

システムマクロトレース ポーティングガイド Linux SoftwareModel 編

株式会社DTSインサイト

ご注意

- システムマクロトレース API ライブラリは、株式会社DTSインサイト(以下DTSインサ イト)の著作物であり、システムマクロトレース API ライブラリにかかる著作権、その他の 権利はすべてDTSインサイトに帰属します。
- 2. 本書に記載されている会社名・製品名は、一般に各社の登録商標、または商標です。
- 3. システムマクロトレース API ライブラリおよび本書の一部または全部をDTSインサイトの 書面による許可なく複写・複製することは、その形態を問わず禁じます。
- 4. システムマクロトレース API ライブラリの仕様および本書に記載されている事柄は、予告な く変更することがあります。
- 5. システムマクロトレース API ライブラリおよび本書を運用した結果の影響については、いっ さい責任を負いかねますのでご了承ください。

システムマクロトレースポーティングガイド

本マニュアルに記載されている企業名、システム名、製品名は、各社の商標または登録商標です。 なお、本文中では、TM、Rマークは明記していません。 © 2013 DTS INSIGHT CORPORATION

発行:株式会社DTSインサイト

改訂履歴

日 / 反 / 正	-		
	第1版	2013年9月	新規発行
	第2版	2013年11月	誤記修正
	第3版	2014年7月	実行モード追加 誤記修正
	第4版	2015年2月	指定する ccfg について追記 誤記修正
	第5版	2015年7月	API ライブラリバージョン 3.00 対応
	第6版	2016年6月	スタック解析機能について追記 誤記修正

目次

1.	は	じめに	1
	1.1.	システムマクロトレースとは	1
	1.2.	API ライブラリとは	1
	1.3.	ポーティングガイドとは	1
	1.4.	API ライブラリ出力仕様	2
	1.4	.1. 出力情報について	2
	1.5.	動作確認ターゲットボード	2
	1.6.	各ツールのマニュアルについて	2
	1.7.	コマンド入力例の表記について	3
	1.8.	表記規則	3
	1.9.	ディレクトリ表記について	4
	1.10.	バージョン表記について	4
	1.11.	お問い合わせについて	4
2.	ر م بر	別用の前に	5
_	_		
	2.1.	対応パージョンについて	5
	2.2.	GPL(General Public License)の適応範囲について	5
	2.3.	準備するもの	6 _
	2.4.	開発ホスト Linux サーハーとのファイル共有について	/ –
	2.5.	API フイ ノフリの機構	/
3.	作	業の流れについて	8
4.	AP	l ライブラリのインストール	9
	11	淮借	0
	4.1.	半哺	99 م
	4.Z.	インストール	9 10
	ч. <u>э</u> . Л Л	ストノインフリティレクトリ構成	10 11
	4. . .	フライ フェア ビアルアイ レフィ ア構成	12
	4.6	ユー デーガバデマイバデーパコートに因じて	12 12
	4.0.	1 C コンパイラ標準ライブラリのリンク	12
	4.6	$2 \gamma \neq \gamma = \gamma =$	12
	4.6	3. API ライブラリの型宣言について(SMT API ライブラリバージョン 2.12 以降)	
	4.6	.4. 64bitOS 環境について	13
5		リライゴラリのっピー	11
5.			
	5.1.	API ライブラリコピースクリプトの編集	14
	5.2.	API ライブラリコピースクリプトの実行	14
6.	カ-	ーネルランド API ライブラリとデバイスドライバの適用	15
	6.1.	ファイルー覧	16
	6.2	API ライブラリ適用後のディレクトリ構成例	16
	6.3.	ソースファイルの組み込み	17
	6.3	.1. API ライブラリファイルの組み込み確認	17
	6.3	.2. Linux カーネルの Makefile を編集	17
	6.4.	カーネルランドからのスタック解析機能を有効にする(API ライブラリバージョン 3.10 以降)	18

6.4.1SMT_get_SP_address 関数実装	18
6.4.2. 注意点	18
6.5. Linux カーネルのビルド	18
6.6. システムマクロトレースドライバの確認	19
6.7. バッファーサイズの指定方法	19
7. プロセス遷移情報の取得	20
7.1. コンテキストスイッチ関数の挿入	21
7.2. SMT 用ヘッダファイルのインクルード	21
7.3. コンテキストスイッチ情報出力コードの挿入	22
7.4. Linux カーネルのビルド	22
8. ユーザーランド API ライブラリのビルド	23
8.1. ファイルー覧	24
8.2. ユーザーランドからのスタック解析機能を有効にする(API ライブラリバージョン 3.10 以降	夅)25
8.2.1SMT_get_SP_address 関数実装	25
8.2.2. 注意点	25
8.3. ビルド設定	26
8.4. ビルドの実行	27
9. ユーザーランドからの出力確認用アプリのビルド	28
9.1. Makefile の編集	29
9.2. ビルドの実行	29
9.3. ファイルシステムへの組み込み	29
10. SMT ログファイル出力デーモンの組込み	30
10.1. ファイルー覧	31
10.2. SMT ログファイル出力デーモンの設定	31
10.3. Makefile の編集	31
10.4. SMT ファイル出力デーモンのビルド	31
10.5. SMT ログファイル出力デーモンの組込み	31
10.6. 注意事項	31
11. システムマクロトレース取得方法	32
11.1. SMT ログファイル出力デーモンの起動	33
11.1.1. 起動オプション	33
11.1.2. 出力動作モードについて	34
11.1.3. SMT ログファイル出力デーモン起動コマンド例	35
11.2. トレースの開始と終了方法について	36
11.2.1. 有効化方法	
11.2.2. 無効化方法	
11.2.5. 注意争惧	
II.3. ケノノルナーダ 収行	37 20
11.4. IIIdUUTRACE-VIEVVER について	
11.4.2 Index データ生成設定 (onetime チード ring チード)	
11.4.3. Index データ生成設定 (continuous モード)	

11.5.	macroTRACE-VIEWER でのトレース取得例	40
12. シ	·ステムマクロトレース出力確認方法	41
12.1.	プロセス遷移出力確認	42
12.1	.1. プロセス遷移情報の出力確認手順	42
12.2.	ユーザーランド API 出力確認	43
12.2	.1. ユーザーランドからのシステムマクロトレース出力確認方法	43
13. 力	ーネルランドからの関数トレース取得方法	44
13.1.	ネットワークドライブの設定	44
13.2.	Cpe ワークスペース新規作成	45
13.2	.1. Cpe 起動	45
13.2	.2. ワークスペース新規作成	45
13.2	.3. 言語の選択	46
13.2	.4. 新規ワークスペース	47
13.2	<i>.</i> 5. プロジェクト作成	48
13.2	.6. パス変換設定	49
13.3.	ライブラリサーチパス設定の実施	51
13.4.	MakeFile の編集	51
13.5.	ビルドの実行	
13.6.	macroTRACE-VIEWER の設定	53
13.7.	システムマクロトレース取得例	54
14. д	-一ザーランドからの関数トレース取得方法	55
14.1.	サンプルプログラムのビルドの実施	
14.1	.1. ソースコードの確認	55
14.1	.2. Makefile を編集	
14.1	.3. ビルド実施	
14.2.	ライブラリサーチパス設定の実施	
14.3.	Cpe/make cp 挿入ツール設定	
14.3	.1. プロジェクトの追加作成	
14.3	.2. チェックポイント挿入ソース選択の完了	60
14.3	.3. Makefile.smt を編集	61
14.3	.4. make の実行	62
14.4.	出力確認方法	63
14.4	.1. オブジェクトの保存	63
14.4	.2. Linux システムの起動	63
14.4	.3. helloworld の実行	63
14.5.	macroTRACE-VIEWER の設定	64
14.6.	システムマクロトレース取得例	65
15. 效)果的なご利用方法	66
15.1.	システムコール情報の取得	66
15.2.	割込み情報の取得	67
15.3.	printk 情報の取得	68
16 付		60
10. 19	J 邓小	

16.1.	実施項目の確認事項	.69
16.1.1.	カーネルランドからの関数トレース出力関連	69
16.1.2.	ユーザーランドからの関数トレース出力関連	69

1. はじめに

1.1. システムマクロトレースとは

システムマクロトレースは、一般的な Print 文によるデバッグ手法と関数/タスクの実行履歴機能を融合したデバッグ手法です。システムマクロトレースを使用することで、以下のことが解析できます。

- 関数/タスクの実行履歴を可視化
- 関数の処理時間が見える
- システム全体の動作からボトルネックを発見できる
- 関数履歴と print 文を同時に記録/表示可能

1.2. APIライブラリとは

API ライブラリとは、システムマクロトレース対象のアプリケーションからシステムマクロトレース出 力専用デバイスドライバを利用し、関数トレース情報や、デバックプリント文等の出力を行うためのラ イブラリです。

