**[reader\_prefer\_cond] – 조건 : 조건변수 + 1개의 mutex lock**

pthread\_mutex\_t mutex; *// 조건변수 용 락*

pthread\_cond\_t cond; *//조건변수*

int wrt, read; // writer, reader 수 체크

**- Reader**

while (alive) {

**pthread\_mutex\_lock**(&mutex);

        while(wrt)

**pthread\_cond\_wait**(&cond, &mutex);

        read++;

**pthread\_mutex\_unlock**(&mutex);  
 */\* Critical Section \*/*

**pthread\_mutex\_lock**(&mutex);

    read--;

**pthread\_cond\_signal**(&cond);

**pthread\_mutex\_unlock**(&mutex);

**- Writer**

while (alive) {

**pthread\_mutex\_lock**(&mutex);

        while(read || wrt)

**pthread\_cond\_wait**(&cond, &mutex);

        wrt++;

**pthread\_mutex\_unlock**(&mutex);

*/\* Critical Section \*/*

**pthread\_mutex\_lock**(&mutex);

    wrt--;

**pthread\_cond\_signal**(&cond);

**pthread\_mutex\_unlock**(&mutex);

reader는 현재 writing 중이라면 대기합니다. 이 대기는 writer가 끝날 때 signal을 보내는 것으로 해결합니다.

writing 중이 아니라면 reader는 읽고 있는 reader의 수를 올린 후 Critical Section에 진입합니다. 이 때 reader는 중복으로 들어가서 작업할 수 있습니다.

writer는 reader나 writer가 있다면 대기합니다. 이때 reading 중이며 reader가 추가로 들어온 경우 추가로 들어온 reader는 대기중인 writer를 앞질러 read할 수 있습니다.

이 방식에서 reading 중일 때는 reader 선호로 reader가 앞지를 수 있지만, writing 중인 상태이면 reader와 writer가 같이 조건변수로 대기중이기에, signal에 전달에 따라 writer나 reader가 선택될 수 있을 것 같습니다.

**[writer\_prefer\_mutex] – 조건 : 오직 mutex lock**

*\* x의 역할 : writing 중이면 reader을 멈춘다*

*\* read\_mutex의 역할 : reader 내부 lock/unlock*

*\* y의 역할 : reading 중이면 writer를 멈추고, writer가 중복으로 못들어가게 한다.*

*\* write\_mutex의 역할 : writer 내부 lock/unlock*

pthread\_mutex\_t x, y, read\_mutex, write\_mutex;

int reading, writing;

텍스트, 폰트, 친필, 스크린샷이(가) 표시된 사진

자동 생성된 설명

reader, writer의 각각 자기 내부를 관리하는 mutex lock 1개와 상대 쪽에 영향을 주는 mutex lock 을 둬 총 2\*2 = 4개의 뮤텍스 락을 사용합니다.

reader는 reading중이라면 writer가 CS에 진입하지 못하도록 막습니다. 그 후 CS를 중복으로 들어갈 수 있습니다. 단, 여기서 기다리고 있는 writer가 있다면 reader는 lock(x)가 되어 추가 진입을 못하고 writer가 앞지를 수 있습니다.

writer는 총 3가지 파트로 나뉘는데 기다리는 writer가 있다면 reader의 추가 진입을 막는 1번 파트, 다른 writer가 못오게 막으며 현재 진행 중이던 reading을 막는 2번 파트, 더 이상 writer가 없으면 다시 reader 진입을 허용하는 3번 파트로 되어있습니다.

**[writer\_prefer\_cond] – 조건 : 조건변수 + 1개의 mutex lock**

*\* mutex : condition variable을 위한 lock*

*\* reader\_cond : writing 중일 때 대기용 조건 변수*

*\* writer\_cond : reading 중이거나 다른 writer가 있을 때 대기용 조건 변수*

*\* reader\_count : reading 중인 reader수*

*\* waiting\_writers : 기다리고있는 writers 수. reader가 앞지르지 못하게 체크*

*\* writer\_active : 현재 writing 중인지 체크*

pthread\_mutex\_t mutex;

pthread\_cond\_t reader\_cond, writer\_cond;

int reader\_count = 0;

int waiting\_writers = 0;

**bool** writer\_active = **false**;

**- Reader**

**pthread\_mutex\_lock**(&mutex);

        while(writer\_active || waiting\_writers > 0)

**pthread\_cond\_wait**(&reader\_cond, &mutex);

        reader\_count++;

**pthread\_mutex\_unlock**(&mutex);

*/\*Begin Critical Section\*/*

**pthread\_mutex\_lock**(&mutex);

    reader\_count--;

    if(reader\_count == 0)

**pthread\_cond\_signal**(&writer\_cond);

**pthread\_mutex\_unlock**(&mutex);

**- Writer**

**pthread\_mutex\_lock**(&mutex);

        waiting\_writers++;

        while(reader\_count > 0 || writer\_active)

**pthread\_cond\_wait**(&writer\_cond, &mutex);

        waiting\_writers--;

        writer\_active = **true**;

**pthread\_mutex\_unlock**(&mutex);

*/\*Begin Critical Section\*/*

**pthread\_mutex\_lock**(&mutex);

    writer\_active = **false**;

**pthread\_cond\_broadcast**(&reader\_cond);

**pthread\_cond\_signal**(&writer\_cond);

**pthread\_mutex\_unlock**(&mutex);

reader는 writing 중이거나 기다리고 있는 writer가 있다면 더 이상 reader를 받지않고 대기합니다.

writer는 reading 중이거나 writing 중이면 일단 대기합니다. 여기서 writing이 끝나면 기다리고 있는 writer 조건 변수에게 signal을 주고, reader는 중복 진입이 가능하기에 broadcast로 모든 reader 조건 변수를 깨웁니다. 추가적인 방법으로 writer를 체크해 writer가 있으면 signal만 하는 방식도 괜찮은 것 같습니다.

