wolfSSL ポーティングガイド

（この資料は **wolfSSLPorting Guide** の日本語翻訳版です）

目的

このガイドでは、wolfSSL 組込み向け軽量 SSL/TLS ライブラリを新しい組み込みプラットフォーム、オペレーティングシステム、または転送媒体（TCP / IP、Bluetooth など）に移植する開発者、エンジニアのためのリファレンスを提供します。 それらのために、通常、wolfSSL を移植するときに変更が必要とされる wolfSSL コードベース内の領域を呼び出します。それは "ガイド"と見なされる進化する仕事であると考えています。文書に説明や説明を随時追加いたしますので、不足しているものなどお気づきの際はお知らせください。

対象

このガイドでは、デフォルトでサポートされていない新しいプラットフォームや環境に wolfSSL 組込み向け軽量 SSL/TLS ライブラリを移植する開発者やエンジニアを対象としています。

目次

1 はじめに

組み込みプラットフォーム上で wolfSSL を実行するには、いくつかのステップが必要です。これらのステップのいくつかは、wolfSSL Manual（非標準環境でのビルド）のセクション 2.4 で概説されています。

wolfSSL マニュアルの第 2 章の手順とは別に、特定のプラットフォームに対応するために移植や修正が必要なコードがいくつかあります。 wolfSSL は、これらの分野の多くを抽象化して、wolfSSL を新しいプラットフォームに移植するのはできるだけ簡単にしようとしています。

./wolfssl/ctaocrypt/settings.h ファイルに、いくつかの異なるオペレーティングシステム、TCP / IP スタック、およびチップセット（MBED、FREESCALE_MQX、MICROCHIP_PIC32、MICRIUM、EBSNET 等）向けの定義があります。新しい定義は wolfSSL の新しい移植が完了したときに settings.h ファイルに追加されますが、それらは通常 wolfSSL を変更する必要があります。これによって、機能を有効/無効にしたり、ビルド設定をカスタマイズしたりする簡単な方法が提供されます。
wolfSSL のポートを新しいプラットフォームにするときには、このファイルの新しいカスタム定義に自由に追加してください。わたしどもは、ユーザーが wolfSSL のポートに貢献してメインのオープンソースコードブランチに戻していただけるようお勧めしています。これにより、wolfSSL を最新の状態に保つことができ、wolfSSL プロジェクトが改善され、前進する際に、さまざまなポートを最新に維持することができます。

wolfSSL は、直接メール（info@wolfssl.com）か、GitHub のプルリクエスト（https://github.com/wolfssl/wolfssl）を介してパッチやコードの変更の提出をお勧めしています。

2 wolfSSL のポーティング

2.1 データ型

Q:このセクションが必要なのはどういう場合？

A: ポーティング対象のプラットフォームの正しいデータ型のサイズを設定するのは常に重要です。

wolfSSL は、64 ビットタイプが利用可能の場合、スピードに恩恵を受けます。プラットフォーム上の sizeof（long）と sizeof（long long）の結果と一致するように SIZEOF_LONG または SIZEOF_LONG_LONG を設定します。これは、settings.h ファイルのカスタム定義に追加することができます。たとえば、MY_NEW_PLATFORM のサンプル定義の settings.h で次のように指定します。

```
#ifdef MY_NEW_PLATFORM
#define SIZEOF_LONG 4
#define SIZEOF_LONG_LONG 8
...
#endif
```


2.1 データ型

Q:このセクションが必要なのはどういう場合？

A: ポーティング対象のプラットフォームの正しいデータ型のサイズを設定するのは常に重要です。

wolfSSL は、64 ビットタイプが利用可能の場合、スピードに恩恵を受けます。プラットフォーム上の sizeof（long）と sizeof（long long）の結果と一致するように SIZEOF_LONG または SIZEOF_LONG_LONG を設定します。これは、settings.h ファイルのカスタム定義に追加することができます。たとえば、MY_NEW_PLATFORM のサンプル定義の settings.h で次のように指定します。


```
#ifdef MY_NEW_PLATFORM
```

#define SIZEOF_LONG 4

#define SIZEOF_LONG_LONG 8

...

