

2019. 5. 30.

Assignment # 4-1

금요일 - 이성원 교수님
2015722087 컴퓨터정보공학부
김민철

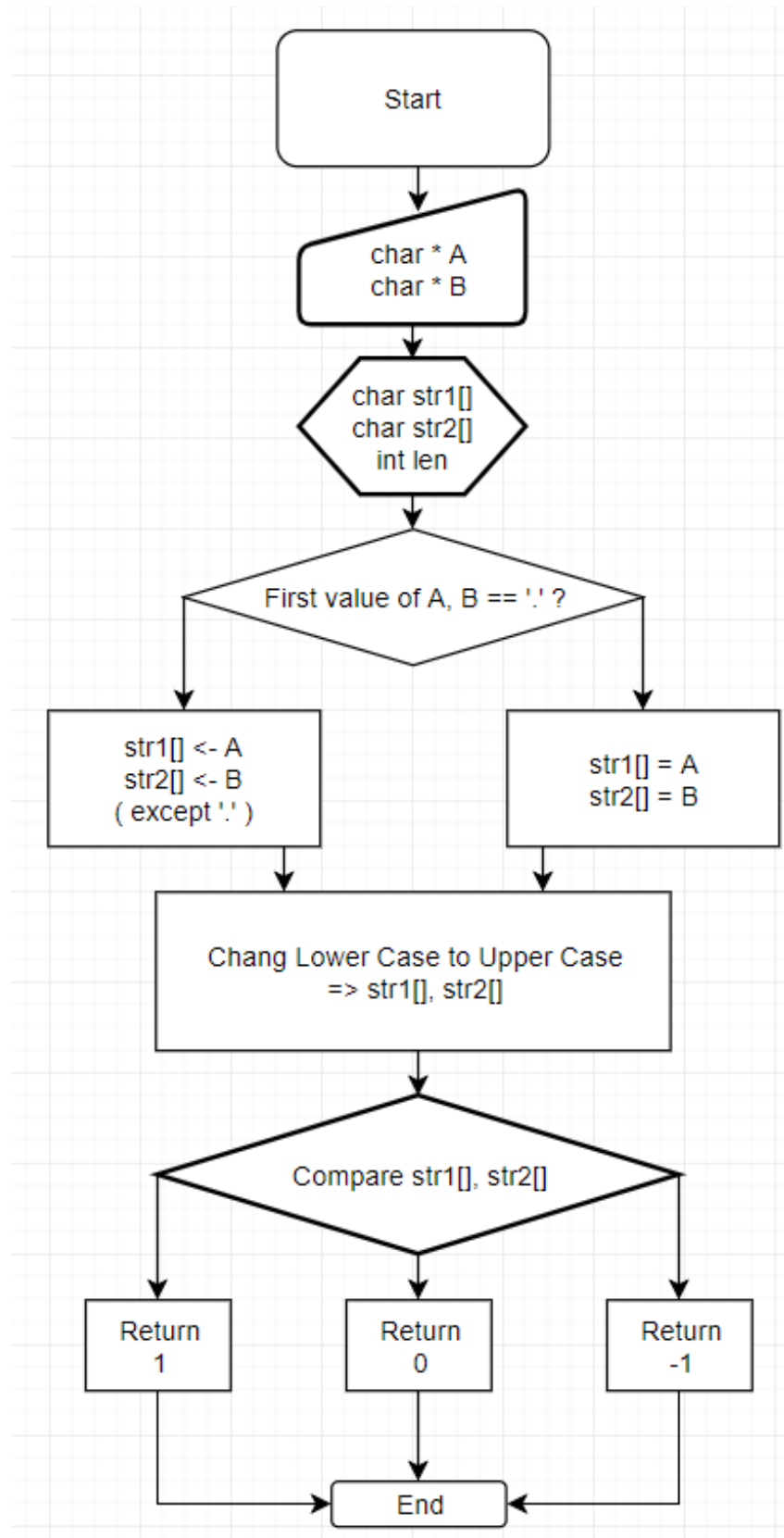
pre-forked web-server 구현

Introduction

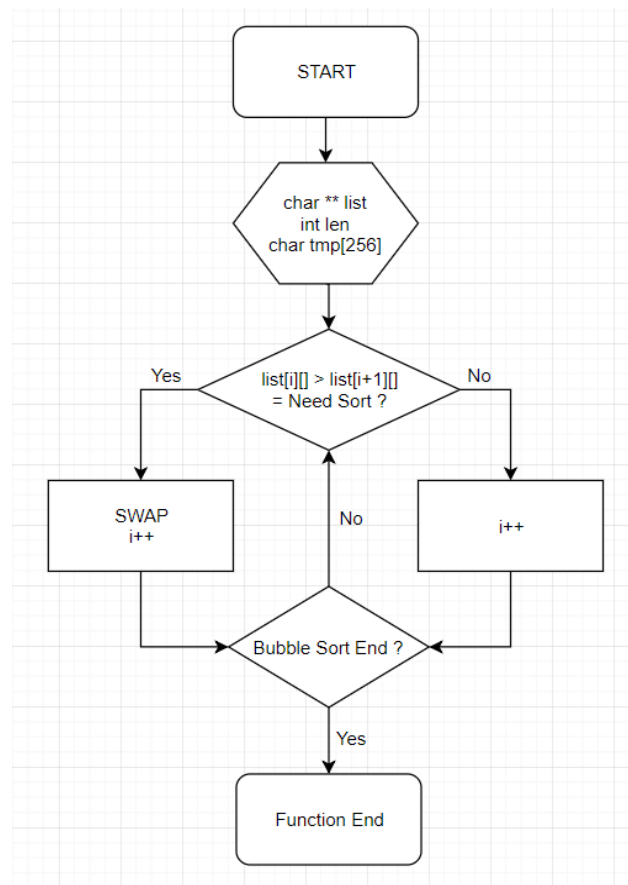
이번 과제에서는 3-3 과제에서 구현하였던 advanced web-server 를 연결 후 fork() 하는 것이 아닌 미리 fork()를 통해 자식 프로세스를 생성하여 연결시 미리 생성된 자식 프로세스로 처리하는 서버를 구현하는 과제이다. 미리 자식 프로세스를 생성해두면 연결 후에 fork() 해야 하는 비용을 줄여 보다 빠르게 처리할 수 있다는 장점이 있다.

