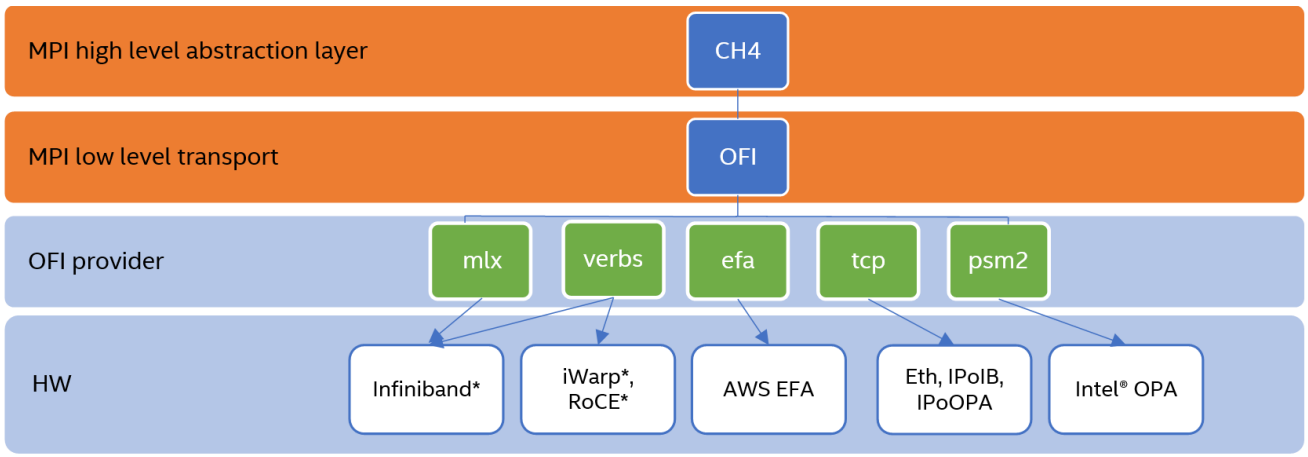
インテル® MPI ライブラリー 2019 の libfabric

この記事は、インテル® デベロッパー・ゾーンで公開されている「Intel® MPI Library 2019 Over Libfabric*」の日本語参考訳です。

概要

バージョン 2019 でインテル® MPI ライブラリーは、Open Fabrics Alliance* (OFA) フレームワークから Open Fabrics Interfaces* (OFI) に変更され、libfabric がサポートされるようになりました。DAPL、TMI、および OFA ファブリックは廃止される予定です。インテル® MPI ライブラリーのパッケージには、`mpivars.sh/csh/bat` スクリプトを実行する際に使用される libfabric ライブラリーが含まれています。

次の図は、インテル® MPI ライブラリーのソフトウェア・スタックを示したものです。



libfabric とは

libfabric は、ハイパフォーマンス・ネットワーク向けの低レベルの通信抽象化レイヤーです。ミドルウェアやアプリケーションからトランスポートやハードウェアの実装の詳細をほとんど隠し、多様なファブリック間でハイパフォーマンスな移植性を提供します。詳細は、プロジェクトの概要と詳細な API ドキュメントを含む OFI ウェブサイト (英語) を参照してください。

インテル® MPI ライブラリーの libfabric の使用

デフォルトでは、Linux* または Windows* で `mpivars.sh/csh/bat` (インテル® oneAPI の場合は `vars.sh/bat`) は、環境変数にインテル® MPI ライブラリーに同梱の libfabric バージョンを設定します。これを無効にするには、`I_MPI_OFI_LIBRARY_INTERNAL` 環境変数または `-ofi_internal` (デフォルトでは `ofi_internal=1`) を使用します。

```
source ./mpivars.sh -ofi_internal=1
I_MPI_DEBUG=4 mpirun -n 1 IMB-MPI1 barrier
```

出力:
```
[0] MPI startup(): libfabric version: 1.9.0a1-impi
[0] MPI startup(): libfabric provider: mlx
```

```
source ./mpivars.sh -ofi_internal=0
I_MPI_DEBUG=4 mpirun -n 1 IMB-MPI1 barrier
```

