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

담당 교수 / 분반 : 박성용 교수님 / 1반

이름 / 학번 : 정진원 / 20161643

개발 기간 : 11/1 ~ 11/15

1. **개발 목표**

Pintos에서 파일 입출력을 위해 필요한 파일 디스크립터를 구현하고, 파일 시스템에 대한 create, remove, open, close, filesize, seek, tell 시스템 콜을 구현한다. 또한, reader/writer problem과 같은 동기화 문제들도 해결한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

**1. File Descriptor**

각각의 process는 독립된 file descriptor를 가지며, process에서 파일에 접근하게 해주기 위해 필요한 file descriptor의 구현이 필요하다.

**2. System Calls**

Pintos의 file system의 구현과 사용을 위해 stdin/stdout에만 작동하던 read/write system call의 기능이 file에 대해 동작하도록 확장해주었다. 또한, create, remove, open, filesize, seek, tell, close에 대한 system call들에 대응하는 기능을 구현하였다.

**3. Synchronization in Filesystem**

파일에 대한 동시 접근을 막기 위해 synchronization을 구현해야 한다. 이는 오직 하나의 process가 file system code를 수행해야 하는 pintos의 요구 조건을 충족시키기 위함이다. 또한, 실행중인 파일에 대한 수정은 예측할 수 없는 결과를 야기할 수 있기 때문에 이를 막아준다.

* 1. **개발 내용**

**1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술**

File Descriptor의 구현을 위해 배열을 사용하였다. 배열의 크기는 131로 설정해주었다. 이는 pintos 매뉴얼에서 필요에 따라 최대로 열 수 있는 파일을 프로세스당 128개로 설정하라고 하여 stdin, stdout, stderr를 위한 3개의 공간을 더해 나온 값이다. 또한 동적할당에 의해 발생할 수 있는 오류를 방지하기 위해 배열을 이용하였다.

**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)**

- SYS\_READ: 기존의 read에서 파일을 읽어야 하는 경우에 대한 처리까지 해준 system call. fd > 2면 fd에 해당하는 파일을 읽는다.

- SYS\_WRITE: 기존의 write에서 파일에 대한 입력을 하는 경우까지 처리해준 system call. fd > 2면 fd에 해당하는 파일에 대한 입력을 수행한다.

- SYS\_CREATE: 인자로 전달받은 파일명과, 파일 크기에 맞는 파일을 생성하게 해주는 system call.

- SYS\_REMOVE: 인자로 전달받은 파일명에 해당하는 파일을 삭제해주는 system call.

- SYS\_OPEN: 인자로 전달받은 파일명에 해당하는 파일을 열어주는 system call. Open에 성공하는 경우 fd 값을 반환한다. Fd는 프로세스마다 독립적이며 이는 child에게 상속되지 않는다. 또한, 동일한 파일이 여러 번 open되는 경우에 open마다 다른 fd가 부여된다.

- SYS\_FILESIZE: 인자로 전달받은 fd에 해당하는 파일의 크기가 몇 byte인지 반환해주는 system call.

- SYS\_SEEK: 인자로 전달받은 fd와 position으로 fd에 해당하는 파일에서 다음 읽기/쓰기가 수행될 위치를 시작점으로부터 떨어진 byte 수로 나타내어진 position으로 옮겨주는 system call.

- SYS\_TELL: 인자로 전달받은 fd에 해당하는 파일에서 다음 읽기/쓰기가 수행될 위치가 어디인지 반환해주는 system call. 이 위치는 시작점으로부터 몇 byte만큼 떨어져있는지로 나타내어진다.

- SYS\_CLOSE: fd에 해당하는 file을 닫아주는 system call. 또한 thread의 file descriptor 배열에서 fd에 해당하는 위치를 NULL로 초기화 해준다.

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

- Lock을 이용하는 방법

Synchronization을 위해 lock을 이용하는 경우에 전역변수 struct lock filesys\_lock을 선언해주고 read() 나 write()가 수행되기 전에 lock을 lock\_acquire()를 통해 획득하도록 한다. Lock이 acquire된 경우 lock이 release되기 전까지 다른 read()나 write()가 수행 될 수 없다. 수행이 완료되었을 때 lock을 lock\_release()를 통해 해제해주어 한번에 하나의 파일에 대한 접근이 이루어질 수 있도록 한다.

