

# 시스템프로그래밍

과제이름: Assignment 3-2

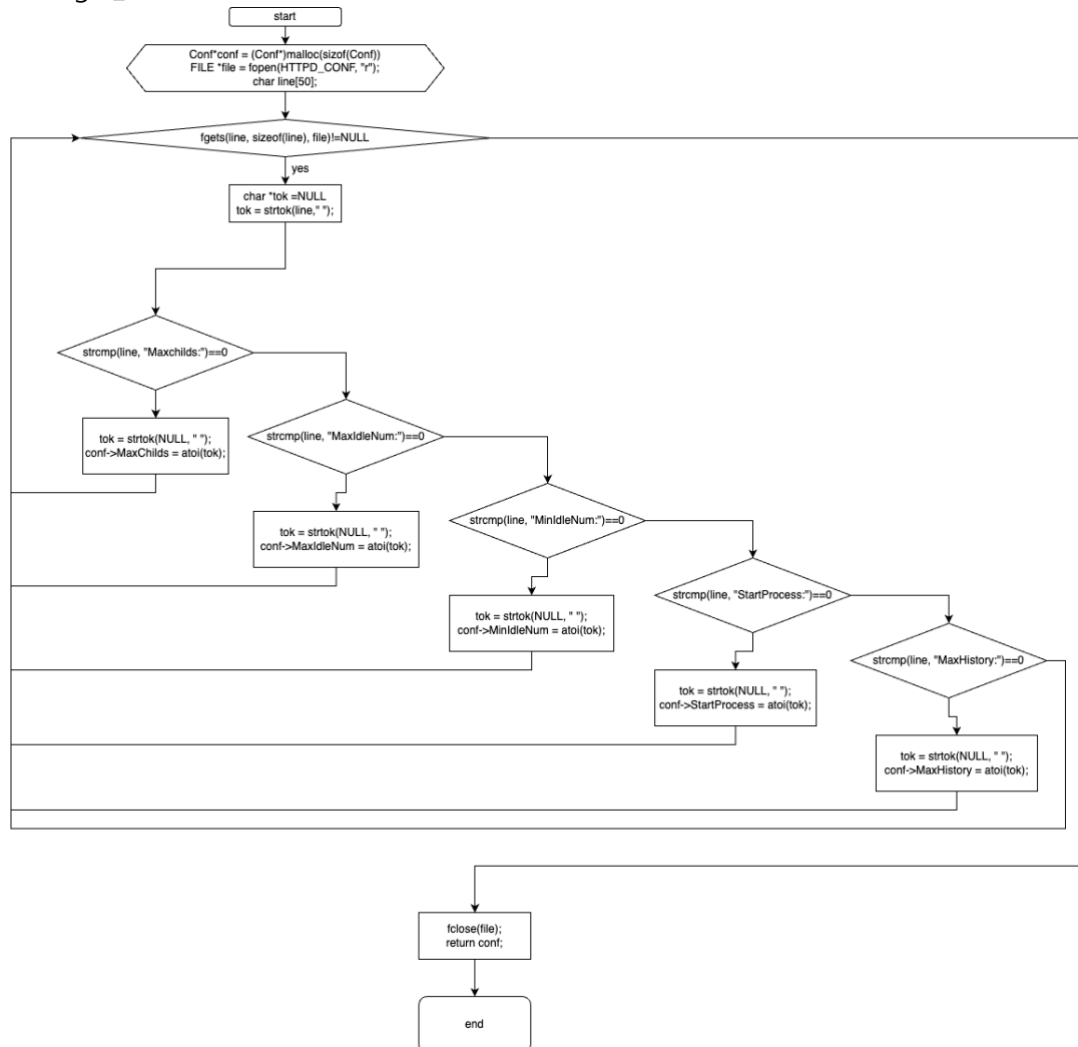
- 담당교수: 김 태 석 교수님
- 학 과: 컴퓨터정보공학부
- 학 번: 2019202021
- 이 름: 정 성 업
- 제출일: 2023/5/24

