Extended model answer

Operating Systems II 2005 – Paper 4 Question 5 (SMH)

This question is on disk scheduling (page 63-65 of notes) and journalled filesystems (pages 83, 92). Some is book work but other parts (partic (c)) require original thought.

SSTF, SCAN, C-SCAN

1 mark each

SSTF is shortest-seek time first; i.e. it always seeks to the closest possible cylinder.

SCAN services requests in order in one direction, and then turns around and services requests in the other direction.

C-SCAN is the circular version of scan; it scans in only one direction (e.g. upward) and then resets to start, not servicing requests on the way.

Basic 2-D scheduling idea

The key idea here is to try to avoid rotational latency as well as just seek latency (which is what traditional algorithms target).

Difficulty implementing 2-D scheduling in OS

Major difficulty here is that the operating system has no idea about the rotational position of the disk.

Implementing 2-D scheduling in an OS

Note: this is rather hard, and so I expect to mark it relatively easily.

The main way one could imagine achieving this is by explicitly building a 'map' of the disk layout – in essence, one would need to seek to every block in the disk from a set of 'reference' markers and determine how long the response time was in each case. This would involve disabling or invalidating the cache between each measurement to ensure real disk parameters are measured. After the set of all distances is computed, one could construct a model which fits the observed data. One would of course also require measurements which measure the effect of the cache since this would be used during operation.

The actual scheduling algorithm then would take as input the past series of disk requests issued and the pending set (or a window of these), and determine the 'closest' request to service modulo a scan-style mechanism to prevent starvation. It could also take into account an estimate of what it believes to be in the disk cache at the current time.

I would expect this to work rather well in practice although the training overhead would be very large. The actual cost/benefit would depend on how simple a model could be used to fit the data and with what error.

Journalling for crash recovery

Journalling basically works by appending entries to the journal before and after you've done a particular filesystem (or db) update. In case of a crash, one can scan the journal and only worry about consistency of those parts of the FS/DB which have 'start' journal entries but no corresponding 'end' ones.

NVRAM journal

This would have large advantages in terms of power saving (one can spin the disk down for relatively long periods of time and aggresivley cache disk blocks in memory without fear of the consequences of failure). It would also aid disk efficiency since frequent seeks to and seeks back from the journal area would be avoided. A disadvantage is the cost of the NVRAM.