Operating Systems(24802)
Project #3

Division of Computer Science
2017011912
Nam jung hoon

Contents

<Algorithm for writer_prefer.c>

Writer_prefer 의 경우 reader 의 중복을 최대한 허용하나 writer 가 들어왔다면 writer 를 우선 처리해주어야 합니다. 다음과 같은 경우를 생각해 볼 수 있습니다.
1) 진행중인 reader 가 있을 경우 : 또다른 reader 도 CS 에 접근 가능 writer 는 CS 에 접근 불가능
2) 대기중인 reader 가 있을 경우 : 대기중인 writer 가 모두 진행 되어야 대기중인 reader 실행 가능
3) 진행중인 writer 가 있을 경우 : reader 는 CS 에 접근 불가능 또다른 writer 또한 CS 에 접근 불가능
4) 대기중인 writer 가 있을 경우 : 대기중인 reader 가 있더라도 대기중인 writer 부터 먼저 실행
-> Reader 가 CS 에 접근할 수 있는 조건 :
진행 중인 writer 가 없으며 대기하고있는 writer 또한 없을 경우
->Writer 가 CS 에 접근할 수 있는 조건 :
진행 중인 reader 가 없으며 진행 중인 writer 가 없을 경우
Reader 는 자신이 CS 에 접근할 수 없는 조건이라면 자신이 실행되어도 된다는 신호가 올 때 까지 pthread_cond_wait(&reader_condition,&rw_mutex)에 의해 대기합니다.
신호가 왔다면 대기중인 모든 reader 들은 CS 에 접근하여 작업을 실행합니다.
모든 reader 들이 작업을 마쳤다면 pthread_cond_signal(&writer_condition) 로 writer 가 CS 에 접근 가능하다는 신호를 보냅니다.
Writer 는 자신이 CS 에 접근할 수 없는 조건이라면 자신이 실행되어도 된다는 신호가 올 때 까지 pthread_cond_wait(&writer_condition,&rw_mutex)에 의해 대기합니다.
신호가 왔다면 하나의 writer 가 CS 에 접근하여 작업을 실행합니다.
해당 writer 가 작업을 마쳤을때, writer 는 다음에 누구에게 신호를 줄 지 결정해야 합니다.
대기중인 writer 가 있다면 pthread_cond_signal(&writer_condition)로 다음 writer 에게 신호를 줍니다.
대기중인 writer 가 없으며 대기중인 reader 가 있다면 pthread_cond_broadcast(&reader_condition)로 대기중인 모든 reader 들이 실행 가능하다는 신호를 줍니다.

<Algorithm for fair_reader_writer.c>

Fair_reader_writer 의 경우 reader 의 중복을 최대한 허용 하며 들어온 순서대로 reader 와 writer 을 번갈아 가며 실행해 주어야 합니다. 다음과 같은 경우를 생각해 볼 수 있습니다.
1) Reader-Reader-Writer-Reader 순서로 들어왔을 경우 : 앞의 두 Reader 의 실행은 중복을 허용해 동시에 진행하며 Writer 가 들어온다면 Writer 를 실행해 준 뒤 Reader 를 마저 실행합니다.
2) Writer-Writer-Reader-Writer 순서로 들어왔을 경우 : 들어 온 순서대로 차근 차근 하나씩 실행해 줍니다.
->Reader 가 CS 에 접근할 수 있는 조건 :
자신의 순서일 경우
->Writer 가 CS 에 접근할 수 있는 조건 :
자신의 순서일 경우

다수의(1 개 이상) 의 Reader 들의 시작의 의미인 첫번째 Reader 는 Writer 의 실행을 막기 위해 pthread_mutex_lock(&rw_mutex)를 해줍니다.
만약 writer 가 실행되고 있다면 rw_mutex 는 lock 상태이므로 Reader 는 writer 가 종료되며 unlock 해줄 때 까지 기다려야 합니다.
다음 차례의 Reader, 즉 Writer 가 들어오기 전까지의 순서의 Reader 들이 동시에 실행됩니다.
실행되던 Reader 들이 모두 실행되어 read_count 가 0 이 되었다면 writer 가 CS 에 접근 할 차례이기 때문에 pthread_mutex_unlock(&rw_mutex)를 해줍니다.

