2019. 6. 4.

Assignment # 4-2

금요일 - 이성원 교수님
2015722087 컴퓨터정보공학부
김민철

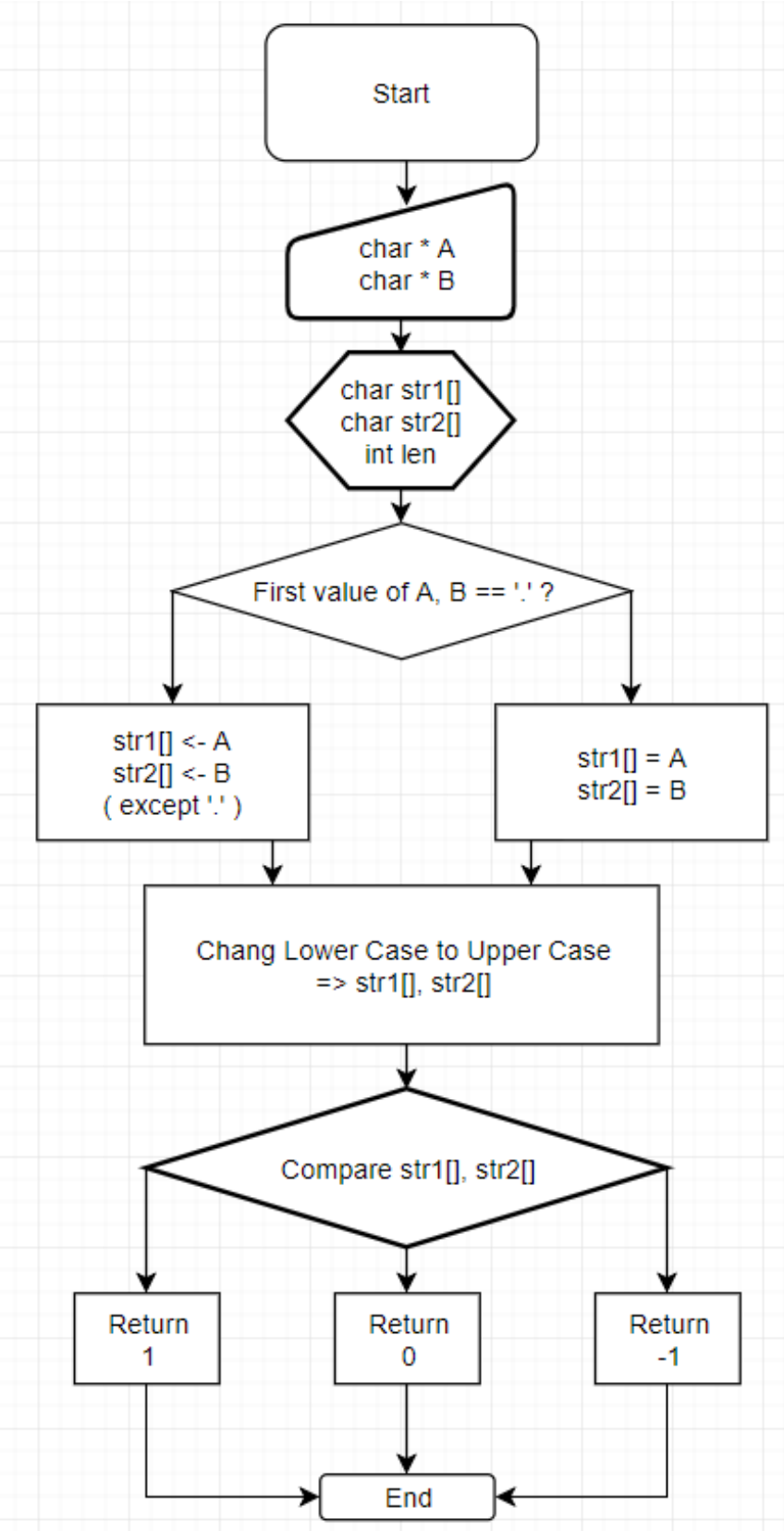
Process_pool_management web-server 구현

Introduction

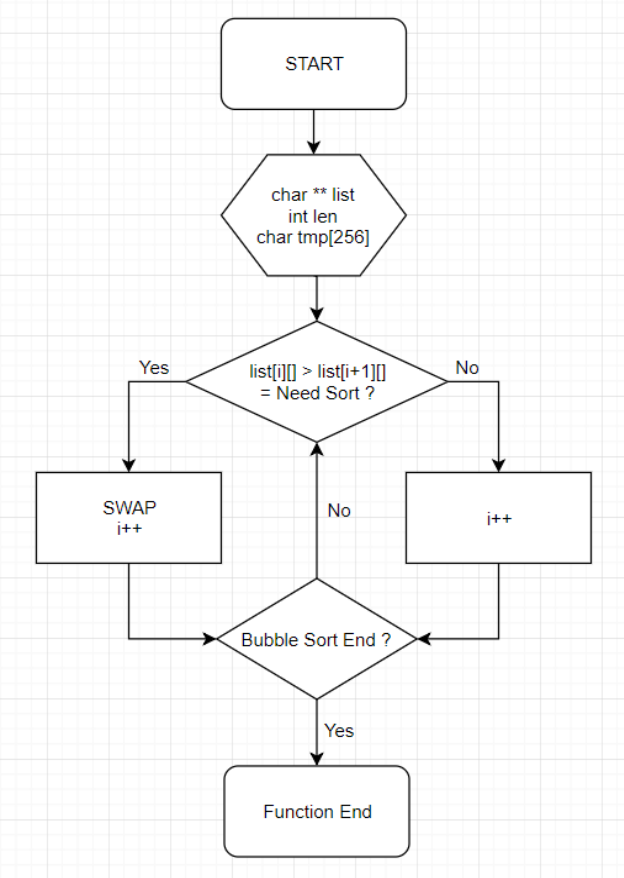
이번 과제에서는 4-1 과제에서 구현하였던 Pre forked web-server 에서, 모든 자식 프로세스에서의 연결 기록을 공유 메모리에 있는 하나의 데이터로 모아, 가장 최신의 기록 10 개 까지 출력하게하며, 공유 메모리에 접근 할때는 스레드를 이용하고, 동기화를 위하여 mutex_lock, unlock 을 이용하고, Idle 프로세스의 개수를 조절하는 기능을 추가한다.

