**Synchronization**

오퍼레이팅 시스템

학 번 12191659

이 름 이화연

소속학과 컴퓨터공학과

교 수 님 정진만 교수님

제출일자 2021.06.03

1. **개발환경**

리눅스 배포판은 ubuntu-20.04.2.0을 사용하였고 가상머신은 Oracle VM VirtualBOX를 사용하였다.

1. **쉘 프로그램 설계 및 구현 내용**

설계 및 구현 내용은 하단의 코드와 주석으로 설명하겠다.

**2-1. synch\_pthread.c**

#include <pthread.h>

#include <stdio.h>

#define ITER 1000

void \*thread\_increment(void \*arg);

void \*thread\_decrement(void \*arg);

pthread\_mutex\_t mutex;

pthread\_cond\_t cond\_space;

pthread\_cond\_t cond\_data;

int x;

int main() {

    pthread\_t tid1, tid2;

    pthread\_mutex\_init(&mutex, NULL);

    pthread\_cond\_init(&cond\_space, NULL);

    pthread\_cond\_init(&cond\_data, NULL);

    pthread\_create(&tid1, NULL, thread\_increment, NULL);

    pthread\_create(&tid2, NULL, thread\_decrement, NULL);

    pthread\_join(tid1, NULL);

    pthread\_join(tid2, NULL);

    if(x != 0) printf("BOOM! counter=%d\n", x);

    else       printf("OK counter=%d\n", x);

    return 0;

}

// 생산자

void \*thread\_increment(void \*arg) {

    int i, val;

    for(i=0; i<ITER; i++) {

        // critical section에 접근하기 전에 mutex lock을 걸어준다.

        pthread\_mutex\_lock(&mutex);

        // 버퍼의 개수(30)가 꽉 찼다면 소비자가 하나 소비할 때까지 기다린다.

        if(x == 30) pthread\_cond\_wait(&cond\_space, &mutex);

        // 생산

        val = x;

        printf("%u: %d\n", (unsigned int)pthread\_self(), val);

        x = val + 1;

        // 버퍼가 비어있지 않다는 신호를 보낸다.

        pthread\_cond\_signal(&cond\_data);

        // mutex lock을 푼다.

        pthread\_mutex\_unlock(&mutex);

    }

    return NULL;

}

// 소비자

void \*thread\_decrement(void \*arg) {

    int i, val;

    for(i=0; i<ITER; i++) {

        // critical section에 접근하기 전에 mutex lock을 걸어준다.

        pthread\_mutex\_lock(&mutex);

        // 버퍼의 개수(30)가 비어있다면 생산자가 하나 생산할 때까지 기다린다.

        if(x == 0) pthread\_cond\_wait(&cond\_data, &mutex);

        // 소비

        val = x;

        printf("%u: %d\n", (unsigned int)pthread\_self(), val);

        x = val - 1;

        //버퍼가 꽉차지 않았다는 신호를 보낸다.

        pthread\_cond\_signal(&cond\_space);

        // mutex lock을 푼다.

        pthread\_mutex\_unlock(&mutex);

    }

    return NULL;

}

**2-2. synch\_semaphore.c**

#include <pthread.h>

#include <stdio.h>

#include <semaphore.h>

#define ITER 1000

void \*thread\_increment(void \*arg);

void \*thread\_decrement(void \*arg);

sem\_t empty;

sem\_t full;

int x;

int main() {

    pthread\_t tid1, tid2;

    // 세마포어 초기값을 30으로 정의

    sem\_init(&empty, 0, 30);

    // 세마포어 초기값을 0으로 정의

    sem\_init(&full, 0, 0);

    pthread\_create(&tid1, NULL, thread\_increment, NULL);

    pthread\_create(&tid2, NULL, thread\_decrement, NULL);

    pthread\_join(tid1, NULL);

    pthread\_join(tid2, NULL);

    if(x != 0) printf("BOOM! counter=%d\n", x);

    else       printf("OK counter=%d\n", x);

    sem\_destroy(&empty);

    sem\_destroy(&full);

    return 0;

}

void \*thread\_increment(void \*arg) {

    int i, val;

    for(i=0; i<ITER; i++) {

        // 세마포어값을 1 감소시킨다. 값이 0인 경우 1이상이 될 때까지 기다리다가 1이상이 되면 값을 감소시키고 대기상태에서 빠져나온다.

