C/C++標準ライブラリのテスト

C/C++言語のプログラムを構成する要素には文字列処理や数学関数のようにプログラム実行時に必要となるソフトウェア部品があります。アプリケーションプログラムが安全であるためには、コンパイラが生成するオブジェクトコードの部分だけでなく、これらも検証することが必要です。実行時に必要となるこのようなソフトウェア部品は実行時ライブラリと呼ばれますが、その中で言語規格で定義されているものを標準ライブラリと呼び、プログラム開始時に必要となるスタートアップルーチンや例外処理ルーチンなどを実行時ライブラリとして区別することもあります。この記事では、標準ライブラリと実行時ライブラリに関する SuperTest での検証について紹介しています。

標準ライブラリ関数をテストする

原文：Functions in the standard library are usually taken for granted　2020/7/2
https://solidsands.com/functions-in-the-standard-library-are-usually-taken-for-granted

たいていの組込みシステムのソフトウェアにとって標準ライブラリは基本部分であり、標準ライブラリがないと組込みシステムは動きません。しかし、標準ライブラリの関数は通常当たり前のものとみなされ、適切にはテストされません。C/C++の言語規格では、個々の機能を呼び出すインターフェースを定義してライブラリ関数を定義しています。ライブラリ関数は SDK によって自動的にアプリケーションに追加され、アプリケーション開発者はコードを手書きで実装するのではなく、よく知られている関数機能としてそれを使用できます。このアプローチによって時間の節約だけでなくコードを短く、よりロバストにできます。つまり、ソフトウェア開発者は標準ライブラリ関数に大きく依存するようになるのです。

ライブラリ関数には、入出力のような深く組込まれたシステムでは多く使用されないものもありますが、セーフティクリティカルなコンポーネントでカギを握るものもあります。たとえば、自動車のブレーキ圧やスロットルバルブ開度の計算では、浮動小数点計算の数学ライブラリ関数が極めて重要な役割を果たします。このように C 標準ライブラリのコードが最終的にデバイス上に残るものでは、その関数が認定されないままであってはなりません。ISO 26262 機能安全ではライブラリの認定に対して 2 つの方法を示しています。

機能安全テスト

機能安全のプロセスで C/C++標準ライブラリについて考えないことは容易です。なぜなら、アプリケーションの構築においてライブラリコードは見えず、そして自動的に追加されるからです。しかし、機能安全の場面では、ライブラリコードは考慮されるべきものなのです。

COTS ソフトウェアに対する ISO 26262 のプロセスは、ASIL C までのライブラリに対して、そのソースコードにアクセスすることなく適用できます。ASIL D に対しては構造コードカバレッジ解析が必要であり、ソースコードへのアクセスが必要になります。COTS でないライブラリに対しては、アプリケーションソフトウェアに対する一般の要件が適用されますが、それは非常に高価になります。

SuperTest は標準ライブラリを考慮しています

いずれにせよ、標準ライブラリのための優良なテストスイートが必要で、SuperTest は C 標準ライブラリの完全な定義である ISO の C 言語規格に基づいており、標準ライブラリの認定で使用できる最も広範なテストスイートです。ライブラリのテストにふさわしい妥当性を与え、SuperTest はライブラリの認定に不可欠なツールです。

C/C++の実行時ライブラリ -「隠れたライブラリ」をお忘れなく

原文：Don’t forget the ‘hidden library’ in C and C++　2020/7/22
https://solidsands.com/dont-forget-the-hidden-library-in-c-and-c

プログラムを書くことは難しく、ありふれたタスクでも再実装することに誤りはつきものです。そこでソフトウェア開発者は、標準ライブラリの関数に非常に依存することになります。それによって開発者の仕事は楽になり、生成されるコードはより信頼できるものになります。ところで、コンパイラも自身の仕事を単純にするためライブラリを使用していることをご存知でしょうか？ それは実行時ライブラリと呼ばれるものです。

この隠れたライブラリは、スタートアップ時に OS とのインターフェースをとる関数やハードウェアの初期化、ターゲットに存在しない操作の実装、そして他の多くのターゲットに関連するタスクなどを提供します。しかし、実行時ライブラリで注目すべき項目は、それが隠れているということです。実行時ライブラリはアプリケーション開発者が使用するようには考えられておらず、言語規格から直接見えることはありません。さらに、ISO 26262 や他の機能安全標準での要件として特定することも困難です。それでも、実行時ライブラリは実装において重要な部分なのです。

実行時ライブラリの関数は、言語が定義する動作の一部分を実装するためのものです。SuperTest のテストスイートは言語の定義を検証する優れた方法であり、それが実行時ライブラリの正しい動作を検証することにもなります。すなわち、SuperTest によってプログラムのスタートアップと終了の動作やターゲットの命令として実装されていないものも含むすべての算術演算の正しさを検証できます。SuperTest は言語仕様も実行時ライブラリも完全に検証するのです。

実行時ライブラリと標準ライブラリを混同しないでください。アプリケーション開発者はコードを実装することなく、明確に定義され、よく知られている機能を得るために標準ライブラリを使用します。それによって時間を節約し、コードを短くよりロバストにもできます。

