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

담당 교수 / 분반 : 문의현 / 2

이름 / 학번 : 엄석훈 / 20181536

개발 기간 : 2022.11.07 ~ 2022.11.14

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술
* 파일 시스템과 관련된 시스템 콜을 구현한다.
* 파일 시스템에 필요한 file descriptor를 구현하고 synchronization을 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* 아래 각 항목을 구현해야 하는 이유, 혹은 구현 시 기대되는 결과를 간략히 서술

1. File Descriptor

프로세스가 파일에 접근할 때 사용하는 값으로 파일이 처음 open되면 3이상의 fd중에서 사용되지 않는 가장 낮은 값을 부여하고 이 fd를 이용해서 다른 시스템 콜에서 파일에 접근할 수 있도록 해준다.

2. (이번 프로젝트에서 구현해야 하는) System Calls

이번 프로젝트에서는 지난 프로젝트에 이어서 다양한 시스템 콜을 구현한다. 파일 시스템과 관련해서 create, remove, open, close, filesize, read, write, seek, tell을 구현한다.

3. Synchronization in Filesystem

두개 이상의 프로세스가 파일에 접근해서 write를 하게 되면 오류가 발생할 수 있다. 따라서 여러 개의 프로세스가 동시에 critical section에 접근하지 못하도록 synchronization을 해주어야 한다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술

File descriptor를 구현하기 위해서 배열을 사용했다. 핀토스 매뉴얼에서 각 프로세스는 최대 128개의 파일을 open하기 때문에 배열의 크기를 128로 해서 쓰레드에 선언해주었다.

2. System Calls: 구현할 각 system call에 대해 간략히 서술 (하나의 system call 당 최대 3문장으로 간략히 설명; 3문장을 넘길 정도로 길게 작성하지 말 것)

1) create : 파일의 이름과 사이즈를 입력 받아 사이즈 만큼의 크기를 가지는 파일을 생성한다. 파일 생성을 완료하면 true를 리턴하고 실패하면 false를 리턴한다.

2) remove : 파일의 이름을 입력을 받아 해당 파일을 삭제한다. 삭제를 완료하면 true를 리턴하고 실패하면 false를 리턴한다.

3) open : 파일의 이름을 입력받아 해당 파일을 oepn하며 할당해준 fd 값을 리턴해준다. open에 실패하면 -1을 리턴한다.

4) close : 주어진 fd를 가진 파일을 close한고 해당 fd는 초기화해준다.

5) filesize : 주어진 fd를 가진 파일의 사이즈를 바이트 단위로 리턴한다.

6) read : 주어진 fd를 가진 파일에서 size만큼 읽어서 buffer에 저장한다. 그리고 읽어드린 size를 반환한다.

7) write : 주어진 fd를 가진 파일에 size만큼 buffer에서 읽어들여 파일에 write한다.

8) seek : 주어진 fd를 가진 파일에서 read나 write하는 위치를 position만큼 이동시킨다.

9) tell : 주어진 fd를 가진 파일에서 read나 write하는 위치를 리턴해준다.

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

1) Lock : 하나의 파일에 여러 프로세스가 접근하는 것을 막기 위해서 파일에 접근 하기 전에 lock\_acquire를 통해서 다른 프로세스가 접근하지 못하도록 막고, 파일에 접근이 끝나면 lock\_release를 통해서 다른 프로세스가 파일의 critical section에 접근할 수 있도록 해준다.

2) Semaphore : 프로세스가 execute될 때 sema\_down을 해주고 프로세스가 start될 때 sema\_up을 해주어서 프로세스 간의 동기화를 해준다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

11.07 ~ 11.10 : File descriptor 구현 및 시스템 콜 핸들러 구현

11.11 ~ 11.12 : synchronization 구현

11.13 : 일부 fail 테스트 케이스 수정.

11.14 : 보고서 작성

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

1. File Descriptor : thread.c의 thread 구조체에 struct file \*fd\_array[128]을 통해서 각 쓰레드 별로 file descriptor를 관리할 수 있도록 해준다. 그리고 thread.c에서 init\_thread()함수에서 초기화 하는 코드를 추가해준다.

