컴퓨터학부 20201662 이정현

목차

1. 과제 개요

리눅스 시스템 상에서 사용자가 지정한 경로에 대해 변경 사항을 추적하는 데몬 프로세스를 생성하고, 해당 데몬 프로세스를 통해 주기적으로 변경 사항을 추적하고 이를 백업 및 기록하는 ssu_sync 프로그램을 작성하였다.

2. 기능

프로그램 실행 시, 사용자 입력을 기다리는 프롬프트를 출력한다. 이 때 입력 가능한 내장 명령어는 add, remove, list, help, exit이며, 이 외의 명령어 입력 시에는 help 명령어의 결과를 출력하고 프롬프트를 재출력한다.

```
// main.c
// 사용자 입력을 기다리는 프롬프트 출력
// exit 입력 시, 프로그램 종료
while(1){
        printf("20201662 > ");
        fgets(input, sizeof(input), stdin);
        input[strlen(input) - 1] = '\0';

        // argList: 공백을 기준으로 자른 토큰들을 가리키는 포인터들를 저장한 리스트
        // argCnt: 토큰의 개수
        // 엔터만 입력한 경우: argList 는 NULL, argCnt 는 0
        if((argList = getArglist(input, &argCnt)) != NULL){
                if(!strcmp(argList[0], "exit")) break;

                // 해당하는 내장 명령어 실행
                for(i = 0; i < NUM_CMD; i++){
                        if(!strcmp(argList[0], commandList[i])){
                                builtin_command(argList, argCnt);
                                break;
                        }
                }
```

```
        // 프롬프트에서 지정한 내장명령어 외 기타 명령어를 입력한 경우: help 명령어 실행
        if(i == NUM_CMD)
            builtin_help(NULL);

        // 내장명령어, 경로 등을 저장했던 메모리 해제
        for(i = 0; i < argCnt; i++)
            free(argList[i]);
        free(argList);
    }

    memset(input, 0, sizeof(input));

    printf("\n");
}
```

1) add <PATH> [OPTION] …

백그라운드에서 입력받은 경로를 모니터링하며, 변경 내역이 있는 경우 백업 디렉토리에 해당 파일을 백업하고 변경 내역을 로그 파일에 기록하는 데몬 프로세스를 생성한다. 기본적으로 모니터링은 1초 간격으로 진행하며, 옵션 t가 입력되었을 때에는 옵션 뒤에 입력된 인자만큼의 주기로 변경 내역을 모니터링 한다.

① initPath(): 현재 홈 디렉토리, 작업 디렉토리, 각 로그파일 등의 경로를 초기화하여 전역 변수에 저장

② isExistDaemon(): monitor_list.log을 바탕으로 입력 받은 경로가 현재 추적 경로인지 확인하며, 만약 존재하는 경우에는 해당 경로를 추적하는 데몬 프로세스를 생성하지 않음

