운영체제 과제 2: Lock-Free Data Structure

컴퓨터과학부 2018920031 유승리 | 운영체제 과제 2 | 2020-05-01-금 제출

** 운영체제 과제 2: Lock-Free Data Structure

- 학번과 이름: 2018920031 유승리
- 제출일: 2020-05-01 금
- CPU Spec: i5-8250U
- Memory: 8GB

세부과제 1) 첨부한 프로그램에서 여러 개(다수의 스레드에서 실험할 것)의 스레드들이 루프를 돌면서 AddVal(1) 함수를 호출하여 원자적으로 값을 증가시킬 때와 단순히 덧셈 연산(i +=1 또는 i++)으로 값을 증가시킬 때 차이가 있는지 확인하라. 프로그램과 결과(화면 캡쳐 또는 프로그램 출력)를 보고서에 기술하라

- 전체 소스코드는 본문의 마지막 부분에 첨부되어 있다.

> ThreadBody() - 다수의 thread가 수행해야 하는 동작

_val++ 연산	AddValue(1) 함수

`static void ThreadBody()`
`{`
`  for (int i = 0; i < 100000; i++)`
`  { // Main 스레드와 병행 수행`
`  _val++;`
`  }`
`}`
	`static void ThreadBody()`
	`{`
	`  for (int i = 0; i < 100000; i++)`
	`  { // Main 스레드와 병행 수행`
	`  AddValue(1);`
	`  }`
	`}`

> 실험 결과 - _val++ 연산

Thread 5개	Thread 10개
========== [세부과제 1-1] _val++ 연산 - 컴퓨터과학부 2018920031 유승리 ========== Hello abcxx ---------- - Thread 개수: 5개 (Main 포함) - 예상 _val 출력: 500000 - 실제 _val 출력: 242281 ==========	========== [세부과제 1-1] _val++ 연산 - 컴퓨터과학부 2018920031 유승리 ========== Hello abcxx ---------- - Thread 개수: 10개 (Main 포함) - 예상 _val 출력: 1000000 - 실제 _val 출력: 238841 ==========

⇒ 단순한 덧셈 연산인 _val++ 연산을 통해 _val의 값을 증가시킨 경우, 예상된 결과 값보다 작은 값이 출력된다. 이는 다수의 thread가 동시에 공유변수 _val에 접근 및 증가 연산을 수행하여 race condition이 발생하기 때문이다. 그리고 실행되는 thread의 개수를 늘려서 실험해본 결과, 동시에 실행되는 thread의 개수가 늘어날수록 실제 결과 값과 예상 결과 값의 차이가 더욱 커지는 것을 확인할 수 있었다.

> 실험 결과 - AddValue(1) 함수

Thread 5개	Thread 10개
	

⇒ Interlocked.CompareExchange()를 이용하여 변수 값을 증가시키는 AddValue() 함수를 통해 _val의 값을 증가시킨 경우, thread의 개수와 상관없이 예상된 결과 값과 같은 값이 출력된다. 이는 Interlocked.CompareExchange() 연산이 변수의 값을 비교 및 교환하는 작업을 atomic하게 수행하기 때문이다.

세부과제 2) Interlocked.CompareExchange()를 이용하여 자료구조 시간에 배운 Linked List 자료구조, 특히 push()와 pop() 연산을 구현하라. (주의) ABA problem이 발생할 수 있다. ABA 문제를 해결책을 적용하여 구현하라.

- 전체 소스코드는 본문의 마지막 부분에 첨부되어 있다.

- 본문의 실험 결과에는 Linked List 내의 노드 개수를 세어 출력하는 방법으로 ABA problem 없이 push()와 pop() 연산이 잘 구현되었는지 확인하였다. 실제로 구현하는 과정에서는 구체적인 Linked List의 상태를 확인하기 위해 매 루프에서 노드의 데이터 값까지 추가로 모두 출력하여 정상적으로 동작하는 것을 확인하였기 때문이다.

- 따라서 실행 중 오류 없이 [완료] 출력 결과에서 (예상 전체 노드 개수) == (실제 전체 노드 개수)인 경우, 정상적인 동작이라고 판단하였다.

