

1. 개요

시스템 호출(system call)은 운영 체제의 커널이 제공하는 서비스에 대해, 응용 프로그램의 요청에 따라 커널에 접근하기 위한 인터페이스이다. 보통 C나 C++과 같은 고급 언어로 작성된 프로그램들은 직접 시스템 호출을 사용할 수 없기 때문에 고급 API를 통해 시스템 호출에 접근하게 하는 방법이다. 시스템호출에는 세가지 기능이 있다. ①사용자 모드에 있는 응용 프로그램이 커널의 기능을 사용할 수 있도록 한다. ②시스템 호출을 하면 사용자 모드에서 커널 모드로 바뀐다. ③커널에서 시스템 호출을 처리하면 커널 모드에서 사용자 모드로 돌아가 작업을 계속한다. ssuos에 있는 기존의 open시스템 콜 함수에 O_APPEND, O_TRUNC 기능 추가하고 fcntl 시스템 콜을 구현하고 F_DUPFD, F_GETFL, F_SETFL 기능을 추가하여 시스템 콜에 대한 이해를 높인다.

2.상세설계

*syscall.c

```
1)void init_syscall(void)
```

init_syscall()에 REGSYS(SYS_FCNTL, do_fcntl, 3); 를 추가하였다. 이 부분을 추가하여 fcntl()의 시스템콜 실행이 가능하게 하였다.

```
2)int fcntl(int fd, int cmd, long arg)
```

syscall.h에 선언만 되어 있고 syscall.c에 정의되어 있지 않은 fcntl()의 정의부분을 추가하였다.

*do_syscall.c

```
3)int do_fcntl(int fd, int cmd, long arg)
```

fcntl()에서 인자로 사용되는 cmd값이 F_DUPFD일 때 arg의 인자가 ssuos에서 허용되는 fd의 최대값인 NR_FILES(5)보다 크면 -1을 리턴하게 하였다. 또한 arg인자가 이미 다른 fd값으로 사용중이라면 file_cursor[arg]값이 NULL이 아님을 이용하여 이후의 fd값 중에 사용중이지 않은 가장 작은 값이 할당되도록 하였다. 그리하여 F_DUPFD 인자를 사용할 때 새로 복제된 fd를 리턴값으로 갖게 하도록 함수를 수정하였다.

fcntl()에서 인자로 사용되는 cmd값이 f_GETFL일 때는 인자로 주어지는 fd로 지정된 파일이 open할 때 지정된 상태플래그를 리턴하기위해 리턴값이 file_cursor[fd]->flags가 되도록 함수를 수정하였다.

fcntl()에서 인자로 사용되는 cmd값이 f_SETFL일 때는 |=로 추가되는 플래그가 O_APPEND만 추가가능하게 하기 위해 arg인자에 O_APPEND플래그가 있는지를 확인하여 이 플래그를 제외하고는 모두 추가되지 않도록 함수를 수정하였다. 또한 인자로주어지는 fd로 지정된 파일의 상태플래그에 인자로 주어지는 arg의 상태플래그 값을 추가한 후 그 상태플래그를 리턴값으로 가질 수 있도록 함수를 수정하였다.

*file.c

```
4)int file_open(struct inode *inode, int flags, int mode)
```

flags에 O_APPEND가 있다면 파일 안에 존재하는 문자열의 길이만큼 오프셋을 옮겨주어 기존 내용의 끝부분에 새로운 내용이 추가되도록 해야한다. 파일안에 존재하는 문자열의 길이는 파일의 크기와 동일하므로 파일의 오프셋인 pos의 값에 파일사이즈를 대입하였다. 그래서 파일의 끝부분에 추가적으로 내용이 들어갈 수 있게 하였다.

flags에 O_TRUNC가 있다면 파일의 기존 내용을 지워버리면 다시 처음부터 원하는 문자열을 써 넣을 수 있을 것이다. 따라서 파일사이즈를 0으로 만들어 기존 내용을 지워버리고 새롭게 파일내용을 덮어쓸 수 있게 하였다.