**[fair\_reader\_writer\_mutex] – 조건 : 오직 mutex lock**

*\* in의 목적 : reader, writer 진입 제어*

*\* mutex의 목적 : reader간 분리*

*\* wrt의 목적 : writer간 분리*

pthread\_mutex\_t in, mutex, wrt;

int read\_count = 0;

**-Reader**

**pthread\_mutex\_lock**(&in);

**pthread\_mutex\_lock**(&mutex);

        read\_count++;

        if(read\_count == 1)

**pthread\_mutex\_lock**(&wrt);

**pthread\_mutex\_unlock**(&mutex);

**pthread\_mutex\_unlock**(&in);

*/\*Begin Critical Section\*/*

**pthread\_mutex\_lock**(&mutex);

    read\_count--;

if(read\_count == 0)

**pthread\_mutex\_unlock**(&wrt);

**pthread\_mutex\_unlock**(&mutex);

**-Writer**

**pthread\_mutex\_lock**(&in);

**pthread\_mutex\_lock**(&wrt);

*/\* Begin Critical Section\*/*

**pthread\_mutex\_unlock**(&wrt);

**pthread\_mutex\_unlock**(&in);

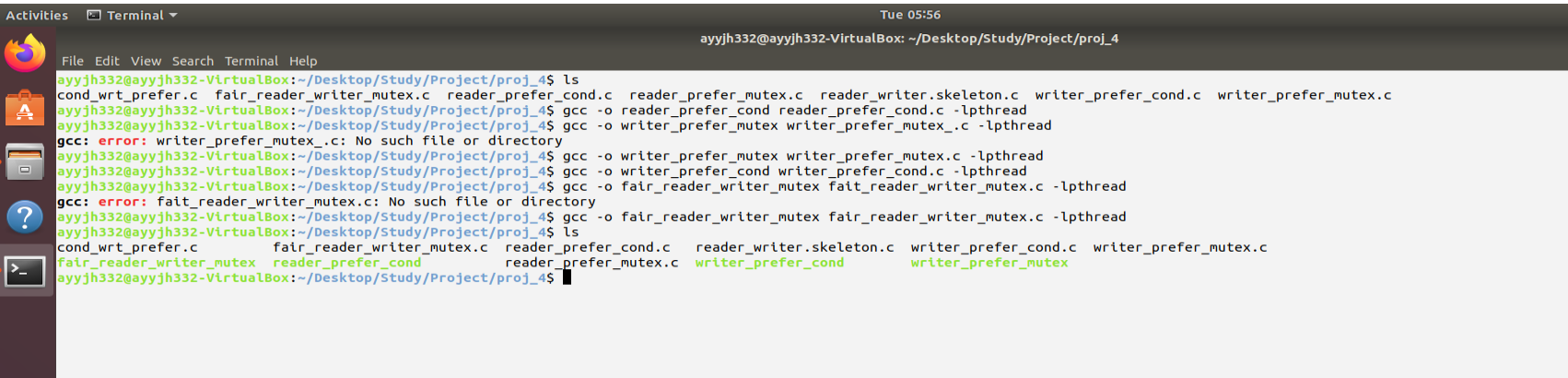
reader는 in, mutex lock으로 감싼 후 만약 reader가 진입했다면 writer를 막습니다. 이후 reader는 CS에서 중복이 가능한 상태로 reading을 진행합니다.

writer는 in을 lock하여 추가 reader에 진입을 막고, 만약 reading 중이면 reading이 끝나 wrt lock이 풀릴 때 까지 대기합니다.

reader와 writer는 in을 통해 누가 lock을 잡냐에 따라 진행을 하며, reader가 in을 잡아서 진입했다 해도 다른 기다리는 reader들이 무조건 앞지를 수 있는게 아닌 in을 잡는 쪽에 따라 결정됩니다.

