운영체제론 PROJ4 보고서

소프트웨어학부 2019007329 이지현

1. 코드 설명

- writer_prefer_mutex.c

뮤택스락으로 writer 선호 방식을 구현하였다. 총 4개의 뮤택스 락을 사용하였고, write_count, read_count 공유 변수를 사용하였다.

reader 함수는 중복 접근은 허용하지만 대기하고 있는 writer를 앞지를 수 없다. 먼저 pthread_mutex_lock(&r_mutex)로 하나의 스레드만 들어올 수 있도록 r_mutex를 잠궈준다. 그 후 read_count 계산을 위한 pthread_mutex_lock(&read_mutex)로 read_mutex를 잠궈준다. read_count를 하나 늘려주고, read_count == 1이면 write_mutex를 잠궈준다. 그 후 read_mutex와 r_mutex를 풀어준다. cs 접근이 끝난 뒤 read_mutex를 사용하여 락을 잠궈준다. read_count 값만 다룰 것이기 때문에 r_mutex를 사용하지 않았다. read_count값을 하나 감소시키고 read_count == 0이면 write_mutex를 풀어준다. 그 후 read_mutex를 풀어준다.

writer 함수는 w_mutex를 잠군 뒤 write_count를 하나 증가시킨다. write_count == 1이면 r_mutex를 잠근다. 그 후 w_mutex를 풀어준다. writer는 하나의 스레드만 cs에 접근가능하므로 write_mutex 락을 건 후 cs에 접근한다. print가 끝나면 write_mutex를 풀어준 뒤 w_mutex를 잠궈준다. write_count를 감소시키고 write_count가 0이면 r_mutex를 풀어준다. 그 후 w_mutex까지 풀어준다.

- reader_prefer_cond.c

cond 변수를 사용하여 reader 선호 방식을 구현하였다. 한개의 뮤택스락과 한개의 조건변수, 한개의 read_count 공유 변수를 사용하였다.

reader 함수는 pthread_mutex_lock(&mutex)로 mutex를 잠근 뒤 read_count를 하나 늘린다. 그 후 mutex를 다시 푼다. writer 함수에서 직접 writer 스레드를 다룰 것이기 때문에 조건문은 사용하지 않았다. cs에 접근 후 print를 마치면 mutex를 다시 잠근다. read_count를 하나 줄이고 read_count == 0이면 더이상 reader 스레드가 없으므로 pthread_cond_signal(&cond_w)로 writer 스레드를 깨운다. 마지막으로 pthread_mutex_unlock(&mutex)로 mutex를 풀어준다.

writer 함수는 pthread_mutex_lock(&mutex)로 mutex를 잠궈준다. while(read_count >= 1)로 read를

진행하고 있다면 pthread_cond_wait(&cond_w, &mutex)로 더이상 writer 스레드가 print 하지 못하도록 대기시킨다. read_count가 0이 되면 while문을 빠져나와 cs에 접근하고, print가 끝나면 cond_w를 깨우고 mutex를 풀어준다.

main 함수에서는 mutex와 cond_w를 pthread_mutex_init(&mutex, NULL)와 pthread_cond_init (&cond_w,NULL)으로 설정해준 뒤 스레드의 함수 실행이 모두 끝나면 destroy로 종료해 준다.

- writer_prefer_cond.c

cond 변수를 사용하여 writer 선호 방식을 구현하였다. 한개의 뮤택스락과 두개의 조건변수, 세개의 공유 변수를 사용하였다.

reader 함수는 pthread_mutex_lock(&mutex)로 mutex를 잠궈준 뒤 while(write_count >= 1 || write_wait >= 1)으로 write 스레드가 cs에 있거나 기다리는 wait 스레드가 있다면 while문에서 pthread_cond_wait(&cond_r,&mutex)를 반복한다. 기다리는 wait 스레드와 cs에 진입한 wait스레드가 없다면 while문을 빠져나오고 read_count를 증가시킨 후 한 스레드가 cs 진입 단계에 도착 후 mutex를 풀어준다. cs 진입 후 print를 끝냈다면 mutex를 다시 잠근 후 read_count를 하나 줄인다. 만약 read_count == 0이라면 더이상 실행중인 read가 없다는 뜻이기 때문에 write 스레드에 신호를 보낸다. 그 후 mutex 락을 풀어준다.

writer 함수는 reader와 같은 mutex를 잠궈주고 write_wait을 하나 늘려준다. while(write_count >= 1 || read_count >= 1)을 사용하여 read가 실행중이지 않고 write가 실행중이지 않으면 빠져나오고, 그렇지 않으면 기다린다. 그 후 writer_count를 늘리고 wait를 줄인다. 그 후 뮤택스를 푼다. cs를 나오면 뮤택스 락을 걸고 write_count를 줄인 후 write_wait == 0이면 read를 broadcast하고 아니면 write에 신호를 보낸다. 그후 뮤택스를 푼다.

