운영체제

Assignment #5

Class : A

Professor : 김태석 교수님

Student ID : 2022202065

Name : 박나림

Introduction

이번 과제는 2 가지로 이루어진다.

첫번째는 Process 들의 정보를 출력하는 processInfo 파일을 작성하는 모듈을 구현하는 것이다. 모듈을 적재하면, proc_(본인학번) 디렉토리에 processInfo 파일을 생성시키고 write 요청에 따라 pid 값을 받아서 read 요청 시 해당 pid 의 프로세스 정보를 출력한다. 이때 write 요청이 없었거나 -1 을 write 할 시엔 pid 순으로 모든 프로세스의 정보를 출력하도록 한다.

두번째는 FAT (File Allocation Table)에 기반한 Virtual File System 을 구현하는 것이다. creation, writing, reading, deleting, listing, saving/restoring file 기능을 제공하도록 각 함수를 구현하여 FAT file system 을 이해하는 것이 목표이다.

결과화면

Assignment 5-1

process 들의 정보를 출력하는 proc 파일을 작성한다. 모듈을 적재할 때 process_2022202065 디렉토리로 processInfo 파일이 생성된다. 해당 파일에 시스템콜을 요청할 때 처리하는 함수를 구현하는데, write 를 요청할 때 pid 값이 전달되므로 이 값을 받으면 copy_from_user()을 통해 복사한 후 정수로 변환하여 targetPid 로 설정한다.

```
static ssize_t proc_info_write(struct file *f, const char __user *data_user, size_t len,
loff_t *off)
{
        char pidBuffer[256];            //save pid
        int pid;

        if (len > sizeof(pidBuffer)) { //size is exceeded
                return -EINVAL;
        }

        if (copy_from_user(pidBuffer, data_user, len)) { //saving the input buffer fails
                return -EFAULT;
        }

        pidBuffer[len]='\0';            //set end string

        if (kstrtoint(pidBuffer, 10, &pid) == 0) { //convert str to int(pid)
                targetPid = pid;        //set tartget Pid
        }
```

read 시스템 콜 요청 시, seq_file API 를 이용하여 seq_read 로서 아래 함수로 연결되게 구현하였다. seq_file 을 사용하면 여러 process 들을 더욱 간편하게 처리할 수 있기 때문이다. 해당 struct 를 인자로 받아서 seq_printf 로 파일에 출력하도록 한다. utime 과 stime 은 jiffies 를 이용하여 초로 변환하여 출력한다. targetPid 가 기본 -1 로 되어 있는데, write 요청이 없었거나 -1 그자체로 들어온 경우 모든 프로세스의 정보를 pid 순으로 출력하고, write 로 -1 이외의 값이 들어온 경우 해당 pid 프로세스 하나만을 출력한다.

```
static int proc_info_print(struct seq_file *sf, void *v)
{
        struct task_struct *task;

        //==================== Information output from the process ====================//
        seq_printf(sf, "%-5s %-5s %-5s %-5s %-10s %-10s %-15s %s\n", "Pid", "PPid", "Uid", "Gid", "utime",
"stime", "State", "Name");

        seq_printf(sf, "-------------------------------------------------------------------------------
\n");

        if(targetPid == -1) { //all process
                for_each_process(task) {
                        //convert to seconds
                        unsigned long Utime = jiffies_to_msecs(task->utime) / 1000;
                        unsigned long Stime = jiffies_to_msecs(task->stime) / 1000;

                        seq_printf(sf, "%-5d %-5d %-5d %-5d %-10lu %-10lu %-15s %s\n",
                        task->pid, task->real_parent->pid, __kuid_val(task->cred->uid), __kgid_val(task-
>cred->gid), Utime, Stime, getState(task), task->comm);
                }
        }
        else { //target process
                task = pid_task(find_vpid(targetPid), PIDTYPE_PID); //Find task by pid
                if (task) {
                        //convert to seconds
                        unsigned long Utime = jiffies_to_msecs(task->utime) / 1000;
                        unsigned long Stime = jiffies_to_msecs(task->stime) / 1000;

                        seq_printf(sf, "%-5d %-5d %-5d %-5d %-10lu %-10lu %-15s %s\n",
                        task->pid, task->real_parent->pid, __kuid_val(task->cred->uid), __kgid_val(task-
>cred->gid), Utime, Stime, getState(task), task->comm);
```

