

2019. 6. 14.

Assignment # 4-3

금요일 - 이성원 교수님

2015722087 컴퓨터정보공학부

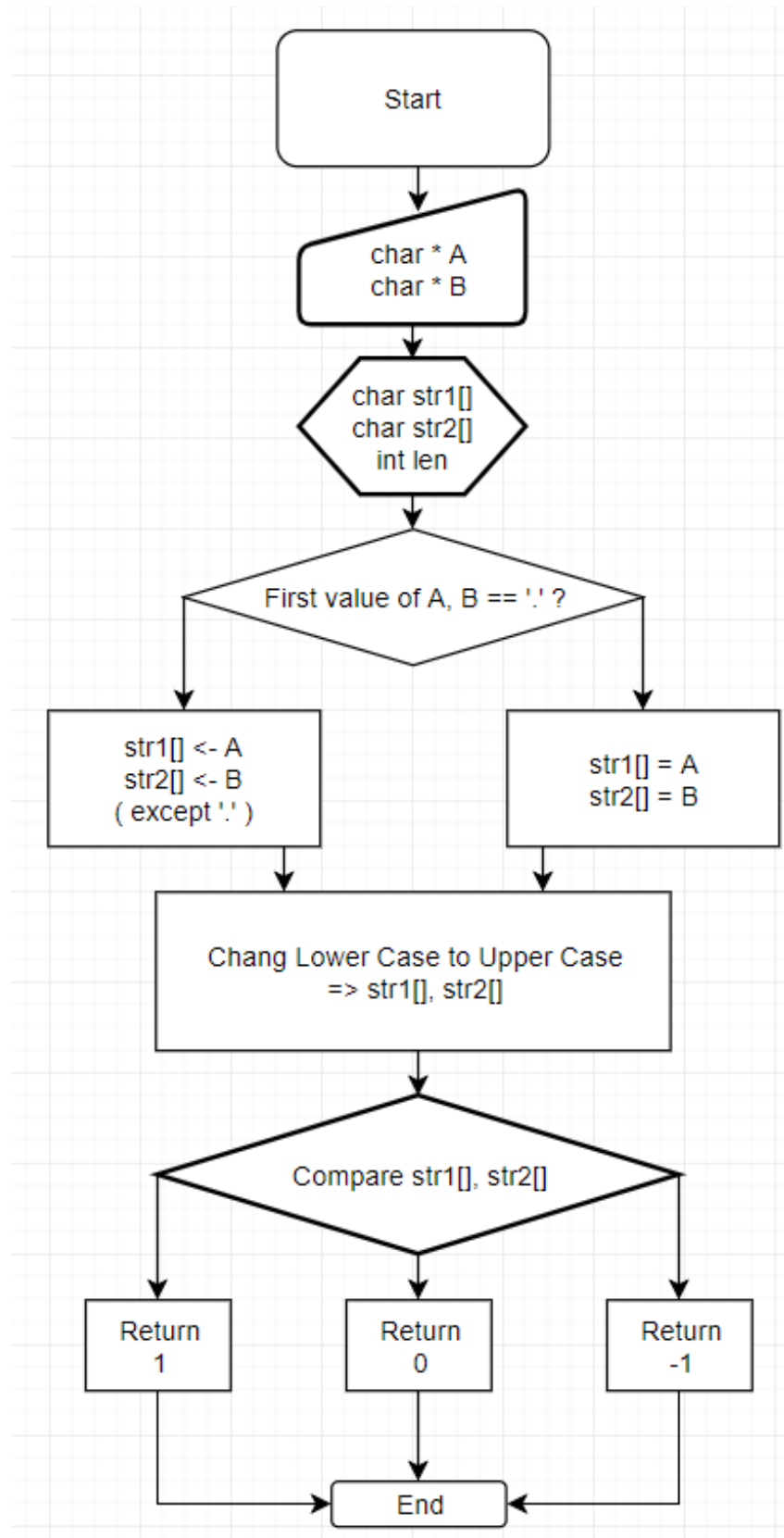
김민철

Introduction

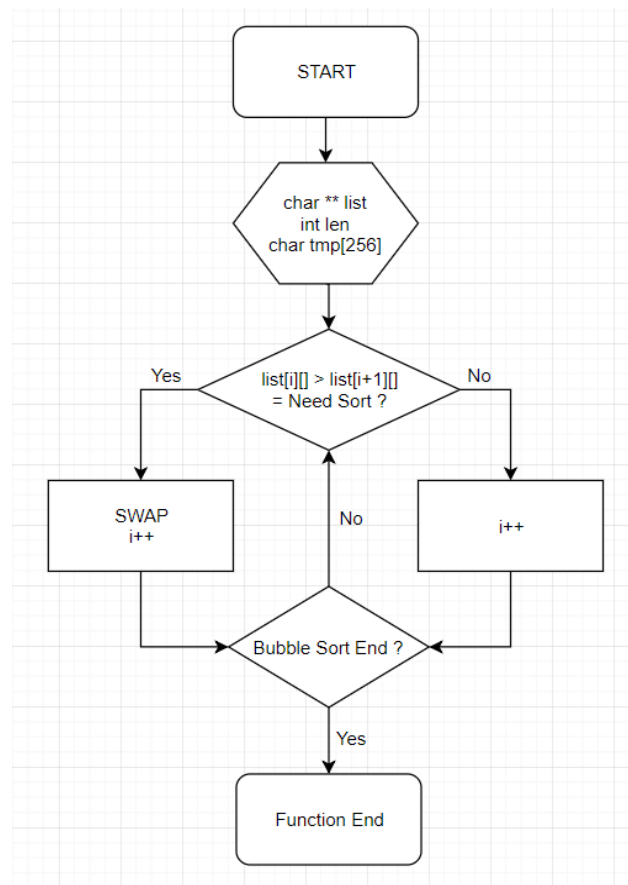
이번 과제에서는 4-2 과제에서 구현하였던 Process_pool_management web-server 에서, 클라이언트가 요청한 URL 경로와 연결이 지속된 시간을 출력하는 것을 추가하고, 콘솔에 출력하는 것을 Log 파일에도 추가로 출력하는 것을 구현한다. Semaphore 를 이용하여 file 에 접근하는 부분을 critical section 으로 하나의 스레드만 접근 가능하도록 보장한다.