Flow Chart

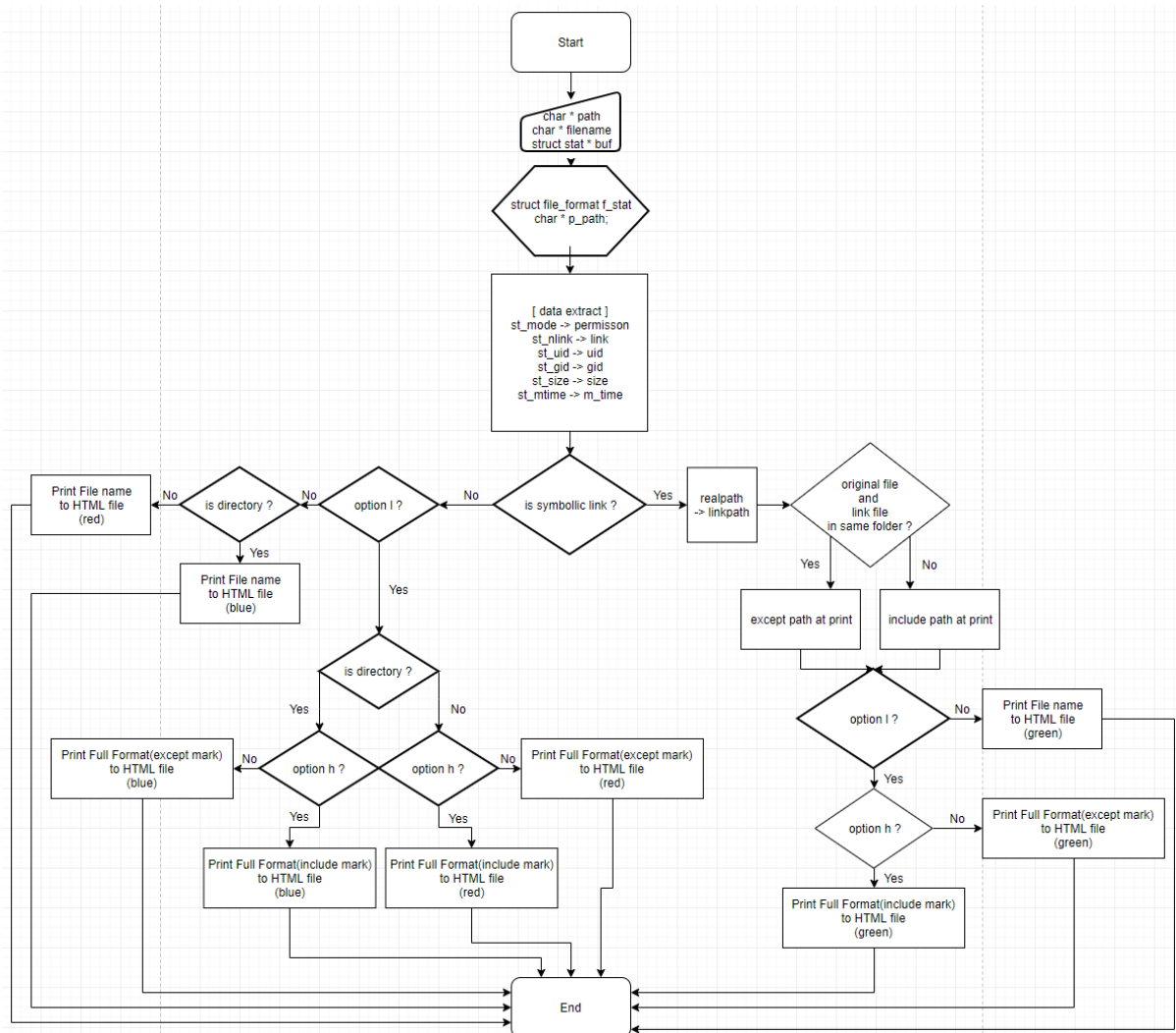
-int strcmp_i



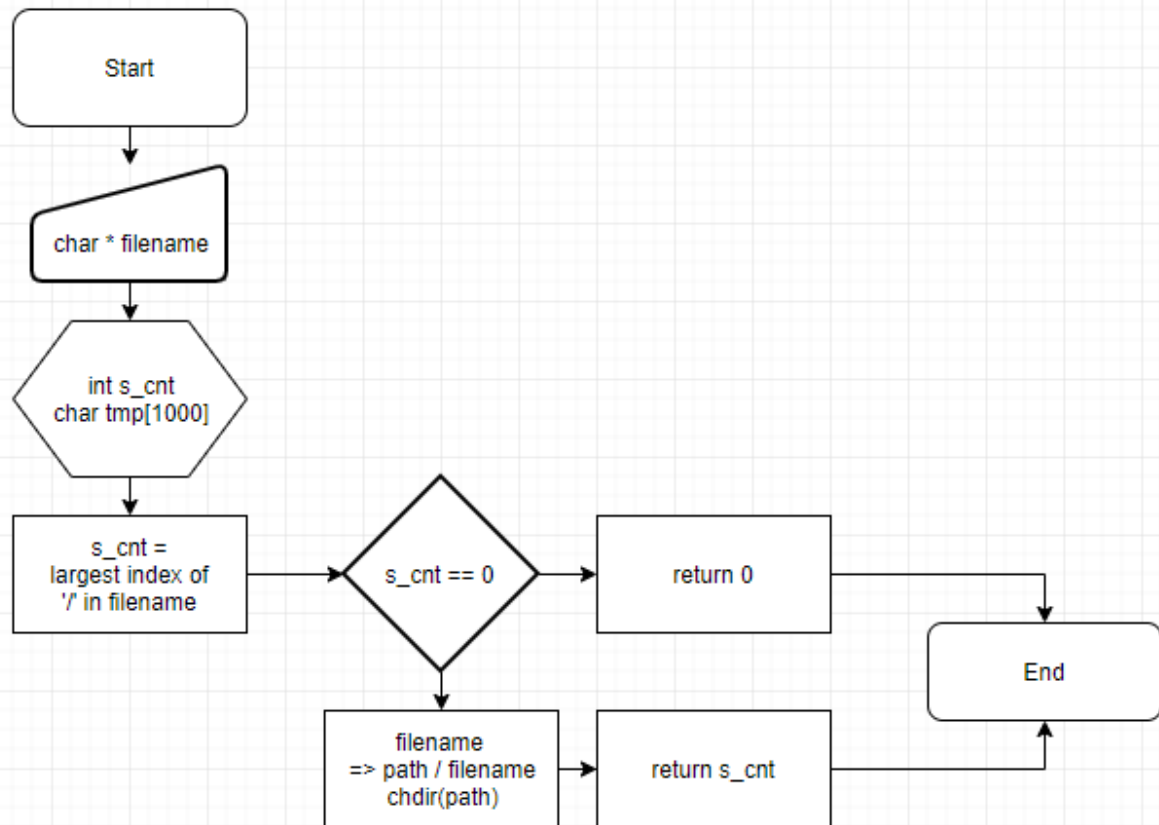
-void sort_list



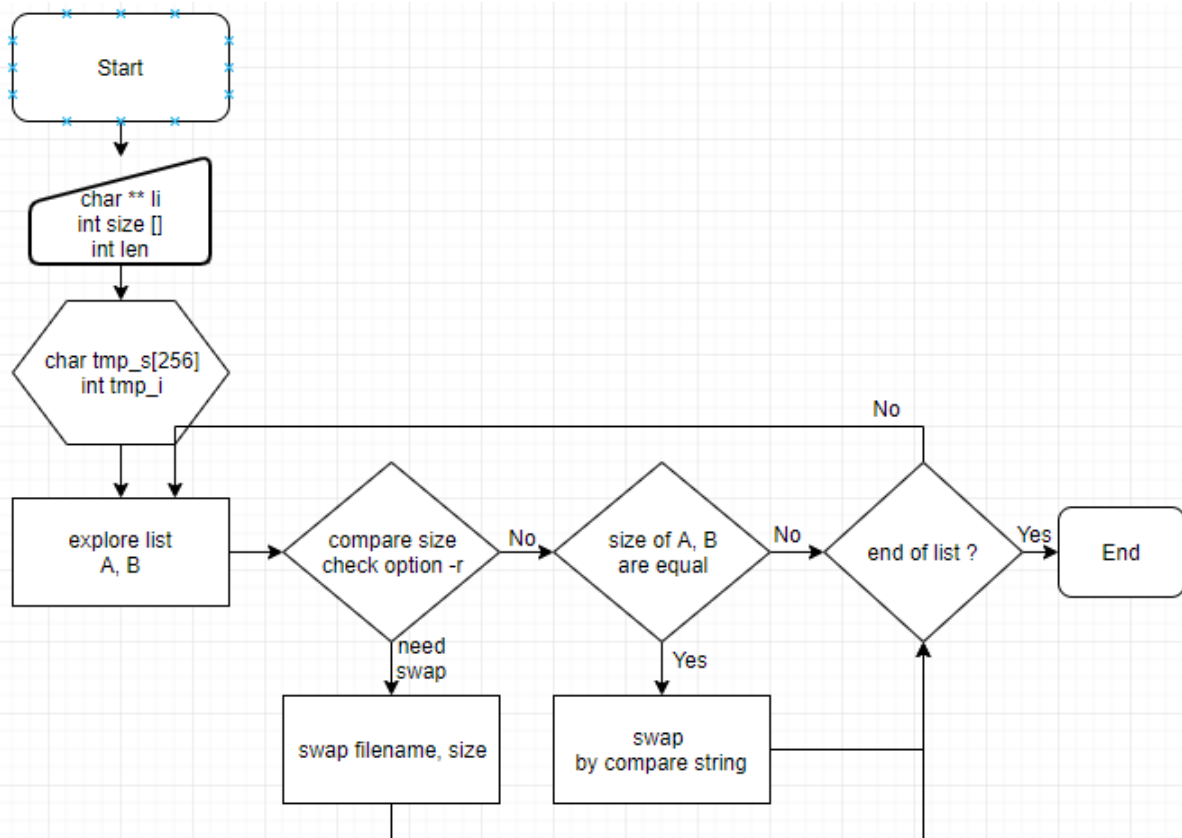
-void view_advanced_list



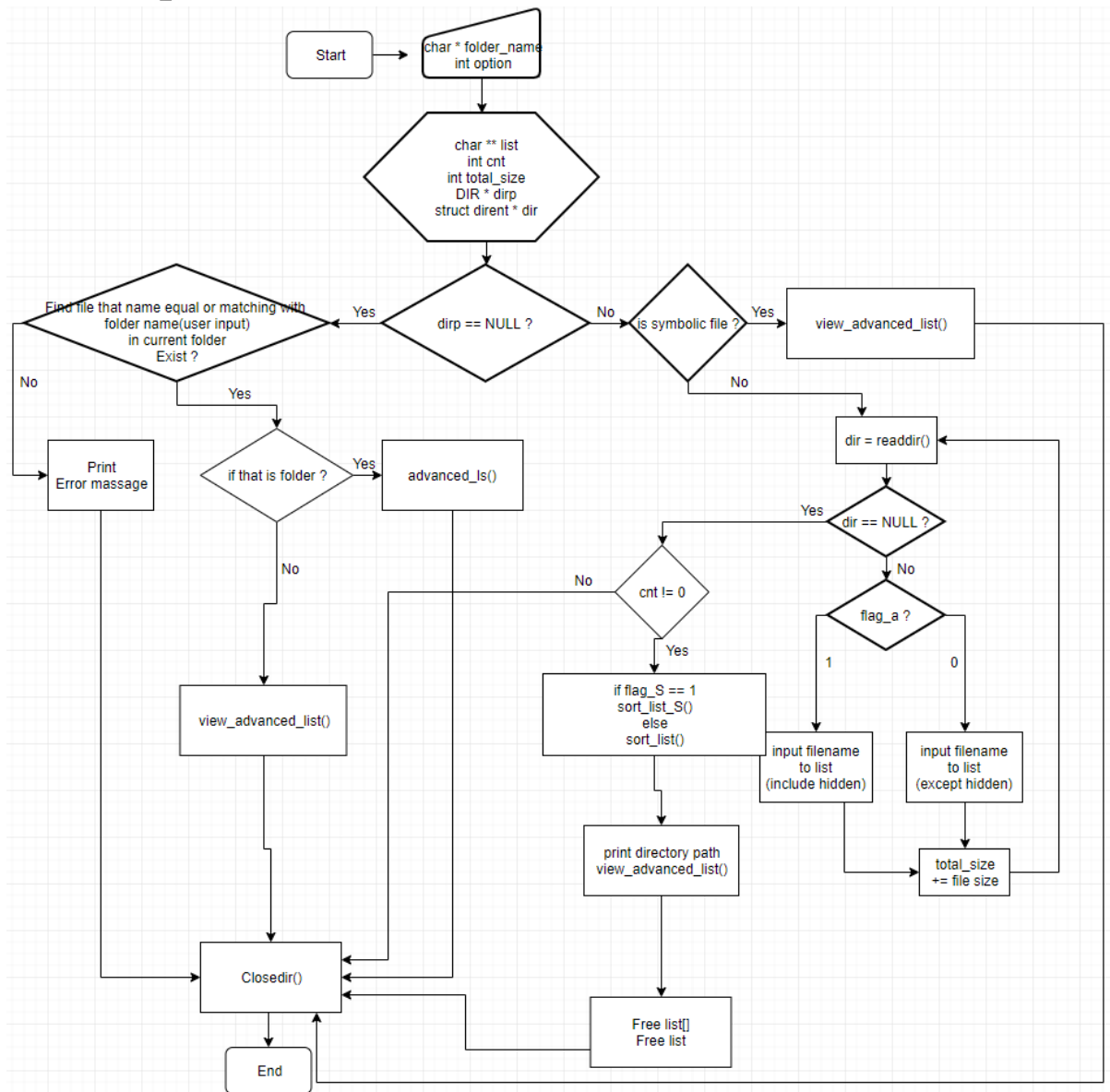
-int check_real_path



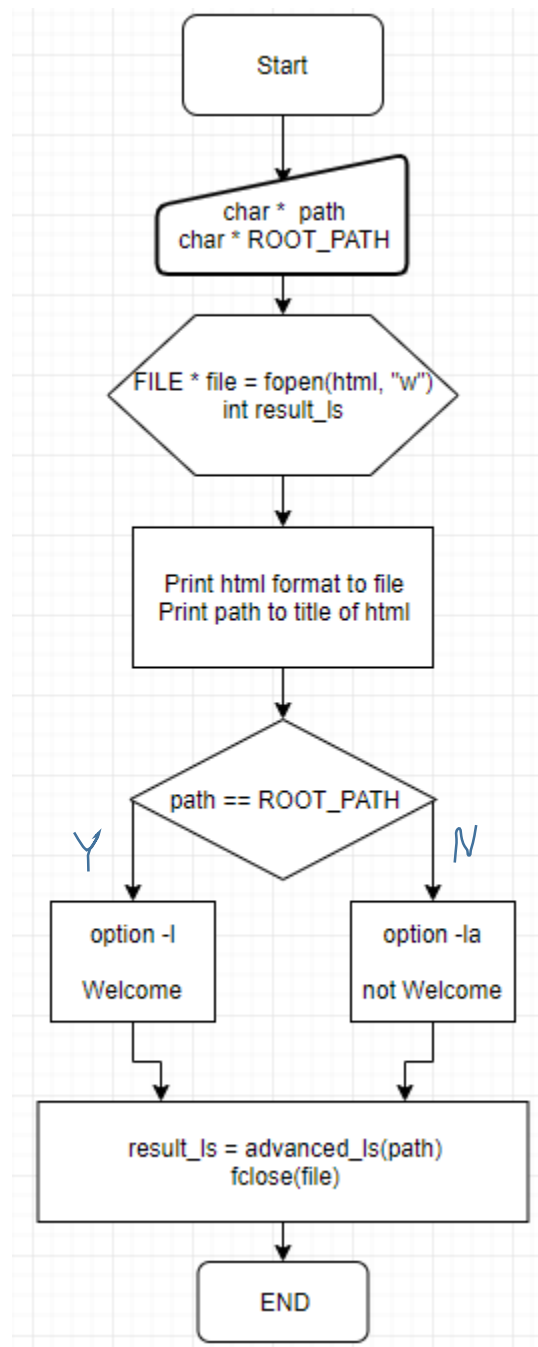
-void sort_list_S



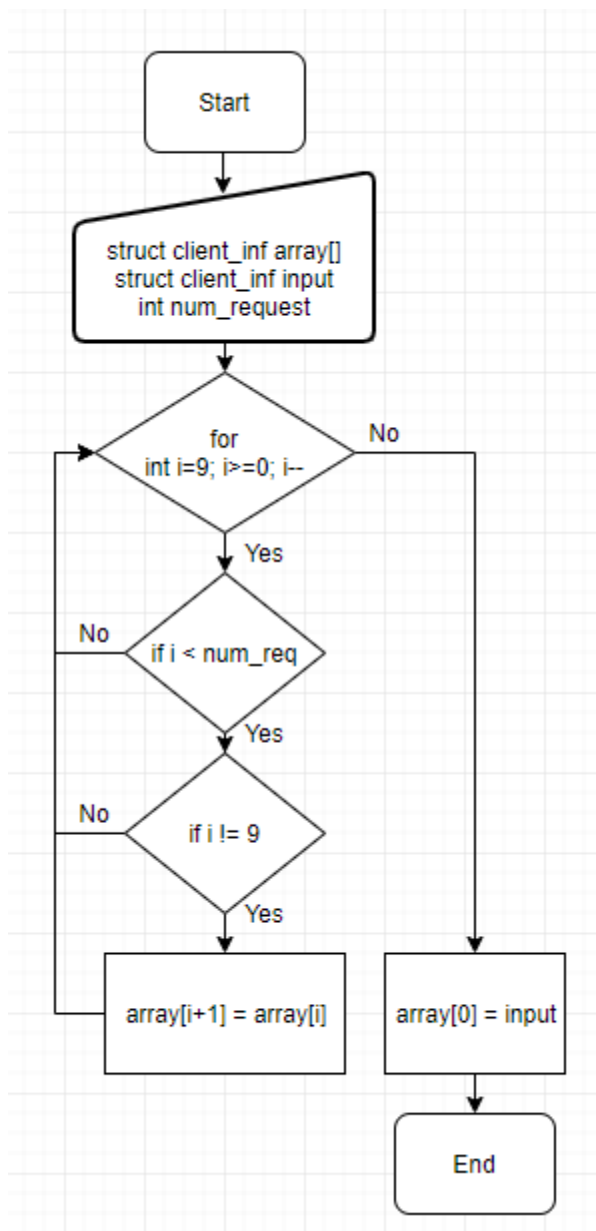
-int advanced_ls



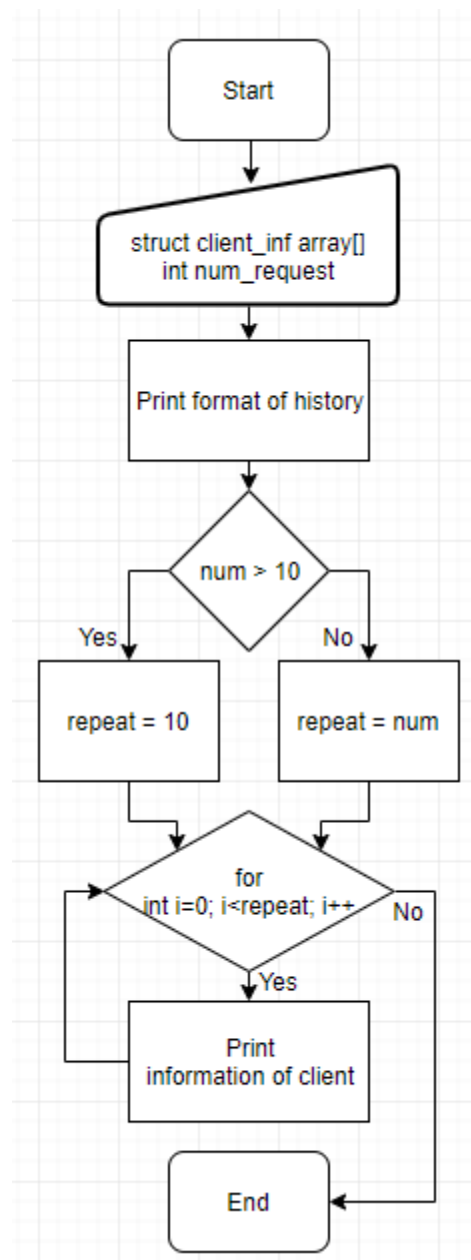
- int html_ls



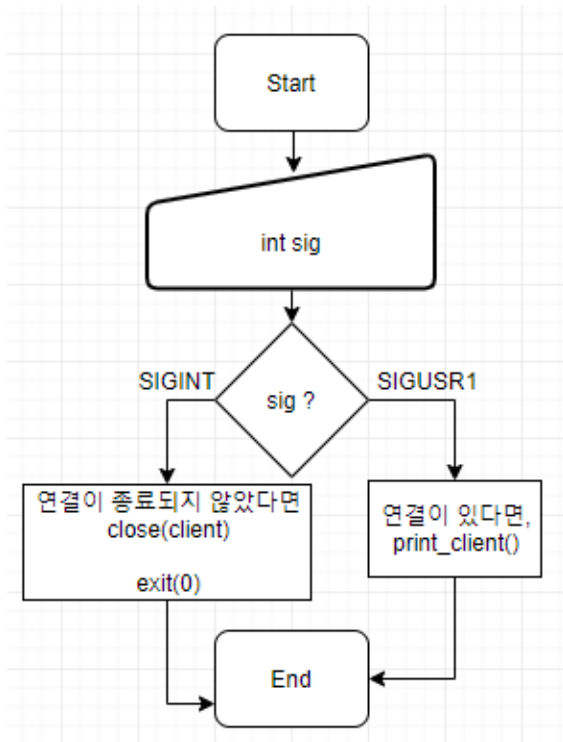
- add_client



- print_client

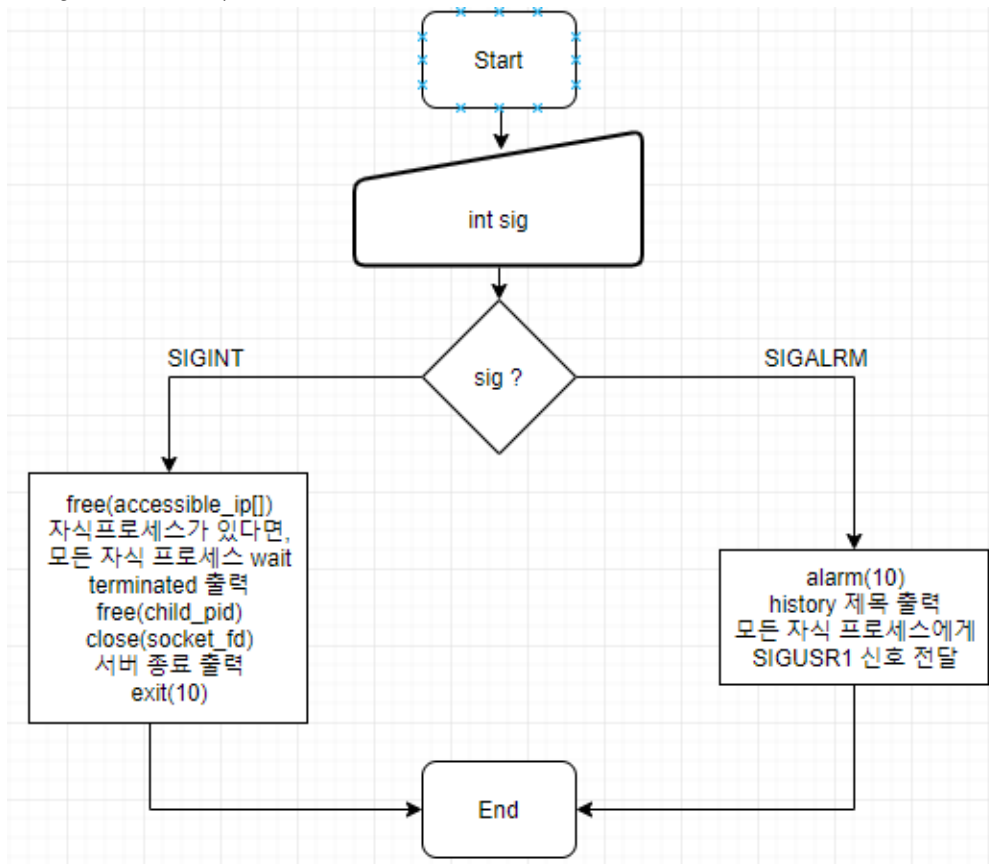


- signal_handler_c



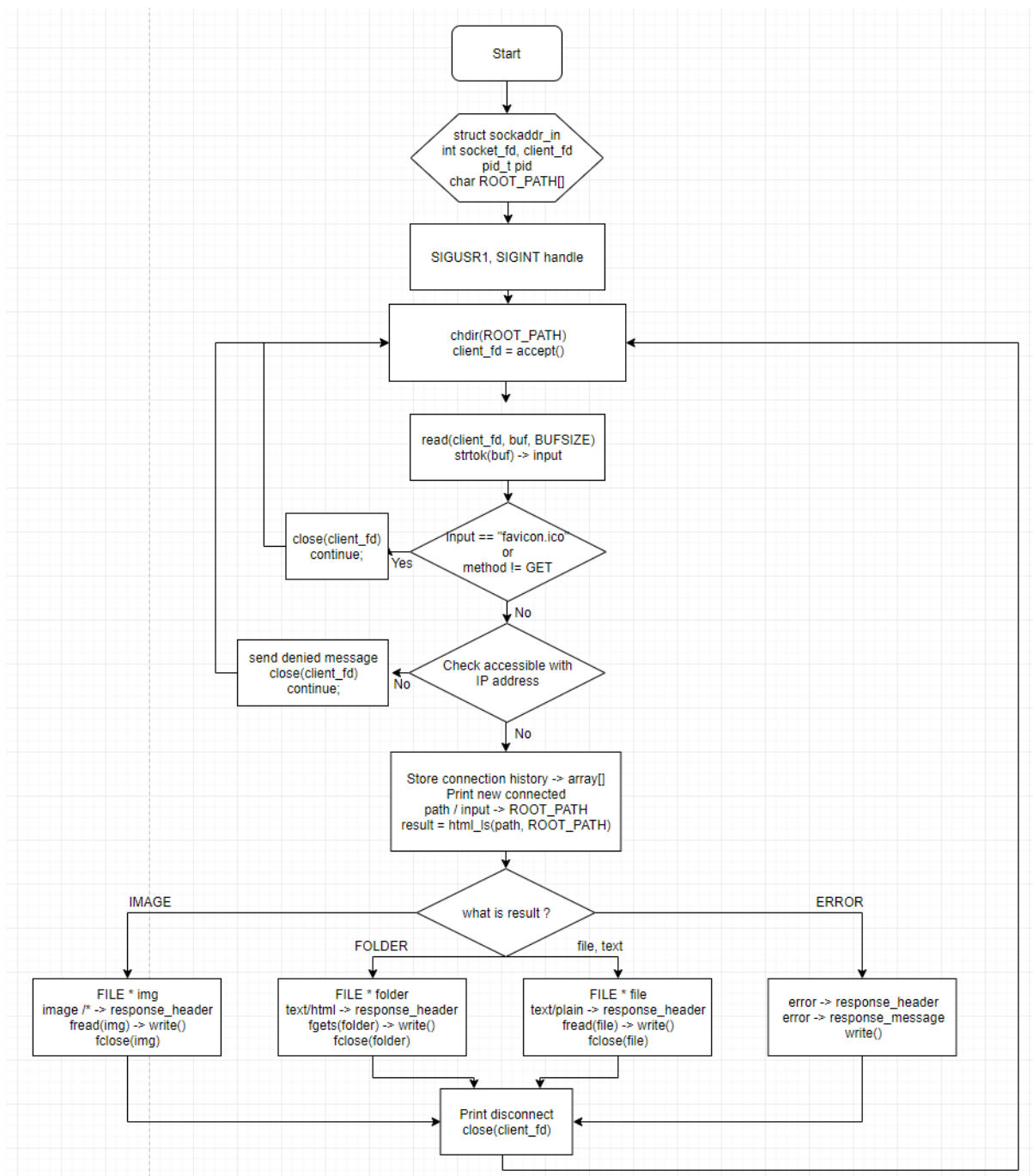
SIGINT -> 무시, 기존의 SIGINT -> SIGUSR2

- signal_handler_p

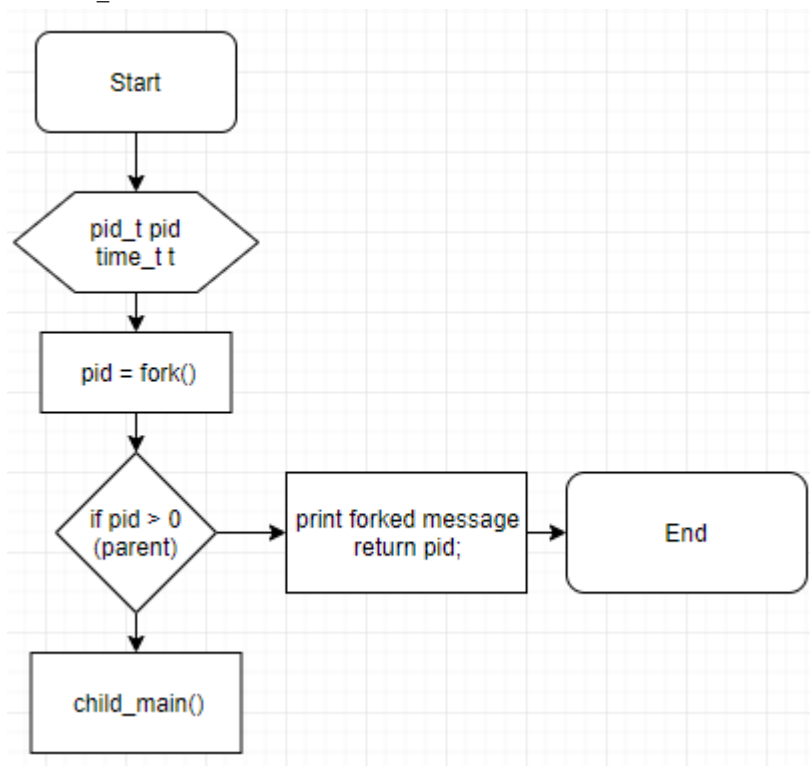


