

파일 시스템 구현

Member:

20202990 이정우

20180610 전준영

20221598 박찬우

01 / 서론

- 프로젝트 배경
- 프로젝트 목적
- 역할 분담

02 / 터미널 기반 기능

- 파일 목록 출력 및 실행 시스템
- 파일 검색 및 정렬
- 파일 및 폴더 관리 시스템

03 / 구현

- 개발 환경 설정
- 파일 탐색기 기능 구현
 - * 파일 목록 출력 및 실행 시스템
 - * 파일 검색 및 정렬
 - * 파일 및 폴더 관리 시스템

04 / 결론

- 기대효과 및 마무리

Linux terminal

```
manseul03@Ubuntu: ~/ros2
Preparing to unpack .../git_1%3a2.34.1-1ubuntu1.10_amd64.deb ...
Unpacking git (1:2.34.1-1ubuntu1.10) ...
liberror-perl (0.17029-1) 설정하는 중입니다 ...
git-man (1:2.34.1-1ubuntu1.10) 설정하는 중입니다 ...
git (1:2.34.1-1ubuntu1.10) 설정하는 중입니다 ...
Processing triggers for man-db (2.10.2-1) ...
manseul03@Ubuntu:~$ git config --global user.name manseul03
manseul03@Ubuntu:~$ git config --global user.email cksdnlab@gmail.com
manseul03@Ubuntu:~$ git config --global color.ui "auto"
manseul03@Ubuntu:~$ git clone https://github.com/ros2/ros2.git
'ros2'에 복제합니다...
remote: Enumerating objects: 1332, done.
remote: Counting objects: 100% (178/178), done.
remote: Compressing objects: 100% (125/125), done.
remote: Total 1332 (delta 105), reused 120 (delta 52), pack-reused 1154
오브젝트를 받는 중: 100% (1332/1332), 353.83 KiB | 6.43 MiB/s, 완료.
웹타를 알아내는 중: 100% (732/732), 완료.
manseul03@Ubuntu:~$ ls
Desktop  Downloads  Pictures  Templates  ros2
Documents Music      Public    Videos    snap
manseul03@Ubuntu:~$ cd ros2
manseul03@Ubuntu:~/ros2$ ls
CODEOWNERS  README.md  ros2.repos
manseul03@Ubuntu:~/ros2$
```

터미널을 통한 조작, 파일 관리

이미지 - 운영체제 과제

System call

```
l03@Ubuntu:~$ ls
p Downloads Pictures Templates fileExplorer ros2 test
nts Music      Public    Videos    miniOS      snap
l03@Ubuntu:~$ ls test
l05.team2
l03@Ubuntu:~$ cd test/ssumini05.team2
l03@Ubuntu:~/test/ssumini05.team2$ ls
le boot drivers kernel minios
.md create_test include lib scripts
l03@Ubuntu:~/test/ssumini05.team2$ cd kernel
l03@Ubuntu:~/test/ssumini05.team2/kernel$ nano kernel.c
l03@Ubuntu:~/test/ssumini05.team2/kernel$ ls
xplorer kernel.o schedule_task.h system.o
.c process.c system.c team2
l03@Ubuntu:~/test/ssumini05.team2/kernel$ cd file_explorer
l03@Ubuntu:~/test/ssumini05.team2/kernel/file_explorer$ ls
reate.c file_search.o schedule_task.c
reate.o file_write_path.c schedule_task.h
earch.c file_write_path.o schedule_task.o
l03@Ubuntu:~/test/ssumini05.team2/kernel/file_explorer$ nano file_writ
l03@Ubuntu:~/test/ssumini05.team2/kernel/file_explorer$ exit
```

직관적이지 못한 System Call

이미지 - 과장된 상황 예시

be intuitive



간단하게, 직관적으로

GUI ver.



