Operating System Project #3

2017011885 김현기
2017011903 남윤재
2019040519 김태형

1. 알고리즘 구현

(1) reader 선호

```
pthread_mutex_t mutex; // 공유변수를 조회하기 위한 뮤텍스락
pthread_cond_t r_cond; // reader의 조건 변수
pthread_cond_t w_cond; // writer의 조건 변수
int r_wait = 0; // cs 진입을 기다리는 reader의 수
int r_act = 0; // cs에 진입한 reader의 수
int w_act = 0; // cs에 진입한 writer의 수 (0 또는 1)
```

문제 조건: 조건변수들과 1개의 뮤텍스락 사용

reader

cs의 상태를 확인하기 위해 뮤텍스락을 걸고, r_wait의 값을 1만큼 증가시킨다. while(w_act == 1) 루프를 이용해 reader 스레드가 실행되었을 때, 이미 cs에 실행 중인 writer가 있다면 pthread_cond_wait(&r_cond, &mutex)로 조건 변수 r_cond에서 signal이 오기를 기다린다. while 루프를 빠져나갔다면, cs에 접근한다는 의미이므로 r_wait의 값을 1 감소, r_act의 값을 1 증가시킨 후, 뮤텍스락을 풀고 임계구역에 진입하여 문자열 출력을 시작한다. 출력을 마친 후, 뮤텍스락을 걸고 r_act를 1 감소시킨다. 만약 자신이 마지막 reader이면서 현재 대기중인 reader가 없다면 writer에게 signal을 보낸 후, 뮤텍스락을 해제한다.

writer

뮤텍스락을 걸고, while(r_wait 〉 0 || w_act+r_act 〉 0) 루프를 이용하여 대기중인 reader가 있거나 cs에서 실행중인 reader나 writer가 있다면, pthread_cond_wait(&w_cond, &mutex)로 조건변수 w_cond에서 대기하며 signal이 오기를 기다린다.
while 루프를 빠져나왔다면 w_act를 1 증가시키고, 뮤텍스락을 해제한 뒤 임계구역에 진입해 사람 얼굴 출력을 시작한다.
출력을 끝마친 후, 뮤텍스락을 걸고 w_act를 1 감소시킨다. 이후, SLEEPTIME 나노초 만큼 랜덤하게 기다린 뒤, 기다리는 reader가 있다면 (r_wait 〉 0), 기다리고 있는 reader 스레드들을 pthread_cond_broadcast(&r_cond)로 깨워주고, 기다리고 있는 reader 스레드가 없다면 pthread_cond_signal(&w_cond)로 writer 스레드를 깨워준다.

(2) writer 선호

```
pthread_mutex_t mutex; // 공유 변수를 조회하기 위한 뮤텍스락
pthread_cond_t r_cond; // reader의 조건변수
pthread_cond_t w_cond; // writer의 조건변수
int r_act = 0; // cs에서 실행 중인 reader의 수
int w_wait = 0; // cs에 진입하기를 대기중인 writer의 수
int w_act = 0; // cs에서 실행 중인 writer의 수 (0 또는 1개)
```

문제 조건: 조건변수들과 1개의 뮤텍스락을 사용

reader

공유변수에 접근하여 임계구역의 상태를 체크하기 위해 뮤텍스락을 건다. 이후, while(w_act == 1 || w_wait 〉 0) 루프를 이용해 임계구역에 진입한 writer 스레드나 임계구역으로의 진입을 기다리고 있는 writer 스레드가 없을 때까지 pthread_cond_wait(&r_cond, &mutex)로 조건변수 r_cond 에서 대기한다. 이로 인해 reader 가 기다리는 writer 를 앞지를 수 없다. 대기를 마쳤다면, 임계구역에 진입하므로 r_act 를 증가시키고, 뮤텍스락을 해제한다. 임계구역에서 문자열의 출력을 마치고 난 후, 뮤텍스락을 걸고 r_act 를 감소시킨다. reader 스레드의 경우, 임계구역에서의 중복을 허용하므로 자신이 마지막 reader 임을 체크할 필요가 있다. 따라서 자신이 cs 에 접근한 마지막 reader 이면서, 대기중인 writer 가 있다면 (r_act == 0 && w_wait 〉 0), pthread_cond_signal(&w_cond)으로 대기중인 writer 스레드 하나를 깨워주고, 뮤텍스락을 해제한다.