> ThreadBody() – 다수의 thread가 수행해야 하는 동작

```
static void ThreadBody()
{
    int k = 0;
    Random r = new Random();
    while (k < 50000)
    {
        for (int i = 0; i < r.Next(0, 101); i++)
        {
            if (!freeList.IsEmpty())
                headList.Push(freeList.Pop());
        }
        for (int i = 0; i < r.Next(0, 101); i++)
        {
            if (!headList.IsEmpty())
                freeList.Push(headList.Pop());
        }
        Console.Write('\b');
        k++;
    }
}
```

> ABA problem을 고려하지 않은 구현

```
public void Push(Node newNode)  // CAS를 이용한 atomic한 Push 연산 (lock-free)
{
    while (true)
    {
        if (newNode != null)
        {
            Node oldHead = new Node();
            oldHead.next = headNode.next;
            newNode.next = oldHead.next;
            if (Interlocked.CompareExchange(ref headNode.next, newNode, oldHead.next) ==
oldHead.next)
                break;
        }
    }
}

public Node Pop()   // CAS를 이용한 atomic한 Pop 연산 (lock-free)
{
    Node currentNode;
    while (true)
    {
        currentNode = headNode.next;
        if (currentNode != null)
        {
            if (Interlocked.CompareExchange(ref headNode.next, currentNode.next, currentNode) ==
currentNode)
                break;
        }
    }
    return currentNode;
}
```

⇒ Interlocked.CompareExchange()를 이용하여 multi-thread 환경에서도 정상적으로 동작하는 lock-free 방식의 LinkedList data structure를 구현하였다. Push() 연산에서는 headNode.next 노드를 새롭게 추가

되는 newNode 노드로 바꾸는 동작을 atomic하게 구현하였으며, Pop() 연산에서는 headNode.next 노드를 삭제되는 현재 노드의 다음 노드인 currentNode.next 노드로 바꾸는 동작을 atomic하게 구현하였다.

기본	Thread 개수 ↑	노드 개수 ↑	루프 반복 횟수 ↑
			
8534ms	14139ms	8973ms	13949ms

⇒ 기본(thread 5개, 노드 100개, 루프 반복 횟수 50000회) 경우에서 각각 thread 8개, 노드 300개, 루프 반복 횟수 80000회로 바꾸어서 실험하였다. Push()와 Pop() 연산이 Interlocked.CompareExchange() 연산을 통해 atomic하게 구현되어 multi-thread 환경에서도 노드의 유실이 발생하지 않았으며, ABA problem 없이 정상적으로 수행 완료되었다. 또한 thread 개수, 노드 개수, 루프 반복 횟수를 증가시키면 수행을 완료하는 데에 걸리는 시간 또한 증가한다는 것을 확인할 수 있었다.

⇒ ABA problem을 고려하지 않은 구현이었으나 실험 결과가 모두 ABA problem 없이 정상적으로 출력되었다. 그래서 C#의 garbage collection(GC) 때문에 ABA problem이 발생하지 않는 것인지 조사한 결과, GC가 모든 케이스의 ABA problem을 방지하는 것은 아니라는 사실을 알게 되었다. 따라서 이러한 경우를 고려한 방법으로도 구현하였다.

(- https://stackoverflow.com/questions/42854116/why-does-automatic-garbage-collection-eliminate-aba-problems

- https://nrhan.tistory.com/entry/ABA-Problem

- https://blog.naver.com/neos_rtos/220054300299)

> ABA problem을 고려한 구현

```
static volatile int counter;

...

public void Push(Node newNode)  // CAS를 이용한 atomic한 Push 연산 (lock-free)
{
    while (true)
    {
        if (newNode != null)
        {
            var oldCounter = counter;   // LinkedList의 counter 값을 oldCounter에 저장
            var newCounter = oldCounter + 1;    // oldCounter의 값에 1을 더해 newCounter에 저장
            Node oldHead = new Node();
            oldHead.next = headNode.next;
            newNode.next = oldHead.next;
            if ((Interlocked.CompareExchange(ref counter, newCounter, oldCounter) == oldCounter)
                && (Interlocked.CompareExchange(ref headNode.next, newNode, oldHead.next) ==
oldHead.next))
                break;
        }
    }
}

public Node Pop()   // CAS를 이용한 atomic한 Pop 연산 (lock-free)
{
```

```
    Node currentNode;
    while (true)
    {
        var oldCounter = counter;   // LinkedList의 counter 값을 oldCounter에 저장
        var newCounter = oldCounter + 1;    // oldCounter의 값에 1을 더해 newCounter에 저장
        currentNode = headNode.next;
        if (currentNode != null)
        {
            if ((Interlocked.CompareExchange(ref counter, newCounter, oldCounter) == oldCounter)
                && (Interlocked.CompareExchange(ref headNode.next, currentNode.next, currentNode)
== currentNode))
                break;
        }
    }
    return currentNode;
}
```

