

운영체제 실습

# [Assignment #5]

Class : 목 3  
Professor : 최상호  
Student ID : 2020202031  
Name : 김재현

# Introduction

이 과제는 파일 시스템의 개념과 구현에 대한 이해를 심화하기 위해 진행되었습니다. Assignment5-1 에서는 ProcFile 시스템 모듈을 설계하고 구현하여 리눅스 커널에서 ProcFile 시스템의 구조와 작동 원리를 학습합니다.

Assignment5-2 에서는 FAT 기반 가상 파일 시스템을 구현하여 FAT 파일 시스템의 데이터 구조와 파일 및 디렉토리 관리 방식을 실습합니다.

이를 통해 파일 시스템의 설계와 작동 원리에 대한 경험을 쌓는 것을 목표로 합니다.

# 결과화면

5-1)

```
// mapping state to string
const char *state_to_str(const struct task_struct *task)
{
    long state = task->state;
    long exit_state = task->exit_state;
    switch(state)
    {
        case TASK_RUNNING:
            return "R (running)";
        case TASK_INTERRUPTIBLE:
            return "S (sleeping)";
        case TASK_UNINTERRUPTIBLE:
            return "D (disk sleep)";
        case TASK_STOPPED:
            return "T (stopped)";
        case __TASK_TRACED:
            return "t (tracing stop)";
        case TASK_PARKED:
            return "P (parked)";
        case TASK_IDLE:
            return "I (idle)";
    }

    switch(exit_state)
    {
        case EXIT_ZOMBIE:
            return "Z (zombie)";
        case EXIT_DEAD:
            return "X (dead)";
    }

    return "etc.";
}
```

task\_struct 의 state 를 문자열로 변환해주는 함수를 구현했습니다.

과제 3-1 에서 구현했던 함수입니다.

```

78 // kernel memory allocation for very big memory
79 buf = vmalloc(80 * 1024);
80 if (!buf) { // fail to allocate memory
81     printk(KERN_ERR "vmalloc failed\n");
82     return 0;
83 }
84
85
86 j = 0;
87 j += snprintf(buf + j, 81920 - j, "%-5s %-5s %-5s %-5s %-15s %-15s %-15s %-15s\n",
88     "Pid", "PPid", "Uid", "Gid", "utime", "stime", "State", "Name");
89 j += snprintf(buf + j, 81920 - j, "-----\n");
90 // visit all processes
91 for_each_process(task) {
92     pid = task->pid;
93     ppid = task->real_parent->pid;
94     uid = task->cred->uid.val;
95     gid = task->cred->gid.val;
96     utime = task->utime;
97     stime = task->stime;
98     state = state_to_str(task);
99     name = task->comm;
100
101     // if target_pid is positive, only print information of target_pid
102     if(target_pid != -1 && pid != target_pid)
103         continue;
104
105     j += snprintf(buf + j, 81920 - j, "%-5d %-5d %-5d %-5d %-15lld %-15lld %-15s %-s\n",
106         pid, ppid, uid, gid, utime, stime, state, name);
107     found = 1;
108 }
109
110 // if there's not target_pid process
111 if(found == 0)
112     j += snprintf(buf + j, 81920 - j, "No process found with PID %d\n", target_pid);
113

```

모든 프로세스의 정보를 담으려면 버퍼의 크기가 매우 커야합니다. 따라서 vmalloc 을 통해 80KB 버퍼를 동적할당합니다.

for\_each\_process 를 통해 존재하는 모든 프로세스를 차례로 방문하면서 pid, ppid, uid, gid, utime, stime, state, name 정보를 변수에 저장합니다.

if(target\_pid != -1 && pid != target\_pid)가 의미하는 바는, pid 가 양수일 땐, 해당 pid 의 프로세스만 정보를 출력하겠다는 뜻입니다.

target\_pid 가 양수이면서 현재 task 의 pid 와 다르면 continue 를 통해 다음 task 를 방문하기 때문에 오직 해당 pid 의 프로세스 정보만 출력하게 됩니다.

target\_pid 가 -1 이면 모든 프로세스를 출력하라는 뜻이기 때문에, target\_pid != -1 조건에 부합하지 않아 continue 를 실행하지 않게되고, 결국 모든 프로세스의 정보를 출력합니다.

