KWANGWOON UNIVERSITY

2024년 2학기 **운영체제실습**

Assignment 5

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

Contents

- **Assignment5-1**
  - Proc File System module
- **Assignment5-2**
  - FAT-based Virtual File System Implementation

Assignment 5-1

- **Process들의 정보를 출력하는 proc 파일 작성**
  - 모듈 적재 시, 다음의 proc 파일 생성: /proc/proc_**본인학번**/processInfo
  - 해당 proc 파일에 읽기 요청 시, 다음 정보 출력
    - **PID** : process ID
    - **PPID** : PID of parent process
    - **UID** : user ID
    - **GID** : group ID
    - **utime** : amount of time that this process has been scheduled in **user mode**
      : measured in clock ticks (i.e. task_struct->utime)
    - **stime** : amount of time that this process has been scheduled in **kernel mode**
      : measured in clock ticks (i.e. task_struct->stime)
    - **State** : process state
      - 다음 중 하나 선택
        - R (running), S (sleeping), D (disk sleep), T (stopped), t (tracing stop), X (dead), Z (zombie), P (parked), I (idle)

Assignment 5-1

- **Process들의 정보를 출력하는 proc 파일 작성**
  - 해당 proc 파일 읽기 요청 시, 다음의 조건으로 처리
    - (1) Proc 파일에 write 요청이 없었거나, **-1** 값을 쓴 경우
      - 모든 프로세스의 정보 출력 (pid가 1인 프로세스부터 순차적으로)
      - e.g. $ **cat /proc/proc_2024123456/processInfo**

```
Pid     PPid    Uid     Gid     utime   stime   State           Name
-------------------------------------------------------------------------------
1       0       0       0       99      808     S (sleeping)    systemd
2       0       0       0       0       7       S (sleeping)    kthreadd
3       2       0       0       0       0       I (idle)        rcu_gp
4       2       0       0       0       0       I (idle)        rcu_par_gp
6       2       0       0       0       0       I (idle)        kworker/0:0H-kblockd
8       2       0       0       0       0       I (idle)        mm_percpu_wq
9       2       0       0       0       7       S (sleeping)    ksoftirqd/0
10      2       0       0       1       6592    I (idle)        rcu_sched
11      2       0       0       0       0       I (idle)        rcu_bh
12      2       0       0       0       69      S (sleeping)    migration/0
14      2       0       0       0       0       S (sleeping)    cpuhp/0
15      2       0       0       0       0       S (sleeping)    cpuhp/1
16      2       0       0       0       67      S (sleeping)    migration/1
17      2       0       0       0       62      S (sleeping)    ksoftirqd/1
19      2       0       0       0       0       I (idle)        kworker/1:0H-kblockd
20      2       0       0       0       0       S (sleeping)    cpuhp/2
21      2       0       0       0       55      S (sleeping)    migration/2
22      2       0       0       0       25      S (sleeping)    ksoftirqd/2
24      2       0       0       0       0       I (idle)        kworker/2:0H-kblockd
25      2       0       0       0       0       S (sleeping)    cpuhp/3
26      2       0       0       0       67      S (sleeping)    migration/3
```

Assignment 5-1

- **Process들의 정보를 출력하는 proc 파일 작성**
  - 해당 proc 파일 읽기 요청 시, 다음의 조건으로 처리 (cont'd)
    - (2) Proc 파일에 특정 프로세스의 PID 값을 입력한 경우
      - 해당 프로세스의 정보만 출력
      - e.g.

