임베디드 시스템용 Windows CE 운영체제에서 메모리 공간 확보 방안에 대한 연구

정동민* · 장승주*

*동의대학교 컴퓨터공학과

A Study of Method of Guaranteeing Enough Memory Space in Windows CE for Embedded System

Dong-Min Jung* · Seung-Ju Jang**

*Dongeui University, Dept. of Computer Enginneering

E-mail : 12074@deu.ac.kr, sjjang@deu.ac.kr

요 약

본 논문에서는 Embedded 시스템에서 Windows CE 내에 사용되고 있는 불필요한 메모리를 최소화 하고 메모리를 좀 더 효율적으로 관리하기 위한 방안으로 가비지컬렉터를 이용한 메모리관리 방법에 대한 연구이다. Embedded 시스템의 힙 메모리 내에 수집할 수 있는 방법 중 전체수집과, 부분수집에 대하여 살펴본다. 가비지컬렉터가 응용 프로그램에서 더 이상 사용하지 않는 힙 내에 개체가 있는지 확인하고, 힙에 대해 사용할 수 있는 메모리가 더 이상 없을 경우 새 연산자가 OutOfMemoryException를 실행하게 된다. 실험부분에서 가비지컬렉터의 개체사용 여부를 실험한다.

키워드

Window CE, .NET, 가비지컬렉터, 메모리효율

Ⅰ. 서 론

요즘 급속도로 발전하는 IT 정보화 산업에서, 최근 많은 발전을 하고 있는 Embedded 시스템내의 가비지컬렉션은 C#프로그램의 실행환경인 CLR(Common Language Runtime)에서 객체(object)의 생성 및 소멸이 반복되면서 필수적으로 발생하는 메모리의 단편화와 이에 따른 메모리 리소스의 감소를 방지하기 위해 메모리를 재사용하는 가비지 컬렉션을 지원한다. 그러나 객체가 소멸할 때 마다 가비지 컬렉션이 실행되면 CPU에 부하를 주어 프로그램 실행에 악 영향을 미쳐서 사용한 것보다 더 안 좋은 결과가 나올수 있다. 가비지컬렉터를 이용하여 WindowsCE 내의 메인 메모리에 상주하고 있는 불필요한 객체들을 삭제하는 것을 가상으로 에뮬레이터 해주는 과정을 알아보기 위하여 임의로 객체를 생성한 후 그 객체가 생성된 만큼 어떻게 가비지 컬렉터를 이용하여 삭제 하는지를 보여준다.

본 논문에서는 Embedded 시스템에서 Windows CE 내에 사용되고 있는 불필요한 메모리를 최소화 하고 메모리를 좀 더 효율적으로 관리하기 위한 방안으로 가비지컬렉터를 이용하여 메모리관리 방법에 대한 연구이다.

Embedded 시스템의 힙 메모리 내에서 필요한 객체(object)와 필요 없는 객체(object)를 분석, 연구하고, 객체의 필요, 불필요성을 체크한 후 가비지 컬렉터를 이용하여 힙 메모리 내의 공간을 어떻게 효율적으로 관리하는지 방법을 분석한다. 가비지 컬렉터를 이용 하였을 때와 이용하지 않았을 때를 비교, 가비지 컬렉터를 적절하게 사용하는 방법과 가비지 컬렉터 환경에서 실행 시 발생할 수 있는 성능 개선 문제를 검증하고, 가비지 컬렉터가 어떤 방식으로 작동하고 가비지 컬렉터

의 내부 작업이 실행 프로그램에 어떤 영향을 주는지에 대해 검증한다.

II. 관련연구

2.1 관리 힙의 단순화된 모델

관리 힙의 단순화된 모델을 살펴보면, 가비지 수집 가능한 모든 객체는 하나의 인접한 주소 고안 범위에서 할당됩니다.

아래 그림 1은 관리 힙의 단순화된 모델을 가상으로 구현해 본 것이다.