writer

공유 변수에 접근하여 임계구역의 상태를 체크하기 위해 뮤텍스락을 건다. 대기 중이므로, w_wait 의 값을 증가시킨다. 이후, while(w_act == 1 || r_act 〉 0) 루프를 이용해 임계구역에 접근한 reader 나 writer 가 없을 때까지 pthread_cond_wait(&w_cond, &mutex)로 조건변수 w_cond 에서 대기한다. 대기를 마쳤다면, 임계구역에 접근하므로 w_wait 의 값은 감소, w_act 의 값은 증가시키고, 뮤텍스락을 해제한다.

임계구역에서 얼굴 이미지 출력을 마친 뒤, 뮤텍스락을 걸고, w_act 의 값을 감소시킨다. 이후, 기다리고 있는 writer 가 있다면 (w_wait 〉 0), pthread_cond_signal(&w_cond)으로 깨워준다. 기다리고 있는 writer 가 없다면, pthread_cond_broadcast(&r_cond)으로 기다리고 있는 reader 들을 깨워주고, 뮤텍스락을 해제한다.

(3) 공정한 reader-writer

```
pthread_mutex_t mutex; //공유 변수 통제를 위한 lock
pthread_mutex_t wrt; //writer중복을 막기 위한 lock
pthread_mutex_t fair; //reader & writer의 starvation을 막기 위한 lock
int readcnt = 0; //중복된 reader의 갯수 변수
```

문제 조건: 뮤텍스락과 공유변수만을 사용하여 구현

공정한 reader-writer를 구현하기 위해서 공유 변수 readcnt를 통제하기 위한 mutex와 writer의 중복을 막는 wrt, reader와 writer의 순서를 만들기 위한 fair 총 3개의 뮤텍스락을 생성했다.

작동 원리: writer와 reader 둘 중 누구나 fair락을 획득할 수 있다. 두 함수 도입에 fair wait을 사용했으므로 fair락을 획득하는 순서대로 프로세스가 진행된다.

reader

reader가 fair를 획득했다면 readcnt의 값을 확인한다. readcnt는 공유변수이므로 mutex를 걸고 확인해야 한다. readcnt가 0이라면 자신이 중복된 reader가 아니므로 writer의 출입을 막고 자신이 곧 실행할 것이므로 readcnt를 하나 늘려준 후 fair와 mutex 둘다 풀어준다. reader 함수는 lock을 걸지 않으므로 reader의 중복은 허용된다. 임계구역에 진입해 문자열 출력을 완료하고, readcnt가 0이라면 더이상 중복해서 실행할 reader가 존재하지 않는 것이므로, wrt를 풀어 writer의 출입을 허용한다. 출입 혹은 중복을 허용한다고해서 이후 실행을 강제하진 않는다. 즉, writer의 출입을 허용하지만 reader가 fair를 획득했다면 reader를 실행한다.

writer

writer가 fair를 획득했다면 writer는 wrt의 획득을 기다린다. 이 과정은 3개의 의미를 가진다.
첫째, 앞에서 실행되고 있는 reader들의 종료를 기다린다. (reader와 writer는 중복될 수 없기 때문) 둘째, 새로운 reader의 출입을 막는다. (기다리는 중에 새로운 reader가 추가되어 먼저 실행된다면 공정하지 않기 때문) 셋째, writer의 중복을 막는다. (writer의 중복은 허용하지 않기 때문)
이후, wrt를 획득했다면 fair를 풀어주어 reader와 writer 출입을 모두 허용한다. (둘 중 fair를 획득하는 프로세스가 실행됨) 해당 과정은, reader와 writer의 순서를 결정해준다.
임계구역에서 사람 얼굴의 출력을 하고난 뒤, wrt를 풀어주어 reader와 writer의 실행을 허용한다. 해당 과정은 reader와 writer의 실행을 결정해준다.

