

The background features a light blue gradient with faint, concentric circular patterns. Overlaid on this are stylized circuit traces in a slightly darker blue. These traces are most prominent on the left and right edges, consisting of vertical and horizontal lines with small circles at the junctions, resembling a printed circuit board (PCB) layout.

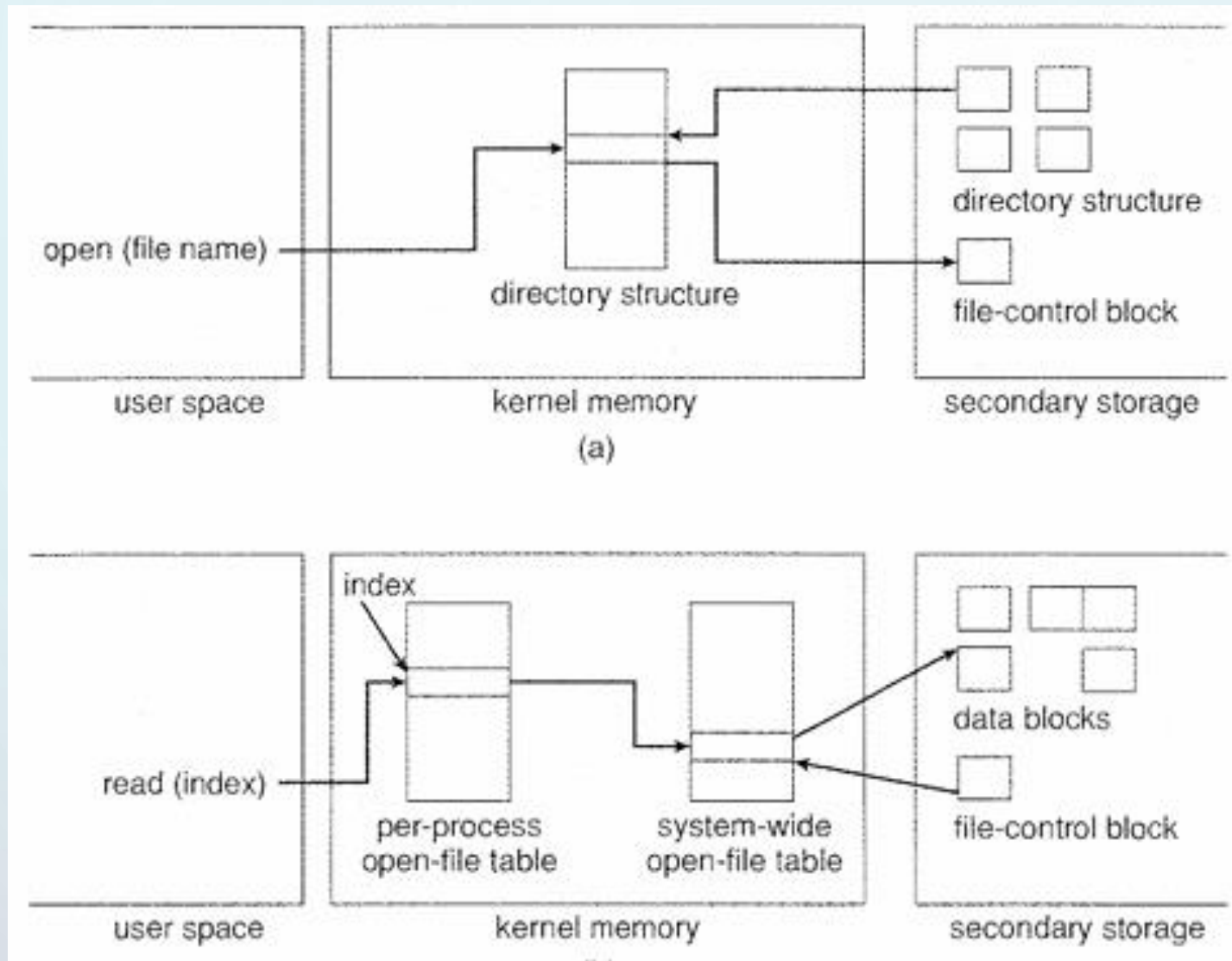
OPERATING SYSTEM

IT-42033

CHAPTER – 11

Q&A

Q. Illustrate in-memory file-system architecture to open and read file.



Q. Explain about the contiguous allocation.

Contiguous Allocation

Contiguous allocation is a file storage method where files are stored in a continuous block of disk space, defined by the starting disk address and the length of the file in blocks. This approach allows for efficient access, as sequential reads require minimal head movement, and direct access to any block can be achieved quickly. Contiguous allocation is to be managed in free space. As files are created and deleted, external fragmentation occurs, leading to small, unusable gaps in storage. This fragmentation can hinder the allocation of new files if there isn't a sufficiently large contiguous block available.

