KWANGWOON UNIVERSITY

2024년 2학기 **운영체제실습**

Fat-based VFS Implementation

System Software Laboratory
School of Computer and Information Engineering
Kwangwoon Univ.

Assignment 5-2

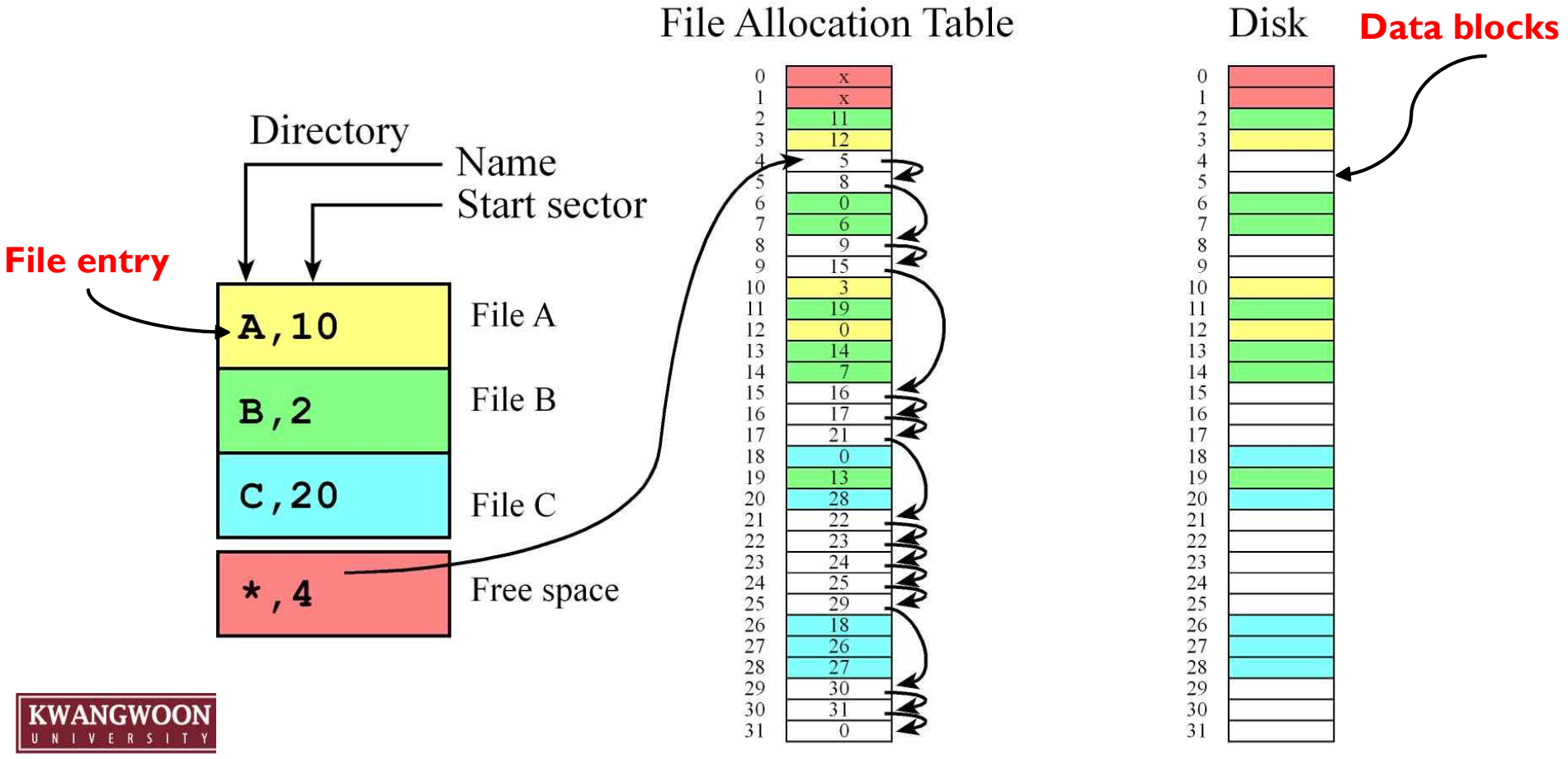
- **Project Title**
  - FAT (File Allocation Table) Based Virtual File System Implementation

- **Objective**
  - Implement a virtual file system using the FAT structure, which
    - performs file creation, writing/reading data, deleting files, and listing files.
    - manages state in memory and be able to save and restore the state from a file.

