Operating Systems

Project #4

2019041094 김정민

본인이 설계한 네 가지 버전에 대한 설명

Reader 선호 (조건변수 버전)

```
bool alive = true;
pthread_mutex_t mutex;
pthread_cond_t writers_cond;
int readers_count = 0;
int writers_count = 0;
```

조건변수를 조회하기 위한 뮤텍스락과, writers_cond 조건변수,

현재 활동하고 있는 readers와 writers의 개수를 나타내는 변수 두개

Reader 함수

```
while (alive) {

    pthread_mutex_lock(&mutex);
    readers_count++;
    pthread_mutex_unlock(&mutex);

    /*
     * Begin Critical Section
     */
    printf("<");
    for (i = 0; i < L0; ++i)
        printf("%c", 'A'+id);
    printf(">");
    /*
     * End Critical Section
     */

    pthread_mutex_lock(&mutex);
    readers_count--;
    if(readers_count == 0) pthread_cond_broadcast(&writers_cond);
    pthread_mutex_unlock(&mutex);

}
```

Readers_count에 1더해주기 위해서 mutex lock을 걸어준 뒤

Critical Section을 지나고 나서 readers_count에 1을 감소시키기 위해서 mutex lock을 또 걸어준다

더 이상 reader가 없다면 조건 변수를 broadcast해 자고 있던 writer를 전부 다 깨워준다.

Writer 함수

```
while (alive) {

    pthread_mutex_lock(&mutex);

    while(readers_count > 0) pthread_cond_wait(&writers_cond, &mutex);
    //pthread_mutex_unlock(&mutex);


    /*
     * Begin Critical Section
     */
    printf("\n");
    switch (id) {
        case 0:
            for (i = 0; i < L1; ++i)
                printf("%s\n", img1[i]);
            break;
        case 1:
            for (i = 0; i < L2; ++i)
                printf("%s\n", img2[i]);
            break;
        case 2:
            for (i = 0; i < L3; ++i)
                printf("%s\n", img3[i]);
            break;
        case 3:
            for (i = 0; i < L4; ++i)
                printf("%s\n", img4[i]);
            break;
        case 4:
            for (i = 0; i < L5; ++i)
                printf("%s\n", img5[i]);
            break;
        default:
            ;
    }
    /*
     * End Critical Section
     */

    pthread_mutex_unlock(&mutex);
    /*
     * 이미지 출력 후 SLEEPTIME 나노초 안에서 랜덤하게 쉰다.
     */
```

Mutex lock을 건 다음에 reader가 1개라도 존재한다면 조건 변수를 사용해 writer를 채운다.

Writer 선호 (뮤텍스락 버전)

```
bool alive = true;
pthread_mutex_t mutex;
int active_writers = 0;
int active_readers = 0;
int waiting_writers = 0;
```

Mutex 하나랑 현재 실행중인 active_writers, active_readers이다. 그리고 기다리는 writer를 표시하는 waiting_writer가 있다.

Reader 함수

```
while (alive) {
    pthread_mutex_lock(&mutex);
    while (active_writers == 1 || waiting_writers > 0) {
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);
    }

    active_readers++;

    pthread_mutex_unlock(&mutex);
    /*
     * Begin Critical Section
     */
    printf("<");
    for (i = 0; i < L0; ++i)
        printf("%c", 'A'+id);
    printf(">");
    /*
     * End Critical Section
     */
    pthread_mutex_lock(&mutex);
    active_readers--;
    pthread_mutex_unlock(&mutex);
}
pthread_exit(NULL);
```

락을 걸고 active_writers == 1 또는 waiting_writers > 0이면 언락하고 다시 락을 건다.

Active_readers 1을 더해주고 다시 락을 건다.

그리고 critical Section을 지난 후에

다시 락 사이에 active_readers을 1 빼준다.

Writer 함수

```
pthread_mutex_lock(&mutex);

waiting_writers++;
while (active_readers > 0 || active_writers == 1) {
    pthread_mutex_unlock(&mutex);
    pthread_mutex_lock(&mutex);
}
waiting_writers--;

active_writers = 1;

pthread_mutex_unlock(&mutex);
```

락을 건 후에 waiting_writers 1을 더해주고 active_readers >0 거나 active_writers == 1이면 언락하고 다시 락을 건다. 그러고 waiting_writers 1을 빼주고 active_writers 값을 1로 바꿔주고 언락한다.

```
pthread_mutex_lock(&mutex);
active_writers = 0;

if (waiting_writers > 0) {
    pthread_mutex_unlock(&mutex);
    continue;
}

pthread_mutex_unlock(&mutex);
```

락을 건 후 active_writers의 값을 0으로 바꿔주고 waiting_writers가 0보다 크다면 언락하고 continue 한다. 그러고 언락한다.