To mitigate fragmentation, systems compact the file system by moving files to create larger contiguous free spaces, though this process can be time-consuming and requires the system to be unmounted. The required space for a file at creation can be difficult, leading to potential issues if the allocated space is insufficient. Users either overestimate their needs, wasting space, or face errors when trying to extend files. While contiguous allocation offers performance benefits, it presents significant issue in space management and file size estimation.

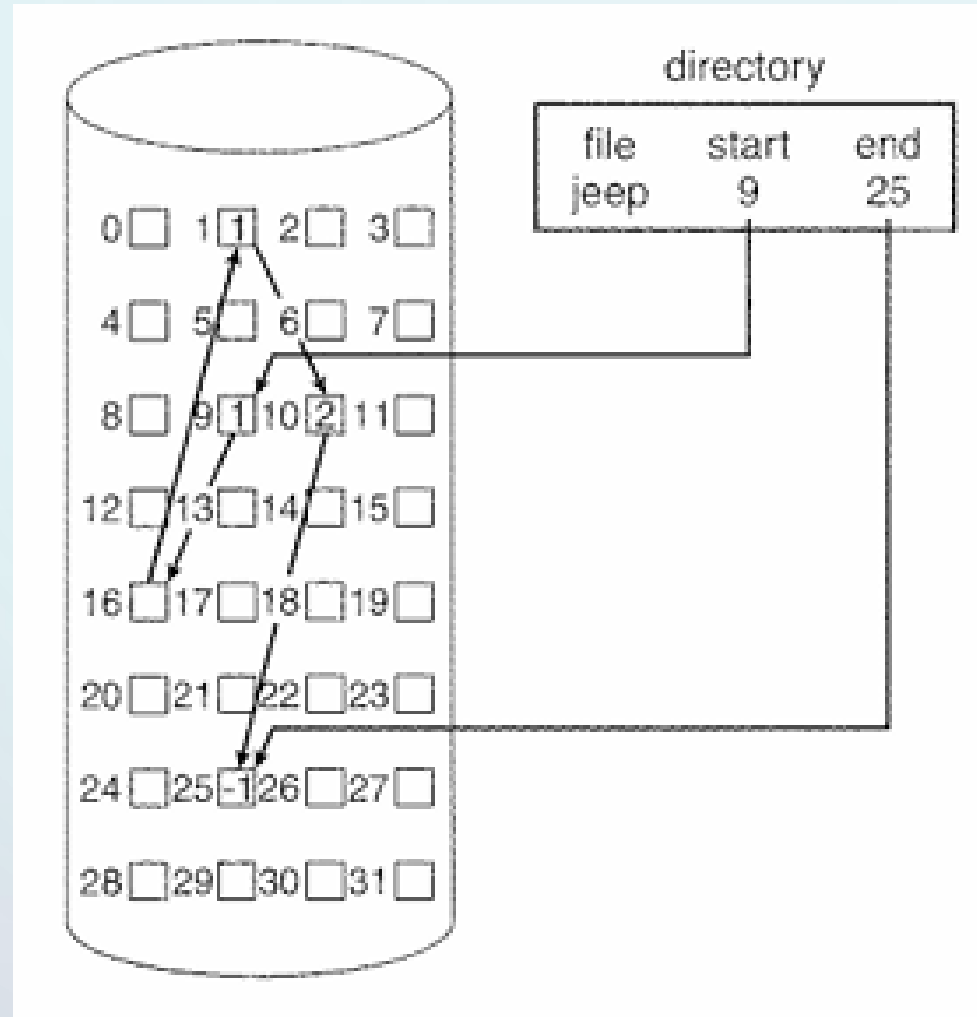
Q. Explain and illustrate about the linked allocation, there is a file *jeep* starting 9 to 25 in linked allocation directory.

