CSE 121 Winter 2006
Home Work 2

Due: Thursday, Feb 16 2006

1. Let's say we are creating a new directory in a file system that uses Soft Updates. In the best case, how many writes are made to disk? In the worst case? What are they? If the file system uses journaling instead, what information would it write to the log?

2. List the operations required by Soft Updates and journaling after a crash. Why are they necessary?

3. The "'Journaling vs. Soft Updates ..."' paper says "'Since a journaling system records a logical operation, such as rename, it will always recover to either the pre-operation or post-operation state. Soft Updates can recover to a state where both old and new names persist after crash."' Show how this can happen using a specific example.

4. Consider the following micro benchmark:

```
// File is already created
fd = open("'file.txt"', O_WRITE);
fstat(fd, &stat_struct);
fileSize = stat_struct.st_size;
while (true) {
   write(fd, buffer, fileSize);
   lseek(fd, 0, SEEK_SET)
}
```

   What would you expect the performance graph (measured in terms of megabytes written per second) to look like as the file size is increased from 1KB to 1GB for each of the Fast File System, Log Structured File System, Soft Updates, and journaling? (Assume that the cache is 512 MB, and LFS has zero cleaning cost.) In addition to drawing the graph, explain why you would expect each system to perform as indicated.

5. LFS provides a bound on the amount of data lost during crashes (it loses at most 30 seconds of data or one segment of data). Is there any bound on the amount of data that can be lost after a crash in the Soft Updates and Journaling approaches? Would the use of NVRAM help?

6. The "Application-controlled File Caching and Prefetching" paper describes four necessary conditions for optimal prefetching. Show, by constructing an example of file access stream, that an algorithm is not optimal if it doesn't satisfy rule 3. (State your assumptions clearly about cache size and the amount of time it takes to read a block)

7. 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 process 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. B25 is first request): B25, C21, C05, B05, B13, C07, B05, A21, C05, A13, C11, C31

   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
B13	B25	C21	C05	B05	B13

8. Give a specific example of priority inversion. Using your example, show how the mutex implementation in lottery scheduling would avoids it.