Operating Systems Project #3 Reader-Writer 문제

2018045241 홍산해, 2020062324 이은비, 2020088568 백홍열

1. 설계한 reader 선호, writer 선호, 공정한 reader-writer 알고리즘

A. Reader 선호

Reader 선호 알고리즘의 전제 조건은 아래와 같다.

1. reader는 CS에 다른 reader가 있어도 중복해서 들어갈 수 있다. 단, writer와는 중복될수 없다 (writer와 중복 불가)
2. writer는 CS에 reader가 1개도 없어야만 들어갈 수 있다. (reader와 중복 불가)
3. writer는 CS에 1개씩만 들어갈수 있다. (writer끼리 중복 불가)

```
int read_count = 0; // 조건변수 : 현재 실행중인 reader 의 개수
pthread_mutex_t mutex; // 조건변수의 상호배타를 위한 mutex lock 1 개
pthread_cond_t writer_cond_var; // POSIX condition
```

Reader 선호 설계 조건 (proj3.pdf, 2p)에 의해, mutex 1개와 POSIX condition을 사용한다.
read_count는 reader와 writer를 위한 조건변수로, 현재 실행중인 reader의 개수이다.
mutex는 조건변수 read_count의 상호배타를 위한 mutex lock 이다.
rw_cond_var은 POSIX condition variable 이다.

이를 통해 설계한 reader 알고리즘은 아래와 같다.

1. reader는 CS에 writer가 없는 한 계속 해서 실행한다 (전제 조건 1번). 이때 조건변수 read_count를 수정하는 과정에서만 mutex lock을 통해 상호 배타를 보장한다.
2. writer는 1번에 1개씩만 실행이 가능하므로 condition을 통해 이를 보장한다 (전제 조건 3번). 이 때 reader에서 read_count를 통해 현재 실행중인 reader의 개수를 검사하고 조건이 충족되면(read_count가 0이면) signal을 통해 대기중인 writer를 깨워준다 (전제 조건 2번). 또한 writer가 실행되는 동안 reader의 실행을 막기 위해 writer의 critical section이 끝난 뒤에 mutex lock을 풀어준다. (전제 조건 1번)

B. Writer 선호

```
pthread_cond_t w_cond;    // writer 들의 대기열
pthread_cond_t r_cond;    // reader 들의 대기열
pthread_mutex_t lock
int writer_cnt = 0;       // 쓰기 중인 writer 수
int reader_cnt = 0;       // 읽기 중인 reader 수
int reader_wait = 0;      // 대기 중인 reader 수
int writer_wait = 0;      // 대기 중인 writer 수
```

i. Reader

1. Reader는 하나 이상의 writer가 대기하거나 실행하는 경우 모두 대기열로 들어간다.
2. CS에서 나온 후 실행 중인 reader가 없다면 대기 중인 writer를 깨운다.
3. signal함수 특성 상 대기 중인 writer가 없더라도 큰 문제가 일어나지 않는다.
4. alive = false가 된 이후에는 대기열에 이미 들어간 스레드들이 알고리즘을 따라
5. 모두 수행한 뒤에 모든 스레드가 종료된다.

ii. Writer

1. writer함수가 조건을 확인해 reader를 모두 깨우거나 다음 writer를 깨운다.
2. 한 번에 깨어난 reader들은 lock획득 순서에 따라 while 조건문을 다시 체크한 후 진행한다. 따라서 깨어난 reader보다 먼저 lock을 획득한 writer가 있다면 reader는 while문을 다시 돌아 대기 상태로 들어간다.
3. alive = false가 된 이후에는 대기열에 이미 들어간 스레드들이 알고리즘을 따라
4. 모두 수행한 뒤에 모든 스레드가 종료된다.

