**Pintos Project 2: User Program (2)**

담당 교수 / 분반 : 박성용 교수님 / 3반

이름 / 학번 : 남유정 / 20191300

개발 기간 : 11/5~11/13

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

Project 1에서 구현하지 않은 나머지 system call을 구현한다. file descriptor을 포함한 구조체를 이용하여 create, remove, open, close, filesize, seek, tell을 구현하고, stdin, stdout이 아닌 read와 write를 구현한다. 이 때, 동기화를 이뤄 critical section에서 문제가 일어나지 않도록 한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

Project 1의 경우, filesystem을 쓰지 않는 시스템 콜을 구현하였다. 이번에는 시스템 콜의 filesystem이 포함된 나머지 부분을 구현을 한다.

* + 1. File descriptor

프로세스가 수행될 때, 어떠한 파일에 접근하기 위해 file descriptor을 사용한다. 그러므로 프로세스는 개별적인 file descriptor을 갖고 있어야 하며, 이를 이용해 스레드가 파일에 접근한다.

* + 1. System calls

Project1과 마찬가지로 user address인지 확인한 뒤, 스택에서 argument를 가져와 시스템 콜에 맞는 적절한 함수를 수행한다. 또한 이때, 유효한 주소인지 확인하는 과정을 거친다.

* + 1. Synchronization in file system

파일을 read, write 할 때, race condition이 발생하는 경우 동기화가 필요하다. 또한, 부모 스레드와 자식 스레드 간의 실행, 종료 순서를 맞추기 위해, 동기화가 필요하다. d를 위해 프로세스가 실행, 종료될 때, 부모 프로세스와 자식 프로세스의 status를 확인한다.

* 1. **개발 내용**
     1. File descriptor

File descriptor는 정해진 크기를 가진 static array로 구현하였다. 이는 인덱싱에 용이하기 때문이다. 프로세스는 각자 개별적인 file descriptor을 갖고 있어야 하므로, thread.h의 struct thread 내에 이러한 member을 선언하였다. 이 크기는 핀토스 매뉴얼의 프로세스 당 open file의 수에 맞게 128로 정하였다.

* + 1. System calls
       1. syscall\_create: 입력된 file이라는 이름을 갖고, initial\_size라는 크기를 가진 파일을 생성한다. 생성에 성공하면 true, 실패하면 false를 리턴한다.
       2. syscall\_remove: 입력된 file이라는 이름을 가진 파일을 삭제한다. 삭제에 성공하면 true, 실패하면 false를 리턴한다.
       3. syscall\_open: 입력된 file이라는 이름을 가진 파일을 연다. 연 파일의 file descriptor를 리턴하고, 여는 데 실패하면 -1을 리턴한다. 하나의 파일이 여러 번 열리면, 각 경우는 계속 새로운 file descriptor을 리턴한다.
       4. syscall\_close: 주어진 file descriptor fd를 닫는다.
       5. syscall\_filesize: 입력된 fd로 열리는 파일의 크기의 byte를 리턴한다.
       6. syscall\_read: 입력된 fd로 열리는 파일을 입력된 size의 byte수 만큼 buffer를 이용해 읽는다. 실제로 읽은 byte 수를 리턴하고, 읽기에 실패하면 -1을 리턴한다.
       7. syscall\_write: 입력된 fd로 열리는 파일에 입력된 size의 byte 수만큼 buffer를 이용해 쓴다. 실제로 쓴 byte 수를 리턴하고, 쓰기에 실패하면 0을 리턴한다.
       8. syscall\_seek: 입력된 fd로 열리는 파일에 다음에 읽거나 쓸 위치를 position으로 만든다. position=0이면 맨 처음을 읽거나 쓰도록 설정하게 된다.
       9. syscall\_tell: 입력된 fd로 열리는 파일에서 읽거나 쓰는 위치를 리턴한다.
    2. Synchronization in filesystem

Filesystem에서 critical section은 같은 파일을 동시에 읽고 쓰거나, 지우는 경우이기 때문에, 이러한 상황을 막기 위해 filesys\_lock이라는 struct lock을 선언하였다. 파일의 내용을 바꾸기 전에 lock\_acquire(&filesys\_lock)을 해주고, 내용 modify가 끝난 후 lock\_release(&filesys\_lock)을 해 주어 read, write, open 명령어에서의 mutual exclusion을 구현하였다.

