ROGUE WAVE SOFTWARE

WHITE PAPER

ハイライト：

- メモリデバッグの問題
- 並列分散アプリケーションにおけるメモリバグの特定と解決
- Heap Interposition Agent (HIA) を使用したメモリ問題の分析

並列分散アプリケーションにおけるメモリデバッグ

概要：

メモリバグは基本的にはヒープメモリ管理の誤りであり、開発、改良、メンテナンスが行われるすべてのプログラムで発生する可能性があります。メモリバグは、エラー状態の確認ミス、非標準動作への依存、メモリ解放ミスを含むメモリリーク、ポインタのクリアミスなどのダングリング参照、配列境界違反、所有していないメモリへの書き込みや配列境界外へのオーバーランなどのメモリ破壊を含む、様々な要因により発生します。これらのメモリバグは、プログラムのクラッシュまたは不適切な結果をランダムに引き起こす場合があります。さらに厄介なことに、長期間にわたりコードベースに潜在し、最悪のタイミングで顕在化することもあります。

メモリ問題は、単純なデスクトップアーキテクチャーでも従来のツールでは追跡することが難しく、分散並列アーキテクチャー上で発生すると頭痛の種となります。本書では、特に並列開発とデバッグの問題に注目しつつメモリデバッグの問題を見直し、開発者が並列分散アプリケーション上でメモリバグを特定し解決するためのツールを紹介し、その特徴と使用のヒントについて記載します。

本資料は、TotalView Technologies (現 Rogue Wave Software, Inc.) によって作成され2008年6月に発行されたのホワイトペーパー「Memory Debugging in Parallel and Distributed Applications」を日本語訳して作成されたものです。英語原文と日本語訳との間で内容に相違がある場合には英語原文が優先されます。Rogue Wave Software, Inc.: http://www.roguewave.com/ よりダウンロードしてご参照下さい。

ROGUE WAVE SOFTWARE

WHITE PAPER

並列開発におけるメモリデバッグの問題

メモリバグはいつでも発生する可能性があり、特に共同で書かれたコードや長期間にわたって保持されているコードなどでは、メモリ管理の前提が変化したり明確に伝わっていない場合があり、メモリデバッグを困難な課題にしています。メモリバグは、直ちに致命的な影響を与えるものではないことも多く、そのような場合はコードベースに長期間潜在し、プログラムが新しいアーキテクチャーにポートされたり、より大きな問題へとスケールアップされたり、プログラム間でコードの適合や再使用が行われたりすることで、突然問題になる可能性もあります。

メモリバグは、常に発生するクラッシュや、時に発生するクラッシュ(不定期)、または単に不適切な結果として顕在化します。一般に使用されている開発ツールや技術(printfや従来のソースコードデバッガなど)はメモリ問題を解決するために設計されていないため、これらのツールや技術でメモリバグを追跡することは困難です。

この状態に並列性が追加されることで状況は更に悪化します。多くの場合、並列プログラムは2つの効果の間にねじ込まれており、メモリの扱いについては十分な慎重さが求められるためです。さらに、並列プログラムはプロブレムセットが「大きい」状況下で書かれているため、プログラムは必然的に大量のデータをロードし、多大なメモリを使用することになります。しかしメモリは高価で、大抵の場合、特殊な高性能計算機(HPC)システムにおける各ノードのメモリ容量は十分ではありません。

メモリエラーの分類

通常、プログラムは異なる方法で管理されている複数のメモリカテゴリを使用します。その中には、スタックメモリ、ヒープメモリ、共有メモリ、スレッドプライベートメモリ、静的メモリまたはグローバルメモリが含まれます。しかし、プログラマはヒープメモリの外側に割り当てられたメモリに特に注意する必要があります。なぜなら、ヒープメモリの管理はコンパイルまたはランタイムに黙示的に行われるのではなく、プログラム内で明示的に行われるからです。

プログラムは、様々な状況で動的に割り当てられたヒープメモリの適正使用に失敗します。今後の議論のため、これらの失敗を簡単に分類し、以降本書ではC malloc() APIと呼びます。ただし、新しいC++ステートメントやFORTRAN 90割り当てステートメントを使用して割り当てられたメモリでは、類似エラーが発生する場合もあることに留意してください。

