컴퓨터학부 20211407 김지호

1. 개요

본 과제에서는 ext2 포맷으로 생성된 디스크 이미지 파일을 직접 읽어 들여, 파일 시스템의 내부 메타데이터를 파싱하고, 루트 디렉토리로부터 시작해 옵션에 따라 모든 하위 디렉토리와 파일을 순회, 탐색하는 쉘 프로그램 'ssu_ext2'를 구현한다.

2. 기능 및 프로토타입

2-1. main.c

2-1-1. 기능 개요

main.c는 프로그램의 진입점으로, 사용자로부터 전달된 이미지 파일 경로를 확인하고 파일 시스템을 초기화한 뒤, 명령어 인터프리터를 실행하여 사용자 입력을 처리한다.

2-1-2. 프로토타입별 기능

(1) int init_ext2_structures(const char *disk_image);
- 이미지 파일(disk_image)을 읽기 전용으로 열어 파일 디스크립터(fs_fd)를 할당한다.
- 슈퍼블록을 1024바이트 오프셋에서 읽어 magic 번호를 검증하고, 블록 크기(block_size) 및 아이노드 크기(inode_size)를 계산한다.
- 그룹 디스크립터를 s_first_data_block + 1 블록 오프셋에서 읽어 이노드 테이블 위치를 초기화한다.
- 정상 처리 시 0을, 실패 시 -1을 반환한다.
(2) int main(int argc, char *argv[]);
- 커맨드라인 인자 갯수를 검사하여 <EXT2_IMAGE> 인자가 없으면 사용법 오류 메시지를 출력하고 종료한다.
- init_ext2_structures() 호출로 EXT2 구조체 초기화; 실패 시 종료한다.
- cmd_loop()를 호출하여 사용자 명령 인터프리터로 진입한다.
- 프로그램 종료 전 이미지 파일 디스크립터를 close()로 해제하고 정상 종료한다.

2-2. command.c

2-2-1. 기능 개요

command.c는 사용자로부터 명령어를 반복적으로 입력받아, 공백 및 개행을 제거한 뒤 토큰화(tokenize)하고, 첫 번째 토큰에 따라 적절한 내장 명령(tree, print, help, exit)으로 분기(dispatch)하여 호출하는 인터프리터 루프를 구현한다. 빈 입력 줄은 무시한다.

2-2-2. 프로토타입별 기능

(1) void cmd_loop(void);
- 프롬프트 출력: printf("20211407> ") 후 fflush(stdout)로 즉시 플러시한다.
- 입력 수신 : fgets()로 한 줄을 읽고, EOF 또는 NULL일 경우 루프를 탈출 및 종료 준비한다.
- 빈 줄 처리: 개행 문자를 제거한 뒤, 길이가0인 줄은 다시 프롬프트로 돌아간다.
- 토큰화: strtok()을 이용해 공백(,)과 개행( )을 구분자로 각 단어를 argv 배열에 저장하고, 마지막에NULL을 붙인다.
- 명령 분기: strcmp()로 argv[0] 값을 비교하여 tree,print,help 명령어를 실행한다. exit일 경우 루프를 종료한다.

2-3. help.c

2-3-1. 기능 개요

help.c는 help 명령어를 구현하는 모듈로, 사용자가 요청하는 명령 또는 전체 명령어 집합의 사용법(Usage)을 출력하는 역할을 한다. 잘못된 인자를 주면 단순히 "invalid command" 메시지를 출력한다.

2-3-2. 프로토타입별 기능

(1) void cmd_help(char *arg);
- arg == NULL이면 all_help()를 호출하여 tree, print, help, exit 네 가지 명령어의 사용법을 모두 출력한다.
- arg가 "tree", "print", "help", "exit" 중 하나면 해당 명령 전용 *_help()를 호출한다.
- 그 외 문자열이면 printf("invalid command -- '%s' ", arg)를 실행한다.\
(2) static void all_help(void);
- tree_help(), print_help(), help_help(), exit_help()를 순차 호출하여 전체 사용법을 출력한다.
(3) static void tree_help(void);
- tree 명령의 기본 사용법 및 -r, -s, -p 옵션 설명을 출력한다.
(4) static void print_help(void);
- print 명령의 기본 사용법 및 -n 옵션 설명을 출력한다.
(5) static void help_help(void);
- help 명령의 사용법을 출력한다.
(6) static void exit_help(void);
- exit 명령의 사용법을 출력한다.

2-4. tree.c

2-4-1. 기능개요

tree.c는 지정된 경로부터 시작해 ext2 이미지 파일 내부를 디렉토리 트리 형태로 탐색하고, 사용자가 선택한 옵션(-r, -s, -p)에 따라 트리를 출력하는 모듈이다. 탐색 중 수집한 각 엔트리를 링크드 리스트로 관리하고, 출력 후에는 디렉토리, 파일 개수를 요약해 보여준다.

2-4-2. 구조체

(1) TreeNode

- TreeNode는 디렉토리 트리 탐색 결과를 일시적으로 저장하고 순차적으로 처리하기 위한 컨테이너 역할을 한다.

(2) DirEntry

- build_tree() 내부에서 잠시 디스크에서 읽은 디렉토리 엔트리들을 보관하기 위해 사용한다.

2-4-3. 프로토타입별 기능

(1) void cmd_tree(int argc, char *argv[]);

- 사용법 검사 후 옵션을 파싱(-r, -s, -p).
- get_inode_by_path()로 루트 아이노드를 획득하고 디렉토리 여부를 확인한다.
- build_tree()와 print_nodes()를 차례로 호출해 트리를 출력하고,
- 리스트를 순회하며 디렉토리·파일 개수를 집계,출력한다.

(2) static void build_tree(const char *path, int depth, int recursive);

- 주어진 경로의 디렉토리 아이노드를 읽어 각 블록의 디렉토리 엔트리를 파싱한다.
- ".", "..", "lost+found"를 제외한 엔트리를DirEntry 배열에 복사하고, add_node()로 링크드 리스트에 추가, recursive가 참이- 면 하위 디렉토리를 재귀 탐색한다.

