Final Exam
CS460: Operating Systems
13 December, 1995
Jim Plank

Do all questions. Note that all questions are *not* worth the same number of points. In particular:

- Question 1 is worth 10 points and should take about 10 minutes to answer.
- Question 2 is worth 25 points and should take about 30 minutes to answer.
- Question 3 is worth 25 points and should take about 30 minutes to answer.
- Question 4 is worth 40 points and should take about 50 minutes to answer.

Question 1: 10 Points (10 minutes)

Suppose you have written a distributed file system using remote procedure call (RPC).

Part 1: 4 points

Describe how clients use this mechanism to do remote file operations.

Part 2: 6 points

Describe how the server responds to clients to perform remote file operations. Why is it a good idea to have the server be threaded?

Question 2: 25 Points (30 minutes)

Behold the following C file:

```
f()
{
  if (i()) {
    ii();
    if (i()) {
      fifi(i());
      if (i()) {
        f();
      }
    }
  }
  fifi();
  if (i()) {
    fifi(i());
  }
}
```

The character usage count is the following:

```
Freq:       5    Char: '{'
Freq:       5    Char: '}'
Freq:       5    Char: ';'
Freq:      12    Char: 'f'
Freq:      16    Char: ')'
Freq:      16    Char: '('
Freq:      17    Char: '^m'
Freq:      18    Char: 'i'
Freq:      62    Char: ' '
```

Part 1: 8 points

Construct a Huffman tree for the file.

Part 2: 4 points

And how many bits are saved over ASCII coding when you encode with your Huffman tree. You can give an expression instead of a number to save time.

Part 3: 7 points

Suppose you were writing a compressor for C files based on Huffman coding. How would you write it so that you don't have to store the Huffman tree with each file?

Part 4: 6 points

Give two simple ways that you could optimize your compressor for C files, yet still have it be based on Huffman coding. (When I say optimize here, I mean optimize the compression, not the speed of the compressor).

Question 3: 25 Points (30 minutes)

Suppose you have a system with 32-bit pointers and 4096-byte pages that supports two-level paging. All page table entries are 4 bytes. Discuss in detail how the hardware translates virtual addresses to physical addresses. In your discussion, make sure you address the following questions:

- Exactly how is the translation done?
- What do page table entries look like?
- What hardware is necessary so that this system is efficient, and how does this hardware work?
- How much physical memory is taken up by each process?
- What happens on a context switch?

Question 4: 40 Points (50 minutes)

A *log-structured file system (LFS)* is one which works as follows:

- A large area of memory is allocated in the operating system – it is called the *write buffer*.
- Whenever a file is written to, the write is performed in the write buffer.
- When the write buffer is full, the entire buffer is written to contiguous disk blocks.

This is straightforward when you are creating and appending to a file. However, when you are overwriting a file block, it is more subtle. What happens is that you read the old values from the disk block into the write buffer, the writes get performed in the write buffer, and eventually the writes go to disk when the write buffer is written to disk. Of course, this means that they go to a *different* disk block than before. The old disk block for that file block is deemed to be garbage. Every now and then, the operating system executes a "garbage collector process," which takes large, contiguous blocks of the disk and compacts them.

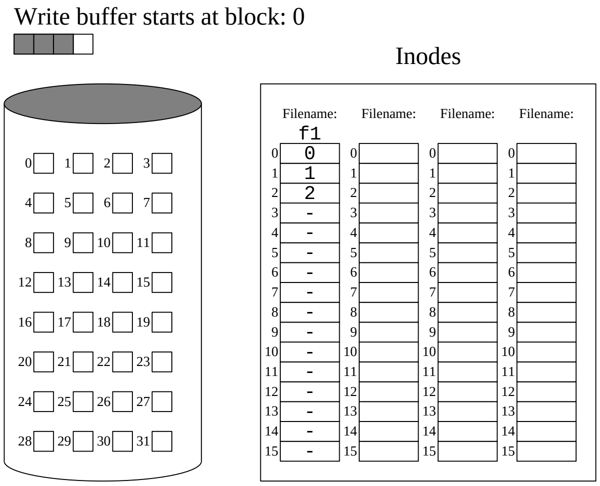
Whenever a file is deleted, its blocks on disk become garbage, and its blocks in the write buffer are freed for reuse.