***Malloc*エラー**

Mallocエラーは、プログラムがC Heap Manager APIにおけるいずれかの操作で無効値を渡した場合に発生します。この状態は、ポインタ値(ブロックのアドレス)が他のポインタに複製され、後に両方のポインタがfree()に渡されると発生する可能性があります。この場合、指定されたポインタは割り当てられたブロックに対応しないため、2番目のfree()は間違いとなります。このような操作を行った場合のプログラムの挙動は明確にはなっていません。

ダングリングポインタ

ポインタが既に解放されているメモリを参照する場合、そのポインタはダングリングしていると言えます。ダングリングポインタによるメモリアクセス(読み込み/書き込みにかかわらず)は、定義されていない挙動を引き起こす可能性があります。ダングリングポインタバグが発生したプログラムは、ダングリングポインタが指すメモリがアクセスされている間に新しい割り当てにリサイクルされない限り、長期間にわたり明確なエラーが発生せず機能しているかのように見えることもあります。

メモリ境界違反

*malloc()*が返す個別のメモリ割り当ては、規定サイズの離散したメモリブロックを表します。ブロックの最下位アドレス直前、またはブロックの最上位アドレス直後のメモリへのアクセスは、不明確な挙動につながる可能性があります。

***Read-Before-Write*エラー**

初期化前にメモリを読み込むのは、一般的なエラーです。一部の言語では初期化されていないグローバルメモリにデフォルト値を割り当て、また多くのコンパイラは初期化前におけるローカル変数の読み込みを認識できます。ポインタ経由でアクセスされたメモリが初期化前に読み込まれたことを検出することは更に難しく、通常は常にできるものではありません。動的メモリは常にポインタ経由でアクセスされ、ほとんどの場合メモリマネージャから取得したメモリコンテンツは未定義であるため、特に影響を受けます。

メモリリークの検出

プログラムがメモリブロックの使用を終え、そのブロックへの参照をすべて破棄したにもかかわらず、ヒープマネージャーに解放して再使用するためのfree()の呼び出しに失敗すると、リークが発生します。その結果、プログラムはメモリを使用することもできず、また新しい目的のために再割り当てすることもできなくなります。

リークの影響度は、アプリケーションの特性により異なります。ほとんど影響を受けない場合もありますが、リーク率が高かったりプログラムのランタイムが長かったりすると、リークによりメモリ動作やプログラムの性能特性が著しく変化することもあります。また、実行時間の長いアプリケーションやメモリが限られているアプリケーションでは、リーク率が低くても深刻な影響が蓄積される場合もあります。このように、少々逆説的ですがリークは十分理解されたコード上に長く潜在するため、非常に厄介なものとなります。複雑なアプリケーションにおいて、メモリ割り当てを必ず1回だけ解放することでmallocエラーおよびリークエラーが発生しないよう動的メモリを管理することはなかなか困難です。

リーク検出は、プログラム実行中いずれのポイントでも実行できます。前述したとおり、リークはプログラムがfreeを呼び出さないままメモリブロックの使用を停止すると発生します。「使用の停止」を定義することは難しいですが、高度なメモリデバッガであればプログラムが特定のメモリ位置への参照を保持しているかどうかを確認することでリークを検出できます。

MemoryScapeデバッガ

MemoryScapeメモリデバッガは、開発者にとって使いやすいツールです。Lightweightなアーキテクチャが採用されているため再コンパイルを必要とせず、プログラムのランタイム性能にほとんど影響を与えません。インターフェイスは帰納的ユーザインターフェイスの概念に基づき設計されており、メモリデバッグのタスクについてユーザを誘導し、分かりやすいグラフィカル表示、パワフルな分析ツール、協働体制のサポート機能(ライブラリベンダー、共同研究者、または問題となっているコードを書いた同僚などに潜在メモリバグを報告しやすくなる機能)を提供します。

MemoryScapeは、並列/マルチプロセスアプリケーション用に設計されており、個別プロセスの詳細情報と、大規模並列アプリケーションを構成する全プロセスの大まかなメモリ稼働統計の両方を提供します。MemoryScapeの特殊機能には、並列ジョブの全プロセスの立ち上げおよび自動アタッチのサポート機能、1つのGUI内における複数プロセスのメモリデバッグ機能、バッチキュー環境を使用するためのスクリプトに基づくデバッグ機能などがあり、これらの機能はMemoryScapeを並列分散アプリケーションのデバッグにより適したものにしています。