(3) static void print_nodes(int show_size, int show_perm);

- 리스트의 노드를 순회하면서 깊이별로 `|`, `└──`, `├──` 기호를 이용해 트리 구조를 그린다.
- show_size/show_perm 옵션에 따라 한 대괄호([]) 안에 권한 문자열과 크기를 함께 출력한다.

(4) static void add_node(const char *name, int depth, int is_last, ext2_inode *inode);

- 주어진 이름과 아이노드 정보를 가진 새 TreeNode를 동적 할당하여 리스트의 뒤에 추가한다.

(5) static void clear_tree_list(void);

- 리스트 전체를 순회하며 메모리를 해제하고, list_head·list_tail을 NULL로 초기화한다.

(6) static int get_inode_by_path(const char *path, ext2_inode *inode);

- "." 또는 "/"는 이노드 2(루트)로 처리한다.
- 그 외 경로는strtok()로 경로 컴포넌트를 분리하여 각 디렉토리 블록에서 일치하는 엔트리를 찾아 아이노드 번호를 반환한다.

(7) static int read_block(int blk, void *buf);

- blk * block_size 오프셋으로 이동한 뒤read()로 한 블록을 읽어buf에 저장한다.

(8) static int read_inode(int ino, ext2_inode *buf);

- 그룹 디스크립터의bg_inode_table * block_size 오프셋에서(ino-1)*inode_size만큼 오프셋을 더해 이노드를 읽어온다.

(9) static int id_dir(ext2_inode *inode);

- inode->i_mode의 디렉토리 비트(EXT2_S_IFDIR)를 검사하여 디렉토리 여부를 반환한다.

(10) static void format_permissions(uint16_t mode, char *buf);

- mode 비트맵을 읽어 "drwxr-xr-x"형태의 문자열을 생성한다.

(11) static void format_size(uint32_t size, char *buf);

- 바이트 단위 크기를 10진수 문자열로 포맷한다.

2-5. print.c

2-5-1. 기능 개요

print.c는 지정된 파일 경로의 이노드를 찾아 ext2 이미지에서 직접 데이터를 읽어와, cat과 유사하게 파일 내용을 출력하는 모듈이다. -n 옵션으로 원하는 줄 수만큼만 출력할 수 있다.

2-5-2. 프로토타입별 기능

(1) void cmd_print(int argc, char *argv[]);

- 파일 경로 인자와 -n <line> 옵션을 파싱하고, 이노드를 조회 후 정규 파일 여부를 확인하여 블록 단위로 내용을 출력한다.

(2) static int read_block(int blk, void *buf);

- 이미지에서 주어진 블록 번호의 내용을 lseek()과 read()로 buf에 읽어들인다.

(3) static int read_inode(int ino, ext2_inode *buf);

- 그룹 디스크립터의 bg_inode_table 위치와 inode_size를 이용해, (ino-1)번째 이노드를 버퍼에 로드한다.

(4) static int get_inode_by_path(const char *path, ext2_inode *inode);

- . 혹은 /는 루트 이노드(2)로 처리하며, 그 외 경로는 디렉토리 블록을 순회해 하위 엔트리를 찾아 이노드를 반환한다.

(5) static int is_file(ext2_inode *inode);

- S_ISDIR(i_mode) 매크로를 사용해 디렉토리가 아니라면 정규 파일로 간주한다.

(6) static void help(void);

- 잘못된 사용 시 Usage : print <PATH> [OPTION]... 메시지를 출력한다.

(7) static void read_indirect(uint32_t blk, int level, uint32_t **out, int *count, int *cap);

- 2중, 3중 간접 인덱싱도 지원하도록 탐색하는 함수

2-6. header.h

2-6-1. 기능개요

header.h는 프로젝트 전반에서 사용되는 매크로 상수, ext2 on-disk 구조체 정의, 전역 변수 선언, 그리고 각 모듈의 함수 프로토타입을 제공하는 헤더 파일이다. 이를 통해 코드 전반에서 공통으로 활용되는 타입과 인터페이스를 일관되게 관리한다.

2-6-2. 매크로

- EXT2_NAME_LEN : 디렉토리/파일 이름 최대 길이
- EXT2_N_BLOCKS : 아이노드당 블록 포인터 개수
- EXT2_FT_DIR : 디렉토리 파일 타입 코드
- EXT2_S_IFDIR : i_mode 필드에서 디렉토리 비트
- MAX_PATH : 경로 버퍼 최대 크기
- MAX_FILE : 파일 이름 버퍼 최대 크기
- MAX_LINE : 입력 라인 버퍼 크기
- MAX_ARG : 명령어 인자 최대 개수

2-6-3. 구조체

(1) ext2_super_block

- ext2 슈퍼블록(on-disk) 구조체로, 파일 시스템 전체 크기·이노드 수·블록 크기·magic 번호 등 핵심 메타데이터를 포함한다.

(2) ext2_group_desc

- ext2 그룹 디스크립터(on-disk) 구조체로, 블록 비트맵·이노드 비트맵·이노드 테이블 시작 블록 번호를 저장한다.

(3) ext2_inode

- ext2 아이노드 구조체로, 파일 타입·권한·크기·타임스탬프·블록 포인터 등을 포함해 파일/디렉토리의 메타데이터를 정의다.