*ssulib.c

```
5)int generic_read(int fd, void *buf, size_t len)
```

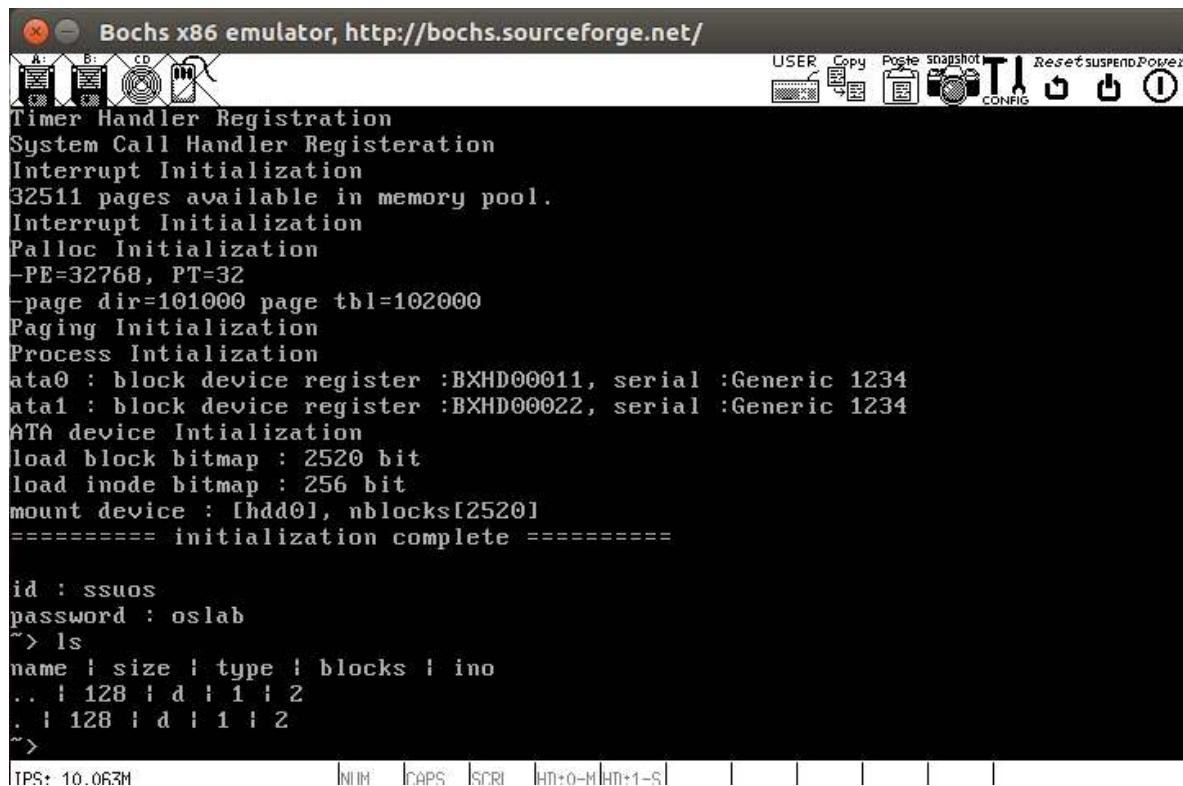
if (*pos + len > cursor->inode->sn_size) 이 부분의 가장 끝에 있는 세미콜론을 지워버려 이 조건을 만족할 때 아래의 문장이 실행될 수 있도록 하였다.

```
6)int generic_write(int fd, void *buf, size_t len)
```

함수 내부에서 file_write()가 실행되기 전에 flag를 읽어 flags에 O_APPEND가 존재한다면 O_APPEND의 플래그를 가지고 file_write()가 실행되어 파일의 제일 끝에 내용이 쓰여질 수 있도록 하였다. 그리고 flags에 O_TRUNC가 존재한다면 O_TRUNC의 플래그를 가지고 file_write()가 실행되어 기존내용을 없애버리고 파일의 처음부터 내용이 쓰여질 수 있도록 하였다.