Writer 가 들어왔을 경우 Reader 의 입장을 잠시 막기 위해 pthread_mutex_lock(&enter_mutex)를 해줍니다.
Writer 는 한번에 한 개 씩만 CS 에 접근할 수 있으므로 pthread_mutex_lock(&rw_mutex)를 해줍니다.
Pthread_mutex_unlock(&enter_mutex)을 해준 뒤 CS 에 접근하여 작업을 수행합니다.
작업을 모두 끝냈다면 다음 순서의 작업을 받기 위해 rw_mutex 를 unlock 해줍니다.

<Program source file – writer_prefer.c>

```c
/*
 * Copyright 2020, 2021. Heekuck Oh, all rights reserved
 * 이 프로그램은 한양대학교 ERICA 소프트웨어학부 재학생을 위한 교육용으로 제작되었습니다.
 */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <pthread.h>

#define N 8192
#define L1 75
#define L2 70
#define L3 70
#define RNUM 10
#define WNUM 3

pthread_mutex_t reader_mutex;
pthread_mutex_t writer_mutex;
pthread_mutex_t rw_mutex;
pthread_mutex_t reader_wait;
pthread_mutex_t writer_wait;
pthread_cond_t reader_condition;
pthread_cond_t writer_condition;

int reader_count = 0;
int writer_count = 0;
int wait_read = 0;
int wait_write = 0;
/*copy & paste 문제 때문에 생략하겠습니다*/
char *t[L1] = {};
char *d[L2] = {};
char *e[L3] = {};
/*
 * alive 값이 1이면 각 스레드는 무한 루프를 돌며 반복해서 일을 하고,
 * alive 값이 0이 되면 무한 루프를 빠져나와 스레드를 자연스럽게 종료한다.
 */
int alive = 1;

/*
 * Reader 스레드는 같은 문자를 N번 출력한다. 예를 들면 <AAA...AA> 이런 식이다.
 * 출력할 문자는 인자를 통해 0이면 A, 1이면 B, ..., 등으로 출력하며, 시작과 끝을 <...>로
나타낸다.
 * 단일 reader라면 <AAA...AA>처럼 같은 문자만 출력하겠지만, critical section에서 reader의
 * 중복을 허용하기 때문에 reader가 많아지면 출력이 어지럽게 섞여서 나오는 것이 정상이다.
 */
void *reader(void *arg)
```

```
{
    int id, i;
    /*
     * 들어온 인자를 통해 출력할 문자의 종류를 정한다.
     */
    id = *(int *)arg;
    /*
     * 스레드가 살아 있는 동안 같은 문자열 시퀀스 <XXX...XX>를 반복해서 출력한다.
     */
    while (alive) {
        pthread_mutex_lock(&rw_mutex);
        while(writer_count != 0 || wait_write != 0){
            pthread_mutex_lock(&reader_wait);
            wait_read++;
            pthread_mutex_unlock(&reader_wait);
            pthread_cond_wait(&reader_condition,&rw_mutex);
            pthread_mutex_lock(&reader_wait);
            wait_read--;
            pthread_mutex_unlock(&reader_wait);
        }
        pthread_mutex_unlock(&rw_mutex);
        pthread_mutex_lock(&reader_mutex);
        reader_count++;
        pthread_mutex_unlock(&reader_mutex);
        /*
         * Begin Critical Section
         */
        printf("<");
        for (i = 0; i < N; ++i)
            printf("%c", 'A'+id);
        printf(">");
        /*
         * End Critical Section
         */
        pthread_mutex_lock(&reader_mutex);
        reader_count--;
        pthread_mutex_lock(&rw_mutex);
        if(reader_count == 0){
            pthread_cond_signal(&writer_condition);
        }
        pthread_mutex_unlock(&reader_mutex);
        pthread_mutex_unlock(&rw_mutex);
    }
    pthread_exit(0);
}

/*
 * Writer 스레드는 어떤 사람의 얼굴 이미지를 출력한다.
 * 이미지는 세 가지 종류가 있으며 인자를 통해 식별한다.
```

```
 * Writer가 critical section에 있으면 다른 writer는 물론이고 어떠한 reader도 들어올 수
없다.
 * 만일 이것을 어기고 다른 writer나 reader가 들어왔다면 얼굴 이미지가 깨져서 쉽게 감지된다.
 */
void *writer(void *arg)
{
   int id, i;
   struct timespec req, rem;

   /*
    * 들어온 인자를 통해 얼굴 이미지의 종류를 정한다.
    */
   id = *(int *)arg;
   /*
    * 이미지 출력을 천천히 하기 위해 한 줄 출력할 때마다 쉬는 시간을 1 나노초로 설정한다.
    */
   req.tv_sec = 0;
   req.tv_nsec = 1L;
   /*
    * 스레드가 살아 있는 동안 같은 이미지를 반복해서 출력한다.
    */
   while (alive) {
      pthread_mutex_lock(&rw_mutex);
      while(reader_count >= 1 || writer_count != 0){