        $ echo 1 > /proc/proc_2017123456/processInfo

        $ cat /proc/proc_2017123456/processInfo

```
Pid     PPid    Uid     Gid     utime   stime   State           Name
-------------------------------------------------------------------
1       0       0       0       99      808     S (sleeping)    systemd
```

      - 단, 이 경우 proc 파일에 하나의 프로세스 ID만 기록한다고 가정

  - [출력 형식]은 p.4 ~ p.5의 예제를 따를 것

PID	PPID	UID	GID	UTIME	STIME	STATE	NAME
[Process ID]	[PID of Parent]	[User ID]	[Group ID]	[User_mode_time]	[Kernel_mode_time]	[Process state]	[Process name]

  - Hint
    - /proc/[pid]/status
    - /proc/[pid]/stat

Assignment 5-2

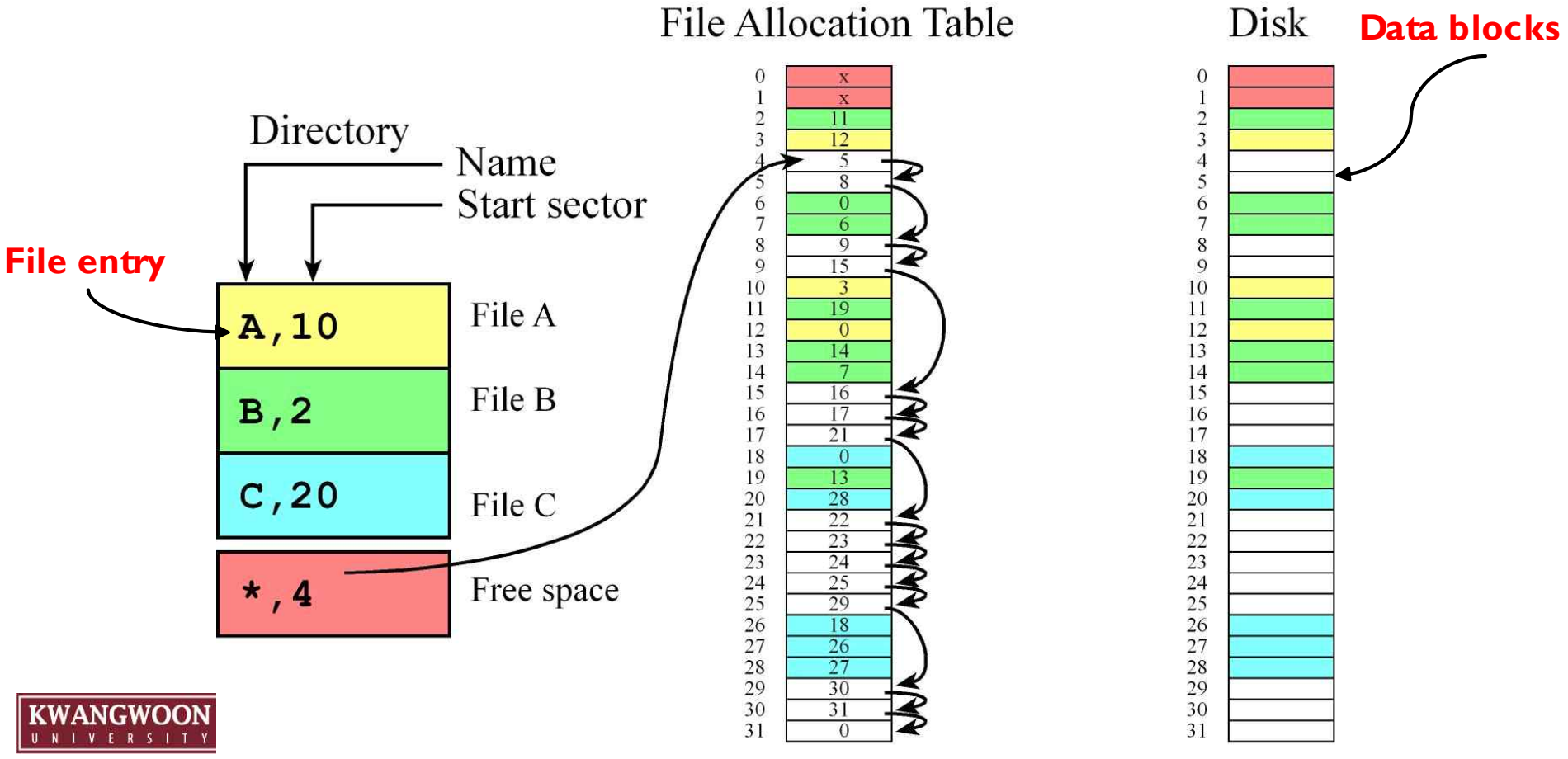
- **Project Title**
  - FAT (File Allocation Table) Based Virtual File System Implementation

