２００５情財第１１６９号

２００５年度　オープンソースソフトウェア
活用基盤整備事業

ＯＳＳを用いた計算機システムの信頼性向上を目指した
イベントトレース機能の開発

分岐命令トレーサ
使用説明書

２００６年８月

（株）日立製作所

～目次～

1 はじめに

1.1 目的

最近、エンタープライズ向けシステムへの Linux の適用ニーズが高まっている。エンタープライズ向けシステムでは、極限まで問題発生の可能性を低減させる必要がある。また、万が一の問題発生の際にも、素早い原因究明が要求される。このような要求に対応するためには、カーネルや実行ファイルを変更することが必要となる場合が多いが、あらゆる制限により、これらを変更できない場合も多い。

このような要求を満足するため、カーネルや実行ファイルを変更しない事および、以下を目的として、分岐命令トレーサの開発を行った。

・動作確認において、各分岐条件を網羅できているか、確認できること

・問題が発生した場合にも、再現環境において障害発生箇所までの実行経路を詳細に表示することで、原因調査を支援できること

1.2 スコープ

分岐命令トレーサの適用範囲を表 1-1 に示す。

表 1-1　分岐命令トレーサの適用範囲

#	項目		適用範囲	備考
1	ハードウェア	アーキテクチャ	IA-32	
2		CPU	Pentium4、Pentium4 ベースの Xeon	UP、SMP、HT に対応
3	ディストリビューション		RHEL4U1、RHEL4U3	
4	カーネル		2.6.9-11.ELsmp 、2.6.9-34.ELsmp	procfs、kallsyms、apic サポートが ON であること
5	ライブラリ		binutils	

2 分岐命令トレーサ概要

2.1 分岐命令トレーサとは

分岐命令トレーサ（以降、btrax と称する）は、カーネルおよび実行ファイルを変更することなく、下記の情報を取得できるツールである。

・分岐カバレッジ情報

・実行経路情報

分岐トレース情報は、Pentium4 および Pentium4 ベースの Xeon プロセッサのデバッグ機能である、最新分岐記録機能（本機能については、Intel のマニュアルを参照のこと）を利用して取得する。

btrax では、トレース対象とする空間（主にユーザ空間の実行をトレース／カーネル空間とユーザ空間の全ての実行をトレース）や、プロセス（特定プロセスのみ／全プロセス）を使い分けることができる。本機能は下記のフック処理をカーネルに動的に埋め込むことにより実現している。

(1) システムコールの入口でトレース OFF にし、出口でトレース ON にする
　　→　カーネル空間でのトレース停止

(2) プロセススイッチ時に、対象プロセスかどうかを判断し、トレース ON/OFF を切替え
　　→　対象以外のプロセスでのトレース停止

トレース結果は、CPU 毎のファイルとして保存されるが、プロセス単位のトレース情報を確認したい場合は、プロセス毎のファイルに変換することができる。

分岐命令トレーサの構成を図 2-1 に示す。また、各構成要素の概要を表 2-1 に示す。

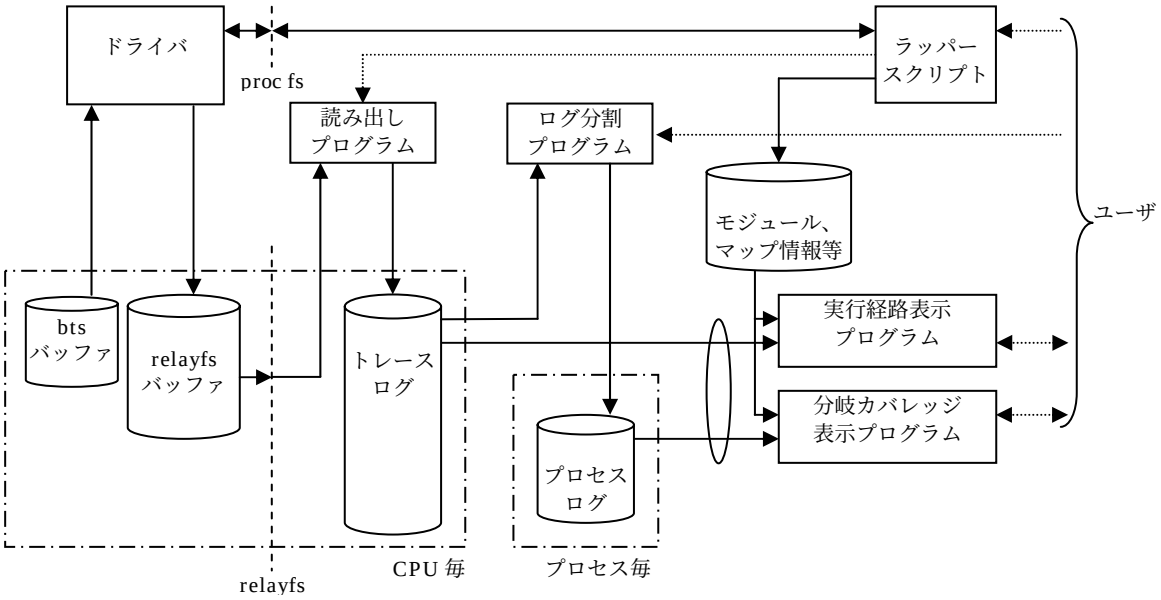


図 2-1　分岐命令トレーサ構成

表 2-1　分岐命令トレーサの構成要素概要

#	構成要素	種別	概要
1	ドライバ	プログラム	プロセッサの機能を用いて分岐情報を収集する。分岐トレーサ本体のドライバ以外に、以下のカーネルモジュールを含む ・ctr（システムコール入口/出口への処理追加用） ・djprobe（フック処理追加用） ・relayfs（カーネル側からユーザ側への大容量データ渡し用）
2	bts バッファ	バッファ	プロセッサが分岐情報を書き出すバッファ。一定容量（スレッシュホールド）を超えると割り込みが発生し、それによってドライバが bts バッファから relayfs バッファへコピーを行う。スレッシュホールドおよびバッファサイズは調整可能
3	relayfs バッファ	バッファ	カーネル側からユーザ側にトレース結果を渡す際のバッファ。プロセッサ毎に多面バッファを持つことができ、面数およびバッファサイズは調整可能