Unix inodes are used to map files to disk. Inodes have the following structure:

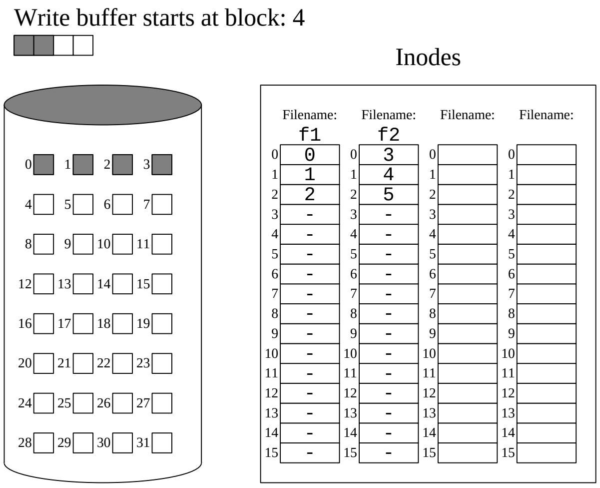
- The first 12 pointers in the inode point directly to the first 12 file blocks.
- The next pointer is to a single indirect block.
- The next pointer is to a double indirect block.
- The next pointer is to a triple indirect block.

Suppose that disk blocks are 1K and that the write buffer is 4K. The following pictures depicts the system after the following events: Free blocks are white. Used blocks are gray. Garbage blocks are black.

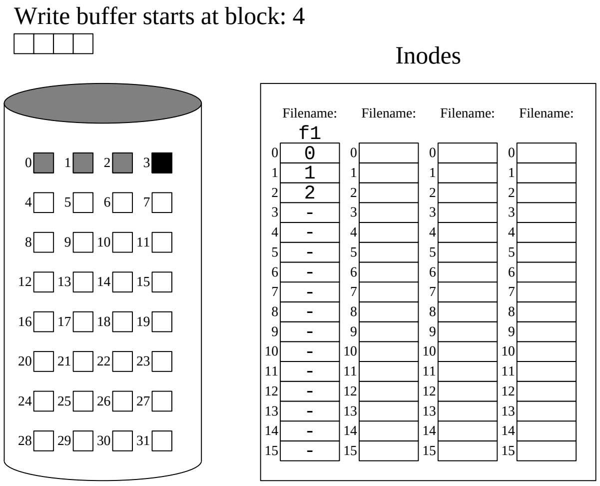
1. Process 1 creates the file "`f1`" and writes blocks 0 through 2.



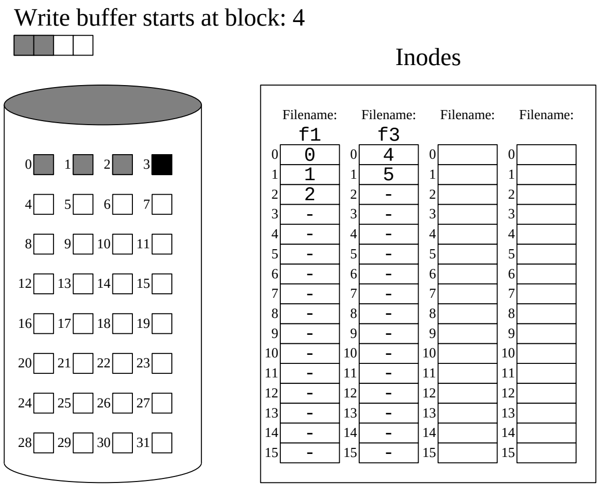
2. Process 2 creates the file "`f2`" and writes blocks 0 through 2.



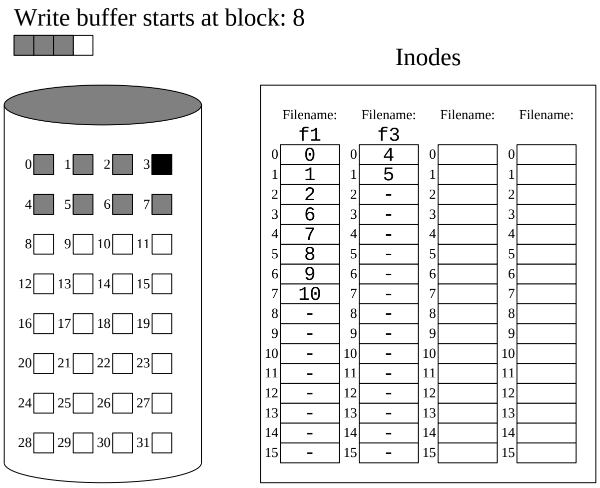
3. Process 2 deletes the file "`f2`".