        // 즉, 세마포어 값은 0아래가 될 수 없다.

        sem\_wait(&empty);

        val = x;

        printf("%u: %d\n", (unsigned int)pthread\_self(), val);

        x = val + 1;

        // 세마포어값을 1 증가시킨다.

        sem\_post(&full);

    }

    return NULL;

}

void \*thread\_decrement(void \*arg) {

    int i, val;

    for(i=0; i<ITER; i++) {

        // 세마포어값을 1 감소시킨다. 값이 0인 경우 1이상이 될 때까지 기다리다가 1이상이 되면 값을 감소시키고 대기상태에서 빠져나온다.

        // 즉, 세마포어 값은 0아래가 될 수 없다.

        sem\_wait(&full);

        val = x;

        printf("%u: %d\n", (unsigned int)pthread\_self(), val);

        x = val - 1;

        // 세마포어값을 1 증가시킨다.

        sem\_post(&empty);

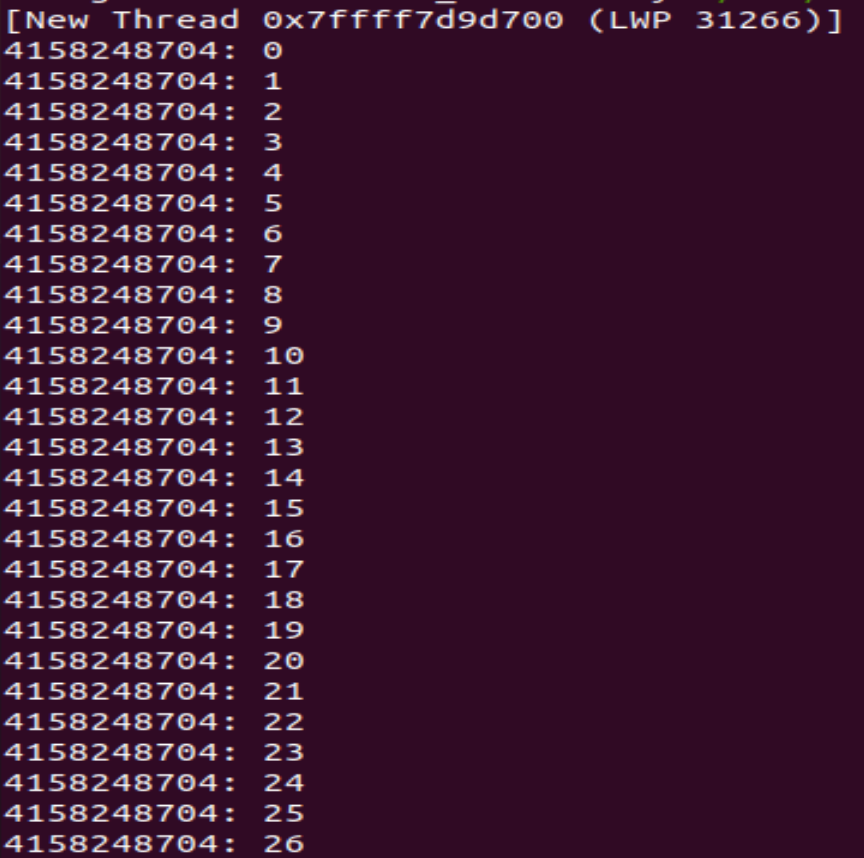
    }

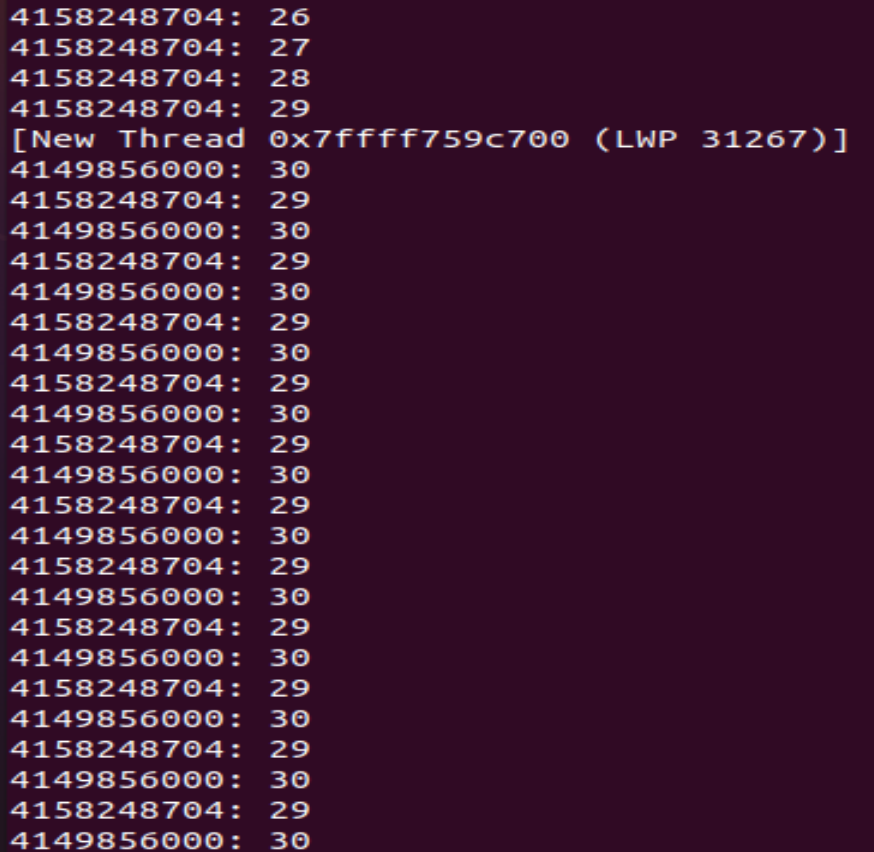
    return NULL;

