시스템프로그래밍

과제이름: Assignment 3-3

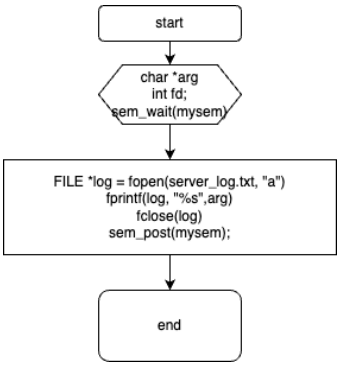
- 담당교수:김 태 석 교수님
- 학 과: 컴퓨터정보공학부
- 학 번: 2019202021
- 이 름: 정 성 엽

- 제출일: 2023/5/30

1. Introduction
   A. 과제 소개

      이번 Assignment 3-3 과제는 이전 Assignment 3-2에서 설계하였던 리눅스 환경에서의 C언어 기반 서버의 부모 프로세스에서 shared memory에서 이제는 공용 txt 파일을 semaphore를 사용하여 접근 제어를 한다. 해당 공용 txt는 로그를 저장하며 history는 제외하고 process 관리 및 client 접속 정보를 작성한다. 추가적으로 disconnect 될 때는 클라이언트 연결 종료 시간까지 출력하도록 한다.

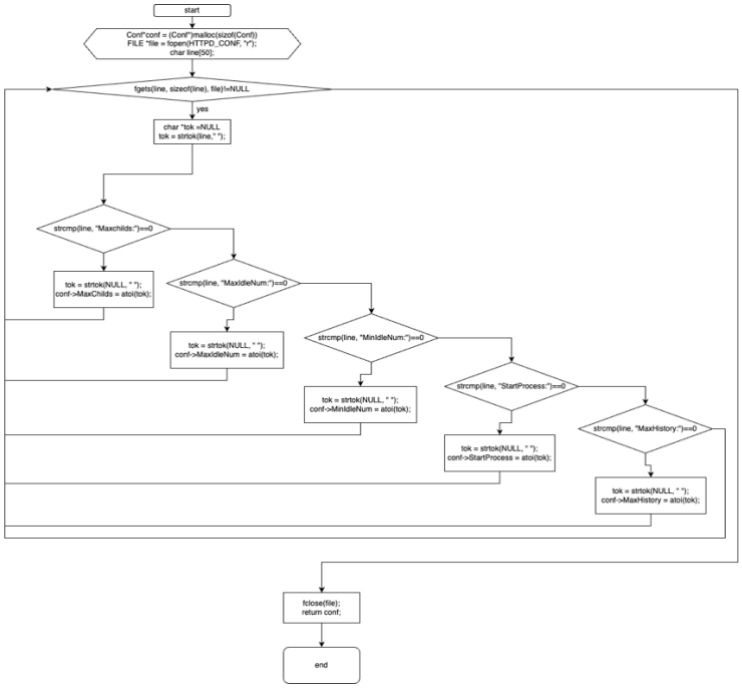
2. Flowchart
   A. write_log



      문자열을 매개변수로 받아서 server_log.txt에 작성하는 함수이다. 이때 이미 존재하는 파일이므로 "a" flag를 이용해서 뒤에 이어서 작성할 수 있도록 한다.

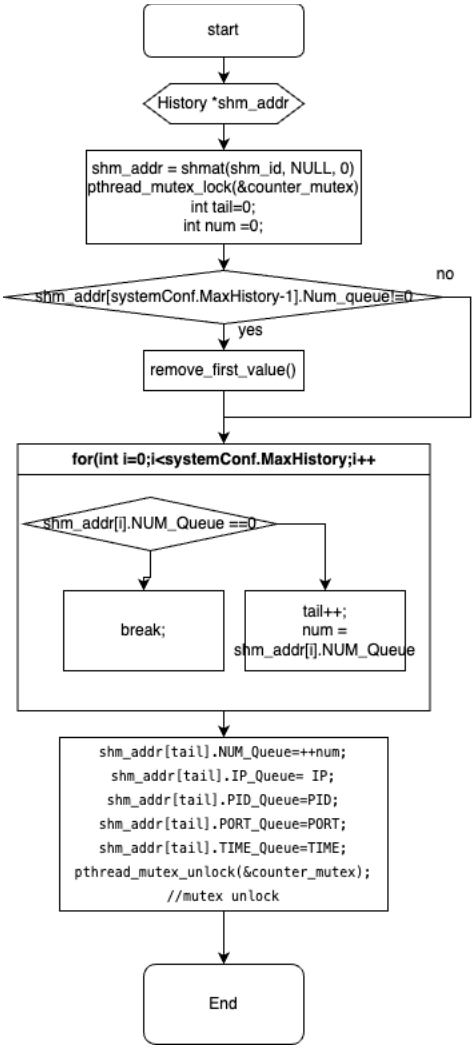
      또한 semaphore를 사용하여 sem_wait로 기다리고 작성이 끝나면 sem_post를 사용하여 해제해준다.

   B. get_Conf



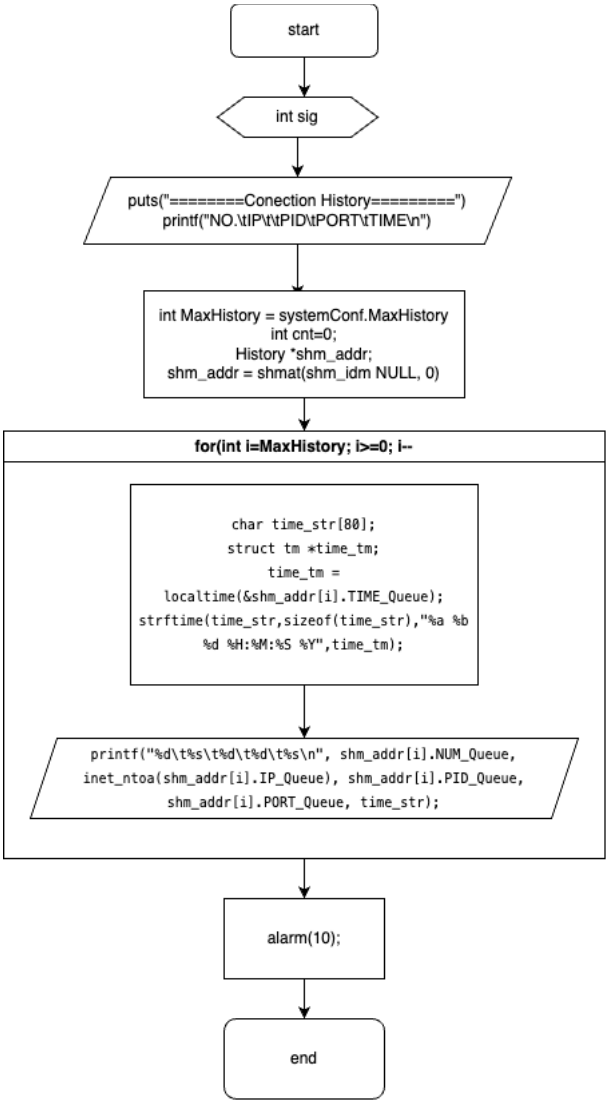
      get_conf()함수는 httpd.conf 파일에 적혀있는 각 요소에 대한 값을 읽고 구조체에 저장한 후 반환하는 함수로 초기에 사용하여 해당 요소 값들을 사용하여 프로그램이 진행되도록 한다.