1.3. ポーティングガイドとは

ポーティングガイドは、Linux システムにシステムマクロトレース API ライブラリを組み込む方法について、実際の手順を追って解説するものです。

[ポーティング内容]

- API ライブラリのビルド
- デバイスドライバの組み込み
- コンテキスト切り替え情報の取得
- SMT ログファイル出力デーモンについて
- Cpe/make_cp での固有設定
- macroTRACE-VIEWER での固有設定

1.4. APIライブラリ出力仕様

1.4.1. 出力情報について

システムマクロトレースの出力を行うためには、「4.API ライブラリのインストール」から「10 SMT ロ グファイル出力デーモンの組込み」までの作業が必須になります。

「4.API ライブラリのインストール」から「10 SMT ログファイル出力デーモンの組込み」を実施すると システムマクロトレースで下記情報出力の確認が出来ます。

- ・ プロセス遷移情報
- ・ サンプルプログラムからの SMT デバッグプリント文出力

1.5. 動作確認ターゲットボード

アーキテクチャ	コア	ボード名
ARM	Cortex-A8	BeagleBoard-XM
ARM	Cortex-A9	Pandaboard

1.6. 各ツールのマニュアルについて

本ドキュメント内で使用する各ツールの詳細についてはそれぞれ下記マニュアルをご覧ください。 ・macroTRACE-VIEWER は『macroTRACE-VIEWER ユーザーズマニュアル』

・Cpe は『システムマクロトレースチェックポイントエディタ(Cpe)ユーザーズマニュアル』

・make_cpは『システムマクロトレースチェックポイント挿入コマンドラインツール(make_cp)ユーザーズマニュアル』

・API ライブラリ関数の詳細は『システムマクロトレース API ライブラリ リファレンスマニュアル』

1.7. コマンド入力例の表記について

このマニュアルには Linux コマンドの入力例が記載されています。その際、コマンドの実行環境を想定 したプロンプト表記をしています。

プロンプト	コマンドの実行環境		
[HOST]# Linux ホスト PC 上の特権ユーザーで実行			
[HOST]\$	Linux ホスト PC 上の一般ユーザーで実行		
[TARGET]#	ターゲット Linux 上の特権ユーザーで実行		
[TARGET]\$	ターゲット Linux 上の一般ユーザーで実行		

ユーザーのホームディレクトリは "~"と表現します。 アンダースコア "_"とハイフン "-"の違いにご注意ください。 コマンドが2行にわたる場合、1行目の最後に"¥"を付記しています。

1.8. 表記規則

本書では以下の表記規則を使用しています

monospace

コマンド、ファイル名、ディレクトリ、関数名、ソースコードを意味しています。

monospace italic

コマンド、ファイル名、ディレクトリ、関数名、ソースコードで、説明の都合上、指定してお り、実際には、お客様の開発環境に依存するものを意味しています。

gothic

GUI上で指定するメニュー名や、項目名を意味しています。

gothic italic

GUI 上で指定するメニュー名や、項目名で、説明の都合上、指定しており、 実際には、お客様の開発環境に依存するものを意味しています。

1.9. ディレクトリ表記について

API ライブラリインストールディレクトリを *\$(API_LIBRARY)*と表記します。
 カーネルツリーのトップディレクトリを*\$(KERNEL)*と表記します。
 開発ホスト Linux のログインユーザー名を *<username>* と表記します。
 出力対象の IF を*\$(SMTIF)*と表記します。

1.10. バージョン表記について

API ライブラリは APILib-Linux -REV.tar.gz 形式で提供します。 REV は、API ライブラリバージョンを示し、バージョン 1.00 の場合は APILib-Linux-100.tar.gz となります。

1.11.お問い合わせについて

マニュアルや製品に関する質問に関しましては弊社 WEB サイトに FAQ がありますので、 はじめに弊社 WEB サイトの FAQ をご覧ください。

http://www.dts-insight.co.jp/support/support_advice/luna/faq/

弊社 WEB サイトの FAQ を探しても解決しない場合、下記お問い合わせフォームよりお問い合わせください。

https://www.dts-insight.co.jp/support/support_advice/

2. ご利用の前に

API ライブラリのバージョンは 2.10 以上を使用してください。

2.1. 対応バージョンについて

システムマクロトレース API ライブラリは、Linux の以下のバージョンに対応しています。

- Linux 2.6 系カーネル
- Linux3.0~3.10.x カーネル

※32bit 環境のみ対応しています。

2.2. GPL(General Public License)の適応範囲について

API ライブラリの kernel ディレクトリは GPL v2 ライセンスです。

2.3. 準備するもの

システムマクロトレース API ライブラリのご利用前に、以下のものを準備してください。

• SMT Software Kit(TLA002)

システムマクロトレース取得に必要なソフトウェアパッケージです。

- ・システムマクロトレースで取得したデータを閲覧するためのツール(macroTRACE-VIEWER)
- ・関数トレースを取得するために必要なツール(CheckPointTools)
- Windows ホスト PC システムマクロトレースの出力結果を見るために必要です。 macroTRACE-VIEWER、Chceck Point Tools をインストール済みの PC を準備してください。
- Linux ホスト PC(以降、ホスト PCと表記)
 開発用 Linux ディストリビューションをインストール済みの PC です。 ユーザーアカウントがあり、ログインできることをご確認ください。
- GNU コンパイラツールチェーン デバッグ対象の Linux アプリケーション、Linux カーネルのソースコードをコンパイルするためのツ ール群です。これらがホスト PC にインストールされており、ソースファイルをコンパイルできるこ とをご確認ください。
- Linux カーネルソース デバッグ対象となる Linux カーネルのソースコードです。ソースファイルを追加、編集できることを ご確認ください。
- システムマクロトレース情報ログファイルを転送する手段 ユーザーシステム上に貯めたシステムマクロトレースログファイルをWindowsマシンへ転送する手段 が必要です。

2.4. 開発ホストLinuxサーバーとのファイル共有について

チェックポイントエディタ(Cpe)をご利用いただく場合は、Linux ホスト PC と Windows ホスト間のファ イル共有が必要になりますので、samba によるファイル共有が可能な設定をお願いします。

2.5. APIライブラリの機構

アプリケーション



3. 作業の流れについて

システムマクロトレース出力を行うための作業フローは下記のようになっています。

- カーネルランド API ライブラリを組み込んだ Linux カーネルを起動する
- システムマクロトレースデバイスドライバをカーネルに登録する
- ユーザーランド API ライブラリを組み込んだプログラムを実行する

開発環境によりますが、通常はホスト PC 上の作業、ターゲット Linux システム上の作業があります。



ホスト PC 作業

4. APIライブラリのインストール

ホストPC上の所定ディレクトリにAPI ライブラリをインストールする手順を記載しています。



4.1. 準備

ホームディレクトリは、/home/<username>ディレクトリと仮定し、/home/<username>に API ライブラリ をダウンロードしたと仮定します。

[HOST]\$ ls /home/<username> APILib-Linux-REV.tar.gz

REV は API ライブラリのバージョンです。

4.2. インストール

アーカイブファイルを tar コマンドで展開してください。

```
[HOST]$ cd ~/
[HOST]$ pwd
/home/<username>
[HOST]$ tar -zxvf APILib-Linux-REV.tar.gz
[HOST]$ cd APILib-Linux-REV
[HOST]$
```

4.3. APIライブラリディレクトリ構成

API ライブラリをインストール後のファイル構成を説明します。



4.4. ソフトウェアモデルディレクトリ構成

ソフトウェアモデル用 API ライブラリのディレクトリ構成について説明します。



4.5. ユーザーカスタマイズソースコードに関して

次に示すソースコードは、お客様の環境及びご利用方法に応じて、修正して頂く事が可能です。 他のソースコード及び Makefile の修正は行わないでください。 編集した場合、動作しない可能性があります。

標準ディレクトリ~/APILib-Linux-REV/SoftwareModel/階層下にあります。

ファイル名	修正内容
smtuser/SMTUser.c	デバッグプリント文レベル設定を行います。
kernel/smt/smtBufferConfig.h	バッファーサイズの設定を行います。
smt_userp/Makefile	SMT デーモンのビルド設定をします。
smt_userp/smtconfig.h	SMT ファイル出力デーモンの設定用ヘッダファイルです。
library/Makefile	ユーザーランド API ライブラリビルドの設定をします。
	設定方法に関しては、「8.3.ビルド設定」をご覧ください。
library/SMTUserDef.h	デバッグプリント文レベル設定初期値の設定を行います。

4.6. APIライブラリの注意事項

4.6.1. Cコンパイラ標準ライブラリのリンク

API ライブラリ関数で提供する_SMT_Printf 関数は、標準ライブラリ関数の vsprintf を使用するように設計されています。

このため、_SMT_Printf 関数を使用する場合は、標準ライブラリのリンクが必要となります。

もしくは、標準ライブラリを使用せずに vsprintf 関数が使用できるように、vsprintf 関数を再実 装する必要があります。

詳しくはご使用になる開発環境のユーザーズマニュアルをご覧ください。

4.6.2. タイムスタンプについて

SMT ログファイルに付与するタイムスタンプはカーネル内で管理している情報を使用しています。タ イムスタンプ精度によっては同一タイムスタンプが付与される可能性があります。 また、SMT 取得中にタイムスタンプを変更しないでください。