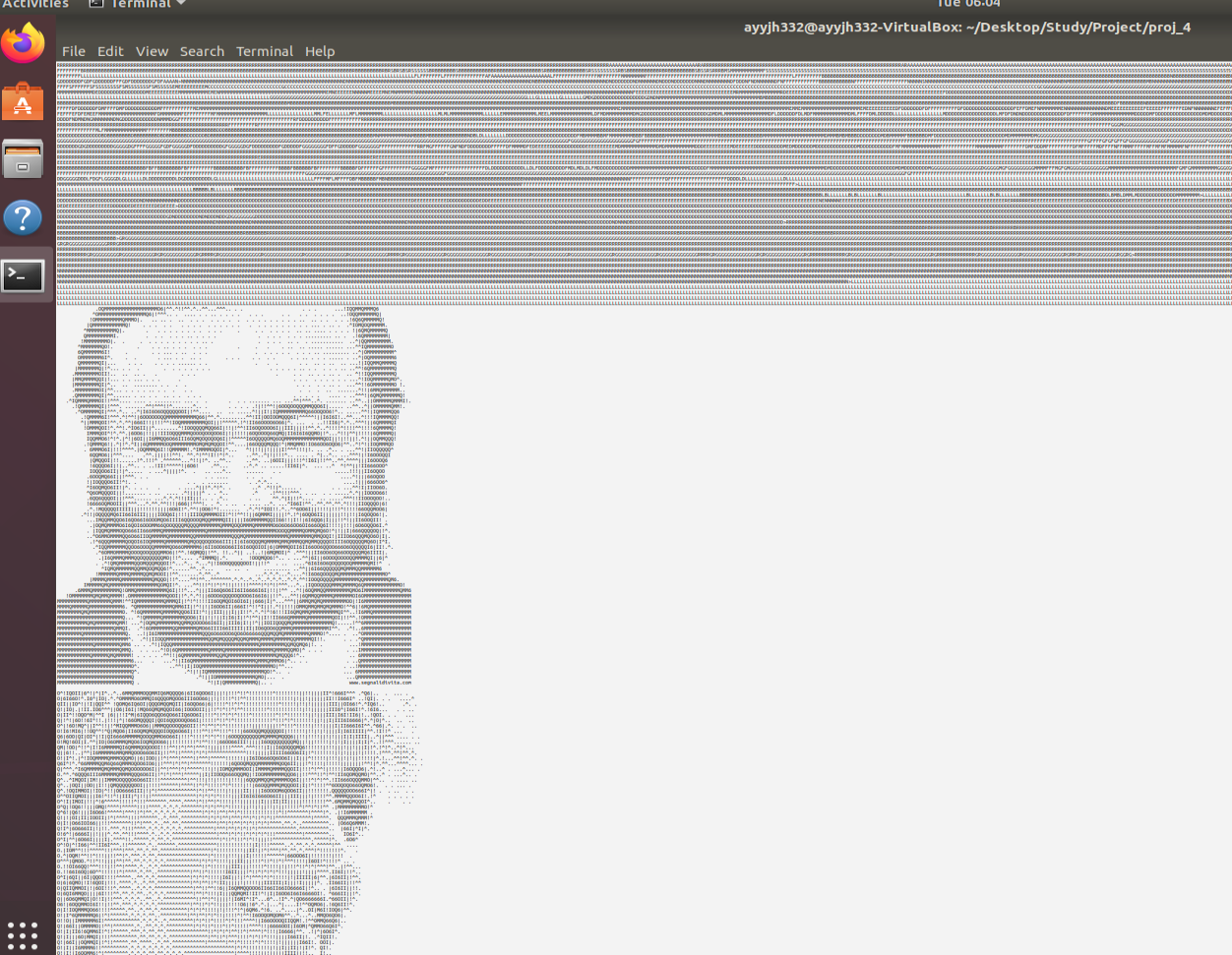
reader와 writer가 다른 한 쪽을 무조건 앞지르지 못하기 때문에 공정한 방식입니다.

**-컴파일 과정**



**- 실행 결과물 상세 설명 및 캡쳐**

**[reader\_prefer\_cond]**

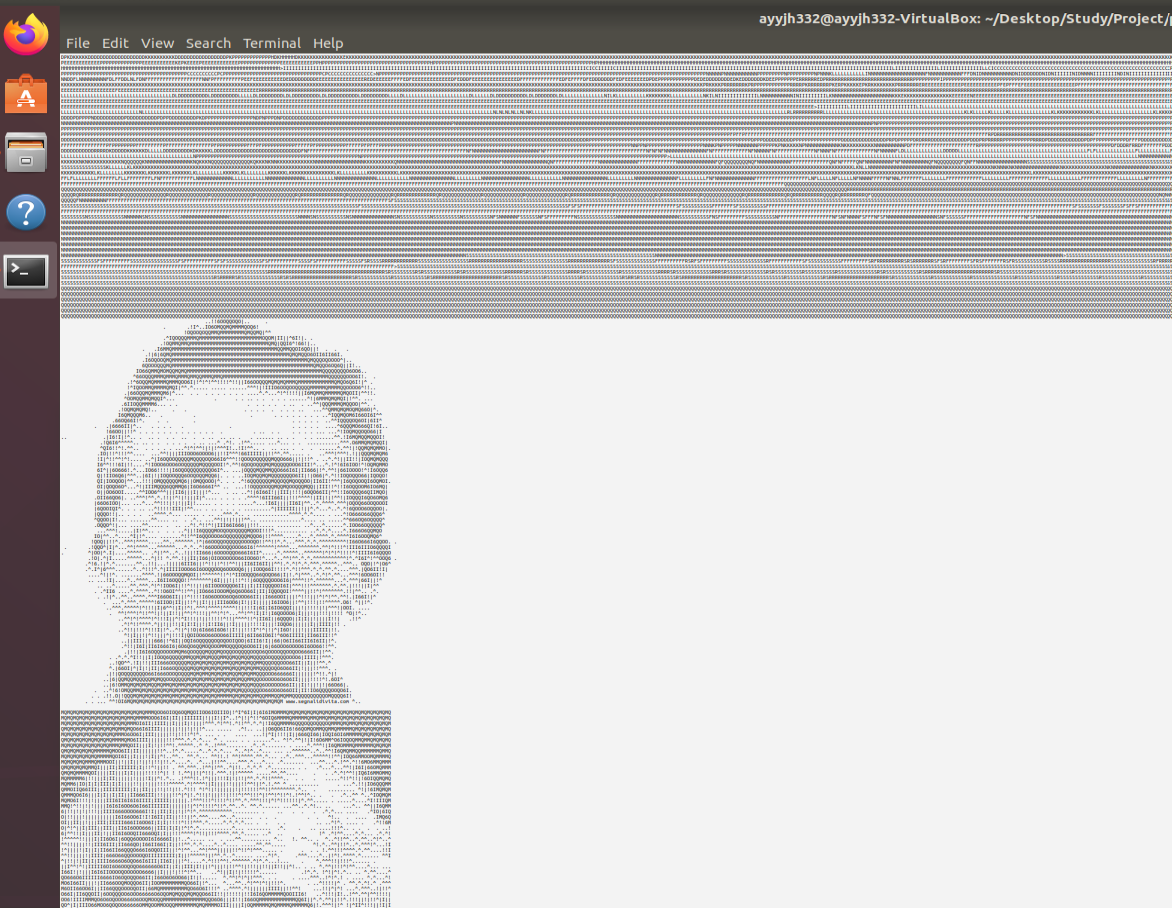


reader 선호 방식은 새로 들어온 reader가 먼저 들어와 대기 중인 writer를 앞지를 수 있기 때문에 코드에서 먼저 생성된 reader스레드를 진행 후 writer가 추가되도 reader가 앞질러 일을 한 후 writer코드에 사진을 마지막에 다 출력하게 됩니다.