MemoryScapeアーキテクチャー

MemoryScapeは、インターポジションという技術を応用することで、並列分散アプリケーション上でメモリデバッグを実行します。MemoryScapeは、ユーザのアプリケーションコードとmalloc()サブシステムの間に挿入されるHeap Interposition Agent (HIA)と呼ばれるライブラリを提供します。このライブラリは、メモリ割り当てAPI機能をそれぞれ定義します。これらの機能は、プログラムがメモリブロックの割り当て、再割り当て、解放を実行する場合に最初に呼び出されます。

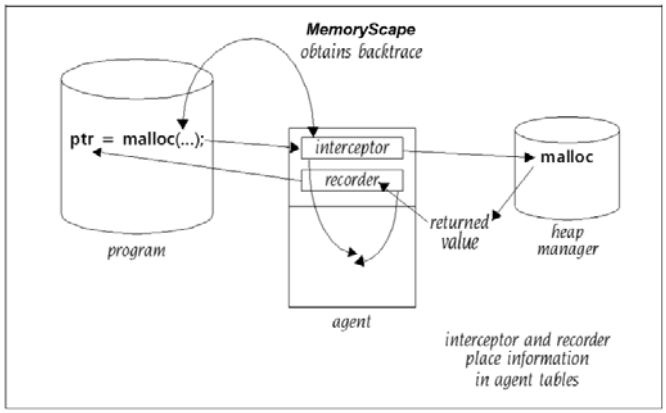

図.1 MemoryScape Heap Interposition Agent (HIA)のアーキテクチャーHIA は、アプリケーションおよび glibc内のメモリ割り当て層の間に存在します。

MemoryScapeでインターポジション技術を採用した理由の一つは、Lightweightなメモリデバッグ機能です。HIAの存在がプログラム性能に影響を与えないためには、オーバーヘッドを小さくすることが重要です。ほとんどの場合、HIAでデバッグを実行しているプログラムとHIAが存在しないプログラムのランタイム性能は類似しています。高性能計算機アプリケーションにおいては、ターゲットプログラムを著しく遅くするheavyweightなアプローチにより、開発者、管理者、およびジョブスケジューラーが我慢できないようなプログラムランタイムになる場合もあり、これは極めて重要な機能です。

インターポジションライブラリはいずれの操作も実際には実行しないため、インターポジションは単純にmalloc()ライブラリをデバッグmallocに置換えるものではありません。プログラムのmalloc()API機能の呼び出しが、HIAがない場合に呼び出される下層のヒープマネージャーに転送されるよう調整します。HIAによるインターポーズの効果は、HIAが存在しない場合と同じようにプログラムが動作することです。ただし、HIAはメモリの呼び出し全てを傍受で

き、下層の機能が呼び出される前後に記録と「サニティーチェック」を実行できます。

HIAライブラリの記録はプログラムが実行されるたびに蓄積され、ヒープ上のアクティブな割り当ての全記録を保管します。ヒープ上の各割り当てについては、ブロックの位置とサイズ以外にも、ブロック割り当て時のプログラムの挙動を表示する完全なコールスタックも記録しています。HIAが実行する「サニティーチェック」は、同一のメモリブロックを2回解放したり、スタックアドレスを指すポインタの再割り当てを試みるなどのmalloc()エラーを、HIAが検出するためのものです。

HIAの構成方法によっては、更に境界エラーが発生したかどうかも検出できます。HIAが収集する情報は、ユーザにヒープ状態を正確に伝えるためにMemoryScapeメモリデバッガが使用します。

***MemoryScape*並列アーキテクチャー**

MemoryScapeは、水面下分散並列アーキテクチャーを使用して、ユーザの並列プログラムとのランタイムインタラクションを管理します。MemoryScapeは、ユーザのコードが実行されているクラスタのノード上で実行される軽量デバッグエージェントプロセス(tvdsvrプロセス)を起動します。これらのtvdsvrプロセスは、それぞれ個別のローカルプロセスとデバッグ対象プロセスにロードされたHIAモジュールとの下位インタラクションに対応します。tvdsvrプロセスは、ほとんどのクラスタ構成においてTCP/IPの上層に位置する独自の最適化されたプロトコルを使用して、MemoryScapeのフロントエンドプロセスと直接通信します。

