1. 개요

Pthread 라이브러리를 이용하여 공유하는 하나의 전역변수에 대해 같은 크기의 값을 증가, 감소시키는 두 가지 쓰레드를 구현한 후, 두 쓰레드를 같은 수만큼 실행하여 전역변수의 초기값과 같은 결과값을 기댓값으로 가지는 프로그램을 구현하였습니다.

unsynced.c에서, 멀티 쓰레드 사용 시 공유 자원에 대한 동기화 문제를 발생시키기 위해, 쓰레드 실행 중에 컨텍스트 스위칭이 일어날 수 있도록 쓰레드 내부에 atomic operation인 변수 값 할당 외에 non atomic operation인 반복문을 추가하여 쓰레드가 종료되기 전에 다른 쓰레드로 전환될 수 있도록 하였습니다.

synced.c에서, mutex를 이용해 여러 쓰레드가 전역변수에 동시에 접근할 수 없도록 코드를 수정하여 동기화 문제가 발생하지 않도록 문제를 해결하였습니다.

1. 프로그램 구조 설명
   1. 함수에 대한 설명
2. unsynced.c

void\* decrease(void\* arg);

* 반복문을 이용하여 전역변수의 값을 0 ~ (maxTry - 1)만큼 감소시킴
* atomic operation인 변수 값 할당 외에 반복에서 함수의 동작이 분할 가능하므로 non atomic operation임

void\* increase(void\* arg);

* 반복문을 이용하여 전역변수의 값을 0 ~ (maxTry - 1)만큼 증가시킴
* atomic operation인 변수 값 할당 외에 반복에서 함수의 동작이 분할 가능하므로 non atomic operation임

long swap(long x);

* 인자로 받은 변수와 임시 변수 사이 값을 100회 주고받음
* 쓰레드의 실행 시간을 충분히 늘려주어 실행 중에 컨텍스트 스위칭이 일어날 수 있도록 함
* 다른 쓰레드에 의해 전역변수 값이 수정되는 동안 현재 쓰레드의 결과값을 잠시 가지고 있다가 다시 전역변수에 대입하여 동기화 문제를 의도적으로 발생시킴

1. synced.c

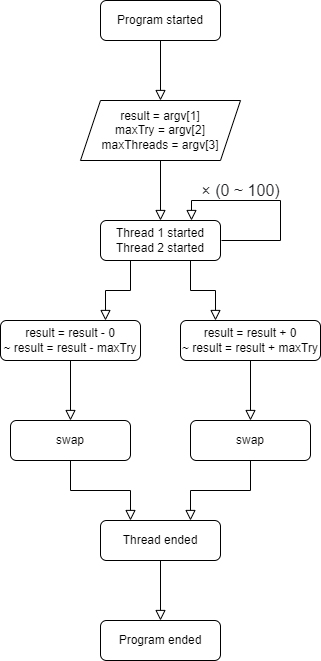
void\* decrease(void\* arg);

* 반복문을 이용하여 전역변수의 값을 0 ~ (maxTry - 1)만큼 감소시킴
* 동기화 문제를 해결하기 위해 내부의 수행문을 mutex로 감싸 atomic operation으로 구현함

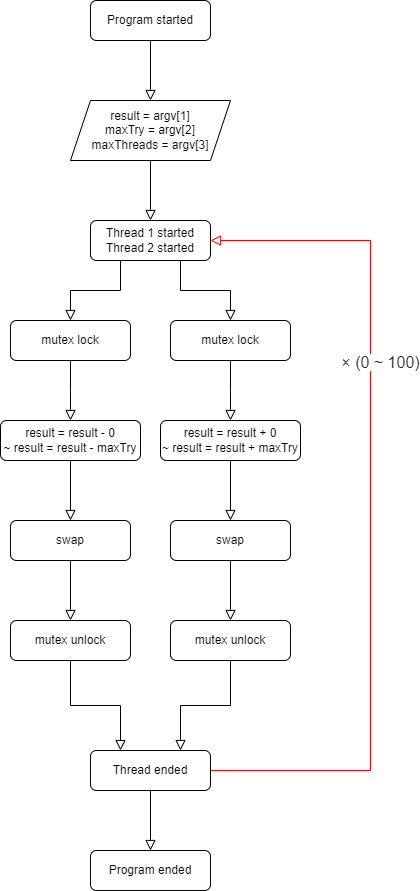
void\* increase(void\* arg);