3. Synchronization in Filesystem: Lock, Semaphore를 어떻게 이용할 수 있는지 각각에 대해 설명 (다른 방법을 서술해도 되지만 lock과 semaphore는 반드시 포함해야 함)

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

11/4 ~ 11/9 : Project 1의 fail 부분들 구현

11/10~11: system call 추가 구현

11/12~14: 디버깅, 보고서 작성

* 1. **개발 방법**
* II. B.의 개발 내용을 구현하기 위해 각각에 대해 다음 사항들을 포함하여 설명
  + 수정해야하는 소스코드

syscall.c, syscall.h에 추가적인 시스템 콜을 구현한다. 이 때, 각 시스템 콜의 address가 커널 영역을 침범하지 않는지 확인하는 과정이 포함되어야 한다.

process.c에서 filesystem을 추가하여 tid\_t process\_execute와 bool load 함수를 추가 구현해야 한다. 또한 semaphore의 up과 down을 추가하여 critical section이 발생하지 않도록 해야 한다.

* + 수정하거나 추가해야 하는 자료구조

thread.h의 struct thread에 file descriptor을 정적 배열인 멤버로 선언한다. 그리고, struct semaphore load\_sema는 자식 프로세스가 실행 중인 경우, 부모 프로세스가 terminate 되지 않고 기다리도록 하는 semaphore이다. 또한, 자식의 종료가 잘 이루어졌는지 알아보기 위해 bool load\_status 멤버를 선언한다.

syscall.c에 synchronization을 위한 struct lock filesys\_lock을 선언한다.

* + 수정하거나 추가해야 하는 함수
    - 1. syscall.c

syscall\_handler(struct intr\_frame \*f UNUSED)

switch 문을 연장하여 각 system call number에 따라 case를 나눠 유효한 주소값인지 확인하고, 유효하다면 함수를 호출하도록 한다.

syscall\_create, syscall\_remove …

주어져 있는 system call api를 이용해 각 시스템 콜이 호출하는 함수를 구현한다.

syscall\_init(void)에 lock\_init(&filesys\_lock)을 추가해 시스템 콜에서 lock을 사용할 수 있도록 초기화한다.

read, write 문에 stdin, stdout의 경우 외의 파일 입출력을 filesys 함수를 이용해 구현한다.

* + - 1. thread.h

file descriptor, load\_sema, load\_status 등의 추가 구현 멤버를 struct thread에포함한다.

* + - 1. thread.c

init\_thread()함수에서 추가 선언한 멤버들의 초기화를 진행한다.

* + - 1. process.c

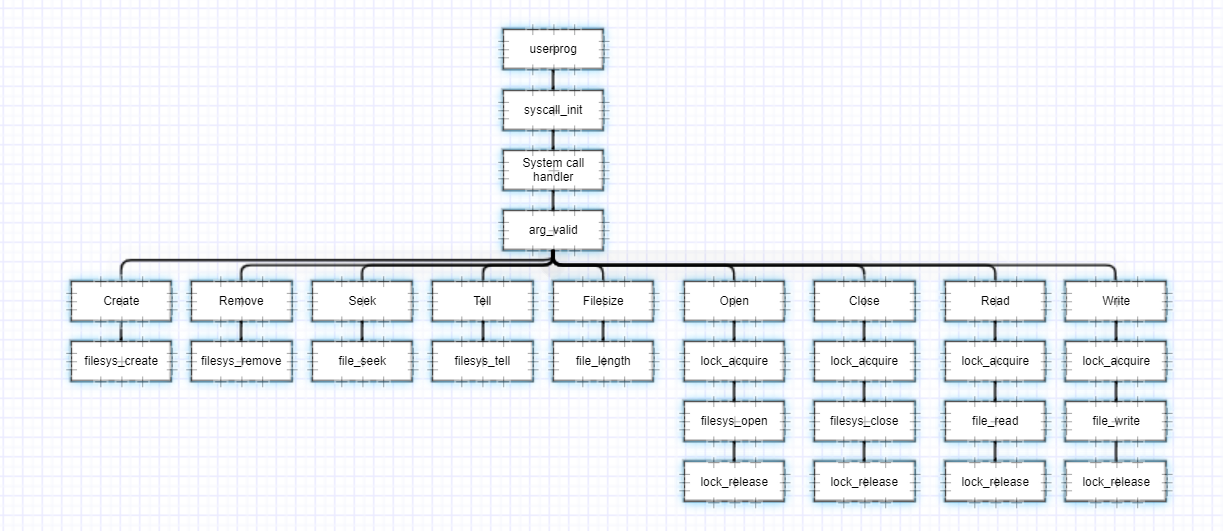
load\_sema, load\_status를 이용해 부모 프로세스와 자식 프로세스 간의