## 1. Introduction

### A. 과제 소개

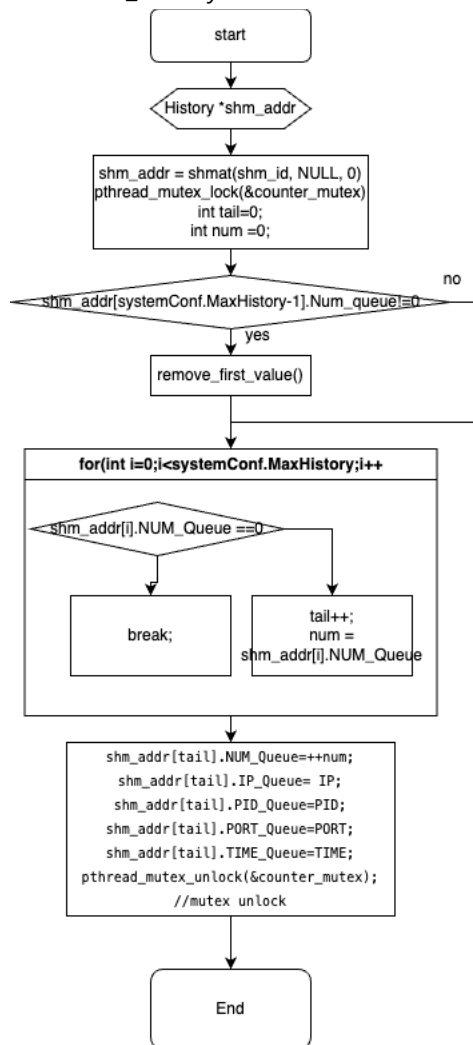
## 2. Flow chart

### A. get\_Conf



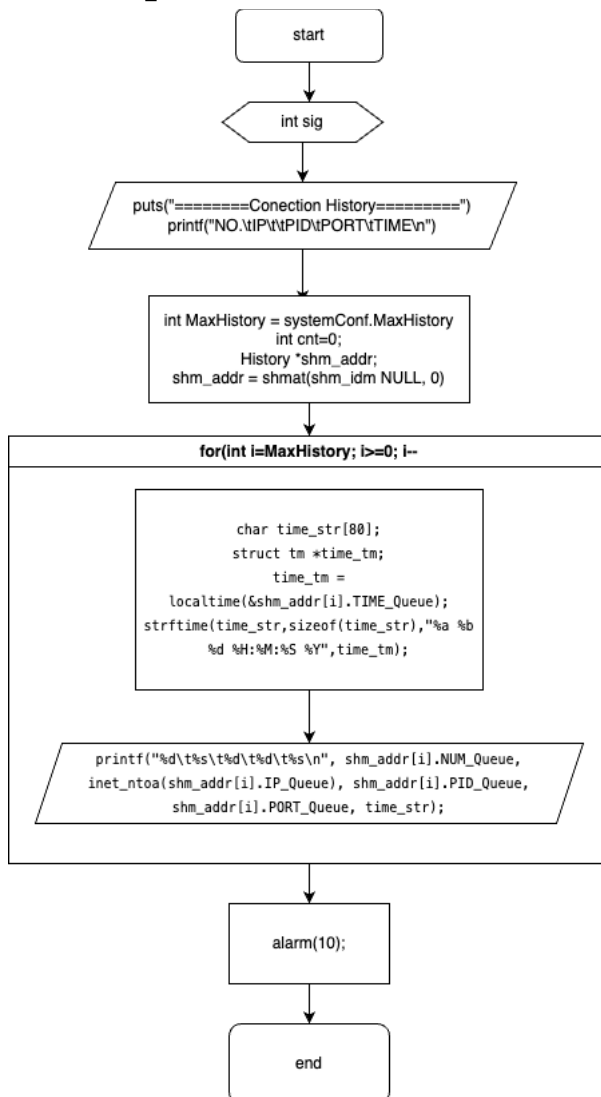
get\_conf()함수는 httpd.conf 파일에 적혀있는 각 요소에 대한 값을 읽고 구조체에 저장한 후 반환하는 함수로 초기에 사용하여 해당 요소 값들을 사용하여 프로그램이 진행되도록 한다.

## B. add\_history



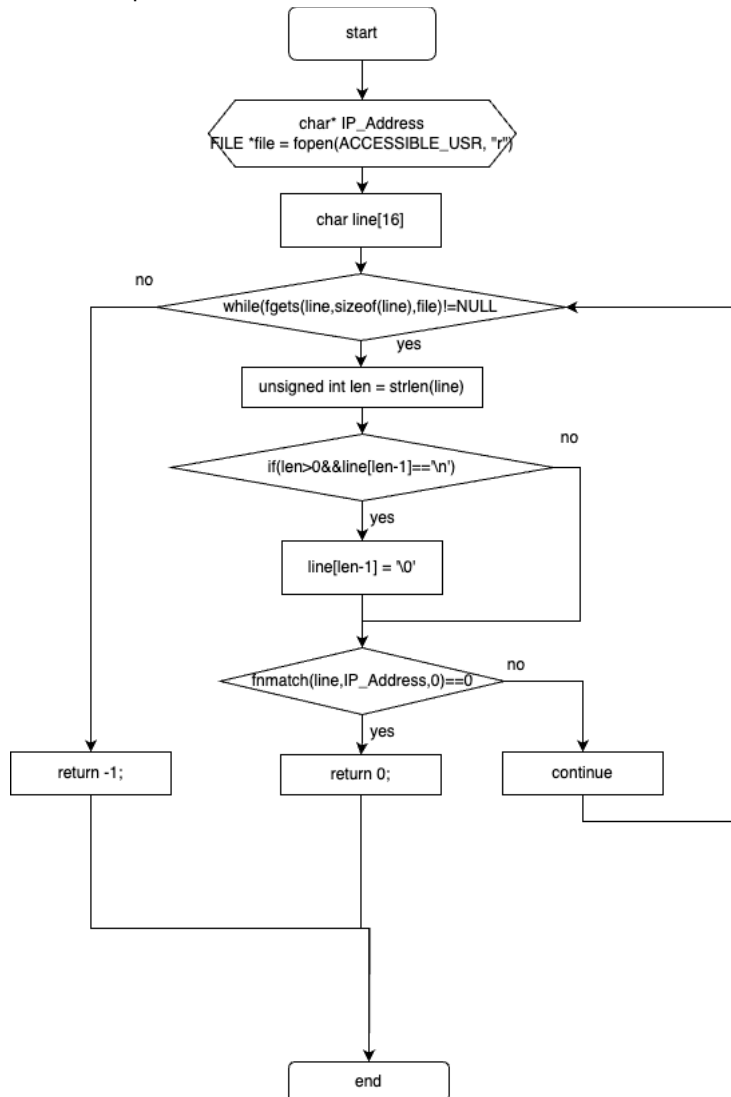
add\_history()함수는 자식 프로세스에서 accept가 발생하고 정상적으로 이뤄진 경우 해당 경우를 공유메모리 구조체인 history에 추가하는 함수이다. 만약 history의 개수가 maxHistory만큼 있다면 첫번째 요소를 삭제한다. 아니라면 공유메모리 구조체 배열에 접근하여 해당 값을 저장한다.

### C. alarm\_handler



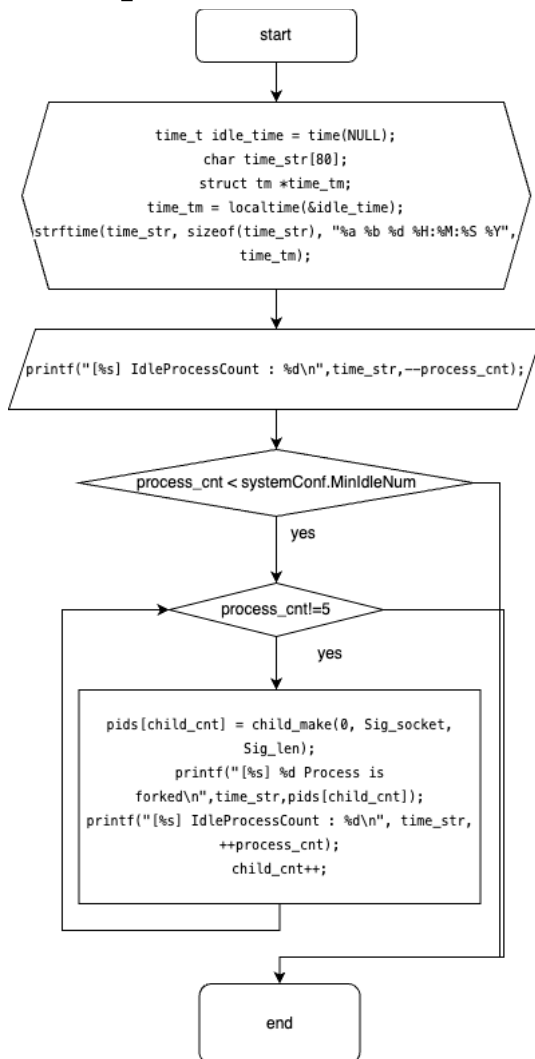
alarm\_handler() 함수는 SIGALRM이 발생했을 때 동작하는 handler 함수로 공유메모리에 있는 값을 가져와 출력하는 동작을 진행한다. 10초마다 작동하며 작동 후에 새로운 alarm을 둔다.