SIGINT 일 경우 wait 이전에 kill(모든 자식, SIGUSR2) 추가

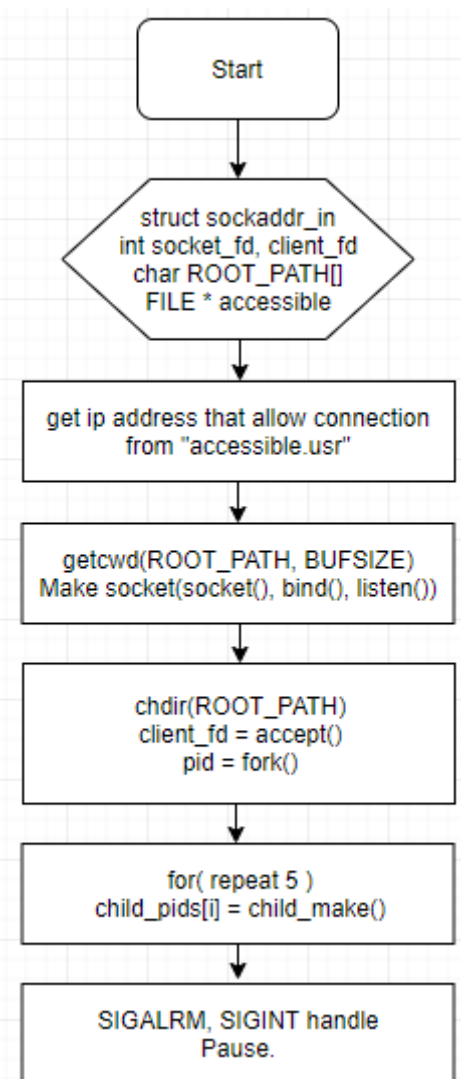
-child_main



- child_make



- int main



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 bubble 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->nlink
    st_uid -> f_stat->uid
    st_gid -> f_stat->gid
    st_size -> f_stat->size
    st_mtime -> f_stat->mtime

```

```

    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 | 옵션이 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 | 옵션이 아니라면
            directory일 경우 파일 이름만 HTML 파일로 출력 (blue)
            아닐 경우 파일 이름만 HTML 파일로 출력 (red)
        else | 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 -l 옵션하고 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
