컴퓨터학부 20160548 이승현 나반

1. 과제 개요

가상 디스크를 위한 간단한 파일 시스템을 구현한다. 주어진 소스코드 (ssufs-disk.h, ssufs-disk.c)를 이용하여 다섯가지 함수들을 구현한다. 다섯가지 함수는 다음과 같다. 파일을 생성하는 ssufs_create(), 파일을 삭제하는 ssufs_delete(), 파일을 여는 ssufs_open(), open된 파일에서 데이터를 읽는 ssufs_read(), open된 파일에 데이터를 쓰는 ssufs_write().

구현된 기능은 주어진 ssufs_test.c, ssufs_test.sh 파일과 추가로 만든 테스트 프로그램을 이용하여 테스트한다.

- **구현 방법**

ssufs_write() 이외의 함수들은 작성된 코드를 보면 간단히 알 수 있는 내용이므로 생략한다

- **ssufs_write()**

ssufs_write()를 구현하는데 있어 까다로운 부분은 "쓰기가 실패한 경우 실패가 발생하기 전에 할당 된 모든 데이터 블록을 해제하고 해당 블록은 파일시스템에 반환해야 함. -> 실패한 쓰기는 파일시스템이 시작된 상태와 동일한 상태를 유지"하도록 해야 한다는 것이다.

이를 쉽게 하기 위하여 먼저 write를 하기 전에 실패가 발생할 수 있는 상황들을 모두 체크하였다. 첫번째로 파일의 최대 크기를 넘어서 write를 하게 될 경우 실패하므로, 먼저 최종 파일 크기를 확인하였다. 두번째로 새로운 data block을 할당하지 못한 경우에 실패하므로 필요한 data block을 모두 할당할 수 있는지 확인하도록 했다. 새로운 data block 할당이 가능한지 확인하는 과정은 다음과 같다. inode의 direct_blocks 배열을 순차적으로 확인하며 새로운 data block이 필요한 경우 ssufs_allocDataBlock() 함수를 호출하여 할당 받는다. 이 때 할당이 불가능하면 ssufs_allocDataBlock()가 -1을 리턴 한다. -1이 리턴 됐을 때 앞서 새롭게 할당 받았던 data block이 존재한다면 해당 block들을 다시 free하여 ssufs_write()함수가 호출되기 이전 상태로 되돌린다. 그 후 -1을 리턴 하여 함수를 종료한다.

위의 두가지 경우를 모두 처리하면 더 이상 쓰기 실패 상황은 발생하지 않는다. 따라서 위의 두가지 경우를 확인한 후에는 별도의 예외 처리 없이 write를 진행하도록 구현했다.

2. 결과

- 1. 제공된 테스트 프로그램 이용한 테스트

```
shlee@shlee-virtual-machine:~/workspace/ssuos/project6$ ./ssufs_test.sh
Write Data: 0
Seek: 0
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<DISK STATE>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
DISK NAME: ssufs
INODE FREELIST:      1 x x x x x x x
DATA BLOCK FREELIST: 1 x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x
INODE 0
STATUS: 1       NAME    f1.txt  SIZE    64      DATABLOCK       0 -1 -1 -1
DATA BLOCK 0: !-------32 Bytes of Data-------!!-------32 Bytes of Data-------!

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<DISK STATE>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
DISK NAME: ssufs
INODE FREELIST:      x x x x x x x x
DATA BLOCK FREELIST: x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
shlee@shlee-virtual-machine:~/workspace/ssuos/project6$ cat ssufs_test.output
Write Data: 0
Seek: 0
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<DISK STATE>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
DISK NAME: ssufs
INODE FREELIST:      1 x x x x x x x
DATA BLOCK FREELIST: 1 x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x
INODE 0
STATUS: 1       NAME    f1.txt  SIZE    64      DATABLOCK       0 -1 -1 -1
DATA BLOCK 0: !-------32 Bytes of Data-------!!-------32 Bytes of Data-------!

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<DISK STATE>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
DISK NAME: ssufs
INODE FREELIST:      x x x x x x x x
DATA BLOCK FREELIST: x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
shlee@shlee-virtual-machine:~/workspace/ssuos/project6$
```

주어진 테스트 결과와 정확히 일치하는 것을 확인하였음.