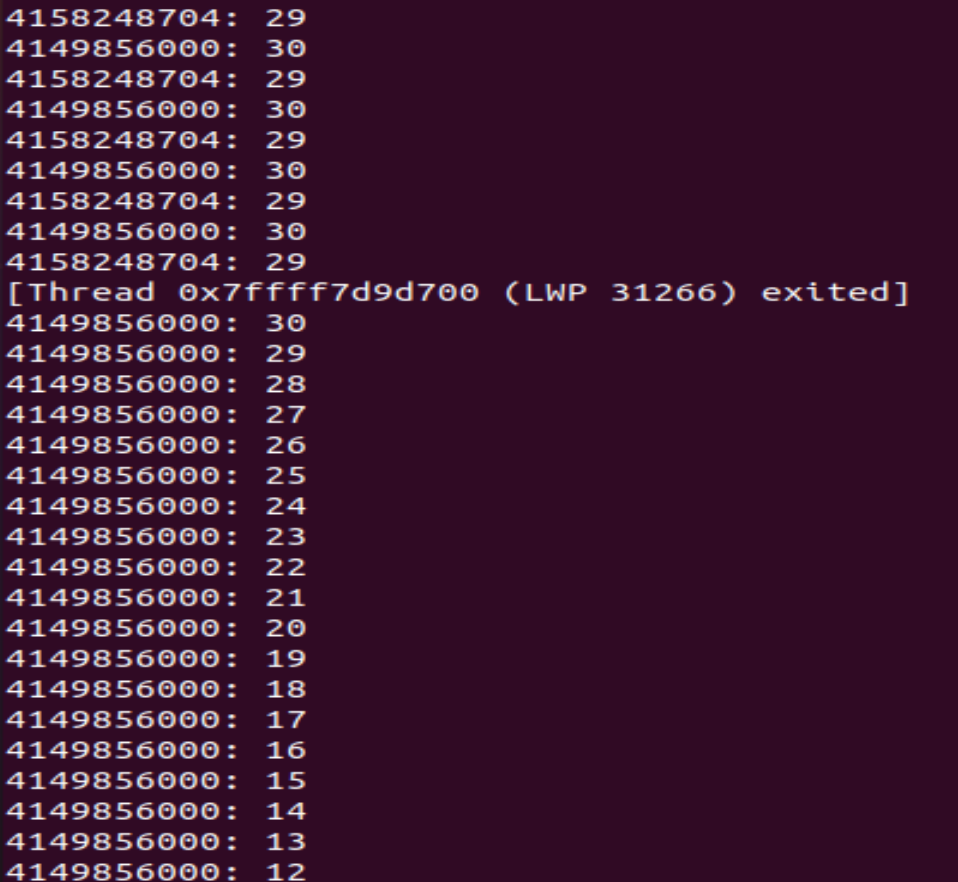
}

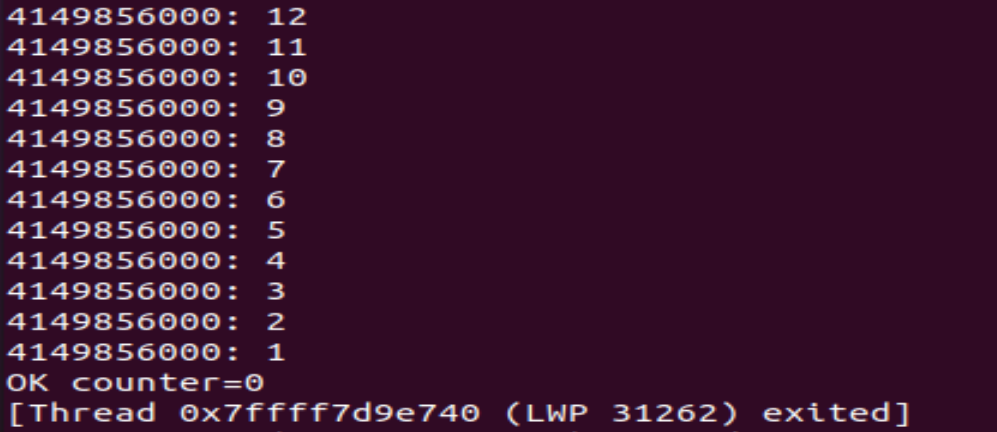
1. **동작과정**

**3-1. synch\_pthread.c**



