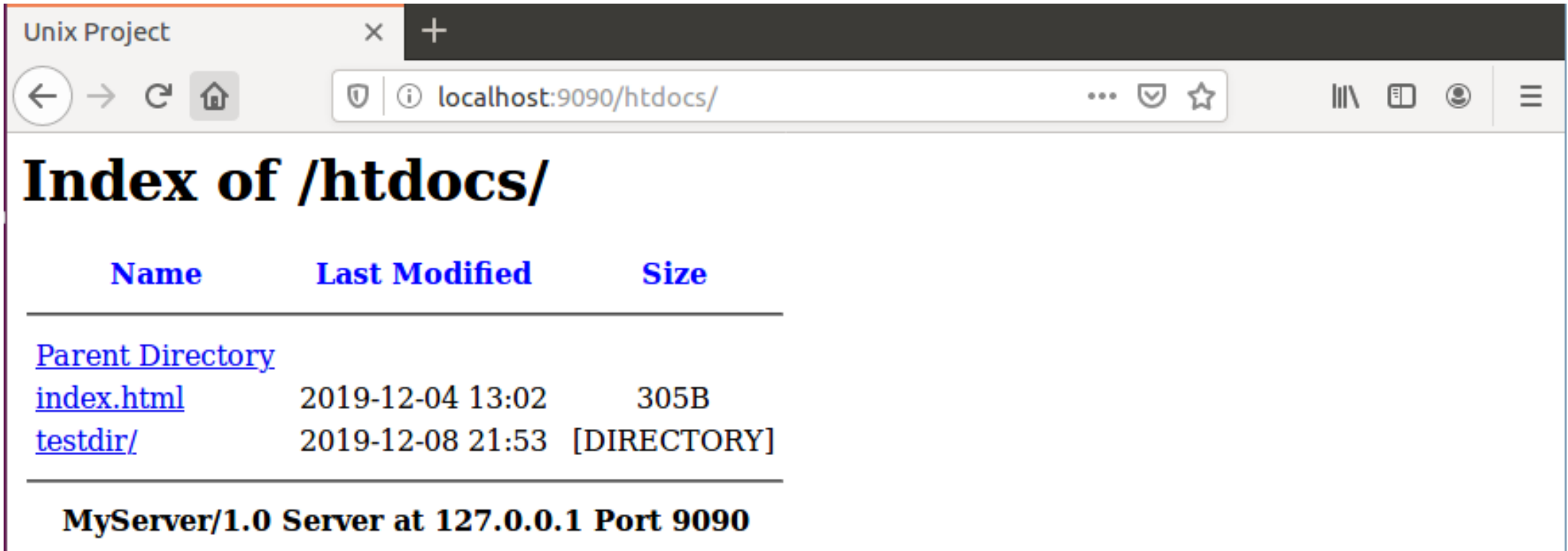
# 클라이언트 웹브라우저로 서버 접속

- 인터넷 브라우저 URI 주소 창에 서버주소:포트번호 입력, 접속

192.168.56.101:9090



# 다른 디렉토리로 이동(htdocs/)



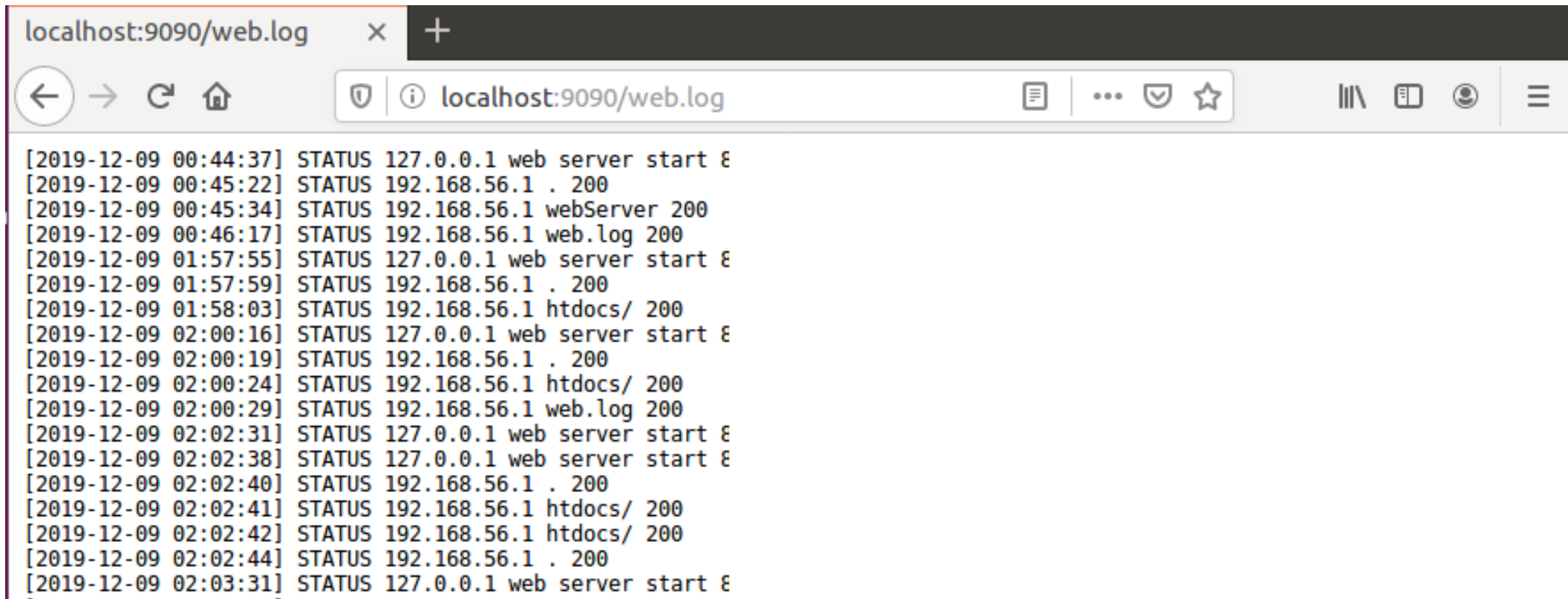
The screenshot shows a web browser window with the title 'Unix Project'. The address bar displays 'localhost:9090/htdocs/'. The main content area shows the 'Index of /htdocs/' page. It features a table with three columns: 'Name', 'Last Modified', and 'Size'. The table lists three items: 'Parent Directory', 'index.html', and 'testdir/'. Below the table, it says 'MyServer/1.0 Server at 127.0.0.1 Port 9090'.

Name	Last Modified	Size
<a href="#">Parent Directory</a>		
<a href="#">index.html</a>	2019-12-04 13:02	305B
<a href="#">testdir/</a>	2019-12-08 21:53	[DIRECTORY]

