

시스템프로그래밍

과제이름: Assignment 3-1

- 담당교수: 김 태 석 교수님
- 학 과: 컴퓨터정보공학부
- 학 번: 2019202021
- 이 름: 정 성 업
- 제출일: 2023/5/15

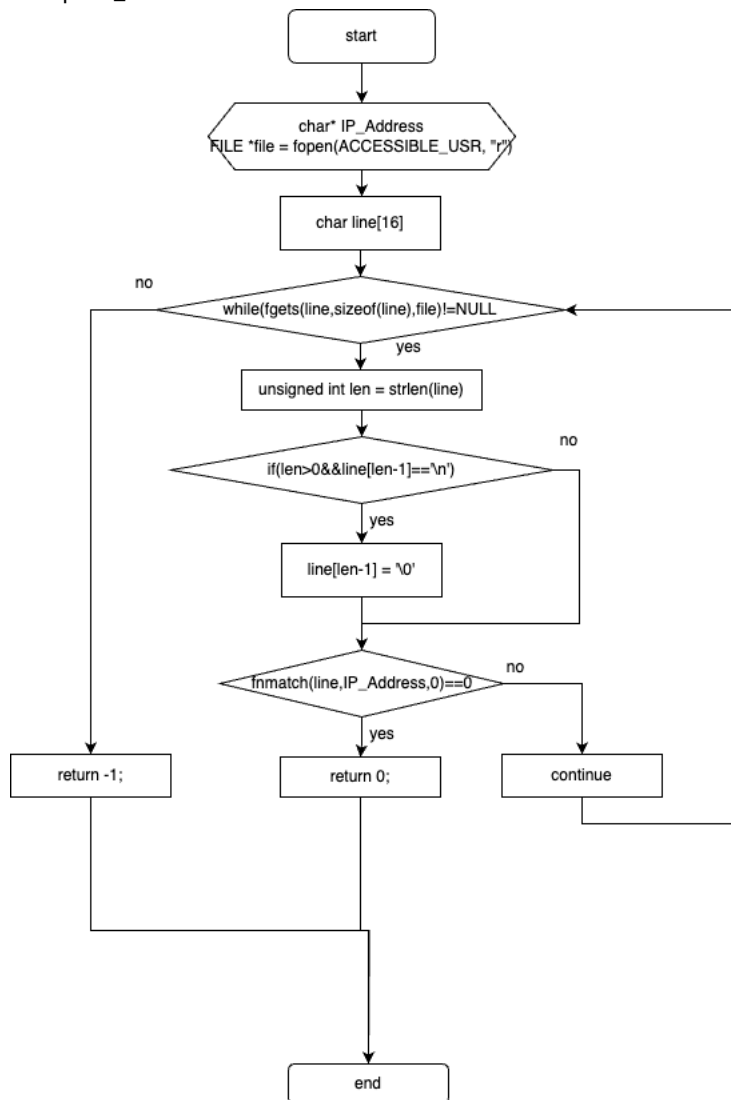
1. Introduction

A. 과제 소개

이번 Assignment 3-1 과제는 이전 Assignment 2-3에서 설계하였던 리눅스 환경에서의 C언어 기반 다중 접속과 접근 제어 지원 socket server를 5개의 child process를 미리 fork 하여 선언하고 후에 5개 프로세스에서 돌아가며 처리하도록 설계하였다. 이전과 다르게 connection history는 parent process에서는 format만 출력하고 연결 정보는 child process에서 출력한다.

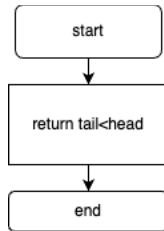
2. Flow chart

A. compare_WhiteList



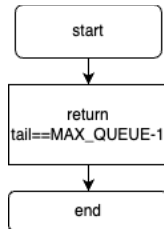
accessible.usr에 저장된 whitelist에 대한 정보를 읽고 패턴을 비교하여 일치하는 경우 0을 반환하고 일치하지 않는 경우 -1을 반환한다.

B. is_empty



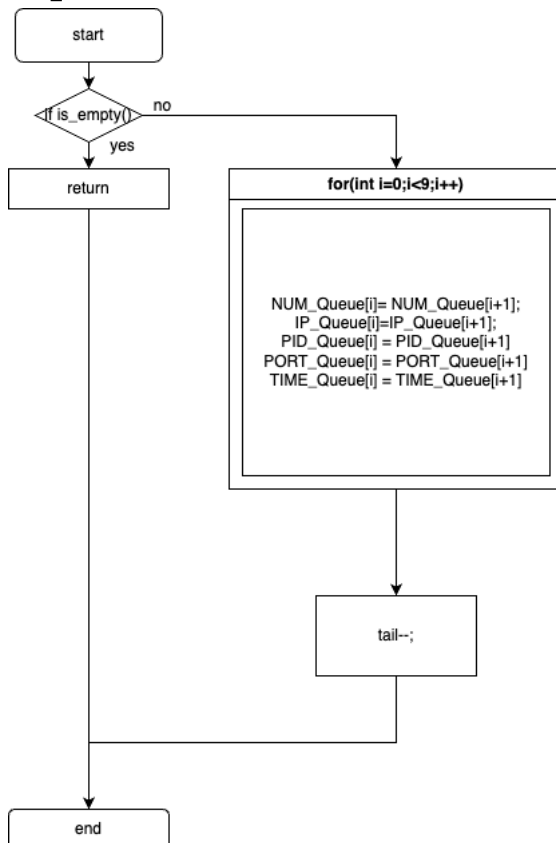
Queue가 비어 있는지 확인하는 함수이다.

C. is_full



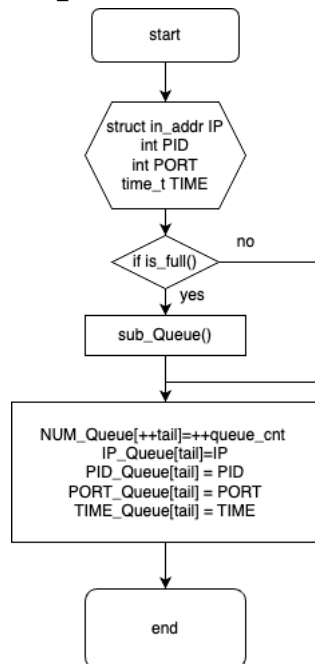
Queue가 가득 찼는지 확인하는 함수이다.