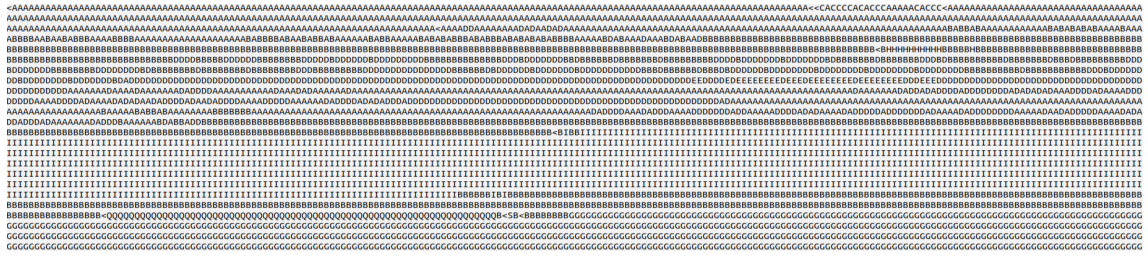
2. 컴파일 과정

```
mac@Macui-MacBookAir task3 % ls
fair_reader_writer.c
reader_prefer.c
writer_prefer.c
~$체 제  과 제 3.docx
운 영 체 제  과 제 3.docx
mac@Macui-MacBookAir task3 % gcc -o reader reader_prefer.c -lpthread
mac@Macui-MacBookAir task3 % gcc -o writer writer_prefer.c -lpthread
mac@Macui-MacBookAir task3 % gcc -o fair fair_reader_writer.c -lpthread
mac@Macui-MacBookAir task3 % ls
fair
fair_reader_writer.c
reader
reader_prefer.c
writer
writer_prefer.c
~$체 제  과 제 3.docx
운 영 체 제  과 제 3.docx
```

lpthread 옵션을 주어 파일들을 컴파일 한 결과, 별다른 오류 메시지 없이 각각 reader, writer, fair 파일들이 생성된 것을 확인할 수 있다.