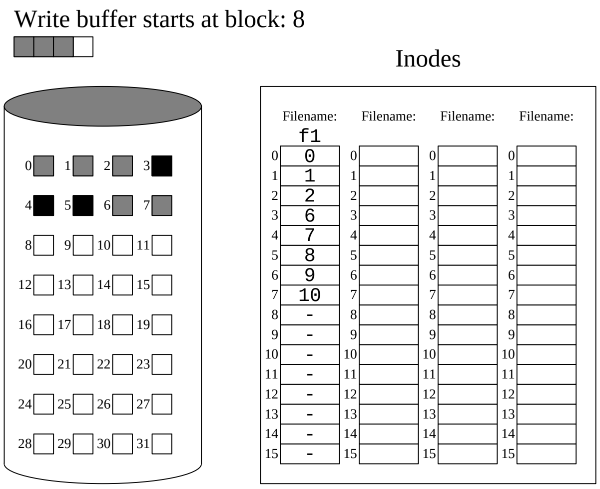
4. Process 3 opens the file "`f3`" and writes blocks 0 and 1.



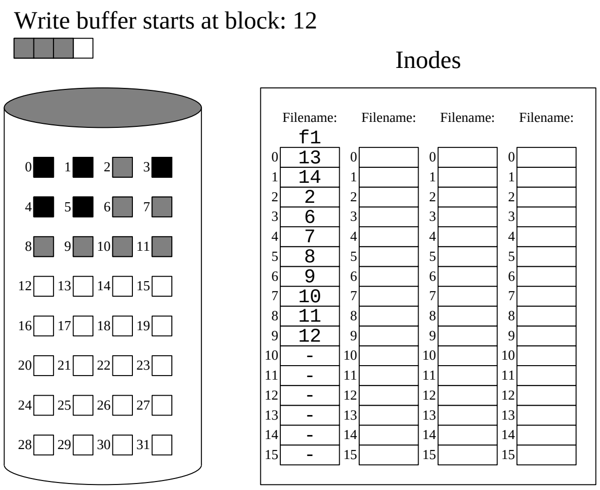
5. Process 1 writes blocks 3 through 7 of "`f1`" and closes it.



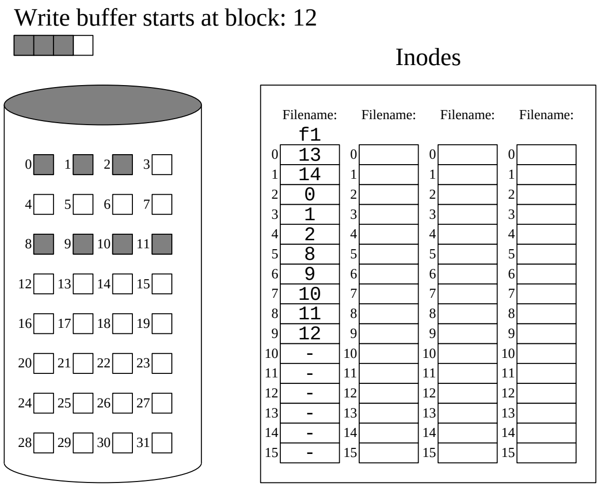
6. Process 3 deletes the file "`f3`".



7. Process 4 opens "`f1`", writes blocks 8 and 9, and overwrites blocks 0 and 1.



8. The garbage collector process runs and compacts disk blocks zero through 7



The following questions relate to log-structured file systems as described above:

Part 1: 6 points

Using the sheets at the end of this paper, show what the state of the system looks like after each of the the following events happen:

- 8. Process 4 overwrites blocks 0, 1, 3, and 4.
- 9. Process 5 creates the file `"f4"` and writes blocks 0 through 7. Then process 4 overwrites block 0 again.
- 10. Process 5 deletes the file `"f4"` and opens the file `"f5"` and writes blocks 0 through 5. Then process 4 overwrites block 0 again.

Part 2: 6 points

We'll define a "normal" file system as follows: Inodes are used to map files to disk. There is no write buffer. A standard free list is used to allocate disk blocks, and when a block is freed, it goes on the end of the free list. When a file overwrites an existing block, the disk block corresponding to that file block is overwritten. Show the state of the system after events 2, 4, 6 and 10 above on a normal file system. Use the sheets at the end of the test to hand this in.

