Q1. A computer has six tape drives, with n processes competing for them. Each process may need up to two drives. For which values of n is the system guaranteed to be deadlock free? Explain.

If n <= 3, then clearly there can be no deadlock, since even in the worst case, if 3 processes all simultaneously get a tape drive, there is still enough to go around. But note that even for n = 5, there can't be any deadlock. In the worst case scenario with n = 5, all five will get one tape drive, and there is still an extra that can be given to any single process. That process would then finish, freeing up two tape drives for the remaining four, etc. Alternatively, just create a banker's algorithm table with 5 processes, max=2 for each of them, and check that no possible allocations of six tape drives will be unsafe. So n < 6 is our final answer.

Q2. Synchronization :
(a) Consider the following three scenarios. One of them is best described as deadlock, one as livelock
and one as starvation. Explain which is which.
(i) Philosophers A and B want to pick up a chopstick. They try to be nice by waiting for one second if the other one is about to take the chopstick. They both try to pick up the chopstick, wait for a second, try again, wait again, ad infinitum.
(ii) Process A is waiting for the result of a computation performed by process B. However, A has higher priority than B, and so the OS prefers A and refuses to give CPU time to B. Therefore, both A and B are stuck.
(iii) Processes A and B communicate through a buffer. Process A is waiting for the buffer to be more full before it reads it. Process B is waiting for the buffer to be less full before it writes more data to it. Therefore, both A and B are stuck.
Answer: (i) livelock (ii) starvation (iii) deadlock

Q3. In a 32-bit machine we subdivide the virtual address into 4 segments as follows:

8 bit	8 bit	6 bit	10 it

We use a 3-level page table, such that the first 8 bits are for the first level and so on. In the following questions, sizes are in bytes.
Note: The problem asked for the actual allocated page table size, not the number of valid PTE. The solution below gives both answers.
(a) What is the page size in such a system?
2^10 = 1K
(b) What is the size of the page tables for a process that has 256K of memory starting at address 0? Assume each page-table entry is 4 bytes.

(b) **Using the subdivision above, the first level page table points to 256 2nd level page tables, each pointing to 256 3rd page tables, each containing 64 pages. The program's address space consists of 256K/1k = 256 pages, thus we need we need 256/64 = 4 third-level page tables. Therefore we need 4 entries in a 2nd level page table, and one entry in the first level page table. Therefore the size is: 256 entries for the first table, 256 entries for the 2nd level page**

table, and 4 3rd level page table containing 64 entries each. Assuming 4 bytes per entry, the space required is 256 * 4 + 256 * 4 (one second-level page table) + 4 * 64 * 4 (4 third-level page tables) = 3072 bytes.

(c) Also assume that each page-table entry is 4 bytes. What is the size of the page tables for a process that has a code segment of 48KB starting at address 0x01000000, a data segment of 600KB starting at address 0x80000000 and a stack segment of 64KB starting at address 0xF0000000 and growing upward (address of top of stack increases)?

(c) First, the stack, data and code segments are at addresses that require having 3 page tables entries active in the first level page table. For 64K, you need 64 pages, or 1 third-level page tables. For 600K, you need 600 pages, or 10 third-level page tables and for 48K you need 48 pages or 1 third-level page tables. Assuming 2 bytes per entry, the space required is 256 * 4 + 256 * 1 * 4 (1 second-level page tables) + 64 * (10+1+1)* 4 (10 third-level page tables for data segment, 1 for stack and 1 for code segment) = 7168 bytes.

Q4. An engineer has designed a FAT-like system and he has used 24 bits for each entry. For a 32-GB disk, what is the minimum size of a file allocation in this system? Justify your answer.

Solution:

A 32 GB disk has 2^35 bytes of storage; if each entry in the FAT has 24 bits, then there can be at most 2^24 allocation chunks (one per FAT entry), so each allocation chunk must be 2^11 bytes = 2KB

Q5. Consider an index-based file system with the inode containing 64 indexes, 1 indirect index pointing to a disk block containing an array of direct indexes, and 1 2-level index in the usual way. Assume that each index takes 4 bytes. What is the maximum file size under this arrangement, if a disk block is 1024 bytes?

Solution:

An indirect block has 1024 bytes * 1 index/4 bytes = 256 pointers

So a file has at most 64 + 256 + 256*2 blocks and (64 + 256 * 256^2) * 1024 bytes.

Q6. Given a disk with 100 cylinders (0 to 99). Exactly one time unit is required to move the read/write head from one cylinder to the next. At time 0 the heads are cylinder 0 and no requests are pending. The following six requests arrive at tmes shown in:

Time (units)	0	10	20	70	80	90
Cylinder	21	75	16	68	2	17

Note that once heads are moving arrival of new request will not alter the intended distination of the heads in movement. Assuming, that rotational and transfer times are negligible, give the order in which the cylinders will be visited and the total time required for the Scan algorithm.

Solution :

Time of request	**t=0**	**t=10**	**t=20**	**t= 21**	**t=70**	**t=75**	**t=80**	**t=84**	**t=90**	**t=135**	**t=136**	**t=150**	
Pending request	**21**	**21,75**	**21,75, 16**	**21**	**75,16,68**	**16,68**	**16,68,2**	**16,2,**	**16,2, 17**	**16,2**	**2**	**0**	
Cylinders accessed				**21**		**75**		**68**		**17**	**16**	**2**	

Cylinders will be accessed in the order : 21 , 75 , 68 , 17, 16 , 2