MyServer/1.0 Server at 127.0.0.1 Port 9090

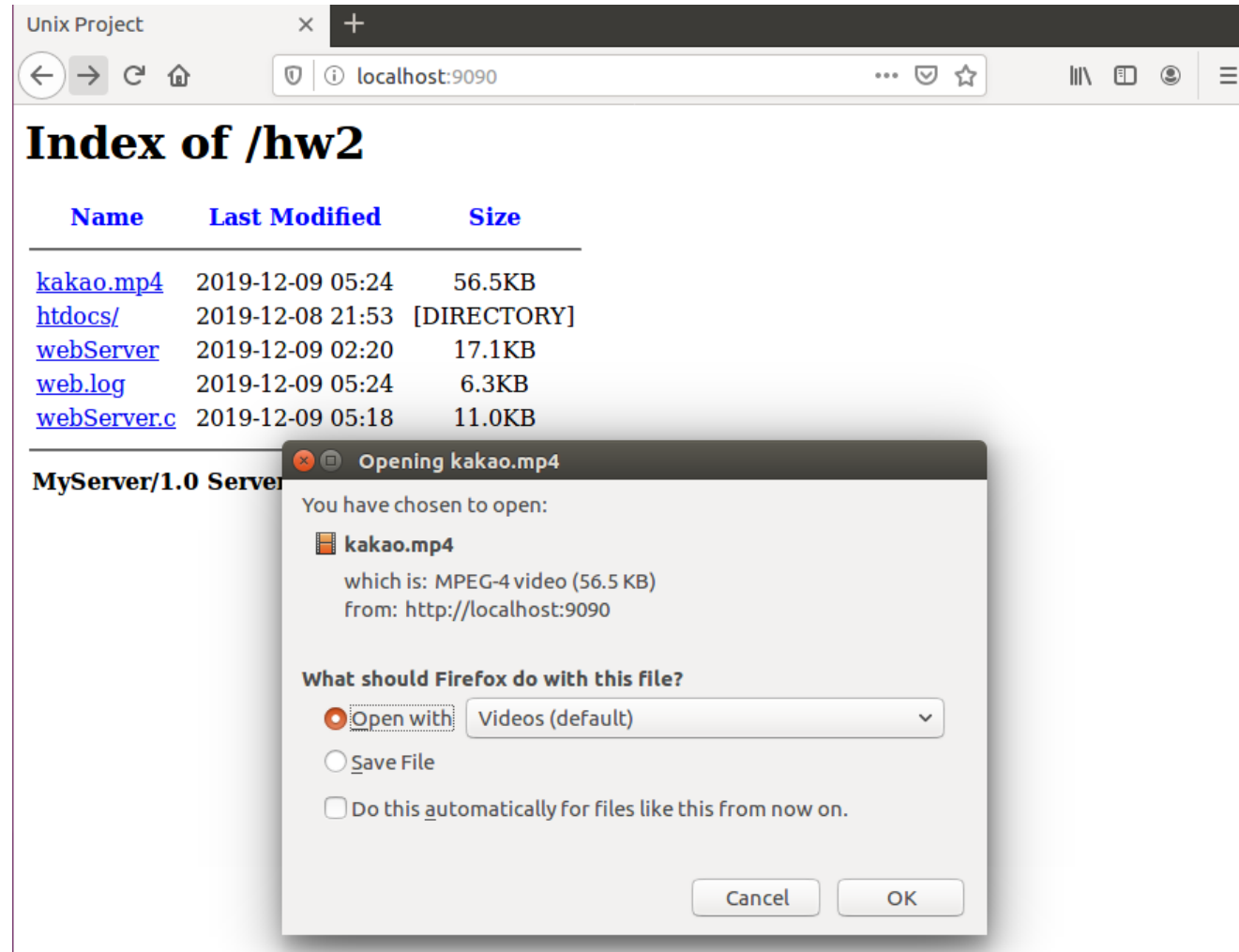


# web.log 파일 실행(text/plain)

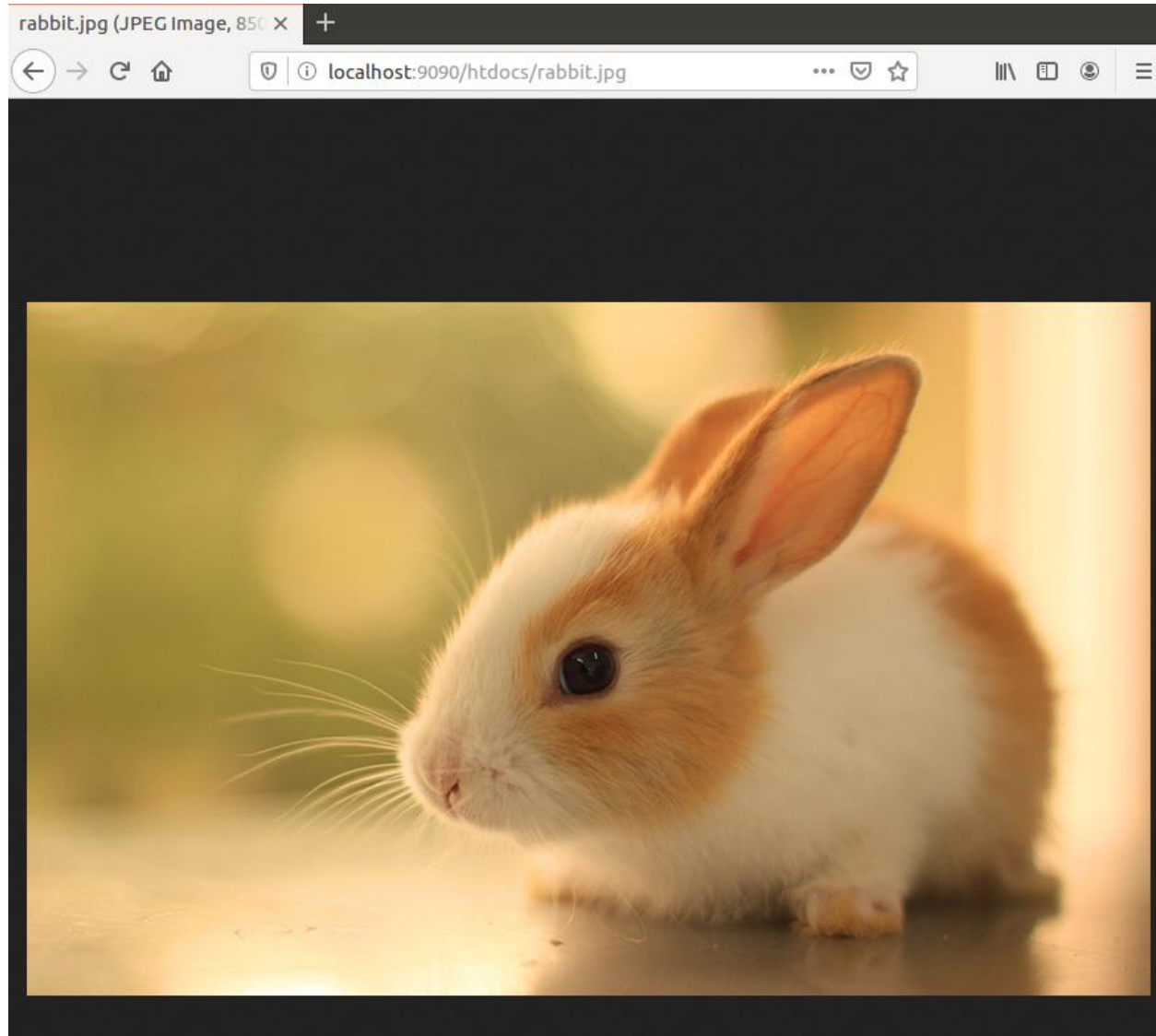
A screenshot of a web browser window showing a log file. The browser's address bar displays 'localhost:9090/web.log'. The page content consists of a series of log entries, each starting with a timestamp in square brackets, followed by the word 'STATUS', and then details about a web server start or a request. The log entries are as follows:

```
[2019-12-09 00:44:37] STATUS 127.0.0.1 web server start 8
[2019-12-09 00:45:22] STATUS 192.168.56.1 . 200
[2019-12-09 00:45:34] STATUS 192.168.56.1 webServer 200
[2019-12-09 00:46:17] STATUS 192.168.56.1 web.log 200
[2019-12-09 01:57:55] STATUS 127.0.0.1 web server start 8
[2019-12-09 01:57:59] STATUS 192.168.56.1 . 200
[2019-12-09 01:58:03] STATUS 192.168.56.1 htdocs/ 200
[2019-12-09 02:00:16] STATUS 127.0.0.1 web server start 8
[2019-12-09 02:00:19] STATUS 192.168.56.1 . 200
[2019-12-09 02:00:24] STATUS 192.168.56.1 htdocs/ 200
[2019-12-09 02:00:29] STATUS 192.168.56.1 web.log 200
