CS 5348 Operating Systems, Homework #3

1. Consider the page reference sequence for a program P: 0 1 2 3 0 1 2 4 0 1 2 5 3 5 6 3. Load the pages following the page replacement policies given below. Also, compute the number of page faults incurred in each policy. Assume that the working set size is 4.
   (a) LRU

After 0 1 2 3, 4 pages faults already (left most is the least recently used page)

```
0 1 2 3  | 0
1 2 3 0  | 1
2 3 0 1  | 2
3 0 1 2  | 4   page fault
0 1 2 4  | 0
1 2 4 0  | 1
2 4 0 1  | 2
4 0 1 2  | 5   page fault
0 1 2 5  | 3   page fault
1 2 5 3  | 5
1 2 3 5  | 6   page fault
2 3 5 6  | 3
2 5 6 3  |
```

Total 8 page faults

   (b) Clock

After 0 1 2 3, 4 pages faults already

```
0* 1* 2* 3*   | 0
0* 1* 2* 3*   | 1
0* 1* 2* 3*   | 2
0* 1* 2* 3*   | 4  page fault, scan, clear used: 0   1   2   3
4* 1  2  3    | 0  page fault
4* 0* 2  3    | 1  page fault
4* 0* 1* 3    | 2  page fault
4* 0* 1* 2*   | 5  page fault, scan, clear used: 4   0   1   2
5* 0  1  2    | 3  page fault
5* 3* 1  2    | 5
5* 3* 1  2    | 6  page fault
5* 3* 6* 2    | 3
5* 3* 6* 2    |
```

Total 11 page faults

   (c) Aging: Assume that the page reference is for the entire system, not from a single program and the memory has only 4 pages. Apply aging policy to the access scenario above and give the aging vectors for all 4 pages. Assume that the system maintains 8 aging bits and before the above accesses the aging vectors for all 8 pages were 0. Also, assume that the background scanning and updating of aging vectors happened after every 2 accesses.

page	frame0	frame1	frame2	frame3
0	1000 0000 (0)	0000 0000	0000 0000	0000 0000
1	1000 0000	1000 0000 (1)	0000 0000	0000 0000

	0100 0000	0100 0000	0000 0000	0000 0000
2	0100 0000	0100 0000	1000 0000 (2)	0000 0000
3	0100 0000	0100 0000	1000 0000	1000 0000 (3)
	0010 0000	0010 0000	0100 0000	0100 0000
0	1010 0000	0010 0000	0100 0000	0100 0000
1	1010 0000	1010 0000	0100 0000	0100 0000
	0101 0000	0101 0000	0010 0000	0010 0000
2	0101 0000	0101 0000	1010 0000	0010 0000
4	0101 0000	0101 0000	1010 0000	1000 0000 (4)
	0010 1000	0010 1000	0101 0000	0100 0000
0	1010 1000	0010 1000	0101 0000	0100 0000
1	1010 1000	1010 1000	0101 0000	0100 0000
	0101 0100	0101 0100	0010 1000	0010 0000
2	0101 0100	0101 0100	1010 1000	0010 0000
5	0101 0100	0101 0100	1010 1000	1000 0000 (5)
	0010 1010	0010 1010	0101 0100	0100 0000
3	1000 0000 (3)	0010 1010	0101 0100	0100 0000
5	1000 0000	0010 1010	0101 0100	1100 0000
	0100 0000	0001 0101	0010 1010	0110 0000
6	0100 0000	1000 0000 (6)	0010 1010	0110 0000
3	1100 0000	1000 0000	0010 1010	0110 0000
	0110 0000	0100 0000	0001 0101	0011 0000

Total 8 page faults

2. Consider a Unix-like file system with the I-node structure to index file blocks. The I-node data structure is given in the following: Each I-node contains 7 direct addresses, 2 single indirect address pointers, and one double indirect address pointer. (There is no triple indirect address pointer.) Assume that each address is 32 bits long. Also assume that each logical file block is of size 8KB.
Given a file of size 100MB, is the inode sufficient to address all the blocks of the file.

