HW8. Concurrency Practice

학번: 20170404

이름: 한종수

제출일: 2021. 5. 25

1.과제 개요

Counter 와 Bounded Buffer 를 Lock 을 사용하여 올바르게 구현하는 프로그램을 만든다.

2.소스코드

(수정한 코드를 캡처하고 간단히 설명함)

1) counter.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

volatile int counter = 0;
int loops;

void *worker(void *arg)
{
    int i;
    for (i = 0; i < loops; i++)
    {
        counter++;
    }

    return NULL;
}

int main(int argc, char *argv[])
{
    if (argc != 2)
    {
        fprintf(stderr, "usage : threads <value>\n");
        exit(1);
    }

    loops = atoi(argv[1]);
    pthread_t p1, p2;
    printf("Initial value : %d\n", counter);

    pthread_create(&p1, NULL, worker, NULL);
    pthread_create(&p2, NULL, worker, NULL);
    pthread_join(p1, NULL);
    pthread_join(p2, NULL);
    printf("Final value : %d\n", counter);

    return 0;
}
```

main에서 thread 두 개를 생성하여 각 thread가 worker 함수를 실행한다. worker 함수에서는 counter 공유변수를 loops 수 만큼 증가시킨다. counter 변수는 lock이 없어 mutual exclusion이 보장이 되지 않는다.

2) counter_lock.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <pthread.h>

volatile int counter = 0;
int loops;

void *worker(void *arg)
{
    int i;
    pthread_mutex_lock((pthread_mutex_t *)arg);
    for (i = 0; i < loops; i++)
    {
        counter++;
    }
    pthread_mutex_unlock((pthread_mutex_t *)arg);

    return NULL;
}

int main(int argc, char *argv[])
{
    if (argc != 2)
    {
        fprintf(stderr, "usage : threads <value>\n");
        exit(1);
    }

    loops = atoi(argv[1]);
    pthread_t p1, p2;
    printf("Initial value : %d\n", counter);

    pthread_mutex_t lock;
    int rc = pthread_mutex_init(&lock, NULL);
    assert(rc == 0);

    pthread_create(&p1, NULL, worker, &lock);
    pthread_create(&p2, NULL, worker, &lock);
    pthread_join(p1, NULL);
    pthread_join(p2, NULL);
    printf("Final value : %d\n", counter);

    return 0;
}
```

main 에서 lock을 선언하고 초기화 시킨다. 그리고 lock을 thread의 인자로 넘겨준다.

worker는 lock을 받아 lock에 대한 권한을 얻은 상태에서 counter를 증가시켜 shared variable에 대한 mutual exclusion을 보장한다.

3) bounded_buffer.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <pthread.h>

#define MAX 5

int buffer[MAX];
int fill = 0;
int use = 0;
int count = 0;
int psum = 0;
int csum = 0;

void put(int value)
{
    buffer[fill] = value;
    fill = (fill + 1) % MAX;
    count++;
    psum += (value - fill);
}

int get()
{
    int tmp = buffer[use];
    use = (use + 1) % MAX;
    count--;
    csum += (tmp - use);

    return tmp;
}

int loops = 0;
pthread_mutex_t _empty = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t _fill = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t _mutex = PTHREAD_COND_INITIALIZER;

void *producer(void *arg)
{
    int i;
    for (i = 0; i < loops; i++)
    {
        pthread_mutex_lock(&_mutex);
        while (count == MAX)
        {
            pthread_cond_wait(&_empty, &_mutex);
        }
        put(i);
        pthread_cond_signal(&_fill);
        pthread_mutex_unlock(&_mutex);
    }
}
```

```c
void *consumer(void *arg)
{
    int i;
    for (i = 0; i < loops; i++)
    {
        pthread_mutex_lock(&_mutex);
        while (count == 0)
        {
            pthread_cond_wait(&_fill, &_mutex);
        }
        int tmp = get();
        pthread_cond_signal(&_empty);
        pthread_mutex_unlock(&_mutex);
        printf("%d\n", tmp);
    }
}

int main(int argc, char *argv[])
{
    if (argc != 2)
    {
        fprintf(stderr, "usage : threads <value>\n");
        exit(1);
    }
    loops = atoi(argv[1]);
    pthread_t p1, p2, p3, p4;

    pthread_create(&p1, NULL, producer, NULL);
    pthread_create(&p2, NULL, producer, NULL);
    pthread_create(&p3, NULL, consumer, NULL);
    pthread_create(&p4, NULL, consumer, NULL);
    pthread_join(p1, NULL);
    pthread_join(p2, NULL);
    pthread_join(p3, NULL);
    pthread_join(p4, NULL);

    if (psum == csum)
    {
        printf("Test Ok\n");
    }
    else
    {
        printf("Wrong\n");
    }

    return 0;
}
```

Max의 크기를 5로 정의하고 buffer의 크기를 그에 맞게 설정한다.

그리고 나머지 공유변수들 fill, use, count, psum, csum 을 초기화한다.

put, get 함수는 buffer에 데이터를 넣고 뺀다. put은 count, fill 를 증가시키고, get은 count를 감소시키고 use를 증가시킨다. psum, csum에는 총 생산, 소비 횟수가 기록된다.

producer는 loop를 돌며 loops수 만큼 데이터를 생산한다. 이때 condition variable에 대한 lock 을 이용하여 mutual exculsion을 보장한다. producer는 empty queue에서 consumer 가 데이터를 소비하는 것을 기다린다. 일을 마친 후 consumer 가 대기하는 fill queue 에서 consumer 하나를 깨운다.

consumer는 loop를 돌며 loops수 만큼 데이터를 소비한다. 이때 condition variable에 대한 lock 을 이용하여 mutual exculsion을 보장한다. consumer는 fill queue에서 producer 가 데이터를 생산하는 것을 기다린다. 일을 마친 후 producer 가 대기하는 empty queue 에서 producer 하나를 깨운다.

main에서 producer, consumer thread를 각각 두개 생성한다. 실행 이후 psum csum이 같으면 mutual exclusion이 잘 보장된 것이다.

3.결과

(테스트 실행결과를 캡처)

1) counter.c 의 잘못된 결과

```
jongsoo@DESKTOP:~/2021_OperatingSystem/Lab8_Concurrency/src$ ./thread 100000
Initial value : 0
Final value : 136388
jongsoo@DESKTOP:~/2021_OperatingSystem/Lab8_Concurrency/src$
```

2) counter_lock.c 의 올바른 결과

```
jongsoo@DESKTOP:~/2021_OperatingSystem/Lab8_Concurrency/src$ ./thread 100000
Initial value : 0
Final value : 200000
jongsoo@DESKTOP:~/2021_OperatingSystem/Lab8_Concurrency/src$
```

3) bounded_buffer.c 의 올바른 결과

```
jongsoo@DESKTOP:~/2021_OperatingSystem/Lab8_Concurrency/src$ ./thread 10
0
1
2
3
4
5
6
7
0
8
1
2
3
9
4
5
6
7
8
9
Test Ok
jongsoo@DESKTOP:~/2021_OperatingSystem/Lab8_Concurrency/src$
```