3. 실행결과

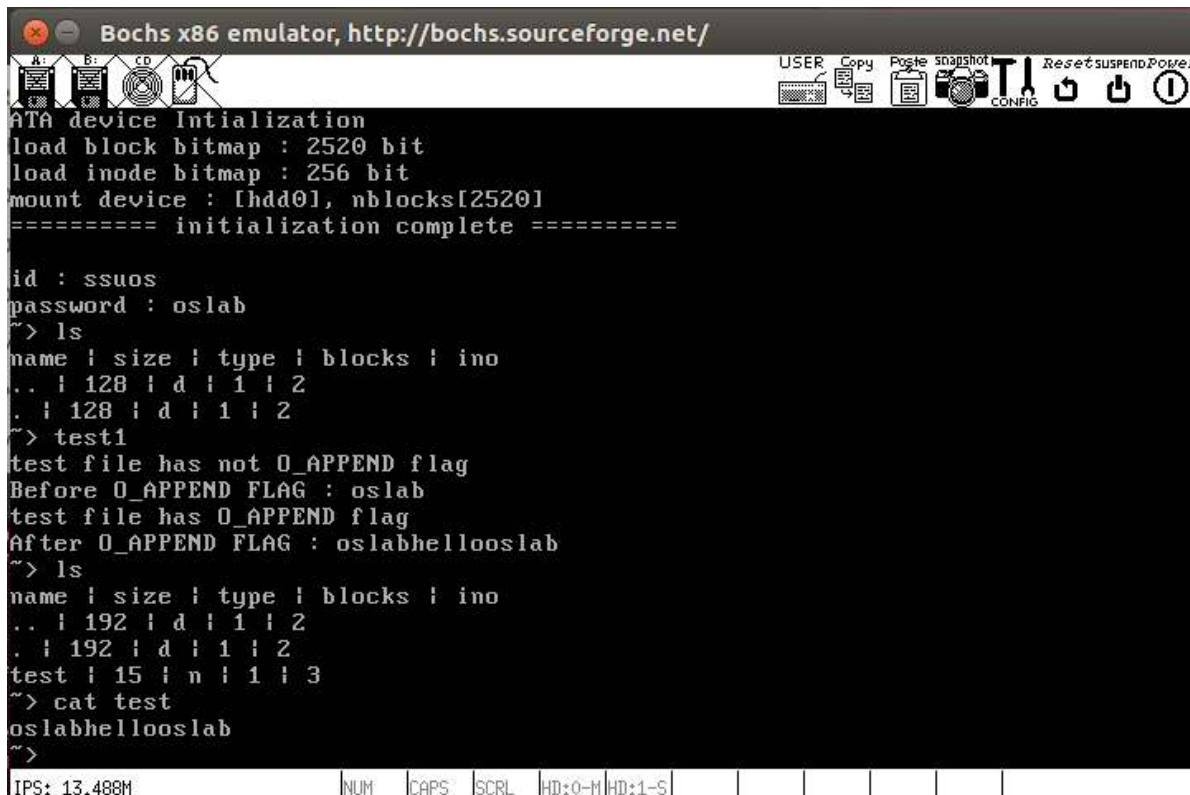
1) 처음 SSUOS 실행시 화면



```
Bochs x86 emulator, http://bochs.sourceforge.net/
Timer Handler Registration
System Call Handler Registration
Interrupt Initialization
32511 pages available in memory pool.
Interrupt Initialization
Palloc Initialization
-PE=32768, PT=32
-page dir=101000 page tbl=102000
Paging Initialization
Process Initialization
ata0 : block device register :BXHD00011, serial :Generic 1234
ata1 : block device register :BXHD00022, serial :Generic 1234
ATA device Initialization
load block bitmap : 2520 bit
load inode bitmap : 256 bit
mount device : [hdd0], nblocks[2520]
===== initialization complete =====

id : ssuos
password : oslab
~> ls
name | size | type | blocks | ino
.. | 128 | d | 1 | 2
. | 128 | d | 1 | 2
~>
```