③ daemonize(): 현재 프로세스를 데몬 프로세스로 전환

④ trackFile()/trackDir(): 변경사항을 일정 주기에 맞추어 추적

```
// 홈 디렉토리, 현재 실행 디렉토리 경로 등을 초기화
    initPath();

    // 입력 받은 경로를 절대 경로(fullPath)로 변환하고, 경로에 해당하는 파일/디렉토리명(name)을 저장하고
    // 해당 파일/디렉토리가 올바른 경로인지 확인
    if(processPath(argv[1], fullPath, name) < 0 || checkPath(fullPath, name) < 0){
        fprintf(stderr, "ERROR: %s is wrong path\n", argv[1]);
        exit(1);
    }

    // 입력 받은 경로의 정보 획득
```

```
if(lstat(fullPath, &statbuf) < 0){
        fprintf(stderr, "lstat error for %s\n", argv[1]);
        exit(1);
}

// 옵션이 있는 경우: 옵션 추출 및 옵션에 따른 뒤의 인자 추출
// 옵션이 올바르지 않은 경우: add 사용법 출력 후 비정상 종료
period = 1; // default 주기: 1 초
if(getOpt(&opt, &period, argc, argv) < 0){
        fprintf(stderr, "ERROR: wrong option\nUsage: %s\n", USAGE_ADD);
        exit(1);
}

// 입력받은 경로에 맞는 올바른 옵션이 입력되었는지 확인
if(checkOpt(statbuf.st_mode, opt) < 0){
        if(S_ISREG(statbuf.st_mode))
                fprintf(stderr, "\"%s\" is file\n", fullPath);
        else if(S_ISDIR(statbuf.st_mode))
                fprintf(stderr, "\"%s\" is directory\n", fullPath);

        exit(1);
}

// monitor_list.log 을 바탕으로 입력 받은 경로가 현재 추적 경로인지 확인
// 노드가 존재하는 경우: 에러 처리
if(isExistDaemon(fullPath)){
        fprintf(stderr, "ERROR: %s is already exist\n", argv[1]);
        exit(1);
}

// 현재 프로세스를 데몬 프로세스로 전환
daemonize();

// monitor_list.log 파일에 경로 및 pid 기록
if((fp = fopen(monitorPATH, "a+")) != NULL){
        pid = getpid();

        fprintf(fp, "%d : %s\n", pid, fullPath);
        fprintf(stdout, "monitoring started (%s) : %d\n", fullPath, pid);
```

```
        fclose(fp);
    }
    else{
        fprintf(stderr, "fopen error for %s\n", monitorPATH);
        exit(1);
    }


    // 추적할 경로 내의 파일을 저장할 백업 파일 및 로그 파일 생성
    sprintf(backupDirPath, "%s/%d", backupPATH, pid);
    sprintf(backupLogPath, "%s.log", backupDirPath);

    if(access(backupDirPath, F_OK)){
        int fd;

        mkdir(backupDirPath, 0777);
        if((fd = creat(backupLogPath, 0666)) < 0){
            fprintf(stderr, "creat error for %s\n", backupLogPath);
            exit(1);
        }
        close(fd);
    }

    // 입력 받은 경로에 대해 변경사항 추적
    // 입력받은 경로가 일반 파일인 경우
    if(S_ISREG(statbuf.st_mode))
        trackFile(backupDirPath, backupLogPath, fullPath, period);
    // 입력받은 경로가 디렉토리인 경우
    else if(S_ISDIR(statbuf.st_mode))
        trackDir(backupDirPath, backupLogPath, fullPath, period, opt);
```

2) remove <DAEMON_PID>

pid를 이용해 실행 중인 데몬 프로세스를 삭제한다. 즉, 특정 경로에 대한 모니터링을 중지하고, 해당 경로와 관련된 백업 파일 및 로그 파일을 모두 삭제한다. 이 때 내장명령어 remove는 pid에 해당하는 데몬 프로세스에 SIGUSR1 시그널을 보냄으로써 데몬 프로세스를 삭제한다.

① initPath(): 현재 홈 디렉토리, 작업 디렉토리, 각 로그파일 등의 경로를 초기화하여 전역 변수에 저장

② isExistPid(): 입력 받은 pid에 해당하는 데몬 프로세스가 있는지 확인하고, 만약 존재하지 않는다면 에러 처리

③ removeDaemon(pid): pid에 해당하는 데몬 프로세스를 삭제하고, 해당 데몬 프로세스와 관련이 있는 백업 파일 및 로그를 모두 삭제

```
// 홈 디렉토리, 현재 실행 디렉토리 경로 등을 초기화
    initPath();

    // 입력받은 pid 에 해당하는 데몬 프로세스가 있는지 확인
    pid = atoi(argv[1]);
    if(!isExistPid(pid, fullPath)){
        fprintf(stderr, "ERROR: %s is wrong pid\n", argv[1]);
        exit(1);
    }

    // pid 에 해당하는 데몬 프로세스 삭제
    kill(pid, SIGUSR1);
    fprintf(stdout, "monitoring ended (%s) : %d\n", fullPath, pid);

    // pid 에 해당하는 데몬 프로세스와 관련된 로그 파일 및 디렉토리 삭제
    removeDaemon(pid);
```

3) list [DAEMON_PID]

현재 모니터링 중인 데몬 프로세스 또는 데몬 프로세스가 관리 중인 경로에 대해 트리 형태로 출력한다. 이 때 첫 번째 인자인 DAEMON_PID는 생략 가능하며, 생략 시에는 현재 실행 중인 데몬 프로세스들의 목록을 출력하고, 생략하지 않은 경우에는 해당 데몬 프로세스가 관리 중인 경로 내의 정규 파일들을 트리 형태로 출력한다.

