Concurrent Programming Assignment3, Wait-Free Snapshot implementation  
컴퓨터공학부 성예닮

Prof. 정형수

TAs. 김종빈, 오현석, 장형원

Run:

cd project3  
make  
./run [N\_THREAD]

Ex)   
./run 1  
./run 32

Task

Wait-Free Snapshot을 구현한다.

각 Thread는 자신의 로컬 변수에 대해 계속 업데이트를 수행한다.

Data structures

StampedSanp:  
T stamp // 업데이트 된 시각을 저장하는 스탬프  
T value // 업데이트 된 값  
T snap[] // 가장 최근에 저장한 스냅샷

WFSnapshot:  
 StampedSnap a\_table[] // 원자적인 스냅샷들의 배열  
 capcacity // 스냅샷의 크기, 이번 과제에서는 쓰레드의 수와 같다.

Functions

StampedSnap[] collect():  
 WFSnapshot의 스냅샷 배열을 레지스터에 복사하여 반환한다.  
 원자적이지 않은 동작이다.

T[] scan():  
 업데이트가 완료된 쓰레드에 대해 move라는 연산을 수행한다.  
 move 연산이 완료되었다고, 해당 쓰레드의 스냅샷을 현재의 최신 스냅샷으로 간주할 수는 없다.  
 업데이트가 완료되지 않은 상황이라면, 다시 중간 루프인 collect를 반복하며 move를 기다린다.

void update():  
 scan()을 호출하여 snapshot을 저장한다.  
 새로운 값을 value에 대입하고, stamp를 1씩 증가시키는 연산을 한다.  
 이전 값은 더 이상 필요하지 않은 값이므로 메모리에서 해제한다.

Performance Graph

전체적인 관점에서 보면, 쓰레드 수가 증가함에 따라 총 업데이트 수는 감소함을 알 수 있다.  
여러가지 이유가 있을 수 있지만, 순차적으로 반복하며 레지스터를 복사하는 작업과   
scan()시 쓰레드가 많을 경우 계속 충돌이 나는 경우가 잦아 느려진다고 볼 수 있을 것 같다.