         pthread_mutex_lock(&writer_wait);
         wait_write++;
         pthread_mutex_unlock(&writer_wait);
         pthread_cond_wait(&writer_condition,&rw_mutex);
         pthread_mutex_lock(&writer_wait);
         wait_write--;
         pthread_mutex_unlock(&writer_wait);
      }
      pthread_mutex_unlock(&rw_mutex);

      pthread_mutex_lock(&writer_mutex);
      writer_count++;
      pthread_mutex_unlock(&writer_mutex);
       /*
        * Begin Critical Section
        */
       printf("\n");
       switch (id) {
          case 0:
             for (i = 0; i < L1; ++i) {
                printf("%s\n", t[i]);
                nanosleep(&req, &rem);
             }
             break;
          case 1:
             for (i = 0; i < L2; ++i) {
```

```
                printf("%s\n", d[i]);
                nanosleep(&req, &rem);
            }
            break;
        case 2:
            for (i = 0; i < L3; ++i) {
                printf("%s\n", e[i]);
                nanosleep(&req, &rem);
            }
            break;
        default:
            ;
    }
    /*
     * End Critical Section
     */
    pthread_mutex_lock(&writer_mutex);
    writer_count--;
    pthread_mutex_unlock(&writer_mutex);
    pthread_mutex_lock(&rw_mutex);
    if(wait_write == 0 && wait_read != 0){
        pthread_cond_broadcast(&reader_condition);
    }
    else if(wait_write != 0){
        pthread_cond_signal(&writer_condition);
    }
    pthread_mutex_unlock(&rw_mutex);
  }
  pthread_exit(0);
}

/*
 * 메인 함수는 RNUM 개의 reader 스레드를 생성하고, WNUM 개의 writer 스레드를 생성한다.
 * 생성된 스레드가 일을 할 동안 0.2 초 동안 기다렸다가 alive 의 값을 0 으로 바꿔서 모든 스레드가
 * 무한 루프를 빠져나올 수 있게 만든 후, 스레드가 자연스럽게 종료할 때까지 기다리고 메인을
종료한다.
 */
int main(void)
{
  int i;
  int rarg[RNUM], warg[WNUM];
  pthread_t rthid[RNUM];
  pthread_t wthid[WNUM];
  struct timespec req, rem;

  pthread_mutex_init(&reader_mutex,NULL);
  pthread_mutex_init(&writer_mutex,NULL);
  pthread_mutex_init(&rw_mutex,NULL);
  pthread_mutex_init(&reader_wait,NULL);
  pthread_mutex_init(&writer_wait,NULL);
```

```
    pthread_cond_init(&reader_condition,NULL);
    pthread_cond_init(&writer_condition,NULL);

    /*
     * Create RNUM reader threads
     */
    for (i = 0; i < RNUM; ++i) {
        rarg[i] = i;
        if (pthread_create(rthid+i, NULL, reader, rarg+i) != 0) {
            fprintf(stderr, "pthread_create error\n");
            exit(-1);
        }
    }
    /*
     * Create WNUM writer threads
     */
    for (i = 0; i < WNUM; ++i) {
        warg[i] = i;
        if (pthread_create(wthid+i, NULL, writer, warg+i) != 0) {
            fprintf(stderr, "pthread_create error\n");
            exit(-1);
        }
    }
    /*
     * Wait for 0.2 second while the threads are working
     */
    req.tv_sec = 0;
    req.tv_nsec = 300000000L;
    nanosleep(&req, &rem);
    /*
     * Now terminate all threads and leave
     */
    alive = 0;
    for (i = 0; i < RNUM; ++i)
        pthread_join(rthid[i], NULL);
    for (i = 0; i < WNUM; ++i)
        pthread_join(wthid[i], NULL);
    pthread_mutex_destroy(&reader_mutex);
    pthread_mutex_destroy(&writer_mutex);
    pthread_mutex_destroy(&rw_mutex);
    pthread_mutex_destroy(&wait_read);
    pthread_mutex_destroy(&wait_write);
    exit(0);
}
```