7 direct pointers: Total file size = 7 x 8K = 56 KB $\Rightarrow$ not enough
2 single indirect pointers points to 2 blocks of direct pointers, each block can have 2K addresses:
Total file size = 2*2K*8K = 32 MB $\Rightarrow$ not enough
1 double indirect pointer can have 2K*2K = 4M total addresses:
Total file size = 4M*8K = 32 GB $\Rightarrow$ enough

3. Consider a Unix-like file system. It has file block size 4KB. Each inode is of size 128B and its format is as defined in Question 2. The plan is to support the storage of 1M files in the system. The average file size is 16KB.
A disk is used to host the file system, which has 512 sectors on each track and each sector is 512B. The file system starts at track 0. Also, the disk uses the C-Scan algorithm for its arm scheduling. Currently, the disk head is at track 600 with a forward direction.

(a) What is the rough number of file data blocks to be allocated for the file system?
1M files, average size is 16KB. So we need 16GB file storage, which is 4M file blocks.

(b) How many blocks are needed for the file block map?
We have 4M file blocks, which needs 4M bits for the map, so it requires 128 blocks for file block map.

(c) How many inode blocks are needed for the file system?

1M files, we need 1M inodes. Each block 4KB can have 32 inodes. So we need 32K inode blocks.

(d) How many blocks are needed for the inode map?

Each block has 32Kbits. The inode map needs 1M bits, so, we need 32 blocks for inode map.

(e) How many blocks total are required in the file system?

1 super block + 32+128 map blocks + 32K inode blocks + 4M file blocks
It is a total of 161 + 32K + 4M blocks

(f) How many tracks are required to host the file system?

Each track is 256KB, which can have 64 blocks
3 tracks for superblock and maps, 512 tracks for inodes, 64K tracks for file content
Total 64K+515 tracks

(g) Assume that a file F was nonexisting and a command is just issued to created F in the given file system. F got an inum 1024 and its inode is as shown in the figure below after it is created. Assume that the pointers in the i-node is the block offset from the beginning of the file blocks. Also assume that the OS, upon creating a file, will calculate all the required updates to the disk and issue all the updates to the disk manager (arm scheduler) all together. Which tracks have been visited during the creation of F? Give these tracks in the order they are visited.

file attributes	direct pointer 3500	direct pointer 1000	direct pointer 8000	direct pointer null	direct pointer null	direct pointer null	direct pointer null	single indirect pointer null	single indirect pointer null	double indirect pointer null

Creation of F includes: (i) update F's data blocks (ii) update F's inode, (iii) update the maps.
(i) F's file blocks include:
1000: is actually the (1000+1+32+128+32K)th block in the file system, i.e., the (32K+1161)th block.
It is on the (512+18+1)th track on the disk, i.e., track 530.
3500 is at (32K+3661)th block, (512+57+1)th track, i.e., track 569.
8000 is at (32K+8161)th block, (512+127+1)th track, i.e., track 639.
(ii) F's inode block:
F's inum 1024 is in inode block (1024/32=32).
It is block (1+32+128+32=193) in the file system.
It is on the (3+1)th track, i.e., track 3.
Note: though first 2 inum are reserved, here we say inum=1024, the first 2 are already factored in
(iii) F's map blocks
32-th inode block is in the first block of the i-node map, which is the (1+1)th block of the file system.
It is on track 0.
A block can have 4K*8 map bits, so all F's file blocks are in the 1st data block map.
It is block (1+32+1) in the file system and it is on track 0.
Tracks visited in their visiting order: 600-639-0-3-530-569.

4. Consider a 500GB, 7200rpm disk. Each track on the disk has 128 sectors and each sector is 1KB. The disk arm traverses the tracks at 0.1 msec per track and requires a start up time 0.2msec. A file system is built on top of this disk and the size of each file block is 8KB. Note that a file block is always allocated contiguously on the disk. Consider a file of size 80KB. The file is allocated on the disk using indexed allocation scheme and the index table is kept in memory. The file contains 10 file blocks, which are allocated on tracks 105, 120, 112, 101, 113, 106, 116, 108, 111, and 115. Assume that the disk has a 1MB buffer and it uses the Scan algorithm for disk arm scheduling. Compute the

time required to access the entire file. (Assume that there are no other disk access requests in the system and the current disk arm is at track 0.)

