EEL-4736/EEL-5737 Principles of Computer System Design

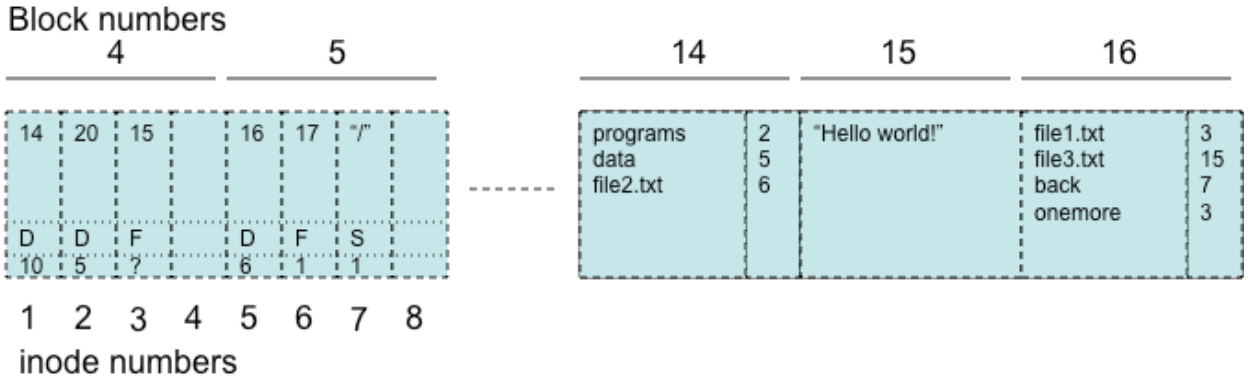
Homework #2

Assigned: 8/30/2019; Due on 9/13/2019 – To be done individually

This assignment is divided into two parts, Part-A will test your understanding of the concepts covered in class by working out problems and Part-B involves implementing the inode layer of a file system.

Part A (30% of HW#2 grade)

A-1) Textbook exercise 2.3
A-2) Textbook exercise 2.4
A-3) Consider the figure below representing data stored on a computer's hard disk and mounted as the *root file system*. At the bottom of each inode, its **type** (D (directory), F (file), and S (symlink)) and **reference count** are shown (e.g. D, 10 for inode 1), based on the UNIX file system described in class.



i) Suppose blocks are 8KBytes (8192 Bytes) in size, and there are 8 inodes per block. Assume an inode uses 8 Bytes to store type and reference count; the rest of the inode space is used to store block numbers. Suppose the disk has 2^32 blocks, and 2048 blocks are used to store inodes.
   a. What is the largest number of files that can be stored?

   b. What is the largest file size that can be stored (in KBytes)?

ii) Consider the LOOKUP resolution procedure as discussed in class.
   a. What is returned from LOOKUP("data",14)?

   b. What is returned from LOOKUP("file1.txt",5)?

   c. What is returned from LOOKUP("programs",5)?

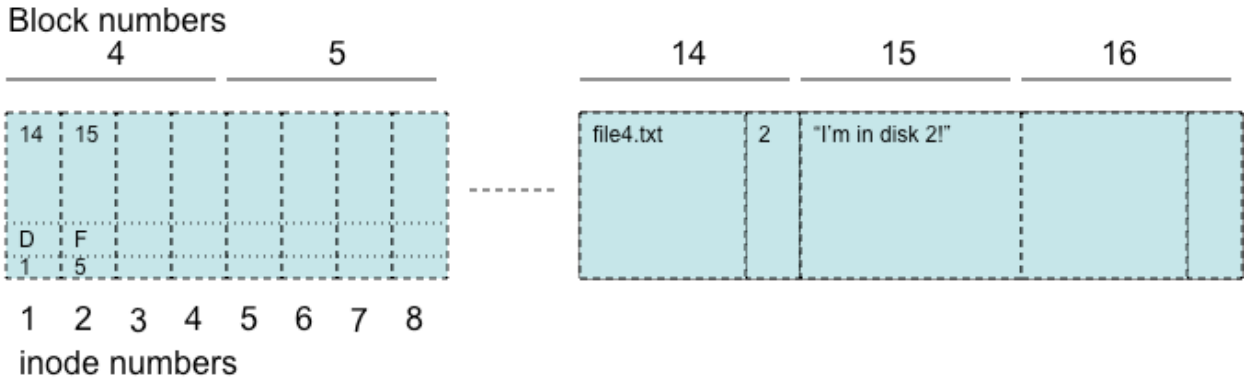
iii) Suppose you create a new hard link “xyz” in directory “/data” that links to “/data/file1.txt”. Describe (in words, or use a drawing based on the figure above) the new state *of all blocks that may have changed* due to this operation

