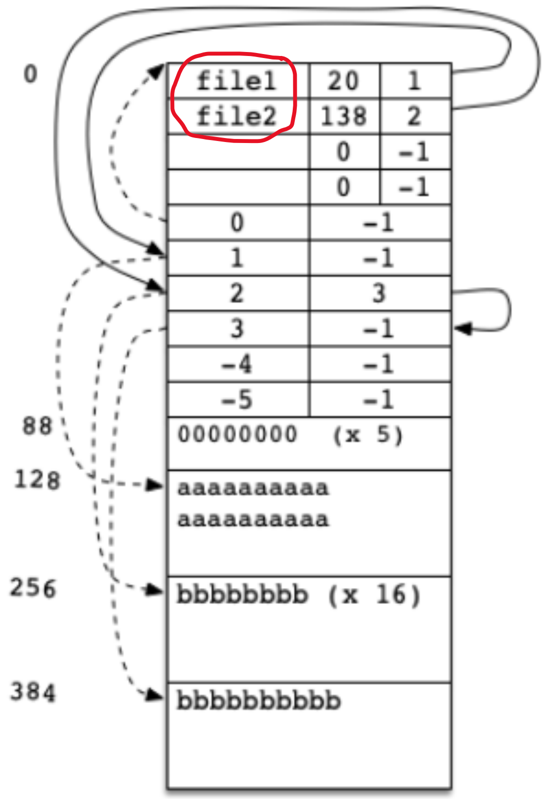
CSC209 ASSIGNMENT 1 OVERVIEW

ASSIGNMENT OVERVIEW

- The goal of this assignment is for you to design a simple simulated file system.
- Essentially, this file system will consist of an array of **blocks**, with the first block, or first few blocks, in the system containing information about the system and its files (**metadata**) in the form of fnodes and fentries
- Your job is to write functions that will **create, delete, read, and write** files. The function to initialize the filesystem itself has already been written for you, in initfs.c. You will build off of this structure.
- This assignment is fairly open ended – none of the signatures for the functions you will write have been written, so you may design your program however you wish. **You may edit any of the files in the starter code except for simfstypes.h and the Makefile.**
- **I recommend you read through initfs.c before getting started to understand how to build your functions into the existing file system.**

UNDERSTANDING THIS DIAGRAM

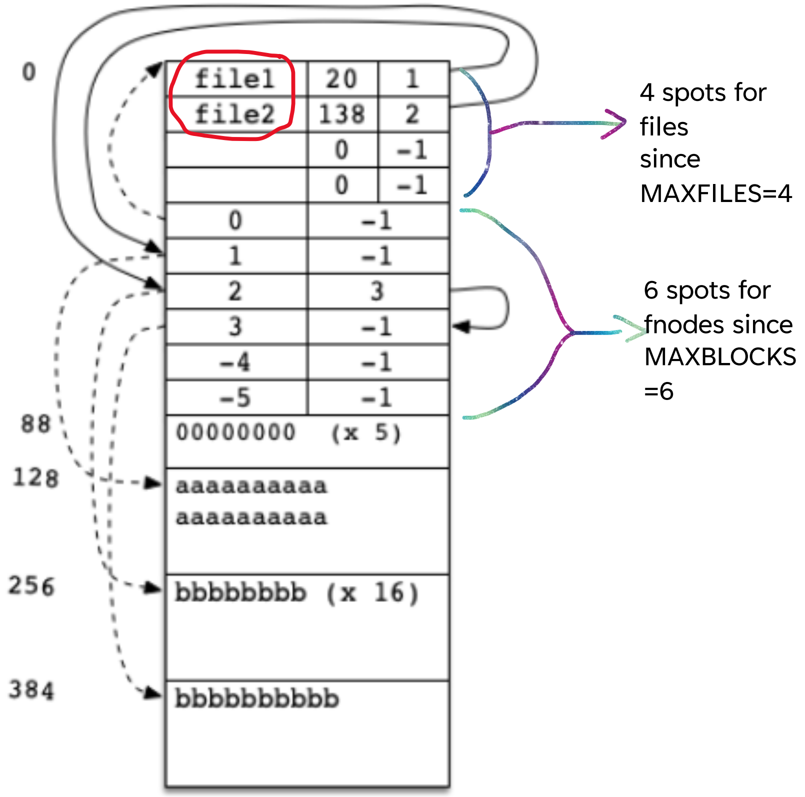
This diagram shows an example file system which contains two files. Those files are named ‘file1’ and ‘file2’.



