

COMP3301

Lecture 9

File systems: implementation

School of Information Technology and Electrical Engineering
The University of Queensland

Last week

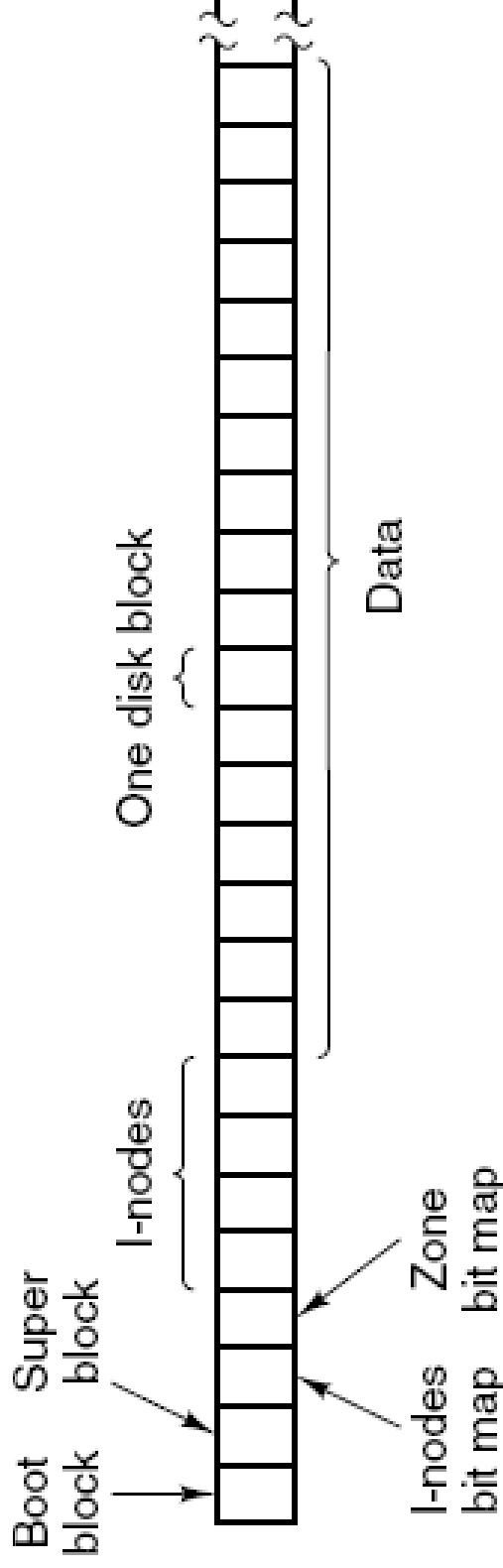
- Looked at generic file systems implementations
- possible file structures
- how these would be stored on disk
- This week: details of a simple file system: Minix 3

Minix filesystem

- As expected now:
 - user-mode process
 - communicates via messages
 - runs in a loop, receiving messages, doing work, replying
- Messages:
 - all system calls expected (open, chdir, mount etc)
 - some others from **pm** (exec, exit etc)

Minix file system

- Uses i-nodes
- Boot block for executable code
- Super block for system information
- bit maps and then i-nodes and data



- Figure 5-34. Disk layout for a floppy disk or small hard disk partition, with 64 i-nodes and a 1-KB block size (i.e., two consecutive 512-byte sectors are treated as a single block).

File system components

- **Boot block:**
 - always 1KB
 - contains magic number to indicate executable code
- **superblock**
 - also always 1KB
 - records how big all the rest of the pieces of the file system are

Minix 3 superblock

- Magic number identifies valid Minix file system

Present on disk and in memory	Number of i-nodes
	(unused)
	Number of i-node bitmap blocks
	Number of zone bitmap blocks
	First data zone
	\log_2 (block/zone)
	Padding
	Maximum file size
	Number of zones
	Magic number
	padding
	Block size (bytes)
	FS sub-version
	Pointer to i-node for root of mounted file system
Present in memory but not on disk	Pointer to i-node mounted upon
	i-nodes/block
	Device number
	Read-only flag
	Native or byte-swapped flag
	FS version
	Direct zones/i-node
	Indirect zones/indirect block
	First free bit: in i-node bitmap
	First free bit: in zone bitmap

Figure 5-35. The MINIX 3 superblock.