[2019-12-09 02:02:31] STATUS 127.0.0.1 web server start 8
[2019-12-09 02:02:38] STATUS 127.0.0.1 web server start 8
[2019-12-09 02:02:40] STATUS 192.168.56.1 . 200
[2019-12-09 02:02:41] STATUS 192.168.56.1 htdocs/ 200
[2019-12-09 02:02:42] STATUS 192.168.56.1 htdocs/ 200
[2019-12-09 02:02:44] STATUS 192.168.56.1 . 200
[2019-12-09 02:03:31] STATUS 127.0.0.1 web server start 8
```

# mp4 파일 다운로드하기



# 이미지 파일 보기



# 미완성 기능

- Name, Last Modified, Size 별로 정렬 기능은 구현하지 못하였음. 원래 클릭하면 정렬되도록 하려고 하였음.

# 서버 종료 시

```
pintos@pintos-VirtualBox:~/hw2$ kill webServer
pintos@pintos-VirtualBox:~/hw2$ ps
  PID TTY          TIME CMD
 8904 pts/4        00:00:00 bash
10035 pts/4        00:00:00 ps
pintos@pintos-VirtualBox:~/hw2$
```

# 어려웠던 점

- HTTP 프로토콜 및 브라우저와 관련해서는 배운 것이 없어서, 직접 알아봐야했는데,
- 파일을 다운로드 하기 위한 또는 파일을 조회하기 위한 HTTP 상태 코드 관련해서 공부가 필요했고,
- 브라우저가 프로그램을 중단하거나 계속 TCP RST이 되거나 소켓이 닫히거나 등의 문제가 발생했었는데 이를 위해서 해결하기 위한 시간이 정말 많이 걸렸음.
- 주로 HTTP request, response 에서의 Header 관련 지식의 부재로 인한 시간 소요가 많았음.
- 원래 이름 순, 최근 수정된 순, 크기 순으로 정렬하려고 하였으나 시간부족으로 구현하지 못하였음.