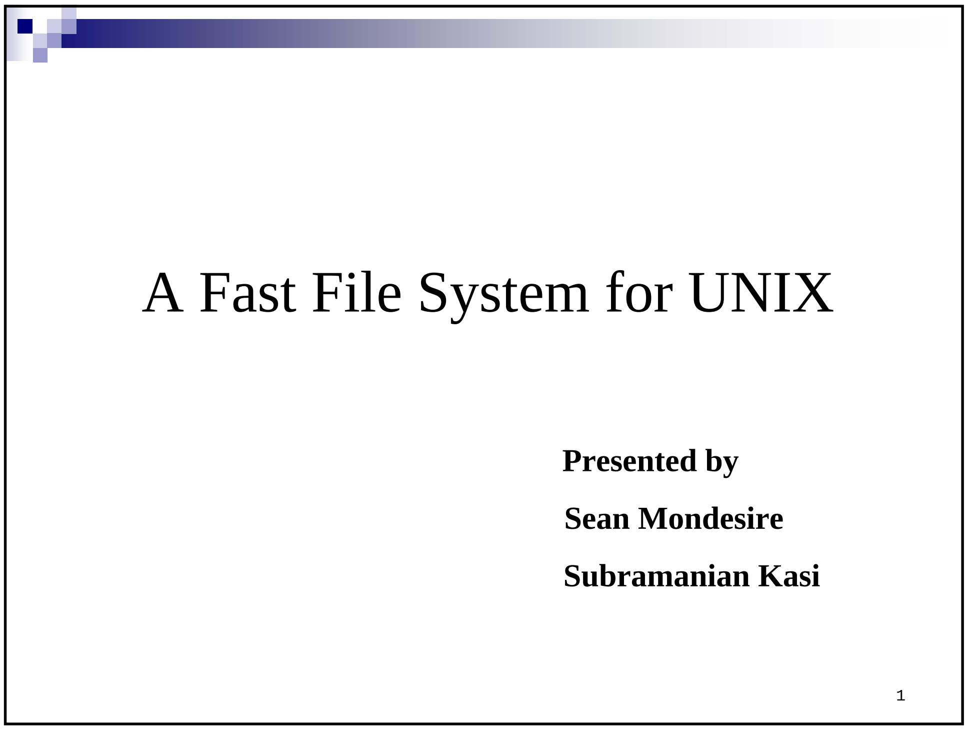


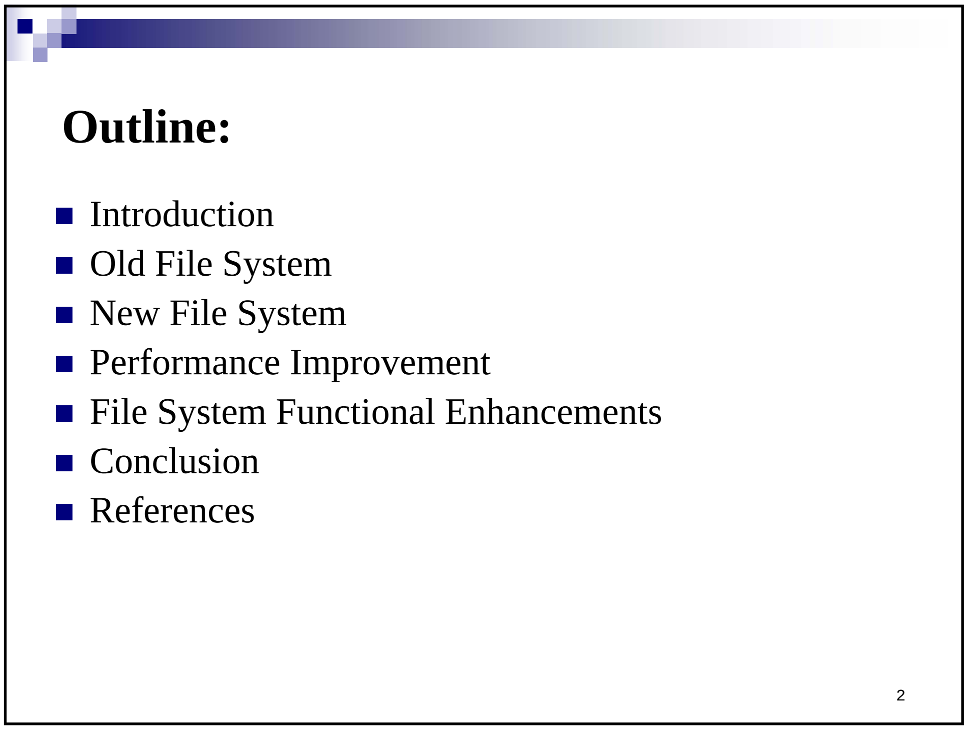
A Fast File System for UNIX

Presented by

Sean Mondesire

Subramanian Kasi

1



Outline:

- Introduction
- Old File System
- New File System
- Performance Improvement
- File System Functional Enhancements
- Conclusion
- References

2

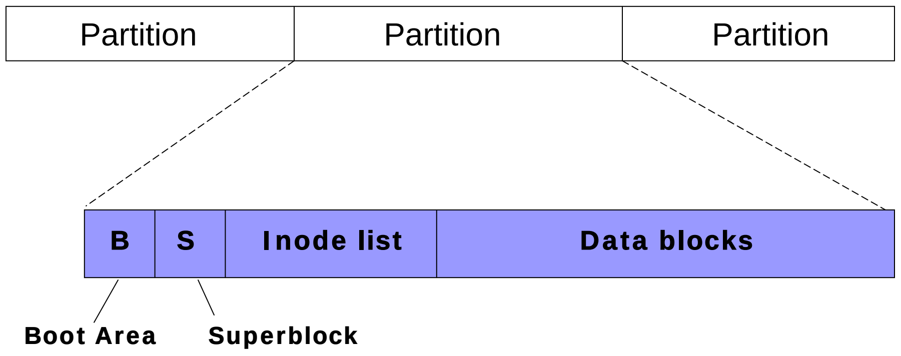
Introduction

- The Fast File system was developed by the Computer Systems Research Group (CSRG) at the University of California Berkeley
- The work was done under grants from the NSF and ARPA
- The main goal was increase the throughput of old 512-byte UNIX file system by changing the underlying implementation.

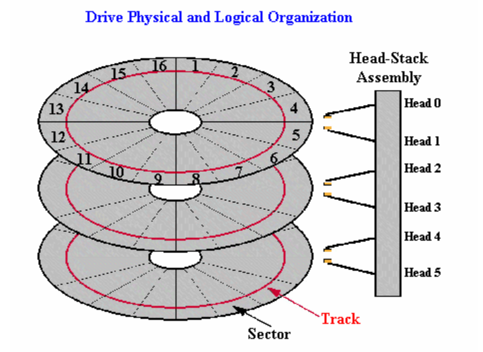
Old File System

- Each disk drive is divided into one or more "*partitions*"
- Each partition has one File System
- File system consists of
  - Boot Area
  - Super block
  - Inode list
  - Data blocks.

Old File System



Old File System



- One or more sectors in a track are grouped to form a "*data block*"

Old File System

- The **boot area** stores objects that are used in booting the system.
- If a file system is not to be used for booting, the boot area is left blank
- **Superblock** contains basic parameters of the file system.

- number of data blocks in the file system

- maximum number of files

- pointer to the *free list* – A link list of all free blocks in a system. Traversed during block allocation for a file

7

Old File System

- Within the file system are *files*
- Each file is described by an *inode*
- An inode contains information about:

-ownership information

-time stamps

-array of indices to data blocks

8

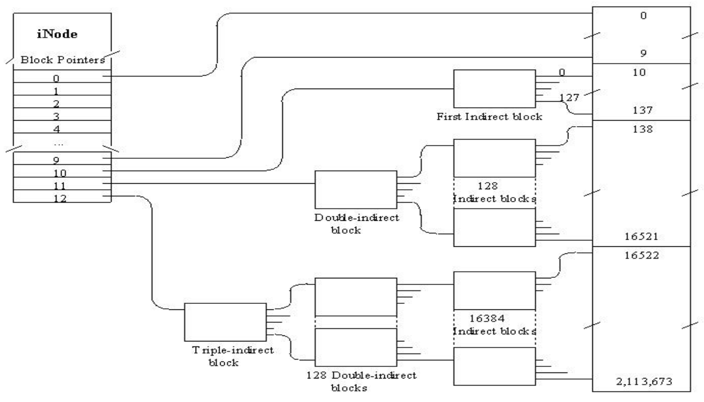
Old file System

```
 struct  inode
{
   u_short   di_mode;      /* mode and type of file */
  short       di_nlink;       /* number of links to file */
  short       di_uid_lsb;   /* owner's user id */
  short       di_gid_lsb;   /* owner's group id */
  quad        di_size;        /* number of bytes in file */
  time_t      di_atime;     /* time last accessed */
  long         di_atspare;
  time_t      di_mtime;     /* time last modified */
  long         di_mtspare;
  time_t      di_ctime;     /* time of last file status change */
  long         di_ctspare;
  daddr_t   di_db[NDADDR];  /* disk block addresses */
  .
  .
};
```