#endif

2.2 エンディアン

Q:このセクションが必要なのはどういう場合？

A: プラットフォームがビッグエンディアンの場合です

あなたのプラットフォームはビッグエンディアン、リトルエンディアン、どちらですか？ wolfSSL はデフォルトではリトルエンディアンシステムです。システムがビッグエンディアンの場合は、wolfSSL をビルドするときに BIG_ENDIAN_ORDER を定義してください。これを settings.h で設定する例：

#ifdef MY_NEW_PLATFORM

...

#define BIG_ENDIAN_ORDER

...

#endif

2.3 WRITEV

Q:このセクションが必要なのはどういう場合？

A: <sys/uio.h> が提供されていない場合です

デフォルトでは、wolfSSL API はアプリケーションに対して writev 関数のセマンティクスをシミュレートする wolfSSL_writev（）を提供します。使用可能な<sys / uio.h>ヘッダーを持たないシステムでは、この機能を除外するために NO_WRITEV を定義してください。

2.4 (ネットワーク) 入出力

Q：どういう場合このセクションが必要ですか？
A：BSD スタイルのソケット API が使用できない場合。また、特別なポート層または TCP / IP スタックを使用したい場合、静的バッファーのみを使用したい場合です。

wolfSSL はデフォルトでは BSD スタイルのソケットインターフェイスを使用します。トランスポート層が BSD ソケットインタフェースを提供する場合、カスタムヘッダが必要な場合を除いて、wolfSSL はそのままの状態で統合する必要があります。

wolfSSL は、ユーザーがシステムに wolfSSL の I / O 機能を合わせることが可能となるようにカスタム I / O 抽象レイヤーを提供しています。詳細は、wolfSSL マニュアルのセクション 5.1.2 にあります。

単に、ビルド時オプションに WOLFSSL_USER_IO を指定して、独自の I / O コールバック関数を、テンプレートとして wolfSSL のデフォルト EmbedSend（）と EmbedReceive（）を参照して記述してください。これら 2 つの関数は./src/io.c にあります。

wolfSSL は、入出力時に動的バッファを使用します。デフォルトは 0 バイトです。バッファよりサイズが大きい入力レコードが受信された場合は、動的バッファを使用して要求を処理してから解放します。

ダイナミックメモリを使用せず、大きな 16kB スタティックバッファを使用したい場合は、LARGE_STATIC_BUFFERS オプションを指定します。

ダイナミックバッファが使用されている時は、ユーザがバッファサイズより大きい wolfSSL_write（）を要求すると、最大 MAX_RECORD_SIZE までの動的ブロックがデータを送信するために使用されます。 RECORD_SIZE で定義されているように、バッファー・サイズのデータを最大でのみ送信したい場合は、STATIC_CHUNKS_ONLY を定義します。この定義を使用する場合、RECORD_SIZE のデフォルトは 128 バイトです。

2.5 ファイルシステム

Q：どういう場合このセクションが必要ですか？
A： 使用可能なファイルシステムがない場合、標準のファイルシステム機能が使用できない場合、または、カスタムファイルシステムを使用する場合です。

wolfSSL は鍵と証明書を SSL セッションまたはコンテキストにロードするためにファイルシステムを使用します。 wolfSSL では、これらをメモリバッファからロードすることもできます。メモリバッファだけを使用する場合、ファイルシステムは必要ありません。

ライブラリをビルドするときに NO_FILESYSTEM を定義することにより、ファイルシステムの使用を無効にすることができます。この場合、ファイルではなくメモリバッファから証明書と鍵をロードする必要があります。これを settings.h で設定する例：

```
#ifdef MY_NEW_PLATFORM
...
#define NO_FILESYSTEM
...
#endif
```