* 반복문을 이용하여 전역변수의 값을 0 ~ (maxTry - 1)만큼 증가시킴
* 동기화 문제를 해결하기 위해 내부의 수행문을 mutex로 감싸 atomic operation으로 구현함
  1. 다이어그램

1. unsynced.c



1. synced.c



1. 실행 결과

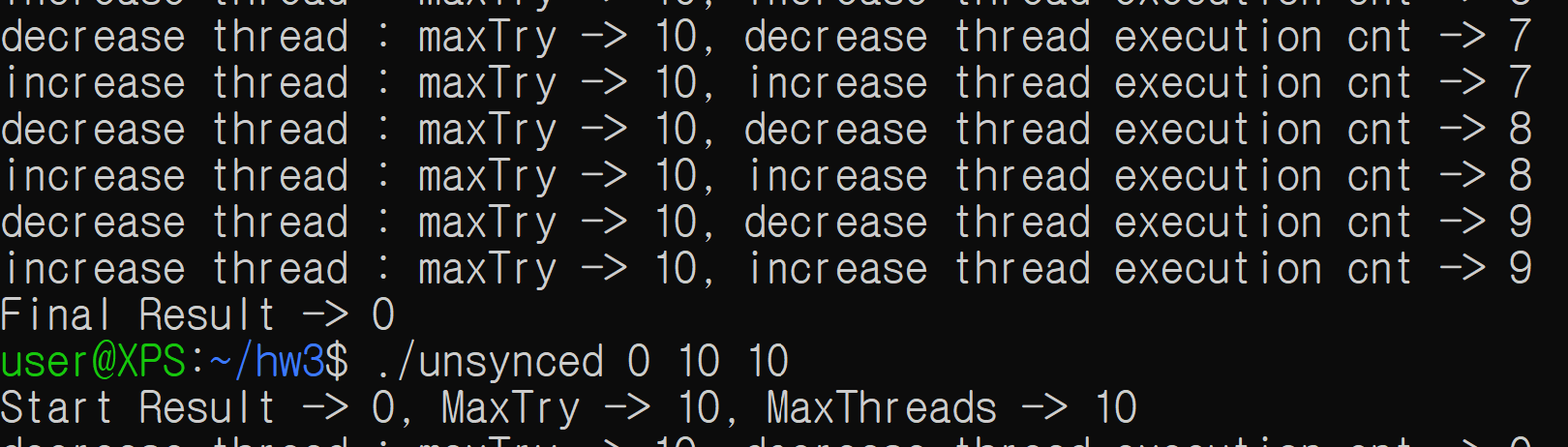
동기화 문제가 발생하는 unsynced.c, 문제를 해결한 synced.c 두 가지 프로그램에서 초기값은 0으로 고정하고 maxTry가 10으로 작은 경우, 100으로 큰 경우, maxThreads가 10으로 작은 경우, 100으로 큰 경우 총 4가지 상황으로 결과를 확인해 보았습니다. unsynced.c는 전역변수에 volatile 속성을 사용한 경우와 사용하지 않은 경우 두 가지 상황을 추가로 확인해 보았습니다. 오류가 발생하는 빈도를 확인하기 위해 각 상황별로 5번 실행하였습니다.

unsynced.c, volatile 사용하지 않은 경우)

maxTry = 10, maxThreads = 10)

텍스트이(가) 표시된 사진

자동 생성된 설명



텍스트, 전자기기, 키보드, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

텍스트, 점수판, 명판이(가) 표시된 사진

자동 생성된 설명

텍스트, 컴퓨터, 파일이(가) 표시된 사진

자동 생성된 설명

오류가 발생하지 않았습니다.

maxTry = 10, maxThreads = 100)

텍스트, 점수판이(가) 표시된 사진

자동 생성된 설명

텍스트, 점수판이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트, 점수판, 파일이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