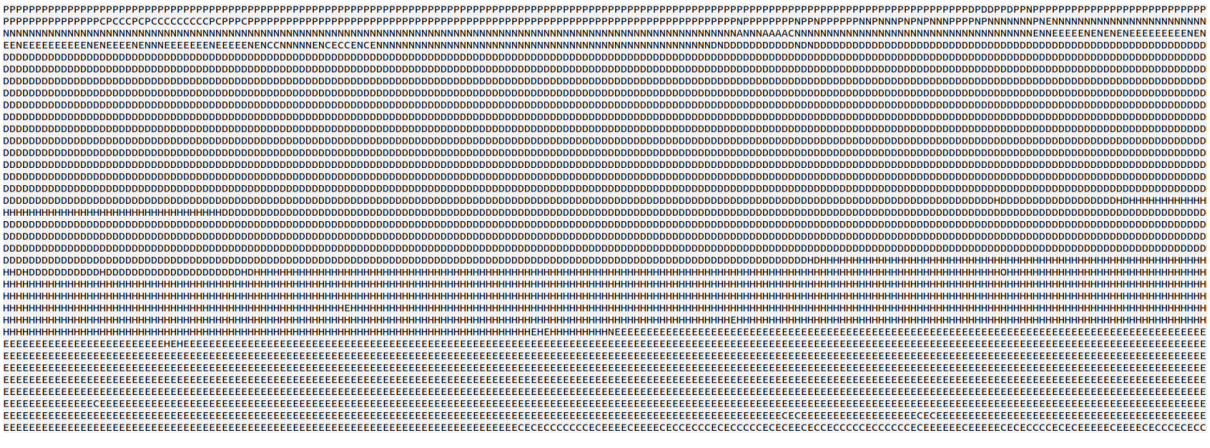
3. 실행 결과물 주요 장면 및 설명

1) reader 선호



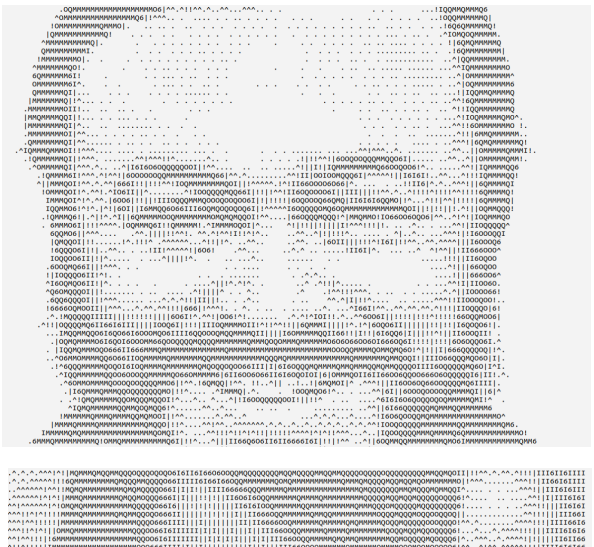
메인함수에서 reader 스레드들이 먼저 생성되어, reader 들이 문자열 출력을 시작한다.

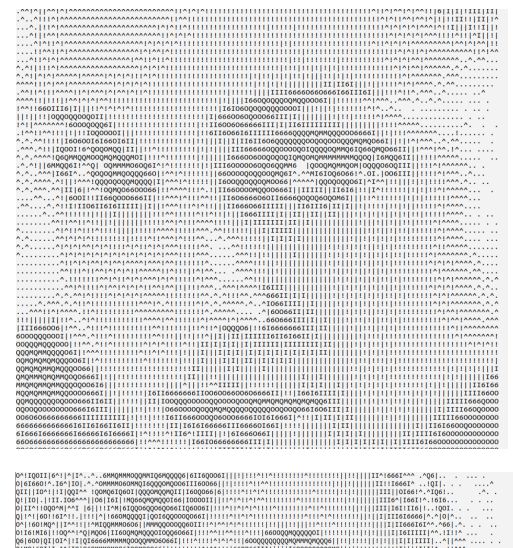
reader 는 임계구역에서 중복을 허용하므로, 문자열의 출력이 겹쳐져서 나타나는 것을 확인할 수 있다.

