컴퓨터학부 20211527 박창현

honey_bee

1. **과제개요**

리눅스 시스템 환경에서 사용자의 홈 디렉토리 하위 특정 디렉토리를 감시(모니터링)하며, 파일 확장자 또는 조건에 따라 주기적으로 자동 정리하는 데몬 프로세스 기반 정리 프로그램을 구현한다. 사용자는 ssu_cleanupd 명령어를 통해 원하는 디렉토리를 등록(add), 수정(modify), 삭제(remove)하고, 감시중인 프로세스를 조회(show)하거나 도움말(help)을 요청할 수 있다.

//

2. 기능(구현한 기능 요약)

-디렉토리 감시 및 정리 기능

-데몬 프로세스 관리 및 등록 해제

-설정 파일/로그 파일 자동 생성 및 관리

-확장자 및 제외 경로 필터링

-로그 최대 줄 수 제한, 중복 처리 방식 선택

파일 기반 설정/로그 관리

```
int read_config(const char *dir_path, DaemonConfig *config, int fd);
int write_config(const char *dir_path, DaemonConfig *config, int fd);
int lock_config(const char *dir_path);
int unlock_config(int fd);
int write_log_entry(const char *dir_path, const char *src, const char *dest, pid_t pid);
int trim_log_file(const char *dir_path, int max_lines);
```

데몬 리스트 관리

```
int add_to_daemon_list(const char *dir_path, pid_t pid);
int remove_from_daemon_list(const char *dir_path);
int is_path_in_daemon_list(const char *dir_path);
int show_daemon_list();
```

파일 정리 기능

```
FileNode* create_file_node(const char *path, const char *name, const char *ext, time_t mtime);
void add_file_node(const char *path, const char *ext, time_t mtime, DaemonConfig *config);
void free_file_list();
void handle_duplicates(FileNode **head, DaemonConfig *config);
void scan_directory(const char *dir_path, DaemonConfig *config);
void organize_files(const char *dir_path, DaemonConfig *config);
```

명령어 핸들링

```
void handle_show();
void handle_add(char *args);
void handle_modify(char *args);
void handle_remove(char *args);
void handle_help();
void handle_exit();
```

유틸리티

```
char *get_user_id();
char *get_home_directory();
char *get_absolute_path(const char *path);
int is_inside_home_directory(const char *path);
int is_directory(const char *path);
int file_exists(const char *path);
char *get_file_extension(const char *filename);
time_t get_file_mtime(const char *path);
char *get_current_time();
void trim_newline(char *str);
int create_directory(const char *path);
int is_subdirectory(const char *parent, const char *child);
```

데몬 실행

```
void run_as_daemon(DaemonConfig config);
```

//

3. 상세설계(함수 및 모듈 구성, 순서도, 구현한 함수 프로토타입 등

(1) handle_add()

[사용자 입력] → [경로 검증] → [중복/하위 디렉토리 검사] →
[설정값 파싱 및 저장] → [출력 디렉토리 생성] →
[fork 후 run_as_daemon 호출 → 백그라운드 실행] → [종료]

(2) run_as_daemon()

[초기 설정파일 읽기] → [루프 시작]
↓
[scan_directory()] → [handle_duplicates()] → [정리 대상 생성]
↓
[파일 copy + 로그 작성] → [로그 줄 수 초과 시 trim]
↓
[time_interval 만큼 sleep] → [루프 반복]

(3) scan_directory()

[디렉토리 열기] → [파일 및 하위 디렉토리 순회]
↓
[제외 경로 여부 검사] → [확장자 필터 확인]
↓
[정리 대상이면 파일 노드 생성 → add_file_node()]

(4) handle_duplicates()

[링크드 리스트 순회] → [중복 파일 검색]

↓

[mode에 따라 최신/오래된/제외 조건에 따라 삭제 or 유지]

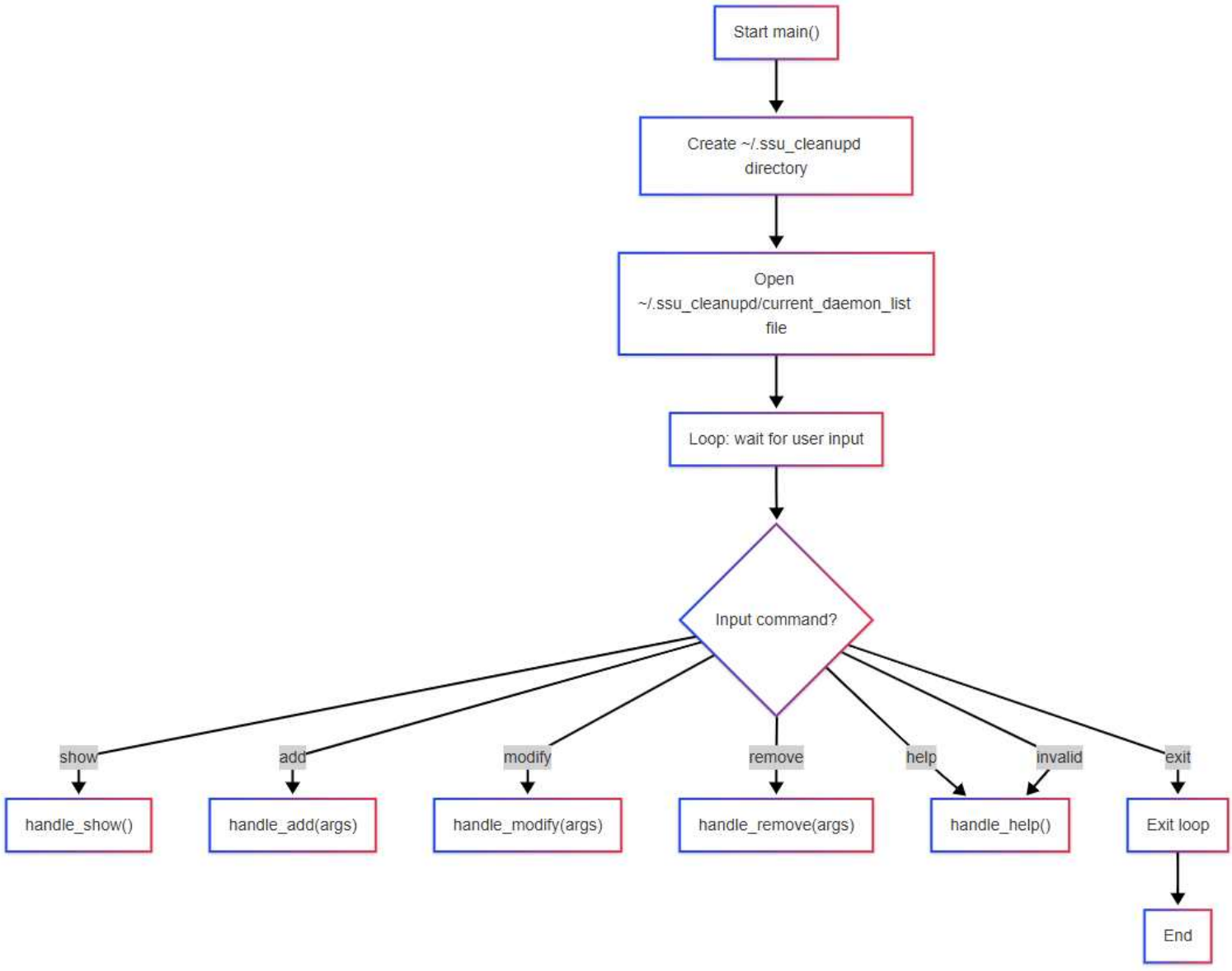
(5) organize_files()

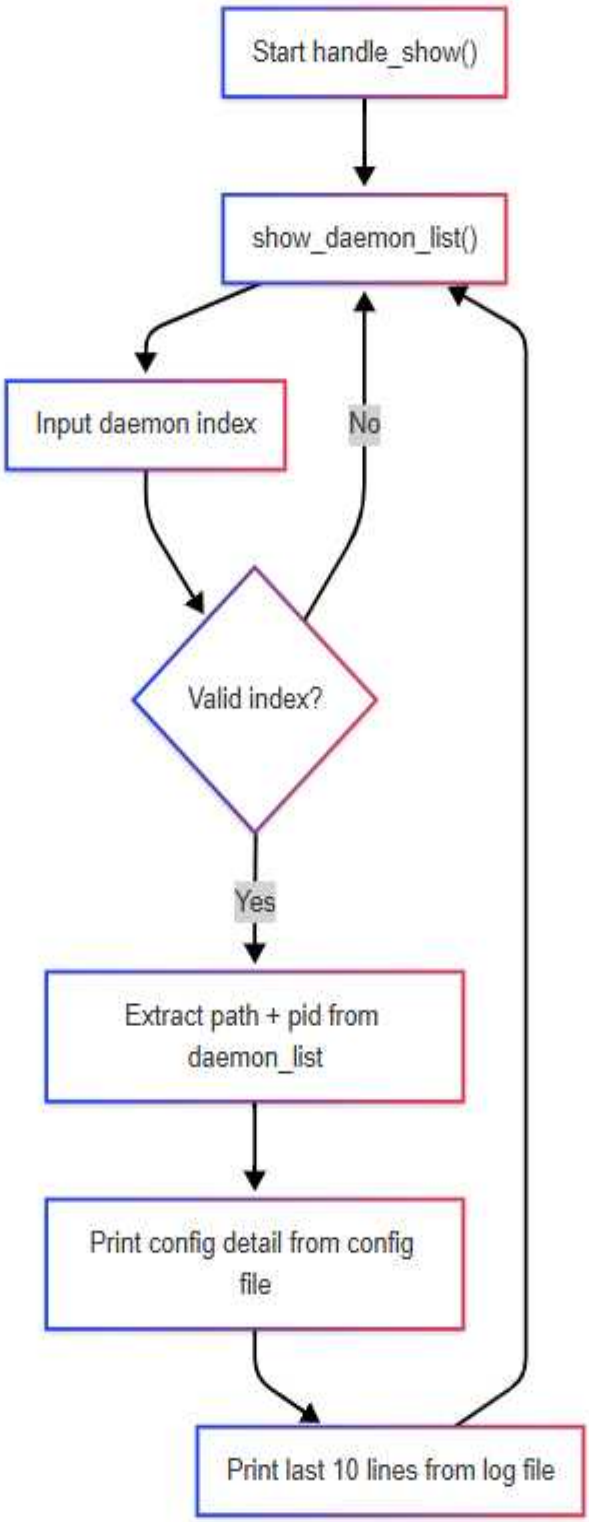
[config 잠금 → read_config] → [scan_directory()] → [handle_duplicates()]

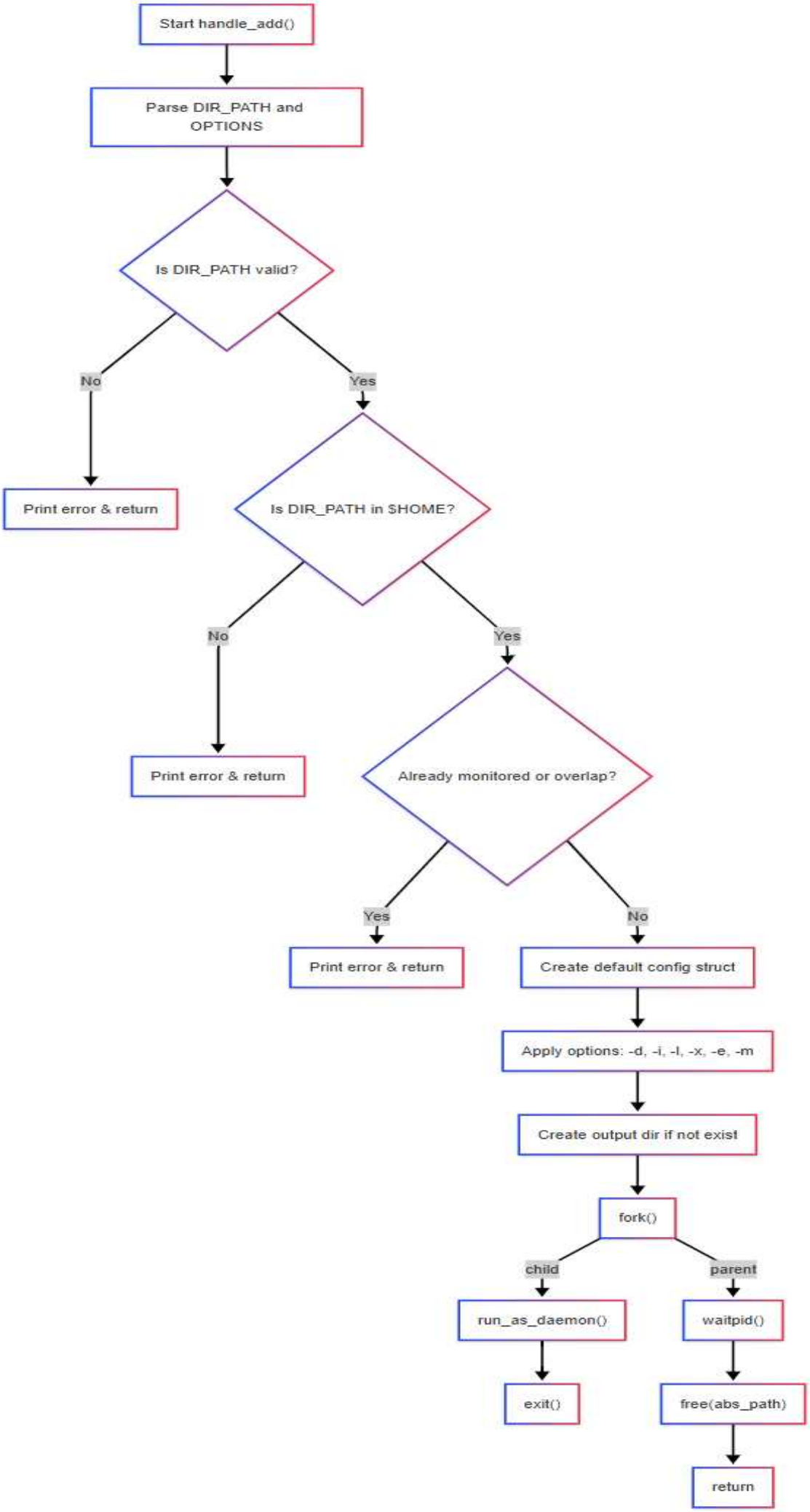
↓

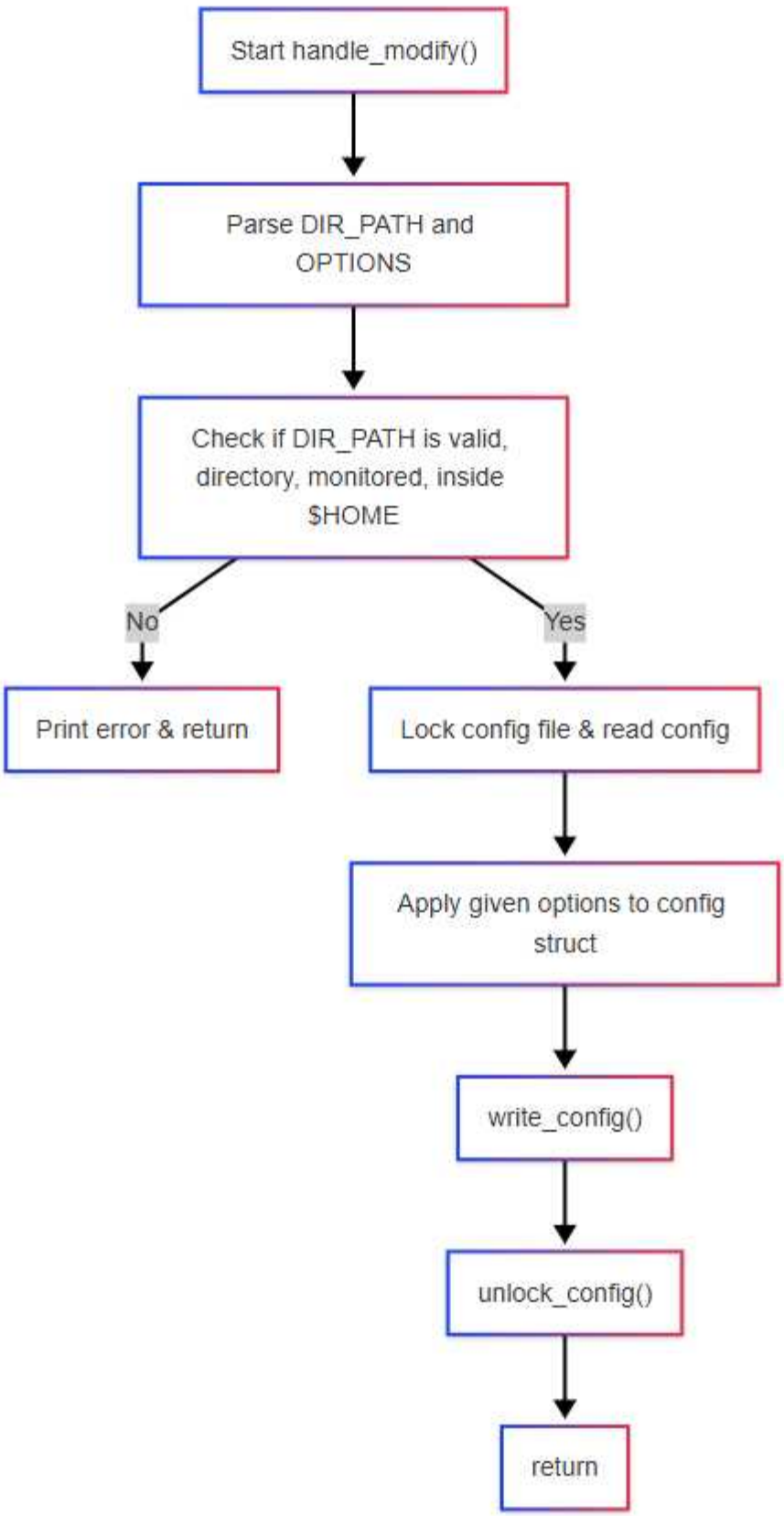
[링크드 리스트 순회 → 파일 복사 및 로그 작성]