모든 실행에서 오류가 발생하였습니다.

maxTry = 100, maxThreads = 10)

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트, 점수판, 명판이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

모든 실행에서 오류가 발생하였습니다.

maxTry = 100, maxThreads = 100)

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트, 점수판, 명판이(가) 표시된 사진

자동 생성된 설명

텍스트, 점수판, 명판이(가) 표시된 사진

자동 생성된 설명

텍스트, 점수판, 명판이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

모든 실행에서 오류가 발생하였습니다.

unsynced.c, volatile 사용한 경우)

maxTry = 10, maxThreads = 10)

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트, 점수판이(가) 표시된 사진

자동 생성된 설명

텍스트, 점수판, 명판이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트, 전자기기이(가) 표시된 사진

자동 생성된 설명

오류가 발생하지 않았습니다.

maxTry = 10, maxThreads = 100)

텍스트, 전자기기, 키보드이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트, 전자기기, 점수판이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트, 전자기기이(가) 표시된 사진

자동 생성된 설명

5회 중 4회 오류가 발생하였습니다.

maxTry = 100, maxThreads = 10)

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

모든 실행에서 오류가 발생하였습니다.

maxTry = 100, maxThreads = 100)

텍스트, 점수판, 파일이(가) 표시된 사진

자동 생성된 설명

텍스트, 점수판이(가) 표시된 사진

자동 생성된 설명

텍스트, 점수판, 명판이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

모든 실행에서 오류가 발생하였습니다.

synced.c)

maxTry = 10, maxThreads = 10)

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트, 점수판, 명판이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

오류가 발생하지 않았습니다.

maxTry = 10, maxThreads = 100)

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트, 점수판, 명판이(가) 표시된 사진

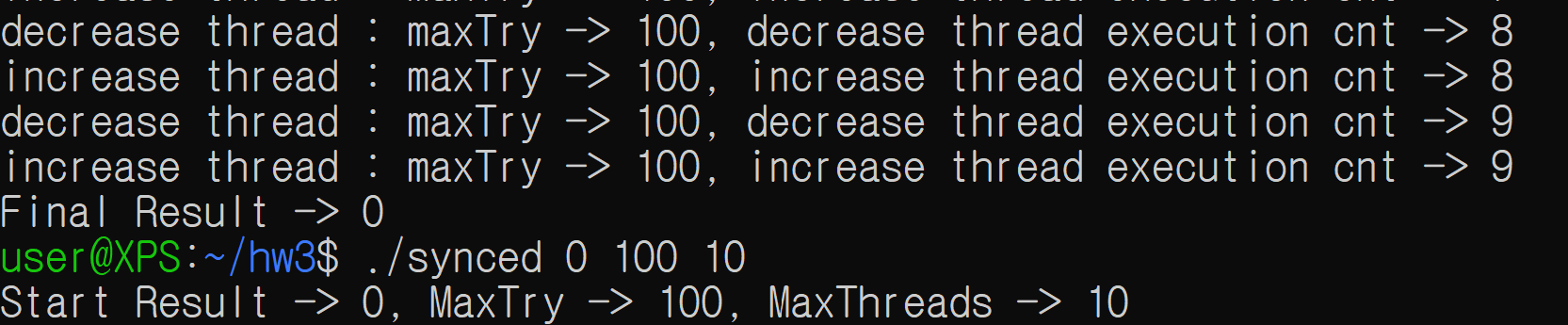
자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

오류가 발생하지 않았습니다.

maxTry = 100, maxThreads = 10)



텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트, 전자기기, 컴퓨터, 키보드이(가) 표시된 사진

자동 생성된 설명

오류가 발생하지 않았습니다.

maxTry = 100, maxThreads = 100)

텍스트, 점수판이(가) 표시된 사진

자동 생성된 설명

텍스트, 점수판, 명판이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

오류가 발생하지 않았습니다.

