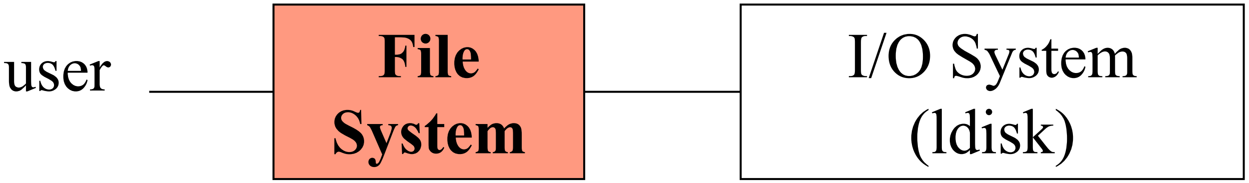
Project: File System

Textbook: pages 501-506

Lubomir Bic

Assignment

- Design and implement a simple file system using ldisk to emulate disk
- Overall organization



user

File System

I/O System (ldisk)

- Input

```
cr foo
op foo
wr 1 y 10
sk 1 0
rd 1 3
```

- Output

```
file foo created
file foo opened, index=1
10 bytes written
current position is 0
3 bytes read: yyy
```

I/O System

- I/O system presents disk as a linear sequence of blocks:
  - We will refer to the logical disk as *ldisk[L][B]*
    - L is the number of logical blocks on ldisk
    - B is the block length (in bytes)
  - It can be implemented as a byte array or integer array
    - Type casting or conversion is necessary
- I/O system interface – provided by your driver:

```
read_block(int i, char *p)
write_block(int i, char *p)
```

- Each command reads or writes an entire block (B bytes)
- FS can access the emulated disk using only these functions (no direct access to ldisk is allowed)

File System -- User Interface

- create(symbolic_file_name): return ok/error
- destroy(symbolic_file_name) : return ok/error
- open(symbolic_file_name): return index/error
- close(index): return ok/error
- read(index, mem_area, count): return #bytes read/error
- write(index, mem_area, count): return #bytes written/error
- lseek(index, pos) : return ok/error
- directory: return list of files/error

- init/save: create or restore ldisk/save ldisk

Review of concepts

- directory structure
  - tree, DAG, graph, symbolic links, path names
  - this project: single flat list of all files (=one special file)
- organization of entries within directory
  - array of slots, linked list, hash table, B-tree
  - this project: unsorted array of fixed-size slots
- each directory entry contains
  - all descriptive info, parts of info, name only
  - this project: symbolic name plus index of descriptor

Review of concepts

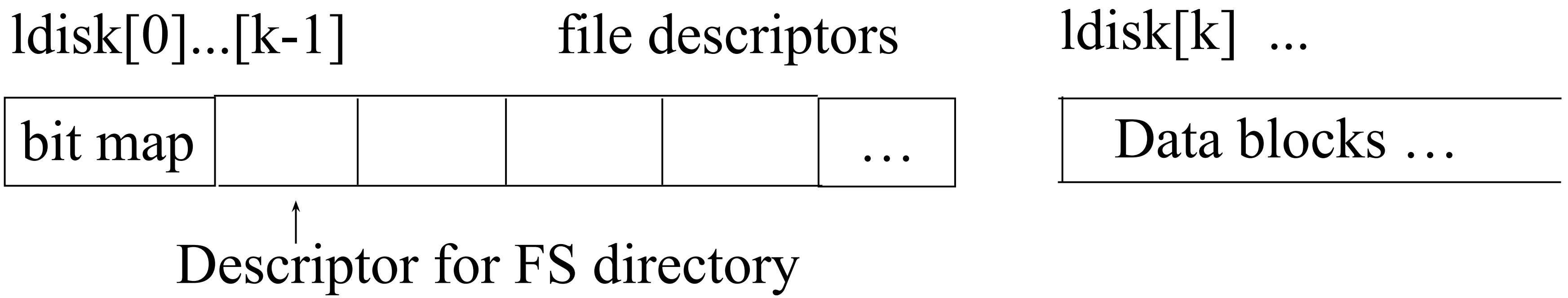
- file descriptor contents
  - owner, file type, protection, length, disk map, access times
  - this project: length (bytes), disk map
- disk map (physical organization)
  - contiguous, linked list, indexed, multi-level
  - this project:
    - flat index (fixed list of max 3 disk blocks)
    - 1-level incremental (for teams)

Review of concepts

- location of file descriptors
  - dedicated portion of disk, special files, in directories
  - this project: first k disk blocks (ldisk[0]..[k-1])

- free storage management
  - linked lists, bit map
  - this project: bit map

Organization of the file system



ldisk[0]...[k-1]

file descriptors

ldisk[k] ...