① initPath(): 현재 홈 디렉토리, 작업 디렉토리, 각 로그파일 등의 경로를 초기화하여 전역변수에 저장

② isExistPid(): 입력 받은 pid에 해당하는 데몬 프로세스가 있는지 확인하고, 만약 존재하지 않는다면 에러 처리

```
initPath();

    // list 만 입력된 경우
    if(argc == 1){
        if((fp = fopen(monitorPATH, "rb")) != NULL){
            while(fgets(buf, BUFFER_SIZE, fp) != NULL){
                fp(stdout, "%s", buf);
            }
        }
    }
    // list [DAEMON_PID]
```

```
    else if(argc == 2){
        // 입력받은 pid 에 해당하는 데몬 프로세스가 있는지 확인
        pid = atoi(argv[1]);
        if(!isExistPid(pid, fullPath)){
            fprintf(stderr, "ERROR: %s is wrong pid\n", argv[1]);
            exit(1);
        }

        ;
    }
```

4) help [COMMAND]

ssu_repo 프로그램의 내장 명령어에 대한 설명을 출력하는 명령어이다. 명령행 인자가 없는 경우에는 모든 내장 명령어에 대한 설명을 출력하고, 인자 COMMAND가 입력된 경우에는 해당하는 내장 명령어의 설명을 출력한다. 만약 입력 받은 인자에 해당하는 내장 명령어가 존재하지 않는 경우에는 에러 처리를 한다.

```
// 모든 내장 명령어에 대한 설명을 출력하는 경우
    if(argc == 1){
        printf("Usage: \n");
        printf("    > %s\n", USAGE_ADD);
        printf("    > %s\n", USAGE_REMOVE);
        printf("    > %s\n", USAGE_LIST);
        printf("    > %s\n", USAGE_HELP);
        printf("    > %s\n", USAGE_EXIT);
    }
    // 특정 명령어에 대한 설명을 출력하는 경우
    else{
        if(!strcmp(argv[1], "add")) printf("Usage: %s\n", USAGE_ADD);
        else if(!strcmp(argv[1], "remove")) printf("Usage: %s\n", USAGE_REMOVE);
        else if(!strcmp(argv[1], "list")) printf("Usage: %s\n", USAGE_LIST);
        else if(!strcmp(argv[1], "help")) printf("Usage: %s\n", USAGE_HELP);
        else if(!strcmp(argv[1], "exit")) printf("Usage: %s\n", USAGE_EXIT);
        else printf("ERROR: help [add | remove | list | help | exit]\n");
    }
```

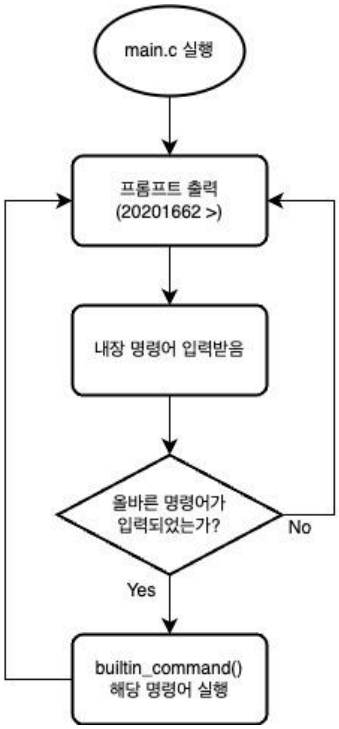
5) exit

현재 실행 중인 ssu_sycn 프로그램을 종료하는 명령어이다.