4.6.3. APIライブラリの型宣言について(SMT APIライブラリバージョン 2.12 以降)

SMT で扱うデータは 32bit 符号なし整数型を使用します。

SMT API ライブラリバージョン 2.12 以降のカーネルランド API ライブラリ、ユーザーランド API ライ ブラリ、SMT ログファイル出力デーモンはデフォルト unsigned long が 32bit の符号なし整数型と仮定し て_SMT_UNSIGNED_32BIT_INTEGER に typedef 宣言しています。

ご利用の環境で異なる場合にはそれぞれ変更を行ってください。

編集箇所の変更例は unsigned int が 32bit の符号なし整数型だった時のものです。

4.6.3.1. カーネルランド API ライブラリの編集箇所

\$(API_LIBRARY)/SoftwareModel/kernel/include/SMTAPI.h の 22 行目付近

typedef unsigned int __SMT_UNSIGNED_32BIT_INTEGER;

4.6.3.2. ユーザーランド API ライブラリの編集箇所

\$(API_LIBRARY)/SoftwareModel/library/SMTAPI.hの17行目付近

typedef	unsigned	int	_SMT_UNSIGNED_32BIT_INTEGER;

4.6.3.3. SMT ログファイル出力デーモンの編集箇所

\$ (API_LIBRARY)/SoftwareModel /smt_userp/smtconfg.h の1行目付近

4.6.3.4. SMT のユーザーランド用ヘッダファイル編集箇所

\$ (API_LIBRARY)/SoftwareModel /smtuser /SMTAPI.h の19行目付近

typedef unsigned int __SMT_UNSIGNED_32BIT_INTEGER;

4.6.4. 64bitOS環境について

システムマクロトレースの 64bit データ幅には対応していないため、関数トレースで 64bit 変数の引数/ 戻り値は取得できません。

5. APIライブラリのコピー

SMT API ライブラリは Linux カーネルソースツリーに展開する必要があります。 SMT API ライブラリコピースクリプトを使用してカーネルランド API ライブラリを Linux カーネルツリ ーへ展開します



5.1. APIライブラリコピースクリプトの編集

ユーザー環境に合わせて下記環境変数を編集してください。

コピースクリプトは\$(API_LIBRARY)/ install_Linux_ApiLibrary.sh です。

変数名	説明
SMT_INTERFACE	使用する SMT I/F 名を入力してください
SMT_API_LIBRARY_DIR	API ライブラリの展開先を設定してください
KERNEL_DIR	Linux カーネルソースツリーのトップを設定してください。

5.2. APIライブラリコピースクリプトの実行

実行権限の付与をして API ライブラリコピースクリプトを実行します。 各 API ライブラリコピー前に実施するかどうか確認メッセージが出力されますので、 コピーする場合は y をコピーしない場合は n を押してエンターキーを押してください。

[HOST]\$ cd \$(API_LIBRARY) [HOST]\$ chmod a+x ./install_Linux_ApiLibrary.sh [HOST]\$./install_Linux_ApiLibrary.sh

Config Info SMT_INTERFACE : SoftwareModel SMT_API_LIBRARY_DIR : \$(API_LIBRARY) KERNEL_DIR : \$(KERNEL)

Do you want to copy Kernel Land API Library?[y/n] y

6. カーネルランドAPIライブラリとデバイスドライバの適用

Linux カーネルからデバッグ情報を出力するには、カーネルのソースファイルに以下の3つを追加し、 Linux カーネルビルドする必要があります。

- API ライブラリのカーネル依存ソースファイル
- API ライブラリのヘッダファイル
- デバイスドライバの組み込み

この章ではカーネルランド API ライブラリの適用方法について記載します。

API ライブラリは「5 API ライブラリのコピー」でコピーが完了していることを前提に説明を行います。





6.1. ファイル一覧

カーネルランド API ライブラリのビルドに必要なファイルについて示します。 ※\$(API_LIBRARY)/SoftwareModel 以下にあります。

ディレクトリ	ファイル名	説明
kernel/smt	Makefile	API ライブラリの Makefile です。
kernel/smt	smt.c	API ライブラリのカーネル依存部ソースファイルです。
kernel/smt	smt_sa_data.c	SMT デバイスドライバソースファイルです。
kernel/smt	smtBufferConfig.h	バッファーサイズ設定用ヘッダファイルです。。
kernel/smt	SMTDef.h	カーネルランド API 関数のライブラリヘッダファイルです。
kernel/include/smt	SMTAPI.h	カーネルランド API 関数のライブラリヘッダファイルです。
kernel/smt	smtcp_kernel.c	API ライブラリのカーネル依存部ソースファイルです。
		※API ライブラリバージョン 3.00 以上
kernel/include/smt	SMTAPI_CP.h	カーネルランドAPI関数のライブラリヘッダファイルです。
		※API ライブラリバージョン 3.00 以上

6.2. APIライブラリ適用後のディレクトリ構成例

Linux カーネルに API ライブラリの各ソースを適用した例を以下に示します。



6.3. ソースファイルの組み込み

6.3.1. APIライブラリファイルの組み込み確認

必要なファイルをカーネルツリー下にコピーし、トップディレクトリの Makefile を編集します。 ヘッダファイルとソースファイルは標準のディレクトリ構成では

\$(API_LIBRARY)/SoftwareModel/kernel/

にインストールされています。

既に「5 API ライブラリのコピー」でコピーは完了していますのでファイルが存在しているか確認します。

[HOST]\$ cd \$(KERNEL)
[HOST]\$ ls include/smt smt
include/smt:
SMTAPI.h SMTAPI_CP.h

smt/:

Makefile SMTDef.h smt.c Kconfig smt_sa_data.c smtBufferConfig.h smtcp_kernel.c

6.3.2. LinuxカーネルのMakefileを編集

Linux カーネルツリーのトップディレクトリの Makefile を編集します。<u>下線部</u>を追加してください。 一ヶ所のみ、core-y 変数に追加します。600 行目近辺ですが、バージョンにより前後します。

6.4. カーネルランドからのスタック解析機能を有効にする(APIライブラリバージョン 3.10 以降)

スタック解析機能は初期状態では出力されません。

1.\$(KERNEL)/include/smt/SMTAPI.h の_SMT_OUTPUT_SP_ADDR の定義値を_SMT_ON にしてください。

2.\$(KERNEL)/smt/smt_sa_data.c の_SMT_get_SP_address ()にスタックポインタアドレス値を取得するための実装を行ってください。

6.4.1. _SMT_get_SP_address関数実装

■機能

スタックポインタ値の取得を実施します。

※スタック解析機能を使用する場合にはユーザーシステムに合わせて実装してください。

■呼び出し手順

ユーザーから呼び出しません。API ライブラリ内部からコールされます。

■戻り値

型 : _SMT_UNSIGNED_32BIT_INTEGER 戻り値には取得したスタックポインタ値を指定してください。

6.4.2. 注意点

・スタック解析機能使用するためには関数トレース情報を出力する必要があります。

・スタックポインタのアドレスが 64bit の場合は使用できません。

6.5. Linuxカーネルのビルド

Makefile の編集が完了したらカーネルビルドを実施し、 エラーなくビルド出来た事を確認してください。

6.6. システムマクロトレースドライバの確認

組み込みを行ったシステムマクロトレースデバイスドライバが正しく動作しているか確認をします。 ターゲットコンソールで dmesg コマンドを実行したときのログ内に下の画像のように出力されてい る事を確認してください。

F	0.208523]	msgmni has been set to 931 io scheduler poop registered
	0.209354]	io scheduler deadline registered io scheduler ofg registered (default)
Ē	0.209421]	***** SMT Init ****
L	0.2125671	smtdat module loaded ***** SMT if optry loit *****
Ľ	0.212605]	smtif module loaded
Ē	0.212610]	***** SMT SW Init ****
L r	0.212625	smtsw module loaded

6.7. バッファーサイズの指定方法

バッファーサイズを指定する場合には smtBufferConfig.hの SMT_BUF_SIZE を環境に合わせて編集してください。

バッファーサイズはデフォルト 4MByte ですが、環境によって上限は異なります。

指定のバッファーサイズを確保できなかった場合、カーネルブートメッセージにメッセージが 出力されますので、バッファーサイズを減らして調整を行ってください。

Ē	0.209158]	io scheduler noop registered
Γ	0.209166]	io scheduler deadline registered
C	0.209222]	io scheduler cfg registered (default)
[0.209229]	***** SMT Init *****
Ε	0.209239]	vmalloc Error
Γ	0.209244]	Please reduce SMT_BUF_SIZE
Γ	0.210036]	registerd lp101.h1 LCD Driver.

