CSE 121: Operating Systems - Architecture and Implementation
Homework 1
Winter 2007

Due: Thursday, January 25th, in class

1 Operating Systems Review

a) We talked in class about the difference between kernel threads and user threads. In an operating system with kernel threads, the OS maintains both process control blocks (PCBs) and thread control blocks (TCBs). Identify which of the following are stored in PCBs and which must be maintained in TCBs:

- **(TCB)** Stack pointer
- **(PCB)** Page table
- **(PCB)** Open file descriptors
- **(TCB)** CPU context (registers)
- **(PCB)** PID

b) Operating systems typically abstract physical hardware, instead presenting a set of well defined services and interfaces to user applications. Briefly, what are two major advantages of this structure? Conversely, what are two significant drawbacks?

Two advantages include simplified application implementation (they can leave the details of hardware interaction to the OS) and portability (because the application doesn't traffic in the details, the OS can cleanly change the details underneath it). Two disadvantages are possibly restricted performance (because the abstraction inserts a layer of indirection) and the limited choice of implementation (the application has to use the abstractions provided by the OS, even if they are inconvenient.)

2 Fast File System

- The FFS authors claim doubling block size roughly doubled throughput. Of course, this trend wouldn't continue forever. Give two reasons why you'd expect continuing to double the block size to eventually experience declining returns.

At some point, the disk block becomes as large as a cylinder, and a disk seek will be required. Even before then, however, the block size is likely to exceed the length of most files, decreasing efficiency for "small" files.

- Cylinder groups were one of the key contributions of the Fast File System. They do not specify how big they should be, however. Describe the tradeoffs between large and small groups, and suggest a method to select an appropriate size for a given disk and/or file system.

  Small cylinder groups force files and directories to be spread across cylinder groups, defeating the optimization. In contrast, large cylinder groups make seeks within the group expensive, decreasing the effectiveness of the optimization—in the extreme, the file system has one large cylinder group.

3 Log-Structured File System

- Suppose you have only modified one data block of one file since the last checkpoint was written, and LFS is forced to write out a new checkpoint. Describe, precisely, which data structures need to be written to disk, in what order, and where they would be stored.

  In order: new data block, new inode for that file, new imap file, and new segment usage table all go into a new partial segment. Finally, a checkpoint is written to a special region on disk.

- Is it possible for a cleaning operation to to lose, rather than gain free blocks? Why or why not?

  Yes. Consider a segment S containing 1 dead block and 2 data blocks from 2 different files, each of which is accessed from indirect blocks that reside on different segments. If the cleaner cleans segment S it will gain one free block, but lose 2 blocks from invalidating the indirect blocks. (This observation is due to Margo Seltzer, and was shown in her PhD thesis.)

- Why is in that LFS can postpone meta-data writes until a segment is flushed to disk (asynchronous), yet FFS is forced to write meta-data updates synchronously?

  Because FFS does not enforce any ordering on blocks written from the buffer cache, so it cannot ensure that the meta-data blocks are written in a reasonable order unless it writes them synchronously. LFS, on the other hand, orders things in the log so it can ensure that either none of them get written (because the system crashed), all of them get written (in the normal case that all related meta-data operations fit in a single segment), or those that do get written before the crash are either sensible by themselves or rolled back upon reboot.

4 Journaling and Soft Updates

Imagine a news server that needs to store new articles and remove old ones. These operations impose a lot of meta-data updates on the file system. We will analyze the performance of Soft Updates and Journaling FFS under this type of workload. Suppose the machine consists of a 60 GB disk with 8 KB blocks, and 512 MB memory with 4 KB blocks. For simplicity, assume there is no cache. Further, assume that:

- an inode is 128 bytes, indirect and directory blocks are 8 KB
- the journal is 6 GB and treated as a regular file
- don't worry about the periodic checkpoints for either file system
- both file systems are based on FFS with 8 KB blocks.
- for the purposes of analysis, assume no other objects occupy memory

Given a benchmark that does the following:

- traverses 100 different directories, creating a new 8 KB file in each
- removes an old file from each of the the 100 directories

This question is meant to illustrate larger properties of both file systems, so don't worry about the exact paging properties of the system and the exact numbers, just write down your (reasonable) assumptions and the steps to your answers in detail.

a) For each of the Journaling File System and Soft Updates, how many disk accesses will occur?

The directory blocks need to be updated, and inodes need to be allocated and deallocated. All of these meta-data updates should fit in the journal so the Journaling FFS should require no synchronous disk accesses. Similarly, soft updates can delay these meta-data operations for later. Hence, all the writes can be deferred until after the benchmark completes. Afterwards, the journal can be written out in a few large chunks (depending on its implementation); both file systems will eventually need to push out each updated directory, inode, and data block individually.

b) Now suppose the News server deals with larger files. Repeat the benchmark described above assuming files of size 20 MB. For each of the Journaling File System and Soft Updates, how many times will a block need to be written to disk on the series of creates? Deletes?

This is similar to the first benchmark, except the meta-data updates cannot all be delayed. Assume M blocks of the journal and N blocks of the corresponding data need to be written to disk under journaling FFS. In the Soft Updates scheme there are circular dependencies between the directory blocks and deallocated and allocated inodes (from the deletes and creates respectively) in the inode block. Thus for each meta-data update, the disk write will have to be performed twice because of rollback. Thus a total of $2N + M$ blocks will be written to disk. With a 512 MB memory and approximately 2 GB for the data, a reasonable assumption would be that 1.5 GB gets written to disk, or about 192 Kblocks.