Note to Recruiter

Dear Recruiter,

Thank you for taking your time to review my response to this technical assessment. I understand you probably have a lot of these to get through, so this documentation aims to explain all the features and tests you can run on my program, as well as the implementation details.

This program is written in C++ utilizing smart pointers for automatic memory management. The 'Tests' folder contains some tests that you can use to evaluate this program. Each test file will be explained below in the 'Tests' section of this document.

The program offers the basic set of instructions such as 'read', 'write', 'delete', but also offers some bonus instructions such as 'room' and 'list'. You can find what each command does in the 'Instructions' section of this document. You can compile the program using the Makefile provided; which will create an executable named 'fileSystem'.

In the 'Design' section of this document, you will find a uml diagram and an explanation of how it's implemented.

I look forward to the opportunity to speaking with you live should you deem my submission desirable!

Thanks,
Kefan Cao

(519)722-9687
kefancao@outlook.com

Instructions

To compile and run the program, use the 'Makefile' provided. This will create an executable named 'fileSystem'. The file system supports the following commands

- save (s)
  - Requests a file id to reference to as well as the file size.
  - Saves the requested information in an index node and mark the appropriate indices of a bitmap as occupied.
  - Prints the indices of the bitmap that is now used by the file as an array accompanied by how many blocks was used in total.
- read (r)
  - Given a file id, it will search for the file in a hashtable and try to find a match. If a match is found, the location of the file is printed as an array, and the total number of blocks used is also printed.
  - If a match is not found, an error message is printed to alert the user that the file does not exist and they should try again.
- delete (d)
  - Given a file id, it will delete the file id from the disk and mark the areas containing the file on the disk as unoccupied.
  - If no match is found, an error message is printed to alert the user that the file does not exist and they should try again.
- room (x)
  - Outputs the available memory in the disk as well as the total memory the disk has.
- list (l)
  - Lists all of the files inputted alongside their size in KB.

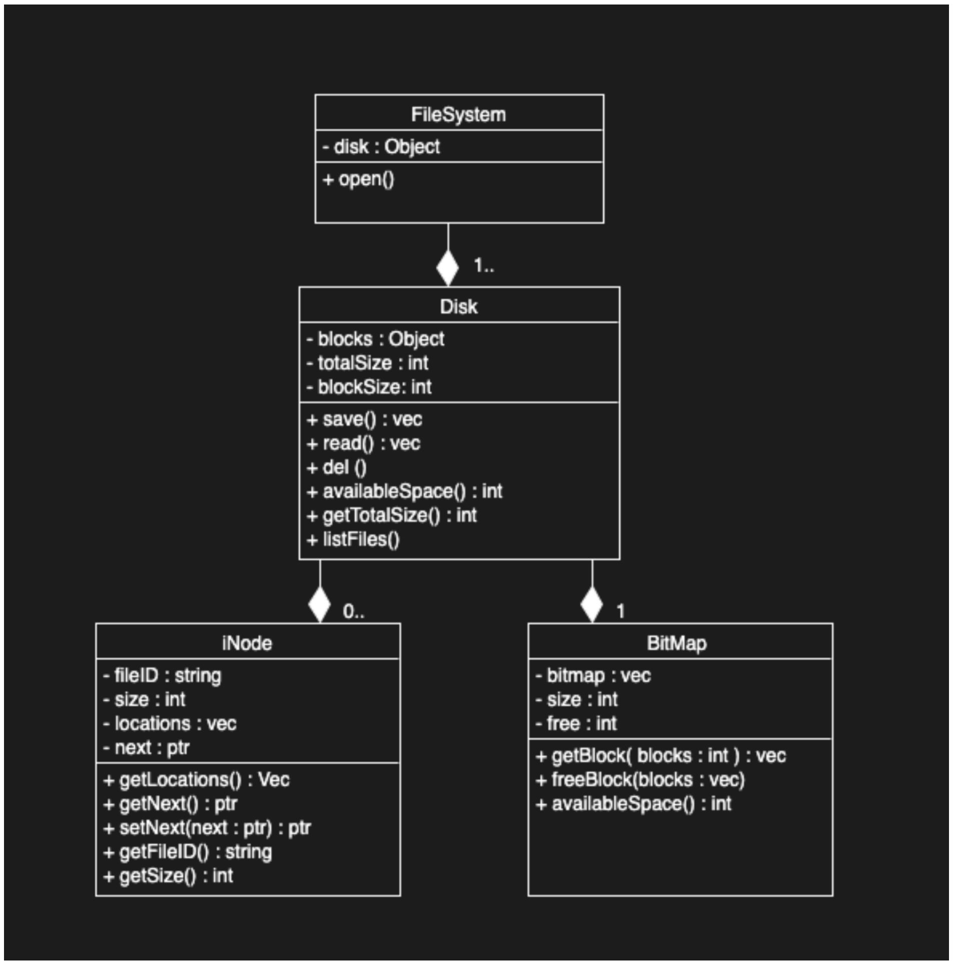
Tests

Below are a list of files included in the 'Tests' directory to test basic functionality.

- saveFull.in
  - Saves 'file1' and 'file2' onto the disk to occupy the entire disk, then tries to save additional files, but is prompted an error message (sent to standard output) that the storage is full. It then checks the files stored and the available space using the list and room commands supplied.
  - See 'saveFull.out' for the output.
- memReclaim.in
  - This test file shows that memory freed can indeed be reused again.
  - It first saves two files, then delete the first one, then saves a third file. Note that the third file occupies the room of the deleted file.
  - See 'memReclaim.out' for the output.
- readFiles.in
  - This test file examines the read command in depth.
  - It first saves a sequence of files, then calls read on one of them, delete that file and attempts to read it again. Because that file is deleted, no result will be outputted and an error message prompts the user letting he/her know that the file does not exist.
  - See 'readFiles.out' for the output.
- storeFiles.in
  - This test file examines the save command in depth.
  - This test will first save a sequence of files varying in size to make sure the proper amount of blocks are allocated. It will then delete files in the middle segment to make sure that that region is reclaimed and can be used again to store new files.
  - See 'storeFiles.in' for the output.
- deleteFiles.in
  - This test file examines the delete command in depth.
  - This test first saves multiple files to a single row of the hash table and performs delete action on the head of the linkedlist, the tail of the linked list, and the middle of the linked list. To make sure each are deleted, read operation is called right after to ensure it is wiped from the disk.
  - See 'deleteFiles.in' for the output.

Note that further tests have been done to ensure functionality and are not listed here.

Design



Consider the above uml diagram. The program is built with a single disk which owns a bitmap representing the data region of the disk and a iNode class containing data about files which includes file id, file size, and the location of these files. The disk is able to perform write, read, and delete actions on the files and the bitmap to simulate an actual file manager. The Disk stores the iNodes (or files) in a hash table for faster access.

Also note that the Disk object is very independant, so implementing a file manager with multiple Disk objects would not be an issue if desired. Extra features are implemented so that the user is able to see which files are on the disk and how much room they take up; which is achieved by storing the file size in the file metadata.