Part 3: 18 points

There are two important things going on in a LFS. One is the use of the write buffer, and the other is how the blocks are written to disk. For most systems, the following statements are true:

- Most files are small.
- Many files are deleted very soon after they are created.
- Most files are written sequentially.

Given these characteristics, explain why a LFS with a *very* large write buffer (on the order of megabytes) should perform much better than a normal file system as far as disk writes are concerned. There are several reasons for this – try to get them all.

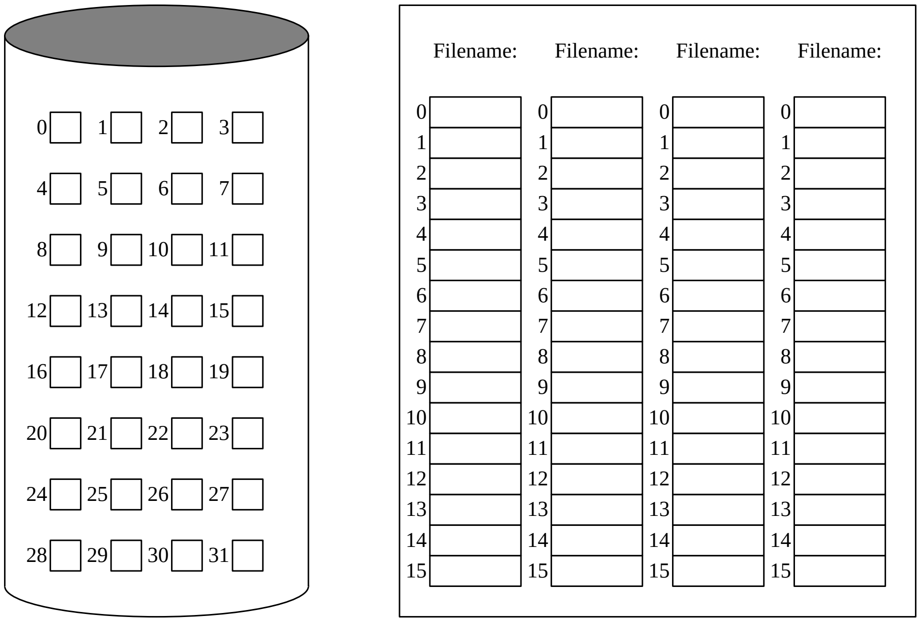
Part 4: 5 points

In most systems, the majority of file reads are either from a small set of files, or from files that have been recently written. Given this, explain how to improve the performance of an LFS.

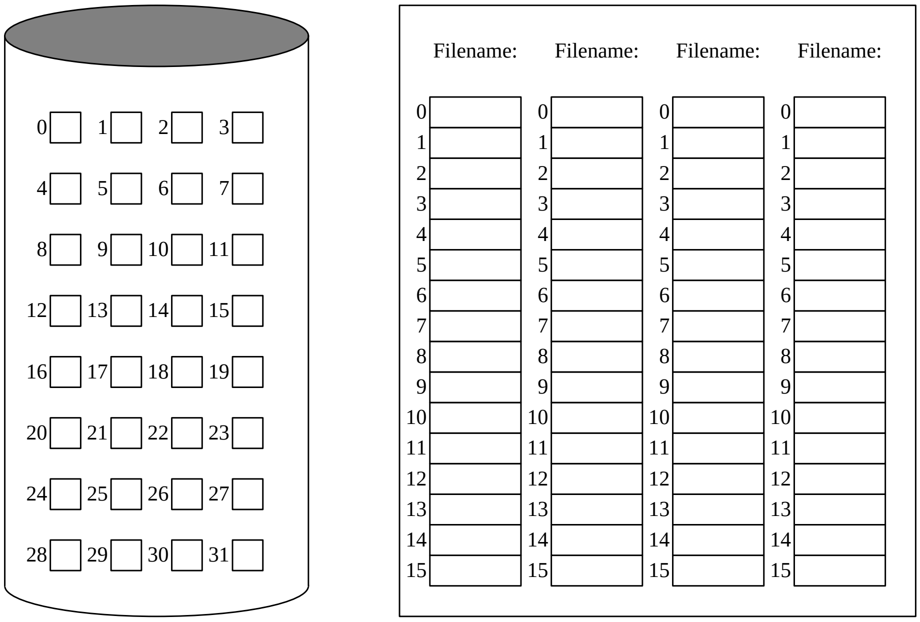
Part 5: 5 points

What kind of disk scheduling would be best for an LFS-based system? Why?