7. プロセス遷移情報の取得

プロセス遷移をシステムマクロトレースに出力する方法を記載します。 スケジューラ内のコンテキストスイッチ処理直前に埋め込む例です。



7.1. コンテキストスイッチ関数の挿入

コンテキストスイッチ情報をシステムマクロトレース出力するために、(KERNEL)/kernel/sched.cを編集します。

※カーネルバージョンが 3.3 以降の場合は\$(KERNEL)/kernel/sched/core.c を編集してください。

7.2. SMT用ヘッダファイルのインクルード

sched.cに、システムマクロトレース用ヘッダファイルのインクルードを追加してください。

\$(KERNEL)/kernel/sched.c 90行目近辺ですが、バージョンにより前後します。 ※編集例は\$(API_LIBRARY)/SD/sample/kernel/sched.cをご覧ください。

```
extern void ct_thread_enter(struct task_struct *next);
extern void ct_thread_exit(struct task_struct *prev);
extern void ct_isr_enter(int irq);
extern void ct_isr_exit(int irq);
#endif /* CONFIG_CODETEST */
#define YDC_SMT_DEBUG
#ifdef YDC_SMT_DEBUG
#include <SMTAPLh>
#endif /* YDC_SMT_DEBUG */
/*
 * Convert user-nice values [ -20 ... 0 ... 19 ]
 * to static priority [ MAX_RT_PRIO..MAX_PRIO-1 ],
....
```

7.3. コンテキストスイッチ情報出力コードの挿入

コンテキストスイッチ情報出力コードを context_switch()関数の switch_to()直前に追加します。 \$(KERNEL)/kernel/sched.c context_switch 関数 2830 行目近辺ですが、 バージョンにより前後します。

※編集例は\$(API_LIBRARY)/SoftwareModel /sample/kernel/sched.c をご覧ください。

```
. . .
#ifndef __ARCH_WANT_UNLOCKED_CTXSW
   spin_release(&rq->lock.dep_map, 1, _THIS_IP_);
#endif
#ifdef YDC_SMT_DEBUG
  if( unlikely(!next->pid)) {
     _SMT_OsSwitch_Idle();
  }
  else{
     _SMT_OsSwitch_ThreadProcess_Name( (unsigned long)(next->pid), (unsigned
long)(next->tgid), next->comm, next->group_leader->comm);
  }
#endif /* YDC_SMT_DEBUG */
   /* Here we just switch the register state and the stack. ^{\prime\prime}
   switch_to(prev, next, prev);
 . .
```

7.4. Linuxカーネルのビルド

プロセス遷移情報出力のソースファイル編集が終了後にカーネルビルドを実施し、 正しくビルド出来た事を確認してください。

8. ユーザーランドAPIライブラリのビルド

ユーザーランドからデバッグ情報を出力するために、ユーザーランド API ライブラリをビルドし、生成 されたユーザーランド API ライブラリをデバッグ対象のアプリケーションにリンクする必要があります。 この章では、ユーザーランド API ライブラリの Makefile の編集、ビルドを実施します。



8.1. ファイル一覧

ユーザーランド API ライブラリのビルドに必要なファイルについて示します。

※\$(API_LIBRARY)/SoftwareModel/library 以下にあります。

ディレクトリ	ファイル名	説明
library	SMTDebugLevel.c	
library	SMTInit.c	
library	SMTPortOut.c	ユーザーランド API ライブラリソースファイルです。
library	SMTPrintf.c	
library	SMTUsrMsgTag.c	
library	TRQHook.c	
library	Makefile	ユーザーランド API ライブラリビルド用 Makefile です。
library	SMTDef.h	トレースフォーマット定義ヘッダファイルです。
library	SMTUser.h	ユーザー組込みヘッダファイルです。
library	SMTUserDef.h	デバッグプリント文レベル制御初期値設定ヘッダファイ
		ルです。
library	SMTAPI.h	ユーザーランド API 関数のライブラリヘッダファイルで
		す。
library	smtif.h	デバイスドライバインターフェースヘッダファイルです。
library	SMTAPI_CP.h	ユーザーランド API ライブラリソースファイルです。
		※API ライブラリバージョン 3.00 以上から対応
library	SMTCP.c	ユーザーランド API 関数のライブラリヘッダファイルで
		す。
		※API ライブラリバージョン 3.00 以上から対応

8.2. ユーザーランドからのスタック解析機能を有効にする(APIライブラリバージョン 3.10 以降)

スタック解析機能は初期状態では出力されません。

1.\$(API_LIBRARY)/SoftwareModel/library/SMTAPI.hの_SMT_OUTPUT_SP_ADDRの定義値を_SMT_ON にしてください。

2.\$(API_LIBRARY)/SoftwareModel/library/SMTInit.cの_SMT_get_SP_address()にスタックポインタアドレス値を取得するための実装を行ってください。

8.2.1. _SMT_get_SP_address関数実装

■機能

スタックポインタ値の取得を実施します。

※スタック解析機能を使用する場合にはユーザーシステムに合わせて実装してください。

■呼び出し手順

ユーザーから呼び出しません。API ライブラリ内部からコールされます。

■戻り値

型 : _SMT_UNSIGNED_32BIT_INTEGER 戻り値には取得したスタックポインタ値を指定してください。

8.2.2. 注意点

・スタック解析機能使用するためには関数トレース情報を出力する必要があります。

・スタックポインタのアドレスが 64bit の場合は使用できません。

8.3. ビルド設定

ユーザーランド API ライブラリをビルドする為に、

\$(API_LIBRARY)/SoftwareModel/library/Makefile ファイル先頭部分に定義されている変数を、

ご利用の環境に合わせて変更してください。

変数名称	説明
CROSS_COMPILE	クロスコンパイラ使用時のgcc プレフィックスを設定してください。

#			
# Set the tool chain.			
#			
CROSS COMPILE=linux-gnu-			
#			
# Set the library file name.			
#			
TARGET = libsmt.a			
#			
# You don't need to make change hereafter.			

8.4. ビルドの実行

次の手順でビルドを実行します。

[HOST]\$ make

実行後に、TARGET で設定された名称で、ライブラリが作成されている事を確認します。

```
[HOST] pwd
/home/<username>/APILib-Linux-REV/$(SMTIF)/library
[HOST $ make
linux-gnu-gcc -Wall -O2 -fomit-frame-pointer -I.-I../include/user
-I../include/common -I../driver -c -o SMTInit.o SMTInit.c
linux-gnu-gcc -Wall -O2 -fomit-frame-pointer -I.-I../include/user
-I../include/common -I../driver -c -o SMTPort.o SMTPort.c
linux-gnu-gcc -Wall -O2 -fomit-frame-pointer -I.-I../include/user
-I../include/common -I../driver -c -o SMTPortOut.o SMTPortOut.c
linux-gnu-gcc -Wall -O2 -fomit-frame-pointer -I../include/user
-I../include/common -I../driver -c -o SMTUsrMsgTag.o SMTUsrMsgTag.c
linux-gnu-gcc -Wall -O2 -fomit-frame-pointer -I../include/user
-I../include/common -I../driver -c -o SMTPrintf.o SMTPrintf.c
linux-gnu-gcc -Wall -O2 -fomit-frame-pointer -I../include/user
-I../include/common -I../driver -c -o SMTDebugLevel.o SMTDebugLevel.c
linux-gnu-gcc -Wall -02 -fomit-frame-pointer -I../include/user
-I../include/common -I../driver -c -o TRQHook.o TRQHook.c
linux-qnu-ar rv libsmt.a SMTInit.o SMTPort.o SMTPortOut.o SMTUsrMsgTag.o
SMTPrintf.o SMTDebugLevel.o TRQHook.o
linux-gnu-ar: creating libsmt.a
a - SMTInit.o
a - SMTPort.o
a - SMTPortOut.o
a - SMTUsrMsqTaq.o
a - SMTPrintf.o
a - SMTDebuqLevel.o
a - TRQHook.o
[HOST]$ ls
Makefile SMTInit.c SMTPort.o SMTPrintf.c SMTUsrMsqTaq.o (libsmt.a
SMTDebugLevel.c SMTInit.o SMTPortOut.c SMTPrintf.o
                                                       TRQHook.c
SMTDebugLevel.o SMTPort.c SMTPortOut.o SMTUsrMsgTag.c TRQHook.o
[HOST]$
```

9. ユーザーランドからの出力確認用アプリのビルド

ユーザーランドからの出力確認を行うため、テストアプリを作成してファイルシステムに組み込みを行 います。





9.1. Makefileの編集

\$(API_LIBRARY)/startupにあるサンプルソースをビルドするためにMakefile先頭部分の以下に示す変数 を、ご利用の環境に合わせて適宜エディタで編集してください。