Writer 선호 (조건변수 버전)

```
bool alive = true;
pthread_mutex_t mutex;
pthread_cond_t readers_cond, writers_cond;
int active_writers = 0;
int active_readers = 0;
int waiting_writers = 0;
```

조건변수를 조회하기 위한 뮤텍스 락과 두 개의 조건 변수가 있고

현재 실행 중인 reader와 writers를 확인할 수 있는 active_writers와 active_readers 두 변수

기다리고 있는 writers의 수를 알 수 있는 waiting_writers 가 있다.

Reader 함수

```
pthread_mutex_lock(&mutex);
while (active_writers == 1 || waiting_writers > 0) {
        pthread_cond_wait(&readers_cond, &mutex);
}

active_readers++;
pthread_mutex_unlock(&mutex);

/*
 * Begin Critical Section
 */
printf("<");
for (i = 0; i < L0; ++i)
    printf("%c", 'A'+id);
printf(">");
/*
 * End Critical Section
 */

pthread_mutex_lock(&mutex);
active_readers--;
if(active_readers == 0) pthread_cond_signal(&writers_cond);
pthread_mutex_unlock(&mutex);
```

뮤텍스 락을 건 후에 현재 실행 중인 writer가 있거나 기다리고 있는 writer가 있다면 reader_cond 조건변수를 wait 상태로 둔다. 그런 다음 wait 상태가 깨지면 active_readers 수를 1 더해준다.

Critical Section을 수행한 후에

다시 뮤텍스락을 건 다음 active_readers의 수를 1 빼준다.

그리고 active_readers가 만약 0이라면 writers_cond에 signal을 보내준다.

Writer 함수

```
while (alive) {
    pthread_mutex_lock(&mutex);

    waiting_writers++;
    while(active_readers > 0 || active_writers == 1) {
            pthread_cond_wait(&writers_cond, &mutex);
    }
    waiting_writers --;

    active_writers = 1;


     pthread_mutex_unlock(&mutex);

/*
 * End Critical Section
 */
 pthread_mutex_lock(&mutex);

active_writers = 0;

if (waiting_writers > 0) {
    pthread_cond_signal(&writers_cond);
} else if(waiting_writers == 0) pthread_cond_broadcast(&readers_cond);


pthread_mutex_unlock(&mutex);
/*
 * 이미지 출력 후 SLEEPTIME 나노초 안에서 랜덤하게 쉬다.
```

뮤텍스락을 건 후에 현재 실행 중인 reader의 수가 0보다 크거나 현재 실행 중인 writer가 있다면

Waiting_writers의 값을 늘려주고 writers_cond 조건변수를 wait 상태로 바꾼다. Wait 상태에서 벗어난다면 waiting_writers의 값을 1 줄여준다.

그런 다음 acitve_writers값을 1로 바꿔준다.

Critical Section을 지난다음에 다시 뮤텍스 락을 걸고 active_writers의 값을 0으로 바꿔준다.

이 때, waiting_writers가 0보다 크다면 writers_cond에 signal을 보내준다. 0이라면 readers_cond 조건변수엥 broadcast해준다.

공정한 reader-writer (뮤텍스락 버전)

```
bool alive = true;

pthread_mutex_t reader_mutex, arrive_mutex, access_mutex;
int readers_count = 0;
```

Reader_mutex : reader의 중복 허용 처리 위한 mutex

Arrive_mutex: reader 와 writer 도착 시 순서대로 실행 위한 mutex

Access_mutex: 접근을 위한 mutex

Readers_count: 실행 중인 reader 개수

```
while (alive) {

    pthread_mutex_lock(&arrive_mutex);
    pthread_mutex_lock(&reader_mutex);

    if(readers_count == 0) {
            pthread_mutex_lock(&access_mutex);
    }
    readers_count++;

    pthread_mutex_unlock(&reader_mutex);
    pthread_mutex_unlock(&arrive_mutex);
    /*
     * Begin Critical Section
     */
    printf("<");
    for (i = 0; i < L0; ++i)
        printf("%c", 'A'+id);
    printf(">");
    /*
     * End Critical Section
     */
    pthread_mutex_lock(&reader_mutex);
    readers_count --;
    if(readers_count == 0) {
            pthread_mutex_unlock(&access_mutex);
    }
    pthread_mutex_unlock(&reader_mutex);
}
pthread_exit(NULL);
```

도착 순서대로 실행을 위해 lock을 걸고 reader는 중복을 허용하기 때문에 readers_count로 개수를 나타낸다. Read_count 조작을 위해 lock을 건 후 read_count가 0이면 처음 도착한 reader이다.