#### D. compare\_WhiteList



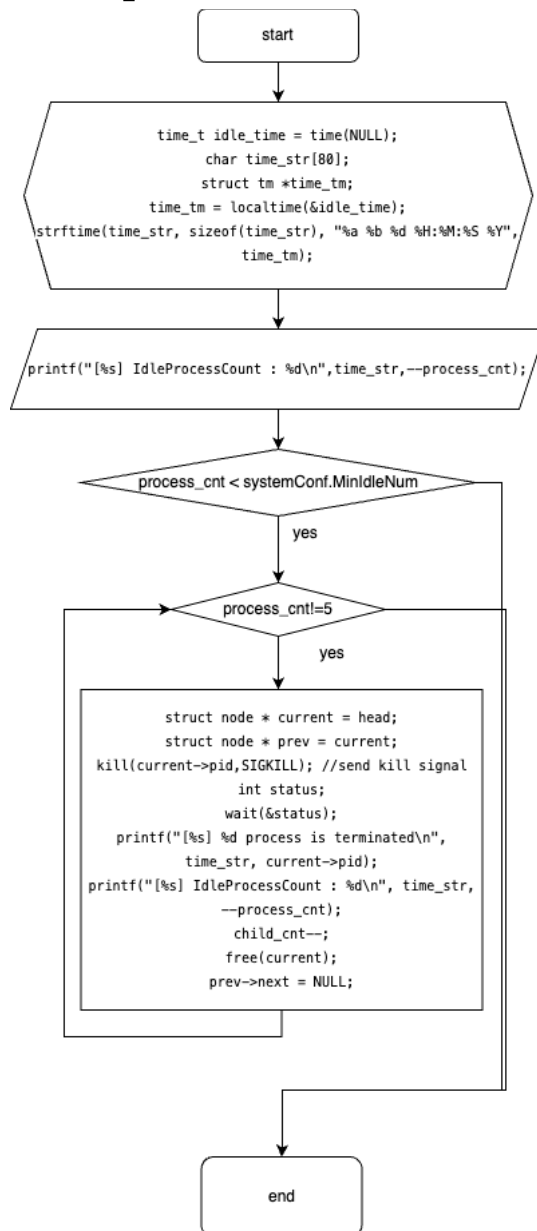
accessible usr에 저장된 whitelist에 대한 정보를 읽고 패턴을 비교하여 일치하는 경우 0을 반환하고 일치하지 않는 경우 -1을 반환한다.

### E. idle\_handler



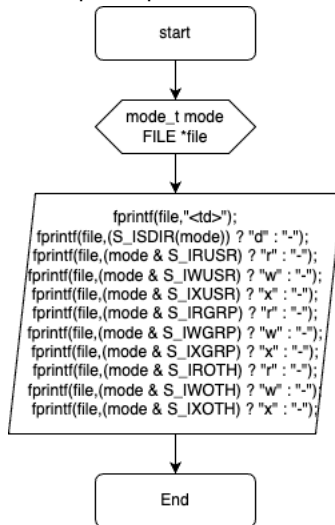
idle\_handler()함수는 add\_history()함수와 마찬가지로 accept 이후에 동작하도록 하며 이는 kill 함수를 통해 자식 프로세스에서 부모프로세스로 SIGUSR1 시그널을 보내서 동작하도록 한다. 프로세스를 사용하면 idleprocess 개수를 줄이고 특정 개수 이하라면 새로운 자식 프로세스를 제작한다.

## F. idle\_handler2



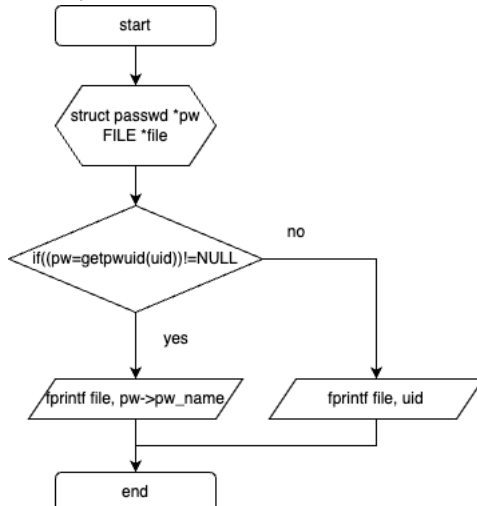
idle\_handler2() 함수는 disconnected client가 출력된 이후에 자식프로세스가 부모 프로세스에게 SIGUSR2 시그널을 보낼 때 동작하는 함수로, 프로세스 동작이 끝나면 idleprocess의 개수를 늘리고, 특정 개수보다 많아진다면 만들어진 프로세스를 삭제한다.

### G. print\_permission



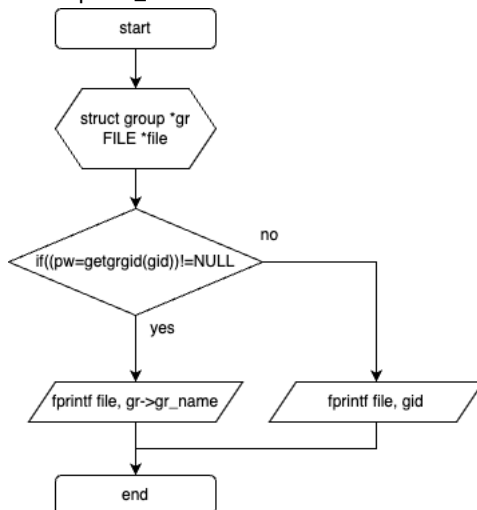
파일 허용정보를 출력하는 함수이다.