ここでは、ユーザーランド API ライブラリファイル名は、libsmt.a で作成されたと仮定します。

変数名称	説明
CROSS_COMPILE	クロスコンパイラ使用時のgcc プレフィックスを設定します。
SMT_LIBDIR	SMT ユーザーランド API ライブラリのあるディレクトリを指定します。
SMT_LIB	ユーザーランド API ライブラリファイル名設定します。

```
CROSS_COMPILE =linux-gnu
SMT_LIBDIR =$(API_LIBRARY)/SoftwareModel/library
SMT_LIB =libsmt.a
MAKE_CP =
MAKE_CP_CFG =
CC = $(CROSS_COMPILE)gcc
...
```

9.2. ビルドの実行

次の手順でビルドを実行します。

```
[HOST]$ cd $(API_LIBRARY)/startup
[HOST]$ make
```

make の実行結果にエラーがないことを確認し、helloworld が生成されたことを確認してください。

9.3. ファイルシステムへの組み込み

生成された helloworld を、ユーザーシステムのルートファイルシステムに組み込んでください。
10. SMTログファイル出力デーモンの組込み

この章では SMT ログファイルをファイル出力するためのデーモンについて記載します。





10.1. ファイル一覧

SMT ログファイル出力デーモンのビルドに必要なファイルについて示します。 ※ \$(API_LIBRARY)/SoftwareModel/smt_userp/以下にあります。

ファイル名	説明
Makefile	SMT ファイル出力デーモンの Makefile です。
smt_proc.c	SMT ファイル出力デーモンのソースファイルです。
smtconfig.h	SMT ファイル出力デーモンの設定用ヘッダファイルです。

10.2. SMTログファイル出力デーモンの設定

ファイル出力先、ファイル出力最大サイズの設定を smtconfig.h を環境に合わせて設定を行ってください。

変数名	説明
SMT_LOG_FILE_OUTPUT_DIREC TORY	SMT ログファイル出力先のディレクトリ指定をします。 ※出力先は必ずライト権限のがある場所を指定してください。
SMT_OUTPUT_MAX_FILE_SIZE	SMT ログファイル出力の最大サイズを指定します。 (デフォルト 1GByte) ※この設定値の値は 4GByte 未満 1MByte 以上の値を設定してくだ さい。

10.3. Makefileの編集

Makefile の CROSS_COMPILE 指定を環境に合わせてを行ってください。

10.4. SMTファイル出力デーモンのビルド

make を実行して SMT ファイル出力デーモンのビルドを実施してください。 ビルド正常に終了すると smtex が生成されます。

[HOST]\$ cd \$ (API_LIBRARY)/SoftwareModel /smt_userp [HOST]\$ make

10.5. SMTログファイル出力デーモンの組込み

生成された smtex をユーザーシステムのファイルシステムへ転送を行って実行権限を付与してください。

10.6. 注意事項

- ・ SMT ログファイル出力先の設定は必ずライト権限がパスを設定してください。
- ・ SMT ログファイル出力最大サイズは環境に合わせて変更する必要がありますが、1MByte 以下には 設定しないでください。
- SMT ログファイル出力デーモンはトレース ON 状態の時には一定時間毎に SMT ログファイルを出 力します。

11. システムマクロトレース取得方法

本章ではシステムマクロトレース出力確認を行います。

「4.API ライブラリのインストール」から「10 SMT ログファイル出力デーモンの組込み」までを実施するとシステムマクロトレースで下記の情報出力の確認が出来ます。

- ・ プロセス遷移情報
- サンプルアプリからの SMT デバッグプリント文出力

取得手順は以下のとおりです。

- 1. SMT ログファイル出力デーモンの起動
- 2. トレースの開始
- 3. トレースの終了
- 4. SMT ログファイルを WindowsPC へ移動
- 5. macroTRACE-VIEWER でデータ参照



11.1.SMTログファイル出力デーモンの起動

SMT ログファイル出力デーモンの起動オプション、実行モード、起動コマンド例について説明します。

11.1.1. 起動オプション

SMT ログファイル出力デーモン実行時には下記表のオプションがあります。

オプション	説明
-d <dir></dir>	SMT ログファイル出力先を dir に指定されたディレクトリに設定します。
	※出力先は必ずライト権限のがある場所を指定してください。
-m <mode></mode>	SMT 出力動作モードを mode に指定されたモードに設定します。
	出力動作モードは以下のいずれかを指定してください。
	continuous:長時間トレースモード
	ring:直近データのみ残すトレースモード
	onetime:指定サイズでトレースを終了するトレースモード
	※デフォルトは onetime となります。
	各モードの詳細については「11.1.2 出力動作モードについて」を参照してください。
-t <sec></sec>	トレース開始から sec 秒後にトレースを終了します。
-h	起動オプションのヘルプを表示します。
-s <size></size>	SMT ログファイルの出力サイズを size に指定されたサイズに設定します。

モード	説明
continuous	トレース開始から終了までデータを取得します。
	continuous モードの場合 smt.iidx と smtXXXX.clog(XXXX部分は数字)の2種類
	のファイルが作成されます。
	smtXXXX.clog ファイルは SMT_OUTPUT_MAX_FILE_SIZE 毎に数字がインクリメン
	トされます。
	(Windows にコピーする際にはこの2種類のファイル全てをコピーしてください)
ring	トレース開始から終了までの間の直近のデータを取得します。
	最大データサイズは SMT_OUTPUT_MAX_FILE_SIZE の 2 倍になります。
onetime	トレース開始から SMT_OUTPUT_MAX_FILE_SIZE で指定されたサイズのデータを取得し
	ます。

11.1.2. 出力動作モードについて

11.1.2.1. continuous モード実行イメージ





34

11.1.3. SMTログファイル出力デーモン起動コマンド例

SMT ログファイル出力デーモン(smtex)を/home/root に配置したと仮定して説明を行います。

11.1.3.1. 実行権限付与について

smtex への実行権限がない場合は chmod コマンドで実行権限を付与してください。

SMT ログファイル出力デーモンへの実行権限付与コマンド例

[TARGET]\$ chmod a+x /home/root/smtex

11.1.3.2. SMT ログファイル出力デーモンの起動コマンド例1

SMT_OUTPUT_MAX_FILE_SIZE までログを取得する時のコマンド例です

[TARGET]\$cd /home/root
[TARGET]\$ls
smtex
[TARGET]\$./smtex &
[SMTD] Logging daemon start !

11.1.3.1. SMT ログファイル出力デーモンの起動コマンド例 2

ring モードで SMT ログファイル出力先を/home/root/にする時のコマンド例です。

[TARGET]\$cd /home/root
[TARGET]\$ls
smtex
[TARGET]\$./smtex -m ring -d /home/root &
[SMTD] Logging daemon start !

11.2. トレースの開始と終了方法について

ターゲットコンソールの/proc/smtswを操作する事による SMT の出力設定を切り替えることができます。 トレース開始する時には必ず SMT ログファイル出力デーモンが起動されていることを確認してから実行 してください。

11.2.1. 有効化方法

ターゲットコンソールで以下のコマンドを実行してください。

```
[TARGET]$echo 1 > /proc/smtsw
または
[TARGET]$echo on > /proc/smtsw
```

11.2.2. 無効化方法

ターゲットコンソールで以下のコマンドを実行してください。

[TARGET]\$echo 0 > /proc/smtsw または

[TARGET]\$echo off > /proc/smtsw

トレース OFF 後全ての SMT ログファイル出力が完了すると

「[SMTD]SMT data output Finished!!」と出力されます。

11.2.3. 注意事項

・SMT ログファイル出力デーモンを実行せずにトレースの ON/OFF した場合には下記のようなメッセ ージが出力されます。

[TARGET]\$echo 1 > /proc/smtsw Please exec smtex(daemon for outputting SMT log file)

・SMT ログファイル出力前に SMT_BUF_SIZE 以上のデータを出力した場合には下記のようなメ ッセージを出力しトレースを終了します

```
[SMT] buffer full
[SMT] smt sw off
```

11.3. サンプルデータ取得

ここまでの作業でプロセス遷移情報、helloworld アプリケーションからの SMT 情報出力ができる環境が 整いましたので確認のためのデータ取得を行います。

/home/root に helloworld アプリ、smtex があり、SMT ログファイルの出力先とファイル名を smt.slog と 仮定して説明をします。

[TARGET]\$cd /home/root [TARGET]\$ls smtex helloworld [TARGET]\$./smtex & [SMTD] Logging daemon start ! [TARGET]\$echo 1 > /proc/smtsw [TARGET]\$/home/root/helloworld & [TARGET]\$echo 0 > /proc/smtsw [SMTD]SMT data output Finished!! [TARGET]\$ls smtex helloworld smt.slog

11.4. macroTRACE-VIEWERについて

取得したサンプルデータをWindowsPCにコピーしてmacroTRACE-READERで開く方法について記載し ます。(continuous モードで記録した場合、インデックスファイル(smt.iidx)と共にログファイル (smtXXXXXXXX.clog)もコピーしてください。)