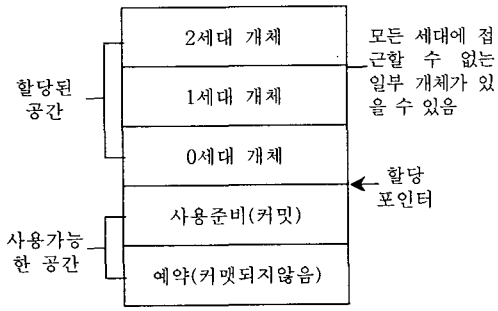

그림 1. 관리 힙의 단순화된 모델

그림 1에서 보면 힙의 작은 부분만 확인하여 대부분의 가비지를 제거할 수 있도록 힙은 여러 세대(0세대, 1세대, 2세대)로 구분된다.

각 세대별(0,1,2세대) 내의 모든 개체는 거의 동일한 나이때를 가지고, 더 높은 번호의 세대는 더 오래된 개체가 있는 힙 영역을 나타냅니다. 개체는 오래될수록 더 안정적입니다, 가장 오래된 개체는 가장 낮은 주소에 위치하며 새 개체가 만들어 질 때마다 주소가 늘어납니다. 위의 그림 1에서는 밑으로 내려갈수록 주소가 늘어난다.

새 개체에 대한 할당 포인터는 메모리의 사용(할당) 영역과 사용되지 않는(사용가능) 영역 간의 경계를 포시하고, 데드 개체를 제거하는 라이브 개체를 힙의 하위 주소 끝으로 이동하여 힙의 정기적으로 압축합니다. 이렇게 하면 새 개체가 만들어지는 다이어그램 아래족의 사용되지 않는 영역이 확장된다. 적절한 위치를 위해 메모리의 개체 순서는 개체가 만들어진 순서를 유지하고, 힙의 개체 사이에는 간격이 존재하지 않는다.

사용 가능한 공간 중 일부만 커밋되고, 필요할 경우 예약된 주소 범위의 운영체제에서 추가 메모리를 얻을 수 있습니다.

2.2 가비지 수집

2.2.1 전체수집

전체 수집에서는 프로그램 실행을 중단하고 GC 힙의 모든 루트를 확인해야 합니다. 이러한 루트는 다양한 형태를 취하지만 대부분 힙을 가리키는 스택 및 전역 변수입니다. 루트로부터 시작하여 모든 개체를 방문하게 되며 방문한 모든 개체에 포함된 해당 개체를 표시하는 개체 포인터를 따라 이동합니다. 이러한 방식으로 수집기는 접근 가능한 개체 또는 라이브 개체를 확인합니다. 접근할 수 없는 다른 개체는 이제 폐기됩니다.

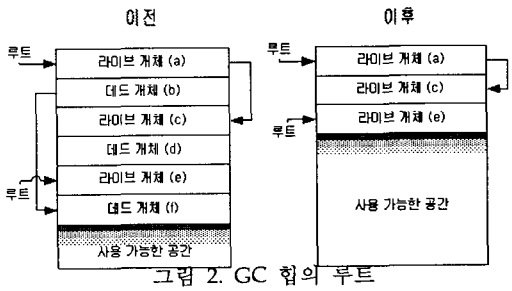

그림 2. GC 힙의 루트

그림 2는 접근할 수 없는 개체가 확인되고 나면 나중에 사용하기 위해 해당 공간을 회수할 수 있습니다.

즉, 이 시점에서 수집기의 목표는 라이브 개체를 위로 올리고 낭비된 공간을 제거하는 것입니다. 실행이 중단되면 수집기가 안전하게 이러한 개체를 모두 이동하고 새 위치로 올바르게 연결되도록 모든 포인터를 수정할 수 있습니다. 남아있는 개체가 다음 세대 번호로 수준이 올라가면(즉, 세대의 경계가 업데이트되면) 실행을 다시 시작할 수 있습니다.

2.2.2 부분수집

최근에 전체 수집이 수행되었고 힙이 적절하게 압축되었다고 가정한다.

이 경우, 프로그램 실행이 다시 시작되고 약간의 할당이 수행. 아주 많은 할당이 수행되어 충분한 할당이 이루어지면 메모리 관리 시스템은 수집 시간이 되었다는 것을 결정한다.

이러한 상황에서 마지막 수집 이후 전체 실행 과정에서 이전 개체에 전혀 기록하지 않았으며 새로 할당된 0세대 개체만 기록되었다고 가정. 이러한 경우는 가비지 수집 과정을 획기적으로 단순화할 수 있다.