MemoryScapeの特性

***MemoryScape*を使用したメモリ統計の比較**

多くの並列分散アプリケーションのメモリ使用に関しては、既知または期待される挙動があります。それらは、すべてのノードが同量のメモリを割り当てるように構成されているか、メモリ使用が対象プロセスの MPI COMMWORLDランクに基づくように構成されています。このようなパターンが予想される場合、またはユーザが単にプロセスセットを確認してパターンを把握したい場合、MemoryScapeはメモリ統計ウィンドウを使用します。メモリ統計ウィンドウは、デバッグセッションを構成する1つ、すべて、または任意のサブセットのプロセスについて、総合的なメモリ使用統計を多様なグラフ(線グラフ、棒グラフ、円グラフ)で表します。ユーザは、特定のブレークポイントやバリアまでプログラムを実行でき、また単に実行中の任意のポイントですべてのプロセスを停止することもできます。

ユーザは、"generate view"をクリックして、統計情報の表示を希望するプロセスセットと表示タイプを選択できます。生成された画面には、その時点でのプログラムの状態が表示されます。ユーザはデバッガプロセスコントロールを使用して、新しいポイントまでプログラムを実行し、表示を更新して変化を確認できます。プロセスが線外に表示された場合、ユーザはヒープメモリの詳細な状態を確認できます。

***MemoryScape*を使用したヒープ状態の確認**

MemoryScapeは多様なヒープステータスレポートを提供しており、最も有名なレポートがヒープグラフィカル表示です。プロセスが停止したいずれかのポイントで、ユーザはヒープのグラフィカル表示を取得できます。その場合、ヒープステータスタブを選択し、1つ以上のプロセスを選択し、グラフィカル表示を選択して"generate view"をクリックします。

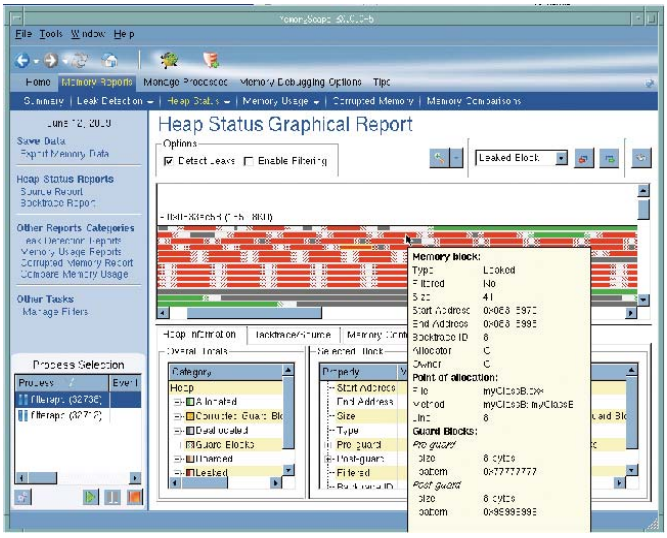

図2. *MemoryScape*グラフィカルインターフェイスが提供するヒープのインタラクティブ表示。メモリの割り当て状態は色分けして表示されます。

選択したプロセスのヒープメモリの状態が表示されます。現在のヒープメモリ割り当ては、割り当ての一部であるアドレス全体にかかる緑の線で表示されます。これにより、ユーザは一目でプログラムのヒープメモリ構成を把握できます。この表示はインタラクティブです。ブロックを選択すると関連する割り当てが反転表示され、選択されたブロックと関連ブロックのフルセットに関する詳細情報がユーザに表示

されます。また、サイズや割り当てられた共有オブジェクトなどのプロパティに基づきフィルタリングを行い、割り当てを灰色表示することも可能です。ベースラインを設定し、そのベースライン前後に発生した割り当ておよび割り当て解除も表示できます。

MemoryScapeを使用したリーク検出

MemoryScapeは、プログラムを既知の状態(ブレークポイントなど)まで実行するか、あるいは単にGUIまたはCLIで"halt"コマンドを使用して並列アプリケーション実行プロセスを停止して、ヒープメモリリークを検出します。メモリデバッグウィンドウでリーク検出タブを選択すると、並列ジョブの1つ以上のプロセスが選択でき、リークレポートが生成されます。

