CSC/CEG 3150 Tutorial

Last Milestone in Assignment 3 & tips

XIAO Zigang
zgxiao@cse.cuhk.edu.hk

Department of Computer Science and Engineering
The Chinese University of Hong Kong

November 18, 2008

Outline

1. Outline
2. Milestone 3
3. Debug
4. Tips
5. Q & A
6. Acknowledgement

Tasks in Milestone 3

- **Objective**: Writing to an **existing** file
  - Truncation: **Discard** original data and write new content
  - Appendage: **Preserve** original data and write from the end of the target

Truncation Mode work flow

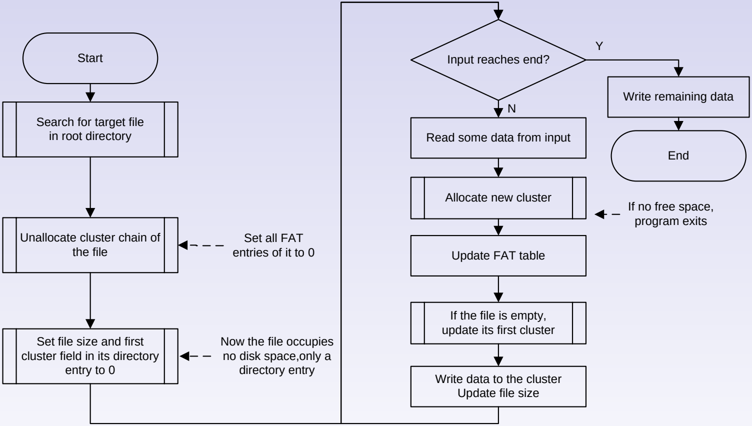
Start
Search for target file in root directory
Unallocate cluster chain of the file
Set all FAT entries of it to 0
Set file size and first cluster field in its directory entry to 0
Now the file occupies no disk space,only a directory entry
Input reaches end?
Y
N
Write remaining data
End
Read some data from input
Allocate new cluster
If no free space, program exits
Update FAT table
If the file is empty, update its first cluster
Write data to the cluster Update file size

Figure: Illustration of truncation

FSINFO

- FSINFO data structure includes hints about where the OS can allocate new clusters.
- Its location is given in the boot sector

```c
/* FSINFO,size=512B */
#pragma pack(push,1)
typedef struct FSInfo {
        unsigned long FSI_LeadSig;     /* Signature (0x41615252) */
        unsigned char FSI_Reserved1[480]; /* Not used */
        unsigned long FSI_StrucSig;       /* Signature (0x61417272) */
        unsigned long FSI_Free_Count;     /* Number of free clusters */
        unsigned long FSI_Nxt_Free;       /* Next free cluster */
        unsigned char FSI_Reserved2[12];  /* Not used */
        unsigned long FSI_TrailSig;       /* Signature (0xAA550000) */
}FSInfo;
#pragma pack(pop)
```

Listing 1: FSINFO structure

Cluster allocation

- Different allocation scheme in implementations
- You should implement a **circular, next-available** scheme
- The next free cluster information is stored in FSINFO structure
  - i.e. *FSI_Nxt_Free*
  - Note: it stores the cluster value **just assigned**
- Remember to update FSINFO after a cluster is allcoated!

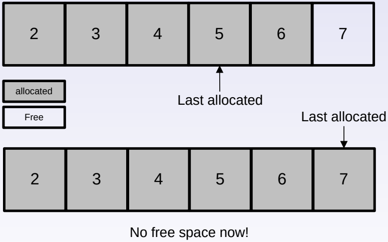

Figure: Illustration of truncation

Appendage Mode work flow

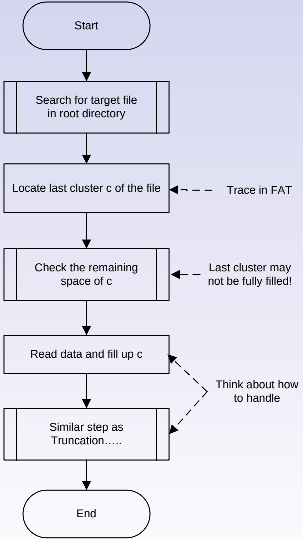
Start
Search for target file in root directory
Locate last cluster c of the file
Trace in FAT
Check the remaining space of c
Last cluster may not be fully filled!
Read data and fill up c
Think about how to handle
Similar step as Truncation…..
End

Figure: Illustration of appendage

How to debug your program?

