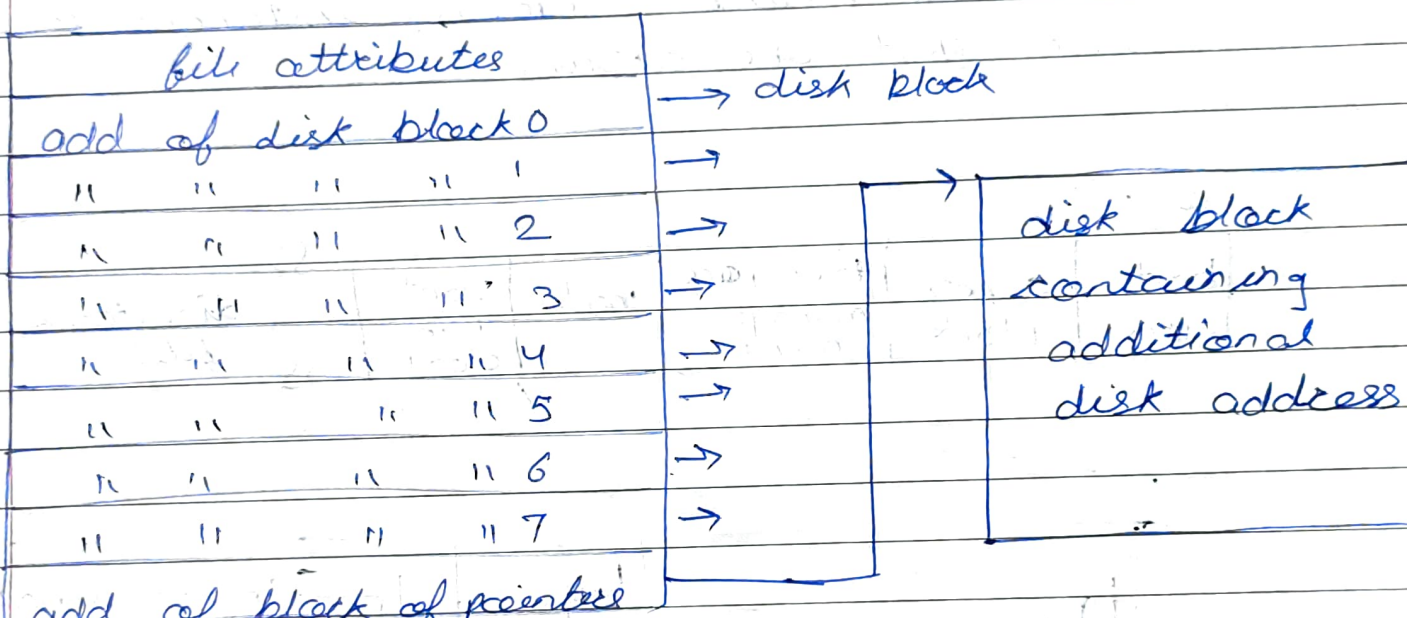
PAGE NO.:
DATE: / /

OS - 2.

Q.1. What is inode? List fields from disk inode & in-core inode.

→ * Inode -
An inode (index-node) lists the attribute & disk addresses of the file's blocks.
I-node exists in a static form on disk & the kernel reads them into an in-core i-node.



* Fields of disk ~~and~~ inodes -
(1) file owner identifier → an individual owner & a grp owner.
(2) file type → reg, directory, character/block special etc.
(3) file access permissions → owner/grp/others read/write/execute.
(4) file access times → last modified/accessed
(5) no. of links to file → no. of names.
(6) table of contents for the disk address
(7) file size.

