

Oracle Solaris Studio 12.3 Discover および Uncover ユーザーズガイド

このソフトウェアおよび関連ドキュメントの使用と開示は、ライセンス契約の制約条件に従うものとし、知的財産に関する法律により保護されています。ライセンス契約で明示的に許諾されている場合もしくは法律によって認められている場合を除き、形式、手段に関係なく、いかなる部分も使用、複写、複製、翻訳、放送、修正、ライセンス供与、送信、配布、発表、実行、公開または表示することはできません。このソフトウェアのリバース・エンジニアリング、逆アセンブル、逆コンパイルは互換性のために法律によって規定されている場合を除き、禁止されています。

ここに記載された情報は予告なしに変更される場合があります。また、誤りが無いことの保証はいたしかねます。誤りを見つけた場合は、オラクル社までご連絡ください。

このソフトウェアまたは関連ドキュメントを、米国政府機関もしくは米国政府機関に代わってこのソフトウェアまたは関連ドキュメントをライセンスされた者に提供する場合は、次の通知が適用されます。

U.S. GOVERNMENT END USERS:

Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

このソフトウェアもしくはハードウェアは様々な情報管理アプリケーションでの一般的な使用のために開発されたものです。このソフトウェアもしくはハードウェアは、危険が伴うアプリケーション（人的傷害を発生させる可能性があるアプリケーションを含む）への用途を目的として開発されていません。このソフトウェアもしくはハードウェアを危険が伴うアプリケーションで使用する場合、安全に使用するために、適切な安全装置、バックアップ、冗長性（redundancy）、その他の対策を講じることは使用者の責任となります。このソフトウェアもしくはハードウェアを危険が伴うアプリケーションで使用したこと起因して損害が発生しても、オラクル社およびその関連会社は一切の責任を負いかねます。

OracleおよびJavaはOracle Corporationおよびその関連企業の登録商標です。その他の名称は、それぞれの所有者の商標または登録商標です。

Intel、Intel Xeonは、Intel Corporationの商標または登録商標です。すべてのSPARCの商標はライセンスをもとに使用し、SPARC International, Inc.の商標または登録商標です。AMD、Opteron、AMDロゴ、AMD Opteronロゴは、Advanced Micro Devices, Inc.の商標または登録商標です。UNIXは、The Open Groupの登録商標です。

このソフトウェアまたはハードウェア、そしてドキュメントは、第三者のコンテンツ、製品、サービスへのアクセス、あるいはそれらに関する情報を提供することがあります。オラクル社およびその関連会社は、第三者のコンテンツ、製品、サービスに関して一切の責任を負わず、いかなる保証もいたしません。オラクル社およびその関連会社は、第三者のコンテンツ、製品、サービスへのアクセスまたは使用によって損失、費用、あるいは損害が発生しても一切の責任を負いかねます。

目次

はじめに	5
1 概要	9
メモリーエラー探索ツール (Discover)	9
コードカバレッジツール (Uncover)	10
2 メモリーエラー探索ツール (Discover)	11
Discover を使用するための要件	11
バイナリは正しく準備される必要がある	11
プリロードまたは監査を使用するバイナリは使用できない	12
標準メモリー割り当て関数を再定義するバイナリを使用できる	12
クイックスタート	13
準備されたバイナリの計測	14
共有ライブラリのキャッシュ	15
共有ライブラリの計測	15
ライブラリの無視	16
コマンド行オプション	16
bit.rc 初期化ファイル	19
SUNW_DISCOVER_OPTIONS 環境変数	19
SUNW_DISCOVER_FOLLOW_FORK_MODE 環境変数	19
計測済みバイナリの実行	20
Discover レポートの分析	20
HTML レポートの分析	20
ASCII レポートの分析	28
メモリーアクセスエラーと警告	31
メモリーアクセスエラー	31
メモリーアクセスの警告	34

Discover エラーメッセージの解釈	34
部分的に初期化されたメモリー	34
スペキュレイティブロード	35
未計測コード	36
Discover 使用時の制限事項	37
注釈付きコードのみが計測される	37
機械命令はソースコードとは異なる場合がある	37
コンパイラオプションは生成されたコードに影響を及ぼす	38
システムライブラリは報告されたエラーに影響を及ぼす可能性がある	38
カスタムメモリー管理はデータの正確さに影響を及ぼす可能性がある	39
静的および自動配列範囲外は削除できない	39
3 コードカバレッジツール (Uncover)	41
Uncover を使用するための要件	41
Uncover の使用法	42
バイナリの計測	42
計測済みバイナリの実行	43
カバレッジレポートの生成と表示	43
例	44
パフォーマンスアナライザのカバレッジレポートを理解する	45
「関数」タブ	45
「ソース」タブ	48
「逆アセンブリ」タブ	50
「命令頻度」タブ	51
ASCII カバレッジレポートを理解する	51
HTML カバレッジレポートを理解する	55
Uncover 使用時の制限事項	57
注釈付きコードのみ計測可能	57
機械命令はソースコードと異なる場合がある	58
索引	61

はじめに

『Oracle Solaris Studio 2.3 Discover および Uncover ユーザーズガイド』では、バイナリのメモリー関連のエラーを検出するメモリーエラー探索ツール (Discover)、およびアプリケーションのコードカバレッジを測定するコードカバレッジツール (Uncover) の使用方法について説明します。

サポートされるプラットフォーム

この Oracle Solaris Studio リリースは、Oracle Solaris オペレーティングシステムを実行する SPARC ファミリーのプロセッサアーキテクチャーを使用するプラットフォームと、Oracle Solaris または特定の Linux システムを実行する x86 ファミリーのプロセッサアーキテクチャーを使用するプラットフォームをサポートします。

このドキュメントでは、次の用語を使用して x86 プラットフォームの違いを示しています。

- 「x86」は、64 ビットおよび 32 ビットの x86 互換製品を指します。
- 「x64」は、特定の 64 ビット x86 互換 CPU を指します。
- 「32 ビット x86」は、x86 ベースシステムで特定の 32 ビット情報を指します。

Linux システムに固有の情報は、サポートされている Linux x86 プラットフォームだけに関連し、Oracle Solaris システムに固有の情報は、SPARC および x86 システムでサポートされている Oracle Solaris プラットフォームだけに関連します。

サポートされるハードウェアプラットフォームとオペレーティングシステムリリースの完全なリストについては、[Oracle Solaris Studio 12.3 リリースノート](#)を参照してください。

Oracle Solaris Studio ドキュメント

Oracle Solaris Studio ソフトウェアの完全なドキュメントは、次のように見つけることができます。

- 製品のドキュメントは、リリースノート、リファレンスマニュアル、ユーザーガイド、チュートリアルも含め、[Oracle Solaris Studio Documentation Web サイト](#)にあります。

- コードアナライザ、パフォーマンスアナライザ、スレッドアナライザ、dbxtool、DLight、およびIDEのオンラインヘルプには、これらのツール内の「ヘルプ」メニューだけでなく、F1キー、および多くのウィンドウやダイアログボックスにある「ヘルプ」ボタンを使用してアクセスできます。
- コマンド行ツールのマニュアルページでは、ツールのコマンドオプションが説明されています。

関連するサードパーティのWeb サイトリファレンス

このドキュメントには、詳細な関連情報を提供するサードパーティの URL が記載されています。

注- このドキュメントで紹介するサードパーティ Web サイトが使用可能かどうかについては、Oracle は責任を負いません。このようなサイトやリソース上、またはこれらを経由して利用できるコンテンツ、広告、製品、またはその他の資料についても、Oracle は保証しておらず、法的責任を負いません。また、このようなサイトやリソースから直接あるいは経由することで利用できるコンテンツ、商品、サービスの使用または依存が直接のあるいは関連する要因となり実際に発生した、あるいは発生するとされる損害や損失についても、Oracle は一切の法的責任を負いません。

開発者向けのリソース

Oracle Solaris Studio を使用する開発者のための次のリソースを見つけるには、[Oracle Technical Network Web サイト](#)にアクセスしてください。

- リソースは頻繁に更新されます。
- ソフトウェアの最近のリリースに関連する完全なドキュメントへのリンク
- サポートレベルに関する情報
- ユーザーディスカッションフォーラム。

Oracle サポートへのアクセス

Oracle のお客様は、My Oracle Support にアクセスして電子サポートを受けることができます。詳細は、<http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> にアクセス、または、聴覚に障害がある方は、<http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> にアクセスしてください。

表記上の規則

このマニュアルでは、次のような字体や記号を特別な意味を持つものとして使用します。

表 P-1 表記上の規則

字体または記号	意味	例
AaBbCc123	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コード例を示します。	.login ファイルを編集します。 ls -a を使用してすべてのファイルを表示します。 system%
AaBbCc123	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して示します。	system% su password:
AaBbCc123	変数を示します。実際に使用する特定の名前または値で置き換えます。	ファイルを削除するには、rm <i>filename</i> と入力します。
『 』	参照する書名を示します。	『コードマネージャ・ユーザーズガイド』を参照してください。
「 」	参照する章、節、ボタンやメニュー名、強調する単語を示します。	第 5 章「衝突の回避」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合に、継続を示します。	sun% grep '^#define \ XV_VERSION_STRING'

Oracle Solaris OS に含まれるシェルで使用する、UNIX のデフォルトのシステムプロンプトとスーパーユーザープロンプトを次に示します。コマンド例に示されるデフォルトのシステムプロンプトは、Oracle Solaris のリリースによって異なります。

- C シェル


```
machine_name% command y|n [filename]
```
- C シェルのスーパーユーザー


```
machine_name# command y|n [filename]
```
- Bash シェル、Korn シェル、および Bourne シェル


```
$ command y|n [filename]
```
- Bash シェル、Korn シェル、および Bourne シェルのスーパーユーザー

command y|n [*filename*]

[] は省略可能な項目を示します。上記の例は、*filename* は省略してもよいことを示しています。

| は区切り文字 (セパレータ) です。この文字で分割されている引数のうち 1 つだけを指定します。

キーボードのキー名は英文で、頭文字を大文字で示します (例: Shift キーを押します)。ただし、キーボードによっては Enter キーが Return キーの動作をします。

ダッシュ (-) は 2 つのキーを同時に押すことを示します。たとえば、Ctrl-D は Control キーを押したまま D キーを押すことを意味します。

概要

『Oracle Solaris Studio 12.3 Discover および Uncover ユーザーズガイド』では、以下のツールの使用方法の詳細について説明します。

- 9 ページの「メモリーエラー探索ツール (Discover)」
- 10 ページの「コードカバレッジツール (Uncover)」

メモリーエラー探索ツール (Discover)

メモリーエラー探索ツール (Discover) ソフトウェアは、メモリーアクセスエラーを検出するための高度な開発ツールです。Discover は、Sun Studio 12 Update 1、Oracle Solaris Studio 12.2、Oracle Solaris Studio 12.3 コンパイラ、または GCC for Sun Systems コンパイラのバージョン 4.2.0 以降を使用してコンパイルされたバイナリ上で機能します。Solaris 10 10/08 オペレーティングシステムまたはそれ以降の Solaris 10 update、または Oracle Solaris 11 を実行しているシステムで機能します。

プログラムのメモリー関連のエラーは、検出が難しいことで知られています。Discover を使用すると、ソースコードに存在している問題の正確な場所を指摘することによって、このようなエラーを簡単に検出できます。たとえば、プログラムが配列を割り当て、それを初期化せずに、ある配列の場所から読み取ろうとする場合、プログラムは動作が不安定になることがあります。Discover は、通常の方法でプログラムを実行するときに、この問題を検出できます。

Discover によって検出される他のエラーには、次のものがあります。

- 非割り当てメモリーからの読み取り、および非割り当てメモリーへの書き込み
- 割り当て済み配列範囲外のメモリーへのアクセス
- 解放されたメモリーの不正使用
- 不正なメモリーブロックの解放
- メモリーリーク

Discover はプログラムの実行中にメモリアクセスエラーを動的に検出して報告するため、ユーザーコードの一部が実行時に行われていない場合、その部分のエラーは報告されません。

Discover は簡単に使用できます。コンパイラによって準備されたすべてのバイナリは (完全に最適化されたバイナリでも)、単一のコマンドを使用して計測し、通常の方法で実行できます。実行中に、Discover は、メモリー異常のレポートを生成し、それをテキストファイル、または Web ブラウザで HTML 形式で表示できます。

コードカバレッジツール (Uncover)

Uncover は、簡単にコマンドラインツールを使用して、アプリケーションのコードカバレッジを計測できます。コードカバレッジは、ソフトウェアのテストの重要な部分です。Uncover はテストで実行される、または実行されないコードの領域に関する情報を提供し、テストスイートを向上させ、より多くのコードをテストできるようにします。Uncover で報告されるカバレッジ情報は、関数、文、基本ブロック、または命令レベルとすることができます。

Uncover は、カバレッジ外と呼ばれる一意の機能を提供し、テストされない主要な機能領域をすばやく検出できます。他の種類の計測より優れた Uncover コードカバレッジの他の利点は、次のとおりです。

- 未計測コードに関連する遅延がかなり少ない。
- Uncover はバイナリ上で動作するため、最適化されたバイナリと併用できる。
- 出荷用バイナリを計測することによって測定が可能である。アプリケーションはカバレッジテスト用に異なる方法で構築する必要がない。
- Uncover は、バイナリの計測、テストの実行、および結果の表示を行うための簡単な手順を提供する。
- Uncover は、マルチスレッドおよびマルチプロセスに対して安全である。

メモリーエラー探索ツール (Discover)