1. **연구 결과**
   1. **Flow Chart**
      1. file descriptor

텍스트, 창문, 실내, 건물이(가) 표시된 사진

자동 생성된 설명

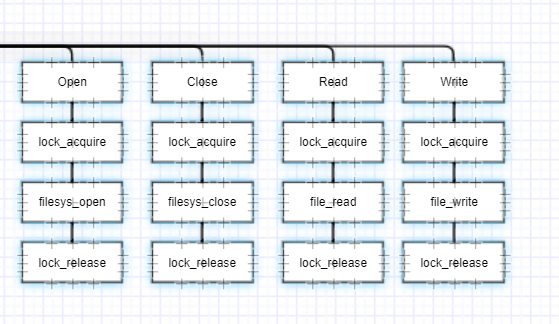
* + 1. system calls



* + 1. synchronization in file system

**텍스트, 음악, 피아노이(가) 표시된 사진

자동 생성된 설명**

**ㅌ`**

* 1. **제작 내용**

II. B. 개발 내용의 각 3가지 항목에 대하여 실직적으로 구현한 코드의 관점에서 작성 (구현 내용, 알고리즘 등을 명확히 서술할 것)

1. file descriptor

threads/thread.h의 struct thread에, struct file\* fd[128]인 file 구조체 정적 배열을 선언해 file descriptor을 구현하였다. 그리고, threads/thread.c의 init\_thread함수에서 for문을 이용해 128개의 요소를 모두 0으로 초기화하였다.

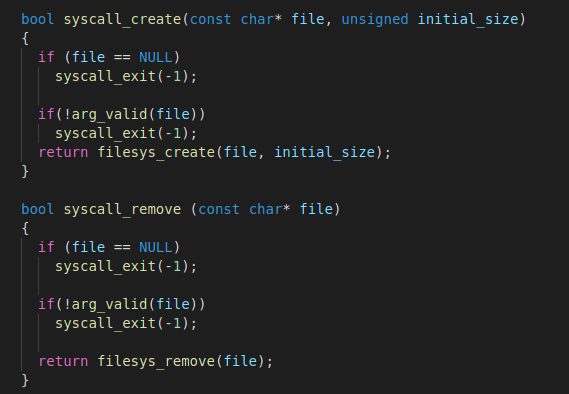
userprog/syscall.c에서 file구조체를 이용하기 위해 file.c에 선언되어 있는 구조체를 복사해 위에 선언하였다. 그리고 요소인 off\_t pos 정의를 위해 off\_t.h를 include해 주었다.

1. system call

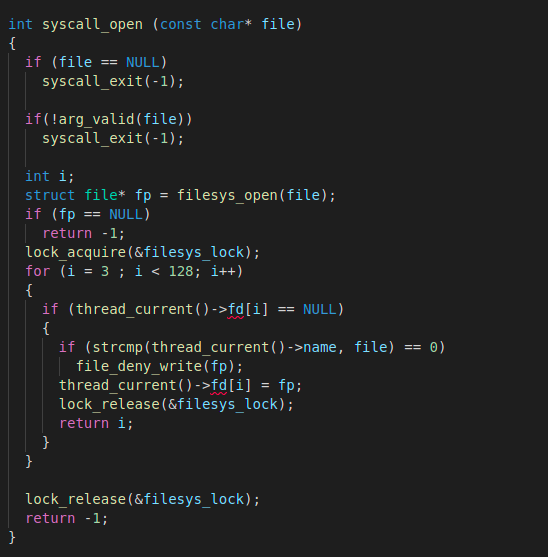
userprog/syscall.h에 추가로 구현한 system call 함수 syscall\_create, syscall\_remove, …, syscall\_close 함수들의 prototype을 추가하였다.