(4) ext2_dir_entry_2

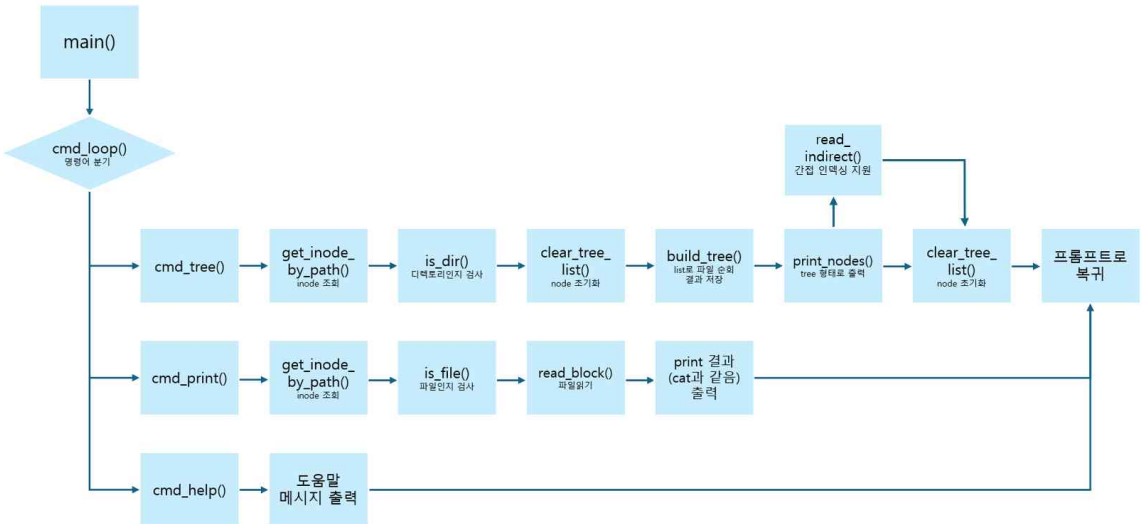
- ext2 디렉토리 엔트리 구조체로, 각 디렉토리 내 항목의 이노드 번호·이름 길이·파일 타입·이름 문자열을 저장한다.

2-6-4 전역변수

- extern int fs_fd; : 디스크 이미지 파일 디스크립터
- extern ext2_super_block sb; : 슈퍼블록
- extern ext2_group_desc gd; : 그룹 디스크립터
- extern int block_size; : 블록 크기(바이트 단위)
- extern int inode_size; : 아이노드 크기(바이트 단위)

3. 상세설계

3-1. flow chart



3-2. 전체 flow

3-2-1 프로그램 실행

(1) main 진입 후 init_ext2_structures() 호출

(2) 슈퍼블록, 디스크립터 읽기 및 block_size, inode_size 초기화.

(3) 모든 과정 성공시 cmd_loop()로 이동

3-2-2 사용자 입력 대기 : cmd_loop()

(1) 프롬프트 출력 후 대기

(2) gets()로 입력 읽음. NULL이면 재시작

(3) strtok로 argv[], argc 구성

3-2-3 명령어 입력 및 분기

- tree 이면 cmd_tree(), print 이면 cmd_print(), help이면 cmd_help(), exit이면 break.

3-2-4 tree 실행 : cmd_tree()

(1) 경로 및 옵션 파싱 및 사용법 검증

(2) get_inode_by_path()로 아이노드 조회

(3) is_dir() 확인. 실패 시 사용법 출력

(4) clear_tree_list()로 이전 결과 초기화

(5) printf(path) 로 루트 출력

(6) build_tree()로 디렉토리 스캔 및 리스트 구축

(7) print_nodes()로 트리형태 출력

(8) 리스트 순회하여 디렉토리, 파일 개수 알아낸 후 출력

(9) clear_tree_list()로 메모리 해제

3-2-5 print 실행 : cmd_print()

(1) 경로 및 -n <lines> 옵션 파싱

(2) get_inode_by_path()로 inode 조회

(3) is_file() 확인, 실 패시 에러 출력

(4) for루프로 i_block[] 순휘, read_block() -> write(), putchar()로 내용 출력

(5) 필요하다면 read_indirect()를 통해 간접 인덱싱 방식으로 큰 파일을 읽어서 출력

(6) -n 옵션일 경우 지정 줄 수만큼 출력 후 조기 종료

3-2-6 프로그램 종료

- exit 명령어 또는 EOF 입력 시 cmd_loop() 탈출 후 main()에서 close(fs_fd) 후 return

4. 실행결과

4-1. ssu_ext2

```
kimjiho@kimjiho:~/hw_p3$ ./ssu_ext2 /home/kimjiho/hw_p3/ext2disk.img
20211407> exit
kimjiho@kimjiho:~/hw_p3$ ./ssu_ext2 ext2disk.img
20211407> exit
```

- 상대경로와 절대경로 모두 실행 가능

```
20211407>
20211407> asdf
  > tree <PATH> [OPTION]... : display the diractory structure if <PATH> is a directory
    -r : display the directory structure recursively if <PATH> is a directory
    -s : display the directory structure if <PATH> is a directory, including the size of each file
    -p : display the directory structure if <PATH> is a directory, including the permissions of each directory and file
  > print <PATH> [OPTION]... : print the contents on the standard output if <PATH> is file
    -n <line_number> : print only the first <line_number> lines of its contents on the standard output if <PATH> is file
  > help [COMMAND] : show commands for program
  > exit : exit program
```

- 등록되지 않은 명령어 입력시 Usage 출력 후 프롬프트 재출력

```
kimjiho@kimjiho:~/hw_p3$ ./ssu_ext2
Usage Error : ./ssu_ext2 <EXT2_IMAGE>
```

- 인자 없을경우 Error 출력 후 종료

4-2. tree

```
20211407> tree .
.
├── A
├── B
├── ssu_text.txt
└── C

4 directories, 1 files

20211407> tree A
A
└── hello.c

1 directories, 1 files
```

- 옵션 없을 때 정상 출력

```
20211407> tree . -r
.
├── A
│   └── hello.c
├── B
│   ├── BB
│   └── nope.py
├── ssu_text.txt
└── C
    ├── CC
    │   └── 1111.txt
    ├── 1.txt
    ├── 2.txt
    └── 3.cc

6 directories, 7 files
```

```
20211407> tree . -s
.
├── [4096] A
├── [4096] B
├── [87] ssu_text.txt
└── [4096] C

4 directories, 1 files
```

```
20211407> tree . -p
.
├── [drwxr-xr-x] A
├── [drwxr-xr-x] B
├── [-rw-r--r--] ssu_text.txt
└── [drwxr-xr-x] C
```