4	ラッパースクリプト	スクリプト	ユーザが簡単な操作で使用できるようにするためのスクリプト
5	読み出しプログラム	プログラム	カーネル側からユーザ側にトレース結果を読み出す。トレース結果は、プロセッサ毎のファイルとなる
6	ログ分割プログラム	プログラム	プロセスモードでのトレース結果を解析する際に、プロセッサ毎のログを、プロセス毎のファイルに分割する
7	トレースログ	ファイル	読み出しプログラムが出力するファイル(プロセッサ毎)
8	モジュール、マップ情報等	ファイル	ログ解析時に必要な情報を各ファイルに保存したもの。ラッパースクリプトが収集する
9	プロセスログ	ファイル	ログ分割プログラムが出力するファイル(プロセス毎)
10	実行経路表示プログラム	プログラム	ログおよびモジュール、マップ情報を解析し、実行経路として表示する
11	分岐カバレッジ表示プログラム	プログラム	ログおよびモジュール、マップ情報を解析し、分岐カバレッジ情報として表示する

2.2 トレース対象とトレース手順の概要

btrax では、トレース対象がカーネルかアプリケーションかによって、必要な手順が異なる。表 2-2 にトレース対象とトレース手順の概要を示す。

表 2-2　トレース対象とトレース手順概要

#	トレース対象		バイナリ再構築（ソース改変）の必要性	トレース手順の概要
	カーネル/アプリ	トレース範囲		
1	カーネル	任意の部分だけをトレース。例えば、ある関数だけトレース	なし	1. トレース開始／終了フック位置の確認 2. --start,--stop の両方を指定し、スクリプト実行
2	カーネル	ある特定のシンボル通過からバッファ分だけトレース	なし	1. トレース開始フック位置の確認 2. --start を指定し、スクリプト実行
3	カーネル	ある特定のシンボル通過までをバッファ	なし	1. トレース終了フック位置の確認 2. --stop を指定し、スクリプト実行

		分だけトレース		
4	アプリ	実行中のアプリをトレース	なし	1. -p で pid を指定し、スクリプト実行
5	アプリ	アプリの実行直後からの全挙動をトレース	フック関数追加要（開始点）	1. main 関数の先頭に、トレース開始関数を埋め込み、バイナリを再構築 2. -c でコマンド（およびその引き数）を指定し、スクリプト実行
6	アプリ	任意の部分だけをトレース	フック関数追加要（開始、終了点）	1. 任意の部分の入口と出口にそれぞれトレース開始／終了関数を埋め込み、バイナリを再構築 2. -c でコマンド（およびその引き数）を指定し、スクリプト実行

2.3 前提条件および制限事項

・プリエンプション OFF（ドライバで使用している djprobe の制限）であること。
・先にプロセッサの機能の一つである、性能モニタリング（PEBS）機能が有効になっていた場合は、使用不可。（プロセッサの仕様により、性能モニタリング機能と分岐命令トレース機能が同じメモリ領域で管理されるため）
・procfs、kallsyms、apic がカーネルでサポートされていること。
・トレース情報の発生頻度が高い場合、読み出しが追いつかず、データが欠落する場合がある。分岐命令トレーサは、データが欠落した場合も、データ読み出しを続行する。

3 分岐命令トレーサ動作環境構築手順

3.1 前準備

ドライバのコンパイルのため、kernel-devel-2.6.9-11.EL パッケージが必要である。また、分岐命令トレーサによるトレース結果解析のため、非圧縮のカーネル（vmlinux）も必要である。非圧縮カーネルは RHEL4U1 の場合、kernel-debuginfo-2.6.9-11.EL に含まれている。これらのパッケージを入手しインストールする。インストールは、スーパーユーザ権限で行う必要がある。（以降、スーパーユーザ権限で実行する場合と、一般ユーザ権限で実行する場合のプロンプトを、それぞれ#と$で示す）

```
# rpm -i kernel-devel-2.6.9-11.EL.i686.rpm
# rpm -i kernel-debuginfo-2.6.9-11.EL.i686.rpm
```

また、binutils も必要となる。binutils は、通常システムインストール後からインストールされているので、インストール済みであることを確認する。インストールされていない場合には、下記名称のパッケージを入手し、上記と同様の手順でインストールする。

```
$ rpm -qa|grep binutils
binutils-2.15.92.0.2-13
```

3.2 インストール

分岐命令トレーサのインストールは、下記の手順による。

(1) 分岐命令トレーサを展開

XXX はバージョン番号

```
$ tar jxvf btrax-XXX.tar.bz2
```

(2) コンパイル

```
$ cd btrax
$ make
```

(3) インストール（スーパーユーザで行う）

```
# make install
```

4 使用例

分岐命令トレーサの使用例を以下に示す。各コマンドの詳細や出力書式等については、以降の章を参照のこと。

4.1 アプリケーションのカバレッジ確認

カバレッジ確認の一例として、DAVL（ディスク割り当て評価ツール）のカバレッジ確認手順を示す。

(1) 前準備その１～relayfs マウントポイント作成

```
# mkdir /mnt/relay
```

(2) 前準備その２～ソースへのトレース開始関数の追加とコンパイル

DAVL コマンドは、小さいパーティションを対象として実行すると、即座に終了してしまうため、pid を取得してからトレースを開始するという手順を踏むことができない。このため、DAVL コマンドの main 関数にトレース開始関数（下記青太字部分）を追加する。ソース追加が終わったら、コンパイルして実行ファイルを作成しておく。

```
# vi $(where_cdavl_rebuild)/cdavl.c
#include <stdio.h>
#include "btrax/bt_for_ap.h"

（省略）
int main(int argc, char* argv[]) {
    int                         rc, ecode = 0;
    t_prog_const                prog_c;
    t_e2info            fs;
    t_chk                       chk;

    bt_start_from_ap();
    /* initialize local variable */
    bzero(&prog_c, sizeof(t_prog_const));
    bzero(&fs, sizeof(t_e2info));
    bzero(&chk, sizeof(t_chk));
（以下、省略）
```

注：コンパイル時に最適化オプションが含まれている場合、トレース結果表示のソース行番号がずれてしまうため、可能な限り最適化オプションは外すこと。

(3) DAVL コマンドを引き数に指定して、トレースを開始

```
# bt_collect_log -d $(ログ出力ディレクトリ)  -c cdavl -Tv /dev/hda5
```

