**Practical - 5**

**Aim:-Case Study on Various Inode Algorithms**

**INTRODUCTION:**

The **inode (index node)** is a fundamental concept in the Linux and UNIX file system.Every file on a UNIX system has a unique inode (contains information necessary for a process to access a file).Inodes exist in a static form on disk and the kernel reads them into an in-core inode to manipulate them.

**Inode Consist of:**

• File owner identifier (individual owner & group owner)

• File type (regular, directory, character or block special, or FIFO – pipes)

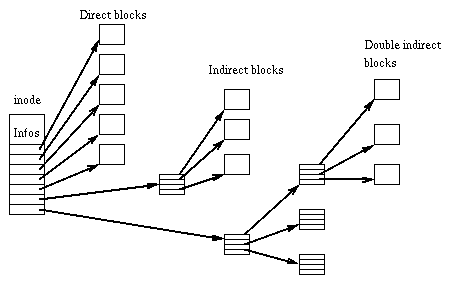
• File access permissions (owner, group, other: read, write, execute)

• File access time

• Number of links to the file

• Table of contents for the disk addresses of data in a file

• File size



**Algorithm for allocation of in-core inodes:**

algorithmiget

input: file system inode number

output: locked inode

{ while (not done)

{ if (inode in inode cache)

{ if (inode locked)

{

sleep (event inode becomes unlocked);

continue; /\*loop back to while \*/

}

/\*special processing for mount points \*/

if (inode on inode free list)

remove from free list;

increment inode reference count;

return (inode);

}

/\*inode not in inode cache \*/

if (no inodes on free list)

return (error);

remove new inode from free list;

reset inode number and fiel system;

remove inode from old hash queue, place on new one;

read inode from disk (algorithm bread);

initialise inode (e.g. reference count to 1);

return (inode);

}

}

**Algorithm for conversion of path name to an inode:**

algorithmnamei /\* convert path name to inode \*/

input: path name

output: locked inode

{

if (path name starts from root)

working inode = root inode (algorithm iget);

else

working inode = current directory inode (algorithm iget);

while (there is more path name)

{

read next path name component from input;

verify that working inode is of directory, access permissions OK;

if (working inode is of root and component is “..”)

continue; /\*loop back to while \*/

read directory (working inode) by repeated use of algorithms

bmap, bread and brelse;

if (component matches an entry in directory (working inode))

{

get inode number for matched component;

release working inode (algorithm iput);

working inode = inode of matched component (algorithm iget);

}

else /\*component not in directory \*/

return (no inode);

}

return (working node);

}

**Algorithm for assigning new inodes:**

algorithmialloc /\* allocate inode \*/

input: file system

output: locked inode

{ while (not done)

{ if (super blocked locked)

{ sleep (event super block becomes free);

continue; /\*loop back to while \*/

}

if (inode list in super block is empty)

{ lock super block;

get remembered inode for free inode search;

search disk for free inodes until super block full, or no more

freeinodes (algorithms bread and brelse);

unlock super block;

wake up (event super block becomes free);

if (no free inodes found on disk)

return (no inode);

set remembered inode for next free inode search;

}

/\*there are inodes in super block inode list \*/

get inode number form super block inode list;

get inode (algorithm iget);

if (inode not free after all)

{ write inode to disk;

release inode (algorithm iput);

continue;

}

/\*inode is free \*/

initialise inode;

write inode to disk;

decrement file system free inode count;

return (inode);}

}