### H. print\_UID



파일의 UID 정보를 출력하는 함수이다

### I. print\_GID



파일의 GID 정보를 출력하는 함수이다



```

graph TD
    Start([start]) --> Init{{time_t time;  
char time_str[80];  
struct tm *time_tm  
FILE *file}}
    Init --> Main[time_tm = localtime(&time);  
strftime(time_str, sizeof(time_str), "%b %d %H:%M", time_tm,  
fprinf file, time_str)]
    Main --> End([end])

```

파일의 마지막 수정일자와 시간을 출력하는 함수이다.

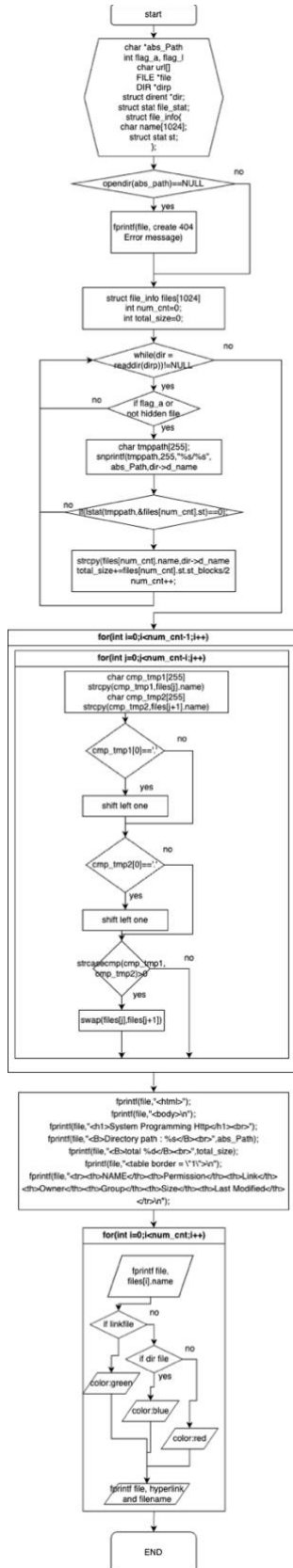
```

graph TD
    Start([start]) --> Init{{char tmpopath[255];  
char cwd[1024];  
int cwd_cnt=0;  
char * tok=NULL}}
    Init --> GetCwd[getcwd(cwd,1024);  
tok=strtok(cwd,"/");]
    GetCwd --> While{while(tok!=NULL)}
    While -- yes --> IncCnt[cwd_cnt++;  
tok = strtok(NULL,"/");]
    IncCnt --> While
    While -- no --> IfStrcmp{if(strcmp(dir_name,"")==0)}
    IfStrcmp -- yes --> Strcpy[strcpy(tmpopath,abs_path);]
    IfStrcmp -- no --> Sprintf[snprintf(tmpopath,255,"%s/%s", abs_path, dir_name)]
    Strcpy --> Strtok[tok=strtok(tmpopath,"/")]
    Sprintf --> Strtok
    Strtok --> For[for(int i=0;i<cwd_cnt-1;i++)  
tok = strtok(NULL,"/")]
    For --> FprintfHref[printf(file,"<td><a href=V/s/s\\>%s</a></td>",tok,dir_name)]
    FprintfHref --> PrintPerm[print_permission(file_stat.st_mode)  
fprintf file, file_stat.st_nlink  
print_UID(file_stat.st_uid)  
print_GID(file_stat.st_gid)]
    PrintPerm --> IfShowSize{if show_size}
    IfShowSize -- yes --> PrintReadSize[print_readaSIZE(file_stat.st_size)]
    IfShowSize -- no --> FprintfSize[/fprintf file, file_stat.st_size/]
    PrintReadSize --> PrintLastMod[print_LastModified_time(file_stat.st_mtime)  
fprintf file, name]
    FprintfSize --> PrintLastMod
    PrintLastMod --> End([end])

```

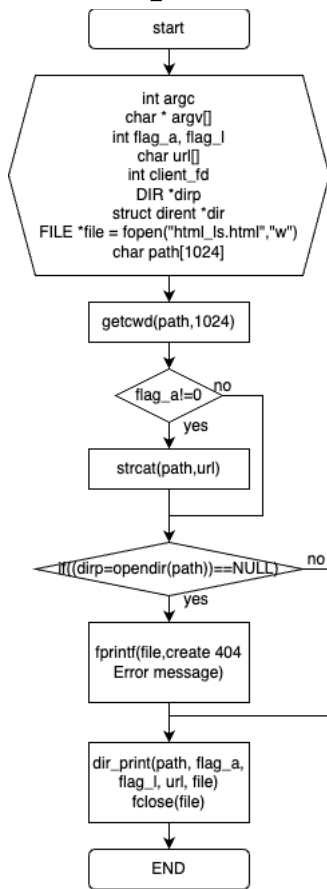
파일의 정보를 long format에 맞춰서 출력하는 함수이다. 현재 working directory와 url 정보를 합쳐서 파일에 접근한다.

