파일 시스템 구현

Member:
20202990 이정우
20180610 전준영
20221598 박찬우

Thanks to 미리 캔버스

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

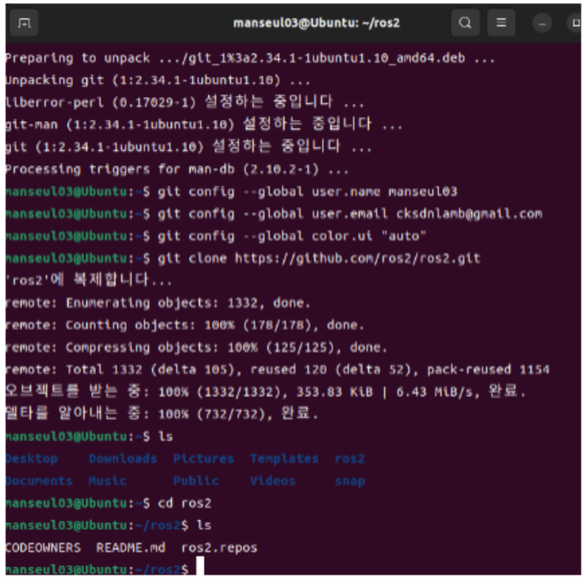
03 / 구현

- 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

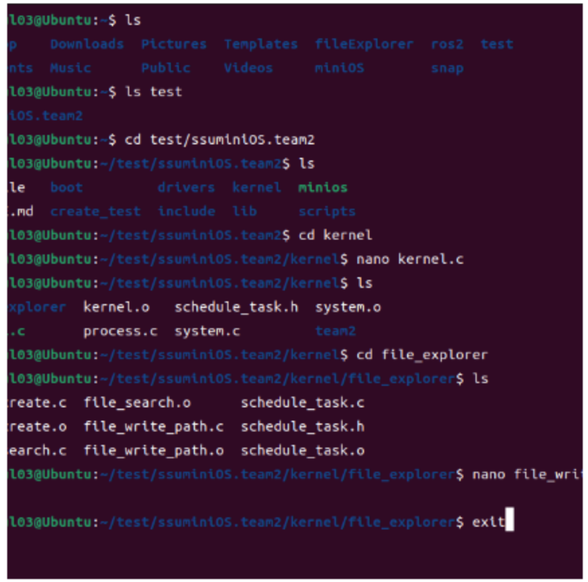
Linux terminal



터미널을 통한 조작, 파일 관리

이미지 - 운영체제 과제

System call



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)

minios

```c
#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
    }
}
```

```c
// 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

```c
#ifndef BASIC_INCLUDE
#define BASIC_INCLUDE
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifdef _WIN32
#include <windows.h>
#include <direct.h>
#define GetCurrentDir _getcwd

#else
#include <unistd.h>
#include <dirent.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

#define GetCurrentDir getcwd
#endif

#define NAMEMAX 255
#define PATHMAX 4096
#define STRMAX 4096
#endif

typedef struct fileLinkedList {
    char path[PATHMAX];
    struct fileLinkedList* prev;
    struct fileLinkedList* next;

} fileLinkedList;
static fileLinkedList* head = NULL;
static fileLinkedList* tail = NULL;

void addLink(char* path) {

    fileLinkedList* rt = (fileLinkedList*)malloc(sizeof(fileLinkedList));
    strcpy(rt->path, path);
    rt->next = NULL;

    if (head == NULL) {
        head = rt;
        tail = rt;
        rt->prev = NULL;
    }
    else {
        rt->prev = tail;
        tail->next = rt;
        tail = rt;
    }
}

void printList() {
    fileLinkedList* temp = head;

    if (temp == NULL) {
        fprintf(stdout, "file not founded \n");
    }

    while (temp != NULL) {
```

```c
    rt->next = NULL;

    if (head == NULL) {
        head = rt;
        tail = rt;
        rt->prev = NULL;
    }
    else {
        rt->prev = tail;
        tail->next = rt;
        tail = rt;
    }
}

void printList() {
    fileLinkedList* temp = head;

    if (temp == NULL) {
        fprintf(stdout, "file not founded \n");
    }

    while (temp != NULL) {
        fprintf(stdout, "%s \n", temp->path);
        temp = temp->next;
    }