- 2. 직접 작성한 테스트 프로그램으로 테스트

```
shlee@shlee-virtual-machine:~/workspace/ssuos/project6$ ./ssufs_test.sh
***write test***
Write Data: 0
Write Data: 0
Write Data: 0
Write Data: 0
Write Data: 0
Write Data: 0
Write Data: 0
Seek: 0
Seek: 0
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<DISK STATE>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
DISK NAME: ssufs
INODE FREELIST:      1 1 x x x x x x
DATA BLOCK FREELIST: 1 1 1 1 1 x x x x x x x x x x x x x x x x x x x x x x x x x x
INODE 0
STATUS: 1       NAME    f1.txt  SIZE    84      DATABLOCK       0 1 -1 -1
DATA BLOCK 0: !-------32 Bytes of Data-------!!-------32 Bytes of Data-------!
DATA BLOCK 1: !-20 Bytes of Data-!

INODE 1
STATUS: 1       NAME    f2.txt  SIZE    152     DATABLOCK       2 3 4 -1
DATA BLOCK 0: !-20 Bytes of Data-!!---24 Bytes of Data---!!-------32 Bytes of
DATA BLOCK 1: Data-------!!-------32 Bytes of Data-------!!-20 Bytes of Data-!
DATA BLOCK 2: !---24 Bytes of Data---!

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
***read test***
Read Data: 0
!-20 Bytes of Da
Read Data: 0
ta-!!---24 Bytes
Read Data: 0
 of Data---!!---
Read Data: 0
----32 Bytes of Data-------!!---
Seek: 0
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<DISK STATE>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
DISK NAME: ssufs
INODE FREELIST:      1 1 x x x x x x
DATA BLOCK FREELIST: 1 1 1 1 1 x x x x x x x x x x x x x x x x x x x x x x x x x x
INODE 0
STATUS: 1       NAME    f1.txt  SIZE    84      DATABLOCK       0 1 -1 -1
DATA BLOCK 0: !-------32 Bytes of Data-------!!-------32 Bytes of Data-------!
DATA BLOCK 1: !-20 Bytes of Data-!

INODE 1
STATUS: 1       NAME    f2.txt  SIZE    152     DATABLOCK       2 3 4 -1
DATA BLOCK 0: !-20 Bytes of Data-!!---24 Bytes of Data---!!-------32 Bytes of
DATA BLOCK 1: Data-------!!-------32 Bytes of Data-------!!-20 Bytes of Data-!
DATA BLOCK 2: !---24 Bytes of Data---!

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

➔ write 테스트 여러 블럭에 걸쳐서 잘 쓰여지는지 확인

➔ read 테스트

➔ read 테스트2 – 여러 블록에 걸쳐있는 데이터 잘 읽어지는 것 확인

```
***read test2***
Read Data: 0
!-20 Bytes of Data-!!---24 Bytes of Data---!!-------32 Bytes of Data-------!!-------32 Bytes of Data-------!!-20 Bytes of Data-!!---24 Bytes of Data---!
Seek: 0
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<DISK STATE>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
DISK NAME: ssufs
INODE FREELIST:      1 1 x x x x x x
DATA BLOCK FREELIST: 1 1 1 1 1 x x x x x x x x x x x x x x x x x x x x x x x x x
INODE 0
STATUS: 1       NAME    f1.txt  SIZE    84      DATABLOCK       0 1 -1 -1
DATA BLOCK 0: !-------32 Bytes of Data-------!!-------32 Bytes of Data-------!
DATA BLOCK 1: !-20 Bytes of Data-!

INODE 1
STATUS: 1       NAME    f2.txt  SIZE    152     DATABLOCK       2 3 4 -1
DATA BLOCK 0: !-20 Bytes of Data-!!---24 Bytes of Data---!!-------32 Bytes of
DATA BLOCK 1: Data-------!!-------32 Bytes of Data-------!!-20 Bytes of Data-!
DATA BLOCK 2: !---24 Bytes of Data---!

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

➔ rewrite 테스트. 이미 데이터가 있는 영역에 덮어쓰기가 잘 되는지 확인

```
***rewrite test***
Seek: 0
Write Data: 0
Seek: 0
Write Data: 0
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<DISK STATE>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
DISK NAME: ssufs
INODE FREELIST:      1 1 x x x x x x
DATA BLOCK FREELIST: 1 1 1 1 1 x x x x x x x x x x x x x x x x x x x x x x x x x
INODE 0
STATUS: 1       NAME    f1.txt  SIZE    84      DATABLOCK       0 1 -1 -1
DATA BLOCK 0: !-------32 Bytes of Data-------!!-------32 Bytes of Data-------!
DATA BLOCK 1: !-20 Bytes of Data-!

INODE 1
STATUS: 1       NAME    f2.txt  SIZE    152     DATABLOCK       2 3 4 -1
DATA BLOCK 0: !-20 Bytes of Data-!!---24 Bytes of Data---!!-------32 Bytes ohe
DATA BLOCK 1: llo world--!!------hello worldf Data-------!!-20 Bytes of Data-!
DATA BLOCK 2: !---24 Bytes of Data---!

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

➔ delete 테스트. 파일이 잘 삭제되는지 확인