Flow Chart

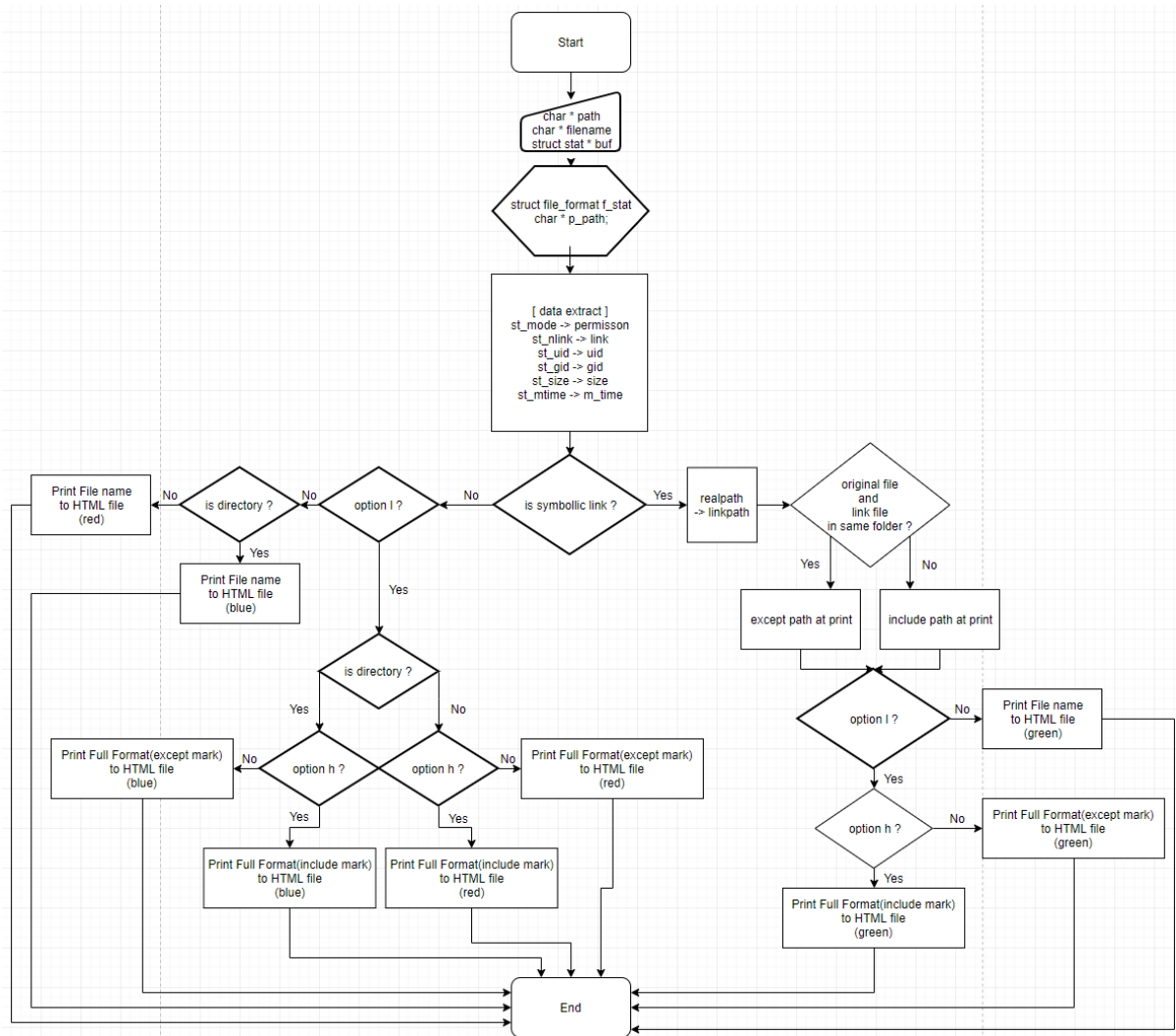
-int strcmp_i



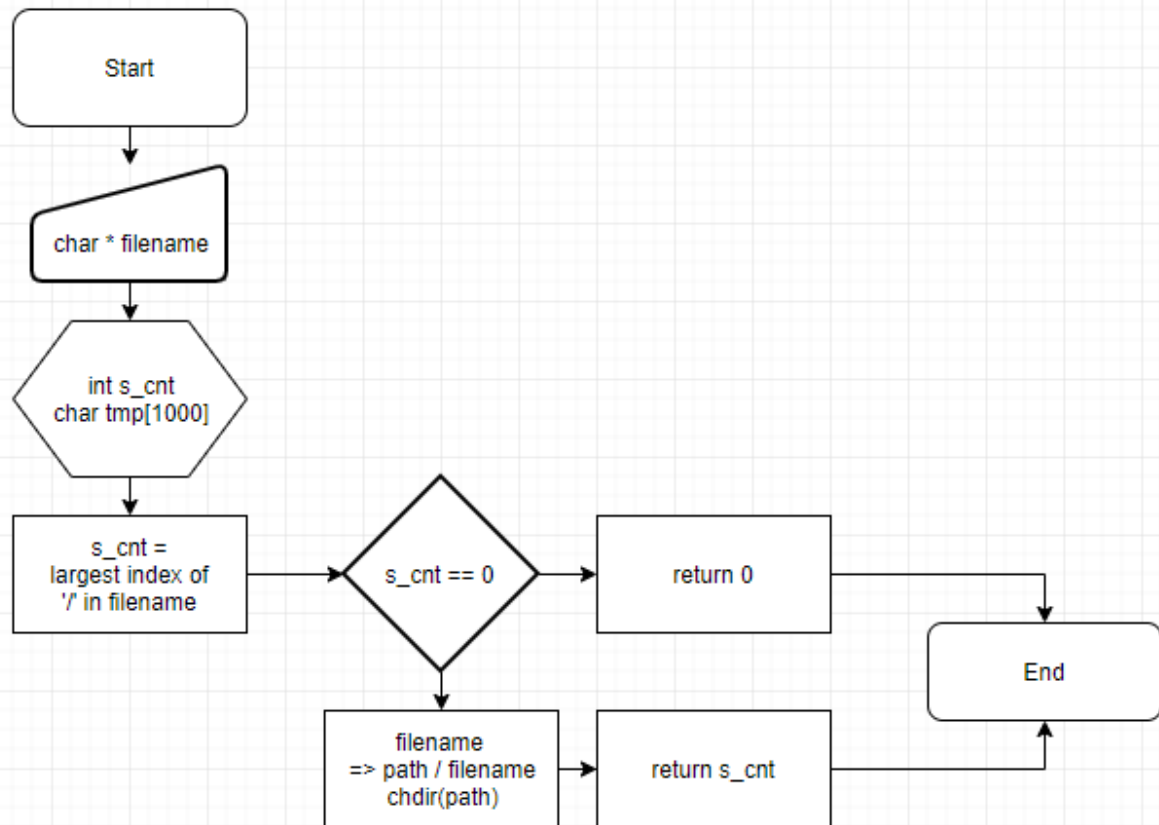
-void sort_list



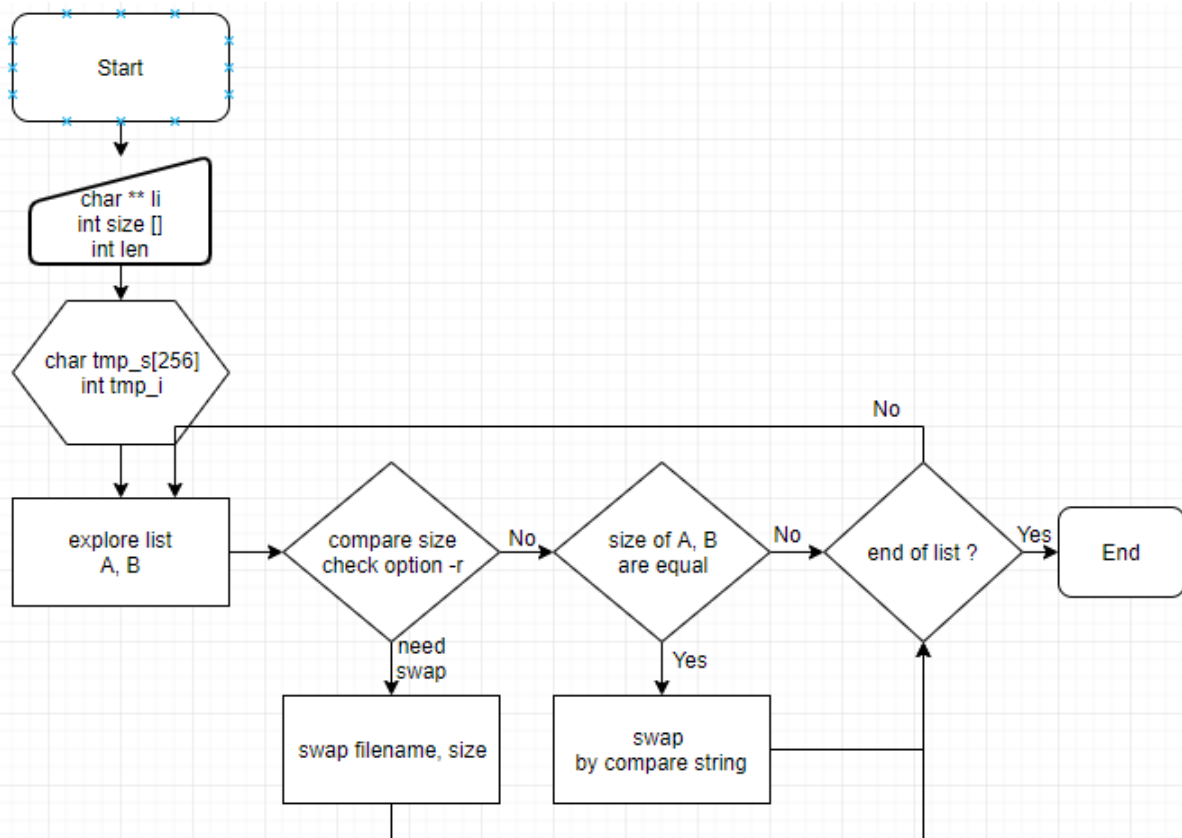
-void view_advanced_list



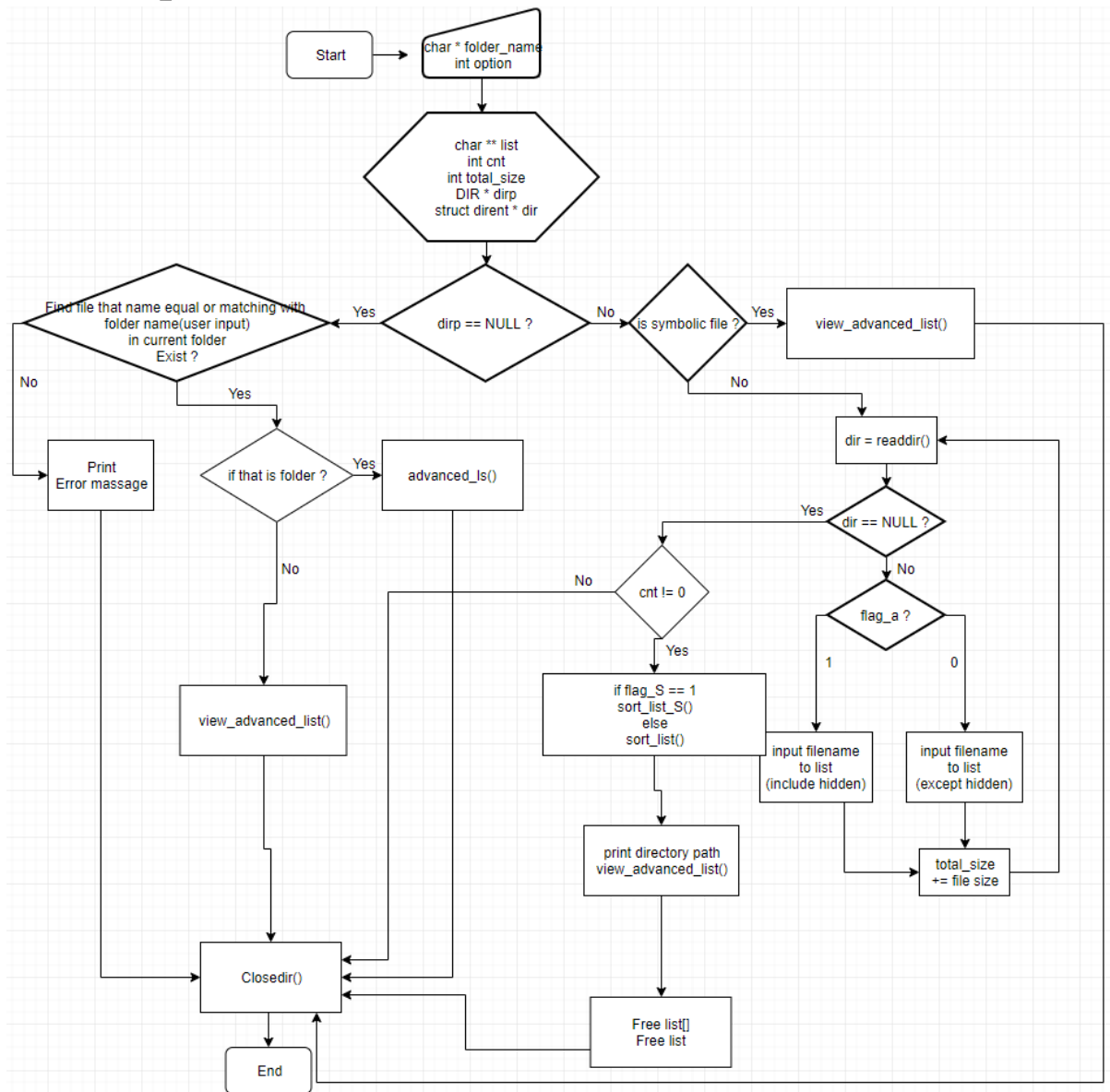
-int check_real_path



