메아리 저널

2009년 7월 23일 할당한 것은 무조건 해제해야 한다?

요즘은 만사가 귀찮아서 글을 잘 안 쓰게 되는 것 같아 좀 몸풀기 겸으로 써 봄.

흔히들 할당한 메모리는 무조건 해제해야 한다고 말한다. 예컨대:

```
Something *foo = new Something();
// foo를 사용한다.
delete foo;
```

요런 식으로 써야 한다는 것이다. (어째 이글루스 html 편집 기능이 엉망이 된 것 같다. 원래대로라면 <pre></pre>로 묶었어야 한다. 윽.)

물론 이 얘기는 일반적으로 틀린 말이 아니고, 아마도 C/C++로 프로그래밍하는 웬만한 사람들이 포인터 내지 메모리 관리를 배울 때 처음으로 듣는 원칙이기도 할 것이다. 근데 그렇다면 왜 이 글을 쓰겠는가? 당연히 아닌 경우가 있으니까 쓰겠지. -_-;

메모리 할당과 해제가 항상 일대일로 대응될 필요는 없다

예를 들어 DOM -- 자바스크립트가 실제 문서와 통신하는 인터페이스 -- 처럼 객체 생성과 해제가 빈번하게 일어나는 경우를 들 수 있다. (사실 DOM 성능은 자바스크립트 성능에 있어 무시 못 할 정도로 큰 비중을 차지한다.) 꼭 DOM까지 들지 않더라도 한 함수를 벗어나면 수많은 객체가 한꺼번에 사라진다거나 하는 경우도 해당하겠다.

이런 경우 보통 메모리 풀(memory pool)을 사용하게 된다. 메모리 풀은 어느 정도 이상의 메모리를 항시 할당해 놓고 대신 최대한 빠른 객체 생성/해제를 지원한 뒤 나중에 모두 필요 없어졌을 때 한꺼번에 날려 버리는 관리자 객체이다. 만약 해제를 일부러 완전히 무시하고 메모리 풀이 날아갈 때 한꺼번에 해제시킨다거나 하면 이 관리자 객체는 바로 가비지 컬렉션(garbage collection; 좀 더 엄밀하게는 여러 개의 메모리 풀을 두면 generational GC가 된다)을 구현하게 된다.

골수 C/C++ 프로그래머는 가비지 컬렉션을 못마땅하게 생각할지 모르지만, 이로 얻는 장점은 단순히 개발자의 편의 뿐만이 아니다. free나 delete 연산자를 호출했을 때 메모리가 항상 해제되어야 한다면 메모리 관리자는 매 free 요청마다 작업을 해야 하지만, 메모리 관리자가 자율적으로 해제를 계획할 수 있다면 여러 개의 free 요청을 묶어서 훨씬 효율적으로 처리를 할 수 있다. 실제로 메모리 할당이 잦은 큰 C/C++ 프로그램을 Bohem GC를 사용하게 살짝 고쳤는데 꽤 성능 향상을 보았다는 얘기도 있다. (Bohem GC는 C/C++의 특성 때문에 accurate GC를 쓸 수 없어서 훨씬 비효율적[1]인데도!)

당연하지만 가비지 컬렉션이 만능은 아니다. 일단 좋은 가비지 컬렉터를 만드는 건 무진장 어려우며, 특히 대부분의 경우에 성능이 좋게 하는 게 힘들다. (만약 "대부분"에 속하지 않는 경우라면 직접 상황에 맞게 알고리즘을 만들어야 할지도 모르겠다.) 프로그램 로직이 잘못되었다면 제 아무리 가비지 컬렉터가 좋아도 메모리 누수에 빠질 수도 있다.[2] 또한 가비지 컬

렉션은 정확한 성능 예측을 어렵게 만들고, 특히 실시간(real-time) 처리에 매우 부적합하다. (물론 실시간 처리를 할 때도 메모리 풀은 여전히 유용할 수 있다. 하지만 이 쯤 되면 뭘 해도 이 정도 삽은 파야 겠지.) C/C++ 같이 가비지 컬렉션을 고려하지 않고 만들어진 프로그래밍 언어에서는 선택의 폭이 꽤 줄어든다는 점도 있겠다.

어차피 운영체제도 해제를 한다

또 하나 중요한 사실은 내가 해제를 안 해도 운영체제는 프로그램이 끝날 때 할당된 메모리를 해제한다는 것이다. 이것은 현대 운영체제들이 가상 메모리 시스템을 구현하기 때문에 가능한 것으로, 이게 안 되면 운영체제에 버그가 있는 것이다. 물론 오래 돌아 가는 프로그램은 프로그램이 도는 동안 사용하는 메모리를 일정 수준 이하로 유지해야 하지만, 짧은 시간 동안 동작하고 만다면 그냥 free를 안 하는 게 오히려 성능에 도움이 될 수도 있다. (하지만 따라 하지는 말자. 이런 걸 고려할 정도면 이미 대부분의 다른 코드를 최적화했다는 소리니까.)