UNDERSTANDING THIS DIAGRAM

This diagram shows an example file system which contains two files. Those files are named ‘file1’ and ‘file2’.

In this system, MAXFILES=4 and MAXBLOCKS=6 (the real values of these constants are defined for you in simfstypes.h and will be higher than in the example.

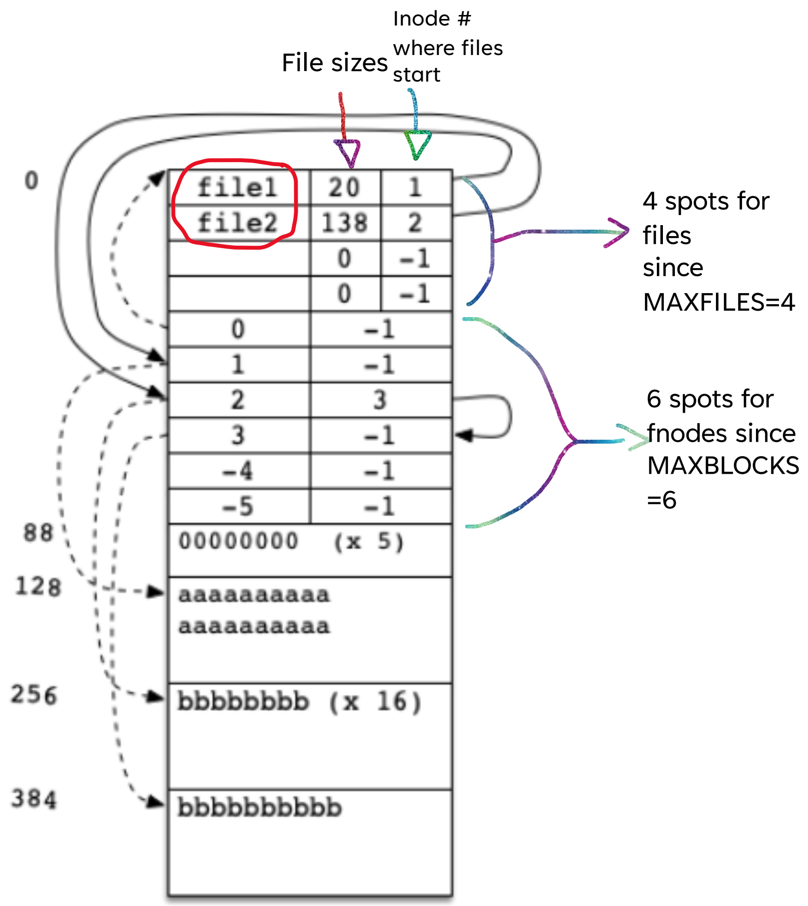


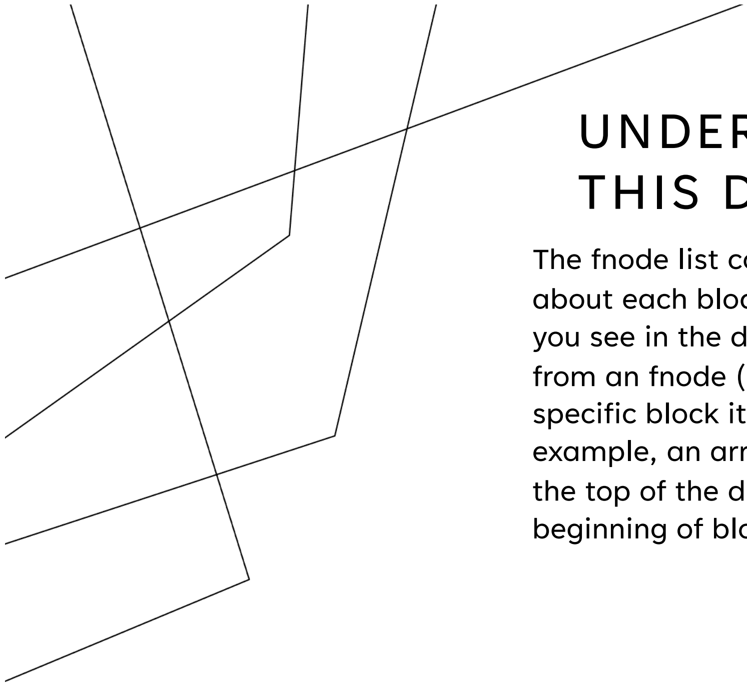
UNDERSTANDING THIS DIAGRAM

This diagram shows an example file system which contains two files. Those files are named ‘file1’ and ‘file2’.

In this system, MAXFILES=4 and MAXBLOCKS=6 (the real values of these constants are defined for you in simfstypes.h and will be higher than in the example.