11.4.1. macroTRACE-READERを起動する

スタートアップメニューの「プログラム」→「YDC」→「macroTRACE-VIEWER」→「macroTRACE-READER」より起動します。



ログ選択ダイアログを表示されるので、プロジェクト名(StartUp)を設定します。

取得した SMT ログファイル(smt.slog または smt.iidx)を選択し「OK」ボタンを押してください。 C:¥YDC¥macroTRACE-VIEWER にコピーしたと仮定します。

ログ選択ダイアログ			×	
 	artUp		▼	
 新規プロジェクトの作成 プロジェクト名 StartUp 保存位置 			•	
C¥YDC¥macroTRACE-VIEWER ログの選択 Date マ	LogName	FileSize(KB)	•	出力動作モードが onetime、ring で取得し た ログは TVDE が
水 6/25/2014 04:42 午後	smt	3009	SLOG	にロクはTIPEか SLOG と表示されます
水 6/25/2014 04:35 午後	smt	1	IIDX	BEOG CANCHUR /
C#YDC#macroTRACE-VIEWER	<u>Ок</u> ++уен		•	出力動作モードが continuous で取得した ログは TYPE が IIDX と 表示されます

11.4.2. Indexデータ生成設定 (onetimeモード、ringモード)

Index データ生成設定を行います。

ユーザーシステムの Endian を選択して「OK」ボタンを押してください。 出力ファイルのテキストボックスに Index データの出力先を指定してください。

Indexデータ生成設定	×
エンディアン ・レディアン ・ビックエンディアン 	
出力ファイル C:¥YDC¥macroTRACE-VIEWER¥smt.ilog 進捗	参照
ОК	++>th

11.4.3. Indexデータ生成設定 (continuousモード)

Index データ生成設定を行います。

ユーザーシステムの Endian を選択して「OK」ボタンを押してください。

出力ファイルのテキストボックスに Index データの出力先ディレクトリを指定してください。

Indexデータ生成設定	×	
-エンディアン ●リトルエンディアン ◎ビックエンディアン		
出力ディレクトリ C:¥YDC¥macroTRACE-VIEWER	参照	
進捗		
	OK キャンセル	

11.5. macroTRACE-VIEWERでのトレース取得例

ファイ	Ίν ι(<u>E</u>)	表示(⊻)	検索(<u>S</u>) 記録(<u>R</u>) ツール(<u>T</u>	:) プロファイル(<u>P</u>)		プ(<u>H</u>)					
) a		🗊 🚊 🗖) 🗛 🔳 📨 🤅 💿 🔟 🖉 🔾	き 🗓 🥜 🗄 テーマ	•	:# 🗲	*	🗟 💸 🛪	>		
チャ									-		
: -											п,
: "	JUXE						_				÷ /
Mar	k Index	Sub	Time	DifferenceTime	Core	Thread	Attr	Level	Message	Information	Ev
<u>A</u> :	-11	3456	00:00:02.526m900u600n		1	<idle></idle>	IDLE				00
à'r i	-11	3488	00:00:02.527m144u940n	244u340n	1	TID=0000003	SW	P/T	TID=000003E(irg/106		00
2	-11	3664	00:00:02527m144u940n	On	1	<idle></idle>	IDLE				00
ŝ.	-11	3680	00:00:02.527m175u480n	30u540n	0	<idle></idle>	IDLE				00
2	-11	3712	00:00:02.579m612u900n	52m437u420n	0	TID=00000169	SW	P/T	TID=00000169(kworker/		00
ż	-11	3856	00:00:02:579m735u060n	122u160n	0	<idle></idle>	IDLE				00
ý.	-11	3888	00:00:02:587m430u560n	7m695u500n	0	TID=000000C	SW	P/T	TID=000000CB(sync_su		00
2	-11	4032	00:00:02.587m430u560n	On	0	<idle></idle>	IDLE				00
	-11	4064	00:00:02.679m100u440n	91 m669u880n	1	TID=0000003	SW	P/T	TID=0000003E(irq/106		00
Ş.	-11	4240	00:00:02.679m100u440n	On	0	TID=00000169	SW	P/T	TID=00000169(kworker/		00
	-11	4400	00:00:02.679m130u960n	30u520n	1	<idle></idle>	IDLE				00
ξ.	-11	4416	00:00:02.679m130u960n	On	0	TID=0000021	SW	P/T	TID=0000021 A(ash)/PI		00
ξ π υ	-11	4496	00:00:02.679m192u040n	61u080n	1	TID=0000003	SW	P/T	TID=0000003E(irq/106		00
2	-11	4672	00:00:02.679m222u560n	30u520n	1	<idle></idle>	IDLE				00
2	-11	4688	00:00:02.679m436u280n	213u720n	1	TID=0000003	SW	P/T	TID=0000003E(irq/106		00
5	-11	4864	00:00:02.679m466u820n	30u540n	1	<idle></idle>	IDLE				00
2	-11	4880	00:00:02.679m466u820n	On	0	<idle></idle>	IDLE				00
3	-11	4912	00:00:03.321 m780u880n	642m314u060n	0	TID=0000021	SW	P/T	TID=0000021D(flush-17		00
ζ.	-11	5056	00:00:03.321 m811u400n	30u520n	0	<idle></idle>	IDLE				00
3	-11	5088	00:00:03.775m112u180n	453m300u780n	1	TID=0000003	SW	P/T	TID=000003E(irg/106		00
2	-11	5264	00:00:03.775m112u180n	On	0	TID=00000169	SW	P/T	TID=00000169(kworker/		00
2	-11	5408	00:00:03.775m142u700n	30u520n	1	<idle></idle>	IDLE				00
3	-11	5424	00:00:03.775m142u700n	On	0	TID=0000021	SW	P/T	TID=0000021 A(ash)/PI		00
3	-11	5504	00:00:03.775m203u740n	61u040n	1	TID=0000003	SW	P/T	TID=0000003E(irq/106		00
•											•

macroTRACE-VIEWER でのトレース取得例を示します。

12. システムマクロトレース出力確認方法

本章ではシステムマクロトレース出力確認を行います。

・ プロセス遷移情報

「4.API ライブラリのインストール」から「10 SMT ログファイル出力デーモンの組込み」までを実施するとシステムマクロトレースで下記の情報出力の確認が出来ます。

・ サンプルアプリからの SMT デバッグプリント文出力
 c ユーザーランドからの出力実装

 d システムマクロトレース出力確認
 [11 システムマクロトレース取得方法]
 [12 システムマクロトレース出力確認方法]

 e 関数トレース取得方法

12.1. プロセス遷移出力確認

プロセス遷移出力が正しく出力されているか確認を行います。

12.1.1. プロセス遷移情報の出力確認手順

- 1. macroTRACE-VIEWER のメニュー「表示」→「チャート」または 「ボタンをクリックしてチャート表示をします。
- 2. チャートウィンドウにタスクスイッチ出力が表示されているか確認します。
- 3. ログリスト内のトレース結果が出力されていますので、Attr 列に SW または IDLE 表示の物が出力 されているかどうか確認してください。

出力サンプル例を以下に示します。



12.2. ユーザーランドAPI出力確認

「12.1 プロセス遷移出力確認」でタスク情報の出力確認が出来ている事を前程として説明を行います。

12.2.1. ユーザーランドからのシステムマクロトレース出力確認方法

- 1. macroTRACE-VIEWER のメニュー「検索」→「絞り込み検索」を選択してください。
- 2. 「カテゴリ」欄から「デバッグプリント」を選択して「実行」ボタンを押してください。
- 3. 「絞込みリスト」にデバッグプリント出力結果のみ表示されます。
- 4. 「絞込みリスト」の Messege 列に「main() called」等が表示されていればユーザーランドからのシス テムマクロトレース出力確認は終了です。

取得例をを以下に示します。



13. カーネルランドからの関数トレース取得方法

本章では、チェックポイント挿入ツールを使用してカーネルランドのデバイスドライバの関数履歴を取 得するための手順について、\$(KERNEL)/drivers/serialを対象とした場合を例に説明します。

[※]チェックポイント挿入ツール(Cpe,make_cp)は、関数の入り口/出口に対して、チェックポイントを挿入することで関数履歴を取得する為のツールです。



事前に、チェックポイント挿入ツールのインストール及び環境設定を行ってください。

13.1. ネットワークドライブの設定

開発 Linux ホストと Windows ホストのファイル共有は、ネットワークドライブの割り当てを実施して Windows ホストからアクセスが出来る事をご確認ください。

ここでは、Windows ホストのWドライブを開発Linux ホストの/home/<username>に割り当てたと仮 定します。

各ファイルのパス設定は以下の設定で実施しています。

設定内容	開発 Linux ホスト設定値	Windows ホスト設定値
ネットワークドライブ設定	/home/ <username></username>	W:¥
チェックポイントワークス	/home/ <username>/WorkSpace</username>	W:¥WorkSpace
ペース設定		
チェックポイント挿入対象	/home/ <username>/kernel/dr</username>	W:¥kernel¥drivers¥serial
ディレクトリ	ivers/serial	

