학과 : 컴퓨터학부 학번 : 20192392 이름 : 김한승

1. 과제 개요

디몬 프로세스를 통한 사용자가 원하는 디렉토리를 모니터링하는 프로그램입니다.

해당 프로그램의 내장 명령어는 총 5가지입니다.

첫째로, add입니다. 사용자는 add 명령어를 통해 원하는 경로를 모니터링 할 수 있게 됩니다.

이 명령어는 절대 경로 와 상대 경로를 모두 지원하며, -t 옵션을 통해 해당 디몬 프로세스가 전체 모니터링이 완료된 후, 재 시작하는데 <TIME> 만큼 걸리게 설정할 수 있습니다. 해당 옵션을 사용하지 않는 경우 디몬 프로세스는 기본 설정으로 전체 모니터링이 완료된 후, 1초 후 재시작 됩니다.

사용자가 프롬프트 상에서 해당 명령어를 사용하게 되면, 사용자가 모니터링 프로그램을 실행 한 작업 디렉토리 내에 “monitor\_list.txt” 상에 디몬 프로세스가 관리하는 모니터링 경로 및 디몬프로세스의 pid를 기록하게 됩니다. “monitor\_list.txt”는 전체 프로그램이 종료된 이후에도 남아있으며, 모니터링이 진행되는 동안, 모니터링 경로 상의 변동사항은 마찬 가지로 사용자가 모니터링 프로그램을 실행 한 작업 디렉토리 내에 “log.txt” 상에 로그를 남기게 됩니다.

두번째로, delete입니다. 사용자는 delete 명령어를 통해 모니터링 중이던 디몬 프로세스를 종료할 수 있습니다.

사용자는 “monitor\_list.txt”내에 기록되지 않은 디몬 프로세스를 종료 시킬 수 없으며, “monitor\_list.txt” 내에 존재하는

모니터링 중이던 디몬 프로세스를 SIGUSR1 시그널을 통해 종료합니다. 명령어를 실행한 이후, 해당 DAEMON\_PID 에

해당하는 디몬 프로세스가 종료되며, “monitor\_list.txt” 내에 종료된 디몬 프로세스의 정보 또한 삭제됩니다.

세번째로, tree입니다. 사용자는 tree 명령어를 통해 현재 모니터링 중인 디몬 프로세스의 정규 파일 및 디렉토리 파일 트리를 확인할 수 있습니다. delete 명령어와 마찬가지로 사용자는 “monitor\_list.txt”내에 기록되지 않은 모니터링 중이지 않은 경로에 대해서는 확인할 수 없으며, 사용자가 프로그램 실행 후, 프롬프트 사용도중 모니터링 경로 상에 변동 사항이 있는 경우에도 명령어를 재 실행할 경우, 결과가 반영 되어 새로운 tree를 출력합니다.

네번째로, help입니다. 사용자는 해당 프로그램의 사용법을 help 명령어를 통해 Usage를 확인할 수 있습니다. 예외로, 해당 프로그램 사용도중 5가지 명령어 외의 정의되지 않은 명령어를 입력할 경우 help 명령어 입력과 같이 Usage를 확인할 수 있습니다.

마지막으로, exit입니다. 사용자는 exit 명령어를 통해 프로그램을 종료할 수 있습니다. 해당 프로그램 사용 도중 add 명령어를 통해 사용자 설정 경로의 모니터링 중인 디몬 프로세스는 종료되지 않고 유지됩니다.

파일이 많을 경우 해당 프로그램의 결과가 “log.txt”에 생각보다 늦게 반영될 수 있습니다. 두 파일 트리 형태의 링크드 리스트를 비교하기 위한 함수가 재귀적으로 호출되며, 해당 함수는 O(N^2)의 시간 복잡도를 가지고 있습니다.

모든 예전 노드 트리 수 \* 모든 새로운 노드 트리 수

그리고, 접근권한이 없는 경로의 경우에는 파일을 모니터링 하지 않습니다. 해당 프로그램이 opendir() 및 수정시간 체크를 위해 파일의 정보를 확인하기 위해 lstat()를 사용하기 전, realpath()를 통해 현재 모니터링 하려는 경로가 접근 가능한 경로인지 체크합니다.

2. 구현 기능.

2-1. 프로그램에 정의된 노드 구조체 형태

2-1-1. node 구조체

typedef struct node{

    char path[1024];

    pid\_t pid;

    struct node \*next;

    struct node\_sub \*child; // 현재 디몬 프로세스가 관리하는 경로 하위 파일 트리의 헤드.

}node;

node는 디몬 프로세스가 관리하는 경로와 디몬 프로세스의 pid를 담고 있는 노드입니다. next 포인터를 통해

다른 디몬 프로세스가 관리하는 경로와 pid를 가진 노드를 형제 노드로 가지며, 자식 노드는 node\_sub 노드로 이루어진

파일 트리를 가집니다. 즉, node는 node\_sub로 이루어진 파일 트리의 최상단 노드입니다.