D. sub_queue



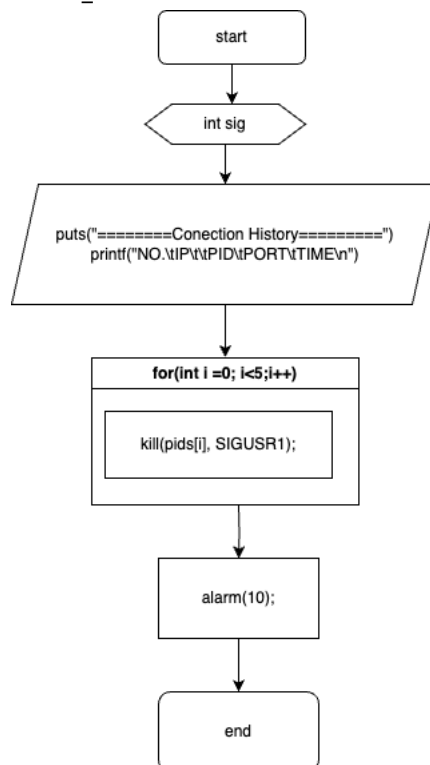
Queue의 첫 번째 값을 제거하는 함수이다.

E. add_Queue



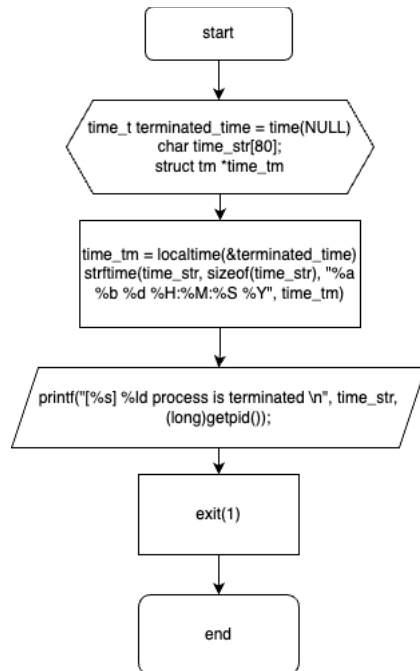
새로운 클라이언트 연결 시 queue에 연결정보를 저장하는 함수이다.

F. Alarm_handler



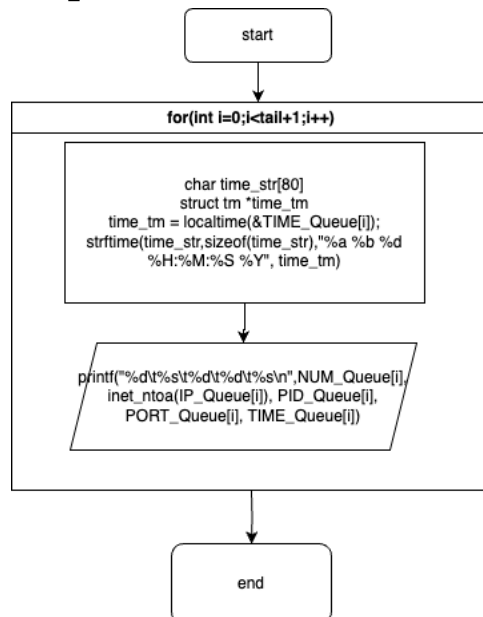
alarm이 10초가 되었을 때 작동하는 함수로 connection history format을 출력하고 자식 프로세스에게 SIGUSR1 시그널을 보낸다.

G. exit_handler



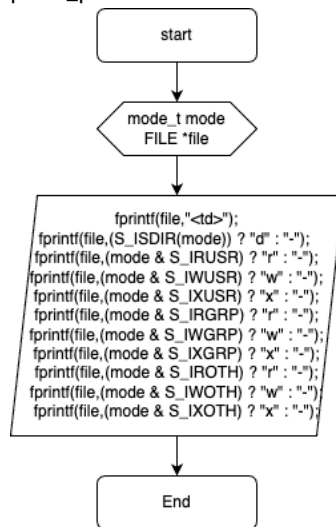
SIGINT 시그널이 발생했을 때 처리하는 함수로 종료되는 자식 프로세스 정보를 출력 후 종료한다.

H. child_handler



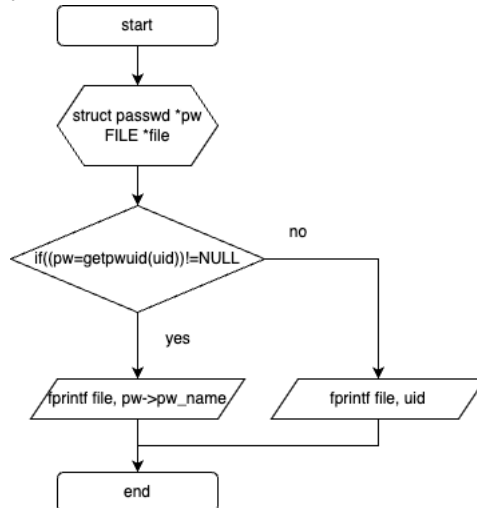
SIGUSR1의 시그널을 받았을 때 처리하는 함수로 connection history를 출력한다.

I. print_permission



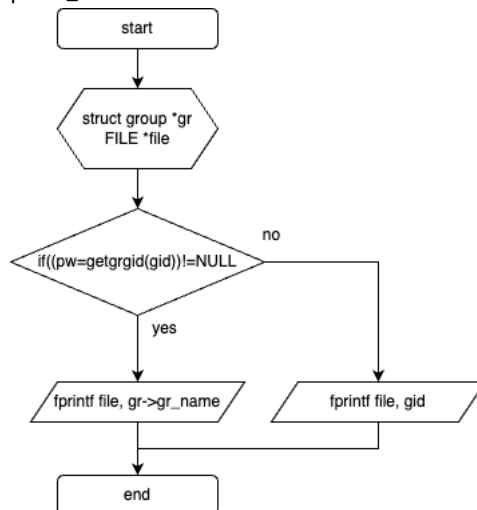
파일 허용정보를 출력하는 함수이다.

J. print_UID



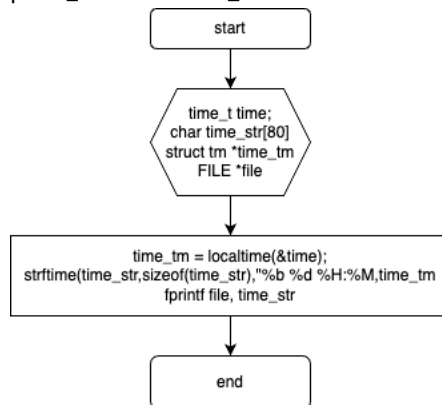
파일의 UID 정보를 출력하는 함수이다.