- **Key Learning Objectives**
  - Understand file system structures (FAT table, block allocation)
  - Manage files (creation, reading, writing, deleting)
  - Understand data persistence between memory and disk

File Allocation Table (FAT)

- **What is FAT?**
  - A simple file system structure that maps how files are stored on disk using a **table.**
  - Each file is divided into blocks, and these blocks are linked together in the FAT table.
  - Example: MS-DOS, Windows (FAT12, FAT16, FAT32)



File Allocation Table (FAT)

- **Key Components of FAT File System:**
  - FAT Table:
    - A mapping table that keeps track of which blocks of data belong to a file.
    - Each entry in the table corresponds to a block of data and points to the next block of the file. The last block of a file is marked with -1.
  - File Entries:
    - Contains metadata about each file, such as:
      - File name
      - File size
      - First block: The starting block in the FAT table where the file's data begins.
  - Data Blocks:
    - The actual storage location where file data is written.
    - Each block has a fixed size (e.g., 512 bytes).
    - If a file exceeds the block size, additional blocks are allocated, and these blocks are linked via the FAT table.

Assignment 5-2 Description

- **A FAT-based FS that provides the following functionalities:**
  - **1. File Creation (create <filename>):**
    - Create a new file and register it in the file system.
    - Allocate the first available block in the FAT table.
  - **2. Writing to a File (write <filename> <data>):**
    - Write data to the file. If the data exceeds the block size, allocate new blocks and link them in the FAT table.
  - **3. Reading from a File (read <filename>):**
    - Read the file's data and display it, reading through all linked blocks.
  - **4. Deleting a File (delete <filename>):**
    - Delete the file and release the blocks occupied by the file in the FAT table.
  - **5. Listing Files (list):**
    - Display the list of all files in the file system along with their sizes.
  - **6. Saving/Restoring File System State:**
    - Save the state of the file system (FAT table, file entries, data) to disk when the program terminates and restore it upon program startup.

Main Components of the File System

- **1. FAT Table**
  - Manages the blocks where file data is stored.
  - Each block points to the next block, with the last block marked as -1.

- **2. File Entries**
  - Stores metadata for each file
  - E.g. the file name, size, first block number, and usage status.

- **3. Data Blocks**
  - Holds the actual data of files.
  - Each block is of size BLOCK_SIZE.
  - If the data exceeds the block size, it spans across multiple blocks.

Functionality Overview & Examples

- **1. File Creation Example:**
  - **$ ./fat create file1**
  - After file creation, the first free block in the FAT table is allocated.
- **2. Writing to a File Example:**
  - **$ ./fat write file1 "Hello, World!"**
  - If the data exceeds the block size, new blocks are allocated and linked in the FAT table.
- **3. Reading from a File Example:**
  - **$ ./fat read file1**
  - All data stored in linked blocks will be read sequentially.
- **4. Listing Files Example:**
  - **$ ./fat list**
  - Display all files currently in the file system along with their sizes.

Requirements

- **1. Implement the FAT-based file system**
- **2. Manage files in memory and store the file system state on disk when the program terminates.**
- **3. Implement file creation, reading, writing, and deletion functions according to the provided function definitions.**
- **4. Ensure the file system state is saved to disk and restored upon program restart.**

File System Parameters

- **1. Maximum Files: 100 files**
  - Max number of files in the system
- **2. Num. of Data Blocks: 1024 blocks**
  - Total blocks for storing file data
- **3. Block Size: 32 bytes**
  - Size of each block of data
- **4. Maximum File Name: 100 characters**
  - Max length of file names

Code Structure (example)

- **create_file(const char *name):**
  - Creates a file and allocates a block.
- **write_file(const char *name, const char *content):**
  - Writes data to a file, allocating and linking blocks in the FAT table.
- **read_file(const char *name):**
  - Reads data from the file, following the linked blocks.
- **delete_file(const char *name):**
  - Deletes the file and releases its blocks in the FAT table.
- **list_files():**
  - Displays a list of all files in the system.
- **save_file_system() / load_file_system():**
  - Saves and restores the state of the file system.

