

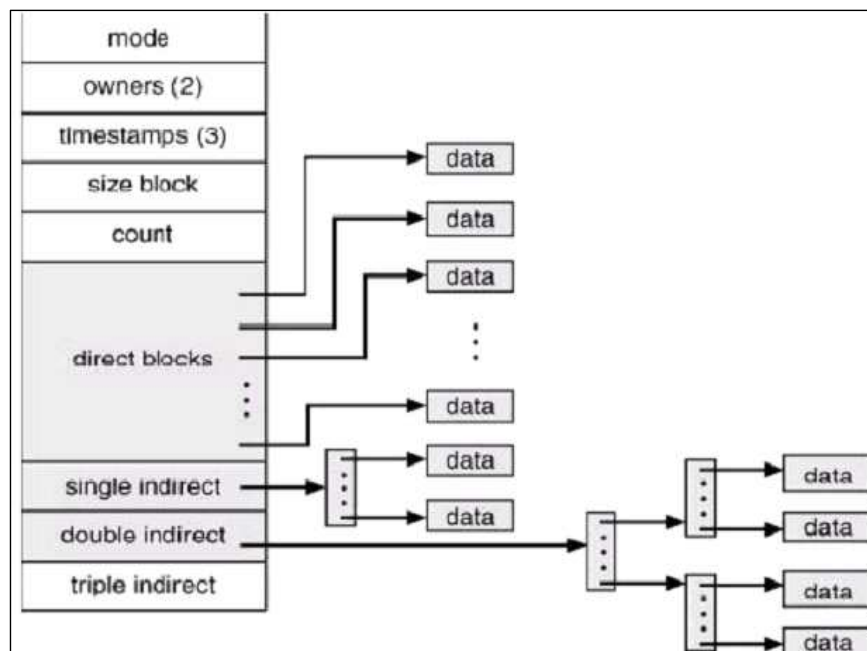
## 과제 #3 : System Call

### ○ 과제 목표

- 시스템 콜 이해
  - ✓ 기존 open 시스템 콜 함수에 O\_APPEND, O\_TRUNC 기능 추가
  - ✓ fcntl 시스템 콜 구현 및 추가