2) test1 명령어 실행



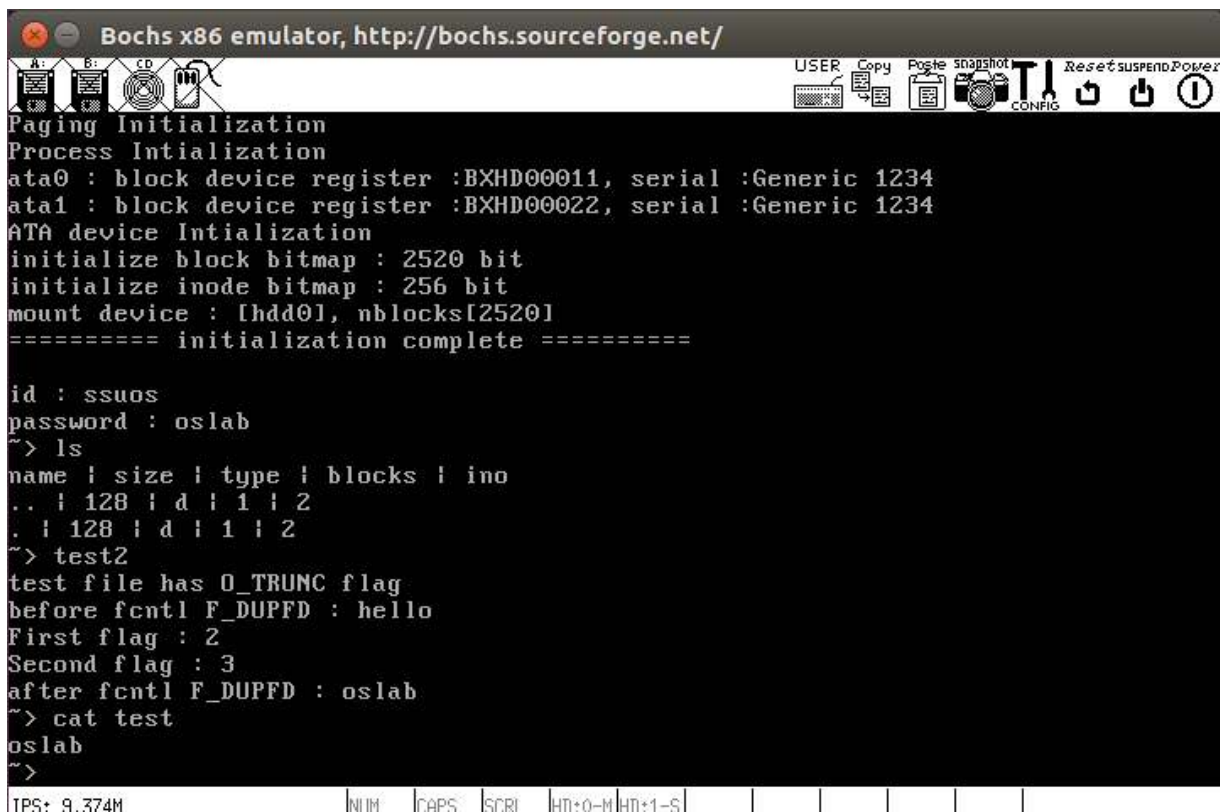
The screenshot shows the Bochs x86 emulator window with the title "Bochs x86 emulator, http://bochs.sourceforge.net/". The window has a menu bar with "USER", "Copy", "Paste", "snapshot", "CONFIG", "Reset", "SUSPEND", and "Power". The main display area shows the following text:

```
ATA device Initialization
load block bitmap : 2520 bit
load inode bitmap : 256 bit
mount device : [hdd0], nblocks[2520]
===== initialization complete =====

id : ssuos
password : oslab
~> ls
name | size | type | blocks | ino
.. | 128 | d | 1 | 2
. | 128 | d | 1 | 2
~> test1
test file has not O_APPEND flag
Before O_APPEND FLAG : oslab
test file has O_APPEND flag
After O_APPEND FLAG : oslabhellooslab
~> ls
name | size | type | blocks | ino
.. | 192 | d | 1 | 2
. | 192 | d | 1 | 2
test | 15 | n | 1 | 3
~> cat test
oslabhellooslab
~>
```

At the bottom of the window, there is a status bar showing "IPS: 13,488M" and several checkboxes for "NUM", "CAPS", "SCRL", "HD:0-M", "HD:1-S", and others.