メモリーエラー探索ツール (Discover) ソフトウェアは、メモリーアクセスエラーを検出するための高度な開発ツールです。

この章には、次の情報が含まれます。

- 11 ページの「Discover を使用するための要件」
- 13 ページの「クイックスタート」
- 14 ページの「準備されたバイナリの計測」
- 20 ページの「計測済みバイナリの実行」
- 20 ページの「Discover レポートの分析」
- 31 ページの「メモリーアクセスエラーと警告」
- 34 ページの「Discover エラーメッセージの解釈」
- 37 ページの「Discover 使用時の制限事項」

Discover を使用するための要件

バイナリは正しく準備される必要がある

Discover は、Sun Studio 12 Update 1、Oracle Solaris Studio 12.2、Oracle Solaris Studio 12.3 コンパイラ、または GCC for Sun Systems コンパイラのバージョン 4.2.0 以降を使用してコンパイルされたバイナリ上で機能します。Solaris 10 10/08 オペレーティングシステムまたはそれ以降の Solaris 10 update、または Oracle Solaris 11 を実行している、SPARC ベースシステムまたは x86 ベースシステムで機能します。

これらの要件が満たされない場合、Discover でエラーが発生するか、またはバイナリが計測されません。ただし、`-l` オプション (17 ページの「計測オプション」を参照) を使用することによって、これらの要件を満たさないバイナリを計測し、それを実行して限定された数のエラーを検出できます。

前述のようにコンパイルされたバイナリには、注釈と呼ばれる情報が含まれ、Discover がバイナリを正しく計測するのに役立っています。このわずかな情報が追加されることで、バイナリのパフォーマンスまたは実行時のメモリー使用量に影響を及ぼすことはありません。

バイナリのコンパイル時に `-g` オプションを使用してデバッグ情報を生成することにより、Discover はエラーおよび警告を報告しながらソースコードおよび行番号情報を表示し、より正確な結果を生成することができます。バイナリが `-g` オプションを使用してコンパイルされない場合、Discover には、対応する機械レベルの命令のプログラムカウンタのみが表示されます。また、`-g` オプションを使用してコンパイルすることにより、Discover はより正確なレポートを生成できます (34 ページの「[Discover エラーメッセージの解釈](#)」を参照)。

プリロードまたは監査を使用するバイナリは使用できません

Discover は実行時リンカーの一部の特定の機能を使用するため、プリロードまたは監査を使用するバイナリと併用することはできません。

プログラムが `LD_PRELOAD` 環境変数の設定を必要とする場合は、Discover を使用して適切に機能しない可能性があります。それは Discover は特定のシステム関数に割り込む必要があり、関数がプリロードされている場合は割り込めないためです。

同様に、プログラムが実行時監査を使用している (バイナリが `-p` オプションまたは `-P` オプションとリンクされているか、`LD_AUDIT` 環境変数を設定する必要がある) 場合、この監査は Discover の監査の使用と衝突します。バイナリが監査とリンクされている場合、Discover は計測時に失敗します。実行時に `LD_AUDIT` 環境変数を設定している場合、結果は定義されません。

標準メモリー割り当て関数を再定義するバイナリを使用できる

Discover では、標準メモリー割り当て関数 `malloc()`、`calloc()`、`memalign()`、`valloc()`、および `free()` を再定義するバイナリがサポートされます。

クイックスタート

次の内容は、プログラムを準備し、Discover を使用して計測を行い、それを実行して、検出したメモリアクセスエラーに関するレポートを生成する例です。この例は初期化されていないデータにアクセスする単純なプログラムを使用します。

```
% cat test_UMR.c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    // UMR: accessing uninitialized data
    int *p = (int*) malloc(sizeof(int));
    printf("*p = %d\n", *p);
    free(p);
}

% cc -g -O2 test_UMR.c
% a.out
*p = 131464
% discover a.out
% a.out
```

Discover 出力は、初期化されていないメモリーが使用された場所、およびそのメモリーが割り当てられた場所を、結果の概要とともに示します。

込みます。バイナリを計測してレポートを ASCII ファイルまたは stderr に書き込むように要求する場合は、`-w` オプションを使用できます。

`-n` オプションを使用すると、Discover でバイナリに書き込み専用の計測を行うように指定できます。

Discover がバイナリを計測する際に、注釈が付けられていないために計測できないコードを検出する場合、次のような警告が表示されます。

```
discover: (warning): a.out: 80% of code instrumented (16 out of 20 functions)
```

注釈付きではないコードは、バイナリにリンクされているアセンブリ言語コード、またはコンパイラでコンパイルされたモジュール、または [11 ページの「バイナリは正しく準備される必要がある」](#) にリストされているシステムより古いオペレーティングシステム上から来ている可能性があります。

共有ライブラリのキャッシュ

Discover がバイナリを計測する際には、コードを追加し、実行時リンカーを使用して、実行時にロードされる場合には依存共有ライブラリを計測できるようにします。計測済みライブラリは、元のライブラリが最後に計測されてから変更されていない場合には再使用可能なキャッシュに格納されます。デフォルトでは、キャッシュディレクトリは `$HOME/SUNW_Bit_Cache` です。このディレクトリは `-D` オプションを使用して変更できます。

共有ライブラリの計測

すべての共有ライブラリを含む、プログラム全体が計測される場合、Discover は最も正確な結果を生成します。デフォルトでは、Discover は実行可能ファイルのメモリーエラーだけを検査して報告します。`-c` オプションを使用すると、依存共有ライブラリおよび `dlopen()` によって動的に開かれたライブラリのエラーを Discover で検査するように指定できます。`-n` オプションを使用すると、Discover で実行可能ファイルのエラーの検査をスキップするように指定できます。

`-c` オプションを使用して特定のライブラリのエラー検査を無効にすると、Discover はそのライブラリのエラーを報告しません。ただし、Discover はメモリーエラーを正しく検出するためにアドレス空間全体のメモリー状態を追跡する必要があるため、すべての共有ライブラリも含めたプログラム全体で割り当てとメモリーの初期化を記録します。

プログラムで使用されるすべての共有ライブラリは、[11 ページの「バイナリは正しく準備される必要がある」](#) で説明されているように準備される必要があります。デフォルトで、実行時リンカーが準備されていないライブラリを検出する場合、致命的なエラーが発生します。ただし、Discover に 1 つ以上のライブラリを無視するように指示できます。

ライブラリの無視

一部のライブラリは、準備できないか、または何らかの理由で計測できない場合があります。このような場合に、多少、正確さを低下させるために、`-s`、`-T`、または `-N` オプション（「17 ページの「計測オプション」」を参照）、`bit.rc` ファイル（「19 ページの「`bit.rc` 初期化ファイル」」を参照）の仕様を使用して、Discover にこれらのライブラリを無視するように指示できます。

ライブラリが計測できず、「無視可能」と指定されていない場合、Discover は計測時に失敗するか、またはプログラムが実行時にエラーメッセージを伴って失敗します。

デフォルトで、Discover はシステムの `bit.rc` ファイルの仕様を使用して、特定のシステムおよびコンパイラが提供するライブラリを、準備されていないため「無視可能」として設定します。Discover は最も一般的に使用されるライブラリのメモリー特性を知っているため、正確さに対する影響は最小限です。

コマンド行オプション

`discover` コマンドとともに次のオプションを使用して、バイナリを計測できます。

出力オプション

- `-a` コードアナライザで使用するためにエラーデータを `binary_name.analyze/dynamic` ディレクトリに書き込みます。
- `-b browser` 計測機構の組み込まれたプログラムを実行するとき、Web ブラウザ `browser` を自動的に起動します (デフォルトでは off)。
- `-o file` 計測済みのバイナリを `file` に書き込みます。デフォルトで、計測済みのバイナリは入力バイナリを上書きします。
- `-w text_file` バイナリ上の Discover のレポートを `text_file` に書き込みます。計測済みのバイナリを実行するときに、ファイルが作成されます。`text_file` が相対パス名である場合、ファイルは計測済みバイナリを実行する作業ディレクトリを基準として相対的に配置されます。バイナリを実行するたびにファイル名を一意にするには、文字列 `%p` をファイル名に追加して、Discover ランタイムにプロセス ID を含めるように求めます。たとえば、オプション `-w report.%p.txt` は `report.process_id.txt` を持つレポートファイルを生成します。2 回以上、ファイル名に `%p` を含む場合、最初のインスタンスだけがプロセス ID と置き換えられます。

このオプションまたは `-H` オプションを指定しない場合、レポートは HTML 形式で `output_file.html` に書き込まれます。`output_file` は、計測済みバイナリのベース名です。ファイルは、計測済みバイナリを実行する作業ディレクトリに配置されます。

このオプションおよび `-H` オプションを両方指定して、テキストおよび HTML 形式の両方でレポートを書き込みます。

- `-H html_file` Discover のバイナリのレポートを HTML 形式で *html_file* に書き込みます。このファイルは計測済みバイナリの実行時に作成されます。*html_file* が相対パス名である場合、計測済みバイナリを実行する作業ディレクトリを基準として相対的に配置されます。バイナリを実行するたびにファイル名を一意にするには、文字列 `%p` をファイル名に追加して、Discover ランタイムにプロセス ID を含めるように求めます。たとえば、オプション `-H report.%p.html` は、ファイル名 `report.process_id.html` を持つレポートファイルを生成します。2 回以上、ファイル名に `%p` を含む場合、最初のインスタンスだけがプロセス ID と置き換えられます。

このオプションまたは `-w` オプションを指定しない場合、レポートは HTML 形式で `output_file.html` に書き込まれます。*output_file* は計測済みバイナリのベース名です。ファイルは、計測済みバイナリを実行する作業ディレクトリに配置されます。

このオプションおよび `-w` オプションを指定して、テキストおよび HTML ファイル形式の両方でレポートを書き込むことができます。

- `-e n` レポートに *n* メモリーエラーのみを表示します (デフォルトでは、すべてのエラーを表示します)。
- `-E n` レポートに *n* メモリーリークのみを表示します (デフォルトは 100 です)。
- `-f` レポートのオフセットを表示します (デフォルトは非表示です)。
- `-m` レポートの符号化された名前を表示します (デフォルトは符号化されていない名前の表示です)。
- `-S n` レポートに *n* スタックフレームのみを表示します (デフォルトは 8 です)。

計測オプション

- `-c [- | library | file]` すべてのライブラリ、指定された *library*、または指定された *file* に列挙されているライブラリのエラーを検査します。
- `-n` 実行可能ファイルのエラーを検査しません。
- `-l` Discover を簡易モードで実行します。このオプションは、プログラムのより高速な実行を提供し、11 ページの「[バイナリは正しく準備される必要がある](#)」で説明されるように、プログラムが特別に準備される必要はないが検出されるエラー数は制限されます。

- F [parent | child] Discover で計測機構を組み込んだバイナリが実行中にフォークした場合に行う処理を指定します。デフォルトでは、Discover は親プロセスからメモリアクセスエラーのデータを収集し続けます。Discover でフォークを追尾し、子プロセスからメモリアクセスデータを収集するには、-F child を指定します。
- i スレッドアナライザを使用してデータ競合を検出するために計測します。このオプションを使用する場合は、データ競合検出のみが実行時に行われ、他のメモリー検査は行われません。計測機構付きバイナリを collect コマンドを使用して実行し、パフォーマンスアナライザで表示可能な実験を生成する必要があります(『Oracle Solaris Studio 12.3: スレッドアナライザ ユーザーズガイド』を参照)。
- s 計測不可能なバイナリの計測を試みる場合は、警告を発するが、エラーのフラグは立てないでください。
- T 指定されたバイナリのみを計測します。依存共有ライブラリを実行時に計測しないでください。
- N library 接頭辞 *library* に一致する依存共有ライブラリを計測しないでください。ライブラリ名の最初の文字が *library* に一致する場合、ライブラリは無視されます。*library* が / で始まる場合、ライブラリの完全な絶対パス名でマッチングが行われます。それ以外の場合、ライブラリのベース名でマッチングが行われま
- K bit.rc 初期化ファイルを読み取らないでください(「19 ページの「bit.rc 初期化ファイル」」を参照)。

キャッシュオプション

- D *cache_directory* キャッシュされた計測済みバイナリを格納するためのルートディレクトリとして *cache_directory* を使用します。デフォルトでは、キャッシュディレクトリは \$HOME/SUNW_Bit_Cache です。
- k キャッシュで検出されたライブラリの再計測を強制します。

その他のオプション

- h または -? ヘルプ。短いヘルプメッセージを出力して、終了します。
- v 冗長。Discover が実行している内容のログを出力します。詳細については、オプションを繰り返してください。
- V Discover バージョン情報を出力して終了します。

bit.rc 初期化ファイル

Discoverは、起動時に一連のbit.rc ファイルを読み取ることによってその状態を初期化します。システムファイル、*Oracle_Solaris_Studio_installation_directory* /prod/lib/postopt/bit.rc は、特定の変数のデフォルトの値を提供します。Discover は最初にこのファイルを読み取り、次に、存在する場合は \$HOME/.bit.rc、および存在する場合は *current_directory*/.bit.rc を読み取ります。

bit.rc ファイルには特定の変数を設定、追加、または削除するコマンドが含まれています。Discover が set コマンドを読み取る場合、変数の前の値がある場合には、それを無効にします。append コマンドを読み取る場合、変数の既存の値に(コロンセパレータの後に)引数を追加します。remove コマンドを読み取る場合、変数の既存の値から引数とそのコロンセパレータを削除します。

bit.rc ファイルの変数セットには、計測時に無視するライブラリのリスト、およびバイナリ内の注釈の付いていない(準備されていない)コードの割合を計算する場合に無視する関数または関数接頭語のリストが含まれます。