Flow Chart

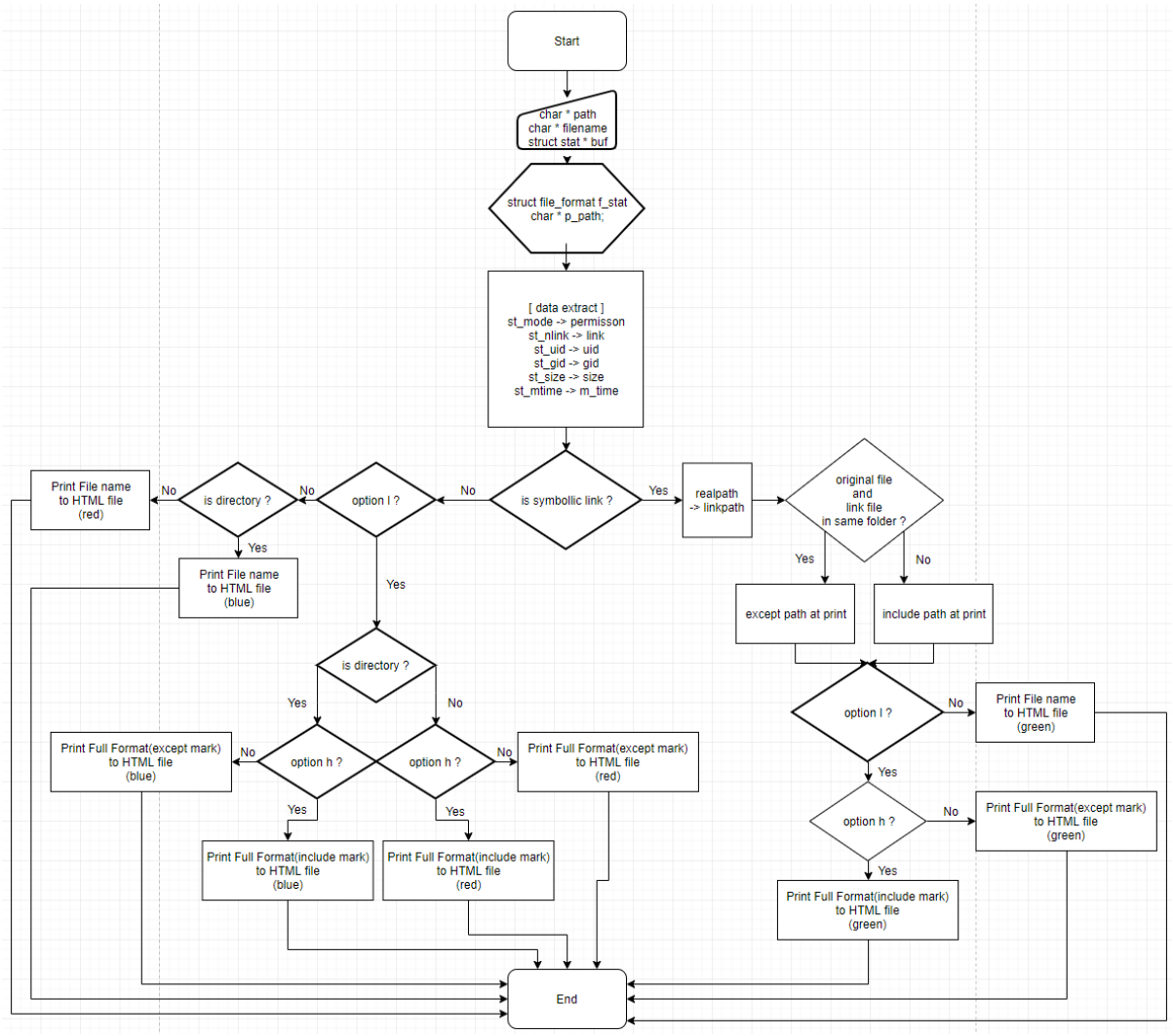
-int strcmp_i



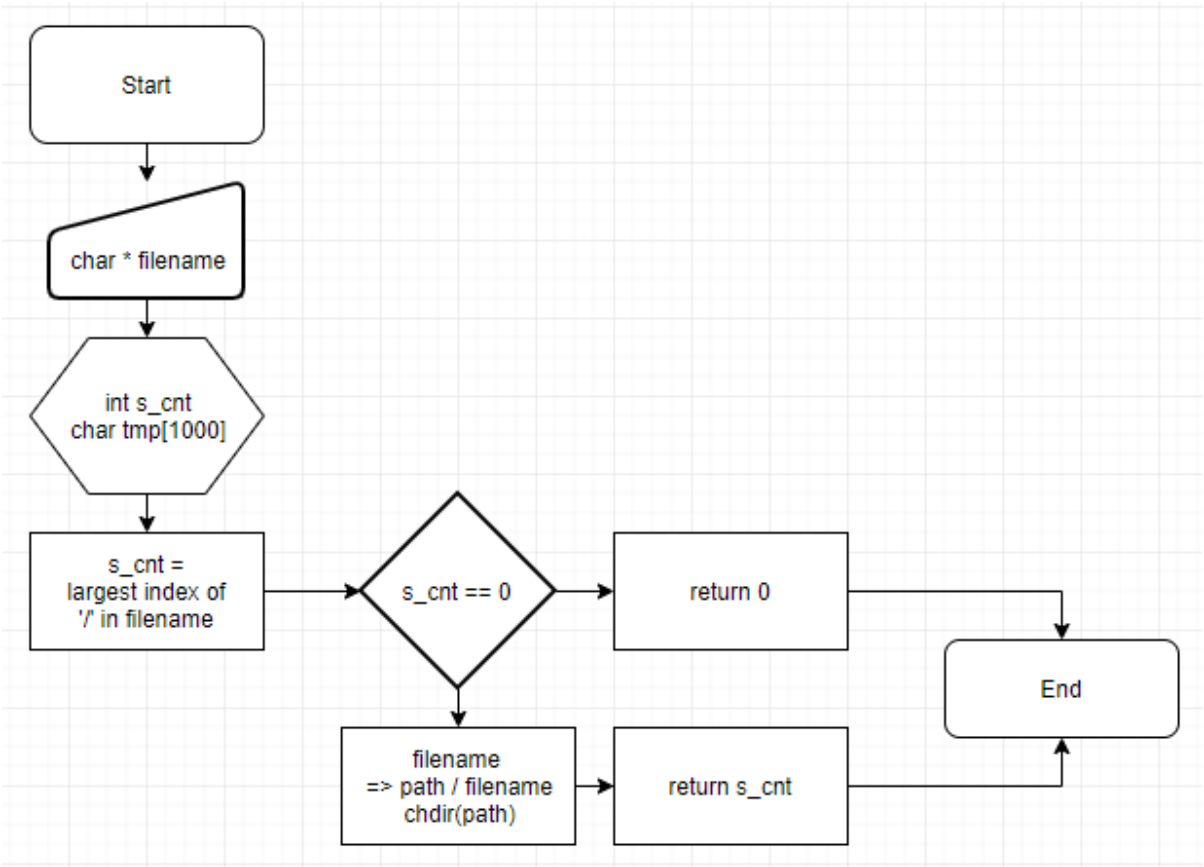
-void sort_list



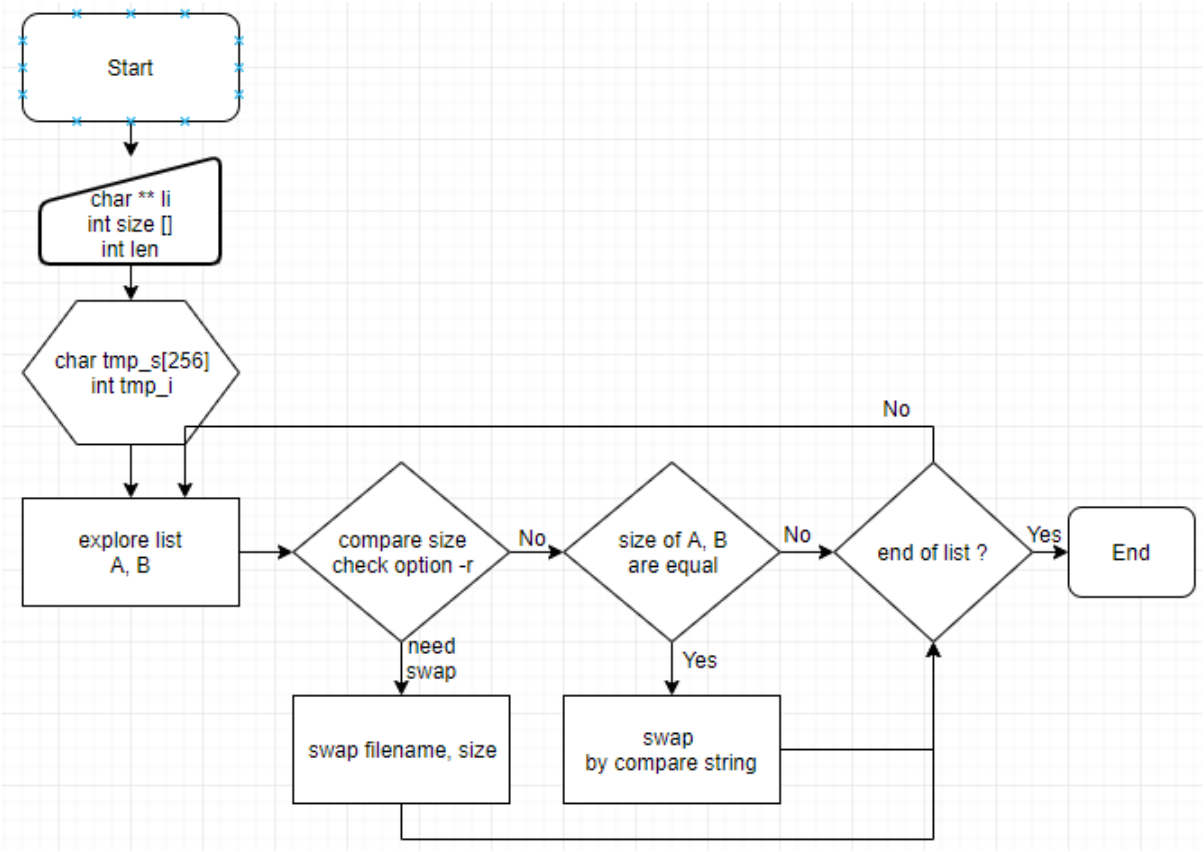
-void view_advanced_list



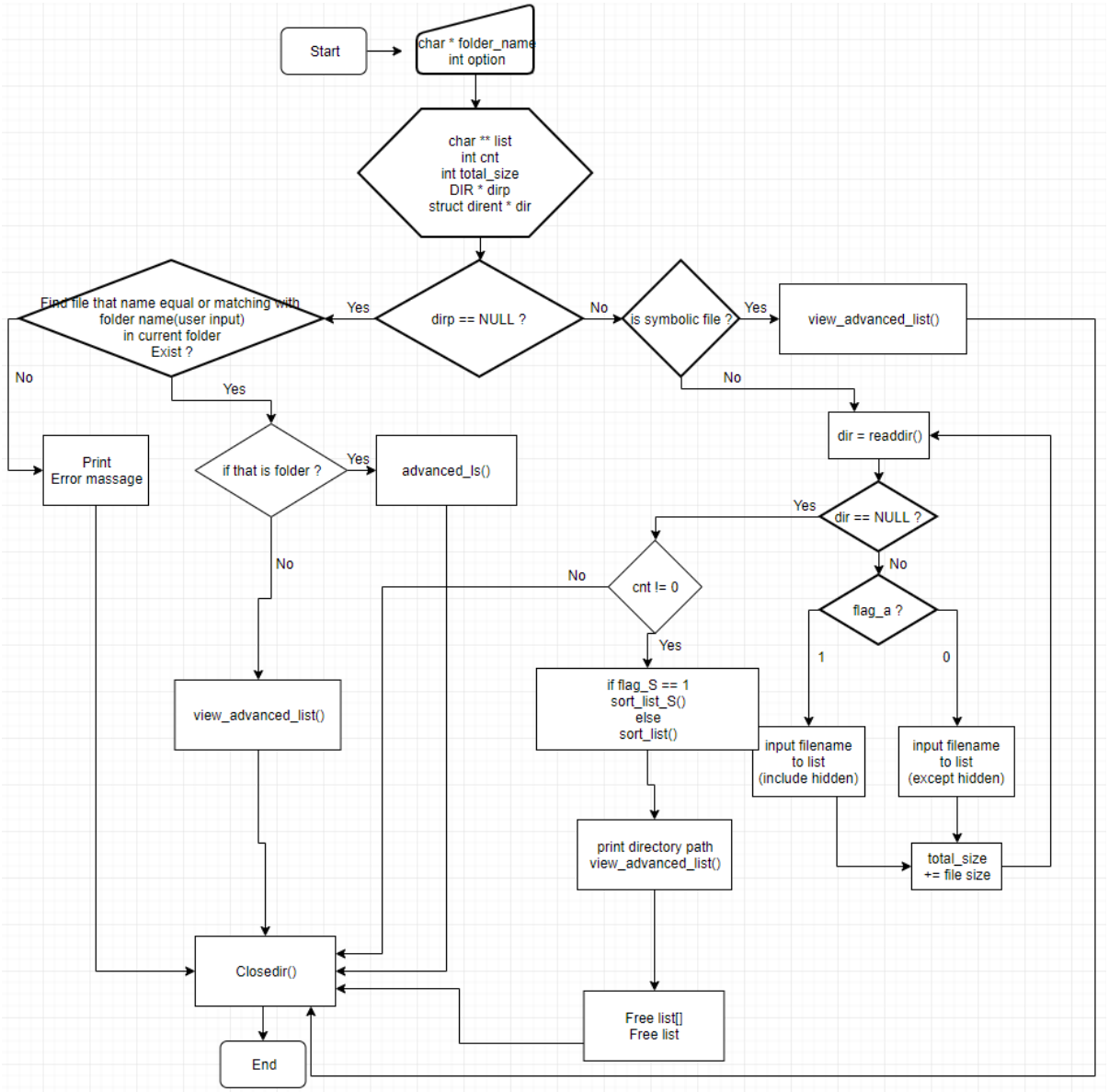
-int check_real_path



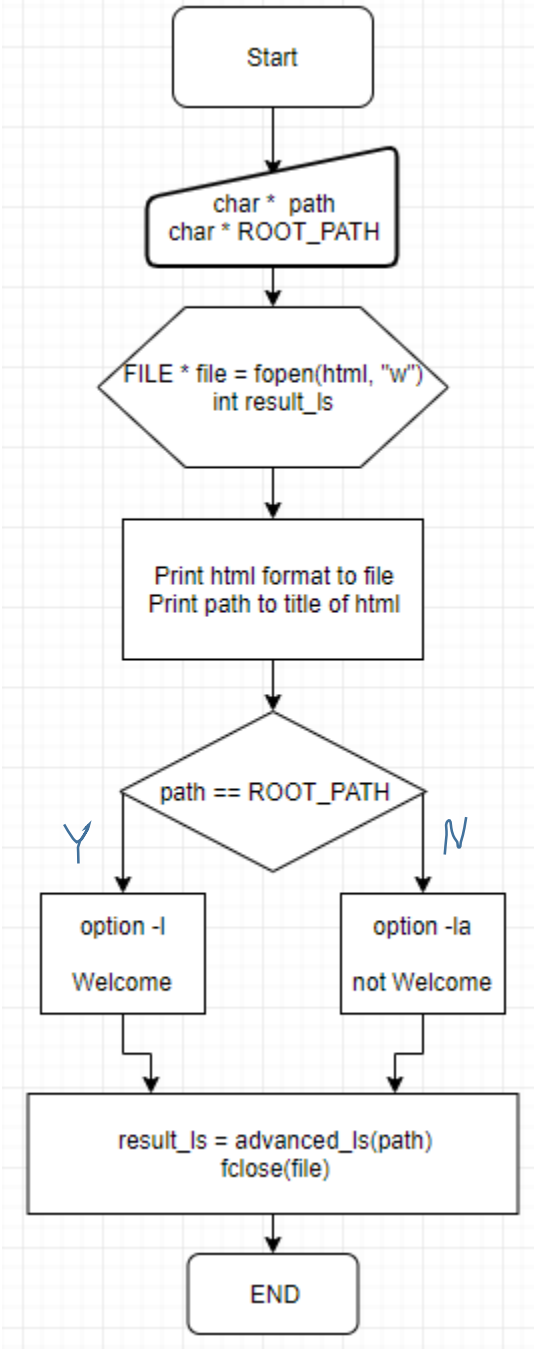
