Homework #2 Solutions

June 27th, 2004

1. Disk scheduling

(a) FCFS: |78 – 50| + |56 – 78| + |12 – 56| + |1 – 12| + |78 – 1| + |45 – 78| + |25 – 45| + |78 – 25|
(b) SSTF: |45 – 50| + |56 – 45| + |78 – 56| + |78 – 78| + |78 – 78| + |25 – 78| + |12 – 25| + |1 – 12|
(c) SCAN: |45 – 50| + |25 – 45| + |12 – 25| + |1 – 12| + |0 – 1| + |56 – 0| + |78 – 56| + |78 – 78| + |78 – 78|
(d) C-LOOK: |56 – 50| + |78 – 56| + |78 – 78| + |78 – 78| + |1 – 78| + |12 – 1| + |25 – 12| + |45 – 25|

2. The rational of anticipatory scheduling.

The disk requests from the same processes are likely close to one another. Those from different processes are likely in different areas of the disk. Switching from one process’s requests to those from a different process causes disk head movements, which are very slow. Asynchronous requests can be issued continuously. Synchronous requests are scattered over time. After serving a synchronous requests, it is worth waiting for a shore while for the same process to issue following requests, in hope to avoid moving read/write heads. The huge overheads of moving read/write heads outweigh the short waiting times.

3. See below

- (Y) Looking up interrupt vector is done by hardware
- (N) Processor follows the instructions that are parts of the OS to store registers to PCB.
- (Y) Jump to a memory location upon RESET: by hardware
- (N) Processor follows the instructions that are part of the BIOS to load MBR.
- (N) Processor follows the instructions that are part of the BIOS to check memory.
- (Y) No software is involved in TLB lookup.
- (Y) No software is involved updating TLB entries
- (N) Page tables are updated by instructions that are part of the OS.
4. B-tree sizes.

(a) min # of keys at level 1: 1
    min # of keys at level 2: 2x100
    min # of keys at level 3: 2x101x100
    min # of keys at level 4: 2x101^2x100
(b) the sum of the above four levels.
(c) max # of keys at level 1: 200
    max # of keys at level 2: 201x200
    max # of keys at level 3: 201^2x200
    max # of keys at level 4: 201^3x200
(d) the sum of the above

5. Each Reiser key is 16-byte long, and a pointer 4 bytes. Where there are \( x \) keys in a node, we have \( 16x + 4(x + 1) \) bytes. Since a node must fit in an 8K block, we have

\[
20x + 4 \leq 8 \times 1024 \implies x \leq 409.4
\]

Notice that the maximum number of keys in a B-tree node must be an even integer. We have \( \text{MAX}=408 \) and \( \text{MIN}=204 \).

6. See below

(a) \( 4K \times 10 = 40960 \) bytes
(b) An indirect block contains 1024 pointers. We have \( 40960 + 1024 \times 4096 = 40960 + 2^{22} \) bytes (about 4 mega bytes).
(c) The first level indirect table points to 1024 second level indirect tables, each of which points 1024 data blocks. We have \( 40960 + 2^{22} + 1024^2 \times 4096 = 40960 + 2^{22} + 2^{32} \) bytes (about 4 giga bytes).
(d) FYI, I next compute the maximum file size allowed by the Unix file system. The first level indirect table points to 1024 second level tables, each of which points 1024 third-level tables, each of which in turn points to 1024 data blocks. We have \( 40960 + 2^{22} + 2^{32} + 1024^3 \times 4096 = 40960 + 2^{22} + 2^{32} + 2^{42} \) bytes (about 4 tera bytes).

7. See below

- (Y) to update the bit map
- (N) Large files do not use direct items
- (Y) indirect items are used by large files to point to unformatted data blocks
- (Y) changes to the B-tree structure are always journaled
- (N) mp3 header is just data
• (N) mp3 encodings are also data

8. Initialization: `semop(S, 1);

do {
    `semop (S, -1);
    Critical Section
    `semop (S, 1);
    Remainder Section
} while (1);

9. Please see the textbook or handouts for the details of the Bakery algorithm.

Consider $N$ processes trying to enter a critical section repeatedly and consider a particular process $P_i$. After the $P_i$ gets its ticket number, processes that have obtained smaller numbers will have priority over $P_i$. We don’t have to worry about those processes with larger numbers, for they will lose to $P_i$. Therefore we focus on a process $P_a$ who won the first round. $P_a$ will enter the critical section while $P_i$ is waiting. Let us consider what happens when $P_a$ leaves the critical section and returns to draw the ticket a second time. If $P_a$ obtains a larger ticket number, then $P_a$ nows waits after $P_i$. If $P_a$ obtains the same ticket number as $P_i$, then in executing the for loop it has to wait for $P_i$ (notice the ticket comparison in the while loop is $a < \text{not} \leq$; thus $P_a$ will have to wait even if it is a tie). On the other side, $P_i$ had once waited for $P_a$ and since then the integer $j$ in its for loop has grown greater than $a$. $P_i$ therefore will not wait for $P_j$ a second time. In summary, everyone ahead of $P_i$ in the first round will have to wait behind $P_i$ when it tries to enter the critical section a second time. This ensures that $P_i$ will not wait indefinitely.