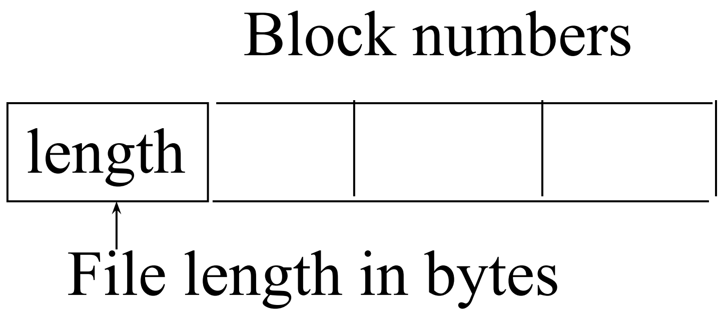
bit map

…

Data blocks …

Descriptor for FS directory

Each descriptor



Block numbers

length

File length in bytes

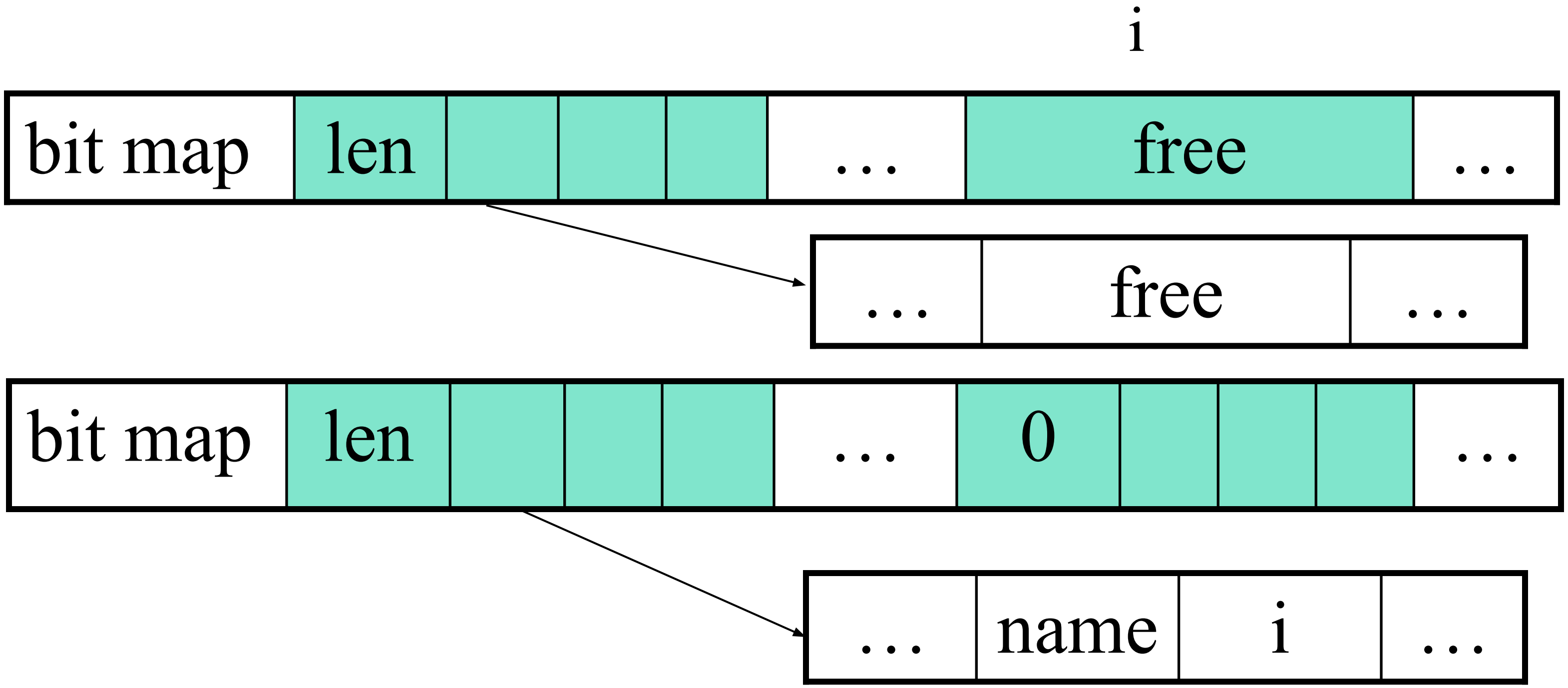
- teams: additional task: 1-level incremental index
  - last entry points to another file descriptor (2+4 block)