Average rotational delay is 4.2ms.
Time to transfer 128KB = 8.3ms, so time to transfer 8KB = 8.3ms / 16 = 0.52ms
Disk buffer is sufficiently large, so the blocks can be fetched in any order.
Total seek time = 10 start up time + 120 tracks traversal = 10*0.2ms + 120*0.1ms = 14ms
Total time = 14ms + (4.2+0.52)*10 = 61.2ms
OR
Total seek time = 1 start up time + 120 tracks traversal = 0.2ms + 120*0.1ms = 12.2ms
Total time = 12.2ms + (4.2+0.52)*10 = 59.4ms

5. Consider a log-structured file system with each file block of size 4KB. Now we are updating file F1, and a new segment is started just before this update. The segment starts at block 10000 in the file system. The inode design and F1's inode content are as given in Q3. We add 3 data blocks to F1 at offset 4KB. We also created a new file F2 with 4 data blocks. Now a flush occurred and the current part of the segment is written to disk. The inum for F1 and F2 are 87 and 233, respectively.

(a) Give the current layout of the new segment (at a high level), indicating the data blocks for each file, the inode blocks, and the imap locations.

CR imap	F1-D	F1-D	F1-D	F2-D	F2-D	F2-D	F2-D	inode block	imap	F1-D

(b) Give the inodes for F1.

file attributes	3500	10001	10002	10003	1000	8000	…

(c) Describe the imap content at a high level, no need to worry about the specific format.
i-node for inum 87 (F1): at block #10008, 0th inode
i-node for inum 233 (F2): at block #10008, 1st inode
(generally, one write will contain many file updates, gaining the sequential write benefits)

6. Consider the clock device in Unix system

(d) Assume that the current time is March 4, 1971, 10:00:00 a.m. What would be the value in the clock register?
Difference between 1/1/1970 and 3/4/1971 = 365 + 31 + 28 + 3 = 427
427*24*60*60 seconds = 36892800 seconds
10 hrs = 36000 seconds
the clock should be 36928800

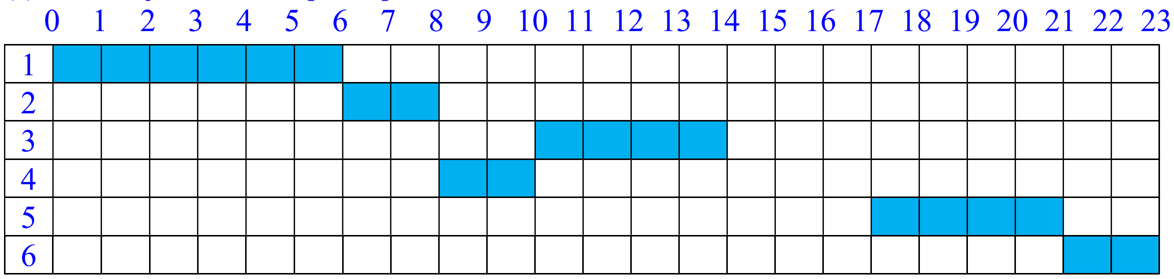
(e) Assume that current time is 10:22:30. Also, assume that the clock timer unit is 1 second. Consider the following timer requests being issued to the clock. Show the timer data structure.
(A) Process A issued command "sleep (100)" at time 10:22:00.
(O) OS set time quantum for current process at time 10:22:10. Time quantum is 30seconds.
(B) Process B issued command "set_timer (10:25:00)" at time 10:22:20.

(A) is 70 seconds after current time
(O) is 10 seconds after current time
(B) is 150 seconds after current time
The list of timers should be: 10 → 60 → 80
10 is at the timer register. 60 → 80 is in the list of timer requests.

7.

Job Number	1	2	3	4	5	6
Arrival Time	0	1	4	7	17	18
Service Time	6	2	4	2	4	2

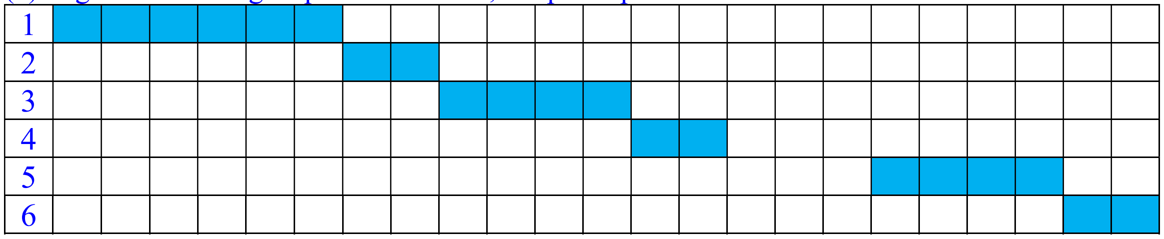
For (a) Shortest job first, non-preemptive