C. 공정한 reader-writer

```
pthread_mutex_t arrive;
pthread_mutex_t r_mutex;
pthread_mutex_t rw_lock
int reader_cnt = 0;
```

도착 순서대로 실행되도록 도와주는 arrive 뮤텍스를 사용한다. 처음 arrive 뮤텍스에 lock을 시도한다. 만약 다른 스레드가 이미 arrive 뮤텍스를 소유중이라면 대기 상태에 있게된다. critical section에 들어가기전 잠금이 해제되고, 대기중이던 다른 스레드가 arrive 뮤텍스를 소유할 수 있게된다.
reader끼리 중복을 허용하게 도와주는 r_mutex를 사용한다. 현재 실행중인 reader의 수를 알려주는 reader_cnt 공유변수를 접근할 때 사용한다.
writer-writer, writer-reader 간 간섭을 막기 위한 rw_lock 뮤텍스를 사용한다. writer 스레드 실행시 critical section은 해당 뮤텍스에 의해 보호된다. reader는 중복될 수 있기 때문에, 현재 실행중인 첫번째 reader라면 해당 뮤텍스를 소유하고, 마지막 reader라면 소유한 rw_lock 뮤텍스를 해제한다

이를 통해 설계한 공정한 reader-writer 알고리즘은 아래와 같다.

i. Reader

1. arrive 뮤텍스에 lock 을 건다. 만약 다른 스레드가 arrive 뮤텍스를 소유하고 있다면 대기한다. 해당 reader 스레드가 arrive 뮤텍스를 소유했다면, 이후 도착한 다른 스레드는 arrive 뮤텍스의 해제를 기다리며 대기하게 된다.
2. reader_cnt 공유변수에 접근하기 위해 r_mutex를 잠근다. 만약 1 증가시킨 reader_cnt 결과값이 1이라면 rw_lock 뮤텍스에 lock 을 건다. 이로써 다른 writer 의 접근을 막고, 다음에 오는 reader 의 접근은 허용할 수 있다.
3. arrive 뮤텍스에 건 lock을 해제한다. arrive 뮤텍스를 기다리던 스레드가 동작한다. r_mutex lock도 해제후 critical section 에 들어간다.
4. CS 이후 eader_cnt 공유변수에 접근하기 위해 r_mutex 를 잠근다. 만약 1 감소시킨 reader_cnt 결과값이 0 이라면, 기다리고 있는 다른 writer의 접근을 가능케하기 위해 rw_lock 뮤텍스에 건 lock을 푼다.

ii. Writer

1. arrive 뮤텍스에 lock 을 건다. 만약 다른 스레드가 arrive 뮤텍스를 소유하고 있다면 대기한다. 다른 reader 가 cs 에 들어가있고, writer 도착 직후 reader 가 도착했다면 writer 가 arrive 뮤텍스를 소유하고 있고 도착한 reader 는 writer 가 arrive 뮤텍스를 반환하기만을 기다리게 된다. writer 가 도착하더라도 마찬가지이다.
2. writer가 arrive 뮤텍스를 소유했더라도, 앞서 reader나 writer가 rw_lock 뮤텍스를 소유하고 있다면 해당 뮤텍스의 반환을 기다린다. 만약 rw_lock 뮤텍스를 소유하게 되었다면, arrive 뮤텍스의 lock 을 풀고, critical section 에 들어간다.
3. CS 이후 rw_lock 뮤텍스의 잠금을 해제한다.

