**ABA 문제** : 메모리를 사용하려는 과정에서 해당하는 메모리가 재사용되었을 때, 일어나는 문제

* 포인터(p)가 예전과 동일하다고 해서, 포인터의 데이터(\*p)가 예전과 동일한 보장x
* 예) Thread A 가 NODE a를 사용하려 하는데, Thread B가 Node a가 있던 메모리를 재사용하면 Node a는 새로운 데이터를 가지고 있지만, Thread A는 인지할 수 없기 때문에 그대로 이용하게 됨. 그래서 Thread A는 NODE a의 이전 데이터로 계산하기 때문에 정확하지 않은 데이터를 이용.
* 해결) 메모리에 태그를 달아서 같은 메모리 주소라도 태그가 다르면 다른것으로 취급.
* 하드웨어에 종속적인 방법

그래서 사용하는 **Hazard pointer**!

* 사용할 메모리 블록을 hazard pointer에 등록 후 사용하고, 사용이 끝나면 hazard pointer에서 제거. 다른 쓰레드는 해제할 메모리가 있으면 hazard pointer에서 검색하고, 없다면 삭제.
* Hazard pointer에 등록되어 있다면, 다른 쓰레드가 안전하게 사용 중이므로, 해제하면 안됨.
* 해제할 메모리가 hazard pointer에 있다면, 자신의 retire list에 저장하고 해제하지 않는다. 추후에 쓰레드가 메모리 해제할 기회가 생길 때, retire list와 hazard pointer를 비교해 삭제. (retire list에 존재하고, hazard pointer에 존재하지 않는 메모리)

hazard pointer도 lock-free로 구현되면**, hazard pointer에서 ABA 문제는?**

* 쓰레드에서 메모리를 해제할 메소드에서 hazard pointer이용. 메소드 시작부분에 hazard pointer를 등록하고, 끝부분에서 hazard pointer를 해제한다. Hazard pointer에는 할당만 있고, 해제가 없다.따라서, ABA 문제가 생기지 않는다.

class HpRec {

int active; 재사용 유무 : 사용중 (1) / 미사용중 (0)

void\* pNode;

HpRec\* pNext;

}

HpRec\* AllocHp() // HpRec 할당

{

// \*\*\*재사용할 HpRec가 없으면 생성

        HpRec\* p;

        //기존에 레코드 중에 빈레코드가 있는지 찾아본다.  
        for( p = pHpHead; p != NULL; p = p->pNext )  
        {  
            if( p->active == 0

&& InterlockedCompareExchange(

(LONG\*)&(p->active), (LONG)1, (LONG)0 ) == 0 )  
                return p;

}

        //새레코드를 하나 생성.  
        p = new HpRec();

        p->active = 1;  //사용중이라고 세팅.  
        p->pNode = NULL;

        //새로 생성한 레코드를 Head 위치에 삽입.  
        HpRec\* oldHead;  
        do  
        {  
            oldHead = pHpHead;  
            p->pNext = oldHead;  
        }  
        while( InterlockedCompareExchangePointer(

(PVOID\*)&pHpHead, p, oldHead ) != oldHead );

        return p;

}

void ReleaseHp( HpRec\* p ) // HpRec 반납

{

// \*\*\*실제로 HpRec를 삭제하지 않고, 재사용

        p->active = 0;         //미 사용중이라는 세팅만.  
        p->pNode = NULL;

}

SimpleList\* GetRetireList()  
{

// \*\*\* 쓰레드 로컬의 Retire list를 가져온다.(없으면 생성)  
        SimpleList\* rList = (SimpleList\*)TlsGetValue( tlsSlot );

        if( rList == NULL )  
        {  
            rList = new SimpleList();  
            TlsSetValue( tlsSlot, rList );  
        }

        return rList;  
}

void RetireNode( Node\* node )

{

// \*\*\* 사용하지 않을 노드를 Retire list에 넣는다.

        SimpleList\* rList = GetRetireList();  
        rList->Append( node );    //Retire List에 넣는다.

        //너무 자주 Hp 검색 작업을 하지 않도록 Retire된 Node가 적절한 수(R)가 되었을 경우에만 Scan을 수행.  
        if( rList->Size() >= R )  
            Scan();

}

void Scan()

{

// \*\*\* Hazard pointer list에서 사용중인 pointer을 모아서 retire list에 있는 값들과 비교해, harzard pointer list에 없는 것들만 삭제한다.

        // 현재 사용중인 Hazard Pointer들을 모은다.

        vector<PVOID> hpList;  
        for( HpRec\* p = pHpHead; p != NULL; p = p->pNext )  
        {  
            if( p->pNode != NULL )  
                hpList.push\_back( p->pNode );  
        }

        // 검색을 빠르게 하기 위해 모아놓은 HpRec들을 정렬.  
        sort( hpList.begin(), hpList.end(), less<PVOID>() );

        // 현재 Thread의 RetireList에 있는 요소마다 검색을 수행.

        SimpleList\* rList = GetRetireList();  
        SimpleList::Node\* p = rList->pHead;  
        SimpleList::Node\* temp;

        while( p != NULL )  
        {  
            // 모아놓은 Hp List에서 검색.  
            if( binary\_search( hpList.begin(), hpList.end(), p->pData ) == false )  
            {  
                // Hp List에서 검색되지 않았기 때문에 메모리에서 삭제.  
                delete p->pData;

                temp = p->pNext;  
                rList->Delete( p );  
                p = temp;  
            }  
            else  
            {  
                p = p->pNext;  
            }  
        }

}

Scan()중에 다른 쓰레드에 의해서 harzard list가 추가/제거 된다면??

* 다음 scan() 실행에서 반영될 뿐, 메모리 해제를 못하는 것은 아님.