컴퓨터학부 20160548 이승현 나반

1. 과제 개요

이 과제의 목표는 pthread 라이브러리를 이용한 쓰레드 프로그래밍을 통해 다중 쓰레드 프로그래밍의 기본 개념을 이해하는 것이다. 1번 문제에서는 lock과 조건변수를 이용하여 간단한 마스터-워커 쓰레드 풀을 구현한다. 2번 문제에서는 두가지 종류의 읽기-쓰기 lock을 구현한다. 3번 문제에서는 pthread 라이브러리를 이용하여 세마포어를 구현하고, 구현한 세마포어를 이용해 쓰레드간 동기화를 구현한다.

- **구현 방법**

1. Master-Worker Thread Pool 구현

master 쓰레드가 수행하는 generate_requests_loop() 함수를 다음과 같이 구현하였다. 무한 루프를 수행하며 mutex lock을 건 뒤 버퍼에 남은 공간이 있으면 다음 아이템을 집어넣는다. 아이템은 mutex lock을 건 뒤에만 접근해 쓰레드간 동기화 문제가 발생하지 않도록 했다. 버퍼에 공간이 없는 경우에는 pthread_cond_wait()으로 버퍼에 공간이 생길 때를 기다린다. 반복문의 마지막에는 다른 쓰레드에 signal을 보낸 뒤 wait을 호출해 다음 순서를 기다린다. 모든 아이템을 생성한 경우에는 다른 쓰레드에 signal을 보낸 뒤 반복을 종료하고 쓰레드를 종료한다.

worker 쓰레드가 수행하는 consume_requests_loop() 함수는 다음과 같이 구현하였다. 무한 루프를 수행하며 버퍼에 소비해야 할 아이템이 있는 경우에는 해당 아이템을 소비한다. 이 때에도 mutex를 이용하여 쓰레드간 동기화 문제가 발생하지 않도록 하였다. 버퍼가 비어서 소비할 아이템이 없을 경우에는 wait 상태로 들어가 대기한다. 모든 아이템을 소비했을 때에는 다른 쓰레드에 signal을 보낸 뒤 반복을 종료한다. 반복문의 마지막에는 다른 쓰레드에게 signal을 보낸 뒤 wait 상태로 들어간다.

위와 같이 master 쓰레드와 worker 쓰레드를 구현하여 master-worker thread pool을 구현하였다.

2. Reader-Writer Locks 쓰레드 프로그램 구현

두가지 종류의 read/write lock을 구현한다. 첫번째는 쓰기 쓰레드가 lock을 요청한 뒤에도 읽기, 쓰기 쓰레드가 모두 lock을 획득할 수 있는 lock이다. 이 경우에는 쓰기 쓰레드의 대기시간이 매우 길어질 수 있다. 두번째는 쓰기 쓰레드가 lock을 요청했으면 읽기 쓰레드는 lock을 획득할 수 없는 lock이다. 첫번째 lock은 rw_lock-r-test.c에, 두번째 lock은 rw_lock-w-test.c에 구현하였다.

struct rw_lock을 다음과 같이 정의하였다.

```
struct rw_lock
{
        pthread_rwlock_t rwlock;
        pthread_spinlock_t lock;
        int num_of_readers;
```

```
        int num_of_writers;
        int num_of_write_requests;
};
```

여기에서 pthread_rwlock_t rw_lock은 첫번째 lock에, pthread_spinlock_t lock은 두번째 lock에 사용한다. num_of_readers는 읽기 lock을 획득한 쓰레드의 수, num_of_writers는 쓰기 lock을 획득한 쓰레드의 수, num_of_write_requests는 쓰기 lock을 요청한 쓰레드의 수이다. 이 중 num_of_write_requests는 두번째 lock에서만 사용하고, 나머지 변수들은 두 lock에서 모두 사용한다.

첫번째 lock은 pthread_rwlock_t를 이용하여 구현하였다. pthread의 rwlock은 스케줄러의 종류에 따라 다르게 작동한다고 man page에 나와있다. 리눅스의 기본 스케줄러(CFS)에서는 우리가 구현해야 하는 첫번째 스케줄러와 동일하게 작동하기 때문에 pthread_rwlock_t를 그대로 가지고 구현하였다. 스케줄러를 RR 스케줄러 등으로 바꾸면 두번째 스케줄러와 같은 방식으로 작동하게 되지만, 두번째 스케줄러를 구현하는 데에는 pthread_rwlock_t를 사용하지 않고 그냥 spinlock을 이용하여 구현하였다.