텍스트이(가) 표시된 사진

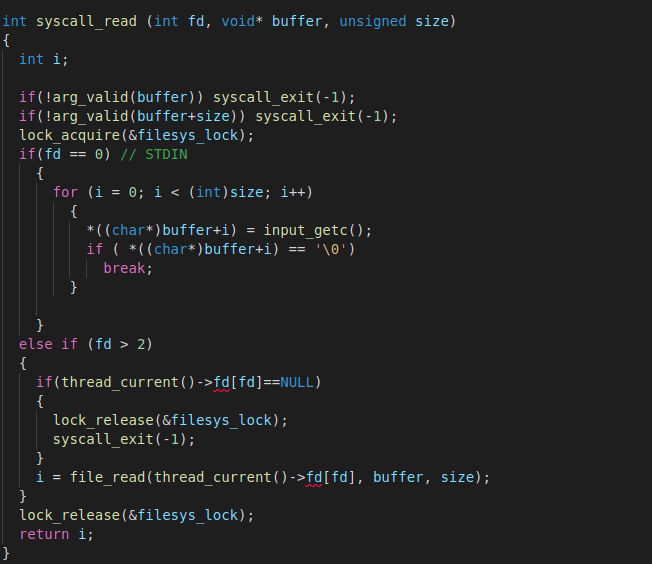
자동 생성된 설명userprog/syscall.c에는 syscall\_handler 함수 내의 switch문을 연장하여 명령어를 구분하였다. create, remove, open, filesize, seek, tell, close를 추가하여 함수를 호출하도록 하였다. 리턴값이 있는 함수의 경우, f->eax에 넘겨주었다. 각 argument는 esp의 스택에서 꺼내오며, 각 argument가 모두 커널을 침범하거나 비어 있지 않은지 확인하는 과정을 거친다.



syscall\_create의 경우, filesys/filesys.c의 filesys\_create함수를 호출하도록 한다. 이때 비어 있는 파일 혹은 없는 파일을 호출하는지 arg\_valid함수로 확인한다. 이 함수는 입력된 이름과 입력된 크기의 파일을 생성하는 함수이다. sydcall\_remove의 경우, filesys\_remove 함수를 호출한다. 마찬가지로 비어 있는 파일이거나 적절하지 않은 위치인지 확인한다. 이 함수는 입력된 이름을 가진 파일을 제거하는 함수이다.



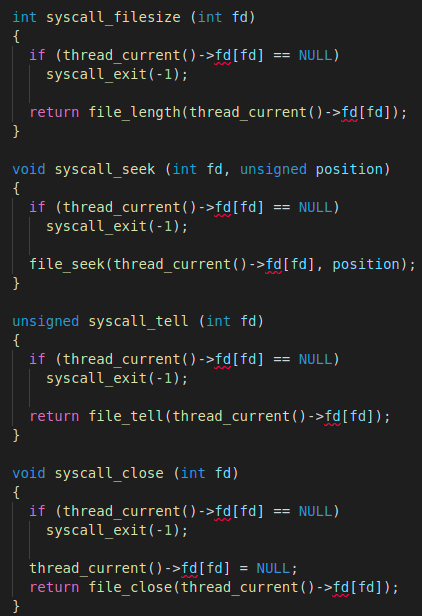
syscall\_open은 filesys/filesys.c의 filesys\_open함수를 이용하며, 입력된 이름의 파일을 여는 함수이다. 마찬가지로 file이 null이거나 적절한 주소에 있는지 확인한다. 파일을 여는 데 성공한다면 해당 파일을 리턴하고, 그렇지 않다면 NULL을 리턴하므로, syscall\_open의 리턴값도 실패한 경우에는 -1을 리턴하도록 한다. NULL이 아닌 fd를 for문을 돌며 찾고, 열린 파일을 해당 fd에 저장하고 fd값을 리턴하도록 한다.



stdin인 fd==0인 경우를 제외하고, 나머지 fd>2인 경우에 대해 추가 구현을 하였다. filesys/file.c에 구현된 file\_read 함수를 이용했다. buffer의 처음과 끝의 주소값이 유효한지 확인했으며, fd가 NULL인 경우, read가 불가능하므로 syscall\_exit(-1)을 호출했다. 읽는 데 성공한다면, 읽은 size만큼을 반환한다.