    // Clear the linked list after printing
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
    tail = NULL;
}

void filesearch(char* path, char* target) {
    char filename[NAMEMAX];
    char pathbuf[PATHMAX];
    char pathbuf2[PATHMAX];

    if (path == NULL && target == NULL) {
        fprintf(stdout, "input filename: ");
        fgets(filename, NAMEMAX, stdin);
        char* temp = strchr(filename, '\n');
        if (temp) *temp = 0;

#ifdef _WIN32
        strcpy(pathbuf, getenv("USERPROFILE"));
        _chdir(getenv("USERPROFILE"));

#else
        strcpy(pathbuf, getenv("HOME"));
        chdir(getenv("HOME"));

#endif
        head = NULL;
        tail = NULL;
    }

    else {
        strcpy(pathbuf, path);
        strcpy(filename, target);
    }

#ifdef _WIN32
    WIN32_FIND_DATA findFileData;
    HANDLE hFind = INVALID_HANDLE_VALUE;
```

```c

    // Prepare directory search pattern
    char searchPattern[PATHMAX];
    sprintf(searchPattern, "%s/*", pathbuf);
    hFind = FindFirstFile(searchPattern, &findFileData);

    do {
        if (strcmp(findFileData.cFileName, ".") == 0 || strcmp(findFileData.cFileName, "..") == 0) {
            continue;
        }
        sprintf(pathbuf2, "%s/%s", pathbuf, findFileData.cFileName);
        if (findFileData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) {
            filesearch(pathbuf2, filename);
        }
        else {
            if (strcmp(findFileData.cFileName, filename) == 0) {
                addLink(pathbuf2);
            }
        }
    }
    while (FindNextFile(hFind, &findFileData) != 0);
        FindClose(hFind);

#else
    struct dirent** list;
    struct stat statbuf;
    int cnt = scandir(pathbuf, &list, NULL, alphasort);

    for (int i = 0; i < cnt; i++) {
        if (strcmp(list[i]->d_name, ".") == 0 || strcmp(list[i]->d_name, "..") == 0) {
            free(list[i]);
            continue;
        }

        sprintf(pathbuf2, "%s/%s", pathbuf, list[i]->d_name);
        if (lstat(pathbuf2, &statbuf) != 0) {
            fprintf(stderr, "lstat err \n");
            exit(1);
        }
        if (S_ISREG(statbuf.st_mode)) {
            if (strcmp(list[i]->d_name, filename) == 0) {
                addLink(pathbuf2);
            }
        }
        else if (S_ISDIR(statbuf.st_mode)) {
            filesearch(pathbuf2, filename);
        }
        free(list[i]);
    }
    free(list);

