CSE 121 Winter 2005
Homework 2

Due: Thursday, February 10 2005

LFS

1. Explain why LFS performs poorly when re-reading sequential bytes of a large file that were written randomly. How would a journaling file system compare to LFS in this same benchmark?

2. Intuitively explain the exponential curve graphed in figure 3. Why does the write cost approach infinity when u approaches 1?

3. This problem will run through a very simple example of a segment cleaner that operates on the simple greedy policy - "always clean segments with 50% or less utilization". Assume we are currently considering 4 segments, each segment containing 4 blocks. Show how the below 4 segments would appear after cleaning. Assume an number means the block is live and a * means the block is not live.

1	2	*	*	4	5	*	6	7	*	8	*	9	*	10	11

4. The LFS paper states that, "the key to achieving high performance at low cost in a log-structured file system is to force the disk into a bimodal segment distribution ..." Explain why a bimodal distribution is so important.

Journaling & Soft Updates

5. As seen by the Netnews example from the paper, Soft Updates decreases performance when there are a large number of dependency-required rollbacks. Assume the following file system charactistics:

   - 4096 byte block size
   - 512 inode size
   - 512 fixed directory entry size
   - All entries in a single directory block point to inodes in the same block
   - For simplicity, ignore details of indirect pointers and assume that no matter what the file size, a single inode can access all the required data.
   - Infinite file cache
   - Assume both file systems asynchronously execute all data writes then flush data to disk after the writes complete

For the following questions, the exact numeric answer is not as important as the reasoning behind the answer. Please explicity state any assumptions regarding either Soft Updates or Journaling FS made when solving the problem.

(a) Assuming 200 files of size 4096 are created, filled and deleted one after the other all in the same directory, how many total blocks are eventually written to disk in an LFFS-wafs file system (not counting blocks written to the auxiliary file system).

(b) Assuming 200 files of size 4096 are created, filled and deleted, how many block writes occur in the SoftUpdates file system?

(c) Assuming 200 files of size 4194304 (4 megs) are created, filled and deleted, how many block writes eventually occur with the journaling file system?

(d) Assuming 200 files of size 4194304 (4 megs) are created, filled and deleted, how many block writes eventually occur with the Soft Updates file system?

Rio Filecache

6. Assuming power is not an issue (i.e. we have a reliable UPS), briefly explain why RAM is vulnerable during a crash while disk is considered more protected. What part of memory is RIO interested in protecting and what is the basic method taken to protect it?

7. One of the neat things about the Rio paper is their methods of injecting faults. Without giving a list of examples, explain the general difference between low-level faults and high-level faults. Which of the two do you think is more representative of crashes due to device driver faults?

8. What is the difference between direct corruption and indirect corruption? Why do check-sum detections only detect direct corruption as opposed to both direct and indirect corruption?

Caching & Prefetching

9. For this problem we will look at application-controlled prefetching and a pre-fetching policy based on spacial locality. Like in the readings, the application-controlled prefetching will act aggresively and read ahead as soon as possible to the next block that will be accessed. The second prefetching policy will act equally as aggressive, however, for each access it will choose to prefetch the next logically lettered block. For example, if block A is accessed, it will attempt to prefetch block B right away. However, if there is currently a block being pre-fetched, no additional prefetches will be queued up. Make the same assumptions made in the paper: there are 2 blocks of cache, we are using an optimal replacement and the time spent fetching data from memory is 3 time units. Below shows an example where application-controlled pre-fetching and read-ahead prefetching act exactly the same. The access string for this example is : A B C D and we are assuming cache initially has the first two entries - A and B.

Application-controlled pre-fetching

time	1	2	3	4	5	6	7	8
access	A	B			C			D
fetch		C	*	*	D	*	*	
cache0	A	A	X	X	C	C	C	C
cache1	B	B	B	B	B	X	X	D

So the time spent reading blocks A, B, C, D was 8 time units. Answer the following two questions using similar methods as above.

(a) How many time units does the application-controlled pre-fetching strategy take to read the blocks: B,C,A,C assuming cache is initially filled with blocks B and C?

(b) How many time units does it take the read-ahead pre-fetching strategy to read blocks B, C, A, C assuming cache is initially filled with blocks B and C?

10. This question will run through an example of the LRU-SP policy highlighted in the Caching, Prefetching and Disk Scheduling paper. For this question assume that the size of cache is 5 blocks and there are 3 processes running - A, B and C. For readability, a read request is given as a process/block pair. For example, the read request A12 means that process A is trying to read block 12. Finally, assume a basic priority scheme where each processer gives a higher priority to a higher numbered block. For example, if the least recently accessed block is A15, and A has another more recently accessed block in cache A10, A will choose to have A10 evicted instead of A15.

    For this problem we will use the following string of read requests (order is from left to right i.e. A11 is first request): A11, B20, C31, A10, C30, A12, C32, A09, C33, C31, C30, A10

    Assuming an initially empty cache, for EACH read request, give the the contents of cache as well as any placeholders (and specify the cache blocks each placeholder points to). Start showing cache contents after the first 5 requests (since the first 5 trivially fill up cache) that occurs as a result of servicing that request. For example, the following table gives the cache contents after the 5th read request:

Request	Cache4	Cache3	Cache2	Cache1	Cache0
C30	A11	B20	C31	A10	C30