The Directory

- only one directory (root)
- regular file, i.e., use regular file operations:
  - read, write, lseek
  - but the directory is always open (OFT[0])
- described by the first descriptor
- contains array of entries:
  - symbolic file name (characters)
  - index of the descriptor (integer)

Create a file

- find a free file descriptor
- find a free directory entry
- fill both entries

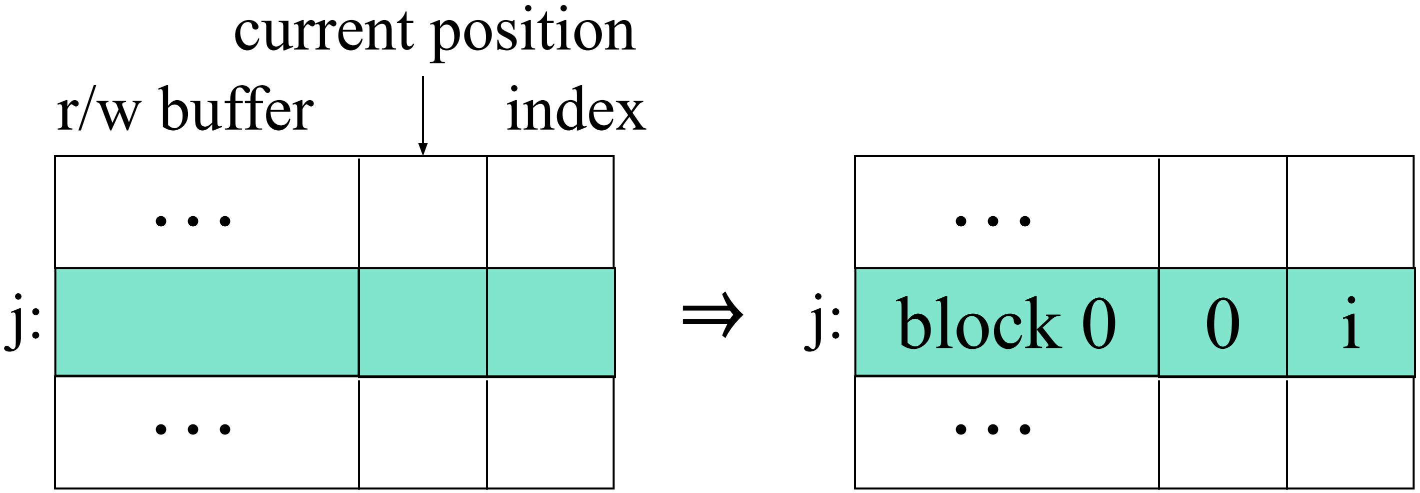


Destroy a file

- search directory to find file descriptor
- remove directory entry
- update bit map (if file was not empty)
- free file descriptor
- return status

Open a file

OFT:



current position

r/w buffer

index

. . .

j:

. . .

⇒

. . .

j: block 0 | 0 | i

. . .

- search directory to find index of file descriptor (i)
- allocate a free OFT entry (reuse deleted entries)
- fill in current position (0) and file descriptor index (i)
- read block 0 of file into the r/w buffer (read-ahead)
- return OFT index (j) (or return error)
- consider adding a file length field (to simplify checking)

Close a file

- write buffer to disk
- update file length in descriptor
- free OFT entry
- return status

Read a file

- compute position in the r/w buffer
- copy from buffer to memory until
  - desired count or end of file is reached:

    update current pos, return status
  - end of buffer is reached
    - write the buffer to disk
    - read the next block
    - continue copying

Write a file

- write into buffer
- when full, write buffer to disk
  - if block does not exist (file is expanding):
    - allocate new block
    - update file descriptor
    - update bit map
- update file length in descriptor

Seek in a file

- if the new position is not within the current block
  - write the buffer to disk
  - read the new block
- set the current position to the new position
- return status

List the directory

- read directory file
- for each entry:
  - find file descriptor
  - print file name and file length

Presentation shell

- develop presentation shell:
  - repeatedly accept command (e.g. `cr abc`)
  - invoke corresponding FS function (e.g. `create(abc)`)
  - display status/data on screen

    (e.g. `file abc created` or `error`)
- project will be tested using an input file and it must produce an output file