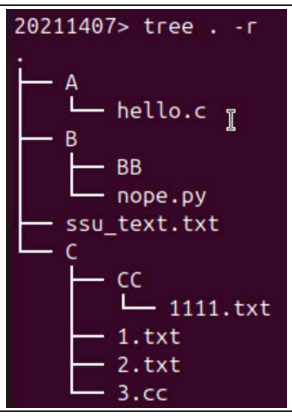
```
20211407> tree . -r -s -p
.
├── [drwxr-xr-x 4096] A
│   └── [-rw-r--r-- 149] hello.c
├── [drwxr-xr-x 4096] B
│   ├── [drwxr-xr-x 4096] BB
│   └── [-rw-r--r-- 29] nope.py
├── [-rw-r--r-- 87] ssu_text.txt
└── [drwxr-xr-x 4096] C
    ├── [drwxr-xr-x 4096] CC
    │   └── [-rw-r--r-- 0] 1111.txt
    ├── [-rw-r--r-- 0] 1.txt
    ├── [-rw-r--r-- 0] 2.txt
    └── [-rw-r--r-- 0] 3.cc

6 directories, 7 files
```

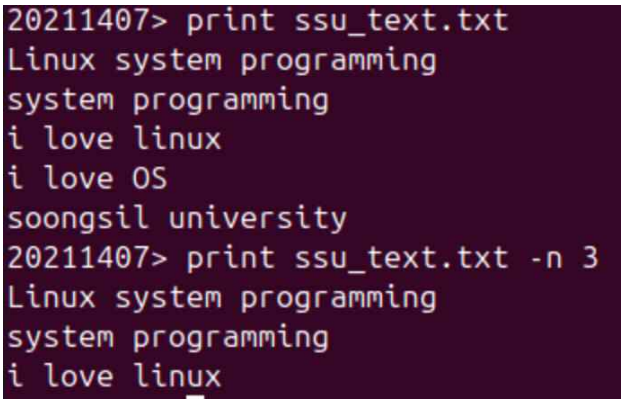
```
20211407> tree . -rsp
.
├── [drwxr-xr-x 4096] A
│   └── [-rw-r--r-- 149] hello.c
├── [drwxr-xr-x 4096] B
│   ├── [drwxr-xr-x 4096] BB
│   └── [-rw-r--r-- 29] nope.py
├── [-rw-r--r-- 87] ssu_text.txt
└── [drwxr-xr-x 4096] C
    ├── [drwxr-xr-x 4096] CC
    │   └── [-rw-r--r-- 0] 1111.txt
    ├── [-rw-r--r-- 0] 1.txt
    ├── [-rw-r--r-- 0] 2.txt
    └── [-rw-r--r-- 0] 3.cc

6 directories, 7 files
```

- r, s, p 옵션 실행 시 정상 출력
- 옵션 여러개인 경우 정상 작동

```
20211407> tree Z
Usage : tree <PATH> [OPTION]...
20211407> tree ssu_text.txt
Error: 'ssu_text.txt' is not directory
20211407> tree . -a
Usage : tree <PATH> [OPTION]...
```

- 올바르지 않은 경로, 올바르지 않은 옵션이면 Usage 출력 후 프롬프트 재출력
- 디렉토리가 아닐경우 error 출력 후 프롬프트 재출력





4-3. print

- (좌) 현재 폴더 상황, print 실행시 정상 출력
- 옵션으로 -n <num> 주면 num 줄만큼 출력

```
20211407> print Z
Usage : print <PATH> [OPTION]...
20211407> print A
Error: 'A' is not file
20211407> print ssu_text.txt -a
Usage : print <PATH> [OPTION]...
20211407>
```

- 잘못된 경로, 잘못된 옵션일 경우 Usage 출력 후 프롬프트 재출력
- file 아닐경우 error 출력 후 프롬프트 재출력

```
20211407> print ssu_text.txt -n
print: option requires an argument -- 'n'
```

- 옵션으로 <line_num> 없을 시 에러 출력

4-4. help

```
20211407> help
  > tree <PATH> [OPTION]... : display the diractory structure if <PATH> is a directory
    -r : display the directory structure recursively if <PATH> is a directory
    -s : display the directory structure if <PATH> is a directory, including the size of each file
    -p : display the directory structure if <PATH> is a directory, including the permissions of each directory and file
  > print <PATH> [OPTION]... : print the contents on the standard output if <PATH> is file
    -n <line_number> : print only the first <line_number> lines of its contents on the standard output if <PATH> is file
  > help [COMMAND] : show commands for program
  > exit : exit program
20211407> help tree
  > tree <PATH> [OPTION]... : display the diractory structure if <PATH> is a directory
    -r : display the directory structure recursively if <PATH> is a directory
    -s : display the directory structure if <PATH> is a directory, including the size of each file
    -p : display the directory structure if <PATH> is a directory, including the permissions of each directory and file
20211407> help print
  > print <PATH> [OPTION]... : print the contents on the standard output if <PATH> is file
    -n <line_number> : print only the first <line_number> lines of its contents on the standard output if <PATH> is file
20211407> help exit
  > exit : exit program
```

- help, help <command> 모두 정상출력

```
20211407> help a
invalid command -- 'asd'
  > tree <PATH> [OPTION]... : display the diractory structure if <PATH> is a directory
    -r : display the directory structure recursively if <PATH> is a directory
    -s : display the directory structure if <PATH> is a directory, including the size of each file
    -p : display the directory structure if <PATH> is a directory, including the permissions of each directory and file
  > print <PATH> [OPTION]... : print the contents on the standard output if <PATH> is file
    -n <line_number> : print only the first <line_number> lines of its contents on the standard output if <PATH> is file
  > help [COMMAND] : show commands for program
  > exit : exit program
```

- 등록되지 않은 명령어 물어볼 시 에러메세지 출력

4-5. exit

```
kimjiho@kimjiho:~/hw_p3$ ./ssu_ext2 ext2disk.img
20211407> exit
kimjiho@kimjiho:~/hw_p3$
```

- 정상종료

5. 소스코드

5-1. main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdint.h>
#include "header.h"

// Global variables defined in header.h
int fs_fd;
struct ext2_super_block sb;
struct ext2_group_desc gd;
int block_size;
int inode_size;

// init_ext2_structures: open image, read superblock and group descriptor
int init_ext2_structures(const char *disk_image) {
    // Open disk image read-only
    fs_fd = open(disk_image, O_RDONLY);
    if (fs_fd < 0) {
        perror("open");
        return -1;
    }

