컴퓨터학부, 20190511, 배준형

# 과제 개요

|  |
| --- |
| ssu\_monitoring 프로그램은 디몬 프로세스를 이용하여 백그라운드로 프로세스가 실행되면서 디렉토리 폴더를 지정하면 해당 디렉토리내부의 파일들이 수정, 삭제, 생성 되었는지 확인하는 프로그램이다. 예를들어 add temp 라고 입력하면 현재경로가 /home/junhyeong 라고 헀을 때 /home/junhyeong/temp 폴더를 모니터링하여 해당 폴더 내의 a.txt를 생성하면 [시간][create][/home/junhyeong/temp/a.txt] 라고 temp 폴더 내의 log.txt 파일에 기록이 될 것이다.  지원하는 옵션으론 {add, delete, tree, exit, help} 명령어가 존재한다.   * add <DIRPATH> * add <DIRPATH> -t <TIME> * delete <DAEMON\_PID> * tree <DIRPATH> * help * exit   생성된 디몬 프로세스는 모니터링을 시작할 때, 종료할 때 출력을 해준다. 실제 구현상으론 A터미널에서 실행한 디몬프로세스를 B터미널에서 꺼도 종료 출력 메시지를 B터미널에서 받을 수 있도록 작업파일 (프로그램 실행 중 사용하는 임시작업파일 (daemon\_msg.txt) 을 FIFO 처럼 이용하여 출력하도록 구현되었다. (이렇게 설계한 이유는 상세설계에서 설명하겠습니다.)  add 옵션은 DIRPATH 경로를 디몬프로세스가 모니터링을 하도록 시키는 명령어로 -t 옵션을 하나 가질 수 있다. -t 옵션은 <TIME> (unsigned int) 시간마다 모니터링을 재개할 수 있도록 주는 옵션이다.  delete 옵션은 모니터링이 수행중인 디몬프로세스의 pid를 주면 SIGUSR1 시그널 디몬에게 보내 종료하도록 해주는 옵션이다.  tree 옵션은 monitor\_list.txt 에 있는 경로에 한정하여 해당 경로의 작업파일 계층 트리를 출력해주는 옵션으로 구현되어있다. 빠른 속도를 지원하기 위해 재귀호출을 이용하여 구현되었다.  help 옵션은 위의 옵션을 한 눈에 볼 수있도록 #P1 설계과제, #P2 설계과제 를 참고하여 구현하였다.  exit 옵션은 인자를 받던 안받던 exit를 입력하면 ssu\_monitoring프로세스가 종료되도록 설계하였다. |