(4) 別のシェルで DAVL コマンド終了を確認してから、トレースを終了

```
# ps -u root|grep cdavl
（cdavlの実行が終了しており、何も表示されないことを確認する）
#  （bt_collect_logを実行したシェルに戻って、Ctrl-Cを入力）
```

(5) ログをプロセス単位に分割

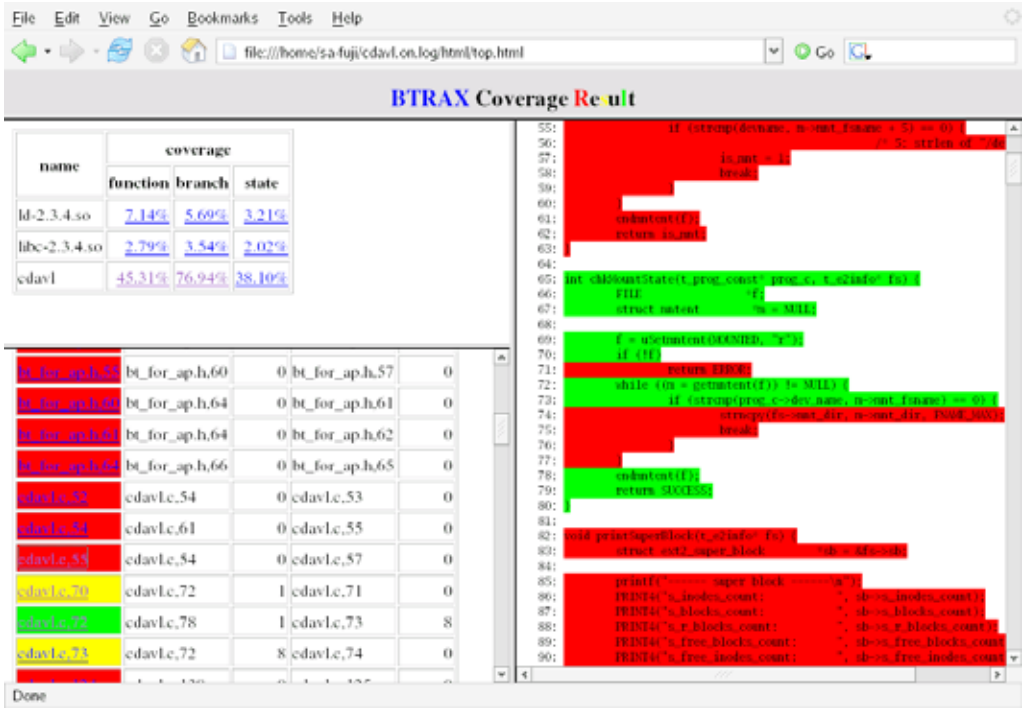
```
# cd $（ログ出力ディレクトリ）
# bt_split -d .
```

(6) ソース html でカバレッジを確認

5245 は、bt_split 実行により生成されたプロセス単位のログファイル

```
# bt_coverage --usr -f 5245 -o html
# firefox file:///$(ログ出力ディレクトリ）/html/top.html
```

html 上のリンクを辿ることで、下図のようなソース html が表示される（赤色が実行されていない行を、緑色が実行された行を示す）



4.2 ドライバの実行経路確認

実行経路確認の一例として、ext3 ドライバの ext3_unlink 関数の実行経路を確認する手順を示す。

(1) 前準備～削除するファイルを用意

```
# touch hogehoge
```

(2) フックポイントの確認

ext3_unlink のアドレスと djprobe でプローブ挿入しても問題ないか確認する。djprobe がプローブ可能なコードの条件は、djprobe のドキュメントを参照のこと。

```
# grep ext3_unlink /proc/kallsyms
f88dceb5 t ext3_unlink  [ext3]
# objdump -S /lib/modules/`uname -r`/kernel/fs/ext3/ext3.ko|grep -A 10 ext3_unlink|head
00008eb5 <ext3_unlink>:
    8eb5:       55                              push   %ebp
    8eb6:       57                              push   %edi
    8eb7:       89 c7                           mov    %eax,%edi
    8eb9:       56                              push   %esi
    8eba:       53                              push   %ebx
```

この 5 バイトが jmp コードに置き換わる

上記より、プローブアドレスは 0xf88dceb5、サイズは 5 であることが分かる。

(3) トレース開始（ext3_unlink 関数の通過待ち）

```
# bt_collect_log --START 0xf88dceb5,5 -d $(ログ出力ディレクトリ)
```

(4) 手順(1)で準備しておいたファイルを削除

```
（以下、他のシェルから実行する）
# rm hogehoge
```

(5) フックアドレス通過の確認とトレースの終了

```
（以下、他のシェルから実行する）
# cat /proc/btrax/cpu*/on_off_cnt
1
（bt_collect_logを実行したシェルに戻って、Ctrl-Cを入力）
```

いずれかの CPU の ON/OFF カウンタ値が 1 になっていることを確認してから、Ctrl-C

を入力しトレースを終了する。

(5) 実行経路を確認

```
# bt_execpath --ker -f $(ログ出力ディレクトリ）/cpu0
checking /lib/modules/2.6.9-11.ELsmp/kernel/fs/ext3/ext3.ko...
checking /usr/lib/debug/lib/modules/2.6.9-11.ELsmp/vmlinux...
start
（省略）
--------      0xf6858f44
ext3.ko       0x00008eba        <ext3_unlink+0x5>  push   %ebx
ext3.ko       0x00008eca        <ext3_unlink+0x15> jne                    0x00008ed4
<ext3_unlink+0x1f>
ext3.ko       0x00008ed4        <ext3_unlink+0x1f> testb  $0x3,0xac(%edx)
ext3.ko       0x00008edb        <ext3_unlink+0x26> je                     0x00008eee
<ext3_unlink+0x39>
ext3.ko       0x00008eee        <ext3_unlink+0x39> mov    0xa4(%edi),%eax
ext3.ko       0x00008ef9        <ext3_unlink+0x44> call                   0x00008efa
<ext3_journal_start_sb>
ext3.ko       0x000096c8        <ext3_journal_start_sb>     testb  $0x1,0x34(%eax)
ext3.ko       0x000096df        <ext3_journal_start_sb+0x17>         je
0x000096f6 <ext3_journal_start_sb+0x2e>
ext3.ko       0x000096f6        <ext3_journal_start_sb+0x2e>         jmp
0x000096f7 <journal_start>
--------      0xf88a0404
--------      0xf88a04a1
ext3.ko       0x00008efe        <ext3_unlink+0x49> mov    %eax,%esi
ext3.ko       0x00008f1f        <ext3_unlink+0x6a> je                     0x00008f25
<ext3_unlink+0x70>
ext3.ko       0x00008f25        <ext3_unlink+0x70> movl   $0xfffffffe,(%esp)
ext3.ko       0x00008f32        <ext3_unlink+0x7d> call                   0x00006ef5
<ext3_find_entry>
（省略）
```