found 의 초기값은 0 입니다. 프로세스의 정보를 하나라도 출력하게 되면 found 는 1 이 됩니다. found 가 초기값 0 을 유지하고 있다는 것은 정보를 하나도 출력하지 못했다는 뜻이므로, 버퍼에 오류메시지를 담아줍니다.

```

114 // copy to user side
115 len_written = strlen(buf);
116 copy_to_user(data_usr, buf, len_written);
117
118 oslab_is_processed++;
119
120 // free memory
121 vfree(buf);
122
123 return len_written;
124 }
125

```

이렇게 프로세스의 정보를 담은 버퍼를 copy\_to\_user 를 통해 유저버퍼에 복사해줍니다. 복사를 완료하면 vfree 를 통해 동적할당한 메모리를 해제해줍니다.

```

126 ssize_t oslab_write(struct file *f, const char __user * data_usr, size_t len, loff_t *off)
127 {
128     ssize_t len_copied;
129     int value;
130
131
132     len_copied = len;
133     copy_from_user(oslab_buffer, data_usr, len_copied);
134
135     // oslab_buffer represents integer
136     if (kstrtoint(oslab_buffer, 10, &value) == 0) {
137         target_pid = value;
138     // oslab_buffer doesn't represent integer
139     } else {
140         target_pid = -1;
141     }
142     printk(KERN_INFO "target_pid: %d\n", target_pid);
143
144     return len_copied;
145 }
146

```

유저버퍼를 통해 입력받은 값을 oslab\_buffer 에 복사합니다.

kstrtoint 함수를 통해 oslab\_buffer 에 복사된 값을 정수로 변환해줍니다.

만일 oslab\_buffer 값이 정수가 아닌 값(ex. 공백)이면 target\_pid 에 -1 을 저장함으로써 모든 프로세스 정보를 출력하도록 구현했습니다.

```

os2020202031@ubuntu:~/Assignment5-1$ echo 1 > /proc/proc_2020202031/processInfo
os2020202031@ubuntu:~/Assignment5-1$ cat /proc/proc_2020202031/processInfo
Pid  PPid  Uid   Gid   utime      stime      State      Name
-----
1    0      0      0      516000000  3200000000 S (sleeping) systemd
os2020202031@ubuntu:~/Assignment5-1$