Bitmaps

- i-node bitmap:
 - records which of the i-nodes are used
 - create a file – find free i-node (which is stored in superblock)
- zone bitmap
 - zones are similar to blocks
 - records use of disk blocks
 - zones v blocks

i-nodes

- 64 bytes
- very similar to UNIX i-nodes
- 32-bit zone pointers

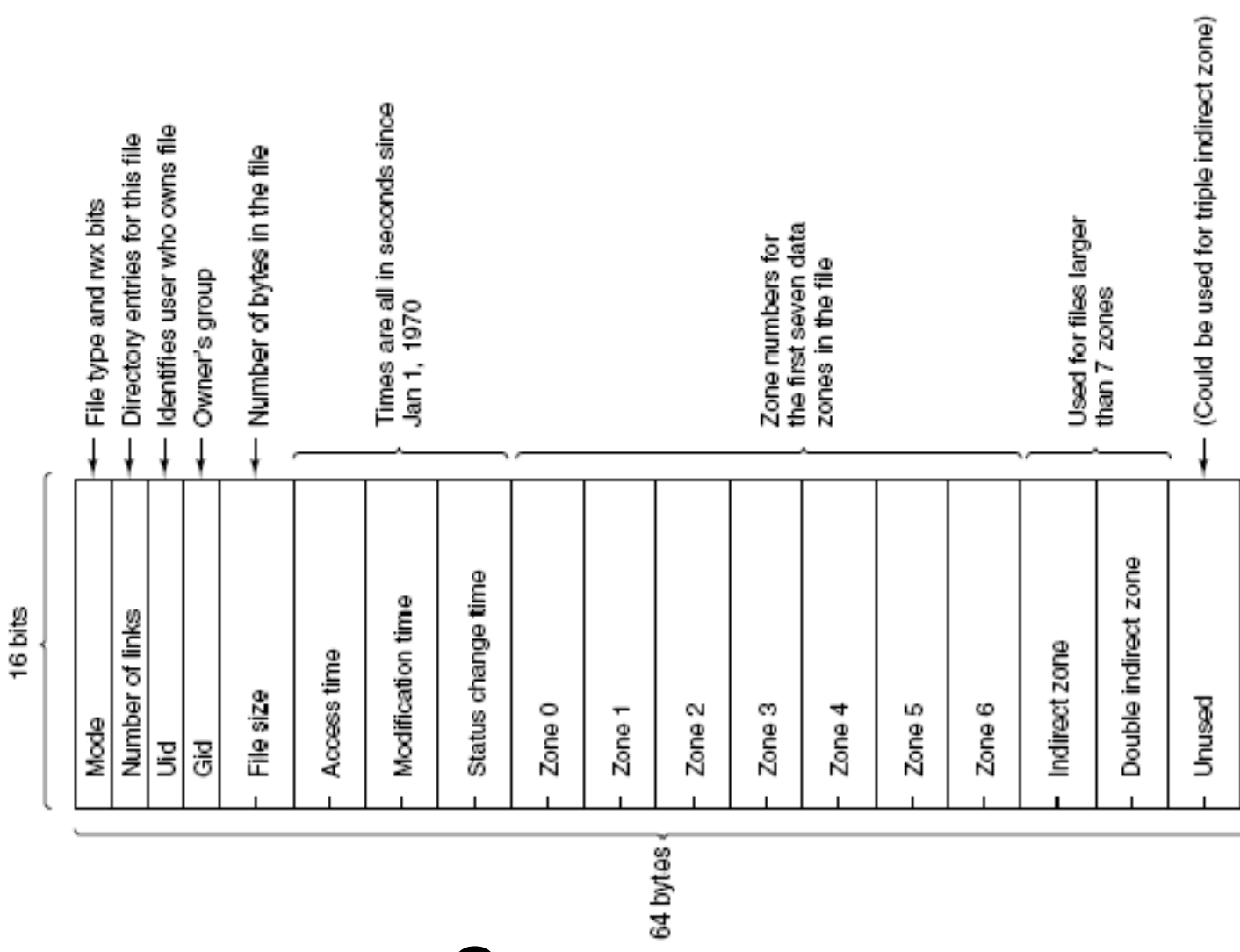


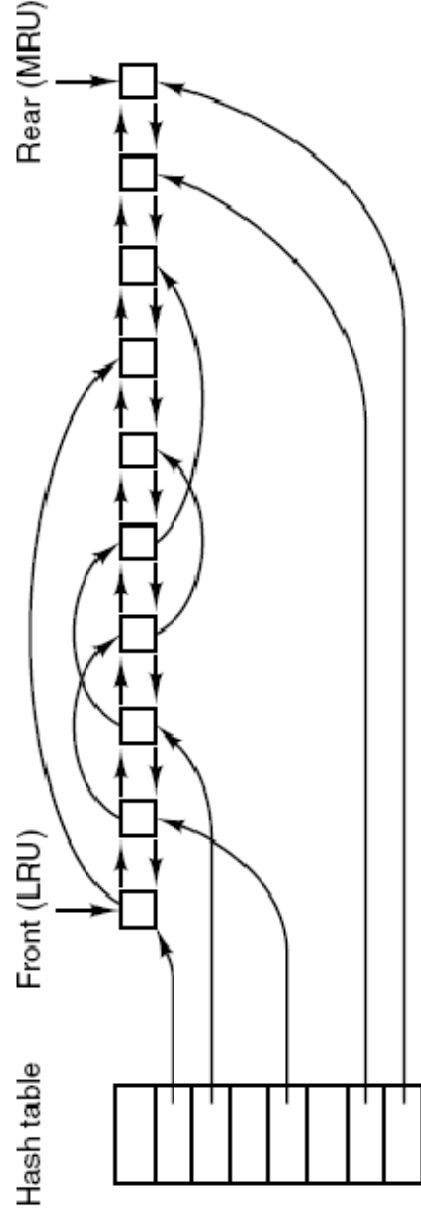
Figure 5-36. The MINIX i-node.

More details

- When file is opened:
 - locate i-node (using root directory if necessary)
 - load i-node into table in memory
 - increment counter
- File size:
 - With 1KB blocks, 7KB files stored using direct pointers
 - single indirect pointers allow 256 more blocks
 - double indirect points to 256 single indirect...
 - But...

Block cache

- Cache the blocks in use (within main memory)
- Blocks are chained together in a linked list, MRU to LRU
- Hash table used for more efficient lookup



Finding a block

- compute hash code for that block
- search the appropriate linked list
- If buffer containing block is found...
 - indicate that it's in use
- If not found...
 - use LRU buffer
 - check if block is modified – write back to disk if it is
- Releasing a block
 - decrement use counter

Block cache

- Depending on class of block:
 - front or rear of LRU list
 - write to disk immediately
- sync call traverses list and writes back modified blocks

Directories

- How do we find the correct i-node, given a file name?
- Each directory entry contains 64 bytes:
 - 4 bytes for i-node number
 - 60 bytes for file name
 - So what are the limits on number of files? What about file names?
- Is space being wasted?

Mounting

- How do we have different devices incorporated into a single directory tree?
- i-node of mounted on directory indicates the file system is not of the same device
- The superblock of the mounted filesystem is loaded – indicates it is mounted
- Finding `/usr/ast/f2`:
 - i-node for `/usr` indicates this is mounted – need to find root of device
 - search for superblock of device which is mounted on `/usr`
 - search from there...