Linked Allocation

Linked allocation solves all problems of contiguous allocation. With linked allocation, each file is a linked list of disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file.

To create a new file, we simply create a new entry in the directory. With linked allocation, each directory entry has a pointer to the first disk block of the file. This pointer is initialized to nil (the end-of-list pointer value) to signify an empty file. The size field is also set to 0. A write to the file causes the free-space management system to find a free block, and this new block is written to and is linked to the end of the file. To read a file, we simply read blocks by following the pointers from block to block. There is no external fragmentation with linked allocation, and any free block on the free-space list can be used to satisfy a request. The size of a file need not be declared when that file is created. A file can continue to grow as long as free blocks are available. It is never necessary to compact disk space.

Q. Explain and illustrate about the linked allocation, there is a file *jeep* starting 9 to 25 in linked allocation directory. (Cont ...)



Linked allocation of disk space

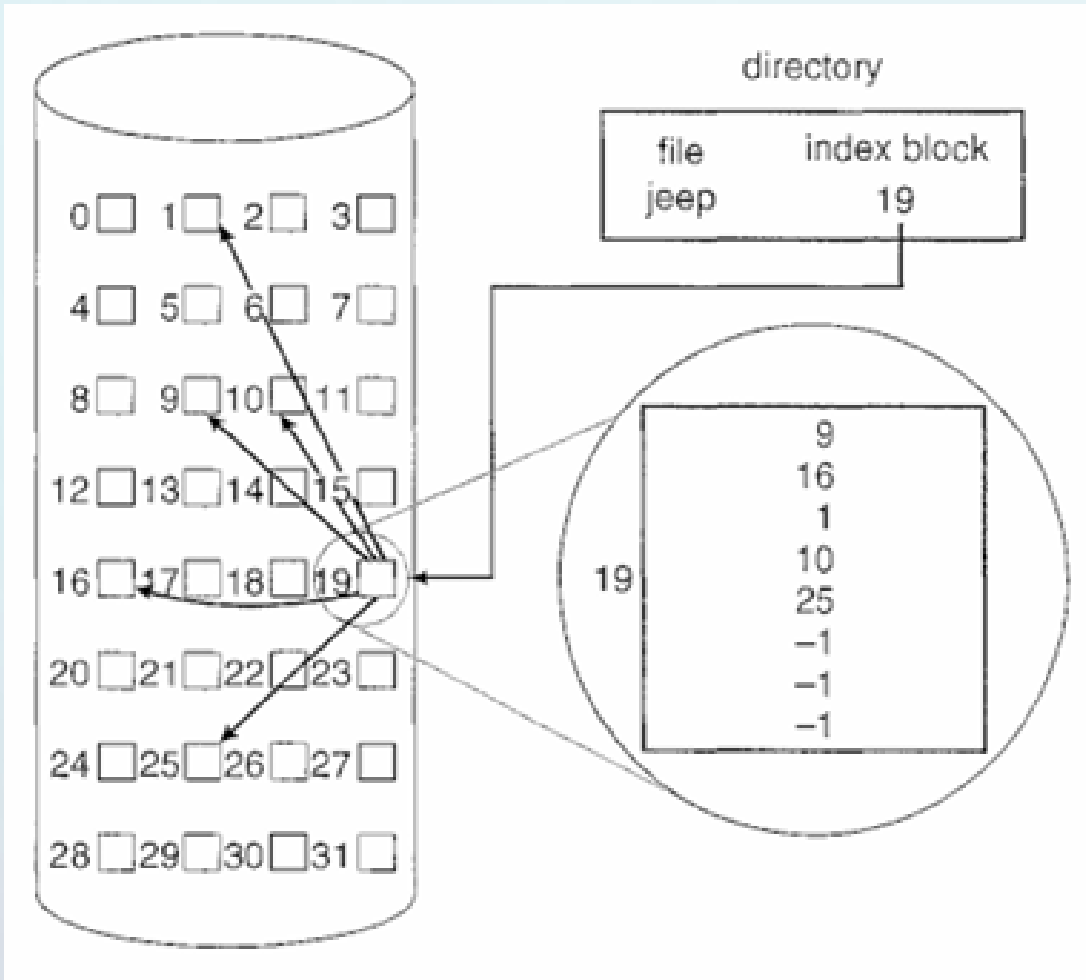
Q. Explain and illustrate about the indexed allocation, there is a file *jeep* index block 19 which is an array $i[9, 16, 1, 10, 25]$ in directory.