- Semaphore를 이용하는 방법

강의 시간에 배웠던 readers / writers problem을 해결하는 방법으로, read는 동시에 진행 될 수 있지만 write는 한번에 하나만 진행될 수 있게 mutex, wrt semaphore와 readcnt라는 변수를 사용한다. Read가 시작되면 wrt semaphore를 down 시켜 write의 수행이 이루어질 수 없도록 하고 read의 수행이 모두 완료되어 readcnt가 0이 되면 wrt semaphore를 up 시켜주어 wrt가 수행 될 수 있도록 해준다. Read의 경우 read에 진입할 때, mutex를 down 시켜주고 readcnt가 늘어나면 mutex를 up 시켜주어 다른 read가 진행될 수 있도록 해준다. Read가 끝나면, 모든 read가 끝났는지 확인하기 위해 mutex를 down 시키고 up 시켜주는 과정을 거친다. 또한 syn-read, syn-write을 통과하기 위해서 child thread들이 제대로 load 될 때까지 parent thread에서 기다려주는 과정이 필요한데, 이를 semaphore를 이용해서 프로젝트 1에서 구현하였다. 간략히 설명하면 process\_execute() 함수에서 semaphore l\_lock (load\_lock)을 down 시켜주고 start\_process에서 child thread가 load에 제대로 load되면 parent thread의 semaphore l\_lock을 up 시켜주어 load가 제대로 될 때까지 parent thread가 기다리게 해준다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* 11/1 ~ 11/7 프로그램 요구사항 파악
* 11/8 ~ 11/14 file descriptor, system call, synchronization에 관한 전반적인 내용을 구현했다.
* 11/15 ~ 11/16 test case 중 통과하지 못한 bad-read, bad-write과 같은 경우를 해결하였다.
  1. **개발 방법**

1. **File Descriptor**

Src/threads/thread.h의 struct thread에 file descriptor 배열을 추가해주었다. 또한 src/threads/thread.c의 init\_thread 함수에 thread가 새로 생성되는 경우 file descriptor 배열을 초기화 해주는 코드를 추가해주었다.

1. **System calls**

Src/userprog/syscall.c에 구현해야 하는 system call read, write, create, remove, open, filesize, seek, tell, close에 대응되는 system call number에 맞는 기능을 수행하도록 함수들을 구현해주었다.

Bad-read / bad-write과 같이 기본 주소에 NULL이 들어오는 경우를 해결하기 위해 src/userprog/exception.c의 page\_fault 함수에 조건을 추가해주었다.

파일 이름으로 NULL이 들어오는 경우를 src/userprog/syscall.c에 구현한 check 함수에 조건을 추가해주어 확인해주었다.

Thread가 종료되는 경우, thread에서 열었던 파일을 모두 닫아주어야 하기 때문에 src/userprog/syscall.c의 exit() 함수에 thread에서 열고 있는 file을 닫아주었다.

1. **Synchronization in Filesystem**

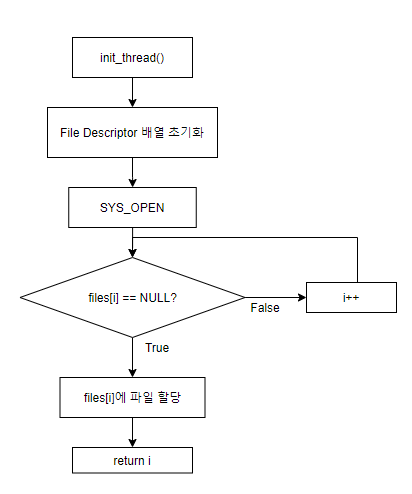
Child가 완전히 load 되기 전에 parent의 execute이 끝나는 경우를 방지하는 방법을 프로젝트 1에서 semaphore를 이용하여 구현했었다.

Read가 진행되는 동안 write의 접근을 막고 write가 진행되는 동안 read의 접근을 막기 위해 mutex, wrt semaphore와 readcnt라는 변수를 사용하여 src/userprog/syscall.c의 read와 write함수에 내용을 추가해주었다. 이 변수들을 전역변수로 syscall.c에 선언하였으며 syscall\_init()함수에서 초기화해주었다.