두번째 스케줄러는 다음과 같이 구현하였다. 우선 lock 관리 구조체인 struct rw_lock에 접근할 때에는 쓰레드간 동기화 문제가 일어나지 않도록 하기 위하여 spinlock을 이용에 한번에 한 쓰레드만 접근할 수 있도록 하였다. r_lock() 함수에서는 쓰기 lock을 갖고 있는 쓰레드가 없고, 쓰기 lock을 요청한 쓰레드가 없을 때에만 읽기 lock을 획득할 수 있도록 구현하였다. w_lock() 함수에서는 일단 num_of_write_requests를 1 증가시켜 쓰기 lock 획득을 요청한 쓰레드가 있음을 표시하였다. 그 다음 읽기, 쓰기 lock을 갖고 있는 쓰레드가 하나도 없을 때 쓰기 lock을 획득하도록 하였다.

lock의 획득 및 해제의 상대적 순서를 확인하기 위하여 새로운 테스트 프로그램을 원래 테스트 용으로 주어졌던 테스트 프로그램과, 셸 스크립트를 일부 수정하여 작성하였다. 해당 테스트 프로그램에서는 쓰레드가 lock을 요청한 시점, 획득한 시점, 반납한 시점에 메시지를 출력한다. 출력된 메시지를 통해 제대로 동작하고 있는지 확인할 수 있다. 테스트 결과는 보고서의 "2.결과"에서 확인할 수 있다.

3. pthread를 이용한 사용자 수준 세마포어 (SSU_Sem) 구현

pthread 라이브러리를 이용하여 사용자 수준 세마포어를 구현하고, 구현한 세마포어를 이용하여 쓰레드가 차례대로 돌아가며 수행되는 SSU_Sem_toggle_test 프로그램을 구현한다.

```
  typedef struct SSU_Sem {
        pthread_mutex_t lock;
        pthread_cond_t cond;
        int count;
} SSU_Sem;
```

세마포어 SSU_Sem은 위와 같이 정의하여 세마포어의 count값이 정상적으로 증가/감소할 수 있도록 하였다.

SSU_Sem_down(), SSU_Sem_up() 함수는 위 구조체의 mutex와 조건변수를 이용하여 쓰레드간 동기화 문제없이 counter값을 증가, 감소시키도록 구현하였다.

SSU_Sem_toggle_test.c 에서는 정해진 쓰레드 수만큼 세마포어를 만들어 각각의 쓰레드가 자신만의 세마포어를 갖게 하였고, 그 값을 0으로 초기화 하였다. 쓰레드를 생성하여 justprint() 함수를 수행하게 되면 일단 SSU_Sem_down()을 호출하여 대기하게 된다. 메인 쓰레드에서 모든 쓰레드가 생성된 후에 첫번째 세마포어에 대하여 SSU_Sem_up()을 호출하여 첫번째 쓰레드가 수행되도록 한다. 각각의 쓰레드는 wait상태에서 깨어나 수행을 한 뒤 다음 쓰레드의 세마포어에 대하여 SSU_Sem_up()을 호출하여 다음 쓰레드가 수행되도록 한다 (예를 들면 1번 쓰레드가 수행을 마친 뒤에는 2번 쓰레드를 깨우고, 2번 쓰레드가 수행을 마친 뒤에는 3번 쓰레드를 깨운다). 다음 쓰레드를 깨운 뒤에는 본인의 세마포어를 1 감소시켜 대기한다. 이 과정을 반복하며 모든 쓰레드가 순차적으로 수행되도록 구현하였다.

2. 결과

- 1. Master-Worker Thread Pool 구현 test

```
shlee@shlee-virtual-machine:~/workspace/ssuos/project4/1$ ./test-master_worker.sh 100 100 3 5
OK. All test cases passed!
shlee@shlee-virtual-machine:~/workspace/ssuos/project4/1$ ./test-master_worker.sh 300 200 3 5
OK. All test cases passed!
shlee@shlee-virtual-machine:~/workspace/ssuos/project4/1$ ./test-master_worker.sh 100 10 3 5
OK. All test cases passed!
shlee@shlee-virtual-machine:~/workspace/ssuos/project4/1$ ./test-master_worker.sh 500 20 2 6
OK. All test cases passed!
```

여러 상황에서 정상적으로 작동하는 것 확인.