-void sort_list_S



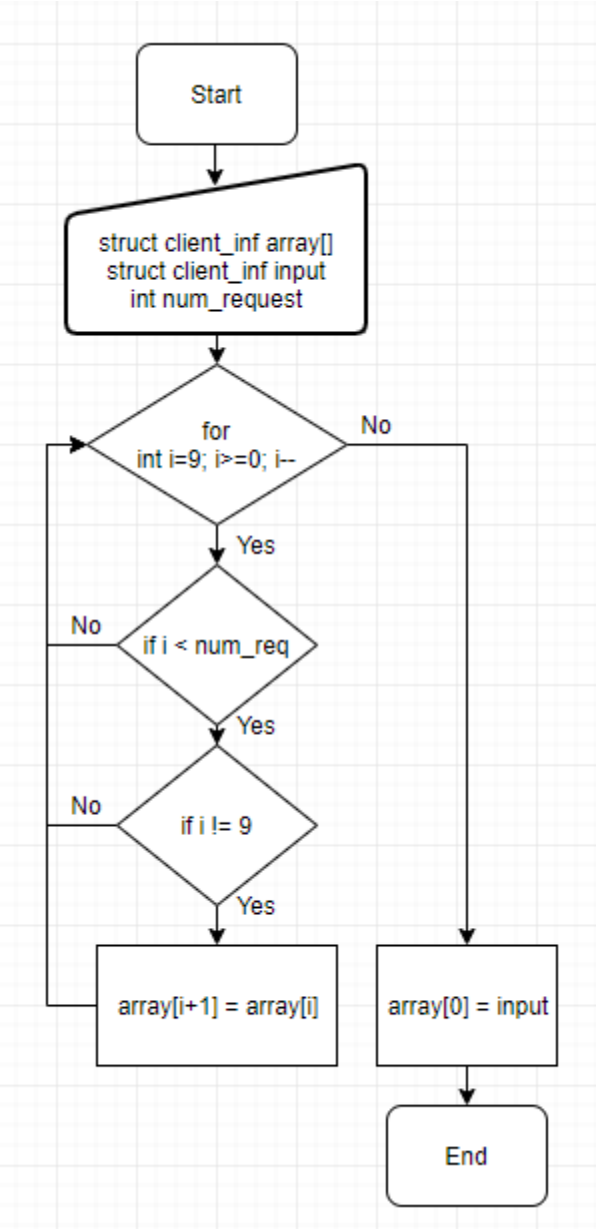
-int advanced_ls



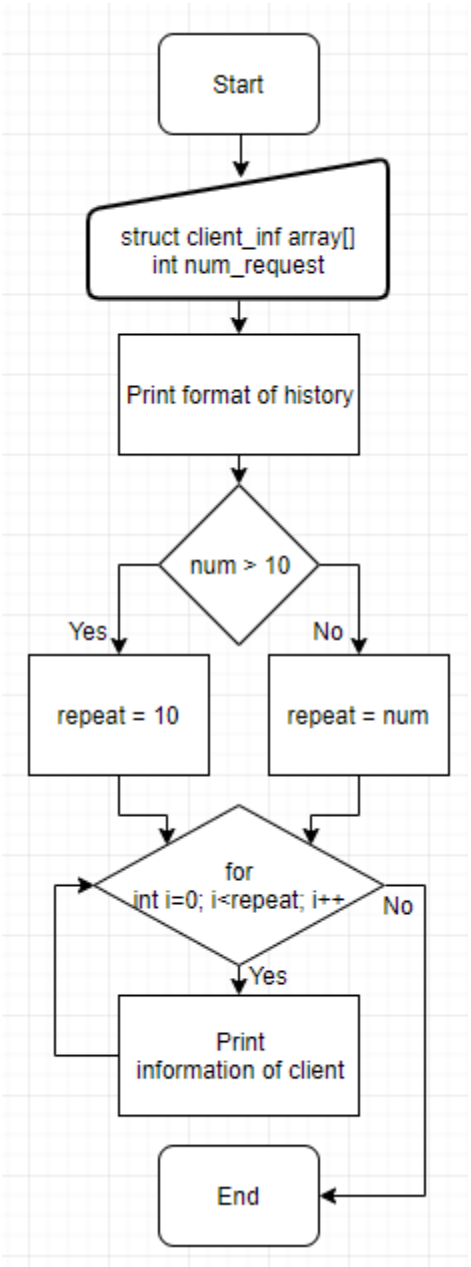
- int html_ls


Start
char * path
char * ROOT_PATH
FILE * file = fopen(html, "w")
int result_ls
Print html format to file
Print path to title of html
path == ROOT_PATH
Y
N
option -l
Welcome