出力:
```
[0] MPI startup(): libfabric version: 1.10.0a1
[0] MPI startup(): libfabric provider: verbs;ofi_rxm
```

注: `I_MPI_DEBUG=4` で libfabric バージョンを表示できます。

一般的な OFI コントロール

プロバイダーの選択

libfabric ライブラリーから OFI プロバイダーを選択するには、ロードする OFI プロバイダー名を定義する `FI_PROVIDER` 環境変数を使用します。

```
export FI_PROVIDER=<name>
```
ここで、*<name>* はロードする OFI プロバイダーです。

IMPI 環境変数で特定の OFI プロバイダーを指定できます。

```
export I_MPI_OFI_PROVIDER=<name>
```

注: `I_MPI_OFI_PROVIDER` よりも `FI_PROVIDER` が推奨されます。

ランタイム・パラメーター

RxM、MLX、および EFA プロバイダーの `FI_UNIVERSE_SIZE=<number>` 環境変数は、分散型 OFI アプリケーションで使用される最大プロセス数を定義します。プロバイダーは、この値を使用してリソースの割り当てを最適化します。

ログ・インターフェイス

`FI_LOG_LEVEL=<level>` は、出力されるログデータ量を制御します。次のログレベルが定義されています。

- **Warn**: は、データ量が最も少なく、エラーや警告をレポートします。
- **Trace**: は、データ量がやや多く、プログラム実行のトレースに役立つ詳細ではない情報を出力します。
- **Info**: は、データ量が多く、詳細な情報を出力します。
- **Debug**: は、データ量が多く、アプリケーションのパフォーマンスに影響する可能性があります。Debug 出力は、ライブラリーがデバッグを有効にしてコンパイルされている場合にのみ利用できます。

詳細は、fi_fabric(7) (英語) を参照してください。

Linux* オペレーティング・システム

インテル® MPI ライブラリーでサポートされる OFI プロバイダー

このセクションでは、インテル® MPI ライブラリーでサポートされる OFI プロバイダーについて説明します。プロバイダーごとに、インテル® MPI ライブラリーに関連する説明とパラメーターがあり、オプションで必要条件や制限事項があります。

MLX

説明

MLX プロバイダーは、バージョン 2019 Update 5 で内部 libfabric のバイナリーとして追加されました。MLX プロバイダーは、Mellanox* InfiniBand* ハードウェアで利用可能な UCX 上で動作します。

InfiniBand* での MLX の使用に関する詳細は、「InfiniBand* でインテル® MPI ライブラリーのパフォーマンスと安定性を向上する」(英語) を参照してください。

一般的なコントロール・パラメーター

名前	説明
`FI_MLX_INJECT_LIMIT`	tinject/inject メッセージの最大サイズを設定します。
`FI_MLX_ENABLE_SPAWN`	動的プロセスのサポートを有効にします。
`FI_MLX_TLS`	MLX プロバイダーが利用可能なトランスポートを指定します。 **注**: ランク数やその他のパラメーターに応じて、最適なトランスポートが選択されます。どのトランスポートを FI_MLX_TLS に設定すべきかは、「InfiniBand* でインテル® MPI ライブラリーのパフォーマンスと安定性を向上する」(英語) を参照してください。

必要条件

UCX 1.4 以降 (MOFED の一部またはスタンドアロン・パッケージとして利用可能) がインストールされている必要があります。

制限事項

MLX プロバイダーは、インテル® MPI ライブラリーの MPI connect/accept プリミティブとマルチ EP モードをサポートしません。UCX API の最小バージョンは 1.4 です。

PSM2

説明

PSM2 プロバイダーは、インテル® Omni-Path ファブリックでサポートされる PSM 2.x インターフェイス上で動作します。

一般的なコントロール・パラメーター

名前	説明
`FI_PSM2_INJECT_SIZE`	`fi_inject` および `fi_tinject` 呼び出しの最大メッセージサイズを定義します。
`FI_PSM2_LAZY_CONN`	OFI エンドポイントのベースとなる PSM2 エンドポイント間で確立される接続モードを制御します。 • **0 (eager 接続モード):** アドレスがアドレスベクトルに挿入されるときに接続が確立されます。 • **1 (lazy 接続モード):** アドレスが初めて通信で使用されるときに接続が確立されます。 **注:** lazy 接続モードは、データ・パス・オーバーヘッドの増加と引き換えに、大規模システムの起動時間を短縮します。