- 2. Reader-Writer Locks 쓰레드 프로그램 구현 test

```
shlee@shlee-virtual-machine:~/workspace/ssuos/project4/2$ ./rw_lock_test.sh
TESTSET: Running Testcases with reader test
CASE1: Reader Test with 5 reader and 1 writer
PASSED
CASE2: Reader Test with 5 reader and 3 writer
PASSED
CASE3: Reader Test with 5 reader and 5 writer
PASSED
TESTSET: Running Testcases with writer test
CASE1: Writer Test with 5 reader and 1 writer
PASSED
CASE2: Writer Test with 5 reader and 3 writer
PASSED
CASE3: Writer Test with 5 reader and 5 writer
PASSED
Test Cases Passed: 6
Test Cases Total: 6
shlee@shlee-virtual-machine:~/workspace/ssuos/project4/2$ ./rw_lock_test.sh
TESTSET: Running Testcases with reader test
CASE1: Reader Test with 5 reader and 1 writer
PASSED
CASE2: Reader Test with 5 reader and 3 writer
PASSED
CASE3: Reader Test with 5 reader and 5 writer
PASSED
TESTSET: Running Testcases with writer test
CASE1: Writer Test with 5 reader and 1 writer
PASSED
CASE2: Writer Test with 5 reader and 3 writer
PASSED
CASE3: Writer Test with 5 reader and 5 writer
PASSED
Test Cases Passed: 6
Test Cases Total: 6
shlee@shlee-virtual-machine:~/workspace/ssuos/project4/2$ ./rw_lock_test.sh
TESTSET: Running Testcases with reader test
CASE1: Reader Test with 5 reader and 1 writer
PASSED
CASE2: Reader Test with 5 reader and 3 writer
PASSED
CASE3: Reader Test with 5 reader and 5 writer
PASSED
TESTSET: Running Testcases with writer test
CASE1: Writer Test with 5 reader and 1 writer
PASSED
CASE2: Writer Test with 5 reader and 3 writer
PASSED
CASE3: Writer Test with 5 reader and 5 writer
PASSED
Test Cases Passed: 6
Test Cases Total: 6
```

테스트를 위해 세번 연속으로 실행했을 때 정상적으로 작동하는 것 확인.

- **lock의 획득 및 해제의 상대적 순서 확인**

```
shlee@shlee-virtual-machine:~/workspace/ssuos/project4/2$ ./my_rw_lock_test.sh
************** TESTSET: Running Testcases with READER TEST **************
Reader Test with 5 reader and 5 writer
Reader: 0 has requested the lock
Reader: 0 has acquired the lock
Reader: 1 has requested the lock
Reader: 1 has acquired the lock
Reader: 2 has requested the lock
Reader: 2 has acquired the lock
Writer: 0 has requested the lock ^
Writer: 1 has requested the lock ^
Reader: 3 has requested the lock
Reader: 3 has acquired the lock
Reader: 4 has requested the lock
Reader: 4 has acquired the lock
Writer: 2 has requested the lock ^
Writer: 3 has requested the lock ^
Writer: 4 has requested the lock ^
Reader: 5 has requested the lock
Reader: 5 has acquired the lock
Reader: 6 has requested the lock
Reader: 6 has acquired the lock
Reader: 7 has requested the lock
Reader: 7 has acquired the lock
Reader: 8 has requested the lock
Reader: 8 has acquired the lock
Reader: 9 has requested the lock
Reader: 9 has acquired the lock
Reader: 7 has released the lock
Reader: 0 has released the lock
Reader: 1 has released the lock
Reader: 3 has released the lock
Reader: 4 has released the lock
Reader: 2 has released the lock
Reader: 5 has released the lock
Reader: 6 has released the lock
Reader: 8 has released the lock
Writer: 0 has acquired the lock &
Reader: 9 has released the lock
Writer: 0 has released the lock
Writer: 1 has acquired the lock &
Writer: 1 has released the lock
Writer: 2 has acquired the lock &
Writer: 2 has released the lock
Writer: 3 has acquired the lock &
Writer: 3 has released the lock
Writer: 4 has acquired the lock &
Writer: 4 has released the lock
Reader 0 Lock Time: 0 Unlock Time: 11
```