현재 예제코드에서는 reader가 계속 alive 동안 일을 진행하고, alive가 끝난후 reader가 while을 탈출하며 스레드가 종료되고, 그 후 writer가 조건 변수 전달을 받아 writer 스레드에서 사진을 1개씩 출력하고 끝난 것으로 생각됩니다.

그렇기에 해당 예제의 출력 결과에서는 reader만이 일을 계속 진행했으며 reader선호로 writer를 제쳐 일을 했으며, 모든게 끝난 후 남은 writer 잔재가 사진을 1개씩 출력하고 끝난 것으로 보입니다.

**[writer\_prefer\_mutex]**

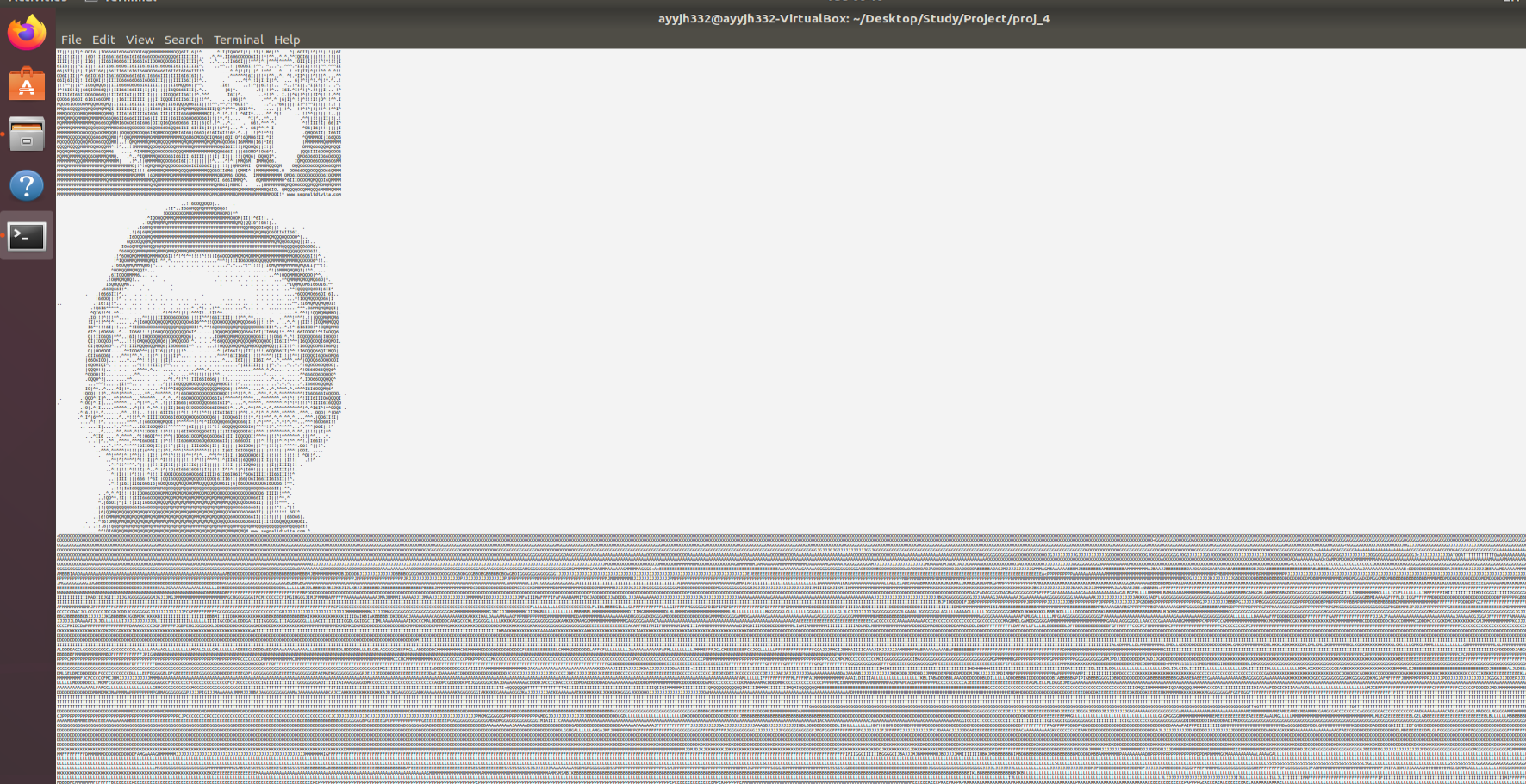


첫 시작부분은 먼저 생성된 reader가 진행되다가 writer가 reader가 앞지르지 못하게 막은 후 writing을 진행합니다.