カーネル空間の実行を表示する--ker
オプションを指定して実行経路を表示

5 プログラム仕様

5.1 ラッパースクリプト

ラッパースクリプト（bt_collect_log）は、トレース情報収集のためのユーザ操作を簡略化するためのスクリプトである。

bt_collect_log は、スクリプト実行直後から、Ctrl-C キーで終了するまでの間、トレース対象の分岐情報を収集する。bt_collect_log スクリプトの説明を表 5-1 に示す。

表 5-1　bt_collect_log スクリプト説明

<table>
<tr><td>コマンド</td><td colspan="3">bt_collect_log</td></tr>
<tr><td>書式</td><td colspan="3">bt_collect_log [共通オプション] [ユーザ空間用オプション | 全空間用オプション]
ユーザ空間用オプション：
-p pid [...pid] -d log_dir
-d log_dir -c cmd [arg...]
全空間用オプション：
[--start symbol] [--stop symbol] -d log_dir
[--START addr,size] [--STOP addr,size] -d log_dir
--fr -d log_dir
--exit-fr [-d logdir]</td></tr>
<tr><td>オプション</td><td>共通オプション</td><td>-S bts_size</td><td>ドライバが内部で使用する BTS バッファのサイズ（単位：バイト数、デフォルト：786KByte、min：デフォルトと同じ、max：※）</td></tr>
<tr><td></td><td></td><td>-M number</td><td>割り込み発生時の BTS バッファの空きレコード数（１レコードは 12Byte）。割り込み発生後でも、この数分の分岐だけは保存できる。BTS バッファがオーバフローしてしまう場合に、調整用として使用する（デフォルト：8192、min：1024、max：－）</td></tr>
<tr><td></td><td></td><td>-s size</td><td>relayfs バッファサイズ（単位：バイト数、デフォルト：2MByte、min：1 以上、max：※）</td></tr>
<tr><td></td><td></td><td>-n number</td><td>relayfs バッファ面数（デフォルト：16、min：1 以上、max：※）</td></tr>
<tr><td></td><td></td><td>-N number</td><td>ここで指定した relayfs バッファ数分の未読み出しログがカーネル内に溜まると、対象プロセスをスリープさせる。relayfs バッファがオーバフローしてしまう場合に、調整用として使用する。ユーザ空間トレース（-p, -c オプション指定）時のみ有効（デフ</td></tr>
</table>

			ォルト：8、min：1、max：-n 指定数）
		-m mnt_dir	relayfs をマウントする際に使用するマウントポイント（デフォルト：/mnt/relay）
	-d log_dir		ログ出力先ディレクトリ名。存在しない場合は生成する。ログが出力された後のディレクトリを指定すると、エラーになる。
	--start symbol		指定シンボル通過後から、使用可能なバッファ分だけのトレース情報を取得する。このオプション指定時は、トレース情報は上書きされない
	--stop symbol		指定シンボルに到達する直前のトレース情報を、バッファ分だけ取得する
	--START addr,size		--start symbol と同じ。--start symbol ではシンボルが見つからない場合に、アドレスとサイズを直接指定することが可能
	--STOP addr,size		--stop symbol と同じ。--stop symbol ではシンボルが見つからない場合に、アドレスとサイズを直接指定することが可能
	-p pid [...pid]		ユーザ空間トレース指定。pid はスペースで区切って複数指定可能
	-c cmd [arg...]		ユーザ空間トレース指定。実行するアプリケーションには、事前にトレース開始関数を組み込んでおく必要がある。また、トレース終了関数も合わせて組み込むことで、アプリケーションの任意の部分をトレースすることも可能（ただし、同時に複数のトレース開始/終了関数が動くと誤動作するため、fork プロセス等には使用しないこと）
	--fr		フライトレコーダモード指定。スクリプト終了後も die 関数実行までトレースし続ける
	--exit-fr		フライトレコーダモード終了指定。そこまでのトレース情報が必要な場合のみ-d でログ出力ディレクトリを指定する

※IA-32 の場合、Linux の制限により、（relayfs バッファサイズ×面数＋BTS バッファサイズ）×プロセッサ数の合計が、約 100MByte までは確保可能

各オプションによりトレース対象空間等が異なるため、適切なオプションを使い分ける必要がある。表 5-2 に各オプションによるトレースモードの詳細を示す。

表 5-2　トレースモードの詳細

#	オプション	トレース空間	対象プロセス	トレース容量制限	relayfs バッファオーバフローの可能性
1	--start,	全空間	start 通過プロセス	なし	あり

	--stop				
2	--start	全空間	start 通過プロセス	最大 relayfs バッファ分まで	なし
3	--stop	全空間	全プロセス	最大 relayfs バッファ分まで	なし
4	--fr	全空間	全プロセス	最大 relayfs バッファ分まで	なし
5	-p または-c	ユーザ空間	指定プロセス	なし	なし※

※適切に-N オプション値を設定している場合に限る。

本スクリプトの実行により、プロセッサ毎のトレースログ、モジュール/マップ情報等が指定したディレクトリに出力される。表 5-3 に本スクリプトの出力ファイル一覧を示す。

表 5-3　bt_collect_log 出力ファイル一覧

#	出力ファイル名	ファイル内容
1	$pid.maps	/proc/$pid/maps のコピー。$pid は pid 値を意味する（例：1234.maps）
2	modules	/proc/modules のコピー
3	kallsyms	/proc/kallsyms のコピー
4	dropped	欠落した relayfs サブバッファ数
5	cpuN	トレースログファイル。N は CPU 番号（0～）を意味する