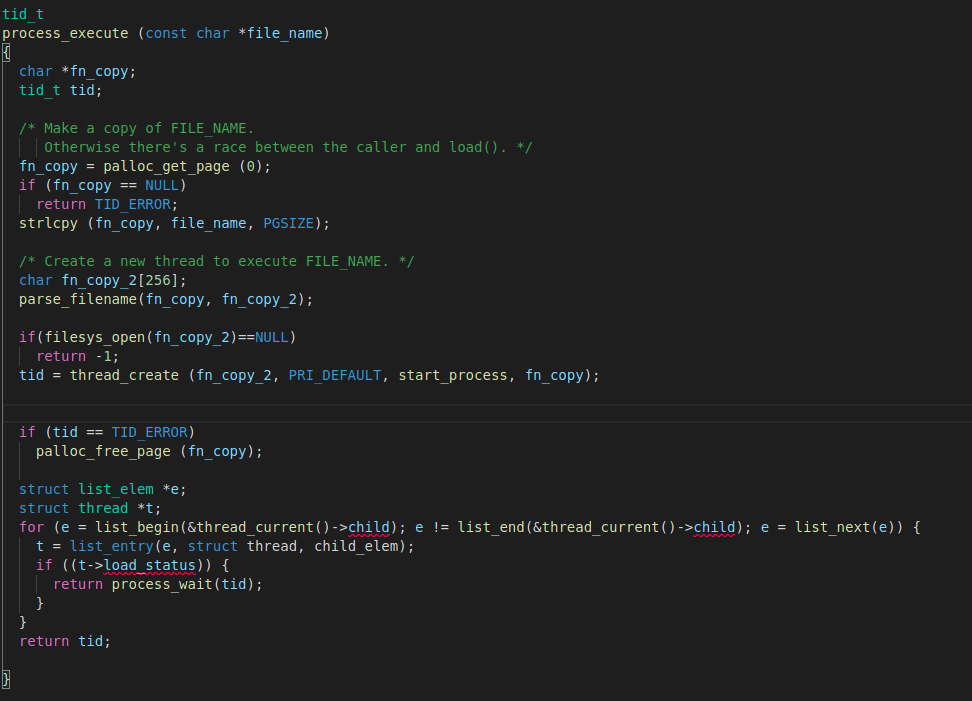
마찬가지로 stdout인 fd ==1을 제외한 나머지 경우에 대해 구현하였다. fd가 NULL인 경우, write가 불가능하므로 syscall\_exit(-1)을 호출하도록 하며, 그렇지 않은 경우, filesys/file.c에 구현된 file\_write함수를 이용해 쓰는 데 성공한 바이트 수를 리턴하도록 한다.



나머지 filesize, seek, tell의 경우 구현 방식은 동일하다. fd가 NULL인지 판별하고 null인 경우 filesystem 접근이 불가능하므로 syscall\_exit(-1)을 호출하고, 그렇지 않은 경우 filesys/file.c에 구현된 file\_length, file\_seek, file\_tell 함수의 수행 결과를 리턴하도록 한다.

syscall\_close의 경우, fd가 null이면 마찬가지로 syscall\_exit(-1)을 호출하며, 주어진 파일을 닫ㄷ도록 한다. 이 때 file\_close는 스스로 fd를 초기화해주지 않으므로 fd를 초기화해 주어야 한다.

1. synchronization



userprog/process.c의 tid\_t process\_execute()에 file = filesys\_open(fn\_copy\_2)를 추가해 file open이 정상적으로 이루어지는 데 실패하면 -1을 리턴하도록 한다.

parent thread가 먼저 terminate되지 않도록 하기 위해 load\_status라는 Boolean flag를 사용하였다. 이 값이 true인 경우, 아직 자식 프로세스가 execute 되고 있음을 의미하므로 process\_wait을 호출한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

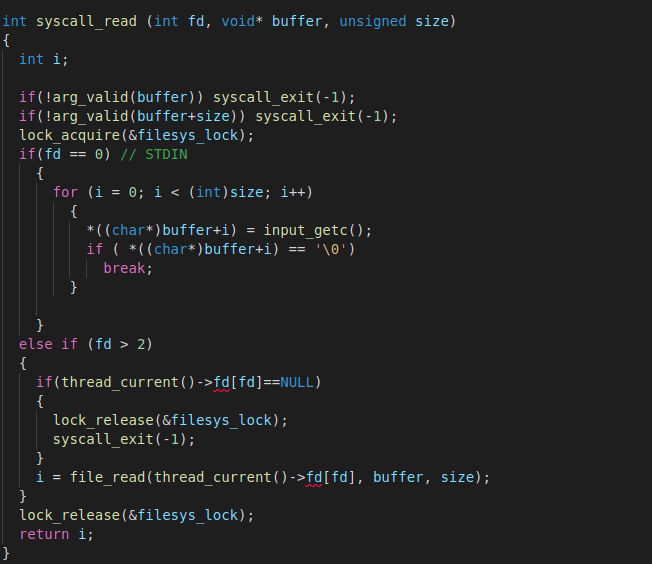
userprog/process.c의 static void start\_process()에서는 load가 잘 이루어지면 success가 true, 그렇지 않으면 false 값을 갖게 된다. load가 제대로 이루어지지 않으면, success가 false가 되고, 이 경우 load\_status를 true로 바꿔줌으로써 자식 프로세스가 제대로 수행되지 않은 것을 보일 수 있다. 이 load\_status는 위의 process\_execute()에서 감지되는데, 이 멤버가 true인 경우, wait하도록 하여 error가 구분되도록 한다.