C. add_history



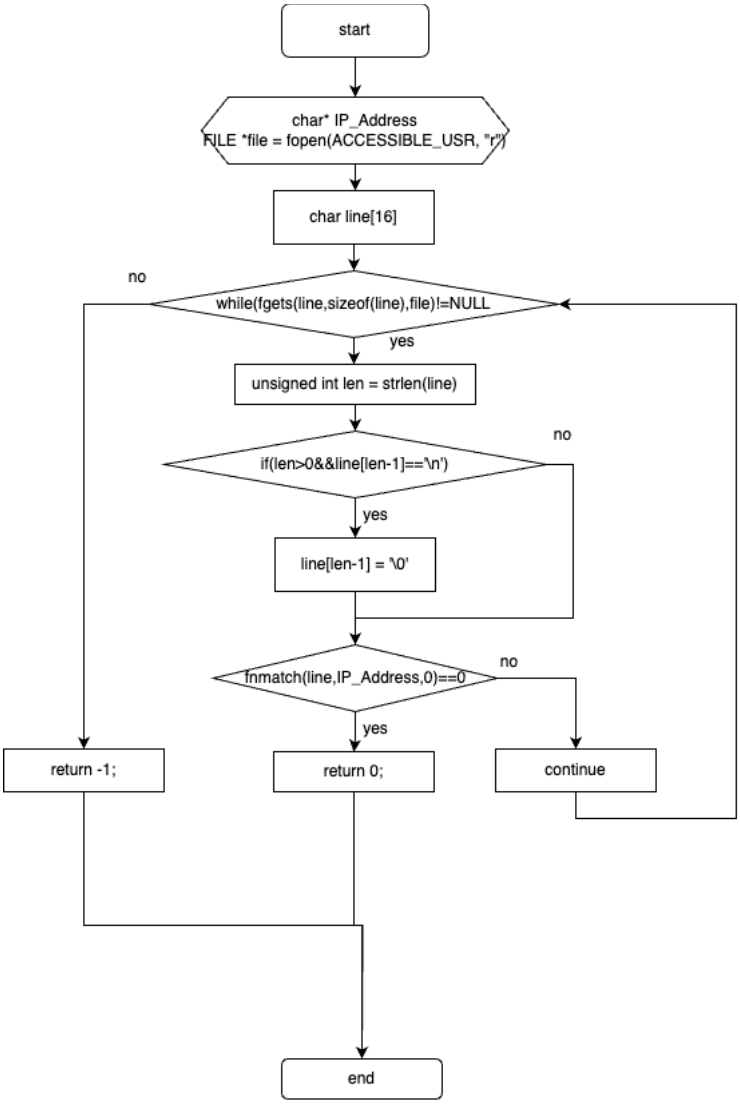
add_history()함수는 자식 프로세스에서 accept가 발생하고 정상적으로 이뤄진 경우 해당 경우를 공유메모리 구조체인 history에 추가하는 함수이다. 만약 history의 개수가 maxHistory만큼 있다면 첫번째 요소를 삭제한다. 아니라면 공유메모리 구조체 배열에 접근하여 해당 값을 저장한다.

D. alarm_handler



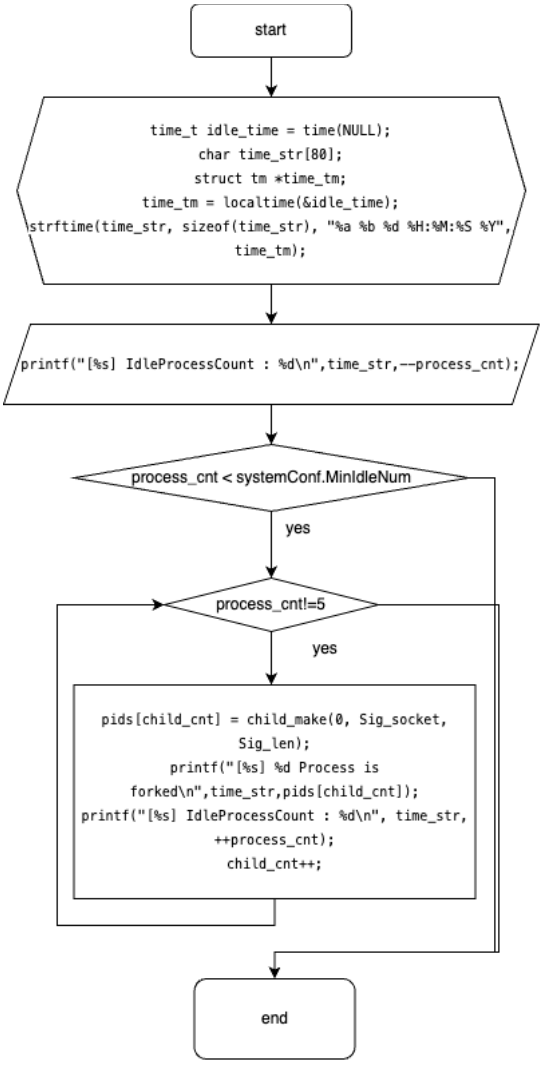
alarm_handler() 함수는 SIGALRM이 발생했을 때 동작하는 handler 함수로 공유메모리에 있는 값을 가져와 출력하는 동작을 진행한다. 10초마다 작동하며 작동 후에 새로운 alarm 을 둔다.