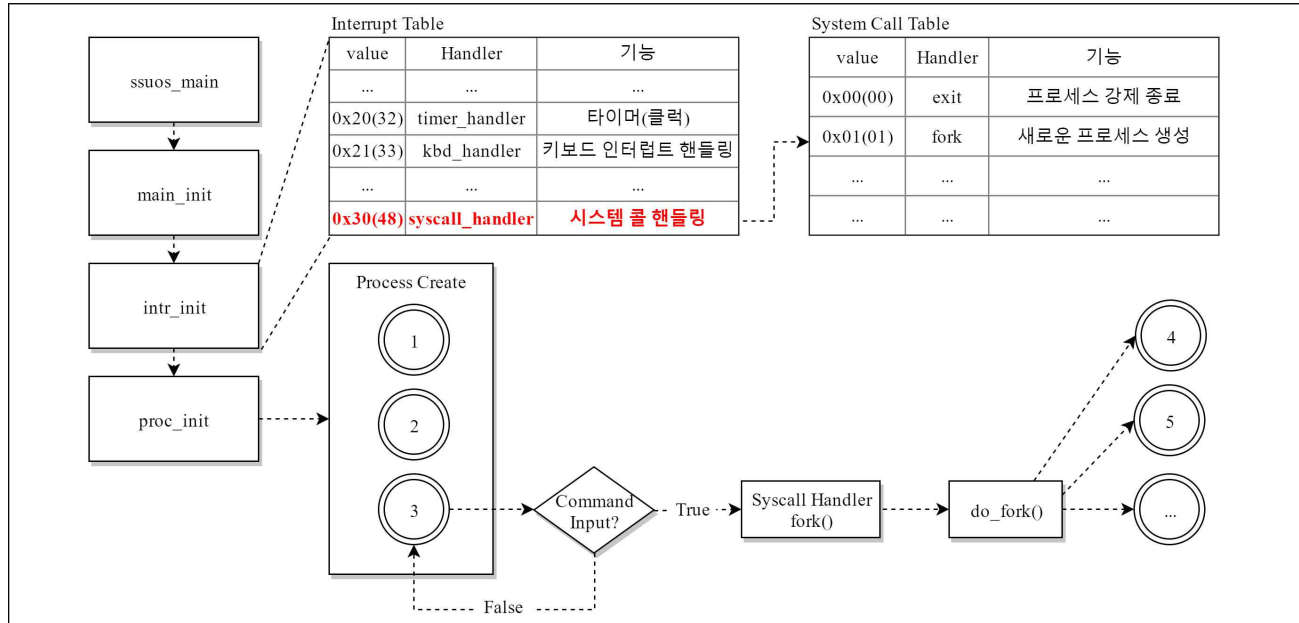
### ○ 기본 배경 지식

- 시스템 콜 함수
  - ✓ 시스템 콜은 사용자 영역이 커널 영역의 기능들을 사용할 수 있게 해주는 인터페이스
  - ✓ 크게 프로세스 제어, 파일 조작, 장치 관리, 정보 유지, 통신 유형으로 분류
- inode 정의
  - ✓ 파일을 관리하기 위한 객체로, 파일이 새로 생성되면 만들어짐
  - ✓ 파일의 모든 정보를 관리함
    - 파일에 속한 블록 위치 (index block 방법과 유사)
    - 파일 소유자 및 접근 권한
    - 파일 시간 정보
    - 파일 유형 : 커널은 정규 파일 뿐만 아니라 디렉토리, 디바이스, 파이프, 소켓 등도 파일이라는 추상화 객체로 관리
  - ✓ 디스크에 정적으로 존재 -> 메모리에 로딩
  - ✓ 일반적인 inode 구조의 예