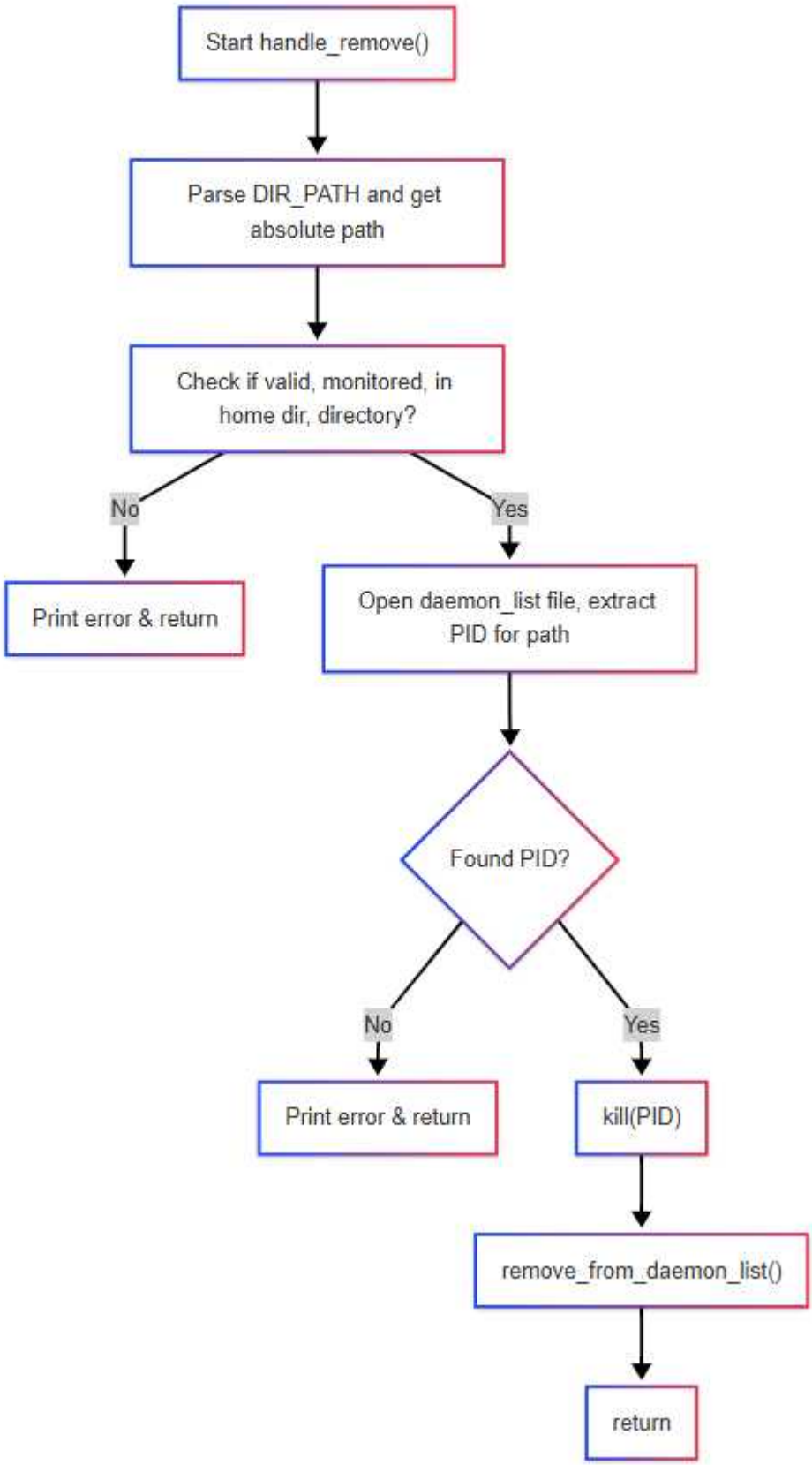
3-1 순서도

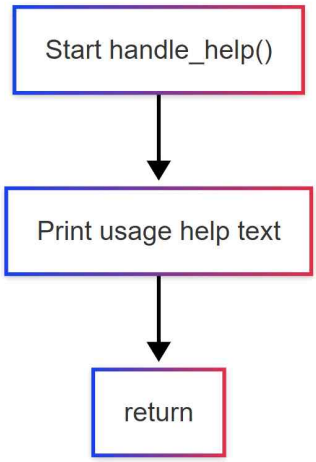


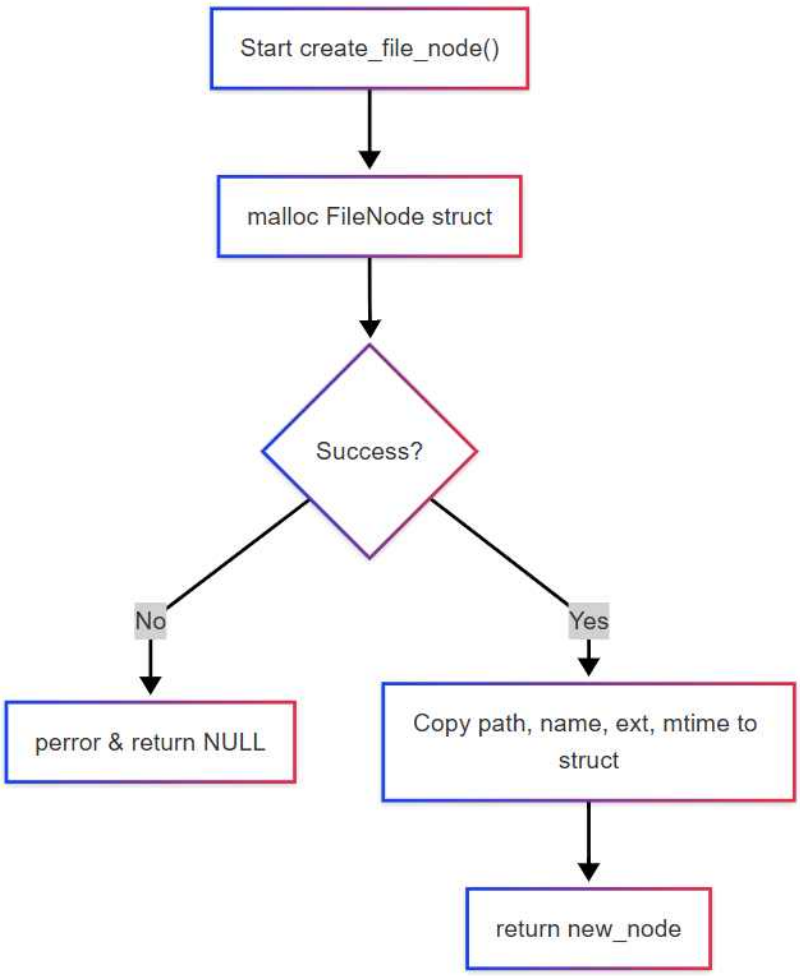

Start handle_show()
show_daemon_list()
Input daemon index
No
Valid index?
Yes
Extract path + pid from daemon_list
Print config detail from config file
Print last 10 lines from log file

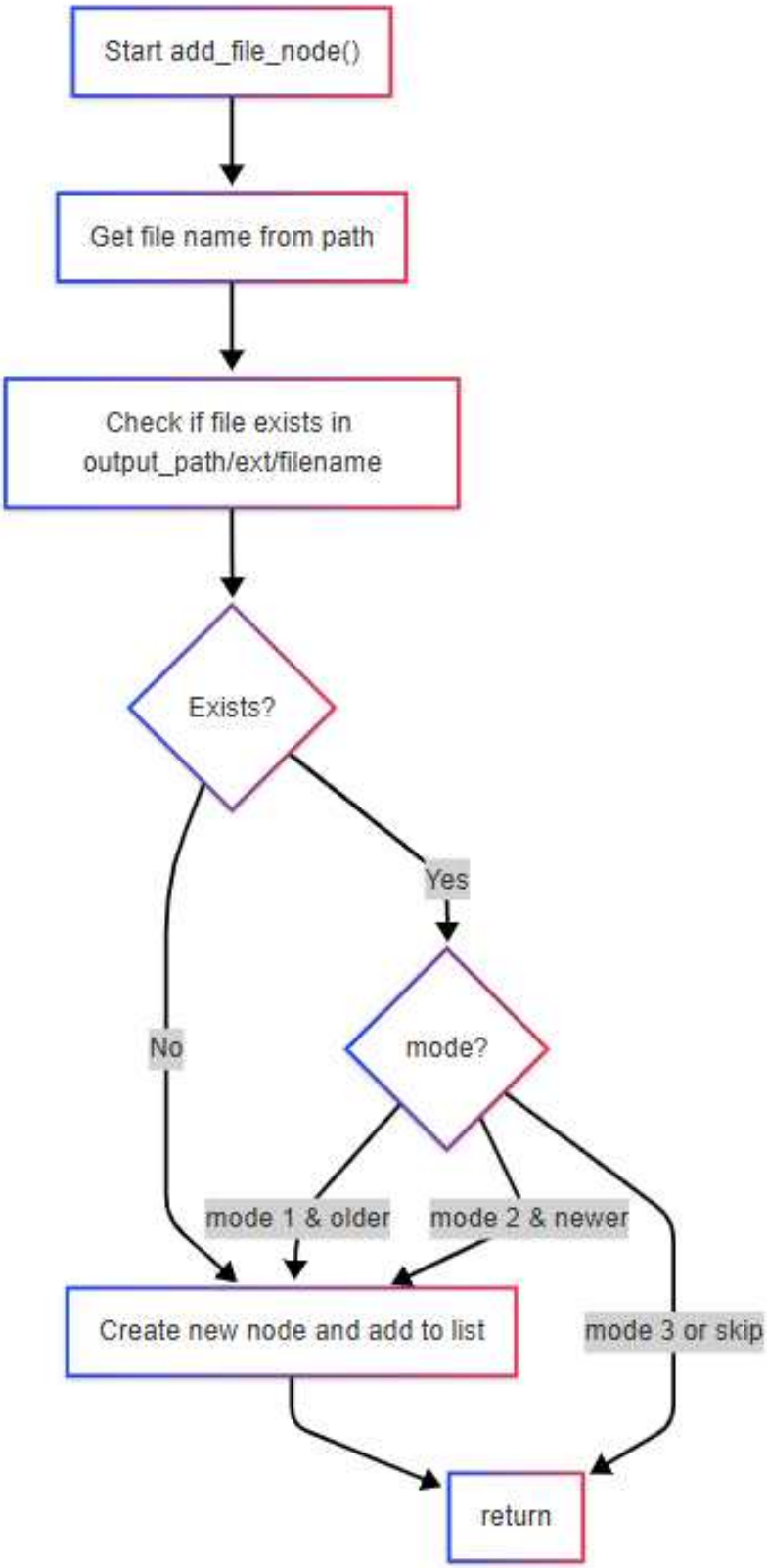

Start handle_add()
Parse DIR_PATH and OPTIONS
Is DIR_PATH valid?
No
Yes
Print error & return
Is DIR_PATH in $HOME?
No
Yes
Print error & return
Already monitored or overlap?
Yes
No
Print error & return
Create default config struct
Apply options: -d, -i, -l, -x, -e, -m
Create output dir if not exist
fork()
child
parent
run_as_daemon()
waitpid()
exit()
free(abs_path)
return

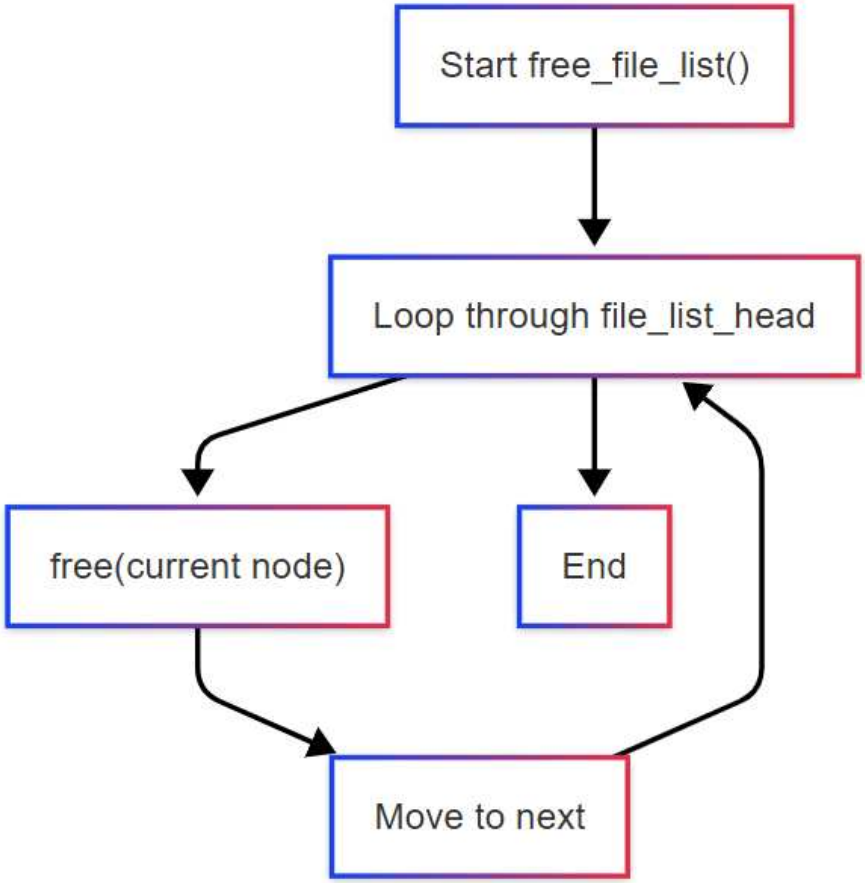

Start handle_modify()
Parse DIR_PATH and OPTIONS
Check if DIR_PATH is valid, directory, monitored, inside $HOME
No
Yes
Print error & return
Lock config file & read config
Apply given options to config struct
write_config()
unlock_config()
return

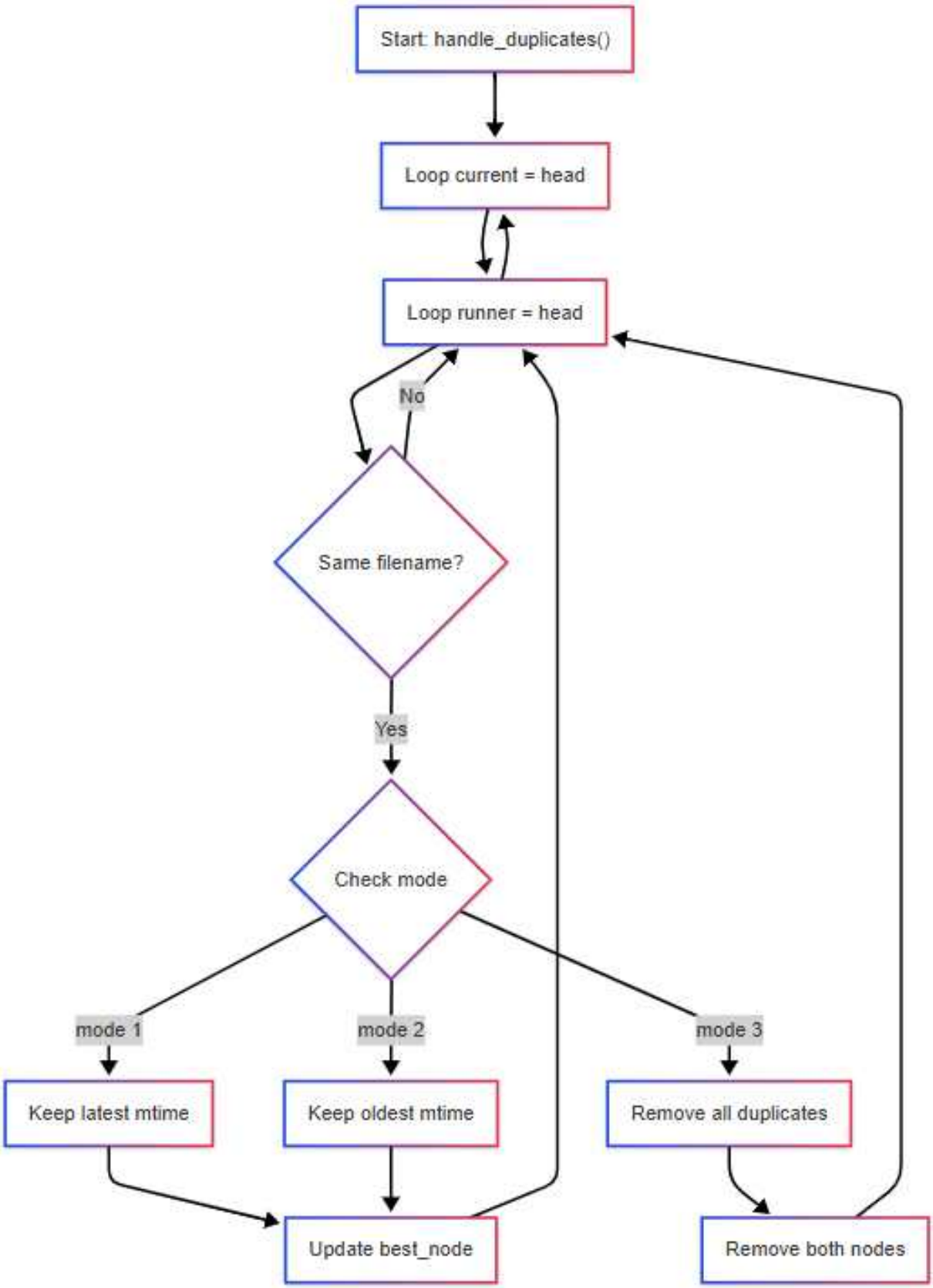

Start handle_remove()
Parse DIR_PATH and get absolute path
Check if valid, monitored, in home dir, directory?
No
Yes
Print error & return
Open daemon_list file, extract PID for path
Found PID?
No
Yes
Print error & return
kill(PID)
remove_from_daemon_list()
return

