

CSC 262  
Homework 3 *Solutions*  
Due Oct. 9, 2008

1. The following questions are based on the following simplified disk diagram:

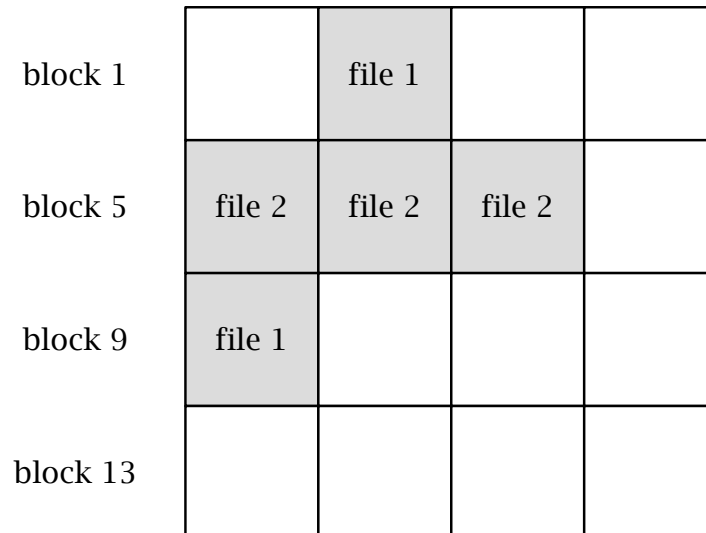
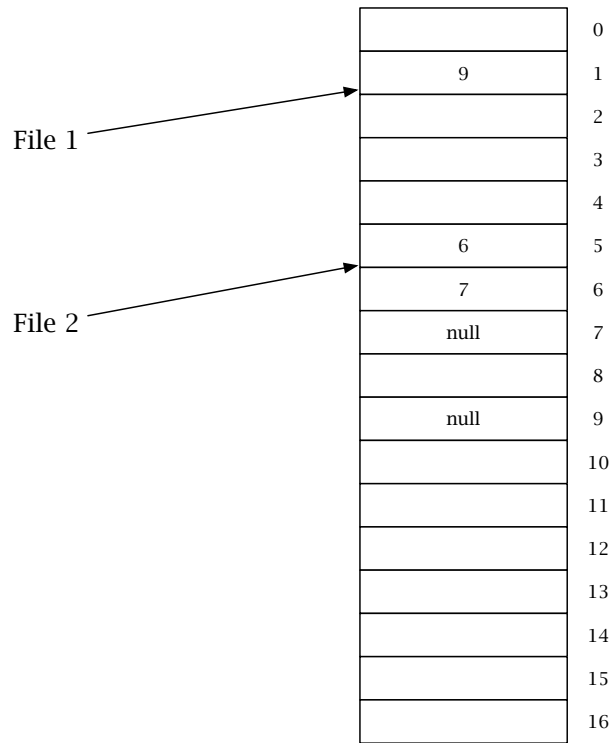


Figure 1: A teeny disk

This disk has only 16 blocks, and two files on it.

**A** What would the free block bitmap for this disk be?  
*010011101000000*

**B** What would the file allocation table look like? (Be sure to include the pointers for file 1 and file 2).



2. On a system with a block size of 2048 bytes, how many entries would a FAT require for a 250 GB hard drive (assume 1 GB is  $2^{30}$  bytes)? Assuming 4 bytes per entry, how much memory would the FAT consume?

*2048 bytes is  $2^{11}$  bytes, so the number of entries is the total size of the disk divided by the block size or:  $\frac{250 \cdot 2^{30}}{2^{11}}$  which simplifies to:  $250 \cdot 2^{19} = 125 \cdot 2^{20} = 131072000 \approx 131$  M entries.*

*If the FAT uses 4 bytes per entry, then the size consumed is equal to the number of entries times 4 bytes or:  $125 \cdot 2^{20} \cdot 4 = 500 \cdot 2^{20}$  and because  $2^{20}$  bytes is a megabyte, this FAT will consume 500MB of memory.*

3. How large would the bitmap be for the hard drive mentioned above?

*The bitmap would have the same number of entries as the FAT, but each entry is only 1 bit large. Therefore, it would take 131M bits. Because memory is byte-addressable, it would be more convenient to represent this as bytes, so the storage required is:  $\frac{125 \cdot 2^{20}}{2^3} = 125 \cdot 2^{17}$  bytes. Because  $2^{10}$  bytes is 1K, that means that this will take up  $125 \cdot 2^7 = 125 \cdot 128 = 16000$ K, or just under 16MB.*

4. The following questions concern the order of operations on an ext2 filesystem. Although ext2 is technically unreliable, efforts are made so that an ext2 filesystem won't get too in-

consistent. In general, you are trying to prevent ‘dangling’ references. An example of a dangling reference is a block pointer in an inode referring to a block that doesn’t contain file data (i.e. the power went off before the bytes were written to disk).

- A** Describe the actions and their order that are taken when a user creates a file. The actions must be ordered such that if the power goes out at any point (even in the middle of an action), there will be no dangling references in any inodes. *HINT: think in terms of allocating space, writing out meta-data/inodes, and writing file data*

*There are several ways of formulating this, but the basic idea is that first you write out the data to the disk, then only when it’s actually committed to the disk do you write out the meta-data that references that data. In that case, an acceptable order of operations for file creation would be:*

- 1) Allocate data blocks (if any are needed).*
  - 2) Write out the updated free-block bitmap.*
  - 3) Write out the data blocks (if any)*
  - 4) Allocate a new inode*
  - 5) Write out the updated free-inode bitmap.*
  - 6) Write out the new file’s inode. (At this point the file is consistent with respect to itself, but the containing directory is unaware of its existence).*
  - 7) Write out the updated directory data for the parent directory.*
  - 8) Write out updated group desc. (Mostly for free-space numbers).*
  - 9) Write out updated superblock (if necessary)*
- B** Describe the actions and their order when deleting a file. For the purposes of this question assume that the file is just large enough to require one pointer in the indirect block. The actions must be ordered such that if the power goes out at any point (even in the middle of an action), there will be no dangling references in any inodes. *For deletion, the same rule-of-thumb applies. First you write out the lowest-level data, then you update the meta-data that directly references it.*
- 1) Write out updated inode (set the deleted flag).*
  - 2) De-allocate data blocks*
  - 3) Write out updated free-block bitmap*
  - 4) Write out updated free-inode bitmap*
  - 5) Write out updated parent directory inode*
  - 6) Write out updated group desc. (if necessary)*
  - 7) Write out updated superblock (if necessary)*