```
***delete test***
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
DISK NAME: ssufs
INODE FREELIST:      x 1 x x x x x x
DATA BLOCK FREELIST: x x 1 1 1 x x x x x x x x x x x x x x x x x x x x x x x x x
INODE 1
STATUS: 1       NAME    f2.txt  SIZE    152     DATABLOCK       2 3 4 -1
DATA BLOCK 0: !-20 Bytes of Data-!!---24 Bytes of Data---!!-------32 Bytes ohe
DATA BLOCK 1: llo world--!!------hello worldf Data-------!!-20 Bytes of Data-!
DATA BLOCK 2: !---24 Bytes of Data---!

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

```
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<DISK STATE>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
DISK NAME: ssufs
INODE FREELIST:      x x x x x x x x
DATA BLOCK FREELIST: x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
shlee@shlee-virtual-machine:~/workspace/ssuos/project6$ git add --all
```

테스트 결과 정상적으로 작동됨을 확인하였음.

3. 소스코드

- **ssufs-ops.c**

```
#include "ssufs-ops.h"


extern struct filehandle_t file_handle_array[MAX_OPEN_FILES];


int ssufs_allocFileHandle() {
	for(int i = 0; i < MAX_OPEN_FILES; i++) {
		if (file_handle_array[i].inode_number == -1) {
			return i;
		}
	}
	return -1;
}


int ssufs_create(char *filename){
	/* 1 */
	struct inode_t *tmp;
	int inode_number;


	if (strlen(filename) > MAX_NAME_STRLEN) {
		return -1;
	}


	if (open_namei(filename) != -1) { // 동일한 이름의 파일이 존재하는지 확인한다.
		return -1;
	}
```

```
	if ((inode_number = ssufs_allocInode()) == -1) { // 새로운 inode를 할당한다
		return -1;
	}

	tmp = (struct inode_t *) malloc(sizeof(struct inode_t)); // 임시로 inode 내용 저장할 메모리 공간 할당
	ssufs_readInode(inode_number, tmp);
	tmp->status = INODE_IN_USE; // inode의 status 변경
	strcpy(tmp->name, filename); // 파일명 저장
	tmp->file_size = 0; // 새로 생성된 파일이므로 크기 0으로 초기화
	ssufs_writeInode(inode_number, tmp); // 새로운 inode 내용을 inode block에 저장

	return inode_number;
}

void ssufs_delete(char *filename){
	/* 2 */
	int inode_number;

	if (strlen(filename) > MAX_NAME_STRLEN) {
		return;
	}

	if ((inode_number = open_namei(filename)) == -1) { // 해당 파일의 inode 번호를 구한다
		return; // 해당 파일이 존재하지 않으면 종료한다
	}
```

```
	ssufs_freeInode(inode_number); // inode free

	return;
}

int ssufs_open(char *filename){
	/* 3 */
	int inode_number;
	int new_handle_index = -1;

	if ((inode_number = open_namei(filename)) == -1) { // 해당 파일의 inode 번호 구함
		return -1; // 파일 존재하지 않으면 -1 리턴
	}

	if ((new_handle_index = ssufs_allocFileHandle()) == -1) { // 새로운 file handle 할당
		return -1; // file handle을 할당받지 못했다면 -1 리턴
	}
	file_handle_array[new_handle_index].inode_number = inode_number; // file handle에 inode 번호 저장
	file_handle_array[new_handle_index].offset = 0; // offset 0으로 초기화

	return new_handle_index; // 새로운 file handle의 index를 리턴함
}

void ssufs_close(int file_handle){
	file_handle_array[file_handle].inode_number = -1;
	file_handle_array[file_handle].offset = 0;
}
```

```
int ssufs_read(int file_handle, char *buf, int nbytes){
	/* 4 */
	struct inode_t *tmp;
	char *block_buf;
	int file_size, offset;
	int read_bytes;
	int start_byte, end_byte;
	int start_block_index, end_block_index;

	if (file_handle_array[file_handle].inode_number == -1) { // 잘못된 file_handle 번호를 전달받았으면 -1 리턴
		return -1;
	}

	tmp = (struct inode_t *) malloc(sizeof(struct inode_t));
	ssufs_readInode(file_handle_array[file_handle].inode_number, tmp); // inode 읽어옴

	offset = file_handle_array[file_handle].offset;
	file_size = tmp->file_size;
	start_byte = offset; // 읽기 시작할 위치의 오프셋
	end_byte = offset + nbytes - 1; // 읽기 종료할 위치의 오프셋 (여기까지 읽고 종료)
	start_block_index = offset / BLOCKSIZE; // 읽기 시작할 블록 index
	end_block_index = end_byte / BLOCKSIZE; // 읽기 종료할 블록 index

	if (offset + nbytes > file_size) { // 파일 크기를 넘어서 읽으려고 하는 경우에는 아무것도 읽지 않아야함 -> -1 리턴하며 함수 종료
		free(tmp);
```

```
            return -1;
        }

        for (int i = start_block_index, read_bytes = 0; i <= end_block_index; ++i) {
            int read_start_byte, read_end_byte;
            int data_block_index = tmp->direct_blocks[i];

            block_buf = (char *)calloc(BLOCKSIZE, sizeof(char)); // 읽은 데이터 임시로 저장할 공간 할당
            ssufs_readDataBlock(data_block_index, block_buf); // 블럭 통째로 데이터 읽어옴

            // 읽어온 블럭의 데이터들 중에서 우리에게 필요한 데이터의 시작위치, 끝 위치를 구한다
