

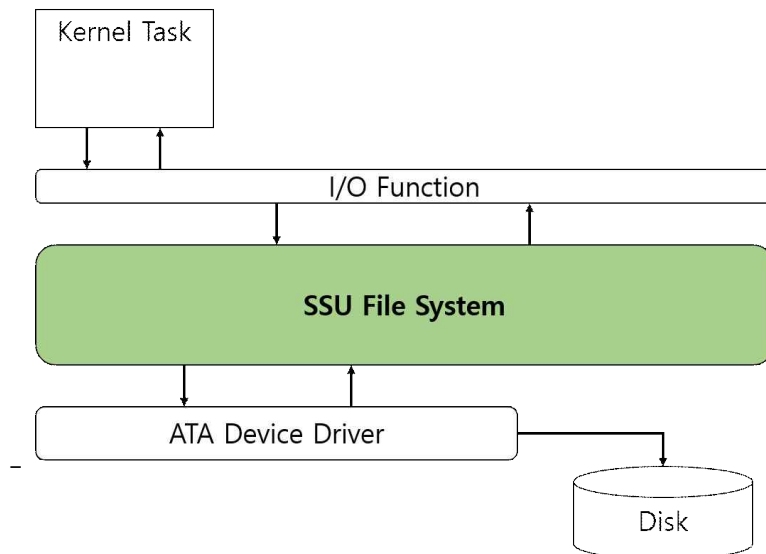
과제 #7 : File System · System Call

○ 과제 목표

- 시스템 콜 이해
- 파일 디스크립터를 이용한 lseek 시스템콜 구현 및 추가
- lseek 시스템콜 함수를 활용한 다양한 lseek 옵션 구현

○ 기본 배경 지식

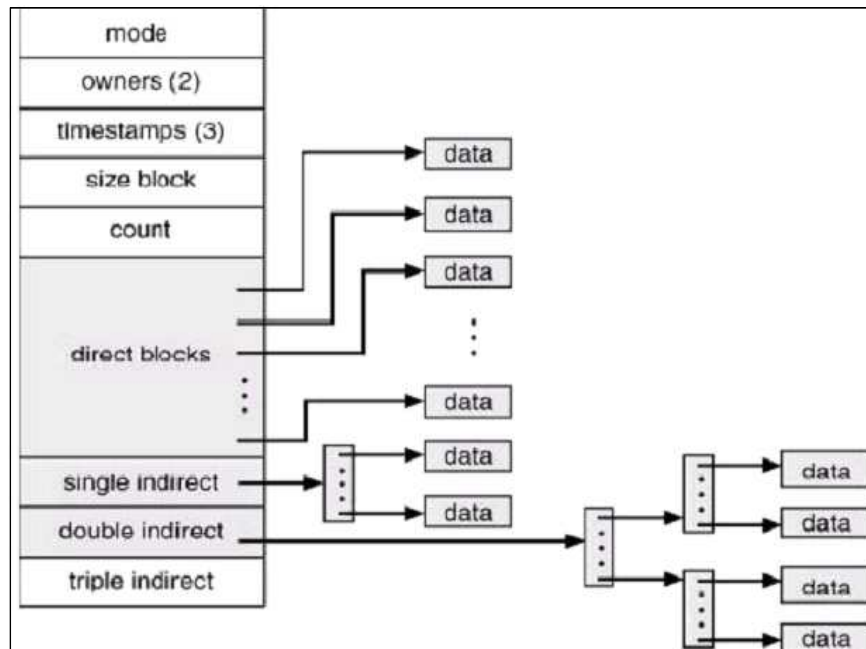
- 시스템 콜 함수
 - ✓ 시스템 콜은 사용자 영역이 커널 영역의 기능들을 사용할 수 있게 해주는 인터페이스
 - ✓ 크게 프로세스 제어, 파일 조작, 장치 관리, 정보 유지, 통신 유형으로 분류
- SSU File System