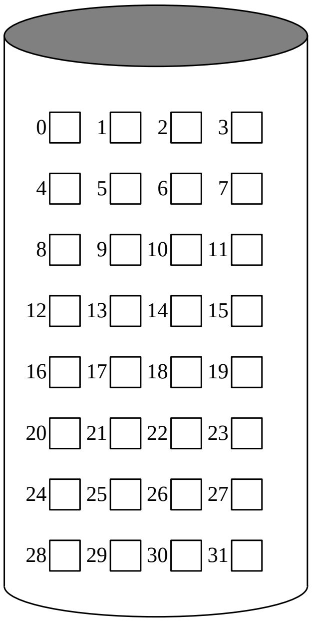
Inodes

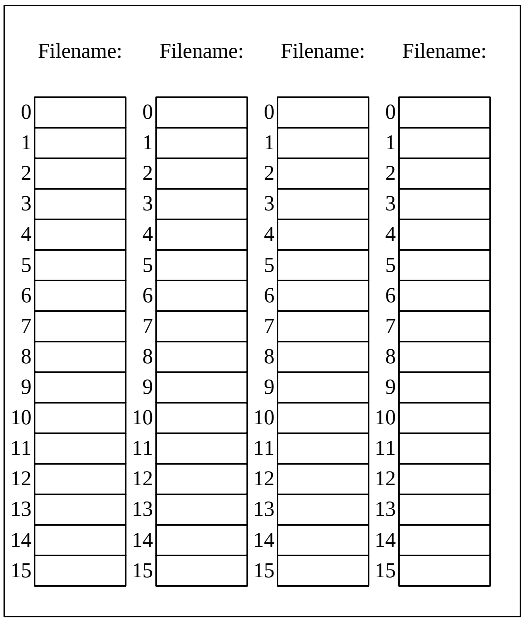


Inodes

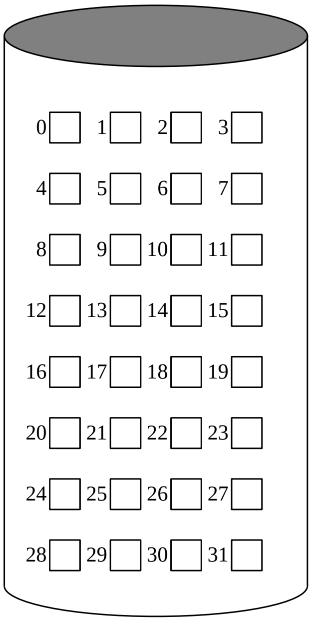


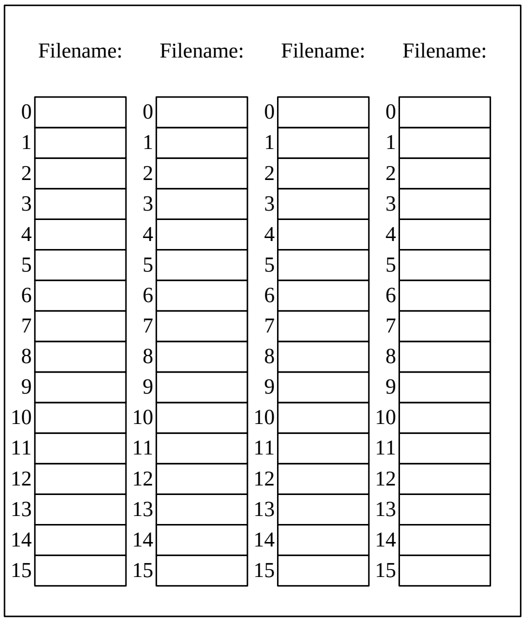
Inodes