必要条件

追加で、インテル® OPA-IFS (英語) ソフトウェア・スタックの一部である `libpsm2` ライブラリーが必要です。

詳細は、fi_psm2(7) (英語) を参照してください。

TCP

説明

TCP プロバイダーは、libfabric API を実装するため TCP ソケットをサポートする任意のシステムで使用できるインテル® MPI ライブラリーの汎用プロバイダーです。このプロバイダーでは、特定の高速インターコネクトを持たないクラウド環境 (例えば、GCP、イーサネット接続の Azure*、AWS*インスタンス) や IPoIB を使用して、通常のイーサネット上でインテル® MPI ライブラリー・アプリケーションを実行できます。

一般的なコントロール・パラメーター

名前	説明
`FI_TCP_IFACE`	特定のネットワーク・インターフェイスを指定します。
`FI_TCP_PORT_LOW_RANGE` `FI_TCP_PORT_HIGH_RANGE`	TCP プロバイダーがパッシブ・エンドポイントの作成に使用するポートの範囲を設定します。これは、ファイアウォールで TCP 接続用のポート範囲が指定されている場合に便利です。

必要条件

必要条件はありません。

詳細は、fi_tcp(7) (英語) を参照してください。

EFA

説明

EFA プロバイダーは、AWS* EFA (Elastic Fabric Adapter) ハードウェア上の AWS* EC2* インスタンスで OFI を使用するアプリケーションを実行できるようにします。詳細は、Elastic Fabric Adapter ページを参照してください。

EFA は、ユーザースペースからの直接ハードウェア・アクセス (OS バイパス) で、信頼性の高い/低いデータグラムの送受信を提供します。このプロバイダーは、インテル® MPI ライブラリーの新しい Scalable (順不同) Reliable Datagram (SRD) プロトコルで *FI_EP_RDM* エンドポイント・タイプをサポートします。SRD は、ほかの Reliable Datagram (RD) 実装と比較して、通常、信頼性の高いデータグラムとより完全なエラー処理をサポートします。EFA プロバイダーは、*FI_EP_RDM* エンドポイントを介してアプリケーションに送信後の順序を保証するため、順不同パケットのセグメンテーションと再構築を提供します。

一般的なコントロール・パラメーター

名前	説明
FI_EFA_EFA_CQ_READ_SIZE	プログレスエンジンの 1 回の反復において 1 つのループで読み取る EFA 完了エントリーの数を設定します。デフォルトは 50 です。
FI_EFA_MR_CACHE_ENABLE	max_memcpy_size よりも大きい IOV に対して、バウンスバッファーの代わりに、メモリー・レジストレーション・キャッシュとインライン・レジストレーションを使用します。デフォルトは true です。無効にすると、バウンスバッファーのみ使用します。
FI_EFA_MR_CACHE_MERGE_REGIONS	オーバーラップおよび隣接するメモリー・レジストレーション領域をマージします。デフォルトは true です。
FI_EFA_MR_MAX_CACHED_COUNT	一度にキャッシュ可能なメモリー・レジストレーションの最大数を設定します。
FI_EFA_RECVWIN_SIZE	スライディング受信ウィンドウのサイズを定義します。デフォルトは 16384 です。 **注:** 送信完了を待たずに大量の送信操作を行った場合、パフォーマンスに影響する可能性があります。
FI_EFA_TX_MIN_CREDITS	送信者が受信者に要求する最小クレジット数を定義します。デフォルトは 32 です。