3) test2 명령어 실행



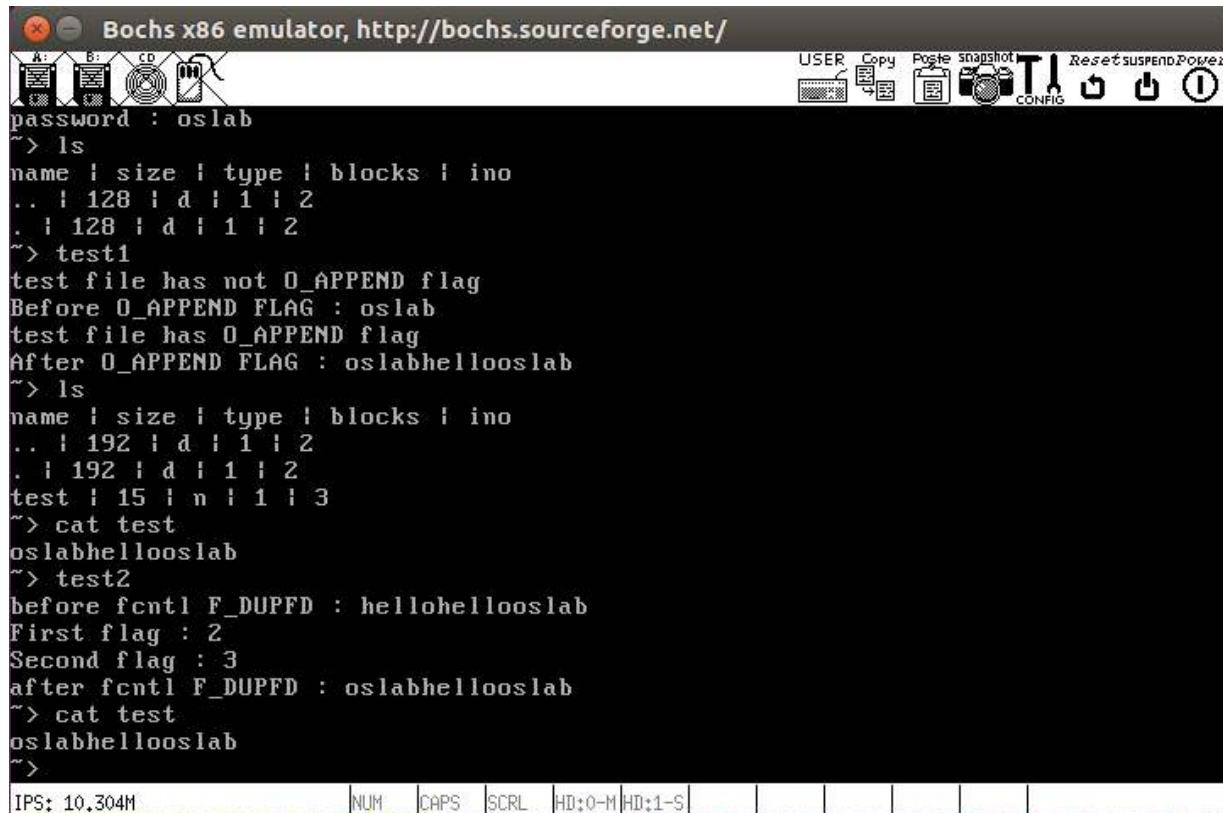
The screenshot shows the Bochs x86 emulator window with the title "Bochs x86 emulator, http://bochs.sourceforge.net/". The window has a menu bar with "USER", "Copy", "Paste", "snapshot", "CONFIG", "Reset", "SUSPEND", and "Power". The main display area shows the following text:

```
Paging Initialization
Process Initialization
ata0 : block device register :BXHD00011, serial :Generic 1234
ata1 : block device register :BXHD00022, serial :Generic 1234
ATA device Initialization
initialize block bitmap : 2520 bit
initialize inode bitmap : 256 bit
mount device : [hdd0], nblocks[2520]
===== initialization complete =====

id : ssuos
password : oslab
~> ls
name | size | type | blocks | ino
.. | 128 | d | 1 | 2
. | 128 | d | 1 | 2
~> test2
test file has O_TRUNC flag
before fcntl F_DUPFD : hello
First flag : 2
Second flag : 3
after fcntl F_DUPFD : oslab
~> cat test
oslab
~>
```

At the bottom of the window, there is a status bar showing "IPS: 9,374M" and several checkboxes for "NUM", "CAPS", "SCRL", "HD:0-M", "HD:1-S", and others.