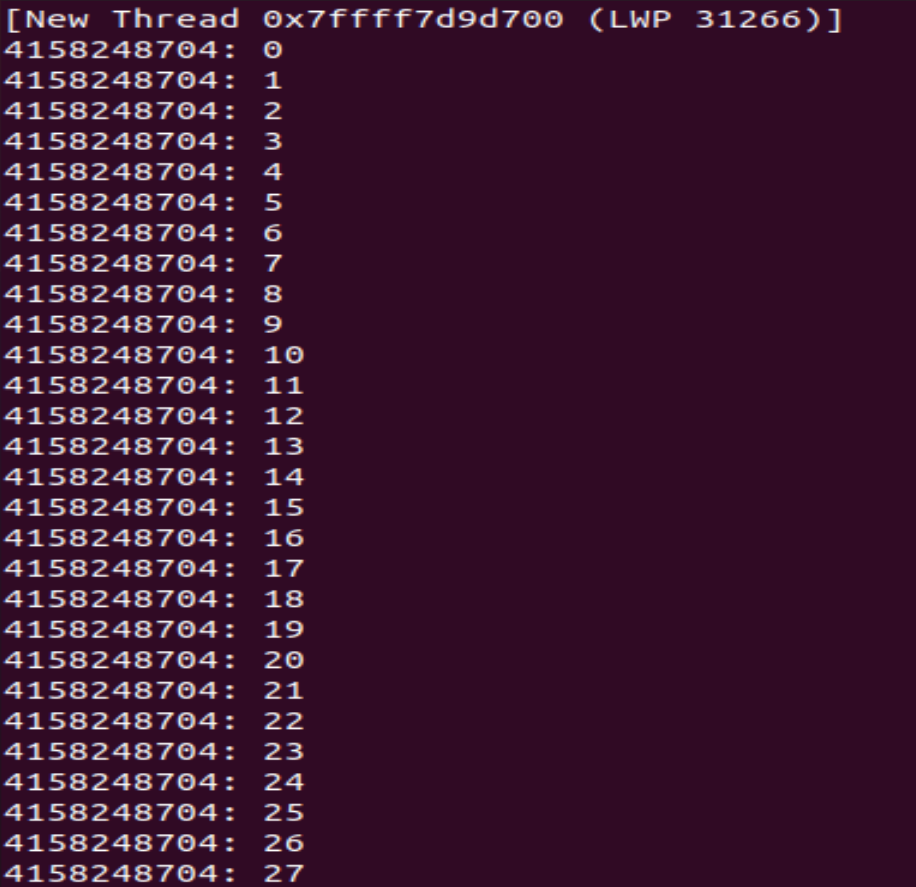


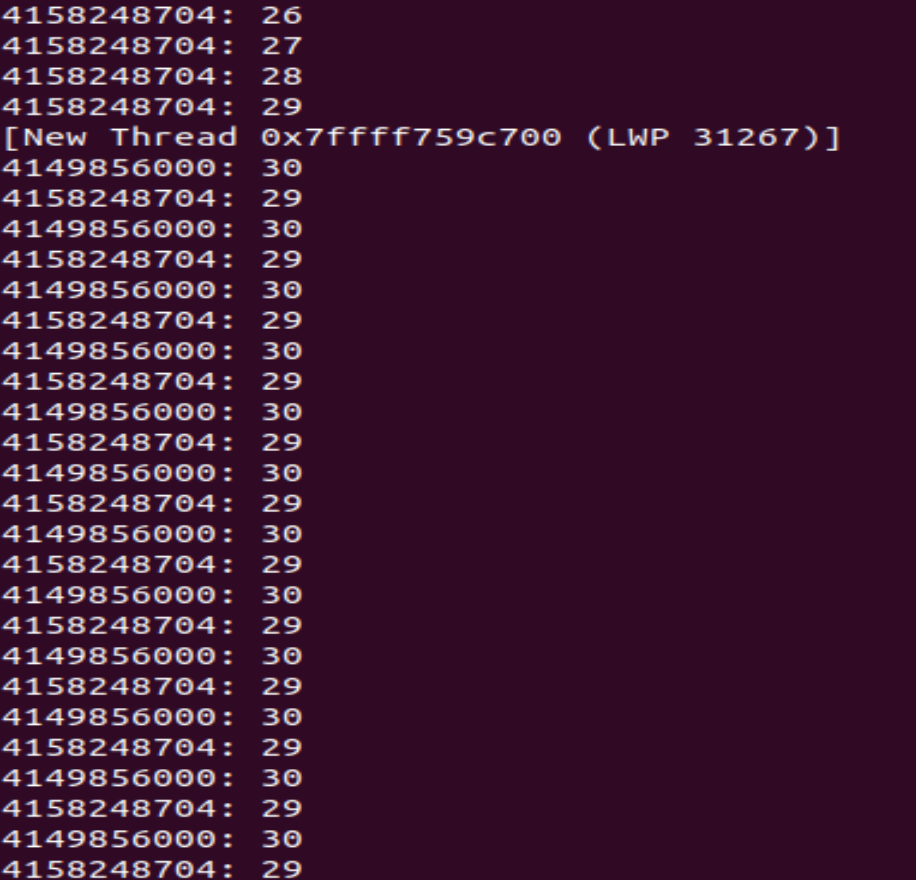


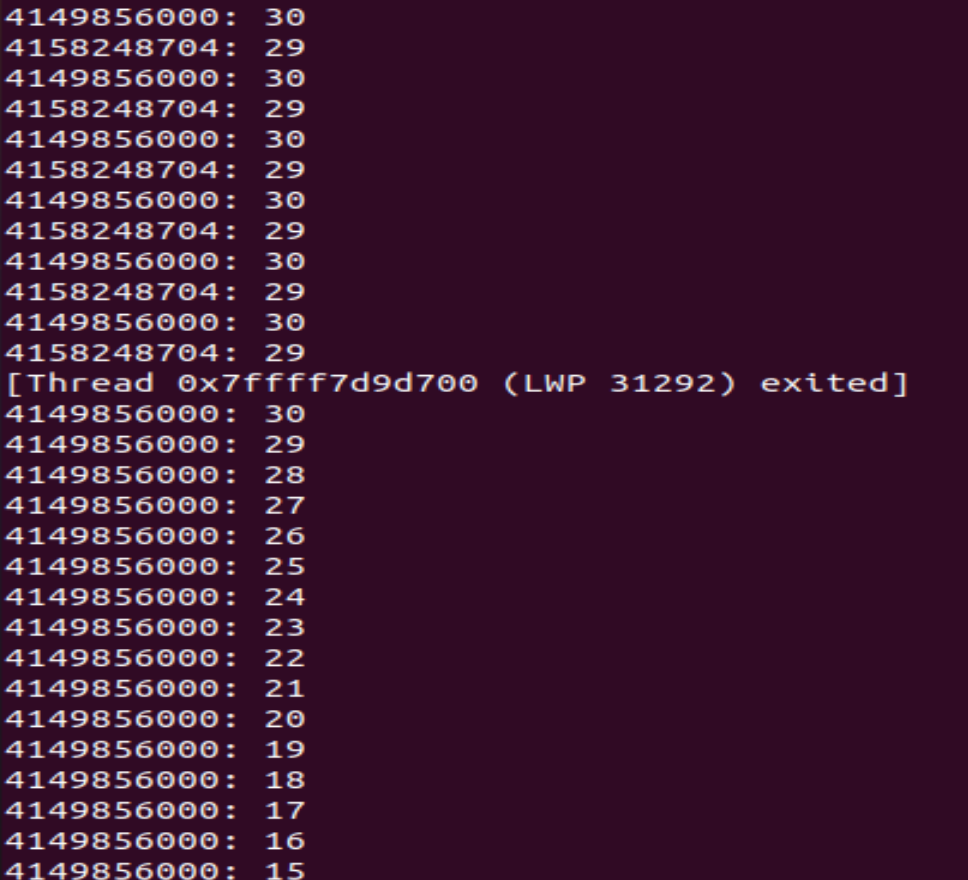


