Study Set for Lecture 12: File System Implementation

Due Nov 16 at 11:59pm **Points** 10 **Questions** 12
Available Nov 10 at 12pm - Dec 8 at 8pm 28 days **Time Limit** None
Allowed Attempts Unlimited

Instructions

Review lecture notes from **lect12_File_System_Implementation.pdf**.

Then answer the questions from this study set and submit them to gain access to the further part of the course.

Take the Quiz Again

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	69 minutes	10 out of 10

Correct answers are hidden.

Score for this attempt: **10** out of 10
Submitted Nov 16 at 11:11am
This attempt took 69 minutes.

Question 1 — 0 / 0 pts

Describe with details typical memory-resident structures that support file system after it has been mounted.

What is a File Control Block (FCB)? What is its purpose?

Using the memory organization and data structures that you have just described, provide pseudocode for opening a file, and for reading from an

opened file.

Your Answer:

The two main memory-resident structures that support the file system after mounting are the directory structure and the open file tables. The directory is used to indicate where file contents are on the disk. The open file tables hold file descriptors of opened files, can be either system-wide open or only show files open per process.

A **File Control Block** (**FCB**) is a file system structure in which the state of an open **file** **(https://en.wikipedia.org/wiki/Computer_file)** is maintained. A FCB is managed by the operating system, but it resides in the memory of the program that uses the file, not in operating system memory. This allows a process to have as many files open at one time as it wants to, provided it can spare enough memory for an FCB per file. A file control block is an on-disk file system structure that contains details about and individual file or directory.

if (file is in the directory and has a file control block)

if (the file is already opened)

return file handle

else

create entry in the local/global file table for the file using the directory entry and the info in the file control block

return handle for the new entry

Question 2

0 / 0 pts

Describe the algorithms to

1. add
2. remove

a file or folder to a file system directory based on hashing functions.

Discuss its advantages and disadvantages.

Your Answer:

Hash Table

● direct access through a hash function
applied to the name
● decreases directory search time
● must deal with the collisions
● efficient solutions do exist though
● fixed size

1. **Hash Table –**
   The hash table takes a value computed from the file name and returns a pointer to the file. It decreases the directory search time. The insertion and deletion process of files is easy. The major difficulty is hash tables are its generally fixed size and hash tables are dependent on hash function on that size.

How can one store a large number of files while maintaining a high level of performance during access? One solution is file name hashing.

File name hashing in the simplest terms can be defined as, creating a known and reproducible path, based on the name of the file.

Lets do a test. The following java code will create a 2-level directory hash for the String “cat.gif”.

```
public static void main(String[] args) {

    String fileName = "cat.gif";
    int hash = fileName.hashCode();    int mask = 255;
    int firstDir = hash & mask;
    int secondDir = (hash >> 8) & mask;    String path = new StringBui
lder(File.separator)
            .append(String.format("%03d", firstDir))
            .append(File.separator)
            .append(String.format("%03d", secondDir))
            .append(File.separator)
            .append(fileName)
            .toString();    System.out.println(path);
}
```

The code generates the following output:

```
/172/029/cat.gif
```

If we ever need to find “cat.gif”, we can easily reproduce this path using the same algorithm.

Question 3

0 / 0 pts

Describe the algorithms to

1. add
2. remove

a file or folder to a file system using a linear directory.

Discuss its advantages and disadvantages.

Your Answer:

1. **Linear List –**
   It maintains a linear list of filenames with pointers to the data blocks.It is time-consuming also. To create a new file, we must first search the directory to be sure that no existing file has the same name then we add a file at end of the directory. To delete a file, we search the directory for the named file and release the space. To reuse the directory entry either we can mark the entry as unused or we can attach it to a list of free directories.

