Operating Systems (COMPSCI 3SH3) Assignment 5
File Block Allocation Simulator [40 marks]

Prof. Neerja Mhaskar and Samkith Jain

Fall 2025

Rules and Guidelines

- No late assignment accepted, unless an MSAF is provided.
- If an MSAF is provided, then you will get 5 days extension on the assignment
- It is advisable to start your assignment early.
- Make sure to submit a version of your assignment ahead of time to avoid last minute uploading issues.
- Note that students/groups copying each other's solution will get a zero.
- The assignment should be submitted on Avenue under:

  Assessments → Assignments → Assignment 5 → [Group #] folder

- In your C programs, you should follow good programming style, which includes providing instructive comments and well-indented code. If this is not followed, marks will be deducted
- If working in a group of two, a `Readme` file containing information on your individual contributions should be provided.
- Use of generative AI is not allowed.

Overview

In this assignment, the goal is to implement the ***Indexed Allocation Scheme*** for a simulated flat file system using C. This scheme uses an index block to store pointers to data blocks, enabling efficient random access. *It is recommended that you use helper function where needed for modularity and readability.*

File System Specifications

Use the file system specifications from the `fs_os.h` header file created in Lab 5.

File System Implementation in `fs_os.c` file [30 marks]

Your `fs_os.c` file must implement all functions defined in `fs_os.h` file with following requirements. **Declare the `fileSystem` structure (`fs`) globally.**

File system operations [20 marks]

`initFS()` function:

1. initialize the volume control block in the file system structure, including the block numbers.
2. Display a message if initialization is successful including the total number of blocks created and the block size; otherwise, display and error message.

`createFile(filename, size)` function:

1. Check that the total number of files does not exceed the maximum allowed.
2. Then, verify that the number of blocks required for the new file is $\leq$ free blocks available. If yes, allocate data blocks using the indexed allocation scheme and the `allocateFreeBlock()` function. Allocate an index block to store pointers to data block.
3. Update the free block list.
4. Allocate a file information block ID to the new file by creating the function `getFileInformationBlockID()` to achieve this task.
5. Update file system's File Information Block available in the file system.
6. Display a message if delete is successful; otherwise display an error message.

`deleteFile(filename)` function

1. Locate the file information block ID (FIB ID) for the file.
2. Return all the blocks used by the file, including the index block, to the free block list using the `returnFreeBlock()`.
3. Return the FIB ID to the file system, and update the FIB IDs available in the file system.
4. Display a message if delete is successful; otherwise display an error message.

`listFiles()` function:

– Display all the files in the flat file system in the below format.

```
Root Directory Listing (2 files):
alpha.txt  |   3072 bytes |  3 data blocks | FIBID=0
beta.txt   |   5120 bytes |  5 data blocks | FIBID=1
```

File System Utility Functions [10 marks]

1. `allocateFreeBlock()` – Removes a free block from the head of the free block list and returns it.
2. `returnFreeBlock(Block)` – Adds a free block back to the tail of the free block list.
3. `printFreeBlocks()` – Displays all free block numbers and the total count of free block (as shown in the output.txt file).

Test Cases [5 marks]

Write a `main.c` program to perform the below tasks and use the `printFreeBlocks()` function to verify the correctness of your code:

- Initialize the file system.
- Create multiple files of different sizes.
- Delete some files and verify that blocks are reclaimed.
- Display the directory and free block list after each operation.

Makefile [5 marks]

You must provide a Makefile to compile and run the program, ensuring that all source files (`fs_indexed.c`, `main.c`, and any headers) are correctly linked.

Deliverables

- Submit the below files:
    1. `fs_indexed.h` (header file with structure definitions and function prototypes)
    2. `fs_indexed.c` (implementation of file system functions)
    3. `main.c` (driver program to test your implementation)
    4. Makefile
    5. Readme (if applicable)