시스템프로그래밍실습 보고서

Assignment 2-3

과 목 시스템프로그래밍실습

담당교수 이기훈교수님

학 과 컴퓨터정보공학부

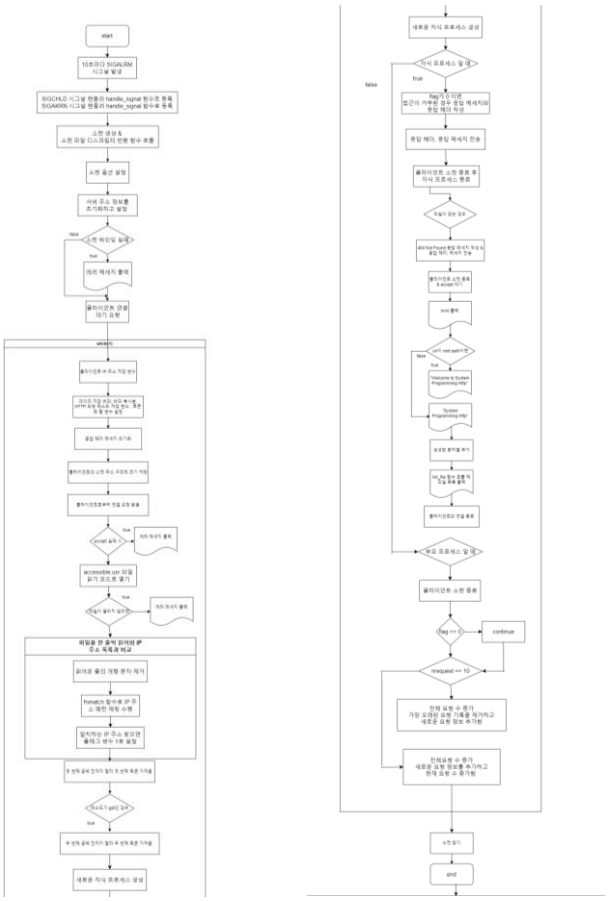
학 번 2021202058

이 름 송채영

1. Introduction

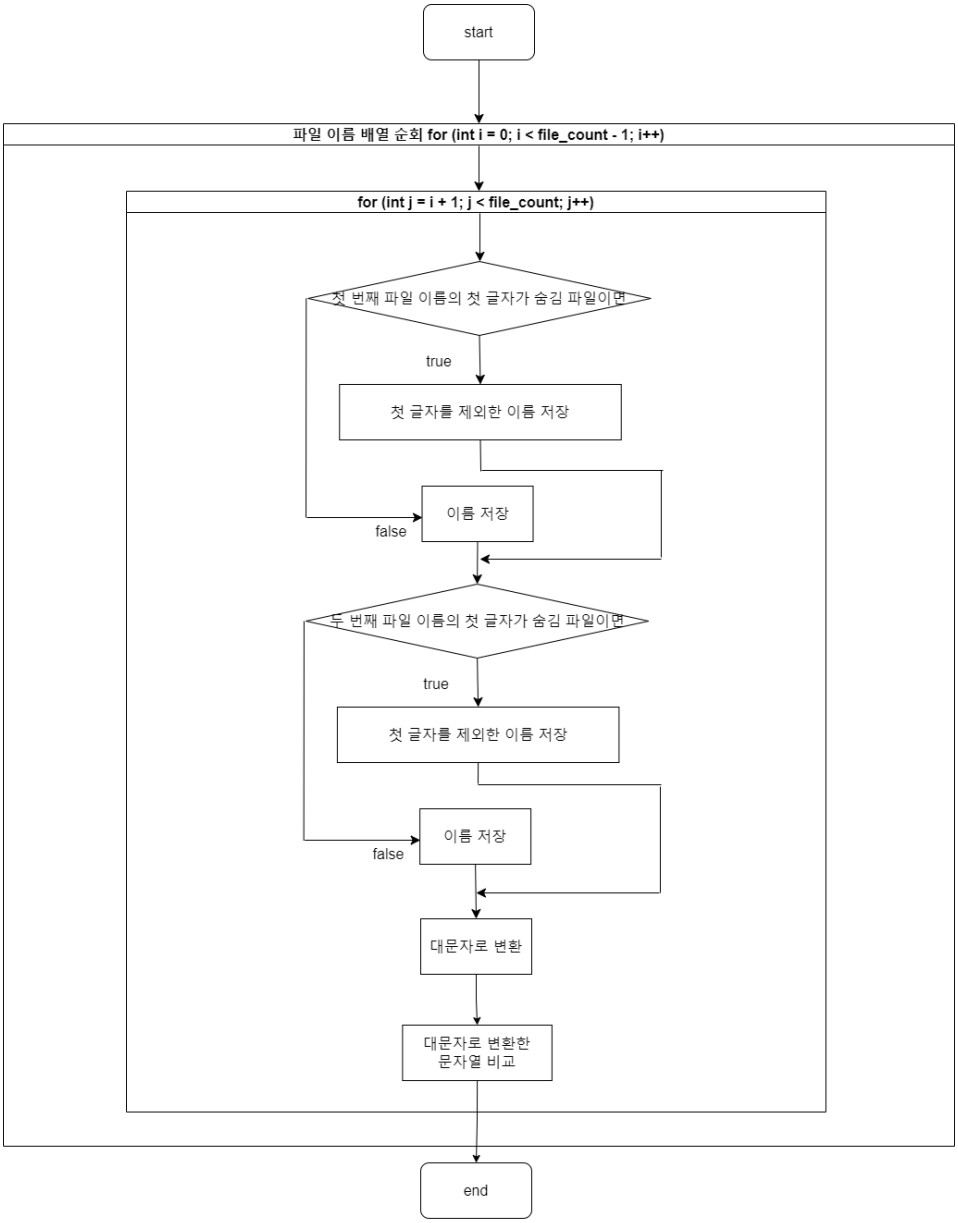
이번 과제는 여러 개의 동시 연결 및 access 제어를 지원하는 웹 서버 프로그램을 구현하는 것이다. 이전 Assignment 2-2, 웹 서버 구현을 기반으로 해 여러 클라이언트 연결을 동시에 처리하는 기능을 추가한다. 서버는 클라이언트 연결 및 연결 끊김에 대한 실시간 정보를 표시하고, 최근 연결 요약을 주기적으로 출력하며, 누적 요청 수를 유지하며, 최근 10 개의 연결 레코드를 기록한다. 추가적으로 fork(), alarm() 함수에 대해 알아본다.