2-1-2. node\_sub 구조체

typedef struct node\_sub{ // 현재 디몬 프로세스가 모니터링 중인 경로의 하위 파일 및 서브 디렉토리의 경로.

    int type; // 파일이면 0 서브 디렉토리면 1

    int state; // UNCHECKED ~ MODIFIED 0 ~ 4 번 상태. 2,3,4의 경우 로그 파일의 작성.

    char path[1024]; // node의 경로(디렉토리 경로, 파일 경로)

    char name[256]; // node의 이름.(디렉토리 명, 파일 명)

    time\_t node\_mtime;

    struct node\_sub \*next; // 현재 자신과 같은 레벨의 파일 이나 서브 디렉토리 node\_sub

    struct node\_sub \*child; // 현재 자신이 디렉토리 타입이라면 NULL이 아닐수 있음. ".", ".." 제외

}node\_sub;

node\_sub는 디몬 프로세스가 관리하는 node 하위에 관리 하고 있는 서브 디렉토리 및 파일들을 위한 노드입니다. 해당 node\_sub로 이루어진 파일 트리를 node가 child 포인터를 통해 가리키고 있으며, 해당 node\_sub 구조체의 정보로는 type : TYPE\_DIR 혹은 TYPE\_FILE을 통해 서브 디렉토리 노드인지 파일 노드인지를 정보를 가지며, state를 통해 총 5가지 상태(UNCHECKED, CHECKED, CREATED, DELETED, MODIFIED) 에 해당하는 0~4번 정수 정보를 가집니다.

모니터링 중인 파일의 경로와 이름 정보를 가지며, 수정 되었는지 비교를 위해 lstat()을 통해 멤버 st\_mtime 최종 수정 시각 정보를 정보로 가집니다. next를 통해 현재 같은 깊이의 다른 파일이나 서브디렉토리 node\_sub를 가리키며, 해당 node\_sub가 TYPE\_DIR 즉 서브디렉토리 일 경우 하위로 다른 파일 및 서브디렉토리 노드를 가리키는 child 포인터를 가지고 있습니다.

2-2. ssu\_monitor.h 내에 함수 프로토타입 및 기능 설명

2-2-1. void ssu\_monitor()

main 함수에서 해당함수를 호출하며 프로그램을 시작합니다. 해당 함수는 monitor\_list.txt 를 읽어 실행 중인 디몬 프로세스들의 정보를 노드 링크드 리스트로 만듭니다. 전역 변수로 선언된 node \*head에 링크드리스트를 만들며, 이때 read\_daemon()을 호출합니다. 디몬 프로세스들이 관리하는 노드 링크드 리스트를 만든 후, 사용자의 입력을 받기 위한 입력 대기 상태가 되며, 사용자의 입력을 통해 받은 문자열을 통해 do\_order()를 호출합니다. 이때 사용자가 입력한 문자열이 exit 일 경우 입력 대기 상태를 종료 하고 함수를 끝냅니다.

2-2-2. void do\_order(char \*buf)

ssu\_monitor() 함수에서 20192392> 출력 후 사용자의 입력을 buf 버퍼에 담은 후, 해당 함수를 호출 하여 어떤 내장 명령어인지 판단하며, 명령어 수행을 위한 함수를 호출하고 buf 버퍼에 담긴 문자열을 빈칸을 통해 나누어 토큰화 합니다. 해당 토큰화 된 문자열들은 나중에 free\_token() 함수를 통해 메모리 할당 해제합니다.

2-2-3. void print\_usage()

usage 출력을 위한 함수입니다. 내장 명령어 help 수행을 위해 사용되며 사용자가 5가지외의 정의 되지 않은 명령어를 입력 할 경우 usage 출력을 위해 해당 함수를 호출합니다.

2-2-4. void free\_token(int argc, char \*argv[])

사용자의 입력을 argc 와 argv를 통해 동적으로 메모리 할당 하여 토큰화 한 후, 해당 토큰들을 메모리 할당 해제를 위한 함수입니다. argc 만큼 for문을 돌며 argv[] 내에 0번 인덱스부터 argc-1 인덱스까지 메모리를 할당하여 썻던 토큰들을 free() 하여 메모리 할당 해제합니다.

2-2-5. void ssu\_add(int argc, char \*argv[])

내장 명령어 add 수행을 위한 함수입니다. 경로에 해당하는 인자는 상대/절대 경로 무엇이 오던지 get\_path() 함수를 통해 절대 경로로 변환한다. -t 옵션이 있을경우 get\_time() 함수를 통해 -t 옵션 사용을 위한 time 인자가 제대로 되었는지 체크하고, 값을 리턴받는다. 최종적으로, 현재 모니터링 되고 있는 경로들의 상위/하위/동일 경로인지 check\_daemon() 을 통해 확인한 후, init\_daemon()을 실행하여 디몬 프로세스를 생성해 모니터링을 시작한다. 마지막으로, free\_token() 을 통해 인자로 받은 argc, argv 토큰들을 메모리 할당 해제 후 끝마친다.

2-2-6. void ssu\_delete(int argc, char \*argv[])

내장 명령어 delete 수행을 위한 함수입니다. 해당 함수는 크게 3가지 일을 한다. 만약 인자로 받은 pid와 동일한 디몬프로세스가 존재한다면,

1. pop\_node()를 통해 디몬 프로세스의 정보를 가진 링크드리스트에서 해당 디몬 프로세스의 노드를 삭제한다.

2. kill()을 통해 해당 디몬프로세스에 SIGUSR1 시그널을 보내며, 디몬 프로세스를 종료시킨다.

3. erase\_daemon()을 통해 해당 디몬 프로세스의 정보를 “monitor\_list.txt” 에서 제거 후

4. free\_token() 을 통해 인자로 받은 argc, argv 토큰들을 메모리 할당 해제 후 끝마친다.

만약 인자로 받은 pid와 동일한 디몬 프로세스가 존재하지 않거나, 인자의 입력이 잘못 되었다면 해당 4가지 기능을 수행하지않습니다.