K. print_GID



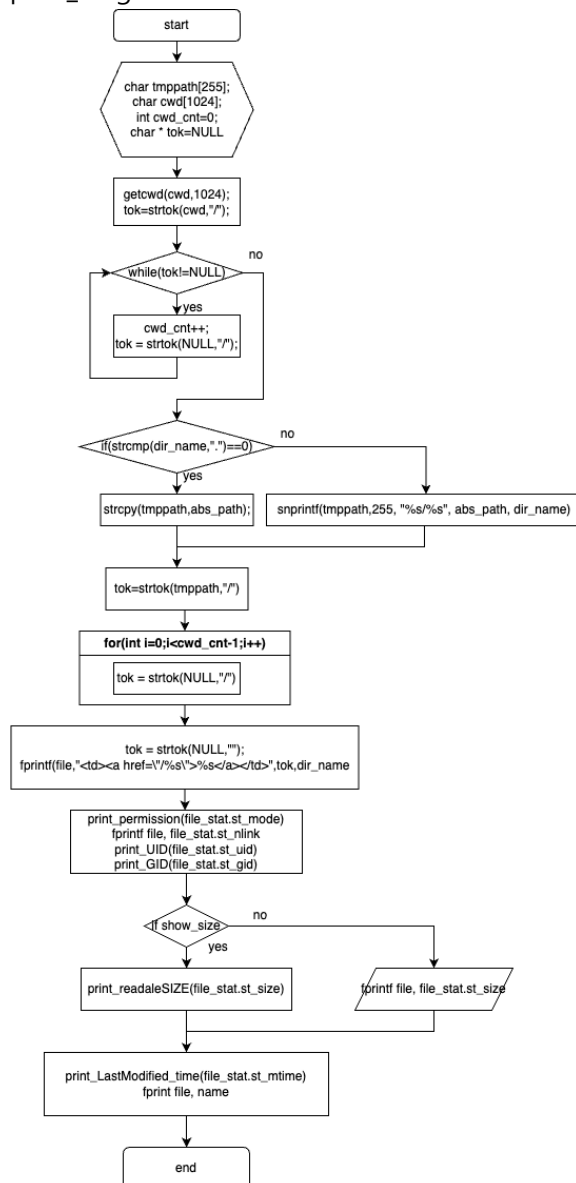
파일의 GID 정보를 출력하는 함수이다.

L. print_lastModified_time



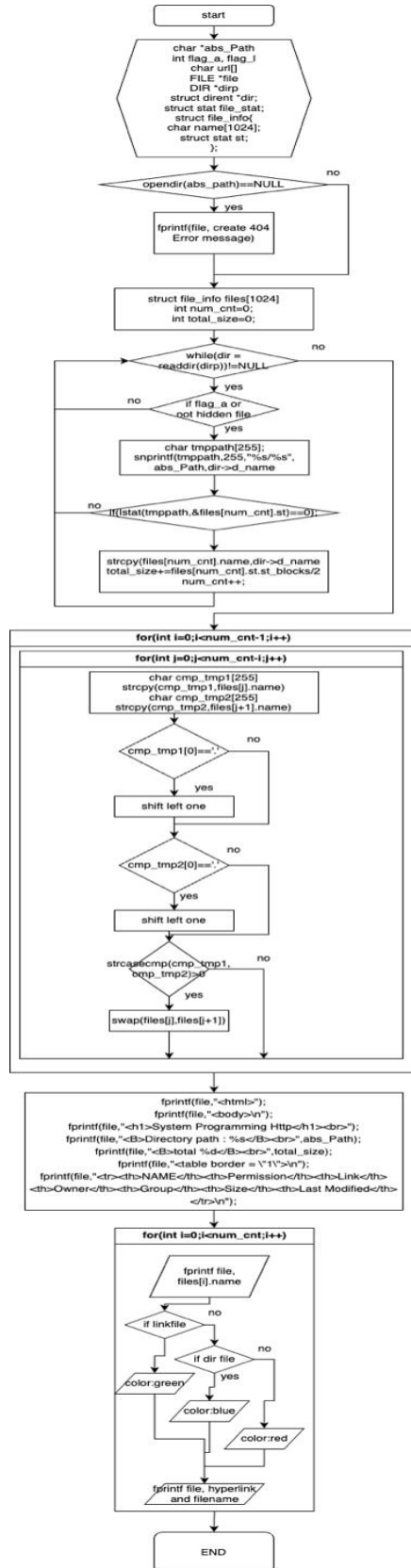
파일의 마지막 수정일자와 시간을 출력하는 함수이다.

M. print_long



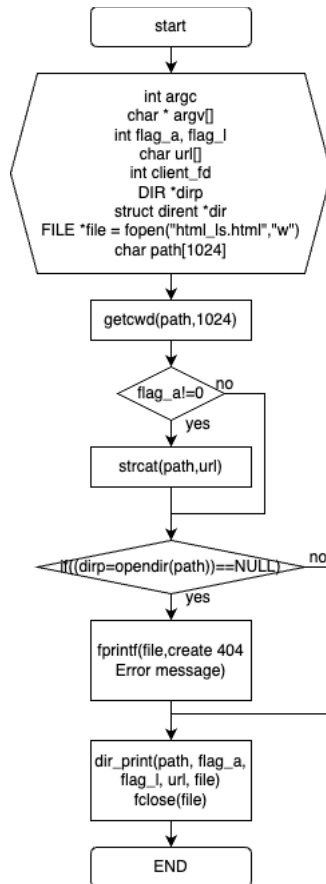
파일의 정보를 long format에 맞춰서 출력하는 함수이다. 현재 working directory와 url 정보를 합쳐서 파일에 접근한다.