Question 4

0 / 0 pts

Describe the algorithms to

1. add
2. remove

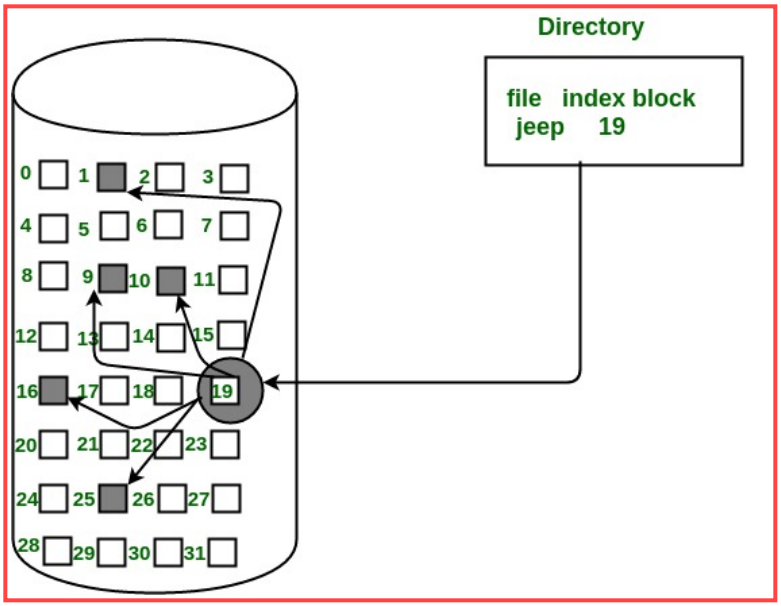
a file or folder to a file system using indexed allocation scheme for allocating storage space.

Discuss its advantages and disadvantages.

Your Answer:

3. Indexed Allocation

In this scheme, a special block known as the **Index block** contains the pointers to all the blocks occupied by a file. Each file has its own index block. The ith entry in the index block contains the disk address of the ith file block. The directory entry contains the address of the index block as shown in the image:



Advantages:

- This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
- It overcomes the problem of external fragmentation.

Disadvantages:

- The pointer overhead for indexed allocation is greater than linked allocation.
- For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.

Question 5

0 / 0 pts

Describe with details how to use linked lists for managing free space in a file system.

What are the advantages and disadvantages of this approach?

Your Answer:

Linked list-based free space management involves starting from the head of the list and traversing through the list starting from the head. Because of this traversals are very expensive, but fortunately traversing is not needed frequently, since it's a list of free blocks so the first block will usually be sufficient. No space is wasted since there won't be any disconnections as long as the list is properly managed and there are no gaps. But unfortunately it's hard to get contiguous space in a file system, and contiguous space speeds up reading and writing.

Question 6

0 / 0 pts

Describe with details how to use a bit vector for managing free space in a file system.

What are the advantages and disadvantages of this approach?
(i.e., compare to other methods of managing free storage)

Your Answer:

Bitmap or Bit vector –
A Bitmap or Bit Vector is series or collection of bits where each bit corresponds to a disk block. The bit can take two values: 0 and 1: *0 indicates that the block is allocated* and 1 indicates a free block.
The given instance of disk blocks on the disk in *Figure 1* (where green blocks are allocated) can be represented by a bitmap of 16 bits as: **0000111000000110**.

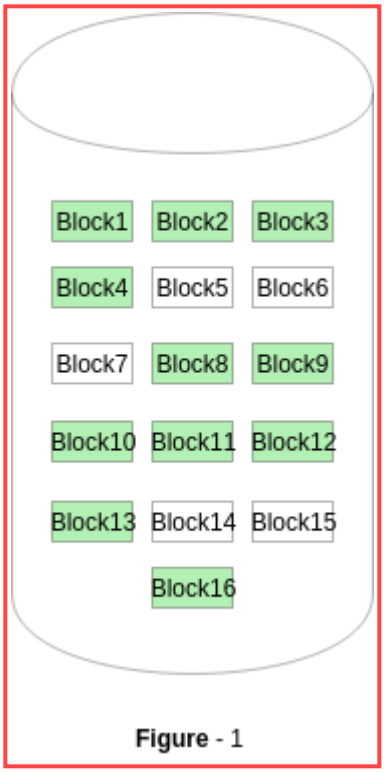


Figure - 1

When the **operating system** **(https://en.wikipedia.org/wiki/Operating_system)** (OS) needs to write a file, it will scan the bitmap until it finds enough free locations to fit the file. If a 12 KiB file were stored on the example drive, three zero bits would be found, changed to ones, and the data would be written across the three sectors represented by those bits. If the file were subsequently truncated down to 8 KiB, the final sector's bit would be set back to zero, indicating that it is again available for use.