Indexed Allocation

Indexed allocation is a file storage method that addresses the limitations of contiguous and linked allocation by organizing pointers to file blocks in a centralized index block. Each file has its own index block, which contains an array of disk-block addresses. This structure allows for efficient direct access to any block of the file, as the i^{th} entry in the index block points directly to the i^{th} block of the file. When a file is created, all entries in the index block are initially set to *nil*, and as data is written to the file, the corresponding index entries are updated with the addresses of the allocated blocks.

One of the key advantages of indexed allocation is its ability to eliminate external fragmentation. Since any free block on the disk can be used to satisfy a request for more space, the system can efficiently allocate storage without being constrained by the need for contiguous space. The overhead of maintaining an index block can lead to wasted space, especially for small files. If a file consists of only one or two blocks, the entire index block must still be allocated, resulting in unused pointers for the remaining entries.

Q. Explain and illustrate about the indexed allocation, there is a file *jeep* index block 19 which is an array $i[9, 16, 1, 10, 25]$ in directory. (Cont ...)



Indexed allocation of disk space

Q. What are some common methods used by file systems to ensure data integrity during recovery?

Recovery

Recovery is important for maintaining the integrity of files and directories stored in both main memory and on disk. It ensures that a system failure does not lead to data loss or inconsistencies. When a system crashes, it can cause problems with the data structures on the disk, such as directory structures and free-block pointers. Many file systems make changes directly to these structures. For example, creating a file involves several changes, like modifying directory structures, allocating File Control Blocks (FCBs), and updating free block counts. If a crash happens during these changes, inconsistencies can occur.

The free FCB count shows that an FCB has been allocated, but the directory structure may not reflect this. Some changes may be saved directly to the disk, while others may be stored in cache. If a crash occurs before the cached changes are written to disk, further corruption can happen. Besides crashes, bugs in the file system, disk controllers, or user applications can also corrupt the file system. Different file systems have various methods to handle corruption based on their data structures and algorithms.

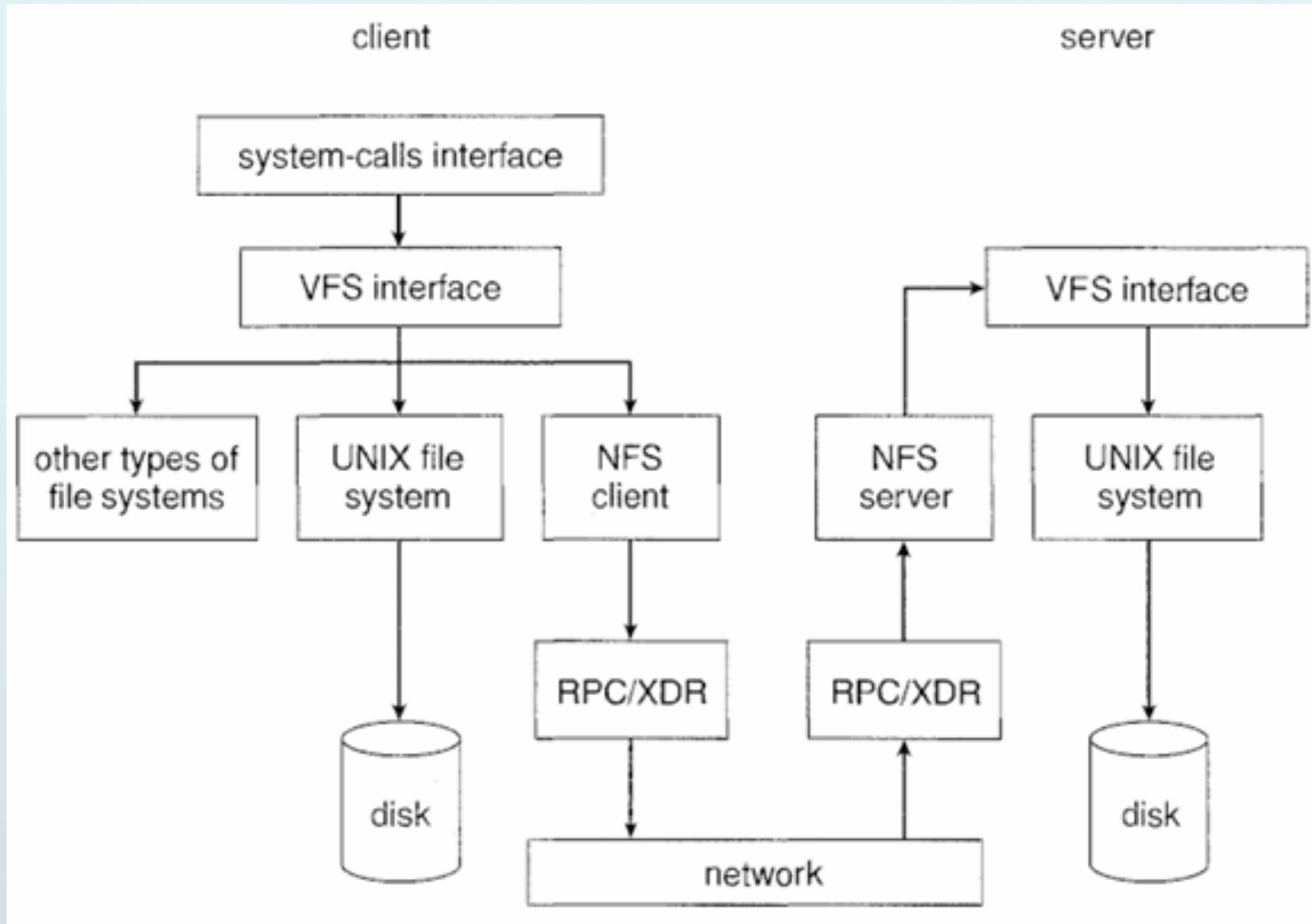
Q. What are the key features of the Network File System (NFS) and design its architecture?