PAGE NO.:
DATE : / /

* fields of in-core inode -

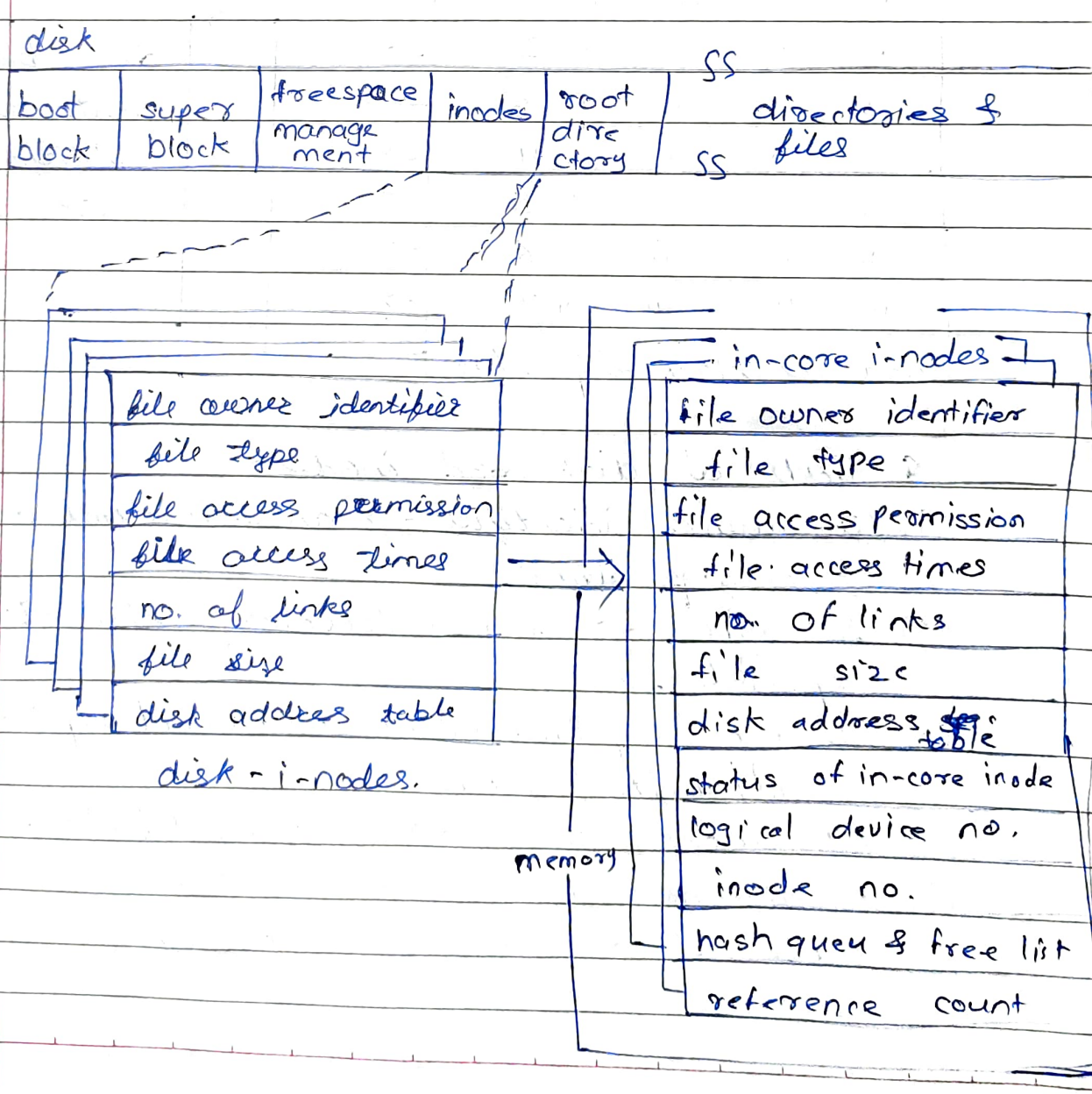
(1) status of in-core inode indicating :-
- the inode is locked
- a process is waiting for the inode to become unlocked.

(2) logical device no. of file system

(3) inode no.

(4) pointers to other in-core i-node : hash queue & free list.

(5) reference count : no. of instances of the file that are active.