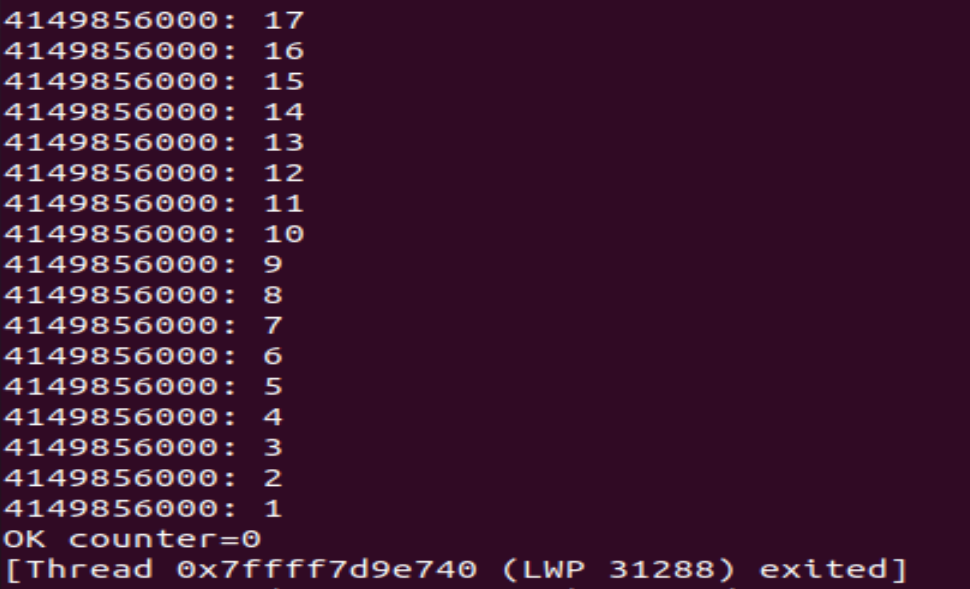
x의 값이 30을 넘지 않고 두 개의 thread가 동시에 critical section을 접근하지 않으면서 동기화 문제를 해결한 것을 확인할 수 있다.

**3-2. synch\_semaphore.c**









x의 값이 30을 넘지 않고 두 개의 thread가 동시에 critical section을 접근하지 않으면서 동기화 문제를 해결한 것을 확인할 수 있다.