2. Flow chart

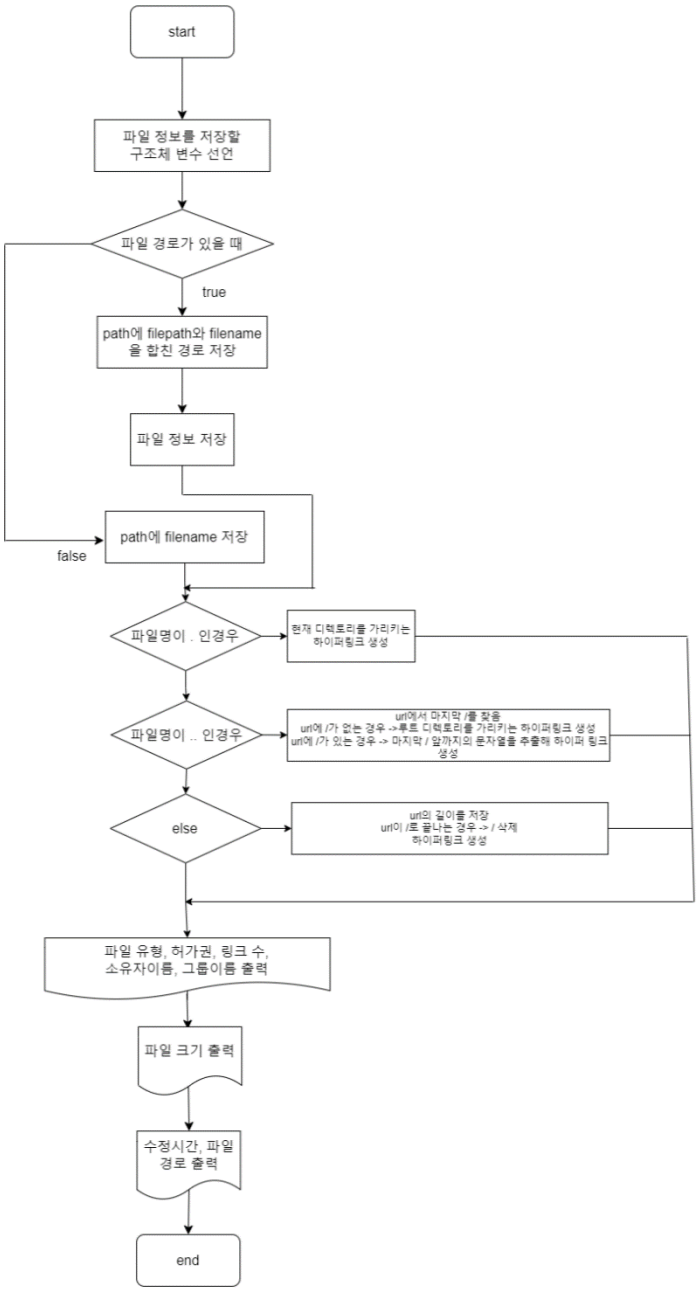


코드의 전체적인 흐름을 flow chart 로 나타내었다. 우선 main 함수에 대해 자세히 설명해보면 다음과 같다. 프로그램이 시작하면, 10 초마다 SIGALRM 시그널을 발생하고 SGICHLD 와 SIGALRM 을 handle_signal 함수로 등록한다. 소켓을 생성하고 소켓 파일 디스크립터 반환 함수를 호출한 후 소켓 옵션을 설정한다. memset 을 사용하여 서버 주소를 초기화 하고 설정해준다. 소켓 바인딩을 실패하면 에러 메세지를 출력해주고, listen() 함수를 사용해 클라이언트의 연결 요청을 대기한다. while 이 1 일 때 클라이언트 IP 주소 저장 변수와 클라이언트로부터 받은 데이터를 저장할 버퍼, 버퍼 복사본, HTTP 요청 메소드 저장 변수, HTTP 요청 메시지 토큰화 할 포인터를 선언한 후 응답 헤더와

메시지를 초기화한다. 클라이언트의 소켓 주소 구조체 크기를 저장한 후 클라이언트로부터 연결 요청을 받으며, accept 를 실패하면 에러 메세지를 출력해준다. accessible.usr 파일을 읽기 모드로 열어 파일을 열지 못했을 경우 오류 메시지를 출력한다. 파일을 한 줄씩 읽어와 IP 주소 목록과 비교한다. 읽어온 줄의 개행 문자레르 제거하고 fnmatch 함수를 사용하여 IP 주소 패턴 매칭을 수행한다. 이때 일치하는 IP 주소를 찾으면 플래그 변수 1 로 설정한다. 첫 번째 공백 전까지 문자열을 잘라서 첫 번째 토큰을 가져오는데 이때 토큰이 없는 경우 무시한다. url 배열을 초기화한 후 메서드가 GET 인 경우 두 번째 공백 전까지 문자열을 잘라서 두 번째 토큰을 가져온다. 새로운 자식 프로세서를 생성하고 자식프로세서 일 때와 부모 프로세서 일 때로 나눈다. 자식 프로세서에서는 flag 가 0 이면 접근이 거부된 경우 응답 메시지와 응답 헤더를 작성하고 응답 헤더와 메시지를 전송한다. 클라이언트 소켓을 종료하고 다음 클라이언트를 요청하고 자식 프로세스를 종료한다. 현재 작업 디렉토리를 cwd 에 저장하고 요청한 파일의 경로를 만들어, 경로가 없는 경우 404 Not Found 응답 메시지를 작성한다. 응답 헤더와 메시지를 전송한 후 클라이언트 소켓을 종료한다. 다시 accept 대기로 돌아간 후 html 출력을 위해 필요한 부분을 구현한다. url 이 root path 인 경우 lflag 를 1 로 설정해주고 "Welcome to System Programming Http"을 출력한다. 하위디렉토리일 경우 aflag 와 lflag 를 1 로 설정해주고 "System Programming Http"를 출력한다. cwd 의 파일 목록을 출력한 후 클라이언트와 연결과 소켓과의 연결을 종료한다. 다음으로 부모 프로세서의 경우 클라이언트 소켓을 종료한 후 flag 가 0 이면 다음 클라이언트 요청 처리를 위해 반복문의 처음으로 이동한다. nrequest 가 10 일 경우 전체 요청 수를 증가하고 가장 오래된 요청 기록을 제거한 후 새로운 요청 정보를 추가한다. else 일 경우 전체 요청 수를 증가하고 새로운 요청 정보를 추가하고 현재 요청 수를 증가한다. 소켓을 닫고 프로그램을 종료한다.