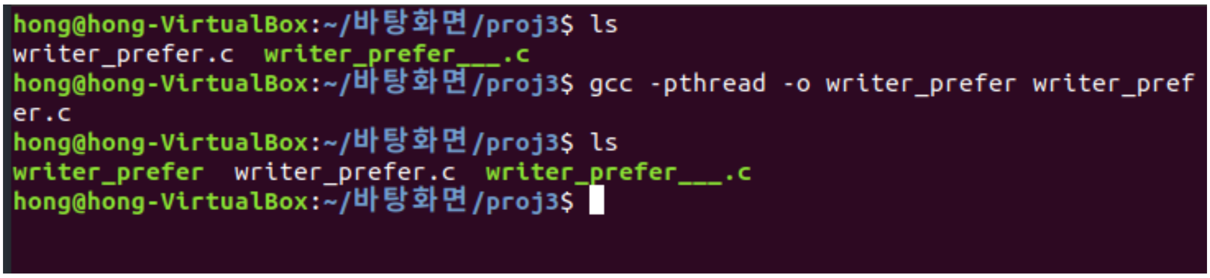
2. 컴파일 과정

gcc를 통해 reader_prerfer.c / writer_prerfer.c / fair_reader_writer.c를 컴파일한다. 이때 각각의 프로그램은 pthread 라이브러리를 통해 다중 스레드를 사용하므로, '-lpthread' 옵션을 붙여준다.

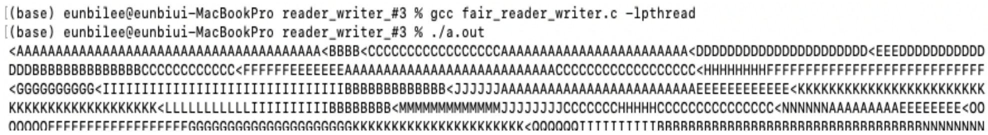
A. Reader 선호

```
sanhaehong@admin:~/operation_system/proj3$ ls
reader_prefer.c
sanhaehong@admin:~/operation_system/proj3$ gcc -o reader_prefer reader_prefer.c -lpthread
sanhaehong@admin:~/operation_system/proj3$
```