3. 상세 설계

명령어	add	remove	list	help	exit
구현	○	○	△	○	○

main.c에서 프롬프트를 출력하고 내장 명령어와 명령어에 따른 인자를 입력 받는다. 입력 받은 내장 명령어가 존재하는 내장 명령어인지 확인하고, builtin_commnad()를 통해 각 내장 명령어들을 실행시킬 프로세스를 생성하고 이를 실행한다.



builtin_commnad()의 코드는 다음과 같으며, command.c 파일에서 확인할 수 있다. 즉, 모든 내장 명령어는 아래의 코드를 통해 실행된다.

```
// 내장 명령어를 실행하는 함수
void builtin_command(char **argList, int argc)
{
    char **argv = (char **)malloc(sizeof(char *) * (argc + 1));
    int i;

    // execv()를 실행할 때, 실행할 프로그램에 전달할 인자들을 나타내는 배열의 마지막 원소는 NULL 포인터여야함
    for(i = 0; i < argc; i++)
        argv[i] = argList[i];
    argv[i] = NULL;

    // help 명령어인 경우: help 다음에 온 인자 또는 NULL 전달
    if(!strcmp(argv[0], "help"))
```

```
            builtin_help(argv[1]);
        // 그 외의 다른 명령어인 경우: 해당 명령어 프로세스 실행
        else{
            pid_t pid;

            // 내장 명령어 실행 -> 뒤의 인자만 전달 (명령어 자체는 전달하지 않음)
            if((pid = fork()) < 0){
                fprintf(stderr, "fork error for %s\n", argv[0]);
                exit(1);
            }
            else if(pid == 0){
                execv(argv[0], argv);
                exit(0);
            }
            else{
                wait(NULL);
            }

        }

        free(argv);
}
```

1) 구조체와 연결리스트

ssu_sync 프로그램에서 생성된 데몬 프로세스는 연결리스트를 통해 추적하는 경로 내의 파일 및 디렉토리를 관리하였다. 연결리스트의 노드를 위한 구조체와 연결리스트의 선언 및 연결리스트 관련 함수들은 모두 monitoring.c와 monitoring.h에서 확인할 수 있다.

- DirNode *trackList
  - ❿ 데몬 프로세스가 관리하는 경로들의 정보를 담고 있는 연결 리스트
  - ❿ 각각의 데몬 프로세스마다 연결리스트를 하나씩 관리하고 있음

- Struct DirNode: 디렉토리의 이름, 경로, 상위/하위/형제 디렉토리 노드, 하위 파일 등

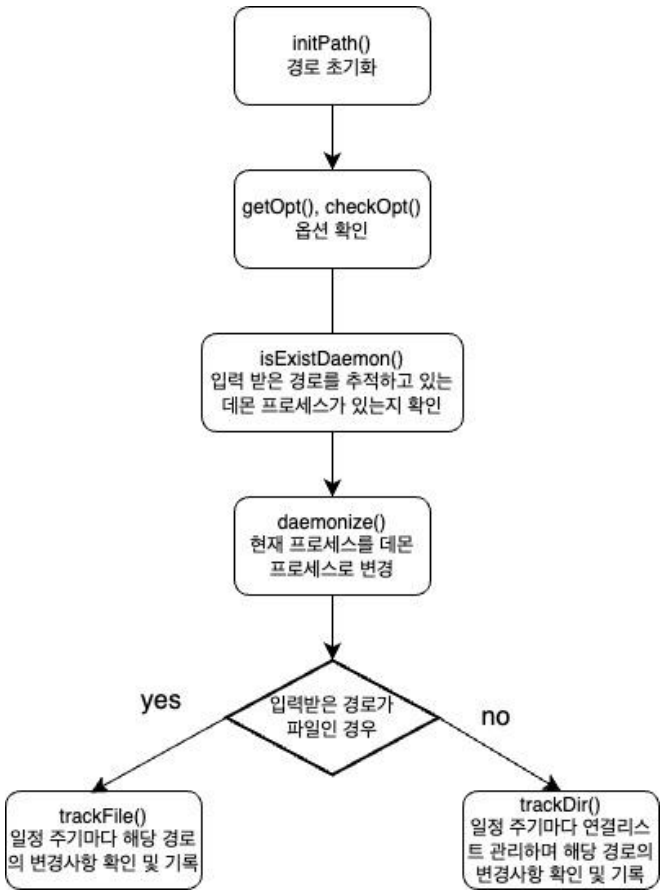
```
// 디렉토리 노드
typedef struct DirNode{
    char path[MAX_PATH + 1]; // 해당 디렉토리의 원본 경로

    struct DirNode *parentDir; // 상위 디렉토리
    struct DirNode *subDir; // 하위 디렉토리
    struct FileNode *fileHead; // 하위 파일들의 리스트
```

```
    // 형제 디렉토리
    struct DirNode *next;
    struct DirNode *prev;
}DirNode;
```