iv) Assume the sequence of events for *two different processes* (A and B) *running in the same computer where this file system is mounted*:

Time T1: A issues fdA=open(“/data/file1.txt”), and obtains fdA=3
Time T2: B issues fdB=open(”/data/file1.txt”) and obtains fdB=3
Time T3: thread A issues read(fdA,bufA,n=5)
Time T4: thread A issues close(fdA)
Time T5: thread B issues read(fdB,bufB,n=5)

Where fdA, fdB are file descriptors, bufA and bufB are memory buffers, and n is the number of characters to read. What values (if any) are returned in the buffer bufA of thread A at time T3, and bufB in thread B at time T5?

v) Suppose now that the system has a *second hard disk*, with a second file system as depicted in the figure below. Suppose you mount this second file system under directory “/mnt” of the root file system. Is it possible to create a hard link “file4.txt” under “/data” that links to “file4.txt” in the second file system? Is it possible to create a symbolic link “file4.txt” under “/data” that links to “file4.txt” in the second file system? Describe (in words, or use a drawing based on the figure above) the new state *of all blocks that may have changed* due to these operations, if they are possible.



Part B (70% of HW#2 grade)

In HW#1, you used the raw MemoryInterface to read and write strings across consecutive blocks in a storage system. In this assignment, you will implement the inode layer discussed in class. This will provide a basis to implement file system objects aggregating non-contiguous blocks, with different types, and varying sizes.

For this assignment, you are given BlockLayer.py, which implements the block layer described in class atop the MemoryInterface layer, as well as a "skeleton" code as a starting point for InodeLayer.py. Your assignment is to complete the implementation of InodeLayer.py by programming the **read()** and **write()** methods that allow reading and writing blocks of data from a given offset. You will also implement a **copy()** method that builds on the read() and write() methods:

read(self,inode,offset,length):

- takes as an input an "inode" object (you can create using new_inode)
- reads up to "length" bytes, starting from "offset", from the blocks indexed by the inode object
- returns the updated "inode" object, and the data read

write(self,inode,offset,data):

- takes as an input an "inode" object
- writes "data", starting from "offset", to the blocks indexed by the inode object.
- Returns the updated "inode" object

copy(self,inode):

- takes as an input an "inode" object
- copies the entire contents associated with the input inode to a new "inode" object
- Returns the new "inode" object

Requirements:

1) Your design must return an error if the inode type is not a file
2) Your design must return an error if reading/writing from an offset larger than the file size
3) Your design must truncate data if attempting to write beyond the maximum size of a file
4) Your design must update "accessed" (read or write) and "modified" (write) times in the inode
5) Depending on the offset, writes may start in the middle of an existing block – in this case, you need to read and update the block
6) To simplify implementation, you may assume writes can "clear out" (invalidate) all old data starting from the offset, before writing the new data (feel free to use free_data_block())

Submission guidelines:

Turn in through Canvas a zip file named HW2.zip containing the following three files:

1. HW2partA.pdf – Solution to the questions in Part A
2. All the Python files with updated InodeLayer.py.
3. HW2design.pdf – a PDF document with a short (1-pager) description of how you went about your design and the tests you have conducted to check the functionality of your code

Make sure your Python code is well commented and tested before submission.