System Programming

Project 6 Hybrid Lock

2014004248 강준규

2013011253 김기원

1. Project Goal

Hybrid lock을 구현해야 한다. 조건은 세가지가 있는데,

1. 락 획득 대기 시, 1초 동안 spin하며, 락을 대기한다.
2. 1초가 지나도 락을 획득하지 못한다면 해당 쓰레드를 블락시킨다.
3. 락은 블락 상태에 있는 쓰레드에게 우선적으로 넘겨준다.

를 만족시켜야 한다.

우선 테스트 환경은 서버에서 구동하였으며, 4코어입니다.

Hybrid\_lock은 mutex lock과 spin lock 두 가지를 이용하여 구현하였습니다.

#include "hybrid.h"

unsigned long long default\_try = 0;

void hybrid\_lock\_lock(hybrid\_lock \* lock)

{

struct timeval startT, endT;

double Timeuse;

unsigned long long try\_count = 0;

gettimeofday(&startT, NULL);

while(!gettimeofday(&endT, NULL) &&

(Timeuse = 1000000\*((endT.tv\_sec - startT.tv\_sec) + (endT.tv\_usec - startT.tv\_usec)) < 1000000)){

//무사히 시간 측정이 되고, 사용 시간이 1초 미만이라면 spin\_try를 합니다.

if(pthread\_spin\_trylock(&(lock->slock)) == 0) {

if(pthread\_mutex\_trylock(&(lock->mlock)) != 0) {

pthread\_spin\_unlock(&(lock->slock));

continue;

//spin\_lock을 획득하면 if 문 내부로 진입하게 됩니다. 만일 뮤텍스락을 본인이 소유하고 있거나, 새로 획득에 성공했다면 return 시키고, 그러지 못했다면 스핀락을 다시 풀어줍니다. 즉 데드락을 방지하기 위한 부분입니다.

}

return;

}

}

pthread\_mutex\_lock(&(lock->mlock));

pthread\_spin\_lock(&(lock->slock));

만일 스핀 락을 획득하지 못하거나, 획득했어도 뮤텍스락이 없다면 이제 뮤텍스 -> 스핀 순서로 락을 획득하게 합니다. 따라서 두가지 종류의 락이 모두 있어야 critical section에 진입할 수 있습니다. 또한 스핀이 끝난 후엔 뮤텍스 락의 획득을 시도하게 되므로 자동으로 스레드는 블락 상태로 진입하게됩니다.

}

void hybrid\_lock\_unlock(hybrid\_lock \* lock)

{

pthread\_mutex\_unlock(&(lock->mlock));

pthread\_spin\_unlock(&(lock->slock));

}

언락 함수는 뮤텍스 - >스핀 순으로 풀어주게 됩니다. 따라서 블락된 스레드들이 먼저 락을 획득할 수 있으며, 뮤텍스를 획득한 스레드들이 이제 스핀락까지 획득하게 됩니다. 이로서 과제의 hybrid\_lock에 대한 모든 명세를 구현하였습니다.

위의 설명과 같이 구현하였습니다. Init과 destroy 함수는 단순히 mutex\_lock과 spin\_lock을 init 시키고 destroy시키기 때문에 설명을 생략하였습니다.

위의 소스코드는 모두 <https://github.com/etual118/sysprogpj3>의 pj6 폴더에 업로드 하였습니다.

2.

1. 각각의 프로그램에서 mutex, spin lock, hybrid lock을 사용했을 때의 runtime을 비교한 그래프를 보고서에 포함
2. 각 프로그램에서 각 락을 사용했을 때, 성능이 달라지는 이유를 보고서에 작성

우선 test1입니다.

Test1의 가장 큰 특징은 critical section이 짧아서 context switch에 따른 오버헤드보다 busy waiting을 하면서 spin하는게 더 효율적일 수 있다는 것이다. Spin lock의 알려진 문제 중 하나로는, cpu의 논리 스레드 개수 이상으로 스레드를 생성하여 프로세스를 실행시키면 기하급수적으로 느려지기 시작한다. 따라서 주어진 조건을 조금 바꾸어 8개의 스레드가 10000000만번씩의 증가 연산을 수행하도록 했다. 예상과 같게 spin lock이 효율이 가장 좋았고, 그 다음이 mutex, hybrid 순서였다. Mutex는 context switch에 따른 오버헤드 때문에 spin보다 속도가 느린 것으로 판단할 수 있고, hybrid는 다른 lock에 비해서 성능이 떨어지고있다. 여기에 대한 원인을 분석해보면 우선 spin\_trylock을 하는데 따른 오버헤드입니다. 이를 해결하기 위해서는 아예 새로 락을 inline assembly로 구현해야할 것 같습니다. 두번째로는 매번 gettimeofday를 호출하면서 생기는 오버헤드입니다. 이를 해결한 과정은 밑에 서술되어있습니다.

Test2의 결과물입니다. Test2의 가장 큰 특징은 critical section의 실행시간을 길게하였기 때문에 cpu를 점유하는 시간이 길다는 것입니다. 따라서 1초 이상의 시간을 cpu에서 점유할 수 밖에 없고, 그렇다면 락을 대기하는 스레드들은 block되어 락의 해제를 기다리는 것이 가장 효율적일 것입니다. 따라서 스레드를 즉시 블록하는 mutex의 효율이 가장 좋았고, 1초 후에 블락하는 hybrid의 효율이 그 다음, 계속 스레드를 busy waiting 시키는 spin이 효율이 떨어졌습니다.

3번은 hybrid\_lock2 코드에 구현되어 있습니다. 처음으로 1초 동안 spin한 스레드가 있으면, 그 스레드의 try\_count를 기록해뒀다. 두번째 스레드부터는 count만큼만 try를 하도록 하였고, 그 시간과 1초의 gap을 보았습니다. Gettimeofday는 상당히 편차가 컸는데, 10000000부터 15000000까지 측정되었습니다. 따라서 10번 실행 후 평균값인 13200000번으로 테스트를 수행하였고 그 결과 수치가 대략 105600정도로 측정되었습니다.