Oracle Solaris Studio 12.2 Discover および Uncover ユーザーズガイド

Part No: 821–2498
2010 年 9 月

ORACLE

110412@25097

目次

はじめに

『Oracle Solaris Studio 12.2 Discover および Uncover ユーザーズガイド』では、Sun メモリエラー探索ツール (Discover) ツールを使用して、SPARC® ベースのプラットフォーム上でコンパイルされたバイナリにおけるメモリー関連のエラーを検出し、コードカバレッジツール (Uncover) を使用して、アプリケーションのコードカバレージを測定する方法について説明します。

注 – この Oracle Solaris Studio のリリースは、SPARC および x86 ファミリ (UltraSPARC、SPARC64、AMD64、Pentium、Xeon EM64T) プロセッサアーキテクチャを使用するシステムをサポートしています。使用の Solaris オペレーティングシステムのバージョンに対するシステムのサポート状況は、ハードウェア互換性リスト (`http://www.sun.com/bigadmin/hcl`) をご参照ください。ここには、すべてのプラットフォームごとの実装の違いについて説明されています。

このドキュメントでは、x86 関連の用語は次のものを指します。

- 「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、AMD 64 または EM64T システムで、特定の 64 ビット情報を指します。
- 「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。

サポートされるシステムについては、ハードウェアの互換性に関するリストを参照してください。

対象読者

このマニュアルは、Discover を使用してプログラムにおけるメモリーアクセスエラーを検出し、Uncover を使用してコードカバレージを測定するプログラマ向けに設計されています。Discover および Uncover のユーザーには、Fortran、C、または C++ による開発経験を持ち、Oracle Solaris または Linux オペレーティングシステムと UNIX コマンドについてある程度の知識が必要です。

Oracle Solaris Studio のマニュアルへのアクセス

マニュアルには、次の場所からアクセスできます。

- マニュアルは、次に示すマニュアル索引のページからアクセスできます。http://www.oracle.com/technetwork/server-storage/solarisstudio/documentation/index.html。
- IDE の全コンポーネントのオンラインヘルプは、IDE 内の「ヘルプ (Help)」メニューだけでなく、F1 キー、および多くのウィンドウおよびダイアログにある「ヘルプ (Help)」ボタンを使用してアクセスできます。
- パフォーマンスアナライザのオンラインヘルプは、「ヘルプ (Help)」メニューだけでなく、F1 キー、パフォーマンスアナライザの多くのウィンドウおよびダイアログボックスにある「ヘルプ (Help)」ボタンを使用してアクセスできます。
- dbxtool および DLight のオンラインヘルプは、「ヘルプ (Help)」メニューだけでなく、F1 キー、およびこれらのツールの多くのウィンドウおよびダイアログボックスにある「ヘルプ (Help)」ボタンを使用してアクセスできます。
- dbxtool のオンラインヘルプは dbxtool の「ヘルプ (Help)」メニューからアクセスできます。

アクセシブルな製品マニュアル

マニュアルは、技術的な補足をすることで、ご不自由なユーザーの方々にとって読みやすい形式のマニュアルを提供しております。アクセシブルなマニュアルは次の表に示す場所から参照することができます。

マニュアルの種類	アクセシブルな形式と格納場所
マニュアルとチュートリアル	HTML 形式。http://docs.sun.com にある Oracle Solaris Studio 12.2 Collection - Japanese から選択
『Oracle Solaris Studio 12.2 リリースの新機能』(旧リリースのコンポーネント Readme ファイルに記載されていた情報)	HTML 形式。http://docs.sun.com にある Oracle Solaris Studio 12.2 Collection - Japanese から選択
マニュアルページ	インストール済み製品の man コマンドから
オンラインヘルプ	HTML 形式。IDE、パフォーマンスアナライザ、DLight、および dbxtool の「ヘルプ (Help)」メニュー、「ヘルプ (Help)」ボタン、および F1 キーを使用して表示。
リリースノート	HTML 形式。http://docs.sun.com にある Oracle Solaris Studio 12.2 Collection - Japanese から選択

関連するサードパーティの Web サイト参照

サードパーティの URL はこのマニュアルで参照され、追加の関連情報を提供します。

注 – このマニュアルで紹介する Oracle 以外の 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。

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` *filename* 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。

表 P–1 表記上の規則 (続き)

字体または記号	意味	例
\	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'`

コード例は次のように表示されます。

- 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]`

- Bourne シェルおよび Korn シェル

  `$ command y|n [filename]`

- Bourne シェルおよび Korn 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。

マニュアル、サポート、およびトレーニング

追加のリソースについては、次の Web サイトを参照してください。

- マニュアル (http://docs.sun.com)
- サポート (http://www.oracle.com/us/support/systems/index.html)
- トレーニング (http://education.oracle.com) – 左側のナビゲーションバーで Sun へのリンクをクリックしてください。

ご意見の送付先

マニュアルの品質や使いやすさに関するご意見やご提案をお待ちしています。間違いやその他の改善すべき箇所がありましたら、http://docs.sun.com で「Feedback」をクリックしてお知らせください。ドキュメント名とドキュメントの Part No.、および、可能な場合は章、節、ページ番号を記載してください。返答が必要な場合はお知らせください。

Oracle 技術ネットワーク (http://www.oracle.com/technetwork/index.html) では、Oracle ソフトウェアに関するさまざまなリソースを提供しています。

- 技術上の問題やソリューションについては、ディスカッションフォーラム (http://forums.oracle.com) を参照してください。
- 実践的なステップ・バイ・ステップのチュートリアルについては、Oracle By Example (http://www.oracle.com/technology/obe/start/index.html) を参照してください。
- サンプルコードのダウンロードについては、サンプルコード (http://www.oracle.com/technology/sample_code/index.html) を参照してください。

◆ ◆ ◆ 第 1 章

概要

『Oracle Solaris Studio 12.2 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。

Sun メモリエラー探索ツール (Discover)

Sun メモリエラー探索ツール (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、Solaris 10 5/08 オペレーティングシステムまたはそれ以降の Solaris 10 update を実行しているシステムで、Sun Studio 12、Sun Studio 12 Update 1、Oracle Solaris Express 6/10、Oracle Solaris Studio 12.2 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。

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ー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ージを計測できます。コードカバレージは、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ージ情報は、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、カバレー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コードカバレージの他の利点は、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ー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。

◆ ◆ ◆ 第 2 章

Sun メモリエラー探索ツール (Discover)

Sun メモリエラー探索ツール (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。

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、Solaris 10 5/08 オペレーティングシステムまたはそれ以降の Solaris 10 update を実行している SPARCベースまたは x86 ベースシステム上で Sun Studio 12、Sun Studio 12 Update 1、Oracle Solaris Studio 12.2 コンパイラ、または GCC for Sun Systems コンパイラバージョン 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。

Sun Studio または Oracle Solaris Studio コンパイラを使用する際には、`-O` オプションまたは `-xO[n]` オプションを使用することによって最適化してコンパイルを行う必要があります。GCC コンパイラを使用する際には、特別な最適化レベルは必要ありません。

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 (18 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 (35 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い

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。

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、検出したメモリー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 -02 test_UMR.c
% a.out
*p = 131464
% discover -w - a.out
% a.out
ERROR (UMR): accessing uninitialized data from address 0x50010 (4 bytes) at:
        main() + 0x54  <test_UMR.c:7>
                 4:    {
                 5:      //  UMR:  accessing uninitialized data
                 6:      int *p = (int*) malloc(sizeof(int));
                 7:=>    printf(**p = %d\n", *p);
                 8:      free(p);
                 9:    }
        _start() + 0x108
    block at 0x50010 (4 bytes long) was allocated at:
        malloc() + 0x220
        main() + 0x1c  <test_UMR.c:6>
                3:    int main()
                4:    {
                5:      //  UMR:  accessing uninitialized data
                6:=>    int *p = (int*) malloc(sizeof(int));
                7:      printf(**p = %d\n", *p);
                8:      free(p);
                9:    }
        _start() + 0x108
*p = 327704
*************** Discover Memory Report *****************
No allocated memory left on program exit.
DISCOVER SUMMARY:
        unique errors   : 1 (1 total, 0 filtered)
        unique warnings : 0 (0 total, 0 filtered)
FILTERS: type=ML:func=_rt_boot:depth=*; type=*:func__f90_esfw:depth=*;
type=ML:func=__f90_ssfw:depth=*;
```

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。

準備されたバイナリの計測

ターゲットバイナリを準備したら、次の手順はその計測です。計測は戦略的な場所にコードを追加して、Discover がバイナリの実行中にメモリー操作を追跡できるようにします。

`discover` 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` を計測済みの `a.out` で上書きします。