# 구현기능

## 실질적 구현 내용

|  |
| --- |
| **디몬 프로세스 모니터링 과정에서의 구현기능**   * 모니터링 받을 경로를 입력받아 해당 경로에 log.txt를 생성 (해당과정은 daemon\_io 프로세스에 위임해서 파일 입출력을 함, 아래부터 서술할 파일입출력은 모두 daemon\_io 에서 대행함) 해서 모니터링 관리 * <TIME>에 따른 모니터링 간격을 조율할 수 있음. 단, <TIME> 이 음수가 오면 에러처리를 하고,<TIME> 이 0초가 온다면 DEFAULT\_TIMER 로 타이머로 설정함. (명세에 없는 내용) * 모니터링 과정을 수행하기 위해 모니터링 경로 내부 디렉토리 파일 및 디렉토리를 재귀호출 하면서 LogPathList 링크드리스트에 파일을 연결하여 관리함. * 디몬 프로세스는 기본적으로 모니터링 시점의 경로들을 링크드리스트로 관리하고 있음. * 디몬 프로세스 시작과 종료 시 시작/종료를 알리는 출력을 함 (대신 디몬은 표준출력이 안되므로, 임시파일인 **damon\_msg.txt 파일에 출력 내용을 알리고 ssu\_monitor 프로세스에서 파일이 내용이 적힐 때 까지 기다렸다가 내용을 읽고 삭제**) * 모니터링 하는 경로의 파일이 삭제/수정/생성 되는 경우 해당 경로에 log.txt 파일에 내용을 기록. 삭제와 생성 시간은 모니터링으로 생성/삭제가 확인된 시점을 출력하고, 수정시간의 경우에는 실제로 수정된 시간을 출력하도록 하였음. * ssu\_monitor 로부터 delete (SIGUSR1) 시그널을 받으면 프로세스를 종료함을 알리고 프로세스 종료.   **ssu\_monitor 프로세스에서의 옵션처리 구현기능**   * 사용자로부터 입력받는 문자열을 바탕으로 토큰을 분리해서 옵션처리를 할 수 있도록 구현 * 그냥 개행 \n 을 받는 경우 재프롬포트 출력 * exit, help 를 제외한 tree, add, delete의 경우 인자처리가 잘못된 경우 “wrong arguement” 를 출력하고 help 메시지를 띄운 후 재프롬포트 출력 * 기본적으로, 상대경로는 절대경로로 바꿔주는 기능이 있음. * **자기가 관리하는 log.txt, monitor\_list.txt, 그리고 swp 파일은 모니터링 하지 않도록 설계. (swp 파일은 파일을 수정하려고 vim을 입력해도 creat 생성이 되는 등 modify 와 역할이 겹치는 등의 지저분함 요소가 있어 제외하도록 구현)**   **Add의 경우**   * 디몬 프로세스를 생성해서 해당 경로를 모니터링 하도록 구현 * 받은 경로 <DIR\_PATH> 가 없는 경로이거나 디렉토리가 아니면 에러처리 후 재프롬포트 * -t 옵션을 받았는데 <TIME> 이 없거나, <TIME> 이 음수 혹은 0을 포함한 양의정수 (Natural Number) 이 아니면 에러메시지를 출력하고 프로세스 종료 * Add 과정을 수행할 때마다 디몬 프로세스는 monitor\_list.txt 에 자기가 모니터링할 경로와 자신의 PID 를 기록 * 만약, monitor\_list.txt에 있는 <DIR\_PATH> 나 포함된거나 포함하는 관계가 있는 경로가 입력 시 에러 처리 * 인자 개수가 맞지 않는 경우 에러처리 * **add 하고 daemon 메시지가 5초안에 오지 않으면 alarm 시그널을 보내 while loop를 종료**   **Delete 의 경우**   * 실행중인 디몬프로세스 <DAEMON\_PID> 에대하여 디몬프로세스를 종료시키기 위해 SIGUSR1 을 kill 함수를 통해 보내면 디몬프로세스가 종료하면서 종료메시지를 임시파일 (damon.txt) 에출력하고 ssu\_monitor 프로세스에서 해당 내용을 읽고 임시파일 삭제 * <DAEMON\_PID> 가 ssu\_monitor에 없는 경우 에러처리 후 재프롬포트 * delete를 통해 삭제한 DAEMON\_PID 를 적용하여 monitor\_list.txt 에 적용해줌 (연결리스트 이용 출력) * **A터미널에서 생성한 디몬을 B터미널에서 종료해도 출력메시지를 받을 수 있게 임시파일을 이용한 출력메시지 전달을 받도록 구현**   **tree의 경우**   * monitor\_list.txt 에서 모니터링하고 있는 디렉토리의 트리계층구조를 트리형태로 출력해주는 옵션 구현< * 만약, <DIRPATH> 가 monitor\_list.txt 에 없는 경우 에러 처리함 * 재귀호출 방식으로 모니터링 디렉토리 구조를 출력하도록 설계 * 인자 개수가 잘못된 경우 에러출력하고 재프롬포트 출력 * **delete하고 daemon 메시지가 5초안에 오지 않으면 alarm 시그널을 보내 while loop를 종료**   **help**   * #1, #2 설계과제와 비슷한 양식으로 옵션 사용방식을 출력   **exit**   * 인자 개수와 상관없이 exit가 처음에 오는 경우 ssu\_monitor 프로세스를 종료하도록 설계 (while loop를 빠져나오도록 설계) * **daemon\_io 프로세스에서의 구현기능** * 디몬 프로세스에서 파일 입출력을 지원해주는 프로세스이다. * argv[1] 번 인자로 온 option 에따라 자동으로 fopen인지 open인지 판단. * ex) argv[1] = “O\_CREAT | O\_WRONLY | O\_TRUNC” 라고 하면 token 함수가 자동으로 {“O\_CREAT”, “O\_WRONLY”, “O\_TRUNC} 로 분리하고 인자를 보고 fopen인지 open인지 판단해준다. * 해당 인자를 받는 중의 좌/우 공백을 자동으로 제거해준다. |