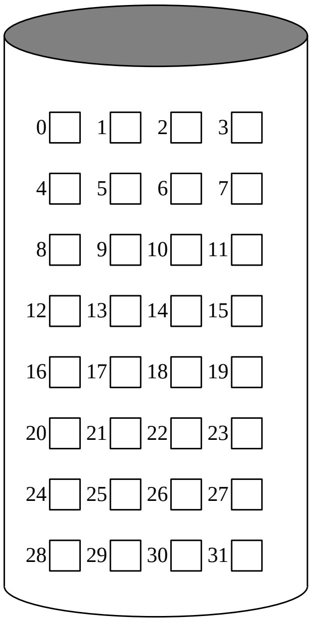




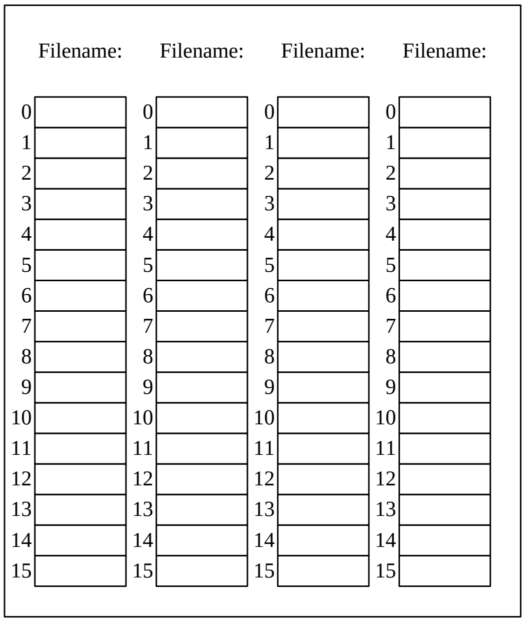
Inodes

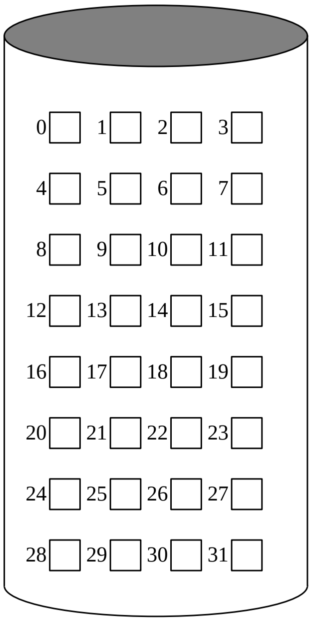




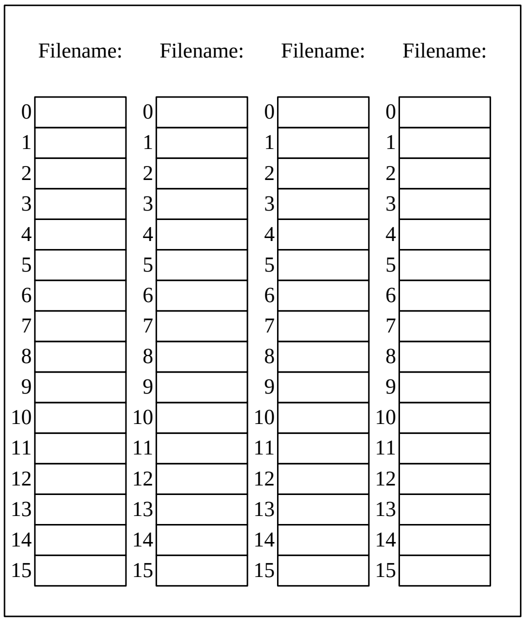


Inodes

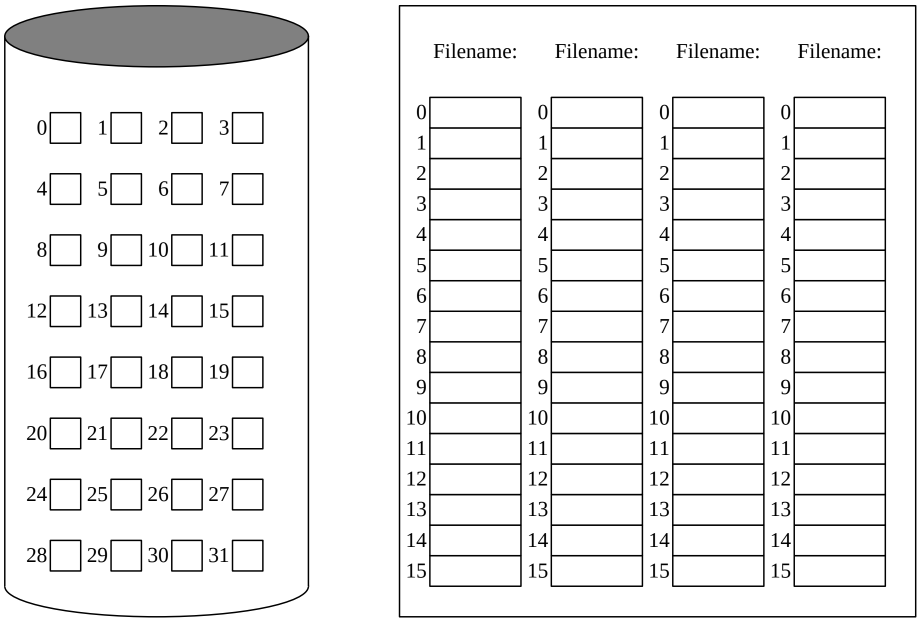