```
discover a.out
```

計測済みのバイナリを実行する場合、Discover はプログラムのメモリーの使用を監視します。実行時に、Discover は Web ブラウザで表示可能な HTML ファイル (この場合、デフォルトで `a.out.html`) にメモリーアクセスエラーを詳述するレポートを書き込みます。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。

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13 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、主要な実行可能ファイルを計測する際に、Discover はプログラムの実行時のようなコードを挿入し、Discover は共有ファイルが開かれるとすべて自動的に計測します。それらのファイルがプログラムに静的にリンクされているか、`dlopen()` によって動的に開かれているかは関係ありません。

プログラムで使用されるすべての共有ライブラリは、13 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(「18 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

`-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

`-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、13 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。

`-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。計測済みのバイナリは、`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 12.2: 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す。

`-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、-H、-l、-L、-m, -S および -w のリストに設定することによって、計測済み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。

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際には、-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 (17 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、20 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。

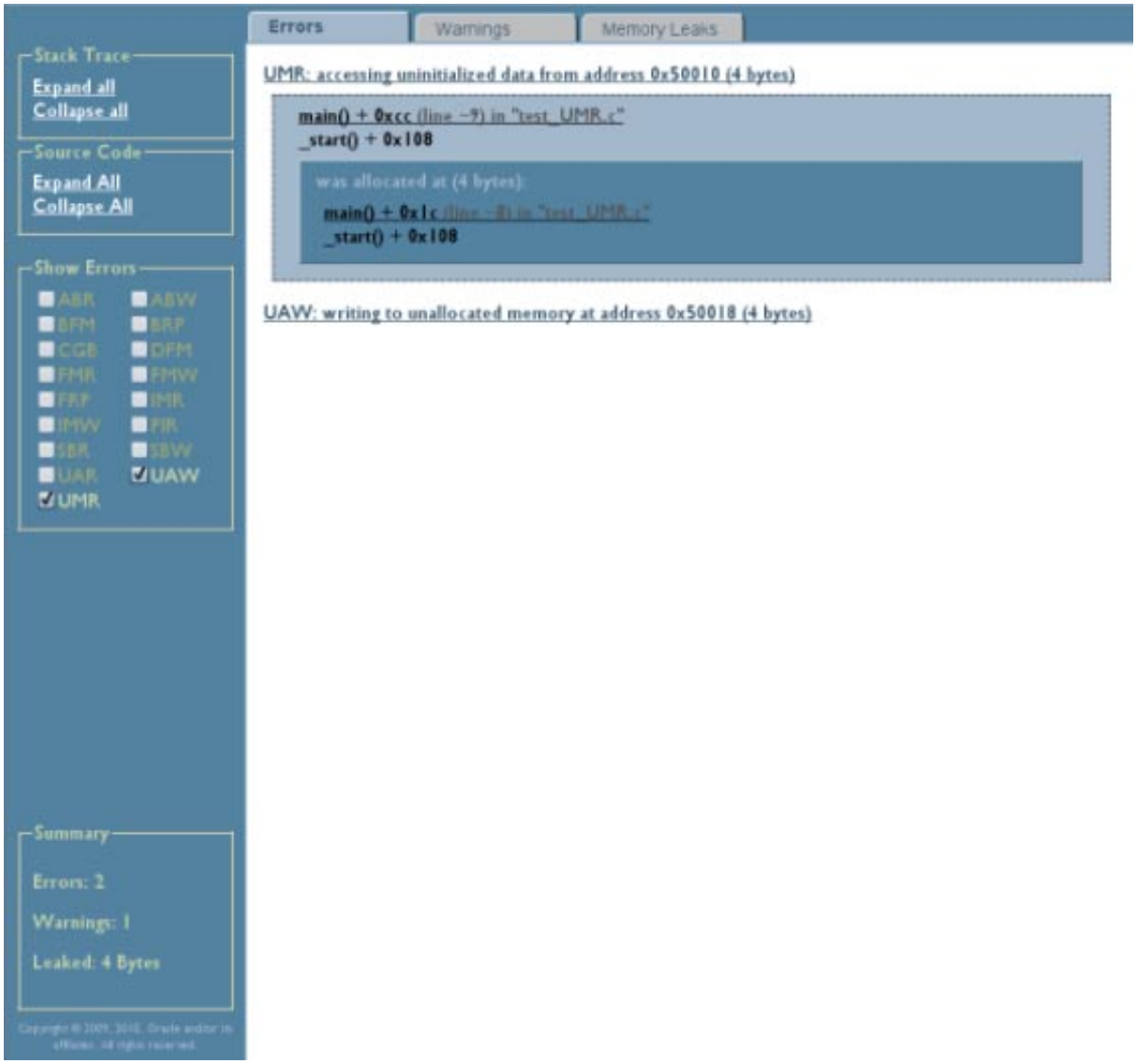
「エラー (Errors)」タブ (21 ページの「「エラー (Errors)」タブの使用法」を参照)、「警告 (Warnings)」タブ (24 ページの「「警告 (Warnings)」タブの使用法」を参照)、および「メモリーリーク (Memory Leaks) タブ」 (25 ページの「「メモリーリーク (Memory Leaks)」タブの使用法」を参照) では、エラーメッセージ、警告メッセージ、およびメモリーリークレポートをそれぞれ検索できます。

