SuperGuard: セーフティクリティカルなアプリケーションで使用されるC標準ライブラリの認定

はじめに

ソフトウェアソリューションの役割は、セーフティクリティカルシステムやセーフティ関連システムでますます重要なものとなっています。今や、ソフトウェアの誤動作は責任を負うべきものであり、そして傷害や生命の損失、必要不可欠なサービスの中断などの観点から現実の脅威となっています。そこで、ISO（国際標準化機構）やIEC（国際電気標準会議）などの国際標準化団体では、ソフトウェア開発者が自社のソフトウェアの安全性を証明するための規格を公表しており、それらは広く認知・採用されるようになっています。自動車向けのものではISO 26262（*自動車－機能安全*）、鉄道輸送向けではEN 50128（*通信、信号および処理システム－鉄道の制御、保護システム用ソフトウェア*）、産業アプリケーション向けではIEC 61508（JIS C 0508 *電気・電子・プログラマブル電子安全関連系の機能安全*）がそのような規格の例です。

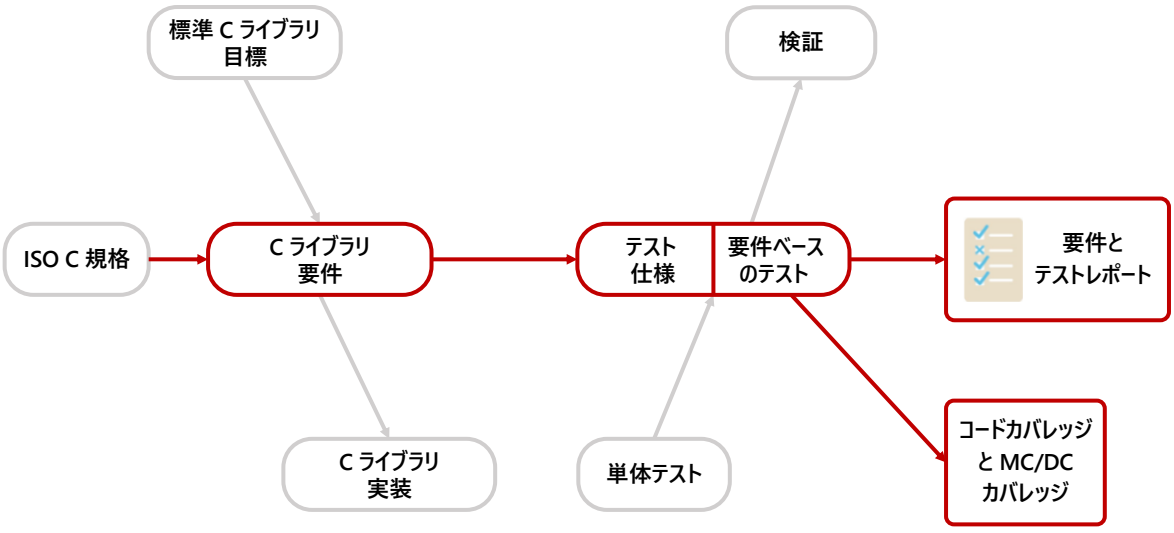
アプリケーションソフトウェアとその開発に使用するソフトウェア手法・プロセス・ツールチェーンが、関連する機能安全規格に準拠していることを実証する責任は、はっきりとアプリケーション開発者にあり、一方で事実として、ツールチェーンの重要な部分は開発者のコントロール外です。このことがセーフティクリティカルシステムの開発者にとって、コンパイラの検証が重要な課題となっている理由の一つであり、このコンパイラ検証の分野ではSolid Sands社が世界のリーダーです。事実上バグのないコンパイラはないので、コンパイラが誤動作する箇所を知ってコンパイラエラーを回避することは非常に重要です。

完全なアプリケーションの一部となるコードの重要な部分が、アプリケーション開発者が使用するのとは異なるユースケース、コンパイラオプション群、そしてコンパイル環境を使用してコンパイルされる可能性が高いことも事実です。これは、最終のアプリケーションコードの一部にはC標準ライブラリ（libc）にあるようなコンパイル済みのライブラリ関数が含まれており、それはソフトウェア開発キット（SDK）の一部としてバイナリ形式で提供されることが多いからです。