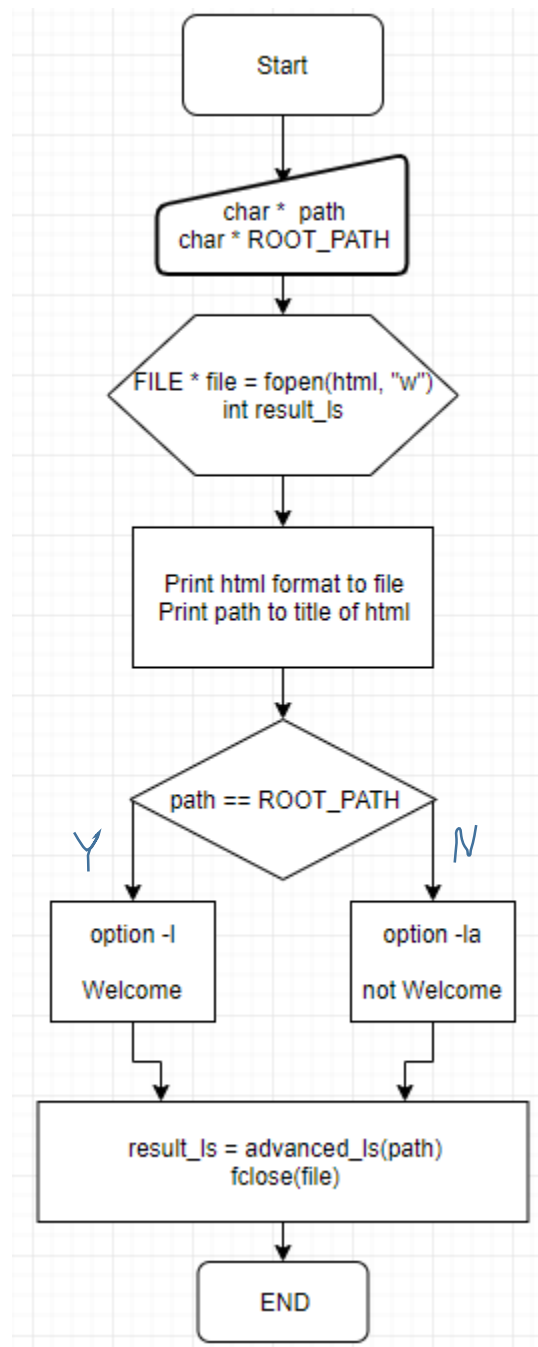
-void sort_list_S



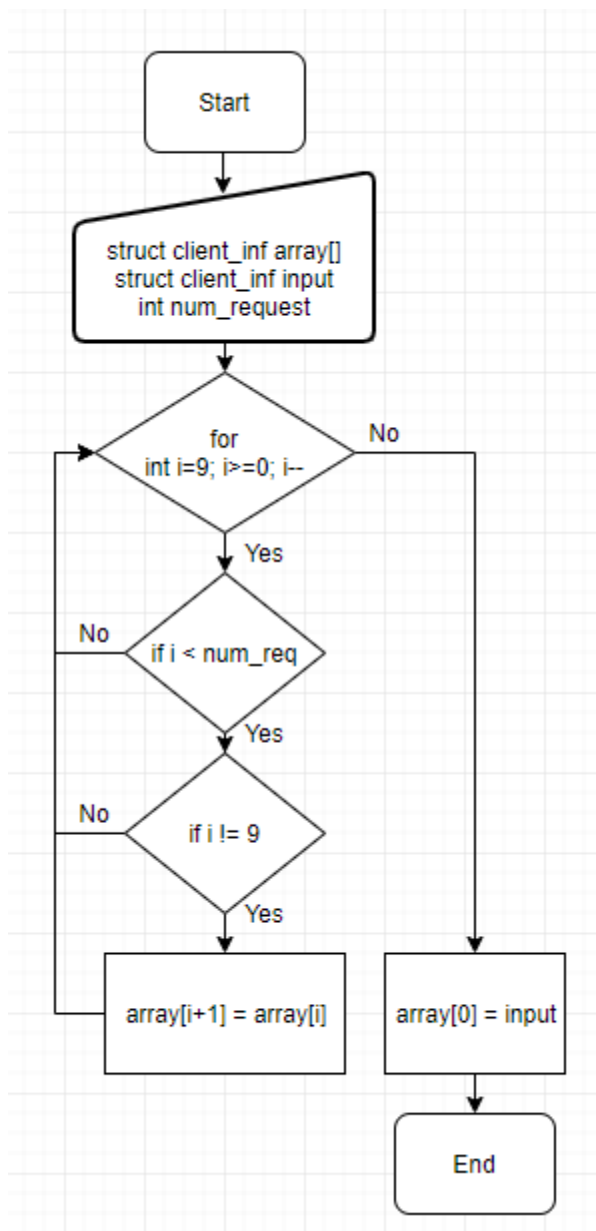
-int advanced_ls



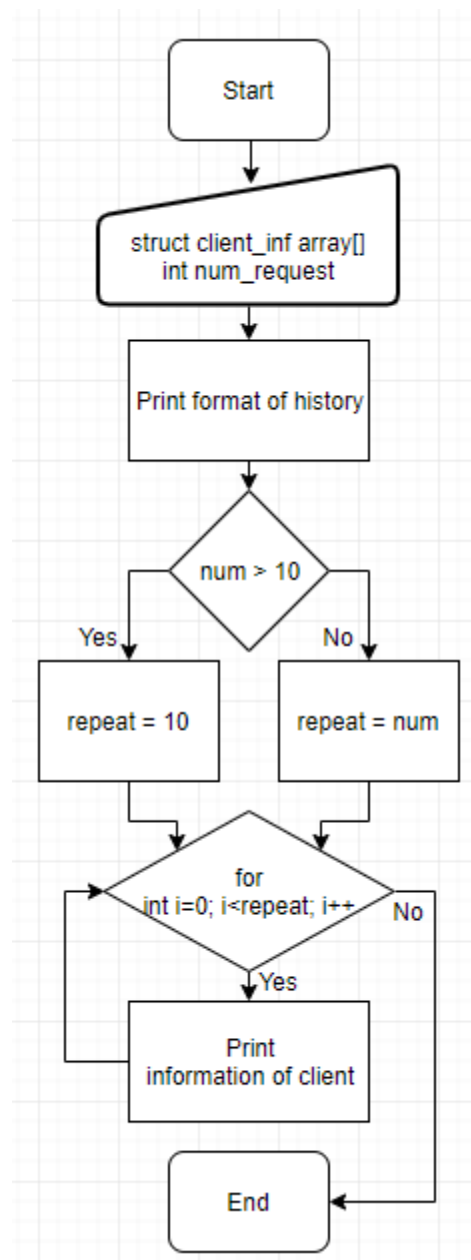
- int html_ls



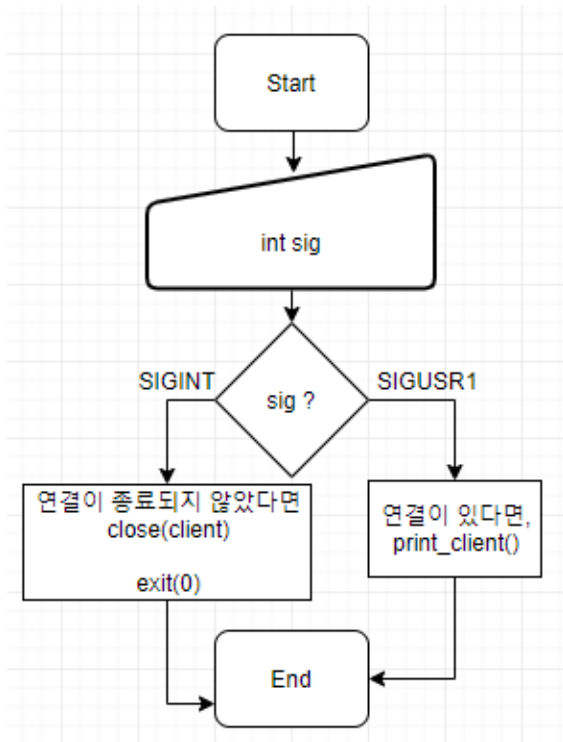
- add_client



- print_client

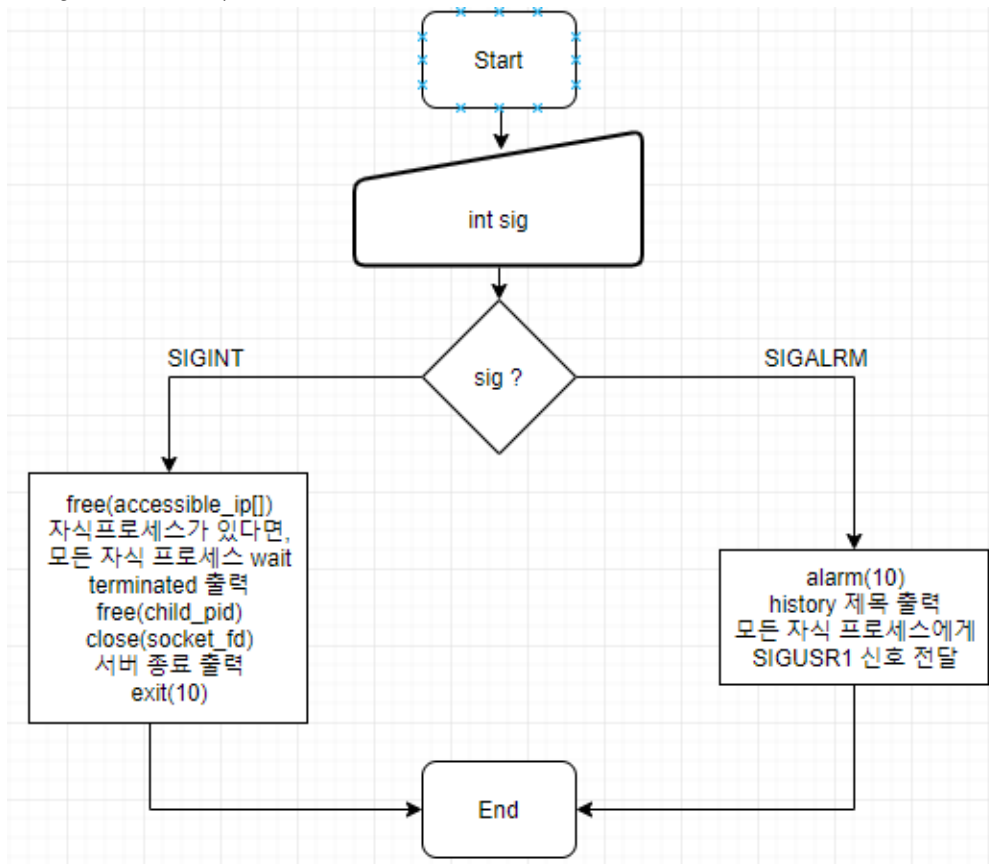