실행중인 파일에 대한 수정을 막기 위해 src/userprog/syscall.c의 open 함수에서 현재 수행중인 thread와 file의 이름이 같다면 file\_deny\_write()함수를 이용하여 수정이 불가능하도록 해주었다.

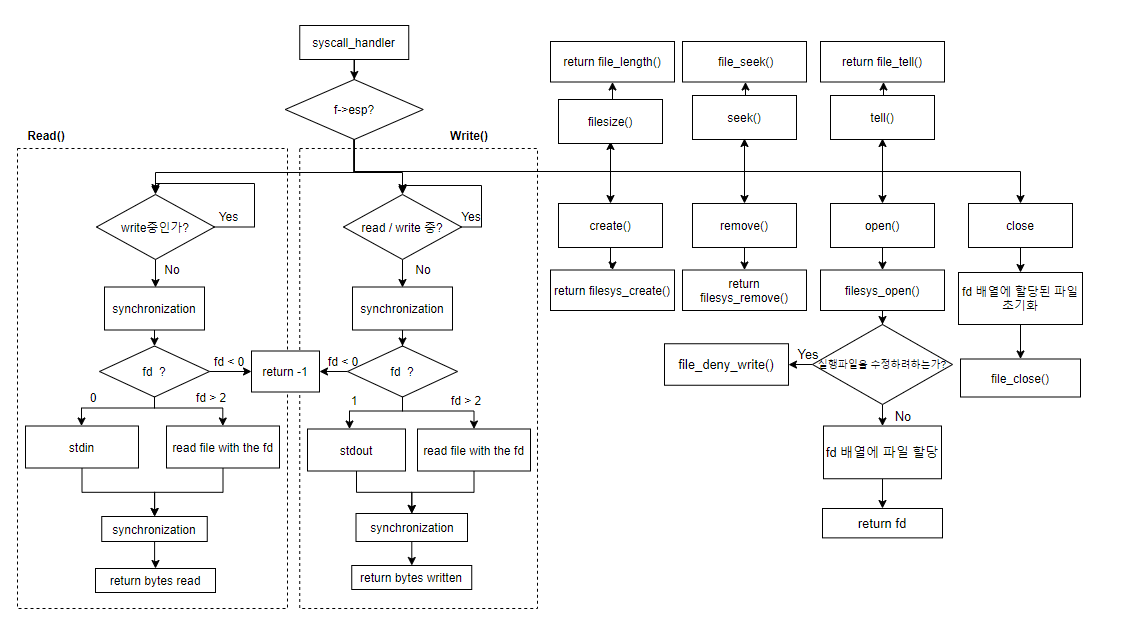
1. **연구 결과**
   1. **Flow Chart**

**1) File Descriptor**

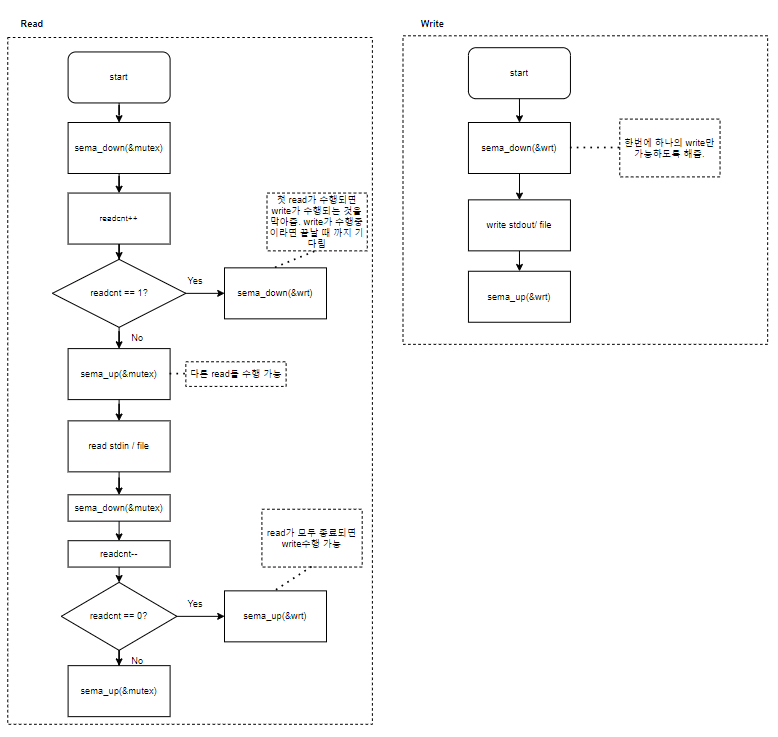


System call open이 수행되면, 수행중인 thread의 file descriptor 배열에서 stdin / stdout / stderr를 위해 존재하는 세 개의 공간을 제외한 files의 네 번째 원소부터 확인하여, 만약 files[i] 가 NULL이라면 배열에 파일을 할당해주고, non-negative integer 값인 fd값 i를 반환해준다. 할당된 파일들이 존재한다면, files[i] == NULL이 될 때까지 i를 증가시켜준 뒤, files[i]에 파일을 할당해주고 fd값 i를 반환해준다.