詳細については、システム bit.rc ファイルのヘッダーのコメントを参照してください。

SUNW_DISCOVER_OPTIONS 環境変数

SUNW_DISCOVER_OPTIONS 環境変数をコマンド行オプション

-b、-e、-E、-f、-F、-H、-l、-L、-m、-s、および -w のリストに設定することによって、計測機構付きバイナリの実行時動作を変更できます。たとえば、レポートされるエラー数を 50 に変更し、レポート内のスタックの深さを 3 に制限する場合、環境変数を -e 50-s 3 に設定します。

SUNW_DISCOVER_FOLLOW_FORK_MODE 環境変数

デフォルトでは、Discover で計測機構を組み込んだバイナリが実行中にフォークした場合、Discover は親プロセスからメモリアクセスエラーのデータを収集し続けます。Discover でフォークを追尾し、子プロセスからメモリアクセスデータを収集するには、SUNW_DISCOVER_FOLLOW_FORK_MODE 環境変数を設定します。

計測済みバイナリの実行

Discover を使用してバイナリを計測した後で、それを通常の場合と同じ方法で実行します。通常、特定の組み合わせの入力により、プログラムが別の動作を行う場合、そのプログラムを Discover を使用して計測し、同じ入力を使用して実行して、潜在的なメモリーの問題を調べます。計測済みのプログラムの実行中に、Discover は、選択した形式 (テキスト、HTML、またはその両方) の指定された出力ファイルに、検出されるメモリーエラーに関する情報を書き込みます。レポートの解釈に関する詳細については、「[20 ページの「Discover レポートの分析」](#)」を参照してください。

計測のオーバーヘッドのため、プログラムは計測後に大幅に低速に実行されます。メモリーアクセスの頻度に応じて、50 倍も低速に実行される場合があります。

Discover レポートの分析

Discover レポートは、ソースコードで効果的に自動補完して問題を修正する情報を提供します。

デフォルトでは、レポートは `output_file.html` に HTML 形式で書き込まれます。`output_file` は、計測済みバイナリのベース名です。ファイルは、計測済みバイナリを実行する作業ディレクトリに配置されます。

バイナリを計測するには、`-H` オプションを使用して、HTML 出力を指定されたファイルに書き込むように要求するか、または `-w` オプションを使用して、テキストファイルに書き込むように要求できます ([16 ページの「コマンド行オプション」](#)を参照)。

バイナリの計測後に、たとえば、プログラムを後続の実行用に異なるファイルにレポートを書き込みたい場合に、[19 ページの「SUNW_DISCOVER_OPTIONS 環境変数」](#) でレポートの `-H` および `-w` オプションの設定を変更できます。

HTML レポートの分析

HTML レポート形式では、プログラムの対話型分析が可能です。HTML 形式のデータは、電子メールを使用するか、Web ページ上に配置して、開発者間で容易に共有できます。JavaScript インタラクティブ機能と組み合わせると、Discover のメッセージを検索する便利な方法が提供されます。

「エラー (Errors)」タブ ([21 ページの「「エラー \(Errors\)」タブの使用法」](#)を参照)、[「警告 \(Warnings\)」タブ \(24 ページの「「警告 \(Warnings\)」タブの使用法」を参照\)](#)、および「メモリーリーク (Memory Leaks) タブ」 ([25 ページの「「メモ](#)

リーク (Memory Leaks)」タブの使用方法」を参照)では、エラーメッセージ、警告メッセージ、およびメモリーリークレポートをそれぞれ検索できます。

左側のコントロールパネル(27ページの「コントロールパネルの使用方法」を参照)では、右側に現在表示されているタブの内容を変更できます。

「エラー (Errors)」タブの使用方法

ブラウザで最初に HTML レポートを開く場合、「エラー (Errors)」タブが選択され、計測済みのバイナリの実行中に検出されたメモリアクセスエラーのリストが表示されます。

The screenshot displays the Oracle Solaris Studio 12.3 Discover interface. On the left, there are three main sections: 'Stack Trace' with 'Expand all' and 'Collapse all' buttons; 'Source Code' with 'Expand All' and 'Collapse All' buttons; and 'Show Errors' with a grid of checkboxes for various error types: ABR, ABV, BFM, BRP, CGB, DFM, FMR, FMW, FRP, IMR, IMW, IR, SSR, SBW, UAR, UAW, and UMR. The 'UAW' and 'UMR' checkboxes are checked. At the bottom left, a 'Summary' box shows 'Errors: 2', 'Warnings: 1', and 'Leaked: 4 Bytes'. On the right, the 'Errors' tab is active, showing two error messages: 'UMR: accessing uninitialized data from address 0x50010 (4 bytes)' and 'UAW: writing to unallocated memory at address 0x50010 (4 bytes)'. The 'Warnings' and 'Memory Leaks' tabs are also visible but inactive.

エラーをクリックすると、エラー時のスタックトレースが表示されます。

The screenshot shows the Discover tool interface. On the left is a sidebar with the following sections:

- Stack Trace**: Expand all, Collapse all
- Source Code**: Expand All, Collapse All
- Show Errors**: A list of error types with checkboxes: ABR, ABW, BFM, BRP, CGB, DFM, FMR, FMW, FRP, IHR, IMV, IIR, SBR, SBW, UAR, UAW (checked), UMR (checked).
- Summary**: Errors: 2, Warnings: 1, Leaked: 4 Bytes

The main area has tabs for Errors, Warnings, and Memory Leaks. The Errors tab is active, showing:

- UMR: accessing uninitialized data from address 0x50010 (4 bytes)**
- Stack trace snippet:

```
main() + 0xccc (line -9) in "test_UMR.c"  
_start() + 0x108  
was allocated at (4 bytes):  
main() + 0x1c (line -8) in "test_UMR.c"  
_start() + 0x108
```
- UAW: writing to unallocated memory at address 0x50018 (4 bytes)**

At the bottom left of the sidebar, it says: Copyright © 2007, 2010, Oracle and/or its affiliates. All rights reserved.

-g オプションを使用してコードをコンパイルした場合、関数をクリックすることによってスタックトレースの関数ごとのソースコードを表示できます。

Stack Trace
Expand all
Collapse all

Source Code
Expand All
Collapse All

Show Errors

- ABR
- ABW
- BFM
- BRP
- CGB
- DFM
- FMR
- FMW
- FRP
- IMR
- IPW
- IIR
- SBR
- SBW
- UAR
- UAW
- UMR

Summary

Errors: 2
Warnings: 1
Leaked: 4 Bytes

Copyright © 2009, 2010, Oracle and/or its affiliates. All rights reserved.

Errors Warnings Memory Leaks

UMR: accessing uninitialized data from address 0x50010 (4 bytes)

main() + 0x0c (line -9) in "test_UMR.c"
_start() + 0x100

was allocated at (4 bytes):
main() + 0x1c (line -8) in "test_UMR.c"

```

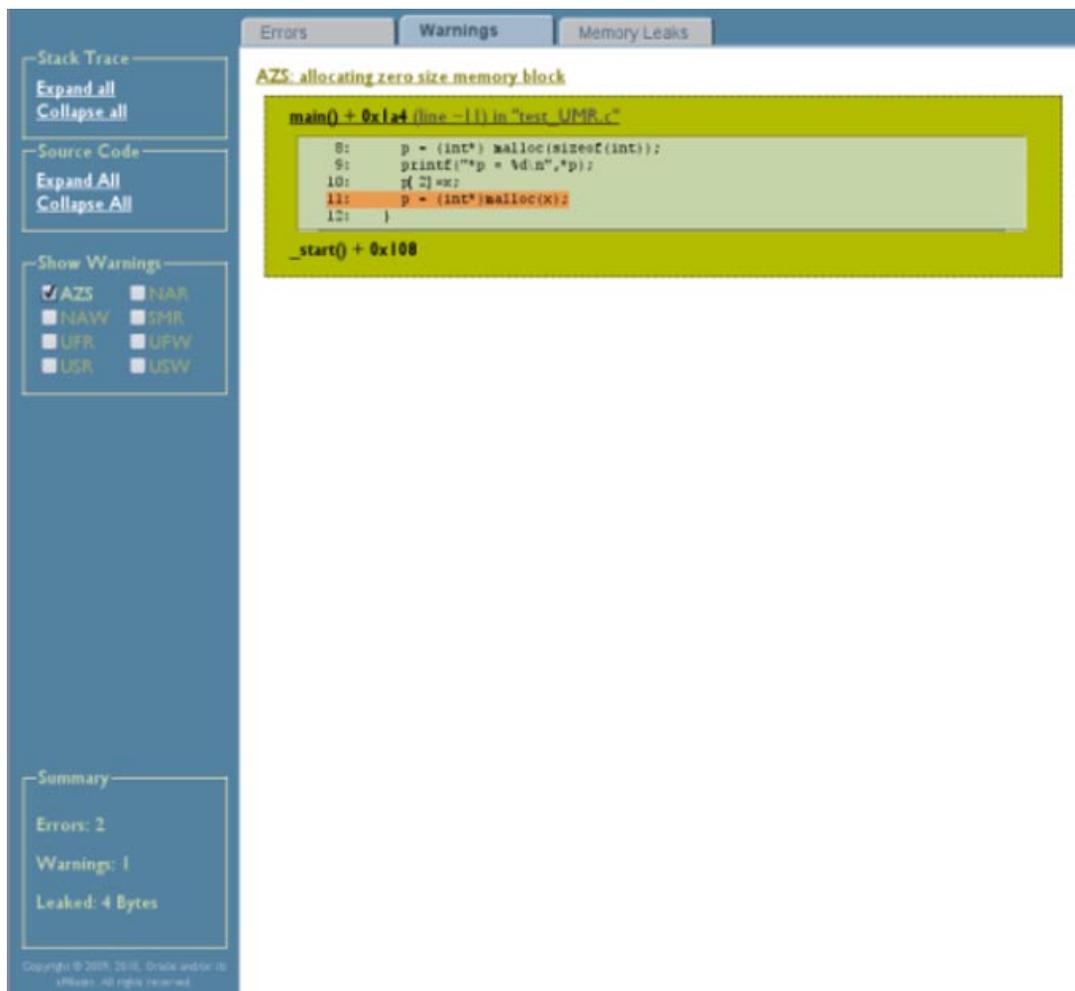
5: int main()
6: {
7: // UMR: accessing uninitialized data
8: p = (int*) malloc(sizeof(int));
9: printf("p = %d\n", *p);
10: printf("%x", *p);
11: p = (int*) malloc(x);
_start() + 0x100

```

UAW: writing to unallocated memory at address 0x50018 (4 bytes)

「警告 (Warnings)」 タブの使用法

「警告 (Warnings)」タブには、起こり得るアクセスエラーの警告メッセージのすべてが表示されます。警告をクリックすると、警告時のスタックトレースが表示されます。コードを `-g` オプションを使用してコンパイルした場合、関数をクリックすることによって、スタックトレースの関数ごとのソースコードを表示できます。



The screenshot displays the Discover tool's interface. On the left, there are three panels: 'Stack Trace' with 'Expand all' and 'Collapse all' buttons; 'Source Code' with 'Expand All' and 'Collapse All' buttons; and 'Show Warnings' with a grid of checkboxes for various warning types (AZS, NAW, UFR, USR, NAR, SMP, UPW, USW). The 'AZS' checkbox is checked. Below these is a 'Summary' panel showing 'Errors: 2', 'Warnings: 1', and 'Leaked: 4 Bytes'. At the bottom left, there is a copyright notice: 'Copyright © 2009, 2010, Oracle and/or its affiliates. All rights reserved.'

The main area shows the 'Warnings' tab selected. The warning title is 'AZS: allocating zero size memory block'. The warning details are as follows:

```
main() + 0x1a4 (line -11) in "test_UMR.c"
8:   p = (int*) malloc(sizeof(int));
9:   printf("p = %d\n", *p);
10:  p[2] = x;
11:  p = (int*)malloc(x);
12:  }
```

The warning is associated with the address '_start() + 0x100'.