E. compare_WhiteList



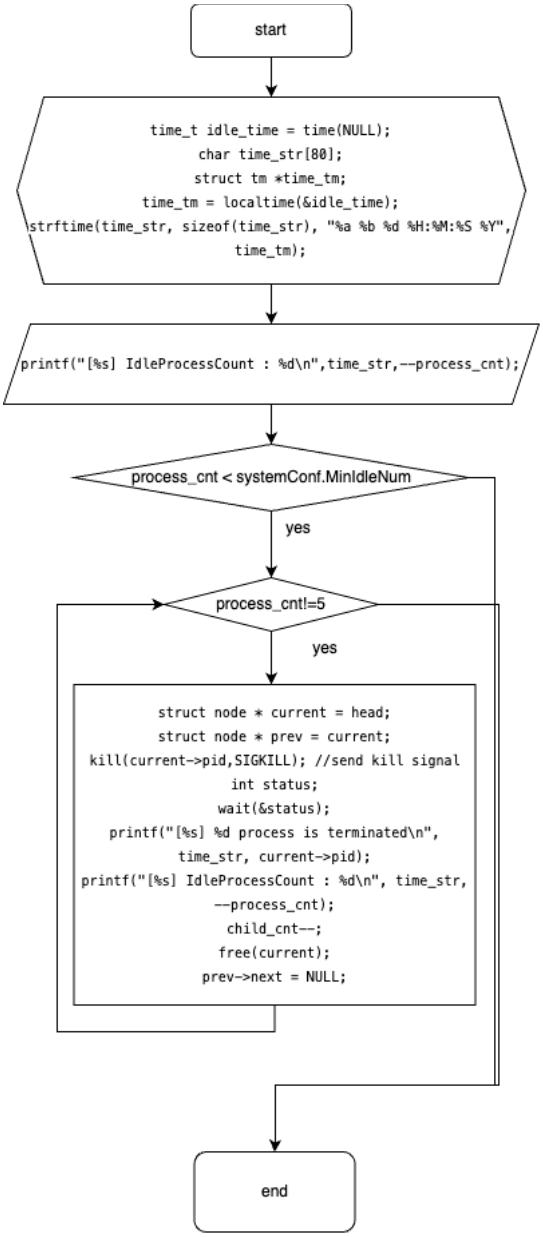
accessible.usr에 저장된 whitelist에 대한 정보를 읽고 패턴을 비교하여 일치하는 경우 0을 반환하고 일치하지 않는 경우 -1을 반환한다.

F. idle_handler



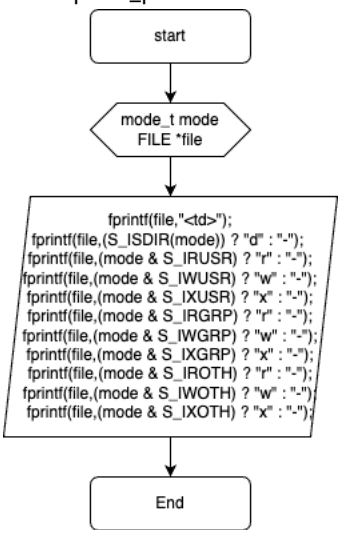
idle_hander()함수는 add_history()함수와 마찬가지로 accept 이후에 동작하도록 하며 이는 kill 함수를 통해 자식 프로세스에서 부모프로세스로 SIGUSR1 시그널을 보내서 동작하도록 한다. 프로세스를 사용하면 idleprocess 개수를 줄이고 특정 개수 이하라면 새로운 자식 프로세스를 제작한다.

G. idle_handler2



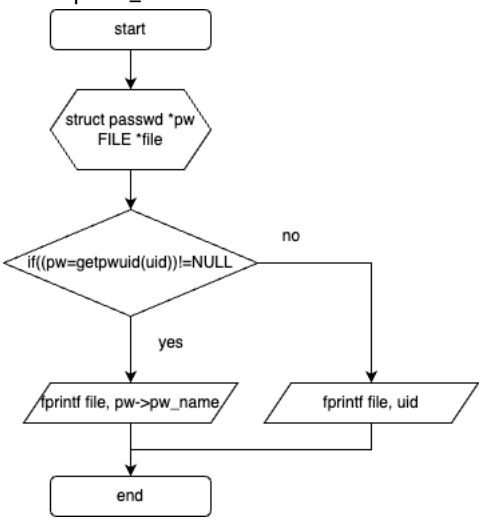
idle_hander2()함수는 disconnected client가 출력된 이후에 자식프로세스가 부모프로세스에게 SIGUSR2 시그널을 보낼 때 동작하는 함수로, 프로세스 동작이 끝나면 idleprocess의 개수를 늘리고, 특정 개수보다 많아진다면 만들어진 프로세스를 삭제한다.

H. print_permission



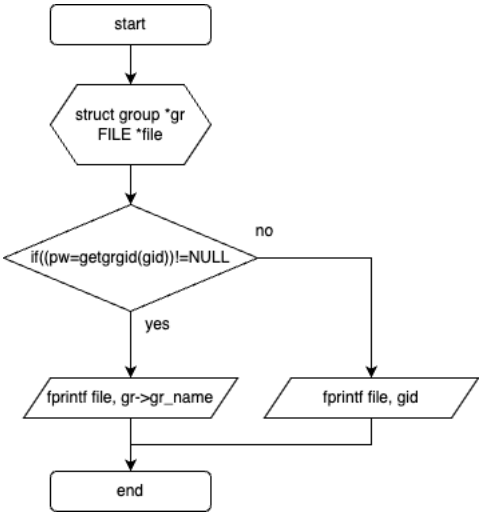
파일 허용정보를 출력하는 함수이다.