$pid.maps、modules および kallsyms ファイルは、後でトレースログの解析に用いるため、スクリプトの起動時に保存する。このため、スクリプト起動後にロードしたモジュールやライブラリは解析できない。

5.2 ログ分割プログラム

ログ分割プログラム（bt_split）は、プロセッサ毎に出力されるトレースログ（ファイル名は cpuX）を、プロセス毎に分割する。bt_split コマンドの説明を表 5-4 に示す。

表 5-4　bt_split コマンド説明

コマンド	bt_split	
書式	bt_split -d dir	
引き数	-d dir	ログ入出力先ディレクトリ名。存在しない場合はエラーとなる。
オプション	なし	―

本コマンドの実行により、トレースログと同じディレクトリに、プロセス毎のプロセスログ（ファイル名は pid 値）が出力される。

5.3 実行経路表示プログラム

実行経路表示プログラム（bt_expath）は、トレース情報を解析し、実行経路として表示する。コマンド実行の際には、解析/表示対象とするアドレス範囲を指定できる。トレース対象にデバッグ情報が含まれている場合は、ソースファイル名や行番号が表示され、含まれていない場合は、アセンブラニーモニックが表示される。

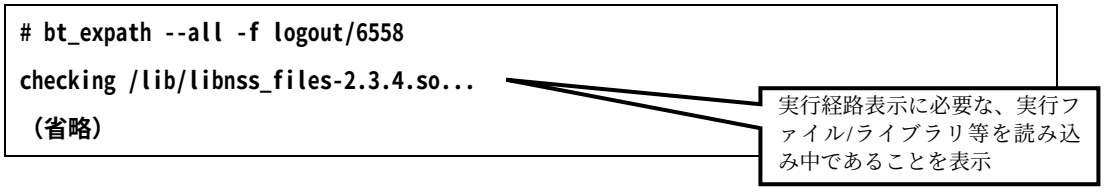
bt_execpath は、簡易版のサマリ表示機能を持っており、関数コール／ジャンプ／割り込み実行だけをネストさせて表示させることができる。本機能は簡易版であり、ネストがずれることがある。

また、通常表示／サマリ表示共に、繰り返しチェック機能があり、処理が繰り返されている部分は省略して表示される。bt_expath コマンドの説明を表 5-5 に示す。

表 5-5　bt_expath コマンド説明

コマンド	bt_expath	
書式	bt_expath [-s] [-a top:end [...]] [-d top:end [...]] [--usr | --ker | --all] -f logfile	
引き数	-f logfile	解析対象のログファイル名。プロセスモードの場合はプロセスログを、カーネルモードの場合はトレースログを指定する。
オプション	-s	サマリ表示指示
	-a top:end	解析/表示対象とするアドレス範囲を指定する。複数範囲の指定が可能。何も指定しないと何も表示されない
	-d top:end	-a オプションで追加したアドレス範囲から、任意のアドレス範囲を除外したい場合に使用する。複数範囲の指定が可能
	--usr	ユーザ空間アドレス範囲指定。-a 0:0xbfffffff のエイリアス
	--ker	カーネル空間アドレス範囲指定。-a 0xc0000000:0xffffffff のエイリアス
	--all	全空間アドレス範囲指定。-a 0:0xffffffff のエイリアス

