운영체제 실습

[Assignment #5]

Class : **[B]**

Professor : **[최상호 교수님]**

Student ID : **[2021202089]**

Name : **[오나균]**

Introduction

Assignment 5-1 에서는 Linux 커널의 proc 파일 시스템을 활용하여 프로세스의 정보를 user 공간에서 확인할 수 있도록 하는 것을 목표로 한다. 이를 위해 /proc/proc_학번/processInfo 형태의 가상 파일을 생성하고, 파일을 읽는 시점에 커널 내부의 task_struct 정보를 탐색하여 프로세스의 PID, PPID, UID, GID, utime, stime, state 및 프로세스 이름을 출력하도록 해야한다.

Assignment 5-2 에서는 FAT(File Allocation Table) 구조를 기반으로 하는 user 수준 파일 시스템을 설계하고 구현하는 것을 목표로 한다. FAT 테이블, 파일 Entry 영역, 그리고 데이터 영역으로 구성된 파일 시스템을 메모리 상에서 관리하며, 파일 생성, 데이터 쓰기, 데이터 읽기, 파일 삭제, 저장된 파일 목록 출력 기능을 제공하도록 구현 해야한다. 파일이 저장될 때 고정된 크기의 데이터 블록을 사용하며, 블록 크기를 초과하는 경우 FAT 테이블을 통해 추가 블록을 연결하는 방식으로 파일 데이터를 저장한다. 또한 프로그램 종료 시 파일 시스템 상태(FAT 테이블, 파일 엔트리 정보, 데이터 영역)를 디스크에 저장하고, 실행 시 해당 상태를 복원하여 지속성을 유지하도록 구현해야 한다.

결과화면

Assignment 5-1

1) Assignment5-1 디렉토리 생성 & 이동

```
os2021202089@ubuntu:~$ mkdir Assignment5-1
```

```
os2021202089@ubuntu:~$ cd Assignment5-1
os2021202089@ubuntu:~/Assignment5-1$
```

2) proc_info.c 작성

```
os2021202089@ubuntu:~/Assignment5-1$ vi proc_info.c
```

os2021202089@ubuntu: ~/Assignment5-1

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/syscalls.h>
#include <linux/proc_fs.h>
#include <linux/string.h>
#include <linux/uaccess.h>
#include <linux/jiffies.h>
#include <linux/slab.h>
#include <linux/sched/signal.h>
#include <linux/uidgid.h>
#include <linux/sched/cputime.h>

struct proc_dir_entry *oslab_fp_dir;  // /proc/proc_2021202089
struct proc_dir_entry *oslab_fp_info;  // proc/proc_2021202089/processInfo

static int oslab_is_processed = 0; // 연산 종료 지시자
static int filter_pid = -1;// write를 통해 넘어온 PID 값 저장, -1이면 프로세스 전체 출
력
static int write_flag = 0; // proc 파일에 wrtie 요청이 있었는지 여부 확인
```

3) Makefile 작성

```
os2021202089@ubuntu:~/Assignment5-1$ vi Makefile
```

os2021202089@ubuntu: ~/Assignment5-1

```
obj-m := proc_info.o

KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)

all:
	$(MAKE) -C $(KDIR) M=$(PWD) modules

clean:
	$(MAKE) -C $(KDIR) M=$(PWD) clean
~
~
```

4) Make

```
os2021202089@ubuntu:~/Assignment5-1$ make clean && make
make -C /lib/modules/5.4.282-os2021202089/build M=/home/os2021202089/Assignment5-1 clean
make[1]: Entering directory '/usr/src/linux-5.4.282'
  CLEAN   /home/os2021202089/Assignment5-1/Module.symvers
make[1]: Leaving directory '/usr/src/linux-5.4.282'
make -C /lib/modules/5.4.282-os2021202089/build M=/home/os2021202089/Assignment5-1 modules
make[1]: Entering directory '/usr/src/linux-5.4.282'
  CC [M]  /home/os2021202089/Assignment5-1/proc_info.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/os2021202089/Assignment5-1/proc_info.mod.o
  LD [M]  /home/os2021202089/Assignment5-1/proc_info.ko
make[1]: Leaving directory '/usr/src/linux-5.4.282'
```

5) 모듈 삽입

```
os2021202089@ubuntu:~/Assignment5-1$ sudo insmod proc_info.ko
```

```
os2021202089@ubuntu:~/Assignment5-1$ lsmod | grep proc_info
proc_info              16384  0
```

⇨ lsmod | grep proc_info 명령어를 통해서, proc_info 모듈이 삽입되었음을 알 수 있습니다.

6) Proc 파일 작성

a) Proc 파일에 write 요청이 없었거나, -1 값을 쓴 경우