            read_start_byte = 0;
            read_end_byte = BLOCKSIZE - 1;
            if (i == start_block_index) {
                read_start_byte = start_byte % BLOCKSIZE;
            }
            if (i == end_block_index) {
                read_end_byte = end_byte % BLOCKSIZE;
            }

            memcpy(buf + read_bytes, block_buf + read_start_byte, read_end_byte - read_start_byte + 1); // 데이터를 buf에 copy
            read_bytes += read_end_byte - read_start_byte + 1; // 이번 블럭에서 읽어온 데이터의 크기를 구한다

            free(block_buf);
        }
```

```
	file_handle_array[file_handle].offset = end_byte + 1; // 새로운 offset 저장


	free(tmp);
	return 0;
}


int ssufs_write(int file_handle, char *buf, int nbytes){
	/* 5 */
	struct inode_t *tmp;
	int file_size, offset;
	int write_bytes;
	int start_byte, end_byte;
	int start_block_index, end_block_index;

	if (file_handle_array[file_handle].inode_number == -1) { // 잘못된 file_handle 번호를 전달받았으면 -1 리턴
		return -1;
	}

	tmp = (struct inode_t *) malloc(sizeof(struct inode_t));
	ssufs_readInode(file_handle_array[file_handle].inode_number, tmp); // inode 읽어옴

	offset = file_handle_array[file_handle].offset;
	file_size = tmp->file_size;
	start_byte = offset; // 쓰기 시작할 위치의 오프셋
	end_byte = offset + nbytes - 1; // 쓰기 종료할 위치의 오프셋 (여기까지 쓰고 종료)
	start_block_index = offset / BLOCKSIZE; // 쓰기 시작할 블록의 index
	end_block_index = end_byte / BLOCKSIZE; // 쓰기 종료할 블록의 index
```

```
	if (end_byte > BLOCKSIZE * MAX_FILE_SIZE) { // 요청된 바이트 수를 쓰면 최대 파일 크기 제한을 초과하는 경우 -1 리턴하고 함수 종료
		free(tmp);
		return -1;
	}

	int new_alloced_data_blocks[MAX_FILE_SIZE] = {0,};
	for (int i = start_block_index; i <= end_block_index; ++i) {
		int data_block_index = tmp->direct_blocks[i];

		if (data_block_index == -1) {
			data_block_index = tmp->direct_blocks[i] = ssufs_allocDataBlock(); // 새로운 data block이 필요하면 새로 할당함
			if (data_block_index == -1) { // 할당 실패 시
				// 앞서 새로 할당했던 data block들 다시 반환하여 쓰기 하기 전 상태를 유지하도록 한다
				for (int j = 0; j < MAX_FILE_SIZE; ++j) {
					if (new_alloced_data_blocks[j]) {
						ssufs_freeDataBlock(tmp->direct_blocks[j]);
					}
				}

				return -1; // -1 리턴하며 종료

			} else { // 할당 성공 시 새로 할당된 데이터 블럭임을 표시
				new_alloced_data_blocks[i] = 1;
			}
```

```
        }
}

for (int i = start_block_index, write_bytes = 0; i <= end_block_index; ++i) {
        char *block_buf;
        int write_start_byte, write_end_byte;
        int data_block_index = tmp->direct_blocks[i];

        block_buf = (char *)calloc(BLOCKSIZE, sizeof(char)); // write할 데이터 임시로 저장할 공간 할당

        if (!new_alloced_data_blocks[i]) {
                ssufs_readDataBlock(data_block_index, block_buf);
        }

        // 데이터를 쓸 위치의 인덱스를 구한다
        write_start_byte = 0;
        write_end_byte = BLOCKSIZE - 1;
        if (i == start_block_index) {
                write_start_byte = start_byte % BLOCKSIZE;
        }
        if (i == end_block_index) {
                write_end_byte = end_byte % BLOCKSIZE;
        }

        memcpy(block_buf + write_start_byte, buf + write_bytes, write_end_byte - write_start_byte + 1); // write
        ssufs_writeDataBlock(data_block_index, block_buf); // write한 내용 디스크에 쓴다
```

```
            write_bytes += write_end_byte - write_start_byte + 1; // write한 바이트 수 계산

            free(block_buf);
        }

        if (end_byte > file_size) { // write 후에 파일의 크기가 커진 경우 file size 증가시킴
            tmp->file_size = end_byte + 1;
        }

        file_handle_array[file_handle].offset = end_byte + 1; // 새로운 offset 저장
        ssufs_writeInode(file_handle_array[file_handle].inode_number, tmp); // 변경된 idnode 내용 저장

        free(tmp);
        return 0;
}

int ssufs_lseek(int file_handle, int nseek){
        int offset = file_handle_array[file_handle].offset;

        struct inode_t *tmp = (struct inode_t *) malloc(sizeof(struct inode_t));
        ssufs_readInode(file_handle_array[file_handle].inode_number, tmp);

        int fsize = tmp->file_size;

        offset += nseek;

        if ((fsize == -1) || (offset < 0) || (offset > fsize)) {
```

```
                free(tmp);
                return -1;
        }

        file_handle_array[file_handle].offset = offset;
        free(tmp);

        return 0;
}
```