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。

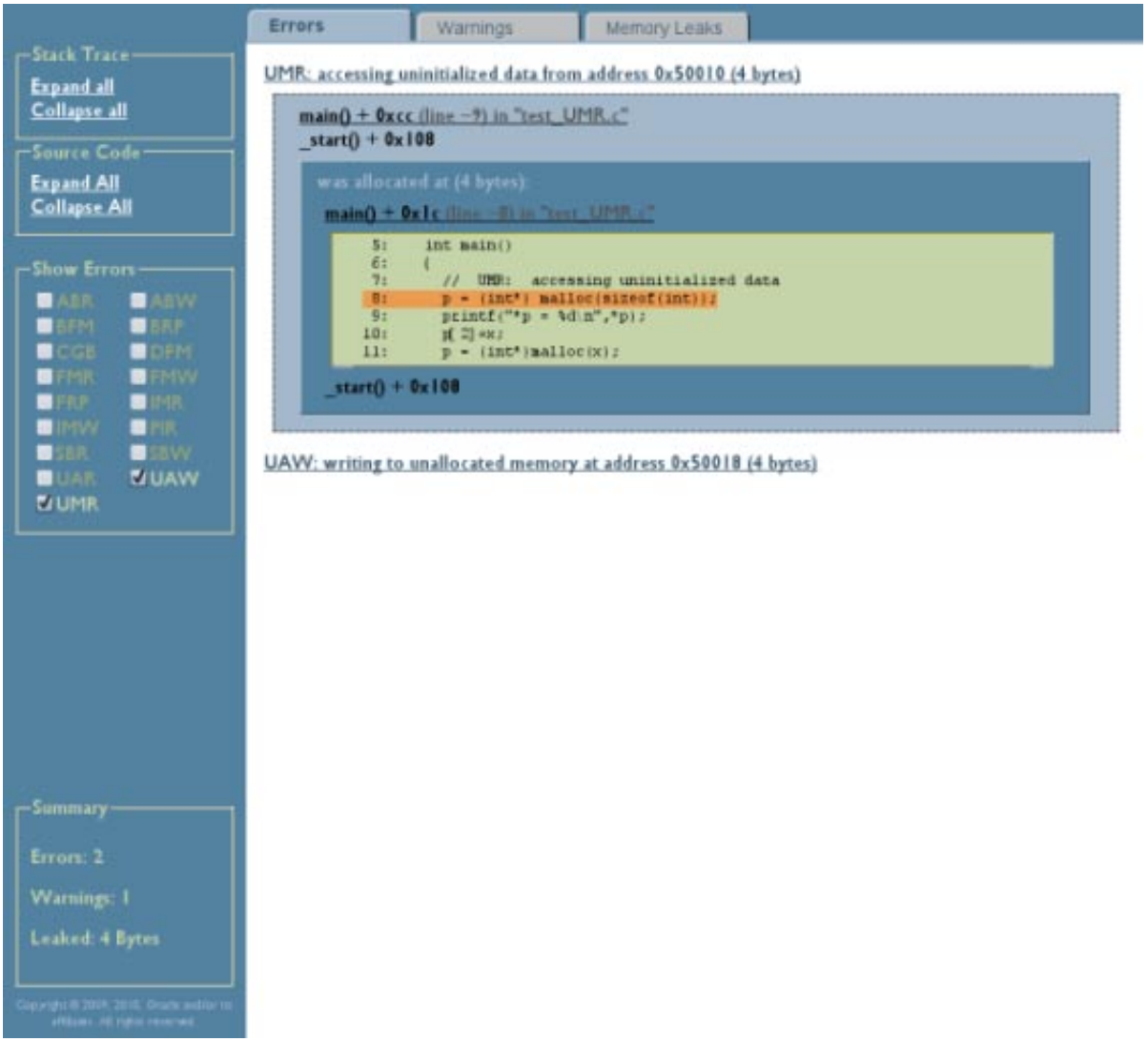
「エラー (Errors)」タブの使用法

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ーアクセスエラーのリストが表示されます。

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。

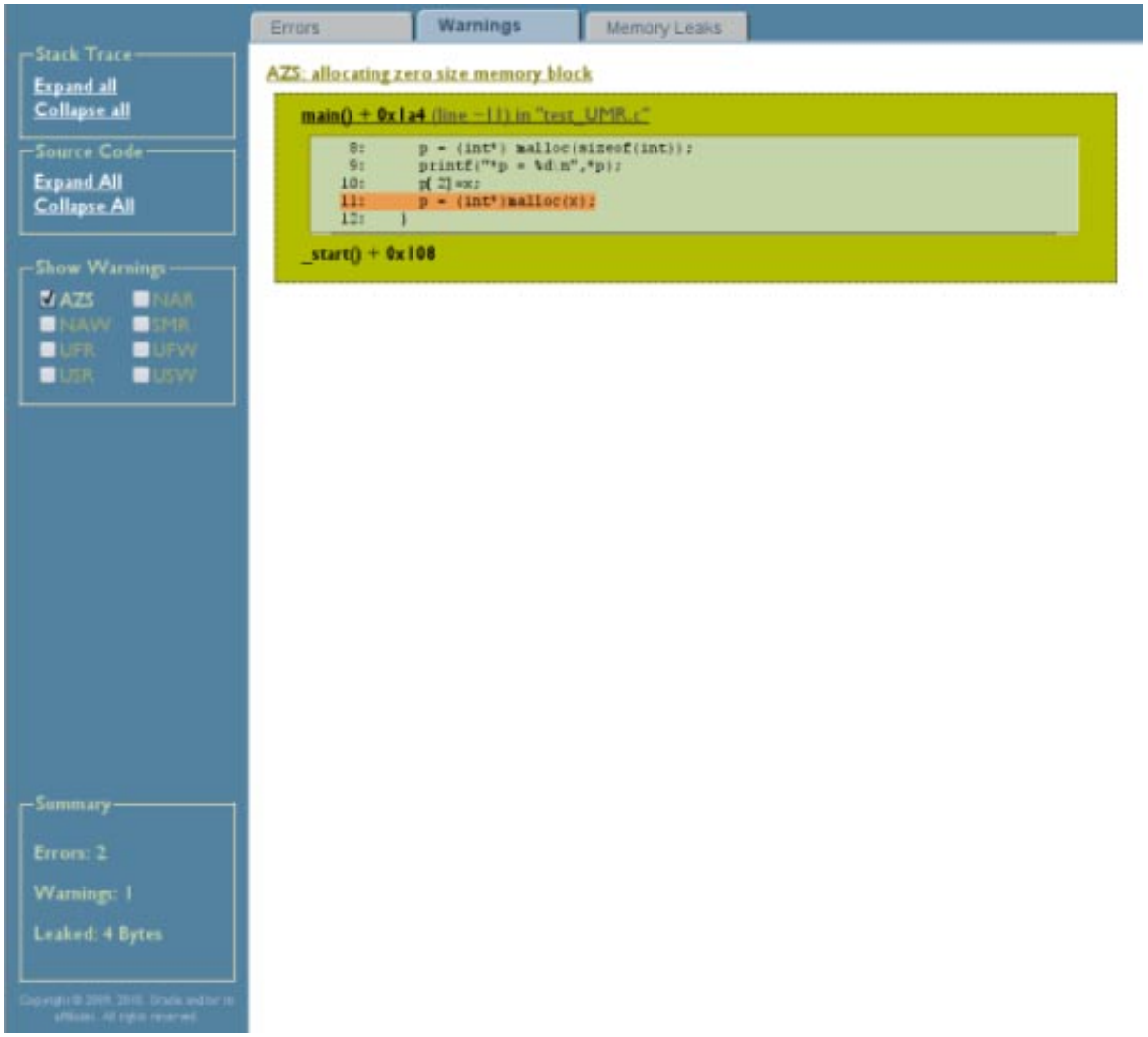


-g オプションを使用してコードをコンパイルした場合、関数をクリックすることによってスタックトレースの関数ごとのソースコードを表示できます。



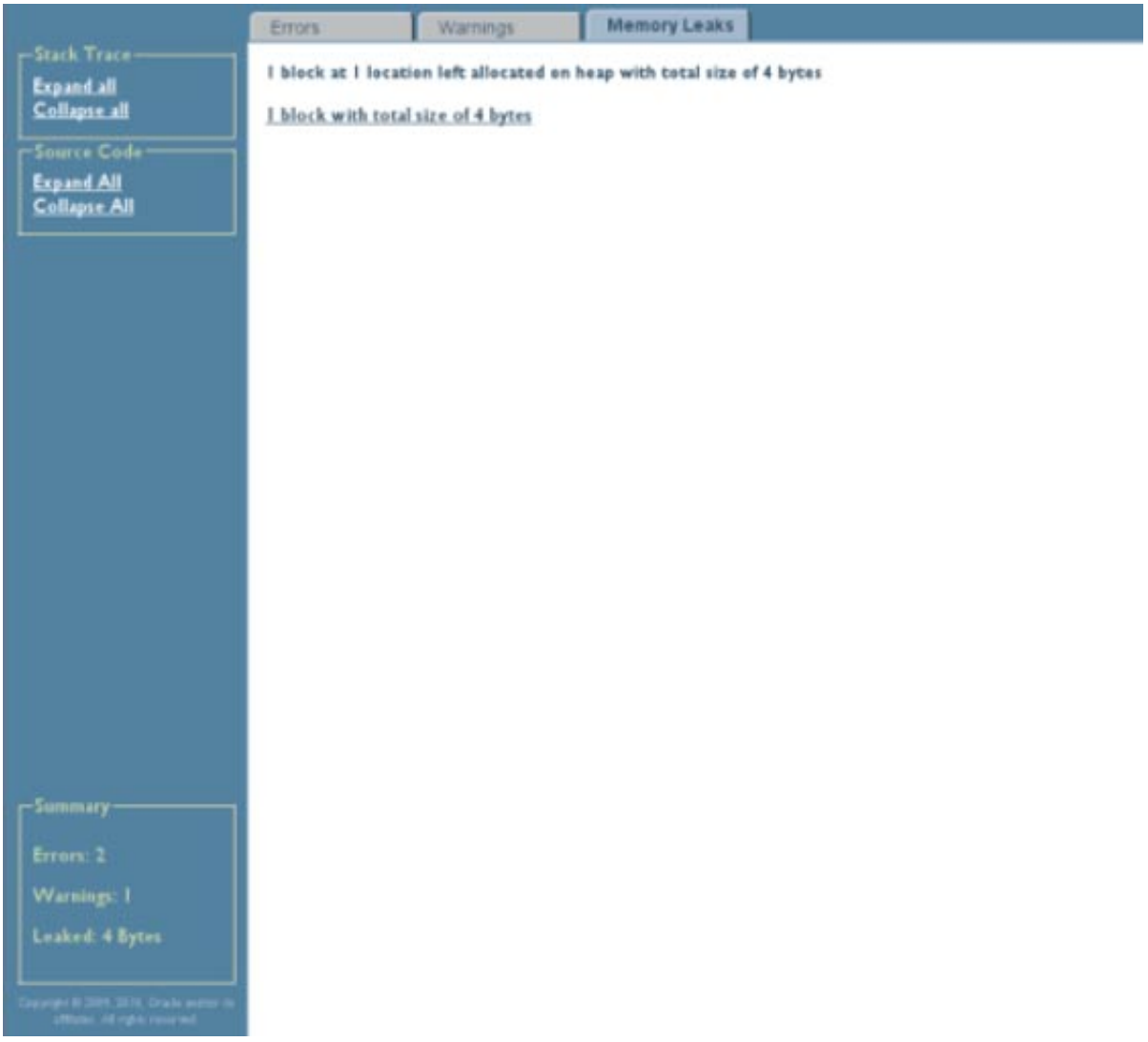
「警告 (Warnings)」タブの使用法

「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。

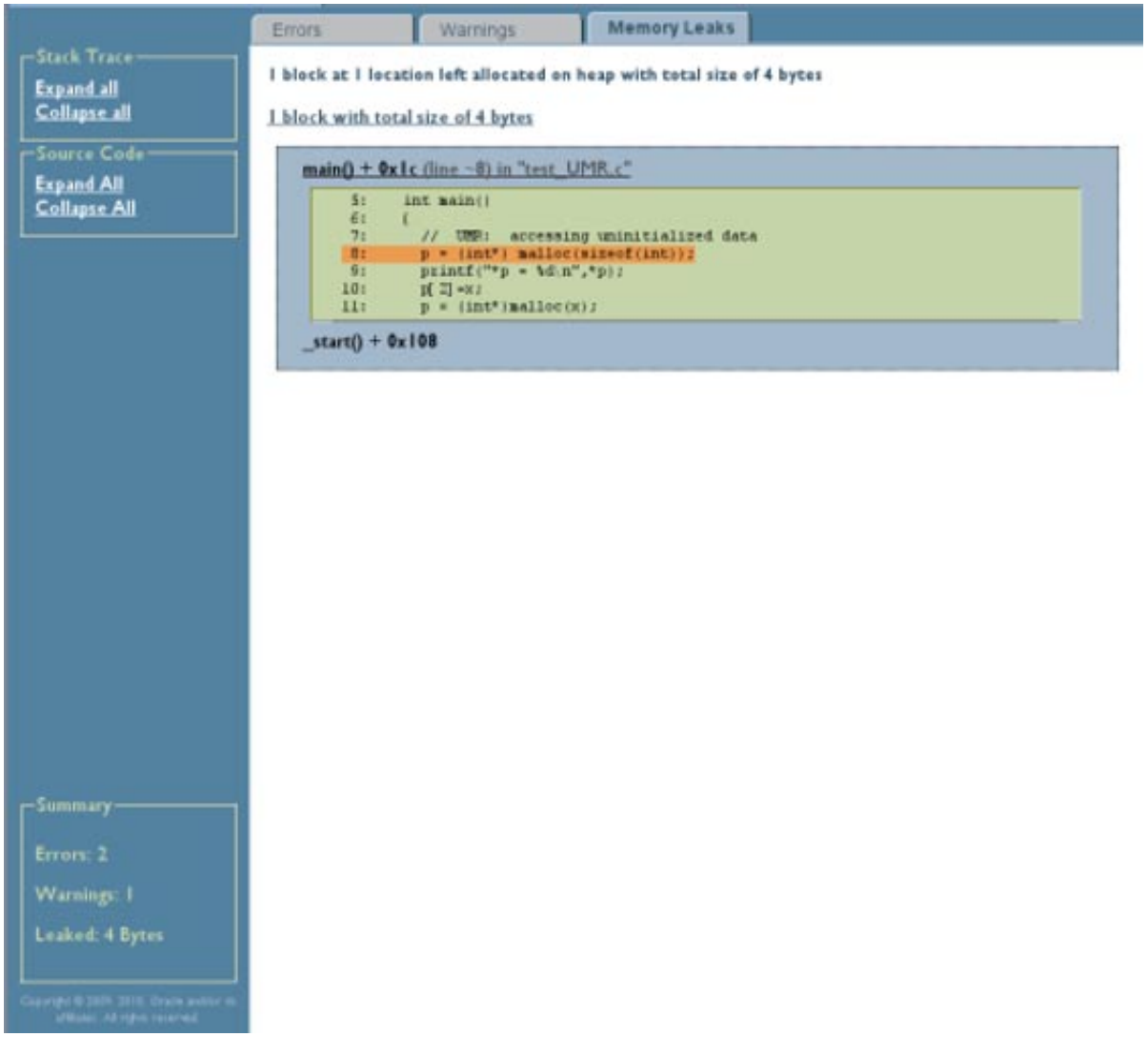


「メモリーリーク (Memory Leaks)」タブの使用法

「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。



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ること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、すなわちメモリー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。次に例を示します。

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

メモリーアクセスエラーと警告

Discover は多数のメモリーアクセスエラー、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。

メモリーアクセスエラー

Discover は次のメモリー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:不正なメモリーブロックの解放 (freeing the wrong memory block)
- 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)
- 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

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  //  ABR: reading memory beyond array bounds at address 0x%1x (%d byte%s)"
  int *a = (int*) malloc(sizeof(int[5]));
  printf("a[5] = %d\n",a[5]);
)
```