⇒ 첫 번째 방법으로 구현하여 ABA problem이 발생한다면 그 이유는 객체의 주소 값으로만 CAS 연산(Interlocked.CompareExchange())을 수행하였기 때문일 것이다. 따라서 Push(), Pop() 연산을 수행할 때, 즉 headNode.next가 변경될 때마다 LinkedList 클래스 내에 static으로 선언된 counter 변수를 CAS 연산을 통해 atomic하게 증가시키는 코드를 추가하면 ABA problem을 해결할 수 있을 것이라고 생각하였다. 이는 oldCounter에 기존 counter 값을, newCounter에 oldCounter+1 값을 복사한 후 기존 counter 값이 oldCounter 값과 같을 때 counter 값을 newCounter 값으로 atomic하게 업데이트하는 과정으로 수행된다. 그리고 이 과정이 성공적으로 수행되면 비로소 headNode.next를 CAS 연산으로 변경한다. 이러한 방법으로 Push(), Pop() 연산을 구현하면 ABA problem으로 인해 LinkedList data structure가 깨지는 상황을 방지할 수 있을 것이다. 또한, counter 변수의 자료형을 수정하거나 관련 연산을 추가한다면 규모가 훨씬 큰 multi-thread 상황에서도 오버플로우 등의 문제 상황 없이 정상적으로 수행될 수 있을 것이다.

기본	Thread 개수 ↑	노드 개수 ↑	루프 반복 횟수 ↑
[세부과제 2] - 컴퓨터과학부 2018920031 유승리 - Thread 개수: 5개 - 노드 개수: 100개 - 루프 반복 횟수: 50000회 [초기] - Free List 노드 개수: 100개 - Head List 노드 개수: 0개 - 예상 전체 노드 개수: 100개 - 실제 전체 노드 개수: 100개 [완료] - Free List 노드 개수: 36개 - Head List 노드 개수: 64개 - 예상 전체 노드 개수: 100개 - 실제 전체 노드 개수: 100개 - 걸린 시간: 8599ms	[세부과제 2] - 컴퓨터과학부 2018920031 유승리 - Thread 개수: 8개 - 노드 개수: 100개 - 루프 반복 횟수: 50000회 [초기] - Free List 노드 개수: 100개 - Head List 노드 개수: 0개 - 예상 전체 노드 개수: 100개 - 실제 전체 노드 개수: 100개 [완료] - Free List 노드 개수: 13개 - Head List 노드 개수: 87개 - 예상 전체 노드 개수: 100개 - 실제 전체 노드 개수: 100개 - 걸린 시간: 13768ms	[세부과제 2] - 컴퓨터과학부 2018920031 유승리 - Thread 개수: 5개 - 노드 개수: 300개 - 루프 반복 횟수: 50000회 [초기] - Free List 노드 개수: 300개 - Head List 노드 개수: 0개 - 예상 전체 노드 개수: 300개 - 실제 전체 노드 개수: 300개 [완료] - Free List 노드 개수: 214개 - Head List 노드 개수: 86개 - 예상 전체 노드 개수: 300개 - 실제 전체 노드 개수: 300개 - 걸린 시간: 9324ms	[세부과제 2] - 컴퓨터과학부 2018920031 유승리 - Thread 개수: 5개 - 노드 개수: 100개 - 루프 반복 횟수: 80000회 [초기] - Free List 노드 개수: 100개 - Head List 노드 개수: 0개 - 예상 전체 노드 개수: 100개 - 실제 전체 노드 개수: 100개 [완료] - Free List 노드 개수: 39개 - Head List 노드 개수: 61개 - 예상 전체 노드 개수: 100개 - 실제 전체 노드 개수: 100개 - 걸린 시간: 14042ms
8599ms	13768ms	9324ms	14042ms