PAGE NO.:
DATE: / /

Q.1. Explain algorithm for allocation of in-core inodes. (iget algo).

→ algorithm iget
input :- inode no.
output: locked inode
{
while (not done) {
if (inode in inode cache) {
if (inode locked) {
sleep (event inode becomes unlocked);
continue;
}
if (inode on inode free list)
remove from free list;
increment inode reference count;
return (inode);
}
if (no inodes on free list)
return (error);
remove new inode from free list;
reset inode no & file system;
remove inode from old hash queue,
place on new one;
read inode from disk;
(bread)
initialize inode;
return (inode);
}
}

PAGE NO.:
DATE : / /

The kernel identifies particular inodes by their file system & inode no & allocates in-core inodes at the request of higher level algorithms. The algorithm iget allocates in-core copy of inode is identical to getblk algorithm. The kernel maps the device number & inode number into a hash queue & searches the queue for the inode. If it cannot find the inode, it allocates one from the free list & locks it.
If the kernel attempts to take an inode from free list but finds the free list empty, it reports error.

Q.3. Explain algorithm for releasing inode (iput algo).

→ algorithm iput
input: pointer to in-core inode
output: none
{
lock inode if not already locked;
decrement inode reference count;
if (reference count == 0) {
if (inode link count == 0) {
free disk blocks for file (free);
set file type to 0;
free inode (ifree);
}
if (file accessed or inode changed or file changed)
update disk inode;
put inode on free list;
}
release inode lock;
}

When the kernel releases an inode, it determines its in-core reference count. If it drops to 0, the kernel writes inode to disk if in-copy differs from disk copy. They differ if the file data/file access time/file owner has changed.
The kernel places inode on free list. The kernel may also release all data block associated with file.

Q.4. Explain structure of regular file.

→ Unix has 13 entries in the inode table of contents:
- 10 "direct" entries
- 1 "single indirect" entry
- 1 "double indirect" entry
- 1 "triple indirect" entry

Assume that logical block holds 1k bytes & that a block number is addressable by a 32 bit integer.
- 10 direct blocks with 1k bytes each : 10kB
- 1 indirect block with 256 direct blocks : 256kB
- 1 double indirect block with 256 indirect blocks : 64MB
- 1 triple indirect block with 256 double indirect blocks : 16GB

PAGE NO.:
DATE: / /

Q.6. Explain process of conversion of pathname to inode with algorithm "namei".

→ - The initial access to a file is by its path name ; the kernel converts the path names to i-nodes.
- The algorithm namei parses the path name one component at a time, converting each component into i-node based on its names & directory being searched, & eventually returns the inode of the input path name.
- Every process is associated with a current directory ; the uarea contains a pointer to the current directory inode.

```
algorithm namei
input : path name
output : locked inode
{
if (path name starts from root)
    working inode = root inode ; (iget)
else
    working inode = current dir inode ; (iget)
while (there is more path name) {
    read next path name component ;
    verify that working inode is of dir,
            access permissions OK ;
    if (working inode is of root &
            component is "..")
        continue ;
    read directory by repeated use of
        algorithms bmap, bread & brelse ;
    if (component matches an entry in
        directory (working inode)) {
```

PAGE NO.:
DATE: / /

```
        get inode no. for matched component;
        release working inode; (iput);
        working inode = inode of matched
                                component (iget)
    }
    else
        return (no. inode);
    }
    return (working inode);
}
```

- The kernel does a linear search of directory file associated with the working inode.
  - Starting at byte offset 0, the kernel converts the byte offset in the directory to the appropriate disk block & according to bmap.
  - If the kernel finds the path name component in the directory block, it records the inode number of the matched directory entry.
  - If the kernel does not match the path name in the block, it converts the new offset to a disk block no. (bmap) & reads the next block.

PAGE NO.:
DATE : / /

Q.7. What is superblock? List fields of superblock.

→ - The superblock binds an inode to a file & allocates disk blocks to files.

- Fields of superblock.