일반적인 전체 수집 대신에 모든 이전 개체가 계속해서 라이브 상태이거나 최소한 이러한 개체가 작동 중이어서 확인할 필요가 없다고 가정할 수 있다. 또한 이러한 개체 중 기록된 것이 없기 때문에 이전 개체에서 새 개체로의 포인터가 존재하지 않는다.

따라서 이 상황에서는 모든 루트를 보통 때와 같이 확인하고 이전 개체를 가리키는 루트가 있을 경우 단순히 이러한 루트를 무시할 수 있다.

다른 루트의 경우 평상시처럼 모든 포인터를 따라 이동하며 이전 개체로 향하는 내부 포인터는 항상 무시된다.

이러한 프로세스가 완료되면 이전 세대의 개체를 열지 않고 0세대의 모든 라이브 개체를 열 수 있다. 그런 다음 0세대 개체를 평상시처럼 폐기할 수 있으며 해당 메모리 영역을 위로 이동함으로써 이전 개체의 상태를 그대로 유지한다.

이제 대부분의 데드 공간이 최신 개체에 존재한다는 것을 알 수 있으므로 상황이 매우 유리하다고 할 수 있다.

대부분의 클래스는 자체의 반환 값을 위한 임시 개체, 임시 문자열 및 혼합된 다른 유틸리티 클래스(예: 열거자 등)를 만든다. 0세대를 보면 개체의 극히 일부만을 확인하여 대부분의 데드 공간을 간단하게 되돌리는 방법을 알 수 있다.

3. GC 알고리즘

가비지 컬렉터는 응용 프로그램에서 더 이상 사용하지 않는 힙 내의 개체가 있는지 확인한다. 해당되는 개체가 있을 때는 이들 개체에서 사용한 메모리를 재생할 수 있다. 힙에 대해 사용할 수 있는 메모리가 더 이상 없을 경우 새 연산자가 OutOfMemoryException를 던지게 된다.

가비지 컬렉터가 실행되기 시작하면 힙 내 모든 개체는 가비지라는 가정을 전제로 하게 된다. 즉, 응용 프로그램 루트 중 어느 것도 힙 내 개체를 나타내지 않는다는 것이다. 이제, 가비지 컬렉터는 루트로 가서 루트에서 접근할 수 있는 모든 개체의 그래프를 구성하게 된다. 예를 들어, 가비지 컬렉터는 힙 내 개체를 가리키는 글로벌 변수의 위치를 나타낼 수 있다.

그림 3에서는 응용 프로그램 루트에서 직접 A, C, D, F 개체를 참조하는 몇 가지 할당된 개체를 힙으로 나타내고 있다.

이 개체는 모두 그래프의 일부가 된다. 개체 D를 추가하면 수집기는 이 개체가 개체 H를 참조하는 것으로 인식하고, 개체 H도 그래프에 추가된다. 수집기는 접근할 수 있는 모든 개체를 반복하여 돌아다니게 된다.

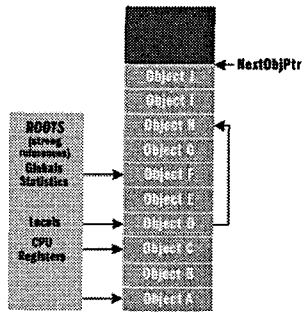

그림 3. 힙 내 할당된 개체

그래프에서 이 부분이 완성되면, 가비지 컬렉터는 다음 루트를 확인하고 다시 개체 사이를 돌아다닙니다. 가비지 수집기가 개체 사이를 다닐 때 이전에 추가된 그래프에 개체를 추가하려고 하면, 해당 가비지 수집기는 이동을 멈출 수 있습니다. 이는 두 가지 목적 때문인데, 하나는 한 번 이상 개체 집합 사이를 다니지 않음으로써 성능을 훨씬 증대시키기 위한 것이고, 다른 하나는 원형으로 된 개체의 연결 목록이 있을 때 무한 루프가 발생하지 않도록 하려는 것입니다.

그림 4에서는 수집 후의 관리 힙을 보여줍니다.