- signal_handler_c



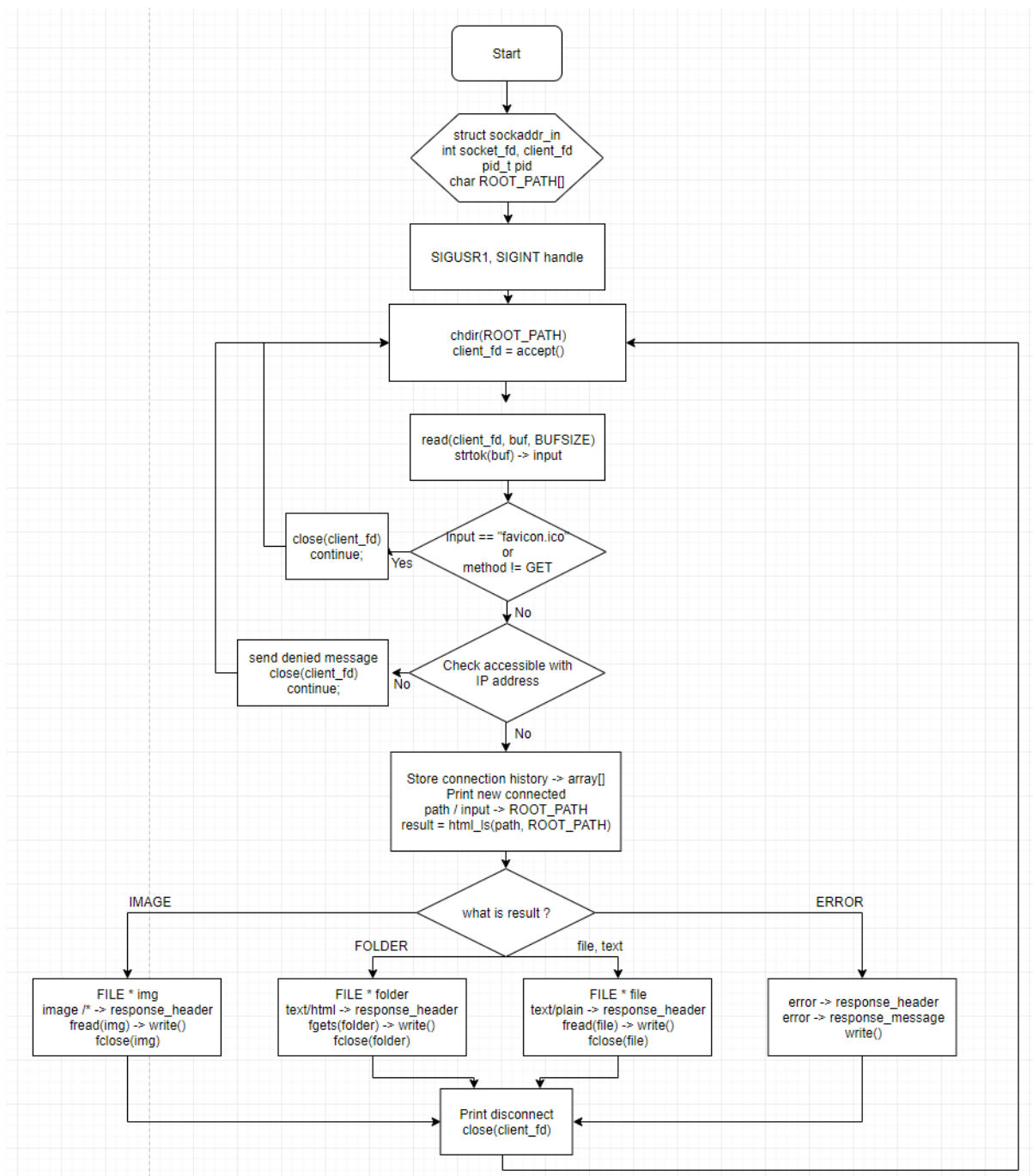
SIGINT -> 무시, 기존의 SIGINT -> SIGUSR2

- signal_handler_p

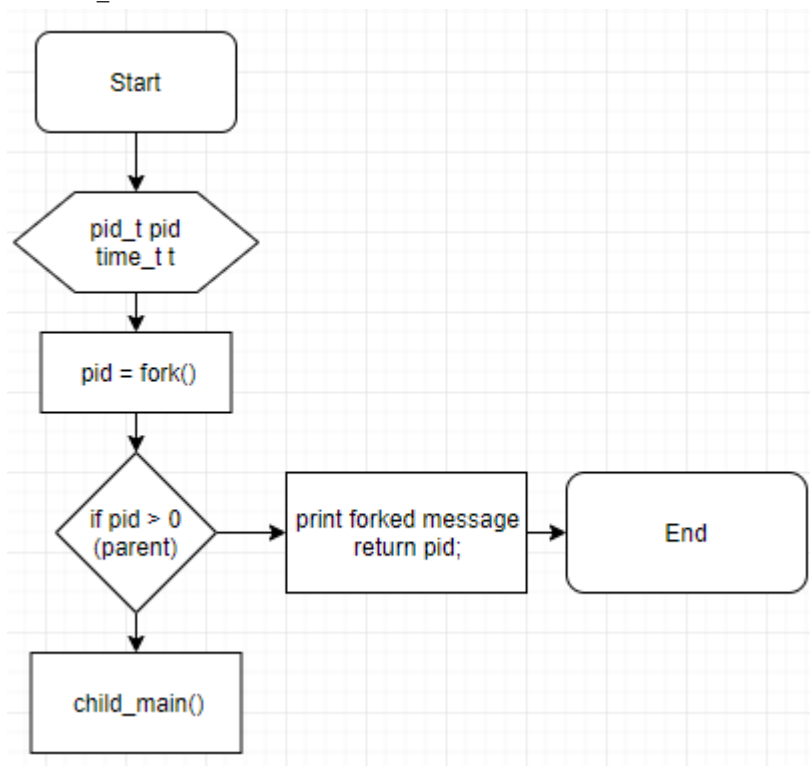


SIGINT 일 경우 wait 이전에 kill(모든 자식, SIGUSR2) 추가

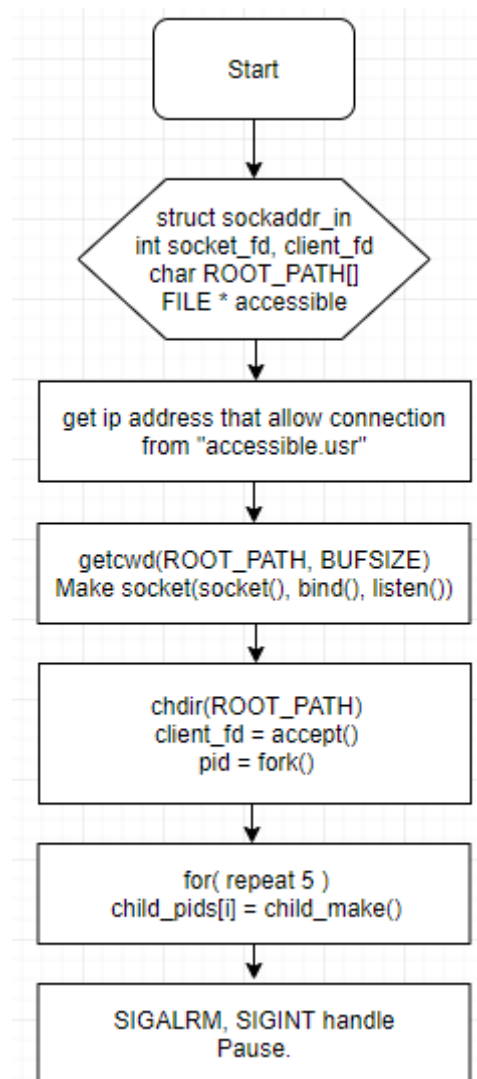
-child_main



- 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
}

```

"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

Semaphore 이름 설정 및 생성

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
연결 메시지를 파일에도 출력하는 스레드 생성
-> 스레드는 log_write() 함수를 사용

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 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

    5 초간 대기
    현재 클라이언트의 연결 끊김을 입력하는 스레드 생성
        -> 스레드는 disconnect_client() 함수를 사용

    연결 종료 메시지 출력
    연결 종료 메시지를 파일에도 출력하는 스레드 생성
        -> 스레드는 log_write() 함수를 사용

    연결 종료.
}
}