ライブラリはバイナリ形式で提供されているため特定のユースケースに依存しない、すなわちコードは不変であると一般には思われていますが、実際はそうではないのです。マクロ

や型汎用のテンプレートを含めることで、ライブラリコンポーネントがユースケースに依存することはよくあります。そのため、SDK のサプライヤーによって SDK と同じコンパイラを使用してライブラリが事前に認定されている場合でも、合致するユースケースやコンパイラオプション、ターゲットハードウェア環境の要件が満たされていないという可能性は、ほぼ確実にあり、そのため機能安全規格に準拠していることの実証は困難なのです。

この制限を克服するため、Solid Sands 社は新しいライブラリ認定ツール SuperGuard C Library Safety Qualification Suite（SuperGuard C ライブラリ安全認定スイート）を開発しました。これは、個々のテスト結果と ISO C 言語仕様から派生する要件とを結びつける完全なトレーサビリティを備えた C 標準ライブラリに対する要件ベースのテストスイートです。これは、サードパーティ製のライブラリ実装をそのまま使用する場合と自己開発/自己保守型で実装を行う場合との両方について、セーフティクリティカルアプリケーションに対する C 標準ライブラリの認定をサポートするために使用できます。ソフトウェア開発のための V モデルにおける SuperGuard の役割を下図に示します。



認定の目標

ライブラリのコードはアプリケーションとリンクしてターゲットデバイス上で実行されます。そのため、ライブラリコンポーネントに欠陥があるとアプリケーション全体の機能安全が危険にさらされるので、ソフトウェアライブラリの認定が不可欠です。ソフトウェアライブラリの使用に対して機能安全規格それぞれで独自の目標がありますが、一般的には、ライブラリの実装がその仕様に準拠していることを検証するという目標は共通です。ISO 26262 にはライブラリの認定へのルートが二つあり、それぞれ Part 8 と Part 6 で詳述されています。SuperGuard C Library Safety Qualification Suite は、そのどちらにも使用できます。

ISO 26262 Part 8 Clause 12: ソフトウェアコンポーネントの認定

COTS (既製品) であるという点で、ライブラリは Part 8 Clause 12「ソフトウェアコンポーネントの認定」でカバーされます。この条項は、「再利用に適していることの証拠を提供する」ため、ライブラリなど既存ソフトウェアコンポーネントを認定することの必要性を扱うものです。具体的には、商用 SDK に付属する標準ライブラリを含む「サードパーティサプライヤー製のソフトウェアライブラリ」について述べているもので、この条項は再利用される内製ソフトウェアやオープンソースソフトウェアにも適用されます。

Part 8 Clause 12 では、認定の前提はソフトウェアコンポーネントの要件の言明である、としています。また、ソフトウェアコンポーネントがこれら要件に準拠していることの証拠が「第一には要件ベースのテストに基づく」ものでなければならないこと、それが「専用の認定テストスイートの適用」で達成できること、「正常時の動作条件と故障時の挙動の両方をカバー」しなければならないこと、そして「安全要件の違反につながり得る既知のエラーがないことを表明」しなければならないことを示唆しています。

C/C++ 言語にはライブラリの仕様が存在し、一般に公開されているため、それが厳格な要件ベースのテストの開始点になります。実際、言語とライブラリの両方の仕様が存在し、かつ両者をテストできることが、開発者コミュニティで C/C++ 言語が広く普及している理由の一つです。ただし、言語仕様は要件リストとしては記述されていません。Solid Sands 社の SuperGuard テストスイートの重要な特徴は、そのすべてのライブラリテストが ISO C 言語仕様から導出される要件に基づいているということです。

テストスイートでは正常条件と故障条件の両方をカバーしなければならないという要求を満たすには、境界条件の内外両側で各関数を実行してエラー処理を検証する必要があります。SuperGuard を作るために ISO C規格から導出した要件には、その仕様で定義されている必要な故障動作の検証が含まれています。