[그림 1] inode 구조의 예시

○ 과제 구현을 위한 기본 지식



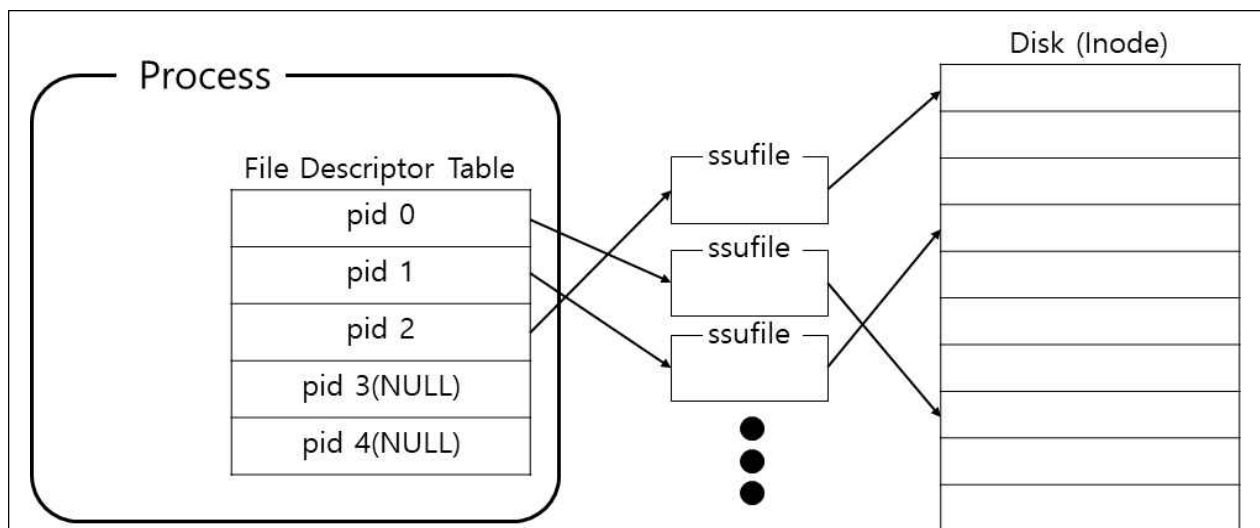
[그림 2] SSUOS에서 fork() 시스템 콜 처리 루틴

- SSUOS에서 fork() 시스템 콜 처리 루틴

- ✓ `intr_init()`에서 시스템콜을 처리하기 위한 `syscall_handler()`를 IDT에 등록
- ✓ 각 시스템콜을 처리하기 위한 시스템콜 테이블 정의
- ✓ `fork()`를 통하여 로그인 프로세스가 생성되어 로그인 함수인 `login_proc()` 호출
- ✓ 로그인 후 `shell_proc()` 호출 후 사용자 명령어 입력 시 각각 명령어에 맞는 시스템콜(`fork`) 호출
- ✓ 시스템콜 처리 함수 `syscall_handler()` 동작
- ✓ 실제 `fork()` 동작을 수행하는 `do_fork()`를 호출하여 자식 프로세스 생성

○ 과제 내용

- 현재 SSUOS는 `open()`, `read()`, `write()` 등 시스템 콜들이 구현되어 있으며 `fd`(파일 디스크립터)를 통하여 파일에 접근함



[그림 3] SSUOS에서 fd를 통한 디스크 접근

- ✓ open, read, write 시스템 콜은 fd를 통해 파일 I/O를 수행
- ✓ proc.h의 process 구조체에 fd 테이블 확인
- ✓ 파일 오픈 시 각 파일에 fd 테이블의 한 항목번호가 할당됨
- ✓ fd 테이블의 한 항목은 inode 정보를 포함하고 있는 ssufile 구조체를 가리킴
- 시스템 콜 등록하는 과정을 분석하여 기존 open() 시스템 콜에 O\_APPEND, O\_TRUNC 플래그 구현
- 시스템 콜 등록하는 과정을 분석하여 fcntl() 시스템 콜 구현

## ○ 과제 수행 방법

- (1) open() 시스템 콜 O\_APPEND, O\_TRUNC 플래그 구현
  - ✓ open() 함수의 O\_APPEND, O\_TRUNC 플래그 기능 추가
  - ✓ test1 명령어로 생성되는 test 파일의 EOF 다음에 파일 내용이 새로 추가가 된다면 O\_APPEND 플래그 구현 인정
  - ✓ test1 명령어로 생성한 test 파일을 test2 명령어로 기존 test 파일의 내용이 지워진다면 O\_TRUNC 플래그 구현 인정
  - ✓ SSUOS가 제공하는 함수 이외 불필요한 함수를 사용하였을 경우 감점처리
- (2) fcntl() 시스템 콜 구현
  - ✓ fcntl() 함수의 프로토타입
    - int fcntl(int fd, int cmd, long arg);
    - ☞ fcntl()은 이미 오픈한 파일의 속성을 가져오거나 변경할 때 사용하는 시스템 콜 함수
    - ☞ 반환 값 : cmd에 따라 다름
  - ✓ fcntl() 함수에서 인자로 사용되는 cmd
    - F\_DUPFD
      - ☞ 인자로 지정된 fd를 arg로 복사
      - ☞ 새로 복제된 fd를 리턴
      - ☞ arg 인자가 SSUOS에서 허용하는 fd 범위보다 큰 경우 -1 리턴
      - ☞ arg 인자가 이미 다른 fd 값으로 사용 중일 때에는 이후의 fd 값 중에 사용 중이지 않은 가장 작은 값을 할당
      - ☞ [그림 6] 과 같이 나오면 F\_DUPFD 구현 인정
    - ✓ SSUOS가 제공하는 함수 이외 불필요한 함수를 사용하였을 경우
      - F\_GETFL
        - ☞ 인자로 주어지는 fd로 지정된 파일이 open할 때 지정된 상태 플래그를 리턴
        - ☞ arg 인자는 무시
        - ☞ test1, test2 명령어에서 해당 fd의 상태 플래그를 올바르게 출력하면 F\_GETFL 구현 인정
      - F\_SETFL
        - ☞ 인자로 주어지는 fd로 지정된 파일의 상태 플래그에 인자로 주어지는 arg 상태 플래그 값 추가
        - ☞ (1) open() 시스템 콜에서 추가한 O\_APPEND 플래그만을 추가가능하도록 구현
        - ☞ arg 인자에서 접근 권한 플래그(O\_RDONLY, O\_WRONLY, O\_RDWR), 파일 생성 플래그(O\_CREAT, O\_EXCL, O\_NOCTTY, O\_TRUNC), 파일 상태 플래그(O\_NONBLOCK, O\_ASYNC)를 지정하면 -1을 리턴
        - ☞ test1 명령어에서 O\_APPEND 플래그가 fcntl 시스템 콜을 사용해 올바르게 추가되면 구현 인정