따라서 read 들어간다. Reader 추가하고 reader_mutex unlock한다. 그리고 arrive_mutex도 unlock 해준다.

Critical section 을 지난 후

Readers_count 조작을 위해 reader_mutex lock을 걸고, 실행 중인 reader가 없는 경우 access_mutex를 언락한다.

그리고 reader_mutex를 언락해준다.

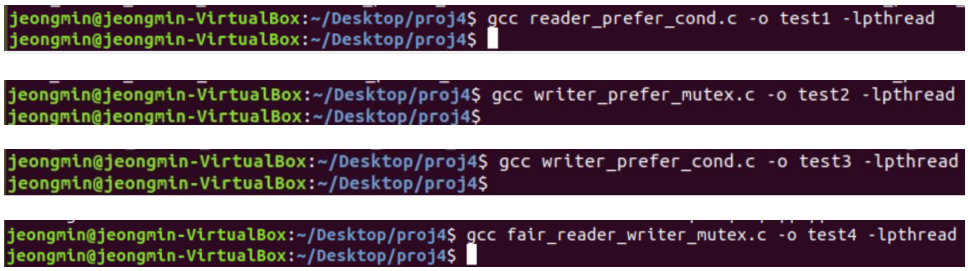
```
while (alive) {

    pthread_mutex_lock(&arrive_mutex);
    pthread_mutex_lock(&access_mutex);
    pthread_mutex_unlock(&arrive_mutex);
    /*
     * Begin Critical Section
     */
    printf("\n");
    switch (id) {
        case 0:
            for (i = 0; i < L1; ++i)
                printf("%s\n", img1[i]);
            break;
        case 1:
            for (i = 0; i < L2; ++i)
                printf("%s\n", img2[i]);
            break;
        case 2:
            for (i = 0; i < L3; ++i)
                printf("%s\n", img3[i]);
            break;
        case 3:
            for (i = 0; i < L4; ++i)
                printf("%s\n", img4[i]);
            break;
        case 4:
            for (i = 0; i < L5; ++i)
                printf("%s\n", img5[i]);
            break;
        default:
            ;
    }
    /*
     * End Critical Section
     */
    pthread_mutex_unlock(&access_mutex);
    /*
     * 이미지 출력 후 SLEEPTIME 나노초 안에서 랜덤하게 쉰다.
```

Arrive_mutex를 락하고 writer는 중복이 안되므로 바로 write들어간다 (access_mutex 락을 건다).

그런 후에 arrive_mutex를 언락한다. Critical section을 지난 후에 access_mutex 언락한다.

컴파일 과정을 보여주는 화면 캡처



느낀점과 문제점

내가 reader-writer 문제를 해결하면서 배운 것은 reader 선호 방식, writer 선호 방식, 그리고 공정한 reader-writer 방식을 구현할 때 상호 배제와 동기화가 얼마나 중요한지 알게 되었습니다.

그러나, 이러한 방식을 구현하는 것은 쉬운 일이 아닙니다. 예를 들어, reader와 writer가 동시에 접근하는 경우 경쟁 조건이 발생할 수 있습니다. 이를 해결하기 위해서는 reader와 writer 간의 상호 배제와 동기화를 보장하는 메커니즘이 필요합니다.

또한, reader-writer 방식을 구현할 때에는 애플리케이션의 특성에 따라 적절한 방식을 선택해야 합니다. 예를 들어, reader가 많이 존재하는 경우 reader 선호 방식을 사용하는 것이 더 효율적일 수 있습니다.

따라서, reader-writer 방식을 구현할 때에는 상호 배제와 동기화를 고려하고, 애플리케이션의 특성에 맞는 방식을 선택하는 것이 중요합니다.

실행 결과물의 주요 장면을 발췌해서 그에 대한 상세한 설명 및 차이점

reader_prefer_cond.txt에서 ,

문자열이 나오다가 writer 스레드가 마지막에 한 번에 출력된다.

writer_prefer_mutex.txt , writer_prefer_cond.txt

위가 뮤텍스락을 활용해 구현한 writer 선호 방식이고, 아래는 조건변수를 활용해 구현한 writer 선호 방식이다.

둘 다 writer 스레드에서 출력 후에 reader 스레드에서 출력하는 모습

Fair_reader_writer_mutex.txt

Reader와 writer가 선착순으로 와서 출력되는 모습