生成されるレポートには、プログラムのレジスタまたはアクセス可能なメモリにおいて既に無効な参照となっている、プログラム内のヒープ割り当てがすべてリストアップされます。プログラムがどこにも参照を保存しないメモリブロックは、非常に高い確率でその後free()コールの対象となり、リークとなります。前述したとおり、リークはヒープグラフィカル表示でも確認できます。その場合、"Detect Leaks"と表示されたチェックボックスを切り替えると、グラフィカル表示上でリークブロックが赤色表示されます。

MemoryScapeを使用したヒープ境界違反検出

前述したメモリエラーの分類で診断が難しい傾向にあるのがプログラムロジック内のエラーが原因で、プログラムがヒープ上に割り当てられたメモリブロックの境界を越えて書き込むエラーです。malloc APIは、別のメモリ割り当てで返されたメモリブロックの関連スペーシングやアラインメント、または指定ブロック前後のメモリの使途について保証していません。メモリブロックの開始前または終了後における読み込み/書き込みの結果は定義されていません。

実際には、大抵のブロックはプログラムデータの他のブロックに隣接しています。そのため、プログラムが配列の終了点を超えて書き込んだ場合、通常は他の関連しない割り当てのコンテンツを上書きすることになります。プログラムが再実行されて同じエラーが発生する場合、割り当ての順番が変更され、異なる配列上で上書きされる可能性があります。その結果、例えば5回はプログラムのクラッシュ、時に不適切なデータ、時にまったく影響のない形でメモリを書き換えるなど、毎回異なる方法で顕在化する非常に苛立たしく「際どい」バグにつながります。

MemoryScapeは、割り当て時にヒープメモリブロックの前後にメモリビットを設定するメカニズムを提供します。このメモリビットはガードブロックと呼ばれ、割り当ての一部ではありません。そのため、プログラムはこの場所の読み込み/書き込みは実行しません。HIAはガードブロックをあるパターンで初期化し、このパターンの変化に対して(ユーザがチェックを要求したり、個別のブロック割り当てが解放されたりした場合はいつでも)ガードブロックをチェックするよう調整できます。何らかの変化があった場合、プログラムが配列境界を越えて書き込んだことを意味します。

MemoryScapeの協働機能

MemoryScapeは、分散して作業する開発チームが効果的に協働して問題解決にあたり、製品品質を改善できる2つの形式のメモリレポートを提供しています。ヒープメモリ表示は、ウェブブラウザで読むためにHTMLファイルでエクスポートできます。これらのHTMLファイルには、画面上でレポートを使用して作業するのと同じような方法で、ユーザ個人のブラウザでレポートを使用して作業できるJavaScriptアプレットが含まれています。これによりレポート読者は、興味のあるセクションについては詳細を表示し、それ以外のセクションについては概要のみ表示することができます。

MemoryScapeでは、メモリデバッグデータファイルの作成もサポートされています。これは、プロセスに関してMemoryScapeが持つすべてのデータを2進法表示したものです。これらのファイルは後日ロードバックし、ライブプロセスのように作業することができます。この機能により、開発者は再現されたプロセスを保存し、後で比較や精査できます。

メモリデバッグデータファイルは、TotalView® Source Code Debuggerのメモリモジュールを使用してロードすることも可能です。この機能により、上級ユーザはよりパワフルで正確なデバッグ技術を適用でき、メモリデバッグ機能とライブプロセスの全変数およびステートデータへのアクセスを最大限に活用できます。

MemoryScape使用のヒント

前述したとおり、MemoryScapeはプログラムがエラーによりヒープ配列の範囲を超えて書き込む事例を多く検出します。開発者および科学者は、ヒープまたはグローバルプログラムメモリへ自動的に割り当てられた配列のためにコンパイラが生成した境界確認コードを使用して、MemoryScapeのヒープ境界チェックを実行できます。Intel™ Fortran Compilerやオープンソースのgfortranコンパ

イラを含む多くのコンパイラは、コンパイルラインオプション(例えばifortでは-checkbounds、gfortranでは-fbounds-check)を使用して、境界チェックコードを自動的に生成できます。

