Question 1.
List two ways a system could use to determine which disk blocks are free. Give advantages in each way.

Two ways a system could use to determine which disk blocks are free are:

a. Free-Space List:
In this method, a list is maintained where each entry indicates a disk block or sector that is available for use. The list can be implemented in various ways such as a linked list, a stack, or a queue.

Advantages:
Simple Implementation: The free-space list is relatively straightforward to implement and understand.
Easy Allocation and Deallocation: Allocating and deallocating disk blocks is straightforward; simply add or remove entries from the list.
No Overhead of Tracking Used Blocks: Unlike a used-sector list, a free-space list only concerns itself with free sectors, potentially reducing the overhead associated with tracking used sectors.

b. Bit Vector (Bitmap):
A bit vector or bitmap is a data structure where each bit represents the status of a disk block - whether it's free or occupied. If a bit is set to 1, it indicates that the corresponding block is occupied, and if it's set to 0, it indicates that the block is free.

Advantages:
Compact Representation: Bitmaps are space-efficient as they require only one bit per block to represent the status of all disk blocks, making them a compact representation of free and occupied blocks.
Fast Lookup: Bitmaps allow for quick lookups to determine the status of a block, making allocation and deallocation operations relatively fast.
No Links: There are no links that can be broken or corrupted as in a linked-list representation, which can reduce the chance of errors and the complexity of handling links.
Ease of Finding Contiguous Blocks: Bitmaps make it easier to find a range of contiguous free or occupied blocks, which can be beneficial for certain allocation strategies.
Both the free-space list and the bit vector have their own sets of advantages depending on the particular requirements and constraints of the system in question.

1. Free-Space List Implementation:

```
#include <iostream>
#include <queue>

const int TOTAL_BLOCKS = 100;

// A simple implementation using a queue
class FreeSpaceList {
public:
    FreeSpaceList() {
        // Initialize all blocks as free
        for (int i = 0; i < TOTAL_BLOCKS; ++i) {
            freeBlocks.push(i);
        }
    }

    int allocateBlock() {
        if (freeBlocks.empty()) {
            std::cerr << "No free blocks available!" << std::endl;
            return -1;
        }
        int block = freeBlocks.front();
        freeBlocks.pop();
        return block;
    }

    void deallocateBlock(int block) {
        freeBlocks.push(block);
    }

private:
    std::queue<int> freeBlocks;
};
```

2. Bit Vector (Bitmap) Implementation:

```
#include <iostream>
#include <bitset>

const int TOTAL_BLOCKS = 100;

class BitVector {
public:
    BitVector() : bitmap(TOTAL_BLOCKS) {}  // Initialize all bits to 0 (free)

    int allocateBlock() {
        for (int i = 0; i < TOTAL_BLOCKS; ++i) {
            if (!bitmap[i]) {
                bitmap.set(i);
                return i;
            }
        }
        std::cerr << "No free blocks available!" << std::endl;
        return -1;
    }

    void deallocateBlock(int block) {
        bitmap.reset(block);
    }

private:
    std::bitset<TOTAL_BLOCKS> bitmap;
};
```

Question 2.
What is contiguous allocation? Name one advantage and one disadvantage of contiguous allocation.

Contiguous allocation is a method used in operating systems and file systems to allocate space to files such that each file occupies a set of contiguous addresses or blocks in memory or on disk. In this method, each file is stored in a contiguous section of storage, with no gaps between the files.

Advantage:
Fast Access:
Accessing data in contiguous allocation is usually faster since the system can calculate the address of any part of a file based on its starting address and its size, reducing the need for complex data structures to keep track of file locations. This is particularly advantageous for sequential access.

Disadvantage:
External Fragmentation:
Over time, as files are created, deleted, and resized, gaps can appear between the allocated areas which may not be large enough to accommodate new files or file expansions. This situation, known as external fragmentation, can result in inefficient utilization of the storage space.

Contiguous allocation simplifies the design and access to files but can lead to challenges in efficiently managing the available space, particularly in long-running systems or systems with frequently changing file sizes.

Question 3.
What is the difference between a hard link and a symbolic link? Give an advantage of each one. How does a symbolic link work?

Hard Link: A hard link is a directory entry that associates a name with a file on a file system. All directory-based file systems must have at least one hard link giving the original name for each file. When a hard link is created, it essentially provides an additional name for a file that already exists.
Symbolic Link (Symlink): A symbolic link is a file that contains a reference to another file or directory in the form of an absolute or relative path. Unlike a hard link, a symbolic link can point to a file or directory on a different file system or partition.

Advantage:

Hard Link Advantage: Hard links share the same inode (the data structure that holds the file's metadata and data blocks' location) with the original file. Thus, even if the original file is deleted, the content remains accessible through the hard link until all links to the inode are deleted.

Symbolic Link Advantage: Symbolic links can link to directories or across different file systems, providing a level of indirection and flexibility that hard links don't offer.

How Symbolic Link Works:
A symbolic link works by creating a special file that contains a path to another file or directory. When you access the symbolic link, the operating system redirects the operation to the path specified within the symbolic link.

Real Life Scenario:
Imagine a library with many books and a central catalog that helps you find where each book is located.

Hard Link Scenario:
Suppose there's a popular book that's often sought by visitors. To make it easier to find, the library creates additional catalog entries for this book under different categories. These extra catalog entries are like hard links - they all point directly to the actual book. If the original catalog entry is removed, the book is still easily found through the additional catalog entries.

Symbolic Link Scenario:
Now, suppose the library decides to move the popular book to a more prominent location. Instead of updating all the catalog entries, the library could leave a note in the original location saying "This book has been moved to the new location XYZ." This note acts like a symbolic link,

pointing visitors to the new location of the book. Unlike hard links, if the original book is moved or removed, the symbolic link (the note) becomes outdated and may point to a non-existent location.