**2) System calls**

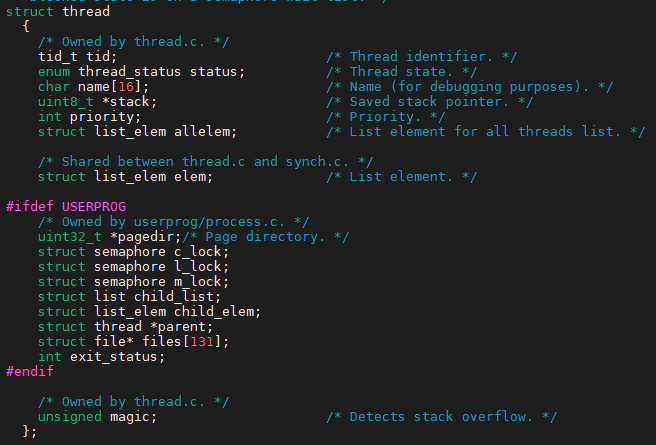


**3) Synchronization in Filesystem**

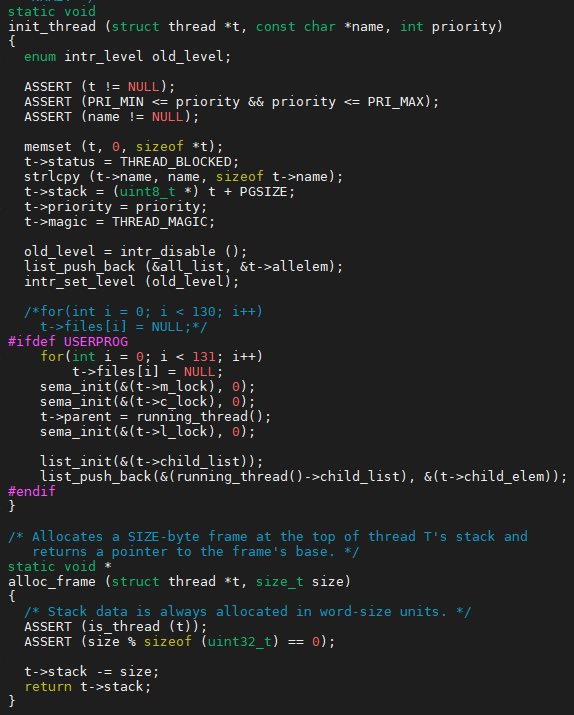


* 1. **제작 내용**

**1) File Descriptor**



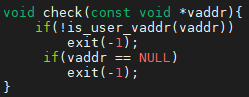
File descriptor들을 저장할 files 배열을 thread 구조체에 선언해주었다.



Thread가 initiate 될 때, 해당 thread의 files 배열의 원소들을 null로 초기화 해주었다.

**2) System calls**

먼저 기존의 check 함수를 수정해주어, null 값이 들어온 경우 exit(-1)을 수행하게 해주었다.