安全要件の違反につながり得る既知のエラーがないことの信頼できる証拠は、効果的な要件ベースのテストに依存するだけでなく、自動車用アプリケーションの最高の安全レベルである ASIL D で求められる構造カバレッジにも依存します。完全性の証拠を確保するため、SuperGuard のテストは高い構造カバレッジと Modified Condition/Decision Coverage (MC/DC) を満たします。

SuperGuard には、同値クラスと境界値の分析とテスト、そしてライブラリ関数の動作に関する最良の知識と経験に基づいたエラー推測も含まれています。

ISO 26262 Part 6: ソフトウェアレベルでの製品開発

ISO 26262 Part 6では並列アプローチを採用し、ターゲット上で実行される他のすべてのアプリケーションコードと同じようにライブラリコードを処理します。皮肉なことに、この規格ではライブラリをコードの別カテゴリのものとしては言及しておらず、そのためライブラリ認定の必要性がしばしば見落とされるのかもしれません。Part 6はソフトウェア開発のすべてのフェーズに対応するものなので、そこで説明されているC標準ライブラリの認定はPart 8で説明されているアプローチよりも複雑です。

Part 6の表7「ソフトウェアユニットの検証方法」では、ASILのすべてのレベルに対して要件ベーステストを推奨しています。実際に開発者がC標準ライブラリの実装を検証するためには、この表に記載されている複数の手法を使用することが推奨されていますが、先に「**ソフトウェアコンポーネントの認定**」で述べたように、SuperGuardは要件ベーステストを包括的にサポートしています。

SuperGuardではPart 6の表8「ソフトウェア単体テストのテストケース導出方法」に記載されている手法もすべて採用しています。方法 1a「要件の分析」は、C標準ライブラリ仕様をテスト可能な要件に分解するために使用されます。C標準ライブラリの記述は、(時には暗黙である)定義と関数の使用に対する制約、そして関数の動作の定義が混在したものです。そこで、SuperGuardでは、それらを言語規格が定義する例外ケースの処理に対するものも含めてテスト可能な要件に変換し、テストスイートの基盤としています。

方法 1b「同値クラスの生成と分析」と 1c「境界値の分析」は関数への入力値の変域を分割することに関連しており、SuperGuardのテスト仕様に記載されています。このテスト仕様は要件とテストの間のリンクを提供するものです。

方法 1d「知識や経験に基づくエラー推測」は、Cライブラリの難しい実装分野に関するSolid Sands社の知識と、SuperTestコンパイラテスト・検証スイートの開発当初から30年を超えて様々なC/C++言語仕様やそのコンパイラに対して繰り返し実行してきた一種のリグレッションテストの経験とその蓄積に基づいています。

このアプローチにおいて、SuperGuardはソフトウェア開発のVモデルの右側で大きな役割を果たしており、ISO 26262 Part 6 Clause 9「ソフトウェア単体検証」に関連して、以下の項目を扱います。

- 標準Cライブラリ実装が、その要件に準拠すること
- ターゲットハードウェアでのテスト実行によるハードウェアーソフトウェアインターフェースの検証
- 故障ケースの検証とコードカバレッジ解析による意図しない機能がないことの確信
- ターゲットハードウェアでのテスト実行によるリソース要件の検証

SuperGuardテストの開発法

ISO 26262 の Part 8 と Part 6 で推奨される要件ベーステストを実装する場合、C/C++ 標準ライブラリの仕様で主として問題になるのは、各関数の詳細な動作記述はあるが、そのどれもが明確な要件のセットを定義していないということです。したがって、各関数に必要な要件は、その動作記述から作成する必要があります。

SuperGuard C Library Safety Qualification Suite には、Solid Sands 社が 30 年以上 ISO 言語規格に対応してきた世界最高水準の SuperTest コンパイラテスト・検証スイートがもつ実証済みの C 標準ライブラリテストスイートが組み込まれています。SuperGuard は、さらに ISO 26262、EN 50128、IEC 61508 などの機能安全規格に準拠し、そのレポート機能、要件の文書化、またテストの内容について、SuperTest と比較して、ライブラリの認定向けに強化されています。