proc_info.c 코드를 작성한 후 make 한 뒤 insmod 로 해당 모듈을 적재한다.

```
os2022202065@ubuntu:~/Assignment5/Assignment5-1$ make
make -C /lib/modules/5.4.282-os2022202065/build M=/home/os2022202065/Assignment5
/Assignment5-1 modules
make[1]: Entering directory '/usr/src/linux-5.4.282'
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-5.4.282'
os2022202065@ubuntu:~/Assignment5/Assignment5-1$ sudo insmod proc_info.ko
```

#case 1: proc 파일에 write 요청이 없었던 경우

pid 순서대로 모든 프로세스의 정보를 파일에 출력한 것을 볼 수 있다.

```
os2022202065@ubuntu:~/Assignment5/Assignment5-1$ cat /proc/proc_2022202065/processInfo
Pid     PPid    Uid     Gid     utime       stime       State           Name
--------------------------------------------------------------------------------
1       0       0       0       1504000     1627098     S (sleeping)    systemd
2       0       0       0       0           64000       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           48000       S (sleeping)    ksoftirqd/0
10      2       0       0       0           96000       I (idle)        rcu_sched
11      2       0       0       0           144000      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           112000      S (sleeping)    migration/1
18      2       0       0       0           96000       S (sleeping)    ksoftirqd/1
20      2       0       0       0           0           I (idle)        kworker/1:0H
21      2       0       0       0           0           S (sleeping)    kdevtmpfs
22      2       0       0       0           0           I (idle)        netns
23      2       0       0       0           16000       S (sleeping)    kauditd
24      2       0       0       0           656000      I (idle)        kworker/0:2
25      2       0       0       0           0           S (sleeping)    khungtaskd
26      2       0       0       0           0           S (sleeping)    oom_reaper
27      2       0       0       0           0           I (idle)        writeback
28      2       0       0       0           0           S (sleeping)    kcompactd0
29      2       0       0       0           0           S (sleeping)    ksmd
```

#case 2: proc 파일에 특정 process pid 값을 입력한 경우

해당 프로세스의 정보만을 파일에 출력한 것을 볼 수 있다.

```
os2022202065@ubuntu:~/Assignment5/Assignment5-1$ echo 1 > /proc/proc_2022202065/processInfo
os2022202065@ubuntu:~/Assignment5/Assignment5-1$ cat /proc/proc_2022202065/processInfo
Pid     PPid    Uid     Gid     utime       stime       State           Name
--------------------------------------------------------------------------------
1       0       0       0       1504000     1627098     S (sleeping)    systemd
os2022202065@ubuntu:~/Assignment5/Assignment5-1$ echo 16 > /proc/proc_2022202065/processInfo
os2022202065@ubuntu:~/Assignment5/Assignment5-1$ cat /proc/proc_2022202065/processInfo
Pid     PPid    Uid     Gid     utime       stime       State           Name
--------------------------------------------------------------------------------
16      2       0       0       0           0           S (sleeping)    idle_inject/1
```

#case 2-2: proc 파일에 -1 값을 write 한 경우

기본 상태와 동일하게 모든 프로세스의 정보를 pid 순으로 출력한다.

```
os2022202065@ubuntu:~/Assignment5/Assignment5-1$ echo -1 > /proc/proc_2022202065/processInfo
os2022202065@ubuntu:~/Assignment5/Assignment5-1$ cat /proc/proc_2022202065/processInfo
Pid     PPid    Uid     Gid     utime       stime       State           Name
--------------------------------------------------------------------------------
1       0       0       0       1504000     1627098     S (sleeping)    systemd
2       0       0       0       0           64000       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           48000       S (sleeping)    ksoftirqd/0
10      2       0       0       0           96000       I (idle)        rcu_sched
11      2       0       0       0           144000      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           112000      S (sleeping)    migration/1
18      2       0       0       0           96000       S (sleeping)    ksoftirqd/1
20      2       0       0       0           0           I (idle)        kworker/1:0H
```