必要条件

EFA ソフトウェア・コンポーネントがインストールされている必要があります。詳細は、「AWS* EC2* ユーザーガイド」を参照してください。

制限事項

FI_UNIVERSE_SIZE を、8192 ランク (または 227 の C5n インスタンス) を超える実行で想定されるランク数に設定します。

詳細は、fi_efa(7) (英語) を参照してください。

Verbs

説明

Verbs プロバイダーは、OFI を使用するアプリケーションを任意の Verbs ハードウェア (InfiniBand*、iWarp*/RoCE*) 上で実行できるようにします。このプロバイダーは、ネットワーク転送に Linux* Verbs API を使用し、OFI 呼び出しを適切な Verbs API 呼び出しに変換します。Verbs プロバイダーは、RxM ユーティリティー・プロバイダーを使用していくつかの特定の環境を実装します。

一般的なコントロール・パラメーター

名前	説明
`FI_VERBS_INLINE_SIZE`	`fi_inject` および `fi_tinject` 呼び出しの最大メッセージサイズを定義します。
`FI_VERBS_IFACE`	Verbs デバイスに関連付けられているネットワーク・インターフェイスのプリフィクスまたは完全な名前を定義します。デフォルト値は **`ib`** です。
`FI_VERBS_MR_CACHE_ENABLE`	メモリー・レジストレーション・キャッシュを使用します。デフォルト値は **0** です。 **注:** 1 に設定すると、中規模および大規模メッセージの帯域幅を増やすことができます。
`FI_MR_CACHE_MONITOR`	デフォルトのメモリー・レジストレーション・モニターを定義します。モニターは、仮想および物理メモリーアドレスの変更をチェックします。設定可能なオプションは次のとおりです。 • **userfaultfd:** システムで利用可能な場合、この Linux* カーネル機能がデフォルトです。 **注:** userfaultfd は、カーネルバージョン 4.11 以降、Linux* ディストリビューション RHEL 7.3 以降で利用できます。 • **memhooks:** デフォルトのメカニズムとして、このオプションはメモリーの割り当てと解放呼び出しを検出して動作します。 • **disabled:** メモリーキャッシュを無効にします。

必要条件

追加で、次のライブラリーがインストールされている必要があります。

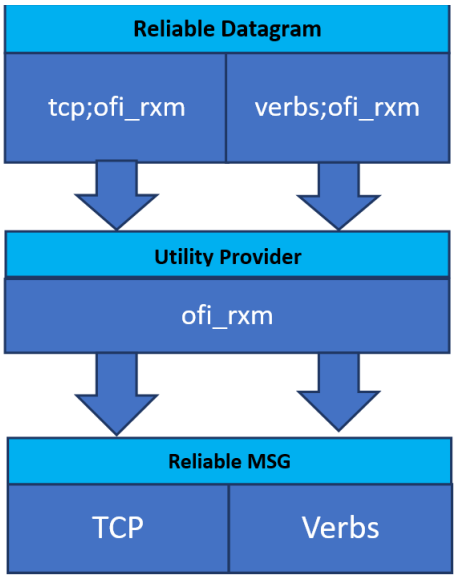
- `IPoIB` がインストールされ、適切に設定されている必要があります。
- 接続管理に `librdmacm` 1.0.16 以降の `ibacm` が必要です。

注: libfabric をソースからコンパイルして Verbs サポートを有効にする場合、上記の 2 つのライブラリーに対応するヘッダーファイルが必要です。ライブラリーとヘッダーファイルがデフォルトのパスにない場合、`CFLAGS`、`LDFLAGS`、および `LD_LIBRARY_PATH` 環境変数で指定します。

詳細は、fi_verbs(7) (英語) を参照してください。

RxM

説明



RxM プロバイダー (ofi_rxm) は、Verbs または TCP コア・プロバイダー上のユーティリティー・プロバイダーです。ユーティリティー・プロバイダーは、通常、特定のエンドポイント・タイプをサポートするために使用されます。例えば、RxM ユーティリティー・プロバイダーは、MSG エンドポイント上の Reliable Datagram Sockets メカニズムをサポートします。