Directories and Paths

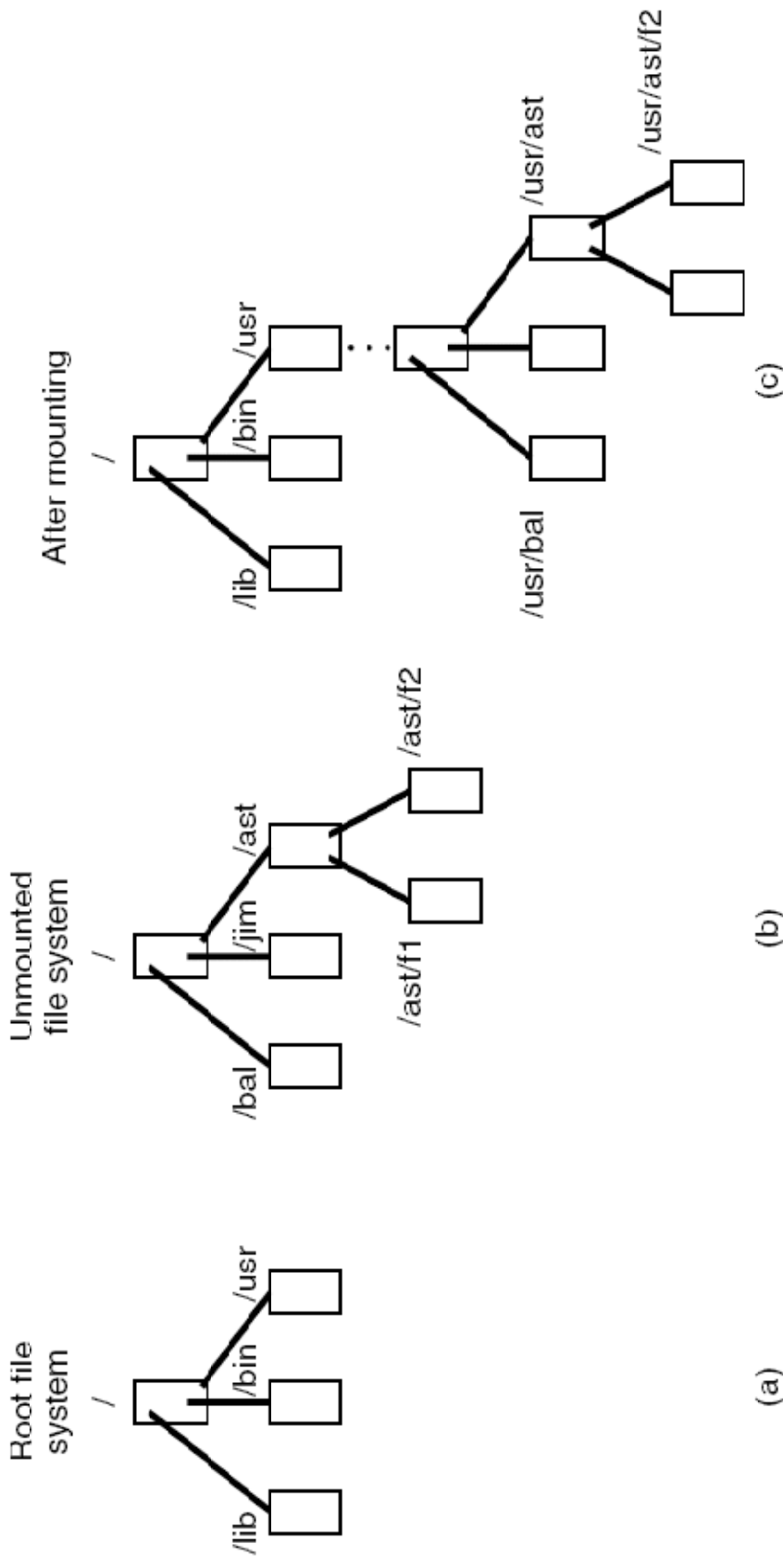


Figure 5-38. (a) Root file system. (b) An unmounted file system. (c) The result of mounting the file system of (b) on `/usr/`.