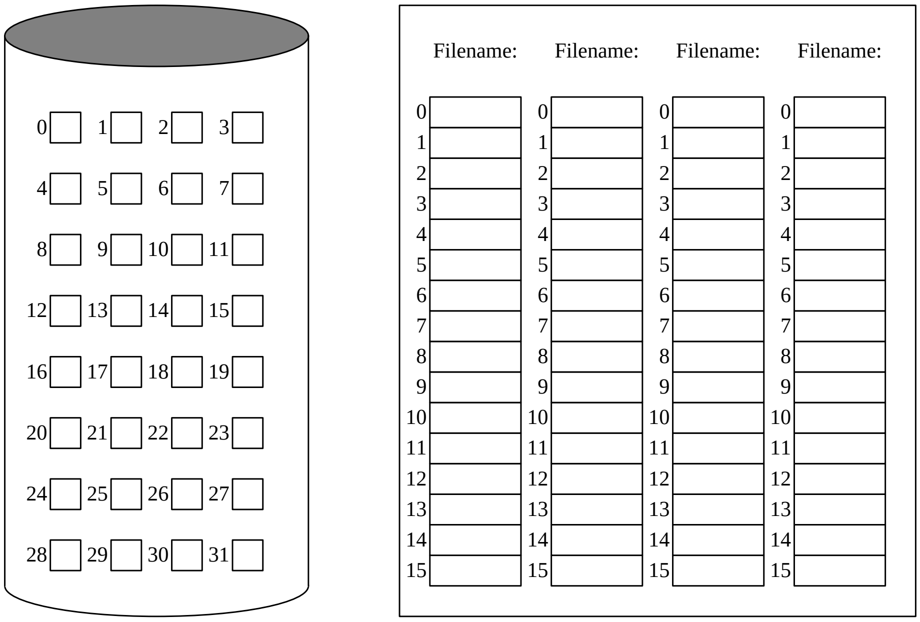
Inodes

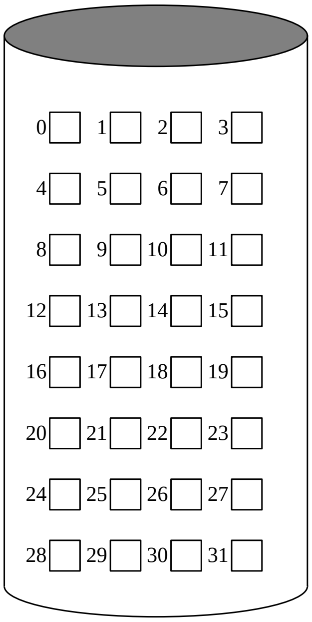


Inodes

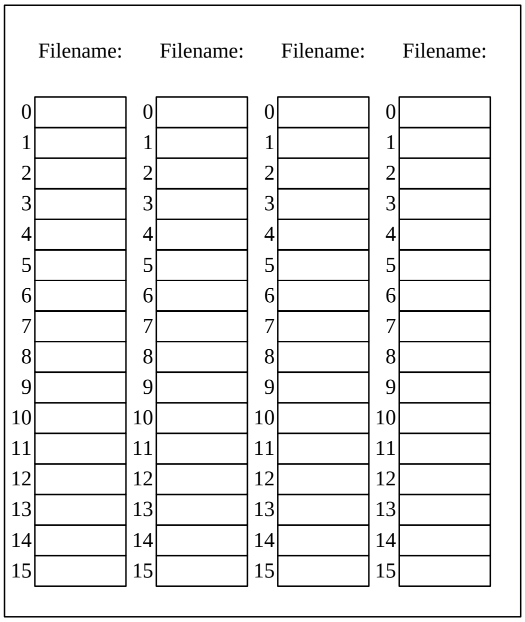


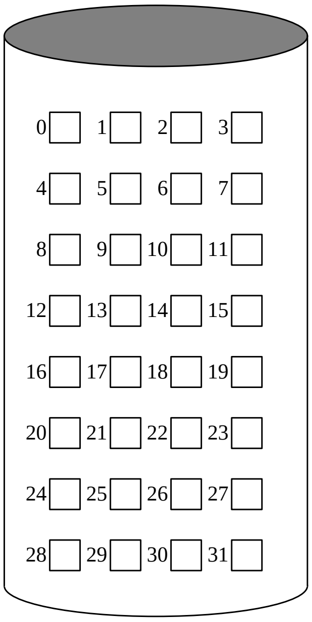
Inodes