➔ write lock 요청시에도 읽기 쓰레드가 lock 획득이 가능한 read/write lock test

➔ 쓰기 쓰레드가 write lock을 요청했음

➔ 쓰기 쓰레드가 write lock을 요청한 상태에서 읽기 쓰레드가 read lock을 획득함

➔ 읽기 쓰레드들은 동시에 read lock을 획득할 수 있음.

```
Reader 1 Lock Time: 1 Unlock Time: 12
Reader 2 Lock Time: 2 Unlock Time: 15
Reader 3 Lock Time: 3 Unlock Time: 13
Reader 4 Lock Time: 4 Unlock Time: 14
Reader 5 Lock Time: 5 Unlock Time: 16
Reader 6 Lock Time: 6 Unlock Time: 17
Reader 7 Lock Time: 7 Unlock Time: 10
Reader 8 Lock Time: 8 Unlock Time: 18
Reader 9 Lock Time: 9 Unlock Time: 19
Writer 0 Lock Time: 20 Unlock Time: 21
Writer 1 Lock Time: 22 Unlock Time: 23
Writer 2 Lock Time: 24 Unlock Time: 25
Writer 3 Lock Time: 26 Unlock Time: 27
Writer 4 Lock Time: 28 Unlock Time: 29
max_reader_acquire_time: 9
min_reader_release_time: 10
max_reader_release_time: 19
min_writer_acquire_time: 20
PASSED

************** TESTSET: Running Testcases with WRITER TEST **************
Writer Test with 5 reader and 5 writer
Reader: 0 has requested the lock
Reader: 0 has acquired the lock
Reader: 1 has requested the lock
Reader: 1 has acquired the lock
Reader: 2 has requested the lock
Reader: 2 has acquired the lock
Reader: 3 has requested the lock
Reader: 3 has acquired the lock
Writer: 0 has requested the lock ^
Writer: 1 has requested the lock ^
Writer: 2 has requested the lock ^
Writer: 3 has requested the lock ^
Writer: 4 has requested the lock ^
Reader: 4 has requested the lock
Reader: 9 has requested the lock
Reader: 8 has requested the lock
Reader: 7 has requested the lock
Reader: 6 has requested the lock
Reader: 5 has requested the lock
Reader: 0 has released the lock
Reader: 2 has released the lock
Reader: 3 has released the lock
Reader: 1 has released the lock
Writer: 4 has acquired the lock &
Writer: 0 has acquired the lock &
Writer: 4 has released the lock
Writer: 0 has released the lock
Writer: 1 has acquired the lock &
Writer: 1 has released the lock
Writer: 2 has acquired the lock &
```

➔ write lock 요청시에는 읽기 쓰레드가 lock 획득 불가능한 read/write lock test

➔ 쓰기 쓰레드가 write lock을 요청했음

➔ 쓰기 쓰레드가 write lock을 요청한 상태에서 읽기 쓰레드가 read lock을 요청하지만 획득하지는 못함.

```
Writer: 2 has released the lock
Writer: 3 has acquired the lock &
Writer: 3 has released the lock
Reader: 5 has acquired the lock
Reader: 4 has acquired the lock
Reader: 7 has acquired the lock
Reader: 8 has acquired the lock
Reader: 6 has acquired the lock
Reader: 9 has acquired the lock
Reader: 5 has released the lock
Reader: 7 has released the lock
Reader: 9 has released the lock
Reader: 6 has released the lock
Reader: 8 has released the lock
Reader: 4 has released the lock
Reader 0 Lock Time: 0 Unlock Time: 4
Reader 1 Lock Time: 1 Unlock Time: 7
Reader 2 Lock Time: 2 Unlock Time: 5
Reader 3 Lock Time: 3 Unlock Time: 6
Reader 4 Lock Time: 22 Unlock Time: 28
Reader 5 Lock Time: 21 Unlock Time: 24
Reader 6 Lock Time: 20 Unlock Time: 26
Reader 7 Lock Time: 23 Unlock Time: 29
Reader 8 Lock Time: 19 Unlock Time: 27
Reader 9 Lock Time: 18 Unlock Time: 25
Writer 0 Lock Time: 10 Unlock Time: 11
Writer 1 Lock Time: 12 Unlock Time: 13
Writer 2 Lock Time: 14 Unlock Time: 15
Writer 3 Lock Time: 16 Unlock Time: 17
Writer 4 Lock Time: 8 Unlock Time: 9
28
18
8
17
PASSED

************** TEST END **************
```

➔ 모든 쓰기 쓰레드가 요청했던 write lock을 획득, 반납하고 나서야 읽기 쓰레드가 lock을 획득함.

또한 읽기 쓰레드들은 동시에 read lock을 획득하고 있음.

위와 같이 요구사항 대로 두 종류의 read/write lock을 구현한 것을 확인함.