4) test1 명령어 수행 후 test2 명령어 수행(O_TRUNC 적용 x)

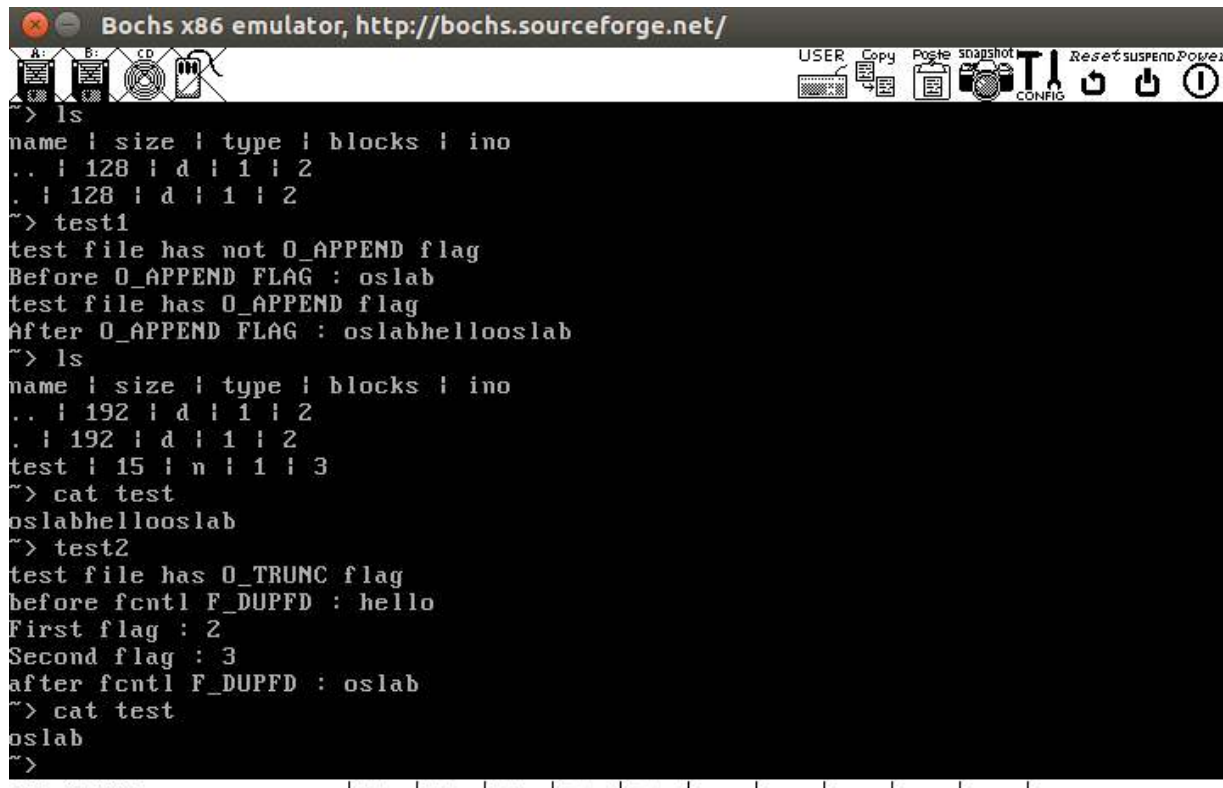


The image shows a Bochs x86 emulator window with a black terminal background. The title bar reads "Bochs x86 emulator, http://bochs.sourceforge.net/". The terminal shows the following sequence of commands and outputs:

```
password : oslab
~> ls
name | size | type | blocks | ino
.. | 128 | d | 1 | 2
. | 128 | d | 1 | 2
~> test1
test file has not O_APPEND flag
Before O_APPEND FLAG : oslab
test file has O_APPEND flag
After O_APPEND FLAG : oslabhellooslab
~> ls
name | size | type | blocks | ino
.. | 192 | d | 1 | 2
. | 192 | d | 1 | 2
test | 15 | n | 1 | 3
~> cat test
oslabhellooslab
~> test2
before fcntl F_DUPFD : hellohellooslab
First flag : 2
Second flag : 3
after fcntl F_DUPFD : oslabhellooslab
~> cat test
oslabhellooslab
~>
```

At the bottom of the window, a status bar shows "IPS: 10,304M" and several checkboxes for NUM, CAPS, SCRL, HD:0-M, HD:1-S, and others.

5) test1 명령어 수행 후 test2 명령어 수행(O_TRUNC 적용 o)



```
Bochs x86 emulator, http://bochs.sourceforge.net/
> ls
name | size | type | blocks | ino
.. | 128 | d | 1 | 2
. | 128 | d | 1 | 2
~> test1
test file has not O_APPEND flag
Before O_APPEND FLAG : oslab
test file has O_APPEND flag
After O_APPEND FLAG : oslabhellooslab
~> ls
name | size | type | blocks | ino
.. | 192 | d | 1 | 2
. | 192 | d | 1 | 2
test | 15 | n | 1 | 3
~> cat test
oslabhellooslab
~> test2
test file has O_TRUNC flag
before fcntl F_DUPFD : hello
First flag : 2
Second flag : 3
after fcntl F_DUPFD : oslab
~> cat test
oslab
~>
```

4. 소스코드

- SSUOS_P3/src/kernel/arch/syscall.c

1) void init_syscall(void)

```
{
    REGSYS(SYS_FORK, do_fork, 3);
    REGSYS(SYS_EXIT, do_exit, 1);