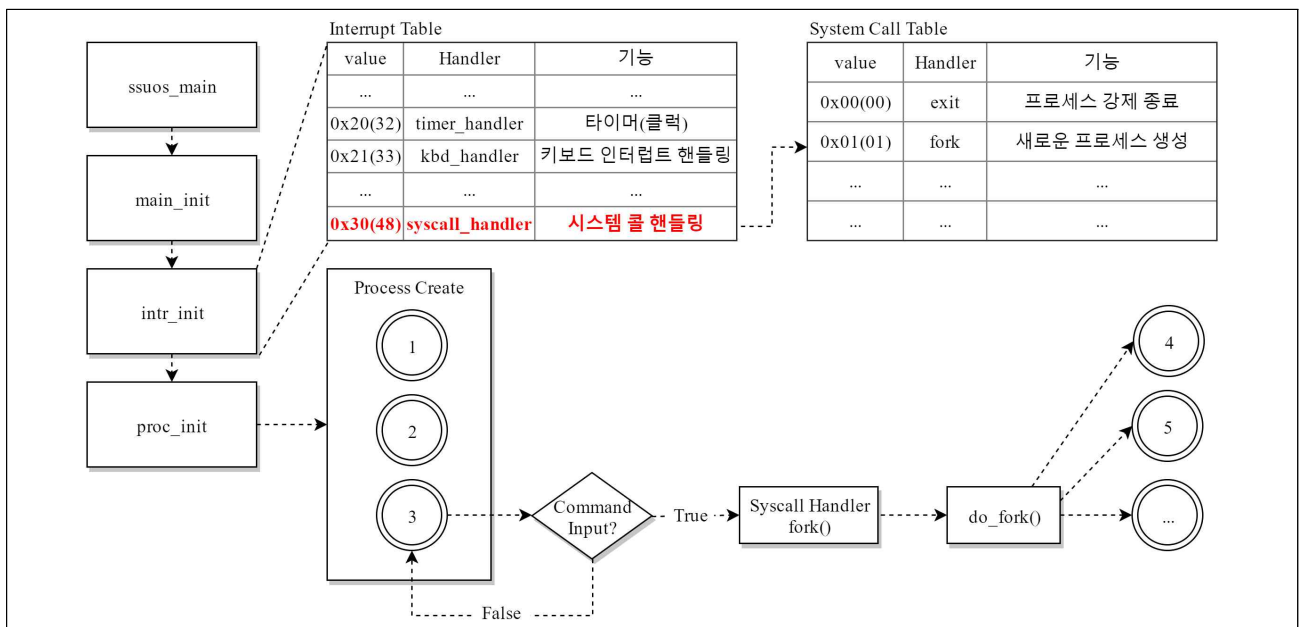
- ✓ 일반적인 파일 시스템에서는 가상파일시스템(VFS)를 통해 여러 파일시스템을 다룰 수 있음

