**항목 50. new 및 delete를 언제 바꿔야 좋은 소리를 들을지를 파악해 두자.**

C++가 제공해주는 new/delete 는 지극히 대중적이고 모든 사용자에게 욕먹지 않게끔 적당히 무난하게 동작한다. 하지만 어느 누구에게도 칭찬들을 정도로 훌륭하진 못하다. 그렇기 때문에 개발자가 자신의 프로그램이 동적 메모리를 어떤 성향으로 사용하는 지를 제대로 이해하고 있다면, 직접 만들어 쓰는 편이 훨씬 더 우수한 성능을 낼 확률이 높다. 간혹 실행 속도에서 10의 몇 제곱 배 단위로 빨라 진다 던지 메모리를 50%나 덜 차지 하는 등의 성능 향상을 발휘하도록 개선하는 것이 어렵지 않게 구현된다.

버퍼 오버런 및 언더 런을 탐지 하기 쉬운 형태로 만들어 주는 전역 operator new를 후다닥 만들어 봤다.

|  |
| --- |
| static const int SIGNATURE = oxDEAD8EEF;  typedef unsigned char BYTE;  void\* operator new(std::size\_t size) throw(std::bad\_alloc){  using namespace std;  size\_t realSize = size + 2 \* sizeof(int); // 경계표시 2개 추가  void \*pMem = malloc(realSize);  if (!pMem)  throw bad\_alloc();  \*(static\_cast<int\*>(pMem)) = SIGNATURE;  \*(reinterpret\_cast<int\*>(static\_cast<BYTE\*>(pMem) + realSize - sizeof(int))) = SIGNATURE;  return static\_cast<BYTE\*>(pMem) + sizeof(int); // 경계표시 바로 다음 메모리를 반환  } |

위 코드는 좀 까다로운 문제를 무시했는데 그 중 하나가 바이트 **정렬(alignment)** 이다. 많은 컴퓨터의 경우 아키텍쳐적으로 특정 타입의 데이터가 특정 종류의 메모리 주소를 시작 주소로 하여 저장될 것을 요구하는 경우가 많다. 예를 들어 포인터는 4의 배수에 해당되는 주소에 맞추어 저장되어야 하며 double은 8의 배수에 해당하는 주소에 맞추어 저장되어야 한다는 것이다. 이걸 지키지 않은 경우 프로그램이 다운 될 수도 있고, 정말 다행스러울 경우 실행속도가 엄청나게 느려지는 것으로 마무리 될 수도 있다.

new/delete를 직접 구현하는 것이 생각만큼 쉽지는 않다. 꼭 만들어 쓸 경우가 없다면 굳이 들이댈 필요는 없다. 농담이 아니라 진짜 필요가 없다. 메모리 관리 함수만 전문으로 다루는 상업용 제품들도 많이 있다. 링크 한번 다시 하는 것만으로 개선된 기능과 향상된 성능을 맛 볼 수 있다는 거다.

그럼 언제 new/delete를 다른 것으로 대체하는 작업을 해야 의미가 있겠는가?

- 잘못된 힙 사용을 탐지하기 위해 : 버퍼 오버런, 언더런 (할당된 메모리 블록 앞이나 뒤를 넘어서 기록하는 것) 탐지

- 동적 할당 메모리의 실제 사용에 관한 Log를 수집하기 위해

- 할당 및 해제 속력을 높이기 위해 : 부스트의 Pool라이브러리 같이 고정된 크기의 객체만 만들어주는 등…

- 기본 메모리 관리자의 공간 오버헤드를 줄이기 위해

- 적당히 타협한 기본 할당자의 바이트 정렬 동작을 보장하기 위해 : 바이트 정렬을 보장해 주지 않는 Compiler 사용 시

- 임의의 관계를 맺고 있는 객체들을 한 곳에 나란히 모아 놓기 위해 : 페이지 부제(page fault) 발생 횟수 최소화

- 그때 그때 원하는 동작을 수행하도록 하기 위해 : 보안 강화를 위해 delete 한 메모리를 0으로 덮어쓰기 등

**요약**

- 개발자가 스스로 사용자 정의 new 및 delete를 작성하는 데는 여러가지 나름대로 타당한 이유가 있습니다. 여기에는 수행 성능을 향상시키려는 목적, 힙 사용 시의 에러를 디버깅하려는 목적, 힘 사용 정보를 수집하려는 목적 등이 포함됩니다.