```

/proc/proc\_2020202031/processInfo 파일에 1 을 write 하면, pid1 에 대한 정보를 출력하는 것을 확인할 수 있습니다.

```

os2020202031@ubuntu:~/Assignment5-1$ echo -1 > /proc/proc_2020202031/processInfo
os2020202031@ubuntu:~/Assignment5-1$ cat /proc/proc_2020202031/processInfo
Pid  PPid  Uid   Gid   utime      stime      State      Name
-----
1    0      0      0      516000000  3200000000 S (sleeping) systemd
2    0      0      0      0           240000000  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
8    2      0      0      0           0           I (idle)    mm_percpu_wq
9    2      0      0      0           320000000  S (sleeping) ksoftirqd/0
10   2      0      0      0           524000000  I (idle)    rcu_sched
11   2      0      0      0           880000000  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           760000000  S (sleeping) migration/1
18   2      0      0      0           480000000  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           840000000  S (sleeping) migration/2
24   2      0      0      0           360000000  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           480000000  S (sleeping) migration/3
30   2      0      0      0           1360000000 S (sleeping) ksoftirqd/3
32   2      0      0      0           0           I (idle)    kworker/3:0H
33   2      0      0      0           800000000  S (sleeping) kdevtmpfs
34   2      0      0      0           0           I (idle)    netns
35   2      0      0      0           0           S (sleeping) kauditd
37   2      0      0      0           400000000  S (sleeping) khungtaskd
2076 1      0      0      1256000000 280000000  S (sleeping) fwupd
2533 1420 1000 1000 800000000 160000000  S (sleeping) gvfsd-metadata
2536 1630 1000 1000 516000000 200000000  S (sleeping) update-notifier
2583 1420 1000 1000 8752000000 4328000000 R (running) gnome-terminal-
2591 2583 1000 1000 2000000000 220000000  S (sleeping) bash
2653 1420 1000 1000 15668000000 17916000000 S (sleeping) code
2655 2653 1000 1000 280000000 400000000  S (sleeping) code
2656 2653 1000 1000 320000000 240000000  S (sleeping) code
2674 1420 1000 1000 0           200000000  S (sleeping) chrome_crashpad
2691 2690 1000 1000 8000000000 1568000000 S (sleeping) code
2696 2690 1000 1000 2000000000 800000000  S (sleeping) code
2705 2656 1000 1000 143676000000 73928000000 S (sleeping) code
2746 2690 1000 1000 19848000000 18676000000 S (sleeping) code
2756 2690 1000 1000 2488000000 2168000000  S (sleeping) code
2757 2690 1000 1000 1004000000 776000000  S (sleeping) code
2823 2746 1000 1000 272000000 1240000000 S (sleeping) cpptools
2855 1420 1000 1000 24000000 360000000  S (sleeping) cpptools-srv
3391 2      0      0      0           208000000  I (idle)    kworker/3:0
4594 1      100 102 680000000 960000000  S (sleeping) systemd-network
4595 2      0      0      0           464000000  I (idle)    kworker/2:1
4635 2      0      0      0           0           I (idle)    kworker/2:0
5631 2      0      0      0           148000000  I (idle)    kworker/0:1
6267 2      0      0      0           0           I (idle)    kworker/1:1
6302 2      0      0      0           244000000  I (idle)    kworker/0:2
6488 2      0      0      0           324000000  I (idle)    kworker/u256:0
7874 2746 1000 1000 432000000 172000000  S (sleeping) code
8419 2      0      0      0           116000000  I (idle)    kworker/u256:1
11075 2      0      0      0           360000000  I (idle)    kworker/2:2
11099 2      0      0      0           0           I (idle)    kworker/2:3
11258 2      0      0      0           240000000  I (idle)    kworker/u256:2
11709 2591 1000 1000 0           0           R (running) cat
os2020202031@ubuntu:~/Assignment5-1$

```

/proc/proc\_2020202031/processInfo 파일에 -1 을 write 하면, 전체 프로세스에 대한 정보를 출력하는 것을 확인할 수 있습니다.

```

os2020202031@ubuntu:~/Assignment5-1$ echo > /proc/proc_2020202031/processInfo
os2020202031@ubuntu:~/Assignment5-1$ cat /proc/proc_2020202031/processInfo
Pid  PPid  Uid    Gid    utime      stime      State      Name
-----
1      0      0      0      516000000  3216000000 S (sleeping)  systemd
2      0      0      0      0          240000000  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
8      2      0      0      0          0          I (idle)      mm_percpu_wq
9      2      0      0      0          320000000  S (sleeping)  ksoftirqd/0
10     2      0      0      0          532000000  I (idle)      rcu_sched
11     2      0      0      0          880000000  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          760000000  S (sleeping)  migration/1
18     2      0      0      0          480000000  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)  kworker/1:1H
2076   1      0      0      1256000000 280000000  S (sleeping)  fwupd
2533   1420   1000   1000   800000000  160000000  S (sleeping)  gvfsd-metadata
2536   1630   1000   1000   516000000  204000000  S (sleeping)  update-notifier
2583   1420   1000   1000   9240000000 4504000000 S (sleeping)  gnome-terminal-
2591   2583   1000   1000   2080000000 224000000  S (sleeping)  bash
2653   1420   1000   1000   1578800000 1799600000 S (sleeping)  code
2655   2653   1000   1000   280000000  400000000  S (sleeping)  code
2656   2653   1000   1000   320000000  240000000  S (sleeping)  code
2674   1420   1000   1000   0          200000000  S (sleeping)  chrome_crashpad
2691   2690   1000   1000   816000000  1576000000 S (sleeping)  code
2696   2690   1000   1000   204000000  840000000  S (sleeping)  code
2705   2656   1000   1000   14438400000 7419600000 S (sleeping)  code
2746   2690   1000   1000   19868000000 1872000000 S (sleeping)  code
2756   2690   1000   1000   2496000000  2168000000 S (sleeping)  code
2757   2690   1000   1000   1004000000  788000000  S (sleeping)  code
2823   2746   1000   1000   272000000  1240000000 S (sleeping)  cpptools
2855   1420   1000   1000   240000000  400000000  S (sleeping)  cpptools-srv
3391   2      0      0      0          208000000  I (idle)      kworker/3:0
4594   1      100   102   680000000  960000000  S (sleeping)  systemd-network
4595   2      0      0      0          464000000  I (idle)      kworker/2:1
5631   2      0      0      0          148000000  I (idle)      kworker/0:1
6267   2      0      0      0          0          I (idle)      kworker/1:1
6302   2      0      0      0          256000000  I (idle)      kworker/0:2
6488   2      0      0      0          324000000  I (idle)      kworker/u256:0
7874   2746   1000   1000   440000000  172000000  S (sleeping)  code
8419   2      0      0      0          132000000  I (idle)      kworker/u256:1
11075  2      0      0      0          480000000  I (idle)      kworker/2:2
11258  2      0      0      0          400000000  I (idle)      kworker/u256:2
11716  2591   1000   1000   0          0          R (running)   cat
os2020202031@ubuntu:~/Assignment5-1$