テスト用の鍵と証明書バッファーは、./wolfssl/certs_test.h ヘッダーファイルにあります。これらは、これらの証明書と./certs ディレクトリにある証明書と同じものです。

certs_test.h ヘッダーファイルは、必要に応じて./gencertbuf.pl スクリプトを使用して更新できます。 gencertbuf.pl には、fileList_1024 と fileList_2048 という 2 つの配列があります。鍵のサイズに応じて、それぞれの配列に追加の証明書または鍵を追加することができ、DER 形式でなければなりません。上記の配列は、目的のバッファ名を持つ証明書/鍵ファイルの場所にマップされます。 gencertbuf.pl を変更した後、wolfSSL ルートディレクトリからそれを実行すると、./wolfssl/certs_test.h の証明書と鍵バッファが更新されます：

```
./gencertbuf.pl
```

デフォルト以外のファイルシステムを使用したい場合、ファイルシステム抽象化レイヤーは./src/ssl.c にあります。ここでは、EBSNET、FREESCALE_MQX、MICRIUM などのさまざまなプラットフォームのファイルシステムが表示されています。 XFILE、XFOPEN、XFSEEK などでファイルシステム関数を定義できるように、必要に応じてプラットフォームにカスタム定義を追加できます。たとえば、Micrium のμC/ OS（MICRIUM）の ssl.c のファイルシステム層は次のとおりです。

```
#elif defined(MICRIUM)
#include <fs.h>
#define XFILE          FS_FILE*
#define XFOPEN         fs_fopen
#define XFSEEK         fs_fseek
#define XFTELL         fs_ftell
#define XREWIND        fs_rewind
#define XFREAD         fs_fread
#define XFCLOSE        fs_fclose
#define XSEEK_END    FS_SEEK_END
#define XBADFILE     NULL
```

2.6 スレッド

Q：どういう場合このセクションが必要ですか？
A：マルチスレッド環境で wolfSSL を使用したい場合、またはシングルスレッドモードでコンパイルしたい場合です。

wolfSSL がシングルスレッド環境でのみ使用される場合、wolfSSL をコンパイルするときに SINGLE_THREADED を定義して wolfSSL の排他制御を無効にすることができます。これにより、wolfSSL 排他制御層を移植する必要がなくなります。

wolfSSL をマルチスレッド環境で使用する必要がある場合は、wolfSSL 排他制御層を新しい環境に移植する必要があります。排他制御層は、./wolfssl/ctaocrypt/wc_port.h と./ctaocrypt/src/wc_port.c にあります。 wolfSSL_Mutex は、wc_port.c の port.h の新しいシステムと排他制御関数（InitMutex、FreeMutex など）に定義する必要があります。

wc_port.h および wc_port.c で、いくつかの既存のプラットフォーム（EBSNET、FREESCALE_MQX など）を例として検索します。

2.7 乱数シード

Q：どういう場合このセクションが必要ですか？

A：/ dev / random または/ dev / urandom のいずれかが利用できないか、RNG ハードウェアを統合したい場合です。

デフォルトでは、wolfSSL は/dev/urandom または/dev/random を使用して RNG シードを生成します。 NO_DEV_RANDOM の定義は、デフォルトの wc_GenerateSeed（）関数を無効にするときにビルド時に指定します。これが指定されている場合は、ターゲットプラットフォームに固有の./wolfcrypt/src/random.c にカスタム wc_GenerateSeed（）関数を記述する必要があります。これにより、ハードウェアベースのランダムエントロピーソースがあれば、wolfSSL の PRNG にシードすることができます。

wc_GenerateSeed 関数をどのように記述する必要があるかの例については、wolfSSL の既存の wc_GenerateSeed 関数の実装を./wolfcrypt/src/random.c で参照してください。