Shell commands and Output

- `cr <name>`
  - **Output:** `file <name> created`
- `de <name>`
  - **Output:** `file <name> destroyed`
- `op <name>`
  - **Output:** `file <name> opened, index=<index>`
- `cl <index>`
  - **Output:** `file <name> closed`
- `rd <index> <count>`
  - **Output:** `<count> bytes read: <xx...x>`
- `wr <index> <char> <count>`
  - **Output:** `<count> bytes written`

Shell commands and Output

- `sk <index> <pos>`
  - Output: `current position is <pos>`
- `dr`
  - Output: `file0 <len0>,..., fileN <lenN>`
- `in <no_cyl> <no_surf> <no_sect> <sect_len> <disk_cont>`
  - `disk_cont` is a text file; it holds copy of ldisk
  - If file does not exist, output: `disk initialized`
  - If file does exist, output: `disk restored`
- `sv <disk_cont>`
  - Output: `disk saved`
- If any command fails, output: `error`

Sample Interaction

- Input

```
in 4 2 8 64 dsk.txt
cr foo
op foo
wr 1 x 60
wr 1 y 10
sk 1 55
rd 1 10
dr
sv dsk.txt
in 4 2 8 64 dsk.txt
op foo
rd 1 3
cr foo
```

- Output

```
disk initialized
file foo created
file foo opened, index=1
60 bytes written
10 bytes written
current position is 55
10 bytes read: xxxxxyyyyy
foo 70
disk saved
disk restored
file foo opened, index=1
3 bytes read: xxx
error
```

Handling the Bit Map (pg 217)

- determine BM size (# of bits needed = # of ldisk blocks)
- represent bit map as an array of int (32 bits each): `BM[n]`
- **How to set, reset, and search for bits in BM?**
- prepare a mask array: MASK[16]
  - diagonal contains “1”, all other fields are “0”
  - use bit operations (bitwise or/and) to manipulate bits

Handling the Bit Map

- MASK

0	10…
1	010…
2	0010…
3	00010…
…	…
16	0 … 01

- to set bit i of BM[j] to “1”:

$$BM[j] = BM[j] \mid MASK[i]$$

Handling the Bit Map

- how to create MASK?

  MASK[0] = 0x8000 (1000 0000 0000 0000)

  MASK[1] = 0x4000 (0100 0000 0000 0000)

  MASK[2] = 0x2000 (0010 0000 0000 0000)

  MASK[3] = 0x1000 (0001 0000 0000 0000)

  MASK[4] = 0x0800 (0000 1000 0000 0000)

  …

  MASK[15] = 0x0001 (0000 0000 0000 0001)

- another approach:

  MASK[15] = 1;

  MASK[i] = MASK[i+1] <<

Handling the Bit Map

- to set a bit to “0”:
  - create MASK2, where MASK2[i] = ~MASK[i]

    e.g., 0010 0000 0000 0000 → 1101 1111 1111 1111

    (use “~” operator or declare using hex constants like MASK)
- set bit i of BM[j] to “0”:

  BM[j] = BM[j] & MASK2[i]

Handling the Bit Map

- to search for a bit equal to “0” in BM:

```
for (i=0; …          /* search BM from the beginning
     for (j=0; …     /* check each bit in BM[i] for “0”
          test = BM[i] & MASK[j])
          if (test == 0) then bit j of BM[i] is “0”; stop search
```

Assumptions for Testing

- Disk: 4 cylinders, 2 surfaces, 8 sectors/track, 64 Bytes/sector
  - sector = block = 64 B =16 int
- What is k?
  - Bitmap: 4*2*8=64 bits (2 integers)
  - Descriptor: 4 integers (file length plus 3 block #s)
- How many descriptors (files) can we have?
  - How many fit into directory?
  - Each dir entry: 2 int
    - File name: maximum 4 chars, no extension (=1 int)
    - Descriptor index: 1 integer
  - dir = 3 blocks = 3*64 B = 48 int = 24 entries (= 24 files)
  - 24 descriptors = 96 int = 6 blocks
- k=7: 1 block for bitmap, 6 blocks for descriptors

Assumptions for Testing

- Number of open files: 3 plus the directory, i.e., the size of the OFT is exactly 4 entries.
- Directory should be opened automatically when program starts (index=0)
- It should close automatically with sv command
- All other files should also close at that time

Summary of tasks

- design and implement I/O system (ldisk plus read/write ops)
- design and implement FS using ldisk
- develop test/presentation shell
- teams:
  - error checks on all commands
  - additional task (pg 506): 1-level expanding index
- submit documentation
- schedule testing