2-2-7. void ssu\_tree(int argc, char \*argv[])

내장 명령어 tree 수행을 위한 함수입니다. 해당함수는 프로그램이 시작되면 어떤 디몬 프로세스 들이 있는지 전역 변수 node \*head에 read\_deamon() 함수를 통해 링크드리스트를 만들 듯이, 프로그램 실행 도중 모니터링 되고 있는 경로의 변화가 생길 수 있으므로, ssu\_tree() 함수를 실행 할 때마다 ,read\_daemon() 함수를 실행하여 새롭게 링크드 리스트를 초기화합니다. 새롭게 링크드 리스트를 초기화 하기 전, 이전 링크드리스트는 메모리 할당 해제 후, NULL로 초기화합니다.(segmentation fault 대비) ssu\_tree() 함수는 ssu\_add() 함수와 마찬가지로 절대/상대 경로를 지원해야하므로, get\_path() 함수를 통해 상대/절대 경로 상관없이 절대 경로로 변환합니다. 그리고, “monitor\_list.txt”에 인자로 받은 경로가 있는지 확인 후, read\_daemon()을 통해 만든 디몬 노드 링크드 리스트 중 해당 디몬 프로세스의 노드 하위 node\_sub 파일 트리를 재귀적으로 순회하며 트리를 출력합니다. 이때, 재귀적 호출로 순회하며 트리 출력을 위해 print\_tree() 함수를 사용합니다.

2-2-8. void read\_daemon(node \*\*head)

프로그램 처음 실행 직후, 및 ssu\_tree() 함수를 호출 할 때 마다 해당 함수를 호출합니다. 인자로 들어가는 링크드리스트에 head가 될 node 이중 포인터를 인자로 받는데, 전역 변수로 선언된 node \*head 의 주소 &head 를 인자로 받아 read\_daemon()을 수행 후, “monitor\_list.txt”가 있다면 해당 파일에 기재된 모니터링 중인 디몬프로세스의 정보를 한줄씩 읽어 링크드리스트로 만듭니다. read\_daemon() 내에서 “monitor\_list.txt” 가 있다면 한줄 씩 fgets()를 통해 읽은 후, push\_back\_node()를 통해 링크드리스트 맨 뒤에 노드들을 하나씩 이어갑니다.

2-2-9. void erase\_daemon(const char \*path, pid\_t pid)

ssu\_delete 함수에서 내장 명령어 delete 수행을 위해 해당 함수를 호출합니다. 해당 함수는 “monitor\_list.txt”에서 해당 하는 디몬프로세스가 존재할 경우 해당 정보를 삭제하는데, 이때 monitor\_list\_tmp.txt 라는 임시파일을 만들어 삭제할 디몬 프로세스의 정보를 제외하고 파일에 쓰는 방법으로 해당 정보를 삭제합니다. 삭제할 디몬프로세스를 제외한 다른 디몬 프로세스들의 정보를 작성한 임시파일은 원래의 파일을 unlink()후, rename() 하여 “monitor\_list.txt”를 새로운 결과로 유지합니다.

2-2-10. void init\_daemon(node \*\*head, char \*path, int tOption, int time)

ssu\_add() 함수에서 새롭게 디몬 프로세스를 생성하기 위해 만든 함수입니다. 인자로 링크드리스트의 head를 받는 이유는 디몬 프로세를 새로 생성한 후, 링크드리스트를 새롭게 만드는 것이 아닌, 해당 추가된 디몬 프로세스의 정보를 가진 노드를 추가하기 위함입니다. tOption과 time을 통해 디몬 프로세스가 전체 모니터링 된 후, 재시작 될 시간을 설정합니다.(옵션 없는 경우, 기본 1초)

2-2-11. int check\_daemon(node \*\*head, const char \*path)

ssu\_add() 함수내에서 현재 모니터링 중인 경로들을 확인하면서 path가 모니터링 중인 경로들의 상위/하위/동일 경로인지 확인하는 함수입니다. 해당함수를 통해 모든 모니터링 경로를 확인하면서 path가 상위/하위/동일 경로가 아니라면, FALSE를 리턴하여서 ssu\_add() 함수가 새로운 디몬 프로세스를 생성 할 수 있도록 합니다.

2-2-12. void push\_back\_node(node \*\*head, char \*path, pid\_t pid)

새로운 디몬프로세스의 정보를 가진 노드를 링크드리스트에 추가하기 위한 함수. 해당 함수의 인자로 모니터링 되는 경로와 경로를 모니터링하는 디몬프로세스의 pid값을 통해 노드를 만듭니다.

2-2-13. void pop\_node(node \*\*head, pid\_t pid)

ssu\_delete() 함수에서 쓰이는 함수. 디몬 프로세스의 정보를 가진 노드들의 링크드 리스트를 순차 순회하며, 인자로 넘겨받은 pid에 해당하는 디몬 프로세스가 있다면 링크드 리스트에서 해당 디몬 프로세스 정보를 가진 노드를 삭제한다.

2-2-14. void print\_node(node \*\*head)

디몬 프로세스의 정보를 가진 노드들의 링크드 리스트를 순차 순회하며, 노드들을 출력하기 위한 함수.

해당 함수를 호출하는 함수는 없으며, 테스트 도중 프롬프트에서 현재 디몬 프로세스들이 관리하는 경로 수 만큼 노드가 생성되어 관리 되고 있는지 제대로 확인을 위해 만든 함수입니다.