「メモリーリーク (Memory Leaks)」 タブの使用方法

「メモリーリーク (Memory Leaks)」タブには、下記にリストされているブロック数とともに、最上部にプログラムの実行終了時に割り当てられている残存ブロック総数が表示されます。

The screenshot shows the Oracle Solaris Studio Discover interface with the 'Memory Leaks' tab selected. The main area displays two memory leak entries:

- 1 block at 1 location left allocated on heap with total size of 4 bytes
- 1 block with total size of 4 bytes

On the left side, there are two expandable sections:

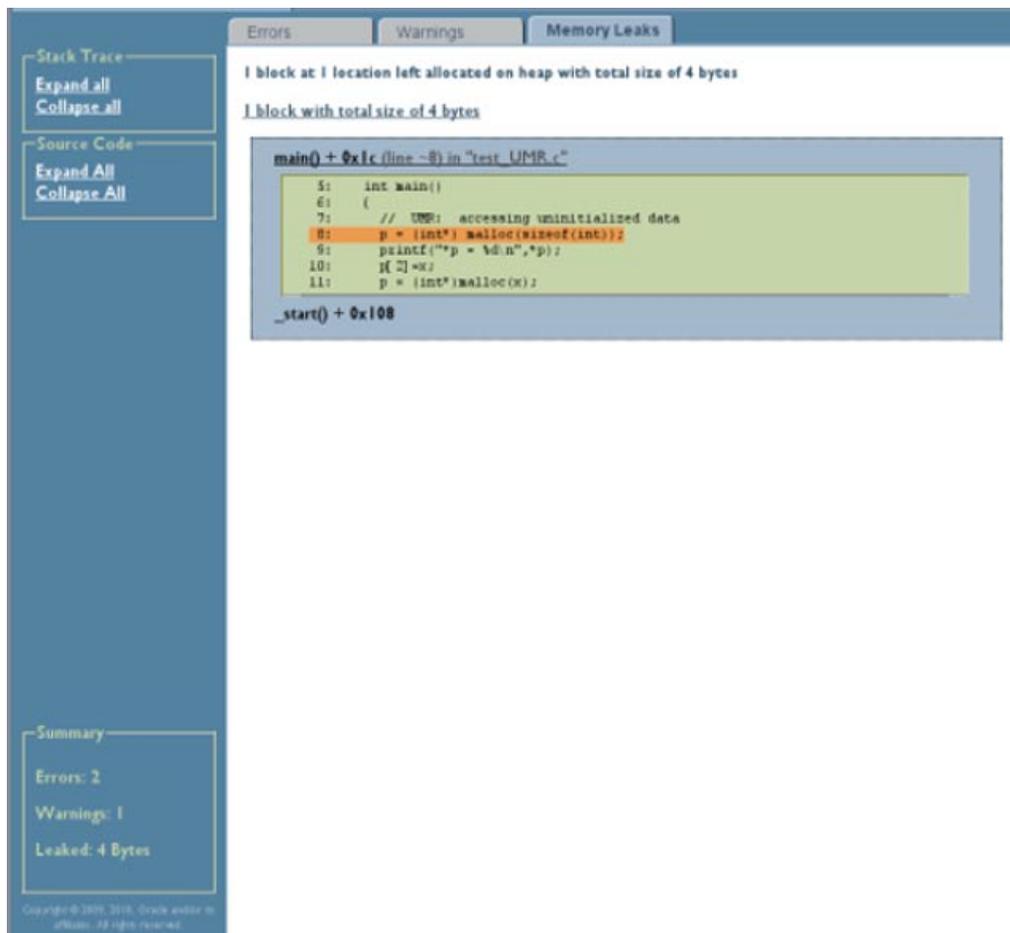
- Stack Trace**: Expand all, Collapse all
- Source Code**: Expand All, Collapse All

At the bottom left, a **Summary** box shows:

- Errors: 2
- Warnings: 1
- Leaked: 4 Bytes

Copyright © 2007, 2013, Oracle and/or its affiliates. All rights reserved.

ブロックをクリックすると、ブロックのスタックトレースが表示されます。-g オプションを使用してコードをコンパイルした場合、関数をクリックすることによって、スタックトレースの関数ごとのソースコードを表示できます。



コントロールパネルの使用方法

エラー、警告、およびメモリーリークのすべてのスタックトレースを表示するには、コントロールパネルの「スタックトレース (Stack Traces)」セクションの「すべて展開 (Expand All)」をクリックします。関数のすべてのソースコードを表示するには、コントロールパネルの「ソースコード (Source Code)」セクションの「すべて展開 (Expand All)」をクリックします。

エラー、警告、およびメモリーリークのすべてのスタックトレースまたはソースコードを非表示にするには、対応する「すべて折りたたむ (Collapse All)」をクリックします。

コントロールパネルの「エラーの表示 (Show Errors)」セクションは、「エラー (Errors)」タブが選択され、表示されるエラーのタイプを制御できる場合に表示され

ます。デフォルトでは、検出されたエラーのすべてのチェックボックスがオンになっています。エラーのタイプを非表示にするには、チェックボックスをクリックして、チェックマークを外します。

コントロールパネルの「警告の表示 (Show Warnings)」セクションは、「警告 (Warnings)」タブが選択され、表示される警告のタイプを制御できる場合に表示されます。デフォルトで、検出された警告のすべてのチェックボックスがオンになっています。警告のタイプを非表示にするには、チェックボックスをクリックして、チェックマークを外します。

エラーおよび警告の総数を一覧表示するレポートの概要、およびリークしたメモリー量がコントロールパネルの下方に表示されます。

ASCII レポートの分析

Discover レポートの ASCII (テキスト) 形式は、スクリプトで処理したり、Web ブラウザにアクセスできない場合に適しています。次に示すのは ASCII レポートの例です。

```
$ a.out
```

```
ERROR 1 (UAW): writing to unallocated memory at address 0x50088 (4 bytes) at:
```

```
main() + 0x2a0 <ui.c:20>
17:   t = malloc(32);
18:   printf("hello\n");
19:   for (int i=0; i<100;i++)
20:=>   t[32] = 234; // UAW
21:   printf("%d\n", t[2]); //UMR
22:   foo();
23:   bar();
_start() + 0x108
```

```
ERROR 2 (UMR): accessing uninitialized data from address 0x50010 (4 bytes) at:
```

```
main() + 0x16c <ui.c:21>$
18:   printf("hello\n");
19:   for (int i=0; i<100;i++)
20:   t[32] = 234; // UAW
21:=> printf("%d\n", t[2]); //UMR
22:   foo();
23:   bar();
24: }
```

```
_start() + 0x108
was allocated at (32 bytes):
```

```
main() + 0x24 <ui.c:17>
14:   x = (int*)malloc(size); // AZS warning
15: }
16: int main() {
17:=>   t = malloc(32);
18:   printf("hello\n");
19:   for (int i=0; i<100;i++)
20:   t[32] = 234; // UAW
_start() + 0x108
```

```

0
WARNING 1 (AZS): allocating zero size memory block at:
foo() + 0xf4 <ui.c:14>
11: void foo() {
12:   x = malloc(128);
13:   free(x);
14:=> x = (int*)malloc(size); // AZS warning
15: }
16: int main() {
17:   t = malloc(32);
main() + 0x18c <ui.c:22>
19:   for (int i=0; i<100;i++)
20:     t[32] = 234; // UAW
21:   printf("%d\n", t[2]); //UMR
22:=>   foo();
23:   bar();
24: }
_start() + 0x108

***** Discover Memory Report *****

1 block at 1 location left allocated on heap with a total size of 128 bytes

1 block with total size of 128 bytes
bar() + 0x24 <ui.c:9>
6:   7: void bar() {
8:     int *y;
9:=>   y = malloc(128); // Memory leak
10:  }
11: void foo() {
12:   x = malloc(128);
main() + 0x194 <ui.c:23>
20:   t[32] = 234; // UAW
21:   printf("%d\n", t[2]); //UMR
22:   foo();
23:=>   bar();
24: }
_start() + 0x108

ERROR 1: repeats 100 times
DISCOVER SUMMARY:
  unique errors   : 2 (101 total, 0 filtered)
  unique warnings : 1 (1 total, 0 filtered)

```

このレポートはエラーと警告メッセージ、およびその概要で構成されます。

エラーメッセージは、ERROR という単語で開始され、3文字のコード、ID番号、およびエラーの説明(例では、writing to unallocated memory)が含まれます。その他の詳細には、アクセスされたメモリーアドレス、および読み取られた、または書き込まれた数またはバイト数が含まれます。説明の後には、プロセスライフサイクルでエラーの場所を自動補完するエラー時のスタックトレースが表示されます。

プログラムが `-g` オプションを使用してコンパイルされた場合、スタックトレースには、ソースファイルおよび行番号が含まれます。ソースファイルにアクセス可能な場合、エラー付近のソースコードが出力されます。各フレームのターゲットソース行は `=>` シンボルによって示されます。

同じバイト数を持つ同じメモリーの場所の同じエラーの種類が繰り返される場合、スタックトレースを含む完全なメッセージが1度だけ出力されます。後続のエラーの出現が数えられ、次の例に表示されるように繰り返し数が、複数回発生する同一エラーごとにレポートの末尾に一覧表示されます。

```
ERROR 1: repeats 100 times
```

不正なメモリーアクセスのアドレスがヒープ上にある場合、対応するヒープブロックに関する情報がスタックトレース後に出力されます。その情報には、ブロック開始アドレスとサイズ、およびブロックが割り当てられた時点のスタックトレースが含まれます。ブロックが解放された場合、解放ポイントのスタックトレースも含まれます。

警告メッセージは、`WARNING`という単語で開始されている場合を除いて、エラーメッセージと同じ形式で出力されます。概して、これらのメッセージは、アプリケーションの正確さに影響を及ぼさない状態に対して警告しますが、問題を改善するために使用可能な役立つ情報を提供します。たとえば、0サイズのメモリーを割り当てては有害ではありませんが、頻繁すぎると、パフォーマンスを低下させる可能性があります。

メモリーリークレポートには、ヒープ上に割り当てられているがプログラムの終了時にリリースされないメモリーブロックに関する情報が含まれます。次に示すのは、メモリーリークレポートの例です。

```
$ DISCOVER_MEMORY_LEAKS=1 ./a.out
...
***** Discover Memory Report *****

2 blocks left allocated on heap with total size of 44 bytes
  block at 0x50008 (40 bytes long) was allocated at:
    malloc() + 0x168 [libdiscover.so:0xea54]
    f() + 0x1c [a.out:0x3001c]
    <discover_example.c:9>:
      8:   {
      9:=>   int *a = (int *)malloc( n * sizeof(int) );
     10:   int i, j, k;
main() + 0x1c [a.out:0x304a8]
    <discover_example.c:33>:
     32:   /* Print first N=10 Fibonacci numbers */
     33:=>   a = f(N);
     34:   printf("First %d Fibonacci numbers:\n", N);
_start() + 0x5c [a.out:0x105a8]
...

```

ヘッダーに続く最初の行は、ヒープ上に割り当てられて残されているヒープブロック数とその合計サイズを要約しています。レポートされるサイズは、開発者の見解であり、すなわちメモリアロケータのブックキーピングのオーバーヘッドは含まれません。

メモリーリークの概要の後に、割り当てポイントのスタックトレースを持つ未解放ヒープブロックごとの詳細情報が出力されます。スタックトレースレポートは、エラーおよび警告メッセージに対して説明されるレポートと同様です。

Discover レポートには概要全体が記載されています。かっこ付きの一意の警告およびエラー数、繰り返しを含むエラーおよび警告の総数を報告します。例:

```
DISCOVER SUMMARY:
    unique errors   : 3 (3 total)
    unique warnings : 1 (5 total)
```

メモリアクセスエラーと警告

Discover は多数のメモリアクセスエラー、およびエラーである可能性のあるアクセスに関する警告を検出および報告します。

メモリアクセスエラー

Discover は次のメモリアクセスエラーを検出します。

- ABR: 配列範囲外からの読み取り (beyond Array Bounds Read)
- ABW: 配列範囲外への書き込み (beyond Array Bounds Write)
- BFM: 不正な空きメモリー (Bad Free Memory)
- BRP: 不正な realloc アドレスパラメータ (Bad Realloc address Parameter)
- CGB: 破壊された配列ガードブロック (Corrupted array Guard Block)
- DFM: メモリーの二重解放 (Double Freeing Memory)
- FMR: 解放されたメモリーからの読み取り (Freed Memory Read)
- FMW: 解放されたメモリーへの書き込み (Freed Memory Write)
- FRP: 解放された realloc パラメータ (Freed Realloc Parameter)
- IMR: 無効なメモリーからの読み取り (Invalid Memory Read)
- IMW: 無効なメモリーへの書き込み (Invalid Memory Write)
- メモリーリーク
- OLP: 送り側と受け側の重複 (OverLaPping source and destination)
- PIR: 部分的に初期化された領域からの読み取り (Partially Initialized Read)
- SBR: スタックフレームの範囲外からの読み取り (beyond Stack frame Bounds Read)
- SBW: スタックフレームの範囲外への書き込み (beyond Stack frame Bounds Write)
- UAR: 非割り当てメモリーからの読み取り (UnAllocated memory Read)
- UAW: 非割り当てメモリーへの書き込み (UnAllocated memory Write)

- UMR: 非初期化メモリーからの読み取り (Unitialized Memory Read)

次のセクションに、これらのエラーの一部を生成する簡単なサンプルプログラムを一覧表示します。

ABR

```
// ABR: reading memory beyond array bounds at address 0x%lx (%d byte%s)"
int *a = (int*) malloc(sizeof(int[5]));
printf("a[5] = %d\n",a[5]);
```

ABW

```
// ABW: writing to memory beyond array bounds
int *a = (int*) malloc(sizeof(int[5]));
a[5] = 5;
```

BFM

```
// BFM: freeing wrong memory block
int *p = (int*) malloc(sizeof(int));
free(p+1);
```

BRP

```
// BRP is "bad address parameter for realloc 0x%lx"
int *p = (int*) realloc(0,sizeof(int));
int *q = (int*) realloc(p+20,sizeof(int[2]));
```

DFM

```
// DFM is "double freeing memory"
int *p = (int*) malloc(sizeof(int));
free(p);
free(p);'
```

FMR

```
// FMR is "reading from freed memory at address 0x%lx (%d byte%s)"
int *p = (int*) malloc(sizeof(int));
free(p);
printf("p = 0x%h\n",p);
```

FMW

```
// FMW is "writing to freed memory at address 0x%lx (%d byte%s)"
int *p = (int*) malloc(sizeof(int));
free(p);
*p = 1;
```

FRP

```
// FRP: freed pointer passed to realloc
int *p = (int*) malloc(sizeof(int));
free(0);
int *q = (int*) realloc(p, sizeof(int[2]));
```

IMR

```
// IMR: read from invalid memory address
int *p = 0;
int i = *p; // generates Signal 11...
```

IMW

```
// IMW: write to invalid memory address
int *p = 0;
*p = 1; // generates Signal 11...
```

OLP

```
char *s=(char *) malloc(15);
memset(s, 'x', 15);
memcpy(s, s+5, 10);
return 0;
```

PIR

```
// PIR: accessing partially initialized data
int *p = (int*) malloc(sizeof(int));
*((char*)p) = 'c';
printf("%*(p = %d\n", *(p+1));
```