## L. dir\_print

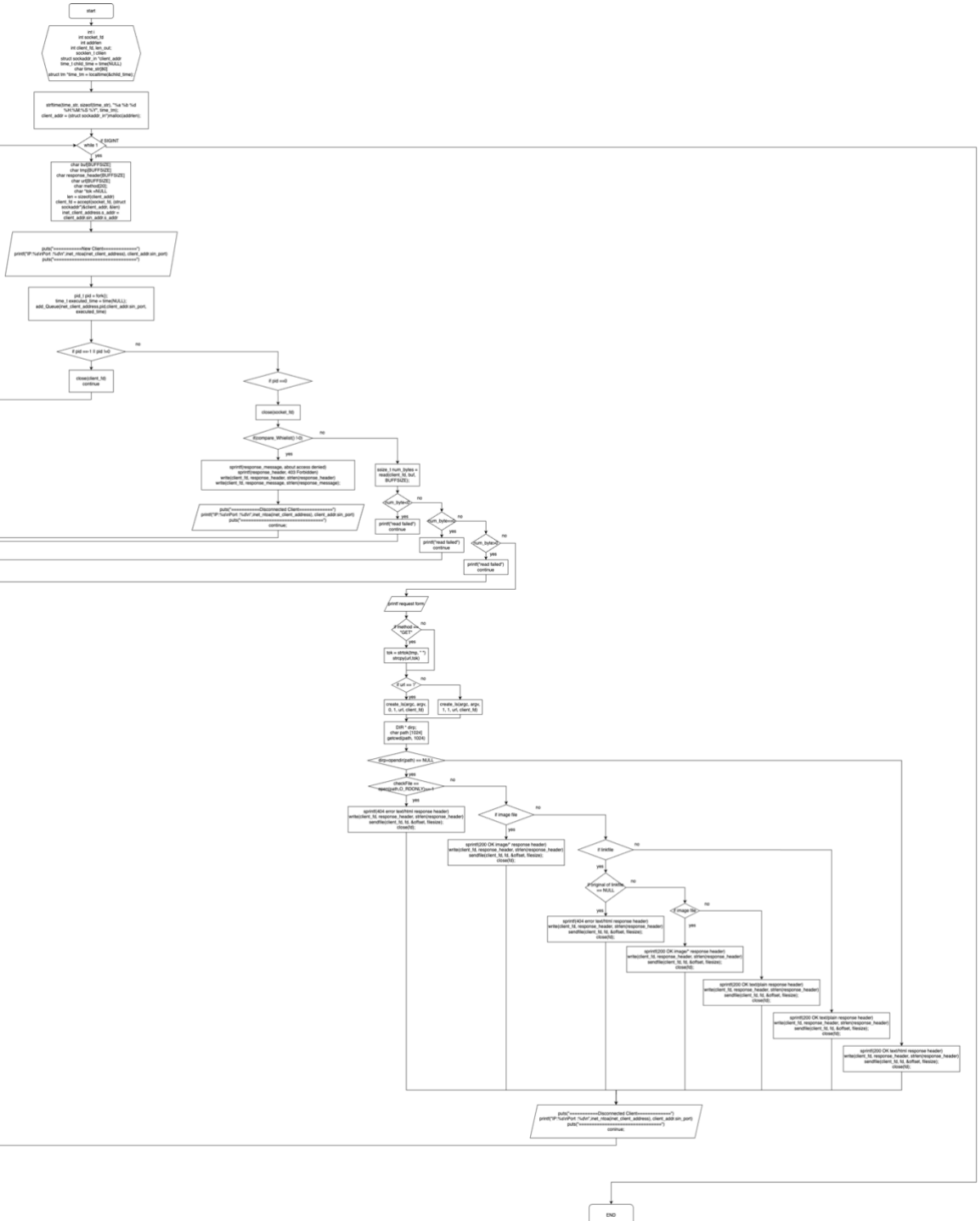


입력 받은 인자가 경로인 경우 실행하는 함수로 없는 경로인 경우에 404 not found html을 작성하고 정상 경로인 경우 ls-l or -al 의 결과를 html 파일에 작성한다.

# M. create\_ls

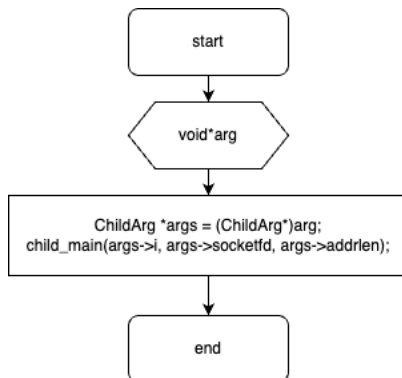


ls를 구현하기 위한 중간 단계의 함수로 directory가 존재하지 않는 경우 404 not found html을 작성하고 아니라면 flag\_a와 flag\_l에 따라 ls의 결과를 작성하는 dir\_print 함수를 호출한다.



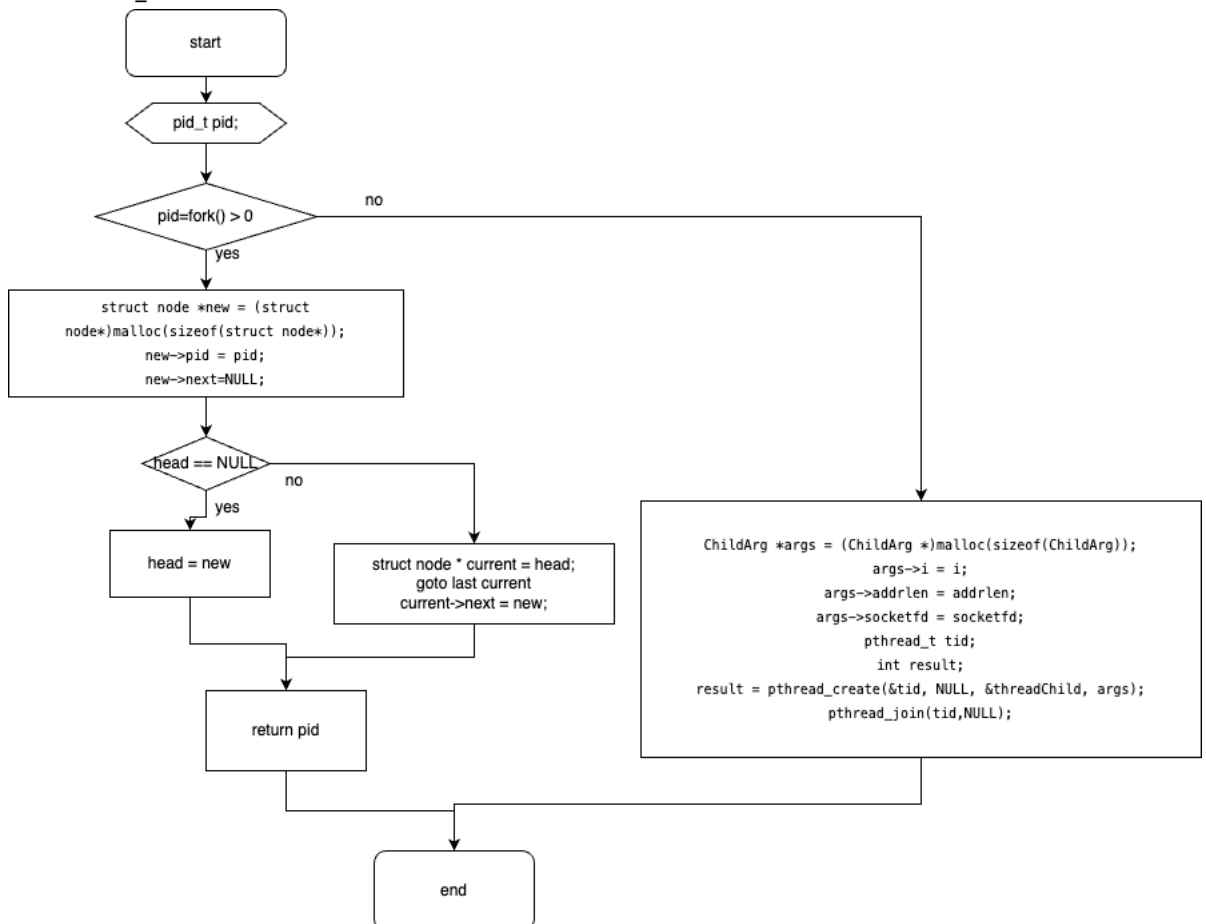
이전 main함수에 있던 코드의 일부로 child 전용으로 함수를 빼두어 continue로 반복한다. SIGINT 시그널을 받았을 때 child process가 종료되도록 설계되었다. accessible\_usr에 있는 접근 가능 IP 주소를 확인 후 패턴과 일치하면 접속하도록 하고 아니라면 403 Forbidden을 보내도록 한다. response는 directory, 파일, 이미지파일, 링크파일 등에 따라 나뉘어서 response\_header를 작성하고 이에 따른 html 파일 또는 response\_message를 client로 보낸다.