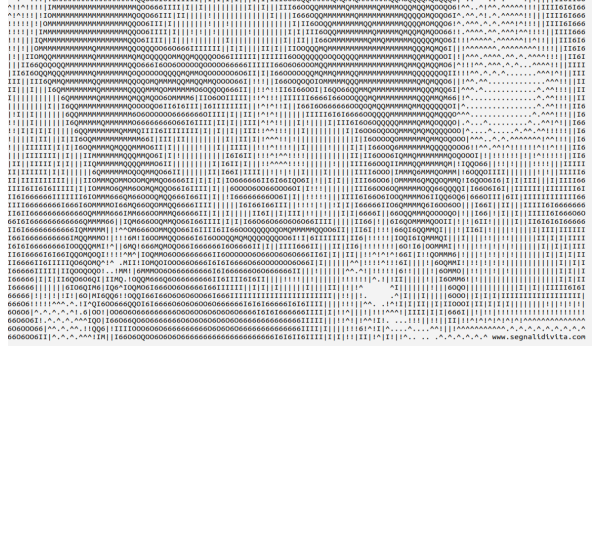


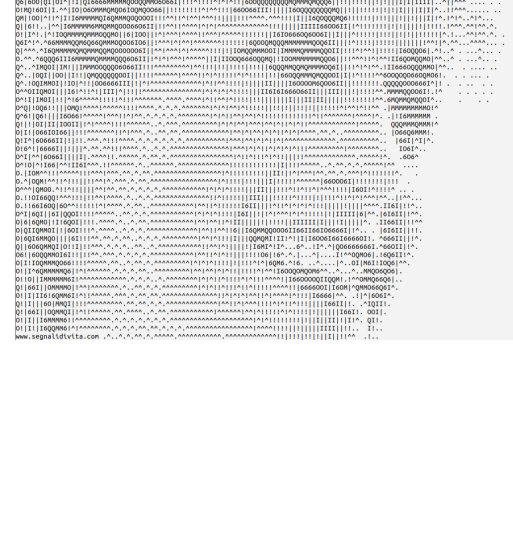
reader 선호 알고리즘은 늦게 온 reader 가 기다리고 있는 writer 를 앞지를 수 있기 때문에 writer 의 굶주림이 나타난다. 따라서, writer 는 출력을 계속 하지 못하고, reader 의 출력이 계속해서 나타난다.

reader 들이 모두 종료되고나서야 writer 의 출력이 나타난다. 아래와 같이 writer 들의 출력이 나오고 프로그램이 종료된다.



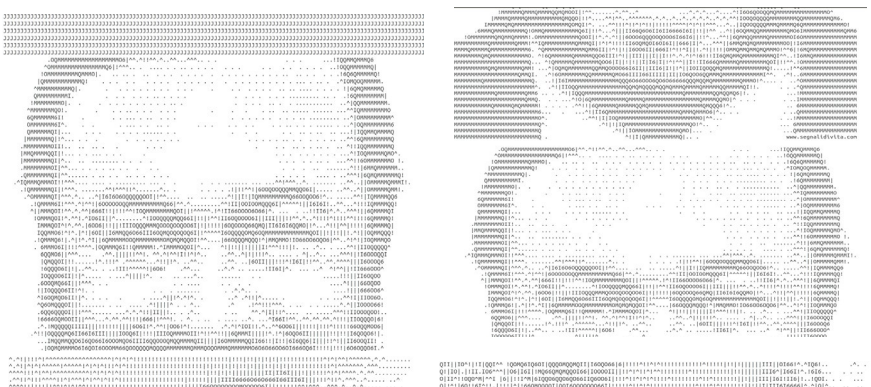




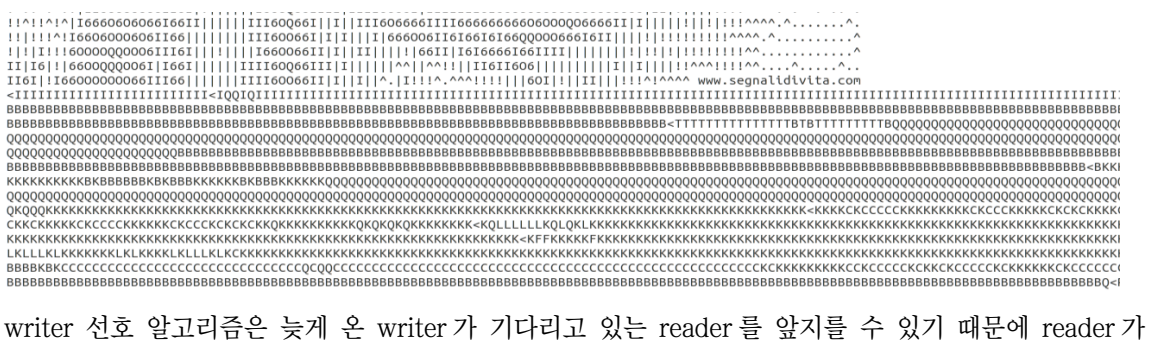


2) writer 선호

메인 함수에서 reader 스레드들이 먼저 생성되므로, reader 들이 출력을 먼저 시작한다. reader 들끼리는 임계구역에서의 중복을 허용하므로, 여러 reader 가 동시에 임계구역에 진입하면서 문자열이 섞여 출력되는 경우가 발생한다.