## ssu\_monitor 프로세스 프로토타입

|  |
| --- |
| [**프로토타입 및 함수 구현기능]**  typedef struct pn { //상대경로-->절대경로 이중포인터 구조체      char path[MAXPATHLEN];      struct pn \*prev;      struct pn \*next;  }pathNode;    typedef struct mt { //ssu\_montitor 파일 이중연결리스트      char path [MAXPATHLEN];      char pid [PIDSIZE];      struct mt\* next;      struct mt\* prev;  }Mnode;  typedef struct mtl{ //Mnode 연결리스트 관리 구조체      Mnode \*head;      Mnode \*tail;      int cnt;  }Mlist;  char\*\* path\_arr(char\* str); // str을 / 기준으로 토큰배열화  int realpathS (char \*str); // 링크드리스트기반 경로찾기 함수  char\*\* prom\_args (char\* str); // 프롬포트 토큰 배열 생성  char\* tokinzer (char\* str, char\* del); // del tokenizer  int path\_setting ();   // 현재경로, 디몬경로, log.txt 파일 경로를 알려주는 함수  int print\_tree (char\* path, int depth);  Mnode\* newM (char\* path, char\* pid);  Mlist\* newMlist ();  Mnode\* appendM (Mlist\* list, char\* path, char\* pid);  int delNode (Mlist\* list, char\* path);  // return 은 해당 pid  int free\_mlist (Mlist\* list); //list를 요소를 모두 free 하는 함수.  void printM (Mlist \*list); // list의 모든 요소를 출력해주는 함수.  Mlist\* read\_monitorfile(); //리스트의 있는 데이터를 moitor\_list.txt 로부터 입력  int isExistMlist (Mlist\* list, char\* path); // 해당 리스트에 path가 있는지 확인  int canAddMlist (Mlist\* list, char\* path); // 해당 리스트에 path가 같거나 포함되거나, 포함하거나 하면 오류처리  int write\_monitorfile(Mlist\* list); //리스트의 있는 데이터를 moitor\_list.txt 에 출력  int open\_monitor\_read();    //monitor\_list.txt read 오픈 전용  int open\_monitor\_write();   //monitor\_list.txt write 오픈 전용  void print\_help(); //help 출력  void daemonMessage(); //디몬프로세스로부터 전달받은 죽으면서 출력한 다잉메시지 받아보는 함수.  int directory\_check (char\* path); //딕셔너리가 실행중인 디몬 프로세스 경로와 겹치는지 확인  int do\_add(int argc, char\* args[]); // add옵션 수행  int do\_delete(int argc, char\* args[]); // delete 옵션 수행  int do\_help(int argc, char\* args[]); // help 옵션 수행  int do\_tree(int argc, char\* args[]); // tree 옵션수행  int do\_module(int option, int argc, char\* args[]); // add,delete,help,tree 옵션 수행 |