Job #	1	2	3	4	5	6
Arrival time	0	1	4	7	17	18
Finish time	6	8	14	10	21	23
Response time	6	7	10	3	4	5
Service time	6	2	4	2	4	2

Average response time = (6+7+10+3+4+5)/6 = 35/6 = 5.83
Average (response time/service time) = (1+7/2+10/4+3/2+4/4+5/2)/6 = 2

For (b) Highest remaining response ratio first, nonpreemptive



Time 7: Job 2 = (5+2)/2 = 3.5; Job 3 = (3+4)/4 = 1.75; Job 4 = 1
Time 9: Job 3 = (5+4)/4 = 2.25; Job 4 = (2+2)/2 = 2

Job #	1	2	3	4	5	6
Arrival time	0	1	4	7	17	18
Finish time	6	8	12	14	21	23
Response time	6	7	8	7	4	5
Service time	6	2	4	2	4	2

Average response time = (6+7+8+7+4+5)/6 = 37/6 = 6.17

Average (response time/service time) = (1+7/2+2+7/2+1+5/2)/6 = 2.25

For (c) Multilevel feedback queue (number of queue levels = 4, time quantum = 1)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	🟦		🟦				🟦				🟦		🟦	🟦									
2		🟦		🟦																			
3					🟦	🟦				🟦		🟦											
4								🟦	🟦														
5																		🟦		🟦		🟦	🟦
6																			🟦		🟦		

	1	2	3	4	5	6	7	8	9	10	11	…	18	19	20	21	22	23
Q1	1	2			3			4				…	5	6				
Q2		1	1,2	2		3			4					5	5,6	6		
Q3				1	1	1	1,3	3	3	3						5	5	
Q4								1	1	1	1,3							5

An easier way to do this is to maintain the queue level in the scheduling diagram. This way you do not need to maintain the queues separately.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	1		2				3				4		4	4									
2		1		2																			
3					1	2				3		4											
4								1	2														
5																		1		2		3	4
6																			1		2		

Job #	1	2	3	4	5	6
Arrival time	0	1	4	7	17	18
Finish time	14	4	12	9	23	21
Response time	14	3	8	2	6	3
Service time	6	2	4	2	4	2

Average response time = (14+3+8+2+6+3)/6 = 36/6 = 6
Average (response time/service time) = (7/3+3/2+2+1+3/2+3/2)/6 = 1.64

8.

Job name	A	B	C	D
Period	6	4	12	6
Service time	1	1	2	1

There will be a feasible schedule to allow the execution of all jobs. More formally, we can use rate monotonic theorem to verify the schedulability of the tasks. If $U < n\,(2^{1/n} - 1)$, then the jobs are schedulable, where U is CPU utilization and n is the number of jobs.

$U = 2/12 + 1/4 + 1/6 + 1/6 = 3/4 <= n\,(2^{1/n} - 1) = 0.7568$.

So, the jobs are schedulable.
Note: after scheduling B, you can either choose A to schedule first or choose D to schedule first.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A		🟧					🟧							🟧	
B	🟦				🟦				🟦				🟦		
C				🟪		🟪									
D			🟩					🟩							🟩

9.

0	1	2	3	P1
4	5	6	7	P2
8	9	10	11	P3
12	13	14	15	P4
16	17	18	19	P5
Disk 0	Disk 1	Disk 2	Disk 3	Disk 4

(a) If a sequence of read requests retrieve data blocks 1, 6, 8, 9, 13, 15, 16, how many rounds of parallel accesses will need to be performed? Which blocks will be accessed in each round?
Round 1: read 1, 6, 8, 15
Round 2: read 9, 16
Round 3: read 13

(b) If a sequence of write requests access data blocks 1, 3, 4, 6, 7, how many rounds of parallel accesses will need to be performed? Which blocks will be accessed in each round? Give an answer that will require the least number of block accesses.
Round 1: read 0, 2, 5;
Round 2: write 1, 3, P1
Round 3: write 4, 6, 7, P2