13.2. Cpe ワークスペース新規作成

13.2.1. Cpe起動

「スタートメニュー」→「プログラム」→「YDC」→「CheckPointTools」→「Cpe」をクリックし、Cpe を起動します。

YDC
🛃 APClient.exe
🐌 CheckPointTools
🗏 Cpe

13.2.2. ワークスペース新規作成

「ファイル」→「新規作成」をクリックすると、「新規プロジェクト」ダイアログが開きます。



13.2.3. 言語の選択

	(1)	[言語選択]より、	専用コンパイラ版	$\mathcal{D} \left[C/C + + \right]$	を選択します
--	-----	-----------	----------	--------------------------------------	--------

言語選択	×
言語とチェックポイント挿入方法を選択してください。	
(チェックポイント挿入とは関数履歴を取得するためのAPIコールを挿入することです)	
C C/C++	
ソー人ファイルを書き換えチェックボイントを挿入します。 書き換えられたソースファイルをビルドしてください。	
IDEやコンパイラ環境を問わず使用できます。	
注意: Javaは、Android対応のAPI Libraryのみご利用頂けます。	
▶ 専用コンパイラ版	
© C/C++	
ビルトサに、コノハイラと連携しチェックポイントを挿入します。	
連携対応しているコンパイラは以下になります。	
・ARM社製 C/C++コンパイラ ・GNU Compiler Collection	
・Microsoft社 Platform Builder	
• RENESASKI Superh RISC C/C++_J)/(1.5	
< 戻る(B) 次へ(N) > キャ	ンセル

② 設定が完了したら、「次へ」をクリックします。

13.2.4. 新規ワークスペース

設定項目	設定内容	
ワークスペース名(<u>W</u>)	myWorkspace	
保存先ディレクトリ(<u>D</u>)	W:¥Workspace	
コンパイラ HOST (<u>H</u>)	Linux	
CCFG ファイル(<u>R</u>)	GCC を選択してください。	
	もし異なるコンパイラを使用している場合には TL 以外で始まる	
	CCFG ファイルの中から該当するコンパイラを選択してください	

	myWorkspace			
(男友生=シルカロ)(の)。	, W:¥workspace		泰昭(B)	
1#1777710717(<u>0</u>)			39 Att (D/	
チェックホペイント開始番号(S):	1			
▼ ソースファイルの更新日付	の変更を許可する(工)			
コンパイラHOST(<u>H</u>)	Linux	•		
ccfgファイル(<u>R</u>):	,			
CCFG File		Remark	s	
ARMCC	ARM C/C+	+ compiler 2.0-5.01		:
CL	Microsoft P	'latform Builder		
GCC	GNU Comp	iler Collection 3.0-4.8		
SHC	SH4 SHC 9	.1.1.0-9.4.0.0		
TLH400	SH2A SHC	9.1.1.0-9.4.0.0 (BUS IF, H	łI7000/4)	
	SH4 SHC 9	.1.1.0-9.4.0.0 (BUS IF, RT	TOS)	
TLH401	GCC AM34 3.0-4.8 (BUS IF, Linux)			
TLH401 TLU200_TLJ002	GUU AM34			

設定が完了したら、「次へ」をクリックします。

13.2.5. プロジェクト作成

ここではプロジェクト名を"kernel_serial"として作成した例を紹介します。

- ① [プロジェクト名]に、新規プロジェクト名「kernel_serial」を入力してください。
- ② [次のディレクトリから]に、ソースファイルを検索するディレクトリを入力してください。
- ③ ネットワークドライブ割当された Linux ホスト PC 上のディレクトリを指定してください。
- ④ [Search] を押してソースファイルの検索を開始してください。
- ⑤ [Source File]に見つかったソースファイルが表示されます。
- ⑥ プロジェクトに登録するファイルにチェックを入れます。
- ⑦ [次へ]をクリックします。

プロジェクトイ作成				— ×
プロジェクト名(<u>P</u>):	kernel_serial			
)欠のディレクトリから(<u>D</u>)	W:¥kernel¥drivers¥tty¥serial		参照(<u>B</u>)	
ファイル拡張子(<u>E</u>):	c,cc,cp,cxx,cpp,CPP,c++,C		<u>S</u> earch	
93 items found				
	Source File	•		
🗖 🗹 🔂 serial				
🕀 🗖 🧰 cpm_uart				
🗄 🗖 🧰 jsm				
⊞ 🔲 🖹 21285.c				
🖽 🗖 🖹 68328serial.	;			
🕀 🗖 🖹 68360serial.c	•			
🕀 🗹 🖹 8250.c				
🕀 🗖 🖹 8250_accent	c			
🕀 🗖 🖹 8250_acom.c				
🕀 🗖 🖹 8250_boca.c				
🕀 🗹 🖹 8250_early.c				
🕀 🗖 🖹 8250_exar_st	16c554.c			
📃 🕀 🗖 🖹 8250_fourpor	tc			
		< 戻る(<u>B</u>))次へ(<u>N</u>) >	キャンセル

13.2.6. パス変換設定

Windows パスと Linux ホスト PC 上のパスを対応させるための変換条件設定です。 複数の条件を設定した場合、上方の設定が優先されます。



WorkSpace ディレクトリが新規作成されます。「**すべて はい」**をクリックします。



新規作成後、以下のような画面状態になります。



新規ワークスペース作成時に指定したディレクトリにワークスペースファイル(拡張子.ctx)が出来ていることを確認します。

O ♥ W:¥myWorkspace	✓ ← myWorkspaceの検	索 ዖ
ファイル(<u>E</u>) 編集(<u>E</u>) 表示(Y) ツール(I) ヘルプ(<u>H</u>)		
整理 ▼ 新しいフォルダー		•
☆ お気に入り 1 名前 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	更新日時	種類
kernel_corial	2013/08/19 15:52	ファイル フォル…
溒 ライブラリ 🕺 myWorkspace.ctx	2013/08/19 15:52	Cpe Workspace
■ ドキュメント		
E ビクチャ ▼ < III		4

13.3. ライブラリサーチパス設定の実施

『システムマクロトレースチェックポイント挿入ツールインストールガイド』に従って開発 Linux ホストにインストールした make_cp のパスをライブラリサーチパスに追加設定します。 ここでは make_cp を/home/smt にインストールしたと仮定します。

[HOST]\$ export LD_LIBRARY_PATH=/home/smt/make_cp:\$LD_LIBRARY_PATH

13.4. MakeFileの編集

\$KERNEL/ Makefile に対して make 変数(MAKE_CP、MAKE_CP_CFG)を追加し、 make_cp(チェックポイント挿入ツール)を呼び出す設定を追加します。

make 変数	説明
MAKE_CP	make_cp 実行ファイルを指定します。
	『システムマクロトレースチェックポイント挿入ツールインストールガイ
	ド』に従って開発 Linux ホストにインストールした make_cp のパスを指定し
	てください。
MAKE_CP_CFG	make_cp 設定ファイルを指定します。
	「13.2 Cpe ワークスペース新規作成」 で作成したワークスペースのパスを指
	定してください。
CC	make_cpの動作設定値を追加設定します。
CXX	

Makefile 修正前

CC = \$(CROSS_COMPILE)gcc

Makefile 修正後

```
YDC_SMT_MAKECP := yes
ifeq ($(YDC_SMT_MAKECP),yes)
MAKE_CP = /home/<username>/make_cp/make_cp
MAKE_CP_CFG = -ctx /home/<username>/WorkSpace/myWorkSpace.ctx -cc
CC = $(MAKE_CP) $(MAKE_CP_CFG) $(CROSS_COMPILE)gcc
else
CC = $(CROSS_COMPILE)gcc
endif
```

13.5.ビルドの実行

トップディレクトリに移動してビルドを行います。ビルドが完了したら、イメージを作成します。 ビルド中の画面に make_cp 動作中メッセージが表示されます。

[HOST]\$make •••(中略)•••
make cp: As for /home/ <username>/kernel/drivers/video/cfbcopvarea_c_oply the compiling is peformed</username>
make cp: As for /home/susernames/kernel/net/inv4/fib semantics c only the compiling is peformed
make on: Compile only due to compiler ention
make_cp. Compile only due to compiler option.
CC drivers/video/cfbimgblt.o
<pre>make_cp: As for /home/<username>/kernel/drivers/video/cfbimgblt.c, only the compiling is peformed.</username></pre>
CC net/ipv4/inet_fragment.o
<pre>make_cp: As for /home/<username>/kernel/net/ipv4/inet_fragment.c, only the compiling is peformed.</username></pre>
<pre>make_cp: Compile only due to compiler option.</pre>
<pre>make_cp: Compile only due to compiler option.</pre>
LD drivers/video/fb.o
LD drivers/video/built-in.o
CC drivers/watchdog/omap_wdt.o
 ・・・(中略)・・・
<pre>make_cp: Compile only due to no specified source files.</pre>
make_cp Warning: Because of no specified source files, execute compilation only.
LD [M] drivers/gator/gator.ko
LD arch/arm/boot/compressed/vmlinux
OBJCOPY arch/arm/boot/zImage
Kernel: arch/arm/boot/zImage is ready
[HOST]\$