### O. threadChild



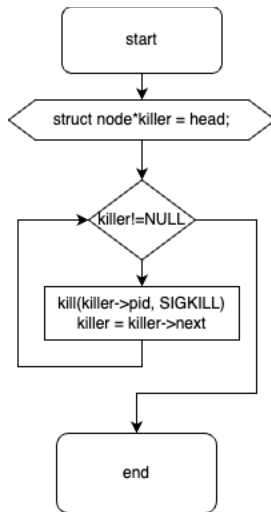
새로 추가된 함수 중 중요한 것들 중 하나로 child\_main을 pthread\_create로 쓰기에는 고려해야할 요소가 많아서 중간에 중계 역할을 하여 pthread도 안정적으로 만들어지고 child\_main도 호출하도록 설계하였다.

### P. child\_make



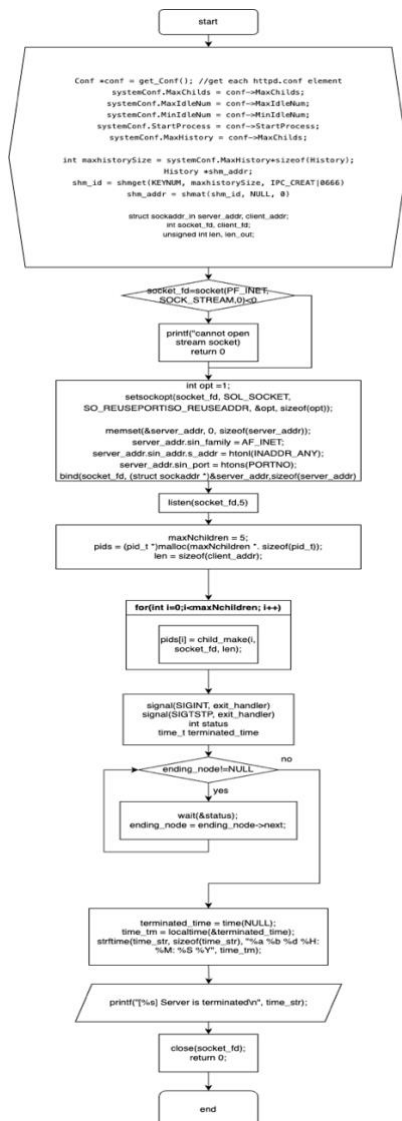
chile\_make()함수는 새로운 자식 프로세스를 생성하는 함수로 이번 코드에서는 각 함수들이 linked list로 연결되도록 설계하였다. 자식프로세스는 pthread를 제작하여 threadchild 함수를 통해 child\_main에 접근하여 실행하도록 인자를 구조체로 정리하여 전달한다.

## Q. exit\_handler



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

## R. 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.

### 3. Pseudo Code

#### A. 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;
```

#### B. 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()
```

#### C. alarm\_handler

```
printf(alarm handler format);
int MaxHistory = systemConf.MaxHistory
int cnt =0;
```

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

D. 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;
```

E. 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;
    }
}
```

F. 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;
}
```



G. 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, fprint file, "r" else "-"
if have write permission by other, fprint file, "w" else "-"
if have execute permission by other, fprint file, "x" else "-"
```

H. print\_UID

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

I. print\_GID

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

J. 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
```

K. 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

```

#### L. 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"
```

#### M. 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)
```

#### N. 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;

```

#### O. threadChild

```

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

```

#### P. 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)
}

```

#### Q. exit\_handler

```

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

```

R. 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;

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;
}

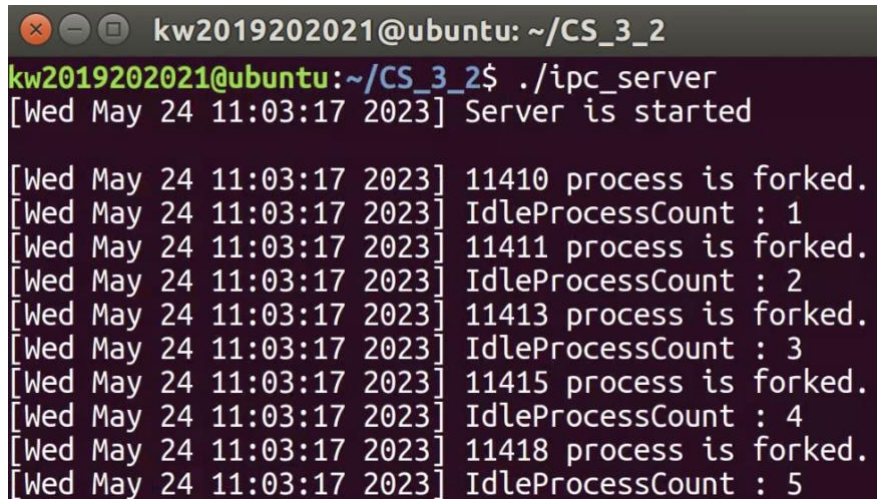
printf "server is. terminated";
close(socket_fd);
shmdt(shm_add)
```