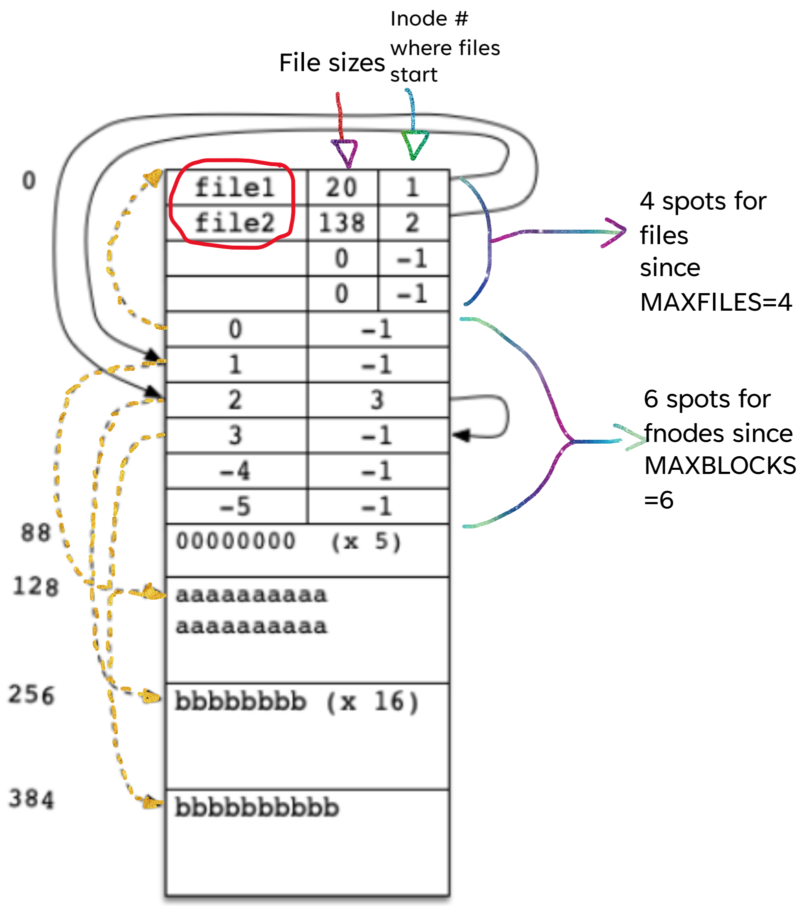
BLOCKSIZE is set to 128 (the same as in your code). Block 0 of the file system contains metadata (information about the files and fnodes), but not any actual content. This consumes 88 bytes of one block, and the remaining bytes of that block are filled with 0s.





UNDERSTANDING THIS DIAGRAM

The fnode list contains information about each block. The dotted arrows you see in the diagram are pointing from an fnode (block number) to the specific block it is referencing. For example, an arrow runs from fnode 0 to the top of the diagram since that is the beginning of block 0.