option -la
not Welcome
result_ls = advanced_ls(path)
fclose(file)
END

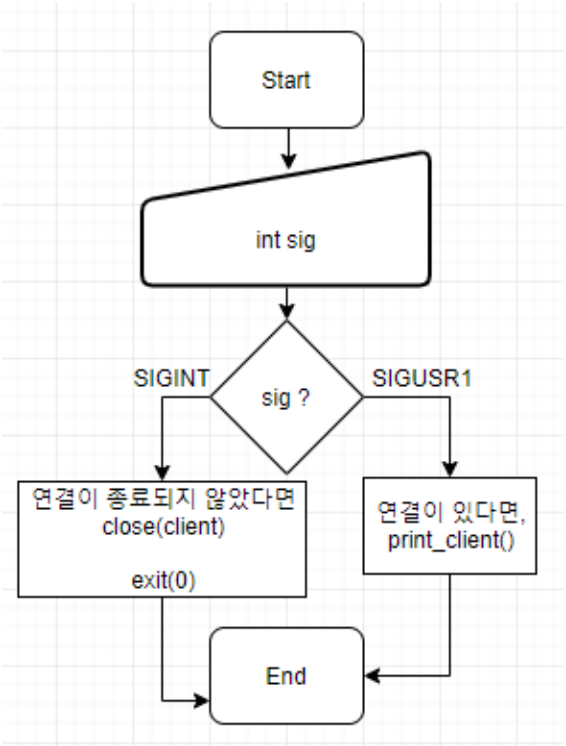
- add_client


Start
struct client_inf array[]
struct client_inf input
int num_request
for
int i=9; i>=0; i--
No
Yes
No
if i < num_req
Yes
No
if i != 9
Yes
array[i+1] = array[i]
array[0] = input
End