- 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



○ 과제 구현을 위한 기본 지식

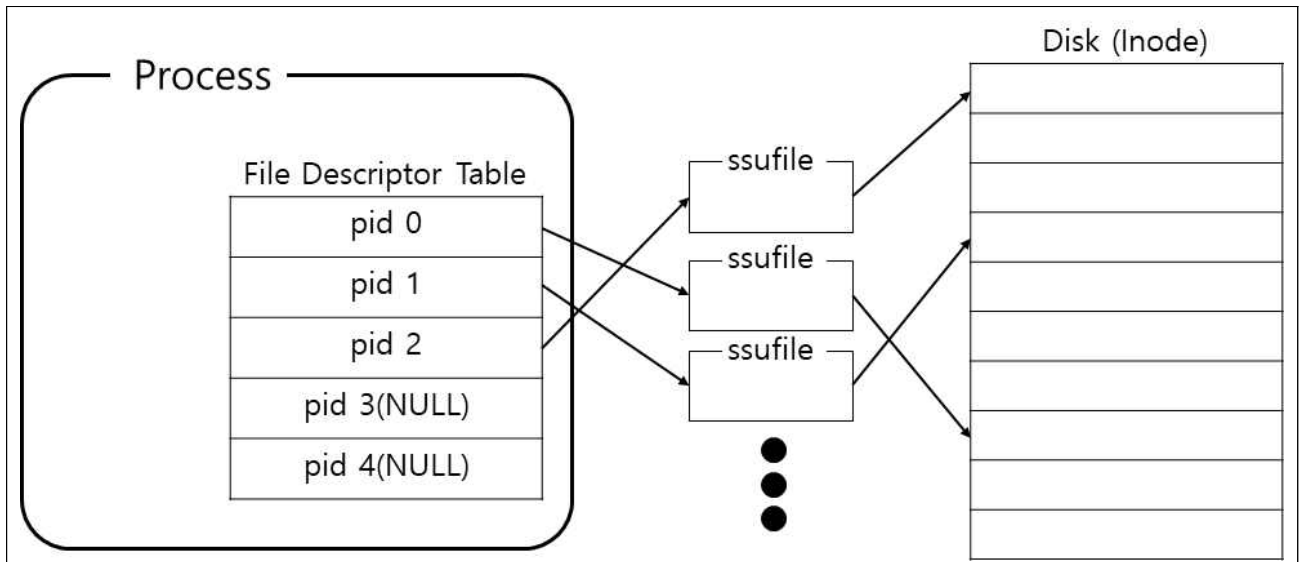


[그림1] SSUOS에서 `fork()` 시스템 콜 처리 루틴

- 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

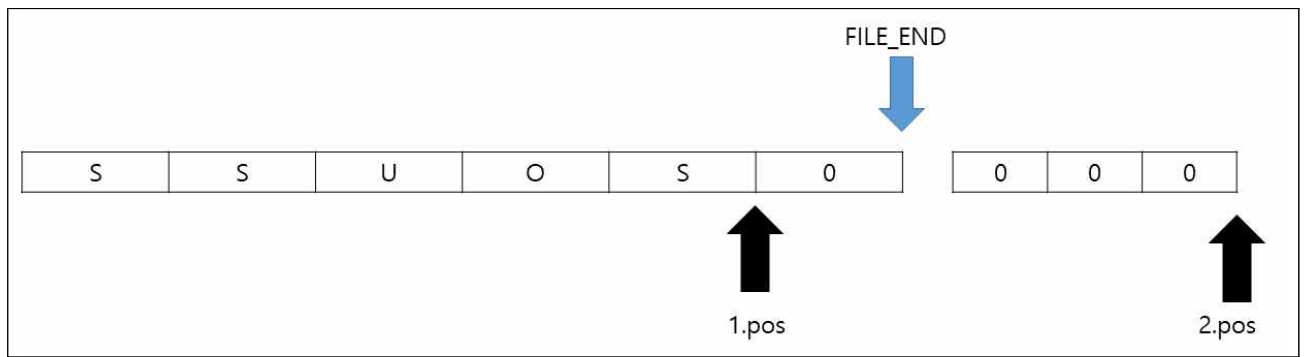
○ 과제 내용

- ✓ 시스템 콜 등록하는 과정을 분석하여 lseek() 시스템 콜 구현
- ✓ 현재 SSUOS는 open(), read(), write() 등 시스템 콜들이 구현되어 있으며 fd(파일 디스크립터)를 통하여 파일에 접근함