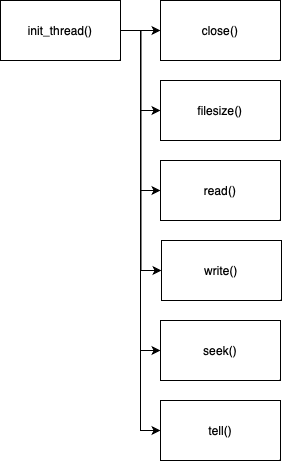
2. System calls : syscall.c에서 syscall\_handler()함수에 각 시스템 콜 넘버가 호출되었을 때 각 시스템 콜 기능을 수행하는 함수가 실행될 수 있도록 switch문에 추가해준다. 그리고 syscall.h와 syscall.c에 각 시스템 콜이 호출되었을 때 수행해야 할 동작을 선언한 함수들을 추가해준다. 이러한 함수들에는 create, remove, open, close, filesize, read, write, seek, tell이 존재한다.

3. Synchronization in Filesystem : 먼저 동기화를 위해서 lock과 semaphore를 사용한다. Lock은 thread.h에 file\_lock을 선언해주고 thread.c의 thread\_init()함수에서 lock\_init()함수를 통해서 락을 초기화해준다. 그리고 각 시스템 콜에서 파일에 접근할 때 lock\_acquire()함수를 통해 critical section을 보호해주고 파일에 대한 접근이 끝나면 lock\_release()함수를 통해 lock을 풀어준다. 그리고 semaphore는 thread.h에 file\_sema를 선언해주고 thread.c에서 file\_sema를 초기화해준다. 그리고 process.c함수에서 strat\_process()에서 sema\_up을 해주고 process\_execute()함수에서는 sema\_down을 통해서 동기화가 되도록 해준다.

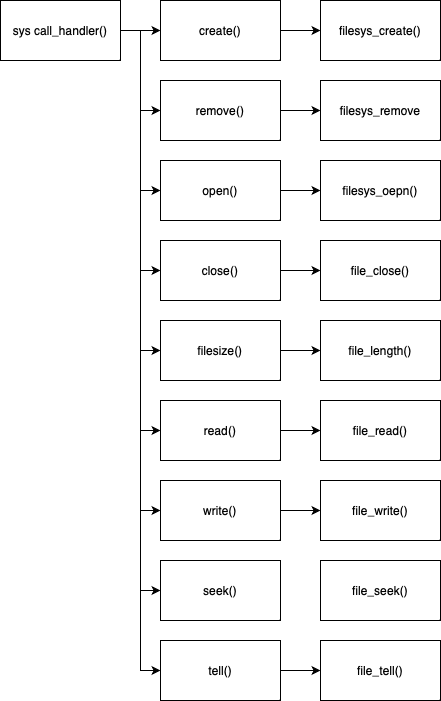
1. **연구 결과**
   1. **Flow Chart**

* II. B. 개발 내용의 각 3가지 항목에 대하여 Flow Chart 작성

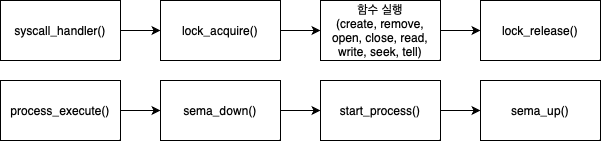
1. File Descriptor



2. System calls

****

3. Synchronization in Filesystem

****

* 1. **제작 내용**
* II. B. 개발 내용의 각 3가지 항목에 대하여 실직적으로 구현한 코드의 관점에서 작성 (구현 내용, 알고리즘 등을 명확히 서술할 것)
* 구현에 있어 Pintos에 내장된 라이브러리나 자체 제작한 함수를 사용한 경우 이에 대해서도 설명
* 개발 중 발생한 문제나 이슈가 있으면 이를 간략히 설명하고 해결한 방식에 대해 설명

