Operating System (CSE231)

Section-B

Quiz 4

Total Marks: 40 **Time: 1 hr**

---

1. (12 points) You are given the following code that inspects a file "data.txt" using system calls. Assuming that the file "data.txt" contains the content:

   ABCDEFGHIJKL

   Answer the following questions.

```c
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <dirent.h>
#include <assert.h>
#include <string.h>

int main() {

    int fd1 = open("data.txt", O_RDWR);
    assert(fd1 >= 0);
    int fd2 = open("data.txt", O_RDONLY);
    assert(fd2 >= 0);
    lseek(fd1, 5, SEEK_SET);
    char buf1[6];
    char buf2[6];
    read(fd1, buf1, 5);
    buf1[5] = '\0';
    read(fd2, buf2, 5);
    buf2[5] = '\0';
    printf("fd1 read: %s\n", buf1); // A
    printf("fd2 read: %s\n", buf2); // B

    // C

    return 0;
}
```

   (a) (4 points) What would be the outputs of code line //A and //B? Explain your answer.

   (b) (4 points) Insert code at code line //C to force synchronize fd1. Also, verify whether the operation was successful.

   (c) (4 points) In UNIX operating systems, which data structure represents a directory entry and is returned by function calls like `readdir()`? Also, what is an inode number?

   Ans.

   (a) fd1 read: FGHIJ

fd2 read: ABCDE

Explanation: After opening the file twice, each file descriptor receives its own open file table entry with its own independent current offset. Each open() creates a new entry with offset = 0.

fd1 seeks to offset 5, then reads 5 bytes, "FGHIJ". fd2 is still at offset 0, so it reads "ABCDE".

Grading guide: 1 mark each for correct output. 1 mark each for correct explanation.

(b)

```
int rc = write(fd1, buf1, 5);
assert(rc == 5);
rc = fsync(fd1);
asset(rc == 0);
```

Grading guide: 2 marks for mentioning fsync(fd1). 2 more marks for complete correct code.

(c) The `dirent` structure. The inode number is the index node which is an OS-level identifier.

Grading guide: 2 marks for each correct answer.

2. (20 points) A disk runs at 7200 RPM, and the seek time per track is 8 ms. The disk has 600 tracks, each track has 120 sectors, numbered 0–119. The **SCAN (Elevator) scheduling algorithm** is used. The disk head is initially positioned at track 320, sector 40, moving toward the higher-numbered tracks (toward track 600). The disk rotates from the lower-numbered sector to the higher-numbered sector. The following disk I/O requests arrive and must be scheduled:

Track	**Sector**
500	60
120	30
320	10
410	80
250	90

   (a) (3 points) Determine the order in which the requests will be serviced using the SCAN algorithm.

   (b) (15 points) For each request in that order, calculate the Seek Time, Rotational Latency/Delay, and the Total access time (this requires rotational latency per sector). Show your calculations.

   (c) (2 points) What issue might arise if a scheduler optimizes only for rotational latency and not for seek time?

Ans. Seek time = 8 ms
RPS = RPM/60 = 7200/60 = 120 revolutions/sec
Time for one rotation = 1/RPS = 1/120 = 8.3333 ms

Avg. rotational latency = revolution time/2 = 4.1667 ms
Rotational latency/sector = revolution time/sectors = 8.3333 ms/120 = 0.069444 ms/sector

(a) Given: must use Elevator SCAN moving upwards, therefore

SCAN will service requests at tracks greater than or equal to 320 in increasing track order, then reverse and service remaining lower-track requests in decreasing track order.

Requests with track greater than or equal to 320: tracks 320, 410, 500 and then reverse direction and serve lower tracks in decreasing order: remaining tracks 250 and 120.

Final service order:

(320, 10)
(410, 80)
(500, 60)
(250, 90)
(120, 30)

Grading guide: 3 marks for correct order. No partial marking.

(b) **Request 1:** (320, 10) starting from (320, 40)
Seek distance: |320 - 320| = 0
Seek time: 0 × 8 ms = 0.000 ms
Rotational latency: moving from sector 40 to 10. Since target < current and the rotation cannot go backwards, a full rotation is required (10+120), minus the current sector (-40). Therefore, forwardSteps = 10 + 120 - 40 = 90 sectors
Then, rotational latency = 90 × 0.069444 ms = 6.250 ms
Total time = Seek + Rotational = 0 + 6.250 = 6.250 ms

Request 2: (410, 80)
Seek distance: |410 - 320| = 90 tracks
Seek time = 90 × 8 ms = 720.000 ms
Rotational latency: sector 10 to 80
target sector > current sector, therefore: forwardSteps = 80 - 10 = 70 sectors
Rotational latency = 70 × 0.069444 ms = 4.861 ms
Total time = 720.000 + 4.861 = 724.861 ms

Request 3: (500, 60)
Seek distance: |500 - 410| = 90 tracks
Seek time = 90 × 8 ms = 720.000 ms
Rotational latency: sector 80 to 60
target < current, so forwardSteps = 60 + 120 - 80 = 100 sectors
Rotational latency = 100 × 0.069444 ms = 6.944 ms
Total time = 720.000 + 6.944 = 726.944 ms

Request 4: (250, 90) after reversing direction: track 500 to 250, sector 60 to 90
Seek distance: |250 - 500| = 250 tracks
Seek time = 250 × 8 ms = 2000.000 ms
Rotational latency: sector 60 to 90
forwardSteps = 90 - 60 = 30 sectors
Rotational latency = 30 × 0.069444 ms = 2.083 ms
Total time = 2000.000 + 2.083 = 2002.083 ms

Request 5: (120, 30)
Seek distance: |120 - 250| = 130 tracks
Seek time = 130 × 8 ms = 1040.000 ms
Rotational latency: sector 90 to 30
target < current, so forwardSteps = 30 + 120 - 90 = 60 sectors
Rotational latency = 60 × 0.069444 ms = 4.167 ms
Total time = 1040.000 + 4.167 = 1044.167 ms

Grading guide: For each request, 1 mark each for seek time, rotational latency, and total time.

(c) The scheduler might repeatedly prefer requests on the same track or nearby sectors, causing **excessive head movement (long seeks) or large average seek times and degraded throughput**. In the extreme, optimizing purely for rotational latency could cause **starvation** of requests far from the current head position because the scheduler always picks ones with minimal rotational wait.

Grading guide: 1 mark for reference to longer seek times, and 1 mark for starvation.

3. (8 points) A filesystem uses VSFS-style layout with 4 KB block size.
   (a) (4 points) Each inode is 256 bytes and the inode table start address is 0 B. The disk sector size is 512 B. What is the block number and disk sector number that contain inode number 5000? Assume 1 KB = 1000 B.
   (b) (4 points) Given block size 4 KB with 4-byte pointers, (i) what is the size of a file that uses 16 direct pointers and 200 pointers in a single indirect block? (ii) What level of indirection would be required for a file of size 4.2 TB?

Ans. (a) Inode 5000's relative position = 5000*256 = 1280000 Block number = 1280000/4k = 1280000/4000 = 320 Sector Number = (1280000 + 0)/512 = 2500

Grading guide: 1 mark for correct relative position. 1 mark for correct block number. 2 marks for correct sector number.

(b) (i) (16 + 200) x 4000 B = 864000 B = 864 KB (ii) Triple indirection.

Grading guide: 2 marks for each part.