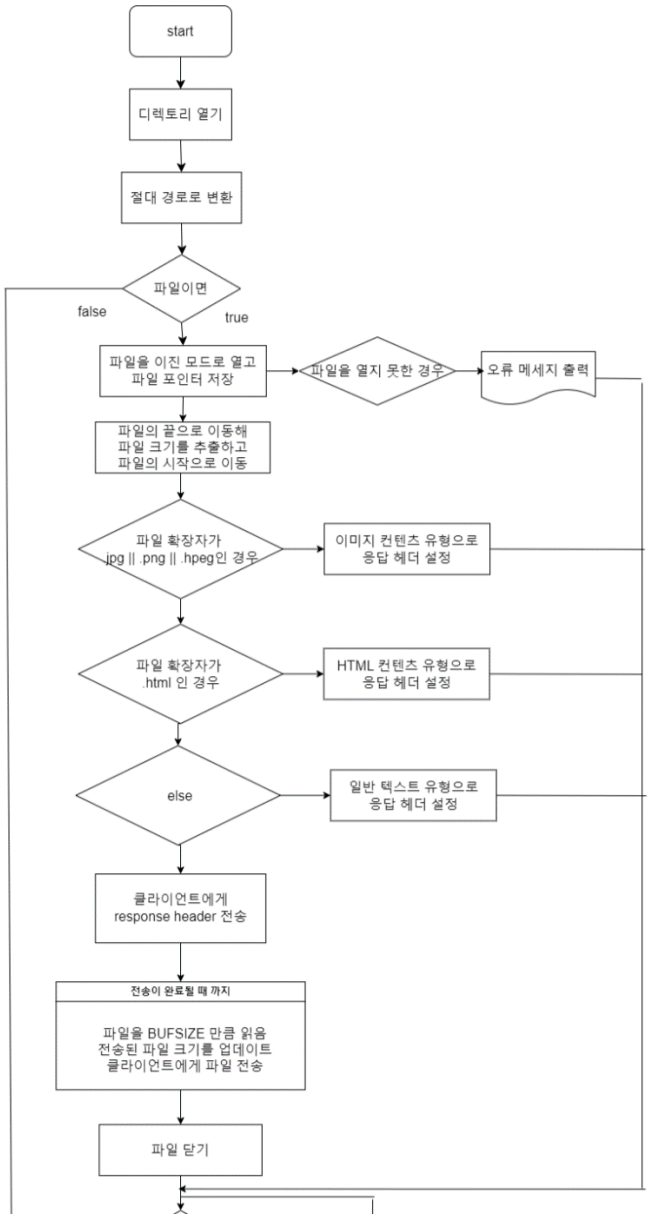


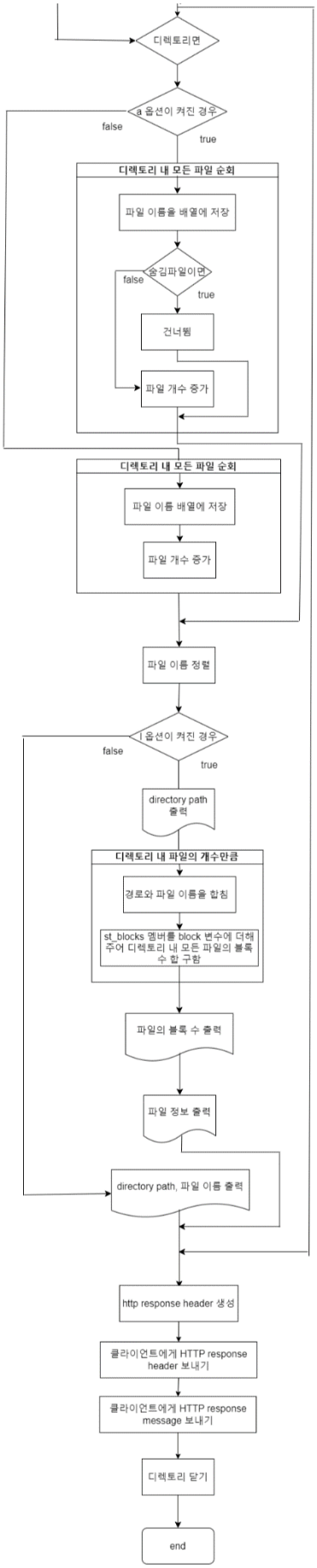
다음은 sort 함수의 flow chart 이다. 우선 파일의 개수만큼 반복한다. 첫 번째 파일이 숨김 파일일 경우, 파일의 이름을 길이만큼 반복하고 첫 글자를 제외한 이름을 저장한다. 숨김파일이 아닐 경우 이름을 저장해준다. 두 번째 파일도 동일하게 진행한다. 먼저 대문자로 변환해준 후 대문자로 변환한 문자열을 비교한다.