9

Old File System

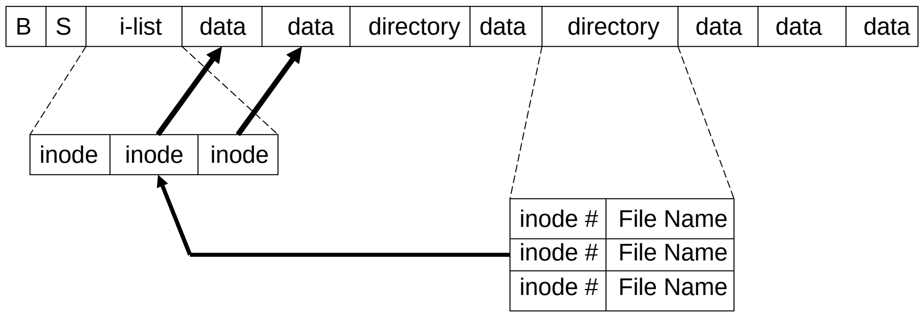
- An inode may contain references to single, double and triple indirect blocks.



10

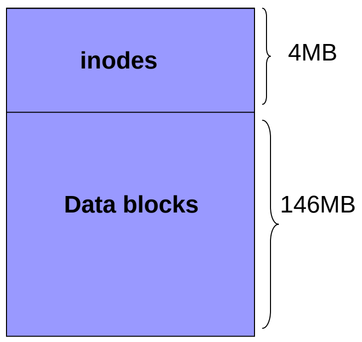
Old File System

- Certain files are distinguished as directories which contain a list of file names and their corresponding inodes



Old file System-Layout Problems

- A 150MB traditional file system contains
  - 4 MB of inodes
  - 146 MB of data blocks
  - causes long seek time from files inode to its data.
  - files within a directory are not allocated sequential slots in the 4MB of inodes.



Old File System

- Disk transfers were only 512 byte (block size)
- Next sequential data block not on the same cylinder causes seek time between transfers
- Reason: Due to suboptimum allocation of data blocks to files.
- Problem with free list- Scrambled

  free list – A link list of all free blocks in a file system stored in the superblock.

13

Old File System

- Free list initially ordered
- As files created and deleted became scrambled.
- Eventually became totally random
- Files had their block randomly distributed over the disk.
- Caused seek time for every block access
- 175 kb/s (initial) $\Rightarrow$ 30kb/s

14

Old File System

Summary of problems:

- Long seek time from inode to actual data
- Files in a directory not allocated consecutive slots in the inode list
- Small block size (512 bytes)
- Allocation of blocks to a file suboptimum
- Resulted in too many seeks between block transfers.

15

Old File System

- First work at Berkeley was to increase the block size from 512 to 1024
- File system performance doubled!- though it was only using 4% of disk bandwidth
- Reason:

  - Each disk transfer twice as much data

  - most files described without need to

    access indirect blocks
- Good indication that increasing block size helps

16

New File System

- Like the Old file system each disk drive contains one or more file systems
- The file system is described by the *superblock*
- *Superbock* is replicated to protect against failure
- Since information present in *superblock* is static no need to access copies unless default superblock becomes unusable.

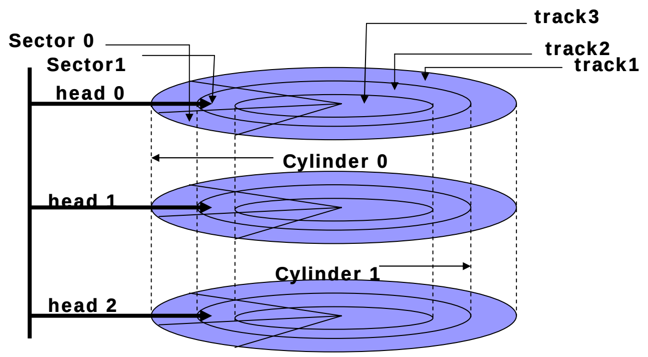
17

New File System

- Larger block size : 4096 bytes
- Block size for each file system recorded in superblock
- File system with different block sizes can be accessed on the same system.
- Decision of the block size made at time the file system is created.

18

New File System



- Tracks with the same radius on different platters form a cylinder
- New file system divided a disk partition into one or more cylinder groups – consecutive cylinders

New File System

- Each cylinder group contains bookkeeping information.

- Redundant copy of *superblock*

- *bit map* of available blocks in the cylinder group (replaced the free list)

- summary information describing the usage of data blocks.

New File System

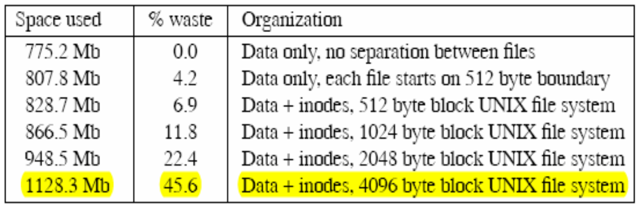
- All cylinder group information could be kept at the top platter $\Rightarrow$ all copies of superblock information on top platter.
- Failure of top platter causes loss of all copies of the superblock
- Solution: Bookkeeping information for each cylinder group at an offset from the previous group - spiral structure

21

New File System

(Optimizing storage utilization) :

- Problem with large block size –Unix systems are composed of many small files.