注: コア・プロバイダーがアプリケーションで要求される機能セットをサポートしない場合、このユーティリティー・プロバイダーは自動的に有効になります。

一般的なコントロール・パラメーター

名前	説明
`FI_OFI_RXM_BUFFER_SIZE`	送信バッファーサイズ/インジェクト・サイズを定義します。小さなサイズのメッセージは、eager プロトコルを介して送信されます。大きなサイズのメッセージは、SAR または rendezvous プロトコルを介して送信されます。送信データは、指定したサイズまでコピーされます。デフォルトのサイズは 16K です。
`FI_OFI_RXM_USE_SRX`	RxM 受信パスを制御します。変数を **1** に設定すると、RxM はコア・プロバイダーの共有受信コンテキストを使用します。デフォルト値は **0** です。 **注:** このモードはメモリー消費を改善しますが、小規模メッセージのレイテンシーが増える可能性があります。

名前	説明
FI_OFI_RXM_SAR_LIMIT	RxM SAR (セグメンテーションと再構築) プロトコルを制御します。大きなサイズのメッセージは、rendezvous プロトコルを介して送信されます。デフォルト値は 256KB です。 **注:** RxM SAR は、iWarp ではサポートされません。

必要条件

外部の必要条件はありませんが、TCP や Verbs などの信頼性の高いメッセージ・プロバイダーが必要です。

詳細は、fi_rxm(7) (英語) を参照してください。

その他

PSM や GNI プロバイダーを ofiwg/libfabric ソースからビルドして、インテル® MPI ライブラリーとともに使用することが可能です。

インテル® True Scale を使用してインテル® MPI ライブラリーを実行するには、次の手順を実行します。

1. 「ソースからの libfabric のビルドと設定」に示すように、libfabric ソースをダウンロードして設定します。設定時に、必ず PSM プロバイダーを有効にします。設定後、次のメッセージが出力されます。

   ```
   *** Built-in providers: psm
   ```

2. 「ビルドした libfabric の使用」セクションに示すように、インテル® MPI ライブラリーがビルドした libfabric を使用するように強制します。
3. バージョン 2019 Update 6 では、追加のコントロールを指定する必要があります。

   ```
   export FI_PSM_TAGGED_RMA=0
   export MPIR_CVAR_CH4_OFI_ENABLE_DATA=0
   ```

4. 通常どおりに、インテル® MPI ライブラリーを実行します。

ソースからの libfabric のビルドと設定

.tar パッケージは、GitHub* の Releases タブ (英語) から入手できます。libfabric をビルドするには、次の手順を実行します。

1. 任意のリリースパッケージ (例えば、`libfabric-1.9.0.tar.bz2`) をダウンロードします。
2. .tar ファイルを展開します (`tar xvf <package_name>`)。
3. libfabric ディレクトリーに移動します。
4. `autogen.sh` を実行します。
5. `configure` を実行します。
6. `make` を実行します。
7. ユーパーユーザーまたは root 権限で `make install` を実行します。

設定オプション

設定オプションには、多くのビルトインオプションがあります (`./configure --help` を参照)。以下は、いくつかの便利なオプションです。

`--prefix=<directory>`
デフォルトでは、make install は `/usr` 以下にファイルを配置します。`--prefix` オプションは、`<directory>` で指定したディレクトリー以下に libfabric ファイルをインストールするように指示します。実行ファイルは、`<directory>/bin` 以下にあります。

`--with-valgrind=<directory>`
Valgrind のインストール・ディレクトリーを指定します。Valgrind が検出されると、Valgrind アノテーションが有効になります。これにより、パフォーマンスが低下する可能性があります。

`--enable-debug`
デバッグ・コード・パスを有効にします。さまざまな追加のチェックを有効にし、通常はプロダクション・ビルドでコンパイルされる詳細なログ出力を使用できるようにします。

`--enable-<provider>=[yes|no|auto|dl|<directory>]`
<provider> という名前のプロバイダーを有効にします。有効なオプションは、次のとおりです。