```

```

pid_t child_make()
{
    pid = fork() // 자식 프로세스 생성
    부모 프로세스라면
        forked가 되었다는 메시지 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
}
```

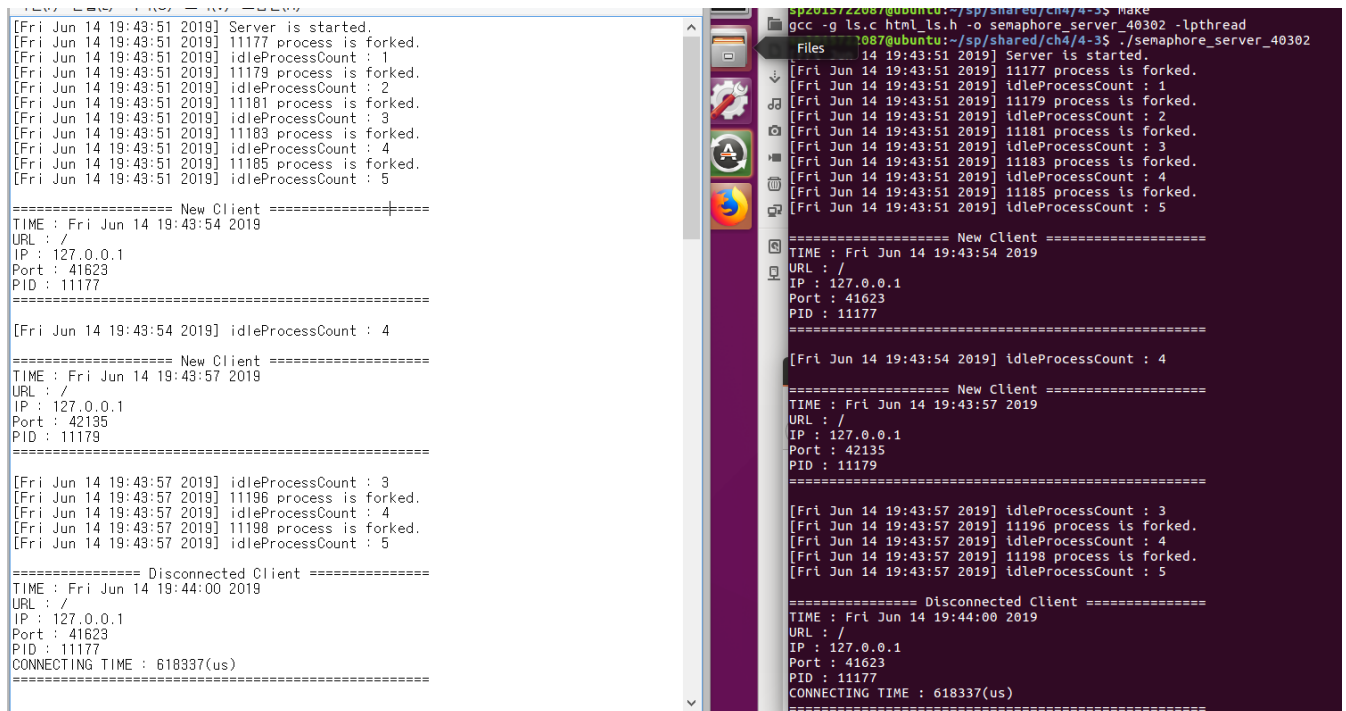
```

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
}

void log_write
{
    semaphore 오픈(name=40302) 후, sem_wait 선언하여 다른 스레드의 접근 차단,
    LOG 파일 오픈 후 인자로 입력된 문자열을 파일에 출력
    LOG 파일 fclose
    sem_post 선언하여 다른 스레드의 접근 허용
    sem_close
    return'
}