Space used	% waste	Organization
775.2 Mb	0.0	Data only, no separation between files
807.8 Mb	4.2	Data only, each file starts on 512 byte boundary
828.7 Mb	6.9	Data + inodes, 512 byte block UNIX file system
866.5 Mb	11.8	Data + inodes, 1024 byte block UNIX file system
948.5 Mb	22.4	Data + inodes, 2048 byte block UNIX file system
1128.3 Mb	45.6	Data + inodes, 4096 byte block UNIX file system

- Space wasted is calculated as % of space on the disk not containing user data
- As block size increases – waste increases
- 45.6 % waste for 4096-byte file system blocks!!

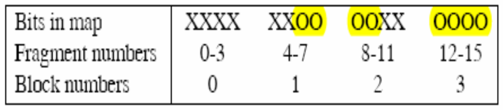
22

New File System

- Need to use large blocks without waste
- Solution: divide the blocks into one or more fragments
- Fragment size specified at the time file system is created
- Block can be broken into 2,4 or 8 fragments
- Lower bound of fragment size is the sector size
- Each individual fragment is addressable.

New File System

- The bit map present for each cylinder group contains the status of the fragments



Bits in map	XXXX	XX00	00XX	0000
Fragment numbers	0-3	4-7	8-11	12-15
Block numbers	0	1	2	3

Figure 1 – Example layout of blocks and fragments in a 4096/1024 file system.

- “X” - fragment in use
- “0” - fragment is available
- Fragments of adjoining blocks cannot be used as one block (6-9 cannot be used as one block)
- 12-15 can be used as one block

New File System

- Example: 11,000 byte file stored in a 4096/1024 file system
- Stored in two full size blocks 4096 x 2 = 8192
- One three fragment portion   1024 x 3 = 3072
- Total space allocated 11,264 as opposed to 12,288

New File System

- Space is allocated to a file every time a program executes a write system call.
- When a file needs to be expanded to hold new data one of the three condition exists.

1. There is enough space in an already allocated   block of fragment –new data written to available space

New File System

2.The file contains no fragmented blocks

the last block has insufficient space to hold new data.

- part of the data is written into the block

- If the remainder of the new data contains more than a full block, a full block is allocated first –data is written

- repeated until less than a full block remains

- if remaining data can fit in less than a block a block with necessary fragments is located

New File System

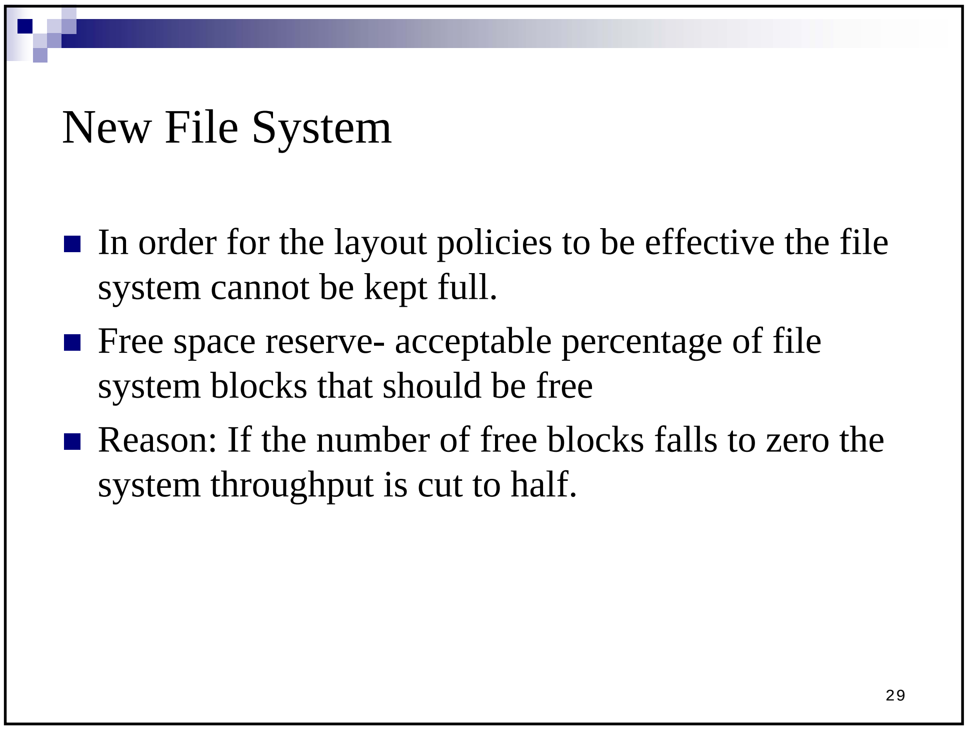
3. The file contains one or more fragments