#endif
    if (path == NULL && target == NULL) {
        printList();
    }
}

#ifdef _WIN32
int main(int argc,char* argv)
{
    filesearch(NULL, NULL);
}

#endif
```

03 구현 - 파일 탐색기 기능 구현 - 파일 검색 및 정렬

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <unistd.h>
#include <errno.h>
#include "schedule_task.h"

#ifdef _WIN32
#include <direct.h>
#define mkdir _mkdir
#define rmdir _rmdir
#else
#include <sys/types.h>
#include <sys/stat.h>
#endif

typedef struct {
    char *folderPath;
    char *folderName;
} FolderArgs;

// 폴더 생성 함수
void *createFolderTask(void *arg) {
    //printf("thread created\n");                //debug-log
    FolderArgs *args = (FolderArgs *)arg;
    char fullPath[1024];

    snprintf(fullPath, sizeof(fullPath), "%s/%s", args->folderPath, args->folderN
ame);
    #ifdef _WIN32
    if (mkdir(fullPath) == 0) {
        printf("폴더 생성 성공: %s\n", fullPath);
        printf(">");
    } else {
        perror("폴더 생성 실패");
        printf(">");
    }
    #else
    if (mkdir(fullPath, 0777) == 0) {
        printf("폴더 생성 성공: %s\n", fullPath);
        printf(">");
    } else {
        perror("폴더 생성 실패");
        printf(">");
    }
    #endif

    free(args->folderPath);
    free(args->folderName);
    free(args);
    //printf("thread exit\n");                //debug-log
    return NULL;
}

// createFolder 함수
void createFolder() {
    char folderPath[1024];
    printf("생성하고자 하는 폴더 경로를 입력하세요: ");
    fgets(folderPath, sizeof(folderPath), stdin);
    folderPath[strcspn(folderPath, "\n")] = '\0';  // 개행 문자 제거
```

```c
    char folderName[256];
    printf("생성하고자 하는 폴더 명을 입력하세요: ");
    fgets(folderName, sizeof(folderName), stdin);
    folderName[strcspn(folderName, "\n")] = '\0';  // 개행 문자 제거

    // FolderArgs 구조체에 동적 저장
    FolderArgs *args = (FolderArgs *)malloc(sizeof(FolderArgs));
    args->folderPath = strdup(folderPath);
    args->folderName = strdup(folderName);

    // scheduleTask 함수를 사용하여 스레드 생성 및 createFolderTask 실행
    scheduleTask(createFolderTask, (void *)args);
}

// 폴더 삭제 함수
void *deleteFolderTask(void *arg) {
    //printf("thread created\n");                //debug-log
    char *folderPath = (char *)arg;

    #ifdef _WIN32
    if (rmdir(folderPath) == 0) {
        printf("폴더 삭제 성공: %s\n", folderPath);
        printf(">");
    } else {
        perror("폴더 삭제 실패");
        printf(">");
    }
    #else
    if (rmdir(folderPath) == 0) {
        printf("폴더 삭제 성공: %s\n", folderPath);
        printf(">");
    } else {
        perror("폴더 삭제 실패");
        printf(">");
    }
    #endif

    free(folderPath);
    //printf("thread exit\n");                //debug-log
    return NULL;
}

void deleteFolder() {
    char folderPath[1024];
    printf("삭제하고자 하는 폴더 경로를 입력하세요: ");
    fgets(folderPath, sizeof(folderPath), stdin);
    folderPath[strcspn(folderPath, "\n")] = '\0';  // 개행 문자 제거

    char *folderPathCopy = strdup(folderPath);
    scheduleTask(deleteFolderTask, (void *)folderPathCopy);
}

// 파일 복사 함수
void *copyFileTask(void *arg) {
    //printf("thread created\n");                //debug-log
    char **fileNames = (char **)arg;
    char *sourceFileName = fileNames[0];
    char *destinationFileName = fileNames[1];
    FILE *sourceFile, *destinationFile;
    char ch;
```

