Question1

a) LIFO is most used in cases where the last data added to the structure must be the first data to be removed or evaluated. It needs to access the data in limited amounts and in a certain order. LIFO structure is accessed in opposite order to a queue. Good for transaction processing systems: the device is given to the most recent user so there should be little arm movement.

b) Possibility of starvation since a job may never regain the head of the line.

Question2

RAID level 2 can not only recover from crashed drives, but also from undetected transient errors. If one drive delivers a single bad bit, RAID level 2 will correct this, but RAID level 3 will not.

Question3

Contiguous allocation: any maximize of file may need to be copied to another location, so this method will greatly reduce performance, in addition the large external fragmentation may be occur. Because the possible file sizes range in the system is relatively small, we need choose a relatively small block size, in order to reduce the internal fragmentation, and save space. If to take a block size of 2KB, the average internal fragmentations of each file will be 1KB.

I-node method: it is possible to create files that are much larger than the given maximum, but because of the much larger "advantage" of the desired, the access to the blocks will be slower because of two indirect pointers (first of all, access in the internal block of pointers and only then, find the requested block). In addition, for files larger than 20 KB, the file system will need to use another block to safe indirect blocks, so for each file we lost 2KB.

FAT (suitable method): need to save space for the FAT tables once, and for each file blocks will be grouped by size. In last two methods, the internal fragmentation cannot be avoided, but, FAT32 (for example), avoids allocating one block that will use for pointers to other blocks, and accessing to larger files will be faster, so this method is more desirable than the i-node method. So if block size 1KB now, then internal fragmentations of each file will be 512 bytes and system work more fast with large files like in i-node.

Question4

Three methods of protecting:

1) Requires a tagged architecture, a hardware design in which each memory word has an extra (or tag) bit that tells whether the word contains a capability or not. The tag bit is not used by arithmetic, comparison, or similar ordinary instructions and it can be modified only by programs running in kernel mode (i.e., the operating system).

2) To keep the C-list inside the operating system. Capabilities are then referred to by their position in the capability list. A process might say: "Read 1 KB from the file pointed to by capability 2." This form of addressing is similar to using file descriptors in UNIX.

3) To keep the C-list in user space, but manage the capabilities cryptographically so that users cannot tamper with them. This approach is particularly suited to distributed systems and works as follows. When a client process sends a message to a remote server, for example, a file server, to create an object for it, the server creates the object and generates a long random number, the check field, to go with it. A slot in the server's file table is reserved for the object and the check field is stored there along with the addresses of the disk blocks. In UNIX terms, the check field is stored on the server in the i-node. It is not sent back to the user and never put on the network. The server then generates and returns a capability to the user of the form.