Question 4.
Three different protection mechanisms that we have discussed are capabilities, access control lists, and the UNIX rwx bits. For each of the following protection problems, tell which of these mechanisms can be used.

(a)Ken wants his files readable by everyone except his office mate.

Access Control Lists (ACLs): This would be the most suitable mechanism for this scenario. ACLs allow for specifying permissions on a per-user or per-group basis. Ken could set an ACL on his files that denies read access to his office mate while allowing read access to everyone else.

Allowing Read Access to Everyone:
First, set the file permissions so that everyone has read access:

chmod a+r filename

Denying Read Access to Office Mate:
Next, use the setfacl command to deny read access to Ken's office mate (let's assume his username is officemate):

setfacl -m u:officemate:r- filename

In this command:

-m stands for modify.
u:officemate:r- denotes that the user officemate is denied read **(r-)** access to the file.
This command will modify the ACL of the file named filename, preventing officemate from reading the file, while all other users will still have read access due to the previous chmod command.

(b)Mitch and Steve want to share some secret files.

UNIX rwx bits: By creating a shared group, adding Mitch and Steve to that group, and setting the group permissions on the secret files to allow read and write access, they can share files effectively. This is a straightforward solution assuming Mitch and Steve are okay with the same level of access to these files.
Access Control Lists (ACLs): If more granular control is needed, ACLs could also be used to specify exactly what permissions Mitch and Steve have on each file. This mechanism allows for more flexibility than the UNIX rwx bits.

Create a Group:
First, create a new group to which both Mitch and Steve will belong. This can be done using the groupadd command:

```
sudo groupadd secretgroup
```

Add Users to Group:
Next, add Mitch and Steve to the newly created group using the usermod command:

```
sudo usermod -aG secretgroup mitch
sudo usermod -aG secretgroup steve
```

Create a Directory:
Create a directory where the secret files will be stored. Set the group ownership of the directory to secretgroup and set the setgid bit on the directory to ensure that all new files created within the directory inherit the group ownership:

```
mkdir /path/to/secretdir
sudo chgrp secretgroup /path/to/secretdir
sudo chmod g+s /path/to/secretdir
```

Set Permissions:
Set the permissions on the directory so that only members of secretgroup (i.e., Mitch and Steve) have read, write, and execute permissions:

```
sudo chmod 770 /path/to/secretdir
```

Explanation:
770 permissions mean that the owner and group members have full read, write, and execute permissions (7), while others have no permissions (0).

Create/Move Secret Files:
Now Mitch and Steve can create or move their secret files into the /path/to/secretdir directory. All files created within this directory will be accessible to both Mitch and Steve, but not to other users.

(c)Linda wants some of her files to be public.

UNIX rwx bits: This mechanism is well-suited for this scenario. Linda can set the permissions on her public files to allow read access for everyone (chmod a+r filename).
Access Control Lists (ACLs): If Linda wants more granular control over who can access her files (e.g., allowing read access to everyone but write access to only a specific group), ACLs could be used.
Change File Permissions:

To make a specific file public (readable by everyone), you can use the chmod command to set the appropriate permissions:

```
chmod a+r filename
```

In this command:
a stands for all, indicating that the following permissions should apply to the owner, group, and others.
+r adds read permission for all users.

If Linda wants to make multiple files public, she can use a wildcard or specify multiple files:

```
chmod a+r file1 file2 file3  # Specific files
chmod a+r *                  # All files in the directory
```

Question 5.
Discuss some of the issues which can mean that even the best system design cannot guarantee invulnerability to malicious attack.

Even with an impeccable system design, it's challenging to ensure absolute invulnerability to malicious attacks due to a variety of inherent and external factors. Here are some issues that contribute to this challenge:

Human Factor:
Human error remains a significant vulnerability in system security. Misconfigurations, weak passwords, or lack of awareness about phishing and other social engineering attacks can expose systems to threats.

Zero-Day Vulnerabilities:
These are unknown vulnerabilities in software or hardware that are not yet discovered by the developers but may be known to attackers. Until these vulnerabilities are identified and patched, they represent a significant security risk.

Complexity:
Modern systems are incredibly complex, involving numerous components and layers of software and hardware. The complexity increases the likelihood of vulnerabilities existing within the system.

Supply Chain Risks:
Systems often rely on external hardware and software components. If any component in the supply chain is compromised, it can introduce vulnerabilities into the system.

Insider Threats:
Malicious or negligent actions by employees or other insiders can pose a significant security risk, often bypassing many of the system's security measures.

Outdated Software:
Running outdated or unsupported software can expose known vulnerabilities that have been fixed in newer versions.

Sophisticated Attackers:
Attackers are becoming increasingly sophisticated, employing advanced techniques to evade detection and exploit vulnerabilities.

Lack of Resources:
Organizations may lack the necessary resources, expertise, or tools to keep up with the evolving threat landscape, leading to inadequate protection and response capabilities.

Legacy Systems:
Older, legacy systems might have inherent security weaknesses or may not support modern security features, yet they are critical for business operations.

Incomplete or Inaccurate Risk Assessment:
Inadequate risk assessment can lead to misallocated resources, where critical vulnerabilities remain unaddressed while less crucial aspects receive attention.

Data Proliferation:
The widespread sharing and storage of sensitive data across various systems and platforms increase the attack surface for malicious actors.

Law and Regulation Lag:
Legal and regulatory frameworks may lag behind technological advancements, providing inadequate guidance or enforcement to ensure system security.

Cross-Border Jurisdictional Issues:
Attackers may operate from regions with lax cybersecurity laws or enforcement, making it challenging to take legal action or prevent attacks.