CS 481: Operating System Principles
Spring '14
Problem Set #2
(due Friday, May 9)

- You **must** typeset your answers and print the results using duplex mode.
- Be concise in your answers but with sufficient detail.
- "Show your work" as appropriate.
- **No late turn-ins will be accepted.**

(From the OSTEP Text): Simulation homeworks come in the form of simulators you run to make sure you understand some piece of the material. The simulators are generally python programs that enable you both to generate different problems (using different random seeds) as well as to have the program solve the problem for you (with the -c flag) so that you can check your answers. Running any simulator with a -h or –help flag will provide with more information as to all the options the simulator gives you.

1 Hard Disks

This homework uses `disk.py` to familiarize you with how a modern hard drive works. It has a lot of different options, and unlike most of the other simulations, has a graphical animator to show you exactly what happens when the disk is in action. See the `README` for details.

1. Compute the seek, rotation, and transfer times for the following sets of requests: -a 0, -a 6, -a 30, -a 7,30,8, and finally -a 10,11,12,13.
2. Do the same requests above, but change the seek rate to different values:-S2,-S4,-S8,-S10,-S40,-S0.1. How do the times change?
3. Do the same requests above, but change the rotation rate: -R 0.1, -R 0.5, -R 0.01. How do the times change?
4. You might have noticed that some request streams would be better served with a policy better than FIFO. For example, with the request stream -a 7,30,8, what order should the requests be processed in? Now run the shortest seek-time first (SSTF) scheduler (-p SSTF) on the same workload; how long should it take (seek, rotation, transfer) for each request to be served?
5. Now do the same thing, but using the shortest access-time first (SATF) scheduler (-p SATF). Does it make any difference for the set of requests as specified by -a 7,30,8? Find a set of requests where SATF does noticeably better than SSTF; what are the conditions for a noticeable difference to arise?
6. You might have noticed that the request stream -a 10,11,12,13 wasnt particularly well handled by the disk. Why is that? Can you introduce a track skew to address this problem (-o skew, where skew is a non-negative integer)? Given the default seek rate, what should the skew be to minimize the total time for this set of requests? What about for different seek rates (e.g., -S 2, -S 4)? In general, could you write a formula to figure out the skew, given the seek rate and sector layout information?

7. Multi-zone disks pack more sectors into the outer tracks. To config- ure this disk in such a way, run with the -z flag. Specifically, try running some requests against a disk run with -z 10,20,30 (the numbers specify the angular space occupied by a sector, per track; in this example, the outer track will be packed with a sector every 10 degrees, the middle track every 20 degrees, and the inner track with a sector every 30 degrees). Run some random requests (e.g., -a -1 -A 5,-1,0, which specifies that random requests should be used via the -a -1 flag and that five requests ranging from 0 to the max be generated), and see if you can compute the seek, rota- tion, and transfer times. Use different random seeds (-s 1, -s 2, etc.). What is the bandwidth (in sectors per unit time) on the outer, middle, and inner tracks?

2 File System Implementation

Use this tool, `vsfs.py`, to study how file system state changes as various operations take place. The file system begins in an empty state, with just a root directory. As the simulation takes place, various operations are performed, thus slowly changing the on-disk state of the file system. See the `README` for details.

1. Run the simulator with some different random seeds (say 17,18,19, 20), and see if you can figure out which operations must have taken place between each state change.
2. Now do the same, using different random seeds (say 21, 22, 23, 24), except run with the -r flag, thus making you guess the state change while being shown the operation. What can you conclude about the inode and data-block allocation algorithms, in terms of which blocks they prefer to allocate?
3. Now reduce the number of data blocks in the file system, to very low numbers (say two), and run the simulator for a hundred or so requests. What types of files end up in the file system in this highly-constrained layout? What types of operations would fail?
4. Now do the same, but with inodes. With very few inodes, what types of operations can succeed? Which will usually fail? What is the final state of the file system likely to be?

3 RAID

This section introduces `raid.py`, a simple RAID simulator you can use to shore up your knowledge of how RAID systems work. See the `README` for details.

1. Use the simulator to perform some basic RAID mapping tests. Run with different levels (0, 1, 4, 5) and see if you can figure out the mappings of a set of requests. For RAID-5, see if you can figure out the difference between left-symmetric and left-asymmetric layouts. Use some different random seeds to generate different problems than above.
2. Do the same as the first problem, but this time vary the chunk size with -C. How does chunk size change the mappings?
3. Do the same as above, but use the -r flag to reverse the nature of each problem.

4. Now use the reverse flag but increase the size of each request with the -S flag. Try specifying sizes of 8k, 12k, and 16k, while varying the RAID level. What happens to the underlying I/O pattern when the size of the request increases? Make sure to try this with the sequential workload too (-W sequential); for what request sizes are RAID-4 and RAID-5 much more I/O efficient? Use the timing mode of the simulator (-t) to estimate the performance of 100 random reads to the RAID, while varying the RAID levels, using 4 disks.

5. Do the same as above, but increase the number of disks. How does the performance of each RAID level scale as the number of disks increases?

6. Do the same as above, but use all writes (-w 100) instead of reads. How does the performance of each RAID level scale now? Can you do a rough estimate of the time it will take to complete the workload of 100 random writes?

7. Run the timing mode one last time, but this time with a sequential workload (-W sequential). How does the performance vary with RAID level, and when doing reads versus writes? How about when varying the size of each request? What size should you write to a RAID when using RAID-4 or RAID-5?