텍스트이(가) 표시된 사진

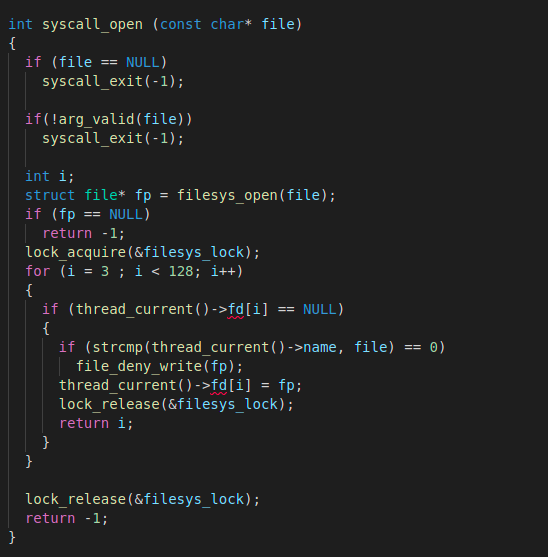
자동 생성된 설명

userprog/exception.c에서, 여러 프로세스가 동시에 움직임에 따른 critical section의 보호를 강화하기 위해 not\_present에 대한 exception을 포함해 exception을 강화한다.

시스템 콜 내부에서도 synchronization이 이루어졌다. 먼저 userprog/syscall.c에 struct lock filesys\_lock을 선언하였다. 그리고, 이는 syscall\_init()에서 lock\_init함수를 통해 초기화 되었다. 파일의 입출력에 따라 mutual exclusion이 만족되지 않는 부분은 read, write open 3 구간으로 보였다. 따라서 userprog/syscall.c의 해당 시스템콜 함수 구현 부분에 lock을 이용한 synchronization을 구현하였다.







이렇게 세 함수의 경우, 주소값의 validity를 확인한 후, file의 접근이 이루어지기 전에 lock\_acquire을 이용해 락을 걸고, 함수값을 리턴하기 전에 lock\_release를 호출하여 다른 프로세스의 파일 접근을 허용하도록 하였다.

또한, filesys의 writer문제를 위해 file\_deny\_write함수를 이용해 synchronization을 구현하엿다. 파일을 오픈하는 경우, 파일에 쓰기 행위가 이루어져서는 안되기 때문에, 현재 실행되는 스레드의 이름이 파일과 동일한 경우, file\_deny\_write 함수를 호출하도록 한다.

1. 구현에 있어 Pintos에 내장된 라이브러리나 자체 제작한 함수를 사용한 경우 이에 대해서도 설명

지난 userprogram(1)과 마찬가지로, 커널 영역을 침범하는지 확인하는 함수 arg\_valid()함수를 이용해 알맞지 않은 주소값의 파라미터들은 syscall\_exit(-1)을 호출하도록 하였다.

1. 개발 중 발생한 문제나 이슈가 있으면 이를 간략히 설명하고 해결한 방식에 대해 설명

multi-oom test를 통과하기 위해 돌아가는 프로세스의 부모 스레드의 load\_sema 를 올리고 내림으로써 race condition에서 mutual exclusion을 만족하려고 했다. 하지만, thread\_current()->parent\_thread->load\_sema에 접근하자, 기존의 userprog test 코드들까지 모두 커널 패닉이 일어나며 핀토스가 정상적으로 작동하지 않아, 해당 코드는 제외하게 되었다. 또한, open, close 둘 다 file\_deny\_write를 호출하자, 핀토스가 구동되지 않았는데, 이는 이중으로 read, write에 접근할 때, file\_deny\_write가 두번 실행되어 아무 행위도 이루어지지 않아 계속 실행되지 않는 문제가 일어난 것으로 보인다. 따라서 write의 file\_deny\_write함수는 주석처리하였다.

* 1. **시험 및 평가 내용**

1. make check 수행 결과를 캡처하여 첨부