다음은 print_file_info, 파일 정보를 출력하는 함수의 flow chart 이다. 파일 정보를 읽어 구조체 변수인 filestat 에 저장한다. 파일 경로가 있고, filepath 와 filename 이 같지 않으면 파일 경로가 존재하는 경우이므로 경로와 파일 이름을 합치며, 파일 경로가 없는 경우 파일이름을 정보로 읽는다. 파일명이 ., .., else 인 경우로 나누어 하이퍼링크를 준다. 먼저 .인 경우 현재 디렉토리를 가리키는 하이퍼링크를 생성하며, ..일 경우 url 에서 마지막 /을 찾고 /이 있으면 루트 디렉토리를 가리키는 하이퍼링크를 생성하고, 없을 경우 마지막 / 앞까지의 문자열을 추출한 후 하이퍼링크를 생성해준다. else 인 경우 url 의 길이를 저장한 후 url 이 /로 끝나면 /을 삭제한 후 하이퍼링크를 생성하도록

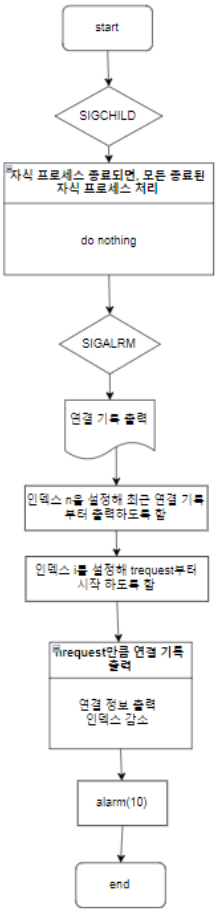
하였다. 파일 유형, 허가권, 링크 수, 소유자,호출해주며, 파일 크기, 수정 시간, 파일 경로를 출력한 후 프로그램을 종료한다.




디렉토리면
a 옵션이 켜진 경우
false
true
디렉토리 내 모든 파일 순회
파일 이름을 배열에 저장
숨김파일이면
false
true
건너뜀
파일 개수 증가
디렉토리 내 모든 파일 순회
파일 이름 배열에 저장
파일 개수 증가
파일 이름 정렬
l 옵션이 켜진 경우
false
true
directory path 출력
디렉토리 내 파일의 개수만큼
경로와 파일 이름을 합침
st_blocks 멤버를 block 변수에 더해 주어 디렉토리 내 모든 파일의 블록 수 합 구함
파일의 블록 수 출력
파일 정보 출력
directory path, 파일 이름 출력
http response header 생성
클라이언트에게 HTTP response header 보내기
클라이언트에게 HTTP response message 보내기
디렉토리 닫기
end

다음은 list files 함수의 flow chart 이다. 파일을 열어준 후 절대경로로 변환한다. 연 파일이 파일일때와 디렉토리 일 때로 나눈다. 파일일 때, 파일을 이진 모드로 열고 파일 포인터를 file 변수에 저장한다. 파일을 열지 못했을 경우 오류 메세지를 출력한다. 파일의 끝으로 이동해 파일 크기를 추출한 후 파일의 시작으로 이동한다. 파일 확장자에 따라 content-type 을 설정하였는데, .jpg, .png, .jpeg 일 때 이미지 컨텐츠 유형으로 응답 헤더를 설정한다. 파일 확장명이 .html 인 경우 HTML 컨텐츠 유형으로 응답 헤더를 설정한다. else 일 경우 일반 텍스트 유형으로 응답 헤더를 설정하고 클라이언트에게 response header 를 전송하였다. 전송이 완료될 때 까지 반복하며 파일을 BUFSIZE 만큼 읽어서 전송된 파일 크기를 업데이트 하고 클라이언트에게 파일을 전송한다. 파일을 닫는다. 디렉토리일 경우 a 옵션이 꺼져 있을 때와 켜져 있을 때, l 옵션이 켜져 있을 때로 나눈다. a 옵션이 꺼져 있으면 디렉토리 내 모든 파일을 순회하며 파일이름을 배열에 저장한다. .과 ..은 건너뛰고 그 외의 파일은 파일 개수를 증가시킨다. a 옵션이 켜져 있을 경우, 디렉토리 내 모든 파일을 파일 이름을 배열에 저장하고 파일 개수를 증가시킨다. 파일 이름 정렬 함수를 호출해 정렬한다. l 옵션이 켜져 있으면, 디렉토리 경로와 total 을 출력해주고 파일 정보 출력 함수를 호출해 출력한다. HTTP response header 를 생성해 작성해 준 후, 클라이언트에게 HTTP response header 와 message 를 보낸 후 디렉토리를 닫고 프로그램을 종료한다.

위 함수에서 html 에 directory path 와 total 그리고 filename 을 table 형태로 출력해주기 위해 html 코드로 출력하는 부분을 추가했다. 또한 html_ls.html 파일일 경우 건너뛰는 코드를 추가해 html_ls.html 파일이 출력되지 않도록 하였다. flow chart 의 흐름은 기존과 변함이 없다.



다음으로 handle_signal 함수에 대한 flow chart 이다. 우선 SIGCHLD 시그널 핸들러일 경우 자식 프로세서가 종료되면, 모든 종료된 자식 프로세스를 처리한다. while 문 안에서는 아무 것도 수행하지 않는다. signal 이 SIGALRM 일 경우 연결 기록을 출력한 후 최근 연결 기록부터 출력하는 인덱스 n 과 trequest 부터 시작하는 인덱스 i 를 변수로 선언한 후 while (i > (trequest - nrequest)) while 문 안에서 연결정보를 출력한다. 이후 인덱스를 감소시키고 연결 기록의 이전 항목으로 이동시킨 후 while 문을 나와 10 초마다 SIGALRM 시그널을 발생시킨다.