SBR

```
int a[2]={0,1};
printf("a[-10]=%d\n",a[-10]);
return 0;
```

SBW

```
int a[2]={0,1}'
a[-10]=2;
return 0;
```

UAR

```
// UAR is "reading from unallocated memory"
int *p = (int*) malloc(sizeof(int));
printf("%*(p+1) = %d\n", *(p+1));
```

UAW

```
// UAW is "writing to unallocated memory"
int *p = (int*) malloc(sizeof(int));
*(p+1) = 1;
```

UMR

```
// UMR is "accessing uninitialized data from address 0x%lx (A%d byte%s)"
int *p = (int*) malloc(sizeof(int));
printf("*p = %d\n", *p);
```

メモリーアクセスの警告

Discover は次のメモリーアクセスの警告を報告します。

- AZS: 0 サイズの割り当て (allocating zero size)
- SMR: 投機的な非初期化メモリーからの読み取り (speculative uninitialized memory read)

次のセクションは、AZS 警告を生成する簡単なプログラム例を一覧表示します。

AZS

```
// AZS: allocating zero size memory block
int *p = malloc();
```

Discover エラーメッセージの解釈

場合によっては、Discover は実際にはエラーでないエラーを報告することがあります。そのようなケースは、擬陽性と呼ばれます。Discover は計測時にコードを分析し、同様なツールと比較して、擬陽性の発生を削減しますが、依然として発生する場合があります。次のセクションでは、Discover レポートでの擬陽性を特定し、可能であれば回避できるようにするヒントを提供します。

部分的に初期化されたメモリー

C および C++ のビットフィールドでは、コンパクトなデータ型を作成できます。例:

```
struct my_struct {
    unsigned int valid : 1;
    char        c;
};
```

この例では、構造メンバー `my_struct.valid` はメモリー内の 1 ビットのみ取得します。ただし、SPARC プラットフォーム上では、CPU はバイト単位でのみメモリーを変更できるため、`struct.valid` を含むバイト全体が構造メンバーにアクセスまたは変更するためにロードされる必要があります。また、コンパイラは 1 度に数バイト (たとえば、4 バイトの機械語) をロードする方がより効率的であることがわかる場合があります。Discover がこのようなロードを検出する場合、追加情報なしに、すべて 4 バイトが使用されると仮定します。また、たとえば、フィールド `my_struct.valid` は初期化されたが、フィールド `my_struct.c` は初期化されず、両方のフィールドを含む機械語がロードされた場合、Discover は部分的に初期化されたメモリーからの読み取り (PIR) にフラグを立てます。

擬陽性の別のソースはビットフィールドの初期化です。1 バイト部分を書き込むには、コンパイラは最初にバイトをロードするコードを生成する必要があります。バイトが読み取るより前に書き込まれていない場合、結果は非初期化メモリーからの読み取りエラー (UMR) となります。

ビットフィールドの擬陽性を回避するには、コンパイル時に `-g` オプションまたは `-g0` オプションを使用します。これらのオプションは、Discover に追加のデバッグ情報を提供し、ビットフィールドのロードと初期化を特定するのに役立ち、多くの擬陽性を削除します。何らかの理由で `-g` オプションを使用してコンパイルできない場合は、`memset()` などの関数を使用して構造を初期化します。例:

```
...
struct my_struct s;
/* Initialize structure prio to use */
memset(&sm 0, sizeof(struct my_struct));
...
```

スペキュレイティブロード

コンパイラは、ロードの結果がすべてのプログラムパスで有効ではない条件下の不明なメモリーアドレスからロードを生成する場合があります。このようなロード命令は分岐命令の遅延スロットに配置できるため、この状況は SPARK プラットフォーム上で発生する場合があります。たとえば、ここに C コードフラグメントがあります。

```
int i'
if (foo(&i) != 0) { /* foo returns nonzero if it has initialized i */
    printf("5d\n", i);
}
```

このコードから、コンパイラは次のコードに等しいコードを生成できます。

```
int i;
int t1, t2'
t1 = foo(&i);
```

```
t2 = i; /* value in i is loaded */
if (t1 != 0) {
    printf("%d\n", t2);
}
```

この例では、関数 `foo()` が `0` を返し、`i` を初期化しないと仮定します。`i` からのロードは、依然として生成されますが、使用されません。しかし、ロードは Discover によって確認され、非初期化変数のロード (UMR) を報告します。

Discover はデータフロー分析を使用して、可能な場合には常にそのようなケースを特定しますが、検出できない場合もあります。

最適化レベルを低くしてコンパイルすることによって、これらのタイプの擬陽性の発生を削減できます。

未計測コード

Discover はプログラムの 100% を計測できない場合があります。おそらく、コードの一部がアセンブリ言語のソースファイルまたは再コンパイルできないサードパーティのライブラリから来ているため、計測できない場合があります。Discover には、未計測コードがアクセスまたは変更しているメモリーブロックの知識がありません。たとえば、サードパーティの共有ライブラリから得られる関数がのちにメイン (計測済み) プログラムによって読み取られるメモリーブロックを初期化すると仮定します。Discover はメモリーがライブラリで初期化されていることを認識していないため、後続の読み取りにより、非初期化メモリーエラー (UMR) が生成されます。

このような場合の解決策を提供するため、Discover API には次の関数が含まれています:

```
void __ped_memory_write(unsigned long addr, long size, unsigned long pc);
void __ped_memory_read(unsigned long addr, long size, unsigned long pc);
void __ped_memory_copy(unsigned long src, unsigned long dst, long size, unsigned long pc);
```

プログラムから API 関数を呼び出し、Discover にメモリー領域への書き込み (`__ped_memory_write()`) やメモリー領域からの読み取り (`__ped_memory_read()`) などの特定のイベントを知らせることができます。どちらの場合も、メモリー領域の開始アドレスは、`addr` パラメータで渡され、そのサイズは `size` パラメータで渡され、`pc` パラメータを `0` に設定します。

`__ped_memory_copy` 関数を使用して、Discover にメモリーがある場所から別の場所にコピーされていることを知らせます。ソースメモリーの開始アドレスは `src` パラメータで渡され、宛先領域の開始アドレスは `dst` パラメータで渡され、サイズは `size` パラメータで渡されます。`pc` パラメータを `0` に設定します。

API を使用するには、プログラムでこれらの関数を `weak` と宣言します。たとえば、ソースコードに次のコードフラグメントを含めます。

```
#ifndef __cplusplus
extern "C" {
#endif

extern void __ped_memory_write(unsigned long addr, long size, unsigned long pc);
extern void __ped_memory_read(unsigned long addr, long size, unsigned long pc);
extern void __ped_memory_copy(unsigned long src, unsigned long dst, long size, unsigned long pc);

#pragma weak __ped_memory_write
#pragma weak __ped_memory_read
#pragma weak __ped_memory_copy

#ifdef __cplusplus
}
#endif
```

API 関数は内部 Discover ライブラリで定義され、計測時にプログラムとリンクされます。ただし、プログラムが計測されない場合、このライブラリはリンクされず、API 関数へのすべての呼び出しによってアプリケーションがハングアップします。そのため、Discover 下でプログラムを実行していない場合は、これらの関数を無効にする必要があります。または、API 関数の空の定義を使用して動的ライブラリを作成し、それをプログラムとリンクすることができます。この場合、Discover を使用しないでプログラムを実行する場合は、ライブラリが使用されますが、Discover 下で実行する場合は、真の API 関数が自動的に呼び出されます。

Discover 使用時の制限事項

注釈付きコードのみが計測される

Discover は、「11 ページの「バイナリは正しく準備される必要がある」」の説明に従って準備されているコードのみを計測できます。注釈の付いていないコードは、バイナリにリンクされているアセンブリ言語コード、またはそのセクションに示されているものより古いコンパイラまたはオペレーティングシステムでコンパイルされたモジュールから来ている場合があります。

準備から特別に除外されているのは、asm 文または .il テンプレートを含むアセンブリ言語モジュールおよび関数です。

機械命令はソースコードとは異なる場合がある

Discover は機械コード上で動作します。ツールは、ロードやストアなどの機械命令でエラーを検出し、それらのエラーをソースコードと相互に関連付けます。一部のソースコード文は関連付けられている機械命令がないため、Discover は明白な

ユーザーエラーを検出していないように思われる場合があります。たとえば、次の C コードフラグメントを考えてみましょう:

```
int *p = (int *)malloc(sizeof(int));
int i;

i = *p; /* compiler may not generate code for this statement */
printf("Hello World!\n");

return;
```

p で示されたアドレスに格納された値を読み取ることは、メモリーが初期化されていないため、潜在的なユーザーエラーとなります。ただし、最適化コンパイラは、変数 i が使用されていないことを検出するため、メモリーから読み取り、i に割り当てる文のコードは、生成されません。この場合、Discover は非初期化メモリーの使用 (UMR) を報告しません。

コンパイラオプションは生成されたコードに影響を及ぼす

コンパイラの生成コードは常に想定どおりになるわけではありません。コンパイラが生成するコードは、`-On` 最適化オプションを含む、使用するコンパイラオプションによって異なるため、Discover によって報告されるエラーも異なる可能性があります。たとえば、`-O1` 最適化レベルで生成されたコードで報告されるレポートは、`-O4` 最適化レベルで生成されたコードによって非表示になる可能性があります。

システムライブラリは報告されたエラーに影響を及ぼす可能性がある

システムライブラリは、オペレーティングシステムとともにインストール済みで、計測用に再度コンパイルできません。Discover は、標準の C ライブラリ (`libc.so`) からの一般的な関数に対するサポートを提供します。すなわち、Discover これらの関数によってどのメモリーにアクセスされ、どのメモリーが変更されるかを把握しています。ただし、アプリケーションが他のシステムライブラリを使用する場合、Discover レポートで擬陽性を検出する可能性があります。擬陽性が報告される場合、コードから Discover API を呼び出してそれらを削除できます。

カスタムメモリー管理はデータの正確さに影響を及ぼす可能性がある

Discover は、`malloc()`、`calloc()`、`free()`、`operator new()`、および `operator delete()` などの標準のプログラミング言語メカニズムによって割り当てられている場合にヒープメモリーを検出できます。

アプリケーションが標準の関数の最上部で動作するカスタムメモリー管理システム (たとえば、`malloc()` とともに実装されるプール割り当て管理) を使用する場合、Discover は機能しますが、適切なリークの報告や、解放されたメモリーへのアクセスは保証されていません。

Discover は、次のメモリーアロケータをサポートしていません:

- `brk(2)()` または `sbrk(2)()` システム呼び出しを直接使用するカスタムヒープアロケータ
- バイナリに静的にリンクされた標準のヒープ管理関数
- `mmap(2)()` および `shmget(2)()` システム呼び出しを使用してユーザーコードから割り当てられたメモリー

`sigaltstack(2)()` 関数はサポートされていません。

静的および自動配列範囲外は削除できない

配列範囲を検出するため Discover が使用するアルゴリズムのため、静的および自動 (ローカル) 配列の範囲外アクセスエラーを検出することはできません。動的に割り当てられた配列のエラーのみ検出できます。

コードカバレッジツール(Uncover)

- 41 ページの「Uncover を使用するための要件」
- 42 ページの「Uncover の使用法」
- 45 ページの「パフォーマンスアナライザのカバレッジレポートを理解する」
- 51 ページの「ASCII カバレッジレポートを理解する」
- 55 ページの「HTML カバレッジレポートを理解する」

Uncover を使用するための要件

Uncover は、Sun Studio 12 Update 1、Oracle Solaris Studio 12.2、Oracle Solaris Studio 12.3 コンパイラ、または GCC for Sun Systems コンパイラのバージョン 4.2.0 以降を使用してコンパイルされたバイナリ上で機能します。Solaris 10 10/08 オペレーティングシステムまたはそれ以降の Solaris 10 update、または Oracle Solaris 11 を実行している、SPARC ベースシステムまたは x86 ベースシステムで機能します。

上記のようにコンパイルされるバイナリには、Uncover がカバレッジデータの収集用に計測するためにバイナリを確実に逆アセンブリする情報が含まれています。

-g オプションを使用して、バイナリのコンパイル時にデバッグ情報を生成することにより、Uncover はソースコードレベルのカバレッジ情報を使用できます。バイナリが -g オプションを使用してコンパイルされない場合、プログラムカウンタ (PC) ベースのカバレッジ情報のみを使用します。

Uncover は、Oracle Solaris Studio コンパイラで構築された任意のバイナリで機能しますが、最適化オプションなしで構築されたバイナリで最適に機能します。(以前のリリースの Uncover では、少なくとも -O1 最適化レベルが必要でした。)最適化オプションを使用してバイナリを構築した場合、Uncover の結果は、最適化レベルが低いほど良くなります (-O1 または -O2)。-g オプションでバイナリを構築したときに生成されたデバッグ情報を使用して、命令を行番号に関連付けることにより、Uncover はソース行レベルのカバレッジ情報を取得します。最適化レベル -O3 以上では、実行されることのないコードや冗長なコードがコンパイラで削除される場合があり、そ

の結果、一部のソースコード行にはバイナリ命令が存在しないことがあります。そのような場合、それらの行に関するカバレッジ情報は報告されません。詳細は、57 ページの「[Uncover 使用時の制限事項](#)」を参照してください。