N. dir_print



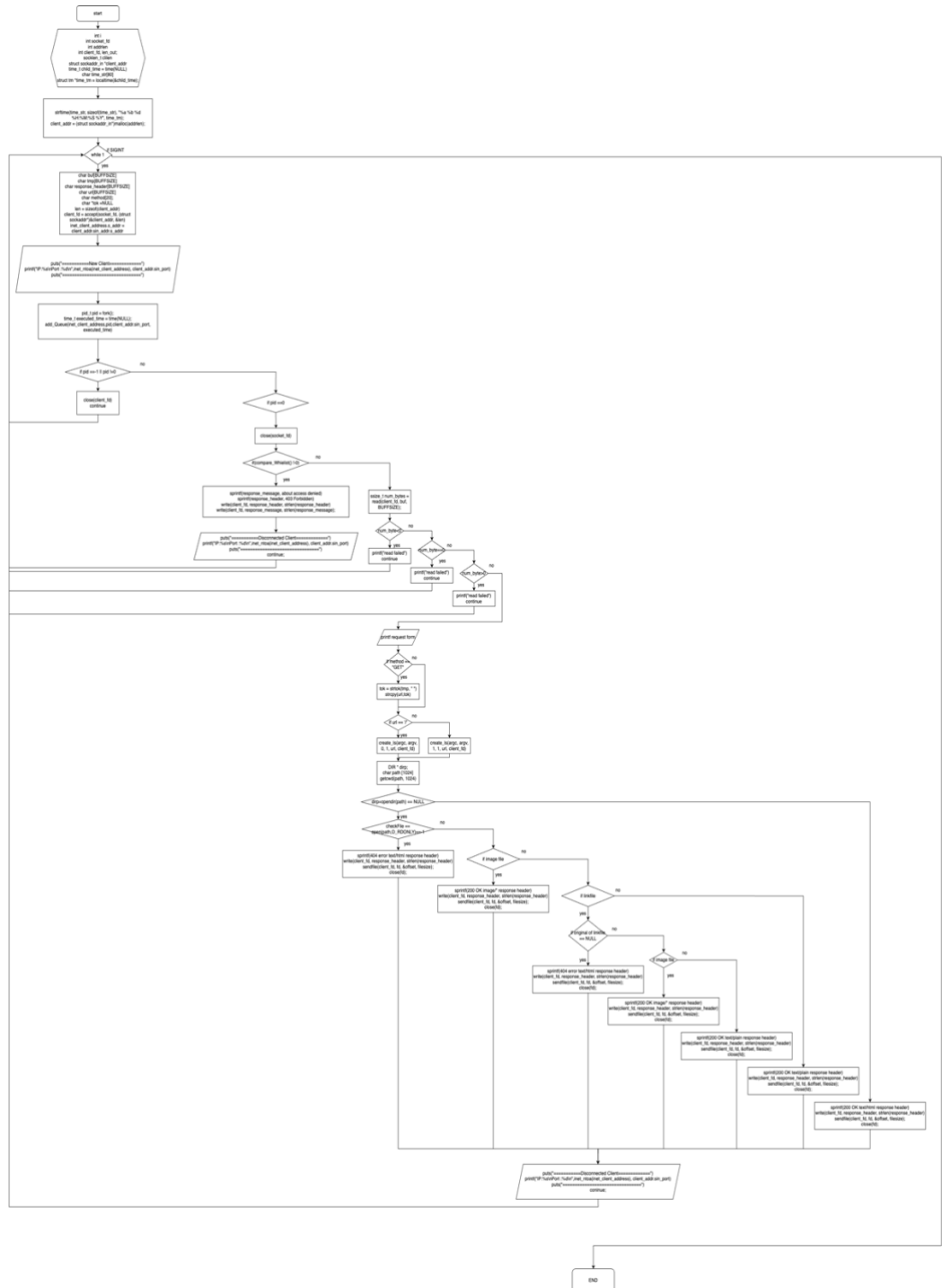
입력 받은 인자가 경로인 경우 실행하는 함수로 없는 경로인 경우에 404 not found html을 작성하고 정상 경로인 경우 ls-l or -al 의 결과를 html 파일에 작성한다.

O. create_ls



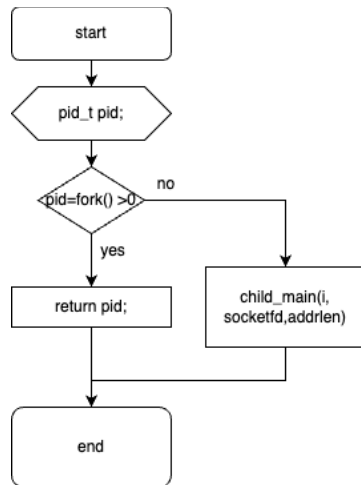
ls를 구현하기 위한 중간 단계의 함수로 directory가 존재하지 않는 경우 404 not found html을 작성하고 아니라면 flag_a와 flag_l에 따라 ls의 결과를 작성하는 dir_print 함수를 호출한다.

P. child_main

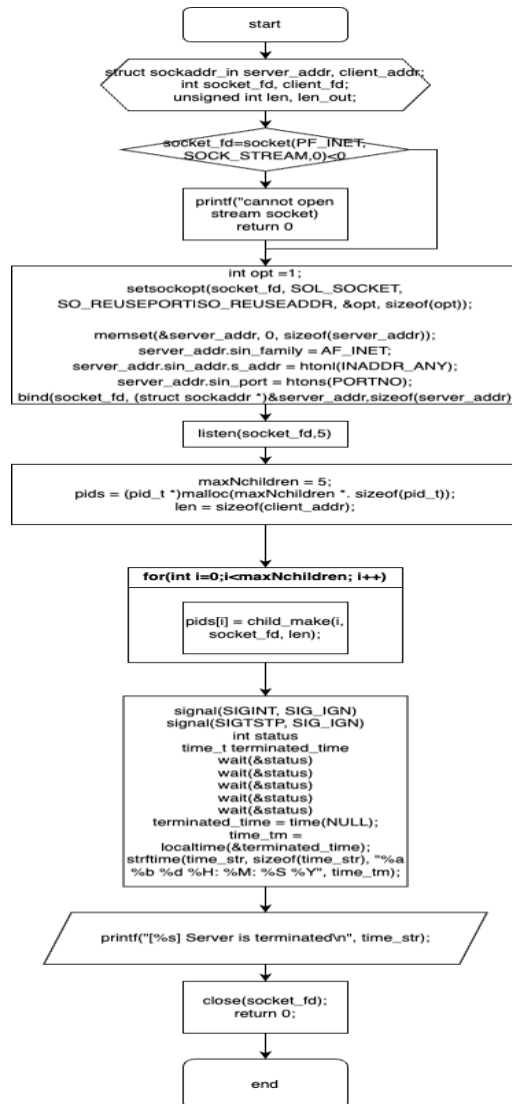


이전 main함수에 있던 코드의 일부로 child 전용으로 함수를 빼두어 continue로 반복한다. SIGINT 시그널을 받았을 때 child process가 종료되도록 설계되었다. accessible_usr 에 있는 접근 가능 IP 주소를 확인 후 패턴과 일치하면 접속하도록 하고 아니라면 403 forbidden을 보내도록 한다. response는 directory, 파일, 이미지 파일, 링크파일 등에 따라 나뉘어서 response_header를 작성하고 이에 따른 html 파일 또는 response_message를 client로 보낸다.

Q. child_make



R. main



기본 socket과 bind, listen 까지 진행하며 childmake로 child process 만든 후에는 SIGINT와 SIGTERM은 자식 프로세스에게 SIGUSR2 시그널을 보낸다.. 그리고 child가 5개 종료될 때까지 wait을 한다.