2.8 メモリー

Q：どういう場合このセクションが必要ですか？

A：標準のメモリ関数を使用できない場合、またはオプションの数学ライブラリ間のメモリ使用量の違いに関心があるような場合です。

wolfSSL は、デフォルトでは malloc（）と free（）を使用しています。通常の整数演算ライブラリを使用する場合、wolfCrypt は realloc（）も使用します。

デフォルトでは、wolfSSL/wolfCrypt は、通常の整数ライブラリを使用します。これは、かなりの動的メモリを使用します。 wolfSSL を構築する場合、FastMath ライブラリのほうを有効にすることができます。こちらは通常速度も速く、暗号操作（すべてのスタック上）には動的メモリーを使用しません。 Fastmath を使う場合、wolfSSL は realloc（）の実装を必要としません。 wolfSSL の SSL 層はそのほかにもいくつかの処理で動的メモリを使用しているので、malloc（）と free（）は依然として必要です。

通常の整数演算ライブラリと FastMath ライブラリ間のリソース使用量（スタック/ヒープ）の比較のドキュメントを参照ご希望のかたはお知らせください。

FastMath を有効にするには、USE_FAST_MATH を定義し ./wolfcrypt/src/integer.c ではなく ./wolfcrypt/src/tfm.c を使用します。 fastmath を使用するときはスタックメモリが大きいので、TFM_TIMING_RESISTANT も定義することをお勧めします。

標準の malloc（）、free（）を、および realloc の（）関数が利用できない場合、XMALLOC_USER を定義します。これによりターゲット環境依存のカスタムフックを ./wolfssl/wolfcrypt/types.h 内に定義することができます。

XMALLOC_USER の使用方法の詳細については、wolfSSL マニュアルのセクション 5.1.1.1 を参照してください。

2.9 時計

Q：どういう場合にこのセクションが必要ですか？

A：標準時間関数（time（）、gmtime（））が利用できない場合、またはカスタムクロックティック関数を指定する必要がある場合です。

デフォルトでは、wolfSSL は./wolfcrypt/src/asn.c で指定されているように、time（）、gmtime（）、および ValidateDate（）を使用します。これらは、XTIME、XGMTIME、XVALIDATE_DATE に抽象化されています。標準時刻関数、および time.h が使用できない場合、ユーザーは USER_TIME を定義できます。 USER_TIME を定義した後、ユーザーは独自の XTIME、XGMTIME、および XVALIDATE_DATE 関数を定義できます。

wolfSSL は、クロックティック機能のデフォルトで time（0）を使用します。これは、LowResTimer（）関数の内部の./src/internal.c にあります。

time（0）が望ましくない場合には、USER_TICKS を定義することでユーザー独自の clock tick 関数を定義することができます。カスタム関数は秒の精度が必要ですが、EPOCH と相関させる必要はありません。 ./src/internal.c の LowResTimer（）関数を参照してください。

2.10 C 標準ライブラリ

Q：どういう場合このセクションが必要ですか？

A：C 標準ライブラリがない場合、またはカスタムライブラリを使用する場合です。

wolfSSL は、C 標準ライブラリを使用しなくても、開発者がより高いレベルの移植性と柔軟性を得ることができます。そのようなとき、ユーザーは C 標準のものの代わりに使用したい機能をマップする必要があります。

上のセクション 2.8 では、メモリ機能について説明しました。メモリ関数の抽象化に加えて、wolfSSL は文字列関数と数学関数も抽象化します。それぞれの関数は抽象化される関数の名前に対応して X<FUNC>の形で定義されます。