SuperGuard テストスイートのテストは以下の原則で設計されており、幅広い開発環境に適したものになっています。

SuperGuard のテストは挙動のテストであり、実装された動作がライブラリの仕様に準拠しているかを検証するものです。各テストでは、テスト対象の構成要素や関数を実行し、実行結果をライブラリ仕様で定義される期待値（モデル）と比較します。テスト自体は、成功または失敗をテストドライバーに報告します。

実装された動作を確認するために、SuperGuard のテストはコンパイルして実行されます。これはターゲットに搭載されるプロセッサ用の開発ツールチェーン全体が関与するということであり、ライブラリが使用される実際のターゲットハードウェア、あるいはそれと等価な実行環境でライブラリを検証します。

ライブラリのフリースタンディング部分（通常はベアメタルシステムで使用される）のテストでは、使用リソースが最小限であることが必要です。SuperGuard のテストのほとんどはメモリが 4K 未満のシステムで実行できるため、非常に小さな組込みシステムでも SuperGuard を使用できます。

要件ベーステストを実装するため SuperGuard は、C 標準ライブラリの仕様をテスト可能な実装要件に詳細に分解し、各要件をどのようにテストするかを説明するテスト仕様とともに提供します。個々のテスト実行結果を対応するテスト仕様・テスト要件・標準ライブラリ関数にリンクすることで、SuperGuard は要件ベーステストに必要な完全なトレーサビリティを提供します。完全性に対する証拠のために、高い MC/DC を実現し、80% 超の関数に対して 100% 近い構造コードカバレッジを満たします。なお、これは OS 層ではなくコンパイラが提供する標準ライブラリの実装自体を対象とするものであることに注意してください。

例

SuperGuardスイートの各ライブラリテストは、以下の**strncpy**関数の例で示すように一貫した方法論に従って開発されています。これはC99言語規格の7.21.2.4項の仕様です。

7.21.2.4　strncpy関数

形式

1 ***#include <string.h>***
char *strncpy(char * restrict s1, const char * restrict s2, size_t n);

機能

2 ***strncpy** 関数は, **s2** が指す配列から **s1** が指す配列に, **n** 個以下の文字（ナル文字に続く文字はコピーしない。）をコピーする。領域の重なり合うオブジェクト間でコピーが行われるとき, その動作は未定義とする。*

3 ***s2** が指す配列が **n** 文字より短い文字列である場合, **s1** が指す配列中のコピーの後ろに, 全体で **n** 文字書き込むまでナル文字を付加する。*

返却値

4 ***strncpy** 関数は, **s1** の値を返す。*

この仕様で最も興味深い点は、段落2は**strncpy()**がコピーする文字数の下限を指定していないことです。「***n** 個以下の文字をコピーする*」とあり、この仕様はどの時点でも **s2** から **s1** にいくつかの文字をコピーすることは要求していません。段落3では **s1** にナル文字を付加するという動作を示しています。この仕様に文字どおりに従うと、単に **s1** に **n** 個のナル文字を書き込むことも正しい実装でしょう。

これでは、この関数が行うことの意図や期待される動作を表していません。一般的な理解は、文字列 **s2** か **n** が尽きるまで、できるだけ多くの文字を **s2** から **s1** にコピーするということです。それについて混乱はなく、同じ文言が C18 に存在しているので、ANSI C89 以来この言い回しについて不満はないようです。しかし、要件を定義するにはもう少し正確さが必要です。

テスト開発プロセスの最初のステップは、この関数が実際に行うべきことを考慮して上記の説明から一連の要件（REQ）を抽出することで、以下のとおりです。

REQ-copystring : ***s2** が **n** より小さい長さ「**l2**」（これは **strlen()** で定義される）の文字列を指している場合、**strncpy()** は、配列 **s2** から **l2** 個の文字を順番どおりに、配列 **s1** にコピーしなければならない。*