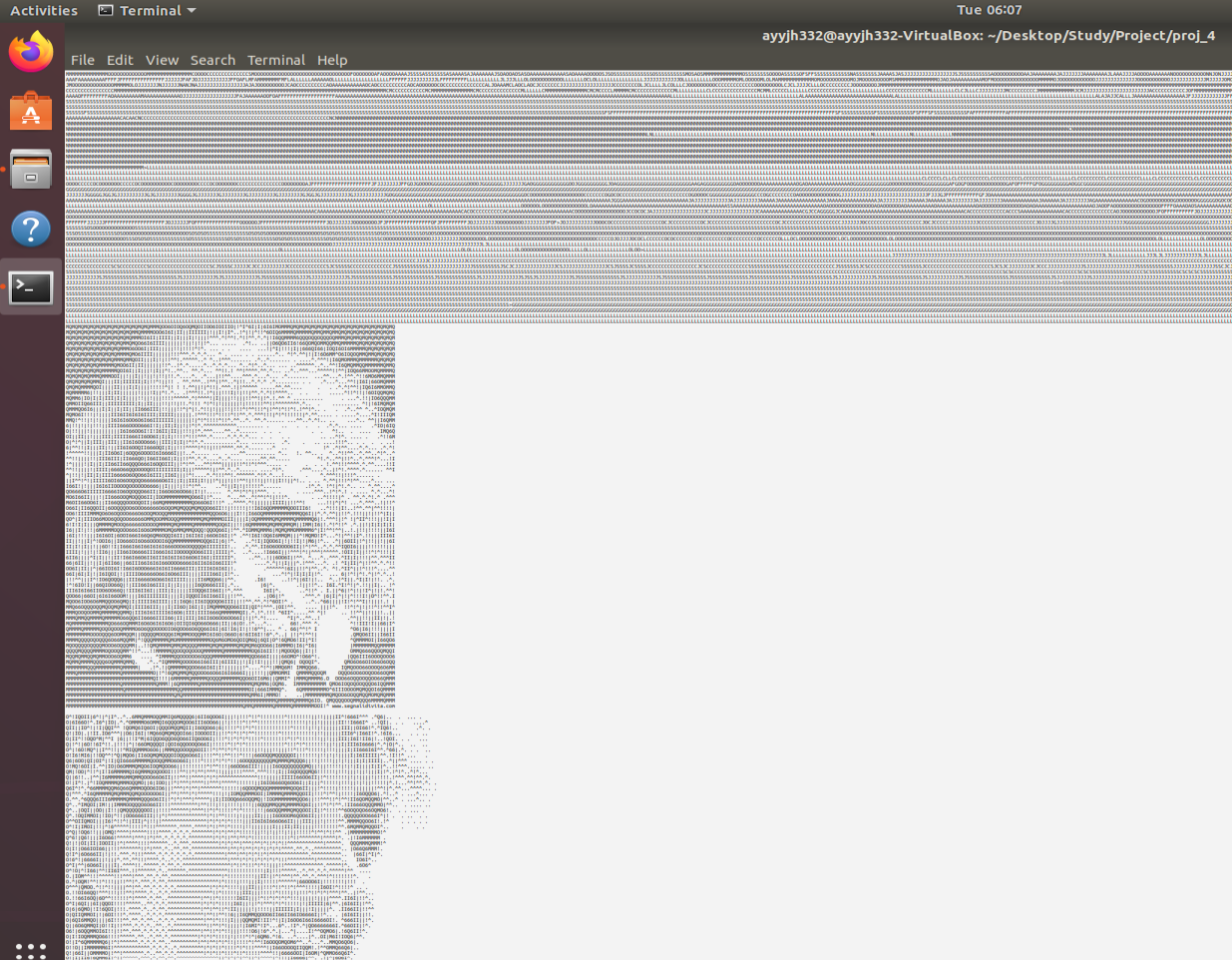
텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