1. File Descriptor

텍스트이(가) 표시된 사진

자동 생성된 설명

thread.h 파일에 struct thread에 file descriptor 역할을 수행하는 struct file\* fd\_array[128]을 선언해주었다. 위에서도 언급하였지만 핀토스 매뉴얼에서 각 쓰레드 별로 파일을 최대 128개만 open한다고 가장 간단하게 배열로 선언을 해주었다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그리고 위에서 선언한 fd\_array를 초기화 하기 위해서 thread.c에서 init\_thread()함수에서 초기화 하는 코드를 넣어주었다. 그리고 syscall\_handler()함수에서 file descriptor를 이용해서 해당 파일을 접근하려 할 때 사용할 수 있도록 해주었다. 자세한 코드는 2번에서 설명할 것이다.

2. System calls

**텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

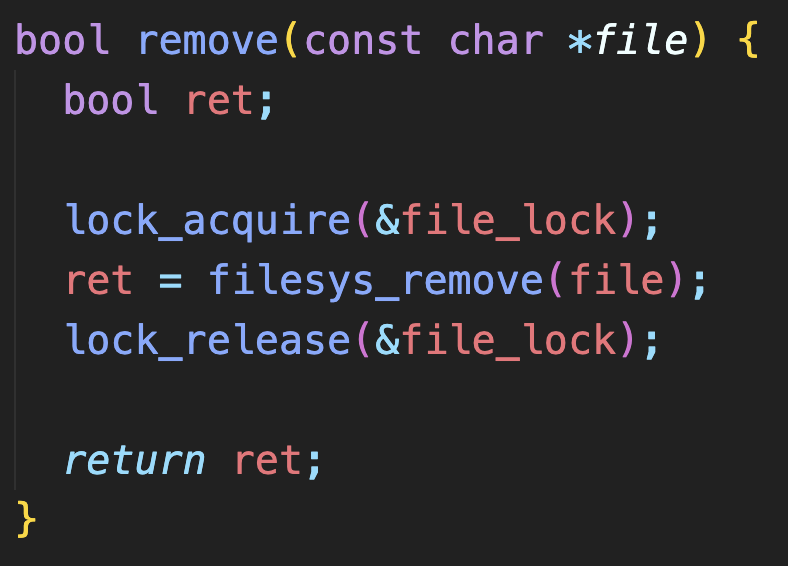
자동 생성된 설명**

먼저 syscall.c에서 syscall\_handler()함수 내부에서 각 시스템 콜 넘버가 불렸을 때 원하는 시스템 콜이 수행될 수 있도록 Proj1에서와 유사하게 스위치 문 안에서 각 함수들을 호출할 수 있도록 해주었다.