I. print_UID



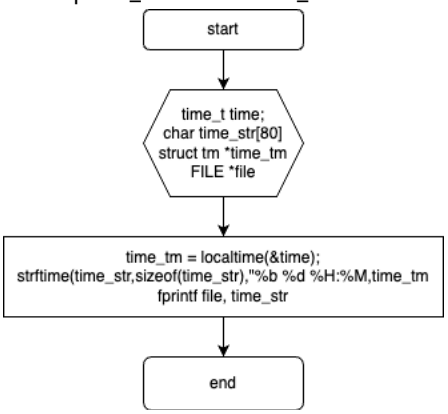
파일의 UID 정보를 출력하는 함수이다

J. print_GID



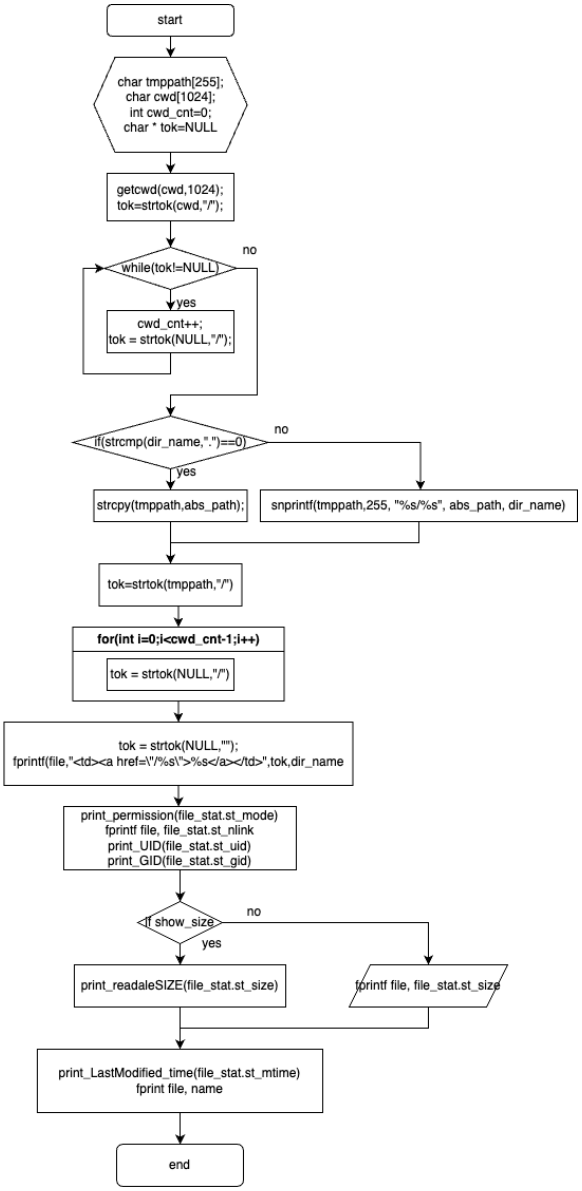
파일의 GID 정보를 출력하는 함수이다

K. print_LastModified_time



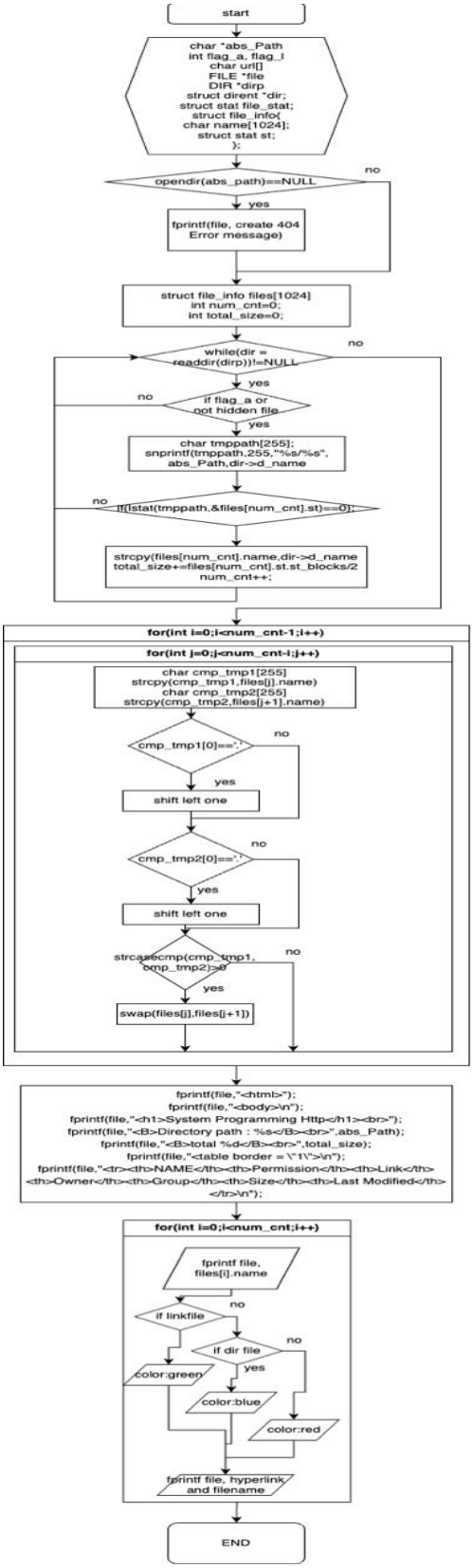
파일의 마지막 수정일자와 시간을 출력하는 함수이다.

L. print_long



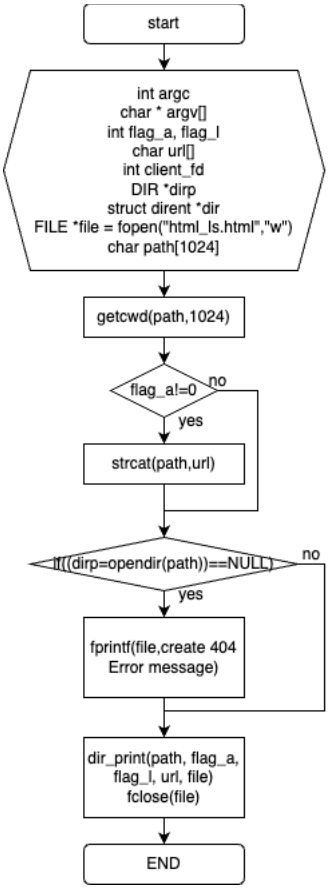
의 정보를 long format에 맞춰서 출력하는 함수이다. 현재 working directory와 url 정보를 합쳐서 파일에 접근한다.