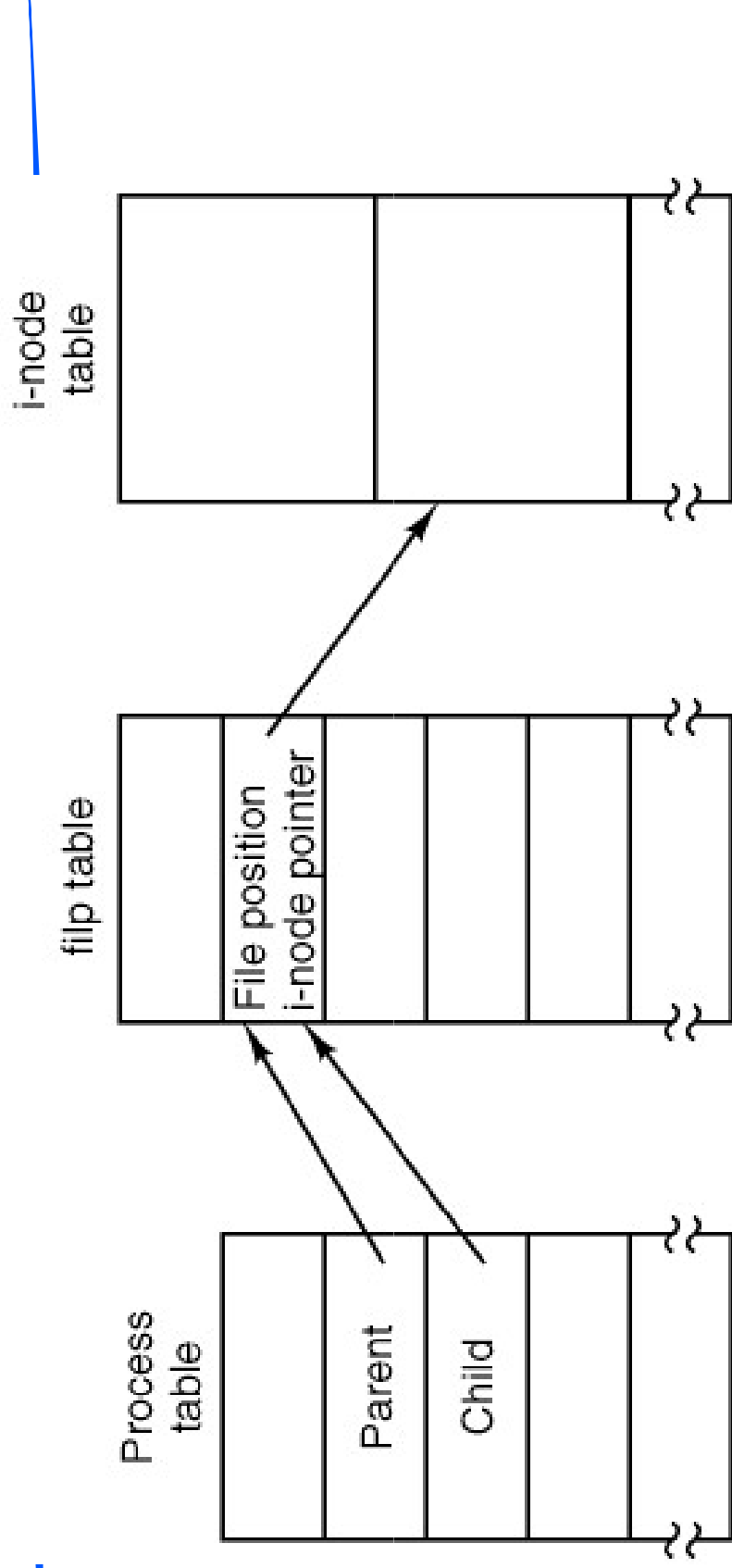
File descriptors

- When file is opened, descriptor is returned
- This is what is used in subsequent references to the file
- Process table within fs:
 - pointers to i-nodes of root
 - working directory
 - array indexed by file descriptor number...