詳細については、wolfSSL マニュアルのセクション 5.1 をお読みください。

2.11 ロギング

Q：どういう場合このセクションが必要ですか？

A：デバッグメッセージを有効にしたいが、stderr は使用できません。

デフォルトでは、wolfSSL は stderr を介してデバッグ出力を提供します。デバッグメッセージを有効にするには、wolfSSL を DEBUG_WOLFSSL でコンパイルし、wolfSSL_Debugging_ON（）をアプリケーションコードから呼び出す必要があります。wolfSSL_Debugging_OFF（）は、アプリケーション層が wolfSSL デバッグメッセージをオフにするために使用できます。

stdder が利用できない環境や、デバッグメッセージを別の出力ストリームや別の形式で出力したい場合、wolfSSL ではアプリケーションはロギングコールバックに登録できます。

詳細については、wolfSSL マニュアルの第 8.1 節をお読みください。

2.12 公開鍵演算

Q：どういう場合このセクションが必要ですか？

A：wolfSSL で独自の公開鍵実装を使用したいとします。

wolfSSL を使用すると、SSL / TLS 層が公開鍵操作を行う必要があるときに呼び出される独自の公開鍵コールバックをユーザーが書くことができます。ユーザーはオプションで 6 つの関数を定義できます。

ECC 符号コールバック
ECC 検証コールバック
RSA 署名コールバック
RSA 検証コールバック
RSA 暗号化コールバック
RSA 復号化コールバック
詳細は、wolfSSL マニュアルのセクション 6.4 を参照してください。

2.13 アトミックレコード層処理

Q：どういう場合このセクションが必要ですか？
A：TLS レコード層の独自の処理、特に MAC /暗号化と解読/検証操作を行いたい場合です。

デフォルトでは、wolfSSL は、暗号化ライブラリ wolfCrypt を使用して、ユーザーの TLS レコード層処理を処理します。 wolfSSL は、MAC /暗号化をより詳細に制御し、SSL / TLS 接続を復号/検証したい場合、アトミックレコード処理コールバックを使用します。

ユーザーは 2 つの関数を定義する必要があります：

MAC /暗号化コールバック関数
コールバック関数の復号化/検証

詳細は、wolfSSL マニュアルのセクション 6.3 を参照してください。

2.14 機能

Q：どういう場合このセクションが必要ですか？
A：機能を無効にする場合。

適切な定義を使用して wolfSSL をビルドするとき、機能を無効にすることができます。利用可能な定義のリストについては、wolfSSL Manual の第 2 章を参照してください。

3. 次のステップ

3.1. wolfCrypt テストアプリケーション

wolfSSL をターゲットプラットフォーム上に適切に構築した後、wolfCrypt テストアプリケーションには良い次のステップがあります。このアプリケーションをターゲットシステムで実行すると、NIST テストベクタを使用して、すべての暗号アルゴリズムが正しく動作していることが確認されます。

この手順をスキップして SSL 接続の確立に直接進むと、基になる暗号化操作が失敗したために発生した問題をデバッグするのがより困難になります。

wolfCrypt テストアプリケーションは、./wolfcrypt/test/test.c にあります。埋め込みアプリケーションに独自の main（）関数がある場合、./wolfcrypt/test/test.c をコンパイルするときに NO_MAIN_DRIVER を定義することができます。これにより、アプリケーションの main（）はそれぞれの暗号/アルゴリズムテストを個別に呼び出すことができます。

組み込みデバイスに wolfCrypt テストアプリケーション全体を実行するのに十分なリソースがない場合、個々のテストを test.c から抜き出して個別にコンパイルすることができます。 test.c から隔離された暗号テストを抽出する際に、ビルドに含まれる特定のテストケースに必要な正しいヘッダファイルがあることを確認してください。

4. サポート

一般的なサポートに関する質問は、電子メール、サポートフォーラムを介して wolfSSL に直接送信することができます。

ウェブサイト：http://www.wolfssl.jp

サポートメール：support@wolfssl.com

フォーラム：http://www.wolfssl.com/forums

wolfSSL は、お客様が新しい環境に wolfSSL を移植できるためのサポートパッケージとコンサルティングサービスを提供しています。

一般的なお問い合わせ：info@wolfssl.jp