M. dir_print



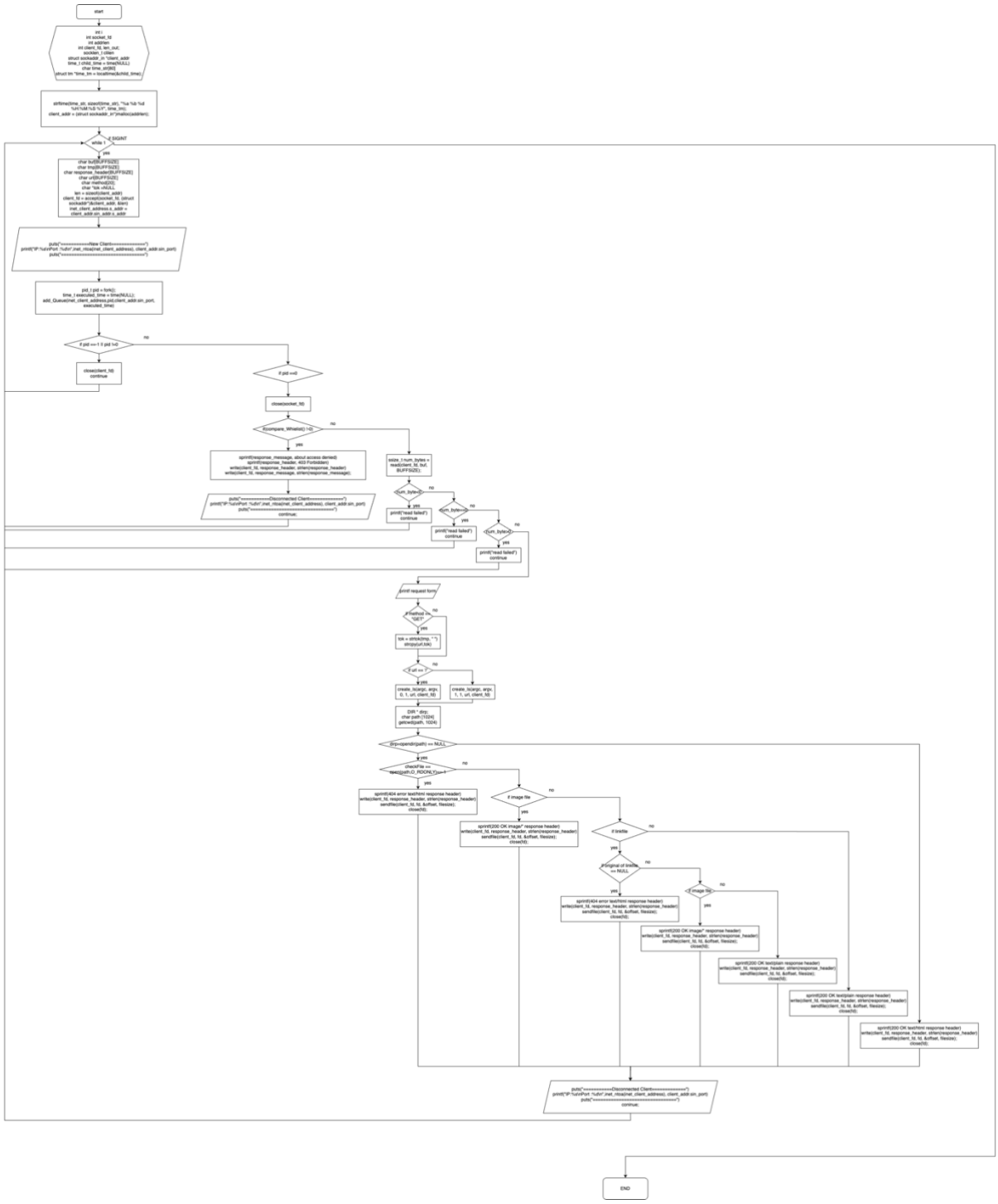
입력 받은 인자가 경로인 경우 실행하는 함수로 없는 경로인 경우에 404 not found html을 작성하고 정상 경로인 경우 ls-l or -al 의 결과를 html 파일에 작성한다.

N. create_ls



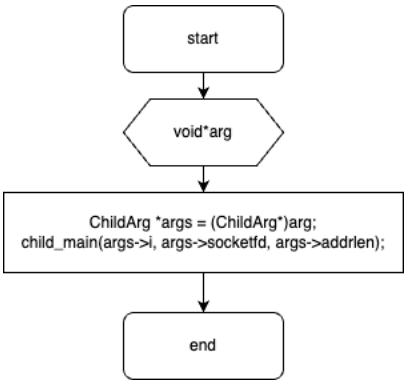
ls를 구현하기 위한 중간 단계의 함수로 directory가 존재하지 않는 경우 404 not found html을 작성하고 아니라면 flag_a와 flag_l에 따라 ls의 결과를 작성하는 dir_print 함수를 호출한다.

O. child_main



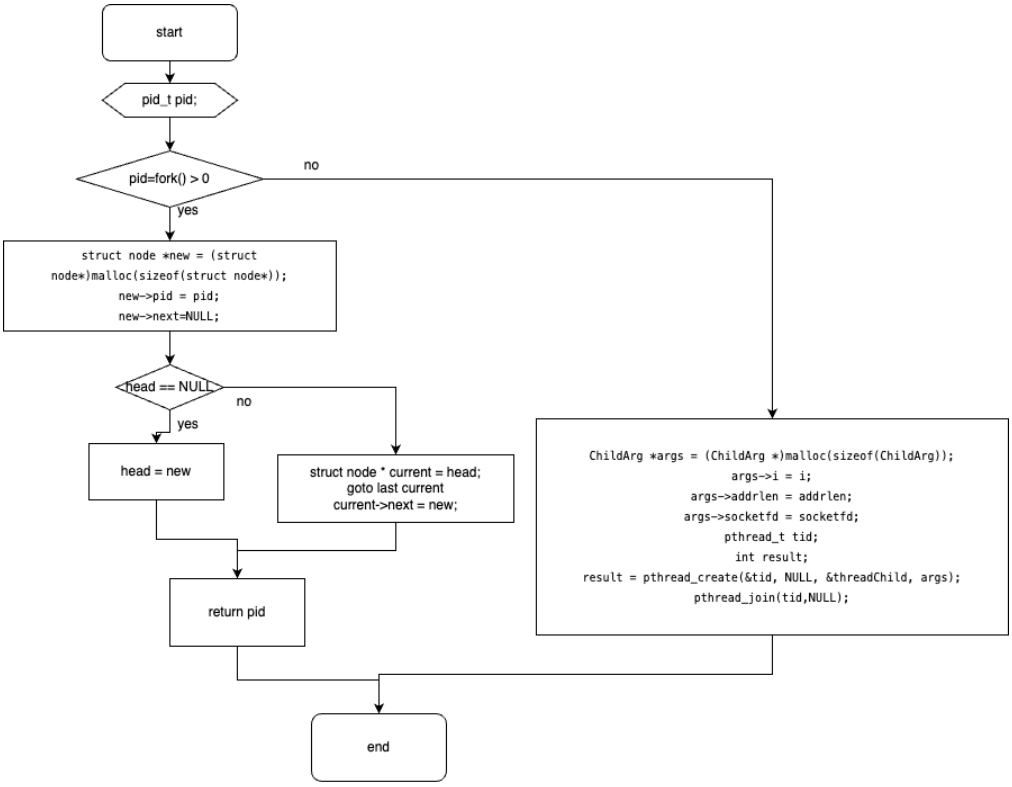
이전 main함수에 있던 코드의 일부로 child 전용으로 함수를 빼두어 continue로 반복한다. SIGINT 시그널을 받았을 때 child process가 종료되도록 설계되었다. accessible.usr 에 잇는 접근 가능 IP 주소를 확인 후 패턴과 일치하면 접속하도록 하고 아니라면 403 forbidden을 보내도록 한다. response는 directory, 파일, 이미지파일, 링크파일 등에 따라 나뉘어서 response_header를 작성하고 이에 따른 html 파일 또는 response_message를 client로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

P. threadChild



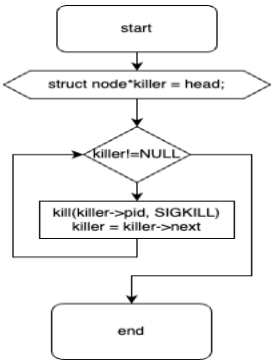
child_main을 pthread_create로 쓰기에는 고려해야할 요소가 많아서 중간에 중계 역할을 하여 pthread도 안정적으로 만들어지고 child_main도 호출하도록 설계하였다.

