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

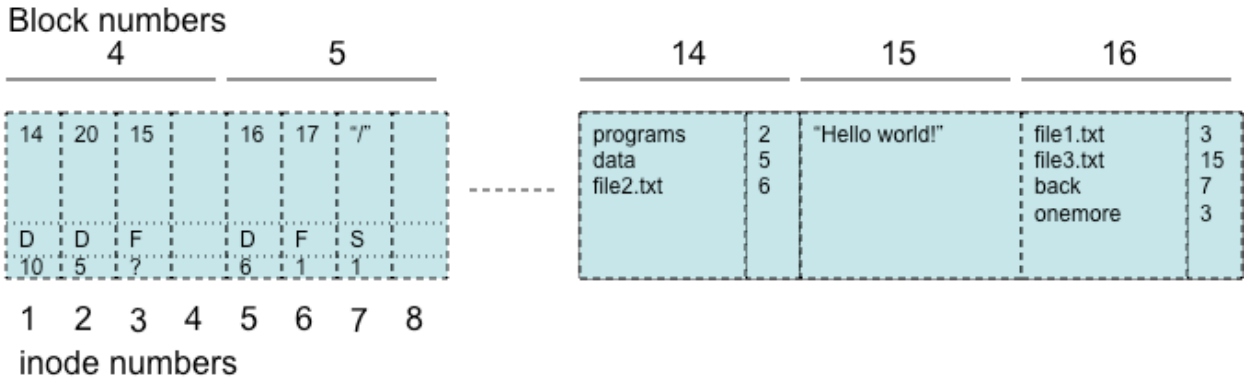
Homework #2

Assigned: 8/28/2017; Due 5 pm on 9/18/2017 – 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 will involve extending the file system from previous assignment to divide data into blocks.

Part A

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 4KBytes (4096 Bytes) in size, and there are 4 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 1024 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("file2.txt",2)?

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

c. What is returned from LOOKUP("Hello world!",5)?

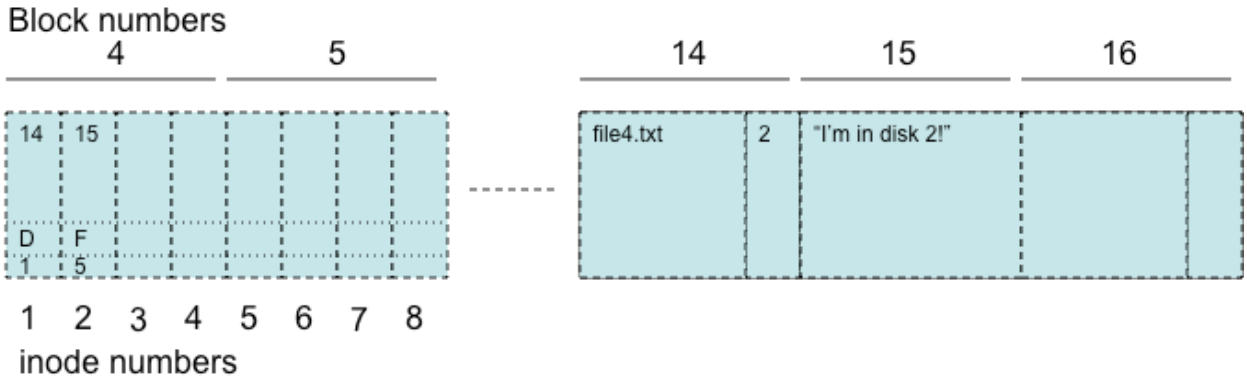
iii) Suppose you create a new hard link "xyz" in directory "/data" that links to "/data/file3.txt". Describe (in words, or use a drawing based on the figure above) the new state *of all blocks that may have changed* due to this operation

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

Time T1: A issues fdA=open("/data/file3.txt"), and obtains fdA=3
Time T2: B issues fdB=open("/data/file3.txt") and obtains fdB=3
Time T3: thread A issues read(fdA,bufA,n=5)
Time T4: thread A issues close(3)
Time T5: thread B issues read(fdB,bufB,n=5)

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 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

In HW#1, you used the flat in-memory file system, in which you were able to do all file system operations including creating, reading and writing files in default constant context. If you look at the constructor of the Memory class in memory.py file (line no 20), it uses two dictionaries (sometimes called 'Hash tables') called "files" (line no 21) which is used to store the metadata of files system objects and another called "data" (line no 22) to store the data in the files.

As discussed in class, data is stored in the form of blocks of fixed size in the disk. Representation of data in the form of blocks of equal size has several advantages like improved I/O performance, error checking and correction and redundancy/replication for fault tolerance. In this homework, you will extend the file system implementation in *memory.py* file to split the file data into several blocks of equal size as they are stored in the physical disk.

Use the fixed block size as 8 bytes. You can define this number as a constant global variable and use it wherever required. Data content of each file should be split into the blocks of 8 bytes each. Though there should not be any functionality difference seen from the upper filesystem layer, you need to split the data into blocks of 8 bytes and operate on these data blocks. You can use the list of strings/bytes to store the data.

Submission guidelines:

Turn in through Canvas the zip file containing the following:

1. homework2partA.pdf – Solution to the questions in Part A
2. memoryBlockFS.py – Your Python code that implements data storage in blocks.
3. homework2design.pdf – PDF describing the design of your implementation and tests you have conducted to check the functionality of your code.

Make sure your Python code is well commented and tested before submission. Copy and the paste the python code (memoryBlockFS.py) at the end of your homework2design.pdf file.