...complete.

①

	P1	P3	P5	P7	P8
22	I/O waiting	Waiting I/O	Ready	I/O Waiting	ready
37	ready	ready	Suspended	I/O waiting	executing
47	ready	ready	Ready	I/O waiting	exited

12

② Assuming on an average page-fault rate although each algorithm works differently in different scenarios. Therefore ranking relatively.

a) LRU: 2, because it is a descent algorithm which replaces according to least recently used but can suffer from page faults in some scenarios. It can also suffer from Belady's anomaly.

b) FIFO: 4, it can suffer from large page faults in many scenarios like simple one: Frame size = 3
Request: 1 2 3 4 1 2 3 4 . . .

c) Optimal: 1, it is perfect algorithm with minimal page faults.

d) Second-Chance: 3, It is approximation to LRU.

How about Belady's anomaly ?? 7

3

Frames →	1	2	3	4	5	6	7
LRU	20	18	16	10	8	7	7
FIFO	20	18	16	14	10	10	7
Optimal	20	~~15~~ 15	11	8	~~8~~ 7	~~8~~ 7	7

12

(4) a) Yes, it can happen when foo() execute semWait(S) and bar() execute semWait(R) simultaneously then ~~B~~ foo() will be waiting for semWait(R) and bar() will be waiting for semWait(S). ~~and~~ Therefore they result in being blocked forever.

12

b) No, because there will come a time when one of them gave a signal (semSignal) and other one will take semWait.

Assuming: It can be possible that execution of one process result in some postponement but for the indefinite postponement it has very very less probability (≃0). Otherwise it can be possible if remains continue before the other one

(5) a) FCFS: 0 + 11 + 63 + 86 + 74 + 55 + 66 + 87 = 442

b) SSTF: 26 → 33 → 37 → 14 → 12 → 88 → 99 → 100
0 + 7 + 4 + 23 + 2 + 76 + 11 + 1 = 124

c) SCAN: 26 → 33 → 37 → 88 → 99 → 100 → 14 → 12
0 + 7 + 4 + 51 + 11 + 1 + 86 + 2 = 162

d) C-SCAN: 26 → 33 → 37 → 88 → 99 → 100 → 0 → 12 → 14
0 + 7 + 4 + 51 + 11 + 1 + ~~100~~ + 12 + 2 = 188

jump does not count

Assuming max = 100

10

(6) a) Transfer time = time required for transferring the data after access.

1 Revolution will transfer = 512 × 400 bytes
= 204800 bytes

Revolutions required = $\frac{1048576}{204800}$ = 5.12

Transfer time = $\frac{5.12}{15000} \times \underbrace{(60 \times 1000)}_{\text{minute to ms}}$ = 20.48 ms ✓

Total time = avg. access time + transfer time = 26.48 ms

b) Average access time = Avg. seek time + Average Rotation latency

$\downarrow$ 4 ms

$\downarrow$ $\frac{1}{2} \times \frac{60 \times 1000}{15000}$ = 2 ms
in average

= 6 ms. ✗

4

c) Rotation delay per rotation = $\frac{60000}{15000}$ = 4 ms on avg. 180° i.e. 2 ms

For file transfer of 1MB = 20.48 ms delay is due to ~~file transfer~~ rotation.

d) Time to read one sector = $\frac{1}{400} \times 4$ = 0.01 ms ✗

e) Total time to read one track = 1 × 4 ms = 4 ms ✗

(7) 13-direct pointers ⇒ $13 \times 8 KB = 104 KB$

1 indirect pointer ⇒ will point to a block of 8 KB which will have ~~2048~~ $\frac{8 \times 1024}{4}$ $\left(\frac{\text{block size (in bytes)}}{\text{pointer size (in bytes)}}\right)$

$= 2 \times 1024$ pointers to ~~B~~ block of 8 KB
direct

⇒ 16 MB

1 double indirect pointer: $\underbrace{\frac{8 \times 1024}{4}}_{\text{No. of entries of indirect pointer}} \times \underbrace{16 MB}_{\text{One indirect pointer can handle}}$

12

$= 32 GB$

1 triple indirect pointer: $\underbrace{\frac{8 \times 1024}{4}}_{\text{No. of entries of double indirect pointer}} \times 32 GB = 64 TB$

Total size $= 64 TB + 32 GB + 16 MB + 104 KB$

(8) Solution: 1. Limit the no. of connections to 1 with a single party, can be one of its solution.

2. Record the challenges that are under process, means sent to other connections and authentication of them is pending and then if a challenge is received respond to that challenge only if it is not present in current records.

3. If it is possible to receive and send challenges in different ways then do that so that some challenge cann't be sent back.

16