Q. child_make



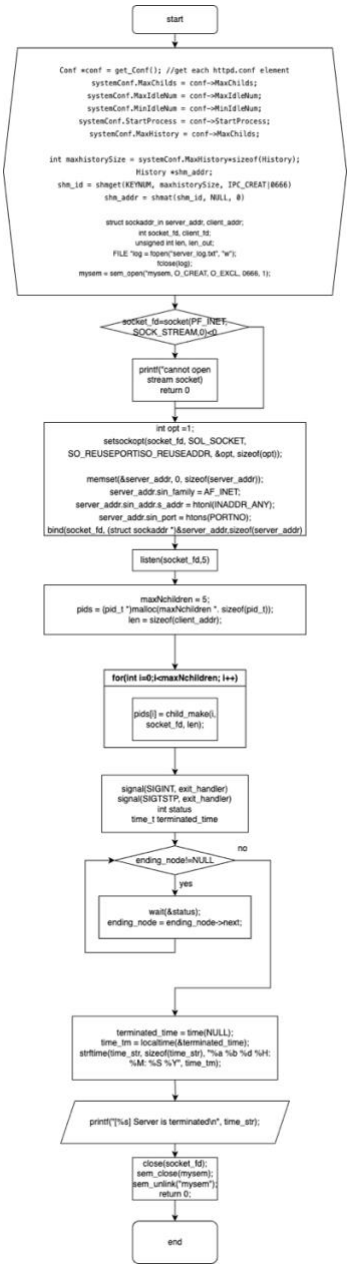
chile_make()함수는 새로운 자식 프로세스를 생성하는 함수로 이번 코드에서는 각 함수들이 linked list로 연결되도록 설계하였다. 자식프로세스는 phtread를 제작하여 threadchild 함수를 통해 child_main에 접근하여 실행하도록 인자를 구조체로 정리하여 전달한다.

R. exit_handler



SIGINT가 부모프로세스에 전달되었을 때 모든 자식 프로세스를 빠짐 없이 죽이도록 설계한 함수이다.

S. main



기본 socket과 bind_listen을 진행하며, 초기에는 공유메모리를 선언하고 초기화한다. SIGINT 신호가 주어졌을 때, 자식프로세스 모두 죽이고 모두 죽인 후에 서버를 닫도록 한다. 이는 wait으로 구현하였다. 또한 이번 과제를 위해 sem_open을 통해 semaphore를 열고 server_log.txt를 만들고 닫아 초기화하였다.

3. Pseudo Code

A. write_log

```
sem_wait(mysem)
FILE *log = fopen("server_log.txt ,"a");
fprintf(log , "%s", arg);
fclose(log)
sem_post(mysem)
return;
```

B. get_Conf

```
Conf *conf = (Conf*)malloc(sizeof(Conf));
FILE *file = fopen(HTTPD_CONF, "r");
while fgets(line, sizeof(line), file)!=NULL{
{
        strtok to get conf value
        get MaxChilds
        get MaxIdleNum
        get MinIdleNum
        get StartProcess
        get MaxHistory
}
fclose(file);
return conf;
```

C. add_history

```
History *shm_addr;
shm_addr = shmat(shm_id, NULL, 0)
pthread_mutex_lock()
remove_first_value();
int tail = 0;
int num =0;
for(int i=0; i<systemConf.MaxHistory;i++){
        get tail
}
add shm_addr[tail].Value = Value

pthread_mutex_unlock()
```

D. alarm_handler

```
printf(alarm handler format);
```

```
int MaxHistory = systemConf.MaxHistory
int cnt =0;

History *shm_addr;
shm_addr = shmat(shm_id, NULL, 0)
```

E. compare_WhiteList

```
FILE *file = fopen(ACESSIBLE_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;
```

F. idle_handler

```
get time to time_str[80]
printf("[%s] IdleProcessCount: %d₩n",time_str, --process_cnt);
if(process_cnt<MinIdleNum
        while(process_cnt!=5){
                pids[child_cnt] = child_make(0,Sig_socket, Sig_len);
                printf("[%s] %d Process is forked₩n", time_str,pids[child_cnt]);
                printf("[%s] IdleProcessCount:%d₩n"time_str, ++process_cnt;
                write_log(string)
        }
}
```

G. idle_handler2

```
get time to time_str[80]
printf("[%s] IdleProcessCount: %d₩n",time_str, ++process_cnt);
if(process_cnt > MaxIdleNum){
        struct node*current = head;
        kill(current->pid, SIGKILL);
        wait(&status)
```

```
        printf("[%s] %d Process is terminated₩n", time_str,pids[child_cnt]);
        printf("[%s] IdleProcessCount:%d₩n"time_str, --process_cnt;
        write_log(string)
}
```

H. 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 "-"
```

I. print_UID

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

J. print_GID

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

K. 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
        fprint file, time
```

L. 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
fprint file, file_stat,st_nlink
use print_UID function
use print GID function
if -h option activated
        use print_readableSIZE function
else
        just fprint file, file_stat.st_size
use print_LastModified_time function
fprint file, file name
```

M. 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"
```

N. 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{
                fprint file, 404 error message html
dir_print(path, flag_a, flag_l, url, file)
fclose(file)
```

O. 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("====================================")
        write_log(string);

        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, BUFFSIZE)
if num_bytes <=0
        continue
else if num_bytes > BUFFSIZE
        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("=====================================")
        write_log(string)
        remove(filename)
        continue;
```

P. threadChild

```
childArg *args = (ChildArgs*)arg;
child_main(args->i, args->socketfd, args->addrlen
```

Q. child_make

```
pid_t pid;
if(pid= fork()>0)
        malloc new node struct
        if(head == NULL)
                head = new;
        else{
                struct node *current = head;
                goto end of current;
                current->next = new;
        }
        return pid;
else{
        malloc new args struct;
        get value i, addrlen, socketfd to args
        pthread_t tid;
```

```
        pthread_create(&tid, NULL, &threadChild, args)
        pthread_join(tid,NULL)
}
```