⇒ 첫 번째 방법과 마찬가지로 기본(thread 5 개, 노드 100 개, 루프 반복 횟수 50000 회) 경우에서 각각 thread 8 개, 노드 300 개, 루프 반복 횟수 80000 회로 바꾸어서 실험하였다. Push()와 Pop() 연산이 Interlocked.CompareExchange() 연산을 통해 atomic 하게 구현되어 multi-thread 환경에서도 노드의 유실이 발생하지 않았으며, ABA problem 없이 정상적으로 수행 완료되었다. 또한 thread 개수, 노드 개수, 루프 반복 횟수를 증가시키면 수행을 완료하는 데에 걸리는 시간 또한 증가한다는 것을 확인할 수 있었다.

세부과제 3) Free List와 Head List에 노드 삽입, 삭제시 뮤텍스 락과 언락으로 상호배제 하는 기법은 다음과 같다.

mutex_lock()

노드 삽입 또는 삭제

mutex_unlock()

위 기법과 CompareExchange()를 사용하는 기법의 성능을 비교하라.

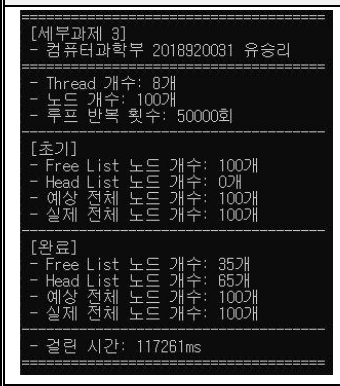
- 전체 소스코드는 본문의 마지막 부분에 첨부되어 있다.

- 세부과제 2와 마찬가지로 실행 중 오류 없이 [완료] 출력 결과에서 (예상 전체 노드 개수) == (실제 전체 노드 개수)인 경우, 정상적인 동작이라고 판단하였다.

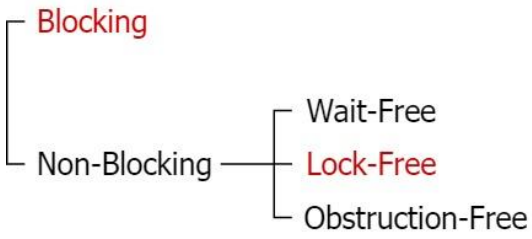
> Mutex를 사용한 구현

```
static Mutex mtx = new Mutex();

...

public void Push(Node newNode)  // Mutex를 이용한 Push 연산
{
    if (newNode != null)
    {
        Node oldHead = new Node();
        mtx.WaitOne();
        oldHead.next = headNode.next;
        newNode.next = oldHead.next;
        headNode.next = newNode;
        mtx.ReleaseMutex();
    }
}

public Node Pop()   // Mutex를 이용한 Pop 연산
{
    Node currentNode;
    mtx.WaitOne();
    currentNode = headNode.next;
    if (currentNode != null)
        headNode.next = currentNode.next;
    mtx.ReleaseMutex();
    return currentNode;
}
```

⇒ Mutex 를 이용하여 Push()와 Pop() 연산에서 headNode.next 노드를 변경할 때 상호배제를 수행하였고, multi-thread 환경에서도 정상적으로 동작하는 lock 방식의 LinkedList data structure 를 구현하였다.

기본	Thread 개수 ↑	노드 개수 ↑	루프 반복 횟수 ↑
			
74718ms	117261ms	141271ms	118189ms
CompareExchange()기법 보다 약 8.689배 오래 걸림	CompareExchange()기법 보다 약 8.517배 오래 걸림	CompareExchange()기법 보다 약 15.151배 오래 걸림	CompareExchange()기법 보다 약 8.417배 오래 걸림

⇒ CompareExchange() 기법과 마찬가지로 기본(thread 5 개, 노드 100 개, 루프 반복 횟수 50000 회) 경우에서 각각 thread 8 개, 노드 300 개, 루프 반복 횟수 80000 회로 바꾸어서 실험하였다. 모든 경우에서 정상적으로 동작하였고, 세부과제 2 의 ABA problem 을 고려한 CompareExchange() 기법보다 수행 시간이 대략 8 배 이상 소요되었다. 이는 특정 thread 가 공유 자원에 접근할 때 Mutex 로 lock 을 걸어 lock 이 release 될 때까지 다른 thread 는 blocked 상태로 대기하게 되기 때문이다.