- Here is one scenario you may follow:

```
# create a disk image file, size = 64M
$ dd if=/dev/zero of=./fs bs=1M count=64
# format as FAT32 filesystem(you may also use /dev/ram[x] here)
$ /sbin/mkfs.vfat -v -F 32 -f 2 -S 512 -s 1 -R 32 ./fs
# you may backup this file here for later use

# mount to a certain mount point
$ mkdir /mnt/rd
$ mount -t vfat -o loop,umask=000 ./fs /mnt/rd
# You can write a script/make rule to do stuff listed above

# make some change here...
$ echo "void main(){}" > /mnt/rd/TEST.C
$ touch /mnt/rd/MAKEFILE
# synchronize change
$ sync

# check your program
$ make
$ ./a3 -m 1 -d ./fs
$ sync
# ... (check your output)
```

utilize `make`

- `make` is very convenient in this assignment, e.g.
  - include `DEBUG` switch in your program, i.e.

```
#ifdef DEBUG
    printf("program reaches here...\n");
    ...
#endif
```

  - when compile your program with
    `#define DEBUG`
    or
    `gcc <other-options> -DDEBUG`
    The codes between `#ifdef/#endif` will be included.
- You can add a **debug rule** in your makefile, then you may use:
  `$ make debug`
  to compile a debug version

Example of test batch

- You are strongly encouraged to write script/make rule to initialize your disk image
- Please be reminded that `mkfs.vfat` will NOT help you to clear original content in the disk image

```
$ cat ./initfs
#!/bin/bash
dd if=/dev/zero of=filesystem bs=1M count=64
/sbin/mkfs.vfat -v -F 32 -f 2 -S 512 -s 1 -R 32 ./filesystem
mount -t vfat ./filesystem ./rd -o loop,umask=000 -v

echo 123 > ./rd/A.TXT
# file ‘‘ascii’’ contains some ascii characters
cat ascii > ./rd/B.TXT
dd if=/dev/urandom of=./rd/C.TXT bs=64 count=1
# file ‘‘raw’’ contains some raw contents
cat raw > ./rd/D.TXT

sync
```

Listing 2: Example of script

Sample output

```
$ ./a3 -m 1 -d ./filesystem
FILENAME        FILE SIZE       STARTING CLUSTER #
--------------------------------------------------
A.TXT           4               3
B.TXT           800             4
C.TXT           100             6
D.TXT           800             7

Total number of entries = 4
$ ./a3 -m 2 -d ./filesystem -t C.TXT | xxd
0000000: cd5e 10e7 e7ec 1988 006b 6974 7e5b f3a5  .^.......kit~[..
0000010: 78cd 7ab3 daa9 2687 8814 269d b2c3 fd7f  x.z...&...&.....
...
$ cat rd/A.TXT
123
$ echo "hello :)" > input_file.txt
# use truncation mode here
$ umount filesystem
$ ./a3 -m 3 -d filesystem -t A.TXT -w t -i input_file.txt && sync
$ mount -t vfat ./filesystem ./rd -o loop,umask=000 -v
$ cat rd/A.TXT
hello :)
# use appendage mode here
$ umount filesystem
$ ./a3 -m 3 -d filesystem -t A.TXT -w a -i input_file.txt && sync
$ mount -t vfat ./filesystem ./rd -o loop,umask=000 -v
$ ./a3 -m 2 -d filesystem -t A.TXT
hello :)
hello :)
```

Listing 3: Sample output

Some facts

- Where is the root directory?
    - Check BootEntry structure
- How many Directory Entries in a cluster?
    - $= cluster\_size/DirEntry\_size$
- What is the usage of the reserved sectors, other than BootEntry,FSINFO?
    - Reserved for system use(e.g. backup, upgrade)
- What is a DirEntry?
    - It contains the name and metadata for a file or directory. Each file or directory is allocated one DirEntry, it is located in the clusters allocated to the file's parent directory

Some facts(cont.)

- How do we know if a DirEntry is available?
  - Allocation status of a directory entry is determined by using the first byte:
    - `0x00` or `0x0e` : unallocated
    - Others: allocated
- What is the size of a directory?
  - The size field in the DirEntry of a directory is not used and should always be 0.
  - The only way to determine the size of the directory is to follow the cluster chain in FAT
- What is the usage of entry 0 and 1 in FAT table?
  - Cluster starts from index 2
  - Generally, 0 records media type, 1 records dirty status of file system

Hints

1. You are not required to update *create time*, *access time*, *etc.* fields in Directory entry in this assignment.
2. After changes are made to the file system(e.g. create/remove a file), you may find that the file system read by your program is not updated.
3. Kernel keeps data in memory in order to avoid slow disk I/O
    - Data will be written back “gradually”
4. Use `sync` to flush the data
5. or use `sync()` in your program
6. If `sync` does not work, try `umount` and then `mount` again.

Q & A Session

Thank You

Now you should understand FAT32
- Have fun in assignment 3 -

Acknowledgement

- Some materials and pictures are from last year's tutorial notes made by *Mr.Cheong Chi Hong*
- The style of this slide is adapted from the template made by *HUANG Zheng-hua* in *Wuhan University*
- Google "vfs" to learn more about filesystem in Linux