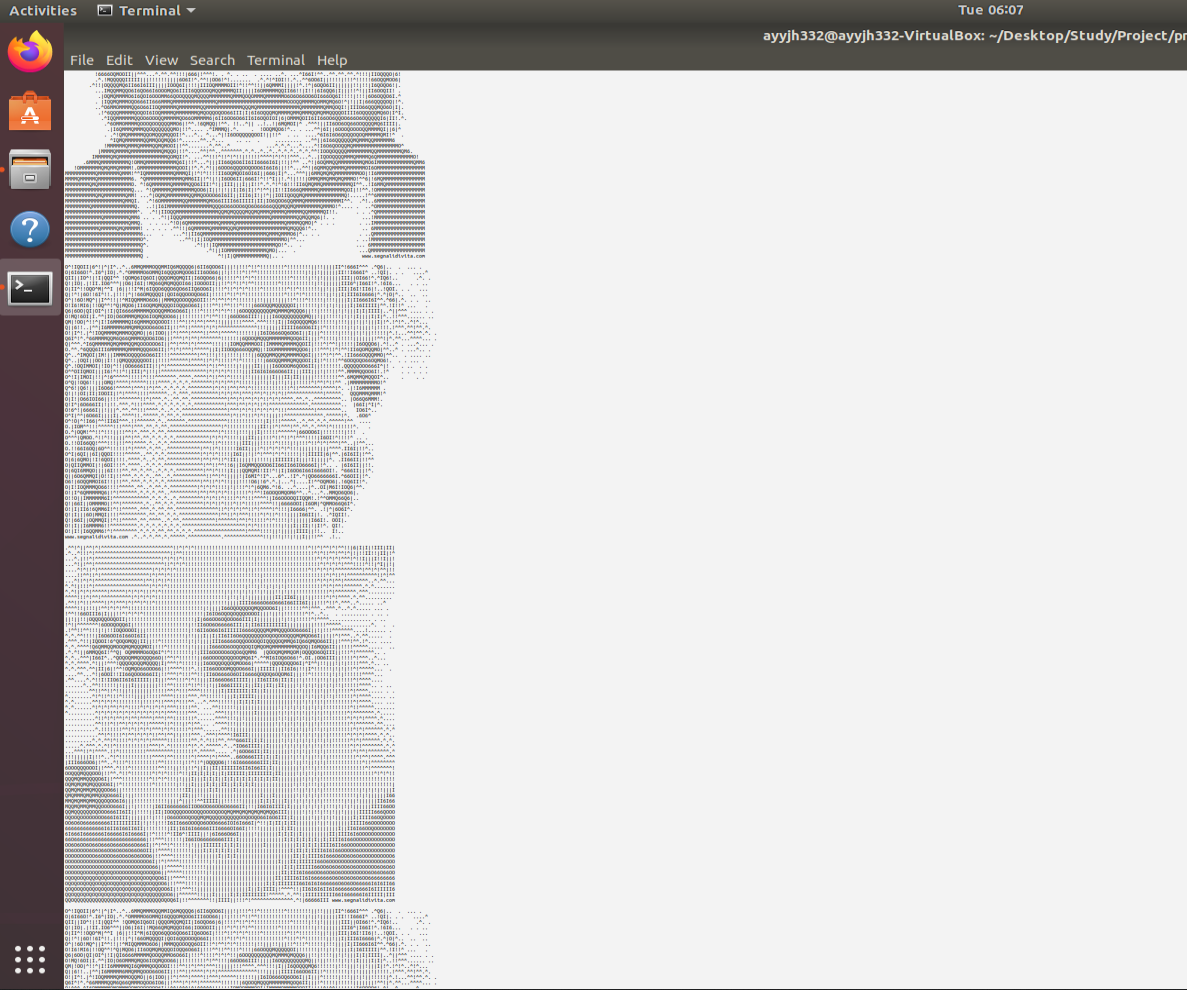
자동 생성된 설명

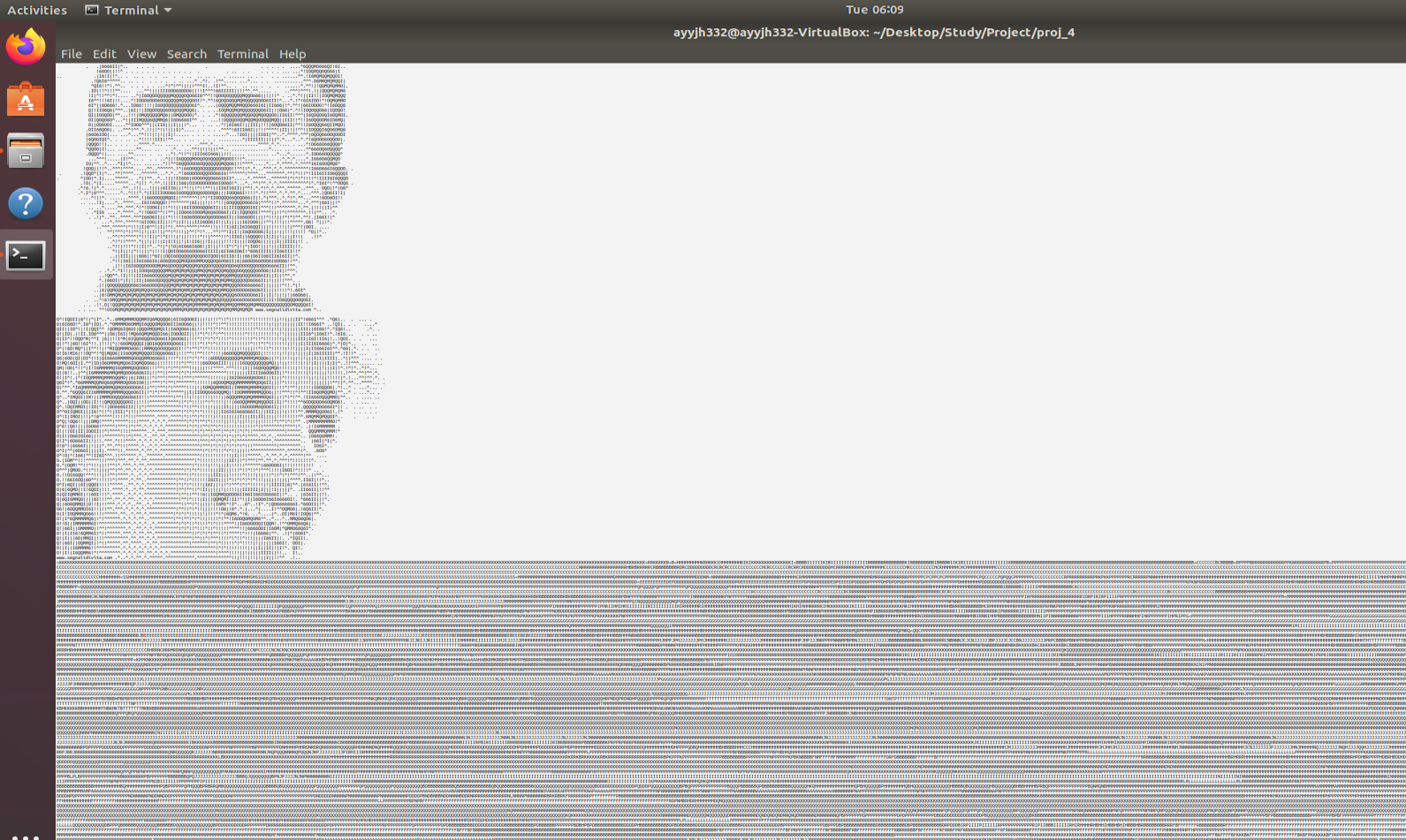
그 후 writer는 계속 출력을 진행합니다. reader는 writer를 앞지를 수 없고 writing 중에는 대기하기에 알파벳은 나오지 않습니다. writer가 없다면 reader는 중간중간 알파벳을 출력합니다



그 후 alive가 false가 되며 남은 writer는 끝나고, reader측 lock을 풀며 남은 reader가 남은 부분을 출력합니다.