本コマンドの実行により、実行経路が標準出力に出力される。実行経路表示プログラムの出力書式を図 5-1 に、サマリ出力時の出力書式を図 5-2 に示す。



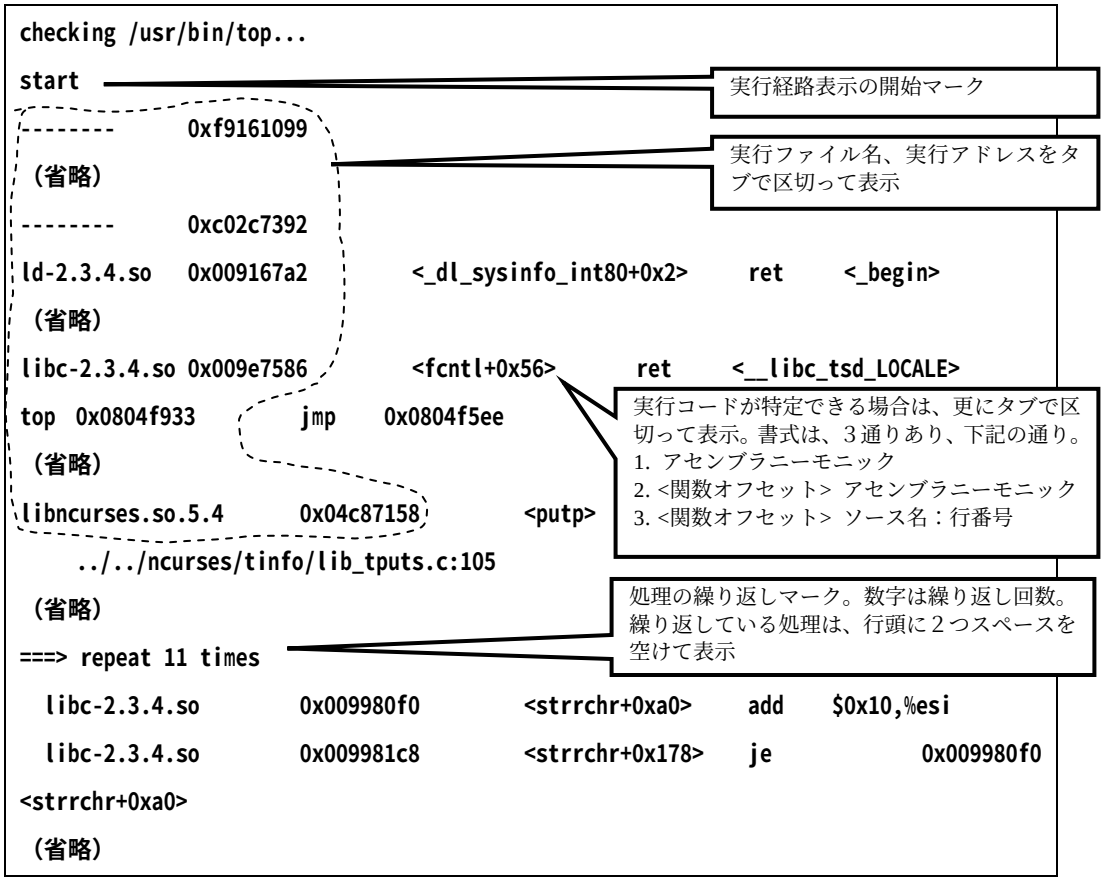


図 5-1　実行経路表示プログラム出力書式

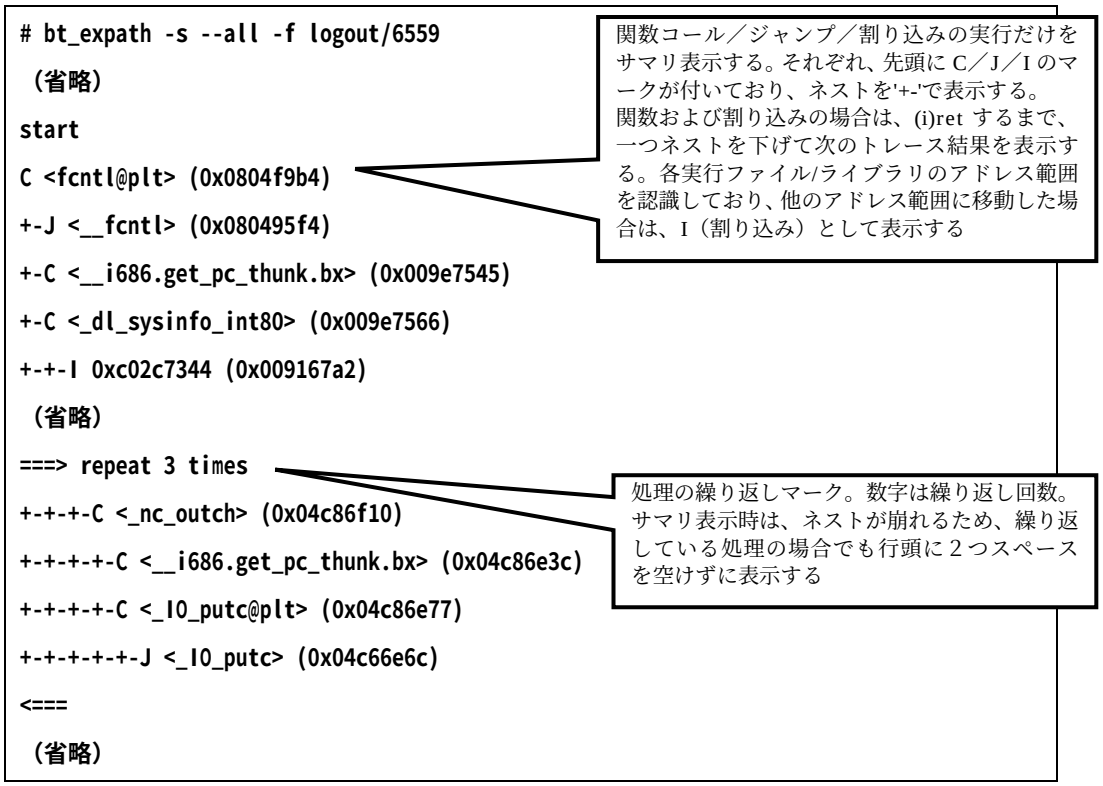


図 5-2　実行経路表示プログラムのサマリ出力書式

5.4 分岐カバレッジ表示プログラム

分岐カバレッジ表示プログラム（bt_coverage）は、実行ファイルおよびトレース情報を解析し、分岐カバレッジ情報として表示する。分岐カバレッジ情報には、全関数中どれだけの関数が実行されているかを示す関数呼び出しカバレッジと、分岐実行カバレッジ、および各ステートメントが実行されているかを示すステートカバレッジの３種類がある。

分岐カバレッジの算出のため、実行ファイルの解析を行うが、これに関して下記の制限がある。

(1) プログラム実行時に分岐先や関数名が決まるような場合（例えば、関数ポインタに任意の関数をセットして実行する等）については、完全な解析は不可能であり、これらについてはトレース情報が記録されているもののみ、出力される。
(2) 各セクションの先頭および各関数シンボルから順次解析を行い、そこで検出された関数や飛び先を更に解析することで、実行ファイル解析を行う。このため、gcc の-s オプション(シンボルテーブルとリロケーション情報の削除)付きでコンパイルされたり、strip されたりした実行ファイルについては、関数エントリが見つからない場合があるため、その部分のカバレッジが表示されない。
(3) アセンブラで記述された関数等、シンボル情報に関数を示すフラグが付いていない場合は、関数か変数か区別が付かないため、関数カバレッジに表示されない。
(4) インライン関数の場合、関数の先頭が検出できないため、インライン関数中の先頭から最初のステートの切れ目（例えば関数コールや分岐等）までのカバレッジ html の色付けは不定となる。

コマンド実行の際には、解析/表示対象とするアドレス範囲を指定できる。トレース対象にデバッグ情報が含まれている場合は、ソースファイル名や行番号が表示され、含まれていない場合は、アセンブラニーモニックが表示される。