## daemon.c | daemon\_io.c 프로세스 프로토타입

|  |
| --- |
| [**프로토타입 및 함수 구현기능]**  //디몬 프로세스 구조체 (이중연결리스트 형태)  typedef struct lp {      char path [MAXPATHLEN];        time\_t mtime; // 최종 수정 시간      int visit;    // 생성 시 1, 디몬에서 monitoring 할 때 0으로 변경-->  다시확인할 때 0인지 확인!        struct lp\* prev;      struct lp\* next;  }LogPath;  typedef struct lpl {      LogPath \*head;      LogPath \*tail;      int cnt;  }LogpathList;    //\*\* 디몬 프로세스꺼  int make\_time (time\_t\* t, char\* buf); // 시간 출력  LogPath\* newL (char\* path);  LogpathList\* newList (); // list 생성자  /\* path (디렉토리만 지원) 경로의 파일 모두 재귀호출하여 Regular File에 대해서 삽입\*/  // list\_arg 리스트 NULL 이면 새로 생성해서 return해줌  LogpathList\* scandir\_list (LogpathList\* list\_arg, char\* path, int depth);  LogPath\* appendL (LogpathList \*list, char\* path);   // fl  LogPath\* pop\_path (LogpathList \*list, char\* path);  // list에서 path에 해당하는 노드 삭제  LogPath\* pop\_node (LogpathList \*list, LogPath\* node); // list에서 node 즉시 삭제  int compare\_list (LogpathList \*list, char\* path);   // 연결리스트 list 와 path를 비교하며 순회  int compare\_node (LogpathList \*list, char\* path); //list 의 있는 모든 노드와 path를 비교해주는 함수.  int check\_listvisit (LogpathList \*list);    // 리스트를 순회하며 1인지 0인지 체킹  // [시간][모드][경로] 를 print\_buf에 만들어 주는 함수  int print\_logchar (LogPath\* node, int option, char\* print\_buf);  // daemon 프로세스에서 실행되는 loop 모듈  int final\_module (LogpathList \*list, char\* path, unsigned int timer);  int printfile (int fd, char\* print\_buf); //fd에 파일 출력 함수  void printL (LogpathList \*list); // list의 모든 요소를 출력해주는 함수.  int exclusive\_swpfile (char\* path); //.swp 파일은 monitoring에서 제외하는 함수.  int log\_fd; //log file descriptor 전역변수  unsigned int time\_sleep = 1; // -t <TIME> 옵션  int daemon\_init(void); //디몬 프로세스 실행 func  //daemon process 에서 처리하는 시그널 핸들러 (SIGUSR1 전용 시그널핸들러)  void handle\_sigusr1(int signo);  void print\_ProcessMessage(); // daemon프로세스 파일입출력을 대행하는 함수  void fprint\_process(char\* opt, char\* path,  char\* str); //process 이용 입출력  char daemon\_path [MAXPATHLEN]; //현재 디몬 프로세스의 경로  char monitor\_path [MAXPATHLEN]; // monitor\_list.txt 의경로  char target\_path [MAXPATHLEN]; //모니터링 대상 경로  char log\_path[MAXPATHLEN+10]; //log.txt 경로  char daemon\_pid [PIDSIZE]; //데몬 프로세스 pid  char daemon\_msg\_path [MAXPATHLEN]; //죽으면서 다잉메시지를 남길 파일.  char child\_path [MAXPATHLEN]; //디몬이 호출하는 표준입출력 전용 프로세스 경로  char\* args[5] = {NULL, }; //입출력 프로세스에 넣을 인자 {"자식프로세스 경로", "출력 옵션", "경로", "출력 내용", NULL};  **[daemon\_io 프로세스 : 디몬 파일입출력 대행 프로세스 내부 함수]**   char\* token(char\* str, char\* del); //del 문자열 기준으로 str을 토큰 분리 (strtok 비슷)  char\* remove\_whitespace (char\* str); // 사용자로 입력받은 문자열을 return |