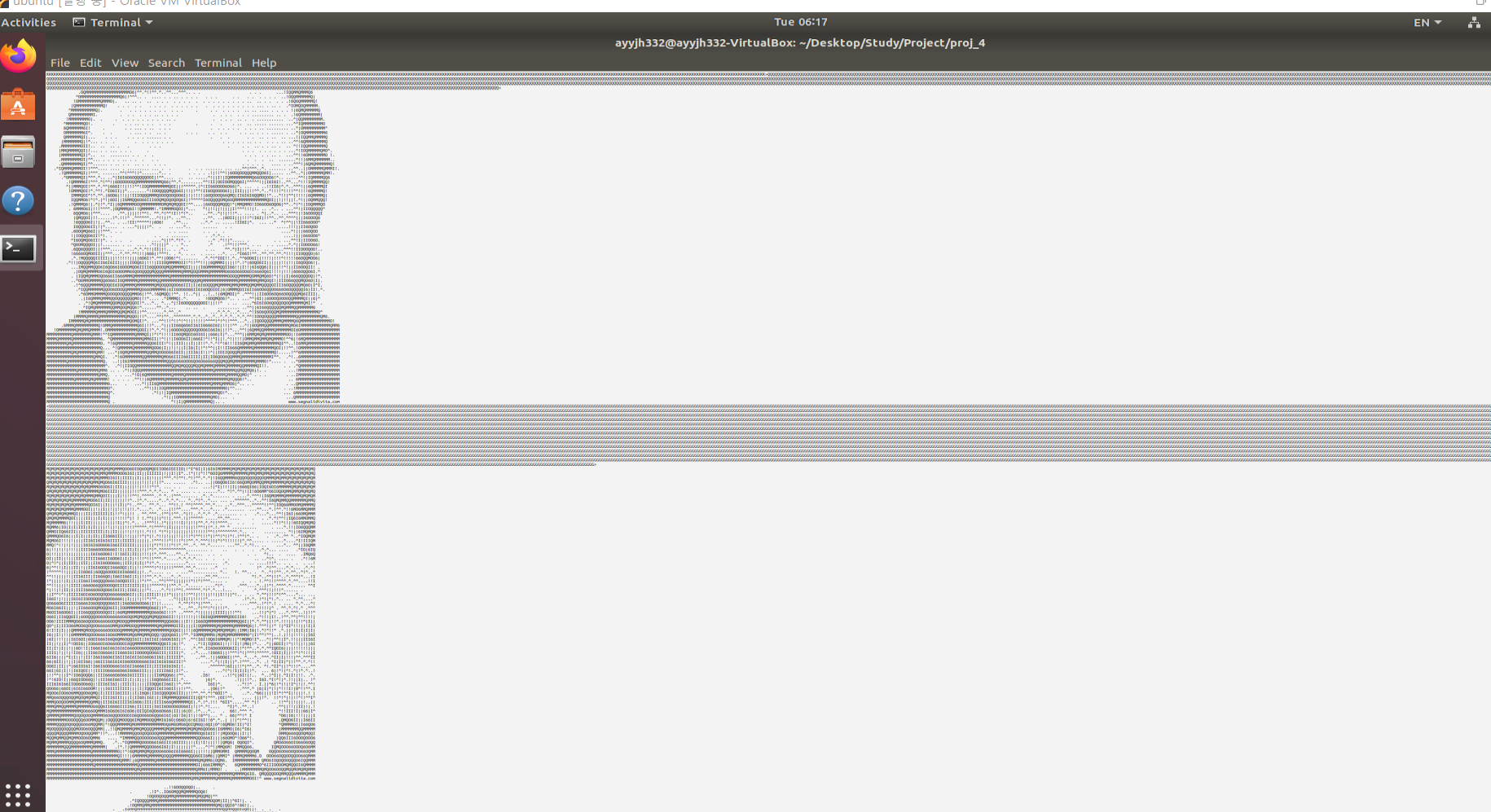
**[writer\_prefer\_cond]**





Mutex lock이 아닌 조건 변수 방법도 출력은 유사하게 나옵니다. 내부 구현의 차이가 속도가 달라서 그런 것인지 mutex 방법의 경우가 writer가 매우 빠르게 진행되어 writing을 잘 내려놓지 않는 것 같았습니다.

**[Fair\_reader\_writer\_mutex]**



공정한 reader writer 방법은 reader의 알파벳 출력과 writer의 사진 출력이 당시 lock을 누가 잡았냐에 따라서 번갈아가며 나옵니다. 이때, reader와 writer는 이전 방법들과 달리 서로를 일방적으로 앞지를 수 없습니다. reader는 중복될 수 있지만 사진 사이 글자를 봤을 때, 많은 reader가 앞지를 수 없기에 보편적으로 1~2개의 스레드가 중복 작업을 진행했습니다.

**- 과제를 수행하면서 경험한 문제점과 느낀점**

과제를 수행하며 경험한 가장 큰 문제점은 확실히 이게 맞는지 틀린지에 대한 의문이었습니다.