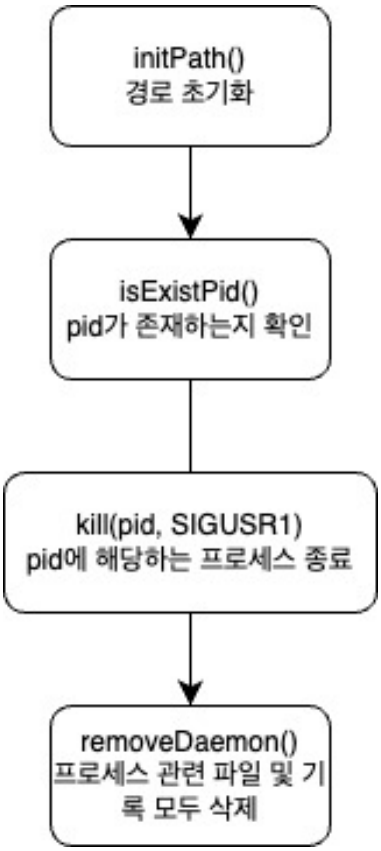
- Struct FileNode: 파일의 이름, 경로, 상위 디렉토리 노드, 형제 파일 노드 등

```
// 파일 노드
typedef struct FileNode{
    char path[MAX_PATH + 1]; // 해당 파일의 경로
    time_t mtime;

    // 해당 파일의 상위 디렉토리 노드
    struct DirNode *parentDir;

    // 형제 파일
    struct FileNode *prev;
    struct FileNode *next;
}FileNode;
```

2) 내장 명령어: add

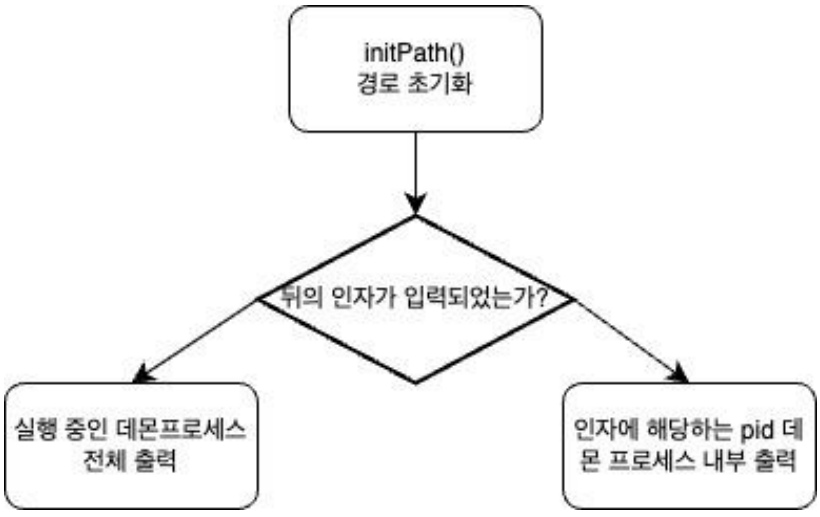


3) 내장 명령어: remove



4) 내장 명령어: list

list의 기능 중, 인자가 입력되지 않은 경우만 구현하였다.



