COP4610
Introduction to Operating Systems
Project #3: Implementing a FAT32 File System

Outline

- Clarification
- Walkthrough
- Program commands
  - `open FILENAME`
  - `close FILENAME`
  - `read FILENAME SIZE`
  - `write FILENAME SIZE “STRING”`

Clarification

- Do I have to program on a virtual machine?
  - **NO**
  - The project is a shell-type single executable
- Does the program have to run on linprog?
  - **YES**
  - Make sure your final version works correctly on linprog
  - Don’t wait for the last time to check this!

Walkthrough

- BPB
- FAT
- Data

Program Commands

`open FILENAME MODE`

- Opens the file for further operations
- Does not write anything to the image
- Must keep a structure that represents the open file
  - First cluster
  - Mode (read, write, read-write, write-read)
  - Offset (in bytes)
- Store structures in a list (multiple open files)

Program Commands

`open FILENAME MODE`

Steps:

- Check if the file exists
  - Get the DIRENTRY associated to FILENAME
    - Compare `FILENAME` with `DIR_Name`
  - Return an error if no DIRENTRY
- Make sure it is not a directory
  - Apply `ATTR_DIRECTORY` mask to `DIR_Attr`
    - Should return 0. Return error otherwise

Program Commands

`open FILENAME MODE`

Steps (2):

- Check file permissions
  - Read-only: Apply `ATTR_READ_ONLY` to `DIR_Attr`
    - If != 0 and open mode is write, read-write, or write-read, return an error.
- Check whether the file is already open
  - Use first cluster number (easier)
  - Search in open file list for same cluster number
    - If found, the file is already open. Return an error.

Program Commands

`open FILENAME MODE`

Steps (3):

- If no error, create a structure and store the file information
  - First cluster
  - Open mode
  - Offset in bytes (Initialized at 0)
- Add the structure created to the open file list

Program Commands

`close FILENAME`

- Find FILENAME's DIRENTRY and compare its first cluster number with the items in the list
  - If found, remove it from the list.
  - If not found, return an error.

Program Commands

`lseek FILENAME OFFSET`

- Sets the offset of a file (in bytes) for further reading/writing
- Find FILENAME's DIRENTRY and compare its first cluster number with the items in the list
  - If found, update the entry's offset field to OFFSET
  - If not found, return an error
  - If OFFSET > file size, return an error.

Program Commands

`read FILENAME SIZE`

- Reads data from FILENAME and prints its content
  - Start at the stored offset for the file and read SIZE bytes
- Print error if:
  - FILENAME does not exist
  - FILENAME is a directory
  - The file is not open or open for reading

Program Commands

`read FILENAME SIZE`

Steps:

- Get the first cluster of the file
  - Either from the open file list by checking path
    - Or get the DIRENTRY for FILENAME and compare with clusters in list
- Move the pointer (in image) to the stored offset
  - hint: use `lseek()` to move within the image file
  - offset > bytes per cluster?
    - Move to next cluster (using FAT)
    - Set temp offset to (offset - bytes_per_cluster) **Don't store it!**
    - Still higher? Repeat

Program Commands

`read FILENAME SIZE`

Steps (2):

- If offset + SIZE > bytes_per_cluster
  - Read only (bytes_per_cluster – offset) bytes from current cluster
- Read to buffer
- Print buffer
  - `printf()`
  - `fwrite()`

Program Commands

`read FILENAME SIZE`

Steps (3):

- Move to next cluster
  - SIZE := SIZE - (bytes_per_cluster – OFFSET)
  - offset := 0
  - Read
  - Repeat until size is 0
- If offset + SIZE is larger than the file size
  - Read file_size - offset bytes, starting at offset

Program Commands

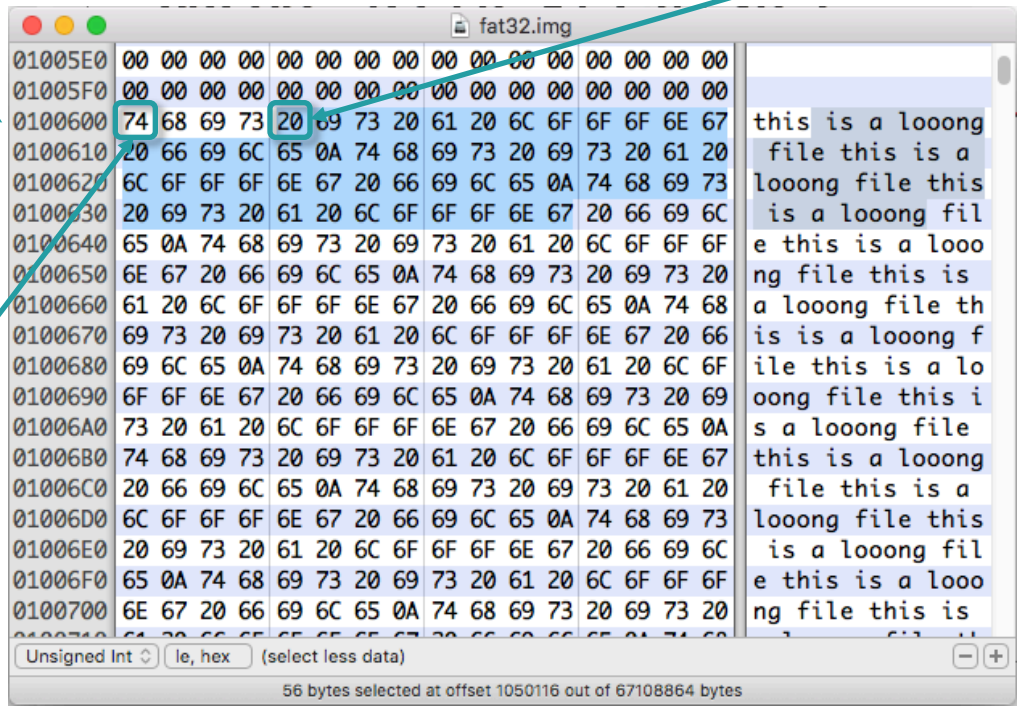
```
lseek LONGFILE 4
read LONGFILE 56
```