}

```

소켓 생성 -> 연결 준비(socket(), bind(), listen())
자식 프로세스 5개 fork() 하고 자식 프로세스의 pid를 child_pids 배열에 저장

SIGALRM, SIGINT, SIGTSTP 시그널 처리, 10초뒤 SIGALRM 신호 발생

```
Pause;  
}
```

```
void child_main()  
{  
    SIGINT, SIGUSR1, SIGUSR2 신호 처리 선언  
  
    while(1)  
    {  
        ROOT PATH로 디렉토리 변경  
        클라이언트로 부터의 연결을 기다림  
  
        연결이 되면 예외처리 확인(EXIT, favicon.ico)  
        접근 허용된 IP 주소 확인(strcmp, fnmatch)  
  
        연결된 클라이언트 정보 array에 저장  
  
        연결 상태 출력  
        루트 경로가 아닌 경우 루트 경로 / input -> path  
        루트 경로인 경우 루트 경로 -> path  
  
        result에 html_ls(path)의 결과 출력 (폴더:3, 파일:1, 오류:-1)  
  
        if 이미지 파일인 경우  
            이미지 파일 open.  
            헤더 메시지 : image/*  
            응답 메시지에 이미지 파일 내용 전송  
        else if 폴더인 경우  
            html_ls의 결과인 html 파일 open  
            헤더 메시지 : text/html  
            응답 메시지에 html 파일 내용 전송  
        else if 파일인 경우  
            파일 open  
            헤더 메시지 : text/plain  
            응답 메시지에 파일 내용 전송  
        else  
            에러 예외처리 메시지 전송  
  
        연결 종료.  
    }  
}
```

```
pid_t child_make()
```

```

{
    pid = fork() // 자식 프로세스 생성
    부모 프로세스라면
        forked 가 되었다는 메시지 출력
        return pid;
    자식 프로세스라면
        child_main() 함수 실행
}

```

```

void add_client(struct client_inf in_array[], struct client_inf input, int num)
{
    if num > 10
        모든 데이터를 앞으로 한칸씩 땡긴다.