⇒ Blocking 알고리즘이란 multi-thread 상황에서 lock 을 걸고 lock 을 획득한 thread 를 제외한 다른 thread 가 해당 자원에 접근하지 못하도록 blocking 하는 알고리즘으로, 이는 보통 Mutex 나 Semaphore 를 이용하여 구현된다. 그러나 Blocking 알고리즘에서는 lock 을 거는 것 자체가 오버헤드이고 나머지 thread 는 blocked 상태로 대기해야 하기 때문에, lock 에 대한 경쟁이 심해질수록 실제로 작업을 수행하는 시간 대비 동기화 작업에 소요되는 시간의 비율이 높아지면서 성능이 저하된다. 또한 lock 을 획득한 thread 가 I/O 작업 등의 이유로 blocked 된다면, lock 을 획득하려 대기하고 있던 다른 모든 thread 가 전부 blocked 상태가 될 가능성이 있다.

⇒ 하지만 Non-Blocking 알고리즘에서는 특정 thread 에서 작업이 실패하거나 blocked 상태에 들어가는 경우에 다른 어떤 thread 라도 그로 인해 실패하거나 blocked 되지 않는다. 이때, 각 단계마다 최소한 하나의 thread 는 항상 작업을 진행할 수 있는 알고리즘을 Lock-Free 알고리즘이라고 한다. Lock-Free 알고리즘은 CAS 와 같은 atomic 연산을 통해 구현되며, 변경하려는 대상이 현재 thread 의 맥락에서 일관적이면 변경을 진행한다. 만약 일관적이지 않다면 blocked 되지 않고 바뀐 상태로 thread 의 local 정보를 갱신한 후 다른 작업을 진행한다. 따라서 공유자원을 안전하게 동시에 사용할 수 있다.

(-https://www.slideshare.net/jinuskr/concurrency-in-action-chapter-7

- https://ozt88.tistory.com/38

- https://effectivesquid.tistory.com/entry/Lock-Free-알고리즘 Non-Blocking-알고리즘)

참고 자료)

- http://15418.courses.cs.cmu.edu/spring2013/article/46
- https://openshortcuts.com/archives/844
- https://blog.naver.com/jungwan82/20179129211
- https://stackoverflow.com/questions/42854116/why-does-automatic-garbage-collection-eliminate-aba-problems
- https://nrhan.tistory.com/entry/ABA-Problem
- https://blog.naver.com/neos_rtos/220054300299
-https://www.slideshare.net/jinuskr/concurrency-in-action-chapter-7
- https://ozt88.tistory.com/38
- https://effectivesquid.tistory.com/entry/Lock-Free-알고리즘 Non-Blocking-알고리즘

세부과제 1) 소스 코드 – _val++, AddValue()

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace OS_Project_2_1
{
    class Program
    {
        private static volatile string _str = "abc";
        private static volatile int _val = 0;

        static void AppendStr(string newstr)
        {
            while (true)
            {
                var original = Interlocked.CompareExchange(ref _str, null, null);
                var newString = original + newstr;
                if (Interlocked.CompareExchange(ref _str, newString, original) == original)
                    break;
            }
        }

        static void AddValue(int value)
        {
            while (true)
            {
                var orgVal = _val;
                var newVal = orgVal + value;
                if (Interlocked.CompareExchange(ref _val, newVal, orgVal) == orgVal)
                    break;
            }
        }

        static void ThreadBody()
        {
            for (int i = 0; i < 100000; i++)
            {  // Main 스레드와 병행 수행
                //_val++;       // 1-1) 단순한 덧셈 연산 _val++로 값 증가
                AddValue(1);    // 1-2) Atomic한 함수인 AddValue(1) 호출
            }
        }