Uncover の使用法

Uncover を使用してカバレッジ情報を生成するには、3 ステップのプロセスがあります。

1. バイナリの計測
2. 計測済みバイナリの実行
3. カバレッジレポートの生成と表示

バイナリの計測

入力バイナリは、実行可能ファイルまたは共有ライブラリとすることができません。分析したいバイナリの計測は個別に行う必要があります。

`uncover` コマンドを使用してバイナリを計測します。たとえば、次のコマンドは、バイナリ `a.out` を計測し、入力 `a.out` を計測済み `a.out` で上書きします。また、このコマンドは、接尾辞 `.uc` を持つディレクトリ (この場合は `a.out.uc`) を作成します。この中に、カバレッジデータが収集されます。入力バイナリのコピーはこのディレクトリに保存されます。

`uncover a.out`

バイナリに計測機構を組み込むとき、次のオプションを使用できます。

- c 命令、ブロック、および関数の実行カウントの報告をオンにします。デフォルトでは、カバーされているコードまたはカバーされていないコードの情報だけが報告されます。(バイナリに計測機構を組み込むときとカバレッジレポートを生成するときの両方で、このオプションを指定します。)
- d *directory* *directory* 内にカバレッジデータディレクトリを作成するよう Uncover に指示します。このオプションは、複数のバイナリ用のカバレッジデータを収集する場合に役立ち、カバレッジデータディレクトリのすべてが同じディレクトリ内に作成されるようにします。また、異なる場所から同じ計測済みバイナリの異なるインスタンスを実行する場合、このオプションを使用すると、これらの実行のすべてから取得されるカバレッジデータが同じカバレッジデータディレクトリに確実に蓄積されるようになります。

- d オプションを使用しない場合、カバレッジデータディレクトリは現在の実行ディレクトリに作成されます。
- m on | off スレッドセーフなプロファイリングをオンまたはオフにします。デフォルトはオンです。このオプションは -c 実行時オプションと組み合わせて使用します。スレッドを使用するバイナリに -m off で計測機構を組み込むと、バイナリは実行時に失敗し、バイナリに -m on で計測機構を組み込み直すよう指示するメッセージが表示されます。
- o *output_binary_file* 指定されたファイルに計測機構付きバイナリファイルを書き込みます。デフォルトでは、入力バイナリファイルが計測機構付きファイルで上書きされます。

すでに計測されている入力バイナリ上で `uncover` コマンドを実行する場合、Uncover はすでに計測されているためバイナリを計測できないことと、そのバイナリを実行してカバレッジデータを生成できることを伝えるエラーメッセージを発行します。

計測済みバイナリの実行

バイナリを計測した後で、それを正常に実行できます。計測済みバイナリを実行するたびに、コードカバレッジデータは、Uncover が計測中に作成した `.uc` 接尾辞を持つカバレッジデータディレクトリに収集されます。Uncover データコレクションは、マルチスレッドおよびマルチプロセスに対して安全であるため、プロセスの同時実行またはスレッド数に制限はありません。カバレッジデータは実行およびスレッドのすべてにわたって蓄積されます。

カバレッジレポートの生成と表示

カバレッジレポートを生成するには、カバレッジデータディレクトリ上で `uncover` コマンドを実行します。例:

```
uncover a.out.uc
```

このコマンドは、`a.out.uc` ディレクトリのカバレッジデータから `binary_name.er` と呼ばれる Oracle Solaris Studio パフォーマンスアナライザ実験ディレクトリを生成し、パフォーマンスアナライザ GUI を起動して、実験を表示します。現作業ディレクトリまたはホームディレクトリに `.er.rc` ファイル (『Oracle Solaris Studio 12.2 パフォーマンスアナライザ』のマニュアルを参照) がある場合は、アナライザが実験を表示する方法に影響を及ぼす場合があります。

また、`uncover` コマンドオプションを使用すると、レポートを HTML 形式で生成して Web ブラウザで表示したり、ASCII 形式で生成して端末ウィンドウで表示したりすることができます。または、データをディレクトリに書き込み、コードアナライザで分析と表示を行うことができます。

- a コードアナライザで使用するためにエラーデータを `binary_name.analyze/coverage` ディレクトリに書き込みます。
- c 命令、ブロック、および関数の実行カウントの報告をオンにします。デフォルトでは、カバーされているコードまたはカバーされていないコードの情報だけが報告されます。(バイナリに計測機構を組み込むときとカバレッジレポートを生成するときの両方で、このオプションを指定します。)
- e on | off カバレッジレポートのための実験ディレクトリを生成し、パフォーマンスアナライザ GUI で実験を表示します。デフォルトではオンになっています。
- H *html_directory* カバレッジデータを指定のディレクトリに HTML 形式で保存し、それを Web ブラウザで自動的に表示します。デフォルトでオフになっています。
- h または -? ヘルプ。
- n カバレッジレポートを生成しますが、パフォーマンスアナライザや Web ブラウザなどのビューアを起動しません。
- t *ascii_file* 指定されたファイルで ASCII カバレッジレポートを生成します。デフォルトでオフになっています。
- V Uncover バージョンを出力して終了します。
- v 冗長。Uncover が実行する内容のログを出力します。

出力形式は 1 つだけ有効になるため、複数の出力オプションを指定する場合、Discover はコマンドの最後のオプションを使用します。

例

`uncover a.out`

このコマンドは、バイナリ `a.out` を計測し、入力 `a.out` を上書きして、現作業ディレクトリに `a.out.uc` カバレッジデータディレクトリを作成し、`a.out.uc` ディレクトリに入力 `a.out` のコピーを保存します。`a.out` がすでに計測されている場合、警告メッセージが表示され、計測は実行されません。

`uncover -d coverage a.out`

このコマンドは、a.out.uc カバレッジディレクトリをディレクトリ coverage に作成する点を除いて、最初の例が行うすべてのことを実行します。

```
uncover a.out.uc
```

このコマンドは、a.out.uc カバレッジディレクトリのデータを使用して、作業ディレクトリにコードカバレッジの実験(a.out.er)を作成し、パフォーマンスアナライザ GUI を起動してその実験を表示します。

```
uncover -H a.out.html a.out.uc
```

このコマンドは、a.out.uc カバレッジディレクトリのデータを使用して、ディレクトリ a.out.html に HTML コードカバレッジレポートを作成し、Web ブラウザにレポートを表示します。

```
uncover -t a.out.txt a.out.uc
```

このコマンドは、a.out.uc カバレッジディレクトリのデータを使用して、ファイル a.out.txt に ASCII コードカバレッジレポートを作成します。

```
uncover -a a.out.uc
```

このコマンドは a.out.c カバレッジディレクトリ内のデータを使用して、コードアナライザで使用するためのカバレッジレポートを *binary_name.analyze/coverage* ディレクトリ内に作成します。

パフォーマンスアナライザのカバレッジレポートを理解する

デフォルトでは、カバレッジディレクトリ上の uncover コマンドを実行する場合、カバレッジレポートが Oracle Solaris Studio パフォーマンスアナライザの実験として開かれます。アナライザは、「関数 (Functions)」、「ソース (Source)」、「逆アセンブリ (Disassembly)」、および「命令頻度 (Inst-Freq)」タブを使用して、カバレッジデータを表示します。

「関数」タブ

アナライザでカバレッジレポートを開くと、「関数 (Functions)」タブが選択されます。このタブには、関数ごとの「カバレッジ外 (Uncoverage)」、「関数カウント (Function Count)」、「命令の実行 (Instr Exec)」、「カバーされているブロックの割合 (%) (Block Covered %）」、「カバーされている命令の割合 (%) (Instr Covered %)」カ

ウインタを一覧表示した列が表示されます。列ヘッダーをクリックすると、どの列でもデータのソートキーとして指定できます。列ヘッダーの矢印をクリックすると、ソート順が逆になります。

The screenshot shows the Oracle Solaris Studio Performance Analyzer interface. The main window displays a table of functions with columns for Uncoverage, Function Count, Instr Exec, Block Covered %, and Instr Covered %. The 'Uncoverage' column is sorted in descending order, with 'main' at the top. A summary panel on the right shows details for the selected object 'main', including its PC address, size, source file, and various metrics for coverage and execution.

Uncoverage	Function Count	Instr Exec	Block Covered %	Instr Covered %	Name
22 424	212 276 980	\$ 030 976 791	3 370	3 158	<Total>
2 076	0	24	5	14	main
1 596	0	0	0	0	forkchild
1 144	0	0	0	0	pagethresh
1 116	0	42	33	47	endcases
1 000	0	0	0	0	forkcopy
980	0	0	0	0	iofile
748	0	0	0	0	do_vforkexec
732	0	0	0	0	callso
712	0	1 603	27	13	commandline
708	0	0	0	0	do_forkexec
648	0	0	0	0	callso
644	0	0	0	0	sigprof
644	0	0	0	0	sigprofh
556	0	0	0	0	do_chdir
548	0	0	0	0	correlate
492	0	0	0	0	do_popen
472	0	62	25	26	stopwatch_print
384	0	0	0	0	so_cpuime
384	0	0	0	0	sk_cpuime
348	0	0	0	0	itimer_realprof
336	0	0	0	0	ldso
328	0	2	33	18	get_ocpus
304	0	0	0	0	hrv
300	0	0	0	0	do_system
300	0	23	33	30	sigtime
300	0	0	0	0	so_burhcpu
300	0	0	0	0	sk_burhcpu
284	0	156	37	33	get_clock_rate
276	0	0	0	0	masksignals
268	0	23	33	34	gpf
268	0	8 054	40	59	stopwatch_stop
256	0	0	0	0	sigprof_handler
256	0	0	0	0	sigprof_sigaction
248	0	18	14	29	init_malloc_attr

Summary Panel:

Selected Object:

- Name: main
- PC Address: 2: 0x0005050
- Size: 224
- Source File: /usr/lib/uncover/stopwatch/stopwatch.c
- Object File: /usr/lib/archives/a.out_SPARCv9/2d
- Load Object: <a.out>
- Mangled Name:
- Aliases:

Metrics for Selected Object:

	Exclusive	Inclusive
Uncoverage:	2076 (9.26%)	2076 (9.26%)
Function Count:	0 (0.0%)	0 (0.0%)
Instr Exec:	24 (0.00%)	24 (0.00%)
Block Covered %:	5 (0.15%)	5 (0.15%)
Instr Covered %:	14 (0.44%)	14 (0.44%)

「カバレッジ外 (Uncoverage)」 カウンタ

「カバレッジ外 (Uncoverage)」 メトリックは、Uncover の非常に強力な機能です。この列を降順のソートキーとして使用する場合、最上部に表示される関数は、カバレッジを増やす可能性が最も高い関数です。例では、main() 関数が「カバレッジ外 (Uncoverage)」列で最大数を持つため、リストの最上部に表示されます。(sigprof() および sigprofh() 関数は同数を持つため、アルファベット順に一覧表示されます。)

main() 関数のカバレッジ外の数は、関数が呼び出される原因となるスイートにテストが追加される場合に潜在的にカバーされる可能性のあるコードのバイト数で

す。カバレッジが実際に増加する量は、関数の構造によって異なります。関数に分岐がなく、呼び出すすべての関数が直線関数でもある場合、カバレッジは実際には定められたバイト数により増加します。ただし、一般的に、カバレッジの増加は潜在的に想定されるよりおそらくずっと少ないです。

「カバレッジ外 (Uncoverage)」列の0以外の値を持つカバーされていない関数は、カバーされていないルート関数と呼ばれ、カバーされている関数によってすべて呼び出されることを意味します。カバーされていない非ルート関数によってのみ呼び出される関数は、独自のカバレッジ外の数を持ちません。テストスイートは、潜在性の高いカバーされていない関数をカバーするように改良されるにつれて、これらの関数は、後続の実行で、カバーされるか、またはカバーされないことが明らかにされると想定されます。

カバレッジ数は排他的ではありません。

「関数カウント (Function Count)」カウンタ

「関数カウント (Function Count)」は、カバーされている関数とカバーされていない関数を報告します。重要なことは数が0か0以外であるかということです。数が0の場合、関数はカバーされません。数が0以外の場合、関数はカバーされます。関数の命令が実行される場合、関数はカバーされるとみなされます。

この列で、トップレベル以外のカバーされていない関数を検出できます。ある関数の関数カウントが0で、カバレッジ外の数も0である場合、関数はトップレベルのカバーされている関数ではありません。

「命令の実行 (Instr Exec)」カウンタ

「命令の実行 (Instr Exec)」カウンタは、カバーされている命令とカバーされていない命令を表示します。0のカウントは命令が実行されないことを意味し、0以外のカウントは命令が実行されることを意味します。

「関数 (Functions)」タブで、このカウンタは関数ごとに実行される命令の総数を表示します。このカウンタは、「ソース (Source)」タブ (48 ページの「[「ソース」タブ](#)」を参照) および「逆アセンブリ (Disassembly) タブ (50 ページの「[「逆アセンブリ」タブ](#)」を参照) にも表示されます。

「カバーされているブロックの割合 (%) (Block Covered %)」カウンタ

関数ごとに、「カバーされているブロックの割合 (%) (Block Covered %)」カウンタに、カバーされている関数の基本的なブロックの割合が表示されます。この数により、関数がいかに適切にカバーされているかがわかります。<Total> 行のこの数は無視してください。これは列のパーセンテージの合計であり、意味がありません。

「カバーされている命令の割合 (%) (Instr Covered %)」カウンタ

関数ごとに、「カバーされている命令の割合 (%) (Instr Covered %)」カウンタに、カバーされている関数の命令の割合が表示されます。この数によっても、関数がいかに適切にカバーされているかがわかります。<Total>行のこの数は無視してください。これは列のパーセンテージの合計であり、意味がありません。

「ソース」タブ

バイナリを `-g` オプションを使用してコンパイルした場合、「ソース (Souece)」タブにはプログラムのソースコードが表示されます。Uncover はバイナリレベルでプログラムを計測し、最適化してプログラムをコンパイルしているため、このタブのカバレッジ情報は解釈するのが困難な可能性があります。

「ソース (Source)」タブの「命令の実行 (Instr Exec)」カウンタには、各ソース行に対して実行される命令の総数が表示され、これは基本的に文レベルのコードカバレッジ情報です。0 以外の値は、文がカバーされていることを意味します。0 の値は、文がカバーされていないことを意味します。変数宣言およびコメントには命令の実行カウントがありません。

The screenshot shows the Oracle Solaris Studio Performance Analyzer interface. The main window displays a list of source code lines with columns for 'Block Covered %' and 'Inst Covered %'. Line 160 is highlighted in yellow, indicating it is covered. The right-hand pane shows a 'Summary' for the selected object, including PC Address, Size, Source File, Object File, and Load Object. Below the summary, 'Metrics for Selected Object' are shown for 'Exclusive' and 'Inclusive' modes, with values for Uncoverage, Function Count, Instr Exec, Block Covered %, and Instr Covered %.