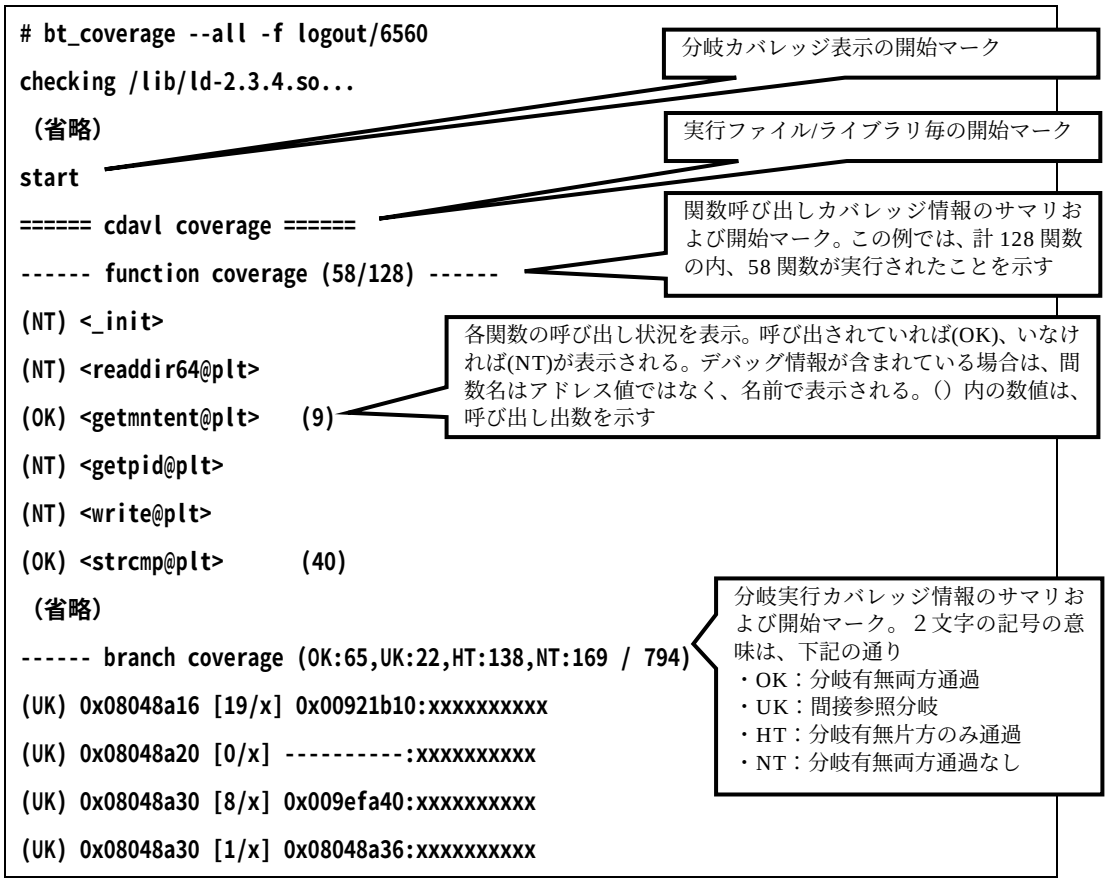
デバッグ情報が含まれておりソースファイルが存在する場合は、html 出力を指定することによって、ソースファイルを色分けし、どの行が実行されたかどうかを確認することができる。bt_coverage コマンドの説明を表 5-6 に示す。

表 5-6　bt_coverage コマンド説明

コマンド	bt_coverage
書式	bt_coverage [-a top:end [...]] [-d top:end [...]] [--usr | --ker | --all] -f logfile [[-s src_dir] -o html_out_dir]

引き数	-f logfile	解析対象のログファイル名。プロセスモードの場合はプロセスログを、カーネルモードの場合はトレースログを指定する。
オプション	-a top:end	解析/表示対象とするアドレス範囲を指定する。複数範囲の指定が可能。何も指定しないと何も表示されない
	-d top:end	-a オプションで追加したアドレス範囲から、任意のアドレス範囲を除外したい場合に使用する。複数範囲の指定が可能
	--usr	ユーザ空間アドレス範囲指定。-a 0:0xbfffffff のエイリアス
	--ker	カーネル空間アドレス範囲指定。-a 0xc0000000:0xffffffff のエイリアス
	--all	全空間アドレス範囲指定。-a 0:0xffffffff のエイリアス
	-s src_dir	次の-o オプション指定時のみ有効。デバッグ情報中にソースファイルの絶対パスが含まれていない場合は、正しく html 出力されない。このような場合に、ソースファイルのパスを指定する
	-o html_out_dir	html 出力指定。指定したディレクトリの中に、top.html およびその他の html ツリーが生成される