- print_client

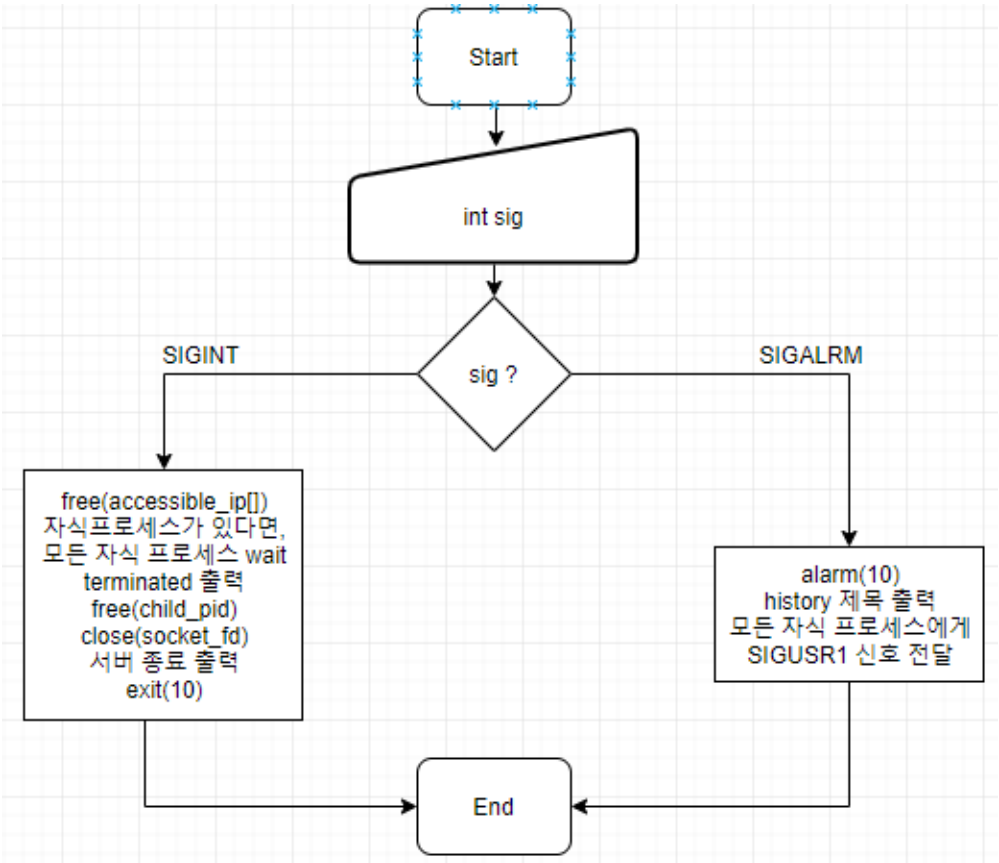


- signal_handler_c



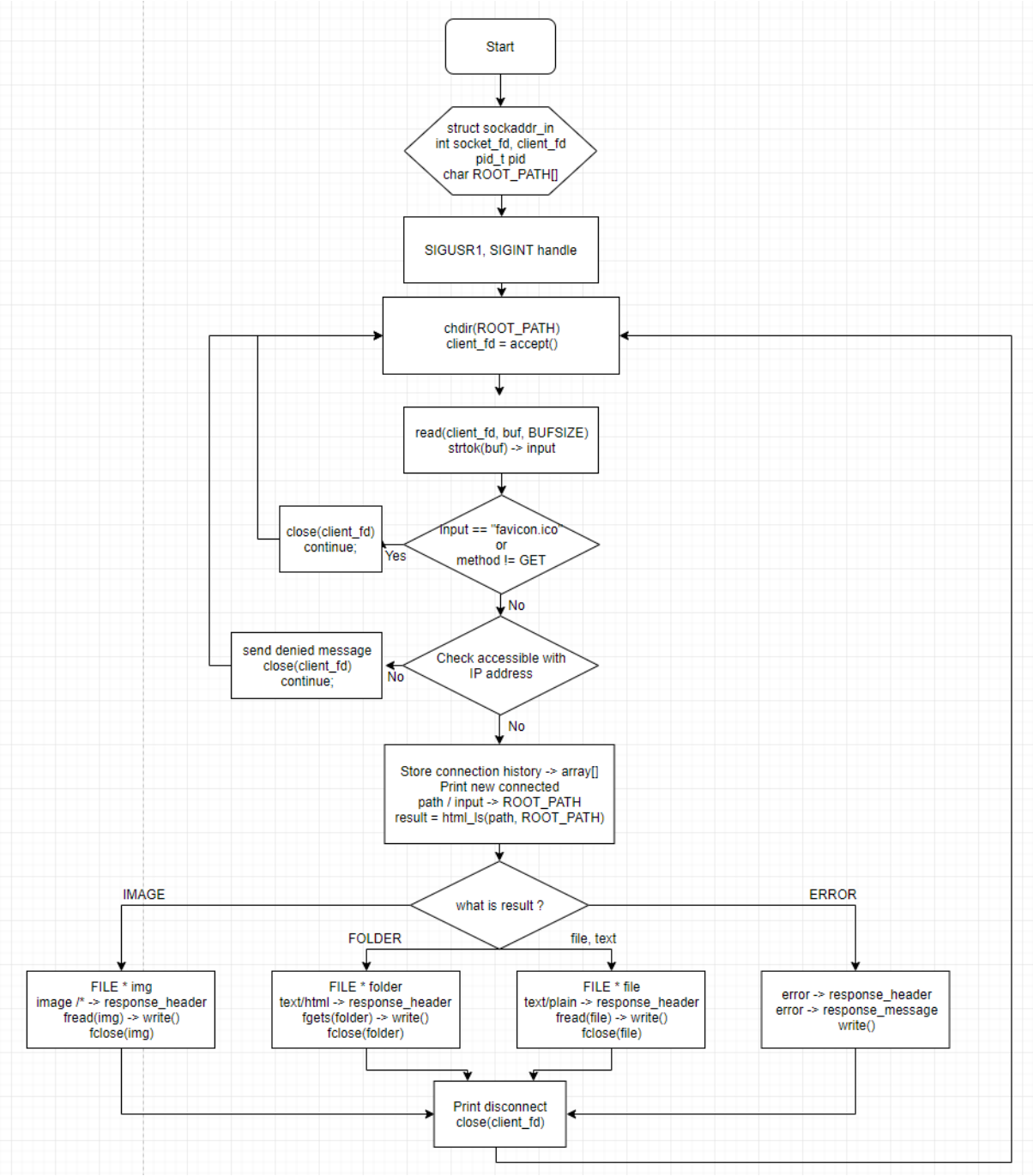
SIGINT -> 무시, 기존의 SIGINT -> SIGUSR2

- signal_handler_p

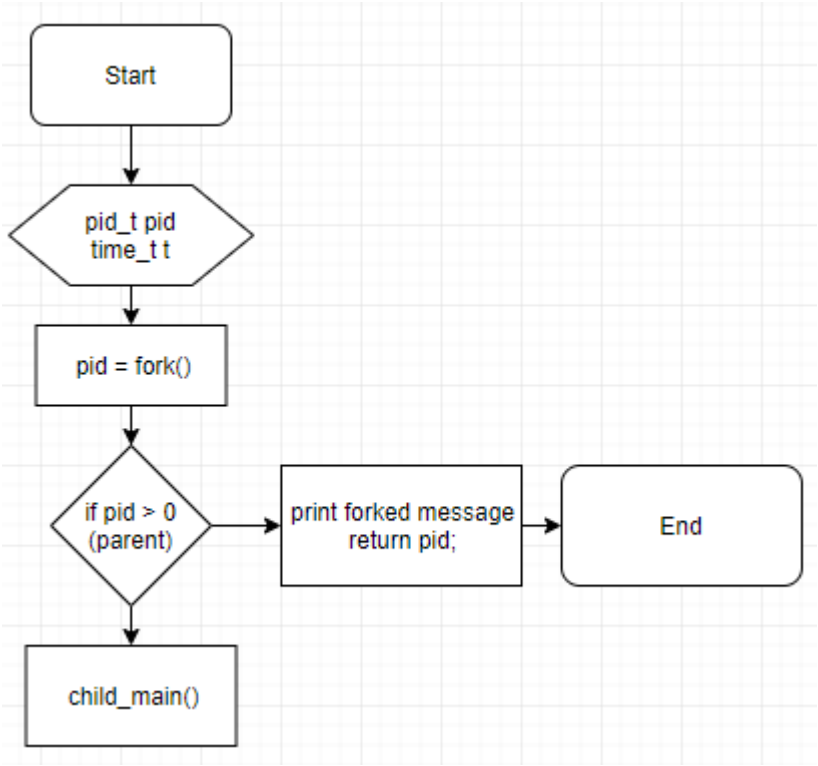


SIGINT 일 경우 wait 이전에 kill(모든 자식, SIGUSR2) 추가

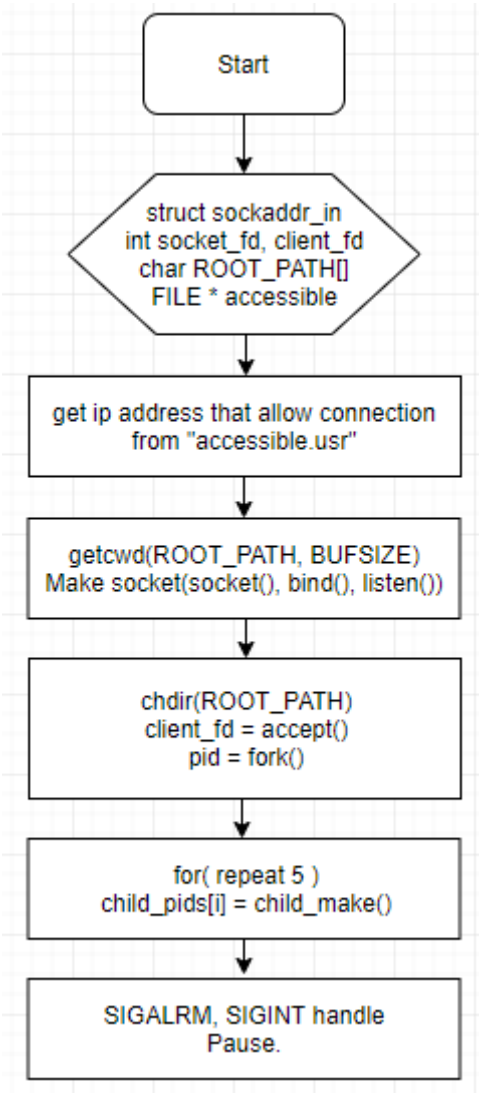
-child_main


Start
struct sockaddr_in
int socket_fd, client_fd
pid_t pid
char ROOT_PATH[]
SIGUSR1, SIGINT handle
chdir(ROOT_PATH)
client_fd = accept()
read(client_fd, buf, BUFSIZE)
strtok(buf) -> input
Input == "favicon.ico"
or
method != GET
Yes
close(client_fd)
continue;
No
Check accessible with
IP address
No
send denied message
close(client_fd)
continue;
No
Store connection history -> array[]
Print new connected
path / input -> ROOT_PATH
result = html_ls(path, ROOT_PATH)
what is result ?
IMAGE
FOLDER
file, text
ERROR
FILE * img
image /* -> response_header
fread(img) -> write()
fclose(img)
FILE * folder
text/html -> response_header
fgets(folder) -> write()
fclose(folder)
FILE * file
text/plain -> response_header
fread(file) -> write()
fclose(file)
error -> response_header
error -> response_message
write()
Print disconnect
close(client_fd)

- child_make


Start
pid_t pid
time_t t
pid = fork()
if pid > 0
(parent)
print forked message
return pid;
End
child_main()

- int main


Start
struct sockaddr_in
int socket_fd, client_fd
char ROOT_PATH[]
FILE * accessible
get ip address that allow connection
from "accessible.usr"
getcwd(ROOT_PATH, BUFSIZE)
Make socket(socket(), bind(), listen())
chdir(ROOT_PATH)
client_fd = accept()
pid = fork()
for(repeat 5)
child_pids[i] = child_make()
SIGALRM, SIGINT handle
Pause.