Advantages

- Simple: Each bit directly corresponds to a sector
- Fast random access allocation check: Checking if a sector is free is as simple as checking the corresponding bit
- Fast deletion: Data need not be overwritten on delete; flipping the corresponding bit is sufficient
- Fixed cost: Both an advantage and disadvantage. Other techniques to store free space information have a variable amount of overhead depending on the number and size of the free space extents. Bitmaps can never do as well as other techniques in their respective ideal circumstances, but don't suffer pathological cases either. Since the bitmap never grows, shrinks or moves, fewer lookups are required to find the desired information
- Low storage overhead as a percentage of the drive size: Even with relatively small sector sizes, the storage space required for the bitmap

is small. A 2 TiB (https://en.wikipedia.org/wiki/Tebibyte) drive could be fully represented with a mere 64 MiB (https://en.wikipedia.org/wiki/Mebibyte) bitmap.

Disadvantages

- Wasteful on larger disks: The simplistic design starts wasting large amounts of space (in an absolute sense) for extremely large volumes[1] (https://en.wikipedia.org/wiki/Free_space_bitmap#cite_note-bonwick-1)
- Poor scalability: While the size remains negligible as a percentage of the disk size, finding free space becomes slower as the disk fills. If the bitmap is larger than available memory (https://en.wikipedia.org/wiki/Random-access_memory), performance drops precipitously on all operations[1] (https://en.wikipedia.org/wiki/Free_space_bitmap#cite_note-bonwick-1)
- Fragmentation (https://en.wikipedia.org/wiki/File_system_fragmentation): If free sectors are taken as they are found, drives with frequent file creation and deletion will rapidly become fragmented. If the search attempts to find contiguous blocks, finding free space becomes much slower for even moderately full disks.

NOTE - Other way

Linked List –
In this approach, the free disk blocks are linked together i.e. a free block contains a pointer to the next free block. The block number of the very first disk block is stored at a separate location on disk and is also cached in memory.

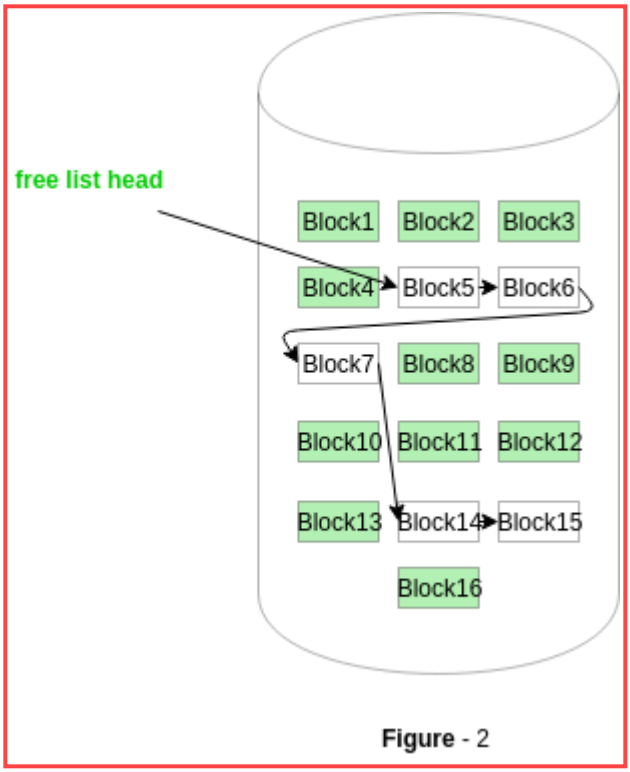


Figure - 2

In *Figure-2*, the free space list head points to Block 5 which points to Block 6, the next free block and so on. The last free block would contain a null pointer indicating the end of free list.
A drawback of this method is the I/O required for free space list traversal.