2-2-15. void ssu\_signal\_handler(int signo);

디몬 프로세스의 7가지 룰을 통해 디몬 프로세스를 생성한 후, 새로운 액션이 될 sigaction 구조체내에 해당 함수를 SIGUSR1의 핸들러 함수로 등록하여 사용합니다. 해당 함수를 통해 디몬 프로세스는 SIGUSR1 시그널을 받으면 모니터링을 중단하고 종료합니다.

2-3. ssu\_daemon.h 내에 함수 프로토타입 및 기능 설명

2-3-1. node\* get\_node(const char\* path, pid\_t pid)

디몬 프로세스 내에서 while 문 내에서 반복적으로 모니터링 경로에 해당하는 디렉토리의 파일 트리를 갖는 node를 생성하는 함수입니다. 해당 함수를 통해 리턴 받은 노드를 compare\_node(예전 노드, 새로 생성한 노드) 를 인자로 받아 비교 하여 변동사항을 모니터링 합니다.

2-3-2. void initialize\_nodes(node \*monitoring\_node) / void initialize\_node(node\_sub \*now\_node)

해당 함수는 compare\_node() -> compare\_node\_sub() -> compare\_info() 순으로 재귀 적으로 모니터링 중인 경로의 파일 트리를 비교 할 때 쓰는 state 값을 초기화 하기 위해 만든 함수 입니다. 해당 함수를 통해 비교가 완료 된 노드를 initialize\_nodes() 함수를 통해 인자로 받은 후, 파일 트리를 순차적으로 탐색하며 initialize\_node()를 호출 하여 해당 node의 하위 node->child (node\_sub 트리) 내에 모든 파일 트리를 이루는 node\_sub 노드들을 초기화합니다.

2-3-3. void compare\_node(node \*old\_node, node \*new\_node)

디몬 프로세스 내에서 get\_node()를 통해 새로 받은 new\_node를 인자로 넣어 비교를 위한 함수입니다. 해당 함수는 현재 프로그램을 실행한 작업 디렉토리 내에 “log.txt” 파일을 열어 파일 디스크립터를 compare\_node\_sub() 함수를 호출 하는데 사용합니다. 현재 모니터링 중인 경로의 새로운 파일 트리가 존재 한다면 compare\_node\_sub()를 호출 하여 비교 한 후, 비교가 완료되면 비교 할 old\_node 파일 트리의 UNCHECKED(새로운 파일 트리에는 삭제되어서 체크 되지 못한) node\_sub 노드들을 삭제 변동사항 작성을 위한 check\_node\_deleted() 함수를 실행 한 후, 파일 디스크립터를 닫은 후, 종료합니다.

2-3-4. void compare\_node\_sub(int fd, node\_sub \*old\_node\_sub, node\_sub \*new\_node\_sub)

compare\_node()를 통해 만들어진 log.txt의 파일 디스크립터와 비교를 위한 두 파일 트리 노드의 head를 인자로 받아 실행합니다. compare\_info()를 통해 비교할 예전 파일 트리의 head와 현재 비교할 새로운 파일 트리 내에 파일을 비교하며, 초기화 된 state를 확인하여서 수정/생성 됬는지 확인합니다. 삭제된 파일의 경우 모든 새로운 파일 트리를 탐색 후 예전 파일 트리 내에 방문 되지 않은 파일 노드를 삭제 처리 해야 하므로, compare\_node() 함수에서 check\_node\_deleted() 함수를 통해 나중에 처리됩니다. 해당 함수가 수정/생성 된 파일을 처리 할 때 get\_time\_string() 과 get\_state\_string()을 통해 로그 작성을 위한 문자열을 time\_string 과 state\_string 문자열에 저장 하고, 인자로 받은 파일 디스크립터에 로그를 작성합니다.

2-3-5. void compare\_info(node\_sub \*old\_node, node\_sub \*new\_node)

compare\_node\_sub()에서 재귀적으로 new\_node\_sub 새로운 파일 트리를 탐색하며, compare\_info()를 호출 하였다면 해당하는 새로운 파일트리의 파일 노드 비교를 위해 compare\_info() 함수에서는 비교 될 예전 파일 트리를 재귀적으로 탐색하며 같은 정보를 가진 노드를 찾습니다. 해당 함수를 통해 같은 노드를 발견한다면 수정 되었는지 여부를 체크하고, node\_sub 파일 노드의 state를 초기화 합니다.

2-3-6. void check\_node\_deleted(int fd, node\_sub \*old\_node)

모든 compare\_node\_sub()를 통해 새로운 파일 트리내에 파일 노드들과 예전 파일 트리 노드들의 비교가 완료 되면, 해당 함수를 통해 예전 파일 트리 노드들 중 UNCHEKCED 된 즉, 새로운 파일 트리에서 찾을 수 없기 때문에 방문 처리 되지 않은 파일 노드들을 모아 삭제 처리를 위한 로그를 작성합니다. 해당 함수는 compare\_node() 함수에서 사용됩니다.

2-3-7. void make\_tree(node\_sub \*\*head, const char \*path)

디몬 프로세스가 관리하는 노드 node를 생성할때마다 해당 node가 가진 모니터링 경로의 하위 파일 트리를 만들기 위해 호출되는 함수입니다. push\_back\_node() 과 get\_node() 함수에서 디몬 프로세스가 관리하는 노드를 생성할 때 해당 함수를 호출하여 관리 중인 경로의 하위 파일 트리 node\_sub 트리를 만듭니다. node\_sub \*\*head는 node->child 포인터의 주소가 들어가 디몬 프로세스가 관리하는 노드가 해당 파일 트리를 가리 킬 수 있도록 합니다.