```
os2021202089@ubuntu:~/Assignment5-1$ sudo insmod proc_info.ko
os2021202089@ubuntu:~/Assignment5-1$ echo -1> /proc/proc_2021202089/processInfo
os2021202089@ubuntu:~/Assignment5-1$ cat /proc/proc_2021202089/processInfo
Pid     PPid    Uid     Gid     utime       stime       State           Name
-------------------------------------------------------------------------------------------
1       0       0       0       47          718         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           2           I (idle)        kworker/0:0H
8       2       0       0       0           0           I (idle)        mm_percpu_wq
9       2       0       0       0           3           S (sleeping)    ksoftirqd/0
10      2       0       0       0           7           I (idle)        rcu_sched
11      2       0       0       0           9           S (sleeping)    migration/0
12      2       0       0       0           0           S (sleeping)    idle_inject/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           0           S (sleeping)    idle_inject/1
17      2       0       0       0           7           S (sleeping)    migration/1
18      2       0       0       0           3           S (sleeping)    ksoftirqd/1
20      2       0       0       0           0           I (idle)        kworker/1:0H
21      2       0       0       0           0           S (sleeping)    cpuhp/2
22      2       0       0       0           0           S (sleeping)    idle_inject/2
23      2       0       0       0           6           S (sleeping)    migration/2
24      2       0       0       0           1           S (sleeping)    ksoftirqd/2
26      2       0       0       0           0           I (idle)        kworker/2:0H
27      2       0       0       0           0           S (sleeping)    cpuhp/3
28      2       0       0       0           0           S (sleeping)    idle_inject/3
29      2       0       0       0           9           S (sleeping)    migration/3
30      2       0       0       0           2           S (sleeping)    ksoftirqd/3
32      2       0       0       0           0           I (idle)        kworker/3:0H
33      2       0       0       0           1           S (sleeping)    kdevtmpfs
34      2       0       0       0           0           I (idle)        netns
35      2       0       0       0           2           S (sleeping)    kauditd
37      2       0       0       0           0           S (sleeping)    khungtaskd
38      2       0       0       0           0           S (sleeping)    oom_reaper
```

```
os2021202089@ubuntu:~/Assignment5-1$ sudo insmod proc_info.ko
os2021202089@ubuntu:~/Assignment5-1$ cat /proc/proc_2021202089/processInfo
Pid     PPid    Uid     Gid     utime       stime       State           Name
-------------------------------------------------------------------------------------------
1       0       0       0       47          718         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           2           I (idle)        kworker/0:0H
8       2       0       0       0           0           I (idle)        mm_percpu_wq
9       2       0       0       0           3           S (sleeping)    ksoftirqd/0
10      2       0       0       0           8           I (idle)        rcu_sched
11      2       0       0       0           9           S (sleeping)    migration/0
12      2       0       0       0           0           S (sleeping)    idle_inject/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           0           S (sleeping)    idle_inject/1
17      2       0       0       0           7           S (sleeping)    migration/1
18      2       0       0       0           3           S (sleeping)    ksoftirqd/1
20      2       0       0       0           0           I (idle)        kworker/1:0H
```

⇨ Proc 파일에 write 요청이 없었거나, -1 값을 쓴 경우에는 pid 가 1 인 프로세스부터 순차적으로 출력됨을 알 수 있습니다.

b) Proc 파일에 특정 프로세스의 PID 값을 입력한 경우

```
os2021202089@ubuntu:~/Assignment5-1$ echo 1 > /proc/proc_2021202089/processInfo
os2021202089@ubuntu:~/Assignment5-1$ cat /proc/proc_2021202089/processInfo
Pid     PPid    Uid     Gid     utime       stime       State           Name
-------------------------------------------------------------------------------------------
1       0       0       0       47          719         S (sleeping)    systemd
```

⇨ 해당 프로세스의 정보만 출력됨을 알 수 있습니다.

Assignment 5-2

1) 디렉토리 생성 및 이동

```
os2021202089@ubuntu:~$ mkdir Assignment5-2
```

```
os2021202089@ubuntu:~$ cd Assignment5-2
os2021202089@ubuntu:~/Assignment5-2$
```

2) fat.c 작성

```
os2021202089@ubuntu:~/Assignment5-2$ vi fat.c
```

os2021202089@ubuntu: ~/Assignment5-2

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

#define MAX_FILE_NUM    100             // MAX_file_number  : 100
#define NUM_BLOCKS      1024            // MAX_NUM_BLOCK    : 1024
#define BLOCK_SIZE      32              // block_size(byte) : 32
#define MAX_FILE_NAME   100             // 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;
}FileEntry;

// File System structure
typedef struct{
        int fat_table[NUM_BLOCKS];      //  FAT Table
        FileEntry directory[MAX_FILE_NUM];
        char data_area[NUM_BLOCKS * BLOCK_SIZE];  // 1024 blocks * 32 byte
}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 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 in the FAT table
void list_files(void);                                  // Display a list of all files in the system
```