3. Pseudo Code

```
선택정렬() //sort
{
	for(파일 수만큼 파일 이름 배열 순회)
	{
		if(첫번째 파일 첫 글자가 숨김 파일이면)
		{
			filename[i]의 길이만큼 반복하고 첫 글자 제외하고 저장
		}
		else(숨김파일이 아니면)
		{
			이름 저장
		}
		if(두번째 파일 첫 글자가 숨길 파일이면)
		{
			filename[i]의 길이만큼 반복하고 첫 글자 제외하고 저장
		}
		else(숨김파일이 아니면)
		{
			이름 저장
		}
		hidden_file 1 대문자로 변환
		hidden_file 2 대문자로 변환
		대문자로 변환한 문자열 비교
}
```

sort 함수의 pseudo code 이다.

```
파일 정보 출력() //print_file_info
{
	파일 정보 읽어 구조체 변수 filestat에 저장
	파일 권한 정보 저장할 변수 선언
	if(파일 경로가 있으면)
	{
		파일 경로와 파일 이름을 합치고 stat함수로 파일 정보 읽음
	}
	else
	{
		파일 이름으로 정보 읽음
	}
	if(파일명이 .인 경우) 현재 디렉토리를 가리키는 하이퍼링크 생성
	else if(파일명이 ..인 경우)
	{
		url에서 마지막 /를 찾음
		url에 /가 없는 경우 -> 루트 디렉토리를 가리키는 하이퍼링크 생성
		url에 /가 있는 경우 -> 마지막 / 앞까지의 문자열을 추출 후 하이퍼링크 생성
	}
	else
	{
		url의 길이 저장
		url이 /로 끝나는 경우 -> /를 삭제
		하이퍼링크 생성
	}
	// 파일 유형, 허가권, 링크 수 , 소유자, 그룹, 출력
	if(파일 유형이 디렉토리면) permission 배열의 0번째는 d
	else if(파일 유형이 링크면) permission 배열의 0번째는 l
	else permission 배열의 0번째는 -
	파일 허가권 정보 설정

	if(디렉토리면) 파란색으로 출력
	else if(link 파일이면) 초록색으로 출력
	else 빨간색으로 출력

	파일 크기 출력
	수정시간, 파일 경로 출력
}
```

print_file_info 함수의 pseudo code 이다.

```
파일 목록 출력() //list_files
{
	디렉토리 열기
	절대경로로 변환
	if(파일이면)
	{
		파일을 이진 모드로 열고 파일 포인터 저장
		if(파일 열지 못하면) 에러메세지 출력
		파일 끝으로 이동해 파일 크기 추출한 후 파일 시작으로 이동

		파일 확장자에 따라 content-type 설정
		if (jpg, png, jpeg)
		{
			이미지 콘텐츠 유형으로 응답 헤더 설정
		}
		else if(.html)
		{
			HTML 컨텐츠 유형으로 응답 헤더 설정
		}
		else
		{
			일반 텍스트 유형으로 응답 헤더 설정
		}
		클라이언트에게 response header를 전송
		while(전송이 완료될 때까지)
		{
			파일을 BUFSIZE 만큼 읽음
			전송된 파일 크기를 업데이트
			클라이언트에게 파일을 전송
		}
		파일 닫기
	}
	else //디렉토리면
	{
		if(a 옵션이 꺼져있는 경우)
		{
			while(디렉토리 내 모든 파일 순회)
			{
				html_ls.html 파일은 건너뜀
				숨김파일이면 넘어감
				아니면 파일 개수 증가
			}
		}
		else
		{
			while(디렉토리 내 모든 파일 순회)
			{
				html_ls.html 파일은 건너뜀
				파일 이름 배열에 저장
			}
		}
		파일 이름 정렬
		if(l 옵션이 켜져있는 경우)
		{
			directory path, total 출력
			HTML 파일에 테이블 태그 출력
			HTML 파일에 테이블 헤더 삽입
			파일 정보 출력
		}
		HTTP response header 생성
		클라이언트에게 response header, message 보내기
	}
	디렉토리 닫기
}
```

list_files 함수의 pseudo code 이다.

```
handle_signal
{
	if(signal == SIGCHLD)
	{
		자식 프로세서가 종료되면, 모든 종료된 자식 프로세서 처리
	}
	if(signal == SIGALRM)
	{
		연결 기록 출력
		while(i > (trequest - nrequest))
		{
			연결 정보 출력
			인덱스 감소
			연결 기록 이전 항목으로 이동
		}
		10초마다 SIGALM 시그널 발생
	}
}
```

handle_signal 함수의 pseudo code 이다.

```
main(argc, argv)
{
	10초마다 SIGALRM 시그널 발생
	SIGCHLD 시그널 핸들러 handle_signal 함수로 등록
	SIGAKRN 시그널 핸들러 handle_signal 함수로 등록

	소켓을 생성하고 소켓 파일 디스크립터 반환 함수를 호출함

	소켓 옵션을 설정함

	memset을 사용해 서버 주소 정보를 초기화하고 설정함

	if(소켓 바인딩 실패) 오류메세지 출력
	클라이언트 연결 요청 대기

	while(1)
	{
		응답 헤더, 메세지 초기화
		클라이언트로부터 연결 요청 받음
		if(accept 실패 시) 에러 메세지 출력

		accessible.usr 파일을 읽기 모드로 열기
		if(파일을 열지 못하면) 오류메세지 출력

		while(파일에서 한 줄씩 읽어와 IP 주소 목록과 비교)
		{
			읽어온 줄의 개행 문자 제거
			fnmatch 함수 사용해서 IP 주소 패턴 매칭 수행
				일치하는 IP 주소 찾으면 플래그 변수 1로 설정
				break;
		}
		파일 닫기

		첫번째 공백 전까지 문자열 잘라서 첫 번째 토큰 가져옴
		if(토큰이 없는 경우) continue
		url 배열 초기화
		if(method가 GET인 경우)
		{
			두번째 공백 전까지 문자열 잘라서 두 번째 토큰 가져옴
		}
		현재 작업 디렉토리 cwd변수에 저장

		새로운 자식 프로세스 생성
```

```
                if(자식 프로세서일 때)
                {
                        접근이 거부된 경우 응답 메세지와 헤더 작성
                        응답 헤더와 메세지 전송
                        클라이언트 소켓 종료
                        continue
                        자식 프로세스 종료

                        if(파일이 없는 경우)
                        {
                                404 Not Found 응답 메세지 작성
                                응답 헤더, 메세지 전송
                                클라이언트 소켓 종료
                                accpet대기로 돌아감
                                continue
                        }
                        if(url이 root path)
                        {
                                lflag = 1
                                "Welcome to System Programming Http" 태그
                        }
                        else //하위 디렉토리
                        {
                                aflag =1, lfalg = 1
                                "System Programming Http"
                        }
                        list_files 함수 호출해 cwd의 파일 목록 출력
                        클라이언트와 연결 종료
                        소켓 연결 종료
                        자식 프로세스 종료
                }
                else // 부모 프로세스 일 때
                {
                        클라이언트 소켓 종료
                        if(flag == 0) continue
                        if(nrequest == 10)
                        {
                                전체 요청 수 증가
                                가장 오래된 요청 기록 제거, 새로운 요청 정보 추가
                        }
                        else
                        {
                                전체 요청 수 증가
                                새로운 요청 정보를 추가
                                현재 요청 수 증가
                        }
                }
        }
        소켓 닫기
}
```