ABW

```
#include <stdlib.h>
int main()
{
  //  ABW:  writing to memory beyond array bounds
  int *a = (int*) malloc(sizeof(int[5]));
  a[5] = 5;
}
```

BFM

```
#include <stdlib.h>
int main()
{

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

BRP

```
#include <stdlib.h>
int main()
{
  //  BRP is "bad address parameter for realloc 0x%1x"
  int *p = (int*) realloc(0,sizeof(int));
  int *q = (int*) realloc(p+20,sizeof(int[2]));
}
```

DFM

```
#include <stdlib.h>
int main()
{
```

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

FMR

```
#includ <stdio.h>
#include <stdlib.h>
int main()
{
  //  FMR is "reading from freed memory at address 0x%1x  (%d byte%s)"
  int *p = (int*) malloc(sizeof(int));
  free(p);
  printf("p = 0x%h\n",p);
}
```

FMW

```
#include <stdlib.h>
int main()
{
  //  FMW is "writing to freed memory at address 0x%1x (%d byte%s)"
  int *p = (int*) malloc(sizeof(int));
  free(p);
  *p = 1;
}
```

FRP

```
#include <stdlib.h>
int main()
{
  //  FRP:  freed pointer passed to realloc
  int *p = (int*) malloc(sizeof(int));
  free(0);
  int *q = (int*) realloc(p,sizeof(int[2]));
}
```

IMR

```
#include <stdlib.h>
int main()
{
  //  IMR:  read from invalid memory address
  int *p = 0;
  int i = *p;   // generates Signal 11...
}
```

IMW

```
#include <stdlib.h>
int main()
```

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

PIR

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  //  PIR:  accessing partially initialized data
  int *p = (int*) malloc(sizeof(int));
  *((char*)p) = 'c';
  printf("*(p = %d\n",*(p+1));
}
```

UAR

```
#include <stdio.h>
#include <stdlib>
int main()
{
  //  UAR is "reading from unallocated memory"
  int *p = (int*) malloc(sizeof(int));
  printf("*(p+1) = %d\n",*(p+1));
}
```

UAW

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  // UAW is "writing to unallocated memory"
  int *p = (int*) malloc(sizeof(int));
  *(p+1) = 1;
}
```

UMR

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  // UMR is "accessing uninitialized data from address 0x%1x (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)
- NAR: 注釈の付かない領域からの読み取り (non-annotated read)
- NAW: 注釈の付かない領域への書き込み (non-annotated write)
- SMR: 投機的メモリーからの読み取り (speculative memory read)
- SMW: 投機的メモリーへの書き込み (speculative memory write)
- UFR: 不明なスタックフレームからの読み取り (unknown stack frame read)
- UFW: 不明なスタックフレームへの書き込み (unknown stack frame write)
- USR: 読み取り中の不明ステータス (unknown status while reading)
- USW: 書き込み中の不明なステータス (unknown status while writing)

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

```
#include <stdlib.h>
int main()
{
  //  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。次に例を示します。

```
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。次に例を示します。