B. Writer 선호



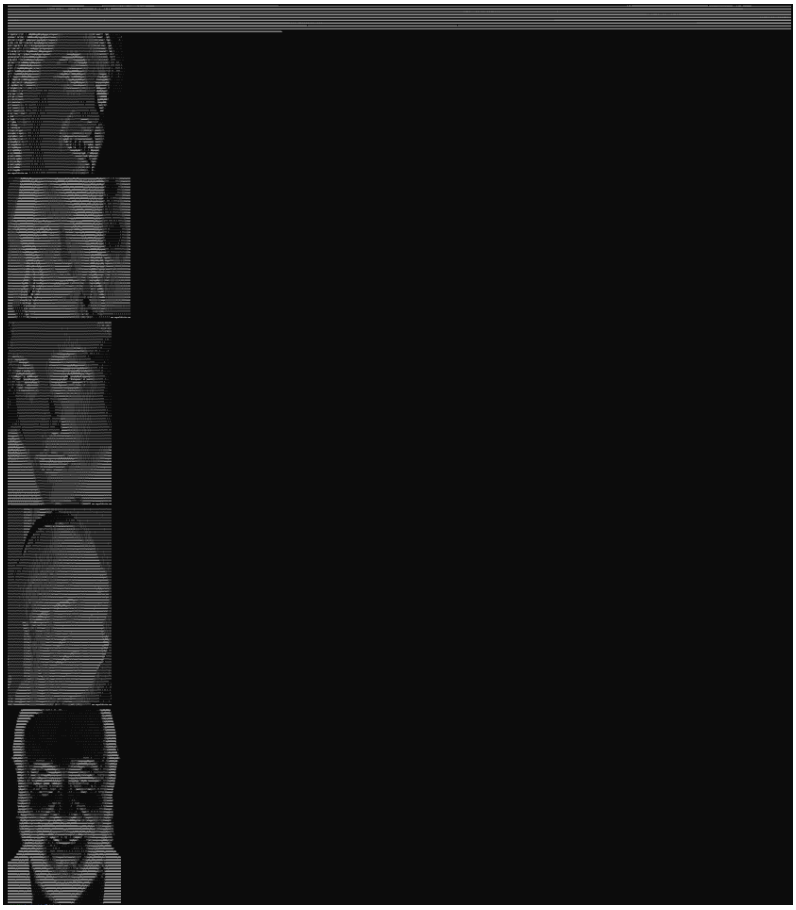
C. 공정한 reader-writer



3. 실행 결과물

A. Reader 선호

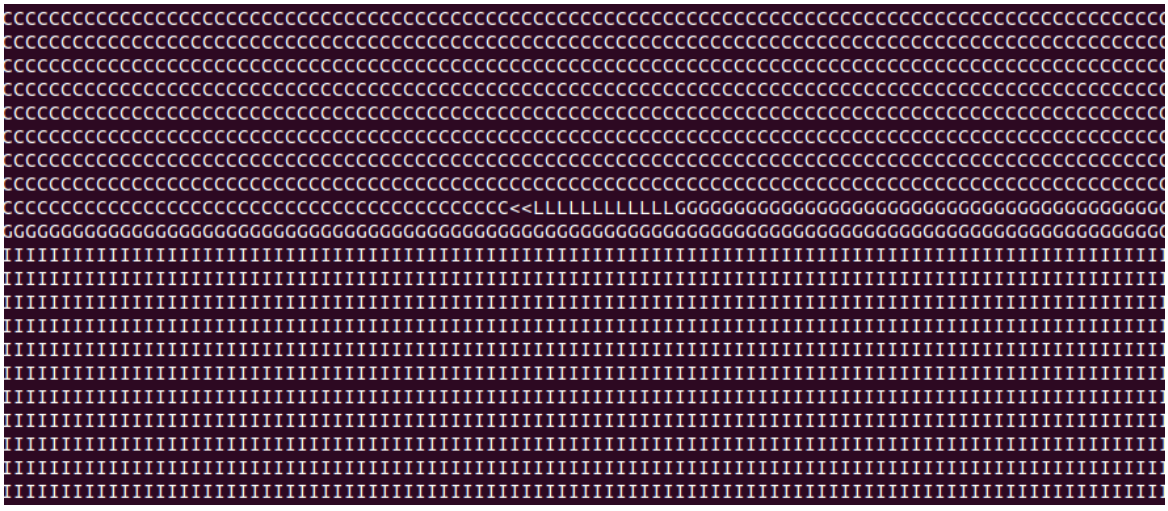
처음에는 reader들이 먼저 실행되며 문자들을 출력한다
writer_prefer, 공정한 reader-writer과는 반대로 reader가 먼저 실행되며, reader는 writer를 앞지를수 있으므로 일반적인 경우 모든 reader가 writer보다 먼저 출력된다.
이때 reader_prefer의 조건에 따라 reader끼리의 중복을 허용했으므로, 〈AAA…AA〉처럼 같은 문자만 출력하지 않고 어지럽게 섞여서 나온다.



reader가 모두 끝난 뒤에는 writer가 얼굴 이미지를 출력한다.
이 때 POSIX condition을 통해 writer 간 중복을 방지하여 이미지가 깨지지 않고 출력된다.
또한 늦게 들어온 reader가 writer를 앞서가므로 writer가 계속해서 대기하는 현상 (starvation)이 발생하여 writer는 모든 reader가 끝난 뒤에 문자를 출력한다.
반면 writer_prefer은 writer가 우선권을 가지므로 반대로 reader가 계속해서 대기하는 현상이 발생하며, fair_reader_writer는 어느 한 쪽이 일방적으로 대기하지 않고 쓰레드의 속도에 따라 순서가 결정된다.

