Operating System 과제

학번: 12170584

이름: 이진호

**문제 정의:**

* ➢Perform write and read operations on same data array
* ➢If two threads should access the same array
* ➢If any of these threads are writing data in the array, no other thread should be to read or write data
* ➢If there are no writes on the array, both threads should be able to read concurrently

**소스 코드:**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

#include <semaphore.h>

#include <sched.h>

//define the rwlock struct

typedef struct \_rwlock\_t {

sem\_t writelock;

sem\_t lock;

int readers;

} rwlock\_t;

//rwlock init function

void rwlock\_init(rwlock\_t \*lock) {

lock->readers = 0;

sem\_init(&lock->lock,0,1);

sem\_init(&lock->writelock,0,1);

}

//rwlock acquire readlock function

void rwlock\_acquire\_readlock(rwlock\_t \*lock) {

sem\_wait(&lock->lock);

lock->readers++;

if (lock->readers == 1)

sem\_wait(&lock->writelock);

sem\_post(&lock->lock);

}

//rwlock release readlock function

void rwlock\_release\_readlock(rwlock\_t \*lock) {

sem\_wait(&lock->lock);

lock->readers--;

if (lock->readers == 0)

sem\_post(&lock->writelock);

sem\_post(&lock->lock);

}

//rwlock acquire writelock function

void rwlock\_acquire\_writelock(rwlock\_t \*lock) {

sem\_wait(&lock->writelock);

}

//rwlock release writelock function

void rwlock\_release\_writelock(rwlock\_t \*lock) {

sem\_post(&lock->writelock);

}

int read\_loops;

int write\_loops;

int counter = 0;

rwlock\_t mutex;

//reader function

void \*reader(void \*arg) {

int i;

int local = 0;

int thread\_numb = \*(int \*)arg; //save current thread number

for (i = 0; i < read\_loops; i++) {

rwlock\_acquire\_readlock(&mutex); //readlock acquire

sleep(rand()%5); //sleep random time 0~5

local = counter; //critical section

rwlock\_release\_readlock(&mutex); //readlock release

printf("thread%d-read %d\n",thread\_numb,local); //print thread num, local

}

printf("read%d done: %d\n",thread\_numb ,local);

return NULL;

}

//writer function

void \*writer(void \*arg) {

int i;

int thread\_numb = \*(int \*)arg; //save current thread number

for (i = 0; i < write\_loops; i++) {

rwlock\_acquire\_writelock(&mutex); //writelock acquire

sleep(rand()%5); //sleep random time 0-5

counter++; //critical section

printf("thread%d-write %d\n",thread\_numb,counter); //print thread num, counter

rwlock\_release\_writelock(&mutex); //writelock release

}

printf("write done\n");

return NULL;

}

//main function

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

//exception

if (argc != 3) {

fprintf(stderr, "usage: rwlock readloops writeloops\n");

exit(1);

}

read\_loops = atoi(argv[1]); //save read loop

write\_loops = atoi(argv[2]); //save write loop

int thread\_numb[2] = {1,2}; //define thread number

rwlock\_init(&mutex); //rwlock initialize

pthread\_t c1, c2;

//create 1 reader , 1 writer thread

printf("1 reader, 1 writer\n");

pthread\_create(&c1, NULL, reader,thread\_numb);

pthread\_create(&c2, NULL, writer,thread\_numb+1);

pthread\_join(c1, NULL);

pthread\_join(c2, NULL);

printf("==============done==============\n");

//create 2 reader thread

printf("2 reader\n");

pthread\_create(&c1,NULL,reader,thread\_numb);

pthread\_create(&c2,NULL,reader,thread\_numb+1);

pthread\_join(c1,NULL);

pthread\_join(c2,NULL);

printf("===========done==============\n");

//create 2 writer thread

printf("2 writer\n");

pthread\_create(&c1,NULL,writer,thread\_numb);

pthread\_create(&c2,NULL,writer,thread\_numb+1);

pthread\_join(c1,NULL);

pthread\_join(c2,NULL);

printf("===========done==============\n");

printf("all done\n");

return 0;

}

**출력 결과:**

스크린샷, 텍스트이(가) 표시된 사진

자동 생성된 설명

결론 - read,write lock을 구현하기 위해서 7개의 함수를 구성하였다. 먼저 rwlock을 초기화 하기 위한 함수를 구성하였고, readlock을 acquire하는 함수와 release 하는 함수, writelock을 acquire하는 함수와 release 하는 함수를 구현하였다. 여기에 각각 thread 생성시 critical section의 값을 read를 하는 역할과 critical section의 값을 write하는 역할을 하는 reader, writer 함수를 구성하였다. 이를 이용하여 , reader 1 writer 1을 동작하는 스레드 실행, reader 2 가 동작하는 스레드 실행, writer 2 가 동작하는 스레드를 실행 하였고, 결과는 writer 를 실행 할때는 다른 스레드가 lock이 걸려 동작하지 않았고 reader 2개가 스레드 동작할 때는 두개가 동시에 실행 되는 것을 확인 할 수 있었다.