```
...
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よくあります。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 lond 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ます。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。

```
#ifdef __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);

#prgama 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、「13 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、-O*n* 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あす。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。

◆ ◆ ◆ 第 3 章

3 コードカバレッジツール (Uncover)

Uncover を使用するための要件

Uncover は、Solaris 10 5/08 オペレーティング システム、またはそれ以降の Solaris 10 update を実行している SPARC ベースまたは x86 ベースのシステム上の少なくとも -O1 最適化オプションを使用した、Sun Studio 12 Update 1、Oracle Solaris Studio Express 6/10、Oracle Solaris Studio 12.2 コンパイラ、または GCC for Sun Systems 4.2.0 以降のコンパイラを使用してコンパイルされたバイナリ上で機能します。

上記のようにコンパイルされるバイナリには、Uncover がカバレー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ー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) ベースのカバレージ情報のみを使用します。

Uncover の使用法

Uncover を使用してカバレー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. カバレー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す。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、カバレー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
```

-d *directory* カバレージデータディレクトリを *directory* に作成するよう Uncover に指示することができます。このオプションは、複数のバイナリ用のカバレージデータを収集する場合に役立ち、カバレー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ージデータが同じカバレー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、カバレー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。

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ー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、コードカバレージデータは、Uncover が計測中に作成した .uc 接尾辞を持つカバレー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。カバレー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。

カバレージレポートの生成と表示

カバレージレポートを生成するには、カバレージデータディレクトリ上で uncover コマンドを実行します。次に例を示します。

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

このコマンドは、a.out.uc ディレクトリのカバレージデータから *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。

-e on | off	カバレー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*	カバレー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	カバレー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 カバレー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 カバレー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 カバレー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 カバレー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ージの実験 (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 カバレー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 コードカバレー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 カバレー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 コードカバレージレポートを作成します。

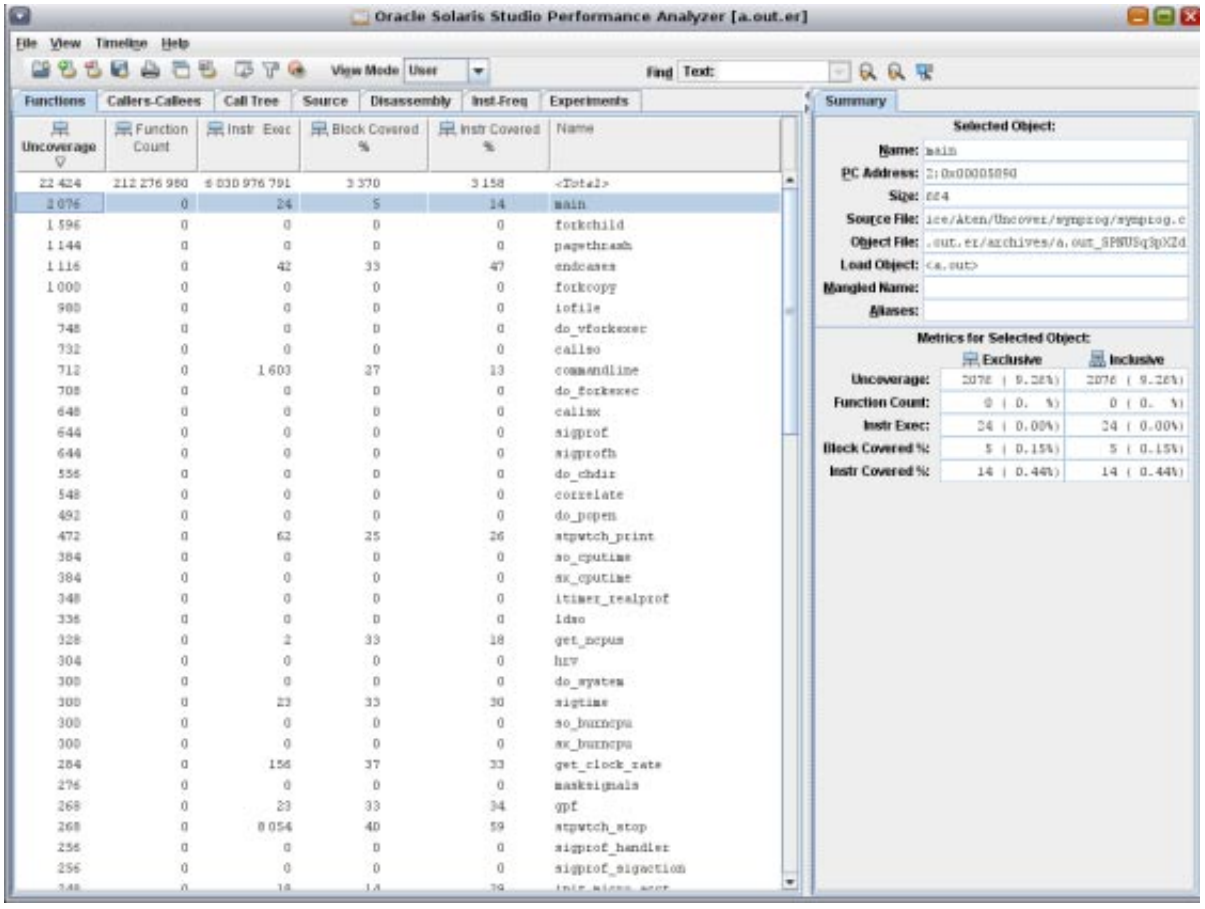
パフォーマンスアナライザのカバレージレポートを理解する

デフォルトでは、カバレージディレクトリ上の uncover コマンドを実行する場合、カバレージレポートが 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、カバレージデータを表示します。

「関数」タブ

アナライザでカバレー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「カバレージ外 (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。



「カバレージ外 (Uncoverage)」カウンタ

「カバレージ外 (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、カバレージを増やす可能性が最も高い関数です。例では、`main()` 関数が「カバレージ外 (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()` 関数のカバレージ外の数は、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。カバレー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、カバレー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、カバレー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。

「カバレージ外 (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、独自のカバレー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。

カバレー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

関数カウントは関数が呼び出された回数ですが、カバレージ分析のコンテキストでは、実際の数は重要ではありません。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、カバレージ外の数も 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)」カウンタには、動的な命令カウントが表示されます。「関数 (Functions)」タブに、関数ごとに実行される命令の総数が表示されます。このカウンタは、「ソース (Source)」タブ (47 ページの「「ソース」タブ」を参照) および「逆アセンブリ (Disassembly) タブ (48 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。

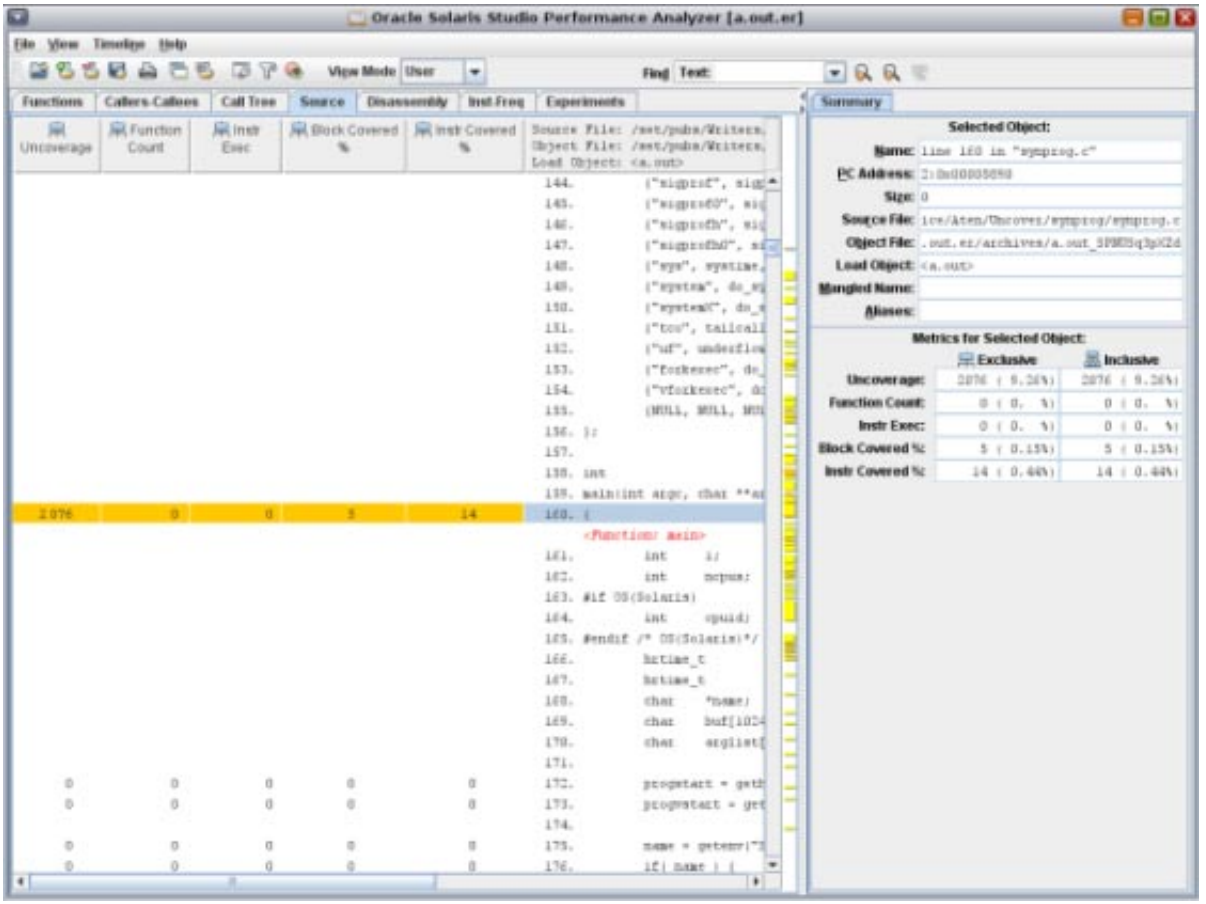
「カバーされている命令の割合 (%) (Instr Covered %)」カウンタ