2-3-8. void print\_tree(node\_sub \*\*head, int depth)

ssu\_tree() 함수에서 모니터링 되고 있는 경로 중 하나의 파일 트리를 출력 하기 위해 사용하는 함수입니다. 해당 함수를 통해 현재 출력하려는 노드가 디렉토리인 경우. 본인을 출력 후, 재귀적으로 자신의 자식이 되는 파일 및 서브 디렉토리들을 출력 하기 위해 다시 호출 합니다. 현재 인자로 받은 노드의 본인과 자식을 모두 출력했다면 다음 형제 노드를 출력합니다.

2-3-9. void get\_time\_string(char \*time\_string)

노드의 state가 CREATED, DELETED, MODIFIED 3가지 경우 중 하나에 해당하여 “log.txt”에 변동사항을 작성하기 위해 현재 변동 시점을 작성하기 위해 time\_string 배열내에 저장하는 함수입니다.

2-3-10. void get\_state\_string(int state, char \*state\_string)

get\_time\_string()과 마찬 가지로, 노드의 state가 CREATED, DELETED, MODIFIED 3가지 경우 중 하나에 해당하여 “log.txt”에 변동사항을 작성하기 위해 어떤 변동사항을 가졌는지 인자로 받은 state\_string 배열내에 저장하는 함수입니다.

2-3-11. void print\_state(node\_sub \*monitoring\_node);

node\_sub 노드를 인자로 받아 해당 노드의 하위 파일 및 서브 디렉토리를 순차 적으로 탐색하며, 노드의 state를 출력 하기 위한 함수입니다. 해당 함수는 프로그램 도중 파일 트리의 노드들이 정상적으로 state가 초기화 되며, 비교 후 새로 생긴 노드, 삭제 된 노드, 수정 된 노드의 state가 정상적으로 초기화 됐는지 확인을 위한 함수입니다. 실제 프로그램 실행 도중에는 호출 되는 일이 없습니다. 테스트를 위한 함수입니다.

3. 상세 설계

함수가 다른 함수를 호출하지 않고, 단일로 동작하여 함수 자체에서 정의된 작업만을 수행하는 함수의 경우에는 콜 그래프에서 제외하였습니다. 프로그램 실행 후, 흐름에 따라 함수가 다른 함수를 호출하는 경우 상세 설계에 콜 그래프를 통해 설명하였습니다.

3-1. main 함수의 콜 그래프

텍스트, 도표, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

main 함수에서는 ssu\_monitor() 함수를 호출 하여서 모니터링 프롬프트를 통해 사용자의 입력을 받기 위한 입력 대기 상태가 됩니다. exit 명령어를 통해 ssu\_monitor() 함수를 종료 하였을 경우 main() 함수로 돌아와 프로그램을 종료 합니다.

3-2. ssu\_monitor() 함수의 콜 그래프

도표, 텍스트, 라인, 패턴이(가) 표시된 사진

자동 생성된 설명

main() 함수에서 호출을 받은 직후, “monitor\_list.txt” 가 존재한다면 해당 txt 파일내에 모니터링 중인 디몬 프로세스를 read\_daemon() 함수를 통해 읽어 전역 변수로 선언된 node \*head를 링크드 리스트에 헤드로 하여서 모니터링 중인 디몬 프로세스에 대한 링크드리스트를 만듭니다. 그 후, while() 문을 통해 입력 대기 상태가 되며, 엔터만을 사용자가 눌렀을 경우에는 다시 프롬프트를 재 출력 하고, exit 명령어를 입력 받았다면 while 반복문을 탈출하여 함수를 종료합니다.그 외에 엔터(개행문자)와 “exit” 내부 명령어를 제외한 입력에 대해서는 do\_order() 함수를 호출하여 명령어를 수행합니다.

3-3. do\_order() 함수의 콜 그래프

도표, 스크린샷, 라인, 디자인이(가) 표시된 사진

자동 생성된 설명

do\_order() 함수에서는 ssu\_monitor() 함수에서 명령어 처리를 위해 만든 함수입니다. 해당 함수를 통해 인자로

받은 buf 문자열은 argv 배열에 문자열을 토큰화하여 저장하며, 토큰화 된 문자열의 개수는 argc입니다.

토큰화 된 문자열 중 첫번째 문자열을 통해 내부 명령어인지 판단 후, ssu\_add(), ssu\_delete(), ssu\_tree() 함수를 각각 내부

명령어 add / delete / tree 일 경우 호출합니다. 그 외의 정의되지 않은 명령어 거나, help 내부 명령어 일 경우 print\_usage()를 통해 프로그램의 사용법(usage)를 출력하여 함수를 종료합니다.