- 3. pthread를 이용한 사용자 수준 세마포어(SSU_Sem) 구현 test

```
shlee@shlee-virtual-machine:~/workspace/ssuos/project4/3$ ./SSU_Sem_test.sh
Running SSU_Sem_test.c
This is main thread. This should print first
This is thread 1
One thread has printed
This is thread 2
Second thread has printed

Running SSU_Sem_toggle_test.c
This is thread 0
This is thread 1
This is thread 2
This is thread 0
This is thread 1
This is thread 2
This is thread 0
This is thread 1
This is thread 2
This is thread 0
This is thread 1
This is thread 2
This is thread 0
This is thread 1
This is thread 2
This is thread 0
This is thread 1
This is thread 2
This is thread 0
This is thread 1
This is thread 2
This is thread 0
This is thread 1
This is thread 2
This is thread 0
This is thread 1
This is thread 2
This is thread 0
This is thread 1
This is thread 2
This is thread 0
This is thread 1
This is thread 2
shlee@shlee-virtual-machine:~/workspace/ssuos/project4/3$
```

SSU_Sem_test에서 메인 쓰레드가 먼저 출력되는 것, SSU_Sem_toggle_test에서 차례대로 돌아가며 출력되는 것 확인.

3. 소스코드 (수정된 부분만)

- **1. Master-Worker Thread Pool 구현**

- **master-worker.c**

```
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER; // mutex 선언, 초기화
pthread_cond_t cond = PTHREAD_COND_INITIALIZER; // 조건변수 선언, 초기화

//produce items and place in buffer
//modify code below to synchronize correctly
void *generate_requests_loop(void *data)
{
	int thread_id = *((int *)data);

	while(1)
	{
		pthread_mutex_lock(&lock); // mutex lock

		if(item_to_produce >= total_items) { // 아이템 모두 만들었으면
			pthread_cond_signal(&cond); // send signal
			pthread_mutex_unlock(&lock); // mutex unlock
			break;
		}

		if (curr_buf_size < max_buf_size) { // 버퍼에 빈 공간이 있을 때
			buffer[curr_buf_size++] = item_to_produce; // 새로운 아이템을 버퍼에 넣음
			print_produced(item_to_produce, thread_id);
			item_to_produce++;
		}
```

```
                pthread_cond_signal(&cond); // send signal
                pthread_cond_wait(&cond, &lock); // wait
                pthread_mutex_unlock(&lock); // mutex unlock
        }
        return 0;
}

void *consume_requests_loop(void *data)
{
        int thread_id = *((int *)data);

        while(1)
        {
                pthread_mutex_lock(&lock); // mutex lock

                if(item_to_consume >= total_items) { // 모든 아이템을 소비했으면
                        pthread_cond_signal(&cond); // send signal
                        pthread_mutex_unlock(&lock); // mutex unlock
                        break;
                }

                if (0 < curr_buf_size) { // 버퍼가 비어있지 않다면
                        print_consumed(buffer[--curr_buf_size], thread_id); // 아이템 하나 소비
                        ++item_to_consume;
                }
```

```
                pthread_cond_signal(&cond); // send signal
                pthread_cond_wait(&cond, &lock); // wait
                pthread_mutex_unlock(&lock); // mutex unlock
        }
        return 0;
}

//write function to be run by worker threads
//ensure that the workers call the function print_consumed when they consume an item

int main(int argc, char *argv[])
{
        int *master_thread_id;
        int *worker_thread_id;
        pthread_t *master_thread;
        pthread_t *worker_thread;
        item_to_produce = 0;
        curr_buf_size = 0;

        int i;

        if (argc < 5) {
                printf("./master-worker #total_items #max_buf_size #num_workers #masters e.g. ./exe 10000 1000 4 3₩n");
                exit(1);
        }
        else {
                num_masters = atoi(argv[4]);
```

```
        num_workers = atoi(argv[3]);
        total_items = atoi(argv[1]);
        max_buf_size = atoi(argv[2]);
}

buffer = (int *)malloc (sizeof(int) * max_buf_size);

//create master producer threads
master_thread_id = (int *)malloc(sizeof(int) * num_masters);
master_thread = (pthread_t *)malloc(sizeof(pthread_t) * num_masters);
for (i = 0; i < num_masters; i++)
        master_thread_id[i] = i;

for (i = 0; i < num_masters; i++)
        pthread_create(&master_thread[i], NULL, generate_requests_loop, (void *)&master_thread_id[i]);

//create worker consumer threads
worker_thread_id = (int *)malloc(sizeof(int) * num_workers);
worker_thread = (pthread_t *)malloc(sizeof(pthread_t) * num_workers);
for (i = 0; i < num_workers; i++)
        worker_thread_id[i] = i;

for (i = 0; i < num_workers; i++)
        pthread_create(&worker_thread[i], NULL, consume_requests_loop, (void *)&worker_thread_id[i]);

//wait for all threads to complete
```

```
	for (i = 0; i < num_masters; i++)
	{
		pthread_join(master_thread[i], NULL);
		printf("master %d joined₩n", i);
	}

	for (i = 0; i < num_workers; i++)
	{
		pthread_join(worker_thread[i], NULL);
		printf("worker %d joined₩n", i);
	}

	/*----Deallocating Buffers---------------------*/
	free(buffer);
	free(master_thread_id);
	free(master_thread);
	free(worker_thread_id);
	free(worker_thread);

	pthread_mutex_destroy(&lock);
	pthread_cond_destroy(&cond);

	return 0;
}
```