Pseudo code

```
int check_real_path(char * filename)
{
        for (int i = 0; i < strlen(filename); i++) // check directory.
        {
                입력된 파일 이름에 '/' 가 있을 경우 해당 index => s_cnt
        }
        if 파일 이름에 '/' 가 있다면 -> real path
        {
                devide < path > / < file >

                <path> 로 작업중인 디렉토리 변경
                s_cnt 값 반환
        }
        else real path 가 아니라면
                0 반환
}
```

```
void swap(char * str1, char * str2)
{
        str1과 str2 swap
}
```

```
int strcmp_i ( char * 비교할 첫 번째 문자열 A, char * 비교할 두 번째 문자열 B )
{
        <예외 처리> : r 옵션이면 거꾸로.
        A 랑 B 가 "." , ".." 이면 ".." 이 뒤로,
        둘 중 하나만 "." 이면 "."이 앞으로
        둘 중 하나만 ".." 이면 ".."이 앞으로

        if( A 가 B 보다 길면 )
                len = B 의 길이;
        else
                len = A 의 길이

        for(len 만큼 반복)
        {
                if(소문자라면)
                        대문자로 바꾼다
        }
        If 옵션이 –r 이라면
```

```
                if( A > B )         return = -1;
                else if( B > A )    return = 1;
                else                return = 0;

        else
                if( A > B )         return = 1;
                else if( B > A )    return = -1;
                else                return = 0;

}
```

```
void sort_list(char ** li, int len)
{
        for (저장된 파일이름 개수-1 만큼 반복)
        {
                if (더 이상 내용이 없다)
                        break;

                for (저장된 파일이름 개수-1 만큼 반복)
                {
                        if ( 앞에 저장된 파일이름 > 뒤에 저장된 파일 이름)
                                두 파일 이름을 swap
                }
        };
}
```

```
void sort_list_S(char ** li, int size[], int len)
{
        // sort like bublle sort
        for (int i = 0; i < len - 1; i++) // total cycle
        {
                if 리스트가 끝났다면
                        break;

                for (int j = 0; j < len - 1; j++) // one cycle
                {
                        if 파일 사이즈가 다르다면, 정렬 필요 -r 옵션에 따라서.
                        {
                                swap file name
```

```
                        swap file size
                }
                else if 파일 사이즈가 같다면, -> 문자열로 정렬
                {
                        if 정렬이 필요하다면
                        {
                                swap file name
                        }
                }
        }
    }
    return;
}
```

```
void view_advanced_list(char * 파일 경로, char * 파일 이름, struct stat * 파일 정보)
{
    St_mode에 저장되어 있는 파일 타입에 따라 permission[0] 결정
    St_mode에 저장되어 있는 user 권한에 따라 permission[1~3] 결정
    St_mode에 저장되어 있는 group 권한에 따라 permission[4~6] 결정
    St_mode에 저장되어 있는 other 권한에 따라 permission[7~9] 결정
    ➔ Permission = "- --- --- ---"

    st_nlink -> f_stat->link
    st_uid -> f_stat->uid
    st_gid -> f_stat->gid
    st_size -> f_stat->size
    st_mtime -> f_stat->m_time

    if 옵션이 –h 라면
    size = size / 1024.0, check_h++(단위가 몇번 상승했는지 체크)
    check_h 값에 따라서 단위(K, M, G) 결정

    // Full Format 출력
    if 파일 type 이 symbolic link 라면
    {
        if l옵션이 아니라면
            파일 이름만 HTML 파일로 출력 (green)
        else Full Format 출력
            If 옵션이 –h 라면, 사이즈 출력시 단위와 함께 출력
            Symbolic link 파일과 가리키는 경로 파일이 같은 폴더에 있다면 ->
가리키는 파일명만 HTML 파일로 출력(green)
            Symbolic link 파일과 가리키는 경로 파일이 다른 폴더에 있다면 -> 절대
경로 모두 HTML 파일로 출력(green)
    }
    else symbolic link 가 아니라면
    {
        if l옵션이 아니라면
            directory일 경우 파일 이름만 HTML 파일로 출력 (blue)
            아닐 경우 파일 이름만 HTML 파일로 출력 (red)
        else l 옵션 이라면
            if directory라면 (blue)
                If 옵션이 –h 라면, FullFormat 출력시 단위도 HTML 파일로 출력
```

```
                        else FullFormat HTML 파일로 출력
                else directory가 아니라면 (red)
                        If 옵션이 –h 라면, FullFormat 출력시 단위도 HTML 파일로 출력
                        else FullFormat HTML 파일로 출력

}
```

```
int advanced_ls(char * 입력한 폴더)
{
        if 폴더가 아니라면
        {
                혹시 절대경로로 입력된 폴더인지 확인       -> 맞으면 해당 디렉토리로 변경
                                                        -> 아니면 현재 디렉토리
                HTML 파일이면 return
                if table이 만들어져 있지 않고, flag_p가 -1이 아니라면,
                        옵션 l 에 따라 테이블 생성 -> flag_table=0;

                while 디렉토리 탐색
                {
                        if 파일을 찾았다면
                                view_advanced_list()
                        If fnmatch 로 매칭이 된다면,
                                폴더면 다시 advanced_ls()
                                파일이면 view_advanced_list()
                }
                If 출력한 파일이 없다면
                        에러 메시지 출력 -> 반환 -1
                else 1 반환
        }
        else // is folder
        {
                p 플래그가 표시되어있다면 return 3 => 폴더
                flag_table == 0 이라면 테이블이 열려있으므로 close
                flag_table = 1 set -> 테이블이 안만들어져있다.

                심볼릭 링크로 연결된 폴더인지 확인
                -> 심볼릭 링크 폴더이고 옵션이 l 이면 심볼릭 링크만 Full Format 출력

                아니면 탐색 시작
                while 입력된 폴더 탐색
                {
                        HTML 파일은 처리하지 않음.

                        if 파일이 없다면
                                반복문 탈출

                        if 히든 파일이고 옵션이 –l 이나 default 라면
                        {
                                List에 파일 이름 입력
                                Total size += 파일 size
                        }
                        else if 옵션이 –a 이나 –la 라면 모든 파일 입력
```

```
                        {
                                List에 파일 이름 입력
                                Total size += 파일 size

                        }
                }

                Directory path 출력
                if 파일이 존재한다면
                {
                        List 정렬.
                        Total size 출력

                        if l 옵션이라면 -> Full Format Table 생성
                        else -> 파일 이름 Table 생성
                        for 파일 개수 만큼 반복
                                view_advanced_list()

                        테이블 close
                        flag_table=1
                }
        }
        폴더 닫기
}
```

```
int html(char * 찾을 경로, char * 루트 경로)
{
        origin_wd에 루트 경로 복사
        HTML 파일 기본 포맷 입력, 타이틀에 현재 디렉토리 입력.

        HTML_FILE 에 쓰기위한 fopen

        title에 찾을 경로 입력

        찾을 경로가 루트 경로이면 -l옵션하고 Welcome, 아니면 -la 옵션하고 Welcome 생략

        result = advanced_ls(path);

        테이블 close
        html 파일 포맷 close
        open한 파일 close

        result 반환.
}
```

```
- int main
{
        "accessible.usr" 파일 오픈
        접근 허용된 IP 주소 읽어와서 accessible_ip 배열에 저장
```

```
        "httpd.conf" 파일 오픈
        각 요소의 데이터를 읽어와서 각 요소에 맞는 데이터 저장

        소켓 생성 -> 연결 준비(socket(), bind(), listen())
        공유 메모리 생성 후 현재 프로세스에 연결, SH_DATA 구조체 형태로 사용
        공유 메모리의 데이터값 0 으로 초기화

        10 초 뒤 알람 설정
        SIGPIPE 무시 신호 처리

        자식 프로세스 5 개 fork() 하고 자식 프로세스의 pid 를 child_pids 배열에 저장

        SIGALRM, SIGINT, SIGUSR1, SIGUSR2, 30 시그널 처리

        Pause;
}
```

```
void child_main()
{
        29, SIGUSR1, SIGUSR2 신호 처리 선언

        while(1)
        {
                ROOT PATH 로 디렉토리 변경
                클라이언트로 부터의 연결을 기다림

                연결이 되면 예외처리 확인 (EXIT, favicon.ico)
                접근 허용된 IP 주소 확인(strcmp, fnmatch)

                공유 메모리 get 하고 현재 프로세스에 연결 -> SH_DATA 구조체 형태로 사용
                현재 클라이언트의 연결 정보를 공유 메모리에 입력하는 스레드 생성
                                        -> 스레드는 connect_client() 함수를 사용

                        연결 상태 출력
                        루트 경로가 아닌 경우 루트 경로 / input -> path
                        루트 경로인 경우 루트 경로 -> path

                        result 에 html_ls(path) 의 결과 출력 (폴더:3, 파일:1, 오류:-1)

                        if  이미지 파일인 경우
                                이미지 파일 open.
                                헤더 메세지 : image/*
                                응답 메세지에 이미지 파일 내용 전송
                        else if 폴더인 경우
                                html_ls 의 결과인 html 파일 open
                                헤더 메세지 : text/html
                                응답 메세지에 html 파일 내용 전송
                        else if 파일인 경우
                                파일 open
                                헤더 메세지 : text/plain
```

```
                                응답 메세지에 파일 내용 전송
                        else
                                에러 예외처리 메세지 전송

                5 초간 대기
                현재 클라이언트의 연결 끊김을 입력하는 스레드 생성
                                                -> 스레드는 disconnect_client() 함수를 사용
                연결 종료 메세지 출력
                연결 종료.
        }
}
```

```
pid_t child_make()
{
        pid = fork() // 자식 프로세스 생성
        부모 프로세스라면
                forked가 되었다는 메세지 출력
                return pid;
        자식 프로세스라면
                child_main() 함수 실행
}
```

```
void connect_client()
{
        mutex_lock
        현재 idle --
        현재 busy ++
        요청개수 ++

        히스토리에 연결 기록 저장

        mutex_unlock
        부모에게 30 신호를 전달
}
```

```
void disconnect_client()
{
        mutex_lock
        현재 idle ++
        현재 busy --
        mutex_unlock
        부모에게 30 신호를 전달
}
```

```
void check_idle()
{
```

```
        mutex_lock

        현재자식개수 > 최대자식 이라면, 첫번째 자식에게 SIGUSR2 전달
        현재 idle > 최대 idle 이라면, 첫번째 자식에게 SIGUSR2 전달
        현재 idle < 최대 idle 이라면, child_make()

        mutex_unlock
}
```

```
void reduce_idle()
{
        mutex_lock
        tmp_pid = 아무 자식이든 죽을때까지 wait 후 pid 반환
        mutex_unlock
}
```

```
void print_client()
{
        if (num_request > MaxHistory)
                repeat = MaxHistory
        else
                repeat = num_request

        히스토리 제목 출력

        repeat 만큼 반복
                array 에 들어있는 연결 정보 출력
}
```

```
void signalHandler_c(int sig)
{
        if sig == 29
                client 와의 연결이 종료되지 않았다면 -> 종료
                exit
        if sig == SIGUSR2
                IDLE 이라면 부모에게 USR1 신호를 보내고 exit.
                BUSY 라면 부모에게 USR2 신호를 보냄
}
```

```
void signalHandler_p(int sig)
{
        if sig == SIGINT
                free accessible_ip
                kill(모든 자식, SIGUSR2)
                waitpid (모든 자식)
                printf(terminated)
```

```
                공유 메모리 제거
                close(소켓)
                exit
        if sig == SIGALRM
                10 초뒤 알람 설정
                스레드를 통해 print_client() 함수 실행
        if sig == SIGUSR1
                스레드를 통해 reduce_idle() 함수 실행
        if sig == SIGUSR2
                flag 값을 1 증가시키고 flag 번째 자식에게 USR2 신호 전달
        if sig == 30
                스레드를 통해 check_idle() 함수 실행
}
```