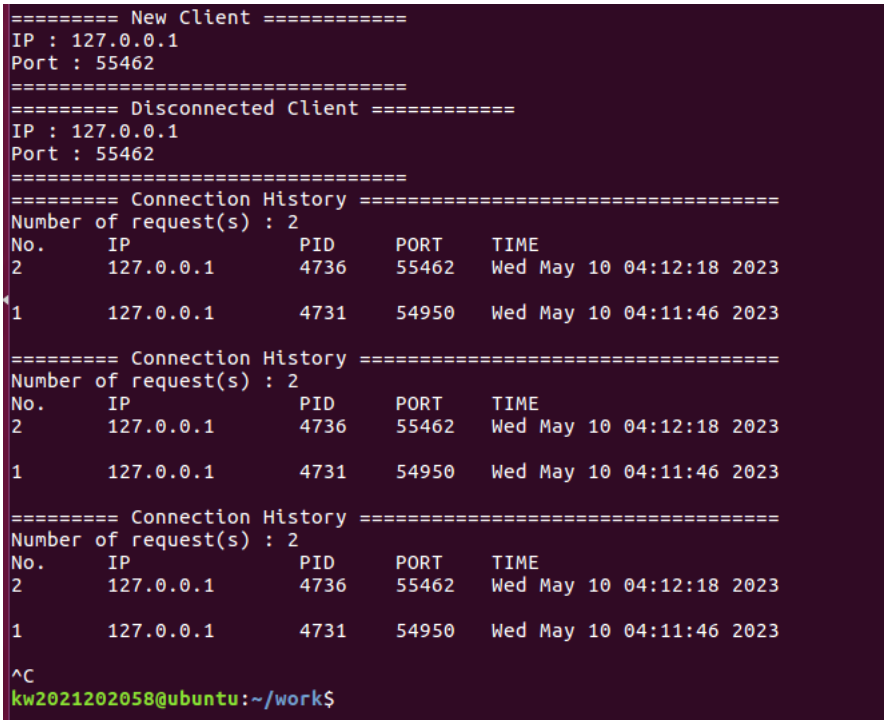
main 함수의 pseudo code 이다.

4. 결과화면

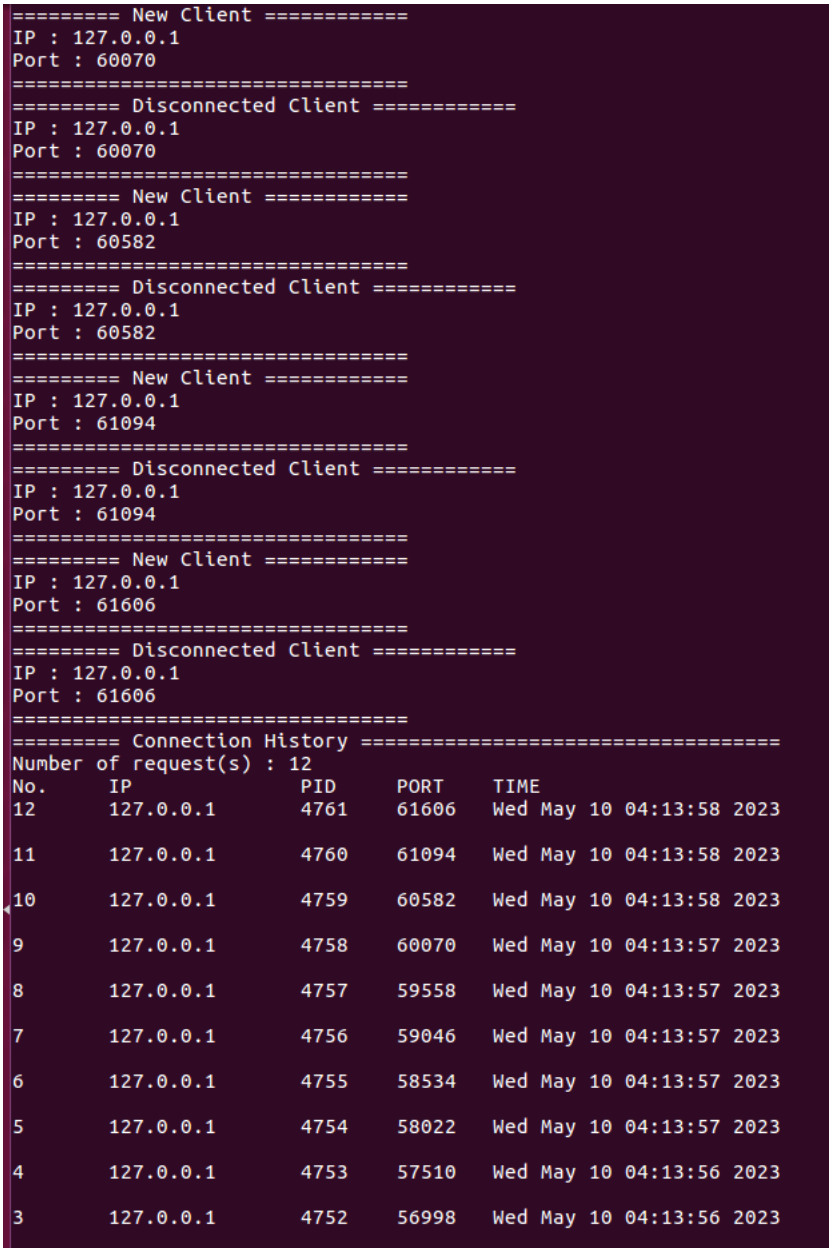
```
kw2021202058@ubuntu:~/work$ ./adv_server
========== New Client ============
IP : 127.0.0.1
Port : 38566
==================================
========== Disconnected Client ============
IP : 127.0.0.1
Port : 38566
==================================
========== New Client ============
IP : 127.0.0.1
Port : 40102
==================================
========== Disconnected Client ============
IP : 127.0.0.1
Port : 40102
==================================
========== Connection History ====================================
Number of request(s) : 2
No.     IP              PID     PORT    TIME
2       127.0.0.1       4613    40102   Wed May 10 04:09:02 2023

1       127.0.0.1       4608    38566   Wed May 10 04:08:56 2023
```