    // Read superblock at offset 1024
    if (lseek(fs_fd, 1024, SEEK_SET) < 0) {
        perror("lseek superblock");
        close(fs_fd);
        return -1;
    }
    if (read(fs_fd, &sb, sizeof(sb)) != sizeof(sb)) {
        perror("read superblock");
```

```
        close(fs_fd);
        return -1;
    }

    // Verify EXT2 magic number
    if (sb.s_magic != 0xEF53) {
        fprintf(stderr, "magic number failed to initialize ext2\n");
        close(fs_fd);
        return -1;
    }

    // Compute block size and inode size
    block_size = 1024 << sb.s_log_block_size;
    inode_size = sb.s_inode_size;

    // Read first group descriptor
        off_t gd_offset = (sb.s_first_data_block + 1) * block_size;
    if (lseek(fs_fd, gd_offset, SEEK_SET) < 0) {
        perror("lseek group_desc");
        close(fs_fd);
        return -1;
    }
    if (read(fs_fd, &gd, sizeof(gd)) != sizeof(gd)) {
        perror("read group_desc");
        close(fs_fd);
        return -1;
    }

    return 0;
}

// main: Program entry point
int main(int argc, char *argv[]) {
    // Validate command-line usage
    if (argc < 2) {
        fprintf(stderr, "Usage Error : %s <EXT2_IMAGE>\n", argv[0]);
        return EXIT_FAILURE;
    }

    // Initialize EXT2 structures
    if (init_ext2_structures(argv[1]) < 0) {
        return EXIT_FAILURE;
    }

    // Enter command loop
    cmd_loop();

    // Clean up: close the filesystem image file descriptor
```

```
    close(fs_fd);
    return EXIT_SUCCESS;
}
```

5-2. command.c

```
#include <stdio.h>
#include <string.h>

#include "header.h"

void cmd_loop(void){
    char line[MAX_LINE];
    char *argv[MAX_ARG];
    char *token;
    int argc;

    while(1){
        // prompt
        printf("20211407> ");
                fflush(stdout);

        // input
        if(fgets(line, sizeof(line), stdin) == NULL) break;

        // continue : blank line
        line[strcspn(line,"\n")] = '\0';
                if(line[0] == '\0') continue;

        // tokenize
        argc = 0;
        token = strtok(line, " \t\n");
        while(token && argc < MAX_ARG){
            argv[argc++] = token;
            token = strtok(NULL, " \t\n");
        }
        argv[argc] = NULL;

        // call control function
                if(strcmp(argv[0], "tree") == 0){
            cmd_tree(argc, argv);
        }else if(strcmp(argv[0], "print") == 0){
            cmd_print(argc, argv);
        }else if(strcmp(argv[0], "help") == 0){
            cmd_help((argc > 1) ? argv[1] : NULL);
        }else if(strcmp(argv[0], "exit") == 0){
            break;
        }else{
            cmd_help(NULL);
        }
```

```
    }
}
```

5-3. tree.c

```
#include <stdio.h>
#include "header.h"

/* -- Prototypes -- */
static void add_node(const char *name, int depth, int is_last, ext2_inode *inode);
static void clear_tree_list(void);
static void build_tree(const char *path, int depth, int recursive);
static void print_nodes(int show_size, int show_perm);
static int is_dir(ext2_inode *inode);
static int read_block(int blk, void *buf);
static int read_inode(int ino, ext2_inode *buf);
static int get_inode_by_path(const char *path, ext2_inode *inode);
static void format_permissions(uint16_t mode, char *buf);
static void format_size(uint32_t size, char *buf);
static void help(void);

/* -- Tree node -- */
typedef struct TreeNode {
    char *name;
    int depth;
    int is_last;
    ext2_inode inode;
    struct TreeNode *next;
} TreeNode;

static TreeNode *list_head = NULL;
static TreeNode *list_tail = NULL;

// add_node: append a node to the linked list
static void add_node(const char *name, int depth, int is_last, ext2_inode *inode) {
    TreeNode *node = malloc(sizeof(TreeNode));
    if (!node) { perror("malloc"); exit(EXIT_FAILURE); }
    node->name = strdup(name);
    node->depth = depth;
    node->is_last = is_last;
    node->inode = *inode;
    node->next = NULL;
    if (!list_head) list_head = node;
    else list_tail->next = node;
    list_tail = node;
}

// clear_tree_list: free all nodes
static void clear_tree_list(void) {
    TreeNode *cur = list_head;
```

```
    while (cur) {
        TreeNode *next = cur->next;
        free(cur->name);
        free(cur);
        cur = next;
    }
    list_head = list_tail = NULL;
}

// build_tree: scan directory entries and recurse
static void build_tree(const char *path, int depth, int recursive) {
    ext2_inode dir_inode;
    // Retrieve inode and verify directory
    if (get_inode_by_path(path, &dir_inode) < 0 || !is_dir(&dir_inode))
        return;

    int blocks = (dir_inode.i_size + block_size - 1) / block_size;
    char *buf = malloc(block_size);
    if (!buf) { perror("malloc"); return; }

    // Collect directory entries except '.' '..' and 'lost+found'
    typedef struct { uint32_t ino; char name[EXT2_NAME_LEN+1]; } DirEntry;
    DirEntry *entries = NULL;
    int count = 0, cap = 0;

    for (int i = 0; i < blocks; i++) {
        uint32_t blk = dir_inode.i_block[i];
        if (!blk) continue;
        if (read_block(blk, buf) < 0) continue;
        int off = 0;
        while (off < block_size) {
            ext2_dir_entry_2 *e = (ext2_dir_entry_2*)(buf + off);
            if (e->inode) {
                char name[EXT2_NAME_LEN+1] = {0};
                memcpy(name, e->name, e->name_len);
                // Skip special entries
                if (strcmp(name, ".") && strcmp(name, "..") && strcmp(name, "lost+found")) {
                    if (count >= cap) {
                        cap = cap ? cap * 2 : 8;
                        entries = realloc(entries, cap * sizeof *entries);
                        if (!entries) { perror("realloc"); exit(EXIT_FAILURE); }
                    }
                    entries[count].ino = e->inode;
                    strncpy(entries[count].name, name, EXT2_NAME_LEN);
                    count++;
                }
            }
            off += e->rec_len;
```