Assignment 5-2

FAT file system 을 구현하기 위해, 제시된 size 들을 define 하고 file entry 를 구조체로 작성한 다음 그 외 fatTable 과 dataBlock, write 시 block 할당 계산을 위한 blockOffset 배열들을 만든다.

```
#define MAX_FILE 100
#define NUM_DATA_BLOCK 1024
#define BLOCK_SIZE 32
#define MAX_FILE_NAME 100

typedef struct {
        char fileName[MAX_FILE_NAME];
        int fileSize;
        int firstBlock; //starting block in the FAT table (start sector)
        bool use; //use of free
} file;
file fileEntry[MAX_FILE];
int fatTable[NUM_DATA_BLOCK]; //File Allocation Table
char dataBlock[NUM_DATA_BLOCK][BLOCK_SIZE]; //actual storage location where file
  data is written
int blockOffset[NUM_DATA_BLOCK] = {0};
```

fat.c 를 실행하게 되면 그때마다 저장된 file 기록들을 불러오는 식으로 관리해야 되기 때문에, load_file_system()으로 file 정보들을 fread 로 저장한 뒤 명령어를 수행하고 save_file_system()으로 fwrite 를 통해 저장하고 종료하는 방식으로 구현한다.

create 를 실행하게 되면 파일이 존재하는 지 확인한 후 없으면 생성하는 식으로 한다. 이때 file entry 의 free 부분에 파일 이름을 저장하고, data block 의 free 부분 index 를 entry 의 firstBlock 에 저장하여 해당 위치를 가리키도록 한다. 또한 use 변수를 통해 사용중임을 표시한다. fat table 에선 현재 끝 위치이니 기본 0 에서 -1 로 바꾼다.

```
void create_file(const char *name) {
        //Check if file exists
        for (int i=0; i<MAX_FILE; i++) {
                if (fileEntry[i].use && strcmp(fileEntry[i].fileName, name)==0) {
                        printf("file exist!\n"); exit(1);
                }
        }

        //Find file entries that are free
        int fileIndex = -1;
        for (int i=0; i<MAX_FILE; i++) {
                if (!fileEntry[i].use) { //free file
                        fileIndex = i;
                        break;
                }
        }
        if (fileIndex == -1) {
                printf("All file entries are full!\n"); exit(1);
        }

        //Find first block(fatTable) that are free
        int blockIndex = -1;
        for (int i=0; i<NUM_DATA_BLOCK; i++) {
                if (fatTable[i] == 0) {
                        blockIndex = i;
                        break;
                }
        }
        if (blockIndex == -1) {
                printf("All data block(FAT table) are full!\n"); exit(1);
        }

        //Create file entry
        strcpy(fileEntry[fileIndex].fileName, name);
        fileEntry[fileIndex].fileSize = 0;
        fileEntry[fileIndex].firstBlock = blockIndex;
        fileEntry[fileIndex].use = true;

        fatTable[blockIndex] = -1; //first block of a file is marked
```