```
shmctl(shm_id, IPC_RMID, NULL);  
  
return(0);
```

각 함수에 대한 설명은 위의 flow chart에서 설명한 것과 동일하다.

#### 4. 결과화면

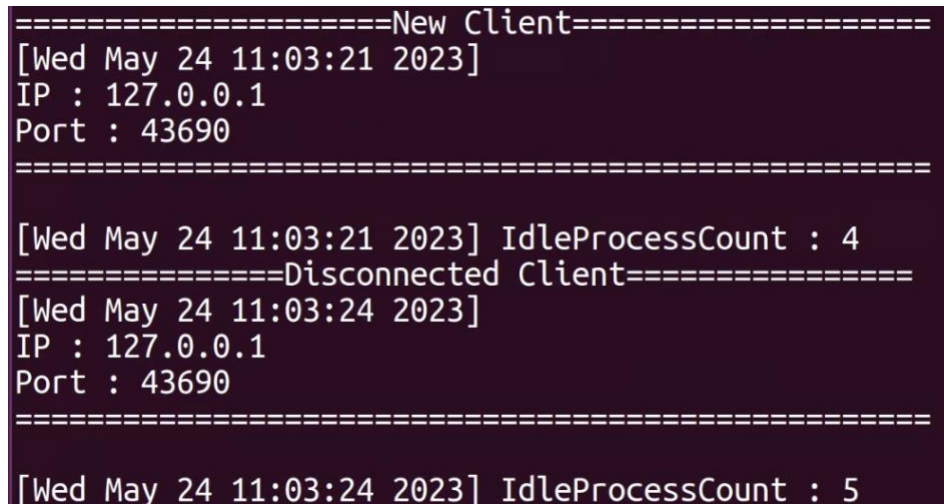
##### A. StartProcess 값 만큼 프로세스 생성하기



```
kw2019202021@ubuntu: ~/CS_3_2  
kw2019202021@ubuntu:~/CS_3_2$ ./ipc_server  
[Wed May 24 11:03:17 2023] Server is started  
  
[Wed May 24 11:03:17 2023] 11410 process is forked.  
[Wed May 24 11:03:17 2023] IdleProcessCount : 1  
[Wed May 24 11:03:17 2023] 11411 process is forked.  
[Wed May 24 11:03:17 2023] IdleProcessCount : 2  
[Wed May 24 11:03:17 2023] 11413 process is forked.  
[Wed May 24 11:03:17 2023] IdleProcessCount : 3  
[Wed May 24 11:03:17 2023] 11415 process is forked.  
[Wed May 24 11:03:17 2023] IdleProcessCount : 4  
[Wed May 24 11:03:17 2023] 11418 process is forked.  
[Wed May 24 11:03:17 2023] IdleProcessCount : 5
```

StartProcess가 5일 때 주어진 만큼 자식 프로세스를 fork한 것을 확인 할 수 있다.

##### B. 일반적으로 client 연결을 한 경우



```
=====New Client=====  
[Wed May 24 11:03:21 2023]  
IP : 127.0.0.1  
Port : 43690  
=====
```

```
[Wed May 24 11:03:21 2023] IdleProcessCount : 4  
=====Disconnected Client=====  
[Wed May 24 11:03:24 2023]  
IP : 127.0.0.1  
Port : 43690  
=====
```

```
[Wed May 24 11:03:24 2023] IdleProcessCount : 5
```

일반적으로 하나씩 연결할 때는 자식 프로세스 추가 없이 idle process에 대한 count만 변한다.

##### C. MinIdleNum > IdleProcessCount인 경우

```

=====New Client=====
[Wed May 24 11:04:10 2023]
IP : 127.0.0.1
Port : 52906
=====

[Wed May 24 11:04:10 2023] IdleProcessCount : 4
=====New Client=====
[Wed May 24 11:04:11 2023]
IP : 127.0.0.1
Port : 53418
=====

[Wed May 24 11:04:11 2023] IdleProcessCount : 3
[Wed May 24 11:04:11 2023] 11656 Process is forked
[Wed May 24 11:04:11 2023] IdleProcessCount : 4
[Wed May 24 11:04:11 2023] 11657 Process is forked
[Wed May 24 11:04:11 2023] IdleProcessCount : 5

```

여러 client에서 server에 접속했을 때 순식간에 idle Process는 지정한 4 미만의 개수를 가지게 되고 이에 대해 idleprocescount가 5개 되도록 새로운 자식 프로세스를 fork하고 idle Process의 개수 또한 증가하는 것을 확인할 수 있다. 이 예시는 sleep()함수를 통해 확인할 수 있다.

D.  $\text{MaxIdleNum} < \text{IdleProcessCount}$ 인 경우

```

=====Disconnected Client=====
[Wed May 24 11:04:13 2023]
IP : 127.0.0.1
Port : 52906
=====

[Wed May 24 11:04:13 2023] IdleProcessCount : 6
=====Disconnected Client=====
[Wed May 24 11:04:14 2023]
IP : 127.0.0.1
Port : 53418
=====

[Wed May 24 11:04:14 2023] IdleProcessCount : 7
[Wed May 24 11:04:14 2023] 11657 process is terminated
[Wed May 24 11:04:14 2023] IdleProcessCount : 6
[Wed May 24 11:04:14 2023] 11656 process is terminated
[Wed May 24 11:04:14 2023] IdleProcessCount : 5

```

프로세스 사용이 종료되면 IdleProcessCount가 증가하는데 지정한 6 초과인 개수를 가지게 된다면 남은 IdleProcess를 종료하고 IdleProcessCount 또한 감소하게 하였다. 삭제하는 순서는 마지막에 fork된 순서로 하도록 하여 초기 preforked 한 프로세스는 간섭하지 않도록 설계하였다.



#### E. history 출력