Question 7

0 / 0 pts

If you have a disk with 1,048,576 blocks, how many bytes do you need to allocate space for the bit vector in a file system using a bit vector-based management of free space assuming that a single bit in a byte indicates the status (free or taken) of one disk block?

Assuming that bit `1` is used to indicate a free block, and bit `0` to indicate a taken block, write a C function that implements the algorithm to find, and to claim, the first (from the beginning of the logical address space) free block. The function should:

- use bit-wise and bit shifting operations to run fast,
- flip the corresponding bit to indicate that the block is taken (not free anymore), and
- return the LBA (logical block address) of the free block to use.

Your Answer:

1 byte = 8 bits

and 1 bit representing the status (free or taken) of 1 block in disk.

So,

Total number of bits required = number of blocks = 1048576

Converting into bytes = 1048576/8 = 131072 bytes

C function to find the block number of first free block:-

```
// Assuming size of "int" data type is 4 bytes.
// So value of n = 32768.
int find(int bitmap[], int n)
{
   for(int i = 0; i < n; i++)
   {
      // If all the bits of the integer presented at i'th index are '1' means,
      // All blocks are taken represented by bits of this integer.
      // Otherwise some blocks are free.
      if(bitmap[i] < 4294967295)
      {
         // Assuming that system is big endian.
         int j = 31;
         while(j >= 0 && ((1 << j) && bitmap[i]) != 0)
         {
            j--;
         }
         return (i*32 + 31 - j);

         /* If system is little endian, remove the above code and uncomment
below code segment.
         int j = 0;
         while(j < 32 && ((1 << j) && bitmap[i]) != 0)
         {
            j++;
         }
         return (i*32 + j);
         */
      }
```

```
  }
}
```

Question 8

0 / 0 pts

Describe what Virtual File System is. What's its purpose? How is it used?

Your Answer:

A virtual file system (VFS) or virtual filesystem switch is an abstraction layer on top of a more concrete file system. The purpose of a VFS is to allow client applications to access different types of concrete file systems in a uniform way. A VFS can, for example, be used to access local and network storage devices transparently without the client application noticing the difference. It can be used to bridge the differences in Windows, classic Mac OS/macOS and Unix filesystems, so that applications can access files on local file systems of those types without having to know what type of file system they are accessing.

A VFS specifies an interface (or a "contract") between the kernel and a concrete file system. Therefore, it is easy to add support for new file system types to the kernel simply by fulfilling the contract. The terms of the contract might change incompatibly from release to release, which would require that concrete file system support be recompiled, and possibly modified before recompilation, to allow it to work with a new release of the operating system; or the supplier of the operating system might make only backward-compatible changes to the contract, so that concrete file system support built for a given release of the operating system would work with future versions of the operating system.

Question 9

0 / 0 pts

Describe with details how a log-structured file system works.

Your Answer:

Log structured file systems write information to media into no blocks without rewriting. The file structure is a circular list buffer where new or modified information is added always at the head and when a new version is written, then the old data is marked as obsolete. A "garbage collector is run to make room at the tail. If there are no blocks with obsolete information then the disk is full. Even if the file system crashes then the system is guaranteed to remain consistent since the old data is still there and will only be made obsolete if the new data is successfully written. It is possible to use logging in parts of a file system.