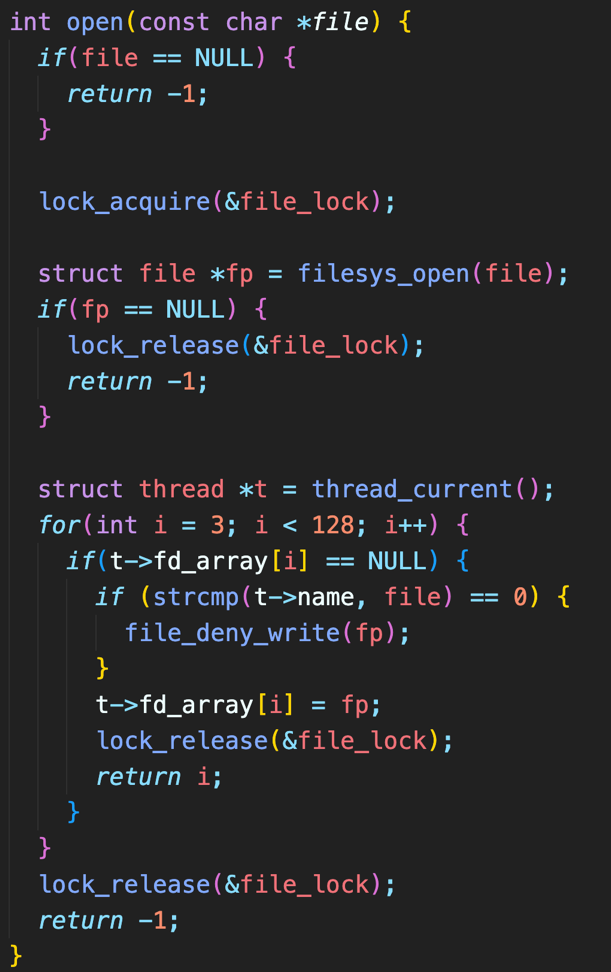
텍스트이(가) 표시된 사진

자동 생성된 설명

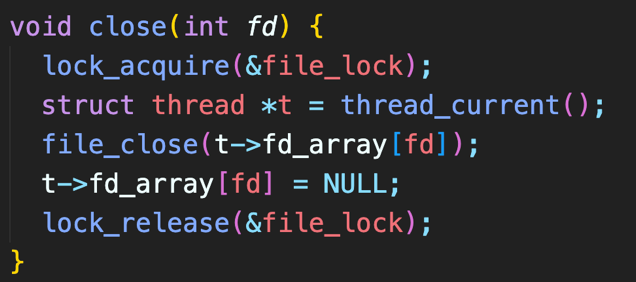
create() 함수는 이미 구현되어 있는 새로운 파일을 create하는 filesys\_create()함수를 호출하는 간단한 함수로 짜주었다.



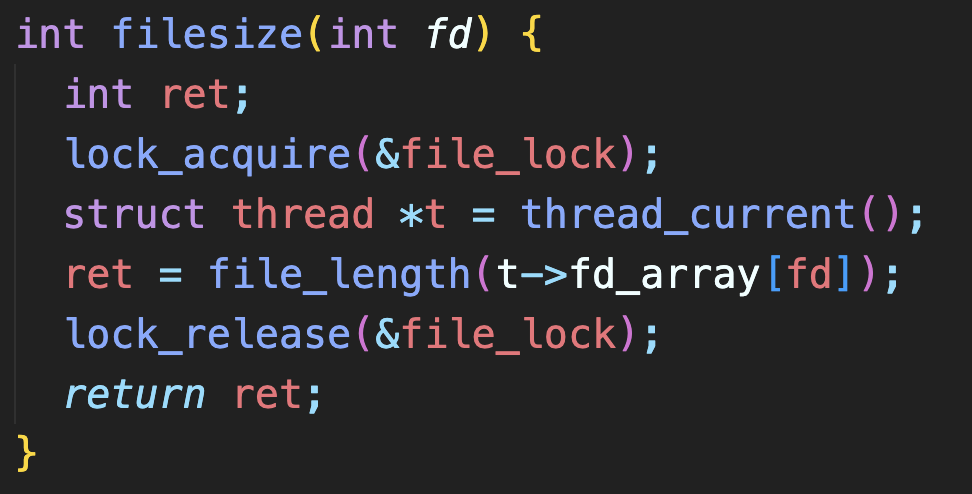
remove()함수는 간단하게 원하는 파일을 remove하는 filesys\_remove()함수를 호출 하도록 간단하게 짜주었다.