1. 고찰

동기화 문제가 일어날 수 있도록 구현한 unsynced 프로그램을 volatile 속성을 적용한 경우, 쓰레드 내 반복 횟수가 큰 경우, 작은 경우 그리고 쓰레드의 개수가 많은 경우, 적은 경우로 각각 나누어 실행해 보았습니다.

maxTry = 10, maxThreads = 10 인 경우 volatile 속성과 관계없이 오류가 발생하지 않았습니다. volatile 속성을 적용하였고 maxTry = 10, maxThreads = 100 인 경우에서 한 번의 실행을 제외하고 모든 경우 오류가 발생하였습니다.

maxTry를 증가시키는 경우, maxThreads를 증가시키는 경우 모두 실행해야 할 반복문의 반복 횟수를 증가시키는데 반복문은 실행 도중에 분할 가능한 non atomic operation이므로 쓰레드 내 반복 횟수를 증가시키는 경우, 쓰레드의 개수를 증가시키는 경우 모두 동기화 문제의 발생 확률을 높이는 같은 결과를 불러오는 것이라 추측해 보았습니다.

제가 작성한 코드에서는 과제 설명에서 말씀해주신 volatile 속성과 관계없이 반복 횟수에 따라 동기화 문제의 발생 확률이 결정되는 것처럼 보여 검색을 통해 volatile 속성이 어떤 역할을 하는가 확인해 보았습니다.

보통 컴파일러에서 높은 최적화 옵션을 적용했을 때, 어떤 변수의 값을 반복문을 통해 여러 번 수정하는 경우 반복문을 수행하여 매 회차마다 변수의 값을 수정하는 것이 아니라 반복이 완료되었을 때의 값을 바로 변수에 대입하여 프로그램의 실행시간을 줄이는 식으로 컴파일된다는 것을 알게 되었습니다. volatile 속성을 적용한 변수는 프로그램이 실행될 때 변수가 저장된 메모리에 반드시 접근해야 하는데 반복문에 사용되는 변수에 이 속성을 적용하면 실행 과정에서 생략되지 않고 모든 회차가 실행될 수 있음을 알게 되었습니다. 교수님이 수업시간에 보여주신 코드를 참고하여 작성한 이 프로그램에서는 쓰레드 내부의 반복문에서 값을 대입하는 명령어 외에 추가로 swap 함수를 실행하는데 이 부분으로 인해 최적화가 이루어지지 않고 반복문이 그대로 실행되어 volatile 속성을 적용한 경우와 같이 동작하여 volatile 속성을 적용한 경우와 적용하지 않은 경우 사이에 차이점이 없는 것이 아닌가 추측해 보았습니다.

1. 프로그램 소스 코드

unsynced.c

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

long maxTry = 999; // number of operations per thread

int maxThreads = 10; // number of threads

volatile long result = 0; // variables to be modified using threads

long t0cnt = 0; // number of executions of the "decrease" thread

long t1cnt = 0; // number of executions of the "increase" thread

// decrease result by 0 ~ maxTry

void\* decrease(void\* arg);

// increase result by 0 ~ maxTry

void\* increase(void\* arg);

// allowing threads to run long enough to cause synchronization problems

long swap(long x);

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

{

pthread\_t dec[100], inc[100];

int retval;

int i;

if (argc <= 1) {

printf("Usage: unsynced [init.result] [no.of.try] [no.of.threads]\n");

exit(1);

}

if (argc >= 2) {

result = atoi(argv[1]);

}

if (argc >= 3) {

maxTry = atoi(argv[2]);

}

if (argc >= 4) {

maxThreads = atoi(argv[3]);

}

if (argc > 4) {

printf("Usage: unsynced [init.result] [no.of.try] [no.of.threads]\n");

exit(1);

}

printf("Start Result -> %ld, MaxTry -> %ld, MaxThreads -> %d\n", result, maxTry, maxThreads);

// create a sufficient number of threads so that synchronization problems can occur

for (i = 0; i < maxThreads; i++) {

retval = pthread\_create(&dec[i], NULL, decrease, NULL);

retval = pthread\_create(&inc[i], NULL, increase, NULL);

}

// join the threads

for (i = 0; i < maxThreads; i++) {

retval = pthread\_join(dec[i], NULL);

retval = pthread\_join(inc[i], NULL);