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、このタブのカバレージ情報は解釈するのが困難な可能性があります。

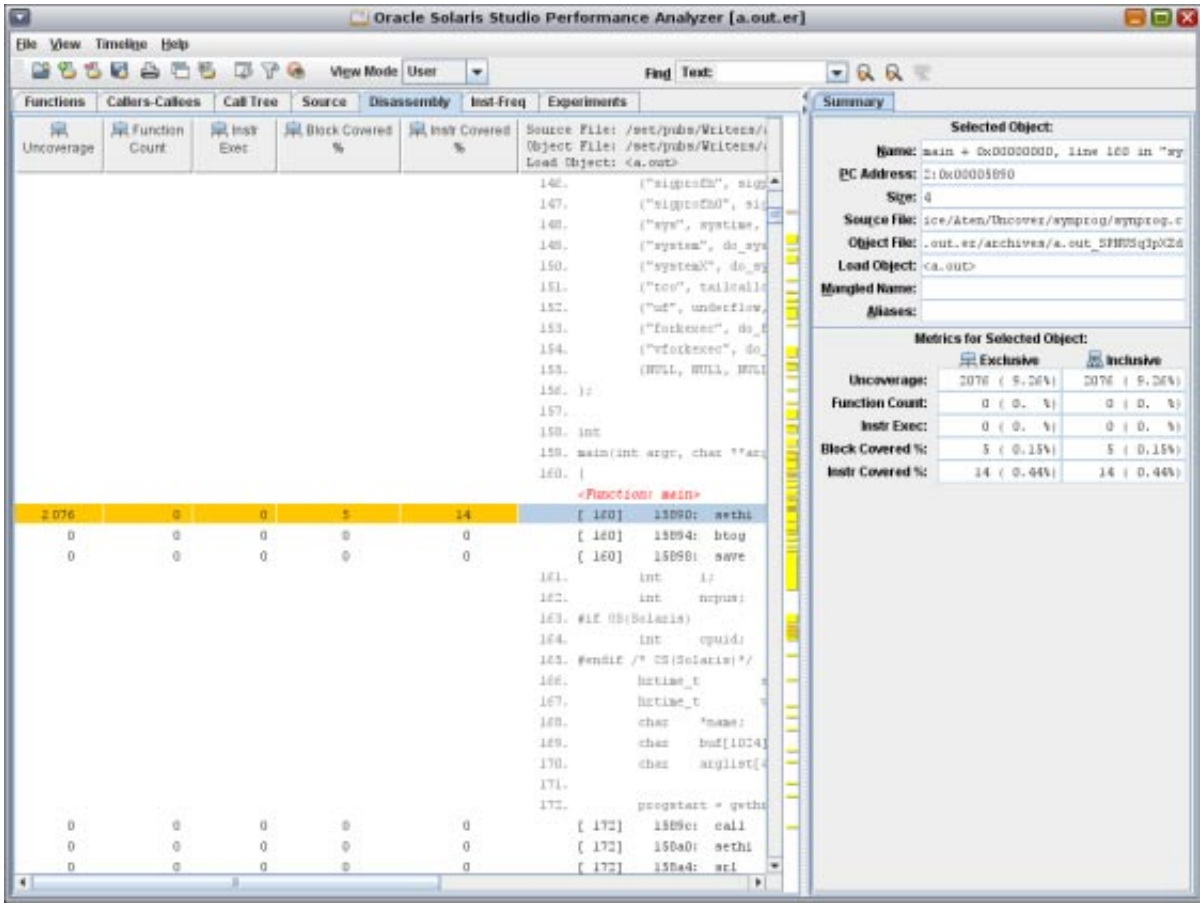
「ソース (Source)」タブの「命令の実行 (Instr Exec)」カウンタには、各ソース行に対して実行される命令の総数が表示され、これは基本的に文レベルのコードカバレージ情報です。0 以外の値は、文がカバーされていることを意味します。0 の値は、文がカバーされていないことを意味します。変数宣言およびコメントには命令の実行カウントがありません。



「逆アセンブリ」タブ

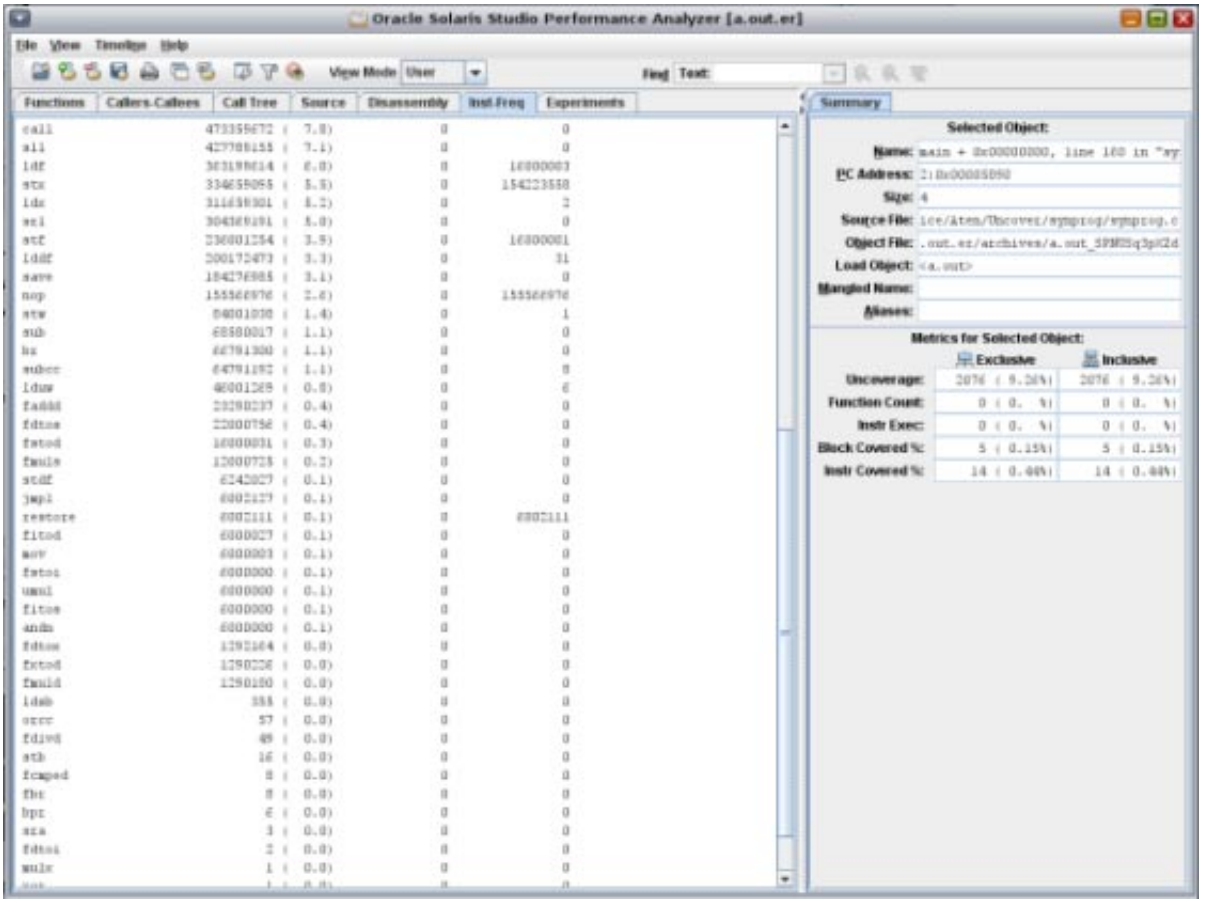
「ソース (Source)」タブのある行を選択し、次に「逆アセンブリ (Disassembly)」タブを選択する場合、アナライザは、バイナリで選択した行を検出し、その逆アセンブリを表示しようとします。