<Program source file – fair_reader_writer.c>

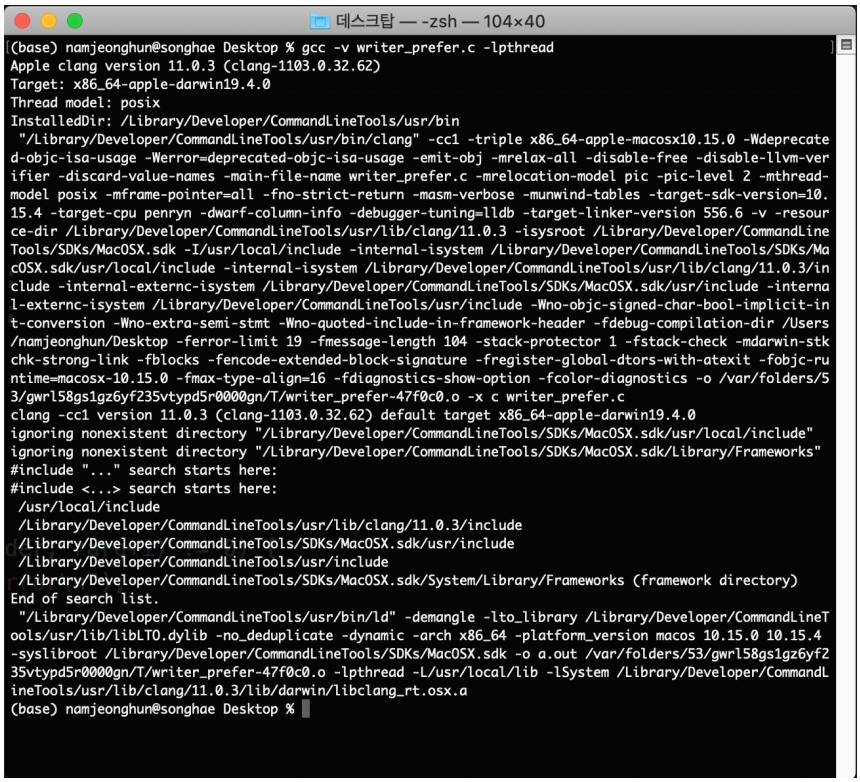
```c
/*
 * Copyright 2020, 2021. Heekuck Oh, all rights reserved
 * 이 프로그램은 한양대학교 ERICA 소프트웨어학부 재학생을 위한 교육용으로 제작되었습니다.
 */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <pthread.h>

#define N 8192
#define L1 75
#define L2 70
#define L3 70
#define RNUM 10
#define WNUM 3

pthread_mutex_t enter_mutex;
pthread_mutex_t reader_mutex;
pthread_mutex_t rw_mutex;

int read_count = 0;
/*copy & paste 문제 때문에 생략하겠습니다*/
char *t[L1] = {};
char *d[L2] = {};
char *e[L3] = {};

/*
 * alive 값이 1이면 각 스레드는 무한 루프를 돌며 반복해서 일을 하고,
 * alive 값이 0이 되면 무한 루프를 빠져나와 스레드를 자연스럽게 종료한다.
 */
int alive = 1;

/*
 * Reader 스레드는 같은 문자를 N번 출력한다. 예를 들면 <AAA...AA> 이런 식이다.
 * 출력할 문자는 인자를 통해 0이면 A, 1이면 B, ..., 등으로 출력하며, 시작과 끝을 <...>로
나타낸다.
 * 단일 reader라면 <AAA...AA>처럼 같은 문자만 출력하겠지만, critical section에서 reader의
 * 중복을 허용하기 때문에 reader가 많아지면 출력이 어지럽게 섞여서 나오는 것이 정상이다.
 */
void *reader(void *arg)
{
    int id, i;

    /*
     * 들어온 인자를 통해 출력할 문자의 종류를 정한다.
     */
    id = *(int *)arg;
```