R. exit_handler

```
struct node*killer =head
kill all child process by SIGKILL signal
```

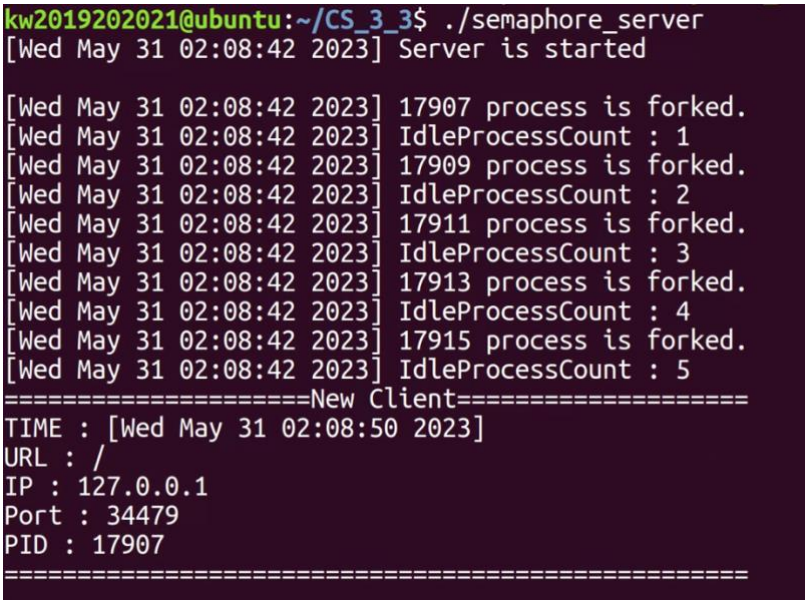
S. main

```
Conf *conf = getconf()
shm_id = shmget(KEYNUM, maxhistorySize, IPC_CREAT|0666)
shm_add = shmat(shm_id, NULL, 0);
parent_pid = getpid();
struct sockaddr_in server_addr, client_addr
int socket_fd, client_fd
unsigned int len, len_out;

mysem = sem_open("mysem", O_CREAT, O_EXCL, 0666, 1);

FILE *log.= fopen("server_log.txt, "w");
fclose(log);

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, exit_handler);
```

```
signal(SIGTERM, exit_handler);
signal(SIGUSR1, idle_handler);
signal(SIGUSR2, idle_handle2);

while (ending_node !- NULL){
        wait(&status);
        ending_node = ending_node->next;
        write_log();
}

printf "server is. terminated");
close(socket_fd);
shmdt(shm_addr)
shmctl(shm_id, IPC_RMID,NULL);

sem_close(mysem)
sem_unlink("mysem);

return(0);
```

각 함수에 대한 설명은 위의 flow chart에서 설명한 것과 동일하다.

4. 결과화면



terminal 화면

```
[Wed May 31 02:08:42 2023] Server is started

[Wed May 31 02:08:42 2023] 17907 process is forked.
[Wed May 31 02:08:42 2023] IdleProcessCount : 1
[Wed May 31 02:08:42 2023] 17909 process is forked.
[Wed May 31 02:08:42 2023] IdleProcessCount : 2
[Wed May 31 02:08:42 2023] 17911 process is forked.
[Wed May 31 02:08:42 2023] IdleProcessCount : 3
[Wed May 31 02:08:42 2023] 17913 process is forked.
[Wed May 31 02:08:42 2023] IdleProcessCount : 4
[Wed May 31 02:08:42 2023] 17915 process is forked.
[Wed May 31 02:08:42 2023] IdleProcessCount : 5
======================New Client====================
TIME : [Wed May 31 02:08:50 2023]
URL : /
IP : 127.0.0.1
Port : 34479
PID : 17907
====================================================
```

server_log.txt 기록

```
[Wed May 31 02:08:52 2023] IdleProcessCount : 4
==================Connection History=====================
NO.     IP              PID     PORT    TIME
1       127.0.0.1       17907   34479   Wed May 31 02:08:50 2023
====================New Client====================
TIME : [Wed May 31 02:08:54 2023]
URL : /
IP : 127.0.0.1
Port : 36015
PID : 17909
==================================================

[Wed May 31 02:08:54 2023] IdleProcessCount : 3
[Wed May 31 02:08:54 2023] 18099 Process is forked
[Wed May 31 02:08:54 2023] IdleProcessCount : 4
[Wed May 31 02:08:54 2023] 18100 Process is forked
[Wed May 31 02:08:54 2023] IdleProcessCount : 5
==============Disconnected Client===============
[Wed May 31 02:08:57 2023]
IP : 127.0.0.1
Port : 34479
PID : 17907
CONNECTING TIME : 1872467(us)
==================================================

[Wed May 31 02:08:57 2023] IdleProcessCount : 6
==============Disconnected Client===============
[Wed May 31 02:08:59 2023]
IP : 127.0.0.1
Port : 36015
PID : 17909
CONNECTING TIME : 1118(us)
==================================================
```