[그림2] SSUOS에서 fd를 통한 디스크 접근

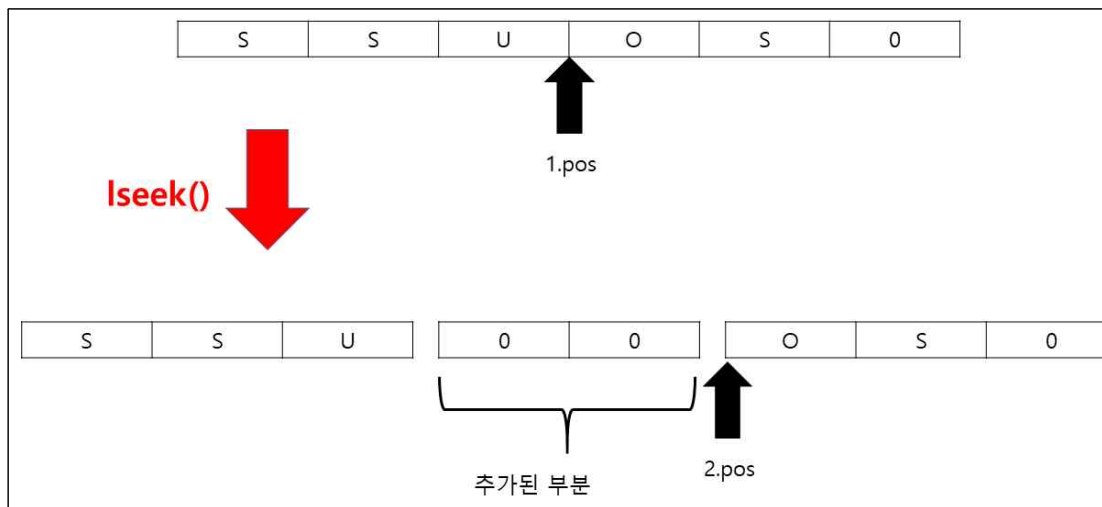
- open, read, write 시스템 콜은 fd를 통해 파일 I/O를 수행
- proc.h의 process 구조체에 fd 테이블 확인
- 파일 오픈 시 각 파일에 fd 테이블의 한 항목번호가 할당됨
- fd 테이블의 한 항목은 inode 정보를 포함하고 있는 ssufile 구조체를 가리킴
- int lseek(int fd, int offset, int whence);
 - ✓ lseek() 는 오픈 fd로부터 인자로 주어진 offset 만큼 파일 포인터 위치를 변경함
 - ✓ 반환 값: 성공했을 경우 파일의 시작으로부터 떨어진 byte만큼의 offset을 리턴하며 실패했을 경우 -1을 리턴
 - ✓ 예1) 파일의 마지막을 초과하여 lseek() 으로 인자로 주어진 파일(fd)의 EOF를 넘어 파일 포인터 위치를 사용했을 경우, lseek에서 -1을 리턴. 단, 아래 옵션처리 시, 정상 종료
 - ✓ 예2) 파일의 처음을 넘어서 lseek()으로 인자로 주어진 파일(fd)의 파일 포인터 위치를 사용했을 경우, lseek에서 -1을 리턴. 단, 아래 옵션처리 시, 정상 종료
- whence의 옵션
 - ☞ SEEK_SET: 파일의 처음을 기준
 - ☞ SEEK_CUR: 파일의 현재위치를 기준
 - ☞ SEEK_END: 파일의 마지막을 기준
- 추가 옵션
 - ✓ '-[e / a / re / c]'
 - 'e'



[그림3] FILE의 끝에서 확장하는 기능의 e 옵션

- ✓ e 옵션은 인자로 주어진 offset이 파일(fd)의 EOF를 넘을 경우 에러 처리를 하지 않고 파일의 원래 크기보다 초과된 부분만큼 파일을 확장
- 예. [그림 3]에서 현재 파일 포인터 위치가 X라고 가정할 때 e옵션으로 `lseek(fd, 4, SEEK_CUR)` 함수를 사용하면 Y까지 파일의 크기가 확장되며 추가적으로 확장된 공간은 0으로 채워짐