- (3) proc.c 에 추가된 'test1' , 'test2' 명령어로 수행 결과 출력
  - ✓ [그림 5] 은 O\_APPEND 플래그, fcntl(F\_GETFL, F\_SETFL) 구현 완료 시 출력되는 화면
  - ✓ [그림 6] 은 O\_TRUNC 플래그, fcntl(F\_GETFL, F\_DUPFD) 구현 완료 시 출력되는 화면
  - ✓ [그림 7]은 본 과제에서 요구하는 모든 사항을 구현 완료 시 출력되는 화면

○ 과제 수행 결과

```

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
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
~>
IPS: 24,404M  NUM  CAPS  SCRL  HD:0-M HD:1-S

```

[그림 4] 처음 SSUOS 실행 시 화면

Bochs x86 emulator, <http://bochs.sourceforge.net/>

ATA device Intialization  
 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  
 ~> 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  
 ~>

IPS: 25.005M    NUM    CAPS    SCRL    HD:0-M    HD:1-S

[그림 5] test1 명령어 수행

Bochs x86 emulator, <http://bochs.sourceforge.net/>

Paging Initialization  
 Process Intialization  
 ata0 : block device register :BXHD00011, serial :Generic 1234  
 ata1 : block device register :BXHD00022, serial :Generic 1234  
 ATA device Intialization  
 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  
 ~>

IPS: 23.003M    NUM    CAPS    SCRL    HD:0-M    HD:1-S

[그림 6] \*\*\* 변경 \*\*\* test2 명령어 수행

Bochs x86 emulator, <http://bochs.sourceforge.net/>

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
~>

```

IPS: 24,374M

[그림 7] \*\*\* 변경 \*\*\* test1 명령어를 수행 후 test2 명령어 수행 (O\_TRUNC 적용 X)

Bochs x86 emulator, <http://bochs.sourceforge.net/>

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
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
~>

```

IPS: 22,303M

[그림 9] \*\*\* 변경 \*\*\* test1 명령어를 수행 후 test2 명령어 수행 (O\_TRUNC 적용 O)

○ 과제 제출 마감

- 2019년 9월 22일 (일) 23시 59분 59초까지 과제 게시판으로 제출

○ 최소 기능 구현

- (1) open() 시스템 콜의 O\_APPEND, O\_TRUNC 플래그 구현
- (2) fcntl() 시스템 콜 F\_GETFL 플래그 구현
- (3) proc.c 에 추가된 'test1' , 'test2' 명령어로 수행 결과 출력

○ 배점 기준

- 보고서 10점
  - ✓ 개요 2점
  - ✓ 상세 설계 명세(기능 명세 포함) 5점
  - ✓ 실행 결과 3점
- 소스코드 90점
  - ✓ 컴파일 여부 5점(설계 요구에 따르지 않고 설계된 경우 0점 부여)
  - ✓ 실행 여부 85점
    - (1) open() 시스템 콜의 O\_APPEND 플래그 구현 15점
    - (2) open() 시스템 콜의 O\_TRUNC 플래그 구현 15점
    - (3) fcntl() 시스템 콜 F\_GETFL 플래그 구현 15점
    - (4) fcntl() 시스템 콜 F\_SETFL 플래그 구현 20점
    - (5) fcntl() 시스템 콜 F\_DUPFD 플래그 구현 20점