3. Pseudo code

A. compare_WhiteList

```
FILE *file = fopen(ACCESSIBLE_USR,"r");
char line[16]
while(fgets(line, sizeof(line),file)!=NULL){
    unsigned int len = strlen(line)
    if(len>0&&line[len-1]=='\n')
        line[len-1]='\0'
    if(fnmatch(line,IP_Address,0)==0)
        return 0;
    else
        continue;
}
fclose(file)
return -1;
```

B. is_empty

```
return tail<head
```

C. is_full

```
return tail== MAX_QUEUE-1
```

D. sub_Queue

```
if is_empty(){
    return;
}
for(int i=0;i<0;i++){
    NUM_Queue[i] = NUM_Queue[i+1];
    IP_Queue[i] = IP_Queue[i+1]
    PID_Queue[i] = PID_Queue[i+1]
    PORT_Queue[i] = PORT_Queue[i+1]
    TIME_Queue[i] = TIME_Queue[i+1]
}
tail--
```

E. add_Queue

```
if is_full(){
    sub_Queue()
}
NUM_Queue[++tail] = ++queue_cnt;
IP_Queue[tail] = IP
PID_Queue[tail] = PID
PORT_Queue[tail] = PORT
TIME_Queue[tail] = TIME
```

```
}
```

F. Alarm_handler

```
puts("=====connection history=====")
printf("Number of request(s) : %d\n" queue_cnt);
printf("NO.WtIPWtWtPIDWtPORTWtTIME\n")
inet_ntoa(IP_Queue[i]), PID_Queue[i], PORT_Queue[i], time_str
```

G. exit_handler

```
time_t terminated_time = time(NULL)
char time_str[80]
struct tm *time_tm
time_tm. = localtime(terminated_time)
strftime(time_str, sizeof(time_str), convert to time format)
print time_str, pid , process is terminated
exit(0);
```

H. child_handler

```
for(int i=0; i<tail+1; i++)
    printf("%dWt%sWt%dWt%dWt%s\n",NUM_Queue[i],
```

I. print_permission

```
fprint file, <td>
if file is dir, print "d" else "-"
if have read permission by user, fprint file, "r" else "-"
if have write permission by user, fprint file, "w" else "-"
if have execute permission by user, fprint file, "x" else "-"
if have read permission by group, fprint file, "r" else "-"
if have write permission by group, fprint file, "w" else "-"
if have execute permission by group, fprint file, "x" else "-"
if have read permission by other, fprintf file "r" else "-"
if have write permission by other, fprintf file, "w" else "-"
if have execute permission by other, fprint file, "x" else "-"
```

J. print_UID

```
get struct passwd pointer pw
if pw=getpwuid(uid) is not NULL
    fprint file, pw->pw_name
else
    fprint file, uid
```

K. print_GID

```
get strcut group pointer gr
if gr=getgrgid(gid) is not NULL
    fprint file, pw -> pw_name
```

```
else
    fprintf file, gid
```

L. print_lastModified_time

```
make array 80
get struct tm pointer time_tm
time_tm is localtime(&time)
    run strftime to get date and time format
    fprintf file, time
```

M. print_long

```
getcurrent working directory to cwd
count cwd's slash
if dir_name == "."
    tmppath=abs_path
else
    tmppath = abs_path,dir_name

get relative directory about root directory

use print_permission function
fprintf file, file_stat,st_nlink
use print_UID function
use print_GID function
if -h option activated
    use print_readableSIZE function
else
    just fprintf file, file_stat.st_size
use print_LastModified_time function
fprintf file, file name
```

N. dir_print

```
struct file_info{
    char name[1024];
    struct stat st;
};
int num_cnt=0
int total_size=0
struct file_info files[1024]
if dirp=opendir(abs_Path) ==NULL
    fprintf file, 404 error message
while dir = readdir(dirp)!=NULL
```

```

        if(flag_a || !hidden file)
            files[num_cnt].name = dir->d_name;
            total_size+= files[num_cnt++].st_blocks/2

sort files ascending order by name

fprintf file, html_ls.html result
for(int i=0; i<num_cnt;i++){
    if linkfile
        style ="color:green"
    else if dir file
        style ="color:blue"
    else
        style="color:red"

```

O. create_ls

```

get current working directory to path
if not root path
    strcat(path,url)
if dirp=opendir(path)==NULL
    if checkFile=open(path,O_RDONLY))-1{
        fprintf file, 404 error message html
    }
dir_print(path, flag_a, flag_l, url, file)
fclose(file)

```

P. child_main

```

while(1)
    struct in_addr inet_client_address
    len = sizeof(client_addr)
    client_fd = accept socket_fd, (struct sockaddr*)&client_addr, &len
    inet_client_address.s_addr = client_addr.sin_addr.s_addr

    puts("====New Client====")
    printf("IP : %s\nPORT: %d\n", inet_ntoa(inet_client_address),client_addr.sin_port)
    puts("====")

    pid_t pid=fork()
    time_t executed_time = time(NULL)
    add_Queue(inet_client_address,pid,client_addr.sin_port,executed_time)

if(pid != 0)

```

```

        close(client_fd)
        continue;
else if(pid ==0)
    close(socket_fd)
    if(compare_WhiteList(inet_ntoa(client_addr.sin_addr))!=0){
        sprintf(response_message, about access denied)
        sprintf(response_header, about 403 forbidden)
        write(client_fd, response_header, strlen(response_header))
        write(client_fd, response_message, strlen(response_message))
        continue
    }
    ssize_t num_bytes = read(client_fd, buf, BUFSIZE)
    if num_bytes <=0
        continue
    else if num_bytes > BUFSIZE
        continue

    printf request form
    if method == "GET"
        tok = strtok(tmp, " ")
        strcpy(url, tok)
    if url == '/'
        create_ls(argc, argv, 0, 1, url, client_fd)
    else
        create_ls(argc, argv, 1, 1, url, client_fd)

    if not directory
        if not file
            make 404 error text/html response header
            write(client_fd, response_header, strlen(response_header))
            send(client_fd,fd,&offset, filesize)
        else
            if image file
                make 200 OK image/* response header
                write(client_fd, response_header, strlen(response_header))
                send(client_fd,fd,&offset, filesize)
            else
                if link file
                    if original of linkfile == NULL
                        make 404 error text/html response header
                        write(client_fd, response_header, strlen(response_header))
                        send(client_fd,fd,&offset, filesize)

```



```

        else
            if image file
                make 200 OK image/* response header
                write(client_fd, response_header, strlen(response_header))
                send(client_fd,fd,&offset, filesize)
            else
                make 200 OK text/plain response header
                write(client_fd, response_header, strlen(response_header))
                send(client_fd,fd,&offset, filesize)
            else
                make 200 OK text/plain response header
                write(client_fd, response_header, strlen(response_header))
                send(client_fd,fd,&offset, filesize)
    else
        make 200 OK text/html response header
        write(client_fd, response_header, strlen(response_header))
        send(client_fd,fd,&offset, filesize)
    puts("=====Disconnected Client=====")
    printf("IP : %s\nPORT: %d\n", inet_ntoa(inet_client_address),client_addr.sin_port)
    puts("=====")
    remove(filename)
    continue;

```

Q. child_make

```

pid_t pid;
if(pid=fork()>0)
    return pid
child_main(i, sockfd, addrlen)

```

R. main

```

struct sockaddr_in server_addr, client_addr
int socket_fd, client_fd
unsigned int len, len_out;

socket_fd=socket(PF_INET, SOCK_STREAM,0)

setsockopt  socket_fd,  SOL_SOCKET,  SO_REUSEPORT|SO_REUSEADDR,  &opt,
sizeof(opt)

memset(&server_addr,0,sizeof(server_addr))
server_addr.sin_family = AF_INET
server_addr.sin_addr.s_addr = htonl(INADDR_ANY)
server_addr.sin_port = htons(PORTNO)