NFS

Network file systems are commonly used and are usually integrated with the directory structure and interface of the client system. The Network File System (NFS) is a well-known example of a client-server network file system. It serves as a good case study for understanding how network file systems are implemented.

NFS is both a specification and an implementation that allows users to access remote files over Local Area Networks (LANs) or Wide Area Networks (WANs). It is part of the Open Network Computing (ONC+) suite, which is supported by many UNIX vendors and some PC operating systems. The implementation is based on the Solaris operating system, a modified version of UNIX System that runs on SUN workstations and other hardware. NFS can use either the TCP or UDP/IP protocol, depending on the network configuration.

Q. What are the key features of the Network File System (NFS) and design its architecture? (Cont ...)



Q. What is NFS Protocol?

NFS Protocol

The NFS protocol facilitates remote file operations through a set of Remote Procedure Calls (RPCs) that support various functions, such as searching for files, reading directory entries, manipulating links and directories. NFS omits traditional `open()` and `close()` operations, as it is designed to be stateless. NFS servers do not retain information about clients between accesses, eliminating the need for a server-side open-files table. Each request must include all necessary arguments, such as a unique file identifier and an absolute offset.

NFS allows for efficient handling of requests, as each request is treated independently. NFS request includes a sequence number, enabling the server to identify duplicate requests and manage any missing ones. To maintain data integrity, modified data must be committed to the server's disk before results are returned to the client, which can lead to performance penalties due to the loss of caching advantages. While NFS guarantees that a single write procedure call is atomic, it does not provide concurrency control, meaning that simultaneous writes to the same file

Q. How does the NFS protocol ensure that remote file operations correspond to standard UNIX system calls?

Remote Operations

With the exception of file opening and closing, there is a nearly direct correspondence between standard UNIX system calls for file operations and the Remote Procedure Calls (RPCs) used in the NFS protocol. This means that a remote file operation can typically be translated into a corresponding RPC. While NFS follows the remote-service paradigm conceptually, it employs buffering and caching techniques to enhance performance. As a result, there is no strict one-to-one mapping between remote operations and RPCs; instead, file blocks and attributes are fetched via RPCs and cached to certain consistency constraints.

NFS utilizes two types of caches: the file-attribute cache, which stores *inode* information, and the file-blocks cache. When a file is accessed, the kernel checks with the remote server to determine whether to fetch or revalidate the cached attributes. Cached file blocks are only used if the corresponding attributes are current. The attribute cache is updated whenever new information is received from the server, but cached attributes are discarded by default after 60 seconds. NFS also employs read-ahead and delayed-write techniques to optimize communication between the server and client.