Terminal ver.



minios



GUI와 같은 기능 제공 - 터미널의 고효율화

20202990 이정우 터미널 파일 및 폴더 관리 시스템

20180610 전준영 터미널 파일 탐색 및 정렬 시스템

20221598 박찬우 터미널 기반 파일 탐색 및 실행 시스템 구현



CreateFolder

폴더 생성
원하는 위치에 디렉토리 생성



deleteFolder

폴더 삭제
원하는 위치의 디렉토리 삭제



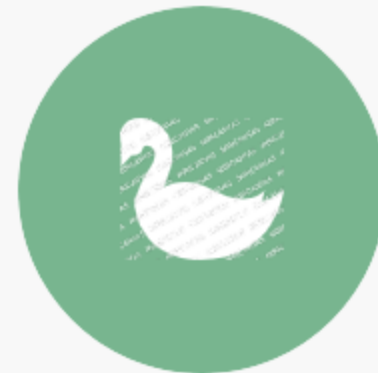
copyFile

파일 복사
복사할 파일을 원하는 위치에 복사



fileSearch

파일 검색
검색어에 맞는 파일들의 경로 표시



listFolder

목록 표시
디렉토리의 하위 목록 표시



execFile

파일 실행
터미널에서 텍스트 편집기 실행(.c, .txt 파일)



Ubuntu_22.04

→ 실행 중



minios

```

#define MAX_THREADS 3 // 최대 스레드 수

static int running_threads = 0; // 현재 실행 중인 스레드 수
static pthread_mutex_t running_threads_mutex = PTHREAD_MUTEX_INITIALIZER;

void scheduleTask(void *(*task)(void *), void *arg) {
    pthread_mutex_lock(&running_threads_mutex);
    if (running_threads < MAX_THREADS) {
        running_threads++;
        pthread_mutex_unlock(&running_threads_mutex);

        pthread_t thread;
        pthread_create(&thread, NULL, task, arg);
        pthread_detach(thread);
    } else {
        pthread_mutex_unlock(&running_threads_mutex);
        usleep(1000000); // 0.1초 대기
        printf("thread is waiting..\n");
        scheduleTask(task, arg); // 재귀적으로 호출하여 다시 시도
    }
}

```

```

// schedule_task.h
#ifndef SCHEDULE_TASK_H
#define SCHEDULE_TASK_H

void scheduleTask(void *(*task)(void *), void *arg);

#endif // SCHEDULE_TASK_H