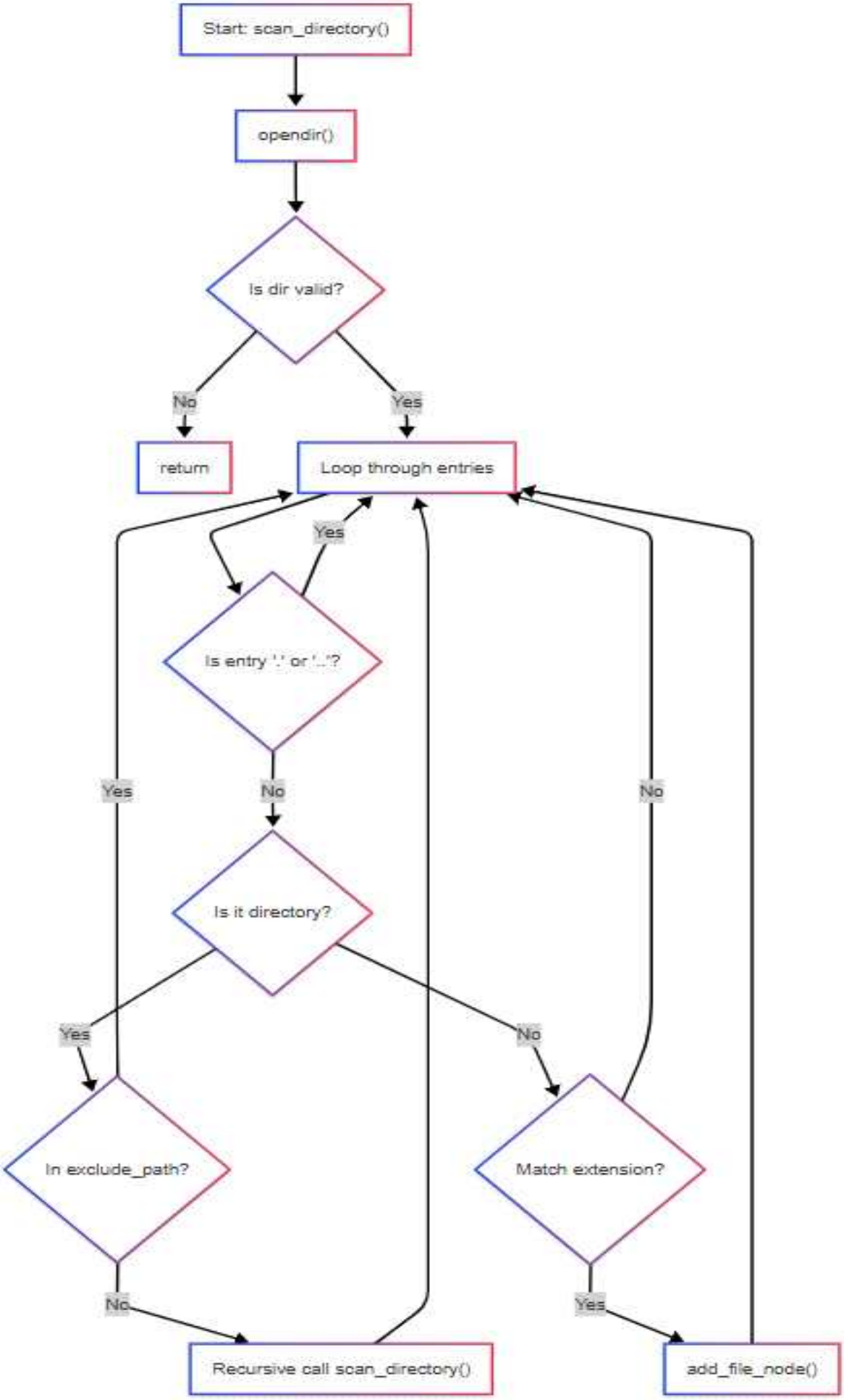

Start handle_help()
Print usage help text
return

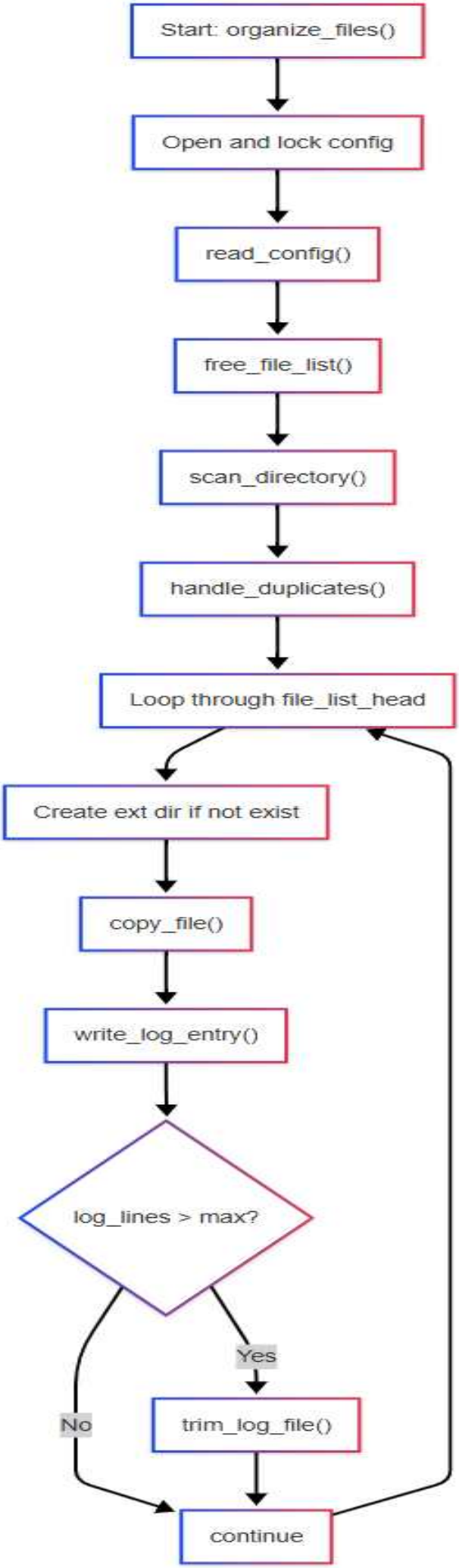

Start create_file_node()
malloc FileNode struct
Success?
No
Yes
perror & return NULL
Copy path, name, ext, mtime to struct
return new_node


Start add_file_node()
Get file name from path
Check if file exists in output_path/ext/filename
Exists?
Yes
No
mode?
mode 1 & older
mode 2 & newer
Create new node and add to list
mode 3 or skip
return


Start free_file_list()
Loop through file_list_head
free(current node)
End
Move to next


Start: handle_duplicates()
Loop current = head
Loop runner = head
No
Same filename?
Yes
Check mode
mode 1
mode 2
mode 3
Keep latest mtime
Keep oldest mtime
Remove all duplicates
Update best_node
Remove both nodes


Start: scan_directory()
opendir()
Is dir valid?
No
Yes
return
Loop through entries
Yes
Is entry '.' or '..'?
Yes
No
No
Is it directory?
Yes
No
In exclude_path?
Match extension?
No
Yes
Recursive call scan_directory()
add_file_node()


Start: organize_files()
Open and lock config
read_config()
free_file_list()
scan_directory()
handle_duplicates()
Loop through file_list_head
Create ext dir if not exist
copy_file()
write_log_entry()
log_lines > max?
Yes
No
trim_log_file()
continue

4. 실행결과(구현한 모든 기능 및 실행 결과 캡쳐)

5-1-0 ssu_cleanupd 실행 시 프롬프트 출력

```
changhyeon@RSP:~/LSP/P2$ ./ssu_cleanupd
20211527>
```

5-1-1 ssu_cleanupd 프로그램 실행 시 사용자의 홈 디렉토리 하위에(~/.ssu_cleanupd/) 디렉토리 및 현재 실행 중인 데몬 프로세스 리스트 파일(~/.ssu_cleanupd/current_deamon_list) 생성

```
changhyeon@RSP:~/LSP/P2$ ./ssu_cleanupd
20211527> exit
changhyeon@RSP:~/LSP/P2$ ls -d ~/.ssu_cleanupd
/home/changhyeon/.ssu_cleanupd
```

5-1-2 프롬프트 상에서 엔터만 입력 시 프롬프트 재출력

```
20211527>
20211527>
```

5-1-3 프롬프트 상에서 지정한 내장명령어 외 기타 명령어 입력 시 help 명령어의 결과를 출력 후 프롬프트 재출력

```
20211527> arrange
Usage:
> show
    <none> : show monitoring daemon process info
> add <DIR_PATH> [OPTION]...
    <none> : add daemon process monitoring the <DIR_PATH> directory
    -d <OUTPUT_PATH> : Specify the output directory <OUTPUT_PATH> where <DIR_PATH> will be arranged
    -i <TIME_INTERVAL> : Set the time interval for the daemon process to monitor in seconds.
    -l <MAX_LOG_LINES> : Set the maximum number of log lines the daemon process will record.
    -x <EXCLUDE_PATH1, EXCLUDE_PATH2, ...> : Exclude all subfiles in the specified directories.
    -e <EXTENSION1, EXTENSION2, ...> : Specify the file extensions to be organized.
    -m <M> : Specify the value for the <M> option.
> modify <DIR_PATH> [OPTION]...
    <none> : modify daemon process monitoring the <DIR_PATH> directory
    -d <OUTPUT_PATH> : Specify the output directory <OUTPUT_PATH> where <DIR_PATH> will be arranged
    -i <TIME_INTERVAL> : Set the time interval for the daemon process to monitor in seconds.
    -l <MAX_LOG_LINES> : Set the maximum number of log lines the daemon process will record.
    -x <EXCLUDE_PATH1, EXCLUDE_PATH2, ...> : Exclude all subfiles in the specified directories.
    -e <EXTENSION1, EXTENSION2, ...> : Specify the file extensions to be organized.
    -m <M> : Specify the value for the <M> option.
> remove <DIR_PATH>
    <none> : remove daemon process monitoring the <DIR_PATH> directory
> help
> exit
20211527>
```

/home/changhyeon/test1 디렉토리 트리 구조

```
changhyeon@RSP:~/LSP/P2$ tree /home/changhyeon/test1
/home/changhyeon/test1
├── A
├── B
└── C
```

5-2-0 show 내장명령어 실행(데몬 프로세스가 파일 2개를 정리했을 때)

```
changhyeon@RSP:~/LSP/P2$ ./ssu_cleanupd
20211527> show
Current working daemon process list

0. exit

Select one to see process info : 0
20211527> add /home/changhyeon/test1/A
20211527> add /home/changhyeon/test1/B
20211527> add /home/changhyeon/test1/C
20211527> show
Current working daemon process list

0. exit
1. /home/changhyeon/test1/A
2. /home/changhyeon/test1/B
3. /home/changhyeon/test1/C

Select one to see process info : abc
Please check your input is valid

0. exit
1. /home/changhyeon/test1/A
2. /home/changhyeon/test1/B
3. /home/changhyeon/test1/C

Select one to see process info : 1
```

```
Select one to see process info : 1

1. config detail
monitoring_path : /home/changhyeon/test1/A
pid : 4291
start_time : 2025-04-13 15:29:36
output_path : /home/changhyeon/test1/A_arranged
time_interval : 10
max_log_lines : none
exclude_path : none
extension : all
mode : 1

2. log detail

[15:29:36][4291][/home/changhyeon/test1/A/2.txt][/home/changhyeon/test1/A_arranged/txt/2.txt]
[15:29:36][4291][/home/changhyeon/test1/A/1.c][/home/changhyeon/test1/A_arranged/c/1.c]



0. exit
1. /home/changhyeon/test1/A
2. /home/changhyeon/test1/B
3. /home/changhyeon/test1/C

Select one to see process info : 0
20211527> exit
changhyeon@RSP:~/LSP/P2$
```

5-2-1 현재 모니터링을 진행하는 데몬 프로세스의 정보를 확인할 때 주어진 번호 이외의 입력이 주어지는 경우 "Please check your input is valid" 출력 후 데몬 프로세스 리스트 재출력 후 다시 번호 입력

```
0. exit
1. /home/changhyeon/test1/A
2. /home/changhyeon/test1/B
3. /home/changhyeon/test1/C

Select one to see process info : 4
Please check your input is valid


0. exit
1. /home/changhyeon/test1/A
2. /home/changhyeon/test1/B
3. /home/changhyeon/test1/C

Select one to see process info :
```

5-3-0 add 명령어 -d, -i, -l 옵션 실행 결과

```
changhyeon@RSP:~/LSP/P2$ mkdir /home/changhyeon/test2/A
changhyeon@RSP:~/LSP/P2$ mkdir /home/changhyeon/test2/A_output
changhyeon@RSP:~/LSP/P2$ tree /home/changhyeon/test2
/home/changhyeon/test2
├── A
└── A_output

3 directories, 0 files
changhyeon@RSP:~/LSP/P2$ ./ssu_cleanupd
20211527> add /home/changhyeon/test2/A -d /home/changhyeon/test2/A_output -i 15 -l 3
20211527> exit
changhyeon@RSP:~/LSP/P2$ tree /home/changhyeon/test2
/home/changhyeon/test2
├── A
│   ├── ssu_cleanupd.config
│   └── ssu_cleanupd.log
└── A_output

3 directories, 2 files
```

```
changhyeon@RSP:~/LSP/P2$ cat /home/changhyeon/test2/A/ssu_cleanupd.config
monitoring_path : /home/changhyeon/test2/A
pid : 4390
start_time : 2025-04-13 15:34:57
output_path : /home/changhyeon/test2/A_output
time_interval : 15
max_log_lines : 3
exclude_path : none
extension : all
mode : 1
changhyeon@RSP:~/LSP/P2$ touch /home/changhyeon/test2/A/1.txt
changhyeon@RSP:~/LSP/P2$ cat /home/changhyeon/test2/A/ssu_cleanupd.log
[15:35:57][4390][/home/changhyeon/test2/A/1.txt][/home/changhyeon/test2/A_output/txt/1.txt]
changhyeon@RSP:~/LSP/P2$ touch /home/changhyeon/test2/A/2.c
changhyeon@RSP:~/LSP/P2$ touch /home/changhyeon/test2/A/3.c
changhyeon@RSP:~/LSP/P2$ touch /home/changhyeon/test2/A/4.c
changhyeon@RSP:~/LSP/P2$ cat /home/changhyeon/test2/A/ssu_cleanupd.log
[15:36:57][4390][/home/changhyeon/test2/A/2.c][/home/changhyeon/test2/A_output/c/2.c]
[15:36:57][4390][/home/changhyeon/test2/A/4.c][/home/changhyeon/test2/A_output/c/4.c]
[15:36:57][4390][/home/changhyeon/test2/A/3.c][/home/changhyeon/test2/A_output/c/3.c]
changhyeon@RSP:~/LSP/P2$
```

5-3-1 add 명령어 -x, -e 옵션 실행 결과

```
changhyeon@RSP:~/LSP/P2$ mkdir ../../test3
changhyeon@RSP:~/LSP/P2$ mkdir ../../test3/A
changhyeon@RSP:~/LSP/P2$ mkdir ../../test3/B
changhyeon@RSP:~/LSP/P2$ mkdir ../../test3/B/B1
changhyeon@RSP:~/LSP/P2$ mkdir ../../test3/C
changhyeon@RSP:~/LSP/P2$ tree ../../test3
../../test3
├── A
├── B
│   └── B1
└── C

5 directories, 0 files
changhyeon@RSP:~/LSP/P2$ ls -d ../../test3_arranged
ls: cannot access '../../test3_arranged': No such file or directory
changhyeon@RSP:~/LSP/P2$ ./ssu_cleanupd
20211527> add ../../test3 -x /home/changhyeon/test3/B/B1 ../../test3/C -e txt
20211527> exit
changhyeon@RSP:~/LSP/P2$ ls -d ../../test3_arranged
../../test3_arranged
changhyeon@RSP:~/LSP/P2$ tree ../../test3
../../test3
├── A
├── B
│   └── B1
├── C
├── ssu_cleanupd.config
└── ssu_cleanupd.log

5 directories, 2 files
```

```
changhyeon@RSP:~/LSP/P2$ cat /home/changhyeon/test3/ssu_cleanupd.config
monitoring_path : /home/changhyeon/test3
pid : 4478
start_time : 2025-04-13 15:46:57
output_path : /home/changhyeon/test3_arranged
time_interval : 10
max_log_lines : none
exclude_path : /home/changhyeon/test3/B/B1,/home/changhyeon/test3/C
extension : txt
mode : 1
changhyeon@RSP:~/LSP/P2$ touch /home/changhyeon/test3/1.txt
changhyeon@RSP:~/LSP/P2$ touch /home/changhyeon/test3/A/2.c
changhyeon@RSP:~/LSP/P2$ touch /home/changhyeon/test3/B/3.txt
changhyeon@RSP:~/LSP/P2$ touch /home/changhyeon/test3/C/4.c
changhyeon@RSP:~/LSP/P2$ touch /home/changhyeon/test3/B/B1/5.txt
changhyeon@RSP:~/LSP/P2$ cat /home/changhyeon/test3/ssu_cleanupd.log
[15:48:07][4478][/home/changhyeon/test3/1.txt][/home/changhyeon/test3_arranged/txt/1.txt]
[15:48:37][4478][/home/changhyeon/test3/B/3.txt][/home/changhyeon/test3_arranged/txt/3.txt]
changhyeon@RSP:~/LSP/P2$
```