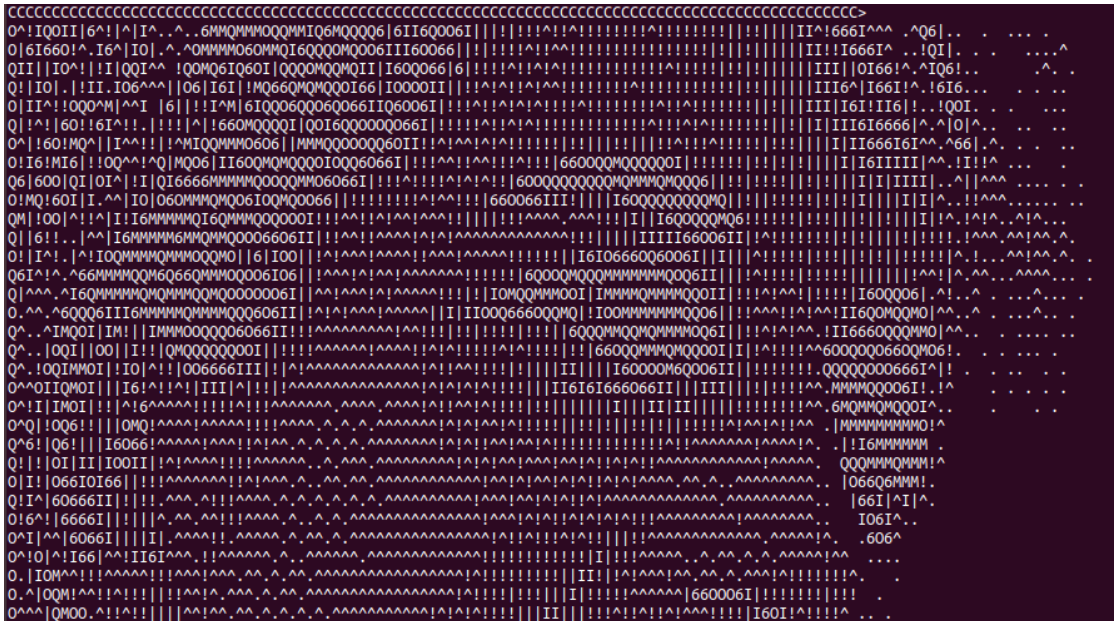
B. Writer 선호

Reader들은 중복을 허용하여 CS에 진입할 수 있다.
따라서 여러 reader 출력이 섞여있는 것이 보인다



- Reader 차례가 끝나고 writer로 전환된 부분

Writer의 출력에는 다른 writer나 reader가 침범하지 않는다.

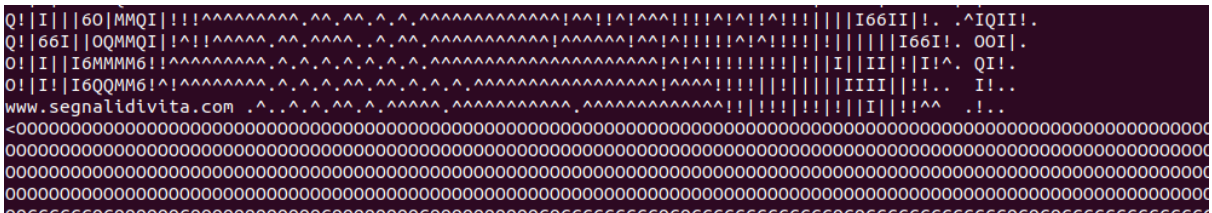


- Writer와 Writer 사이의 전환

writer와 writer는 서로 침범하지 않는다.

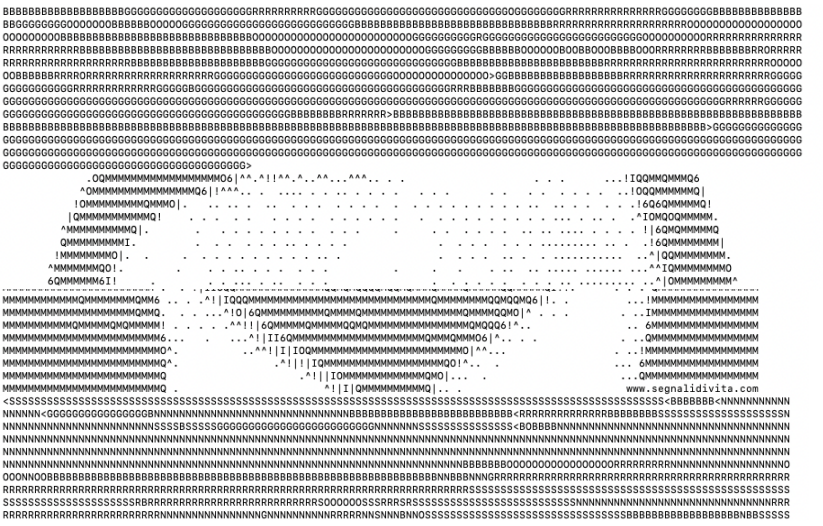
- Writer 후 Reader로 전환된 화면

writer 출력 시에는 침범하지 않지만 reader끼리는 동시 접근이 가능하다.