}

printf("Final Result -> %ld\n", result);

return 0;

} /\* end of main \*/

void\* decrease(void\* arg)

{

long i;

for (i = 0; i < maxTry; i++) {

result -= i;

// allowing threads to run long enough to cause synchronization problems

result = swap(result);

}

printf("decrease thread : maxTry -> %ld, decrease thread execution cnt -> %ld\n", maxTry, t0cnt++);

pthread\_exit(0);

}

void\* increase(void\* arg)

{

long i;

for (i = 0; i < maxTry; i++) {

result += i;

// allowing threads to run long enough to cause synchronization problems

result = swap(result);

}

printf("increase thread : maxTry -> %ld, increase thread execution cnt -> %ld\n", maxTry, t1cnt++);

pthread\_exit(0);

}

long swap(long x)

{

long tmp;

int i;

// add a loop, which is a non atomic operation, to a thread

for (i = 0; i < 100; i++) {

tmp = x;

x = tmp;

}

return x;

}

synced.c

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

long maxTry = 999; // number of operations per thread

int maxThreads = 10; // number of threads

volatile long result = 0; // variables to be modified using threads

long t0cnt = 0; // number of executions of the "decrease" thread

long t1cnt = 0; // number of executions of the "increase" thread

// decrease result by 0 ~ maxTry

void\* decrease(void\* arg);

// increase result by 0 ~ maxTry

void\* increase(void\* arg);

// allowing threads to run long enough to cause synchronization problems

long swap(long x);

pthread\_mutex\_t mutex; // use mutex to make critical section

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

{

pthread\_t dec[100], inc[100];

int retval;

int i;

if (argc <= 1) {

printf("Usage: unsynced [init.result] [no.of.try] [no.of.threads]\n");

exit(1);

}

if (argc >= 2) {

result = atoi(argv[1]);

}

if (argc >= 3) {

maxTry = atoi(argv[2]);

}

if (argc >= 4) {

maxThreads = atoi(argv[3]);

}

if (argc > 4) {

printf("Usage: unsynced [init.result] [no.of.try] [no.of.threads]\n");

exit(1);

}

printf("Start Result -> %ld, MaxTry -> %ld, MaxThreads -> %d\n", result, maxTry, maxThreads);

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

// create a sufficient number of threads so that synchronization problems can occur

for (i = 0; i < maxThreads; i++) {

retval = pthread\_create(&dec[i], NULL, decrease, NULL);

retval = pthread\_create(&inc[i], NULL, increase, NULL);

}

// join the threads

for (i = 0; i < maxThreads; i++) {

retval = pthread\_join(dec[i], NULL);

retval = pthread\_join(inc[i], NULL);

}

retval = pthread\_mutex\_destroy(&mutex);

printf("Final Result -> %ld\n", result);

return 0;

} /\* end of main \*/

void\* decrease(void\* arg)

{

long i;

for (i = 0; i < maxTry; i++) {

pthread\_mutex\_lock(&mutex); // start critical section

result -= i;

// allowing threads to run long enough to cause synchronization problems

result = swap(result);

pthread\_mutex\_unlock(&mutex); // end critical section

}

printf("decrease thread : maxTry -> %ld, decrease thread execution cnt -> %ld\n", maxTry, t0cnt++);

pthread\_exit(0);

}

void\* increase(void\* arg)

{

long i;

for (i = 0; i < maxTry; i++) {

pthread\_mutex\_lock(&mutex); // start critical section

result += i;

// allowing threads to run long enough to cause synchronization problems

result = swap(result);

pthread\_mutex\_unlock(&mutex); // end critical section

}

printf("increase thread : maxTry -> %ld, increase thread execution cnt -> %ld\n", maxTry, t1cnt++);

pthread\_exit(0);

}

long swap(long x)

{

long tmp;

int i;

// add a loop, which is a non atomic operation, to a thread

for (i = 0; i < 100; i++) {

tmp = x;

x = tmp;

}

return x;

}

1. 자료 출처

“85.15 volatile 변수”, C 언어 코딩 도장, <https://dojang.io/mod/page/view.php?id=749>

22.04.08. 강의 녹화