3-4. ssu\_add() 함수의 콜 그래프

도표, 라인, 종이접기이(가) 표시된 사진

자동 생성된 설명

도표, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명

ssu\_add() 함수는 함수를 종료 하기 전, 모든 케이스에서 free\_token() 함수를 통해 do\_order() 함수 내에서 동적 으로 메모리 할당한 argv 배열내에 문자열들을 메모리 할당 해제 합니다. add 명령어 수행은 2가지로 나눌 수 있습니다. -t 옵션이 있는 argc 4인 경우와 옵션이 없는 argc 2인 경우, 그 두 가지 경우가 아니라면 함수를 종료합니다. 옵션이 있다면 -t 옵션이 argv 배열내 “-t” 문자열이 있는지 확인 한 후, -t 문자열 다음 <TIME> 인자에 해당하는 문자열을 get\_time() 함수를 통해 정수형 변수 time에 값을 저장합니다. 이때, time이 음수이거나, “-t” 문자열이 없다면 함수를 종료합니다. 그 뒤, add 명령어의 절대/ 상대 경로 지원을 위해 get\_path() 함수를 통해 절대 경로를 반환하고, 해당 함수가 FALSE를 리턴하며 정상적으로 절대 경로를 저장하지 못했다면, 함수를 종료합니다. lstat() / S\_ISDIR() 두 함수를 통해 절대 경로가 디렉토리 경로인지 확인 하고, 디렉토리 경로가 아니라면, 함수를 종료합니다. check\_daemon()을 통해 마지막으로, 디몬 프로세스를 통해 모니터링 할 경로가 이미 모니터링 중인 경로들의 상위/하위/동일 경로에 해당하는지 확인을 마친 후, 아니라면 init\_daemon()을 통해 디몬 프로세스를 생성합니다. 이미 모니터링 중인 경로들의 상위/하위/동일 경로 일 경우 함수를 종료합니다. init\_daemon()까지 성공적으로 마친 경우 함수를 종료합니다.

3-5. ssu\_delete() 함수의 콜 그래프

도표, 라인, 기술 도면, 평면도이(가) 표시된 사진

자동 생성된 설명

정상적으로 명령어 사용을 위해 인자가 delete <DAEMON\_PID> 로 argc 2개 가 들어온 것이 아니라면 free\_token후 함수를 종료합니다. <DAEMON\_PID>에 해당하는 디몬 프로세스가 있는지, 전역 변수로 선언된 node \*head를 통해 순차 탐색하여서, 확인합니다. 인자에 해당하는 디몬 프로세스가 없다면, free\_token() 후 함수를 종료합니다. 해당하는 디몬 프로세스가 있을 경우, pop\_node()를 통해 디몬 프로세스의 정보를 가진 링크드 리스트에서 해당 디몬 프로세스 정보를 가진 노드를 제거합니다. 그 후, kill()을 통해 해당 디몬 프로세스에 SIGUSR1 시그널을 보내 디몬 프로세스를 종료시킵니다. 마지막으로, erase\_daemon()을 통해 “monitor\_list.txt”에 해당 디몬 프로세스의 정보를 삭제 한 후, free\_token() 후 함수를 종료합니다.

3-6. ssu\_tree() 함수의 콜 그래프

도표, 기술 도면, 라인, 종이접기이(가) 표시된 사진

자동 생성된 설명

tree 함수의 인자로 2개 미만으로 들어오면 free\_token() 후 함수를 종료합니다. get\_path() 함수를 통해 add

과 마찬가지로 상대/절대 경로를 지원하기 위해 함수를 호출하고, 정상적으로 절대 경로를 반환 받지 못했다면 free\_token()후 함수를 종료합니다. lstat()과 S\_ISDIR() 을 통해 경로가 디렉토리 경로인지 검사 하며, 디렉토리 경로가 아니라면 함수를 free\_token() 후 종료합니다. 정상적으로 인자가 들어와있고, 디렉토리 경로임을 확인 하였다면, 전역 변수로 선언된 디몬 프로세스의 링크드 리스트의 노드들을 모두 삭제한 후, read\_daemon() 을 통해 새롭게 링크드 리스트를 만듭니다. (프로그램 실행 도중 모니터링 경로 상의 파일 변동사항의 적용을 위함) 이러한 작업을 통해 프롬프트를 켜놓고 내부 명령어들을 실행하던 와중에 모니터링 경로의 파일의 변동사항이 생겨도 tree 명령어를 사용할 경우 정상적으로 모니터링 경로의 파일 트리를 출력할 수 있습니다. 이 후, tree 명령어를 수행할 디몬 프로세스가 있는지 경로를 확인 한 후,print\_tree() 함수를 통해 트리를 출력한 후 , free\_token() 하여 함수를 종료합니다.

3-7. init\_daemon() 함수의 콜 그래프

텍스트, 도표, 스크린샷, 평면도이(가) 표시된 사진

자동 생성된 설명

ssu\_add() 함수에서 호출하는 init\_daemon() 함수입니다. 해당 함수는 fork()를 통해 자식 프로세스를 생성하고, 자식 프로세스에서 디몬 프로세스 생성을 위한 7가지 룰을 적용하여서 디몬 프로세스로서 동작합니다. fork()를 통해 부모 프로세스는 “monitor\_list.txt”에 새롭게 생성하는 디몬 프로세스 정보를 작성하며 링크드 리스트에 해당 디몬 프로세스의 정보를 가진 노드를 추가 한 후, 프롬프트 출력을 위해 ssu\_add() 함수로 return 합니다. 디몬 프로세스 프로세스는 while 루프를 통해 경로 모니터링을 시작하기 전, sigaction()을 통해 SIGUSR1에 대한 새로운 액션을 설정합니다. 이후, while 루프를 통해 지정된 경로를 모니터링합니다.

while 루프 내부에서는 daemon\_node\_old 와 daemon\_node\_new를 비교합니다. 흐름대로 설명하겠습니다. 첫째로, 모니터링이 재시작 될 때 대기시간 만큼 sleep()을 통해 대기합니다. 이 후, daemon\_node\_new = get\_node()를 통해 최신 상태의 모니터링 경로의 파일 트리를 가진 노드를 리턴받습니다. compare\_node(daemon\_node\_old, daemon\_node\_new) 를 통해 두 노드의 파일 트리를 비교합니다. 비교가 끝나면 , daemon\_node\_old의 자식인 daemon\_node\_old->child 파일 트리를 free\_node\_subtree()를 통해 메모리 할당 해제 후, daemon\_node\_old를 free() NULL로 초기화 하고,

daemon\_node\_old = daemon\_node\_new로 하여서, 새로운 파일 트리를 가진 노드가 daemon\_node\_old가 되게합니다. 이후, daemon\_node\_new = NULL 하고, 파일 트리의 파일 노드들의 상태를 초기화 하기 위해 initialize\_nodes() 를 통해 state를 초기화합니다. 이렇게, 모니터링 경로의 파일 변동사항을 확인합니다.