- **`auto`**: `--enable-<provider>` オプションが指定されていない場合のデフォルトです。プロバイダーは、必要条件をすべて満たしている場合は有効になり、1 つでも満たしていない場合は無効になります。
- **`yes`**: `--enable-<provider>` オプションが指定されている場合のデフォルトです。設定スクリプトは、プロバイダーを有効にできない場合 (例えば、一部の必要条件を満たしていない場合など)、アボートします。
- **`no`**: プロバイダーを無効にします。`--disable-<provider>` と同義です。
- **`dl`**: プロバイダーを有効にして、ロード可能なライブラリーとしてビルドします。
- ***`<directory>`***: プロバイダーを有効にして、*`<directory>`* で指定されたインストールを使用します。

`--disable-<provider>`
`<provider>` という名前のプロバイダーを無効にします。

すべてのプロバイダーは、`dl` オプションを使用するように設定され、個別の依存関係を持つ独立したバイナリーのセットとしてビルドされます。libfabric 自体には外部の依存関係はありません。動的にロード可能なプロバイダーは、`FI_PROVIDER_PATH` 変数を使用して簡単にオーバーライドできます。

通常の方法でビルドされているプロバイダーは、libfabric ライブラリーに静的にリンクされます。この場合、libfabric.so はすべてのビルトイン・プロバイダーの依存関係を持ちます。

例

次の例について考えてみます。

```
$ ./configure --prefix=/opt/libfabric --disable-tcp && make -j 32 && make
install
```

設定が完了すると、次のメッセージが表示されます。

```
*** Built-in providers: rxd rxm verbs
*** DSO providers:
```

これにより、ソケット・プロバイダーを無効にし、libfabric を `/opt/libfabric` 以下にインストールするように libfabric に指示します。可能な場合、ほかのすべてのプロバイダーが有効になり、すべてのプロバイダーが単一のライブラリーにビルドされます。

別の方法:

```
$ ./configure --prefix=/opt/libfabric --enable-debug --enable-tcp=dl && make
-j 32 && make install
```

設定が完了すると、次のメッセージが表示されます。

```
*** Built-in providers: rxd rxm verbs sockets
*** DSO providers: tcp
```

これにより、TCP プロバイダーをロード可能なライブラリーとして有効にし、すべてのデバッグ・コード・パスを有効にし、libfabric を `/opt/libfabric` 以下にインストールするように libfabric に指示します。可能な場合、ほかのすべてのプロバイダーが有効になります。

インストールの検証

fi_info (英語) ユーティリティーを使用して、libfabric とプロバイダーのインストールを検証し、プロバイダー・サポートと利用可能なインターフェイスに関する詳細を得られます。ユーティリティーの使用方法は、libfabric パッケージの一部としてインストールされる fi_info を参照してください。

より包括的なテストスイートは、fabtests (英語) ソフトウェア・パッケージを介して利用できます。また、2 つのプロセス間でデータを送信するピンポンテストの fi_pingpong (英語) を使用して検証することもできます。

ビルドした libfabric の使用

ビルドした libfabric を使用するには、`LD_LIBRARY_PATH` にライブラリーのパスを、`FI_PROVIDER_PATH` にダイナミック・ライブラリーとしてビルドされた OFI プロバイダーのパスを指定します。

構文

```
export LD_LIBRARY_PATH=<ofi-install-dir>/lib:$LD_LIBRARY_PATH
export FI_PROVIDER_PATH=<ofi-install-dir>/lib/libfabric
```

Readme

libfabric のビルドとインストールに関する追加の情報は、libfabric の readme ファイル (英語) を参照してください。

Windows* オペレーティング・システム

インテル® MPI ライブラリーでサポートされる OFI プロバイダー

このセクションでは、インテル® MPI ライブラリーでサポートされる OFI プロバイダーについて説明します。プロバイダーごとに、インテル® MPI ライブラリーに関連する説明とパラメーターがあり、オプションで必要条件や制限事項があります。

TCP

説明

TCP プロバイダーは、インテル® MPI ライブラリーの汎用プロバイダーです。libfabric API を実装するため TCP ソケットをサポートする任意のシステムで使用できます。このプロバイダーでは、特定の高速インターコネクトを持たないクラウド環境 (例えば、GCP、イーサネット接続の Azure*、AWS*インスタンス) や IPoIB を使用して、通常のイーサネット上でインテル® MPI ライブラリー・アプリケーションを実行できます。