5-3-2 add 명령어 -m 옵션 실행결과

```
changhyeon@RSP:~/LSP/P2$ mkdir ../../test4/A
changhyeon@RSP:~/LSP/P2$ mkdir ../../test4/A/A1
changhyeon@RSP:~/LSP/P2$ mkdir ../../test4/A/A2
changhyeon@RSP:~/LSP/P2$ mkdir ../../test4/B
changhyeon@RSP:~/LSP/P2$ mkdir ../../test4/B/B1
changhyeon@RSP:~/LSP/P2$ mkdir ../../test4/B/B2
changhyeon@RSP:~/LSP/P2$ mkdir ../../test4/C
changhyeon@RSP:~/LSP/P2$ mkdir ../../test4/C/C1
changhyeon@RSP:~/LSP/P2$ mkdir ../../test4/C/C2
changhyeon@RSP:~/LSP/P2$ tree ../../test4
../../test4
├── A
│   ├── A1
│   └── A2
├── B
│   ├── B1
│   └── B2
├── C
│   ├── C1
│   └── C2
└── D
    ├── D1
    ├── D2
    └── d.c

13 directories, 1 file
```

```
changhyeon@RSP:~/LSP/P2$ ./ssu_cleanupd
20211527> add ../../test4/D
```

```
20211527> remove /home/changhyeon/test4/D
20211527> exit
changhyeon@RSP:~/LSP/P2$ ls ../../test4/D_arranged/c
d.c
changhyeon@RSP:~/LSP/P2$ ./ssu_cleanupd
20211527> add ../../test4/A -m 1
20211527> add ../../test4/B -m 2
20211527> add ../../test4/C -m 3
20211527> add ../../test4/D -m 3
20211527> exit
changhyeon@RSP:~/LSP/P2$ touch /home/changhyeon/test4/A/A1/a.c
changhyeon@RSP:~/LSP/P2$ touch /home/changhyeon/test4/A/A2/a.c
touch: cannot touch '/home/changhyeon/test4/A/A2/a.c': No such file or directory
changhyeon@RSP:~/LSP/P2$ mkdir ../../test4/A/A2
changhyeon@RSP:~/LSP/P2$ touch /home/changhyeon/test4/A/A2/a.c
changhyeon@RSP:~/LSP/P2$ touch /home/changhyeon/test4/B/B1/b.c
changhyeon@RSP:~/LSP/P2$ touch /home/changhyeon/test4/B/B2/b.c
changhyeon@RSP:~/LSP/P2$ touch /home/changhyeon/test4/C/C1/c.c
changhyeon@RSP:~/LSP/P2$ touch /home/changhyeon/test4/C/C2/c.c
changhyeon@RSP:~/LSP/P2$ touch /home/changhyeon/test4/D/D1/d.c
changhyeon@RSP:~/LSP/P2$ touch /home/changhyeon/test4/D/D2/d.c
changhyeon@RSP:~/LSP/P2$ cat ../../test4/A/ssu_cleanupd.log ../../test4/B/ssu_cleanupd.log ../../test4/C/ssu_cleanupd.log ../../test4/D/ssu_cleanupd.log
[20:39:17][6370][/home/changhyeon/test4/A/A1/a.c][/home/changhyeon/test4/A_arranged/c/a.c]
[20:40:07][6370][/home/changhyeon/test4/A/A2/a.c][/home/changhyeon/test4/A_arranged/c/a.c]
[20:40:15][6371][/home/changhyeon/test4/B/B1/b.c][/home/changhyeon/test4/B_arranged/c/b.c]
[20:37:26][6367][/home/changhyeon/test4/D/d.c][/home/changhyeon/test4/D_arranged/c/d.c]
changhyeon@RSP:~/LSP/P2$
```

5-3-3 add 명령어 예외처리

```
changhyeon@RSP:~/LSP/P2$ ./ssu_cleanupd
20211527> add ~/test999
/home/changhyeon/test999 is not a directory
```

```
changhyeon@RSP:~/LSP/P2$ tree /home/changhyeon/test1/A
/home/changhyeon/test1/A
├── 1.c
├── 2.txt
├── A2
├── ssu_cleanupd.config
└── ssu_cleanupd.log

2 directories, 4 files
```

```
20211527> show
Current working daemon process list

0. exit
1. /home/changhyeon/test1/A

Select one to see process info : 0
20211527> add /home/changhyeon/test1/A
/home/changhyeon/test1/A is already being monitored
20211527> add /home/changhyeon/test1
Directory /home/changhyeon/test1 is already being monitored or overlaps with monitored directory /home/changhyeon/test1/A
20211527> add /home/changhyeon/test1/A/A2
Directory /home/changhyeon/test1/A/A2 is already being monitored or overlaps with monitored directory /home/changhyeon/test1/A
```

```
20211527> add /etc
/etc is outside the home directory
```

```
20211527> add /home/changhyeon/test1/A -d ~/test999
/home/changhyeon/test1/A is already being monitored
20211527> add /home/changhyeon/test1/B -d ~/test999
Invalid output path: ~/test999
20211527> add /home/changhyeon/test1/B -x ~/test999
Invalid exclude path: ~/test999
20211527> add /home/changhyeon/test1/B -d /etc -x /etc
/etc is outside the home directory
```

```
20211527> add /home/changhyeon/test1/C -d /home/changhyeon/test1/C/cc
Output path cannot be a subdirectory of monitoring path
20211527> add /home/changhyeon/test1/C -d ~/test1 -x ~/test1/C
/home/changhyeon/test1/C is not a subdirectory of /home/changhyeon/test1/C
20211527> add /home/changhyeon/test1/C -i 1.2
Invalid time interval: 1.2
20211527> add /home/changhyeon/test1/C -l 1.3
Invalid max log lines: 1.3
20211527> add /home/changhyeon/test1/C -m 4
Invalid mode: 4 (must be between 0 and 3)
```

5-4-0 modify 명령어 실행 결과

```
changhyeon@RSP:~/LSP/P2$ mkdir ../../test5
changhyeon@RSP:~/LSP/P2$ ./ssu_cleanupd
20211527> add ~/test5 -i 5 -l 5
20211527> exit
changhyeon@RSP:~/LSP/P2$ cat ../../test5/ssu_cleanupd.config
monitoring_path : /home/changhyeon/test5
pid : 4244
start_time : 2025-04-15 00:14:42
output_path : /home/changhyeon/test5_arranged
time_interval : 5
max_log_lines : 5
exclude_path : none
extension : all
mode : 1
changhyeon@RSP:~/LSP/P2$ touch /home/changhyeon/test5/a.c
changhyeon@RSP:~/LSP/P2$ ./ssu_cleanupd
20211527> modify ~/test5 -i 10 -e c
20211527> exit
changhyeon@RSP:~/LSP/P2$ cat ../../test5/ssu_cleanupd.config
monitoring_path : /home/changhyeon/test5
pid : 4244
start_time : 2025-04-15 00:14:42
output_path : /home/changhyeon/test5_arranged
time_interval : 5
max_log_lines : 5
exclude_path : none
extension : all
mode : 1
monitoring_path : /home/changhyeon/test5
pid : 4244
start_time : 2025-04-15 00:14:42
output_path : /home/changhyeon/test5_arranged
time_interval : 10
max_log_lines : 5
exclude_path : none
extension : c
mode : 1
```

```
changhyeon@RSP:~/LSP/P2$ touch /home/changhyeon/test5/b.c
changhyeon@RSP:~/LSP/P2$ cat ../../test5/ssu_cleanupd.log
[00:15:28][4244][/home/changhyeon/test5/a.c][/home/changhyeon/test5_arranged/c/a.c]
[00:16:43][4244][/home/changhyeon/test5/b.c][/home/changhyeon/test5_arranged/c/b.c]
changhyeon@RSP:~/LSP/P2$
```

5-4-1 modify 예외처리

```
changhyeon@RSP:~/LSP/P2$ ./ssu_cleanupd
20211527> modify ~/test999
/home/changhyeon/test999 is not a directory
20211527> show
Current working daemon process list

0. exit
1. /home/changhyeon/test5

Select one to see process info : 0
20211527> modify /home/changhyeon/test4
/home/changhyeon/test4 is not being monitored
20211527> modify /etc
/etc is outside the home directory
```

```
0. exit
1. /home/changhyeon/test1/A
2. /home/changhyeon/test1/B

Select one to see process info : 0
20211527> modify ~/test1/B -d /home/changhyeon/test999
Invalid output path: /home/changhyeon/test999
20211527> modify ~/test1/B -x /home/changhyeon/test999
Invalid exclude path: /home/changhyeon/test999
20211527> modify ~/test1/B -d /etc
/etc is outside the home directory
20211527> modify ~/test1/B -x /etc
/etc is outside home directory
20211527> modify ~/test1/B -d ~/test1/B/bb -x ~/test1/B/bb
Output path cannot be a subdirectory of monitoring path
20211527> modify ~/test1/B -i -1
Invalid time interval: -1 (must be positive integer)
20211527> modify ~/test1/B -l 3.14
Invalid max log lines: 3.14 (must be positive integer)
20211527> modify ~/test1/B -m 5
Invalid mode: 5 (must be between 0 and 3)
```

5-5-0 remove 명령어 실행결과

```
changhyeon@RSP:~/LSP/P2$ mkdir ../../test6
changhyeon@RSP:~/LSP/P2$ mkdir ../../test6/A
changhyeon@RSP:~/LSP/P2$ mkdir ../../test6/B
changhyeon@RSP:~/LSP/P2$ ./ssu_cleanupd
20211527> add ~/test6/A
20211527> add ~/test6/B
20211527> show
Current working daemon process list

0. exit
1. /home/changhyeon/test6/A
2. /home/changhyeon/test6/B

Select one to see process info : 0
20211527> remove /home/changhyeon/test6/A
20211527> show
Current working daemon process list

0. exit
1. /home/changhyeon/test6/B

Select one to see process info : 0
20211527> exit
changhyeon@RSP:~/LSP/P2$
```

5-5-1 remove 예외처리

```
changhyeon@RSP:~/LSP/P2$ ./ssu_cleanupd
20211527> show
Current working daemon process list

0. exit
1. /home/changhyeon/test6/B

Select one to see process info : 0
20211527> remove /home/changhyeon/test999
/home/changhyeon/test999 is not a directory
20211527> remove /home/changhyeon/test6/A
/home/changhyeon/test6/A is not being monitored
20211527> remove /etc
/etc is outside the home directory
20211527>
```

5-6-0 help 명령어 실행

```
20211527> help
Usage:
> show
    <none> : show monitoring daemon process info
> add <DIR_PATH> [OPTION]...
    <none> : add daemon process monitoring the <DIR_PATH> directory
    -d <OUTPUT_PATH> : Specify the output directory <OUTPUT_PATH> where <DIR_PATH> will be arranged
    -i <TIME_INTERVAL> : Set the time interval for the daemon process to monitor in seconds.
    -l <MAX_LOG_LINES> : Set the maximum number of log lines the daemon process will record.
    -x <EXCLUDE_PATH1, EXCLUDE_PATH2, ...> : Exclude all subfiles in the specified directories.
    -e <EXTENSION1, EXTENSION2, ...> : Specify the file extensions to be organized.
    -m <M> : Specify the value for the <M> option.
> modify <DIR_PATH> [OPTION]...
    <none> : modify daemon process monitoring the <DIR_PATH> directory
    -d <OUTPUT_PATH> : Specify the output directory <OUTPUT_PATH> where <DIR_PATH> will be arranged
    -i <TIME_INTERVAL> : Set the time interval for the daemon process to monitor in seconds.
    -l <MAX_LOG_LINES> : Set the maximum number of log lines the daemon process will record.
    -x <EXCLUDE_PATH1, EXCLUDE_PATH2, ...> : Exclude all subfiles in the specified directories.
    -e <EXTENSION1, EXTENSION2, ...> : Specify the file extensions to be organized.
    -m <M> : Specify the value for the <M> option.
> remove <DIR_PATH>
    <none> : remove daemon process monitoring the <DIR_PATH> directory
> help
> exit
20211527>
```

5-7-0 exit 명령어 실행

```
20211527> exit
changhyeon@RSP:~/LSP/P2$
```

4. **주석달린 소스코드**(makefile, *.c, *.h 등)

<Makefile>

```
# ssu_cleanupd 프로젝트를 위한 Makefile
# 이 파일은 ssu_cleanupd 데몬의 빌드 규칙과 설정을 정의합니다

# ======================
# 컴파일러 설정
# ======================

# 사용할 컴파일러 (GNU C 컴파일러)
CC = gcc

# 컴파일러 플래그:
# -Wall: 모든 경고 메시지 활성화
# -Wextra: 추가 경고 메시지 활성화
# -g: 디버그 정보 포함
# -I.: 헤더 파일 검색 경로에 현재 디렉토리 추가
CFLAGS = -Wall -Wextra -g -I.

# 링커 플래그 (기본값은 비어 있음)
LDFLAGS =

# 최종 생성할 실행 파일 이름
TARGET = ssu_cleanupd

# ======================
# 소스 파일 설정
# ======================

# 모든 C 소스 파일 목록
SRCS = ssu_cleanupd.c utils.c log.c file_ops.c daemon_process.c daemon_list.c config.c commands.c arrange.c

# 오브젝트 파일 목록 (.c -> .o 변환)
OBJS = $(SRCS:.c=.o)

# 모든 헤더 파일 목록
HEADERS = ssu_cleanupd.h utils.h log.h file_ops.h daemon_process.h daemon_list.h config.h commands.h

# ======================
# 가상 타겟 정의
# ======================
.PHONY: all clean debug release
```

```
# 기본 타겟 (make 명령어만 입력했을 때 실행)
all: $(TARGET)

# ======================
# 메인 타겟 빌드 규칙
# ======================

# 실행 파일 빌드 규칙
$(TARGET): $(OBJS)
	$(CC) $(CFLAGS) $(LDFLAGS) -o $@ $^

# ======================
# 오브젝트 파일 빌드 패턴 규칙
# ======================

# .c 파일을 .o 파일로 컴파일하는 규칙
%.o: %.c $(HEADERS)
	$(CC) $(CFLAGS) -c $< -o $@

# ======================
# 디버그 빌드 설정
# ======================

# 디버그 빌드: DEBUG 매크로 정의 및 디버그 정보 추가
debug: CFLAGS += -DDEBUG -g
debug: clean all

# ======================
# 릴리즈 빌드 설정
# ======================

# 릴리즈 빌드: 최적화 활성화 및 디버그 매크로 제거
release: CFLAGS += -O2 -DNDEBUG
release: clean all

# ======================
# 정리 규칙
# ======================

# 생성된 모든 파일 삭제
clean:
	rm -f $(TARGET) $(OBJS)
```

```
# ======================
# 의존성 생성 설정 (선택적)
# ======================

# 의존성 파일을 저장할 디렉토리
DEPDIR := .d

# 의존성 디렉토리 생성
$(shell mkdir -p $(DEPDIR) >/dev/null)

# 의존성 생성 플래그
DEPFLAGS = -MT $@ -MMD -MP -MF $(DEPDIR)/$*.Td

# 의존성 고려한 컴파일 명령 재정의
COMPILE.c = $(CC) $(DEPFLAGS) $(CFLAGS) -c

# 의존성 생성 규칙
%.o: %.c $(DEPDIR)/%.d
	$(COMPILE.c) $<
	@mv -f $(DEPDIR)/$*.Td $(DEPDIR)/$*.d

# 빈 의존성 파일 생성 규칙
$(DEPDIR)/%.d: ;

# 생성된 의존성 파일 보존 설정
.PRECIOUS: $(DEPDIR)/%.d

# 모든 소스 파일의 의존성 파일 포함
-include $(patsubst %,$(DEPDIR)/%.d,$(basename $(SRCS)))
```