```

/proc/proc\_2020202031/processInfo 파일에 아무 입력도 하지 않으면, 전체 프로세스에 대한 정보를 출력하는 것을 확인할 수 있습니다.

```
os2020202031@ubuntu:~/Assignment5-1$ echo 5000 > /proc/proc_2020202031/processInfo
os2020202031@ubuntu:~/Assignment5-1$ cat /proc/proc_2020202031/processInfo
Pid  PPid  Uid   Gid   utime      stime      State      Name
-----
No process found with PID 5000
os2020202031@ubuntu:~/Assignment5-1$
```

/proc/proc\_2020202031/processInfo 파일에 존재하지 않는 프로세스 pid 를 입력하면,  
"No process found with PID 000" 이라는 오류문구를 출력하는 것을 확인할 수 있습니다.



## 5-2)

fat\_table 의 빈 공간에는 0 값을, 마지막 block 에는 -1 을 저장하도록 구현했습니다.

따라서 fat\_table 의 0 번 인덱스는 사용하지 않습니다.

### save\_file\_system, load\_file\_system

```
40 // Save the File System to Disk
41 void save_file_system(void) {
42     FILE *f = fopen(FS_STAT, "wb");
43     if (f == NULL) {
44         printf("Error: Could not save file system state.\n");
45         return;
46     }
47     fwrite(&myfat, sizeof(FileSystem), 1, f);
48     fclose(f);
49 }
50
51 // Restore the File System from Disk
52 void load_file_system(void) {
53     FILE *f = fopen(FS_STAT, "rb");
54     if (f == NULL) {
55         printf("Warning: No saved state found. Starting fresh.\n");
56         memset(&myfat, 0, sizeof(FileSystem));
57         return;
58     }
59     fread(&myfat, sizeof(FileSystem), 1, f);
60     fclose(f);
61 }
```

load\_file\_system 은 fs\_state.dat 파일에 존재하는 FileSystem 의 상태에 대한 정보를 현재 FileSystem 에 저장합니다. 만약 파일이 존재하지 않는다면 FileSystem 을 초기화합니다.

save\_file\_system 은 현재 FileSystem 의 상태를 fs\_state.dat 파일에 저장합니다.

## create\_file

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

directory 에 파일이름이 저장돼있다는 것은 해당 filename 을 가진 파일이 이미 존재한다는 뜻이므로 -1 을 리턴합니다.

directory 배열에서 파일이름이 NULL 인 경우는 파일이 존재하지 않음을 의미합니다. 따라서 해당 위치에 파일에 대한 정보를 저장할 것입니다.

76 번 줄은 fat table 에 비어있는 공간이 있는지 탐색합니다. fs\_state.dat 파일은 생성됨과 동시에 0 으로 초기화되므로, fat\_table 이 0 이면 빈 공간임을 의미합니다. 따라서 fat\_table 이 0 인 인덱스를 찾으면 해당 테이블의 값을 -1 로 저장합니다. 파일이름과 시작 블록 인덱스, 크기를 directory 에 저장합니다.