초기에 writer 선호에 조건 변수 solution에서 참고 자료에 reader 선호 조건 변수에 추가로 int writers를 추가하여, writing 중인가?에 들어가기 전인 앞부분에 만약에 기다리는 writer인 writers가 있다면 reader를 wait해버리는 방식으로 reader가 writer를 앞지르지 못하게 하는 방식이 있었습니다. 저는 개인적으로 이 방법이 reader와 writer 선호 모두에 도움이 되며 훨씬 단순한 방식이라 생각했으나 실질적인 예상출력과는 다른 형태로 결과가 나왔기에 방식을 변경하게 됐습니다. 이해한 코드의 개념적으로는 문제가 없다 생각했으나 출력물은 달랐기에, 뭔가 문제점이 있다 생각은 했으나 어떤 이유인지, 내부적으로는 어떻게 돌아가는지를 확인하는 것이 수월하지 않았던 것 같습니다.

2번째 문제점은 OS 강의에서 꾸준히 든 생각이지만 양질의 정보를 찾는 과정에 대한 문제점이었습니다. 이번 과제의 경우에는 운이 좋게도 빨리 찾을 수 있었지만 기본적으로 되게 지엽적인 범위에서만 정보가 공개되어 있는 느낌이었습니다.

느낀점으로는 해당 과제를 수행을 했으나 제대로 이해를 했는가? 이것을 마음대로 사용할 수 있는가?에 대해서는 모르겠다는 점입니다. 정확히 방식을 생각하고 만들기보단 이미 완성된 방법을 분석하고 있다는 면이 좀 더 큰 것 같고, 분석을 하면서도 강의와 과제를 통해 atomic하다는게 무엇인가?에 대한 감은 오고 있으나 40%는 이해를 한 것인지 잘 모르겠습니다.

현재 찾은 해답과 그것을 고민하고 정리해본 결과 방법이 틀렸다는 생각은 안들지만 이게 확실히 100% 정답인 코드인가? 잘 돌아가고 있는게 맞는가? 라는 고민은 아직 오리무중인 것 같습니다.

참고자료

*https://stackoverflow.com/questions/56547704/readers-writers-problem-writers-preference-readers-may-starve*

*https://courses.engr.illinois.edu/cs241/sp2012/lectures/25-condition.pdf*

*\* CS 241 prof.Brighten Godfrey. Univ of Illinois*

*https://arxiv.org/abs/1309.4507*

*\* Faster Fair Solution for the Reader-Writer Problem - Vlad Popov, Oleg Mazonka*

*ChatGPT*

*\* Q. reader writer problem을 writer prefernce로 푸는 정석 방법 알려줘*

*\* 사용하는건 phtread\_mutex\_t 1개랑 pthread\_cond\_t 여러개만 사용해서 풀어줘*

Writer preference를 사용하여 Reader-Writer 문제를 해결하는 방법 중에서도 가장 간단한 방법은 'Writer-preference Lock'을 사용하는 것입니다. 이 방법은 작성자가 대기하는 동안 리더가 접근할 수 없도록 보장하여 작성자의 우선권을 부여합니다.

아래에 pthread\_mutex\_t와 pthread\_cond\_t를 사용하여 Writer preference로 Reader-Writer 문제를 해결하는 예시를 제시합니다.

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <pthread.h>

pthread\_mutex\_t lock = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_cond\_t writer\_condition = PTHREAD\_COND\_INITIALIZER;

pthread\_cond\_t reader\_condition = PTHREAD\_COND\_INITIALIZER;

int reader\_count = 0;

bool writer\_active = false;

int waiting\_writers = 0;

void \*reader(void \*arg)

{

int id = \*(int \*)arg;

pthread\_mutex\_lock(&lock);

while (writer\_active || waiting\_writers > 0) {

pthread\_cond\_wait(&reader\_condition, &lock);

}

reader\_count++;

pthread\_mutex\_unlock(&lock);

// Critical section (reading)

printf("Reader %d is reading\n", id);

pthread\_mutex\_lock(&lock);

reader\_count--;

if (reader\_count == 0) {

pthread\_cond\_signal(&writer\_condition);

}

pthread\_mutex\_unlock(&lock);

pthread\_exit(NULL);

}

void \*writer(void \*arg)

{

int id = \*(int \*)arg;

pthread\_mutex\_lock(&lock);

waiting\_writers++;

while (reader\_count > 0 || writer\_active) {

pthread\_cond\_wait(&writer\_condition, &lock);

}

waiting\_writers--;

writer\_active = true;

pthread\_mutex\_unlock(&lock);

// Critical section (writing)

printf("Writer %d is writing\n", id);

pthread\_mutex\_lock(&lock);

writer\_active = false;

pthread\_cond\_broadcast(&reader\_condition);

pthread\_cond\_signal(&writer\_condition);

pthread\_mutex\_unlock(&lock);

pthread\_exit(NULL);

}

int main()

{

pthread\_t readers[5];

pthread\_t writers[2];

int reader\_ids[5];

int writer\_ids[2];

int i;

for (i = 0; i < 5; ++i) {

reader\_ids[i] = i;

pthread\_create(&readers[i], NULL, reader, &reader\_ids[i]);

}

for (i = 0; i < 2; ++i) {

writer\_ids[i] = i;

pthread\_create(&writers[i], NULL, writer, &writer\_ids[i]);

}

for (i = 0; i < 5; ++i) {

pthread\_join(readers[i], NULL);

}

for (i = 0; i < 2; ++i) {

pthread\_join(writers[i], NULL);

}

return 0;

}이 예시에서는 **pthread\_mutex\_t**와 **pthread\_cond\_t**를 사용하여 상호 배제 및 조건 변수를 구현하여 Writer preference를 달성했습니다. 작성자가 대기 중이거나 작성 중일 때 리더는 접근할 수 없습니다. 리더가 대기 중이고 작성자가 없을 때 작성자가 우선권을 갖습니다.