당장 생각나는 예로는 파이썬 인터프리터 코드에서 Py_Main 함수(사실상 인터프리터의 시작에 해당함) 맨 끄트머리에 있는 이 코드를 들 수 있겠다. 설명은 주석으로 대신한다.

```
#ifdef __INSURE__
    /* Insure++ is a memory analysis tool that aids in discovering
     * memory leaks and other memory problems.  On Python exit, the
     * interned string dictionary is flagged as being in use at exit
     * (which it is).  Under normal circumstances, this is fine because
     * the memory will be automatically reclaimed by the system.  Under
     * memory debugging, it's a huge source of useless noise, so we
     * trade off slower shutdown for less distraction in the memory
     * reports.  -baw
     */
    _Py_ReleaseInternedStrings();
#endif /* __INSURE__ */
```

(사실 그 앞의 Py_Finalize()에서도 비슷한 최적화를 시도할 법도 하지만 파이썬 객체는 finalizer가 있을 수 있어서 정신줄 놓고 메모리를 막 해제하긴 좀 그렇다. 코드 관리도 귀찮아지고.)

메모리의 수동 해제를 피해라

"할당한 메모리는 꼭 해제해야 한다"는 말에는, 당연하게도, 프로그래머가 알아서 해제해야 한다는 뉘앙스가 숨어 있다. 하지만 위에서도 봤듯이 메모리의 수동 관리가 성능을 떨어뜨리는 경우도 상당히 있고, 실수도 하기 너무 쉽다는 문제가 있다.

사실 더 현명한 해결책은 "메모리 관리를 좀 더 지능적으로" 하는 것이겠다. 앞에서 언급한 메모리 풀도 비슷한 얘기고, C++라면 std::auto_ptr이나 뭐 이런 류의 자동화된 객체들을 이용할 수도 있다. (물론 auto_ptr과 shared_ptr을 구분하는 정도의 지식은 필요하다.) C라면... C로 긴 프로그램을 짜려는 건 때려 치우고 최대한 잘게 쪼개서 관리하기 쉽게 해라. -_-;;;;; 개인적으로는 C/C++ 수준으로 저수준을 처리할 수 있는 언어 중에서 이걸 잘 해 주는 언어는 그닥 많지 않은 것 같은데 -- Objective C는 메모리 관리는 눈꼽만큼 낫지만 라이브러리가 병신같아서 제외 -- 내가 몇 년동안 생각(만) 하고 있는 새로운 언어의 목표 중 하나이기도 하다.

하여튼 컴퓨터가 할 수 있는 일이라면 최대한 컴퓨터에게 시키는 게 현명한 일이니, 가능한

한 컴퓨터에게 더 많은 일을 시키도록 궁리하는 게 좋다. 귀찮음을 피하기 위해 새로운 걸 궁리해 낸다면 그 귀찮음은 미덕일 뿐만 아니라 칭송할 일이다.

메모리가 아니라면 어떻게 해야 하는가?

"생성-해제"의 쌍은 비단 메모리 관리에서만 나타나는 게 아니다. 이를테면 파일도 열었다 닫아야 하고, 소켓도 열었다 닫아야 하고... 당연한 얘기지만 메모리에 해당하는 얘기를 이런 데서 갖다 쓰려면 관리하는 대상이 메모리랑 비스무리한 특징을 가져야 하는데 -- 비교적 많은 편이고, 해제가 수동으로 행해져야 하며, 정확한 해제 시점을 맞추지 못 했을 때의 페널티가 크지 않은 -- 안타깝게도 대부분의 대상은 이런 "자원"에서 좀 많이 동떨어진 것들이다.

하지만 메모리 관리에 적용되는 일부 얘기는 다른 객체의 관리에도 적용할 수 있다. 예를 들어 메모리 풀은 추후 할당/해제를 쉽게 하기 위해 어느 정도의 메모리를 미리 할당해 두어야 하는데 이는 네트워크 서버에서 사용하는 커넥션 풀의 개념과 동일한 것이다. 언어의 기능을 최대한 활용해서 버그의 가능성을 줄이는 것 -- 이를테면 파일을 특정 블록 안에서만 열었다 블록 끝날 때 닫는 것 -- 역시 메모리 관리랑 다를 바가 없다. 뭐... 일반론인가?

결론

웬만하면 프로그래밍 배울 때 듣는 모든 얘기는 많이 익힌 뒤에 한 번씩은 의심해 봅시다.

---

1. Accurate GC라 함은 어떤 객체가 어떤 다른 객체를 참조하는지 정확히 알 수 있다는 걸 뜻한다. C/C++에서 객체 안에 있는 포인터나 뭐 그런 것들의 목록을 받는 방법은 없으니, Bohem GC는 대신 각 워드가 모두 (일부는 잘못된) 포인터라고 가정하고 최대한 보수적으로 동작하는 conservative GC로 동작한다. 어떤 필드가 포인터는 아닌데 포인터랑 비스무리하게 생겼다면 해제할 수 있는 메모리를 계속 잡아 두고 있을 수도 있으니 당연히 비효율적이다. ↩

2. 가비지 컬렉터에 대한 오해 (덤: 블로그 정주행 내지 역주행 추천.) ↩

이 글은 본래 http://arachneng.egloos.com/1566708에 썼던 것을 옮겨 온 것입니다.

응답들

작성일
2009-07-23T05:00:10+09:00

- 대하여
- 소프트웨어
- 문서

- 저널
  - 소개
  - 모아보기
    - 최신 글들
    - 2011년
    - 2010년
    - 2009년
    - 2008년
    - 2007년
    - 2006년
    - 2005년
    - 2004년
  - 피드
- 잡것

2010-11-26 (rev 1d46270eb038)

구글 검색