## write\_file

```
93 // File Write
94 int write_file(const char *filename, const char *data) {
95     for (int i = 0; i < MAX_FILE_NUM; i++) {
96         if (strcmp(myfat.directory[i].filename, filename) == 0) {
97             int start_block = myfat.directory[i].start_block;
98             int current_block = start_block;
99             int total_size = myfat.directory[i].size;
100
101             int remaining_data = strlen(data);
102             int data_offset = 0;
103
104             if (start_block == -1) {
105                 // File Not Initialized
106                 printf("File '%s' not initialized. Use create command first.\n", filename);
107                 return -1;
108             }
109 }
```

start\_block 은 해당 파일의 시작블록의 인덱스입니다.

current\_block 은 해당 파일의 마지막 블록 인덱스입니다.

total\_size 는 해당 파일의 크기입니다.

remaining\_data 는 입력하려는 data 의 크기입니다.

data\_offset 은 data 를 파일에 저장한 양입니다.

```

110 while(myfat.fat_table[current_block] != -1)
111     current_block = myfat.fat_table[current_block];
112
113 int block_end = total_size % BLOCK_SIZE;
114
115 // Check the data size
116 if (remaining_data <= BLOCK_SIZE - block_end) {
117     strncpy(&myfat.data_area[current_block * BLOCK_SIZE + block_end], &data[data_offset], remaining_data);
118 } else {
119     int read_size = BLOCK_SIZE - block_end;
120     strncpy(&myfat.data_area[current_block * BLOCK_SIZE + block_end], &data[data_offset], read_size);
121
122     block_end = 0;
123     data_offset += read_size;
124     remaining_data -= read_size;
125
126     while(remaining_data > 0)
127     {
128         int j;
129         int found = 0;
130         for(j = 1; j < NUM_BLOCKS; j++)
131         {
132             if(myfat.fat_table[j] == 0){
133                 found = 1;
134                 break;
135             }
136         }
137         if(!found){
138             printf("there's no more block!");
139             return -1;
140         }
141         myfat.fat_table[current_block] = j;
142         myfat.fat_table[j] = -1;
143         current_block = j;
144
145         int read_size = (remaining_data < BLOCK_SIZE) ? remaining_data : BLOCK_SIZE;
146         strncpy(&myfat.data_area[current_block * BLOCK_SIZE + block_end], &data[data_offset], read_size);
147         data_offset += read_size;
148         remaining_data -= read_size;
149     }
150 }
151
152

```

110 번 줄은 해당 파일의 마지막 블록을 찾아냅니다.

block\_end 는 해당 파일이 마지막 블록에 몇 바이트 저장돼있는지를 나타냅니다.

116 번 줄은 데이터의 크기가 현재 블록의 남은 공간에 들어갈 수 있는지를 판별합니다.  
(BLOCK\_SIZE - block\_end 는 블록의 남은 공간을 의미)

맞다면 데이터를 블록의 남은 공간에 저장하고,

데이터의 크기가 커서 새로운 블록을 할당할 경우엔 else 를 실행합니다. 일단 남은 공간에 데이터를 저장하고, 데이터가 남지 않을 때까지 계속해서 새 블록을 할당하고 데이터를 저장하고를 반복합니다.

130 번 줄 for 문이 비어있는 블록을 찾는 과정입니다.

141 번 줄은 마지막 블록과 새 블록을 연결하는 과정입니다.

146 번 줄은 데이터를 새 블록에 저장하는 과정입니다.

```
152  
153     myfat.directory[i].size += strlen(data);  
154     printf("Data written to '%s'.\n", filename);  
155     return 0;  
156  
157 }  
158 printf("File '%s' not found.\n", filename);  
159 return -1;  
160 }  
161
```

데이터를 저장한만큼 파일의 크기도 증가시킵니다.