```

Result



```
[Fri Jun 14 19:43:51 2019] Server is started.
[Fri Jun 14 19:43:51 2019] 11177 process is forked.
[Fri Jun 14 19:43:51 2019] idleProcessCount : 1
[Fri Jun 14 19:43:51 2019] 11179 process is forked.
[Fri Jun 14 19:43:51 2019] idleProcessCount : 2
[Fri Jun 14 19:43:51 2019] 11181 process is forked.
[Fri Jun 14 19:43:51 2019] idleProcessCount : 3
[Fri Jun 14 19:43:51 2019] 11183 process is forked.
[Fri Jun 14 19:43:51 2019] idleProcessCount : 4
[Fri Jun 14 19:43:51 2019] 11185 process is forked.
[Fri Jun 14 19:43:51 2019] idleProcessCount : 5

===== New Client =====
TIME : Fri Jun 14 19:43:54 2019
URL : /
IP : 127.0.0.1
Port : 41623
PID : 11177

[Fri Jun 14 19:43:54 2019] idleProcessCount : 4

===== New Client =====
TIME : Fri Jun 14 19:43:57 2019
URL : /
IP : 127.0.0.1
Port : 42135
PID : 11179

[Fri Jun 14 19:43:57 2019] idleProcessCount : 3
[Fri Jun 14 19:43:57 2019] 11196 process is forked.
[Fri Jun 14 19:43:57 2019] idleProcessCount : 4
[Fri Jun 14 19:43:57 2019] 11198 process is forked.
[Fri Jun 14 19:43:57 2019] idleProcessCount : 5

===== Disconnected Client =====
TIME : Fri Jun 14 19:44:00 2019
URL : /
IP : 127.0.0.1
Port : 41623
PID : 11177
CONNECTING TIME : 618337(us)

===== New Client =====
TIME : Fri Jun 14 19:43:54 2019
URL : /
IP : 127.0.0.1
Port : 41623
PID : 11177

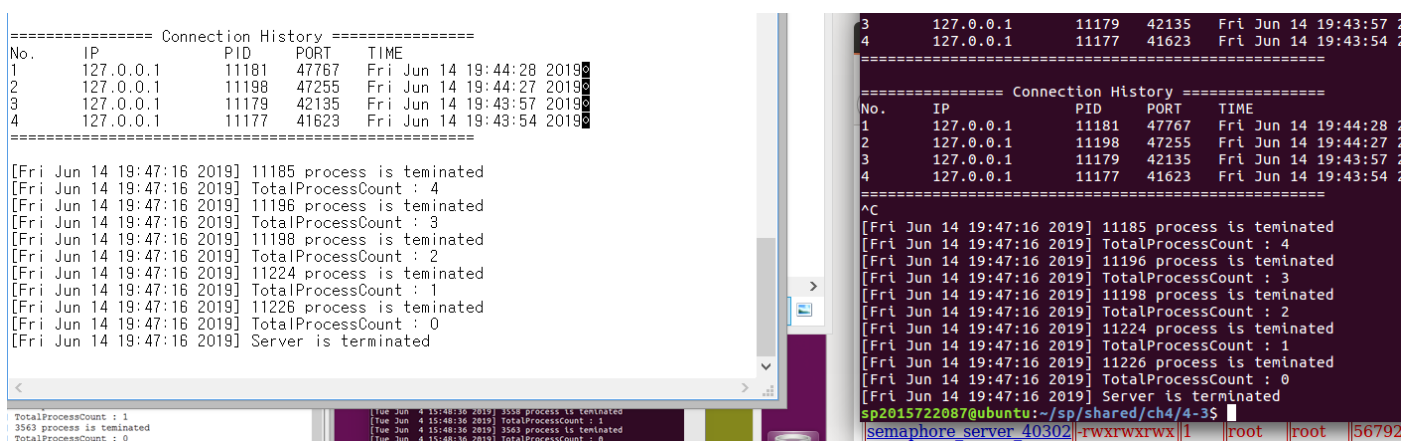
[Fri Jun 14 19:43:54 2019] idleProcessCount : 4

===== New Client =====
TIME : Fri Jun 14 19:43:57 2019
URL : /
IP : 127.0.0.1
Port : 42135
PID : 11179

[Fri Jun 14 19:43:57 2019] idleProcessCount : 3
[Fri Jun 14 19:43:57 2019] 11196 process is forked.
[Fri Jun 14 19:43:57 2019] idleProcessCount : 4
[Fri Jun 14 19:43:57 2019] 11198 process is forked.
[Fri Jun 14 19:43:57 2019] idleProcessCount : 5

===== Disconnected Client =====
TIME : Fri Jun 14 19:44:00 2019
URL : /
IP : 127.0.0.1
Port : 41623
PID : 11177
CONNECTING TIME : 618337(us)
```

- 서버를 실행하고 started 메시지와 forked 메시지의 출력, 그리고 동시에 Log 텍스트 파일에도 동일하게 작성되는 모습이다. 이번 과제에서는 추가로 클라이언트 연결, 연결 종료시에 URL 과 CONNECTING TIME 을 추가로 출력하였는데, 연결 지속 시간은 마이크로초 단위이며, gettimeofday() 함수를 이용하여 5 초간 sleep 을 제외한 시간을 출력하였다.



```
===== Connection History =====
No.  IP      PID    PORT  TIME
1    127.0.0.1 11181  47767 Fri Jun 14 19:44:28 2019
2    127.0.0.1 11198  47255 Fri Jun 14 19:44:27 2019
3    127.0.0.1 11179  42135 Fri Jun 14 19:43:57 2019
4    127.0.0.1 11177  41623 Fri Jun 14 19:43:54 2019

[Fri Jun 14 19:47:16 2019] 11185 process is terminated
[Fri Jun 14 19:47:16 2019] TotalProcessCount : 4
[Fri Jun 14 19:47:16 2019] 11196 process is terminated
[Fri Jun 14 19:47:16 2019] TotalProcessCount : 3
[Fri Jun 14 19:47:16 2019] 11198 process is terminated
[Fri Jun 14 19:47:16 2019] TotalProcessCount : 2
[Fri Jun 14 19:47:16 2019] 11224 process is terminated
[Fri Jun 14 19:47:16 2019] TotalProcessCount : 1
[Fri Jun 14 19:47:16 2019] 11226 process is terminated
[Fri Jun 14 19:47:16 2019] TotalProcessCount : 0
[Fri Jun 14 19:47:16 2019] Server is terminated

TotalProcessCount : 1
3563 process is terminated
TotalProcessCount : 0

===== Connection History =====
No.  IP      PID    PORT  TIME
1    127.0.0.1 11181  47767 Fri Jun 14 19:44:28 2019
2    127.0.0.1 11198  47255 Fri Jun 14 19:44:27 2019
3    127.0.0.1 11179  42135 Fri Jun 14 19:43:57 2019
4    127.0.0.1 11177  41623 Fri Jun 14 19:43:54 2019

[Fri Jun 14 19:47:16 2019] 11185 process is terminated
[Fri Jun 14 19:47:16 2019] TotalProcessCount : 4
[Fri Jun 14 19:47:16 2019] 11196 process is terminated
[Fri Jun 14 19:47:16 2019] TotalProcessCount : 3
[Fri Jun 14 19:47:16 2019] 11198 process is terminated
[Fri Jun 14 19:47:16 2019] TotalProcessCount : 2
[Fri Jun 14 19:47:16 2019] 11224 process is terminated
[Fri Jun 14 19:47:16 2019] TotalProcessCount : 1
[Fri Jun 14 19:47:16 2019] 11226 process is terminated
[Fri Jun 14 19:47:16 2019] TotalProcessCount : 0
[Fri Jun 14 19:47:16 2019] Server is terminated

sp2015722087@ubuntu:~/sp/shared/ch4/4-3$ semaphore_server_40302 -rw-rw-rw- 1 root root 56792
```

- 서버가 종료되는 순간까지도 모든 콘솔에 출력되는 메시지를 Log 텍스트 파일에도 출력한 모습이다.

Conclusion

이번 과제는 지난번의 과제에서 두가지의 출력요소를 추가하고, 콘솔에 출력되는 모든 메시지를 Log 텍스트 파일에도 작성하는 웹서버를 구현하는 과제이다. 텍스트 파일에 출력하는 과제는 이전에도 경험한적이 있어서 어렵지 않았지만, 마이크로초 단위를 사용하는 것은 해본적이 없어 처음에는 어떻게 마이크로 단위까지의 시간을 계산할 수 있을지 고민하는 것이 가장 큰 문제였다.

결국 `gettimeofday` 와 `struct timeval` 구조체를 이용하여 해결하였다. 연결 시작시의 시간을 기록하고, 연결 종료시의 시간을 기록하여 두 시간의 차이가 연결 지속시간이므로 그것을 출력함으로써 과제를 해결하게되었다.