```
        }
    }
    free(buf);
    // Add each entry to node list and recurse if needed
    for (int i = 0; i < count; i++) {
        ext2_inode child;
        read_inode(entries[i].ino, &child);
        int last = (i == count - 1);
        add_node(entries[i].name, depth, last, &child);
        if (recursive && is_dir(&child)) {
            char subpath[1024];
            snprintf(subpath, sizeof(subpath), "%s/%s", path, entries[i].name);
            build_tree(subpath, depth + 1, recursive);
        }
    }

    free(entries);
}

// print_nodes: Iterate the node list and print a tree-like layout.
static void print_nodes(int show_size, int show_perm) {
    int last_flags[MAX_ARG] = {0};

    for (TreeNode *cur = list_head; cur; cur = cur->next) {
        // Print tree branches based on depth
        for (int level = 0; level < cur->depth; level++) {
            if (last_flags[level])
                printf("      ");
            else
                printf("|     ");
        }
        // Print the branch symbol
        printf(cur->is_last ? "└──── " : "├──── ");

        // Print permissions and size if requested
        if (show_perm || show_size) {
            printf("[");
            if (show_perm) {
                char p[16];
                format_permissions(cur->inode.i_mode, p);
                printf("%s", p);
            }
            if (show_perm && show_size) printf(" ");
            if (show_size) {
                char s[16];
                format_size(cur->inode.i_size, s);
                printf("%s", s);
            }
```

```
            printf("] ");
        }

        // Print the entry name
        printf("%s\n", cur->name);

        // Mark this level as last for indentation logic
        last_flags[cur->depth] = cur->is_last;
    }
}

// print usage
static void help() {
        printf("Usage : tree <PATH> [OPTION]...\n");
}

// is_dir: check inode mode for directory
static int is_dir(ext2_inode *inode) {
    return (inode->i_mode & EXT2_S_IFDIR) == EXT2_S_IFDIR;
}

// read_block: read a block from disk image
static int read_block(int blk, void *buf) {
    off_t off = (off_t)blk * block_size;
    if (lseek(fs_fd, off, SEEK_SET) < 0) return -1;
    return read(fs_fd, buf, block_size) == block_size ? 0 : -1;
}

// read_inode: read inode from table
static int read_inode(int ino, ext2_inode *buf) {
    off_t base = (off_t)gd.bg_inode_table * block_size;
    off_t off = base + (ino - 1) * inode_size;
    if (lseek(fs_fd, off, SEEK_SET) < 0) return -1;
    return read(fs_fd, buf, inode_size) == inode_size ? 0 : -1;
}

// get_inode_by_path: resolve path to inode
static int get_inode_by_path(const char *path, ext2_inode *inode) {
    // Handle root or current directory
        if (strcmp(path, ".") == 0 || strcmp(path, "/") == 0) {
                ext2_inode root;
        if (read_inode(2, &root) < 0) return -1;
        *inode = root;
        return 2;
    }

        char *copy = strdup(path);
    if (!copy) return -1;
```

```
    int cur_ino = 2;
    ext2_inode cur;
    if (read_inode(cur_ino, &cur) < 0) { free(copy); return -1; }
    char *tok = strtok(copy, "/");
    // Traverse each component
    while (tok) {
        if (!S_ISDIR(cur.i_mode)) break;
        int blocks = (cur.i_size + block_size - 1) / block_size;
        char *buf = malloc(block_size);
        int found = 0;
        for (int i = 0; i < blocks && !found; i++) {
            uint32_t blk = cur.i_block[i];
                        if (!blk || read_block(blk, buf) < 0) continue;
            int off = 0;
            while (off < block_size) {
                ext2_dir_entry_2 *e = (ext2_dir_entry_2*)(buf + off);
                if (e->inode) {
                    char name[EXT2_NAME_LEN+1] = {0};
                    memcpy(name, e->name, e->name_len);
                    if (strcmp(name, tok) == 0) {
                        cur_ino = e->inode;
                        read_inode(cur_ino, &cur);
                        found = 1;
                        break;
                    }
                }
                off += e->rec_len;
            }
        }
        free(buf);
        if (!found) break;
        tok = strtok(NULL, "/");
    }
    free(copy);
    if (tok) return -1;
    *inode = cur;
    return cur_ino;
}

// format_permissions: build permission string
static void format_permissions(uint16_t mode, char *buf) {
    buf[0] = S_ISDIR(mode) ? 'd' : '-';
    const char *perm = "rwxrwxrwx";
    for (int i = 0; i < 9; i++) buf[i+1] = (mode & (1 << (8 - i))) ? perm[i] : '-';
    buf[10] = '\0';
}

// format_size: file size in bytes
```

```
static void format_size(uint32_t size, char *buf) {
    sprintf(buf, "%u", size);
}

// cmd_tree: entry point for tree command
void cmd_tree(int argc, char *argv[]) {
    int recursive = 0, show_size = 0, show_perm = 0;

    // Usage if no path
    if (argc < 2) { help(); return; }
    const char *path = argv[1];
    // parse options
    for (int i = 2; i < argc; i++) {
        if (argv[i][0] != '-' || argv[i][1] == '\0') { help(); return; }
        for (char *p = &argv[i][1]; *p; p++) {
            if (*p == 'r') recursive = 1;
            else if (*p == 's') show_size = 1;
            else if (*p == 'p') show_perm = 1;
            else { help(); return; }
        }
    }
    // clear old list
    clear_tree_list();
    // validate path
    ext2_inode root;
    if (get_inode_by_path(path, &root) < 0) { help(); return; }
    if (!is_dir(&root)) { fprintf(stderr, "Error: '%s' is not directory\n", path); return; }

    // print and recurse
    printf("%s\n", path);
    build_tree(path, 0, recursive);
        print_nodes(show_size, show_perm);

        // number of directory and files
        int dir_count = 1;
        int file_count = 0;
        for (TreeNode *cur = list_head; cur; cur = cur->next) {
            if (is_dir(&cur->inode)) dir_count++;
                else file_count++;
        }
        printf("\n%d directories, %d files\n\n", dir_count, file_count);