if (sizeof (newdata) + data in fragments) > Size of a block

- the data in the fragment + the new data moved to a new block

- process continues as in 2

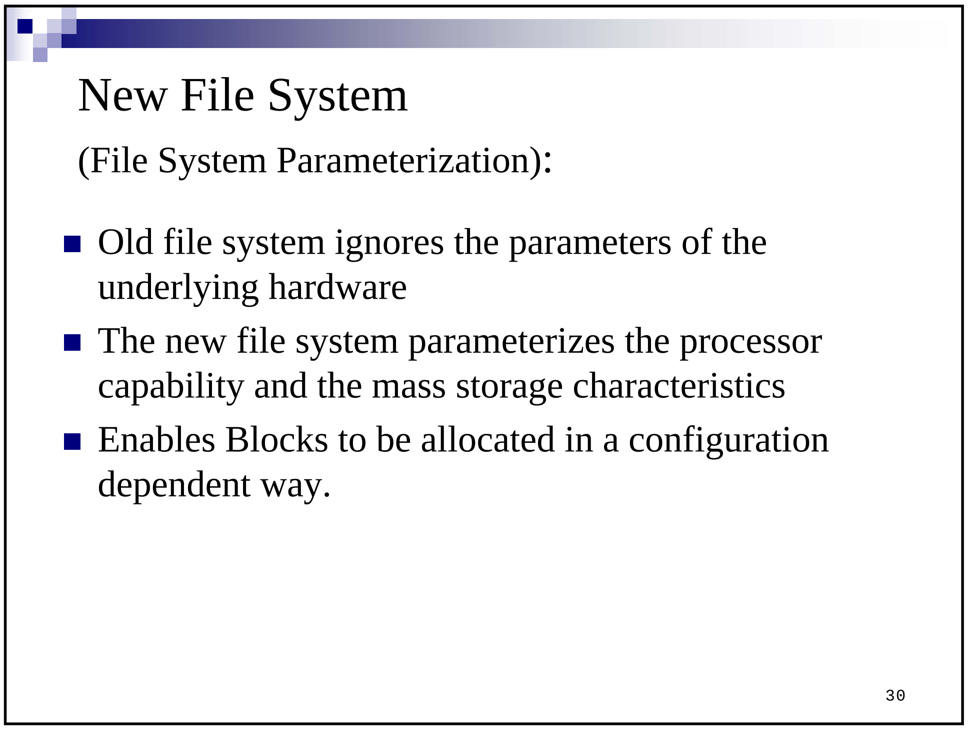
- Problem with expanding a file one fragment at a time
  - data may be copied too many times
- Solution: user program writes one full block at a time except for a block at the end of a file



New File System

- In order for the layout policies to be effective the file system cannot be kept full.
- Free space reserve- acceptable percentage of file system blocks that should be free
- Reason: If the number of free blocks falls to zero the system throughput is cut to half.

29



New File System

(File System Parameterization):

- Old file system ignores the parameters of the underlying hardware
- The new file system parameterizes the processor capability and the mass storage characteristics
- Enables Blocks to be allocated in a configuration dependent way.

30

New File System

Parameters considered:

- Speed of the processor
- Hardware support for mass storage transfers
- Characteristics of the mass storage device.

31

New File System

- Mass storage on disks
- Tries to allocate blocks on the same cylinder as the previous block in the file
- These blocks need to be rotationally well positioned
- Could mean consecutive blocks or rotationally delayed blocks

32

New File System

- If a processor with an I/O channel requires no processor intervention two consecutive blocks can be accessed without any delay
- A processor without an I/O channel will require processor intervention between the disk transfer to prepare for the next disk transfer

33

New File System

- Uses the physical characteristics of a disk like:
  - number of blocks /track
  - rate at which disk spins
- Processor characteristics
  - time to service an interrupt
  - time to schedule next disk transfer

34

New File System

- Using the processor and the disk characteristics
- Allocation routine calculates the number of blocks that needs to be skipped – so that next block in the file will come under the disk head at the appropriate time
- Minimizes the time spent waiting for the disk to position itself

35

New File System

- The cylinder group summary information includes a count of available blocks at different rotation positions
- Superblock contains a vector rotational layout table
- Each component in this table – lists the index into the block map for every data block in its rotational position.

36

New File System

- When looking for a block:
  - first looks through the summary counts for a rotational position with a non zero block count
  - uses the rotational position to index into the rotational layout table to find the list to use to find a free block

cylinder group summary information

Rotational position	Number of blocks available
1	5
2	2
3	1
5	0

Finds a non zero Rotational position from the group summary information

rotational layout table

Rotational position	List of blocks at this position
2	**1,5**
1	**2,7,8,9,10**
3	**11**
5	**0**

Uses the rotational position to index into the vector rotational layout table

New File System

- If a file system is parameterized to lay out blocks with a rotation separation of 2 ms
- If the processor requires 4 ms to schedule disk operations then wasted disk revolutions on every block - throughput drops
- In the new file system the rotation layout delay can be reconfigured based on the target machine.