```

03 구현 - 파일 탐색기 기능 구현 - 파일 목록 출력 및 실행 시스템

```
1 #ifndef BASIC_INCLUDE
2 #define BASIC_INCLUDE
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <string.h>
6
7 #ifdef _WIN32
8 #include <windows.h>
9 #include <direct.h>
10 #define GetCurrentDir _getcwd
11
12 #else
13 #include <unistd.h>
14 #include <dirent.h>
15 #include <sys/stat.h>
16 #include <fcntl.h>
17 #include <sys/types.h>
18 #include <sys/stat.h>
19
20 #define GetCurrentDir getcwd
21 #endif
22
23 #define NAME_MAX 255
24 #define PATH_MAX 4096
25 #define STR_MAX 4096
26 #endif
27
28 typedef struct fileLinkedList {
29     char path[PATH_MAX];
30     struct fileLinkedList* prev;
31     struct fileLinkedList* next;
32 } fileLinkedList;
33
34 static fileLinkedList* head = NULL;
35 static fileLinkedList* tail = NULL;
36
37 void addLink(char* path) {
38     fileLinkedList* rt = (fileLinkedList*)malloc(sizeof(fileLinkedList));
39     strcpy(rt->path, path);
40     rt->next = NULL;
41
42     if (head == NULL) {
43         head = rt;
44         tail = rt;
45         rt->prev = NULL;
46     }
47     else {
48         rt->prev = tail;
49         tail->next = rt;
50         tail = rt;
51     }
52 }
53
54 void printList() {
55     fileLinkedList* temp = head;
56
57     if (temp == NULL) {
58         fprintf(stdout, "file not founded \n");
59     }
60
61     while (temp != NULL) {
62         fprintf(stdout, "%s \n", temp->path);
63         temp = temp->next;
64     }
65 }
```

```
41 rt->next = NULL;
42
43 if (head == NULL) {
44     head = rt;
45     tail = rt;
46     rt->prev = NULL;
47 }
48 else {
49     rt->prev = tail;
50     tail->next = rt;
51     tail = rt;
52 }
53
54 void printList() {
55     fileLinkedList* temp = head;
56
57     if (temp == NULL) {
58         fprintf(stdout, "file not founded \n");
59     }
60
61     while (temp != NULL) {
62         fprintf(stdout, "%s \n", temp->path);
63         temp = temp->next;
64     }
65
66     // Clear the linked list after printing
67     while (head != NULL) {
68         temp = head;
69         head = head->next;
70         free(temp);
71     }
72     tail = NULL;
73 }
74
75 void filesearch(char* path, char* target) {
76     char filename[NAME_MAX];
77     char pathbuf[PATH_MAX];
78     char pathbuf2[PATH_MAX];
79
80     if (path == NULL && target == NULL) {
81         fprintf(stdout, "input filename: ");
82         fgets(filename, NAME_MAX, stdin);
83         char* temp = strchr(filename, '\n');
84         if (temp) *temp = 0;
85     }
86
87 #ifdef _WIN32
88     strcpy(pathbuf, getenv("USERPROFILE"));
89     _chdir(getenv("USERPROFILE"));
90
91 #else
92     strcpy(pathbuf, getenv("HOME"));
93     chdir(getenv("HOME"));
94 #endif
95
96     head = NULL;
97     tail = NULL;
98
99     else {
100         strcpy(pathbuf, path);
101         strcpy(filename, target);
102     }
103
104 #ifdef _WIN32
105     WIN32_FIND_DATA findFileData;
106     HANDLE hFind = INVALID_HANDLE_VALUE;
```

```
108
109 // Prepare directory search pattern
110 char searchPattern[PATH_MAX];
111 sprintf(searchPattern, "%s/*", pathbuf);
112 hFind = FindFirstFile(searchPattern, &findFileData);
113
114 do {
115     if (strcmp(findFileData.cFileName, ".") == 0 || strcmp(findFileData.cFileName, "..") == 0) {
116         continue;
117     }
118     sprintf(pathbuf2, "%s/%s", pathbuf, findFileData.cFileName);
119     if (findFileData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) {
120         filesearch(pathbuf2, filename);
121     }
122     else {
123         if (strcmp(findFileData.cFileName, filename) == 0) {
124             addLink(pathbuf2);
125         }
126     }
127 } while (FindNextFile(hFind, &findFileData) != 0);
128 FindClose(hFind);
129
130 #else
131 struct dirent** list;
132 struct stat statbuf;
133 int cnt = scandir(pathbuf, &list, NULL, alphasort);
134
135 for (int i = 0; i < cnt; i++) {
136     if (strcmp(list[i]->d_name, ".") == 0 || strcmp(list[i]->d_name, "..") == 0) {
137         free(list[i]);
138         continue;
139     }
140
141     sprintf(pathbuf2, "%s/%s", pathbuf, list[i]->d_name);
142     if (lstat(pathbuf2, &statbuf) != 0) {
143         fprintf(stderr, "lstat err \n");
144         exit(1);
145     }
146     if (S_ISREG(statbuf.st_mode)) {
147         if (strcmp(list[i]->d_name, filename) == 0) {
148             addLink(pathbuf2);
149         }
150     }
151     else if (S_ISDIR(statbuf.st_mode)) {
152         filesearch(pathbuf2, filename);
153     }
154     free(list[i]);
155 }
156 free(list);
157
158 #endif
159
160 if (path == NULL && target == NULL) {
161     printList();
162 }
163
164 #ifdef _WIN32
165 int main(int argc, char* argv)
166 {
167     filesearch(NULL, NULL);
168 }
169 #endif
```



```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <pthread.h>
5 #include <unistd.h>
6 #include <errno.h>
7 #include "schedule_task.h"
8
9 #ifdef _WIN32
10 #include <direct.h>
11 #define mkdir _mkdir
12 #define rmdir _rmdir
13 #else
14 #include <sys/types.h>
15 #include <sys/stat.h>
16 #endif
17
18 typedef struct {
19     char *folderPath;
20     char *folderName;
21 } FolderArgs;
22
23 // 폴더 생성 함수
24 void *createFolderTask(void *arg) {
25     //printf("thread created\n"); //debug-log
26     FolderArgs *args = (FolderArgs *)arg;
27     char fullPath[1024];
28
29     snprintf(fullPath, sizeof(fullPath), "%s/%s", args->folderPath, args->folderName);
30
31     #ifdef _WIN32
32     if (mkdir(fullPath) == 0) {
33         printf("폴더 생성 성공: %s\n", fullPath);
34         printf(">");
35     } else {
36         perror("폴더 생성 실패");
37         printf(">");
38     }
39     #else
40     if (mkdir(fullPath, 0777) == 0) {
41         printf("폴더 생성 성공: %s\n", fullPath);
42         printf(">");
43     } else {
44         perror("폴더 생성 실패");
45         printf(">");
46     }
47     #endif
48
49     free(args->folderPath);
50     free(args->folderName);
51     free(args);
52     //printf("thread exit\n"); //debug-log
53     return NULL;
54 }
55
56 // createFolder 함수
57 void createFolder() {
58     char folderPath[1024];
59     printf("생성하고자 하는 폴더 경로를 입력하세요: ");
60     fgets(folderPath, sizeof(folderPath), stdin);
61     folderPath[strcspn(folderPath, "\n")] = '\0'; // 개행 문자 제거