REQ-copyn : ***s2*** *が* ***n*** *未満の長さの文字列を指していない場合、* ***strncpy()*** *は、配列* ***s2*** *から最初の* ***n*** *個の文字を順番どおりに、配列* ***s1*** *にコピーしなければならない。*

REQ-shorter : ***s2*** *が* ***n*** *未満の長さの文字列を指している場合、* ***strncpy()*** *は、配列* ***s1*** *にコピーされた文字の後ろに、合計で* ***n*** *個の文字が書き込まれるまでナル文字 (* ***'\0'*** *) を付加しなければならない。*

REQ-nomore : ***strncpy()*** *は、ターゲットの配列* ***s1*** *に、最初の* ***n*** *文字を超えて書き込んではならない。*

REQ-nochange : ***strncpy()*** *は、配列* ***s2*** *を変更してはならない。*

REQ-return : ***strncpy()*** *は、* ***s1*** *の値を返さなければならない。*

要件 **REQ-nochange** は、**s2** が定数 (**const**) 配列として宣言されたことによるものですが、この宣言自体は、***strncpy()*** の実装で **s2** への書き込みがないことを保証するものではありません。

これらの各要件に対してテスト仕様が作成されます。テスト仕様では、要件が true であることをテストで検証する方法を定義します。一つのテスト仕様は通常、関数の入力と出力の変域をカバーするさまざまなテストケースにつながります。それらテストケースがテストによって実装されます。テスト仕様は要件をテストに結びつけるものです。

たとえば、要件 **REQ-copystring** と **REQ-nomore** に対するテスト仕様は、以下のとおりです。

REQ-copystring に対するテスト仕様: *パラメータ* ***n*** *の値 (****n==0*** *を含む) を元の文字列の長さ以上の異なるものとして* ***strncpy()*** *関数を呼び出す。元の文字列が終端のナル文字までターゲットの配列にコピーされていることを確認する。*

REQ-nomore に対するテスト仕様: *すべてのテストケースで、ターゲットの配列* ***s1*** *の添字* ***n*** *の位置の文字が変更されていないことを確認する。このテストに適合する場合には、****n*** *を超える添字の位置の文字も変更されていないことを確認する。*

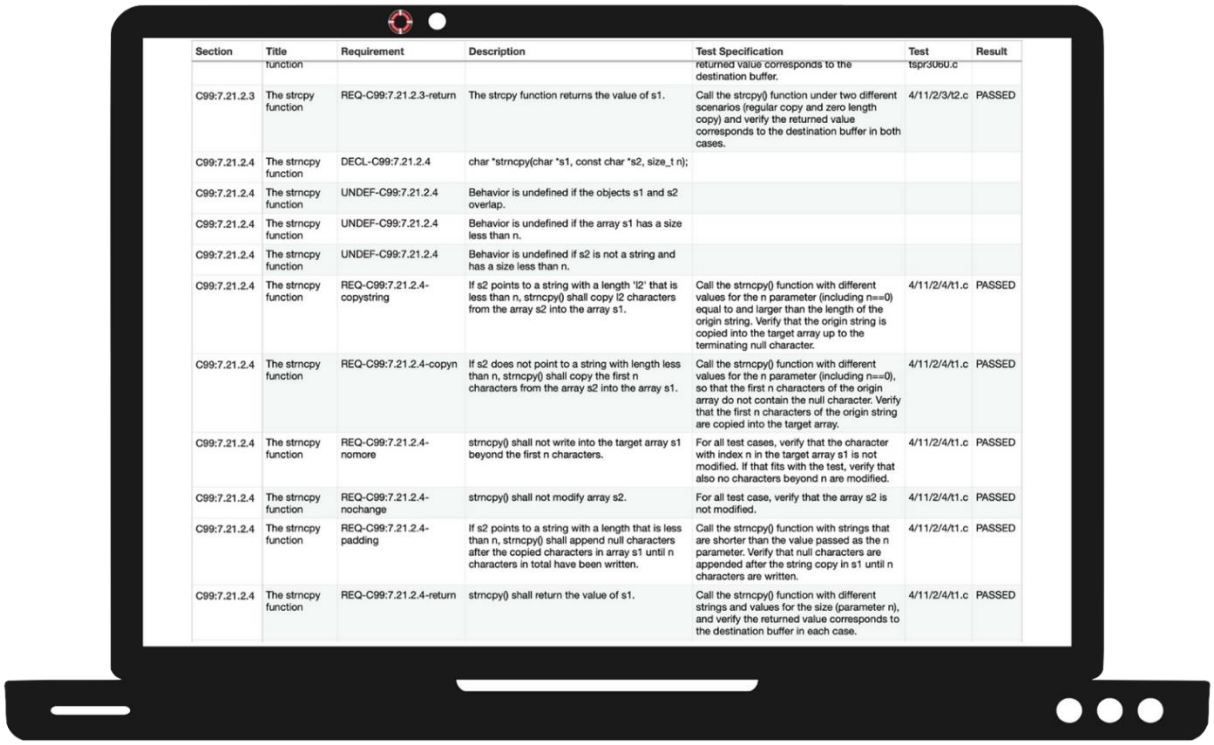
このケースでは、テスト仕様 **REQ-nomore** は、**strncpy()** に対する他のテストケースと同じテストファイルに実装されます。この要件は **strncpy()** の呼び出しごとに無条件に成立しなければならないので、テスト仕様に対して新しいテストを作成するのではなく、他の要件のためのテストケースすべてに対して追加のチェックを抱き合わせることだけで実装されます。

