Homework (Simulation)

This section introduces `ffs.py`, a simple FFS simulator you can use to understand better how FFS-based file and directory allocation work. See the README for details on how to run the simulator.

Questions

1. Examine the file `in.largefile`, and then run the simulator with flag `-f in.largefile` and `-L 4`. The latter sets the large-file exception to 4 blocks. What will the resulting allocation look like? Run with `-c` to check.
2. Now run with `-L 30`. What do you expect to see? Once again, turn on `-c` to see if you were right. You can also use `-S` to see exactly which blocks were allocated to the file `/a`.
3. Now we will compute some statistics about the file. The first is something we call *filespan*, which is the max distance between any two data blocks of the file or between the inode and any data block. Calculate the filespan of `/a`. Run `ffs.py -f in.largefile -L 4 -T -c` to see what it is. Do the same with `-L 100`. What difference do you expect in filespan as the large-file exception parameter changes from low values to high values?
4. Now let's look at a new input file, `in.manyfiles`. How do you think the FFS policy will lay these files out across groups? (you can run with `-v` to see what files and directories are created, or just `cat in.manyfiles`). Run the simulator with `-c` to see if you were right.
5. A metric to evaluate FFS is called *dirspan*. This metric calculates the spread of files within a particular directory, specifically the max distance between the inodes and data blocks of all files in the directory and the inode and data block of the directory itself. Run with `in.manyfiles` and the `-T` flag, and calculate the dirspan of the three directories. Run with `-c` to check. How good of a job does FFS do in minimizing dirspan?
6. Now change the size of the inode table per group to 5 (`-I 5`). How do you think this will change the layout of the files? Run with `-c` to see if you were right. How does it affect the dirspan?
7. Which group should FFS place inode of a new directory in? The default (simulator) policy looks for the group with the most free inodes. A different policy looks for a set of groups with the most free inodes. For example, if you run with `-A 2`, when allocating a new directory, the simulator will look at groups in pairs and pick the best pair for the allocation. Run `./ffs.py -f in.manyfiles -I 5 -A 2 -c` to see how allocation changes with this strategy. How does it affect dirspan? Why might this policy be good?
8. One last policy change we will explore relates to file fragmentation. Run `./ffs.py -f in.fragmented -v` and see if you can predict how the files that remain are allocated. Run with `-c` to confirm your answer. What is interesting about the data layout of file `/i`? Why is it problematic?
9. A new policy, which we call *contiguous allocation* (`-C`), tries to ensure that each file is allocated contiguously. Specifically, with `-C n`, the file system tries to ensure that `n` contiguous blocks are free within a group before allocating a block. Run `./ffs.py -f in.fragmented -v -C 2 -c` to see the difference. How does layout change as the parameter passed to `-C` increases? Finally, how does `-C` affect filespan and dirspan?