        static void Main(string[] args)
        {
            Console.WriteLine("=====================================");
            //Console.WriteLine(" [세부과제 1-1] _val++ 연산");
            Console.WriteLine(" [세부과제 1-2] AddValue(1) 함수");
            Console.WriteLine(" - 컴퓨터과학부 2018920031 유승리");
            Console.WriteLine("=====================================");
            Console.WriteLine("Hello");
            AppendStr("xx");
            Console.WriteLine(_str);
            //          AddValue(5);
            //          Console.WriteLine(_val);
            Thread t1 = new Thread(ThreadBody);
            Thread t2 = new Thread(ThreadBody);
            Thread t3 = new Thread(ThreadBody);
            Thread t4 = new Thread(ThreadBody);
            Thread t5 = new Thread(ThreadBody);
            Thread t6 = new Thread(ThreadBody);
            Thread t7 = new Thread(ThreadBody);
            Thread t8 = new Thread(ThreadBody);
            Thread t9 = new Thread(ThreadBody);
            t1.Start();
            t2.Start();
            t3.Start();
            t4.Start();
            t5.Start();
            t6.Start();
            t7.Start();
            t8.Start();
            t9.Start();
            for (int i = 0; i < 100000; i++)
            {  // 스레드 t와 병행 수행
                //_val++;       // 1-1) 단순한 덧셈 연산 _val++로 값 증가
                AddValue(1);    // 1-2) Atomic한 함수인 AddValue(1) 호출
```