Offset (start reading from here)

Cluster byte offset
(data region)

```
ByteOffset =
[FirstDataSector +
(N – 2) * BPB_SecPerClus]
* BPB_BytsPerSec
```

First byte
of file
(and cluster)



This will read 56 bytes from the offset position (shaded bytes). Output:

```
" is a looong file this is a looong file this is a looong"
```

Program Commands

`write FILENAME SIZE “STRING”`

- Write data to FILENAME
  - Start at the file’s stored offset and write SIZE bytes of STRING
  - STRING is between quotes “”
- Print error if:
  - FILENAME does not exist
  - FILENAME is a directory
  - The file is not open or open for writing
  - OFFSET is larger than the size of the file

Program Commands

`write FILENAME SIZE “STRING”`

Steps:

- Get the first cluster of the file
  - Get the DIRENTRY for FILENAME and compare with clusters in open file list
- Determine if the file size is to be increased
  - Is offset + SIZE larger than the file size?
    - Allocate extra clusters accordingly

Program Commands

`write FILENAME SIZE “STRING”`

Steps (2):

- Allocate extra clusters
  - Determine the number of current clusters
    - current_clusters = file_size / bytes_per_cluster
    - if file_size % bytes_per_cluster > 0, +1 cluster
  - Determine the final number of clusters
    - final_clusters = (offset + SIZE) / bytes_per_cluster
    - if (OFFSET + SIZE) % bytes_per_cluster > 0, +1 cluster
  - Extra clusters = final_clusters - current_clusters

Program Commands

`write FILENAME SIZE “STRING”`

Steps (2):

- Write data in clusters
  - Start at the cluster corresponding to offset
  - Repeat:
    - Write data until end of cluster reached (or all bytes are written)
    - Move to next cluster (if any. Use same logic as for `read`)
      - If no cluster, allocate a new one, move to it

Program Commands

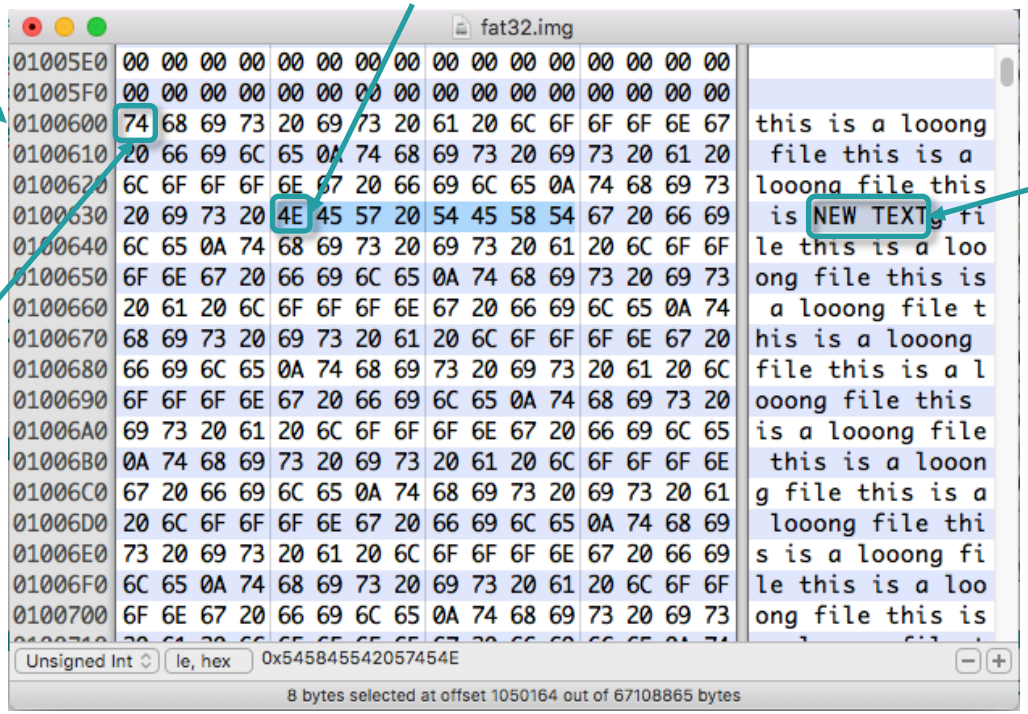
```
lseek LONGFILE 52
write LONGFILE 8 "NEW TEXT"
```

Cluster byte offset
(data region)

```
ByteOffset =
[FirstDataSector +
(N – 2) * BPB_SecPerClus]
* BPB_BytsPerSec
```

Offset (start writing from here)

First byte
of file
(and cluster)



Text written

Questions?