## read\_file

```
162 // Read Files
163 int read_file(const char *filename) {
164     for (int i = 0; i < MAX_FILE_NUM; i++) {
165         if (strcmp(myfat.directory[i].filename, filename) == 0) {
166             int start_block = myfat.directory[i].start_block;
167             int block = start_block;
168             int total_size = myfat.directory[i].size;
169             int bytes_read = 0;
170
171             if (start_block == -1) {
172                 printf("File '%s' not initialized. Use create command first.\n", filename);
173                 return -1;
174             }
175
176             printf("Content of '%s': ", filename);
177
178             while(bytes_read < total_size)
179             {
180                 int remaining_bytes_in_block = total_size - bytes_read;
181                 int read_size = (remaining_bytes_in_block < BLOCK_SIZE) ? remaining_bytes_in_block : BLOCK_SIZE;
182                 printf("%.4s", read_size, &myfat.data_area[block * BLOCK_SIZE]);
183
184                 block = myfat.fat_table[block];
185                 bytes_read += read_size;
186             }
187
188             printf("\n");
189             return 0;
190         }
191     }
192
193     printf("File '%s' not found.\n", filename);
194     return -1;
195 }
196
197
```

read\_file 은 start block 부터 다음 block 으로 차례로 read 하면서 파일의 크기만큼 읽습니다.

## delete\_file

```
198 // File Deletion
199 int delete_file(const char *filename) {
200     for (int i = 0; i < MAX_FILE_NUM; i++) {
201         if (strcmp(myfat.directory[i].filename, filename) == 0) {
202             int start_block = myfat.directory[i].start_block;
203             int block = start_block;
204
205             // Release Cluster at FAT Table
206             while (block != -1) {
207                 int next_block = myfat.fat_table[block];
208                 myfat.fat_table[block] = 0; // Release Cluster
209                 block = next_block;
210             }
211
212             // Remove file at Directory
213             myfat.directory[i].filename[0] = '\0';
214             printf("File '%s' deleted.\n", filename);
215
216             return 0;
217         }
218     }
219     printf("File '%s' not found.\n", filename);
220     return -1;
221 }
222
```

206 번 줄은 start\_block 부터 차례로 fat table 에 0 을 채워줌으로써 해당 블록이 해제되어 사용가능한 빈 공간이 됐음을 나타내줍니다.

또한 directory 의 filename 을 NULL 로 만들어줌으로써 해당 파일이 완전히 삭제되었음을 나타냅니다.

## list\_files

```
223 // Display a list of all files in the system.
224 void list_files() {
225     printf("Files in the file system:\n");
226     for (int i = 0; i < MAX_FILE_NUM; i++) {
227         if (myfat.directory[i].filename[0] != '\0') {
228             printf("File: %s, Size: %d bytes\n", myfat.directory[i].filename, myfat.directory[i].size);
229         }
230     }
231 }
```

directory 에서 파일명이 존재하는 모든 파일의 이름과 크기를 출력합니다.