```

```

63     char folderName[256];
64     printf("생성하고자 하는 폴더 명을 입력하세요: ");
65     fgets(folderName, sizeof(folderName), stdin);
66     folderName[strcspn(folderName, "\n")] = '\0'; // 개행 문자 제거
67
68     // FolderArgs 구조체에 정보 저장
69     FolderArgs *args = (FolderArgs *)malloc(sizeof(FolderArgs));
70     args->folderPath = strdup(folderPath);
71     args->folderName = strdup(folderName);
72
73     // scheduleTask 함수를 사용하여 스레드 생성 및 createFolderTask 실행
74     scheduleTask(createFolderTask, (void *)args);
75 }
76
77 // 폴더 삭제 함수
78 void *deleteFolderTask(void *arg) {
79     //printf("thread created\n"); //debug-log
80     char *folderPath = (char *)arg;
81
82     #ifdef _WIN32
83     if (rmdir(folderPath) == 0) {
84         printf("폴더 삭제 성공: %s\n", folderPath);
85         printf(">");
86     } else {
87         perror("폴더 삭제 실패");
88         printf(">");
89     }
90     #else
91     if (rmdir(folderPath) == 0) {
92         printf("폴더 삭제 성공: %s\n", folderPath);
93         printf(">");
94     } else {
95         perror("폴더 삭제 실패");
96         printf(">");
97     }
98     #endif
99
100     free(folderPath);
101     //printf("thread exit\n"); //debug-log
102     return NULL;
103 }
104
105 void deleteFolder() {
106     char folderPath[1024];
107     printf("삭제하고자 하는 폴더 경로를 입력하세요: ");
108     fgets(folderPath, sizeof(folderPath), stdin);
109     folderPath[strcspn(folderPath, "\n")] = '\0'; // 개행 문자 제거
110
111     char *folderPathCopy = strdup(folderPath);
112     scheduleTask(deleteFolderTask, (void *)folderPathCopy);
113 }
114
115 // 파일 복사 함수
116 void *copyFileTask(void *arg) {
117     //printf("thread created\n"); //debug-log
118     char **fileNames = (char **)arg;
119     char *sourceFileName = fileNames[0];
120     char *destinationFileName = fileNames[1];
121     FILE *sourceFile, *destinationFile;
122     char ch;

