

CSC 262

Homework 2

Due Sep. 25, 2008

1. **Checksums and Error Detection** Because disk drives are subject to transient errors, values can be mis-read. These errors have to be detected and corrected (most of this happens in drive circuitry now, thankfully). For these problems we'll employ the simple checksum algorithm described in class. Recall that the checksum byte is simply the sum of all the data bytes modulo 256 ($B_E = \sum_{i=0}^n B_i \text{ mod } 256$).

- (a) What is the checksum byte for the values: 0x12, 0x45, 0xA, 0xFF, 0x19)?

$$0x12 + 0x45 + 0xA + 0xFF + 0x19 = 0x179 \text{ The checksum is } 0x79$$

- (b) Suppose there was a read error on the first byte and it was read as 0x13 (rather than 0x12). Now suppose there was a read error on the second byte as well (and no other subsequent bytes). What value would the second mis-read have to produce in order to still pass the checksum (i.e. the checksum would not detect this error).

If we treat the error as an addition, e.g. $0x12 + x = 0x13$, then we need to find another value, y added to 0x45 such that $x + y = 0$. Therefore, if the second read produces 0x44, the checksum will not change.

- (c) If, in a stream of N bytes (where $N > 2$), there are exactly two read errors. What is the probability that those errors will go undetected?

As the previous problem illustrated, you can model a read error of a byte x as $x + \Delta_x$, and for any given Δ_x there is exactly one other value that will cancel it out (e.g. produce 0 mod 256). Therefore for any given read error, there is only one out of 256 possible other read errors that will not be detected by our checksum. Therefore the probability is $\frac{1}{256}$

2. **Disk Scheduling** For this problem you will be performing different disk scheduling routines. Assume that the disk has 64 tracks (0-63), and that the read/write heads begin on track 0. The table below lists the tracks requested and the time at which they were requested. Assume that the request buffer starts empty. Also assume that all seeks take 12ms (this simplifies your task). You will emulate several of the algorithms discussed in class. If a request arrives before the current request is finished, then it is buffered. If more than one request are in the buffer then you must make scheduling decisions in accordance with the policy of the algorithm.

Track Requested	15	13	57	20	0	30	41	9	61	11
Time of Request (ms)	0	3	7	12	15	29	31	37	51	73

For each algorithm list the order of tracks serviced, the latency (the time between the request and when it was serviced), and the number of tracks traversed. *HINT: just work through*

each algorithm step-by-step. When done processing a request, figure out how long it took, and which requests are outstanding. Then use the policy to select the next request.

(a) **FIFO**

Track	Time	Queue	# tracks	Latency
0 → 15	12	13, 57, 20	15	15: 12ms
15 → 13	24	57, 20, 0	2	13: 21ms
13 → 57	36	20, 0, 30, 41	44	57: 29ms
57 → 20	48	0, 30, 41, 9	37	20: 36ms
20 → 0	60	30, 41, 9, 61	20	0: 45ms
0 → 30	72	41, 9, 61	30	30: 43ms
30 → 41	84	9, 61, 11	11	41: 53ms
41 → 9	96	61, 11	32	9: 59ms
9 → 61	108	11	52	61: 57ms
61 → 11	120	∅	50	11: 47ms
Total:			393	

(b) **SSTF**

Track	Time	Queue	# tracks	Latency
0 → 15	12	13, 57, 20	15	15: 12ms
15 → 13	24	57, 20, 0	2	13: 21ms
13 → 20	36	57, 0, 30, 41	7	57: 55ms
20 → 30	48	57, 0, 41, 9	10	20: 24ms
30 → 41	60	57, 0, 9, 61	11	0: 105ms
41 → 57	72	0, 9, 61	16	30: 19ms
57 → 61	84	0, 9, 11	4	41: 29ms
61 → 11	96	0, 9	50	9: 71ms
11 → 9	108	0	2	61: 33ms
9 → 0	120	∅	9	11: 23ms
Total:			126	

(c) **LOOK**

Track	Time	Queue	# tracks	Latency
0 → 15	12	13, 57, 20	15	15: 12ms
15 → 20	24	13, 57, 0	5	13: 69ms
20 → 57	36	13, 0, 30, 41	37	57: 29ms
57 → 41	48	13, 0, 30, 9	16	20: 12ms
41 → 30	60	13, 0, 9, 61	11	0: 81ms
30 → 13	72	0, 9, 61	17	30: 31ms
13 → 9	84	0, 61, 11	4	41: 17ms
9 → 0	96	61, 11	9	9: 47ms
0 → 11	108	61	11	61: 69ms
11 → 61	120	∅	50	11: 35ms
Total:			175	

(d) **C-LOOK, increasing track#**

Track	Time	Queue	# tracks	Latency
0 → 15	12	13, 57, 20	15	15: 12ms
15 → 20	24	13, 57, 0	5	13: 69ms
20 → 57	36	13, 0, 30, 41	37	57: 29ms
57 → 0	48	13, 30, 41, 9	57	20: 12ms
0 → 9	60	13, 30, 41, 61	9	0: 33ms
9 → 13	71	30, 41, 61	4	30: 55ms
13 → 30	84	41, 61, 11	17	41: 65ms
30 → 41	96	61, 11	11	9: 23ms
41 → 61	108	11	20	61: 57ms
61 → 11	120	∅	50	11: 47ms
Total:			225	

(e) **C-LOOK, decreasing track#**

Track	Time	Queue	# tracks	Latency
0 → 15	12	13, 57, 20	15	15: 12ms
15 → 13	24	57, 20, 0	2	13: 21ms
13 → 0	36	57, 20, 30, 41	13	57: 41ms
0 → 57	48	20, 30, 41, 9	57	20: 72ms
57 → 41	60	20, 30, 9, 61	16	0: 21ms
41 → 30	72	20, 9, 61	11	30: 43ms
30 → 20	84	9, 61, 11	10	41: 29ms
20 → 11	96	9, 61	9	9: 71ms
11 → 9	108	61	2	61: 69ms
9 → 61	120	∅	52	11: 23ms
Total:			187	

(f) **F-LOOK** For F-LOOK I am assuming a queue of length 8, subdivided into two length 4 queues.

Track	Time	Processing Queue	Adding Queue	# tracks	Latency
0 → 15	12	13, 57, 20	∅	15	15: 12ms
15 → 20	24	13, 57	0	5	13: 45ms
20 → 57	36	13	0, 30, 41	37	57: 29
57 → 13	48	∅	0, 30, 41, 9	44	20: 12ms
13 → 9	60	0, 30, 41	61	4	0: 57ms
9 → 0	72	30, 41	61	9	30: 55ms
0 → 30	84	41	61, 11	30	41: 65ms
30 → 41	96	∅	61, 11	11	9: 23ms
41 → 61	108	11	∅	20	61: 57ms
61 → 11	120	∅	∅	50	11: 47ms
Total:				225	

(g) Which algorithm(s) took the shortest time?

(h) Which algorithm(s) had the lowest average latency?

both the questions are redundant. The answer is all the algorithms took the same amount of time

- (i) Which algorithm(s) had the highest latency?
SSTF, with an astounding 105ms for track 0
- (j) Which algorithm(s) traversed the most tracks?
FIFO, traversing a total of 393 tracks