- fair_reader_writer_mutex.c

reader나 writer가 대기하고 있는 스레드를 앞지를 수 없는 방식을 구현하였다. 세개의 뮤택스락과 한개의 공유변수 read_count를 이용하였다.

reader 함수는 writer 함수와 같은 mutex를 이용하여 락을 걸어주고, read_count를 위한 스레드 제한을 위해 r_mutex 또한 락을 걸어준다. 그 후 read_count를 증가시키고, read_count == 1이라면 w_mutex를 잠군다. 그 후 r_mutex, mutex의 락을 풀어준다. cs에 접근한 뒤 print가 끝나면 r_mutex를 잠군다. r_mutex인 이유는 read_count만을 다룰 것이기 때문이다. read_count를 감소시킨 후 read_count == 0이라면 더이상 reader가 없으므로 writer를 허용하기 위해 w_mutex의 락을 풀어준다. 그 후 r_mutex도 풀어준다.

writer 함수는 mutex를 잠근 뒤 cs에 하나의 스레드만을 허용하기 위해 w_mutex를 잠궈준다.

mutex는 reader의 스레드를 제어할 때 중간에 풀릴 수 있으므로 w_mutex를 따로 사용하였다. cs 접근 후 print가 끝나면 w_mutex와 mutex를 풀어준다.

main 함수에서는 mutex, r_mutex, w_mutex를 pthread_mutex_init으로 초기화 한 뒤, 모든 스레드가 함수를 실행하는 것을 마치면 세개의 mutex들을 모두 destroy 하였다.

2. 컴파일 과정

- writer_prefer_mutex.c

```
os@os-VirtualBox:~/Documents/theory_assignment/proj4$ gcc -o writer_prefer_mutex writer_prefer_mutex.c -lpthread
```

- reader_prefer_cond.c

```
os@os-VirtualBox:~/Documents/theory_assignment/proj4$ gcc -o reader_prefer_cond reader_prefer_cond.c -lpthread
```

- writer_prefer_cond.c

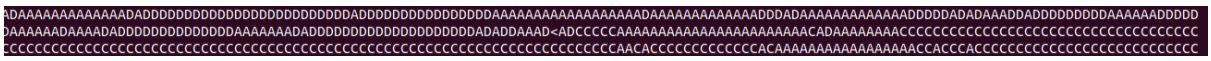
```
os@os-VirtualBox:~/Documents/theory_assignment/proj4$ gcc -o writer_prefer_cond writer_prefer_cond.c -lpthread
```

- fair_reader_writer_mutex.c

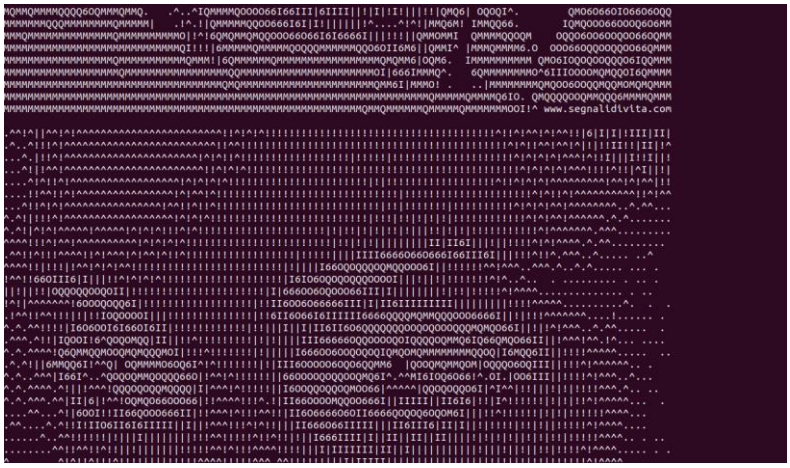
```
os@os-VirtualBox:~/Documents/theory_assignment/proj4$ gcc -o fair_reader_writer_mutex fair_reader_writer_mutex.c -lpthread
os@os-VirtualBox:~/Documents/theory_assignment/proj4$ ./fair_reader_writer_mutex
```

3. 실행 결과

- writer_prefer_mutex.c



서로 다른 알파벳들이 출력된 모습이다.



writer는 중복되어 출력되지 않았다.

실행 결과마다 조금씩 다르지만, 초반에 writer가 주로 출력되다가, 마지막에 reader가 몰려서 출력되는 결과를 보였다.

- reader_prefer_cond.c

reader가 중복되어 보인다.

writer는 중복되지 않았다.

reader 위주로 초반에 출력된 뒤 마지막에 writer가 출력된다.

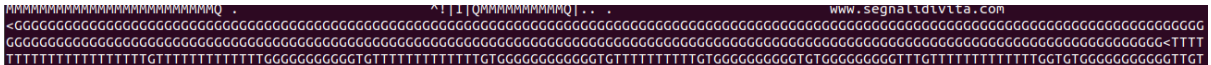
- writer_prefer_cond.c

reader가 중복되어 보인다.