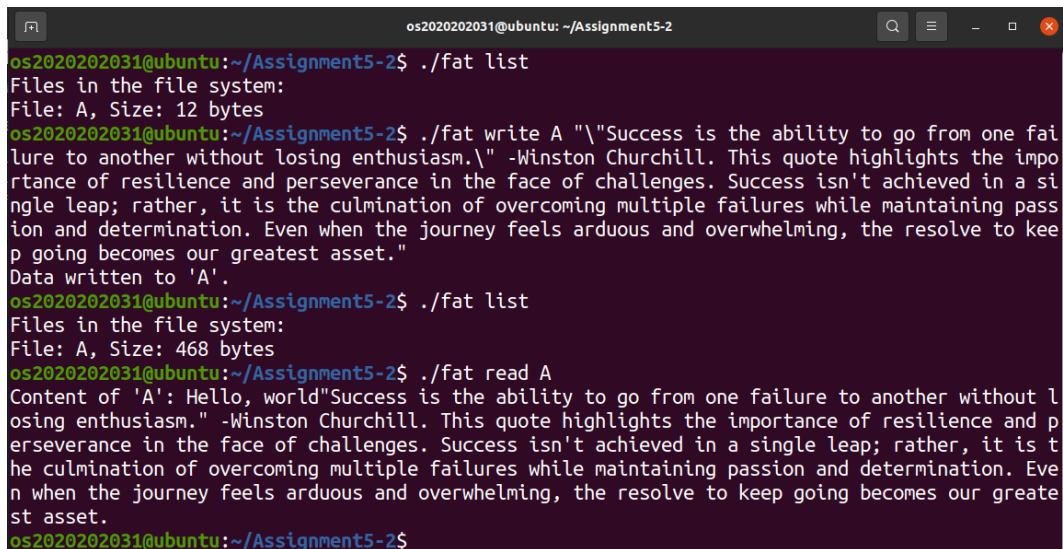


```

os2020202031@ubuntu:~/Assignment5-2$ ./fat create A
Warning: No saved state found. Starting fresh.
File 'A' created.
os2020202031@ubuntu:~/Assignment5-2$ ./fat create B
File 'B' created.
os2020202031@ubuntu:~/Assignment5-2$ ./fat list
Files in the file system:
File: A, Size: 0 bytes
File: B, Size: 0 bytes
os2020202031@ubuntu:~/Assignment5-2$ ./fat write A "Hello, world"
Data written to 'A'
os2020202031@ubuntu:~/Assignment5-2$ ./fat list
Files in the file system:
File: A, Size: 12 bytes
File: B, Size: 0 bytes
os2020202031@ubuntu:~/Assignment5-2$ ./fat write B "Hello, world"
Data written to 'B'
os2020202031@ubuntu:~/Assignment5-2$ ./fat write B "Hola, world!"
Data written to 'B'
os2020202031@ubuntu:~/Assignment5-2$ ./fat list
Files in the file system:
File: A, Size: 12 bytes
File: B, Size: 24 bytes
os2020202031@ubuntu:~/Assignment5-2$ ./fat read A
Content of 'A': Hello, world
os2020202031@ubuntu:~/Assignment5-2$ ./fat read B
Content of 'B': Hello, worldHola, world!
os2020202031@ubuntu:~/Assignment5-2$ ./fat delete B
File 'B' deleted.
os2020202031@ubuntu:~/Assignment5-2$ ./fat list
Files in the file system:
File: A, Size: 12 bytes
os2020202031@ubuntu:~/Assignment5-2$ █

```

제안서의 sample output 과 완전히 동일한 결과를 보이는 것을 확인할 수 있습니다.



```

os2020202031@ubuntu: ~/Assignment5-2
os2020202031@ubuntu:~/Assignment5-2$ ./fat list
Files in the file system:
File: A, Size: 12 bytes
os2020202031@ubuntu:~/Assignment5-2$ ./fat write A "\"Success is the ability to go from one failure to another without losing enthusiasm.\" -Winston Churchill. This quote highlights the importance of resilience and perseverance in the face of challenges. Success isn't achieved in a single leap; rather, it is the culmination of overcoming multiple failures while maintaining passion and determination. Even when the journey feels arduous and overwhelming, the resolve to keep going becomes our greatest asset."
Data written to 'A'.
os2020202031@ubuntu:~/Assignment5-2$ ./fat list
Files in the file system:
File: A, Size: 468 bytes
os2020202031@ubuntu:~/Assignment5-2$ ./fat read A
Content of 'A': Hello, world"Success is the ability to go from one failure to another without losing enthusiasm." -Winston Churchill. This quote highlights the importance of resilience and perseverance in the face of challenges. Success isn't achieved in a single leap; rather, it is the culmination of overcoming multiple failures while maintaining passion and determination. Even when the journey feels arduous and overwhelming, the resolve to keep going becomes our greatest asset.
os2020202031@ubuntu:~/Assignment5-2$

```

block size 를 넘기는 데이터를 입력해도 잘 저장되는 것을 확인할 수 있습니다.

## 고찰

Assignment 5-1 에서 `vmalloc` 으로 동적할당을 진행할 때, 크기를 작게 할당하여, 모든 프로세스의 정보를 출력할 때, 동적할당 공간이 부족하여 오류가 계속해서 났습니다. 평상시 `user mode` 로 프로그램을 작성할 땐, 큰 공간이 필요할 때가 많이 없어서 동적할당 크기에 대해 별 고민을 하지 않았었는데, 이번 실습을 통해, 메모리가 얼마나 필요한지 사전에 예상하고, 이에 대한 예외처리를 잘 하는 것 또한 중요하다는 것을 깨달았습니다.

## Reference

vmalloc: <https://stackoverflow.com/questions/116343/what-is-the-difference-between-vmalloc-and-kmalloc>

kstrtoint: <https://stackoverflow.com/questions/43456885/kstrtoint-in-sysfs-for-c-kernel-module>