

# Assignment 1a: File System Proposal

---

**Due** Oct 2 by 10pm      **Points** 3

---

## Overview

Your task in this two-part assignment is to design and implement a simple extent-based file system. An extent is a contiguous set of blocks allocated to a file, and is defined by the starting block number and the number of blocks in the extent. A single extent is often not sufficient, so a file may be composed of a sequence of extents.

## Part A: Proposal

For the proposal, you will design the data structures needed to implement the file system, describe the layout of data on the disk, and describe several of the important algorithms needed to implement basic file system functionality. The only code you need to write for this assignment is to fill in the missing parts of the [a1fs.h](#) file.

NOTE: you may NOT use the starter code of the file system exercises (namely, the ext2.h header) in your a1fs.h file. The extent-based file system that you are designing is fundamentally different from index-based ext2. You need to think carefully about reusing any ext2 concepts as they might not fit well in an extent-based file system. All the fields in your data structure definitions must have a clear purpose (described in a short comment, if necessary). The a1fs.h file must be valid C code that compiles without errors or warnings.

For context, your file system will implement functionality to support the following file system operations:

- formatting the disk image (mkfs)
- creating and deleting directories (mkdir, rmdir)
- creating and deleting files (creat, unlink)
- displaying metadata about a file or directory (stat)
- setting the size of a file (truncate)
- writing data to files and reading data from files (read, write)


You can expect to use absolute paths on your file system for these operations. Directories can be nested.

NOTES:

- Each used disk block must either contain file system metadata, or the data of a file or directory. Avoid designs that would require storing both data and metadata within the same block.

- There are no limits on the maximum size of the file system, the maximum number of inodes (files and directories), the maximum number of directory entries in a directory, or the maximum size of a file.
- The maximum number of extents for a file or directory is 512.
- The file system should not pre-allocate any metadata (except inodes) or data blocks. Any metadata or data blocks should only be allocated on demand.
- Every file is created empty (with size 0), and its maximum size cannot be known in advance. Files can only be extended or truncated from the end - there is no such thing as removing or adding blocks in the middle of a file.

Your proposal will consist of two parts:

1. The completed [a1fs.h](#)  file. Remember to include comments that explain the purpose of each of the fields and/or structs that you add.
2. A 2-3 page pdf file with the following information.
  - A simple diagram of the layout of your disk image. The diagram will indicate where the different components of your file system will be stored.
  - A description of how you are partitioning space. This will include how you are keeping track of free inodes and data blocks, and where those data structures are stored, how you determine which blocks are used for inodes. Note that the size of the disk image and the number of inodes are parameters to mkfs.
  - A description of how you will store the information about the extents that belong to a file. For example if the data blocks for file A are blocks 2,3,4 and 8,9,10, then the extents are described as (2,3) and (8,3). Where is this information about the extents stored?
  - Describe how to allocate disk blocks to a file when it is extended. In other words, how do you identify available extents and allocate it to a file? What are the steps involved?
  - Explain how your allocation algorithm will help keep fragmentation low.
  - Describe how to free disk blocks when a file is truncated (reduced in size) or deleted.
  - Describe the algorithm to seek to a specific byte in a file.
  - Describe how to allocate and free inodes.
  - Describe how to allocate and free directory entries within the data block(s) that represent the directory.
  - Describe how to lookup a file given a full path to it, starting from the root directory.

Each of these components should be quite short, but still clearly convey your ideas using correct English grammar. We expect that the proposal document should be no more than 3 pages including the diagram. This page limit does not include the a1fs.h file.

We will provide feedback on your proposal. As you start implementing your file system, you may discover that some of the design decisions you included in your proposal no longer make sense. That is totally fine. Your implementation will **not** be graded on how closely it matches your design. The proposal gives you a chance to think through the design and get feedback before you get too far into the implementation.

## Submission:

Submit two files to the a1a directory on MarkUs: a1fs.h and a pdf file containing your proposal.

NOTE: The a1b assignment will be open on MarkUs, so that you can begin to work on your code while you are working on your proposal. You must, however, commit the above to files to the a1a directory to receive credit for them.