이후, writer 스레드들이 계속하여 사람 얼굴을 출력한다. writer 는 임계구역 내에서의 중복이 허용되지 않으므로, 깨끗한 얼굴 출력이 나타난다.



writer 선호 알고리즘은 늦게 온 writer 가 기다리고 있는 reader 를 앞지를 수 있기 때문에 reader 가 출력 기회를 갖지 못하는 굶주림 현상이 발생한다. 따라서 reader 는 출력을 계속 하지 못하다가, 마지막에 이르러서야 출력을 수행한다. reader 의 마지막 부근에서의 출력에서도 임계구역에서의 중복이 허용되어 문자가 겹쳐져서 출력되는 경우가 발생한다.

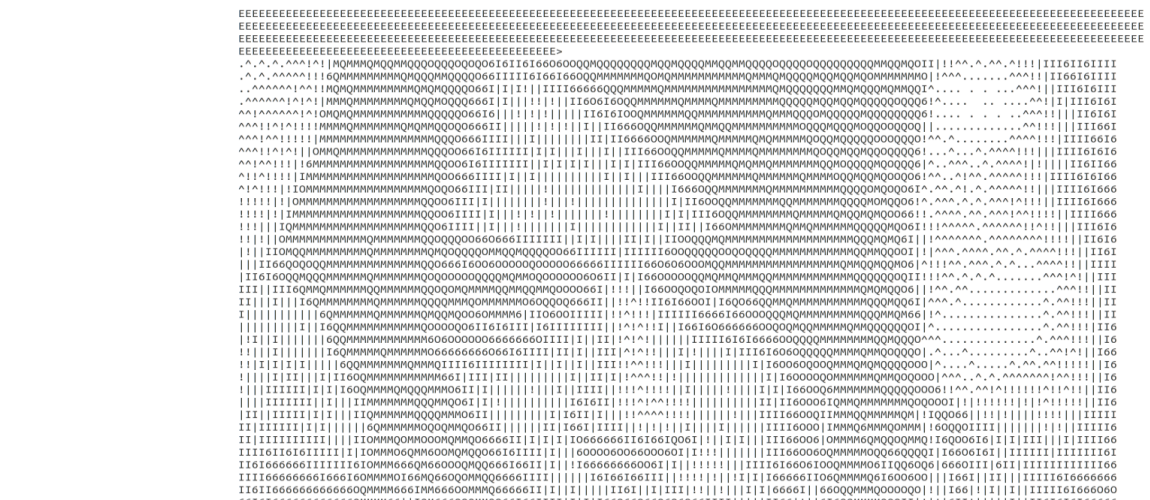
3) 공정한 reader-writer

공정한 알고리즘도 메인함수에서 reader 스레드를 먼저 실행시키므로 reader 의 출력이 먼저 시작된다.

마찬가지로 reader 는 임계구역 내 중복을 허용하므로 알파벳이 섞여 나오는 것을 확인할 수 있다.

이후, writer 가 들어오면 한쪽에 치우치지 않고, 공정하게 writer 의 사람 얼굴이 출력된다.

writer 가 출력을 하고난 뒤, reader 가 출력을 하고 이후 다시 writer 가 출력하는 것을 확인할 수 있다.



reader가 출력을 하고 난 뒤, writer가 출력을 하고 이후 다시 reader가 출력하는 것을 확인할 수 있다. 앞선 출력 결과물들을 통해 reader와 writer 어느 한 쪽이 굶주리지 않고, 공정하게 출력함을 확인할 수 있다.

4) 결과물 정리 및 차이점

공통점:

세 알고리즘 모두 reader들끼리는 임계 구역 내의 중복이 허용되어, 알파벳 문자열들이 섞여서 출력된 것을 확인할 수 있었고, writer는 임계 구역 내의 중복이 일절 허용되지 않아 깨끗한 사람 얼굴이 출력된 것을 확인할 수 있다.