- **직접 작성한 테스트 프로그램**

```
#include "ssufs-ops.h"

char buf[BLOCKSIZE];
char buf2[BLOCKSIZE * MAX_FILE_SIZE];

int main()
{
    char str[] = "!-------32 Bytes of Data-------!!-------32 Bytes of Data-------!";
    char str2[] = "!-20 Bytes of Data-!";
    char str3[] = "!---24 Bytes of Data---!";
    ssufs_formatDisk();

    // create and open test
    ssufs_create("f1.txt");
    int fd1 = ssufs_open("f1.txt");
    ssufs_create("f2.txt");
    int fd2 = ssufs_open("f2.txt");

    // write test
    printf ("***write test***₩n");
    printf("Write Data: %d₩n", ssufs_write(fd1, str, BLOCKSIZE));
    printf("Write Data: %d₩n", ssufs_write(fd1, str2, 20));
    printf("Write Data: %d₩n", ssufs_write(fd2, str2, 20));
    printf("Write Data: %d₩n", ssufs_write(fd2, str3, 24));
    printf("Write Data: %d₩n", ssufs_write(fd2, str, BLOCKSIZE));
    printf("Write Data: %d₩n", ssufs_write(fd2, str2, 20));
```

```
printf("Write Data: %d₩n", ssufs_write(fd2, str3, 24));
printf("Seek: %d₩n", ssufs_lseek(fd1, 0));
printf("Seek: %d₩n", ssufs_lseek(fd2, -152));
ssufs_dump();

// read test
printf ("***read test***₩n");
printf("Read Data: %d₩n", ssufs_read(fd2, buf, 16));
printf("%s₩n", buf);
printf("Read Data: %d₩n", ssufs_read(fd2, buf, 16));
printf("%s₩n", buf);
printf("Read Data: %d₩n", ssufs_read(fd2, buf, 16));
printf("%s₩n", buf);
printf("Read Data: %d₩n", ssufs_read(fd2, buf, 32));
printf("%s₩n", buf);
printf("Seek: %d₩n", ssufs_lseek(fd2, -80));
ssufs_dump();

// read test2
printf ("***read test2***₩n");
printf("Read Data: %d₩n", ssufs_read(fd2, buf2, 152));
printf("%s₩n", buf2);
printf("Seek: %d₩n", ssufs_lseek(fd2, -152));
ssufs_dump();

// rewrite test
printf ("***rewrite test***₩n");
```

```
    printf("Seek: %d₩n", ssufs_lseek(fd2, 62));
    printf("Write Data: %d₩n", ssufs_write(fd2, "hello world", 11));
    printf("Seek: %d₩n", ssufs_lseek(fd2, 10));
    printf("Write Data: %d₩n", ssufs_write(fd2, "hello world", 11));
    ssufs_dump();

    // delete test
    printf ("***delete test***₩n");
    ssufs_delete("f1.txt");
    ssufs_dump();
    ssufs_delete("f2.txt");
    ssufs_dump();
}
```