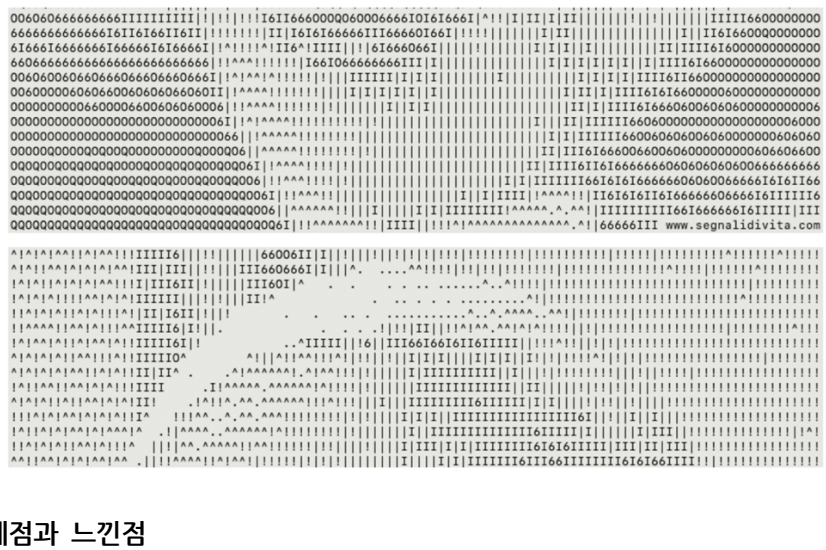


- 초반 reader가 먼저 시작된 시점을 제외하고는 writer의 출력 비중이 훨씬 높음을 알 수 있다.

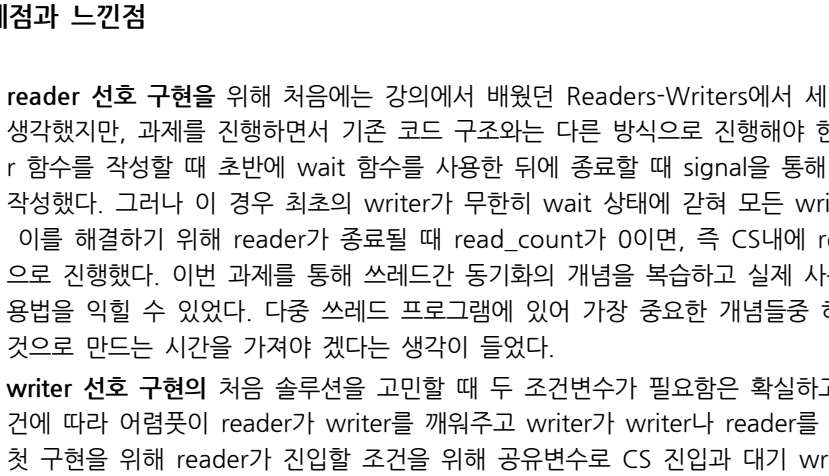
C. 공정한 reader-writer



여러 reader가 동시에 수행되는 모습과 하나의 writer가 다른 스레드 간섭없이 글씨를 출력하는 모습이다. reader 선호, writer 선호 알고리즘과 다르게 도착 순서대로 스레드가 실행되며, 먼저 와 기다리는 writer가 없으면, reader 간에 최대한 중복을 허용한다. 만약 먼저 온 writer가 있을때는 writer의 수행을 기다리고 reader가 수행된다. writer가 출력될때는 rw_lock 뮤텍스에 의해 다른 reader, writer가 간섭할 수 없다.