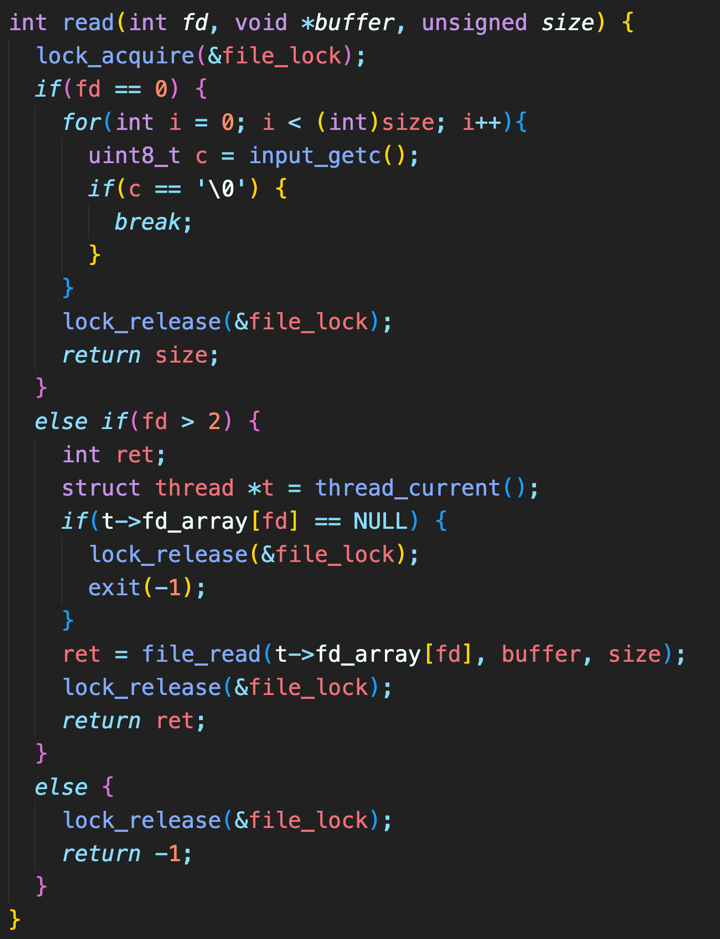
open()함수는 새로운 file을 open하는 함수인 filesys\_open()을 호출하도록 해주었다. 그리고 새로운 file descriptor를 할당해주는데 이 때 1과 2는 STDIN, STDOUT으로 이미 할당되어 있음으로 3 이상의 정수 중에서 아직 할당되지 않은 가장 작은 값을 찾아서 할당해주도록 코드를 짰다.



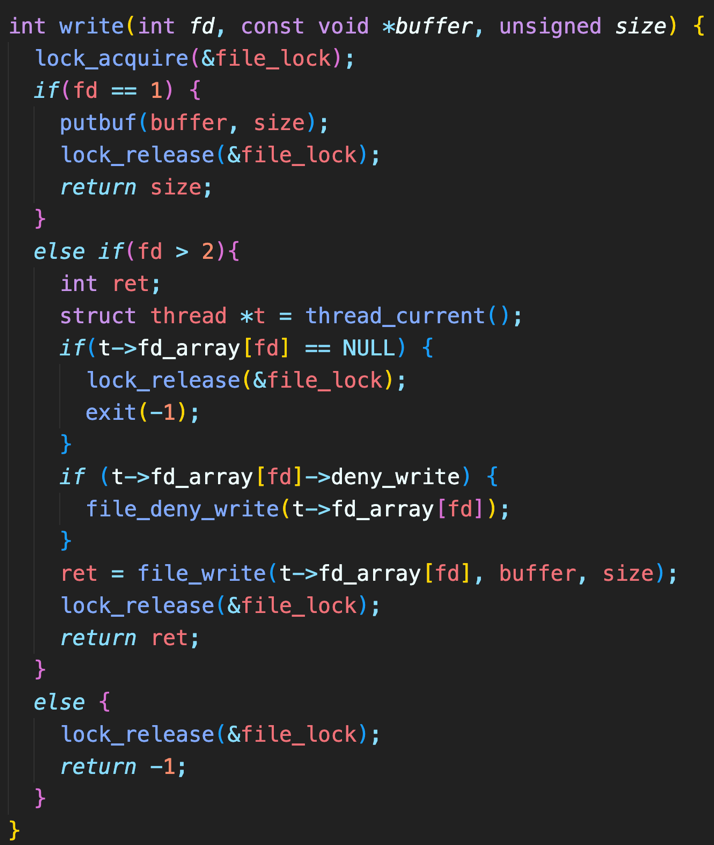
close()함수는 닫고 싶은 파일의 fd값을 받아서 해당 fd값을 가진 파일을 찾은 뒤 file\_close()함수를 호출해서 close하도록 짜주었다.