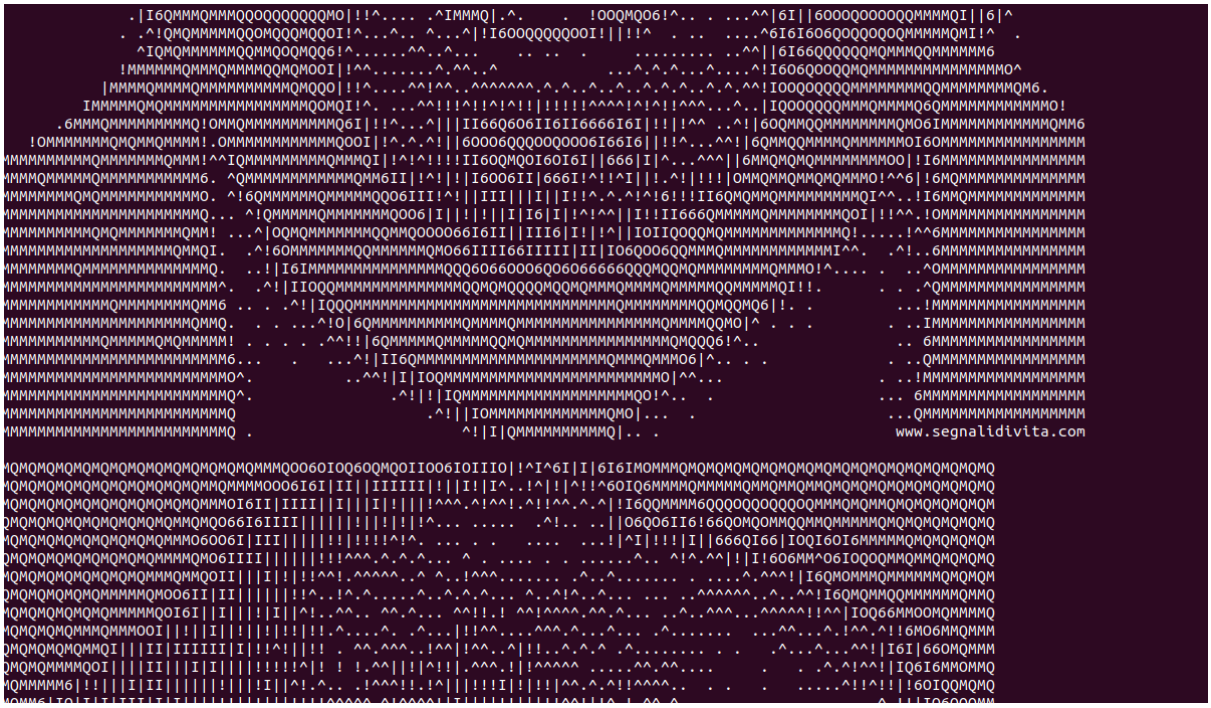
writer는 중복되지 않았다.

writer 위주로 초반에 출력된 뒤 마지막에 reader가 출력된다.

- fair_reader_writer_mutex.c



서로 다른 알파벳이 중간에 출력된 모습이다.



writer는 하나의 스레드가 모두 출력된 뒤 다른 스레드가 출력된다.

prefer 프로그램에 비해 reader와 writer가 번갈아 나오는 빈도수가 높았다.

4. 문제점과 느낀점

하나의 문제를 여러가지 방식으로 풀어보며 락을 이용해 스레드를 제어하는 방식과 조건 변수를 이용하여 스레드를 제어하는 방식을 모두 알 수 있어 동기화를 이해하는데 많은 도움이 되었다. reader와 writer가 각각 동시에 cs에 접근할 수 있는 스레드의 수가 다르기 때문에 공정한 reader writer 문제보다 prefer 문제가 더욱 어렵게 느껴졌다. reader writer 문제는 두 함수 사이에서의 동기화만 고려하면 되지만, 만약 더 많은 개수의 함수들 사이에서 동기화를 진행한다면 알고리즘이 굉장히 복잡해질 것 같다는 생각이 들었다. 동기화 문제가 어렵게 느껴졌던 이유는 내가 구현한 동기화 프로그램을 실행했을 때 올바른 실행 결과가 나왔더라도 사실은 동기화가 이루어지지 않았을 수도 있다는 점 때문이었다. 그렇기에 테스트 결과만으로 동기화가 제대로 이루어졌는지 파악하기 어려웠다. 이러한 특징 때문에 처음부터 알고리즘 계획을 올바르게 수립하는것이 중요했다. reader 함수와 writer 함수의 특징을 제대로 이해하고, 스레드를 제어하는 것이 기초적이지만 가장 중요했던 부분이라 생각한다.