//

```
<ssu_cleanupd.h>
#ifndef SSU_CLEANUPD_H
#define SSU_CLEANUPD_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <dirent.h>
#include <sys/stat.h>
#include <sys/types.h>
```

```c
#include <sys/file.h>
#include <time.h>
#include <pwd.h>
#include <fcntl.h>
#include <signal.h>
#include <errno.h>
#include <ctype.h>
#include <syslog.h>
#include <sys/wait.h>

// 상수 정의
#define MAX_PATH_LEN 4096          // 최대 경로 길이
#define MAX_BUF 1024               // 일반 버퍼 크기
#define MAX_FILENAME_LEN 255       // 최대 파일명 길이
#define DEFAULT_INTERVAL 10        // 기본 모니터링 간격 (초)
#define DEFAULT_MODE 1             // 기본 모드 (1: 최신 파일 유지)
#define CONFIG_FILENAME "ssu_cleanupd.config"   // 설정 파일 이름
#define LOG_FILENAME "ssu_cleanupd.log"         // 로그 파일 이름
#define DAEMON_LIST_FILE ".ssu_cleanupd/current_daemon_list"   // 데몬 리스트 파일 경로
#define STUDENT_ID "20211527"      // 학번

/**
 * @brief 데몬 설정 구조체
 *
 * 데몬 프로세스의 모든 설정 정보를 저장하는 구조체입니다.
 */
typedef struct {
    char monitoring_path[MAX_PATH_LEN];   // 모니터링 대상 디렉토리 경로
    char output_path[MAX_PATH_LEN];       // 정리된 파일이 저장될 디렉토리 경로
    char exclude_paths[MAX_PATH_LEN];     // 제외할 경로 목록 (콤마로 구분)
    char extensions[MAX_PATH_LEN];        // 처리할 파일 확장자 목록 (콤마로 구분)
    int time_interval;                    // 모니터링 간격 (초)
    char max_log_lines[MAX_FILENAME_LEN]; // 최대 로그 라인 수
    int mode;                             // 처리 모드 (1: 최신 파일 유지, 2: 오래된 파일 유지, 3: 중복 무시)
    pid_t pid;                            // 데몬 프로세스의 PID
    char start_time[50];                  // 데몬 시작 시간
} DaemonConfig;

/**
 * @brief 파일 노드 구조체
 *
 * 링크드 리스트로 파일 정보를 관리하기 위한 노드 구조체입니다.
 */
```

```
typedef struct FileNode {
    int excluded;                       // 제외 여부 플래그
    char path[MAX_PATH_LEN];            // 파일 전체 경로
    char name[MAX_FILENAME_LEN];        // 파일 이름
    char extension[32];                 // 파일 확장자
    time_t mod_time;                    // 파일 수정 시간
    struct FileNode *next;              // 다음 노드를 가리키는 포인터
} FileNode;

// Utility functions
char *get_home_directory();                     // 홈 디렉토리 경로 반환
char *get_absolute_path(const char *path); // 절대 경로로 변환
int is_inside_home_directory(const char *path); // 경로가 홈 디렉토리 내에 있는지 확인
int is_directory(const char *path);         // 경로가 디렉토리인지 확인
int file_exists(const char *path);          // 파일이 존재하는지 확인
char *get_file_extension(const char *filename); // 파일 확장자 추출
time_t get_file_mtime(const char *path); // 파일 수정 시간 가져오기
char *get_current_time();                   // 현재 시간 문자열 반환
void trim_newline(char *str);               // 문자열 끝의 개행 문자 제거
int create_directory(const char *path);   // 디렉토리 생성
int is_subdirectory(const char *parent, const char *child); // 하위 디렉토리인지 확인

// Config file operations
int read_config(const char *dir_path, DaemonConfig *config, int fd); // 설정 파일 읽기
int write_config(const char *dir_path, DaemonConfig *config, int fd); // 설정 파일 쓰기
int lock_config(const char *dir_path);      // 설정 파일 잠금
int unlock_config(int fd);                  // 설정 파일 잠금 해제

// Daemon list operations
int add_to_daemon_list(const char *dir_path, pid_t pid); // 데몬 리스트에 추가
int remove_from_daemon_list(const char *dir_path); // 데몬 리스트에서 제거
int is_path_in_daemon_list(const char *dir_path); // 경로가 데몬 리스트에 있는지 확인
int show_daemon_list();                     // 데몬 리스트 출력

// File operations
int copy_file(const char *src, const char *dest); // 파일 복사
int is_newer(const char *file1, const char *file2); // 파일1이 더 최신인지 확인
int is_older(const char *file1, const char *file2); // 파일1이 더 오래된지 확인
int should_clean_file(const char *filename, const char *extensions,
                      const char *exclude_paths, const char *base_path); // 파일 정리 대상인지 확인

// Log operations
int write_log_entry(const char *dir_path, const char *src, const char *dest, pid_t pid); // 로그 항목 추가
```

```
int trim_log_file(const char *dir_path, int max_lines); // 로그 파일 크기 조정

// arrange operations
FileNode* create_file_node(const char *path, const char *name, const char *ext, time_t mtime); // 파일 노드 생성
void add_file_node(const char *path, const char *ext, time_t mtime, DaemonConfig *config); // 파일 노드 추가
void free_file_list();                          // 파일 리스트 메모리 해제
void handle_duplicates(FileNode **head, DaemonConfig *config); // 중복 파일 처리
void scan_directory(const char *dir_path, DaemonConfig *config); // 디렉토리 스캔
void organize_files(const char *dir_path, DaemonConfig *config); // 파일 정리 수행

// Command handlers
void handle_show();                             // show 명령 처리
void handle_add(char *args);                    // add 명령 처리
void handle_modify(char *args);                 // modify 명령 처리
void handle_remove(char *args);                 // remove 명령 처리
void handle_help();                             // help 명령 처리
void handle_exit();                             // exit 명령 처리

// Daemon process
void run_as_daemon(DaemonConfig config); // 데몬 프로세스로 실행

#endif
```

//

<ssu_cleanupd.c>

```
#include "ssu_cleanupd.h"  // ssu_cleanupd 헤더 파일 포함

/**
 * @brief 메인 함수 - 데몬 관리 프로그램의 진입점
 *
 * 이 함수는 다음과 같은 작업을 수행합니다:
 * 1. 프로그램 작업 디렉토리 초기화 (~/.ssu_cleanupd 생성)
 * 2. 데몬 리스트 파일 초기화 (~/.ssu_cleanupd/current_daemon_list 생성)
 * 3. 사용자 명령어 입력 루프 실행
 * 4. 명령어에 따른 핸들러 함수 호출
 *
 * @return int 프로그램 종료 상태 (0: 정상 종료)
 */
int main() {
    // 프로그램 디렉토리 및 파일 초기화
    char home[MAX_PATH_LEN];  // 홈 디렉토리 경로를 저장할 배열
```

```
// 홈 디렉토리 경로에 "/.ssu_cleanupd"를 추가하여 생성
snprintf(home, sizeof(home), "%s/.ssu_cleanupd", get_home_directory());
mkdir(home, 0755);  // 0755 권한으로 디렉토리 생성 (소유자: 읽기/쓰기/실행, 그룹/기타: 읽기/실행)

// 데몬 리스트 파일 경로 설정
char list_path[MAX_PATH_LEN];
// 홈 디렉토리 경로에 데몬 리스트 파일명을 추가하여 전체 경로 생성
snprintf(list_path, sizeof(list_path), "%s/%s", get_home_directory(), DAEMON_LIST_FILE);

// 파일을 추가 모드(a+)로 열기 (읽기/쓰기, 파일이 없으면 생성)
FILE *fp = fopen(list_path, "a+");
if (fp) fclose(fp);  // 파일이 성공적으로 열렸으면 바로 닫기

// 명령어 처리 루프
char input[BUFSIZ];  // 사용자 입력을 저장할 버퍼
char *command, *args;  // 명령어와 인수를 저장할 포인터

while (1) {  // 무한 루프
    printf("%s> ", STUDENT_ID);  // 프롬프트 출력 (학번 표시)
    fflush(stdout);  // 출력 버퍼 강제 비우기

    // 사용자 입력 읽기 (EOF 처리: Ctrl+D)
    if (fgets(input, sizeof(input), stdin) == NULL) {
        break;  // EOF 발생 시 루프 종료
    }

    // 빈 입력 처리 (엔터만 입력된 경우)
    if (strcmp(input, "₩n") == 0) {
        continue;  // 다음 루프로 넘어감
    }

    // 입력 파싱: 공백이나 개행 문자를 기준으로 명령어와 인수 분리
    command = strtok(input, " ₩n");  // 첫 번째 토큰 (명령어)
    args = strtok(NULL, "₩n");          // 두 번째 토큰 (인수)

    if (command == NULL) continue;  // 명령어가 없으면 다음 루프

    // 명령어 분기 처리
    if (strcmp(command, "show") == 0) {
        handle_show();  // show 명령 처리 - 현재 실행 중인 데몬 목록 표시
    } else if (strcmp(command, "add") == 0) {
        handle_add(args);  // add 명령 처리 - 새 데몬 프로세스 추가
```

```
        } else if (strcmp(command, "modify") == 0) {
            handle_modify(args);  // modify 명령 처리 - 기존 데몬 설정 수정
        } else if (strcmp(command, "remove") == 0) {
            handle_remove(args);  // remove 명령 처리 - 데몬 프로세스 제거
        } else if (strcmp(command, "help") == 0) {
            handle_help();  // help 명령 처리 - 도움말 출력
        } else if (strcmp(command, "exit") == 0) {
            break;  // 루프 종료
        } else {
            handle_help();  // 알 수 없는 명령어일 경우 도움말 출력
        }
    }

    return 0;  // 프로그램 정상 종료
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

<utils.c>
#include "ssu_cleanupd.h"

/**
 * @brief 현재 사용자의 홈 디렉토리 경로를 반환
 *
 * @return char* 홈 디렉토리 경로 (정적 메모리 주소)
 */
char* get_home_directory() {
    struct passwd *pw = getpwuid(getuid());
    return pw ? pw->pw_dir : getenv("HOME");
}

/**
 * @brief 상대 경로를 절대 경로로 변환
 *
 * ~, ~/, ., .. 등의 상대 경로를 절대 경로로 변환합니다.
 *
 * @param path 변환할 경로
 * @return char* 절대 경로 (동적 할당 메모리, 사용 후 free 필요)
 */
char* get_absolute_path(const char *path) {
    char resolved_path[MAX_PATH_LEN];
    char *resolved;

    // ~ 처리 (홈 디렉토리)
```

```
    if (path[0] == '~') {
        const char *home = get_home_directory();
        if (path[1] == '/' || path[1] == '₩0') {
            snprintf(resolved_path, sizeof(resolved_path), "%s%s", home, path + 1);
        } else {
            fprintf(stderr, "Unsupported ~ expansion: %s₩n", path);
            return NULL;
        }
    }
    // 절대 경로인 경우
    else if (path[0] == '/') {
        strncpy(resolved_path, path, sizeof(resolved_path));
    }
    // 상대 경로인 경우
    else {
        char cwd[MAX_PATH_LEN];
        getcwd(cwd, sizeof(cwd));
        if(snprintf(resolved_path, sizeof(resolved_path), "%s/%s", cwd, path) >= (int)sizeof(resolved_path))
            fprintf(stderr, "Warning: path is too long₩n");
    }

    // 심볼릭 링크 해결
    resolved = realpath(resolved_path, NULL);
    if (!resolved) {
        resolved = strdup(resolved_path);
    }

    return resolved;
}

/**
 * @brief 경로가 홈 디렉토리 내에 있는지 확인
 *
 * @param path 확인할 경로
 * @return int 1: 홈 디렉토리 내부, 0: 외부
 */
int is_inside_home_directory(const char *path) {
    char *home = get_home_directory();
    char *abs_path = get_absolute_path(path);

    return strncmp(abs_path, home, strlen(home)) == 0;
}
```

```
/**
 * @brief 경로가 디렉토리인지 확인
 *
 * @param path 확인할 경로
 * @return int 1: 디렉토리, 0: 디렉토리 아님
 */
int is_directory(const char *path) {
    struct stat statbuf;
    if (stat(path, &statbuf) != 0) {
        return 0;
    }
    return S_ISDIR(statbuf.st_mode);
}

/**
 * @brief 파일이 존재하는지 확인
 *
 * @param path 확인할 경로
 * @return int 1: 존재, 0: 없음
 */
int file_exists(const char *path) {
    return access(path, F_OK) == 0;
}

/**
 * @brief 파일 확장자 추출
 *
 * @param filename 파일 이름
 * @return char* 확장자 문자열 (정적 메모리 주소)
 */
char* get_file_extension(const char *filename) {
    static char ext[MAX_FILENAME_LEN];
    const char *dot = strrchr(filename, '.');

    if (!dot || dot == filename) {
        strcpy(ext, "noext");
    } else {
        strcpy(ext, dot + 1);
    }

    return ext;
}
```

```
/**
 * @brief 파일 수정 시간 가져오기
 *
 * @param path 파일 경로
 * @return time_t 수정 시간 (초 단위), 실패 시 0 반환
 */
time_t get_file_mtime(const char *path) {
    struct stat statbuf;
    if (stat(path, &statbuf) != 0) {
        return 0;
    }
    return statbuf.st_mtime;
}

/**
 * @brief 현재 시간을 문자열로 반환
 *
 * @return char* "YYYY-MM-DD HH:MM:SS" 형식의 시간 문자열 (정적 메모리 주소)
 */
char* get_current_time() {
    static char time_str[20];
    time_t now = time(NULL);
    struct tm *tm = localtime(&now);
    strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S", tm);
    return time_str;
}

/**
 * @brief 문자열 끝의 개행 문자 제거
 *
 * @param str 처리할 문자열
 */
void trim_newline(char *str) {
    int len = strlen(str);
    if (len > 0 && str[len-1] == '₩n') {
        str[len-1] = '₩0';
    }
}

/**
 * @brief 디렉토리 생성
 *
 * @param path 생성할 디렉토리 경로
```

```
 * @return int 1: 성공, 0: 실패
 */
int create_directory(const char *path) {
    if (mkdir(path, 0755) == 0 || errno == EEXIST) {
        return 1;
    }
    return 0;
}

/**
 * @brief child 경로가 parent 경로의 하위 디렉토리인지 확인
 *
 * @param parent 부모 디렉토리 경로
 * @param child 확인할 디렉토리 경로
 * @return int 1: 하위 디렉토리, 0: 하위 디렉토리 아님
 */
int is_subdirectory(const char *parent, const char *child) {
    char real_parent[MAX_PATH_LEN];
    char real_child[MAX_PATH_LEN];

    if (realpath(parent, real_parent) == NULL) return 0;
    if (realpath(child, real_child) == NULL) return 0;
    size_t parent_len = strlen(real_parent);
    size_t child_len = strlen(real_child);

    if (child_len <= parent_len) return 0;

    if (strncmp(real_parent, real_child, parent_len) != 0) return 0;

    return (real_child[parent_len] == '/' || real_child[parent_len] == '₩0');
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
<daemon_list.c>
#include "ssu_cleanupd.h"

/**
 * @brief 데몬 리스트에 새 항목 추가
 *
 * @param dir_path 모니터링 디렉토리 경로
 * @param pid 데몬 프로세스 PID
 * @return int 0: 성공, -1: 실패
 */
int add_to_daemon_list(const char *dir_path, pid_t pid) {
```

```
    char list_path[MAX_PATH_LEN];
    snprintf(list_path, sizeof(list_path), "%s/%s", get_home_directory(), DAEMON_LIST_FILE);

    int fd = open(list_path, O_WRONLY|O_CREAT|O_APPEND, 0644);
    if (fd == -1) return -1;

    // CSV 형식으로 안전하게 기록 ("경로",PID)
    dprintf(fd, "₩"%s₩",%d₩n", dir_path, pid);

    close(fd);
    return 0;
}

/**
 * @brief 데몬 리스트에서 항목 제거
 *
 * @param dir_path 제거할 모니터링 디렉토리 경로
 * @return int 1: 성공, 0: 실패 (항목 없음)
 */
int remove_from_daemon_list(const char *dir_path) {
    char list_path[MAX_PATH_LEN];
    snprintf(list_path, sizeof(list_path), "%s/%s", get_home_directory(), DAEMON_LIST_FILE);

    // 임시 파일 생성
    char temp_path[MAX_PATH_LEN + 10];
    snprintf(temp_path, sizeof(temp_path), "%s.tmp.%d", list_path, getpid());

    // 원본 파일 열기
    int fd = open(list_path, O_RDONLY);
    if (fd == -1) {
        perror("open daemon list");
        return 0;
    }

    FILE *temp_fp = fopen(temp_path, "w");
    if (!temp_fp) {
        perror("open temp file");
        close(fd);
        return 0;
    }

    // CSV 파싱 및 처리
    char line[MAX_PATH_LEN + 50];
```

```
    int found = 0;
    FILE *list_fp = fdopen(fd, "r");

    while (fgets(line, sizeof(line), list_fp)) {
        // CSV 형식: "path",pid
        char *path_start = strchr(line, '"');
        char *path_end = path_start ? strchr(path_start + 1, '"') : NULL;
        char *comma = path_end ? strchr(path_end + 1, ',') : NULL;

        if (!path_start || !path_end || !comma) {
            fputs(line, temp_fp);  // 형식 오류 시 원본 유지
            continue;
        }

        // 경로 추출 및 비교
        *path_end = '₩0';
        if (strcmp(path_start + 1, dir_path) == 0) {
            found = 1;
        } else {
            *path_end = '"';  // 원복
            fputs(line, temp_fp);  // 일치하지 않으면 임시 파일에 기록
        }
    }

    fclose(temp_fp);
    fclose(list_fp);  // fd도 함께 닫힘

    // 결과 처리
    if (found) {
        if (rename(temp_path, list_path) == -1) {
            perror("rename failed");
            remove(temp_path);
        }
    } else {
        remove(temp_path);
    }

    return found;
}

/**
 * @brief 경로가 데몬 리스트에 있는지 확인
 *
```

```
 * @param dir_path 확인할 디렉토리 경로
 * @return int 1: 존재, 0: 없음
 */
int is_path_in_daemon_list(const char *dir_path) {
    char list_path[MAX_PATH_LEN];
    snprintf(list_path, sizeof(list_path), "%s/%s", get_home_directory(), DAEMON_LIST_FILE);

    int fd = open(list_path, O_RDONLY);
    if (fd == -1) return 0;

    FILE *fp = fdopen(fd, "r");
    if (!fp) {
        close(fd);
        return 0;
    }

    char line[MAX_PATH_LEN + 20];
    int found = 0;

    while (fgets(line, sizeof(line), fp)) {
        char *quote_start = strchr(line, '"');
        if (!quote_start) continue;

        char *quote_end = strchr(quote_start + 1, '"');
        if (!quote_end) continue;

        *quote_end = '₩0';
        if (strcmp(quote_start + 1, dir_path) == 0) {
            found = 1;
            break;
        }
    }

    fclose(fp);
    return found;
}

/**
 * @brief 데몬 리스트 출력
 *
 * @return int 리스트에 있는 데몬 수
 */
int show_daemon_list() {
```

```
char list_path[MAX_PATH_LEN];
snprintf(list_path, sizeof(list_path), "%s/%s", get_home_directory(), DAEMON_LIST_FILE);

// 파일 잠금 획득 (공유 잠금)
int fd = open(list_path, O_RDONLY);
if (fd == -1) {
    printf("No daemon processes running₩n");
    return 0;
}

FILE *fp = fdopen(fd, "r");
if (!fp) {
    close(fd);
    return 0;
}

printf("₩n0. exit₩n");

char line[MAX_PATH_LEN + 20];
int count = 1;

while (fgets(line, sizeof(line), fp)) {
    // CSV 파싱: "path",pid
    char *quote_start = strchr(line, '"');
    if (!quote_start) continue;

    char *quote_end = strchr(quote_start + 1, '"');
    if (!quote_end) continue;

    // 경로 추출 (쌍따옴표 제거)
    *quote_end = '₩0';
    char *path = quote_start + 1;

    // PID 추출 (쉼표 이후)
    char *pid_str = strchr(quote_end + 1, ',');
    if (!pid_str) continue;
    pid_str++;  // 쉼표 건너뛰기

    printf("%d. %s₩n", count++, path);
}

fclose(fp);  // fd도 자동으로 닫힘
return count - 1;
```

```
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
<daemon_process.c>
#include "ssu_cleanupd.h"

/**
 * @brief 데몬 프로세스로 실행
 *
 * 이 함수는 다음과 같은 작업을 수행합니다:
 * 1. 데몬 프로세스 생성 (fork, setsid)
 * 2. 파일 디스크립터 정리
 * 3. 작업 디렉토리 변경 및 umask 설정
 * 4. 설정 파일 및 로그 파일 초기화
 * 5. 데몬 리스트에 등록
 * 6. 주기적으로 파일 정리 수행
 *
 * @param config 데몬 설정
 */
void run_as_daemon(DaemonConfig config)
{
    pid_t pid;

    int fd, maxfd;

    // 1차 fork
    if ((pid = fork()) < 0)
    {
        fprintf(stderr, "fork error₩n");
        exit(1);
    }
    else if (pid > 0)  // 부모 프로세스 종료
        return;

    // 새로운 세션 생성
    setsid();

    // 2차 fork (데몬이 세션 리더가 되는 것 방지)
    if((pid = fork()) < 0)
        exit(1);
    else if(pid > 0)
        exit(0);
```

```
    pid = getpid();
    setsid();

    // 불필요한 시그널 무시
    signal(SIGTTIN, SIG_IGN);
    signal(SIGTTOU, SIG_IGN);
    signal(SIGTSTP, SIG_IGN);

    // 모든 열린 파일 디스크립터 닫기
    maxfd = getdtablesize();
    for (fd = 0; fd < maxfd; fd++)
        close(fd);

    umask(0);   // 파일 생성 권한 마스크 설정
    chdir("/");   // 루트 디렉토리로 이동

    // 표준 입출력 리다이렉션
    fd = open("/dev/null", O_RDWR);
    dup(0);
    dup(0);

    // 설정 파일 경로 생성
    char config_path[MAX_PATH_LEN];
    if (snprintf(config_path, sizeof(config_path), "%s/%s", config.monitoring_path, CONFIG_FILENAME) >= (int)sizeof(config_path))
        fprintf(stderr, "Warning: path is too long₩n");

    // 설정 파일 잠금 및 초기화
    int config_fd = open(config_path, O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (lock_config(config.monitoring_path) == -1)
        exit(EXIT_FAILURE);

    // 로그 파일 경로 생성
    char log_path[MAX_PATH_LEN];
    if (snprintf(log_path, sizeof(log_path), "%s/%s", config.monitoring_path, LOG_FILENAME) >= (int)sizeof(log_path))
        fprintf(stderr, "Warning: path is too long₩n");

    // 로그 파일 생성
    int log_fd = open(log_path, O_WRONLY | O_CREAT | O_EXCL, 0644);
    if (log_fd >= 0)
```

```
        close(log_fd);

    // 데몬 정보 설정
    config.pid = getpid();
    strncpy(config.start_time, get_current_time(), sizeof(config.start_time));

    // 설정 파일에 데몬 정보 기록
    write_config(config.monitoring_path, &config, config_fd);
    unlock_config(config_fd);

    // 데몬 리스트에 등록
    if (add_to_daemon_list(config.monitoring_path, config.pid) != 0)
        exit(EXIT_FAILURE);

    // 주기적으로 파일 정리 수행
    while (1)
    {
        organize_files(config.monitoring_path, &config);
        sleep(config.time_interval);
    }
}
```

