Chapter 1. 전쟁이야기의 추적(Trace)

**1. 초기 추적의 구현**

초기 의도는 함수에 들어가고 함수에서 나가는 것과 같은 이벤트 메시지와 두 이벤트 사이에서 관심 있는 다른 정보를 남기는 것

|  |
| --- |
| #include <iostream>  #include <windows.h>  static bool traceisActive;  class Trace  {  public:  inline  Trace::Trace(const std::string &name) : theFunctionname(name)  {  if (traceisActive)  std::cout << "Enter Function" << theFunctionname.c\_str() << std::endl;  }    inline  Trace::~Trace()  {  if (traceisActive)  std::cout << "Exit function " << theFunctionname.c\_str() << std::endl;  }  inline  void Trace::debug(const std::string &msg)  {  if (traceisActive)  std::cout << msg.c\_str() << std::endl;  }  private:  std::string theFunctionname;  };  void myFunction(int x)  {  std::string name = "MyFunction";  Trace t(name);  // ...  std::string moreInfo = "more interstring info";  t.debug(moreInfo);  // ...  }; |

위와 같이 Trace 객체를 사용할 수 있도록 Trace를 구현하고 debug() 메소드를 호출하여 추가적인 정보 메시지를 남긴다. Trace 객체는 주요 실행 경로에 있는 대부분의 함수에서 호출 되었고 성능 테스트에서 큰 성능 감소를 보였다.

**2. 잘못된 점**

대부분 동의 하는 C++의 기본 법칙

- I/O는 부하를 많이 주는 작업이다.

- 함수 호출 오버헤드도 하나의 원인이므로 짧고 자주 호출되는 함수를 인라인으로 만들어야 한다.

- 객체를 복사하는 것은 부하를 많이 주는 작업이다. 값으로 전달(pass-by-value)보다 참조로 전달(pass-by-reference)을 사용하자.

초기 Trace 구현은 이 세가지 법칙을 모두 고수하고 있지만 성능에 많은 영향을 끼쳤다.

이 Trace는 다음의 연속적인 연산을 수행한다.

1. myFunction의 지역변수 std::string name을 생성 한다.

2. Trace 생성자를 호출한다.

3. Trace 생성자는 std::string 생성자를 호출하여 멤버 std::string을 만든다.

함수가 끝나 변수가 범위를 벗어나게 되면 Trace와 두 개의 std::string 객체는 소멸된다.

1. std::string name을 소멸한다.

2. Trace 소멸자를 호출한다.

3. Trace 소멸자는 멤버 theFunctionName의 std::string 소멸자를 호출한다.

추적이 꺼진 경우 std::string 멤버 객체는 절대 사용되지 않는다. 또한 Trace 객체도 사용되지 않는다. 이러한 객체의 생성과 소멸에 의한 낭비이다.

얼마나 많은 부하를 주는 지 알아 보기 위해 addOne1, addOne2 함수를 추가 하였다.

|  |
| --- |
| int addOne1(int x)  {  return x + 1;  };  int addOne2(int x)  {  std::string name = "addOne";  Trace t(name);  return x + 1;  };  int main()  {  traceisActive = false;  ULONGLONG s = GetTickCount64();    for (int i = 0; i < 1000000; i++)  addOne1(i);  ULONGLONG e = GetTickCount64() - s;  ULONGLONG s\_2 = GetTickCount64();  for (int i = 0; i < 1000000; i++)  addOne2(i);  ULONGLONG e\_2 = GetTickCount64() - s\_2;  std::cout << "case 1 : " << e << std::endl;  std::cout << "case 2 : " << e\_2 << std::endl;  return 0;  } |

addOne1 함수는 단순히 넘어온 x 값을 1 증가 시킨 후 반환하고 addOne2는 Trace 객체를 생성하는 과정을 거치며 traceisActive = false 이기 때문에 로그는 남기지 않는다.

결과

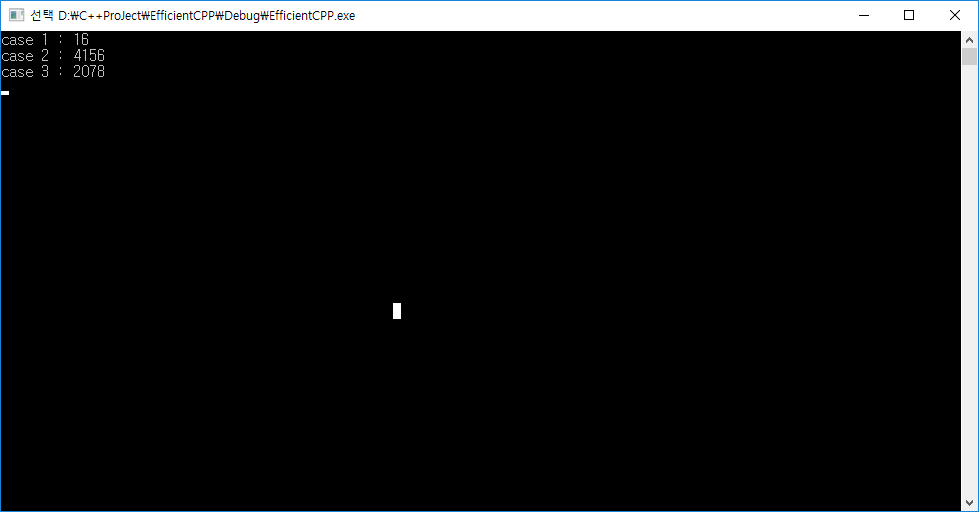


**3. 복구 계획**

위에서 case 1, 2을 살펴 보았다.