Output

- **Sample output**

```
os2024123456@ubuntu:~/assgin5/5-2$ ./fat create A
Warning: No saved state found. Starting fresh.
File 'A' created.
os2024123456@ubuntu:~/assgin5/5-2$ ./fat create B
File 'B' created.
os2024123456@ubuntu:~/assgin5/5-2$ ./fat list
Files in the file system:
File: A, Size: 0 bytes
File: B, Size: 0 bytes
os2024123456@ubuntu:~/assgin5/5-2$ ./fat write A "Hello, world"
Data written to 'A'.
os2024123456@ubuntu:~/assgin5/5-2$ ./fat list
Files in the file system:
File: A, Size: 12 bytes
File: B, Size: 0 bytes
os2024123456@ubuntu:~/assgin5/5-2$ ./fat write B "Hello, world"
Data written to 'B'.
os2024123456@ubuntu:~/assgin5/5-2$ ./fat write B "Hola, world!"
Data written to 'B'.
os2024123456@ubuntu:~/assgin5/5-2$ ./fat list
Files in the file system:
File: A, Size: 12 bytes
File: B, Size: 24 bytes
os2024123456@ubuntu:~/assgin5/5-2$ ./fat read A
Content of 'A': Hello, world
os2024123456@ubuntu:~/assgin5/5-2$ ./fat read B
Content of 'B': Hello, worldHola, world!
os2024123456@ubuntu:~/assgin5/5-2$ ./fat  delete B
File 'B' deleted.
os2024123456@ubuntu:~/assgin5/5-2$ ./fat  list
Files in the file system:
File: A, Size: 12 bytes
```

KWANGWOON UNIVERSITY

Appendix

System Software Laboratory
School of Computer and Information Engineering
Kwangwoon Univ.

Simple Fat-based VFS Samples

- **Sample Code (simple_fat.c)**

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_FILE_NUM 100                            //Requirement - MAX_file_number     : 100
#define NUM_BLOCKS 1024                             //Requirement - MAX_num_block       : 1024
#define BLOCK_SIZE 32                               //Requirement - block_size(byte)    : 32
#define MAX_FILE_NAME 100                           //Requirement - MAX_file_name       : 100
#define FS_STAT "fs_state.dat"                   // DISK FILE

//File Entry
typedef struct{
        char filename[MAX_FILE_NAME];
        int start_block;
        int size;
} FileEnt;

//File system structure
typedef struct{
        int fat_table[NUM_BLOCKS];                  // FAT Table
        FileEnt directory[MAX_FILE_NUM];
        char data_area[NUM_BLOCKS * BLOCK_SIZE];    // 1024blocks * 32byte
} FileSystem;

FileSystem myfat;                               // File System Instance

/*=============================================================================
                            CONTROL API
=============================================================================*/
int create_file(const char *filename);                                  // Create & allocate block
int write_file(const char *filename, const char *data);                 // Write data to file & link blocks in the FAT table
int read_file(const char *filename);                                    // Read data from file, follow linked blocks
int delete_file(const char *name);                                          // Delete the file, release blocks int FAT table.
void list_files(void);                                                          // Display a list of all files in the system.