//

<file_ops.c>

```
#include "ssu_cleanupd.h"

/**
 * @brief 파일 복사
 *
 * @param src 원본 파일 경로
 * @param dest 대상 파일 경로
 * @return int 1: 성공, 0: 실패
 */
int copy_file(const char *src, const char *dest) {
    FILE *src_fp = fopen(src, "rb");
    if (!src_fp) {
        return 0;
    }

    FILE *dest_fp = fopen(dest, "wb");
    if (!dest_fp) {
        fclose(src_fp);
```

```
        return 0;
    }

    char buffer[4096];
    size_t bytes;

    // 버퍼를 사용하여 파일 내용 복사
    while ((bytes = fread(buffer, 1, sizeof(buffer), src_fp)) > 0) {
        fwrite(buffer, 1, bytes, dest_fp);
    }

    fclose(src_fp);
    fclose(dest_fp);

    // 원본 파일 권한 유지
    struct stat statbuf;
    if (stat(src, &statbuf) == 0) {
        chmod(dest, statbuf.st_mode);
    }

    return 1;
}

/**
 * @brief file1이 file2보다 최신인지 확인
 *
 * @param file1 비교할 파일1 경로
 * @param file2 비교할 파일2 경로
 * @return int 1: file1이 최신, 0: 그렇지 않음
 */
int is_newer(const char *file1, const char *file2) {
    time_t mtime1 = get_file_mtime(file1);
    time_t mtime2 = get_file_mtime(file2);

    return mtime1 > mtime2;
}

/**
 * @brief file1이 file2보다 오래된지 확인
 *
 * @param file1 비교할 파일1 경로
```

```
 * @param file2 비교할 파일2 경로
 * @return int 1: file1이 오래됨, 0: 그렇지 않음
 */
int is_older(const char *file1, const char *file2) {
    time_t mtime1 = get_file_mtime(file1);
    time_t mtime2 = get_file_mtime(file2);

    return mtime1 < mtime2;
}

/**
 * @brief 파일이 정리 대상인지 확인
 *
 * @param filename 파일 이름
 * @param extensions 허용된 확장자 목록 (콤마로 구분)
 * @param exclude_paths 제외할 경로 목록 (콤마로 구분)
 * @param base_path 기준 경로
 * @return int 1: 정리 대상, 0: 정리 제외
 */
int should_clean_file(const char *filename, const char *extensions,
                      const char *exclude_paths, const char *base_path) {
    // 설정 파일이나 로그 파일은 제외
    if (strcmp(filename, CONFIG_FILENAME) == 0 ||
        strcmp(filename, LOG_FILENAME) == 0) {
        return 0;
    }

    // 확장자 체크 ("all"이 아니면 필터링)
    if (strcmp(extensions, "all") != 0) {
        char *ext = get_file_extension(filename);
        char ext_list[MAX_PATH_LEN];
        strncpy(ext_list, extensions, sizeof(ext_list));

        int found = 0;
        char *token = strtok(ext_list, ",");
        while (token) {
            // 공백 제거
            while (*token == ' ') token++;

            if (strcmp(token, ext) == 0) {
                found = 1;
```

```
                break;
            }
            token = strtok(NULL, ",");
        }

        if (!found) {
            return 0;
        }
    }

    // 제외 경로 체크 ("none"이 아니면 필터링)
    if (strcmp(exclude_paths, "none") != 0) {
        char full_path[MAX_PATH_LEN];
        snprintf(full_path, sizeof(full_path), "%s/%s", base_path, filename);

        char excl_list[MAX_PATH_LEN];
        strncpy(excl_list, exclude_paths, sizeof(excl_list));

        char *token = strtok(excl_list, ",");
        while (token) {
            // 공백 제거
            while (*token == ' ') token++;

            if (strncmp(full_path, token, strlen(token)) == 0) {
                return 0;
            }
            token = strtok(NULL, ",");
        }
    }

    return 1;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

<log.c>

```
#include "ssu_cleanupd.h"

/**
 * 로그 항목을 파일에 추가하는 함수
 *
 * @param dir_path 로그 파일이 위치한 디렉토리 경로
 * @param src 처리된 원본 파일 경로
```

```
 * @param dest 이동된 대상 파일 경로
 * @param pid 데몬 프로세스 ID
 * @return 성공 시 1, 실패 시 0
 */
int write_log_entry(const char *dir_path, const char *src, const char *dest, pid_t pid) {
    char log_path[MAX_PATH_LEN];
    // 로그 파일 경로 생성: dir_path/LOG_FILENAME
    snprintf(log_path, sizeof(log_path), "%s/%s", dir_path, LOG_FILENAME);

    // 로그 파일 열기 (추가 모드)
    FILE *fp = fopen(log_path, "a");
    if (!fp) {
        return 0;
    }

    // 현재 시간 정보 얻기
    time_t now = time(NULL);
    struct tm *tm = localtime(&now);
    char time_str[9];  // HH:MM:SS 형식 (8문자 + null)
    strftime(time_str, sizeof(time_str), "%H:%M:%S", tm);  // 시간만 표시

    // 로그 형식: [시간][PID][원본경로][대상경로]
    fprintf(fp, "[%s][%d][%s][%s]₩n", time_str, pid, src, dest);
    fclose(fp);
    return 1;
}

/**
 * 로그 파일을 최대 줄 수에 맞게 트리밍하는 함수
 *
 * @param dir_path 로그 파일이 위치한 디렉토리 경로
 * @param max_lines 유지할 최대 줄 수
 * @return 성공 시 1, 실패 시 0
 */
int trim_log_file(const char *dir_path, int max_lines) {
    if (max_lines <= 0) {
        return 1;  // max_lines가 0 이하면 트리밍하지 않음
    }

    char log_path[MAX_PATH_LEN];
    snprintf(log_path, sizeof(log_path), "%s/%s", dir_path, LOG_FILENAME);

    // 임시 파일 경로 생성 (프로세스 ID 추가하여 고유성 보장)
```

```
char temp_path[MAX_PATH_LEN + 10];
snprintf(temp_path, sizeof(temp_path), "%s.tmp%d", log_path, getpid());

// 로그 파일 열기
FILE *src = fopen(log_path, "r");
if (!src) return 0;

// 파일 내용을 메모리에 저장 (동적 할당)
char **lines = NULL;
int line_count = 0;
char buffer[1024];

// 파일 내용 한 줄씩 읽기
while (fgets(buffer, sizeof(buffer), src)) {
    lines = realloc(lines, (line_count + 1) * sizeof(char *));
    lines[line_count] = strdup(buffer);
    line_count++;
}
fclose(src);

// 현재 줄 수가 max_lines 이하인 경우 트리밍 필요 없음
if (line_count <= max_lines) {
    for (int i = 0; i < line_count; i++) free(lines[i]);
    free(lines);
    return 1;
}

// 최신 max_lines 줄만 임시 파일에 저장
FILE *dst = fopen(temp_path, "w");
if (!dst) {
    for (int i = 0; i < line_count; i++) free(lines[i]);
    free(lines);
    return 0;
}

// 오래된 줄은 버리고 최신 줄만 저장
int start = line_count - max_lines;
for (int i = start; i < line_count; i++) {
    fputs(lines[i], dst);
    free(lines[i]);
}
free(lines);
fclose(dst);
```

```
    // 임시 파일을 원본 로그 파일로 교체
    rename(temp_path, log_path);
    return 1;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
<config.c>
#include "ssu_cleanupd.h"

/**
 * 설정 파일을 읽어서 DaemonConfig 구조체에 저장하는 함수
 *
 * @param dir_path 설정 파일이 위치한 디렉토리 경로
 * @param config 설정값을 저장할 구조체 포인터
 * @param fd 파일 디스크립터 (잠금된 파일인 경우 사용)
 * @return 성공 시 1, 실패 시 -1
 */
int read_config(const char *dir_path, DaemonConfig *config, int fd) {
    char config_path[MAX_PATH_LEN];
    // 설정 파일 경로 생성: dir_path/CONFIG_FILENAME
    snprintf(config_path, sizeof(config_path), "%s/%s", dir_path, CONFIG_FILENAME);

    FILE *fp = NULL;
    int fd_dup = -1;

    if (fd != -1) {
        // 파일 잠금이 된 경우: 기존 fd를 복제하여 사용 (원본 fd 유지)
        if ((fd_dup = dup(fd)) == -1) {
            perror("dup failed");
            return -1;
        }
        if ((fp = fdopen(fd_dup, "r")) == NULL) {
            perror("fdopen failed");
            close(fd_dup);
            return -1;
        }
        rewind(fp); // 파일 포지션을 처음으로 리셋
    } else {
        // 일반적인 경우: 파일 직접 열기
        if ((fp = fopen(config_path, "r")) == NULL) {
            perror("fopen failed");
            return -1;
        }
```

```
    }

    char line[255];
    // 설정 파일 한 줄씩 읽기
    while (fgets(line, sizeof(line), fp)) {
        trim_newline(line); // 줄바꿈 문자 제거

        // 키:값 분리 (콜론 기준)
        char *colon_pos = strchr(line, ':');
        if (!colon_pos) continue; // 콜론 없는 줄은 무시

        *colon_pos = '₩0'; // 콜론 위치를 NULL로 변경하여 문자열 분리
        char *key = line;
        char *value = colon_pos + 1;

        // 키와 값 앞뒤 공백 제거
        while (*key == ' ') key++;
        while (*value == ' ') value++;

        // 키에 따라 해당 필드에 값 저장
        if (strcmp(key, "monitoring_path ") == 0) {
            strncpy(config->monitoring_path, value, sizeof(config->monitoring_path));
        }
        else if (strcmp(key, "output_path ") == 0) {
            strncpy(config->output_path, value, sizeof(config->output_path));
        }
        // ... (생략 - 다른 필드들도 동일한 방식으로 처리)
    }

    fclose(fp);
    return 1;
}

/**
 * 설정 파일에 DaemonConfig 구조체 내용을 저장하는 함수
 *
 * @param dir_path 설정 파일을 저장할 디렉토리 경로
 * @param config 저장할 설정 구조체 포인터
 * @param fd 파일 디스크립터 (잠금된 파일인 경우 사용)
 * @return 성공 시 1, 실패 시 -1
 */
int write_config(const char *dir_path, DaemonConfig *config, int fd) {
    char config_path[MAX_PATH_LEN];
```

```
    snprintf(config_path, sizeof(config_path), "%s/%s", dir_path, CONFIG_FILENAME);

    // fd가 있는 경우 복제하여 사용, 없는 경우 새 파일 생성
    FILE *fp = fd == -1 ?
        fopen(dir_path, "w") :
        fdopen(dup(fd), "w");

    if (!fp) {
        perror("Failed to write config");
        return -1;
    }

    // 설정값들을 파일에 기록
    fprintf(fp, "monitoring_path : %s₩n", config->monitoring_path);
    fprintf(fp, "pid : %d₩n", config->pid);
    // ... (생략 - 다른 필드들도 동일한 방식으로 기록)

    fclose(fp);
    return 1;
}

/**
 * 설정 파일에 대한 잠금을 설정하는 함수
 *
 * @param dir_path 설정 파일이 위치한 디렉토리 경로
 * @return 성공 시 파일 디스크립터, 실패 시 -1
 */
int lock_config(const char *dir_path) {
    char config_path[MAX_PATH_LEN];
    snprintf(config_path, sizeof(config_path), "%s/%s", dir_path, CONFIG_FILENAME);

    // 설정 파일 열기 (읽기/쓰기 모드)
    int fd = open(config_path, O_RDWR);
    if (fd == -1) {
        return -1;
    }

    // 파일 잠금 구조체 설정
    struct flock fl;
    memset(&fl, 0, sizeof(fl));
    fl.l_type = F_WRLCK;  // 쓰기 잠금 설정
    fl.l_whence = SEEK_SET;
    fl.l_start = 0;
```

```
    fl.l_len = 0;  // 전체 파일 잠금

    // 파일 잠금 시도 (블로킹 방식)
    if (fcntl(fd, F_SETLKW, &fl) == -1) {
        close(fd);
        return -1;
    }

    return fd; // 잠금된 파일 디스크립터 반환
}

/**
 * 설정 파일 잠금 해제 함수
 *
 * @param fd 잠금 해제할 파일 디스크립터
 * @return 성공 시 1, 실패 시 0
 */
int unlock_config(int fd) {
    if (fd == -1) {
        return 0;
    }