Result

```
sp2015722087@ubuntu:~/sp/shared/ch4/4-2$ ./ipc_server_40302
[Fri Jun  7 16:11:57 2019] Server is started.
[Fri Jun  7 16:11:57 2019] 14908 process is forked.
[Fri Jun  7 16:11:57 2019] idleProcessCount : 1
[Fri Jun  7 16:11:57 2019] 14909 process is forked.
[Fri Jun  7 16:11:57 2019] idleProcessCount : 2
[Fri Jun  7 16:11:57 2019] 14910 process is forked.
[Fri Jun  7 16:11:57 2019] idleProcessCount : 3
[Fri Jun  7 16:11:57 2019] 14911 process is forked.
[Fri Jun  7 16:11:57 2019] idleProcessCount : 4
[Fri Jun  7 16:11:57 2019] 14912 process is forked.
[Fri Jun  7 16:11:57 2019] idleProcessCount : 5

==================== New Client ====================
[Fri Jun  7 16:11:58 2019]
IP : 127.0.0.1
Port : 37026
====================================================

[Fri Jun  7 16:11:58 2019] idleProcessCount : 4
```

- 서버를 실행하고 started 메세지와 forked 메세지의 출력. 새로운 연결이 들어오면 idle 프로세스의 개수가 하나 줄어든다.

```
[Fri Jun  7 16:11:58 2019] idleProcessCount : 4

==================== New Client ====================
[Fri Jun  7 16:11:59 2019]
IP : 127.0.0.1
Port : 37538
====================================================

[Fri Jun  7 16:11:59 2019] idleProcessCount : 3
[Fri Jun  7 16:11:59 2019] 14917 process is forked.
[Fri Jun  7 16:11:59 2019] idleProcessCount : 4
[Fri Jun  7 16:11:59 2019] 14918 process is forked.
[Fri Jun  7 16:11:59 2019] idleProcessCount : 5

==================== New Client ====================
[Fri Jun  7 16:11:59 2019]
IP : 127.0.0.1
Port : 38050
====================================================

[Fri Jun  7 16:11:59 2019] idleProcessCount : 4

==================== New Client ====================
[Fri Jun  7 16:12:00 2019]
IP : 127.0.0.1
Port : 38562
====================================================

[Fri Jun  7 16:12:00 2019] idleProcessCount : 3
[Fri Jun  7 16:12:00 2019] 14923 process is forked.
[Fri Jun  7 16:12:00 2019] idleProcessCount : 4
[Fri Jun  7 16:12:00 2019] 14924 process is forked.
[Fri Jun  7 16:12:00 2019] idleProcessCount : 5
```

- idle 프로세스의 개수가 MinIdleNum 보다 작을 경우 프로세스를 추가로 fork 하여 개수를 유지한다.

```
================ Disconnected Client ===============
[Fri Jun  7 16:12:40 2019]
IP : 127.0.0.1
Port : 47266
====================================================

[Fri Jun  7 16:12:40 2019] idleProcessCount : 7
[Fri Jun  7 16:12:45 2019] 14924 process is teminated
[Fri Jun  7 16:12:45 2019] idleProcessCount : 6
[Fri Jun  7 16:12:45 2019] 14946 process is teminated
[Fri Jun  7 16:12:45 2019] idleProcessCount : 5
```

- 여러번의 연결 종료로 인해 idle 프로세스의 개수가 maxidlenum 보다 커지면서 일부 프로세스를 종료시켜 개수를 유지하는 모습이다.

```
================ Connection History ================
No.     IP              PID     PORT    TIME
1       127.0.0.1       14959   53922   Fri Jun  7 16:16:37 2019
2       127.0.0.1       14953   53410   Fri Jun  7 16:16:37 2019
3       127.0.0.1       14952   52898   Fri Jun  7 16:16:36 2019
4       127.0.0.1       14953   47266   Fri Jun  7 16:12:40 2019
5       127.0.0.1       14946   46754   Fri Jun  7 16:12:40 2019
6       127.0.0.1       14947   46242   Fri Jun  7 16:12:39 2019
7       127.0.0.1       14918   45730   Fri Jun  7 16:12:39 2019
8       127.0.0.1       14917   45218   Fri Jun  7 16:12:39 2019
9       127.0.0.1       14912   44706   Fri Jun  7 16:12:39 2019
10      127.0.0.1       14923   44194   Fri Jun  7 16:12:38 2019
====================================================

^C
[Fri Jun  7 16:17:14 2019] 14953 process is teminated
[Fri Jun  7 16:17:14 2019] TotalProcessCount : 4
[Fri Jun  7 16:17:14 2019] 14958 process is teminated
[Fri Jun  7 16:17:14 2019] TotalProcessCount : 3
[Fri Jun  7 16:17:14 2019] 14959 process is teminated
[Fri Jun  7 16:17:14 2019] TotalProcessCount : 2
[Fri Jun  7 16:17:14 2019] 15025 process is teminated
[Fri Jun  7 16:17:14 2019] TotalProcessCount : 1
[Fri Jun  7 16:17:14 2019] 15026 process is teminated
[Fri Jun  7 16:17:14 2019] TotalProcessCount : 0
[Fri Jun  7 16:17:14 2019] Server is terminated
sp2015722087@ubuntu:~/sp/shared/ch4/4-2$
```

- 서버를 종료시키는 모습이다 현재 전체 프로세스의 개수가 6 개 이므로 6 개를 종료시키는 모습을 출력하고, 공유메모리를 제거한뒤에 서버를 종료시켰다.

Conclusion

이번 과제는 지난번의 과제를 기반으로 기능을 추가하는 과제이지만, 공유 메모리와 스레드를 처음으로 다뤄보게 되어 걱정이 앞서는 과제였다. 공유 메모리를 어떤 식으로 사용해야 하는지, 개념은 무엇인지 잘 몰라서 강의 자료를 많이보고, 구글에서도 많이 검색해서 개념을 익힌 후 여러 가지 방법으로 사용을 해본 뒤에야 이해를 하고 사용하게 되었다.
스레드를 사용하는 것도 매우 어려웠다. 무엇인지도 모르고, 함수는 되게 많아서 이것 또한 공부를 많이하고 나서야 조금 이해하고 사용하게 되었다.

결론적으로는 한번의 과제에서 여러가지를 처음 접하고, 배우고, 사용하면서 간단하게 작성된 코드이지만 개념과 사용법을 익히게 된 것 같아 유익한 과제였다.