11.5

The results are:

a. $100(\text{read}) + 1(\text{write}) = 101$

$1(\text{write new block}) + 1(\text{update pointer}) = 2$

$1(\text{write new block}) + 1(\text{update index}) = 2$

b. $100(\text{read}) + 1(\text{write}) = 101$

$1(\text{write new block}) + 1(\text{update previous pointer}) + 1(\text{update new block's pointer}) = 2$

$1(\text{write new block}) + 1(\text{update index}) = 2$

c. $1(\text{write new block}) = 1$

$1(\text{write new block}) + 1(\text{update last block's pointer}) = 2$

$1(\text{write new block}) + 1(\text{update index}) = 2$

d. $100(\text{read}) = 100$

$1(\text{update pointer}) = 1$

$1(\text{update index}) = 1$

	Contiguous	Linked	Indexed
a.	101	2	2
b.	101	3	2
c.	1	2	2
d.	100	1	1
e.	100	1	1
f.	0	1	1

e. $100(\text{read}) = 100$

$1(\text{update pointer}) = 1$

$1(\text{update index}) = 1$

f. $0(\text{no read}) = 0$

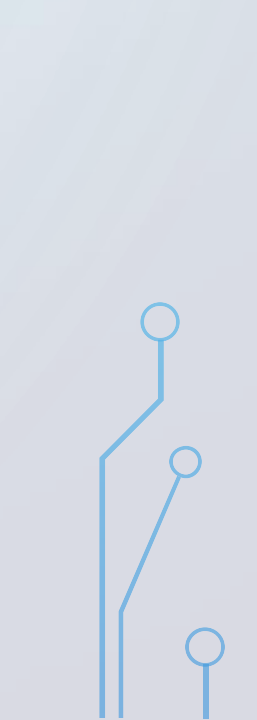
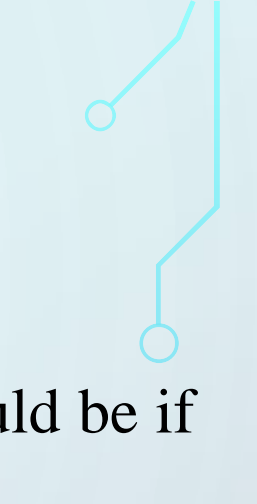

$1(\text{update pointer}) = 1$

$1(\text{update index}) = 1$



11.9

In case of system crash /memory failure the free-space list would not be lost as it would be if the bit map had been stored in main memory.



11.11


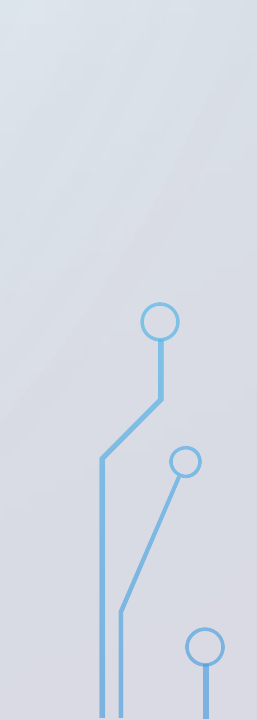
- Dynamic tables allow more flexibility in system use growth, Tables are never exceeded, avoiding artificial use limits.
- Unfortunately, kernel structures and code are more complicated, so there is more potential for bugs. The use of one resource can take away more system resources by growing to accommodate the requests than with static tables.

Q. Explain how the VFS layer allows an operating system to support multiple types of file systems easily.

- VFS introduces a layer of indirection in the file system implementation. In many ways, it is similar to object-oriented programming techniques.
- System calls can be made generically independent of file system type.
- Each file system type provides its function calls and data structures to the VFS layer.
- A system call is translated into the proper specific functions for the target file system at the VFS layer.
- The calling program has no file-system-specific code, and the upper levels of the system call structures likewise are file system-independent.
- The translation at the VFS layer turns these generic calls into file-system-specific operations.



Q. How do caches help improve performance? Why do systems not use more or larger caches if they are so useful?

- Caches allow components of differing speeds to communicate more efficiently by storing data from the slower device, temporarily, in a faster device cache.
 - Caches are, almost by definition, more expensive than the device they are caching for, so increasing the number or size of caches would increase system cost.
- 
- 

References

Images: Internet

Source: Operating System Concepts (8th Edition)