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。



「命令頻度」タブ

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



ASCII カバレージレポートを理解する

カバレージデータディレクトリからカバレー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 はカバレー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
```

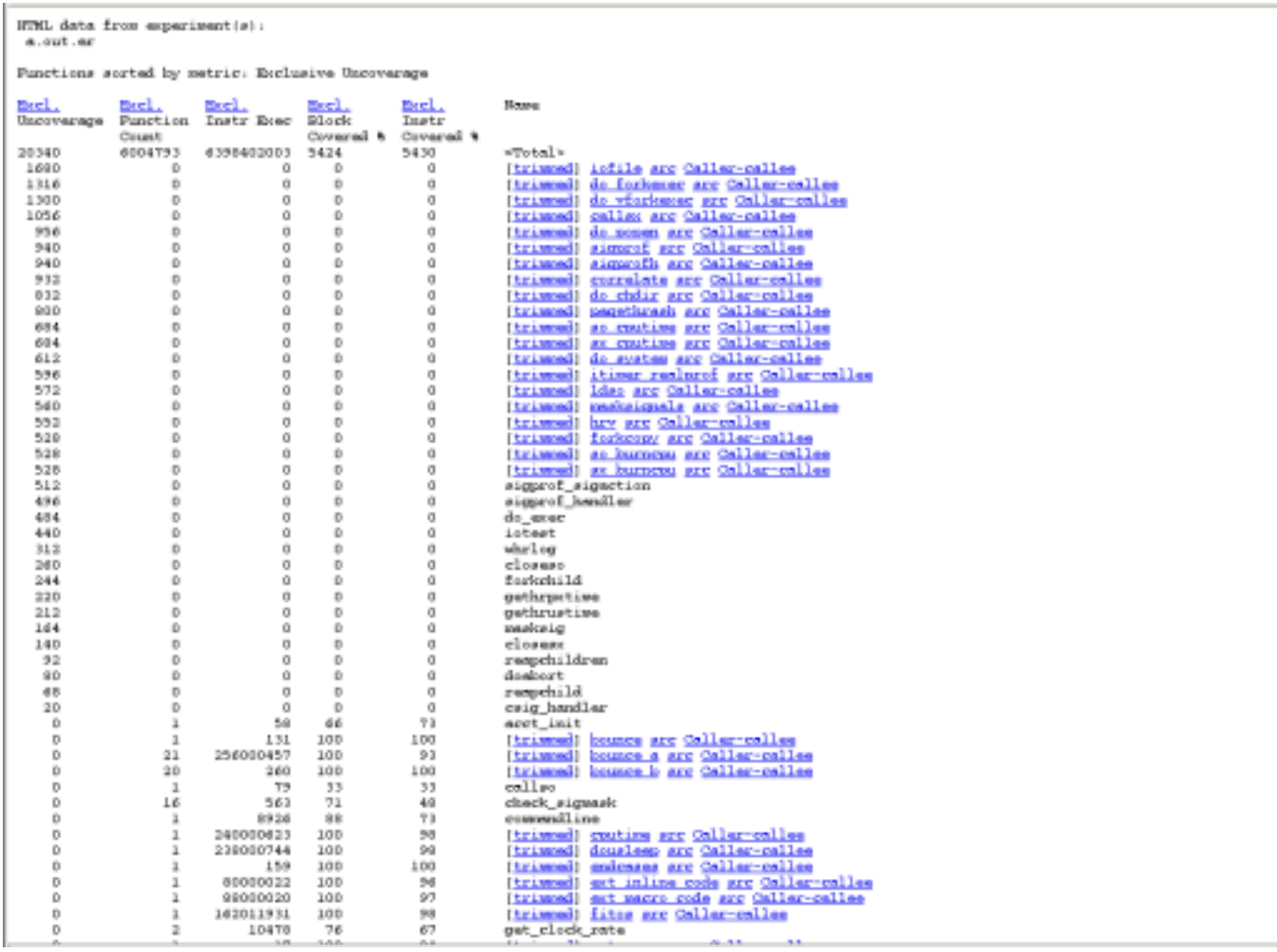
```
Instruction Coverage: 48.5%
Total Instruction Executions: 4,760,934,518
Average Executions per Instruction: 767,768.83
Number of times this program was executed: unavailable
Functions sorted by metric: Exclusive Uncoverage

Excl.       Excl.     Excl.      Excl.       Name
Uncoverage  Function  Block      Instr
            Count     Covered %  Covered %
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        recursedeep
    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        systime
    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
 faddd                    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 カバレー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。