ヘッダファイルと関数形式マクロの扱い

C 言語にはテストを複雑にするもう一つの要素があります。C 標準ライブラリのすべての関数がコンパイル済みのバイナリとしてだけで提供されているわけではないということです。ヘッダファイルのソースに含まれる情報に大きく依存するものも多くあります。

型やグローバル変数、マクロなどが定義されるこれらのヘッダファイルも（コンパイル済みの）関数と同じくライブラリの一部であり、通常の関数とマクロ（関数形式マクロ）の両方として提供されているライブラリ関数が多くあるからです。速度や効率を重視する場合に、マクロで実装されたものを使用することが一般的です。SuperGuard は、通常の関数と関数形式マクロの両者をテストします。

関数形式マクロは、前もってコンパイルされずバイナリでは提供されません。アプリケーションのソースコードと一緒に SDK のコンパイラによってコンパイルされます。したがって、関数形式マクロはヘッダファイル内の他の要素とともに、セーフティクリティカルアプリケーションの個別のユースケースに対して検証することが重要になるのです。C++ のライブラリでは、ヘッダにのみ存在する型汎用のテンプレートを使用することがあり、さらに高いレベルでマクロが使用されることになります。



Section	Title	Requirement	Description	Test Specification	Test	Result
	function			returned value corresponds to the destination buffer.	tspr3060.c	
C99:7.21.2.3	The strcpy function	REQ-C99:7.21.2.3-return	The strcpy function returns the value of s1.	Call the strcpy() function under two different scenarios (regular copy and zero length copy) and verify the returned value corresponds to the destination buffer in both cases.	4/11/2/3/t2.c	PASSED
C99:7.21.2.4	The strncpy function	DECL-C99:7.21.2.4	char *strncpy(char *s1, const char *s2, size_t n);			
C99:7.21.2.4	The strncpy function	UNDEF-C99:7.21.2.4	Behavior is undefined if the objects s1 and s2 overlap.			
C99:7.21.2.4	The strncpy function	UNDEF-C99:7.21.2.4	Behavior is undefined if the array s1 has a size less than n.			
C99:7.21.2.4	The strncpy function	UNDEF-C99:7.21.2.4	Behavior is undefined if s2 is not a string and has a size less than n.			
C99:7.21.2.4	The strncpy function	REQ-C99:7.21.2.4-copystring	If s2 points to a string with a length 'l2' that is less than n, strncpy() shall copy l2 characters from the array s2 into the array s1.	Call the strncpy() function with different values for the n parameter (including n==0) equal to and larger than the length of the origin string. Verify that the origin string is copied into the target array up to the terminating null character.	4/11/2/4/t1.c	PASSED
C99:7.21.2.4	The strncpy function	REQ-C99:7.21.2.4-copyn	If s2 does not point to a string with length less than n, strncpy() shall copy the first n characters from the array s2 into the array s1.	Call the strncpy() function with different values for the n parameter (including n==0), so that the first n characters of the origin array do not contain the null character. Verify that the first n characters of the origin string are copied into the target array.	4/11/2/4/t1.c	PASSED