    // Clean up node list
        clear_tree_list();
}
```

5-4. print.c

```
#include <stdio.h>
#include <stdint.h>
```

```
#include "header.h"

// Prototypes
void cmd_print(int argc, char *argv[]);
static int read_block(int blk, void *buf);
static int read_inode(int ino, ext2_inode *buf);
static int get_inode_by_path(const char *path, ext2_inode *inode);
static int is_file(ext2_inode *inode);
static void help(void);
static void read_indirect(uint32_t blk, int level, uint32_t **out, int *count, int *cap);

// cmd_print: implement "print" command
void cmd_print(int argc, char *argv[]) {
    // Ensure a path argument is provided
    if (argc < 2) {
                help();
        return;
    }
    // parse path
    const char *path = argv[1];
    // parse options
    int max_lines = -1;
    for (int i = 2; i < argc; i++) {
        if (strcmp(argv[i], "-n") == 0) {
            if (i + 1 >= argc) {
                fprintf(stderr, "print: option requires an argument -- 'n'\n");
                return;
            }
            max_lines = atoi(argv[i+1]);
            i++;
        } else {
            help();
            return;
        }
    }
    // get inode for path
    ext2_inode inode;
    if (get_inode_by_path(path, &inode) < 0) {
        help();
        return;
    }
    // ensure it's a file, not directory
    if (!is_file(&inode)) {
        fprintf(stderr, "Error: '%s' is not file\n", path);
        return;
    }
    // open buffer for reading blocks
    char *buf = malloc(block_size);
```

```
if (!buf) { perror("malloc"); return; }
uint32_t remaining = inode.i_size;
int lines_printed = 0;

// array for block num
int cap = 16, count = 0;
uint32_t *blocks = malloc(cap * sizeof *blocks);

// direct (0..11)
for (int i = 0; i < 12; i++) {
    if (inode.i_block[i]) {
        if (count >= cap) {
            cap *= 2;
            blocks = realloc(blocks, cap * sizeof *blocks);
        }
        blocks[count++] = inode.i_block[i];
    }
}

// single indirect
if (inode.i_block[12])
    read_indirect(inode.i_block[12], 1, &blocks, &count, &cap);

// double indirect
if (inode.i_block[13])
    read_indirect(inode.i_block[13], 2, &blocks, &count, &cap);

// triple indirect
if (inode.i_block[14])
    read_indirect(inode.i_block[14], 3, &blocks, &count, &cap);

for (int idx = 0; idx < count && remaining > 0; idx++) {
    uint32_t blk = blocks[idx];
    if (read_block(blk, buf) < 0) break;

    uint32_t to_write = remaining < (uint32_t)block_size
                        ? (uint32_t)remaining
                        : (uint32_t)block_size;

    if (max_lines < 0) {
        if (write(STDOUT_FILENO, buf, to_write) < 0) {
            perror("write");
            break;
        }
    } else {
        for (uint32_t j = 0; j < to_write; j++) {
            putchar(buf[j]);
            if (buf[j] == '\n' && ++lines_printed >= max_lines) {
```

```
                    free(buf);
                    free(blocks);
                    return;
                }
            }
        }
        remaining -= to_write;
    }

    free(blocks);
}

// print usage
static void help(void) {
        printf("Usage : print <PATH> [OPTION]...\n");
}

// is_file: true if inode is regular file
static int is_file(ext2_inode *inode) {
    // use POSIX check on mode bits
    return !S_ISDIR(inode->i_mode);
}

// read_block: read block from disk image
static int read_block(int blk, void *buf) {
    off_t off = (off_t)blk * block_size;
    if (lseek(fs_fd, off, SEEK_SET) < 0) return -1;
    return read(fs_fd, buf, block_size) == block_size ? 0 : -1;
}

// read_inode: read inode from inode table
static int read_inode(int ino, ext2_inode *buf) {
    off_t base = (off_t)gd.bg_inode_table * block_size;
    off_t off = base + (ino - 1) * inode_size;
    if (lseek(fs_fd, off, SEEK_SET) < 0) return -1;
    return read(fs_fd, buf, inode_size) == inode_size ? 0 : -1;
}

// get_inode_by_path: same logic as in tree.c
static int get_inode_by_path(const char *path, ext2_inode *inode) {
    // special-case root
    if (strcmp(path, ".") == 0 || strcmp(path, "/") == 0) {
        if (read_inode(2, inode) < 0) return -1;
        return 2;
    }
    char *copy = strdup(path);
    if (!copy) return -1;
    int cur_ino = 2;
```

```
    ext2_inode cur;
    if (read_inode(cur_ino, &cur) < 0) { free(copy); return -1; }
    char *tok = strtok(copy, "/");
    while (tok) {
        if (!S_ISDIR(cur.i_mode)) break;
        int blocks = (cur.i_size + block_size - 1) / block_size;
        char *buf = malloc(block_size);
        int found = 0;
        for (int bi = 0; bi < blocks && !found; bi++) {
            uint32_t blk = cur.i_block[bi];
            if (!blk || read_block(blk, buf) < 0) continue;
            int off = 0;
            while (off < block_size) {
                ext2_dir_entry_2 *e = (ext2_dir_entry_2*)(buf + off);
                if (e->inode) {
                    char name[EXT2_NAME_LEN+1] = {0};
                    memcpy(name, e->name, e->name_len);
                    if (strcmp(name, tok) == 0) {
                        cur_ino = e->inode;
                        if (read_inode(cur_ino, &cur) < 0) { free(buf); break; }
                        found = 1;
                        break;
                    }
                }
                off += e->rec_len;
            }
        }
        free(buf);
        if (!found) break;
        tok = strtok(NULL, "/");
    }
    free(copy);
    if (tok) return -1;
    *inode = cur;
    return cur_ino;
}

static void read_indirect(uint32_t blk, int level, uint32_t **out, int *count, int *cap) {
    // block num buffer
    uint32_t *buf = malloc(block_size);
    if (!buf) return;
    // read block
    if (read_block(blk, buf) < 0) { free(buf); return; }
    int entries = block_size / sizeof(uint32_t);

    for (int i = 0; i < entries; i++) {
        uint32_t nblk = buf[i];
        if (!nblk) break;
```