        input 데이터를 9 번째 칸에 저장한다.
    else
        input 데이터를 num-1 번째 칸에 저장한다.
}

```

```

void print_client(struct client_inf in_array[], int num)
{
    if (num > 10)
        repeat = 10
    else
        repeat = num

    repeat 만큼 반복
        array 에 들어있는 연결 정보 출력
}

```

```

void signalHandler_c(int sig)
{
    if sig == SIGINT
        무시
    if sig == SIGUSR1
        연결기록이 있다면 print_client();
    if sig == SIGUSR2
        client 가 종료되지 않았다면 종료
        exit
}

```

```

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

```

```
        printf(terminated)
        close(소켓)
        exit
    if sig == SIGALRM
        10 초뒤 알람 설정
        History 제목 출력
        5 개의 자식프로세스에 SIGUSR1 신호 전달 kill()
}
```

Result

```
sp2015722087@ubuntu:~/sp/shared/server$ ./preforked_server_40302
[Mon May 27 20:22:37 2019] Server is started.
[Mon May 27 20:22:37 2019] 13378 process is forked.
[Mon May 27 20:22:37 2019] 13379 process is forked.
[Mon May 27 20:22:37 2019] 13380 process is forked.
[Mon May 27 20:22:37 2019] 13381 process is forked.
[Mon May 27 20:22:37 2019] 13382 process is forked.
```

- 서버를 실행하고 started 메시지와 forked 메시지의 출력.

```
13377 pts/1    00:00:00 preforked_serve
13378 pts/1    00:00:00 preforked_serve
13379 pts/1    00:00:00 preforked_serve
13380 pts/1    00:00:00 preforked_serve
13381 pts/1    00:00:00 preforked_serve
13382 pts/1    00:00:00 preforked_serve
```

- 터미널에서 ps -e 를 입력 후 자식프로세스가 잘 생성되었나 확인해보았다.