두개의 writer 가 서로 간섭없이 수행되는 장면이다. reader 선호, writer 선호 알고리즘과 다르게 오는 순서대로 스레드가 실행되며, writer 선호 알고리즘과 다르게 이미 먼저 온 reader 스레드를 앞질러 실행될 수 없다.



4. 문제점과 느낀점

- **reader 선호 구현을** 위해 처음에는 강의에서 배웠던 Readers-Writers에서 세마포를 condition으로만 바꾸면 된다고 생각했지만, 과제를 진행하면서 기존 코드 구조와는 다른 방식으로 진행해야 한다는 것을 깨달았다. 과제 초반에 write r 함수를 작성할 때 초반에 wait 함수를 사용한 뒤에 종료할 때 signal을 통해 다른 writer를 깨워주는 식으로 코드를 작성했다. 그러나 이 경우 최초의 writer가 무한히 wait 상태에 갇혀 모든 writer가 실행되지 못하는 현상이 발생했다. 이를 해결하기 위해 reader가 종료될 때 read_count가 0이면, 즉 CS내에 reader가 없을경우 writer를 깨워주는 식으로 진행했다. 이번 과제를 통해 쓰레드간 동기화의 개념을 복습하고 실제 사용되는 툴(mutex, condition)에 대한 사용법을 익힐 수 있었다. 다중 쓰레드 프로그램에 있어 가장 중요한 개념들중 하나인 만큼 꼼꼼히 복습하며 온전히 내것으로 만드는 시간을 가져야 겠다는 생각이 들었다.
- **writer 선호 구현의** 처음 솔루션을 고민할 때 두 조건변수가 필요함은 확실하고 bounded_buffer 알고리즘과 같이 조건에 따라 어렴풋이 reader가 writer를 깨워주고 writer가 writer나 reader를 깨워주면 되겠다고 생각을 했다. 하지만 첫 구현을 위해 reader가 진입할 조건을 위해 공유변수로 CS 진입과 대기 writer 모두를 반영한 writer_cnt만을 두고서 알고리즘을 만들어보았을 때 reader가 lock을 풀고 writer가 lock을 가지면 서로 침범하는 문제가 생겼고 여러 방법을 고민해보기 시작했다. writer가 진입할 수 있는 조건을 확인하기 위해 reader_cnt를 추가했지만 writer-prefer 조건을 만족하지 않았고 대기자와 진입자를 분리할 더 많은 공유 변수가 필요함을 느껴서 reader_wait와 reader_cnt 변수로 나눠보았다. 그 후에 'writer_cnt 〉 1 || reader_cnt 〉 0' 조건으로 writer 대기 조건을 설정했지만 항상 true가 되는 오류가 생겼고 마지막으로 writer_wait와 writer_cnt로 나누어 알고리즘을 짜고 완성했다. 주어진 상황에 따라 필요한 변수를 적절히 결정해야함을 느꼈고 구현된 조건이 하나하나 추가되더라도 부족한 다른 조건 때문에 교착상태에 빠질 수 있음을 알게 되었다.
- **공정한 reader-writer 구현을** 위해 arrive 뮤텍스 없이 공유변수인 writer_cnt 을 이용해보았다. writer_cnt 는 대기중인 writer 의 값을 저장하는 변수이다. reader 는 writer_cnt 가 0 이 아닐때만 critical section 에 들어갈 수 있다. 첫번째 reader 가 도착했을때 다음 reader 와 writer 가 차례로 도착하게 되고, reader writer 간의 도착순서와는 상관없이 writer_cnt 의 값이 1 이 되기 때문에 다음 도착한 reader 는 w_mutex 를 확인하며 대기하게 되고, 첫번째 reader 는 다음 reader 를 위해 rw_mutex 를 놓지 않으므로 다음 writer 역시 행동할 수 없게 되어 교착상태에 빠진다. 해당 상황은 도착순서에 따라 수행되지 않아 생긴 문제점으로, arrive 뮤텍스를 추가해 해결하였다. 이처럼 상황을 단순화해 mutex lock 을 설계해야함을 깨달았다.