一部のソースコード行にカバレッジ情報が関連付けられていない場合があります。このような場合、行は空で、どのフィールドにも数値が含まれていません。このような行は次の理由で発生します。

- コメント、空行、宣言など、実行可能コードを含んでいない言語構造。
- コンパイラの最適化により、行に対応するコードが次の理由で削除されている。
 - コードは実行されることがない(デッドコード)。
 - コードは実行可能であるが冗長。

詳細は、57 ページの「Uncover 使用時の制限事項」を参照してください。

「逆アセンブリ」タブ

「ソース (Source)」タブのある行を選択し、次に「逆アセンブリ (Disassembly)」タブを選択する場合、アナライザは、バイナリで選択した行を検出し、その逆アセンブリを表示しようとします。

このタブの「命令の実行 (Instr Exec)」カウンタには、各命令が実行される回数が表示されます。

The screenshot shows the Oracle Solaris Studio Performance Analyzer interface. The main window is titled "Oracle Solaris Studio Performance Analyzer [a.out.er]". The "Disassembly" tab is selected, showing assembly instructions for the function "main". The instructions are listed with their addresses and mnemonics. The columns "Uncoverage", "Function Count", "Instr Exec", "Block Covered %", and "Instr Covered %" are visible. The "main" function is highlighted in yellow. The summary panel on the right shows the following metrics:

Selected Object:	
Name:	main + 0x00000000, line 160 in "sy"
PC Address:	2: 0x00005090
Size:	4
Source File:	ice/Aten/Uncover/symprog/symprog.c
Object File:	.out.er/archives/a.out_5F805q1p02d
Load Object:	a.out
Mangled Name:	
Aliases:	
Metrics for Selected Object:	
	<input type="checkbox"/> Exclusive <input checked="" type="checkbox"/> Inclusive
Uncoverage:	2076 (5.26%) 2076 (5.26%)
Function Count:	0 (0. %) 0 (0. %)
Instr Exec:	0 (0. %) 0 (0. %)
Block Covered %:	5 (0.15%) 5 (0.15%)
Instr Covered %:	14 (0.44%) 14 (0.44%)

「命令頻度」タブ

「命令頻度 (Inst-Freq)」タブにはカバレッジの概要全体が表示されます。

The screenshot shows the Oracle Solaris Studio Performance Analyzer interface. The main window is titled "Oracle Solaris Studio Performance Analyzer [a.out.er]". The "Inst-Freq" tab is selected, displaying a list of instructions. The right-hand pane shows the "Summary" for the selected object, which is "main + 0x00000000, line 100 in 'a.out'". The summary includes details such as PC Address (210x00005D90), Size (4), Source File (ice/Atm/Uncover/ymprog/ymprog.o), Object File (.out.er/archives/a.out_5FM25q3p2d), and Load Object (<a.out). Below the summary, a table shows "Metrics for Selected Object" with columns for "Exclusive" and "Inclusive" coverage, function count, instruction execution, block covered, and instruction covered.

	Exclusive	Inclusive
Uncoverage:	2076 (9.26%)	2076 (9.26%)
Function Count:	0 (0. %)	0 (0. %)
Inst Exec:	0 (0. %)	0 (0. %)
Block Covered %:	5 (0.15%)	5 (0.15%)
Inst Covered %:	14 (0.44%)	14 (0.44%)

ASCII カバレッジレポートを理解する

カバレッジデータディレクトリからカバレッジレポートを生成する際に `-t` オプションを指定する場合、Uncover はカバレッジレポートを指定された ASCII (テキストファイル) に書き込みます。

```
UNCOVER Code Coverage
Total Functions: 95
Covered Functions: 58
```

Function Coverage: 61.1%
 Total Basic Blocks: 568
 Covered Basic Blocks: 258
 Basic Block Coverage: 45.4%
 Total Basic Block Executions: 564,812,760
 Average Executions per Basic Block: 994,388.66
 Total Instructions: 6,201
 Covered Instructions: 3,006
 Instruction Coverage: 48.5%
 Total Instruction Executions: 4,760,934,518
 Average Executions per Instruction: 767,768.83
 Number of times this program was executed: unavailable
 Functions sorted by metric: Exclusive Uncoverage

Excl. Uncoverage	Excl. Function Count	Excl. Block Covered %	Excl. Instr Covered %	Name
13404	6004876	5464	5384	<Total>
1036	0	0	0	main
980	0	0	0	iofile
748	0	0	0	do_vforkexec
732	0	0	0	callso
708	0	0	0	do_forkexec
648	0	0	0	callsx
644	0	0	0	sigprof
644	0	0	0	sigprofh
556	0	0	0	do_chdir
548	0	0	0	correlate
492	0	0	0	do_popen
404	0	0	0	pagethrash
384	0	0	0	so_cputime
384	0	0	0	sx_cputime
348	0	0	0	itimer_realprof
336	0	0	0	ldso
304	0	0	0	hrv
300	0	0	0	do_system
300	0	0	0	do_burncpu
300	0	0	0	sx_burncpu
288	0	0	0	forkcopy
276	0	0	0	masksignals
256	0	0	0	sigprof_handler
256	0	0	0	sigprof_sigaction
216	0	0	0	do_exec
196	0	0	0	iotest
176	0	0	0	closeso
156	0	0	0	gethrustime
144	0	0	0	forkchild
144	0	0	0	gethrpxtime
136	0	0	0	whrlog
112	0	0	0	masksig
92	0	0	0	closesx
84	0	0	0	reapchildren
36	0	0	0	reapchild
32	0	0	0	doabort
8	0	0	0	csig_handler
0	1	66	72	acct_init
0	1	100	100	bounce
0	63	100	96	bounce_a

0	60	100	100	bounce-b
0	16	71	58	check_sigmask
0	1	83	77	commandline
0	1	100	98	cputime
0	1	100	98	dousleep
0	1	100	100	endcases
0	1	100	95	ext_inline_code
0	1	100	96	ext_macro_code
0	1	100	99	fitos
0	2	81	80	get_clock_rate
0	1	100	100	get_ncpus
0	1	100	100	gpf
0	1	100	100	gpf_a
0	1	100	100	gpf_b
0	10	100	93	gpf_work
0	1	100	97	icputime
0	1	100	96	inc_body
0	1	100	96	inc_brace
0	1	100	95	inc_entry
0	1	100	95	inc_exit
0	1	100	96	inc_func
0	1	100	94	inc_middle
0	1	57	72	init_micro_acct
0	1	50	43	initcksig
0	1	100	95	inline_code
0	1	100	95	macro_code
0	1	100	98	muldiv
0	6000000	100	100	my_irand
0	1	100	98	naptime
0	19	50	83	prdelta
0	21	100	100	prhrdelta
0	21	100	100	prhrvdelta
0	1	100	100	prtime
0	552	100	98	real_recurse
0	1	100	100	recurse
0	1	100	100	recursedeeep
0	1	100	95	s_inline_code
0	1	100	100	sigtime
0	1	100	95	sigtime_handler
0	19	100	100	snaptod
0	1	100	100	so_init
0	2	66	75	stpwtch_alloc
0	1	100	100	stpwtch_calibrate
0	2	75	66	stpwtch_print
0	2002	100	100	stpwtch_start
0	2000	90	91	stpwtch_stop
0	1	100	100	sx_init
0	1	100	99	systeme
0	3	100	95	tailcall_a
0	3	100	95	tailcall_b
0	3	100	95	tailcall_c
0	1	100	100	tailcallopt
0	1	100	97	underflow
0	21	75	71	whrvlog
0	19	100	100	wlog

Instruction frequency data from experiment a.out.er

Instruction frequencies of /export/home1/synprog/a.out.uc

```

Instruction      Executed      ( )
TOTAL           4760934518 (100.0)
float ops       2383657378 ( 50.1)
float ld st     1149983523 ( 24.2)
load store      1542440573 ( 32.4)
load            882693735 ( 18.5)
store           659746838 ( 13.9)

```

```

-----
Instruction      Executed      ( )      Annulled      In Delay Slot
TOTAL           4760934518 (100.0)
add              713013787 ( 15.0)      16             1501335
subcc            558774858 ( 11.7)      0              6002
br               558769261 ( 11.7)      0              0
stf              432500661 (  9.1)      726           36299281
ldf              408226488 (  8.6)      40            103000396
fadd             391230847 (  8.2)      0              0
fdtos            366200726 (  7.7)      0              0
fstod            360200000 (  7.6)      0              0
lddf             288250336 (  6.1)      500           282200229
stw              138028738 (  2.9)      26002         25974065
lduw             118004305 (  2.5)      71            94000270
ldx              68212446 (  1.4)      0             2000
stx              68211370 (  1.4)      7            23532716
fitod            36026002 (  0.8)      0              0
sethi            36002986 (  0.8)      0             228
fdtoi            30000001 (  0.6)      0              0
fdivd            26000088 (  0.5)      0              0
call             22250348 (  0.5)      0              0
srl              21505246 (  0.5)      0             21
stdf             21006038 (  0.4)      0              0
or               19464766 (  0.4)      0            10981277
fmuls            6004907 (  0.3)      0              0
jmpl             6004853 (  0.1)      0              0
save             6004852 (  0.1)      0              0
restore          6002294 (  0.1)      0            6004852
sub              6000019 (  0.1)      0              0
xor              6000000 (  0.1)      0              0
fitos            6000000 (  0.1)      0              0
fstoi            6000000 (  0.1)      0              0
and              6000000 (  0.1)      0              0
andn             6000000 (  0.1)      0              0
sll              3505225 (  0.1)      0              0
nop              3505219 (  0.1)      0            3505219
fxtod            7763 (  0.0)      0              0
bpr              6000 (  0.0)      0              0
fcmped           4837 (  0.0)      0              0
fbr              4837 (  0.0)      0              0
fmuld            2850 (  0.0)      0              0
orcc             383 (  0.0)      0              0
sra              241 (  0.0)      0              0
ldsb             160 (  0.0)      0              0
mulx             87 (  0.0)      0              0
stb              31 (  0.0)      0              0
mov              21 (  0.0)      0              0
fdtox            15 (  0.0)      0              0
=====

```

HTMLカバレッジレポートを理解する

HTMLレポートは、パフォーマンスアナライザに表示されるレポートに類似しています。

```
HTML data from experiment(s):
a.out.ar

Functions sorted by metric: Exclusive Uncoverage
```

Excl. Uncoverage	Excl. Function Count	Excl. Instr Exec	Excl. Block Covered %	Excl. Instr Covered %	Name
20340	6004793	4398402003	3424	5430	<Total>
1680	0	0	0	0	[trimmed] iofile arr Caller-callee
1316	0	0	0	0	[trimmed] de_forkexec arr Caller-callee
1300	0	0	0	0	[trimmed] de_vforkexec arr Caller-callee
1056	0	0	0	0	[trimmed] callas arr Caller-callee
956	0	0	0	0	[trimmed] de_spawn arr Caller-callee
940	0	0	0	0	[trimmed] sigprok arr Caller-callee
940	0	0	0	0	[trimmed] sigprok arr Caller-callee
932	0	0	0	0	[trimmed] curstate arr Caller-callee
832	0	0	0	0	[trimmed] de_child arr Caller-callee
800	0	0	0	0	[trimmed] osetuparr arr Caller-callee
694	0	0	0	0	[trimmed] as_creatas arr Caller-callee
694	0	0	0	0	[trimmed] as_creatas arr Caller-callee
612	0	0	0	0	[trimmed] de_spawn arr Caller-callee
596	0	0	0	0	[trimmed] itimer_rearr arr Caller-callee
572	0	0	0	0	[trimmed] ldev arr Caller-callee
540	0	0	0	0	[trimmed] osetuparr arr Caller-callee
532	0	0	0	0	[trimmed] lrv arr Caller-callee
520	0	0	0	0	[trimmed] forkcopy arr Caller-callee
528	0	0	0	0	[trimmed] as_burnout arr Caller-callee
528	0	0	0	0	[trimmed] as_burnout arr Caller-callee
512	0	0	0	0	sigprok_sigaction
496	0	0	0	0	sigprok_handler
484	0	0	0	0	de_err
440	0	0	0	0	iofile
312	0	0	0	0	whaleg
280	0	0	0	0	closeo
244	0	0	0	0	forkchild
220	0	0	0	0	gethrptime
212	0	0	0	0	gethrustime
144	0	0	0	0	waitsig
140	0	0	0	0	closeo
92	0	0	0	0	reapchildren
90	0	0	0	0	abort
88	0	0	0	0	reapchild
20	0	0	0	0	sig_handler
0	1	58	46	73	acct_init
0	1	131	100	100	[trimmed] bounce arr Caller-callee
0	21	25600457	100	93	[trimmed] bounce_a arr Caller-callee
0	20	260	100	100	[trimmed] bounce_b arr Caller-callee
0	1	79	33	33	callo
0	16	563	71	40	check_sigmask
0	1	8928	88	73	osetuparr
0	1	24800423	100	98	[trimmed] creatias arr Caller-callee
0	1	23800744	100	99	[trimmed] duplino arr Caller-callee
0	1	159	100	100	[trimmed] endmas arr Caller-callee
0	1	8000022	100	98	[trimmed] set_inline_code arr Caller-callee
0	1	9800020	100	97	[trimmed] set_macro_code arr Caller-callee
0	1	142011931	100	98	[trimmed] ldev arr Caller-callee
0	2	10470	76	67	get_clock_rate