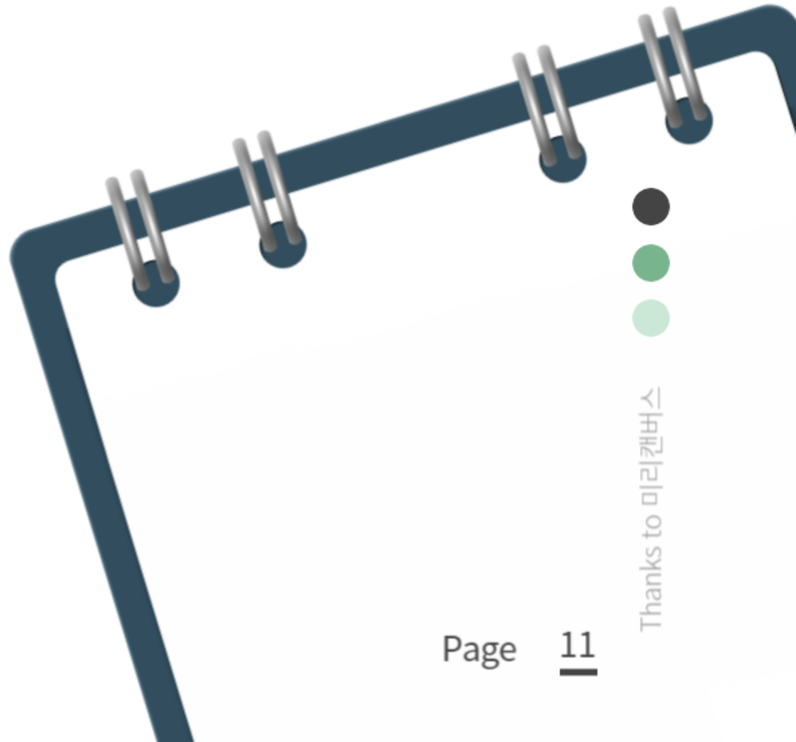
```c
    sourceFile = fopen(sourceFileName, "r");
    if (sourceFile == NULL) {
        perror("원본 그림을 열 수 없음");
        free(sourceFileName);
        free(destinationFileName);
        free(fileNames);
        return NULL;
    }

    destinationFile = fopen(destinationFileName, "w");
    if (destinationFile == NULL) {
        perror("대상 그림을 생성할 수 없음");
        fclose(sourceFile);
        free(sourceFileName);
        free(destinationFileName);
        free(fileNames);
        return NULL;
    }

    while ((ch = fgetc(sourceFile)) != EOF) {
        fputc(ch, destinationFile);
    }

    printf("파일 복사 성공: %s -> %s\n", sourceFileName, destinationFileName);

    fclose(sourceFile);
    fclose(destinationFile);

    free(sourceFileName);
    free(destinationFileName);
    free(fileNames);

    //printf("thread exit\n");                //debug-log
    return NULL;
}

void copyFile() {
    char sourceFileName[1024];
    printf("복사하고자 하는 파일 명을 전체 경로로 입력하세요: ");
    fgets(sourceFileName, sizeof(sourceFileName), stdin);
    sourceFileName[strcspn(sourceFileName, "\n")] = '\0';  // 개행 문자 제거

    char destinationFileName[1024];
    printf("복사하려는 위치를 전체 경로로 입력하세요: ");
    fgets(destinationFileName, sizeof(destinationFileName), stdin);
    destinationFileName[strcspn(destinationFileName, "\n")] = '\0';  // 개행 문자
제거

    char **fileNames = (char **)malloc(2 * sizeof(char *));
    fileNames[0] = strdup(sourceFileName);
    fileNames[1] = strdup(destinationFileName);

    scheduleTask(copyFileTask, (void *)fileNames);
}
```

03 구현 - 파일 탐색기 기능 구현 - 파일 및 폴더 관리 시스템

```
        char *path_copy = strdup(path);
        char *file_name = basename(path_copy);
        char *extension = strrchr(file_name, '.');
        if (extension && (strcmp(extension, ".txt") == 0 || strcmp(extension, ".c") ==
0)) {
            execl("/bin/cat", "cat", path, NULL);
        } else {
            execl(path, file_name, NULL);
        }

        perror("execl");
        free(path_copy);
        exit(EXIT_FAILURE);
    } else if (pid < 0) {
        perror("fork");
    } else {
        waitpid(pid, NULL, 0);
    }
}

void file_explorer() {
    char command[256], path[256];
    while (1) {
        printf("Enter command (list <directory>, exec <file>, or exit): ");
        if (!fgets(command, sizeof(command), stdin)) break;
        command[strcspn(command, "\n")] = 0;
        if (strcmp(command, "exit") == 0) {
            printf("Exiting file explorer.\n");
            break;
        } else if (sscanf(command, "list %s", path) == 1) {
            list_directory(path);
        } else if (sscanf(command, "exec %s", path) == 1) {
            execute_file(path);
        } else {
            printf("Invalid command. Use 'list <directory>', 'exec <file>', or 'exi
t'.\n");}
    }
}

int main() {
    file_explorer();
    return 0;
}
```

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

void list_directory(const char *path) {
    DIR *dp = opendir(path);
    if (!dp) {
        perror("opendir");
        return;
    }

    struct dirent *entry;
    while ((entry = readdir(dp))) {
        printf("%s\n", entry->d_name);
    }

    closedir(dp);
}

void execute_file(const char *path) {
    struct stat sb;
    if (stat(path, &sb) != 0) {
        printf("File does not exist or is not executable\n");
        return;
    }

    pid_t pid = fork();
    if (pid == 0) {
```

결론



Thank You

발표자: 이정우
PPT: 박찬우

Thanks to 미리캔버스