write 시, 해당 파일 위치를 찾은 다음 content 의 시작 주소를 포인터로 가리켜서 while 문으로 주소 위치를 증가하는 식으로 반복한다. 이때 파일에 이어 쓰는 형식으로 되어야 하므로 파일의 마지막 block 을 찾아서 그 block 부터 저장되도록 한다. 한 block 내에서 얼마나 써져있는지는 block offset 배열로 관리된다. 이것과 남은 문자열의 길이를 비교하여 작으면 block 에 이어 쓰고, 크면 다음 block 을 할당하여 작성한다. 완료되면 파일 크기를 문자열만큼 증가시키고 종료한다.

```
112         const char *contentPtr = content;
113         int currBlock = fileEntry[fileIndex].firstBlock;
114
115         //Find the last block of the current file
116         while (fatTable[currBlock] != -1) {
117                 currBlock = fatTable[currBlock];
118         }
119
120         //Continue writing from the last block
121         while (*contentPtr) {
122                 if (currBlock == -1) return;
123
124                 int remainingSpace = BLOCK_SIZE - blockOffset[currBlock];
125                 int writeSize = strlen(contentPtr) < remainingSpace ? strlen(contentPtr) :
    remainingSpace;
126
127                 memcpy(dataBlock[currBlock]+blockOffset[currBlock], contentPtr, writeSize);
128                 contentPtr += writeSize; //move start address of content
129                 blockOffset[currBlock] += writeSize;
130
131                 if (blockOffset[currBlock] == BLOCK_SIZE) {
132                         int nextBlock = -1;
133                         for (int i=0; i<NUM_DATA_BLOCK; i++) {
134                                 if (fatTable[i] == 0) {
135                                         nextBlock = i;
136                                         break;
137                                 }
138                         }
139                         if (nextBlock == -1) {
140                                 printf("All data block(FAT table) are full!\n"); exit(1);
141                         }
142
143                         fatTable[currBlock] = nextBlock;
144                         fatTable[nextBlock] = -1;
145                         currBlock = nextBlock;
146                 }
147         }
148         fileEntry[fileIndex].fileSize += strlen(content);
```

read 시, 해당 파일을 찾은 다음 firstBlock 으로 처음부터 끝 block 이 존재할 때까지 fat Table 을 탐색하며 출력한다.

```
printf("Content of '%s': ", name);

int currBlock = fileEntry[fileIndex].firstBlock;
int readLen = 0;
int totalSize = fileEntry[fileIndex].fileSize;

while(currBlock != -1) {
        for (int i=0; i<BLOCK_SIZE; i++) {
                if (readLen < totalSize) {
                        putchar(dataBlock[currBlock][i]);
                        readLen++;
                }
        }
        currBlock = fatTable[currBlock];
}
printf("\n");
```

delete 시, 파일을 찾은 다음 첫번째 block 부터 마지막 block 까지 탐색하며 해당 fatTable 을 초기화 시키고 파일 정보들도 초기화 시킨다.

```
int currBlock = fileEntry[fileIndex].firstBlock;

while (currBlock != -1) {
        int nextblock = fatTable[currBlock];
        fatTable[currBlock] = 0;
        currBlock = nextblock;
}
strcpy(fileEntry[fileIndex].fileName, "-");
fileEntry[fileIndex].fileSize = 0;
fileEntry[fileIndex].use = false;
```

list 시, use 를 이용하여 사용중인 파일이 존재하면 출력한다.

```
void list_files() {
        printf("Files in the file system:\n");

        for (int i=0; i<MAX_FILE; i++) {
                if (fileEntry[i].use) { //file exist
                        printf("File: %s, Size: %d bytes\n", fileEntry[i].fileName,
fileEntry[i].fileSize);
                }
        }
        return;
}
```

fat.c 작성 완료 후 Makefile 을 만든다.

```
all: fat

fat: fat.c
        gcc -o fat fat.c
```

다음은 각 명령어들을 실행한 결과이다.

#case 1: data 가 한 block 내에 쓰이는 경우

A, B 파일을 생성 후 A 파일에 write 하여 size 를 확인한 결과, 문자열 크기대로 저장된 것을 볼 수 있다.

```
os2022202065@ubuntu:~/Assignment5/Assignment5-2$ ./fat create A
Warning: No saved state found. Starting fresh.
File 'A' created.
os2022202065@ubuntu:~/Assignment5/Assignment5-2$ ./fat create B
File 'B' created.
os2022202065@ubuntu:~/Assignment5/Assignment5-2$ ./fat list
Files in the file system:
File: A, Size: 0 bytes
File: B, Size: 0 bytes
os2022202065@ubuntu:~/Assignment5/Assignment5-2$ ./fat write A "Hello, world"
Data written to 'A'.
os2022202065@ubuntu:~/Assignment5/Assignment5-2$ ./fat list
Files in the file system:
File: A, Size: 12 bytes
File: B, Size: 0 bytes
```