本コマンドの実行により、分岐カバレッジ情報が標準出力または、html ファイルに出力される。分岐カバレッジ表示プログラムの標準出力時の書式は図 5-3 の通り。



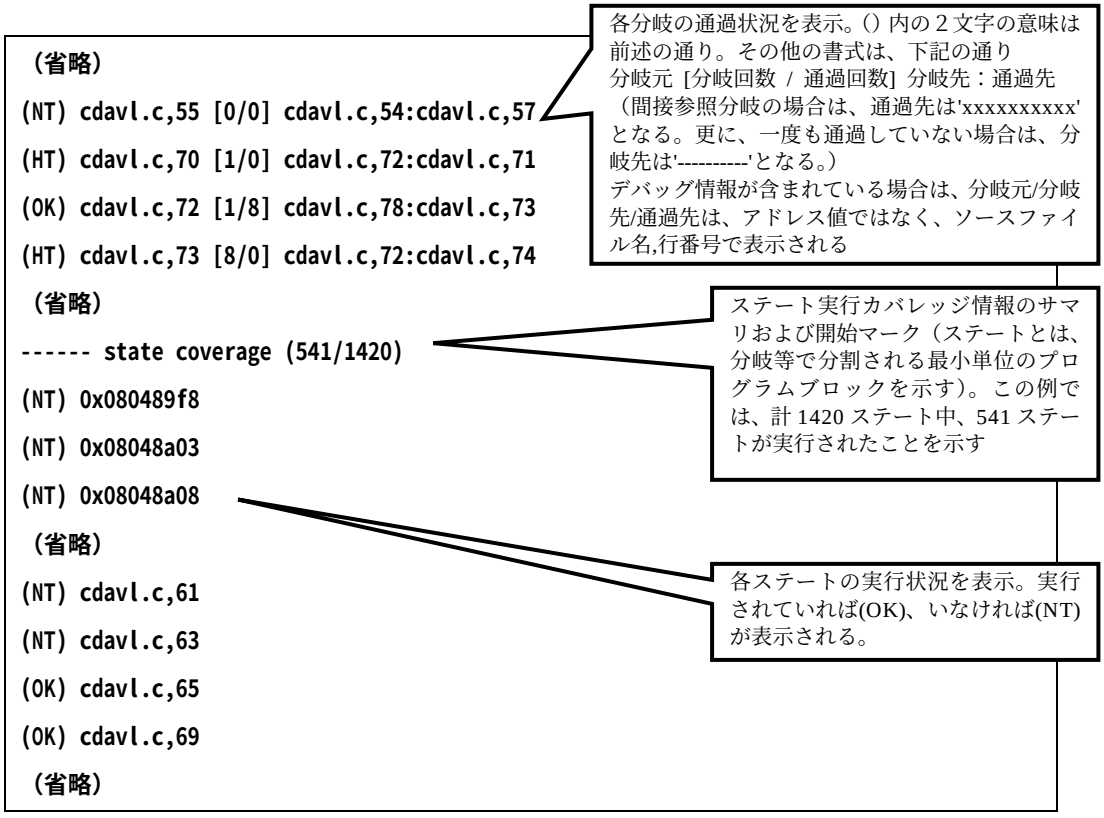


図 5-3　分岐カバレッジ表示プログラム出力書式

また、分岐カバレッジ html ファイル（$(ログ出力ディレクトリ)/top.html）の使用方法は図 5-4～図 5-6 の通り。

図 5-4　分岐カバレッジ html 表示初期画面

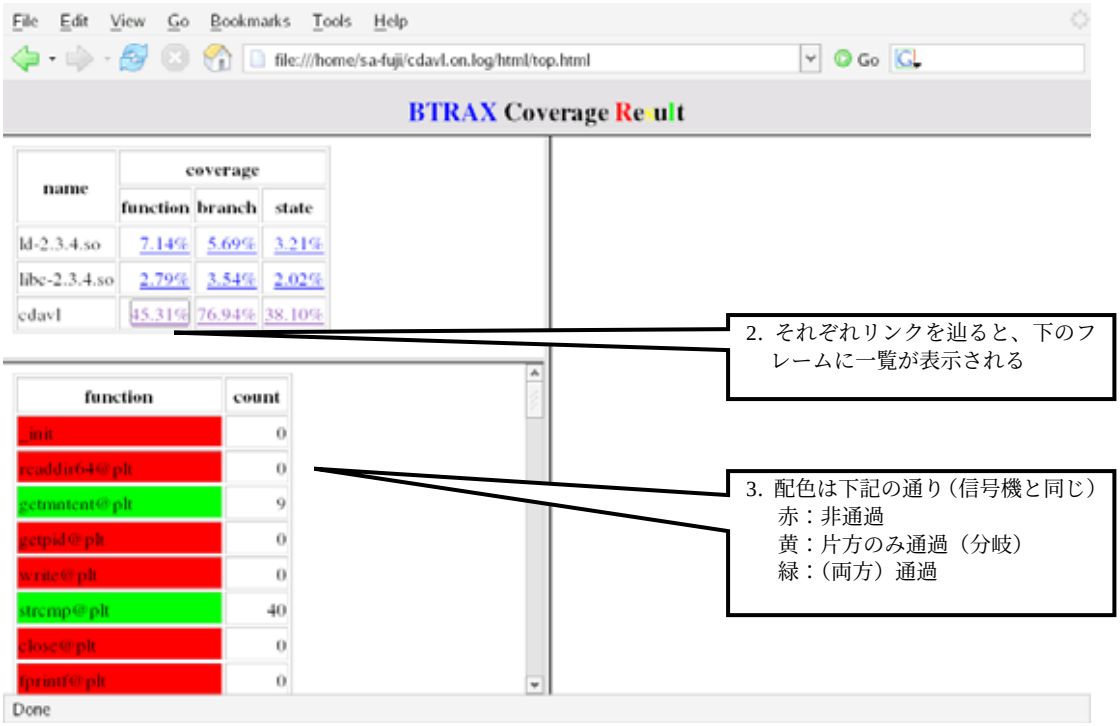


図 5-5　関数分岐カバレッジ html 表示画面

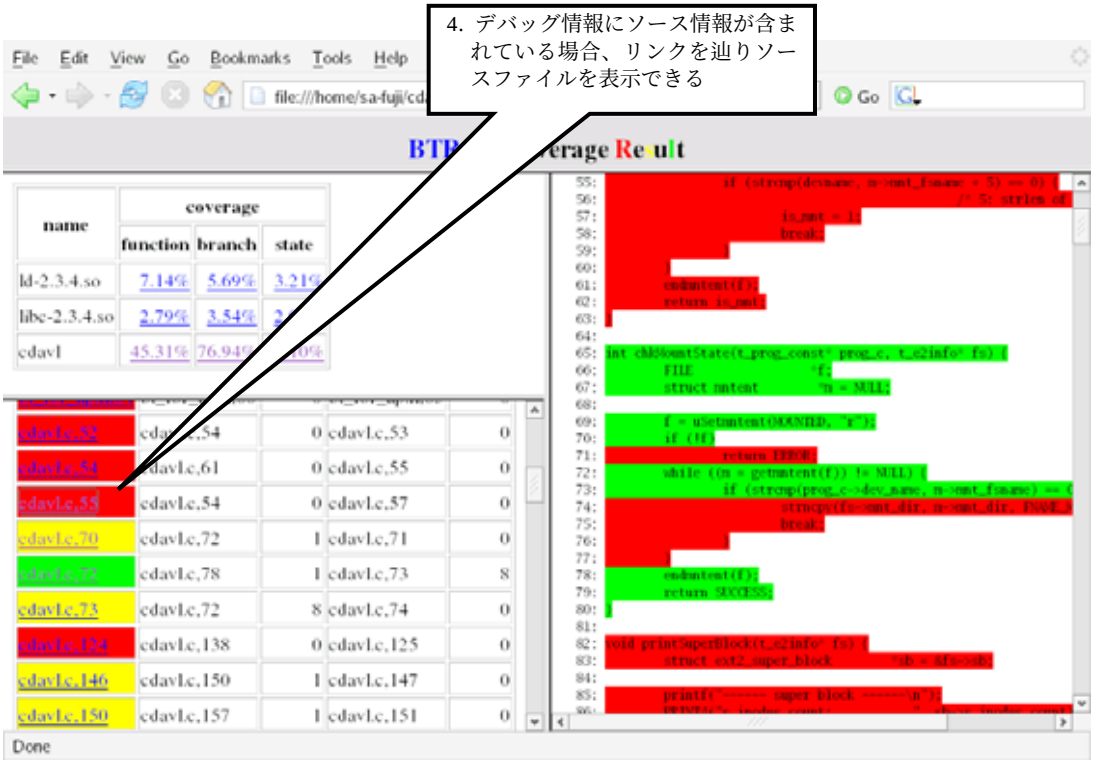


図 5-6　分岐実行カバレッジの一覧とソースファイル html の表示画面