    REGSYS(SYS_WAIT, do_wait, 1);
    REGSYS(SYS_SSUREAD, do_ssured, 0);
    REGSYS(SYS_SHUTDOWN, do_shutdown, 0);
    REGSYS(SYS_OPEN, do_open, 2);
    REGSYS(SYS_READ, do_read, 3);
    REGSYS(SYS_WRITE, do_write, 3);
    REGSYS(SYS_FCNTL, do_fcntl, 3); //fcntl 시스템콜을 실행시키기 위해 추가하였다.
}
```

2) int fcntl(int fd, int cmd, long arg){ //선언만 되어있고 정의가 되어있지 않은 fcntl()를 정의해주었다.

return syscall3(SYS_FCNTL, fd, cmd, arg);

}

-SSUOS_P3/src/kernel/ssulib.c

3) int generic_read(int fd, void *buf, size_t len)

```
{
    struct ssufile *cursor;
```

```

uint16_t *pos =  &(cur_process->file[fd]->pos);

if( (cursor = cur_process->file[fd]) == NULL)
    return -1;

if ((~cursor->flags & O_RDONLY) && (~cursor->flags & O_RDWR)){
    return -1;
}

if (*pos + len > cursor->inode->sn_size) //이곳에 존재하던 세미콜론을 제거하였다.
    len = cursor->inode->sn_size - *pos;

file_read(cur_process->file[fd]->inode,*pos,buf,len);

return len;
}

```

4) int generic_write(int fd, void *buf, size_t len)

```

{
    struct ssufile *cursor;
    uint16_t *pos =  &(cur_process->file[fd]->pos);

    if( (cursor = cur_process->file[fd]) == NULL)
        return -1;

    if ((~cursor->flags & O_WRONLY) && (~cursor->flags & O_RDWR))
    {
        return -1;
    }

    //flag에 O_APPEND가 있을 때 파일의 사이즈만큼 오프셋을 이동하여 파일 뒤에 내용이 추가될 수 있게 한다.
    if(cur_process->file[fd]->flags & O_APPEND){
        cur_process->file[fd]->pos = cur_process->file[fd]->inode->sn_size;
    }
    //flag에 O_TRUNC가 있을 때 파일의 사이즈를 0으로 초기화시켜 파일 내용을 지우고
    //내용을 처음부터 쓸 수 있게한다.
    else if (cur_process->file[fd]->flags & O_TRUNC) {
        cur_process->file[fd]->inode->sn_size = 0;
    }

    *pos=file_write(cur_process->file[fd]->inode,*pos,buf,len);

    return len;
}