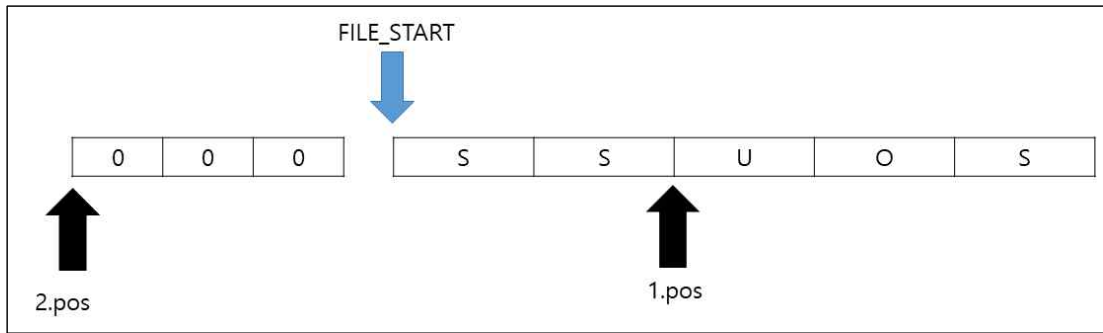
- 'a'



[그림4] FILE의 편입 기능의 a 옵션

- ✓ a 옵션은 인자로 주어진 offset이 사용된 `lseek(fd, 2, SEEK_CUR)`을 사용하면 현재 파일 포인터 위치부터 offset만큼 뒤로 이동함
- 예. [그림 4]에서 현재 파일 포인터 위치가 X라고 가정할 때 a 옵션으로 `lseek(fd, 2, SEEK_CUR)` 함수를 사용하면 Y까지 파일의 크기가 확장되며 추가적으로 확장된 공간은 0으로 채워짐

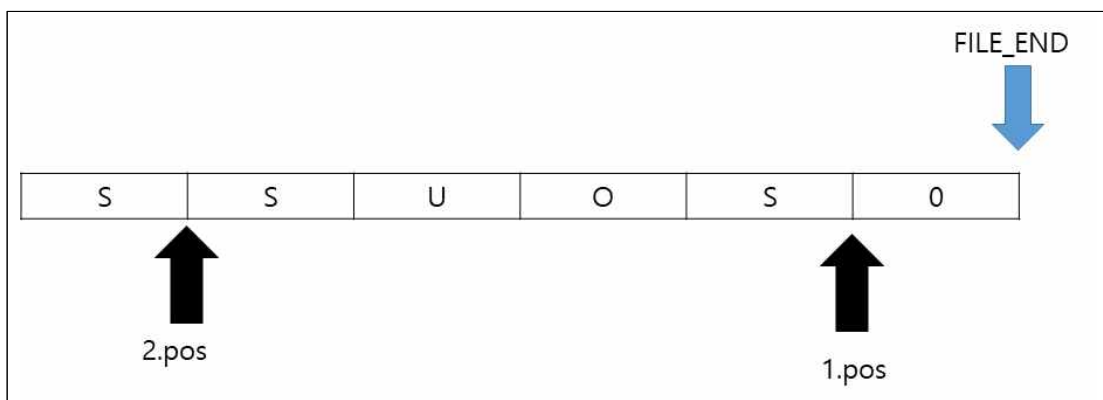
- 're'



[그림5] FILE의 시작점에서 확장하는 기능의 re 옵션

- ✓ re 옵션은 인자로 주어진 offset이 파일(fd)의 시작 지점을 넘을 경우 에러 처리를 하지 않고 파일을 원래 크기보다 초과된 부분만큼 파일을 확장
- 예. [그림 5]에서 현재 파일 포인터 위치가 X라고 가정할 때 re 옵션으로 lseek(fd, -5, SEEK_CUR) 함수를 사용하면 Y까지 파일의 크기가 확장되며 추가적으로 확장된 공간은 0으로 채워짐

- 'c'



