GNU-C++の -Os が壊れた日

フリースタンディング環境の C と C++を SuperTest™でサポートするプロジェクトで作業していて、あるパターンが出現するまでは C++コンパイラテストプログラムのデバッグを問題なく行っていました。

フリースタンディング環境は C/C++ライブラリのサブセットだけをサポートするように定義されており、その目標はちゃんとした OS にあるような豊富な実行時システムをもたないターゲットでアプリケーションを実行させることです。それどころか「ベアメタル（OS のないマシン）」でプログラムが実行できます。このことは組込みアプリケーションすべてで有益です。

このプロジェクトの目標には SuperTest のテスト実行時に必要なメモリを最小にするということもありますので、コンパイラをメモリ最適化オプション ***-Os*** で使用しています。

ところが C++のテストで例外処理や継承に関するもののいくつかが、新しいフリースタンディングの設定では突然フェイルしました。

テストがこのようにフェイルしたとき最初に疑うのは自身が行った変更です。なんといっても使ったコンパイラは完全に最新の x86 Ubuntu 環境でデフォルトの g++コンパイラなのですから。何千(何百万？)ものミッションクリティカルなシステムが GNU-C++と Ubuntu を使っています。注目度の高い例として名前を挙げるなら、Space-X もロケットに x86 上の C++と Linux を使っています(https://www.rankred.com/what-hardware-software-does-spacex-use-topower-its-rockets/によります)。

以下はそのようなテストの一つです。たしかにこれはよい C++ではありませんが、よい C++を書くのが我々の仕事ではありません。コンパイラをテストするためになすべきことは、言語仕様の際どい部分に近づいて実装が正しく動作するのを検証することです。

```
#include <cassert>
class A {
    virtual void f() {};
  public:
    int x;
    A(int in): x(in) {};
};

class B: public A {
  public:
    int y;
    B(int in): A(in-1), y(in) {};
};

int test(void)
{
    int res;
```

```
    B b(2);
    A* bp = &b;
    void* vp = dynamic_cast<void*>(bp);
    if ( ((A*)vp)->x == 1
      && ((B*)vp)->y == 2 ) {
        return 1; // PASS
    } else {
        return 0; // FAIL
    }
}

int main (void)
{
    assert (test());
}
```

このコードは複雑ではありませんが、明確な定義ではあるものの不要な型のキャストがいくつかあります。クラス B はクラス A を継承し、A には x、B には y のインスタンス変数があります。関数 test の 2 行目にあるクラス B のオブジェクトのインスタンス化は、これらの変数をそれぞれ 1 と 2 で初期化します。if 文で、型のキャストを二つ行った後これらの値が確認されます。

最初に、関数 test()に対する g++の -O1 オプションでの x86-64 コード生成の美しさを称賛しましょう：

```
test():
  mov DWORD PTR [rsp-8], 1
  mov DWORD PTR [rsp-4], 2
  mov rdx, QWORD PTR vtable for B[rip]
  movabs rax, 8589934593
  cmp QWORD PTR [rsp-8+rdx], rax
  sete al
  movzx eax, al
  retA::f():
  rep ret
```

このコンパイラは型 **B** のオブジェクトのレイアウトが分かっています。最初の二つの move 命令で変数 **x** と **y** に値が設定されます。そしてオブジェクト **b** の要素を 64 ビットの即値 **8589934593** と比較します。この値は十六進表記で **0x0000000200000001** とするとより分かりやすいでしょう。コンパイラは二つの 32 ビットの値の比較を一つの 64 ビットの値の比較！にします。これは賢い move 命令です。ここまでは順調です。

しかし、次にサイズ最適化のために **-Os** でこれをコンパイルすると、生成されたコードは次です：

```
test():
  mov rdx, QWORD PTR vtable for B[rip]
  mov DWORD PTR [rsp-8], 1
  movabs rax, 8589934593
  cmp QWORD PTR [rsp-8+rdx], rax
  sete al
```

```
movzx eax, al
ret
```

なくなったものが分かるでしょうか？このコンパイラはどういうわけか **b** の変数 **y** を **2** に初期化するのを忘れてしまっています！そしてテストプログラムは `assert` でフェイルするのです。

これは重大なエラーです。二つの比較を一つにする最適化のせいでコンパイラはオブジェクト **b** の要素 **y** が使用されているのを「忘れた」のです。二つの比較を最適化した後の定義-使用解析では **y** の使用は見えません。そこで、コンパイラは要素 **y** の初期化が冗長だと判断し、それを削除したのです。この振る舞いは型キャストと組み合わさった比較の最適化に限ったものでしょうか？そうかもしれませんが、保証はできません。

エラーはこれだけではありません。**-Os** では生成された例外処理のコードにもエラーがあります。Ubuntu の現バージョン[1]ですので、ここでの GNU-C のバージョンは 7.3.0 です。GNU-C の別のバージョンや ARM64 ターゲット、Ubuntu に関連しないものでも試した結果、すべてのものが影響を受けていました。64 ビットのターゲットでは **g++** を **-Os** オプションと組み合わせて使用することは安全ではないと結論づけなければなりません。

コンパイラ開発者は、このようなエラーがすり抜けてしまうことを防ぐために多くのテストを実施します。しかしコンパイラには複雑で非常に多くの設定オプションがあるので、ユーザの特定のユースケースが検証されたと率直に明言できるコンパイラサプライヤーはありません。

ミッションクリティカルかセーフティクリティカルなアプリケーションドメインなら、コンパイラ検証を対象コンパイラと固有のユースケースに対してセットアップする必要があります。そこまでの必要がない場合でも、最低限そのコンパイラサプライヤーが SuperTest を使用していることを確かめてください。SuperTest は私どもが知るどの手法よりもコンパイラエラーに先んじる機会をもたらします。Space-X が **g++** を **-Os** オプションで使用していないことを願うばかりです。

富士設備工業株式会社 電子機器事業部　www.fuji-setsu.co.jp

1 （訳注）Ubuntu18.04