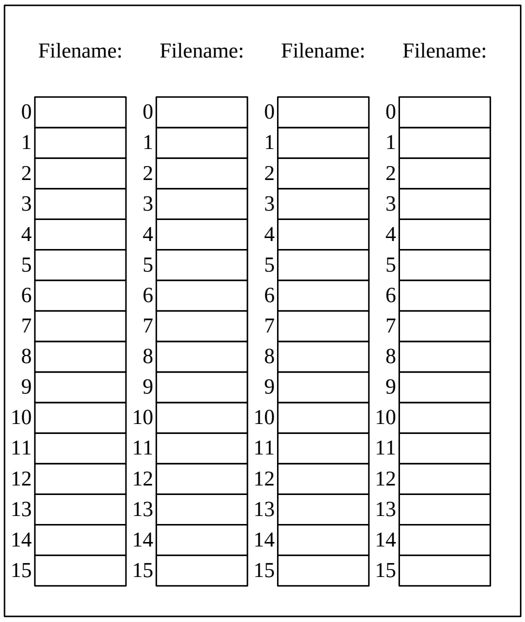


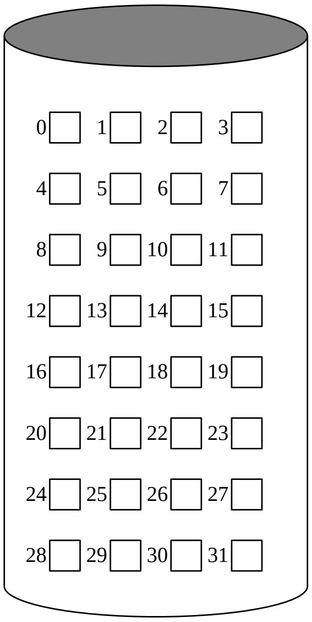
Inodes



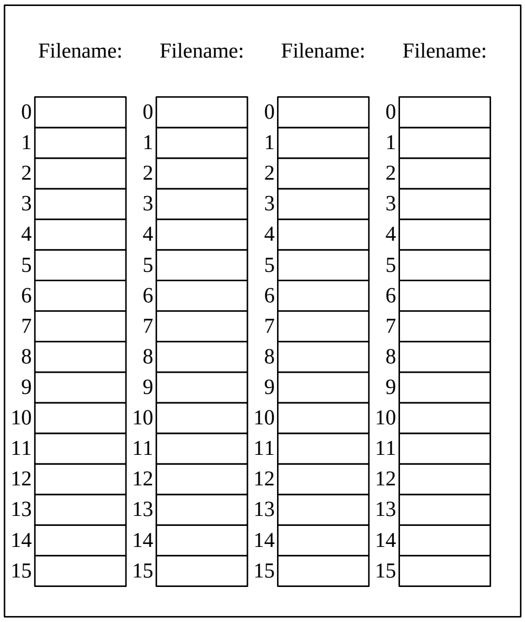


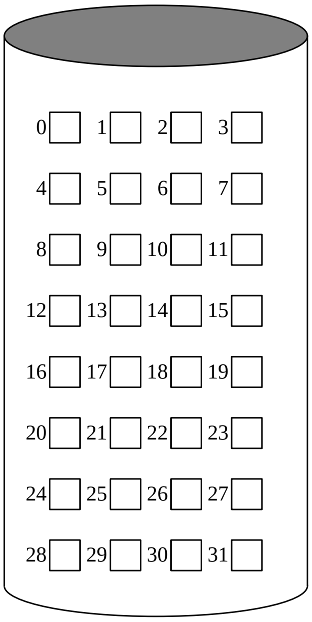
Inodes





Inodes





Inodes