```
===== New Client =====
[Mon May 27 20:24:10 2019]
IP : 127.0.0.1
Port : 36013
=====

===== Disconnected Client =====
[Mon May 27 20:24:10 2019]
IP : 127.0.0.1
Port : 36013
=====

===== Connection History =====
No.      IP          PID      PORT      TIME
1        127.0.0.1    13382    36013     Mon May 27 20:24:10 2019
1        127.0.0.1    13381    30381     Mon May 27 20:24:09 2019
1        127.0.0.1    13380    24749     Mon May 27 20:24:08 2019
```

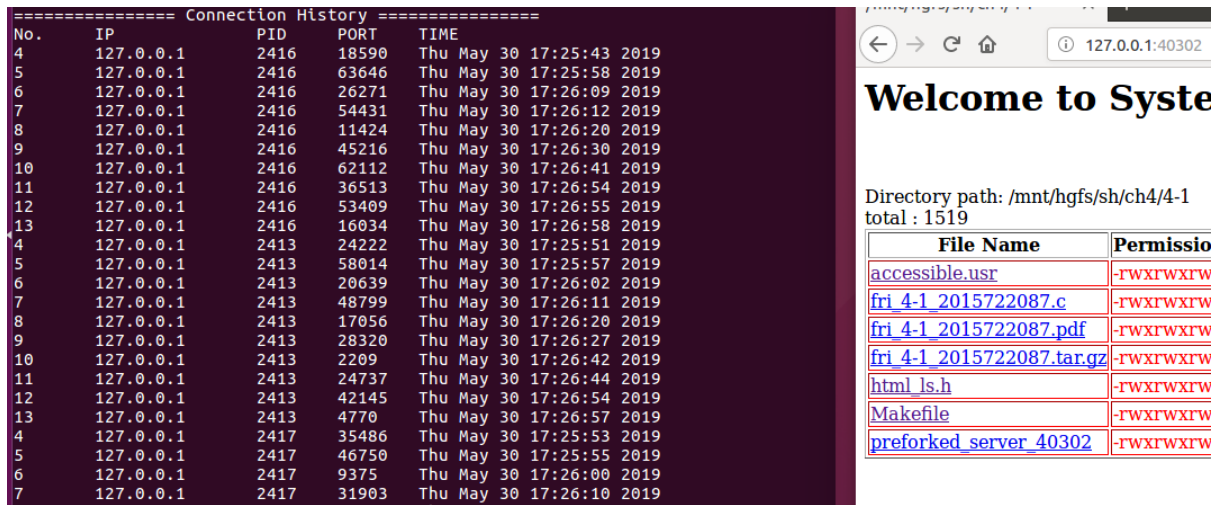
← → ↺ 🏠 ⓘ 127.0.0.1:40302

Welcome to System Program

Directory path: /mnt/hgfs/sh/server
total : 173

File Name	Permission	Link	Owner	Group	Size
accessible usr	-rwxrwxrwx	1	root	root	47
Fri 4-1 2015722087.c	-rwxrwxrwx	1	root	root	164

- 서버에 접속하는 것과, History 출력하는 모습 -> History 의 경우 각 자식 프로세스마다의 기록이 있어 순서 정렬은 신경쓰지 않았다.

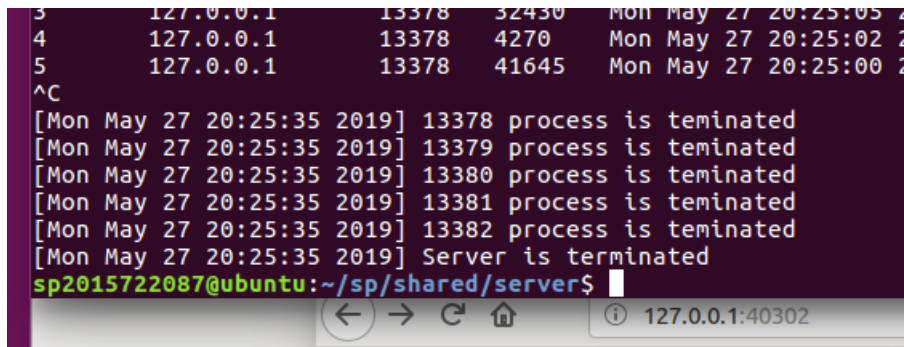


The terminal window displays a 'Connection History' table with columns: No., IP, PID, PORT, and TIME. It lists 26 connections from 127.0.0.1 to the server. The web browser window shows a 'Welcome to Syste' page with a directory path of /mnt/hgfs/sh/ch4/4-1 and a total size of 1519. Below the path is a table of files and their permissions.

File Name	Permissio
accessible usr	-rwxrwxrwx
fri_4-1_2015722087.c	-rwxrwxrwx
fri_4-1_2015722087.pdf	-rwxrwxrwx
fri_4-1_2015722087.tar.gz	-rwxrwxrwx
html ls.h	-rwxrwxrwx
Makefile	-rwxrwxrwx
preforked_server_40302	-rwxrwxrwx

- 여러번의 연결 후 기록 출력의 모습 : 각 자식 프로세스마다 최대 10 개 까지 저장하게 하여, 5 개의 자식 프로세스의 기록을 모두 합친 50 개 까지 출력이 가능하도록 하였다.

(수정) 가장 최근의 연결이 가장 높은 No 를 가지며, No 의 출력은 순서 상관없이 출력하였다.



The terminal window shows a list of processes being terminated. The messages are as follows:

```

[Mon May 27 20:25:35 2019] 13378 process is teminated
[Mon May 27 20:25:35 2019] 13379 process is teminated
[Mon May 27 20:25:35 2019] 13380 process is teminated
[Mon May 27 20:25:35 2019] 13381 process is teminated
[Mon May 27 20:25:35 2019] 13382 process is teminated
[Mon May 27 20:25:35 2019] Server is terminated
sp2015722087@ubuntu:~/sp/shared/server$

```

- 서버를 종료시키는 모습, 컨트롤 + C 의 입력으로 SIGINT 시그널을 부모가 받게 하였으며, 부모는 자식에게 SIGUSR2 신호를 보내면, 자식은 클라이언트의 종료를 확인 후 종료하고, 다시 부모는 waitpid()를 하여 자식의 종료 상태를 확인한 뒤에 terminated 메세지 출력 후 서버 종료 메세지 출력한다.

Conclusion

이번 과제는 지난번의 과제를 기반으로 기능을 추가하는 과제라 처음으로 강의 자료를 받아 보고 과제를 확인하고 나서 걱정이 되지는 않은 과제이다. 코드는 예시 코드와 설명이 주어져있어서 금방 작성하였지만, 자식 프로세스를 생성 후 클라이언트가 접속을 시도하면 어떤 자식 프로세스에게 연결이 되는 것인지가 이해가 되지 않아 코드를 작성하면서도 한참을 생각하였다.

내가 코드로 정해진 것은 없는데, 막상 서버를 실행하면 알아서 겹치지 않도록 자식 프로세스에 연결이 되는 것을 보면서 한편으로는 신기하다는 생각이 들었지만, 다른 한편으로는 어떻게 저렇게 되는건지 알지 못해서 답답하기도 했다.