4. 실행 결과

1) add

```
ubuntu@ip-192-168-0-101:~/p33$ ./ssu_sync
20201662 > add testdir
"/home/ubuntu/p33/testdir" is directory

20201662 > add a.txt

20201662 > monitoring started (/home/ubuntu/p33/a.txt) : 37396
exit
ubuntu@ip-192-168-0-101:~/p33$ cat /home/ubuntu/backup/monitor_list.log
37396 : /home/ubuntu/p33/a.txt
ubuntu@ip-192-168-0-101:~/p33$ vi a.txt
ubuntu@ip-192-168-0-101:~/p33$ rm a.txt
ubuntu@ip-192-168-0-101:~/p33$ cat /home/ubuntu/backup/37396.log
[2024-05-27 11:48:42][create][/home/ubuntu/p33/a.txt]
[2024-05-27 11:49:47][modify][/home/ubuntu/p33/a.txt]
[2024-05-27 11:49:54][remove][/home/ubuntu/p33/a.txt]
ubuntu@ip-192-168-0-101:~/p33$ ./ssu_sync
20201662 > add testdir -d

20201662 > monitoring started (/home/ubuntu/p33/testdir) : 37860
exit
ubuntu@ip-192-168-0-101:~/p33$ cat /home/ubuntu/backup/monitor_list.log
37396 : /home/ubuntu/p33/a.txt
37860 : /home/ubuntu/p33/testdir
ubuntu@ip-192-168-0-101:~/p33$ cat /home/ubuntu/backup/37860.log
[2024-05-27 11:52:14][create][/home/ubuntu/p33/testdir/a.txt]
[2024-05-27 11:52:14][create][/home/ubuntu/p33/testdir/b.txt]
[2024-05-27 11:52:14][create][/home/ubuntu/p33/testdir/c.txt]
ubuntu@ip-192-168-0-101:~/p33$ vi testdir/a.txt
ubuntu@ip-192-168-0-101:~/p33$ rm ./testdir/b.txt
ubuntu@ip-192-168-0-101:~/p33$ touch testdir/d.txt
ubuntu@ip-192-168-0-101:~/p33$ touch testdir/a/b/a.txt
touch: cannot touch 'testdir/a/b/a.txt': No such file or directory
ubuntu@ip-192-168-0-101:~/p33$ touch testdir/a/a/a.txt
touch: cannot touch 'testdir/a/a/a.txt': No such file or directory
ubuntu@ip-192-168-0-101:~/p33$ touch testdir/a/a.txt
ubuntu@ip-192-168-0-101:~/p33$ cat /home/ubuntu/backup/37860.log
[2024-05-27 11:52:14][create][/home/ubuntu/p33/testdir/a.txt]
[2024-05-27 11:52:14][create][/home/ubuntu/p33/testdir/b.txt]
[2024-05-27 11:52:14][create][/home/ubuntu/p33/testdir/c.txt]
[2024-05-27 11:53:03][create][/home/ubuntu/p33/testdir/.a.txt.swp]
[2024-05-27 11:53:05][modify][/home/ubuntu/p33/testdir/.a.txt.swp]
[2024-05-27 11:53:08][modify][/home/ubuntu/p33/testdir/a.txt]
[2024-05-27 11:53:08][remove][[2024-05-27 11:53:08][remove][]
```

2) remove

```
ubuntu@ip-192-168-0-101:~/p33$ cat /home/ubuntu/backup/monitor_list.log
37396 : /home/ubuntu/p33/a.txt
37860 : /home/ubuntu/p33/testdir
ubuntu@ip-192-168-0-101:~/p33$ ./ssu_sync
20201662 > remove 37396
monitoring ended (/home/ubuntu/p33/a.txt) : 37396

20201662 > ^C
ubuntu@ip-192-168-0-101:~/p33$ cat /home/ubuntu/backup/monitor_list.log
37860 : /home/ubuntu/p33/testdir
ubuntu@ip-192-168-0-101:~/p33$ cat /home/ubuntu/backup/37396.log
cat: /home/ubuntu/backup/37396.log: No such file or directory
ubuntu@ip-192-168-0-101:~/p33$
```

3) list

```
ubuntu@ip-192-168-0-101:~/p33$ ./ssu_sync
20201662 > list
37860 : /home/ubuntu/p33/testdir
```