3-8. compare\_node() 함수의 콜 그래프

도표, 텍스트, 라인, 스크린샷이(가) 표시된 사진

자동 생성된 설명

compare\_node() 함수는 디몬 프로세스 내에서 while 루프를 통해 지속적으로 예전 파일 트리와 새로운 파일 트리를 비교하기 위한 함수 입니다. 해당 함수를 통해 비교할 디몬 노드 old\_node 나 new\_node 중 하나라도 NULL 이라면 비교 하지 않고 바로 종료합니다. new\_node 새로운 파일 트리를 가진 디몬 프로세스의 노드의 child 즉, 파일 트리를 가리키는 포인터가 NULL이 아니라면 compare\_node\_sub()를 호출하여 비교합니다. 예전 파일 트리와 새로운 파일 트리의 비교가 완료되면, 예전 파일트리에 state가 UNCHECKED(새로운 파일트리에는 파일 노드가 없어서 방문 처리가 되지 않은 상태)인 파일 노드들을 모아서 삭제 변동사항을 작성합니다.

3-9. compare\_node\_sub() 함수의 콜 그래프

도표, 라인, 텍스트, 종이접기이(가) 표시된 사진

자동 생성된 설명도표, 라인, 텍스트, 그래프이(가) 표시된 사진

자동 생성된 설명

. compare\_node\_sub() 함수는 새로운 파일 트리의 노드들을 탐색하면서 재귀적으로 다음 형제 노드 또는 자식 노드들을 인자로 넣어 재귀적으로 호출하며 모든 새로운 파일 트리의 노드들의 상태를 초기화 합니다. 한번의 함수 흐름은 하나의 새로운 파일 트리의 파일 노드를 모든 예전 파일 트리에서 찾아 비교하는 것입니다.

node\_sub(파일 노드) 의 type은 두가지 입니다. TYPE\_DIR(디몬 프로세스의 모니터링 경로의 서브 디렉토리) / TYPE\_FILE(모니터링 되는 파일) TYPE\_FILE일 경우에만 비교 하여 로그를 작성하며, 서브 디렉토리 인 경우 접근권한 체크 후, 재귀적으로 호출을 어떻게 할 것인지 결정합니다.

compare\_node() 에서 열어서 인자로 넘겨진 “log.txt” 에 대한 파일 디스크립터에 compare\_info() 를 통해 비교가 끝난, 새로운 파일 트리의 해당 노드를 UNCHECKED(예전 파일트리에는 파일 노드가 없어서 방문 처리가 되지 않은 상태) 새로 생성 / MODIFIED(예전 파일 트리에 파일 노드가 존재하지만, 새로운 파일 트리와 최종 수정시각이 다른 상태) 2가지 생성 / 수정 상태에 해당하는 경우인지 확인 후, 변동사항을 작성합니다. 여기서 compare\_node\_sub()를 재귀적으로 호출 하는 것은 현재 새로운 파일트리의 비교중인 현재 노드의 경로가 접근권한이 있다면 현재 파일노드의 하위 서브 디렉토리 및 파일 탐색을 위해 new\_node\_sub->next / new\_node\_sub->child 두번 재귀적으로 호출 하며, 현재 노드의 경로가 접근권한이 없다면, 현재 파일노드의 하위 서브 디렉토리 및 파일 탐색을 건너뛰고, new\_node\_sub->next 형제 노드만 한번 재귀적으로 호출 합니다.

3-10. compare\_info() 함수의 콜 그래프

도표, 라인, 스케치, 종이접기이(가) 표시된 사진

자동 생성된 설명도표, 텍스트, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명

compare\_node\_sub가 새로운 파일 트리의 노드들을 재귀적으로 인자로 넣어 호출하면서 하나의 새로운 파일 트리의 노드를 비교하는 함수라면, compare\_info() 함수는 예전 파일 트리의 노드들을 재귀적으로 인자로 넣어 호출하면서 하나의 예전 파일 트리의 노드를 인자로 받은 새로운 파일 트리의 노드와 비교하는 함수입니다. compare\_node\_sub 와 마찬가지로, realpath()를 통해 valid한 경로라면, 하위 서브디렉토리/파일 노드들도 비교하며, 아니라면 is\_path\_valid를 FALSE로 초기화한 후, compare\_info()를 형제 노드만 재귀적으로 호출합니다.

현재 두 파일 노드가 같은 경로를 모니터링하는 파일 노드라면 노드 구조체내에 최종 수정시각을 비교하여 초기화합니다.