B 파일에는 문자열을 연달아 write 했을 때, read 하면 이어져서 저장된 것을 확인할 수 있다.

```
os2022202065@ubuntu:~/Assignment5/Assignment5-2$ ./fat write B "Hello, world"
Data written to 'B'.
os2022202065@ubuntu:~/Assignment5/Assignment5-2$ ./fat write B "Hola, world!"
Data written to 'B'.
os2022202065@ubuntu:~/Assignment5/Assignment5-2$ ./fat list
Files in the file system:
File: A, Size: 12 bytes
File: B, Size: 24 bytes
os2022202065@ubuntu:~/Assignment5/Assignment5-2$ ./fat read A
Content of 'A': Hello, world
os2022202065@ubuntu:~/Assignment5/Assignment5-2$ ./fat read B
Content of 'B': Hello, worldHola, world!
os2022202065@ubuntu:~/Assignment5/Assignment5-2$ ./fat delete B
File 'B' deleted.
os2022202065@ubuntu:~/Assignment5/Assignment5-2$ ./fat list
Files in the file system:
File: A, Size: 12 bytes
```

#case 2: data 가 한 block 크기를 초과하여 써지는 경우 (FAT table 과정 출력 포함)

A 파일에 block size 인 32byte 까지(0~31) write 했을 때, 0 번째 block 에만 작성된 화면이다.

```
os2022202065@ubuntu:~/Assignment5/Assignment5-2$ ./fat write A "1234567890123456
789012345678901"
Data written to 'A'.
os2022202065@ubuntu:~/Assignment5/Assignment5-2$ ./fat list
Files in the file system:
File: A, Size: 31 bytes
os2022202065@ubuntu:~/Assignment5/Assignment5-2$ ./fat read A
Content of 'A': 1234567890123456789012345678901
FAT Table:
Block 0 -> -1
Block 1 -> 0
Block 2 -> 0
Block 3 -> 0
Block 4 -> 0
Block 5 -> 0
Block 6 -> 0
Block 7 -> 0
Block 8 -> 0
Block 9 -> 0
```

이후 한 문자를 이어 32byte 까지 write 하면 초과되어 다음 block 에 할당되면서 그전 block 은 FAT table 에서 1 번을 가리키게 된다.

```
os2022202065@ubuntu:~/Assignment5/Assignment5-2$ ./fat write A "2"
Data written to 'A'.
os2022202065@ubuntu:~/Assignment5/Assignment5-2$ ./fat list
Files in the file system:
File: A, Size: 32 bytes
os2022202065@ubuntu:~/Assignment5/Assignment5-2$ ./fat read A
Content of 'A': 12345678901234567890123456789012
FAT Table:
Block 0 -> 1
Block 1 -> -1
Block 2 -> 0
Block 3 -> 0
Block 4 -> 0
Block 5 -> 0
Block 6 -> 0
Block 7 -> 0
Block 8 -> 0
Block 9 -> 0
```

B 파일을 생성하여 write 하면 다음 block 인 2 번째에 할당된다. 이때 다른 파일이므로 2 번 block 에선 FAT table 이 가리키지 않은 상태이다. (-1 로 끝을 나타냄)

```
os2022202065@ubuntu:~/Assignment5/Assignment5-2$ ./fat write B "hello"
Data written to 'B'.
os2022202065@ubuntu:~/Assignment5/Assignment5-2$ ./fat list
Files in the file system:
File: A, Size: 32 bytes
File: B, Size: 5 bytes
os2022202065@ubuntu:~/Assignment5/Assignment5-2$ ./fat read B
Content of 'B': hello
FAT Table:
Block 0 -> 1
Block 1 -> -1
Block 2 -> -1
Block 3 -> 0
```