SUMMARY of why we use log-structured file systems:

Summary: The File Systems That We Have Loved
• The original Unix file system had a simple
design, but was slow
- Data layout did not provide spatial locality for
directories and files with temporal locality
- Only provided 4% of the sequential disk
bandwidth
• FFS leveraged knowledge of disk geometry
to improve performance
- To reduce seeks:
-Related files and directories are stored in the
same cylinder group
-Block size increased from 512 bytes to 4KB
- To minimize rotational latency, FFS used “skip
sectors”

• However, FFS had slow failure recovery due
to the horrifying multi-pass nature of fsck

• Journaling file systems use write-ahead logging
to make crash recovery faster
- ext3 performs redo logging of physical blocks
- NTFS performs redo+undo logging of operations
- The journal converts random writes into sequential
ones, but the file system must eventually issue
random writes to perform checkpoints

• LFS turns the entire file system into a log
- Assumes that the buffer cache will handle most reads, so making writes fast is the most important thing
- The log turns all writes (random or sequential, large or small) into large sequential writes
- Fast recovery
- Requires garbage collection, which (depending on the workload) may eliminate benefits from making all writes sequential

Question 10

0 / 0 pts

Describe with details how journaling file system provides data protection from system crashes.

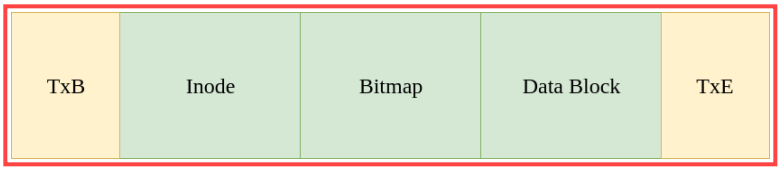
Your Answer:

ournaling, or **write-ahead logging** is a sophisticated solution to the problem of **file system inconsistency (https://www.geeksforgeeks.org/file-system-inconsistency/)** in operating systems. Inspired by database management systems, this method first writes down a summary of the actions to be performed into a “log” before actually writing them to the disk. Hence the name, “write-ahead logging”. In the case of a crash, the OS can simply check this log and pick up from where it left off. This saves multiple disk scans to fix inconsistency, as is the case with FSCK.

Good examples of systems that implement data journaling include Linux ext3 and ext4 file systems, and Windows NTFS.

Data Journaling:

A log is stored in a simple data structure called the journal. The figure below shows its structure, which comprises of three components.



1. **TxB (Transaction Begin Block):**
   This contains the transaction ID, or the TID.
2. **Inode, Bitmap and Data Blocks (Metadata):**
   These three blocks contain a copy of the contents of the blocks to be updated in the disk.
3. **TxE (Transaction End Block)**
   This simply marks the end of the transaction identified by the TID.

As soon as an update is requested, it is written onto the log, and thereafter onto the file system. Once all these writes are successful, we can say that we have reached the **checkpoint** and the update is complete.

What if a crash occurs during journaling ?

One could argue that journaling, itself, is not atomic. Therefore, how does the system handle an un-checkpointed write ? To overcome this scenario, journaling happens in two steps: simultaneous writes to TxB and the following three blocks, and then write of the TxE. The process can be summarized as follows.

1. **Journal Write:**
   Write TxB, inode, bitmap and data block contents to the journal (log).
2. **Journal Commit:**
   Write TxE to the journal (log).
3. **Checkpoint:**
   Write the contents of the inode, bitmap and data block onto the disk.

A crash may occur at different points during the process of journaling. If a crash occurs at step 1, i.e. before the TxE, we can simply skip this transaction altogether and the file system stays consistent.

If a crash occurs at step 2, it means that although the transaction has been logged, it hasn't been written onto the disk completely. We cannot be sure which of the three blocks (inode, bitmap and data block) were

actually updated and which ones suffered a crash. In this case, the system scans the log for recent transactions, and performs the last transaction again. This does lead to redundant disk writes, but ensures consistency. This process is called **redo logging**.

Question 11

0 / 0 pts

Describe with details how copy-on-write file system provides data protection from crashes.

Copy on Write.docx
(https://cilearn.csuci.edu/files/2295535/download)

Question 12

10 / 10 pts

I have submitted answers to all questions in this study set.

- (•) True
- ( ) False

Quiz Score: **10** out of 10