3-11. get\_node() 함수의 콜 그래프

도표, 라인, 텍스트, 디자인이(가) 표시된 사진

자동 생성된 설명

처음 프로그램을 시작한 후, ssu\_monitor() 와 현재 모니터링 경로의 트리 출력을 위한 ssu\_tree()에서 read\_daemon() (“monitor\_list.txt을 읽어 디몬 프로세스들의 정보를 가지고 링크드 리스트를 만드는 함수”)를 호출 하면 push\_back\_node()를 통해 디몬 프로세스의 링크드 리스트에 노드들을 추가합니다. 노드를 생성할 때마다 디몬 프로세스가 관리하는 경로 하위 파일 트리는 node\_sub(파일 노드)로 이루어진 트리를 만들게 됩니다.

node->child는 파일 노드로 이루어진 트리를 가리키게 되며, make\_tree() 함수는 파일 트리를 만들기 위한 함수입니다. 경로로는 디몬 프로세스가 관리하는 경로가 들어가게 되고, 만약 사용자가 디몬 프로세스가 관리하는 디렉토리 자체를 삭제 할 경우 디몬 프로세스가 종료 될 수 있는데 이를 막고자, mkdir()을 통해 모니터링 경로인 디렉토리가 사라지더라도 다시 만들 도록 하였습니다. 그 후, make\_tree()를 통해 모니터링 경로의 하위 파일 및 서브 디렉토리로 이루어진 node\_sub들의 링크드 리스트 트리를 만듭니다.

4. 실행결과

4-1. ssu\_monitor 실행

해당 프로그램은 사용자가 설정한 모니터링 경로 상의 정규 파일을 모니터링 하는 것이므로,

모든 내부 명령어는 모니터링 경로 상의 있는 디렉토리와 정규 파일 만을 모니터링 합니다.

4-1-1. ssu\_monitor 실행 시, 내장 명령어 외 기타 명령어 입력

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

4-1-2. 프롬프트 상에서 엔터만 입력

20192392> 입력 대기 상태에서 엔터를 누르면 다음 20192392> 프롬프트가 출력 된 후, 새로운

입력 대기 상태가 됩니다.

텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명

4-1-3. 모니터링에 대한 결과. (/home/hanseung/test)

모니터링 테스트 도중 /home/hanseung/test 와 /home/hanseung/test/test2 는 서로 다른 디렉토리입니다.

파일 수정 상황.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

파일 생성 상황.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

파일 삭제 상황.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

디렉토리 삭제 상황(디렉토리 삭제 후, 내부 파일 한꺼번에 삭제)

파일 모니터링에 따라 test 경로 하위 서브디렉토리인 test2를 삭제함으로서,

서브디렉토리 내부 ./test2/1.c 와 ./test2/2.c가 모두 삭제된 상황

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

4-2. 내부 명령어add

monitor\_list.txt가 만들어지는 시기는, 처음 모니터링 할 경로를 설정한 후 만들어집니다.

4-2-1. 상대 경로 입력 케이스 (../../../test)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

4-2-2. 절대 경로 입력 케이스 (/home/hanseung/test2)

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

4-2-3. 첫번째 인자가 없는 경우

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

4-2-4. 첫번째 인자가 잘못된 경우(존재 하지 않는 경로)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

4-2-5. 첫번째 인자가 디렉토리가 아닌 경우

텍스트, 폰트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

4-2-6. 첫번째 인자가 이미 모니터링 중인 경로 일 경우 (상위 / 하위 / 동일 경로)

/home/hanseung/test 와 /home/hanseung/test2를 모니터링 중인 경로 일 경우 테스트 입력

상위 경로 입력

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

동일 경로 입력

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

하위 경로 입력

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

4-2-7. -t 옵션이 잘못된 경우

0 이상의 정수를 입력하지 않은 경우(예외 처리 경우이므로, usage 출력이 아닌 다른 에러 메시지 출력)

텍스트, 폰트, 스크린샷, 타이포그래피이(가) 표시된 사진

자동 생성된 설명

-t <TIME> -t 옵션의 시간 인자가 없는 경우(usage에서 제시된 사용방법이 아니므로 usage 출력)

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

4-2-8. -t 옵션을 사용 하여 add 명령어 정상적인 실행

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

4-3. 내부 명령어 delete

4-3-1. 모니터링 중인 디몬 프로세스를 제대로 delete 한 경우.

monitor\_list.txt내 디몬 프로세스 정보 삭제 및 디몬 프로세스 종료.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

4-3-2. 모니터링 되고 있지 않는 <DAEMON\_PID>일 경우(monitor\_list.txt에 없는 프로세스)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

4-3-3. 첫번째 인자 <DAEMON\_PID>의 입력이 없는 경우

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

4-4. 내부 명령어 tree

오직 monitor\_list.txt 내에 디몬 프로세스가 관리 중인 경로에 해당하여서만 tree 명령어 사용 가능.

4-4-1. 정상적으로 모니터링 중인 경로의 디렉토리 및 정규 파일 출력

(디렉토리와 정규 파일 출력) : 상대 경로

텍스트, 스크린샷, 폰트, 도표이(가) 표시된 사진

자동 생성된 설명

4-4-2. 정상적으로 모니터링 중인 경로의 디렉토리 및 정규 파일 출력

(디렉토리와 정규 파일 출력) : 절대 경로

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

4-4-3. 모니터링 되고 있지 않는 경로일 경우(monitor\_list.txt에 없는 프로세스)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

4-4-4. 만약 프롬프트가 켜져있는 상태에서 모니터링 경로의 파일에 변동이 생긴 경우

만약, 프롬프트가 켜져있는 상태에서 모니터링 경로 상의 파일에 변화가 있는 경우 ->

프롬프트에 트리 출력에도 반영 되어야한다. 모니터링 경로의 new\_file.txt 가 생겨 트리에도 반영된 모습

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

4-5. 내부 명령어 help

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

4-6. 내부 명령어 exit

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명