[그림6] FILE의 범위를 초과했을 경우 offset이 순환하는 기능의 c 옵션

- ✓ c 옵션은 인자로 주어진 offset이 파일(fd)의 시작범위 혹은 EOF를 넘을 경우 에러 처리를 하지 않고 순환하여 초과된 부분만큼 파일 포인터 위치를 이동함
- 예1. [그림 6]에서 현재의 파일 포인터 위치가 X라고 가정할 때, c 옵션으로 lseek(fd, 2, SEEK_CUR) 함수를 사용하면 파일 포인터 위치는 파일을 벗어난 범위만큼 순환하여 Y가 됨
- 예2. [그림 6]에서 현재의 파일 포인터 위치가 Y라고 가정할 때, c 옵션으로 lseek(fd, -2, SEEK_CUR) 함수를 사용하면 파일 포인터 위치는 파일을 벗어난 범위만큼 순환하여 X가 됨

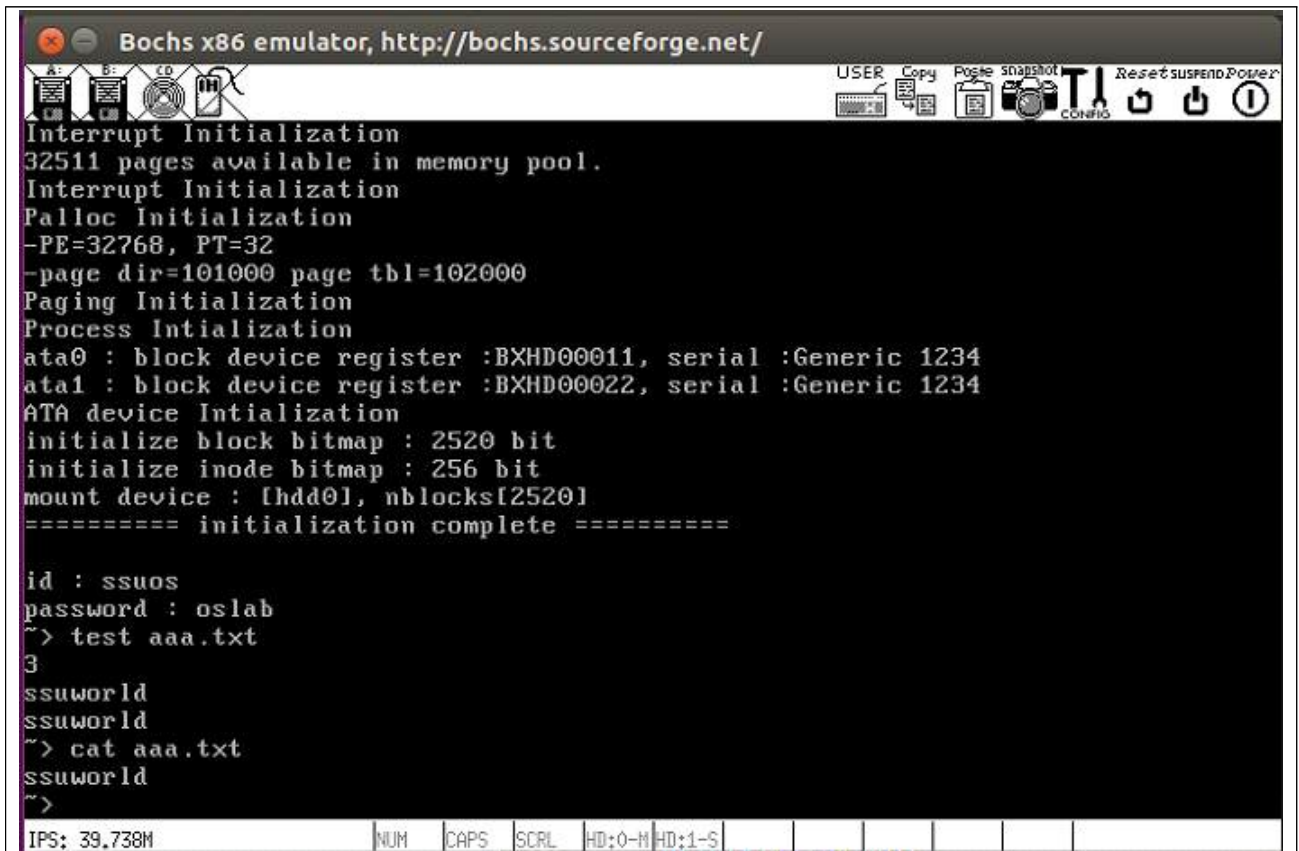
○ 주의 사항

- ✓ lseek() 시스템 콜의 프로토타입 수정 불가
- ✓ 프로세스의 가상메모리 크기가 한정되어 있음 -> 함수 내에 tmp[4096] 같이 큰 배열을 잡을 경우 스택오버플로우가 발생할 수 있음
- 단, 허용 가능한 배열의 크기는 프로세스의 코드 및 데이터 크기에 종속적임
- SSUOS에서는 tmpblock[4096]을 전역변수로 선언함

○ 과제 수행 방법

- (1) lseek() 시스템 콜 구현
 - ✓ 과제 내용에 나와있는 시스템 콜 처리루틴 및 open(), read(), write() 함수의 시스템콜 추가된 것을 분석하여 lseek() 함수를 시스템 콜에 추가
 - ✓ lseek() 함수 구현 후 [그림7] 과 같이 나오면 lseek() 기본 구현 인정
 - ✓ 시스템 콜 추가가 이루어지지 않거나, SSUOS가 제공하는 라이브러리 외 불필요한 함수를 사용하였을 경우 감점처리
 - ✓ lseek() 함수의 프로토타입은 기본 명세에 따름
 - ✓ lseek() 함수 옵션 추가의 경우만 프로토타입의 인자를 추가하는 방식으로 변경 가능