# 상세설계

## 모듈 구성과 동작 원리

|  |
| --- |
| **[모듈 구성 설명]**  ssu\_monitoring이 파일들을 모니터링하고 log.txt와 monitor\_list.txt에 파일입출력을 하도록 하는데 몇 가지 시행착오 끝에 다음과 같은 방식으로 구성되었습니다.  [구성 방식]   1. 모듈 구성 2. ssu\_monitoring 함수에서 옵션처리를 하되 add 옵션의 디몬프로세스 생성은 동일 디렉토리에 디몬 프로세스 프로그램을 만들어 해당 프로세스를 호출하도록 한다. (fork-->exec) 3. 디몬 프로세스가 모니터링을 하되, 디몬 프로세스가 파일 입출력은 하지 않는 것이 좋으니 파일 입출력은 **다른 프로세스**에게 대행을 맡기도록 한다. (fork-->exec); 4. 정리하면, ssu\_monitor.c 파일은 옵션처리와 디몬호출, 디몬종료 시그널 (SIGUSR1) 을 보내주는 파일입니다. 5. daemon.c 파일은 ssu\_monitor에서 디몬을 호출하면 실행되는 디몬프로세스입니다. 하지만 파일 입출력을 할 때는 아래 프로세스를 fork()+exec()하는 fprint\_process 함수를 호출합니다. 6. daemon\_io.c 파일은 디몬 프로세스에서 호출되는 또다른 프로세스로 **디몬의 파일 입출력 부분을 대행해주는 프로세스**입니다. 7. 구현 방식   <**디몬이 파일을 모니터링을 하는 방식 : visit변수가 핵심입니다.**>  우선 디몬 프로세스는 디몬 프로세스 규칙 (부모프로세스 죽이고, setsid(); 를 호출하여 init 프로세스가 부모 프로세스가 되면서 세션리더가 되도록 한 뒤 파일디스크립터 모두 닫고, /dev/null로 오픈하여 표준 입력/출력/에러 를 없애고 umask(0) 으로 만드는 등의 규칙을 모두 수행) 을 모두 마친 후 **모니터링을 시작하는 함수 scandir\_list() 함수를 호출한 뒤 final\_module()함수를 while loop를 돌게하여 시그널이 올 때까지 반복하도록 설계하였습니다.)**  디몬 프로세스는 add 호출을 받자마자 scandir\_list 함수를 호출하여 LogpathList라는 {경로, visit 변수(int), m\_time(현재 최종수정시간)} 을 멤버변수로 가지는 링크드리스트를 이용하여 모니터링할 경로를 모두 탐색하여 링크드리스트로 연결합니다. 이때 visit는 1로 초기세팅되어 연결리스트를 연결합니다.  그리고 **final\_module()** 함수를 호출하여 현재 넘겨받은 링크드리스트에서 [수정,삭제,생성] 이 일어났는지 확인을 합니다. 확인하는 과정은 아래와 같습니다.   1. compare\_list() **함수를 호출**: 다시 한 번 모니터링할 경로를 모두 탐색을 하며 LogPathList 링크드리스트에 해당 경로가 있는지 확인을 합니다. 만약 탐색을 하다가 동일명을 가진 경로가 LogPathList에 존재를 한다면 해당 Node->visit = 0; 으로 변경을 합니다. (1-->0), 하지만 동일명의 경로가 LogPathList에 없다면 이는 파일이 생성됨을 의미하므로 list에 append를 해주고 [creat] 부분을 log.txt에 출력하도록 합니다. (**디몬 프로세스가 출력하는 것은 되도록 피하는게 좋으니, fprint\_process() 함수를 호출하여 daemon\_io 프로세스가 파일 입출력을 대행합니다.)**   **>> 시간복잡도를 줄이기 위해 초기에 head위치를 기록해두었다가 각 링크드 리스트 비교과정에서 자신이 head 위치에 있는 visit를 1-->0 으로 바꾼 경우 head 위치를 앞쪽으로 옮겨 불필요한 연산을 줄이도록 설계하였습니다. (마지막에 저장해둔 head위치를 초기화 시키는 과정도 있음)**   1. check\_listvisit() **함수를 호출**: 앞의 compare\_list 가 돌면서 파일이 존재한다면 LogPathList 링크드리스트의 visit가 0으로 바뀌어있을 것입니다. 0으로 바뀐 노드에 대해서는 stat 시스템 콜을 호출하여 해당 노드의 멤버변수 path 에 대하여 st\_mtime이 동일한지 체크를 합니다. 동일하지 않다면 최종 수정시간이 변경되었음을 의미하므로 [modify] 를 출력합니다. 그리고 다음 노드로 진행하기 전에 visit 를 1로 다시 변경시켜줍니다.   만약 해당 체크 과정에서 visit 가 0으로 바뀌지 않은 노드가 있다면 해당 노드는 삭제된 노드이므로 링크드리스트에서 pop 을 하여 삭제해주고 [delete]를 출력하도록 해줍니다.  final\_module를 호출할 때마다 sleep(timer) 가 적용되어 -t 옵션으로 받은 <TIME> 만큼 프로세스가 sleep을 호출합니다. 만약 <TIME> 옵션에 음수가 온다면 예외처리를 하였고 이 점은 add 옵션 설명부에서 설명하겠습니다.  