/* End of API */

void save_file_system(void);                                    // Save the File System to Disk
void load_file_system(void);                                    // Restore the File System from Disk
```

Simple Fat-based VFS Samples

- **save_file_system( ) / load_file_system( )**

```
42 // Save the File System to Disk
43 void save_file_system(void) {
44     FILE *f = fopen(FS_STAT, "wb");
45     if (f == NULL) {
46         printf("Error: Could not save file system state.\n");
47         return;
48     }
49     fwrite(&myfat, sizeof(FileSystem), 1, f);
50     fclose(f);
51 }
52
53 // Restore the File System from Disk
54 void load_file_system(void) {
55     FILE *f = fopen(FS_STAT, "rb");
56     if (f == NULL) {
57         printf("Warning: No saved state found. Starting fresh.\n");
58         return;
59     }
60     fread(&myfat, sizeof(FileSystem), 1, f);
61     fclose(f);
62 }
```

Simple Fat-based VFS Samples

- **create_file function**

```
64 // File Creation
65 int create_file(const char *filename) {
66     // check file name
67     for (int i = 0; i < MAX_FILE_NUM; i++) {
68         if (strcmp(myfat.directory[i].filename, filename) == 0) {
69             printf("File '%s' already exists.\n", filename);
70             return -1;
71         }
72     }
73 
74     // allocate new file
75     for (int i = 0; i < MAX_FILE_NUM; i++) {
76         if (myfat.directory[i].filename[0] == '\0') {
77             for (int j = 0; j < NUM_BLOCKS; j++) {
78                 if (myfat.fat_table[j] == 0) {
79                     myfat.fat_table[j] = 0xFFFF;
80 
81                     strcpy(myfat.directory[i].filename, filename);
82                     myfat.directory[i].start_block = j;
83                     myfat.directory[i].size = 0;
84                     printf("File '%s' created.\n", filename);
85                     return 0;
86                 }
87             }
88         }
89     }
90     printf("File system full. Cannot create more files.\n");
91     return -1;
92 }
```

Simple Fat-based VFS Samples

- **Write_file Function**

```
94 // File Write
95 int write_file(const char *filename, const char *data) {
96     for (int i = 0; i < MAX_FILE_NUM; i++) {
97         if (strcmp(myfat.directory[i].filename, filename) == 0) {
98             int start_block = myfat.directory[i].start_block;
99             int block = start_block;
100            int remaining_data = strlen(data);
101            int data_offset = 0;
102
103            if (start_block == -1) {
104                // File Not Initialized
105                printf("File '%s' not initialized. Use create command first.\n", filename);
106                return -1;
107            }
108            int block_end = BLOCK_SIZE;
109                while (block_end > 0 && myfat.data_area[block * BLOCK_SIZE + block_end - 1] == 0) {
110                    block_end--;
111                }
112             // Check the data size
113            if (remaining_data <= BLOCK_SIZE - block_end) {
114                strncpy(&myfat.data_area[block * BLOCK_SIZE + block_end], &data[data_offset], remaining_data);
115                myfat.directory[i].size += remaining_data;
116
117                printf("Data written to '%s'.\n", filename);
118                return 0;
119            }
120            else {
121                printf("Data is too Big!!!!\n");
122                return -1;
123            }
124
125        }
126    }
127    printf("File '%s' not found.\n", filename);
128    return -1;
129 }
```

Simple Fat-based VFS Samples

- **Read_file Function**

```
131 // Read Files
132 int read_file(const char *filename) {
133     for (int i = 0; i < MAX_FILE_NUM; i++) {
134         if (strcmp(myfat.directory[i].filename, filename) == 0) {
135             int start_block = myfat.directory[i].start_block;
136             int block = start_block;
137             int total_size = myfat.directory[i].size;
138             int bytes_read = 0;
139 
140             if (start_block == -1) {
141                 printf("File '%s' not initialized. Use create command first.\n", filename);
142                 return -1;
143             }
144 
145             printf("Content of '%s': ", filename);
146 
147             int remaining_bytes_in_block = total_size - bytes_read;
148             int read_size = (remaining_bytes_in_block < BLOCK_SIZE) ? remaining_bytes_in_block : BLOCK_SIZE;
149             printf("%.*s", read_size, &myfat.data_area[block * BLOCK_SIZE]);
150 
151             printf("\n");
152             return 0;
153         }
154     }
155 
156     printf("File '%s' not found.\n", filename);
157     return -1;
158 }
```

Simple Fat-based VFS Samples

- **Delete_file Function**