C99:7.21.2.4	The strncpy function	REQ-C99:7.21.2.4-nomore	strncpy() shall not write into the target array s1 beyond the first n characters.	For all test cases, verify that the character with index n in the target array s1 is not modified. If that fits with the test, verify that also no characters beyond n are modified.	4/11/2/4/t1.c	PASSED
C99:7.21.2.4	The strncpy function	REQ-C99:7.21.2.4-nochange	strncpy() shall not modify array s2.	For all test case, verify that the array s2 is not modified.	4/11/2/4/t1.c	PASSED
C99:7.21.2.4	The strncpy function	REQ-C99:7.21.2.4-padding	If s2 points to a string with a length that is less than n, strncpy() shall append null characters after the copied characters in array s1 until n characters in total have been written.	Call the strncpy() function with strings that are shorter than the value passed as the n parameter. Verify that null characters are appended after the string copy in s1 until n characters are written.	4/11/2/4/t1.c	PASSED
C99:7.21.2.4	The strncpy function	REQ-C99:7.21.2.4-return	strncpy() shall return the value of s1.	Call the strncpy() function with different strings and values for the size (parameter n), and verify the returned value corresponds to the destination buffer in each case.	4/11/2/4/t1.c	PASSED

ISO 26262 Part 8 Clause 12 での要件ベーステストの位置づけ

これらの要件ベースのテスト手法により SuperGuard は、ソフトウェアコンポーネントが認定済みであると判断されるための ISO 26262 Part 8 Clause 12.4.1 に定められた以下の重要な要素が有効であることを保証します。

12.4.1 a) ソフトウェアコンポーネントの仕様

C/C++ 標準ライブラリの仕様は公開されている ISO 規格に基づいており、SuperGuard はその規格における動作の記述に対して明確な機能要件を追加しています。

12.4.1 b) ソフトウェアコンポーネントが要件に準拠することを示す証拠

SuperGuard は、ライブラリ仕様における動作記述から抽出された機能要件をテスト仕様とテスト設計に変換し、得られたテストを包括的な実行可能テストスイートに組み立てることで、C 標準ライブラリの実装が、抽出された要件、そして動作記述に準拠している証拠を提供します。

12.4.1 c) ソフトウェアコンポーネントが使用目的に適していることを示す証拠

SuperGuard を使用して C 標準ライブラリの実装を検証すると、すべてのテストの PASS/FAIL 状況の詳細なレポートが作成され、仕様による動作記述から抽出された機能要件への完全なトレーサビリティが得られます。

SuperGuard は、上記 12.4.1 b で参照される ISO 26262 Part 8 Clause 12 の段落 12.4.2.2、12.4.2.3、12.4.2.4 の要件も満たしています。

12.4.2.2 項はソフトウェアコンポーネントの検証が要件ベースのテストと専用のテストスイートを使用して行うことができることを示しており、SuperGuard によってそれが提供されます。また、検証では「*正常時の動作条件と故障時の挙動の両方をカバーし*」、「*そのソフトウェアコンポーネントに割り当てられた安全要件に違反し得る既知のエラーがないことを表示し*」なければならないとも述べています。SuperGuard は、ISO C 言語規格に記載の定義されたエラー動作すべての機能要件をテストします。

12.4.2.3 項は ASIL-D アプリケーションでソフトウェアコンポーネントを使用するための追加要件を定義しており、テストケースの完全性を評価するために構造カバレッジを測定しなければならないと述べています。