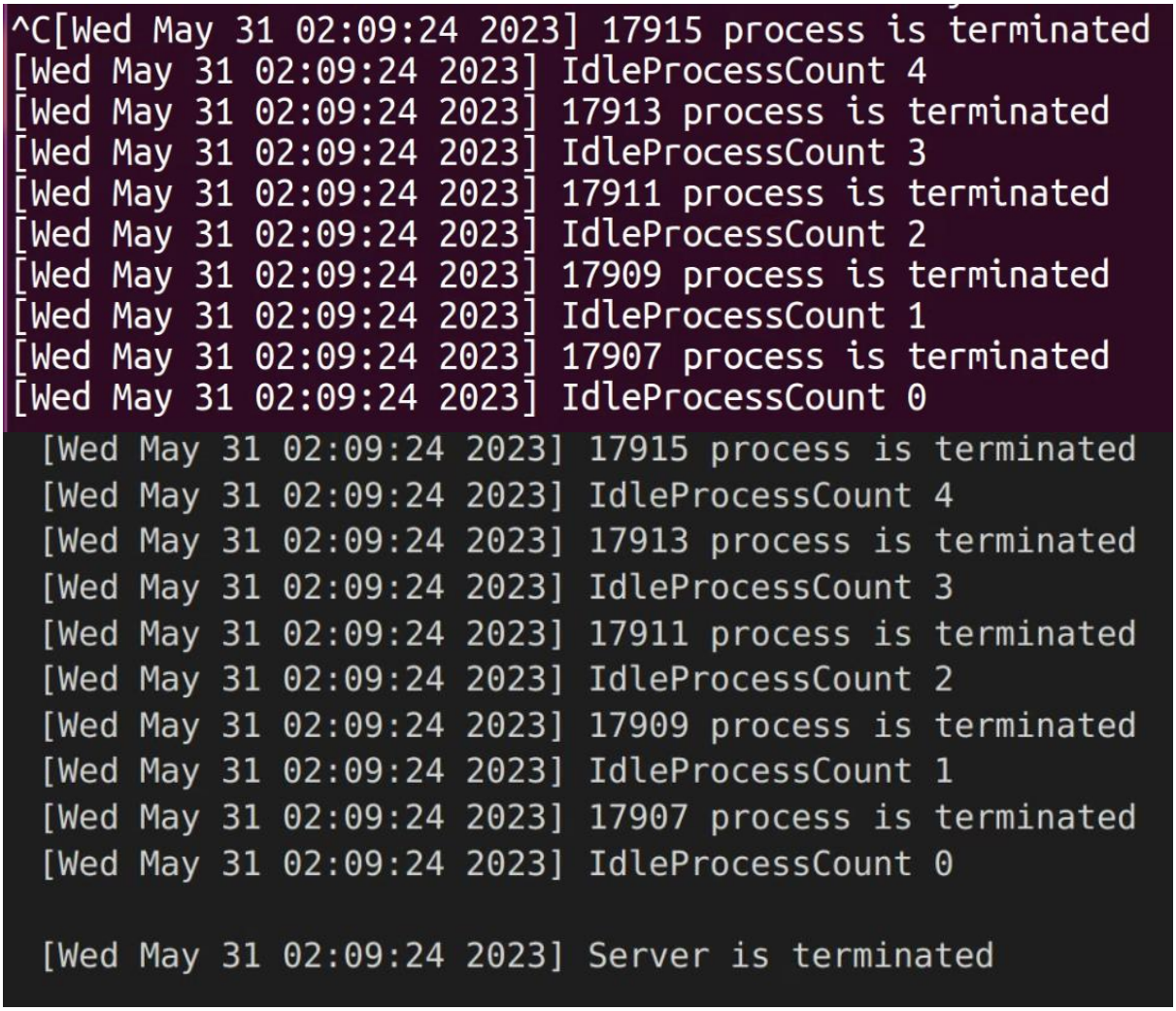
terminal 화면이다. 처음 접근할 때 렉이 걸렸었는데 이에 대한 CONNECTING TIME이 잘 반영된 것을 확인할 수 있다.

```
[Wed May 31 02:08:52 2023] IdleProcessCount : 4
====================New Client====================
TIME : [Wed May 31 02:08:54 2023]
URL : /
IP : 127.0.0.1
Port : 36015
PID : 17909
==================================================

[Wed May 31 02:08:54 2023] IdleProcessCount : 3
[Wed May 31 02:08:54 2023] 18099 Process is forked
[Wed May 31 02:08:54 2023] IdleProcessCount : 4
[Wed May 31 02:08:54 2023] 18100 Process is forked
[Wed May 31 02:08:54 2023] IdleProcessCount : 5
===============Disconnected Client===============
[Wed May 31 02:08:57 2023]
IP : 127.0.0.1
Port : 34479
PID : 17907
CONNECTING TIME : 1872467(us)
==================================================

[Wed May 31 02:08:57 2023] IdleProcessCount : 6
===============Disconnected Client===============
[Wed May 31 02:08:59 2023]
IP : 127.0.0.1
Port : 36015
PID : 17909
CONNECTING TIME : 1118(us)
==================================================
```

server_log.txt 기록으로 terminal과 다르게 history를 작성하여 로그파일에 저장하지 않는다.

```
[Wed May 31 02:08:59 2023] IdleProcessCount : 7
[Wed May 31 02:08:59 2023] 18100 process is terminated
[Wed May 31 02:08:59 2023] IdleProcessCount : 6
[Wed May 31 02:08:59 2023] 18099 process is terminated
[Wed May 31 02:08:59 2023] IdleProcessCount : 5
==================Connection History=====================
NO.     IP              PID     PORT    TIME
1       127.0.0.1       17907   34479   Wed May 31 02:08:50 2023
2       127.0.0.1       17909   36015   Wed May 31 02:08:54 2023
```


```
[Wed May 31 02:08:59 2023] IdleProcessCount : 7
[Wed May 31 02:08:59 2023] 18100 process is terminated
[Wed May 31 02:08:59 2023] IdleProcessCount : 6
[Wed May 31 02:08:59 2023] 18099 process is terminated
[Wed May 31 02:08:59 2023] IdleProcessCount : 5
```



결과를 비교했을 때 지속적으로 histroy를 제외하고는 동일하게 출력하고 있음을 확인할 수 있었다. 또한 각 프로세스 및 쓰레드에서 파일을 작성함에도 꼬이지 않고 정상적으로 순서에 맞게 기록하고 있다.

5. 고찰

이전 과제에 실수로 sleep(5)를 주석 처리하고 제출하여 다시 이를 활성화하였으며 이전에는 html 파일에 대한 remove를 sleep 이후에 하다 보니 새로운 client 접속 시 남아있는 html 파일이 ls 화면에 보이는 문제가 있었으나 앞으로 이동하여 문제를 해결하였다. 이번 과제는 New client일 때 url과 pid를 출력해주고 disconnected client일 때는 자신의 pid와 함께 client와의 접속시간을 출력하도록 바꾸었다.

semaphore로 server_log.txt를 작성할 때는 semaphore에 대한 식별자를 전역변수로 두어 어느 프로세스 어느 쓰레드에서든 open을 새로 하지 않고 이미 존재하는 식별자를 사용하여 wait 와 post를 하도록 하고 모든 자식 프로세스가 종료되고 부모 프로세스도 종료되고 있을 때 semaphore를 닫고, ID에 대해 unlink를 하여 후에 다시 실행하여도 문제가 없도록 한다.

마지막으로 semaphore ID에 대해 문제가 많았는데, 처음에는 mysem으로 진행하였을 때 정상 작동하여 이를 사용하려고 했으나 unlink에 문제가 생긴 것인지 다시 semaphore를 사용할 때 sem_open 함수에서부터 SEM_FAILED가 되고 이를 무시하고 semaphore를 사

용하면 segmentation fault가 발생하였다. 이를 해결하기 위해 terminal에서 semaphore를 강제로 닫아 보았지만 해결되지 않아. ID를 바꿔서 해결하고 이후에는 unlink가 잘 작동하도록 신경 써서 설계하였다.

6. Reference

A. 시스템프로그래밍 실습 Assignment 3-3/광운대학교/ 컴퓨터정보공학부/ 김태석 교수님/2023

B. 시스템프로그래밍 강의자료/광운대학교/컴퓨터정보공학부/김태석교수님 /2023