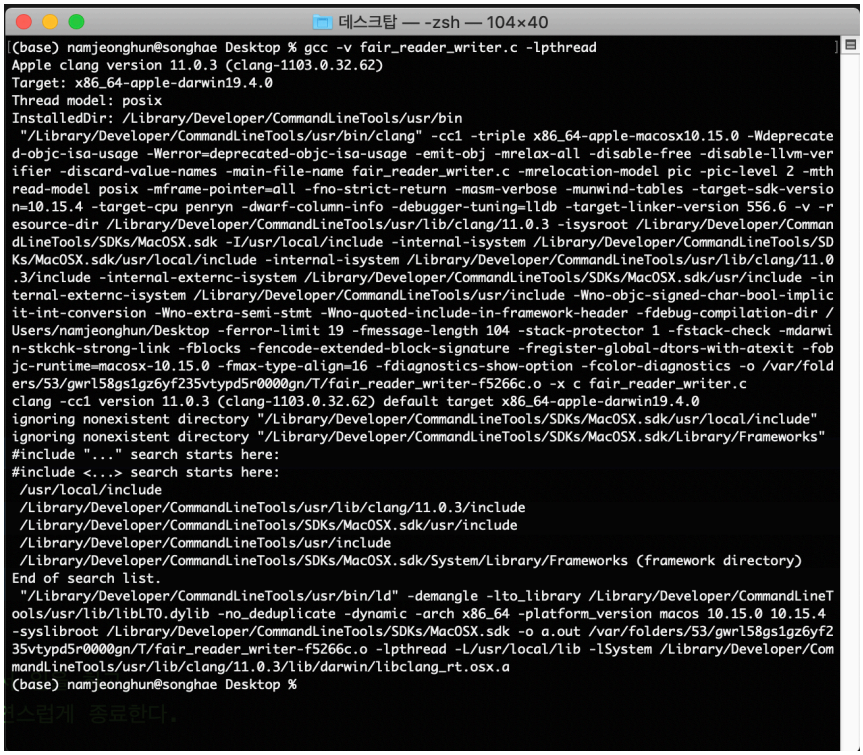
```
    /*
     * 스레드가 살아 있는 동안 같은 문자열 시퀀스 <XXX...XX>를 반복해서 출력한다.
     */
    while (alive) {
        pthread_mutex_lock(&enter_mutex);
        pthread_mutex_lock(&reader_mutex);
        if(read_count++==0){
            pthread_mutex_lock(&rw_mutex);
        }
        pthread_mutex_unlock(&reader_mutex);
        pthread_mutex_unlock(&enter_mutex);
        /*
         * Begin Critical Section
         */
        printf("<");
        for (i = 0; i < N; ++i)
            printf("%c", 'A'+id);
        printf(">");
        /*
         * End Critical Section
         */
        pthread_mutex_lock(&reader_mutex);
        read_count--;
        if(read_count==0){
            pthread_mutex_unlock(&rw_mutex);
        }
        pthread_mutex_unlock(&reader_mutex);
    }
    pthread_exit(0);
}

/*
 * Writer 스레드는 어떤 사람의 얼굴 이미지를 출력한다.
 * 이미지는 세 가지 종류가 있으며 인자를 통해 식별한다.
 * Writer가 critical section에 있으면 다른 writer는 물론이고 어떠한 reader도 들어올 수
없다.
 * 만일 이것을 어기고 다른 writer나 reader가 들어왔다면 얼굴 이미지가 깨져서 쉽게 감지된다.
 */
void *writer(void *arg)
{
    int id, i;
    struct timespec req, rem;

    /*
     * 들어온 인자를 통해 얼굴 이미지의 종류를 정한다.
     */
    id = *(int *)arg;
    /*
     * 이미지 출력을 천천히 하기 위해 한 줄 출력할 때마다 쉬는 시간을 1 나노초로 설정한다.
     */
```

```
    req.tv_sec = 0;
    req.tv_nsec = 1L;
    /*
     * 스레드가 살아 있는 동안 같은 이미지를 반복해서 출력한다.
     */
    while (alive) {
        pthread_mutex_lock(&enter_mutex);
        pthread_mutex_lock(&rw_mutex);
        pthread_mutex_unlock(&enter_mutex);
        /*
         * Begin  Critical Section
         */
        printf("\n");
        switch (id) {
            case 0:
                for (i = 0; i < L1; ++i) {
                    printf("%s\n", t[i]);
                    nanosleep(&req, &rem);
                }
                break;
            case 1:
                for (i = 0; i < L2; ++i) {
                    printf("%s\n", d[i]);
                    nanosleep(&req, &rem);
                }
                break;
            case 2:
                for (i = 0; i < L3; ++i) {
                    printf("%s\n", e[i]);
                    nanosleep(&req, &rem);
                }
                break;
            default:
                ;
        }
        /*
         * End Critical Section
         */
        pthread_mutex_unlock(&rw_mutex);

    }
    pthread_exit(0);
}

/*
 * 메인 함수는 RNUM 개의 reader 스레드를 생성하고, WNUM 개의 writer 스레드를 생성한다.
 * 생성된 스레드가 일을 할 동안 0.2 초 동안 기다렸다가 alive 의 값을 0 으로 바꿔서 모든 스레드가
 * 무한 루프를 빠져나올 수 있게 만든 후, 스레드가 자연스럽게 종료할 때까지 기다리고 메인을
종료한다.
 */
```

```c
int main(void)
{
    int i;
    int rarg[RNUM], warg[WNUM];
    pthread_t rthid[RNUM];
    pthread_t wthid[WNUM];
    struct timespec req, rem;

    pthread_mutex_init(&enter_mutex,NULL);
    pthread_mutex_init(&reader_mutex,NULL);
    pthread_mutex_init(&rw_mutex,NULL);
    /*
     * Create RNUM reader threads
     */
    for (i = 0; i < RNUM; ++i) {
        rarg[i] = i;
        if (pthread_create(rthid+i, NULL, reader, rarg+i) != 0) {
            fprintf(stderr, "pthread_create error\n");
            exit(-1);
        }
    }
    /*
     * Create WNUM writer threads
     */
    for (i = 0; i < WNUM; ++i) {
        warg[i] = i;
        if (pthread_create(wthid+i, NULL, writer, warg+i) != 0) {
            fprintf(stderr, "pthread_create error\n");
            exit(-1);
        }
    }
    /*
     * Wait for 0.2 second while the threads are working
     */
    req.tv_sec = 0;
    req.tv_nsec = 500000000L;
    nanosleep(&req, &rem);
    /*
     * Now terminate all threads and leave
     */
    alive = 0;
    for (i = 0; i < RNUM; ++i)
        pthread_join(rthid[i], NULL);
    for (i = 0; i < WNUM; ++i)
        pthread_join(wthid[i], NULL);
    pthread_mutex_destroy(&enter_mutex);
    pthread_mutex_destroy(&reader_mutex);
    pthread_mutex_destroy(&rw_mutex);
    exit(0);
}
```