해당 while 루프를 돌며 final\_module이 <TIME> 마다 sleep를 해주도록 설계하였습니다. (기본 1초)  **<옵션처리 :ssu\_monitor 프로세스>**  옵션처리는 기본적으로 PROMPT() 매크로로 20190511> 를 띄워준 후 그냥 개행처리가 오면 재프롬포트를 띄우는 형식으로 처리합니다. 그리고 옵션들이 저장된 command\_str 전역변수배열을 활용해서 맞는 옵션을 선택하여 **do\_module()** 함수를 호출하고 함수 호출 내부에서 **do\_add/delete/ ...** 등의 함수로 재호출합니다.  그렇게 각 명령어 실행이 끝나면 다시while 루프로 돌아와 exit가 올 때 까지 무한루프를 돌도록 설정하였습니다.  **<add 옵션 : ssu\_montor 프로세스가 처리>**   1. Arguement 처리를 기본적으로 한 뒤에 -t 옵션이 있는지 없는지 선별검사를 한 후, -t 옵션의 경우 음수값이 들어왔는지 아니면 숫자가 아닌지 확인을 합니다. 2. 이후 상대경로/절대경로로 받은 make\_path를 **realpathS** 함수를 호출하여 상대경로를 절대경로로 바꿉니다. (링크드리스트 기반 상대경로->절대경로화 함수) 그리고 monitor\_list.txt 로부터 현재 실행중인 디몬프로세스가 모니터링 중인 경로와 pid를 **read\_monitorfile() 함수를 호출하여** 연결리스트로 받아온 후, 3. **입력받은 경로가 연결리스트 내부와 겹치거나,포함되거나,포함하는지 canAddMlist 함수를 호출하여** 검사를 합니다. 4. 모든 조건이 만족한다면 fork-->exec() 를통해 디몬 프로세스를 호출하고, 부모 프로세스는 디몬프로세스 부모가 죽은것이 확인되면(**디몬 생성이 확인)** 디몬 프로세스가 간접적으로 전달하는 모니터링 시작 문구를 받아옵니다.(해당 내용 아래에서 설명) 5. 그리고 생성된 디몬 프로세스는 자신의 pid와 경로를 monitoring\_txt 에 저장합니다. 6. list의 모든 요소를 할당해제하기 위해 **free\_mlist()를** 호출시켜 해제합니다.   >> 디몬 프로세스는 기본적으로 생성될 때 “monitoring started(<PID>)” 를 출력하는데 해당 부분을 임시 파일(MSG\_FILE\_NAME 매크로 (“daemon\_msg.txt”)을 통해 메시지를 주고 받을 수 있도록 하였습니다. 그리고 ssu\_monitor 프로세스가 디몬이 보낸 메시지를 읽은 후 해당 “daemon\_mst.txt” 파일을 삭제 (unlink) 하도록 설계하였습니다.   1. **TIMEOUT** 매크로에 설정되어있는 시간(**기본 5sec**) 안에 디몬프로세스로부터 메시지가 오지 않으면 종료시킵니다.   <delete 옵션>   1. 이후Arguement 개수가 맞는지 확인을 한 후 **read\_monitorfile()**함수를 호출하여 ssu\_monitor.txt에 인자로 받은 **pid**가 monitor\_list.txt 로 안에 있는지 **delNode()**를 호출하여 확인합니다. (해당과정에서 리스트에 존재하는 pid면 해당 리스트는 연결리스트에서 pop 되며, 존재하지 않는다면 false 를 리턴합니다.) 2. 이후 수정된 연결리스트를 monitor\_list.txt에 저장하기 위해 **write\_monitorfile()**를 호출합니다. 3. **TIMEOUT** 매크로에 설정되어있는 시간(**기본 5sec**) 안에 디몬프로세스로부터 메시지가 오지 않으면 종료시킵니다. 4. list의 모든 요소를 할당해제하기 위해 **free\_mlist()를** 호출시켜 해제합니다.   <tree 옵션>   1. 인자 체크를 한 후 2. 받아온 경로를 절대경로화 시키기 위해 **realpathS()** 를 호출합니다 3. 그리고 **print\_tree**() 함수를 재귀호출 하여 해당 경로를 재귀호출합니다. 4. 재귀호출이 끝난 후 list의 모든 요소를 할당해제하기 위해 **free\_mlist()를** 호출시켜 해제합니다. 5. **<아쉬운 점> : Modify와 Create 와 달리 탐색하는 경로와 트리를 비교하는 시간복잡도가 O(N2) 가 되다 보니 20,000 개정도 파일이 있는 시스템에서 삭제하는 속도가 버벅임을 조금 볼 수 있다는 아쉬운 점이 있었습니다. (--> 트리를 이용해서 비교했으면 조금 더 좋지않았을까 하는 아쉬운 점이 남습니다.) --> head 가 이미 체크된 대상이면 head를 당기는 방식으로 구현하였으나 한계가 있어보임..**   <help 옵션>   1. #1, #2 설계과제처럼 옵션 사용방식을 출력하도록 설계하였습니다.   <exit 옵션>   1. 첫 부분에 exit 토큰이 오면 ssu\_monitoring 프로그램이 종료되도록 설계하였습니다. |