차이점:

- reader 선호 알고리즘에서는 늦게 도착한 reader가 기다리고 있는 writer를 앞지를 수 있기 때문에 writer의 굶주림이 나타나 writer가 계속 출력을 하지 못하고, reader들의 출력만 나타나다가 reader들이 전부 종료되고나서야 writer들이 사람 얼굴을 출력함을 확인할 수 있다.

- writer 선호 알고리즘에서는 늦게 도착한 writer가 기다리고 있는 reader를 앞지를 수 있기 때문에 reader의 굶주림이 나타나 reader가 출력을 하지 못하고, writer들의 출력만 나타나다가 writer들이 전부 종료되고나서야 reader들이 문자열을 출력함을 확인할 수 있다.

- 공정한 reader-writer 알고리즘에서는 늦게 도착한 reader, writer들이 기다리고 있는 reader, writer를 앞지르지 못하므로, 어느 한쪽이 굶주리지 않고 들어온 순서대로 서로 번갈아가면서 출력이 나타난다.

4. 과제를 수행하면서 경험한 문제점과 느낀점

2017011903 남윤재: 해당 과제를 처음에는 readcnt와 writecnt 공유 변수 2개를 만들어 조건문으로 확인하며 현재 들어온 프로세스가 어느 시점인지를 확인하면서 풀어보았는데, 그 과정이 너무 복잡하고 경우의 수가 너무 많았다. cnt값으로 lock을 걸면 이전에 실행되고 있는 reader가 종료를 하기전에 writer를 실행하는 경우도 있었고 데드락에 빠지는 경우도 많았다. 그래서 lock을 하나 더 만들어 lock을 획득하는 순서대로 프로세스를 실행시키면 어떨까라는 의문에서 코드를 작성해 보았고 잘 작동하였다. 하지만 lock을 획득하는 순서대로 프로세스를 실행시키기 때문에 프로세스의 실행순서를 보장하진 않는다. reader와 writer가 lock을 획득하는 것은 무작위임으로 실행의 순서와 관계없이 누가 lock을 획득했느냐에 따라순서가 결정된다. 교수님이 공유한 원자변수를 구현하거나 순서를 운에 맡겨도 된다고 하셔서 일단은 운에 맡기는 편으로 작성했다. 나중에 순서대로 실행하는 진짜로 공정한 reader-writer코드를 작성해보고싶다.

2019040519 김태형: writer 선호 알고리즘을 맡아 구현하면서, writer들이 같은 얼굴만 출력하는 결과물을 보고, 이상하다는 생각이 들었다. 하지만, 교수님께서 수업시간에 해당 현상이 가능하다고 말씀해주시는 것을 듣고, 잘못된 게 아니라는 것을 알게 되었다. 이번 과제에서는 writer의 출력 순서에 대한 요구는 없었지만, 원자 변수를 사용하여 선착순으로 구현이 가능하다는 것을 알게 된 후, 개인적으로 구현해보고 싶다는 생각이 들었다.

2017011885 김현기: reader 선호 알고리즘을 맡았다. Reader 선호에 대해서는 수업시간에 다룬적이 있기 때문에 조건변수를 사용하여 교수님이 수업시간에 라이브 코딩하신 결과물을 변형해보았지만 공유변수를 사용하는 것에 익숙치 않아서 쉽지 않았다. 또한 교수님이 강조하신 조건을 확인하는 부분을 while이 아닌 if로 하여 한참 고민하기도 하였다. 팀원들의 도움을 받아서 결국 원하는 결과물을 만들었고 전반적인 레포트 작성을 하면서 reader 선호뿐 아니라 나머지 2개의 알고리즘에 대해서도 더 잘 이해하였고 다음과제는 혼자 생각해볼 시간을 더 많이 할애하여 팀원들의 도움을 받기 전에 더 잘 짜봐야겠다고 느꼈다.