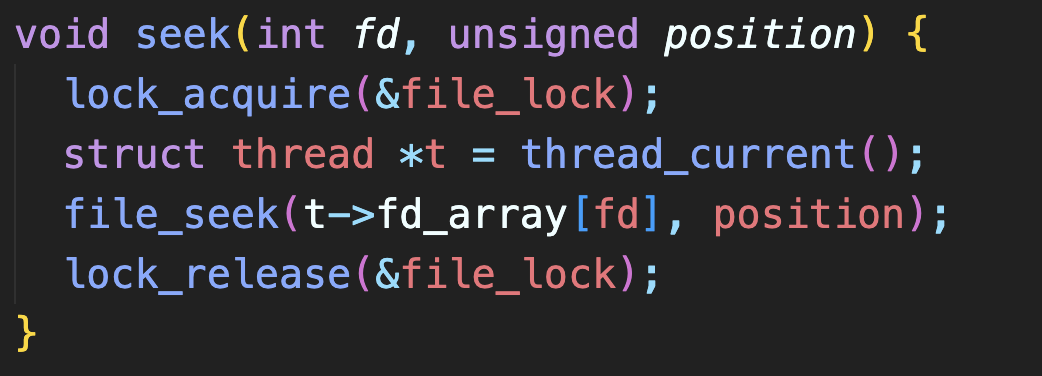
filesize()함수도 간단하게 원하는 fd값을 받은 뒤 file\_length()함수를 호출해서 파일의 사이즈를 리턴하도록 함수를 짜주었다.



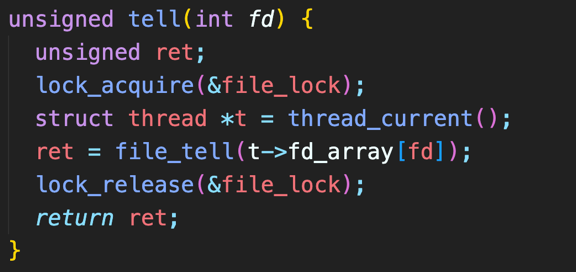
read()함수는 fd값이 0인 경우는 STDIN으로 지난 프로젝트에서 이미 짜주어서 fd가 3부터 해당 fd를 가진 파일을 불러와서 file\_read()를 통해 파일의 내용을 size만큼 읽은 뒤 buffer에 저장하도록 해주었다.



write()함수도 read와 유사하게 fd가 1인 경우는 STDOUT으로 이미 지난 프로젝트에서 짜두었음으로 fd가 3부터 해당 fd를 가진 파일을 불러와 file\_write()함수를 불러와서 size만큼 buffer에서 불러와서 파일에 write하도록 함수를 짜주었다. 이 때 주의할 점이 파일을 write하는 동안 다른 프로세스가 해당 파일에 접근하면 오류가 날 수 있음으로 file\_deny\_write()함수를 호출해서 다른 프로세스에서 이 파일에 접근할 수 없도록 해주었다.



seek()함수도 다른 함수들과 비슷하게 원하는 fd값을 가진 파일을 찾아서 position만큼 이동시키도록 file\_seek()함수를 호출해주었다.



마지막으로 tell()함수도 다른 함수들과 비슷하게 원하는 fd값을 가진 파일을 file\_tell()함수를 통해서 지금 파일의 position을 리턴하도록 해주었다.

3. Synchronization in Filesystem

**텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명**

가장 먼저 thread.h파일에 lock 기능을 사용할 수 있도록 선언해주었다**.**

그리고 thread.c에서 init\_thread()함수에서 해당 lock을 초기화해주었다.

그 뒤 위에서 파일관련 시스템 콜을 호출할 때 매번 lock\_acquire()를 가장 먼저 호출하고 함수가 종료되기 전에 lock\_release()를 해주었다.

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

그리고 thread.h에서 thread구조체에 struct semaphore file\_sema를 선언해주고 thread.c에서 thread\_init()함수에서 해당 semaphore를 초기화 해주는 코드까지 넣어주었다.

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

그리고 해당 세마포어는 process.c의 process\_execute()함수에서 sema\_down을 해주고 start\_process()함수에서는 sema\_up을 할 수 있도록 해주었다.

* 1. **시험 및 평가 내용**
* make check 수행 결과를 캡처하여 첨부

텍스트이(가) 표시된 사진

자동 생성된 설명