C 파일을 생성하여 write 하면 마찬가지로 3 번 block 이 할당되고, 2 번 block 과는 다른 파일이니 연결되지 않는 것을 볼 수 있다.

```
os2022202065@ubuntu:~/Assignment5/Assignment5-2$ ./fat create C
File 'C' created.
os2022202065@ubuntu:~/Assignment5/Assignment5-2$ ./fat write C "world"
Data written to 'C'.
os2022202065@ubuntu:~/Assignment5/Assignment5-2$ ./fat list
Files in the file system:
File: A, Size: 32 bytes
File: B, Size: 5 bytes
File: C, Size: 5 bytes
os2022202065@ubuntu:~/Assignment5/Assignment5-2$ ./fat read C
Content of 'C': world
FAT Table:
Block 0 -> 1
Block 1 -> -1
Block 2 -> -1
Block 3 -> -1
Block 4 -> 0
```

여기서 delete A 를 하면 A 에 할당된 0, 1 block 들이 fat Table 에서 초기화 된다.

```
FAT Table:
Block 0 -> 0
Block 1 -> 0
Block 2 -> -1
Block 3 -> -1
Block 4 -> 0
Block 5 -> 0
Block 6 -> 0
Block 7 -> 0
Block 8 -> 0
Block 9 -> 0
```

고찰

5-1

처음엔 seq_file API 를 사용하는 대신 read 함수를 직접 구현하는 방식으로 하였는데, 여러 오류가 발생하면서 코드가 복잡해져서 다른 방법을 찾아보다 해당 API 를 사용하는 방식으로 수정하였다. seq_file 을 사용하면 seq_printf 를 사용할 수 있어서 sprint 대신에 버퍼를 신경 쓰지 않고 바로 출력할 수 있는 점이 크게 개선되었다. 또한 반복적으로 출력할 때도 for_each_process 매크로를 사용하여 더 간편한 코드로 수정될 수 있었다.

Gid 와 Uid 를 출력할 땐 from_kgid 를 사용했다가 현재 커널에선 지원하지 않는다는 에러가 발생하여 __kgid_val()을 사용하는 방식으로 수정하였다. 여기서는 cred 로 써야하니 헤더에 cred.h 를 추가하였다.

"ISO C90 forbids mixed declarations and code"라는 경고가 발생했었는데, C90 에서는 모든 변수 선언이 함수 처음 부분에 위치해야 한다는 경고였다. 원래는 필요할 때마다 그때그때 변수를 선언했었는데, 이번에 새로 이런 경고를 알게 되어서 앞으로 코드를 작성할 때 순서에도 주의해야겠다고 느꼈다.

코드를 다 작성한 후엔 write 로 특정 pid 를 전달하는 명령에서 계속 잘못된 주소라는 오류가 발생하였다. 이에 디버깅 코드도 추가하면서 코드를 수정해봤지만 여전히 같은 오류가 발생하였는데, 이는 echo 명령어를 입력할 때 pid 입력 후 공백을 같이 입력해야 됐었는데 하지 않아서 처리에 문제가 생긴 것이었다. 명령어를 수정하니 정상적으로 특정 pid process 만 출력될 수 있었다.

5-2

write 함수를 작성할 때, 단순히 content 길이를 계산해서 그걸로 while 문을 작성하고 빈 block 을 순회하는 식으로 하였더니, 이어서 작성되지 않고 덮어써지는 식으로 저장되었다. 따라서 block offset 전역 변수 배열을 따로 만들어서 현재까지 작성한 위치를 기록하였다. 이것도 load 랑 save 할 때 같이 fread, fwrite 를 해줘야 정상적으로 반영되었다.

read 함수를 작성할 때도 현재 block 이 -1 이 아닐 때까지 순회하면서 fat table 을 따라가서 마지막에 기록된 block 까지 출력되는 형식으로 수정하였더니 정상적으로 출력될 수 있었다. 또한 원래는 buffer 를 만들어서 저장하는 방식으로 하였는데, 그렇게 하니 버퍼 비우는 작업까지 필요해서 한 단어씩 출력하는(putchar) 것으로 수정하였다.

Reference

강의자료 참고