- **2. Reader-Writer Locks 쓰레드 프로그램 구현**

- **rw_lcok.h**

```
struct rw_lock
{
	pthread_rwlock_t rwlock; // 첫 번째 방법에 사용할 rwlock - pthread_rwlock을 그대로 사용한다
	pthread_spinlock_t lock; // 두 번째 방법에 사용할 spinlock
	int num_of_readers; // 읽기 작업 하고있는 쓰레드의 개수
	int num_of_writers; // 쓰기 작업 하고있는 쓰레드의 개수
	int num_of_write_requests; // 쓰기 lock 요청한 쓰레드의 개수 - 두 번째 방법에 사용
};
```

- **rw_lock-r-test.c**

```
void init_rwlock(struct rw_lock * rw)
{
	//	Write the code for initializing your read-write lock.
	pthread_rwlock_init(&(rw->rwlock), NULL); // rwlock 초기화
	rw->num_of_readers = 0; // 멤버 변수 초기화
	rw->num_of_writers = 0; // 멤버 변수 초기화

	return;
}

void r_lock(struct rw_lock * rw)
{
	//	Write the code for aquiring read-write lock by the reader.
	pthread_rwlock_rdlock(&(rw->rwlock)); // read lock
```

```
	return;
}

void r_unlock(struct rw_lock * rw)
{
	//	Write the code for releasing read-write lock by the reader.
	pthread_rwlock_unlock(&(rw->rwlock)); // read unlock
	return;
}

void w_lock(struct rw_lock * rw)
{
	//	Write the code for aquiring read-write lock by the writer.
	pthread_rwlock_wrlock(&(rw->rwlock)); // write lock
	return;
}

void w_unlock(struct rw_lock * rw)
{
	//	Write the code for releasing read-write lock by the writer.
	pthread_rwlock_unlock(&(rw->rwlock)); // unlock rwlock
	return;
}
```

- **rw_lock-w-test.c**

```
void init_rwlock(struct rw_lock * rw)
```

```
{
	//	Write the code for initializing your read-write lock.
	pthread_spin_init(&(rw->lock), 0); // spinlock 초기화
	rw->num_of_readers = 0; // 멤버 변수 초기화
	rw->num_of_writers = 0;  // 멤버 변수 초기화
	rw->num_of_write_requests = 0; // 멤버 변수 초기화
}

void r_lock(struct rw_lock * rw)
{
	//	Write the code for aquiring read-write lock by the reader.
	while (1) {
		if (rw->num_of_writers <= 0 && rw->num_of_write_requests <= 0) { // 쓰기 중인 쓰레드, 쓰기 요청한 쓰레드 없다면
			pthread_spin_lock(&(rw->lock)); // lock
			++(rw->num_of_readers); // 읽기 중인 쓰레드 개수 + 1
			pthread_spin_unlock(&(rw->lock)); // unlock
			break; // 반복 종료
		}

		sched_yield(); // 다른 쓰레드에게 넘김
	}
}

void r_unlock(struct rw_lock * rw)
{
	//	Write the code for releasing read-write lock by the reader.
```

```
	pthread_spin_lock(&(rw->lock)); // lock
	--(rw->num_of_readers); // 읽기 중인 쓰레드 개수 - 1
	pthread_spin_unlock(&(rw->lock)); // unlock
}

void w_lock(struct rw_lock * rw)
{
	//	Write the code for aquiring read-write lock by the writer.
	pthread_spin_lock(&(rw->lock)); // lock
	++(rw->num_of_write_requests); // 쓰기 요청한 쓰레드 개수 + 1
	pthread_spin_unlock(&(rw->lock)); // unlock

	while (1) {
		if (rw->num_of_readers <= 0 && rw->num_of_writers <= 0) { // 읽기, 쓰기 중인 쓰레드가 없다면
			pthread_spin_lock(&(rw->lock)); // lock
			--(rw->num_of_write_requests); // 쓰기 요청한 쓰레드 개수 - 1
			++(rw->num_of_writers); // 쓰기 작업중인 쓰레드 개수 + 1
			pthread_spin_unlock(&(rw->lock)); // lock
			break; // 반복 종료
		}
		sched_yield(); // 다른 쓰레드에게 넘김
	}
}

void w_unlock(struct rw_lock * rw)
{
	//	Write the code for releasing read-write lock by the writer.
```

```
        pthread_spin_lock(&(rw->lock)); // lock
        --(rw->num_of_writers); // 쓰기 작업 중인 쓰레드 개수 - 1
        pthread_spin_unlock(&(rw->lock)); // unlock
}
```