 - ✓ ssulib.c의 generic 함수의 구현도 인정하며, 기본적으로 do_syscall을 통한 구현을 원칙으로 함
- (2) 옵션 구현
 - ✓ [e / re / a / c] 옵션 구현
 - ✓ 각 옵션당 배점기준의 소스코드 항목에 10%씩 부여함
 - ✓ 옵션이 정확하게 동작하는지를 보이기 위해 process 혹은 명령어를 만들어 시나리오를 구현해야 함
 - ✓ 해당 옵션에 대해서 구현하였을 경우 구현사항 및 시나리오를 보고서에 기재해야함
 - ✓ 채점 시, 코드를 수정해야하거나 보고서에 기재하지 않을 경우 채점을 진행하지 않음
- (3) proc.c 에 추가된 'test' 명령어로 수행 결과 출력

○ 과제 수행 결과



```
Bochs x86 emulator, http://bochs.sourceforge.net/
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], nbblocks[2520]
===== initialization complete =====

id : ssuos
password : oslab
~> test aaa.txt
3
ssuworld
ssuworld
~> cat aaa.txt
ssuworld
~>

IPS: 39.738M  NUM  CAPS  SCRL  HD:0-M  HD:1-S
```

[그림 7] 기본 과제 수행 후 실행 모습

○ 과제 제출 마감

- 2018년 11월 29일 (목) 23시 59분 59초까지 과제 게시판으로 제출

○ 배점 기준

- 보고서 15%
 - ✓ 개요 2%
 - ✓ 상세 설계 명세(기능 명세 포함) 10%
 - ✓ 실행 결과 3%
- 소스코드 85%
 - ✓ 컴파일 여부 5%(설계 요구에 따르지 않고 설계된 경우 0점 부여)
 - ✓ 실행 여부 80% ((1) 기본 구현 10%, (2) 옵션 구현 70%)

○ 최소 기능 구현

- (1) lseek() 시스템 콜 구현
- (3) 구현된 (1)lseek() 시스템 콜을 proc.c 에 추가된 'test' 명령어로 수행 결과 출력