一般的なコントロール・パラメーター

名前	説明
FI_TCP_IFACE	特定のネットワーク・インターフェイスを指定します。
FI_TCP_PORT_LOW_RANGE FI_TCP_PORT_HIGH_RANGE	TCP プロバイダーがパッシブ・エンドポイントの作成に使用するポートの範囲を設定します。これは、ファイアウォールで TCP 接続用のポート範囲が指定されている場合に便利です。

必要条件

必要条件はありません。

詳細は、fi_tcp(7) (英語) を参照してください。

ソケット

説明

インテル® MPI ライブラリーでは、ソケット・プロバイダーは主に Windows* 用のプロバイダーです。すべての libfabric 必要条件とインターフェイスを適用するため、TCP ソケットをサポートする任意のシステムで使用できます。このプロバイダーでは、通常のイーサネット環境でインテル® MPI ライブラリー・アプリケーションを実行できます。

一般的なコントロール・パラメーター

名前	説明
FI_SOCKETS_IFACE	特定のネットワーク・インターフェイスを指定します。
FI_SOCKETS_CONN_TIMEOUT	1 回の接続確立で待機するミリ秒を整数値で指定します。
FI_SOCKETS_MAX_CONN_RETRY	失敗として報告する前にソケットの接続を再試行する回数を整数値で指定します。

必要条件

必要条件はありません。
詳細は、fi_sockets(7) (英語) を参照してください。

ソースからの libfabric のビルドと設定

Windows* 上で libfabric をビルドするには、NetDirect プロバイダーが必要です。NetworkDirect SDK/DDK は、https://www.nuget.org/packages/NetworkDirect (英語) からナゲットパッケージとして取得するか (推奨)、https://www.microsoft.com/ja-JP/download/details.aspx?id=36043 からダウンロードできます ([ダウンロード] ボタンをクリックして、NetworkDirect_DDK.zip を選択します)。

1. ダウンロードした NetworkDirect_DDK.zip からヘッダーファイルを展開します。\NetDirect\include\ ファイルを <libfabricroot>\prov\netdir\NetDirect\ に展開するか、NetDirect ヘッダーへのパスを VS のインクルード・パスに追加します。
2. コンパイルします。libfabric には 6 つの Visual Studio* ソリューション構成があります。
   - 1-2: Debug/Release ICC (インテル® コンパイラー XE 15.0 専用の制限付きサポート)
   - 3-4: Debug/Release v140 (VS 2015 ツールセット)
   - 5-6: Debug/Release v141 (VS 2017 ツールセット)

   注: 必ず使用するコンパイラーに対応する正しいターゲットを選択してください。デフォルトでは、ライブラリーは *<libfabricroot>*\x64*<yourconfigchoice>* にコンパイルされます。

3. ライブラリーをリンクします。
   1. プロジェクトを右クリックして、プロパティーを選択します。
   2. [C/C++] > [全般] を選択して、[追加のインクルード・ディレクトリー] に *<libfabricroot>*\include を追加します。
   3. [リンカー] > [入力] を選択して、[追加の依存ファイル] に *<libfabricroot>*\x64*<yourconfigchoice>*\libfabric.lib を追加します。

ビルドした libfabric の使用

Windows* でビルドした libfabric を使用する方法は 2 つあります。

1. PATH 環境変数にビルドした dll バイナリーを指定します。
   `set PATH=<path_to_your_dll>;%PATH%`
2. ビルドしたダイナミック・ライブラリー libfabric.dll をターゲットフォルダーに配置します。
   `<impi_build>\intel64\libfabric\bin`

関連情報

1. Open Fabrics Interface プロジェクト (英語)
2. "A Brief Introduction to the OpenFabrics Interfaces" by Paul Grun, Sean Hefty, Sayantan Sur, David Goodell, Robert D. Russell, Howard Pritchard, and Jeffrey M. Squyres
3. 2019 OFA ワークショップ (英語)