```

```

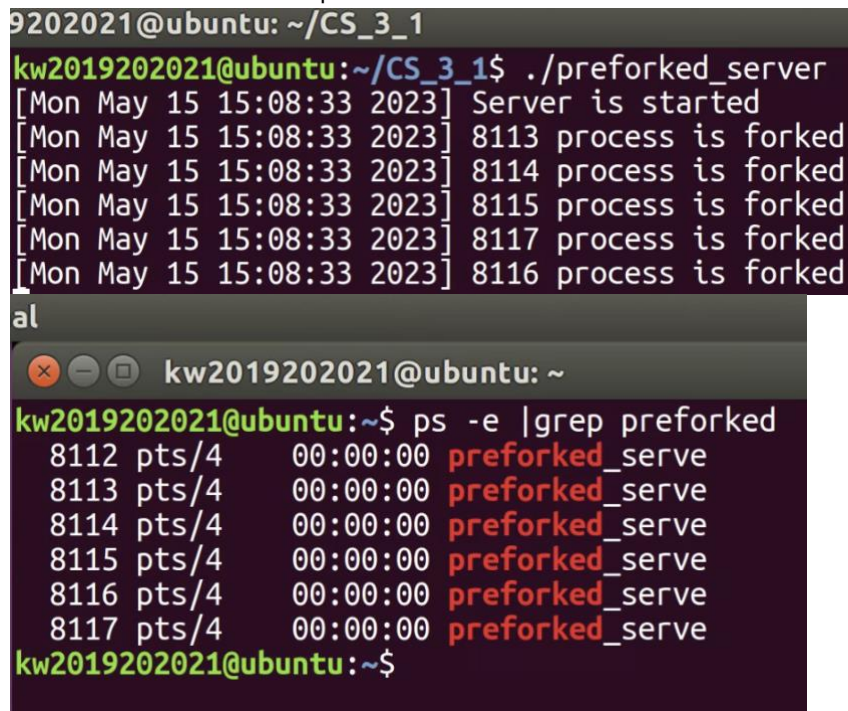
bind socket_fd, (struct sockaddr *)&server_addr,sizeof(server_addr)
listen socket_fd 5
maxNchildren = 5
pids = malloc 5* sizeof(pid_t)
repeat 5 time
    pids[i] = child_make
signal(SIGINT, parent_handler);
signal(SIGTERM, parent_handler);
repeat 5. ime
    wait(%status);
printf "server is. terminated");
close(socket_fd);
return(0);

```

각 함수에 대한 설명은 위의 flow chart에서 설명한 것과 동일하다.

4. 결과화면

A. web server 실행 & child process 생성



```

kw2019202021@ubuntu: ~/CS_3_1
kw2019202021@ubuntu:~/CS_3_1$ ./preforked_server
[Mon May 15 15:08:33 2023] Server is started
[Mon May 15 15:08:33 2023] 8113 process is forked
[Mon May 15 15:08:33 2023] 8114 process is forked
[Mon May 15 15:08:33 2023] 8115 process is forked
[Mon May 15 15:08:33 2023] 8117 process is forked
[Mon May 15 15:08:33 2023] 8116 process is forked
al
kw2019202021@ubuntu: ~
kw2019202021@ubuntu:~$ ps -e |grep preforked
 8112 pts/4    00:00:00 preforked_serve
 8113 pts/4    00:00:00 preforked_serve
 8114 pts/4    00:00:00 preforked_serve
 8115 pts/4    00:00:00 preforked_serve
 8116 pts/4    00:00:00 preforked_serve
 8117 pts/4    00:00:00 preforked_serve
kw2019202021@ubuntu:~$

```

부모 프로세스의 pid는 8112이고 자식 프로세스는 8113~8117까지 5개 존재한다.
web server를 실행했을 때 자식 프로세스에 대한 정보를 알 수 있고 ps -e |grep preforked 명령어를 사용하여 preforked_server 관련 프로세스 실행 정보를 확인하였다.

B. client 접속 & 종료

```

=====New Client=====
[Mon May 15 15:10:59 2023]
IP : 127.0.0.1
Port : 55463
=====

=====Disconnected Client=====
[Mon May 15 15:10:59 2023]
IP : 127.0.0.1
Port : 55463
=====

=====New Client=====
[Mon May 15 15:11:00 2023]
IP : 127.0.0.1
Port : 55975
=====

=====Disconnected Client=====
[Mon May 15 15:11:00 2023]
IP : 127.0.0.1
Port : 55975
=====

=====New Client=====
[Mon May 15 15:11:00 2023]
IP : 127.0.0.1
Port : 56487
=====

=====Disconnected Client=====
[Mon May 15 15:11:00 2023]
IP : 127.0.0.1
Port : 56487
=====

=====New Client=====
[Mon May 15 15:11:00 2023]
IP : 127.0.0.1
Port : 56999
=====

=====Disconnected Client=====
[Mon May 15 15:11:00 2023]
IP : 127.0.0.1
Port : 56999
=====

```

웹 브라우저에서 server 접속 시 client connection과 disconnection을 확인할 수 있다.

C. Connection history

=====Connection History=====						
NO.	IP	PID	PORT	TIME		
5	127.0.0.1	8113	55975	Mon May 15	15:11:00	2023
4	127.0.0.1	8113	53415	Mon May 15	15:10:59	2023
3	127.0.0.1	8113	50855	Mon May 15	15:10:58	2023
2	127.0.0.1	8113	48295	Mon May 15	15:10:57	2023
1	127.0.0.1	8113	45223	Mon May 15	15:10:55	2023
5	127.0.0.1	8114	56999	Mon May 15	15:11:00	2023
4	127.0.0.1	8114	54439	Mon May 15	15:10:59	2023
3	127.0.0.1	8114	51879	Mon May 15	15:10:58	2023
5	127.0.0.1	8115	56487	Mon May 15	15:11:00	2023
2	127.0.0.1	8114	49319	Mon May 15	15:10:58	2023
4	127.0.0.1	8115	53927	Mon May 15	15:10:59	2023
1	127.0.0.1	8114	46759	Mon May 15	15:10:57	2023
3	127.0.0.1	8115	51367	Mon May 15	15:10:58	2023
2	127.0.0.1	8115	48807	Mon May 15	15:10:57	2023
1	127.0.0.1	8115	46247	Mon May 15	15:10:56	2023
5	127.0.0.1	8116	57511	Mon May 15	15:11:00	2023
4	127.0.0.1	8116	54951	Mon May 15	15:10:59	2023
5	127.0.0.1	8117	58023	Mon May 15	15:11:01	2023
3	127.0.0.1	8116	52391	Mon May 15	15:10:58	2023
4	127.0.0.1	8117	55463	Mon May 15	15:10:59	2023
2	127.0.0.1	8116	49831	Mon May 15	15:10:58	2023
3	127.0.0.1	8117	52903	Mon May 15	15:10:59	2023
1	127.0.0.1	8116	47271	Mon May 15	15:10:57	2023
2	127.0.0.1	8117	50343	Mon May 15	15:10:58	2023
1	127.0.0.1	8117	47783	Mon May 15	15:10:57	2023

출력에 대한 동기화를 진행하지 않아서 각 출력이 뒤죽박죽이지만 각 프로세스 별 connection history는 각각 카운트 되고 최신 순으로 출력되고 있음을 확인할 수 있다.