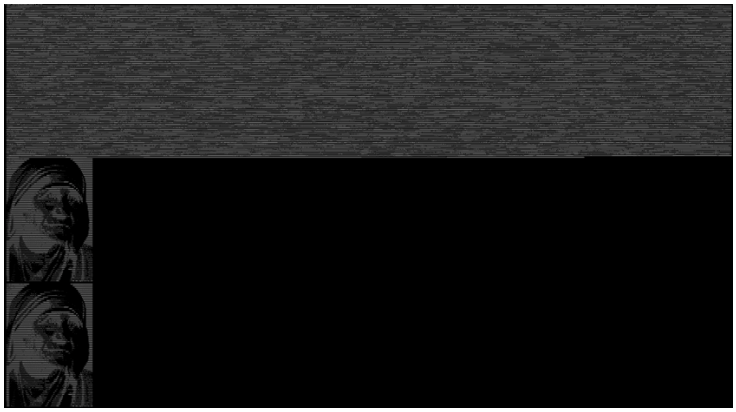
<Compilation Process>





<Deliverables and Descriptions>

1) Writer_prefer.c



Writer 가 들어오기 전에 들어온 reader 들이 동시에 작업을 수행합니다.
Writer 가 들어왔을 경우 reader 는 대기하고 들어온 writer 들을 실행해줍니다.

대기중이었던 writer 들의 작업이 모두 끝났다면,
대기중이였던 reader 들이 동시에 작업을 수행합니다.
작업을 모두 마쳤다면 종료합니다.

2) fair_reader_writer.c

위와 같이 writer 와 reader 가 번갈아가면서 실행됩니다.