```
            }
            t1.Join();
            t2.Join();
            t3.Join();
            t4.Join();
            t5.Join();
            t6.Join();
            t7.Join();
            t8.Join();
            t9.Join();
            Console.WriteLine("-------------------------------------");
            Console.WriteLine(" - Thread 개수: 5개 (Main 포함)");
            Console.WriteLine(" - 예상 _val 출력: 500000");
            //Console.WriteLine(" - Thread 개수: 10개 (Main 포함)");
            //Console.WriteLine(" - 예상 _val 출력: 1000000");
            Console.WriteLine(" - 실제 _val 출력: " + _val);
            Console.WriteLine("=====================================");
        }
    }
}
```

세부과제 2) 소스 코드 (ABA problem을 고려한 구현) - CompareExchange() 이용

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Diagnostics;

namespace OS_Project_2_2
{
    public class Node
    {
        public Node next;
        public int data;

        public Node()
        {
            this.next = null;
            this.data = 0;
        }

        public Node(int data)
        {
            this.next = null;
            this.data = data;
        }
    }

    public class LinkedList
    {
        Node headNode;
        static volatile int counter;

        public LinkedList()
        {
            headNode = new Node();
            counter = 0;
        }

        public void InitFree()  // Free List에 노드 100개 넣어두는 메소드 (0~99)
        {
            for (int i = 0; i < 100; i++)
            {
                Node tempNode = new Node(i);
                if (tempNode == null)
                    Console.WriteLine("경고: 메모리가 부족합니다.");
                tempNode.next = headNode.next;
                headNode.next = tempNode;
            }
        }

        public bool IsEmpty()   // LinkedList가 비어 있는지 여부를 반환하는 메소드
        {
            if (headNode.next == null)
                return true;
            return false;
        }
```

```
        public int GetLength()  // LinkedList의 노드 개수를 반환하는 메소드
        {
            int cnt = 0;
            Node countNode = headNode.next;
            while (countNode != null)
            {
                cnt++;
                countNode = countNode.next;
            }
            return cnt;
        }

        public void PrintNode() // LinkedList 내의 노드들의 데이터를 출력하는 메소드, 테스트용으로 사용
        {
            String content = "";
            Node tempNode = headNode.next;
            while (tempNode != null)
            {
                content = content + tempNode.data + ", ";
                tempNode = tempNode.next;
            }
            Console.WriteLine("Linked List 내용: " + content);
        }

        public void Push(Node newNode)  // CAS를 이용한 atomic한 Push 연산 (lock-free)
        {
            while (true)
            {
                if (newNode != null)
                {
                    var oldCounter = counter;   // LinkedList의 counter 값을 oldCounter에 저장
                    var newCounter = oldCounter + 1;    // oldCounter의 값에 1을 더해 newCounter에
저장
                    Node oldHead = new Node();
                    oldHead.next = headNode.next;
                    newNode.next = oldHead.next;
                    if ((Interlocked.CompareExchange(ref counter, newCounter, oldCounter) ==
oldCounter)
                        && (Interlocked.CompareExchange(ref headNode.next, newNode, oldHead.next)
== oldHead.next))
                        break;
                }
            }
        }

        public Node Pop()   // CAS를 이용한 atomic한 Pop 연산 (lock-free)
        {
            Node currentNode;
            while (true)
            {
                var oldCounter = counter;   // LinkedList의 counter 값을 oldCounter에 저장
                var newCounter = oldCounter + 1;    // oldCounter의 값에 1을 더해 newCounter에 저장
                currentNode = headNode.next;
                if (currentNode != null)
                {
                    if ((Interlocked.CompareExchange(ref counter, newCounter, oldCounter) ==
oldCounter)
                        && (Interlocked.CompareExchange(ref headNode.next, currentNode.next,
currentNode) == currentNode))
                        break;
                }
            }
            return currentNode;
        }
    }

    class Program
    {
        private static volatile LinkedList freeList = new LinkedList(); // Free List
        private static volatile LinkedList headList = new LinkedList(); // Head List

        static void printStatus(String step)   // Free List와 Head List의 노드 개수를 출력하는 메소드
        {
            int freeLength = freeList.GetLength();
            int headLength = headList.GetLength();
            Console.WriteLine("-------------------------------------");
            Console.WriteLine(" [" + step + "]");
            Console.WriteLine(" - Free List 노드 개수: " + freeLength + "개");
            Console.WriteLine(" - Head List 노드 개수: " + headLength + "개");
            Console.WriteLine(" - 예상 전체 노드 개수: 100개");
```

```
            Console.WriteLine(" - 실제 전체 노드 개수: " + (freeLength + headLength) + "개");
        }

        static void ThreadBody()
        {
            int k = 0;
            Random r = new Random();
            while (k < 50000)
            {
                for (int i = 0; i < r.Next(0, 101); i++)
                {
                    if (!freeList.IsEmpty())
                        headList.Push(freeList.Pop());
                }
                for (int i = 0; i < r.Next(0, 101); i++)
                {
                    if (!headList.IsEmpty())
                        freeList.Push(headList.Pop());
                }
                Console.Write('\b');
                k++;
            }
        }

        static void Main(string[] args)
        {
            Console.WriteLine("====================================");
            Console.WriteLine(" [세부과제 2]");
            Console.WriteLine(" - 컴퓨터과학부 2018920031 유승리");
            Console.WriteLine("====================================");
            Console.WriteLine(" - Thread 개수: 5개");
            Console.WriteLine(" - 노드 개수: 100개");
            Console.WriteLine(" - 루프 반복 횟수: 50000회");

            freeList.InitFree();
            printStatus("초기");

            Thread t1 = new Thread(ThreadBody);
            Thread t2 = new Thread(ThreadBody);
            Thread t3 = new Thread(ThreadBody);
            Thread t4 = new Thread(ThreadBody);
            Thread t5 = new Thread(ThreadBody);
            Stopwatch stopwatch = new Stopwatch();

            stopwatch.Start();
            t1.Start();
            t2.Start();
            t3.Start();
            t4.Start();
            t5.Start();
            t1.Join();
            t2.Join();
            t3.Join();
            t4.Join();
            t5.Join();
            stopwatch.Stop();

            printStatus("완료");
            Console.WriteLine("------------------------------------");
            Console.WriteLine(" - 걸린 시간: " + stopwatch.ElapsedMilliseconds + "ms");
            Console.WriteLine("====================================");
        }
    }
}
```

세부과제 3) 소스 코드 – Mutex 이용

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Diagnostics;

namespace OS_Project_2_3
{
    public class Node
```

```
    {
        public Node next;
        public int data;

        public Node()
        {
            this.next = null;
            this.data = 0;
        }

        public Node(int data)
        {
            this.next = null;
            this.data = data;
        }
    }

    public class LinkedList
    {
        Node headNode;
        static Mutex mtx = new Mutex();

        public LinkedList()
        {
            headNode = new Node();
        }

        public void InitFree()  // Free List에 노드 100개 넣어두는 메소드 (0~99)
        {
            for (int i = 0; i < 100; i++)
            {
                Node tempNode = new Node(i);
                if (tempNode == null)
                {
                    Console.WriteLine("경고: 메모리가 부족합니다.");
                }
                tempNode.next = headNode.next;
                headNode.next = tempNode;
            }
        }

