# 성능 비교

C\_LIST(Lock하나로 동기화 객체 전체를 감싸는 경우),

F\_LIST(Lock Unlock을 너무 많이 해서 느리다.),

O\_LIST(오버헤드가 크다. 병렬성(Lock을 한다고 기다리지 않음)이 있어 위 상황들 보다 훨씬 빠르다.) (그리고 Memory Leak이 존재하여 이걸 보완해야 하지만 좋은 방법은 없다.) (스레드가 많아지면 빨라지는 것은 사실이지만 스레드가 적을때에는 느리다.)

L\_LIST (Contain에서 Lock을 걸지 않기 때문에 Lock을 거는 횟수가 줄어들어 앞에 것들보다 1코어 일 때 더 빨라졌다.)

* 메모리 릭의 해결
* Free List
* Delete하지 않고 모아놓음
* Marking이 해제되는 순간 오작동 가능
* shared\_ptr => delete 를 하지 않아도 메모리 릭이 절대 생기지 않는다.
* 멀티스레드에서 사용해도됨? 사용해도 될 것 같다.(매뉴얼에서)
* Shared\_ptr이란?
* C++11에서 제공하는 일종의 스마트 포인터
* 객체에 counter를 두고 이를 통해 쓰이지 않을 객체를 판별하여 자동 삭제
* Atomic 하게 구현
* Shared\_ptr를 사용한 구현
* Node의 next를 shared\_ptr로 구현
* 모든 포인터를 쉐어드로 대체

뭐야? 구현 해봤는데 왤케 느려??

* Shared\_ptr로 복사하면서 나가는 것이 너무나도 느리다.
* 일반 포인터와 쉐어드 포인터는 성능이 10배이상 차이가 난다.
* 게다가 무한루프에 빠져서 결과 값이 나오지 않음
* 디버그 모드에서는 아예 프로그램이 터진다.

멀티스레드에서 주의해야 할 점

* Shared\_pointer의 copy 🡺 Data Race

결론 : 고로.. Shared\_ptr은 해결책이 되지 않는다.

# 복습

* 현실의 멀티쓰레드 프로그램은 ?
* 여러 스레드가 동시에 멀티코어에서 실행된다.
* 스레드간의 데이터 공유 및 동기화는 안전한 lock-free 자료구조를 통해서 이루어진다.
* 안쓰면 효율이 안나오니까… (사용하지 않으면 병렬성 감소)
* Lock-free 알고리즘이란?
* 여러 개의 쓰레드에서 동시에 호출했을 때에도 정해진 단위 시간마다 적어도 한 개의 호출이 완료되는 알고리즘
* STL => 멀티 쓰레드에서 돌렸을 때 제대로 돌아가지 않으므로 탈락
* Non-Blocking 알고리즘
* 다른 스레드가 어떤 상태에 있건 상관없이 호출이 완료된다.
* 호출이 다른 쓰레드와 충돌하였을 경우 적어도 하나의 승자가 있어서, 승자는 딜레이 없이 완료된다. <= 이것이 목표.
* 이 세상의 모든 알고리즘은 싱글쓰레드전용과 멀티쓰레드전용으로 나눌 수 있다.
* Mutex, Lock, Unlock => Blocking
* Lock-Free, … => Non-Blocking
* 왜 Blocking인가?
* Data Ready에 True가 들어가지 않으면 무한 대기이기 때문이다.
* 왜 Non-Blocking
* CAS => 간단히 말하자면… 다른 스레드와 충돌했을 때 내가 졌다.. 포기하고 다시 하자.
* Lock-free알고리즘은 어떻게 구현되는가?

시작 -> 자료구조의 변경 시도 -> 성공했는가? -> (no)시도 전으로 되돌아감, (Yes)완료’’

성공 했는가?를 우리가 어케 알아;;

* 앞의 알고리즘이 불가능 하므로

현재의 자료구조를 파악한다. => 자료구조의 변경을 시도한다. But 다른 쓰레드가 먼저 변경했으면 시도 취소. => 성공 했는가? => (yes) 완료, (no) 현재의 자료구조 파악으로 다시 되돌아 간다.