- **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
```

Report Requirements

- **Ubuntu 20.04.6 Desktop 64bits 환경에서 채점**
- **Copy 발견 시 0점 처리**
- **보고서 구성**
  - **보고서 표지**
    - 수업 명, 과제 이름, 담당 교수님, 학번, 이름 필히 명시
      - 과제 이름 ➔ Assignment #5
  - **과제 내용**
    - Introduction
      - 과제 소개 - 4줄 이상(background 제외) 작성
    - Result
      - 수행한 내용을 캡처 및 설명
    - 고찰
      - 과제를 수행하면서 느낀 점 작성
    - Reference
      - 과제를 수행하면서 참고한 내용을 구체적으로 기록
      - 강의자료만 이용한 경우 생략 가능

Report Requirements

- **Source**
  - Assignment 5-1(2개)
    - proc_info.c
    - Makefile
  - Assigment 5-2(2개)
    - fat.c / Makefile
  - 각 코드 파일은 디렉토리를 나누어서 제출
    - E.g)
      - Assignment2-1/os_ftrace.c
      - Assignment2-2/os_ftracehooking.c
      - Assignment2-3/ ftracehooking.c
        / ftarcehooking.h
        / iotracehooking.c
        / Makefile
- **Copy 발견 시 0점 처리**

Report Requirements

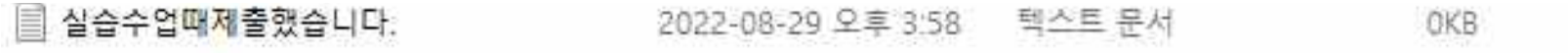
- **Softcopy Upload**
  - 제출 파일
    - 보고서 + 소스파일 **[하나의 압축 파일로 압축하여 제출(tar.xz)]**
    - 보고서(.pdf. 파일 변환)
    - 소스코드(**Comment 반드시 포함**)
  - 보고서 및 압축 파일 명 양식
  - **OS_Assignment1 _*수강분류코드_학번_이름* 으로 작성**

수강요일	이론1 월6수5	이론2 목3	실습 금56
수강분류코드	A	B	C

    - **예시 #1)–이론(월6수5)만 수강하는 학생인 경우**
      - **보고서 OS_Assignment1_A_2024123456_홍길동.pdf**
      - **압축 파일 명: OS_Assignment1_A_2024123456_홍길동.tar.xz**
    - **예시 #2)–이론(월6수5 or 목3)과 실습 모두 수강하는 학생인 경우**
      - **보고서 OS_Assignment1_C_2024123456_홍길동.pdf**
      - **압축 파일 명: OS_Assignment1_C_2024123456_홍길동.tar.xz**
      - **+ "해당 이론반 txt 파일 제출"**

Report Requirements

- **실습 수업을 수강하는 학생인 경우**
  - **실습 과목에 과제를 제출**(.tar.xz)
  - **이론 과목에 간단한 .txt 파일**로 제출



    - **이론 과목에 .txt 파일 미 제출 시 감점**
    - **.tar.xz 파일로 제출 하지 않을 시 감점**

- **과제 제출**
  - **KLAS – 강의 과제 제출**
  - **2024년 12월 5일 목요일 23:59까지 제출**
    - **딜레이 받지 않음**
      - 제출 마감 시간 내 미제출시 해당 과제 **0점 처리(예외 없음)**