39

Layout Policies

- The policies improve performance by:
  - Increasing the locality of reference to minimize seek latency
  - Improving the layout of data to make larger transfers possible
- Two types of policies:
  - Global policy routines
  - Local allocation routines

40

Global Layout Policies

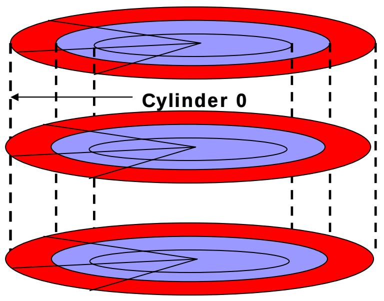
- Attempt to improve performance by clustering related information
  - Make decisions about the placement of new inodes and data blocks
  - Decide the placement of new directories and files
- Distribute unrelated data among different cylinder groups
  - For the fear of too much localization

Local Allocation Routines

- Called by the global policy routines with requests for specific blocks
- Always allocates the requested block if it is free
- If the requested block is not free then the four level allocation strategy must be used

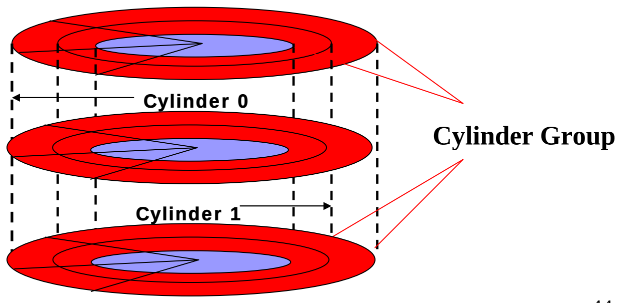
Four Level Allocation Strategy

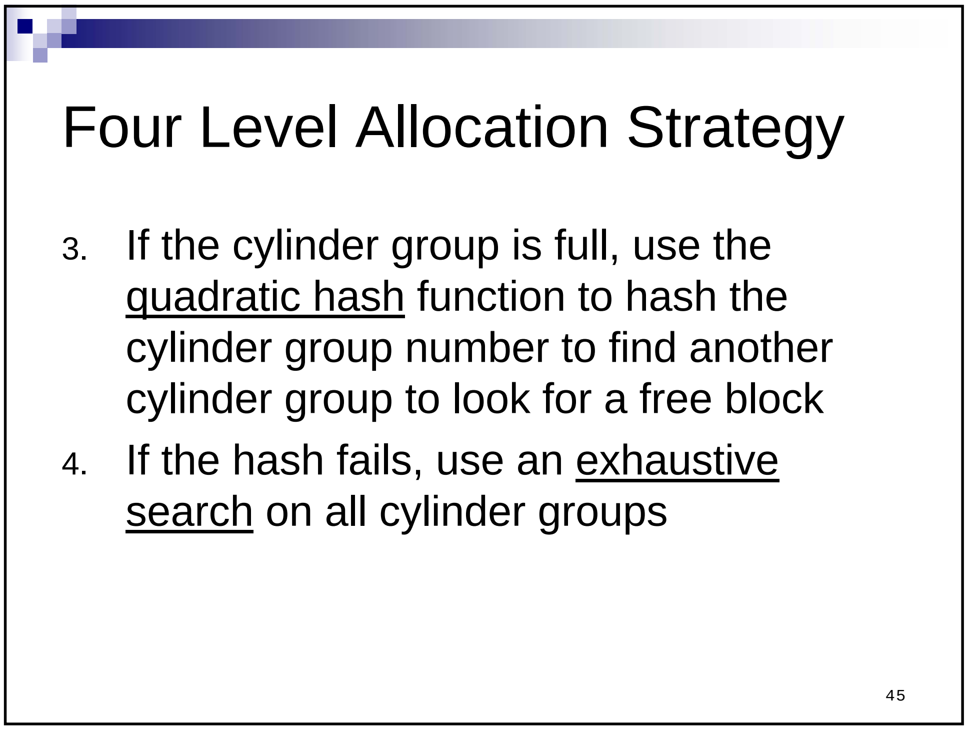
1. Use the next free block that is rotationally closest to the requested block on the <u>same cylinder</u>