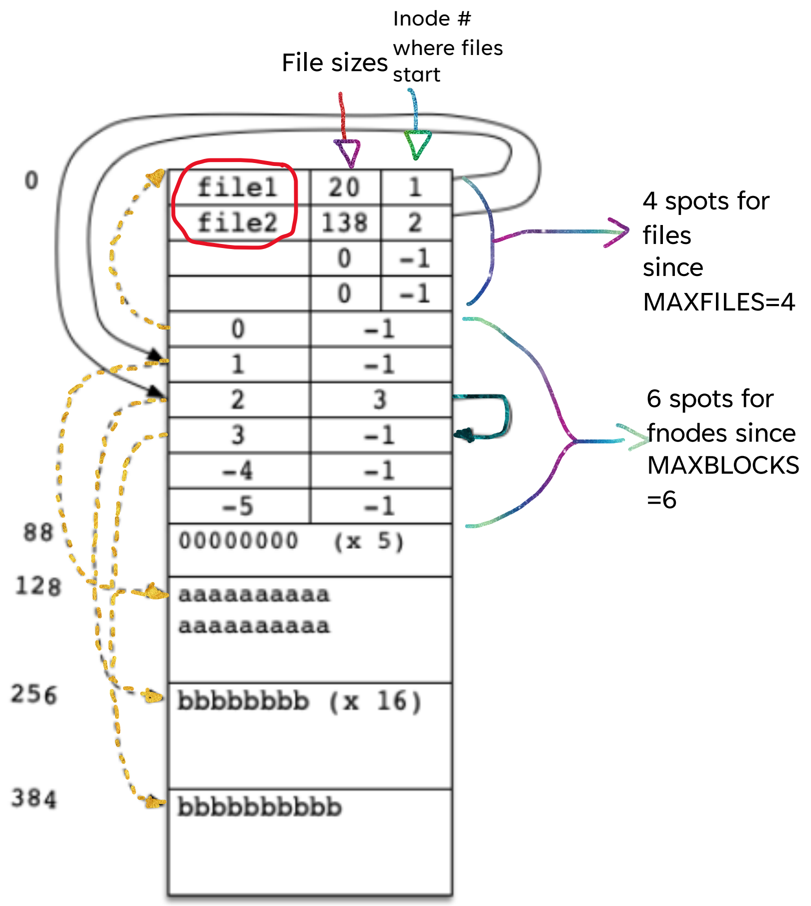


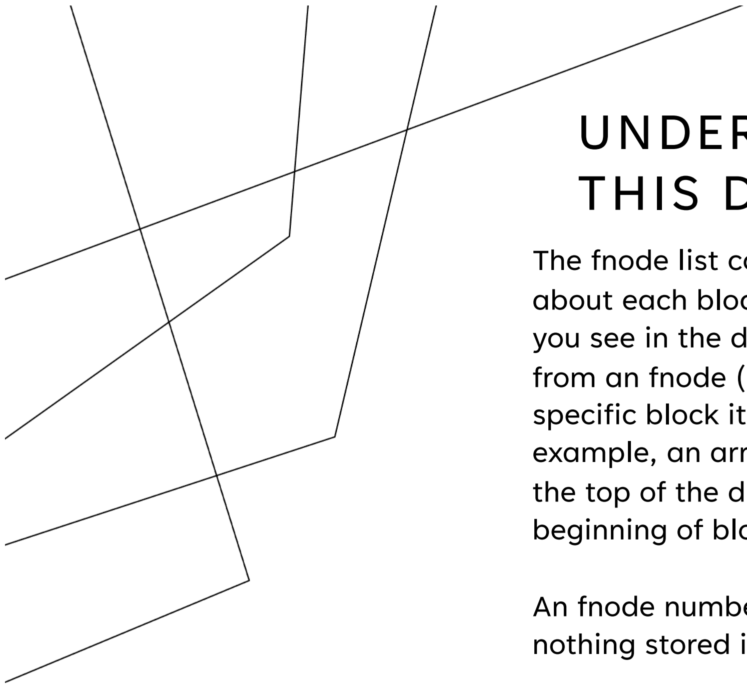
UNDERSTANDING THIS DIAGRAM

The fnode list contains information about each block. The dotted arrows you see in the diagram are pointing from an fnode (block number) to the specific block it is referencing. For example, an arrow runs from fnode 0 to the top of the diagram since that is the beginning of block 0.