D. Connection 50번 이상

=====Connection History=====						
NO.	IP	PID	PORT	TIME		
29	127.0.0.1	8113	51368	Mon May 15	15:13:41	2023
28	127.0.0.1	8113	48808	Mon May 15	15:13:26	2023
27	127.0.0.1	8113	46248	Mon May 15	15:13:26	2023
26	127.0.0.1	8113	43688	Mon May 15	15:13:25	2023
25	127.0.0.1	8113	41128	Mon May 15	15:13:24	2023
27	127.0.0.1	8114	49320	Mon May 15	15:13:26	2023
24	127.0.0.1	8113	38568	Mon May 15	15:13:24	2023
26	127.0.0.1	8114	46760	Mon May 15	15:13:26	2023
23	127.0.0.1	8113	36008	Mon May 15	15:13:23	2023
25	127.0.0.1	8114	44200	Mon May 15	15:13:25	2023
22	127.0.0.1	8113	35496	Mon May 15	15:13:23	2023
24	127.0.0.1	8114	41640	Mon May 15	15:13:25	2023
21	127.0.0.1	8113	32936	Mon May 15	15:13:23	2023
23	127.0.0.1	8114	39080	Mon May 15	15:13:24	2023
20	127.0.0.1	8113	30376	Mon May 15	15:13:22	2023
22	127.0.0.1	8114	36520	Mon May 15	15:13:23	2023
21	127.0.0.1	8114	33448	Mon May 15	15:13:23	2023
20	127.0.0.1	8114	30888	Mon May 15	15:13:22	2023
19	127.0.0.1	8114	28328	Mon May 15	15:13:22	2023
28	127.0.0.1	8115	51880	Mon May 15	15:13:44	2023
18	127.0.0.1	8114	25768	Mon May 15	15:13:21	2023
27	127.0.0.1	8115	49832	Mon May 15	15:13:26	2023
26	127.0.0.1	8115	47272	Mon May 15	15:13:26	2023
25	127.0.0.1	8115	44712	Mon May 15	15:13:25	2023
24	127.0.0.1	8115	42152	Mon May 15	15:13:25	2023
23	127.0.0.1	8115	39592	Mon May 15	15:13:24	2023
27	127.0.0.1	8116	50344	Mon May 15	15:13:26	2023
22	127.0.0.1	8115	37032	Mon May 15	15:13:24	2023
26	127.0.0.1	8116	47784	Mon May 15	15:13:26	2023
21	127.0.0.1	8115	33960	Mon May 15	15:13:23	2023
25	127.0.0.1	8116	45224	Mon May 15	15:13:25	2023

20	127.0.0.1	8115	31400	Mon May 15 15:13:22 2023
24	127.0.0.1	8116	42664	Mon May 15 15:13:25 2023
19	127.0.0.1	8115	28840	Mon May 15 15:13:22 2023
23	127.0.0.1	8116	40104	Mon May 15 15:13:24 2023
22	127.0.0.1	8116	37544	Mon May 15 15:13:24 2023
21	127.0.0.1	8116	34472	Mon May 15 15:13:23 2023
20	127.0.0.1	8116	31912	Mon May 15 15:13:22 2023
27	127.0.0.1	8117	50856	Mon May 15 15:13:27 2023
19	127.0.0.1	8116	29352	Mon May 15 15:13:22 2023
26	127.0.0.1	8117	48296	Mon May 15 15:13:26 2023
18	127.0.0.1	8116	26792	Mon May 15 15:13:22 2023
25	127.0.0.1	8117	45736	Mon May 15 15:13:25 2023
24	127.0.0.1	8117	43176	Mon May 15 15:13:25 2023
23	127.0.0.1	8117	40616	Mon May 15 15:13:24 2023
22	127.0.0.1	8117	38056	Mon May 15 15:13:24 2023
21	127.0.0.1	8117	34984	Mon May 15 15:13:23 2023
20	127.0.0.1	8117	32424	Mon May 15 15:13:23 2023
19	127.0.0.1	8117	29864	Mon May 15 15:13:22 2023
18	127.0.0.1	8117	27304	Mon May 15 15:13:22 2023

각 프로세스 별로 최대 최신 10개, 즉 전체 최신 50개만 connection history로 출력한다.

- E. SIGINT 발생 (Child process & Parent process 종료)

```

^C[Mon May 15 15:16:38 2023] 8115 process is terminated.
[Mon May 15 15:16:38 2023] 8114 process is terminated.
[Mon May 15 15:16:38 2023] 8116 process is terminated.
[Mon May 15 15:16:38 2023] 8113 process is terminated.
[Mon May 15 15:16:38 2023] 8117 process is terminated.
[Mon May 15 15:16:38 2023] Server is terminated
kw2019202021@ubuntu:~/CS_3_1$ 
kw2019202021@ubuntu:~$ ps -e |grep preforked
kw2019202021@ubuntu:~$ 

kw2019202021@ubuntu:~$ top -b -n 1 |grep zombie
Tasks: 250 total,  1 running, 181 sleeping,  0 stopped,  0 zombie
kw2019202021@ubuntu:~$

```

SIGINT가 발생했을 때 preforked_server 관련 모든 프로세스가 종료됨을 확인할 수 있다. 이 때, 부모 프로세스는 자식 프로세스가 모두 종료됨을 확인하고 종료된다.