(1) The size of the file system
(2) the no. of free blocks in the file system
(3) a list of free blocks available on the file system
(4) the index of next free block in the free block list
(5) the size of inode list
(6) the no. of free inodes in file system
(7) list of free inodes in the file system
(8) the index of next free inode in the file system
(9) lock fields for free block & free inode lists
(10) a flag indicating that the super block has been modified.

Q.8. What is remembered inode? How it is useful in inode assignment to a file.

(OR)

Explain algorithm for assignment new inodes (ialloc algo).

→ The kernel reads the inode list on disk, block by block & fills the super block list of inode numbers to capacity, remembering the highest numbered inode that it finds. This is called remembered inode.

PAGE NO.:
DATE: / /

```
algorithm ialloc
input: file system
output: locked inode
{
 while (not done) {
    if (super block locked) {
          sleep (event super block becomes
            free);
          continue;
          }
    if (superblock inode list is empty) {
          lock super block;
          get remembered inode for free inode
                    search;
          search disk for free inode until
                    super block full; (bread, brelse)
          unlock super block;
          wakeup (event super block becomes
            free);
          if (no free inodes found on disk)
            return (no inode);
          set rembered inode for next free
              inode search;
          }
    get inode no. from super block inode list;
    get inode; (iget)
    if (inode not free after all) {
          write inode to disk;
          release inode; (iput)
          continue;
          }
    initialize inode;
    write inode to disk;
```

```
        decrement file system free inode count;
        return (inode);
    }
}
```

- The kernel first verifies that no other processes have locked the super block free inode list.
- If the super block free inode list is not empty, the kernel assigns the next inode no, allocates a free in core inode for the newly assigned disk inode by iget.
- It updates the disk inode to indicate to indicate that disk inode is now in use.
- If the super block free inode is empty, the kernel searches the disk & places as many free inode no.s as possible into super block.
- The kernel reads the inode list on disk, block by block & fills the super block list of inode no. to capacity, remembering the highest numbered inode that it finds.
- The next time the kernel searches the disk for free inodes, it uses the remembered inode as its starting point.
- After gathering a fresh set of inode numbers, it starts the inode assignment algorithm from the beginning.

Q. 9. Explain algorithm for freeing inode (ifree algo).

→ algorithm ifree
input: file system inode no.
output: none
{
increment file system free inode count;
if (super block locked)
return;
if (inode list full) {
if (inode no. less than remembered inode for search)
set remembered inode for search = input inode no.;
} else
store inode no. in inode list;
return;
}

After incrementing the total no. of available inodes in file system, the kernel checks the lock on the super block.
If locked, it avoids race conditions by returning immediately; the inode no. is not put into the super block, but it can be found on disk.
If the list is not locked, the kernel checks if it has room for more inode no.s.
If the list is full, the kernel may not save the newly freed inode there; it compares the no. of freed inode with that of the remembered inode
If the freed inode no. is less than the

remembered inode no., it remembers the newly freed inode no.

3.10. State & explain the algorithm "alloc" for disk blocks.

→ algorithm alloc
input: file system number
output: buffer for new block
{
while (super block locked)
    sleep(event super block not locked);
remove block from super block free list;
if (removed last block from free list) {
    lock super block;
    read block just taken from free list (bread);
    copy block no into super block;
    release block buffer (brelse);
    unlock super block;
    wakeup processes (event super block
                not locked);
}
get buffer for block removed from super
 block list (getblk);
zero buffer contents;
decrement total count of free blocks;
mark super block modified;
return buffer;
}

PAGE NO.:

DATE: / /

- When a process writes data to a file, the kernel must allocate disk block from file system.
- The super block contains an array that is used to cache the numbers of free disk blocks.
- When the kernel wants to allocate a block from a file system by alloc, it allocates the next available block in super block list.
- If the allocated block ~~list~~ is last available block in the super block cache, the kernel treats it as a pointer to a block that contains list of free blocks.
- It allocates a buffer for the block & clears the buffer's data.
- If the file system contains no free blocks, the calling process receives an error.