CSCI4730/6730 – Operating Systems

PA 3: EXT2-like File System

Due date: 11:59pm, 11/23/2025 (Sunday)

Note

- Part #1 (70 points), Part #2 (30 points)

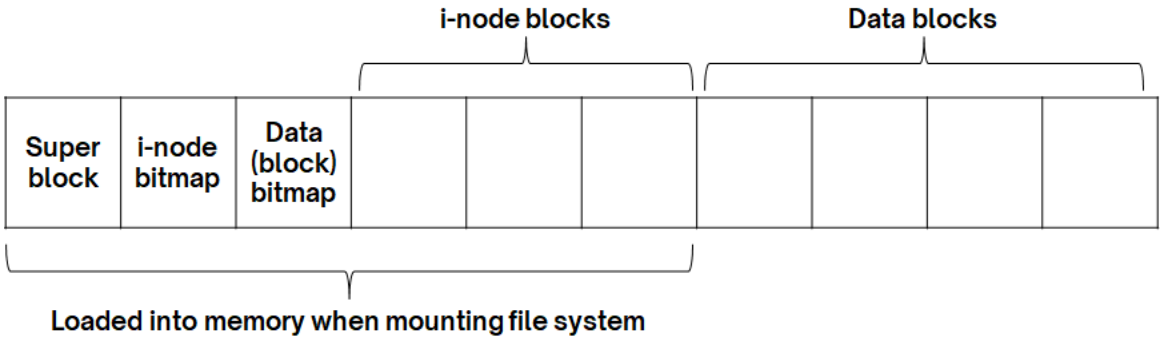
Description

In this programming assignment, you will implement a simplified file system simulator inspired by the EXT2 file system (a no-journaling system). This exercise aims to help you understand the mechanisms behind hierarchical directory structures and i-node management. The file system simulator should be able to

(1) Browse disk information, list files and directories, and display metadata for files and directories,
(2) Create and delete files and directories, and
(3) Read files.

The simulator will behave similarly to the EXT2 file system, but it will not include advanced features such as a per-process open file table, permission handling, or user management.

The figure below illustrates the file system structure of the simulator. The disk consists of 4096 blocks, each with a block size of 512 bytes. The superblock, i-node bitmap, and data bitmap are loaded into memory when the file system is mounted. Each i-node entry is 128 bytes, allowing 4 i-node entries to be stored in a single disk block.



You can start the simulator with the following command:

```
$./fs_sim disk.dat
```

Part #1: File Management: 70 points

The file system only uses direct blocks in the i-node to support small size files (up to 6,144 bytes = 12 * 512 bytes, 12 blocks). Each file in the file system has an i-node number, which is unique in this file system. I-node structure (**`Inode`**) is defined in "**`fs.h`**" file.

The file system needs to support the following functionalities. Please note that "**`df`**", "**`create`**", "**`stat`**", and "**`cat`**" are implemented in **`fs.c`**, and you need to implement "**`read`**", "**`rm`**", and "**`ln`**".

Command	**Arguments**	**Description**	**Func name in `fs.c`**
`df`		Show file system information. e.g., # of free blocks, # of free i-nodes	**`fs_stat()`** (Implemented)
`create`	<name> <size>	Create a file with <filename> as a name in the current directory and fill it with size of random string.	**`file_create()`** (Implemented)
`stat`	<name>	Show the status (metadata) of the file or directory with name <name>. It displays * i-node information * file or directory * # of blocks allocated * other info stored about this file/directory	**`file_stat()`** (Implemented)
`cat`	<name>	Print out the content of the file	**`file_cat()`** (Implemented)
`read`	<name> <offset> <size>	Read <size> bytes from the file <offset>	**`file_read()`** (Not implemented)
`rm`	<name>	Delete <name> file	**`file_remove()`** (Not implemented)
`ln`	<src_file> <new_file>	Create a hard link ("src_file" and "new_file" should be located in the same directory	**`hard_link()`** (Not implemented)

Part #2: Directory Management: 30 points

In the second part, you need to implement hierarchical, tree-structured directory. Each directory has an i-node number and an i-node block. In the simulator, each directory has up to 25 entries (files and sub-directories).

The first entry is always a current directory ("**.**") and the second entry is a parent directory ("**..**"). You will need to implement the following functionalities. In particular, you need to implement "**`mkdir`**" and "**`rmdir`**".

Command	**Arguments**	**Description**	**Func, name in `fs.c`**
`ls`		Show the content of the current directory	**`ls()`** (Implemented)
`cd`	<name>	Change the current directory to <name>	**`dir_change()`** (Implemented)
`mkdir`	<name>	Create a sub-directory <name> under the current directory	**`dir_make()`** (Not implemented)
`rmdir`	<name>	Remove the sub-directory <name>	**`dir_remove()`** (Not implemented)

Submission

Submit a tarball file using the following command

```
$tar czvf p3.tar.gz README.pdf Makefile *.c *.h
```

1. README file with:
    a. Your name
    b. List what you have done and how did you test them. So that you can be sure to receive credit for the parts you've done.
    c. Description of your design
2. The TA and instructor will check if your implementation correctly handles exception cases. e.g., i-node number validation, wrong arguments, etc.
3. All source files are needed to compile, run and test your code
    a. Makefile
    b. All source files
    c. Do not submit object or executable files
4. Your code should be compiled and run correctly in odin machine.
    a. No credit will be given if your code failed to compile with "make" in odin.cs.uga.edu (we will use your makefile).
5. Submit a tarball through ELC.