    // 파일 잠금 해제 구조체 설정
    struct flock fl;
    memset(&fl, 0, sizeof(fl));
    fl.l_type = F_UNLCK;  // 잠금 해제
    fl.l_whence = SEEK_SET;
    fl.l_start = 0;
    fl.l_len = 0;

    // 파일 잠금 해제 시도
    if (fcntl(fd, F_SETLK, &fl) == -1) {
        close(fd);
        return 0;
    }

    close(fd);
    return 1;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

<arrange.c>
#include "ssu_cleanupd.h"
```

```
// 전역 변수: 파일 정보를 저장하는 링크드 리스트의 헤드 포인터
FileNode *file_list_head = NULL;

/**
 * 파일 노드 생성 및 초기화 함수
 */
FileNode *create_file_node(const char *path, const char *name, const char *ext, time_t mtime)
{
    FileNode *new_node = (FileNode *)malloc(sizeof(FileNode));
    if (!new_node)
    {
        perror("Failed to allocate memory for file node"); // 파일 노드 메모리 할당 실패
        return NULL;
    }

    // 파일 정보 복사
    strncpy(new_node->path, path, MAX_PATH_LEN - 1);
    strncpy(new_node->name, name, MAX_FILENAME_LEN - 1);
    strncpy(new_node->extension, ext, 31);
    new_node->mod_time = mtime;
    new_node->next = NULL;

    return new_node;
}

/**
 * 링크드 리스트에 파일 노드 추가 함수
 */
void add_file_node(const char *path, const char *ext, time_t mtime, DaemonConfig *config)
{
    // 경로에서 파일명 추출
    const char *name = strrchr(path, '/');
    if (name)
        name++;
    else
        name = path;

    // _arranged 디렉토리 경로 생성
    char arranged_path[MAX_PATH_LEN];
    if(snprintf(arranged_path, sizeof(arranged_path), "%s/%s/%s", config->output_path, ext, name) >=
(int)sizeof(arranged_path))
        fprintf(stderr, "Warning: path is too long₩n"); // 경로 길이 초과 경고
```

```
    // _arranged 디렉토리 파일 상태 확인
    struct stat arranged_stat;
    if (stat(arranged_path, &arranged_stat) == 0)
    {
        // 모드별 처리
        switch (config->mode)
        {
        case 1: // 최신 파일 유지
            if (mtime <= arranged_stat.st_mtime) return;
            break;
        case 2: // 오래된 파일 보존
            if (mtime >= arranged_stat.st_mtime) return;
            break;
        case 3: // 중복 파일 모두 무시
            return;
        }
    }

    // 새 노드 생성 및 리스트에 추가
    FileNode *new_node = create_file_node(path, name, ext, mtime);
    if (!new_node) return;

    if (!file_list_head)
    {
        file_list_head = new_node;
    }
    else
    {
        FileNode *current = file_list_head;
        while (current->next) current = current->next;
        current->next = new_node;
    }
}

/**
 * 링크드 리스트 메모리 해제 함수
 */
void free_file_list()
{
    FileNode *current = file_list_head;
    while (current)
    {
        FileNode *next = current->next;
```

```
        free(current);
        current = next;
    }
    file_list_head = NULL;
}

/**
 * 중복 파일 처리 함수
 */
void handle_duplicates(FileNode **head, DaemonConfig *config)
{
    FileNode *current = *head;
    FileNode *prev = NULL;

    while (current != NULL)
    {
        FileNode *runner = *head;
        FileNode *runner_prev = NULL;
        FileNode *best_node = current; // 현재 모드에서 가장 적합한 파일
        int duplicate_found = 0;

        // 중복 파일 검색
        while (runner != NULL)
        {
            if (runner != current && strcmp(runner->name, current->name) == 0)
            {
                duplicate_found = 1;

                // 모드별 최적 파일 선택
                switch (config->mode)
                {
                case 1: // 최신 파일 선택
                    if (runner->mod_time > best_node->mod_time) best_node = runner;
                    break;
                case 2: // 오래된 파일 선택
                    if (runner->mod_time < best_node->mod_time) best_node = runner;
                    break;
                case 3: // 중복 파일 모두 제거
                    if (runner_prev)
                        runner_prev->next = runner->next;
                    else
                        *head = runner->next;
                    FileNode *to_delete = runner;
```

```
                        runner = runner->next;
                        free(to_delete);
                        continue;
                    }
                }
                runner_prev = runner;
                runner = runner->next;
            }

            // _arranged 디렉토리 경로 생성
            char arranged_path[MAX_PATH_LEN];
            if(snprintf(arranged_path, sizeof(arranged_path), "%s/%s/%s",config->output_path, current->extension,
current->name) >= (int)sizeof(arranged_path))
                fprintf(stderr, "Warning: path is too long₩n"); // 경로 길이 초과 경고

            // 모드 3에서 중복 파일 처리
            if(config->mode == 3 && duplicate_found){
                FileNode *to_delete = current;
                if (prev == NULL) *head = current->next;
                else prev->next = current->next;
                current = current->next;
                free(to_delete);
            }
            // 다른 모드에서 중복 파일 처리
            else if (duplicate_found && current != best_node && config->mode != 3)
            {
                FileNode *to_delete = current;
                if (prev == NULL) *head = current->next;
                else prev->next = current->next;
                current = current->next;
                free(to_delete);
            }

            prev = current;
            current = current->next;
        }
    }

/**
 * 디렉토리 스캔 함수
 */
void scan_directory(const char *dir_path, DaemonConfig *config)
{
    DIR *dir;
```

```
struct dirent *entry;
struct stat file_stat;

if ((dir = opendir(dir_path)) == NULL) return;

while ((entry = readdir(dir)) != NULL)
{
    // 숨김 파일 및 시스템 파일 필터링
    if (entry->d_name[0] == '.' ||
        strcmp(entry->d_name, "ssu_cleanupd.config") == 0 ||
        strcmp(entry->d_name, "ssu_cleanupd.log") == 0)
    {
        continue;
    }

    // 파일 경로 생성
    char file_path[MAX_PATH_LEN];
    snprintf(file_path, sizeof(file_path), "%s/%s", dir_path, entry->d_name);

    if (stat(file_path, &file_stat) != 0) continue;

    // 디렉토리인 경우 재귀적 처리
    if (S_ISDIR(file_stat.st_mode))
    {
        int is_excluded = 0;
        if (strcmp(config->exclude_paths, "none") != 0)
        {
            char exclude_copy[MAX_PATH_LEN];
            strcpy(exclude_copy, config->exclude_paths);
            char *token = strtok(exclude_copy, ",");

            // 제외 디렉토리 확인
            while (token)
            {
                if (strstr(file_path, token))
                {
                    is_excluded = 1;
                    break;
                }
                token = strtok(NULL, ",");
            }
        }
```

```
            if (!is_excluded) scan_directory(file_path, config);
            continue;
        }

        // 확장자 필터링
        const char *ext = get_file_extension(entry->d_name);
        if (strcmp(config->extensions, "all") != 0)
        {
            char ext_copy[MAX_PATH_LEN];
            strcpy(ext_copy, config->extensions);
            char *token = strtok(ext_copy, ",");
            int ext_match = 0;

            // 허용 확장자 확인
            while (token)
            {
                if (strcmp(ext, token) == 0)
                {
                    ext_match = 1;
                    break;
                }
                token = strtok(NULL, ",");
            }

            if (!ext_match) continue;
        }

        // 파일 노드 추가
        add_file_node(file_path, ext, file_stat.st_mtime, config);
    }
    closedir(dir);
}

/**
 * 파일 정리 메인 함수
 */
void organize_files(const char *dir_path, DaemonConfig *config)
{
    // 설정 파일 경로 생성
    char config_path[MAX_PATH_LEN];
    snprintf(config_path, sizeof(config_path), "%s/ssu_cleanupd.config", dir_path);

    // 설정 파일 읽기
```

```
int config_fd = open(config_path, O_RDONLY);
if (config_fd >= 0)
{
    if (lock_config(dir_path))
    {
        read_config(dir_path, config, config_fd);
        unlock_config(config_fd);
    }
}

// 파일 리스트 초기화 및 처리
free_file_list();
scan_directory(dir_path, config);
handle_duplicates(&file_list_head, config);

FileNode *current = file_list_head;
int changes_detected = 0;

// 파일 정리 수행
while (current)
{
    // 확장자별 디렉토리 생성
    char ext_dir[MAX_PATH_LEN];
    if(snprintf(ext_dir, sizeof(ext_dir), "%s/%s", config->output_path, current->extension) >= (int)sizeof(ext_dir))
        fprintf(stderr, "Warning: path is too long₩n"); // 경로 길이 초과 경고

    if (!create_directory(ext_dir)) {
        fprintf(stderr, "Failed to create directory: %s₩n", ext_dir); // 디렉토리 생성 실패
        continue;
    }

    // 대상 경로 생성
    char dest_path[MAX_PATH_LEN];
    if(snprintf(dest_path, sizeof(dest_path), "%s/%s", ext_dir, current->name) >= (int)sizeof(dest_path))
        fprintf(stderr, "Warning: path is too long₩n"); // 경로 길이 초과 경고

    // 파일 복사 및 로그 기록
    if (copy_file(current->path, dest_path))
    {
        changes_detected = 1;
        if (!write_log_entry(dir_path, current->path, dest_path, config->pid))
        {
            fprintf(stderr, "Failed to write log entry₩n"); // 로그 기록 실패
```

```
            }
        }

        current = current->next;
    }

    // 변경 사항 발생 시 로그 정리
    if (changes_detected)
    {
        if (!trim_log_file(dir_path, atoi(config->max_log_lines)))
        {
            fprintf(stderr, "Failed to trim log file₩n"); // 로그 파일 정리 실패
        }
    }

    // 메모리 해제
    free_file_list();
}
```

//

<commands.c>

```
#include "ssu_cleanupd.h"

/**
 * 현재 실행 중인 데몬 프로세스 목록을 보여주고 상세 정보를 확인하는 함수
 */
void handle_show()
{
    printf("Current working daemon process list₩n"); // 현재 실행 중인 데몬 프로세스 목록 출력
    int count = show_daemon_list(); // 데몬 목록 표시 및 개수 반환

    char input[20];
    while (1)
    {
        printf("₩nSelect one to see process info : "); // 프로세스 선택 프롬프트
        fgets(input, sizeof(input), stdin);
        trim_newline(input);

        // 입력 유효성 검사
        if (!isdigit(input[0]))
        {
            printf("Please check your input is valid₩n₩n"); // 잘못된 입력 알림
            show_daemon_list();
```

```
        continue;
    }

    int choice = atoi(input);
    if (choice == 0) break; // 0 입력 시 종료
    else if (choice < 0 || choice > count)
    {
        printf("Please check your input is valid₩n₩n"); // 범위 외 입력 알림
        show_daemon_list();
        continue;
    }

    // 선택한 데몬 정보 추출
    char list_path[MAX_PATH_LEN];
    snprintf(list_path, sizeof(list_path), "%s/%s", get_home_directory(), DAEMON_LIST_FILE);

    FILE *fp = fopen(list_path, "r");
    if (!fp)
    {
        perror("fopen"); // 파일 열기 실패
        return;
    }

    char line[MAX_PATH_LEN + 50];
    int current = 1;
    char selected_path[MAX_PATH_LEN] = {0};
    pid_t pid = 0;

    // 선택한 데몬의 경로와 PID 추출
    while (fgets(line, sizeof(line), fp))
    {
        if (current == choice)
        {
            char *path_start = strchr(line, '"');
            if (!path_start) continue;

            char *path_end = strchr(path_start + 1, '"');
            if (!path_end) continue;

            *path_end = '₩0';
            strncpy(selected_path, path_start + 1, MAX_PATH_LEN);

            char *pid_start = strchr(path_end + 1, ',');
```

```
            if (pid_start) pid = atoi(pid_start + 1);
            break;
        }
        current++;
    }
    fclose(fp);

    if (selected_path[0] == '₩0' || pid == 0)
    {
        printf("Invalid selection₩n"); // 유효하지 않은 선택
        continue;
    }

    // 1. 설정 파일 내용 출력
    char config_path[MAX_PATH_LEN];
    if(snprintf(config_path, sizeof(config_path), "%s/%s", selected_path, CONFIG_FILENAME) >=
(int)sizeof(config_path))
        fprintf(stderr, "Warning: path is too long₩n"); // 경로 길이 경고

    FILE *config_fp = fopen(config_path, "r");
    if (!config_fp) {
        printf("Failed to open config file: %s₩n", config_path); // 설정 파일 열기 실패
        continue;
    }

    printf("₩n1. config detail₩n"); // 설정 파일 상세 정보
    char config_line[256];
    while (fgets(config_line, sizeof(config_line), config_fp)) {
        printf("%s", config_line);
    }
    fclose(config_fp);

    // 2. 로그 파일 내용 출력 (최근 10줄)
    printf("₩n2. log detail₩n₩n");
    char log_path[MAX_PATH_LEN];
    if(snprintf(log_path, sizeof(log_path), "%s/%s", selected_path, LOG_FILENAME) >= (int)sizeof(log_path))
        fprintf(stderr, "Warning: path is too long₩n"); // 경로 길이 경고

    FILE *log_fp = fopen(log_path, "r");
    if (log_fp)
    {
        // 파일 끝에서부터 10줄 읽기
        char *lines[10] = {0};
        int line_count = 0;
```

```
fseek(log_fp, 0, SEEK_END);
long pos = ftell(log_fp);
long end_pos = pos;

// 역순으로 로그 읽기
while (pos > 0 && line_count < 10)
{
    pos--;
    fseek(log_fp, pos, SEEK_SET);

    if (fgetc(log_fp) == '₩n')
    {
        long line_pos = ftell(log_fp);
        size_t line_len = end_pos - line_pos;

        lines[line_count] = malloc(line_len + 1);
        fseek(log_fp, line_pos, SEEK_SET);
        fread(lines[line_count], 1, line_len, log_fp);
        lines[line_count][line_len] = '₩0';

        if (line_len == 0 || lines[line_count][line_len - 1] != '₩n')
        {
            char *new_line = realloc(lines[line_count], line_len + 2);
            if (new_line) strcat(lines[line_count], "₩n");
        }

        line_count++;
        end_pos = pos;
    }
}

// 처음부터 읽은 경우 처리
if (pos == 0 && line_count < 10)
{
    fseek(log_fp, 0, SEEK_SET);
    size_t line_len = end_pos;
    lines[line_count] = malloc(line_len + 1);
    fread(lines[line_count], 1, line_len, log_fp);
    lines[line_count][line_len] = '₩0';
    if (line_len == 0 || lines[line_count][line_len - 1] != '₩n')
    {
        char *new_line = realloc(lines[line_count], line_len + 2);
```

```
                    if (new_line) strcat(lines[line_count], "₩n");
                }
                line_count++;
            }

            // 로그 역순 출력
            for (int i = line_count - 1; i >= 0; i--)
            {
                printf("%s", lines[i]);
                free(lines[i]);
            }

            fclose(log_fp);
        }
        else
        {
            printf("No log entries found₩n"); // 로그 없음
        }
        show_daemon_list();
    }
}

/**
 * 새로운 데몬 프로세스를 추가하는 함수
 */
void handle_add(char *args)
{
    char *argv[MAX_BUF];
    int argc = 0;
    char *saveptr;

    // 1. 경로 인자 추출
    char *dir_path = strtok_r(args, " ", &saveptr);
    if (!dir_path) {
        printf("Usage: add <DIR_PATH> [OPTION]...₩n"); // 사용법 안내
        return;
    }

    // 2. 절대 경로 변환 및 유효성 검사
    char *abs_path = get_absolute_path(dir_path);
    if (!abs_path) {
        printf("Invalid path: %s₩n", dir_path); // 잘못된 경로
        return;
```

```
    }

    if (!is_inside_home_directory(abs_path)) {
        printf("%s is outside the home directory₩n", abs_path); // 홈 디렉토리 외부
        free(abs_path);
        return;
    }

    if (!is_directory(abs_path)) {
        printf("%s is not a directory₩n", abs_path); // 디렉토리 아님
        free(abs_path);
        return;
    }

    if (is_path_in_daemon_list(abs_path)) {
        printf("%s is already being monitored₩n", abs_path); // 이미 모니터링 중
        free(abs_path);
        return;
    }

    // 3. 상위/하위 디렉토리 중복 검사
    char list_path[MAX_PATH_LEN];
    snprintf(list_path, sizeof(list_path), "%s/%s", get_home_directory(), DAEMON_LIST_FILE);
    FILE *fp = fopen(list_path, "r");
    if (fp) {
        char line[MAX_PATH_LEN + 50];
        while (fgets(line, sizeof(line), fp)) {
            char *path_start = strchr(line, '"');
            char *path_end = path_start ? strchr(path_start + 1, '"') : NULL;
            if (!path_start || !path_end) continue;
            *path_end = '₩0';
            char *registered_path = path_start + 1;

            if (is_subdirectory(abs_path, registered_path) || is_subdirectory(registered_path, abs_path)) {
                printf("Directory %s is already being monitored or overlaps with monitored directory %s₩n",
abs_path, registered_path);
                fclose(fp);
                free(abs_path);
                return;
            }
        }
        fclose(fp);
    }
```

```
// 4. 모든 인자 저장
argv[argc++] = abs_path;
while ((argv[argc] = strtok_r(NULL, " ", &saveptr))) {
    argc++;
    if (argc >= MAX_BUF - 1) break;
}