以下で標準ライブラリとライブラリの認定について詳細をお話しします。

ライブラリコードはセーフティクリティカルアプリケーションの一部分

原文：Library code is part of your safety critical application　2020/9/23
https://solidsands.com/library-code-is-part-of-your-safety-critical-application

セーフティクリティカルなアプリケーションのコードを書くには多くの配慮や注意が必要ですが、何よりも必要なのは非常に効果的な検証です。それは、あなたが書くコードだけに対するものではありません。アプリケーションにリンクされる標準ライブラリのコードも安全性が要求される最終コンポーネントの一部となるため、使用する標準ライブラリにも適用されるのです。すなわち、ライブラリコードも自分のコードに適用するのと同じ基準に準拠していることを確認する必要があります。

Filename	Line Coverage		Functions		Branches	
fmax.c	100.0 %	8 / 8	100.0 %	1 / 1	100.0 %	10 / 10
fmaxf.c	100.0 %	8 / 8	100.0 %	1 / 1	100.0 %	10 / 10
fmaxl.c	100.0 %	8 / 8	100.0 %	1 / 1	100.0 %	10 / 10
fmin.c	100.0 %	8 / 8	100.0 %	1 / 1	100.0 %	10 / 10
fminf.c	100.0 %	8 / 8	100.0 %	1 / 1	100.0 %	10 / 10
fminl.c	100.0 %	8 / 8	100.0 %	1 / 1	100.0 %	10 / 10
fmod.c	100.0 %	42 / 42	100.0 %	1 / 1	100.0 %	32 / 32
fmodf.c	100.0 %	42 / 42	100.0 %	1 / 1	100.0 %	32 / 32
frexp.c	100.0 %	15 / 15	100.0 %	1 / 1	100.0 %	6 / 6
frexpf.c	100.0 %	15 / 15	100.0 %	1 / 1	100.0 %	6 / 6
frexpl.c	100.0 %	15 / 15	100.0 %	1 / 1	100.0 %	6 / 6
hypot.c	100.0 %	37 / 37	100.0 %	2 / 2	100.0 %	14 / 14
hypotf.c	100.0 %	24 / 24	100.0 %	1 / 1	92.9 %	13 / 14
hypotl.c	100.0 %	38 / 38	100.0 %	2 / 2	100.0 %	16 / 16
ilogb.c	100.0 %	15 / 15	100.0 %	1 / 1	90.0 %	9 / 10
ilogbf.c	100.0 %	15 / 15	100.0 %	1 / 1	90.0 %	9 / 10
ilogbl.c	100.0 %	14 / 14	100.0 %	1 / 1	90.0 %	9 / 10

テストによって高いコードカバレッジ（ステートメントカバレッジやラインカバレッジ）を達成することは、ISO 26262 など機能安全規格の要件の一つです。しかし、テストによる機能仕様に対する検証では、必ずしも良好なコードカバレッジが得られるとは限りません。

SuperTest の C 標準ライブラリ用のテストスイートを使用することが良いコードカバレッジのための素晴らしい出発点になります。以下、その理由を示します。

コンフォーマンステストではカバレッジ目標を逃す可能性がある

たいていの場合、ライブラリは選択されたソフトウェア開発キット全体の一部分であり、多くはコンパイラの機能や品質に基づいたものになっています。その結果、アプリケーションにリンクされているライブラリコードをコントロールすることは、通常はできません。

もちろん、コンフォーマンステストを使ってライブラリが期待通りに動作するかどうかを検証することはできますが、コードカバレッジも測定すると失望する結果になるかもしれません。コンフォーマンステストが高いコードカバレッジも達成できれば一石二鳥でいいのではないでしょうか。そこで Solid Sands では、SuperTest のコンフォーマンステストスイートがライブラリのコードカバレッジも高水準で実証することを確実にするよう強化しています。

課題

テストの実施によって高いコードカバレッジを達成するには、実装の点から以下のような課題があります。

1) C コンパイラが関数呼び出しをより効率的なものに置き換えることがあり、必要なカバレッジが得られない
2) 多くの関数がマクロとしても実装されており、両者をテストしてカバレッジをとる必要がある
3) 間接的にのみ呼び出される静的なヘルパー関数があり、それらは文書化されていない
4) 冗長な分岐が存在し、100%のカバレッジを妨げることになっている
5) ターゲット固有の設定コードがある

結果

この機能強化の開始時点で既に、C 言語規格から本質的に導かれるテストスイートによって期待できるものと同程度の結果が得られていました。しかし、世界トップレベルのコンパイラテスト・検証専門会社としては「期待できるものと同程度」では十分ではありません。

そこで開発を強力に推進し、現時点で SuperTest は、標準 C ライブラリの完全な実装に対して 90%を超えるコードカバレッジを提供することができました。全体の中で重要な部分を占める数学ライブラリに対しては、究極の基準である 100%に近づいています。さらに、我々はライブラリ全体で「究極の基準」を達成するよう強力に取り組んでいます。

ご使用のライブラリに対するコンプライアンステストとコードカバレッジについてご興味がございましたらご連絡ください。

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