- F. kill 명령어로 부모프로세스를 종료 시키는 경우

```

8363 pts/4      00:00:00 preforked_serve
8364 pts/4      00:00:00 preforked_serve
8365 pts/4      00:00:00 preforked_serve
8366 pts/4      00:00:00 preforked_serve
8367 pts/4      00:00:00 preforked_serve
8368 pts/4      00:00:00 preforked_serve
8369 pts/11     00:00:00 ps
kw2019202021@ubuntu:~$ kill -2 8363

```

```
kw2019202021@ubuntu:~/CS_3_1$ ./preforked_server
[Mon May 15 15:31:31 2023] Server is started
[Mon May 15 15:31:31 2023] 8364 process is forked
[Mon May 15 15:31:31 2023] 8365 process is forked
[Mon May 15 15:31:31 2023] 8367 process is forked
[Mon May 15 15:31:31 2023] 8368 process is forked
[Mon May 15 15:31:31 2023] 8366 process is forked
=====Connection History=====
NO.      IP      PID      PORT      TIME
[Mon May 15 15:31:43 2023] 8366 process is terminated.
[Mon May 15 15:31:43 2023] 8364 process is terminated.
[Mon May 15 15:31:43 2023] 8367 process is terminated.
[Mon May 15 15:31:43 2023] 8365 process is terminated.
[Mon May 15 15:31:43 2023] 8368 process is terminated.
[Mon May 15 15:31:43 2023] Server is terminated
```

kill 명령어로 SIGINT 시그널을 보냈을 때 결과이다.

```
8386 pts/4      00:00:00 preforked_serve
8387 pts/4      00:00:00 preforked_serve
8388 pts/4      00:00:00 preforked_serve
8389 pts/4      00:00:00 preforked_serve
8390 pts/4      00:00:00 preforked_serve
8391 pts/4      00:00:00 preforked_serve
8392 pts/11     00:00:00 ps
kw2019202021@ubuntu:~$ kill -15 8386
kw2019202021@ubuntu:~$ 
kw2019202021@ubuntu:~/CS_3_1$ ./preforked_server
[Mon May 15 15:35:32 2023] Server is started
[Mon May 15 15:35:32 2023] 8387 process is forked
[Mon May 15 15:35:32 2023] 8388 process is forked
[Mon May 15 15:35:32 2023] 8389 process is forked
[Mon May 15 15:35:32 2023] 8391 process is forked
[Mon May 15 15:35:32 2023] 8390 process is forked
=====Connection History=====
NO.      IP      PID      PORT      TIME
[Mon May 15 15:35:52 2023] 8389 process is terminated.
[Mon May 15 15:35:52 2023] 8387 process is terminated.
[Mon May 15 15:35:52 2023] 8388 process is terminated.
[Mon May 15 15:35:52 2023] 8391 process is terminated.
[Mon May 15 15:35:52 2023] 8390 process is terminated.
[Mon May 15 15:35:52 2023] Server is terminated
```

kill 명령어로 SIGTERM 시그널을 보냈을 때 결과이다.

두 시그널 모두 자식 프로세스 먼저 종료하고 부모 프로세스가 종료되는 것을 확인할 수 있다.

5. 고찰

Assignment 3-1에서는 수업 시간에 배운 fork를 바탕으로 5개의 자식 프로세스를 미리 생성하여 이 자식프로세스로 client의 request에 response 하도록 preforked web server를 구현하였다. server를 시작하고 끝날 때 서버의 간단한 정보를 출력하고 자식 프로세스의 시작과 종료 또한 출력한다. 또한 connection history를 출력할 때 format은 부모 프로세스에서 출력하지만 history는 자식 프로세스에서 동기화를 고려하지 않고 각각 출력하도록 설계하였다. 무엇보다 중요한 것은 부모 프로세스가 자식프로세스를 wait()하지 않아 zombie process를 만드는 것을 방지해야했으며 처음에는 '^c' 가 되었을 때만 처리하기 위해 부모프로세스에서 SIGINT와 SIGTERM에 대해 SIG_IGN를 진행했었다. 그러나 다른 터미널에서 kill 명령어로 SIGINT나 SIGTERM을 보내면 종료되지 않는 문제가 발생하였다. 이를 방지하기 위해 SIGINT와 SIGTERM에 대한 handler를 두어 child 프로세스에

SIGUSR2 시그널을 보내고 이는 SIGINT, SIGTERM 과 같이 exit_handler로 전달된다.

터미널에서 kill 명령어를 다루던 중 SIGKILL에 대한 프로세스 처리를 생각해보았다. 이는 catch와 ignore 모두 불가능한 signal이므로 자식 프로세스를 기다리지 못하고 강제 종료되어 고아프로세스를 만들을 확인했다. ps -e로 프로세스를 확인하면 자식은 종료되지 못하고 남아있음을 확인할 수 있다. 다만 SIGKILL은 이와 같은 상황이 매우 많이 발생할 수 밖에 없기에 예외로 처리하지 않았다.

```
8463 pts/4    00:00:00 preforked_serve
8464 pts/4    00:00:00 preforked_serve
8465 pts/4    00:00:00 preforked_serve
8466 pts/4    00:00:00 preforked_serve
8467 pts/4    00:00:00 preforked_serve
8468 pts/4    00:00:00 preforked_serve
8469 pts/11   00:00:00 ps
kw2019202021@ubuntu:~$ kill -9 8463
kw2019202021@ubuntu:~/CS_3_1$ ./preforked_server
[Mon May 15 15:49:17 2023] Server is started
[Mon May 15 15:49:17 2023] 8464 process is forked
[Mon May 15 15:49:17 2023] 8465 process is forked
[Mon May 15 15:49:17 2023] 8466 process is forked
[Mon May 15 15:49:17 2023] 8467 process is forked
[Mon May 15 15:49:17 2023] 8468 process is forked
=====Connection History=====
NO.      IP          PID      PORT      TIME
=====Connection History=====
NO.      IP          PID      PORT      TIME
Killed
8420 ?        00:00:00 kworker/0:1
8447 ?        00:00:00 kworker/u256:2
8464 pts/4    00:00:00 preforked_serve
8465 pts/4    00:00:00 preforked_serve
8466 pts/4    00:00:00 preforked_serve
8467 pts/4    00:00:00 preforked_serve
8468 pts/4    00:00:00 preforked_serve
8472 pts/11   00:00:00 ps
kw2019202021@ubuntu:~$
```

그래도 각 자식 프로세스를 종료할 때는 종료됨을 나타낸 후 종료된다.

```
kw2019202021@ubuntu:~$ kill -15 8464
kw2019202021@ubuntu:~$ 
kw2019202021@ubuntu:~/CS_3_1$ [Mon May 15 15:51:03 2023] 8464 process is terminated.
[Mon May 15 15:51:03 2023] 8464 process is terminated.
8464 pts/4    00:00:00 preforked_serve
8465 pts/4    00:00:00 preforked_serve
8466 pts/4    00:00:00 preforked_serve
8467 pts/4    00:00:00 preforked_serve
8468 pts/4    00:00:00 preforked_serve
8472 pts/11   00:00:00 ps
kw2019202021@ubuntu:~$ kill -15 8464
kw2019202021@ubuntu:~$ top -b -n 1 |grep zombie
Tasks: 255 total,  1 running, 185 sleeping,  0 stopped,  0 zombie
kw2019202021@ubuntu:~$
```

또한 init 프로세스가 해당 자식프로세스를 wait하기에 zombie 또한 발생하지 않았다.

6. Reference

- 1) 시스템프로그래밍 실습 Assignment 3-1/광운대학교/ 컴퓨터정보공학부/ 김태석 교수님/2023
- 2) 시스템프로그래밍 강의자료/광운대학교/컴퓨터정보공학부/김태석교수님/2023