File descriptor

- Do we just have an array of descriptors for each open file?
- We need some way of recording the position within the file as well
- Position can't be stored within the i-node - two processes can't have different positions in the same file
- Can't be stored in the process table either - parent and child will have different positions, when they should have the same
- shared *filp* table stored positions

File Descriptors



- Figure 5-39. How file positions are shared between a parent and a child.

File locking

- Minix 3 allows locks on files
- Not enforced by operating system
- Similar to file positions...
- *file_lock* table records locks for files...

Pipes

- Pipes slightly different from regular files:
 - Input could be delayed indefinitely
 - Can't just wait until request is completed
 - Need to suspend read request
- Check if request can be completed
 - if not, store details in process table
 - when pipe is modified, process can be woken up and complete request

Special files

- Devices are mounted into the file system...
- How do special files get treated?
- i-nodes for these files stores device class and specifics (major and minor numbers)
- major number indexes into table to determine appropriate device driver
- minor number passed to the driver

Example... READ

- `n=read(fd, buffer, bytes)`
 - library procedure called
 - encodes as a message, sent to fs, blocks
 - fs does read function
 - uses *fd* to find the entry in *filp* table
 - request broken into block-sized pieces
 - check if blocks are in cache
 - if not, select victim, ask driver to write back if necessary and retrieve requested block
 - Once in cache, request system task to copy data to user's buffer
 - reply to user, library procedure returns
 - Additional blocks speculatively read

Implementation details

- Large server...
- Reasonably simple, compared to other systems
- Details in the prac this week.
- And you get to explore it for Assignment 3!

Other Servers

- MINIX = kernel + user-space servers
- Have seen device drivers, pm, fs
- Re-incarnation server
- Information-server – gives debug info from function keys
- INET – network server, comparable in size to rest of MINIX
- Log driver – receives error messages and writes them to a file.

Other file systems- FAT

- FAT – used in older windows machines, poor performance over 200MB volume size for heavily loaded disks.
- Might still be used for floppy disks
- FAT32 commonly used as the format for USB drives

NTFS

- NTFS – modern windows system – quite a complex structure, eg. Wikipedia lists the following key features:-
 - 3.1 Alternate data streams (ADS)
 - 3.2 Quotas
 - 3.3 Sparse files
 - 3.4 Reparse points
 - 3.5 Volume mount points
 - 3.6 Directory junctions
 - 3.7 Hard links
 - 3.8 Hierarchical Storage Management (HSM)

NTFS (cont)

- 3.9 Native Structured Storage (NSS)
- 3.10 Volume Shadow Copy
- 3.11 File compression
- 3.12 Single Instance Storage (SIS)
- 3.13 Encrypting File System (EFS)
- 3.14 Symbolic links
- 3.15 Transactional NTFS
- 3.16 USN Journal

Summary

- File system is large
- Large impacts on system performance if it's slow
- Lots of code for you to read!

Tutorial 8 (out week 10, discussed week 12)

Q1: Assuming 4kB disk blocks, 4kB zones, 32-bit zone numbers, what is the maximum MINIX file size that can be handled:

- (i) Using direct zone numbers
- (ii) + using indirect zone numbers
- (iii) + using doubly indirect zones numbers
- (iv) + using triply indirect zone numbers.

Tutorial

Q2: Assuming 1kB disk blocks, 1kB zones, 64 byte i-nodes, how many bits are needed in zone numbers and file-size-word to fully use the capacity of triple-indirect zone numbers.

Q3 Repeat Q2 assuming 4kB disk blocks and 4kB zones, 64byte i-nodes.