Operating Systems II 2004
Paper 4 Question 5 (SMH)
Extended Model Answer

MPTs

Multi-level page tables are n-ary trees of (typically) constant depth between 2 and 5. The top level page table has pointers to k second level page tables, each of which has pointers to k third level ones. Translation proceeds by taking a fixed prefix of the virtual address and using it to index into the root page table. A subsequent prefix is then used to index into the next level, and so on. *Note: this is easier to explain with a diagram.*

LPTs

MPTs of depth d have the problem that every address translation requires d memory accesses (ignoring superpages). LPTs attempt to overcome this by allowing translation in a single memory access in (hopefully) most cases. They achieve this by holding a single linear page table indexed by page frame number in the *virtual* addres space. Hence a single virtual load can obtain the translation for a given page. However there's a new complication: one may now take a TLB miss when accessing the LPT (viz. a nested miss). To solve this, a second page table of depth $d - 1$ is required to map the pages of the LPT. Once a TLB entry for a page of the LPT is present, however, the single memory access translation case works again.

IPTs

Note: only one of the two problems below need be mentioned.

As with LPTs, IPTs attempt to overcome the d memory accesses required by MPTs. IPTs also handle address space sparsity better. This is acheived by having a single table of entries for each *physical frame* in memory rather than for each page. Translation occurs by hashing the virtual address (well a prefix of it) using the result to index into the IPT. The physical frame corresponding to page number p is $f = h(p)$.

HPTs

IPTs do not allow the operating system any choice about which frame to allocate for which page; the decision is immplicit in the hash function. HPTs overcome this by

storing a full PTE in the hash table rather than simply the page number. Hence the OS can now choose whichever frames it wishes to map to whichever pages.

Buffer Cache vs. Cache Manager

Approx 4 marks for each; the level of detail expected is less than that given below.

The BSD buffer cache operates on a block level; every time a filesystem wishes to access a block, it specifies the device number and the block number on that device. The buffer cache hashes these to determine if the block is already cached. If not, it issues a read from disk and suspends the process until this is complete. Clearly the buffer cache is not of unlimited size, and hence blocks must be replaced when it becomes full. This is done in a LRU manner – every buffer is chained to a LRU list, and pulled to front when accessed. Buffers are replaced from the tail. If a buffer has been modified (i.e. is 'dirty') then replacement requires a write back to disk. Periodicially, dirty blocks are flused to disk anyway, and then marked again as 'clean'.

The W2K cache manager, on the other hand, caches parts of *files*; i.e. when a filesystem wishes to access part of a file, it checks the cache manager first to see if that portion of the file is currently cached. If not, it translates the logical file addresses into block reads, and then updates the cache manager. The cache manager actually caches in the virtual address space in terms of 256K contiguous regions of a file. Hence even something wihch is 'cached' may not actually be resident; if the cache manager services a part of the file from the cache which is not resident, a page fault occurs. The resolution of this is what actually causes data to be loaded in from disk. Hence the cache manager is actually completely unified with the virtual memory system.

The key similarities between these two schemes is that they are attempting to solve the same problem. Key differences are (a) BSD is not unified while W2K is, (b) a hit in the cache manager is even cheaper than a hit in the buffer cache since no metadata translation need occur, and (c) the extra level of indirection in the cache manager makes it more generic and less fs/device number specific. The cache manager also exports a more flexible interface to file system users in terms of synchronous writeback and temporary files.