// 5. 기본 설정 초기화
DaemonConfig config;
memset(&config, 0, sizeof(config));
strncpy(config.monitoring_path, abs_path, MAX_PATH_LEN);

// 기본 출력 경로 설정
char output_path[MAX_PATH_LEN];
char *last_slash = strrchr(abs_path, '/');
if (last_slash) {
    char dir_name[MAX_FILENAME_LEN];
    strcpy(dir_name, last_slash + 1);
    snprintf(output_path, sizeof(output_path), "%.*s/%s_arranged", (int)(last_slash - abs_path), abs_path, dir_name);
} else {
    snprintf(output_path, sizeof(output_path), "%s_arranged", abs_path);
}
strncpy(config.output_path, output_path, MAX_PATH_LEN);

// 기본값 설정
config.time_interval = DEFAULT_INTERVAL;
strcpy(config.max_log_lines, "none");
config.mode = DEFAULT_MODE;
strcpy(config.exclude_paths, "none");
strcpy(config.extensions, "all");

// 6. 옵션 처리
for (int i = 1; i < argc; ) {
    if (strcmp(argv[i], "-d") == 0 && i + 1 < argc) {
        // 출력 경로 옵션 처리
        char *output_abs = get_absolute_path(argv[++i]);
        if (!output_abs || !is_directory(output_abs)) {
            printf("Invalid output path: %s₩n", argv[i]);
            return;
        }
        if (!is_inside_home_directory(output_abs)) {
            printf("%s is outside the home directory₩n", output_abs);
            return;
```

```
        }
        if (is_subdirectory(abs_path, output_abs)) {
            printf("Output path cannot be a subdirectory of monitoring path₩n");
            return;
        }
        strncpy(config.output_path, output_abs, MAX_PATH_LEN);
    }
    else if (strcmp(argv[i], "-i") == 0 && i+1 < argc) {
        // 시간 간격 옵션 처리
        if (i + 1 >= argc) { printf("Missing -i argument₩n"); return; }
        char *interval_str = argv[++i];
        int valid = 1;
        for (int j = 0; interval_str[j] != '₩0'; j++) {
            if (!isdigit(interval_str[j])) valid = 0;
        }
        if (!valid || atoi(interval_str) <= 0) {
            printf("Invalid time interval: %s (must be positive integer)₩n", interval_str);
            return;
        }
        config.time_interval = atoi(interval_str);
    }
    else if (strcmp(argv[i], "-l") == 0 && i+1 < argc) {
        // 최대 로그 라인 옵션 처리
        if(i + 1 >= argc) { printf("Missing -l argument₩n"); return; }
        char *max_lines_str = argv[++i];
        int valid = 1;
        for (int j = 0; max_lines_str[j] != '₩0'; j++) {
            if (!isdigit(max_lines_str[j])) valid = 0;
        }
        if (!valid || atoi(max_lines_str) <= 0) {
            printf("Invalid max log lines: %s (must be positive integer)₩n", max_lines_str);
            return;
        }
        strncpy(config.max_log_lines, max_lines_str, sizeof(config.max_log_lines));
    }
    else if (strcmp(argv[i], "-x") == 0 && i + 1 < argc) {
        // 제외 경로 옵션 처리
        char *exclude_path[MAX_BUF] = {0};
        int exclude_count = 0;
        char built_exclude[MAX_PATH_LEN] = "";
        int first = 1;

        while (i + 1 < argc && argv[i + 1][0] != '-') {
```

```
            char *exclude_abs = get_absolute_path(argv[++i]);
            if (!exclude_abs || !is_directory(exclude_abs)) {
                printf("Invalid exclude path: %s₩n", argv[i]);
                return;
            }
            if (!is_inside_home_directory(exclude_abs)) {
                printf("%s is outside home directory₩n", exclude_abs);
                return;
            }
            if (!is_subdirectory(abs_path, exclude_abs)) {
                printf("%s is not a subdirectory of %s₩n", exclude_abs, abs_path);
                return;
            }

            for (int j = 0; j < exclude_count; j++) {
                if (strcmp(exclude_path[j], exclude_abs) == 0 ||
                    is_subdirectory(exclude_path[j], exclude_abs) ||
                    is_subdirectory(exclude_abs, exclude_path[j])) {
                    printf("Exclude path %s overlaps with %s₩n", exclude_abs, exclude_path[j]);
                    return;
                }
            }

            exclude_path[exclude_count++] = exclude_abs;
            if (!first) strcat(built_exclude, ",");
            strcat(built_exclude, exclude_abs);
            first = 0;
        }
        strncpy(config.exclude_paths, built_exclude, MAX_PATH_LEN);
    }
    else if (strcmp(argv[i], "-e") == 0 && i + 1 < argc) {
        // 확장자 옵션 처리
        char extensions[MAX_PATH_LEN] = "";
        while (i + 1 < argc && argv[i + 1][0] != '-') {
            if (extensions[0]) strcat(extensions, ",");
            strcat(extensions, argv[++i]);
        }
        strncpy(config.extensions, extensions, MAX_PATH_LEN);
    }
    else if (strcmp(argv[i], "-m") == 0 && i + 1 < argc) {
        // 모드 옵션 처리
        char *mode_str = argv[++i];
        for (int j = 0; mode_str[j]; j++) {
```

```
                if (!isdigit(mode_str[j])) {
                    printf("Invalid mode: %s₩n", mode_str);
                    return;
                }
            }
            config.mode = atoi(mode_str);
            if (config.mode < 0 || config.mode > 3) {
                printf("Invalid mode: %d (must be between 0 and 3)₩n", config.mode);
                return;
            }
        }
        else {
            printf("Unknown option: %s₩n", argv[i]); // 알 수 없는 옵션
            return;
        }
        i++;
    }

    // 7. 출력 디렉토리 생성
    if (create_directory(config.output_path) == 0) {
        printf("Failed to create output directory: %s₩n", config.output_path); // 디렉토리 생성 실패
        return;
    }

    // 8. 데몬 프로세스 생성
    pid_t pid = fork();
    if (pid < 0) {
        perror("fork failed"); // fork 실패
        return;
    } else if (pid == 0) {
        // 자식 프로세스: 데몬 실행
        int stdout_backup = dup(STDOUT_FILENO);
        freopen("/dev/null", "w", stdout);
        run_as_daemon(config);
        dup2(stdout_backup, STDOUT_FILENO);
        close(stdout_backup);
        exit(0);
    } else {
        waitpid(pid, NULL, 0); // 부모 프로세스: 자식 프로세스 대기
    }

    free(abs_path);
}
```

```
/**
 * 기존 데몬 프로세스 설정을 수정하는 함수
 */
void handle_modify(char *args)
{
    // 1. 인자 파싱 준비
    char *argv[MAX_BUF];
    int argc = 0;
    char *saveptr;

    // 첫 토큰 분할 (경로 확인용)
    char *dir_path = strtok_r(args, " ", &saveptr);
    if (!dir_path) {
        printf("Usage: add <DIR_PATH> [OPTION]...₩n"); // 사용법 안내
        return;
    }

    // 2. 절대 경로 확인 및 유효성 검사
    char *abs_path = get_absolute_path(dir_path);
    if (!abs_path) {
        printf("Invalid path: %s₩n", dir_path); // 잘못된 경로
        return;
    }

    if (!is_inside_home_directory(abs_path))
    {
        printf("%s is outside the home directory₩n", abs_path); // 홈 디렉토리 외부
        return;
    }

    if (!is_directory(abs_path))
    {
        printf("%s is not a directory₩n", abs_path); // 디렉토리 아님
        return;
    }

    if (!is_path_in_daemon_list(abs_path))
    {
        printf("%s is not being monitored₩n", abs_path); // 모니터링 중 아님
        return;
    }
```

```
// 3. 모든 인자 저장
argv[argc++] = abs_path;
while ((argv[argc] = strtok_r(NULL, " ", &saveptr))) {
    argc++;
    if (argc >= MAX_BUF-1) break;
}

// 4. 설정 파일 잠금 및 읽기
char config_path[MAX_PATH_LEN];
snprintf(config_path, sizeof(config_path), "%s/ssu_cleanupd.config", abs_path);

int fd = lock_config(abs_path);
if (fd == -1) {
    printf("Failed to lock config file₩n"); // 설정 파일 잠금 실패
    return;
}

DaemonConfig config;
if (read_config(abs_path, &config, fd) == -1) {
    printf("Failed to read config file₩n"); // 설정 파일 읽기 실패
    close(fd);
    return;
}

// 5. 옵션 처리
for (int i = 1; i < argc; ) {
    if (strcmp(argv[i], "-d") == 0 && i + 1 < argc) {
        // 출력 경로 옵션 처리
        char *output_abs = get_absolute_path(argv[++i]);
        if (!output_abs || !is_directory(output_abs)) {
            printf("Invalid output path: %s₩n", argv[i]);
            return;
        }
        if (!is_inside_home_directory(output_abs)) {
            printf("%s is outside the home directory₩n", output_abs);
            return;
        }
        if (is_subdirectory(abs_path, output_abs)) {
            printf("Output path cannot be a subdirectory of monitoring path₩n");
            return;
        }
        strncpy(config.output_path, output_abs, MAX_PATH_LEN);
    }
```

```
else if (strcmp(argv[i], "-i") == 0 && i+1 < argc) {
    // 시간 간격 옵션 처리
    if (i + 1 >= argc) { printf("Missing -i argument₩n"); return; }
    char *interval_str = argv[++i];
    int valid = 1;
    for (int j = 0; interval_str[j] != '₩0'; j++) {
        if (!isdigit(interval_str[j])) valid = 0;
    }
    if (!valid || atoi(interval_str) <= 0) {
        printf("Invalid time interval: %s (must be positive integer)₩n", interval_str);
        return;
    }
    config.time_interval = atoi(interval_str);
}
else if (strcmp(argv[i], "-l") == 0 && i+1 < argc) {
    // 최대 로그 라인 옵션 처리
    if(i + 1 >= argc) { printf("Missing -l argument₩n"); return; }
    char *max_lines_str = argv[++i];
    int valid = 1;
    for (int j = 0; max_lines_str[j] != '₩0'; j++) {
        if (!isdigit(max_lines_str[j])) valid = 0;
    }
    if (!valid || atoi(max_lines_str) <= 0) {
        printf("Invalid max log lines: %s (must be positive integer)₩n", max_lines_str);
        return;
    }
    strncpy(config.max_log_lines, max_lines_str, sizeof(config.max_log_lines));
}
else if (strcmp(argv[i], "-x") == 0 && i + 1 < argc) {
    // 제외 경로 옵션 처리
    char *exclude_path[MAX_BUF] = {0};
    int exclude_count = 0;
    char built_exclude[MAX_PATH_LEN] = "";
    int first = 1;

    while (i + 1 < argc && argv[i + 1][0] != '-') {
        char *exclude_abs = get_absolute_path(argv[++i]);
        if (!exclude_abs || !is_directory(exclude_abs)) {
            printf("Invalid exclude path: %s₩n", argv[i]);
            return;
        }
        if (!is_inside_home_directory(exclude_abs)) {
            printf("%s is outside home directory₩n", exclude_abs);
```

```
                return;
            }
            if (!is_subdirectory(abs_path, exclude_abs)) {
                printf("%s is not a subdirectory of %s₩n", exclude_abs, abs_path);
                return;
            }

            for (int j = 0; j < exclude_count; j++) {
                if (strcmp(exclude_path[j], exclude_abs) == 0 ||
                    is_subdirectory(exclude_path[j], exclude_abs) ||
                    is_subdirectory(exclude_abs, exclude_path[j])) {
                    printf("Exclude path %s overlaps with %s₩n", exclude_abs, exclude_path[j]);
                    return;
                }
            }

            exclude_path[exclude_count++] = exclude_abs;
            if (!first) strcat(built_exclude, ",");
            strcat(built_exclude, exclude_abs);
            first = 0;
        }
        strncpy(config.exclude_paths, built_exclude, MAX_PATH_LEN);
    }
    else if (strcmp(argv[i], "-e") == 0 && i+1 < argc) {
        // 확장자 옵션 처리
        if (i + 1 >= argc) { printf("Missing -e argument₩n"); return; }
        char extensions[MAX_PATH_LEN] = "";
        while (i+1 < argc && argv[i+1][0] != '-') {
            if (extensions[0]) strcat(extensions, ",");
            strcat(extensions, argv[++i]);
        }
        strncpy(config.extensions, extensions, MAX_PATH_LEN);
    }
    else if (strcmp(argv[i], "-m") == 0 && i + 1 < argc) {
        // 모드 옵션 처리
        char *mode_str = argv[++i];
        for (int j = 0; mode_str[j]; j++) {
            if (!isdigit(mode_str[j])) {
                printf("Invalid mode: %s₩n", mode_str);
                return;
            }
        }
        config.mode = atoi(mode_str);
```

```
            if (config.mode < 0 || config.mode > 3) {
                printf("Invalid mode: %d (must be between 0 and 3)₩n", config.mode);
                return;
            }
        }
        else {
            printf("Unknown option: %s₩n", argv[i]); // 알 수 없는 옵션
            return;
        }
        i++;
    }

    // 6. 설정 파일 업데이트
    if (write_config(abs_path, &config, fd) == -1) {
        printf("Failed to update config file₩n"); // 설정 파일 업데이트 실패
        close(fd);
        return;
    }

    unlock_config(fd); // 설정 파일 잠금 해제
}

/**
 * 데몬 프로세스를 제거하는 함수
 */
void handle_remove(char *args)
{
    // 1. 인자 파싱
    char *token = strtok(args, " ");
    if (!token)
    {
        printf("Usage: remove <DIR_PATH>₩n"); // 사용법 안내
        return;
    }

    char *dir_path = token;
    char *abs_path = get_absolute_path(dir_path);
    if (!abs_path)
    {
        printf("Invalid path: %s₩n", dir_path); // 잘못된 경로
        return;
    }
```

```
// 2. 유효성 검사
if (!is_inside_home_directory(abs_path))
{
    printf("%s is outside the home directory₩n", abs_path); // 홈 디렉토리 외부
    return;
}

if (!is_directory(abs_path))
{
    printf("%s is not a directory₩n", abs_path); // 디렉토리 아님
    return;
}

if (!is_path_in_daemon_list(abs_path))
{
    printf("%s is not being monitored₩n", abs_path); // 모니터링 중 아님
    return;
}

// 3. 데몬 리스트에서 PID 추출
char list_path[MAX_PATH_LEN];
snprintf(list_path, sizeof(list_path), "%s/%s", get_home_directory(), DAEMON_LIST_FILE);

FILE *fp = fopen(list_path, "r");
if (!fp)
{
    perror("fopen"); // 파일 열기 실패
    return;
}

char line[MAX_PATH_LEN + 50];
pid_t pid = 0;

while (fgets(line, sizeof(line), fp))
{
    line[strcspn(line, "₩n")] = '₩0';  // 개행 제거
    char *path_start = strchr(line, '"');
    if (!path_start) continue;

    char *path_end = strchr(path_start + 1, '"');
    if (!path_end) continue;

    *path_end = '₩0';
```

```
            if (strcmp(path_start + 1, abs_path) == 0)
            {
                char *pid_start = strchr(path_end + 1, ',');
                if (pid_start) pid = atoi(pid_start + 1);
                break;
            }
        }
        fclose(fp);

        if (pid == 0)
        {
            printf("Failed to find daemon process for %s₩n", abs_path); // 데몬 프로세스 찾기 실패
            return;
        }

        // 4. 데몬 프로세스 종료
        if (kill(pid, SIGTERM) == -1)
        {
            perror("kill"); // 프로세스 종료 실패
            return;
        }

        // 5. 데몬 리스트에서 제거
        if (remove_from_daemon_list(abs_path) == 0)
        {
            printf("Failed to remove from daemon list₩n"); // 리스트에서 제거 실패
            return;
        }
}

/**
 * 도움말을 출력하는 함수
 */
void handle_help()
{
        printf("Usage:₩n");
        printf("> show₩n");
        printf("      <none> : show monitoring daemon process info₩n"); // 데몬 프로세스 정보 표시
        printf("> add <DIR_PATH> [OPTION]...₩n");
        printf("      <none> : add daemon process monitoring the <DIR_PATH> directory₩n"); // 데몬 추가
        printf("      -d <OUTPUT_PATH> : Specify the output directory <OUTPUT_PATH> where <DIR_PATH> will be arranged₩n"); // 출력 디렉토리 지정
        printf("      -i <TIME_INTERVAL> : Set the time interval for the daemon process to monitor in seconds.₩n"); // 모니터링 간격 설정
```

```
    printf("      -l <MAX_LOG_LINES> : Set the maximum number of log lines the daemon process will record.₩n"); // 최대 로그 라인 수 설정
    printf("      -x <EXCLUDE_PATH1, EXCLUDE_PATH2, ...> : Exclude all subfiles in the specified directories.₩n"); // 제외 경로 설정
    printf("      -e <EXTENSION1, EXTENSION2, ...> : Specify the file extensions to be organized.₩n"); // 확장자 지정
    printf("      -m <M> : Specify the value for the <M> option.₩n"); // 모드 설정
    printf("> modify <DIR_PATH> [OPTION]...₩n");
    printf("      <none> : modify daemon process monitoring the <DIR_PATH> directory₩n"); // 데몬 설정 수정
    printf("> remove <DIR_PATH>₩n");
    printf("      <none> : remove daemon process monitoring the <DIR_PATH> directory₩n"); // 데몬 제거
    printf("> help₩n"); // 도움말
    printf("> exit₩n"); // 종료
}
```