        public bool IsEmpty()   // LinkedList가 비어 있는지 여부를 반환하는 메소드
        {
            if (headNode.next == null)
                return true;
            return false;
        }

        public int GetLength()  // LinkedList의 노드 개수를 반환하는 메소드
        {
            int cnt = 0;
            Node countNode = headNode.next;
            while (countNode != null)
            {
                cnt++;
                countNode = countNode.next;
            }
            return cnt;
        }

        public void PrintNode() // LinkedList 내의 노드들의 데이터를 출력하는 메소드, 테스트용으로 사용
        {
            String content = "";
            Node tempNode = headNode.next;
            while (tempNode != null)
            {
                content = content + tempNode.data + ", ";
                tempNode = tempNode.next;
            }
            Console.WriteLine("Linked List 내용: " + content);
        }

        public void Push(Node newNode)  // Mutex를 이용한 Push 연산
        {
            if (newNode != null)
            {
                Node oldHead = new Node();
                mtx.WaitOne();
                oldHead.next = headNode.next;
                newNode.next = oldHead.next;
                headNode.next = newNode;
                mtx.ReleaseMutex();
            }
```

```
        }

        public Node Pop()   // Mutex를 이용한 Pop 연산
        {
            Node currentNode;
            mtx.WaitOne();
            currentNode = headNode.next;
            if (currentNode != null)
                headNode.next = currentNode.next;
            mtx.ReleaseMutex();
            return currentNode;
        }
    }

    class Program
    {
        private static volatile LinkedList freeList = new LinkedList(); // Free List
        private static volatile LinkedList headList = new LinkedList(); // Head List

        static void printStatus(String step)    // Free List와 Head List의 노드 개수를 출력하는 메소드
        {
            int freeLength = freeList.GetLength();
            int headLength = headList.GetLength();
            Console.WriteLine("-------------------------------------");
            Console.WriteLine(" [" + step + "]");
            Console.WriteLine(" - Free List 노드 개수: " + freeLength + "개");
            Console.WriteLine(" - Head List 노드 개수: " + headLength + "개");
            Console.WriteLine(" - 예상 전체 노드 개수: 100개");
            Console.WriteLine(" - 실제 전체 노드 개수: " + (freeLength + headLength) + "개");
        }

        static void ThreadBody()
        {
            int k = 0;
            Random r = new Random();
            while (k < 50000)
            {
                for (int i = 0; i < r.Next(0, 101); i++)
                {
                    if (!freeList.IsEmpty())
                        headList.Push(freeList.Pop());
                }
                for (int i = 0; i < r.Next(0, 101); i++)
                {
                    if (!headList.IsEmpty())
                        freeList.Push(headList.Pop());
                }
                Console.Write('\b');
                k++;
            }
        }

        static void Main(string[] args)
        {
            Console.WriteLine("=====================================");
            Console.WriteLine(" [세부과제 3]");
            Console.WriteLine(" - 컴퓨터과학부 2018920031 유승리");
            Console.WriteLine("=====================================");
            Console.WriteLine(" - Thread 개수: 5개");
            Console.WriteLine(" - 노드 개수: 100개");
            Console.WriteLine(" - 루프 반복 횟수: 50000회");

            freeList.InitFree();
            printStatus("초기");

            Thread t1 = new Thread(ThreadBody);
            Thread t2 = new Thread(ThreadBody);
            Thread t3 = new Thread(ThreadBody);
            Thread t4 = new Thread(ThreadBody);
            Thread t5 = new Thread(ThreadBody);
            Stopwatch stopwatch = new Stopwatch();

            stopwatch.Start();
            t1.Start();
            t2.Start();
            t3.Start();
            t4.Start();
            t5.Start();
            t1.Join();
            t2.Join();
            t3.Join();
```

```
            t4.Join();
            t5.Join();
            stopwatch.Stop();

            printStatus("완료");
            Console.WriteLine("-------------------------------------");
            Console.WriteLine(" - 걸린 시간: " + stopwatch.ElapsedMilliseconds + "ms");
            Console.WriteLine("=====================================");
        }
    }
}
```