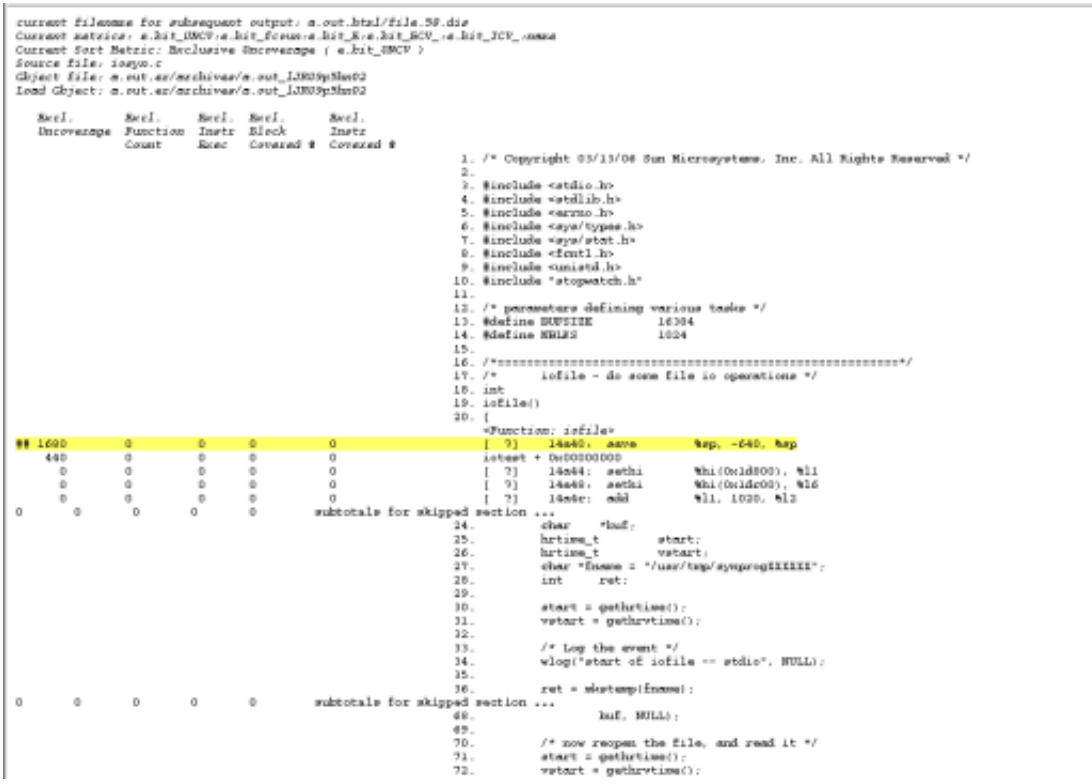


HTML data from experiment(s):
a.out.er

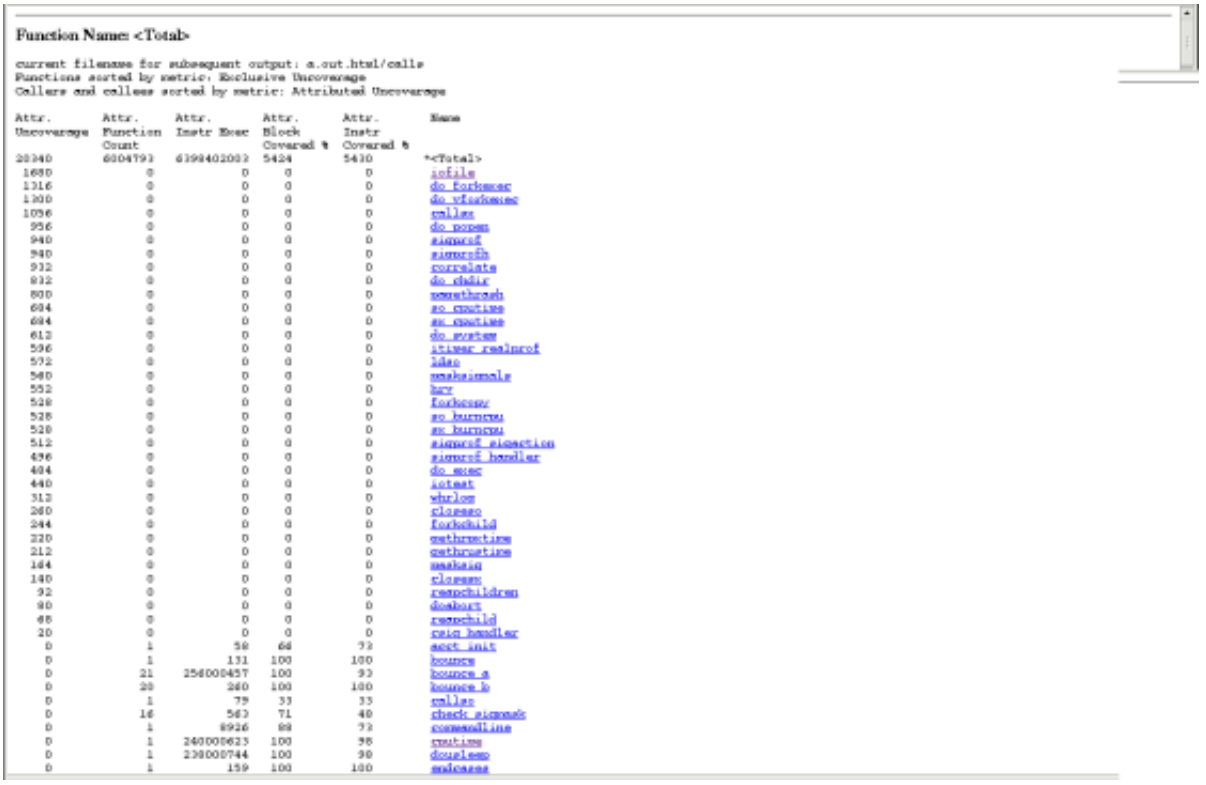
Functions sorted by metric: Exclusive Uncoverage

Excl. Uncoverage	Excl. Function Count	Excl. Instr Exec	Excl. Block Covered %	Excl. Instr Covered %	Name
20340	6004793	6398402003	5424	5430	<Total>
1680	0	0	0	0	[trimmed] isfile src Caller-callee
1316	0	0	0	0	[trimmed] do_forkmesg src Caller-callee
1300	0	0	0	0	[trimmed] do_vforkmesg src Caller-callee
1056	0	0	0	0	[trimmed] callso src Caller-callee
956	0	0	0	0	[trimmed] do_screen src Caller-callee
940	0	0	0	0	[trimmed] sigprof src Caller-callee
940	0	0	0	0	[trimmed] sigwatch src Caller-callee
932	0	0	0	0	[trimmed] correlate src Caller-callee
832	0	0	0	0	[trimmed] do_chdir src Caller-callee
800	0	0	0	0	[trimmed] segsthrash src Caller-callee
684	0	0	0	0	[trimmed] ex_routines src Caller-callee
684	0	0	0	0	[trimmed] ex_routines src Caller-callee
612	0	0	0	0	[trimmed] do_system src Caller-callee
596	0	0	0	0	[trimmed] itimer_realprof src Caller-callee
572	0	0	0	0	[trimmed] ldso src Caller-callee
560	0	0	0	0	[trimmed] emsksignals src Caller-callee
552	0	0	0	0	[trimmed] hrv src Caller-callee
520	0	0	0	0	[trimmed] forkcopy src Caller-callee
528	0	0	0	0	[trimmed] ex_burncpu src Caller-callee
528	0	0	0	0	[trimmed] ex_burncpu src Caller-callee
512	0	0	0	0	sigprof_sigaction
496	0	0	0	0	sigprof_handler
484	0	0	0	0	do_exec
440	0	0	0	0	isteat
312	0	0	0	0	whrlog
280	0	0	0	0	closeso
244	0	0	0	0	forkchild
220	0	0	0	0	gethrgetime
212	0	0	0	0	gethrustime
184	0	0	0	0	maskeig
140	0	0	0	0	closeso
92	0	0	0	0	reapchildren
80	0	0	0	0	doabort
48	0	0	0	0	reapchild
20	0	0	0	0	csig_handler
0	1	50	66	73	sect_init
0	1	131	100	100	[trimmed] bounce src Caller-callee
0	21	256030457	100	93	[trimmed] bounce_a src Caller-callee
0	20	260	100	100	[trimmed] bounce_b src Caller-callee
0	1	79	33	33	callso
0	16	563	71	40	check_sigmask
0	1	8926	88	73	commandline
0	1	240030823	100	98	[trimmed] cputime src Caller-callee
0	1	238030744	100	99	[trimmed] dousleep src Caller-callee
0	1	159	100	100	[trimmed] endsess src Caller-callee
0	1	80000022	100	98	[trimmed] set_inline_code src Caller-callee
0	1	98000020	100	97	[trimmed] set_macro_code src Caller-callee
0	1	162011931	100	98	[trimmed] fitse src 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。



ある関数の `Caller-callee` リンクをクリックする場合、呼び出し元-呼び出し先データが表示されます。



Function Name: <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
28340	6004793	6398402082	5424	5430	*<Total>
1680	0	0	0	0	iofile
1316	0	0	0	0	do_forkmesc
1300	0	0	0	0	do_vforkmesc
1056	0	0	0	0	caller
956	0	0	0	0	do_popen
940	0	0	0	0	sigmesf
940	0	0	0	0	sigmesfh
932	0	0	0	0	correlate
832	0	0	0	0	do_stdio
800	0	0	0	0	newthresh
684	0	0	0	0	mc_creatmes
684	0	0	0	0	mc_creatmes
612	0	0	0	0	do_system
596	0	0	0	0	itimer_realprof
572	0	0	0	0	ldso
560	0	0	0	0	enablesignals
552	0	0	0	0	hcr
528	0	0	0	0	forkmesc
528	0	0	0	0	mc_bumpcnt
520	0	0	0	0	mc_bumpcnt
512	0	0	0	0	sigmesf_sigaction
496	0	0	0	0	sigmesf_handler
484	0	0	0	0	do_mesc
440	0	0	0	0	iotest
312	0	0	0	0	whrlom
280	0	0	0	0	closesc
244	0	0	0	0	forkchild
220	0	0	0	0	gethrvtime
212	0	0	0	0	gethrvtime
184	0	0	0	0	mmksem
140	0	0	0	0	closesc
92	0	0	0	0	reapchildren
90	0	0	0	0	doabort
88	0	0	0	0	reapchild
20	0	0	0	0	cpic_handler
0	1	58	66	72	sscf_init
0	1	131	100	100	bounce
0	21	254000457	100	93	bounce_a
0	20	240	100	100	bounce_b
0	1	79	33	33	caller
0	16	563	71	40	check_signals
0	1	8926	88	72	commandline
0	1	240000623	100	95	routine
0	1	230000744	100	90	doublesc
0	1	159	100	100	mainsesc

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ージを検出し、このカバレー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ージを報告しないと思われる場合があります。たとえば、次のコードフラグメントを考えみましょう:

```
#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ージは報告されません。

索引

ま

よ