## 함수간 흐름도 및 순서도

|  |
| --- |
| [모듈 전체적인 동작 원리]    [디몬 프로세스가 수행하는 함수 루틴]    [ssu\_monitor 프로세스의 함수 흐름]  [각 명령어 함수 루틴]     * add      * delete      * [tree] |

## 함수 프로토타입과 함수 소개 (타입)

|  |
| --- |
| //\*\* ssu\_monitor 프로세스꺼  char\*\* path\_arr(char\* str); // str을 / 기준으로 토큰배열화  return 성공시 토큰 포인터배열, 에러시 NULL  int realpathS (char \*str); // 링크드리스트기반 경로찾기 함수  return 성공시 1 에러시 0  char\*\* prom\_args (char\* str); // 프롬포트 토큰 배열 생성  return 성공시 토큰으로 분리한 문자열배열, 에러시 NULL  char\* tokinzer (char\* str, char\* del); // del tokenizer  return 성공시 del로 분리한 토큰 문자열포인터, 에러시 NULL  int path\_setting ();   // 현재경로, 디몬경로, log.txt 파일 경로를 알려주는 함수  return 성공시 1, 에러시 0  int print\_tree (char\* path, int depth); // 트리 출력함수  return 성공 시 1 , 에러시 0  Mnode\* newM (char\* path, char\* pid);  return 성공시 할당받은 Mnode, 에러시 NULL  Mlist\* newMlist ();  return 성공시 생성된 Mlist 포인터  Mnode\* appendM (Mlist\* list, char\* path, char\* pid);  return 성공시 append 한 Node 포인터, 에러시 NULL;  int delNode (Mlist\* list, char\* path);  // return 은 해당 pid  return 성공 시 1 , 에러시 0  int free\_mlist (Mlist\* list); //list를 요소를 모두 free 하는 함수.  return 성공 시 1 , 에러시 0  void printM (Mlist \*list); // list의 모든 요소를 출력해주는 함수.  //리스트의 있는 데이터를 moitor\_list.txt 로부터 입력  Mlist\* read\_monitorfile();  return 성공 시 monitor\_list.txt에서 읽어들인 {경로,pid} 리스트 Mlist 포인터, 에러시 NULL  int isExistMlist (Mlist\* list, char\* path); // 해당 리스트에 path가 있는지 확인  return 리스트에 존재하면 true, 존재하지 않으면 false  // 해당 리스트에 path가 같거나 포함되거나, 포함하거나 하면 오류처리  int canAddMlist (Mlist\* list, char\* path);  return 리스트에 포함되거나 포함하거나 겹치는 경로 없으면 true, add 수행불가하면 false 리턴  int write\_monitorfile(Mlist\* list); //리스트의 있는 데이터를 moitor\_list.txt 에 출력  return 성공 시 1 , 에러시 0  void print\_help(); //help 출력  int open\_monitor\_read();    //monitor\_list.txt read 오픈 전용  return 성공 시 1 , 에러시 0  int open\_monitor\_write();   //monitor\_list.txt write 오픈 전용  return 성공 시 1 , 에러시 0    int directory\_check (char\* path); //디렉토리인지 체크 + 존재하는지 체크  return path가 디렉토리면 true, 아니거나 경로에 존재하지 않으면 false return  //디몬프로세스로부터 전달받은 메시지 출력  void daemonMessage();  /\*\* 각 모듈들을 수행하는 함수들\*/  int do\_add(int argc, char\* args[]);  return 성공 시 1 , 에러시 0  int do\_delete(int argc, char\* args[]);  return 성공 시 1 , 에러시 0  int do\_help(int argc, char\* args[]);  return 성공 시 1 , 에러시 0  int do\_tree(int argc, char\* args[]);  return 성공 시 1 , 에러시 0  int do\_module(int option, int argc, char\* args[]); // main함수에서 