より高度なメモリデバッグ機能が必要な開発者のために、TotalView Debuggerはプログラムのソースコードコンテキスト内でユーザがメモリ情報を調査できるメモリ問題のデバッグ方法を提供しています。TotalViewソースコードデバッガを使用してメモリデバッグを実行する場合、ユーザはプログラム内のヒープメモリを指すポインタを含有する可能性のあるデータ構造を調査できます。メモリデバッグが有効化されると、これらのポインタには指しているメモリブロックの状態に関する情報が付加されます。

TotalViewソースコードデバッガとメモリデバッガを併用しているユーザが使用できる高度な技術に、ウォッチポイントを使用する方法があります。

ウォッチポイントは、指定したメモリブロックに書き込みが行われた時点でプロセスを停止できるデバッガ機能です。ポインタがメモリ全体に「ワイルド」に(ポインタが書き込みを行うべき場所とはまったく関係のないスペースに)書き込みを行っている場合、それを抑えることは難しいかもしれません。

ガードブロックとウォッチポイントを併用することで、特に微妙なエラーを検出できます。トラブルシューティングには2つのパスがあります。プログラムの最初のパスでは、ガードブロックは誤って書き込まれたメモリブロックの特定に使用されます。次に、プログラムの2番目のパスで、ウォッチポイントが正確なアドレスにセットされます。ウォッチポイントは2回トリガされます。1回目はメモリグロックがメモリデバッガによりペイントされた時、2回目はメモリブロックがワイルドポインタにより上書きされた時です。

ほとんどのユーザは前述したとおりMemoryScapeをインタラクティブに使用することを希望すると思いますが、開発は現在も進行中で、そのためいつ新しいメモリバグが発生してもおかしくありません。開発チームは、現在進行中のテスト戦略に、ヒープメモリテストを追加することをお勧めします。MemoryScapeには、非インタラクティブなコマンドラインバージョンが含まれており、これは自動テストフレームワークに組み込む目的で設計されています。この方法でMemoryScapeを使用する開発チームは、開発時に発生した新しいメモリエラーを即座に、かつ本番に影響が出る前に検出、分析および削除できます。

今後のMemoryScape製品計画

MemoryScapeの今後の開発では、TotalView Workbench Managerアプリケーションとの統合の改善が予定されています。Workbenchは、開発者が必要とする異なる開発ツール間で、構成およびセッションデータを共有するためのサイトを提供しています。Workbenchは、ユーザが最近のデバッグ、メモリデバッグ、および性能分析セッションにアクセスできる、拡張可能なベースを提供しています。統合開発環境(IDE)はよく知られていますが一般的ではないことを考慮し、WorkbenchはIDEで使用できますが必須ではありません。MemoryScape製品の今後のバージョンでは、プログラム、入力値とパラメータ、メモリデータファイルなどの状態および情報をWorkbenchと共有する予定です。

他にもMemoryScapeの今後の開発には、開発者がプログラムのメモリ使用状況を正確に理解できるよう、アプリケーションにおけるメモリ使用履歴の分析機能を改善することが含まれています。最も多くの強化リクエストを受けるのが、ヒープ割り当て範囲を超えたメモリの読み込み/書き込みを検出およびレポートするために開発者が使用できる、追加オプションの開発です。ランタイム性能の代償としてこの領域のより詳細な情報を開発者が得られるよう、TotalView Technologiesは積極的に機能強化の可能性を調査しています。

結論

メモリバグは、開発、改良、メンテナンスが行われるすべてのプログラムに発生する可能性があります。概してこのタイプのバグは、いつでも発生する可能性があり、様々な原因で引き起こされるため、開発者にとっては大きなストレスとなります。また、長期間にわたりコードベースに潜在している可能性もあり、様々な方法で顕在化します。

そのためメモリデバッグは、特に大量のデータを持ち、多くのメモリを使用する並列分散プログラムでは、非常に骨の折れる作業となっています。一般的に使用される開発ツールや技術はメモリ問題を解決するために設計されていないため、メモリバグの検出および修正プロセスはさらに複雑になります。

MemoryScapeは、開発者がメモリバグを特定して解決するための、使いやすいメモリデバッグツールです。

WHITE PAPER

MemoryScapeの特殊機能には、メモリ統計の比較、ヒープ状態の把握とメモリリークの検出などがあり、並列分散アプリケーションにおけるデバッグに適した独自のソフトウェアです。

詳細は、http://www.roguewave.jp/products/TotalView/overview.html を
ご参照下さい。