- **my_rw_lock_test.sh**

```
echo "************** TESTSET: Running Testcases with READER TEST **************"
gcc my_reader_test.c rw_lock-r-test.c rw_lock.c -o rw_lock-r-test -lpthread

echo "Reader Test with 5 reader and 5 writer"
out=`./rw_lock-r-test 5 5`
echo -e $out

echo "************** TESTSET: Running Testcases with WRITER TEST ************** "
gcc my_writer_test.c rw_lock-w-test.c rw_lock.c -o rw_lock-w-test -lpthread

echo "Writer Test with 5 reader and 5 writer"
out=`./rw_lock-w-test 5 5`
echo -e $out

echo "************** TEST END ************** "
```

- **my_reader_test.c (reader_test.c에서 수정한 부분만)**

```
...
void *Reader(void* arg)
{
```

```
	int threadNUmber = *((int *)arg);

	// Occupying the Lock
	printf("Reader: %d has requested the lock₩₩n", threadNUmber);
	r_lock(&rwlock);

	pthread_spin_lock(&spinlock);
	readerAcquireTime[threadNUmber] = indx;
	indx++;
	pthread_spin_unlock(&spinlock);

	printf("Reader: %d has acquired the lock₩₩n", threadNUmber);
	usleep(10000);

	pthread_spin_lock(&spinlock);
	readerReleaseTime[threadNUmber] = indx;
	indx++;
	pthread_spin_unlock(&spinlock);

	// Releasing the Lock
	r_unlock(&rwlock);
	printf("Reader: %d has released the lock₩₩n",threadNUmber);
}

void *Writer(void* arg)
{
	int threadNUmber = *((int *)arg);
```

```
        // Occupying the Lock
        printf("Writer: %d has requested the lock ^₩₩n", threadNUmber);
        w_lock(&rwlock);

        pthread_spin_lock(&spinlock);
        writerAcquireTime[threadNUmber] = indx;
        indx++;
        pthread_spin_unlock(&spinlock);

        printf("Writer: %d has acquired the lock &₩₩n",threadNUmber);
        usleep(10000);

        pthread_spin_lock(&spinlock);
        writerReleaseTime[threadNUmber] = indx;
        indx++;
        pthread_spin_unlock(&spinlock);

        // Releasing the Lock
        w_unlock(&rwlock);
        printf("Writer: %d has released the lock₩₩n",threadNUmber);
}

int main(int argc, char *argv[])
{

        int *threadNUmber;
```

```
        pthread_t *threads;

        setbuf(stdout, NULL); // 정확한 결과 확인을 위해 버퍼링 없이 바로 출력.
...
```

- **my_writer_test.c (writer_test.c에서 수정한 부분만)**

```
...
void *Reader(void* arg)
{
        int threadNUmber = *((int *)arg);

        // Occupying the Lock
        printf("Reader: %d has requested the lock₩₩n", threadNUmber);
        r_lock(&rwlock);

        pthread_spin_lock(&spinlock);
        readerAcquireTime[threadNUmber] = indx;
        indx++;
        pthread_spin_unlock(&spinlock);

        printf("Reader: %d has acquired the lock₩₩n", threadNUmber);
        usleep(100000);

        pthread_spin_lock(&spinlock);
        readerReleaseTime[threadNUmber] = indx;
        indx++;
```

```
        pthread_spin_unlock(&spinlock);

        // Releasing the Lock
        r_unlock(&rwlock);
        printf("Reader: %d has released the lock₩₩n",threadNUmber);
}

void *Writer(void* arg)
{
        int threadNUmber = *((int *)arg);

        // Occupying the Lock
        printf("Writer: %d has requested the lock ^₩₩n", threadNUmber);
        w_lock(&rwlock);

        pthread_spin_lock(&spinlock);
        writerAcquireTime[threadNUmber] = indx;
        indx++;
        pthread_spin_unlock(&spinlock);

        printf("Writer: %d has acquired the lock &₩₩n",threadNUmber);
        usleep(100000);

        pthread_spin_lock(&spinlock);
        writerReleaseTime[threadNUmber] = indx;
        indx++;
        pthread_spin_unlock(&spinlock);
```

```
        // Releasing the Lock
        w_unlock(&rwlock);
        printf("Writer: %d has released the lock₩₩n",threadNUmber);
}

int main(int argc, char *argv[])
{

        int *threadNUmber;
        pthread_t *threads;

        setbuf(stdout, NULL); // 정확한 결과 확인을 위해 버퍼링 없이 바로 출력.
…
```