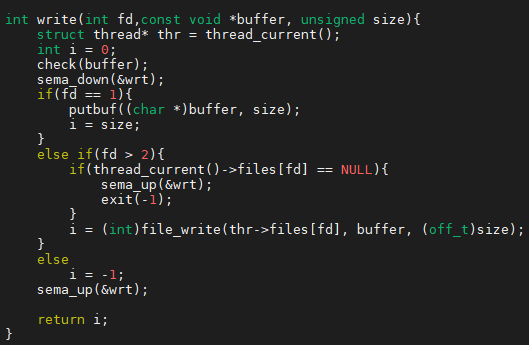
Null 값이 입력되는 경우, check 함수만으로는 부족한 문제가 발생하여 src / userprog / exception.c 의 page\_fault 함수에 아래와 같이 not\_present인 경우 exit(-1) 하도록 조건을 추가해주었다.



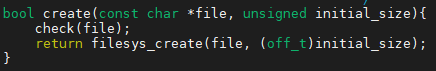
- read(): 기존에 구현했던 read에 fd > 2 이상인 경우, 즉 파일을 읽어야하는 경우에 대해 pintos의 file\_read() 함수를 통해 파일을 읽었다. 파일을 모두 읽은 뒤 실제로 읽은 byte수를 return 해준다. 잘못된 fd가 들어오면 -1을 반환해준다. 또한 synchronization 문제를 해결하기 위해 강의 시간에 배웠던 readers/writers problem에 대한 해결책을 semaphore를 이용해 구현하였다. 이는 3)Synchronization in Filesystem에 더 자세하게 다루겠다.



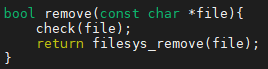
- write(): 기존에 구현했던 write에 fd > 2 이상인 경우, 즉 파일에 입력을 해야하는 경우에 pintos의 file\_write() 함수를 통해 입력을 해주었다. Write가 모두 수행된 뒤, 실제로 write된 byte 수를 return 해준다. 잘못된 fd가 들어온 경우 -1을 반환해준다. 또한 synchronization 문제를 해결하기 위해 파일을 읽는 동안 접근을 못하게 하는 semaphore를 추가해주었다.



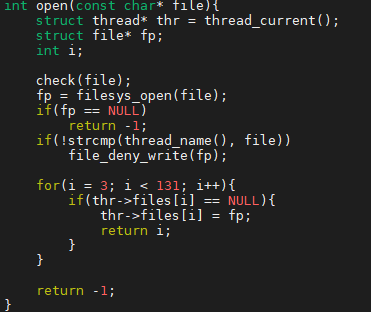
- create(): 인자로 파일 이름과 size를 받아 제대로 된 이름인지 check 함수에서 확인해준 뒤, 이상하다면 에러 처리를 해준다. 이후 pintos의 filesys\_create() 함수를 이용하여 파일을 생성해준다. 생성에 성공하면 true를, 아니면 false의 값을 반환한다.



- remove(): 인자로 파일 이름을 받아 해당하는 파일을 삭제해준다. 이때 제대로 된 이름인지 check 함수에서 확인해주며 pintos의 filesys\_remove() 함수를 통해 파일을 제거한다. 제거에 성공하면 true를 아니면 false를 반환한다.



**-** open(): 인자로 파일 이름을 받아 해당하는 파일이 제대로 된 파일 이름인지 확인 한 뒤, pintos의 filesys\_open() 함수를 이용하여 파일을 열어준다. 수행중인 파일을 수정하는 것을 허용하면 안되기 때문에 현재 thread와 파일 이름이 같다면 pintos의 file\_deny\_write()함수를 불러주었다. 이후, for문을 돌며 수행중인 thread의 file descriptor 배열에서 stdin / stdout / stderr를 위해 존재하는 세 개의 공간을 제외한 files의 네 번째 원소부터 확인하여, 만약 files[i] 가 NULL이라면 배열에 파일을 할당해주고, non-negative integer 값인 fd값 i를 반환해준다. 할당된 파일들이 존재한다면, files[i] == NULL이 될 때까지 i를 증가시켜준 뒤, files[i]에 파일을 할당해주고 fd값 i를 반환해준다.



- filesize(): 인자로 file descriptor를 받아 해당하는 파일의 크기가 몇 byte인지 pintos의 file\_length()함수를 통해 확인한 뒤 반환해준다.