클라이언트와 접속이 연결되거나 해제될 때 마다 클라이언트의 정보를 출력한 결과화면이다. NO.은 해당 클라이언트가 몇 번째로 서버에 연결된 것인지를 나타내므로

역순으로 출력해주었다. 즉 서버 실행 후 몇 번째 request 인지를 나타냈다. 또한 최신 시간 순서대로 정렬되어 잘 출력되는 것을 알 수 있다.



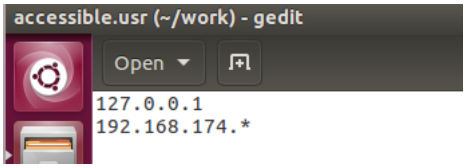
10 초에 한 번씩 연결되었던 클라이언트의 정보를 일괄 출력하는 결과화면으로 10 초마다 잘 출력이 되는 것을 확인할 수 있다.



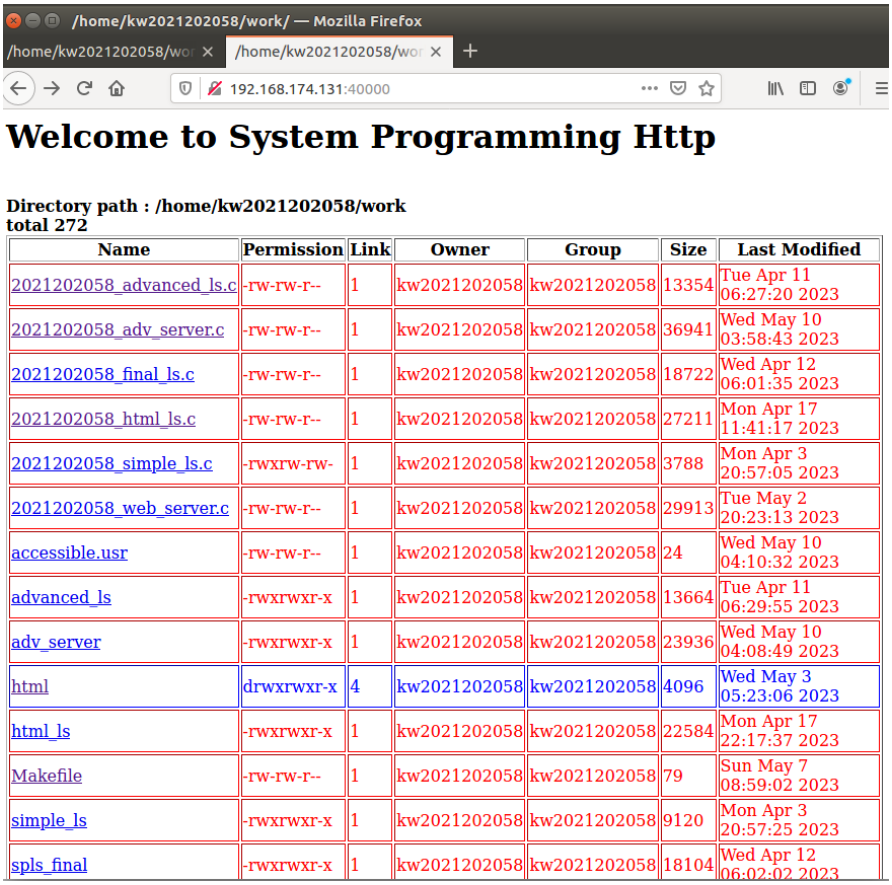
다음으로 최대 10 개의 최근 기록만 출력하는 결과화면이다. 12 개를 입력했기 때문에 12 부터 10 개를 잘라 3 까지만 출력하는 것을 볼 수 있다. 또한 최신 시간 순서대로 정렬되어 출력이 잘 되는 것을 확인할 수 있다.

```
Port : 61606
================================
========= Connection History ==================================
Number of request(s) : 12
No.     IP              PID     PORT    TIME
12      127.0.0.1       4761    61606   Wed May 10 04:13:58 2023

11      127.0.0.1       4760    61094   Wed May 10 04:13:58 2023

10      127.0.0.1       4759    60582   Wed May 10 04:13:58 2023

9       127.0.0.1       4758    60070   Wed May 10 04:13:57 2023

8       127.0.0.1       4757    59558   Wed May 10 04:13:57 2023

7       127.0.0.1       4756    59046   Wed May 10 04:13:57 2023

6       127.0.0.1       4755    58534   Wed May 10 04:13:57 2023

5       127.0.0.1       4754    58022   Wed May 10 04:13:57 2023

4       127.0.0.1       4753    57510   Wed May 10 04:13:56 2023

3       127.0.0.1       4752    56998   Wed May 10 04:13:56 2023

========== New Client ============
IP : 127.0.0.1
Port : 62118
================================
========== Disconnected Client ============
IP : 127.0.0.1
Port : 62118
================================
========= Connection History ==================================
Number of request(s) : 13
No.     IP              PID     PORT    TIME
13      127.0.0.1       4763    62118   Wed May 10 04:14:03 2023

12      127.0.0.1       4761    61606   Wed May 10 04:13:58 2023

11      127.0.0.1       4760    61094   Wed May 10 04:13:58 2023

10      127.0.0.1       4759    60582   Wed May 10 04:13:58 2023

9       127.0.0.1       4758    60070   Wed May 10 04:13:57 2023

8       127.0.0.1       4757    59558   Wed May 10 04:13:57 2023

7       127.0.0.1       4756    59046   Wed May 10 04:13:57 2023

6       127.0.0.1       4755    58534   Wed May 10 04:13:57 2023

5       127.0.0.1       4754    58022   Wed May 10 04:13:57 2023

4       127.0.0.1       4753    57510   Wed May 10 04:13:56 2023

^C
kw2021202058@ubuntu:~/work$
```