- **3. pthread를 이용한 사용자 수준 세마포어(SSU_Sem) 구현**

- **SSU_Sem.h**

```
#include <pthread.h>
```

```
typedef struct SSU_Sem {
        pthread_mutex_t lock; // count 변수 접근 시에 사용할 mutex
        pthread_cond_t cond; // count 변수 접근 시에 사용할 조건변수
        int count;
} SSU_Sem;
```

- **SSU_Sem.c**

```
void SSU_Sem_init(SSU_Sem *s, int value) {
        pthread_mutex_init(&(s->lock), NULL); // mutex 초기화
        pthread_cond_init(&(s->cond), NULL); // 조건변수 초기화
        pthread_mutex_lock(&(s->lock)); // lock
        s->count = value; // 세마포어의 count 값 초기화
        pthread_mutex_unlock(&(s->lock)); // unlock
}

void SSU_Sem_down(SSU_Sem *s) {
        pthread_mutex_lock(&(s->lock)); // lock
        --(s->count); // count 1 감소시킴
        if (s->count < 0) { // count가 음수라면
                pthread_cond_wait(&(s->cond), &(s->lock)); // wait
        }
        pthread_mutex_unlock(&(s->lock)); // unlock
}

void SSU_Sem_up(SSU_Sem *s) {
        pthread_mutex_lock(&(s->lock)); // lock
```

```
        ++(s->count); // count 1 증가시킴

        if (s-> count <= 0) { // 대기중인 쓰레드가 있다면
                pthread_mutex_unlock(&(s->lock)); // unlock
                pthread_cond_signal(&(s->cond)); // send signal
        } else {
                pthread_mutex_unlock(&(s->lock)); // unlock
        }

        return;
}
```

- **SSU_Sem_toggle_test.c**

```
SSU_Sem *sems; // 쓰레드당 하나씩 사용할 세마포어들을 저장할 배열의 인덱스

void *justprint(void *data)
{
        int thread_id = *((int *)data);
        int next_thread = (thread_id + 1) % NUM_THREADS; // 다음에 실행할 쓰레드의 thread_id를 구함

        for(int i = 0; i < NUM_ITER; i++)
        {
                SSU_Sem_down(sems + thread_id); // 세마포어 1 감소시켜 다른 쓰레드가 깨울 때 까지 wait
                printf("This is thread %d₩n", thread_id); // 현재 쓰레드 id 출력
                SSU_Sem_up(sems + next_thread); // 다음 쓰레드의 세마포어 1 증가시켜 다음 쓰레드를 수행하도록 함
        }
        return 0;
```

```
}

int main(int argc, char *argv[])
{
        pthread_t mythreads[NUM_THREADS];
        int mythread_id[NUM_THREADS];

        sems = (SSU_Sem *) malloc (sizeof(SSU_Sem) * NUM_THREADS); // 쓰레드 개수만큼의 세마포어를 동적 할당한다
        for (int i = 0; i < NUM_THREADS; ++i) { // 쓰레드 총 개수만큼 반복하며 세마포어를 초기화
                SSU_Sem_init(sems + i, 0); // 세마포어 count 0으로 초기화
        }

        for(int i =0; i < NUM_THREADS; i++)
        {
                mythread_id[i] = i;
                pthread_create(&mythreads[i], NULL, justprint, (void *)&mythread_id[i]);
        }
        SSU_Sem_up(sems); // 첫번째 세마포어의 count를 1 증가시켜 첫번째 쓰레드를 동작시킨다

        for(int i =0; i < NUM_THREADS; i++)
        {
                pthread_join(mythreads[i], NULL);
        }
        return 0;
}
```