12.4.2.4 項は、ソフトウェアコンポーネントの変更されていない実装にのみ Clause 12 に詳述の検証プロセスが適用できると述べています。ライブラリ関数を独自の実装で置き換える開発者もありますが、その変更がなされる関数の数は通常少ないものです。C 標準ライブラリはモジュール性が高く、すべての関数の実装はライブラリの残りの部分から独立している

とみなせるので、ライブラリ関数の大部分はPart 8 Clause 12の規定の下で検証することができます。これにより、アプリケーション開発者はPart 6で説明されている規定を使用して変更された関数だけを認定することに集中できます。

コードカバレッジ解析

SuperGuardテストスイートの構築においては、長く使われ普及しているオープンソースのC標準ライブラリ実装が12.4.2.3項のASIL D要件を満たすようなコードカバレッジの達成に特に重点を置きました。そのライブラリの多くの関数に対しSuperGuardは、高いMC/DCカバレッジに加えて、100%のカバレッジを達成しています。SuperGuardのカバレッジが低い領域は一般に処理系依存の動作に関連したもので、言語仕様から要件を導出することができないものです。

ライブラリの実装は個々に異なりますが、最終的にはどれについても同様のケース分析処理が必要です。つまり、SuperGuardの高いコードカバレッジは、すべてのC標準ライブラリ実装のコードカバレッジに役立つのです。

例外ケース

SuperGuardのテストで例外ケースを処理する方法は二つあります。一つ目は、例外的な入力に起因する定義された動作に関連します。たとえば、関数**sqrt()**に負の数を渡すと、値**NaN**を返さなければなりません(ISO/IEC 60559算術演算を想定しています)。これは例外ケースですが、それも含めてこの関数の動作は完全に定義されており、他の動作と同様に検証できます。

二つ目はコンパイラで検証できる要件に関連するものです。たとえば、関数に**void**の戻り値型が必要な場合、テストでは戻り値を使用してコンパイラエラーが発生することを期待できます。このようなテストはSuperGuardで*x-テスト*と呼ばれるもので、それらのテストのファイル名は*x*で始まります。X-テストに対して、コンパイラがコンパイル時にエラーメッセージを発生する場合はX-Tests PASSで、コンパイル時にエラーが発生しない場合はFAILとなります。X-テストは実行されません。

まとめと結論

セーフティクリティカルアプリケーションでは、開発プロセスやツールチェーン、そしてアプリケーションコードによって傷害、人命の損失、必須なサービスの中断等のリスクを生じないように、ソフトウェア開発者があらゆることを行う必要があります。コンパイラや標準ライブラリなど、サードパーティ製/COTS ツールやコンポーネントを使用する場合、開発者は、それらのツールやコンポーネントにエラーがないと仮定してはならず、また、事前の認定でエラーなしであることが示されていると仮定してもなりません。アプリケーションで使用されているものとまったく同じ開発環境かつ、まったく同じユースケースシナリオで実行される場合にのみ、認定は真に有効になります。

SuperGuard C Library Safety Qualification Suite では、Solid Sands 社の SuperTest コンパイラテスト・検証ソリューションに含まれるものと同じライブラリテストスイートを使用することで、ISO 26262 など機能安全規格で推奨される要件ベースのテスト結果を C 標準ライブラリに関連づけるトレーサビリティを追加しています。ISO C 標準ライブラリの機能仕様を明確に定義された要件に分解し、その要件をチェックするための適切なテスト仕様を作成し、そして ISO 26262 の推奨に従って実装することで、完全なトレーサビリティを得ることができます。さらに、ソフトウェア開発者は、同じ開発環境、同じユースケースの条件下、およびアプリケーションで使用するものと同じターゲットハードウェアによって、100% に近い構造コードカバレッジでこれらのテストを実施できます。SuperGuard は ISO 26262 認証機関のニーズに合わせた包括的な認定レポートを作成することで、セーフティクリティカルアプリケーションで使用されるライブラリコンポーネントの完全性を実証することの負担を減少させます。

FUJI SETSUBI

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