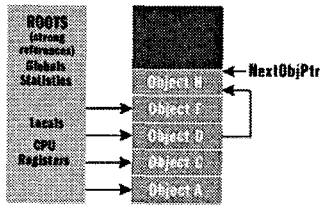

그림 4. 수집후의 관리 힙

가비지가 모두 확인된 후에는 가비지가 아닌 모든 메모리가 압축되고 가비지가 아닌 포인터는 모두 수정되며 NextObjPtr는 가비지 아닌 최종 개체 다음에 나타납니다. 이 때, 새로운 작업이 다시 시도되고 응용 프로그램에서 요청한 리소스는 성공적으로 만들어지게 됩니다.

4. 테스트 환경 및 실험결과

본 논문의 테스트 환경은 Windows XP운영체제 하에서 Microsoft Visual Studio .NET 2003내의 스마트장치 응용프로그램(WindowsCE)를 사용하였으며, C#을 이용하여 가비지컬렉터의 구동을 확인하였다.

다음의 그림 5은 가비지 컬렉터를 이용하여 실험을 해 보았다.

```
using System;
public class Memory
{
  private object BrainCell;
  public Memory(object Experience)
  //← 생성자
  {
    Console.Write(" Memory(). 기억한 내
용은: ");
    BrainCell = Experience;
    Remember();
  }
  public void Remember()
  {
    Console.WriteLine("{0}", BrainCell);
  }
  ~Memory()
//← 소멸자
  {

Console.Write("~Memory() 기억한 내용은:
");
    BrainCell = "기억을 삭제하겠습니다.";
    Remember();
  }
}
public class Experience
{
  public static void Main()
  {
    int i;
    int sum = 0;
    for(i=1;i<=100;i++)
    {
      sum = sum + i;
      Memory xp = new Memory(i);
    }
    GC.Collect();

GC.WaitForPendingFinalizers();
  }
}
```

그림 5. 가비지컬렉터 적용 소스

위의 그림 5은 소멸자는 생성된 객체의 수 만큼 호출되게 하고, 가비지 컬렉터는 LIFO(Last In First Out)의 순서에 따라 소멸자를 실행시키고 있다.

C#의 CLR은 이러한 구조로 객체의 뒤처리를 깔끔하게 마무리 한다.

아래 그림 6은 이 실험에 대한 결과 값이다.

3.1 가비지 컬렉션의 동작을 확인한 결과

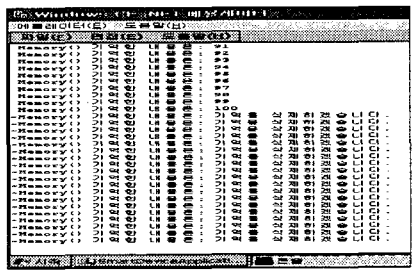

그림 6. 가비지컬렉터의 구동 결과

그림 6에서 나타난 것처럼 먼저 객체를 1~100개까지 소환한 다음, 소환된 갯수만큼 가비지컬렉터가 구동되는 것을 보여주는 결과이다.

III. 결 론

본 논문은 메모리 힙내에서 가비지의 전체수집과 부분수집, 그리고 GC알고리즘에 대하여 알아보았다. 전체수집에서의 목표는 라이브 개체를 위로 올리고 낭비된 공간을 제거하는 것이고, 실행이 중단되면 수집기가 안전하게 이러한 개체를 모두 이동하고 새 위치로 올바르게 연결되도록 모든 포인터를 수정할 수 있는 것을 알수있었다.

본 논문에서 실험은 Visual Studio.Net의 WindowsCE프로젝트로 소스를 구동하였을 때 먼저 New Memory(i) 개체의 인스턴스가 만들어지면서 객체가 메모리 상에 생성되는것을 볼수 있었고, 이후 가비지컬렉터가 생성된 모든 객체를 소멸해 나가는 걸 볼 수 있다.

참고문헌

[1] Mickey Williams, "PROGRAMMING MICROSOFT VISUAL C#.NET"
[2] Jason Price, "Visual C#.NET "
[3] 안원국, "C#.NET Mobile Programming"
[4] Karli Watson, (Beginning) C#
[5] Grant Palmer, C# programmer's reference
[6] Douglas Boling , Windows CE : 2nd Edition
[7] Juval Lowy , Programming.NET Components
[8] Christian Nagel , Professional C# 2005