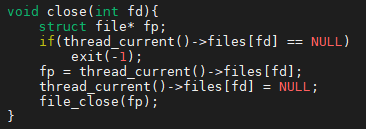
- seek(): 인자로 전달받은 fd와 position으로 fd에 해당하는 파일에서 다음 읽기/쓰기가 수행될 위치를 시작점으로부터 떨어진 byte 수로 나타내어진 position으로 pintos의 file\_seek 함수를 통해 옮겨주는 함수.



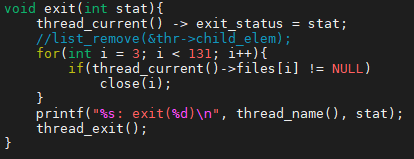
- tell(): 인자로 전달받은 fd에 해당하는 파일에서 다음 읽기/쓰기가 수행될 위치가 어디인지 pintos의 file\_tell() 함수를 이용하여 확인하고 반환해준다. 이 위치는 시작점으로부터 몇 byte만큼 떨어져있는지로 나타내어진다.



- close(): fd에 해당하는 file을 닫아주는 system call. Thread의 file descriptor 배열에서 fd에 해당하는 위치를 NULL로 초기화 해준 뒤, pintos의 file\_close() 함수를 이용하여 파일을 닫아주었다. 전달된 fd에 해당하는 file이 이미 null이라면 파일을 이미 닫힌 파일이라는 뜻이기 때문에 이 경우는 예외처리를 해주었다.



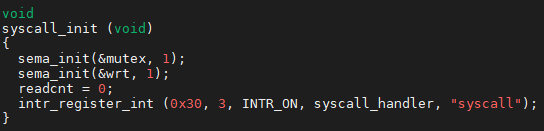
또한 thread가 종료 될 때, thread가 열고 있는 모든 파일을 닫아주어야 하므로 thread 가 종료되는 경우, 즉 exit()함수가 호출되는 경우에 thread의 files 배열을 돌며 파일들을 닫아준다. 이는 exit() 함수에 구현하였다.



**3) Synchronization in Filesystem**



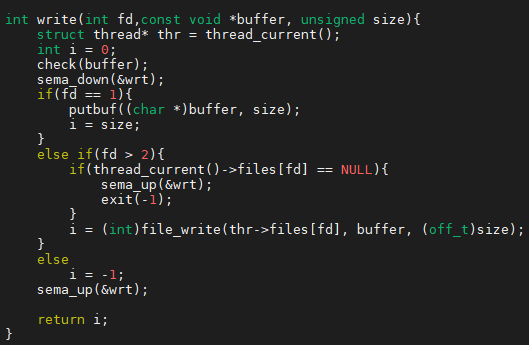
Synchronization을 위해 mutex, wrt 두개의 semaphore를 전역변수로 선언해주고, 몇 개의 read가 수행 중인지를 나타내는 readcnt라는 변수를 전역변수로 선언해주었다.



선언 된 전역변수들의 값을 syscall\_init()이 수행될 때 초기화 해주었다.



Readers/Writers problem을 해결하는 방식으로 synchronization을 구현하였다. 이를 자세히 설명하면 read가 수행 될 때, mutex를 down 시켜주며 진입한다. 이후 readcnt를 증가시켜준다. 만약 첫 read 여서 readcount가 1이 되었다면, write가 수행 될 수 없도록 wrt semaphore를 down 시켜준다. 만약 write가 진행 중이라면 write가 끝날 때까지 기다린다. 이후 다른 read 들이 진입할 수 있도록 다시 mutex를 up 시켜준다. Reading이 종료되면 mutex를 down 시켜주고 readcnt를 감소시킨다. Readcnt가 0이되어 모든 reading이 종료되었다면 write가 수행 가능해야 하므로 wrt semaphore를 up 시켜준다. 이후 다시 read가 접근할 수 있도록 mutex를 up 시켜준다.



Write의 경우 한번에 하나의 write만 수행가능하기 때문에, write에 진입할 때 wrt semaphore를 down 시켜주고, write이 모두 종료되면 up 시켜주어 하나의 write만 수행되도록 한다.

* 1. **시험 및 평가 내용**
* 