```

```

123     char ch;
124
125     sourceFile = fopen(sourceFileName, "r");
126     if (sourceFile == NULL) {
127         perror("원본 파일을 열 수 없음");
128         free(sourceFileName);
129         free(destinationFileName);
130         free(fileNames);
131         return NULL;
132     }
133
134     destinationFile = fopen(destinationFileName, "w");
135     if (destinationFile == NULL) {
136         perror("대상 파일을 생성할 수 없음");
137         fclose(sourceFile);
138         free(destinationFileName);
139         free(fileNames);
140         return NULL;
141     }
142
143     while ((ch = fgetc(sourceFile)) != EOF) {
144         fputc(ch, destinationFile);
145     }
146
147     printf("파일 복사 성공: %s -> %s\n", sourceFileName, destinationFileName);
148
149     fclose(sourceFile);
150     fclose(destinationFile);
151
152     free(sourceFileName);
153     free(destinationFileName);
154     free(fileNames);
155
156     //printf("thread exit\n"); //debug-log
157     return NULL;
158 }
159
160 void copyFile() {
161     char sourceFileName[1024];
162     printf("복사하고자 하는 파일 명을 전체 경로로 입력하세요: ");
163     fgets(sourceFileName, sizeof(sourceFileName), stdin);
164     sourceFileName[strcspn(sourceFileName, "\n")] = '\0'; // 개행 문자 제거
165
166     char destinationFileName[1024];
167     printf("복사하려는 위치를 전체 경로로 입력하세요: ");
168     fgets(destinationFileName, sizeof(destinationFileName), stdin);
169     destinationFileName[strcspn(destinationFileName, "\n")] = '\0'; // 개행 문자
170     제거
171
172     char **fileNames = (char **)malloc(2 * sizeof(char *));
173     fileNames[0] = strdup(sourceFileName);
174     fileNames[1] = strdup(destinationFileName);
175
176     scheduleTask(copyFileTask, (void *)fileNames);
177 }

```

```

34     char *path_copy = strdup(path);
35     char *file_name = basename(path_copy);
36     char *extension = strrchr(file_name, '.');
37     if (extension && (strcmp(extension, ".txt") == 0 || strcmp(extension, ".c") ==
38 0)) {
39         execl("/bin/cat", "cat", path, NULL);
40     } else {
41         execl(path, file_name, NULL);
42     }
43     perror("execl");
44     free(path_copy);
45     exit(EXIT_FAILURE);
46 } else if (pid < 0) {
47     perror("fork");
48 } else {
49     waitpid(pid, NULL, 0);
50 }
51 }
52
53 void file_explorer() {
54     char command[256], path[256];
55     while (1) {
56         printf("Enter command (list <directory>, exec <file>, or exit): ");
57         if (!fgets(command, sizeof(command), stdin)) break;
58         command[strcspn(command, "\n")] = 0;
59         if (strcmp(command, "exit") == 0) {
60             printf("Exiting file explorer.\n");
61             break;
62         } else if (sscanf(command, "list %s", path) == 1) {
63             list_directory(path);
64         } else if (sscanf(command, "exec %s", path) == 1) {
65             execute_file(path);
66         } else {
67             printf("Invalid command. Use 'list <directory>', 'exec <file>', or 'exi
68 t'.\n");
69         }
70     }
71 }
72
73 int main() {
74     file_explorer();
75     return 0;
76 }
77

```

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <dirent.h>
5  #include <sys/stat.h>
6  #include <unistd.h>
7  #include <libgen.h>
8  #include <sys/wait.h>
9
10 void list_directory(const char *path) {
11     DIR *dp = opendir(path);
12     if (!dp) {
13         perror("opendir");
14         return;
15     }
16
17     struct dirent *entry;
18     while ((entry = readdir(dp))) {
19         printf("%s\n", entry->d_name);
20     }
21     closedir(dp);
22 }
23
24
25 void execute_file(const char *path) {
26     struct stat sb;
27     if (stat(path, &sb) != 0) {
28         printf("File does not exist or is not executable\n");
29         return;
30     }
31
32     pid_t pid = fork();
33     if (pid == 0) {

```

결론

Thank You

발표자: 이정우
PPT: 박찬우

