Operating Systems
Project #3

컴퓨터학부

2023년 5월 1일

스핀락

스핀락(spinlock)은 락을 획득할 때까지 쉬지 않고 시도하는 방식이다. 루프를 반복적으로 돌면서 락이 해제되었는지 검사하기 때문에 스핀이라는 이름이 붙었다. 락을 신속하게 얻지 못하면 스핀락은 CPU를 계속 소모해서 비용이 많이 든다. 이런 단점에도 불구하고 스핀락은 이해하기 쉽고 구현이 간단해서 매력적이다. 특히 락의 경쟁이 낮거나 락을 소유한 시간이 짧은 환경에서 매우 유용한 방식이다.

CAE 명령어

CAE (Compare And Exchange)는 비교연산과 교환연산을 원자적으로 수행하는 명령어를 말한다. 어떤 행위가 원자적이라는 것은 그 행위를 마칠 때까지 과정이 분할되지 않는다는 것을 의미이다. 대부분의 CPU는 고유한 CAE 명령어를 가지고 있다. 고급언어로 작성된 프로그램에서 CPU 명령어를 직접 사용하는 것은 용이하지 않으므로 C/C++언어 라이브러리는 함수 형태로 CAE 명령어를 제공한다.

```
_Bool atomic_compare_exchange_weak(_Atomic(T) *object,
                                   T *expected, T desired);
```

- 만일 `object = expected`이면, `object ← desired` 치환, `true` 리턴
- 만일 `object ≠ expected`이면, `expected ← object` 치환, `false` 리턴

`object`는 `T` 타입의 원자변수이고, 나머지 `expected`와 `desired`는 `T` 타입의 일반변수이다. 이 함수는 `object`가 `expected` 값과 같으면 `object`를 `desired` 값으로 바꾸고 true를 리턴한다. 다르면 `expected`를 `object` 값으로 수정하고 false를 리턴한다.

CAE에는 weak과 strong 두 가지 버전이 있다. 앞의 예는 weak 버전으로 속도는 빠르나 `object`가 `expected`와 같아도 false로 잘 못 판단하는 경우가 발생할 수 있다. 우리는 이것을 거짓 부정 (false negative)이라고 한다. 그러나 그 발생 확률이 낮고 속도가 빠르기 때문에 잘 못 판단해도 해가 없는 while문과 같은 루프 안에서 사용한다. 반면에 strong 버전은 잘 못 판단하는 경우가 발생하지 않지만 속도가 느리기 때문에 주로 if문과 함께 사용한다.

이 함수의 한 가지 활용 예를 들어보자. `object`를 상호배타용 락으로 가정한다. 이 경우 `expected`는 락의 해제이고, `desired`는 락의 설정이 된다. 락을 획득하기 위해서는 먼저 락이 해제되었는지 검사해야 한다. 만일 해제가 되었다면 다른 이에게 빼앗기지 않도록 재빠르게 락을 설정해야 한다. `atomic_compare_exchange_weak()` 함수는 이 과정을 원자적으로 수행할 수 있다.

스핀락 구현

POSIX pthread 라이브러리는 뮤텍스락과 함께 스핀락을 제공한다. Pthread 스핀락에 관한 추가 정보는 시스템 매뉴얼을 참조한다. 스핀락 구현이 어렵지 않으므로 우리는 앞에서 언급한 CAE 명령어를 사용하여 다음과 같이 간단하게 구현한다. 여기서 lock은 불타입의 원자변수이고 락을 원하는 다른 스레드와 공유한다.

```
#include <stdbool.h>
#include <stdatomic.h>
...
atomic_bool lock = false;
...
    bool expected = false;
    while (!atomic_compare_exchange_weak(&lock, &expected, true))
        expected = false;
    /* critical section begins */
    ...
    /* critical section ends */
    lock = false;
...
```

while 루프 조건식을 보면 lock이 false이면 true로 만들고 루프를 빠져나와서 임계구역으로 들어간다. 그렇지 않으면 expected가 true로 바뀌기 때문에 루프 바디에서 다시 false를 바꾸고 루프를 돌린다. lock이 false가 될 때까지 쉬지 않고 루프를 반복하기 때문에 우리는 이것을 스핀락이라고 부른다. 임계구역을 벗어나면 lock을 false로 만들어서 해제한다.

스핀락 응용

앞에서도 언급했듯이 스핀락은 락의 경쟁이 낮거나 락을 소유하는 시간이 짧은 환경에서 유용한 방식이다. 이번 과제는 아래에 서술한 두 가지 문제를 스핀락을 사용하여 해결한다. 이 문제는 강의노트에도 있어서 자세한 설명은 노트를 참조한다.

- 공유버퍼 (bounded-buffer) 문제: 여러 개의 생산자와 소비자가 원형버퍼를 공유한 상태에서 생산과 소비를 동시에 진행할 때 발생하는 문제이다. 생산자는 물건을 만들어서 버퍼에 넣고, 소비자는 그 물건을 버퍼에서 가져와서 소비하게 되는데, 공유한 버퍼를 동시에 접근하기 때문에 동기화하지 않으면 문제가 발생한다.
- 유한대기 (bounded-waiting) 문제: 여러 개의 스레드가 임계구역에 동시에 접근할 경우 락을 사용하여 상호배타를 보장할 수 있다. 그러나 락을 얻는 과정에서 어떤 스레드는 경쟁에 밀려 굶주릴 수 있다. 아무리 길어도 어떤 정해진 차례 안에서는 스레드가 락을 얻을 수 있도록 보장하는 것이 유한대기 문제이다.

골격파일

골격파일 bounded_buffer.skeleton.c와 bounded_waiting.skeleton.c는 두 문제를 실험할 수 있는 코드가 완성되어 있다. 다만 동기화 부분이 빠져있어서 원하는 결과를 얻지 못한다. 학생들은 이 파일에 스핀락을 사용한 동기화 기능을 추가하여 제출한다.

제출물

공유버퍼와 유한대기 문제에 대한 스핀락 해법이 잘 설계되고 구현되었다는 것을 보여주는 자료를 각자가 판단하여 PDF로 묶어서 이름_학번_PROJ3.pdf로 제출한다. 여기에는 다음과 같은 것이 반드시 포함되어야 한다.

- 본인이 설계한 두 가지 해법에 대한 설명 (총 1쪽)
- 컴파일 과정을 보여주는 화면 캡처
- 실행 결과물의 주요 장면을 발췌해서 그에 대한 상세한 설명
- 과제를 수행하면서 경험한 문제점과 느낀점
- 프로그램 소스파일 2개 (`bounded_buffer.c`와 `bounded_waiting.c`) 별도 제출
- 실행 결과물 2개 (`bounded_buffer.txt`와 `bounded_waiting.txt`) 별도 제출

평가

- Correctness 50%: 프로그램이 올바르게 동작하는 지를 보는 것입니다. 여기에는 컴파일 과정은 물론, 과제가 요구하는 기능이 문제없이 잘 작동한다는 것을 보여주어야 합니다.
- Presentation 50%: 자신의 생각과 작성한 프로그램을 다른 사람이 쉽게 이해할 수 있도록 프로그램 내에 적절한 주석을 다는 행위와 같이 자신의 결과를 잘 표현하는 것입니다. 뿐만 아니라, 프로그램의 가독성, 효율성, 확장성, 일관성, 모듈화 등도 여기에 해당합니다. 이 부분은 상당히 주관적이지만 그러면서도 중요한 부분입니다. 컴퓨터과학에서 중요하게 생각하는 best coding practices를 참조하기 바랍니다.

HK