3) Makefile 작성

```
os2021202089@ubuntu:~/Assignment5-2$ vi Makefile
```

os2021202089@ubuntu: ~/Assignment5-2

```
CC = gcc
CFLAGS = -Wall -Wextra -std=c11
TARGET = fat
SRCS = fat.c

all: $(TARGET)

$(TARGET): $(SRCS)
        $(CC) $(CFLAGS) -o $@ $^

clean:
        rm -f $(TARGET)
```

4) Make

```
os2021202089@ubuntu:~/Assignment5-2$ make clean && make
rm -f fat
gcc -Wall -Wextra -std=c11 -o fat fat.c
```

5) 각 명령어 실행

```
os2021202089@ubuntu:~/Assignment5-2$ ./fat create A
Warning : No saved state found. Starting fresh.
File 'A' created.
os2021202089@ubuntu:~/Assignment5-2$ ./fat create B
File 'B' created.
os2021202089@ubuntu:~/Assignment5-2$ ./fat list
Files in the file system.
File : A, SIze: 0 bytes
File : B, SIze: 0 bytes
os2021202089@ubuntu:~/Assignment5-2$ ./fat write A "Hello, world"
Data written to 'A'.
os2021202089@ubuntu:~/Assignment5-2$ ./fat list
Files in the file system.
File : A, SIze: 12 bytes
File : B, SIze: 0 bytes
os2021202089@ubuntu:~/Assignment5-2$ ./fat write B "Hello, world"
Data written to 'B'.
os2021202089@ubuntu:~/Assignment5-2$ ./fat write B "Hola, world!"
Data written to 'B'.
os2021202089@ubuntu:~/Assignment5-2$ ./fat list
Files in the file system.
File : A, SIze: 12 bytes
File : B, SIze: 24 bytes
os2021202089@ubuntu:~/Assignment5-2$ ./fat read A
Content of 'A' : Hello, world
os2021202089@ubuntu:~/Assignment5-2$ ./fat read B
Content of 'B' : Hello, worldHola, world!
os2021202089@ubuntu:~/Assignment5-2$ ./fat delete B
File 'B' deleted.
os2021202089@ubuntu:~/Assignment5-2$ ./fat list
Files in the file system.
File : A, SIze: 12 bytes
```

⇨ 처음에 파일이 생성됐을 때, Size = 0 Bytes 로 설정됨을 알 수 있습니다.

⇨ 여러 번 write 명령을 실행했을 때 블록 체인을 따라 기존 데이터 뒤에 새 데이터가 정상적으로 append 되는 것을 확인할 수 있습니다

⇨ 또한 delete 수행 시 FAT 테이블의 블록이 모두 해제됨을 알 수 있습니다.

⇨ Sample output 과 마찬가지로 출력됨을 알 수 있습니다.

고찰

이번 과제를 통해 리눅스의 /proc 파일 시스템이 실제 파일이 아니라 커널 내부 정보를 사용자에게 전달하기 위한 가상 파일 시스템이라는 점을 직접 구현하며 이해할 수 있었다.
Assignment 5-1 에서는 task_struct 를 순회하고 utime, stime, state 등 프로세스 정보를 정리된 형태로 출력하면서, 커널 레벨 자료구조를 안전하게 접근하기 위한 시스템 콜, 커널 메모리등의 사용법을 알게 된 것 같다.

Assignment5-2 에서는 FAT 파일 시스템을 단순화해 직접 구성하면서, 파일이 블록 단위로 저장되고 FAT 테이블이 이를 연결하는 방식이 어떻게 작동하는지를 알 수 있었다.
파일 생성, 삭제, 읽기/쓰기 같은 기본 기능을 모두 구현해 보며, 블록 관리와 디렉토리 관리가 실제 파일 시스템에서 얼마나 중요한 역할을 하는지 알게 되었다
또한 프로그램 종료 후 상태를 파일로 저장하고 재시작 시 파일이 복원됨을 통해 메모리–디스크 간의 데이터 일관성 유지 개념도 알 수 있었다.

전체적으로 이번 과제는 운영체제의 프로세스 관리와 파일 시스템 구조를 코드 레벨에서 직접 다뤄볼 수 있는 좋은 경험이었으며, 단순히 이론으로만 배웠던 개념들이 실제 동작 과정과 연결되면서 이해가 훨씬 깊어진 것 같다.

Reference