module를 수행할 함수.  return 성공 시 1 , 에러시 0, exit 종료 신호면 10 리턴  //\*\* daemon 프로세스꺼  int make\_time (time\_t\* t, char\* buf); // 시간 출력  return 성공 시 1 , 에러시 0  LogPath\* newL (char\* path);  LogpathList\* newList (); // list 생성자  /\* path (디렉토리만 지원) 경로의 파일 모두 재귀호출하여 Regular File에 대해서 삽입\*/  // list\_arg 리스트 NULL 이면 새로 생성해서 return해줌  LogpathList\* scandir\_list (LogpathList\* list\_arg, char\* path, int depth);  LogPath\* appendL (LogpathList \*list, char\* path); // 리스트연결  LogPath\* pop\_path (LogpathList \*list, char\* path);  // list에서 path에 해당하는 노드 삭제  LogPath\* pop\_node (LogpathList \*list, LogPath\* node); // list에서 node 즉시 삭제  // 연결리스트 list 와 path를 비교하며 순회  int compare\_list (LogpathList \*list, char\* path);  return 성공 시 1 , 에러시 0  //list 의 있는 모든 노드와 path를 비교해주는 함수.  int compare\_node (LogpathList \*list, char\* path);  return list에 존재하면 true , 없으면 false 리턴  // 리스트를 순회하며 1인지 0인지 체킹  int check\_listvisit (LogpathList \*list);  return 성공 시 1 , 에러시 0  // [시간][모드][경로] 를 print\_buf에 만들어 주는 함수  int print\_logchar (LogPath\* node, int option, char\* print\_buf);  return 성공 시 1 , 에러시 0  // daemon 프로세스에서 실행되는 loop 모듈  int final\_module (LogpathList \*list, char\* path, unsigned int timer);  return 성공 시 1 , 에러시 0  int exclusive\_swpfile (char\* path); //.swp 파일은 monitoring에서 제외하는 함수.  return swp 파일이면 false return , 아니면 true리턴  int daemon\_init(void); //디몬 프로세스 실행 func  //daemon process 에서 처리하는 시그널 핸들러 (SIGUSR1 전용 시그널핸들러)  void handle\_sigusr1(int signo);  // daemon프로세스 파일입출력을 대행하는 함수  void print\_ProcessMessage();  void fprint\_process(char\* opt, char\* path,  char\* str); //process 이용 입출력  //\*\* daemon\_io 프로세스꺼  // 입력받은 문자열을 del 토큰 단위로 분리  char\* token(char\* str, char\* del);  return 성공시 분리된 토큰 lexeme 리턴, 에러시 NULL 리턴  // 사용자로 입력받은 문자열에서 좌우 공백을 없애줌.  char\* remove\_whitespace (char\* str);  return 성공 시 좌우 white space를 제거한 문자열 포인터, 에러시 NULL 리턴 |

# 실행결과

## add 옵션 // add t 옵션

텍스트, 전자제품, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 웹사이트이(가) 표시된 사진

자동 생성된 설명

## delete 옵션

* 텍스트, 스크린샷, 폰트이(가) 표시된 사진

  자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

## tree 옵션

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

## help 옵션

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

## exit 옵션

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

## makefile

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명