Four Level Allocation Strategy

2. If there are no free blocks on the same cylinder, a free block in the <u>same cylinder group</u> is used

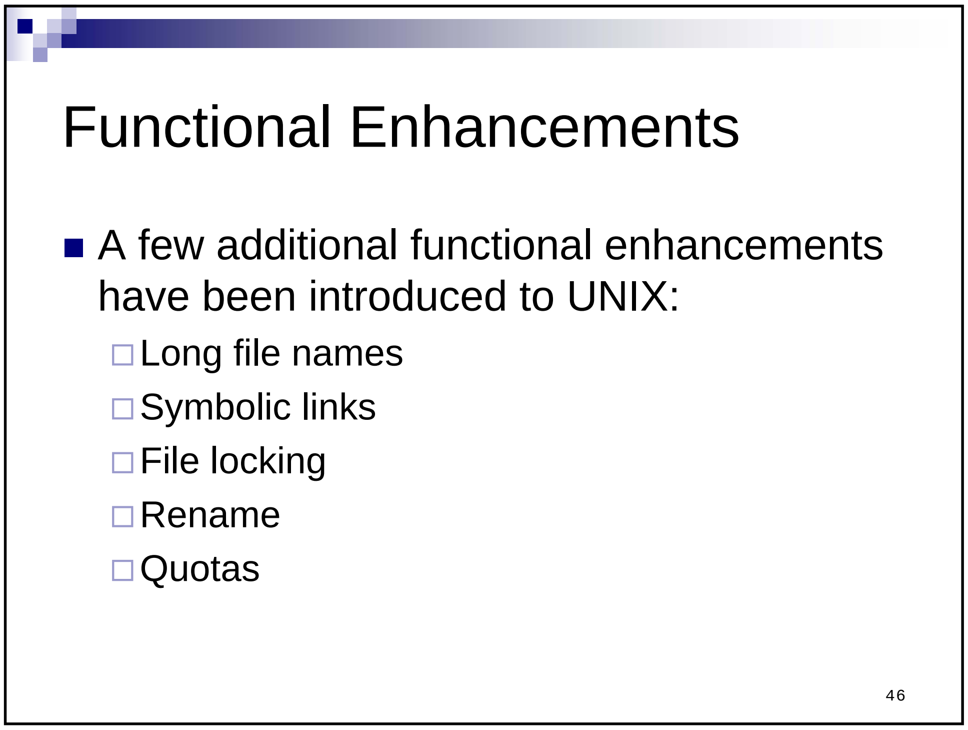




Four Level Allocation Strategy

3. If the cylinder group is full, use the quadratic hash function to hash the cylinder group number to find another cylinder group to look for a free block
4. If the hash fails, use an exhaustive search on all cylinder groups

45



Functional Enhancements

- A few additional functional enhancements have been introduced to UNIX:
  - Long file names
  - Symbolic links
  - File locking
  - Rename
  - Quotas

46

Functional Enhancements (cont.)

- Long File Names:
  - File names can be at most 255 characters
- Symbolic Links:
  - A file that contains the pathname of another file
  - Gives the illusion a remote file is actually local
  - The specified path can be either absolute or relative pathnames
    - Absolute: “C:\SchoolWork\Spring2005\COP5611\HW1.EXE”
    - Relative: “\COP5611\HW1.EXE”

47

Rename

- In the old file system, a file rename required 3 calls to the system to:
    - Create a new copy of the existing file
    - Rename the temporary file
- Posed a threat if the system crashed or interrupted
- FFS added the “rename” system call
  - Guarantees the target name file will be created
  - Handles renaming of files and directories

48

File Locking – Old FS

- Old file system locking:
  - Synchronized processes used a “lock” file
  - Successful locks allowed for immediate updates
  - Failure of lock creation forces the process to keep trying to create the lock file

49

File Locking – Old FS

- Disadvantages:
  - CPU time wasted during creation loop when a lock fails.
  - After a system crash, locks have to be manually removed
  - Since system admin processes can create files, they must use other means for locking

50

File Locking - FFS

- Fast File System's Locking Mechanism:
  - Advisory locks
    - Locks applied to files when a program requests it
    - Lock override is determined by the user program
    - Chosen since many programs need to use locks and run as the system administrator
    - Supports shared and exclusive locks on files

51

Quotas

- In the old file system, users can allocate as much resources as available
- FFS added a quota mechanism to limit the amount of resources a user can obtain
  - Limits the amount of inodes and disk blocks a user may allocate
  - When a user program exceeds its <u>soft limit</u>, a warning is displayed
  - When a user program exceeds its <u>hard limit</u>, it is terminated

52

Performance

- To compare the old file system to the Fast File System, the following measures were taken:
  - The rate a user program can transfer data to or from a file (read/write)
  - Disk utilization by the file system
  - CPU utilization

53

Experiment Conditions

- Processor used: VAX 11/750
- Buses: UNIBUS & MASSBUS
- Disk Drive: AMPEX Capricorn 330-MB Winchester
- Each file system was used for 1 month
- Each test had 10 percent of disk free space