```
        if (level == 1) {
            // data block
            if (*count >= *cap) {
                *cap *= 2;
                *out = realloc(*out, (*cap) * sizeof(uint32_t));
            }
            (*out)[(*count)++] = nblk;
        } else {
            read_indirect(nblk, level - 1, out, count, cap);
        }
    }
    free(buf);
}
```

5-5. help.c

```
#include <stdio.h>
#include "header.h"

// prototype
void all_help();
void tree_help();
void print_help();
void help_help();
void exit_help();

// main command function
void cmd_help(char *arg)
{
        if(arg == NULL){
                all_help();
        printf("\n");
        }else if(strcmp(arg, "tree") == 0){
        tree_help();
        printf("\n");
    }else if(strcmp(arg, "print") == 0){
        print_help();
        printf("\n");
    }else if(strcmp(arg, "help") == 0){
        help_help();
        printf("\n");
    }else if(strcmp(arg, "exit") == 0){
        exit_help();
        printf("\n");
    }else{
        printf("invalid command -- \'asd\'\n");
        all_help();
        printf("\n");
        printf("\n");
    }
```

```
}

// all_help: Display help for all supported commands.
void all_help(){
    tree_help();
    print_help();
    help_help();
    exit_help();
}

// tree_help: Usage instructions for the 'tree' command.
void tree_help(){
    printf("  > tree <PATH> [OPTION]... : display the diractory structure if <PATH> is a directory\n");
    printf("    -r : display the directory structure recursively if <PATH> is a directory\n");
    printf("    -s : display the directory structure if <PATH> is a directory, including the size of each file\n");
    printf("    -p : display the directory structure if <PATH> is a directory, including the permissions of each
directory and file\n");
}

// print_help: Usage instructions for the 'print' command.
void print_help(){
    printf("  > print <PATH> [OPTION]... : print the contents on the standard output if <PATH> is file\n");
    printf("    -n <line_number> : print only the first <line_number> lines of its contents on the standard
output if <PATH> is file\n");
}

// help_help: Usage instructions for the 'help' command itself.
void help_help(){
    printf("  > help [COMMAND] : show commands for program\n");
}

// exit_help: Usage instructions for the 'exit' command.
void exit_help(){
    printf("  > exit : exit program\n");
}
```

5-6. header.h

```
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/stat.h>
#include <stdint.h>
#include <fcntl.h>
#include <ctype.h>
#include <stdint.h>

/* --- macro --- */
#define EXT2_NAME_LEN  255
```

```
#define EXT2_N_BLOCKS   15
#define EXT2_FT_DIR     2
#define EXT2_S_IFDIR 0x4000
#define MAX_PATH 4096
#define MAX_FILE 255
#define MAX_LINE 256
#define MAX_ARG 20

/* --- user defined structure --- */
// On-disk ext2 superblock (fields through inode size)
typedef struct ext2_super_block {
    uint32_t s_inodes_count;
    uint32_t s_blocks_count;
    uint32_t s_r_blocks_count;
    uint32_t s_free_blocks_count;
    uint32_t s_free_inodes_count;
    uint32_t s_first_data_block;
    uint32_t s_log_block_size;
    uint32_t s_log_frag_size;
    uint32_t s_blocks_per_group;
    uint32_t s_frags_per_group;
    uint32_t s_inodes_per_group;
    uint32_t s_mtime;
    uint32_t s_wtime;
    uint16_t s_mnt_count;
    uint16_t s_max_mnt_count;
    uint16_t s_magic;              // 0xEF53
    uint16_t s_state;
    uint16_t s_errors;
    uint16_t s_minor_rev_level;
    uint32_t s_lastcheck;
    uint32_t s_checkinterval;
    uint32_t s_creator_os;
    uint32_t s_rev_level;
    uint16_t s_def_resuid;
    uint16_t s_def_resgid;
    uint32_t s_first_ino;
    uint16_t s_inode_size;
} ext2_super_block;

// On-disk ext2 group descriptor (minimal)
typedef struct ext2_group_desc {
    uint32_t bg_block_bitmap;
    uint32_t bg_inode_bitmap;
    uint32_t bg_inode_table;
    // ... other fields omitted
} ext2_group_desc;
```

```
// On-disk ext2 inode (simplified)
typedef struct ext2_inode {
    uint16_t i_mode;                // file type & permissions
    uint16_t i_uid;                 // owner UID
    uint32_t i_size;                // size in bytes
    uint32_t i_atime;               // access time
    uint32_t i_ctime;               // creation time
    uint32_t i_mtime;               // modification time
    uint32_t i_dtime;               // deletion time
    uint16_t i_gid;                 // group ID
    uint16_t i_links_count;         // hard links count
    uint32_t i_blocks;              // number of 512B blocks allocated
    uint32_t i_flags;               // flags
    uint32_t i_osd1;                // OS dependent 1
    uint32_t i_block[EXT2_N_BLOCKS]; // block pointers
    uint32_t i_generation;          // file version (for NFS)
    uint32_t i_file_acl;            // file ACL
    uint32_t i_dir_acl;             // directory ACL or high 32 bits of file size
    uint32_t i_faddr;               // fragment address
    uint8_t  i_osd2[12];            // OS dependent 2
} ext2_inode;

// On-disk ext2 directory entry (v2)
typedef struct ext2_dir_entry_2 {
    uint32_t inode;
    uint16_t rec_len;
    uint8_t  name_len;
    uint8_t  file_type;
    char     name[EXT2_NAME_LEN];
} ext2_dir_entry_2;

// Global variables for ext2 image state
extern int fs_fd;                       // file descriptor of image
extern ext2_super_block sb;             // superblock
extern ext2_group_desc gd;              // group descriptor
extern int block_size;
extern int inode_size;

// Initialization and main loop
int init_ext2_structures(const char *disk_image);
void cmd_loop(void);

// Command functions
void cmd_tree(int argc, char *argv[]);
void cmd_print(int argc, char *argv[]);
void cmd_help(char *arg);
```