```

-SSUOS_P3/src/kernel/filesys/file.c

5) int file_open(struct inode *inode, int flags, int mode)

```
{
    int fd;
    struct ssufile **file_cursor = cur_process->file;

    for(fd = 0; fd < NR_FILEDES; fd++)
    {
        if(file_cursor[fd] == NULL)
        {
            if( (file_cursor[fd] = (struct ssufile *)palloc_get_page()) == NULL)
                return -1;
            break;
        }
    }

    inode->sn_refcount++;

    file_cursor[fd]->inode = inode;
    file_cursor[fd]->pos = 0;

    //file 오픈 시 O_APPEND라는 플래그가 존재하면 파일의 오프셋을 파일 크기만큼 지정하여
    //기존 내용 뒤에서부터 파일의 오프셋이 존재하도록 한다.
    if(flags & O_APPEND){
        file_cursor[fd]->pos = file_cursor[fd]->inode->sn_size;
    }

    //file 오픈 시 O_TRUNC라는 플래그가 존재하면 파일 크기를 0으로 초기화시켜 기존 내용을 지워버린다.
    else if(flags & O_TRUNC){
        file_cursor[fd]->inode->sn_size=0;
    }

    file_cursor[fd]->flags = flags;
    file_cursor[fd]->unused = 0;

    return fd;
}
```

-SSUOS_P3/src/kernel/arch/do_syscall.c

6) int do_fcntl(int fd, int cmd, long arg)

```
{
    int flag = -1;
    struct ssufile **file_cursor = cur_process->file;
```

```

int n_fd;
if (cmd & F_DUPFD){
    //arg 인자가 ssuos에서 허용하는 fd 범위보다 큰 경우 -1을 리턴한다.
    if(arg >= NR_FILEDES)
        return -1;
    n_fd=arg; //n_fd라는 새로운 변수에 arg값을 저장한다.

    if(file_cursor[arg] != NULL){ //arg 인자가 이미 다른 fd 값으로 사용 중인지 확인한다.
        for(int i=arg;i<NR_FILEDES;i++){
            if(file_cursor[i]==NULL){ //이후의 fd 값 중에 사용중이지 않은 가장 작은 값을 찾는다.
                n_fd=i; //찾은 값을 n_fd에 대입한다.
                break;
            }
        }
        if(n_fd>=NR_FILEDES-1) //arg 인자의 최대값이 이미 사용되고 있을 때 ssuos에서 허용하는
            //fd 범위보다 큰 경우가 되어버리므로 -1을 리턴한다.
            return -1;
    }
    file_cursor[n_fd]=file_cursor[fd]; //인자로 지정된 fd를 arg로 복사한다.
    return n_fd; //새로 복제된 fd값을 리턴한다.
}
else if (cmd & F_GETFL){
    //인자로 주어지는 fd로 지정된 파일이 오픈할 때 지정된 상태 플래그를 리턴한다.
    return file_cursor[fd]->flags;
}
else if(cmd & F_SETFL){
    //추가된 플래그가 O_APPEND가 아니라면 모두 -1을 리턴한다.
    if(!(arg & O_APPEND)){
        return -1;
    }
    //인자로 주어지는 fd로 지정된 파일의 상태플래그에 인자로 주어지는 arg 상태플래그 값을 추가한다.
    file_cursor[fd]->flags |= arg;
    //arg상태플래그 값을 추가한 파일의 상태플래그를 리턴한다.
    return file_cursor[fd]->flags;
}
else{ //그 외의 경우에는 -1을 리턴해줌으로써 예외처리를 해준다.
    return -1;
}
}

```