関数名のリンクまたは関数の trimmed のリンクをクリックする場合、その関数の逆アセンブリデータが表示されます。


```
Function Names: <Total>
current filename for subsequent output: a.out.html/calls
Functions sorted by metric: Exclusive Uncoverage
Callers and callees sorted by metric: Attributed Uncoverage
```

Attr. Uncoverage	Attr. Function Count	Attr. Instr Exec	Attr. Block Covered %	Attr. Instr Covered %	Name
20240	6004793	6199402002	5424	5430	*<Total>
1690	0	0	0	0	acfile
1316	0	0	0	0	do_forbuser
1200	0	0	0	0	do_vforkuser
1096	0	0	0	0	callr
996	0	0	0	0	do_sopen
940	0	0	0	0	sigprocf
940	0	0	0	0	sigprocfh
932	0	0	0	0	correlate
832	0	0	0	0	do_dstat
800	0	0	0	0	semthruash
684	0	0	0	0	as_spoutine
684	0	0	0	0	as_spoutine
612	0	0	0	0	do_system
596	0	0	0	0	itimer_cancelof
572	0	0	0	0	lstat
560	0	0	0	0	semshmole
552	0	0	0	0	hex
528	0	0	0	0	forkuser
528	0	0	0	0	as_burrows
528	0	0	0	0	as_burrows
512	0	0	0	0	sigprocf_sigaction
496	0	0	0	0	sigprocf_handler
484	0	0	0	0	do_ssem
440	0	0	0	0	actual
312	0	0	0	0	whirlw
280	0	0	0	0	classo
244	0	0	0	0	forbidbid
220	0	0	0	0	authractive
212	0	0	0	0	authractive
184	0	0	0	0	mainstat
140	0	0	0	0	classer
92	0	0	0	0	readchildren
88	0	0	0	0	doibact
88	0	0	0	0	readchild
20	0	0	0	0	read_handler
0	1	46	46	73	asct_anat
0	1	131	100	100	hopen
0	21	25600457	100	93	hopen_a
0	20	240	100	100	hopen_b
0	1	79	33	33	callr
0	16	562	71	40	check_sigprocf
0	1	8926	89	72	semshmole
0	1	24000823	100	96	readline
0	1	23800744	100	96	doibact
0	1	159	100	100	mainstat

Uncover 使用時の制限事項

注釈付きコードのみ計測可能

Uncover は 41 ページの「Uncover を使用するための要件」の説明に従って準備されているコードのみを計測できます。注釈の付いていないコードは、バイナリにリンクされているアセンブリ言語コード、またはそのセクションに示されているものより古いコンパイラまたはオペレーティングシステムでコンパイルされたモジュールから来ている場合があります。

準備から特別に除外されているのは、asm 文または .il テンプレートを含むアセンブリ言語モジュールおよび関数です。

機械命令はソースコードと異なる場合がある

Uncover は機械コード上で動作します。Uncover は機械命令のカバレッジを検出し、このカバレッジをソースコードと関連付けます。一部のソースコード文は関連した機械命令を持たないため、Uncover はそのような文のカバレッジを報告しないと思われる場合があります。

例 1

次に示すコードの一部を考えてみましょう。

```
#define A 100
#define B 200
...
    if (A>B) {
        ...
    }
```

Uncover が if 文の 0 以外の実行数を報告することを期待しても、コンパイラはこのコードを削除する可能性があるため、Uncover は計測中にそれを表示しません。そのため、これらの命令に対してカバレッジは報告されません。

例 2

次は、デッドコードの例です。

```
1 void foo()
2 {
3     A();
4     return;
5     B();
6     C();
7     D();
8     return;
9 }
```

B、C、D の呼び出しは実行されることがないため、対応するアセンブリではこのコードが削除されています。

```
foo:
                                .L900000109:
/* 000000          2 */      save    %sp,-96,%sp
/* 0x0004          3 */      call   A      ! params =      ! Result =
/* 0x0008          */      nop
/* 0x000c          8 */      ret     ! Result =
/* 0x0010          */      restore %g0,%g0,%g0
```

そのため、行 5 から 6 までに対してカバレッジは報告されません。

	Excl. Uncoverage	Excl. Function Count	Excl. Instr Exec	Excl. Block Covered %	Excl. Instr Covered %	
## 0		1	1	100	100	1. void foo() 2. {
## 0		0	2	0	0	<Function: foo 3. A(); 4. return; 5. B(); 6. C(); 7. D(); 8. return; 9. }
## 0		0	2	0	0	

例3

次は、冗長なコードの例です。

```

1 int g;
2 int foo() {
3     int x;
4     x = g;
5     for (int i=0; i<100; i++)
6         x++;
7     return x;
8 }

```

低い最適化レベルでは、コンパイラがすべての行にコードを生成する可能性があります。

```

foo:
                                .L900000107:
/* 000000    3 */          save   %sp, -112,%sp
/* 0x0004    5 */          sethi  %hi(g),%l1
/* 0x0008    */          ld     [%l1+%lo(g)],%l3 ! volatile
/* 0x000c    */          add   %l1,%lo(g),%l2
/* 0x0010    6 */          st     %g0, [%fp-12]
/* 0x0014    5 */          st     %l3, [%fp-8]
/* 0x0018    6 */          ld     [%fp-12],%l4
/* 0x001c    */          cmp   %l4,100
/* 0x0020    */          bge,a,pn %icc,.L900000105
/* 0x0024    8 */          ld     [%fp-8],%l1
                                .L17:
/* 0x0028    7 */          ld     [%fp-8],%l1
                                .L900000104:
/* 0x002c    6 */          ld     [%fp-12],%l3
/* 0x0030    7 */          add   %l1,1,%l2
/* 0x0034    */          st     %l2, [%fp-8]
/* 0x0038    6 */          add   %l3,1,%l4
/* 0x003c    */          st     %l4, [%fp-12]
/* 0x0040    */          ld     [%fp-12],%l5
/* 0x0044    */          cmp   %l5,100
/* 0x0048    */          bl,a,pn %icc,.L900000104
/* 0x004c    7 */          ld     [%fp-8],%l1

```

```

/* 0x0050      8 */      ld      [%fp-8],%l1
.L900000105:
/* 0x0054      8 */      st      %l1,[%fp-4]
/* 0x0058      */      ld      [%fp-4],%i0
/* 0x005c      */      ret     ! Result = %i0
/* 0x0060      */      restore %g0,%g0,%g0

```

高い最適化レベルでは、実行可能なソース行のほとんどに、対応する命令がありません。

```

foo:
/* 000000      5 */      sethi  %hi(g),%o5
/* 0x0004      */      ld      [%o5+%lo(g)],%o4
/* 0x0008      8 */      retl   ! Result = %o0
/* 0x000c      5 */      add    %o4,100,%o0

```

そのため、一部の行に対してカバレッジは報告されません。

Excl. Uncoverage	Excl. Function Count	Excl. Instr Exec	Excl. Block Covered %	Excl. Instr Covered %	
0	0	0	0	0	1. int g; 2. int foo() { <Function foo> 3. int x; 4. x = g; Source loop below has tag L1 Induction variable substitution performed on L1 L1 deleted as dead code 5. for (int i=0; i<100; i++) 6. x++; 7. return x; 8. }
## 0	1	3	100	100	
0	0	1	0	0	

索引

B

- bit.rc 初期化ファイル, 19
 - Discover に読み取らないように指示する, 18

D

Discover

- API, 36
- オプション
 - a, 16
 - c, 15, 17
 - D, 15, 18
 - E, 17
 - e, 17
 - F, 17
 - f, 17
 - H, 17, 20
 - h, 18
 - i, 18
 - K, 18
 - k, 18
 - l, 17
 - m, 17
 - N, 16, 18
 - n, 15, 17
 - o, 16
 - S, 17
 - s, 18
 - T, 16, 18
 - V, 18
 - v, 18

Discover, オプション (続き)

- w, 15, 16, 20
 - 概要, 9-10
 - 簡易モードで実行, 17
 - キャッシュされたライブラリの再計測を強制, 18
 - キャッシュディレクトリの指定, 18
 - 共有ライブラリの無視, 16, 18
 - 計測機構付きバイナリがフォークした場合の処理の指定, 17
 - 計測不可能なバイナリの計測を試みる場合は警告を発する, 18
 - コードアナライザで使用するためにエラータデータをディレクトリに書き込む, 16
 - 実行可能ファイルの書き込み専用計測の実行, 17
 - 指定されたバイナリのみを計測する, 18
 - 使用するための要件, 11-12
 - 冗長モードの指定, 18
 - 制限事項, 37-39
 - フォークの追尾, 19
 - メモリアクセスエラー, 31-34
 - メモリアクセスエラーの例, 32
 - メモリアクセスの警告, 34
 - ライブラリの完全な読み取り/書き込み計測の実行, 17
- ### Discover レポート
- ASCII, 28-31
 - エラーメッセージ, 29
 - 概要, 31
 - 書き込み, 16
 - 警告メッセージ, 30

Discover レポート, ASCII (続き)
 スタックトレース, 29, 31
 未解放ヒープブロック, 31
 メモリーリーク, 30
 割り当てられて残されているヒープブ
 ロック, 31
 HTML, 20-28
 「エラー (Errors)」タブ, 21-23
 書き込み, 17
 「警告 (Warnings)」タブ, 24
 コントロールパネル, 27-28
 スタックトレースの表示, 22, 24, 26
 すべての関数のソースコードの表示, 27
 すべてのスタックトレースの表示, 27
 ソースコードの表示, 23, 24, 26
 表示されるエラーのタイプの制御, 27
 表示される警告のタイプの制御, 28
 「メモリーリーク (Memory Leaks)」タ
 ブ, 25-26
 割り当てられて残っている残存ブロック数, 25
 エラーメッセージ、解釈, 34
 オフセットの表示, 17
 擬陽性, 34
 回避, 35
 スペキュレイティブロードにより発生, 35
 未計測コードで発生, 36
 表示されるスタックフレーム数の制限, 17
 符号化された名前の表示, 17
 報告されるメモリーエラー数を制限する, 17
 報告されるメモリーリーク数を制限する, 17
 Discover レポート
 擬陽性
 部分的に初期化されたメモリーによって発
 生, 34-35

S

SUNW_DISCOVER_FOLLOW_FORK_MODE 環境変数, 19
 SUNW_DISCOVER_OPTIONS 環境変数, 19, 20

U

Uncover
 オプション
 -a, 44
 -c, 42, 44
 -d, 42
 -e, 44
 -H, 44
 -h, 44
 -m, 43
 -n, 44
 -o, 43
 -t, 44
 -V, 44
 -v, 44
 概要, 10
 カバレッジレポート、生成, 43-44
 コードアナライザで使用するためにデータを
 ディレクトリに書き込む, 44
 コマンドの例, 44-45
 指定されたディレクトリにカバレッジデータ
 ディレクトリを作成する, 42
 指定されたファイルに計測機構付きバイナリ
 ファイルを書き込む, 43
 使用するための要件, 41
 冗長モードで実行, 44
 スレッドセーフなプロファイリングをオンまた
 はオフにする, 43
 制限事項, 57-60
 命令、ブロック、および関数の実行カウントの
 報告をオンにする, 42, 44
 Uncover ASCII カバレッジレポート, 51-54
 生成, 44
 Uncover HTML カバレッジレポート, 55-56
 保存, 44

き

共有ライブラリ
 Discover に無視するように指示する, 16, 18
 Discover によるキャッシュ, 15
 Discover を使用した計測, 15

ち

- 注釈付きではないコード
 - Discover の処理方法, 15
 - ソース, 15

よ

- 要件
 - Discover, 11-12
 - Uncover, 41

と

- ドキュメント、アクセス, 5-6
- ドキュメントの索引, 5

は

- バイナリ
 - Discover で使用できない, 12
 - Discover の計測, 14-19
 - Discover の準備, 11-12
 - Discover を使用した計測
 - 実行時動作の変更, 19
 - 特定のファイルへの書き込み, 16
 - Discover を使用して計測済み
 - 実行, 20
 - Uncover 用の計測, 42
 - Uncover を使用した計測、実行, 43
- バイナリの計測
 - Discover 用, 14-19
 - Discover を使用したデータ競合検出用, 18
 - Uncover 用, 42
- パフォーマンスアナライザの Uncover カバレッジレポート, 45-51
 - 「関数 (Functions)」タブ, 45-48
 - 「カバーされているブロックの割合 (%) (Block Covered %)」カウンタ, 47
 - 「カバーされている命令の割合 (%) (Instr Covered %)」カウンタ, 48
 - 「カバレッジ外 (Uncoverage)」カウンタ, 46-47
 - 「関数カウント (Function Count)」カウンタ, 47
 - 「命令の実行 (Instr Exec)」カウンタ, 47
 - 「逆アセンブリ (Disassembly)」タブ, 50
- 生成, 44
 - 「ソース (Souece)」タブ, 48-49
 - 「命令頻度 (Inst-Freq)」タブ, 51