An fnode number is negative if there is nothing stored in its block.

The number to the right of each fnode number refers to the fnode which represents the continuation of the data contained in that block. The file ‘file2’ here is 138 bytes so takes up more than one block. So fnode 2 points to fnode 3, which contains the rest of the file. These may not always be in consecutive blocks, however.



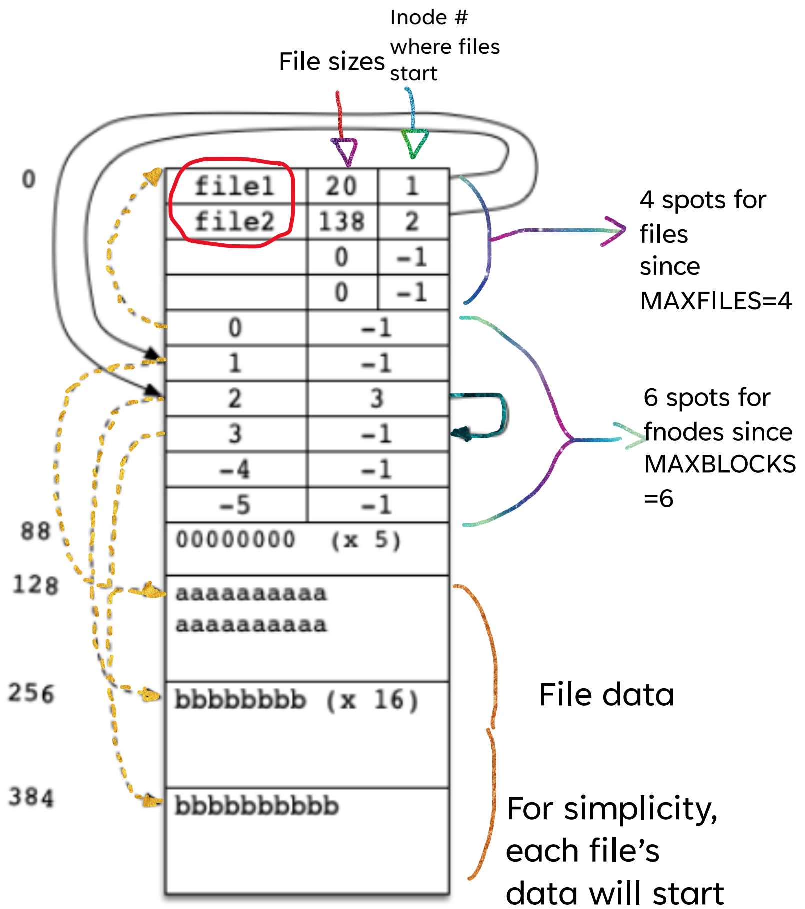


UNDERSTANDING THIS DIAGRAM

The fnode list contains information about each block. The dotted arrows you see in the diagram are pointing from an fnode (block number) to the specific block it is referencing. For example, an arrow runs from fnode 0 to the top of the diagram since that is the beginning of block 0.

An fnode number is negative if there is nothing stored in its block.

The number to the right of each fnode number refers to the fnode which represents the continuation of the data contained in that block. The file ‘file2’ here is 138 bytes so takes up more than one block. So fnode 2 points to fnode 3, which contains the rest of the file. These may not always be in consecutive blocks, however.



File data

For simplicity, each file’s data will start on a new block.

REMEMBER!

- Error check everything! If a command or system call *can* fail, error check it. Error checking is required for this assignment, and bad user input is NOT an excuse for your program segfaulting, crashing, or otherwise malfunctioning.
- Make sure your code runs on the UTM school servers! That is where we will test it, and if it doesn't compile and run on our systems you will lose marks.
- Make sure you compile and test your code right before submitting, even if you only made a small change. Mistakes happen and we would hate to have your code not compile because you made a small change that resulted in an error at the last minute.