**case 3. string 객체 대신 char 포인터를 사용하라.**

|  |
| --- |
| inline  Trace::Trace(const char \*name) : theFunctionname(name)  {  if (traceisActive)  std::cout << "Enter Function" << name << std::endl;  } // case 3  int addOne3(int x)  {  char \*name = "addOne";  Trace t(name);    return x + 1;  }; |

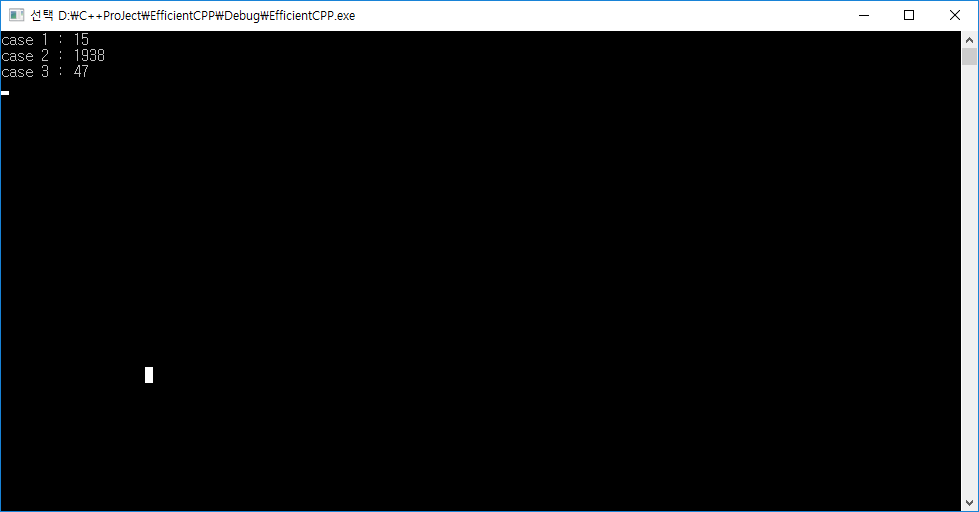


- std::string 대신 const \*char를 입력인자로 받아들이도록 trace::debug() 메소드도 수정하였다. 이제 Trace 객체를 만들기 전에 name string을 만들 필요가 없다. 고려해야 할 객체가 하나 줄었다.

**case 4. string 포인터를 사용하라.**

Trace 클래스의 멤버 std::string theFunctionname 을 std::string \*으로 변경한 후 traceisActive 가 true 상태일 때만 객체를 생성할 수 있도록 수정한다.

|  |
| --- |
| inline  Trace::Trace(const char \*name) : theFunctionname(nullptr)  {  if (traceisActive)  {  std::cout << "Enter Function" << name << std::endl;  theFunctionname = new std::string(name);  }    } // case 4 |



- Trace 객체에 포함된 std::string 멤버 객체의 무조건 적인 생성을 방지하는 것이다.

**키 포인트**

- 객체를 정의하면 객체 생성자와 소멸자의 형태로 조용한 실행(silent execution)이 발생한다. 객체 생성과 소멸은 대게 부하가 아니기 대문에 우리는 “조용한 오버 헤드”라는 말 대신 “조용한 실행”이라는 말을 사용한다. 만약 생성자와 소멸자가 수행하는 연산이 항상 필요하다면, 이 연산 들을 효율적인 코드로 간주되어야 한다. (생성자와 소멸자를 인라인으로 만들면 호출과 반환 오버헤드를 줄여줄 수 있다.) 앞에서 보았듯이, 생성자와 소멸자는 항상 순수한 특성만을 가지는 것은 아니고 현저한 오버헤드를 생성할 수 있다. 어떤 상황에서는 생성자, 소멸자가 수행한 연산이 사용되지 않는다. 이것은 C++ 언어적인 주제라기 보다 디자인 주제에 밀접하다. 하지만, C는 생성자/소멸자의 개념을 갖지 않기 때문에 C코드에서 이러한 상황은 잘 발생하지 않는다.

- 객체를 참조로 전달하는 것도 좋은 성능을 보장하지 않는다. 객체 복사를 수행하지 않으면 도움이 되겠지만, 우선 객체를 생성하고 소멸할 필요가 없다면 더욱 도움이 될 것이다.

- 잘사용되지 않는 결과를 만들어 내는 연산을 방지 한다. 추적이 꺼진 경우, string 멤버의 생성은 가치가 없고 부담이 많이 든다.

- 디자인의 유연성에 너무 집착하지 말자. 여러분이 필요로 하는 것은 문제 영역에 적합한 정도로 유연한 디자인이다. char 포인터는 가끔 string에 비해서 더 효율적이면서 단순한 작업을 잘 수행한다.

- 인라인, 작은 함수를 자주 호출하여 발생하는 함수 호출 오버헤드를 제거한다. Trace 생성자와 소멸자를 인라인으로 만들면 Trace 오버헤드를 줄이기 쉬워진다.