```
=====Connection History=====
NO.      IP          PID      PORT      TIME
1        127.0.0.1    16829    61614     Wed May 24 23:13:32 2023
2        127.0.0.1    16831    62126     Wed May 24 23:13:32 2023
3        127.0.0.1    16824    62638     Wed May 24 23:13:32 2023
4        127.0.0.1    16825    63150     Wed May 24 23:13:32 2023
5        127.0.0.1    16827    63662     Wed May 24 23:13:32 2023
6        127.0.0.1    16829    64174     Wed May 24 23:13:32 2023
7        127.0.0.1    16831    64686     Wed May 24 23:13:33 2023
8        127.0.0.1    16824    65198     Wed May 24 23:13:33 2023
9        127.0.0.1    16825    175       Wed May 24 23:13:33 2023
10       127.0.0.1    16827    687       Wed May 24 23:13:33 2023
```

지난 history 출력과 다르게 공유메모리에서 history를 관리하기 때문에 각 pid 별로 no가 있는 것이 아닌 전체 프로세스에서의 history 기록을 하여 최신 순으로 숫자를 가지고 있음을 확인할 수 있다. 또한 각 프로세스에서 history를 저장할 때 pthread\_mutex\_lock을 걸어 동기화에 문제 없도록 하였다.

#### F. 종료 출력

```
^C[Wed May 24 11:04:21 2023] 11418 process is terminated
[Wed May 24 11:04:21 2023] IdleProcessCount 4
[Wed May 24 11:04:21 2023] 11415 process is terminated
[Wed May 24 11:04:21 2023] IdleProcessCount 3
[Wed May 24 11:04:21 2023] 11413 process is terminated
[Wed May 24 11:04:21 2023] IdleProcessCount 2
[Wed May 24 11:04:21 2023] 11411 process is terminated
[Wed May 24 11:04:21 2023] IdleProcessCount 1
[Wed May 24 11:04:21 2023] 11410 process is terminated
[Wed May 24 11:04:21 2023] IdleProcessCount 0

[Wed May 24 11:04:21 2023] Server is terminated
kw2019202021@ubuntu:~/CS_3_2$
```

종료 또한 프로세스가 생성된 순서로 종료됨을 확인할 수 있고 IdleProcessCount가 0이 되어 모든 자식 프로세스가 종료됨을 확인할 수 있다.

```
9942 ?      00:00:00 code
9944 ?      00:00:00 code
9958 ?      00:00:00 chrome_crashpad
9975 ?      00:00:12 code
9990 ?      00:00:00 code
10000 ?     00:00:48 code
10039 ?     00:00:06 code
10086 ?     00:00:00 code
10182 ?     00:00:03 update-manager
10195 ?     00:00:01 code
10206 ?     00:00:11 code
10274 ?     00:00:01 gnome-terminal-
10281 pts/4  00:00:00 bash
10302 ?     00:00:04 cpptools
10373 ?     00:00:01 cpptools-srv
11128 ?     00:00:01 kworker/1:1
11168 ?     00:00:00 kworker/u256:2
11204 ?     00:00:00 kworker/u256:0
11283 ?     00:00:02 kworker/1:0
11328 ?     00:00:00 kworker/0:2
11384 ?     00:00:00 kworker/0:0
11421 ?     00:00:07 firefox
11473 ?     00:00:01 Privileged Cont
11518 ?     00:00:00 WebExtensions
11564 ?     00:00:00 Web Content
11612 ?     00:00:00 Web Content
11631 ?     00:00:00 Web Content
11718 ?     00:00:00 kworker/1:2
11723 pts/4  00:00:00 ps
kw2019202021@ubuntu:~/CS_3_2$
```

ps -e의 결과에서도 모든 프로세스가 종료됨을 확인하였다.

#### G. ChildMax를 넘어가는 경우

```
=====
[Wed May 24 11:17:27 2023] IdleProcessCount : 3
[Wed May 24 11:17:27 2023] 11858 Process is forked
[Wed May 24 11:17:27 2023] IdleProcessCount : 4
[Wed May 24 11:17:27 2023] Max child : 10
=====
```

sleep()을 길게 두어 한번에 6개 이상의 client 접속이 있을 때 해당 접속을 할 순 있지만 자식 프로세스가 10개가 넘어가므로 10개가 넘어갈 때는 자식 프로세스를 새로 생성하지 않도록 설계하였다.

## 5. 고찰

이번 Assignment 3-2에서는 수업시간에 배운 semaphore 말고 shared memory를 사용하고 이에 대한 pthread\_mutex\_lock & unlock을 사용하여 여러 프로세스에서 공통된 공용 메모리에 접근하여 history를 저장하고 alarm에 따라 출력하도록 하였다. 이전까지는 동기화를 고려할 필요가 없었기 때문에 critical한 부분이 없었지만 여러 프로세스가 공유 메모리에 접근할 때 동기화가 없으면 메모리 조작에 오류가 생길 수 있기 때문에 자식 프로세스 자체를 프로세스 겸 thread로 두어 자식 프로세스에 접근할 때 mutex\_lock을 걸어 다른 프로세스에서의 접근을 막고 값을 저장한 후에는 mutex\_unlock을 두어 다른 프로세스 및 thread에서 접근할 수 있도록 하였다.

또한 저번 과제와 다르게 SIGUSR라는 커스텀 시그널에 대해 accept할 때와 disconnect 할 때로 두어 accept한 후에는 process를 사용하므로 idleProcessCount를 줄이고 disconnect 할 때는 process 사용이 없어지므로 idleProcessCount를 늘렸다. 또한 한 번에 6개 이상의 client 접속이 일어나면 자식 프로세스 개수가 10개가 넘을 수 있는데 이를 예외로 처리하여 이 때는 새로운 자식 프로세스를 만들지 않도록 하였다. 이번 과제에서 다행이었던 점은 child\_make라는 함수를 통해 자식 프로세스를 만들어왔기 때문에 이런 signal이 발생하여도 추가적으로 많은 코드가 필요없이 함수를 사용하여 구현할 수 있게 되었다. 또한 ls 동작 또한 정상 동작함을 확인하였다.

마지막으로 Makefile을 만들 때 -lpthread를 -lpthread를 글자로서 구별하지 못하여 처음에 대문자 I를 두어 컴파일해서 오류가 발생하여 컴파일러에서 추천하는 -pthread를 옵션으로 하여 이를 기준으로 구현하였으나 소문자 l을 사용한 -lpthread에서도 정상 동작함을 확인하였다.

## 6. Reference

- A. 시스템프로그래밍 실습 Assignment 3-2/광운대학교/ 컴퓨터정보공학부/ 김태석 교수님/2023
- B. 시스템프로그래밍 강의자료/광운대학교/컴퓨터정보공학부/김태석교수님 /2023