```
161 // File Delegation
162 int delete_file(const char *filename) {
163     for (int i = 0; i < MAX_FILE_NUM; i++) {
164         if (strcmp(myfat.directory[i].filename, filename) == 0) {
165             int start_block = myfat.directory[i].start_block;
166             int block = start_block;
167 
168             // Release Cluster at FAT Table
169             while (block != 0xFFFF) {
170                 int next_block = myfat.fat_table[block];
171                 myfat.fat_table[block] = 0;  // Release Cluster
172                 block = next_block;
173             }
174 
175             // Remove file at Directroy
176             myfat.directory[i].filename[0] = '\0';
177             printf("File '%s' deleted.\n", filename);
178 
179             return 0;
180         }
181     }
182     printf("File '%s' not found.\n", filename);
183     return -1;
184 }
```

Simple Fat-based VFS Samples

- **list_file / execute_command Function**

```
186 // Display a list of all files in the system.
187 void list_files() {
188     printf("Files in the file system:\n");
189     for (int i = 0; i < MAX_FILE_NUM; i++) {
190         if (myfat.directory[i].filename[0] != '\0') {
191             printf("File: %s, Size: %d bytes\n", myfat.directory[i].filename, myfat.directory[i].size);
192         }
193     }
194 }
195
196 void execute_cmd(char *cmd, char *filename, char *data, int num)
197 {
198         if (strcmp(cmd, "create") == 0) {
199                 if (num != 3)
200                         printf("Usage: create <filename>\n");
201                 else
202                         create_file(filename);
203     }
204         else if (strcmp(cmd, "write") == 0) {
205                 if (num != 4)
206                         printf("Usage: write <filename> <data>\n");
207                 else
208                         write_file(filename, data);
209     }
210         else if (strcmp(cmd, "read") == 0) {
211                 if (num != 3)
212                         printf("Usage: read <filename>\n");
213
214                     else
215                         read_file(filename);
216
217     }
218         else if (strcmp(cmd, "delete") == 0) {
219                 if (num != 3)
220                         printf("Usage: delete <filename>\n");
221                     else
222                         delete_file(filename);
223
224     }
225         else if (strcmp(cmd, "list") == 0) {
226                 list_files();
227     }
228         else {
229                 printf("Invalid command.\n");
230     }
231 }
```

Simple Fat-based VFS Samples

- **Main Function**

```
233 int main(int argc, char *argv[])
234 {
235         if(argc <= 1){
236                 printf("USAGE : ./fat <COMMAND> [ARGS]...\n");
237                 exit(1);
238         }
239
240         load_file_system();                                    //Restore File System from disk
241         execute_cmd(argv[1], argv[2], argv[3], argc);          //execute command
242         save_file_system();                                    //Save File system to Disk
243         exit(0);
244 }
```

Simple Fat-based VFS Samples

- **Output**

```
os2024123456@ubuntu:~$ ./sim_fat create A
Warning: No saved state found. Starting fresh.
File 'A' created.
os2024123456@ubuntu:~$ ./sim_fat create B
File 'B' created.
os2024123456@ubuntu:~$ ./sim_fat list
Files in the file system:
File: A, Size: 0 bytes
File: B, Size: 0 bytes
os2024123456@ubuntu:~$ ./sim_fat write A "Small_data1"
Data written to 'A'.
os2024123456@ubuntu:~$ ./sim_fat write A "Small_data2"
Data written to 'A'.
os2024123456@ubuntu:~$ ./sim_fat read A
Content of 'A': Small_data1Small_data2
os2024123456@ubuntu:~$
os2024123456@ubuntu:~$ ./sim_fat write B "BIIIIIIIIG_DATA_OVER_THE_BLOCK_SIZE"
Data is too Big!!!!
os2024123456@ubuntu:~$ ./sim_fat list
Files in the file system:
File: A, Size: 22 bytes
File: B, Size: 0 bytes
os2024123456@ubuntu:~$ ./sim_fat read B
Content of 'B':
os2024123456@ubuntu:~$
```