하나가 더 늘어났을 때 13 으로 늘어났으며, 13 부터 10 개를 잘라 4 까지만 출력하는 것을 확인할 수 있다. 역시 최신 시간 순서대로 정렬하고 있음을 확인할 수 있다.



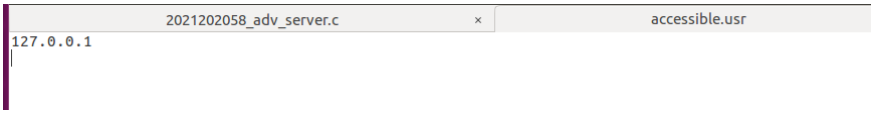
접근제어와 관련된 결과화면이다. 접근가능한 클라이언트의 목록을 파일로 유지한 accessible.usr 파일이다. *가 들어가 있을 경우도 처리하였다.



미리 허용한 IP 인 192.168.174.131:40000 으로 접근했을 때 결과가 잘 나오는 것을 확인할 수 있다.

```
kw2021202058@ubuntu:~/work$ ./adv_server
========= New Client ============
IP : 192.168.174.131
Port : 24794
================================
========= Disconnected Client ============
IP : 192.168.174.131
Port : 24794
================================
========= New Client ============
IP : 192.168.174.131
Port : 25306
================================
========= New Client ============
IP : 192.168.174.131
Port : 25818
================================
========= Disconnected Client ============
IP : 192.168.174.131
Port : 25818
================================
========= New Client ============
IP : 192.168.174.131
Port : 26330
================================
========= Connection History ==================================
Number of request(s) : 3
No.     IP              PID     PORT    TIME
3       192.168.174.131 4812    26330   Wed May 10 04:17:23 2023

2       192.168.174.131 4810    25306   Wed May 10 04:17:20 2023

1       192.168.174.131 4809    24794   Wed May 10 04:17:19 2023

========= New Client ============
IP : 192.168.174.131
Port : 26842
================================
========= Disconnected Client ============
IP : 192.168.174.131
Port : 26842
================================
========= New Client ============
IP : 192.168.174.131
Port : 27354
================================
========= Connection History ==================================
Number of request(s) : 3
No.     IP              PID     PORT    TIME
3       192.168.174.131 4812    26330   Wed May 10 04:17:23 2023

2       192.168.174.131 4810    25306   Wed May 10 04:17:20 2023

1       192.168.174.131 4809    24794   Wed May 10 04:17:19 2023

^C
kw2021202058@ubuntu:~/work$
```

192.168.174.131:40000 으로 접근했을 때도 클라이언트의 정보가 잘 출력되며 위에서 설명한 조건이 모두 동일하게 잘 출력되는 것을 확인할 수 있다.



IP 를 허용하지 않았을 경우를 확인하기 위한 사진이다.

허가되지 않은 IP 가 웹 서버에 접속할 경우 에러메세지가 포함된 페이지를 출력하는 결과화면이다. 과제에서 요구하는 문자를 모두 출력했다.

5. 고찰

우선 이번 과제를 수행하면서 fork 와 alrm 함수에 대해 이해하고 적용하는 것이 가장 어려웠다. 시스템프로그래밍 수업을 통해 개념에 대해서는 이해가 되었지만, 그 개념을 코드의 면에서 어떻게 접근해서 사용하고 구현해야 할지 감이 잘 잡히지 실습 코드를 하나하나 분석하고 이해하려고 했던 것 같다. 과제 ppt 2p 에 사진 상에서는 NO.의 출력이 1 부터 되어있지만, 글에서는 서버 실행 후 몇 번째 request 를 나타내는지를 출력하도록 되어있다. 사진과 글의 내용이 달라 어느 초점에 맞춰야 할지 고민하였지만, 코드에서 원하는 것은, 예를 들어 12 개를 받았을 경우 12 부터 10 개가 잘린 것을 확인하는 것이라 생각했기에 후자의 방식으로 구현하였다. 이 부분을 구현하면서, 변수로 선언했던 trequest 와 nrequest 를 적절하게 활용해서 구현하였기에 seg 오류도 뜨고 좀 더 복잡했던 것 같다. main 에서는 기존의 코드를 조금 수정하여 alarm 부분을 넣어주고, pid 를 child 일때와 부모일 때로 나누어 각각의 조건문 안에서 이전 코드를 활용해서 적절하게 구현해주었다. 이번 과제에서는 구글링을 통해 검색해보고 이해하며 코드를 구현하려고 했던 것 같다. 또한 chrome 에서 192.168.174.131:40000 으로 접근했을 때 처음에는 잘 실행이 되었지만 두 번째부터는 실행이 오래 걸렸으며 약 3 분 뒤 결과가 출력되었다. port 번호를 50000 으로 설정한 후 다시 실행하였을 때는 몇 번을 해도 잘 수행이 되었다. 해당 port 가 처리해야 할 일이 많아서 시간이 오래 걸리는 것이라는 예측을 해보았지만, 이러한 원인에 대해 더 자세히 알고 싶다.

6. Reference

시스템프로그래밍실습 강의자료