54

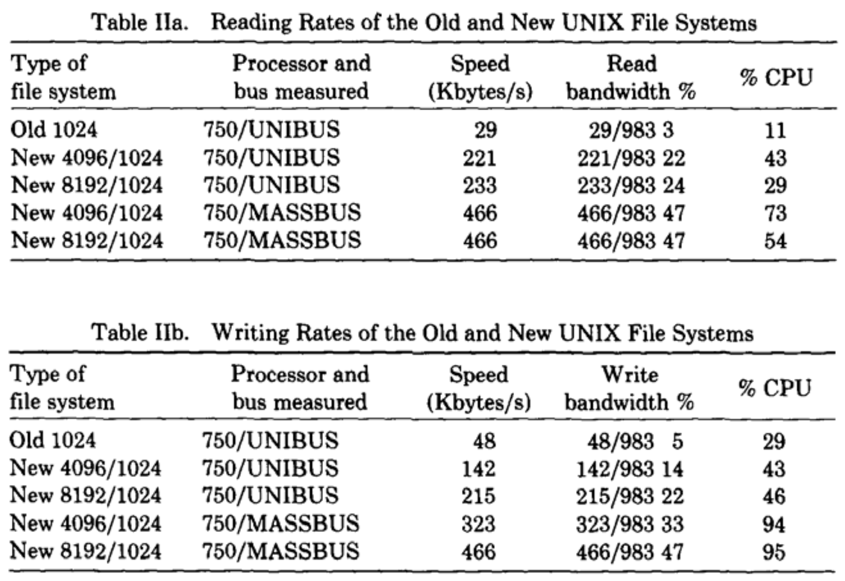


Table IIa. Reading Rates of the Old and New UNIX File Systems

Type of file system	Processor and bus measured	Speed (Kbytes/s)	Read bandwidth %	% CPU
Old 1024	750/UNIBUS	29	29/983 3	11
New 4096/1024	750/UNIBUS	221	221/983 22	43
New 8192/1024	750/UNIBUS	233	233/983 24	29
New 4096/1024	750/MASSBUS	466	466/983 47	73
New 8192/1024	750/MASSBUS	466	466/983 47	54

Table IIb. Writing Rates of the Old and New UNIX File Systems

Type of file system	Processor and bus measured	Speed (Kbytes/s)	Write bandwidth %	% CPU
Old 1024	750/UNIBUS	48	48/983 5	29
New 4096/1024	750/UNIBUS	142	142/983 14	43
New 8192/1024	750/UNIBUS	215	215/983 22	46
New 4096/1024	750/MASSBUS	323	323/983 33	94
New 8192/1024	750/MASSBUS	466	466/983 47	95

Performance: Fast File System

- Uses up to 47 percent of the disk bandwidth
  - Old file system used between 3 and 5 percent of the bandwidth
  - Reason: FFS has larger block sizes
- The read rate is always at least as fast as the write
  - Reason: the kernel must perform more processing to allocate block when writing
  - Old FF: 50 percent faster writes than reads

Performance (cont.)

- FFS has over 16 times faster read speeds than the old FS
- FFS has over 9 times faster write speeds than the old FS
- FFS throughput does not change over time
  - Only when disk has 10% free space
- Throughput decreases to near half the speed if the disk is full

Performance Explanations

- Blocks are more optimally ordered
  - Related data are grouped together
- Larger Blocks
  - Block sizes 4096 and 8192 bytes are used compared to 1024 bytes used in the old FS
  - Larger amounts of related data are pulled in less transfers

Future Expansions

- Better memory management techniques:
  - FFS performance is limited by memory copy operations
  - Current techniques inhibit speeds of accessing and moving data
- Techniques to allocate several blocks to a file at a time
  - Handles file expansion more gracefully
  - Reduces write allocation overhead

59

Conclusion

- New File System
  - Optimally places related data on disk
  - Increases amount of bytes transferred for a given data transfer
- Layout Policies
  - Global Layout Routines
  - Local Allocation Routines

60

Conclusion (cont.)

- Functional Enhancements
  - Longer Filenames
  - Symbolic Links
  - Rename System Call
  - File Locking
  - Quotas
- Performance Results
  - 16 times faster reads than the old file system
  - 9 times faster writes than the old file system

61

References

- McKusick, Marshall K., William N. Joy, Samuel J. Leffler, Robert S. Fabry, “A Fast File System for UNIX”
- McKusick, Marshall K.,“The Design and Implementation of the 4.4BSD Operating System”
- Morgan, David, “Analyzing a File System,” http://homepage.smc.edu/morgan_david/cs40/analyze-ext2.htm
- Nguyen, Thu D., “UNIX Fast File System”, http://www.cs.rutgers.edu/~tdnguyen/courses/cs519/fall2003/
- Duke University, Introduction to Operating Systems http://www.cs.duke.edu/courses/cps110/

62