13.6. macroTRACE-VIEWERの設定

デバッグ情報を記録する際に、macroTRACE-VIEWER で必要な設定について説明します。

(1) 「**ツール」→「パラメータ設定」**または、 22を選択し、「**パラメータ設定」**ダイア ログを表示します。

Mcr	🚾 macroTRACE-VIEWER 【閲覧モード】 【smt.tlog】						
1.5	··· ファイル(F) 表示(V) 検索(S) 記録(R)			ש–	·ル(T) プロファイル(P) へ	リレプ(H)	
E		a i •	i 🖹 🚺	🗛 🔳 👝 i 💿	2	パラメータ設定(S)	Ctrl+S 😑
÷.	: ログ	リスト			d.	イベント設定(E)	Ctrl+E
거방	Mark	Index	Sub			パス設定(P)	
K		-1	802048	00:00:07.196m899		チェックポイントファイルの	更新(C)
Ň	会	-1	802128	00:00:07.196m899i) ±))////) / ////	2
_	会	-1	802208	00:00:07.196m899i		メッセージタグファイルの更	新(M)
	\$	-1	802320	00:00:07.196m929		コンテキストID交換ファイル	の面新(工)
		-1	802384	00:00:07.196m929		コンテキストロを探フティル	

- (1) 左のツリーから「基本」を選択します。
- (2) 「チェックポイントワークスペースファイル」に、「13.2 Cpe ワークスペース新規作 成」で作成した「W:¥WorkSpace¥myWorkSpace¥myWorkSpace.ctx」を指定し ます。
- (3) 「OK」をクリックして、変更を保存します。

パラメータ設定				— ×
+基本マルチコア	ーチェックポイントワークスペースコ	ファイル		
	File	Checkpoint	Path	追加
□	myWorkspace.ctx	1:3	₩¥my₩o	編集
Real-Time Viewer				削除
□ フロファイル □ プロセスプロファイル □ スレッドプロファイル				全削除
 				
- 終了 - ブックマーク	- メッセージタグファイル			
	タイムスタンブ分解能			
	20 nsec 💌			
	エンディアン			
	リトルエンディアン・			
(■ 100 × 100				
	OF		f#)tu	適用

デバッグ情報を記録するための設定が完了しました。

13.7. システムマクロトレース取得例

取得例は下記のようになります。

関数チャートが表示されない場合、

下図の緑枠に示す"関数チャート/スレッドチャートの切り替えボタン"でチャート表示を切り替えて 関数チャートが表示されるかどうか確認してください。



14. ユーザーランドからの関数トレース取得方法

本章では、チェックポイント挿入ツールを使用してユーザーランドプログラムで関数履歴を取得するた めの手順について説明します。



14.1.サンプルプログラムのビルドの実施

14.1.1. ソースコードの確認

\$(API_LIBRARY)/startup/に移動してください。次のサンプルソースファイルがあります。

ファイル名	処理概要		
hellowold.c	メイン関数		
SMTUser.c	デバッグ制御処理		
Makefile	ビルド用 makefile		
[HOST]\$ cd \$(API_I [HOST]\$ ls Makefile SMTUser.	<pre>[HOST]\$ cd \$(API_LIBRARY)/startup/ [HOST]\$ ls Makefile SMTUser.c helloworld.c</pre>		

14.1.2. Makefileを編集

サンプルソースをビルドするために Makefile 先頭部分の以下に示す変数を、ご利用の環境に合わせて適 宜エディタで編集してください。

ここでは、ユーザーランド API ライブラリファイル名は、libsmt.a で作成されたと仮定します。

変数名称	説明
CROSS_COMPILE	クロスコンパイラ使用時のgccプレフィックスを設定します。
SMT_INCDIR	ユーザーランド API ライブラリヘッダファイルのサーチパスを設定します。
SMT_LIBDIR	ユーザーランド API ライブラリのサーチパスを設定します。
SMT_LIB	ユーザーランド API ライブラリファイルを設定します。

編集例

```
CROSS_COMPILE = linux-gnu-
SMT_LIBDIR = ~/APILib-Linux-REV/SoftwareModel/library
SMT_LIB = libsmt.a
```

14.1.3. ビルド実施

編集した Makefile を使用して make をしてください。

```
    コマンドプロンプトにて、コマンドを入力してください。
```

[HOST]\$ make

②実行後、helloworld ファイルが生成されていることを確認してください。

```
[HOST]$ ls
Makefile SMTUser.o helloworld.c
SMTUser.c helloworld helloworld.o
```

14.2. ライブラリサーチパス設定の実施

『システムマクロトレースチェックポイント挿入ツールインストールガイド』に従って開発 Linux ホストにインストールした make_cp のパスをライブラリサーチパスに追加設定します。 ここでは make_cp を/home/smt にインストールしたと仮定します。

```
[HOST]$ export LD_LIBRARY_PATH=/home/smt/make_cp:$LD_LIBRARY_PATH
```

14.3. Cpe/make_cp挿入ツール設定

ここでは、チェックポイント挿入ツールを使用して、サンプルプログラム(helloworld)の関数履歴を 取得するための手順について説明します。

※チェックポイント挿入ツールは、関数の入りロ/出口に対して、チェックポイントを挿入することで 関数履歴を取得する為のツールです。

事前に、チェックポイント挿入ツールのインストール及び環境設定を行ってください。

14.3.1. プロジェクトの追加作成

ここでは「13 カーネルランドからの関数トレース取得方法」で作成したワークスペースにプロジェクト "sampleApp"を追加する例を紹介します。

① Cpe のワークスペース名を右クリックして「プロジェクトの追加」を選択します。

Repervention of the Weekspace of the Wee	ace¥myWorkspace.ctx
· ファイル(F) 編集(E) 昇	表示(V) オプション(O) ソール(T) ヘルプ(H)
i D 🚅 🖬 📮 🙀 💡	1888.
ワークスペース	
Name	
E DemyWork	プロジェクトの追加
	削除
	全て展開
	±7190 ►
	外部エディタで開く
Output	チェックポイント挿入
	チェック木。イント削除
	フ° ロハ° ティ
	プリプロセッサ条件一覧

- ② [プロジェクト名]に、新規プロジェクト名「sampleApp」を入力してください。
- ③ [次のディレクトリから]に、ソースファイルを検索するディレクトリを入力してください。
- ④ [Search] を押してソースファイルの検索を開始してください。
- ⑤ [Source File]に見つかったソースファイルが表示されます。
- ⑥ プロジェクトに登録するファイルにチェックを入れます。
- ⑦ [次へ]をクリックします。

「次のディレクトリから(D)」には、関数履歴取得対象のディレクトリの W:¥\$(API_LIBRARY)¥sartupを設定し、「<u>S</u>earch」ボタンを押してください。 SouerceFile の helloworld.c ファイルをチェックし、「完了」ボタンを押してください。

プロジェクト作成		
プロジェクト名(<u>P</u>):	sampleApp	
)次のディルクトリから(<u>D</u>)	/ W:¥\$(API_LIBRARY)¥startup	参照(<u>B</u>)
ファイル拡張子(<u>E</u>):	c,cc,cp,cxx,cpp,CPP,c++,C	<u>S</u> earch
2 items found		
	Source File	
🖃 🗹 🔂 startup		
🗄 🗹 🖹 helloworld.c		
🗄 🗖 🖹 SMTUser.c		
	〈戻	る(B) 完了 キャンセル

14.3.2. チェックポイント挿入ソース選択の完了

チェックポイント対象のソースファイル登録が完了すると、対象のプロジェクト、ソースファイルが一覧表示されます。

チェックポイントワークスペースファイルを保存して、終了してください。

R Cpe W:¥myWorkspace¥myWorkspace.ctx		- • •
[⋮] ファイル(<u>E</u>) 編集(<u>E</u>) 表示(⊻) オプション(<u>Q</u>) ツール(<u>T</u>) ヘルプ(<u>H</u>)		
- D 🚅 🖬 💭 🐜 💡 I 🗗 🖓 👘 -		
ワークスペース	д х	
Name	Line	
🖃 🗷 🧰 myWorkspace		
++ 🖉 🧰 kernel serial		
L SampleApp		
L 🛛 🖹 helloworld.c		
J : Output		4 x
; output		
Output Search1 Search2		
ντ° 1		

14.3.3. Makefile.smt を編集

変数 MAKE_CP、MAKE_CP_CFG、CC を修正し、make_cp(チェックポイント挿入ツール)を呼び出すようにします。

変数名称	説明
CROSS_COMPILE	クロスコンパイラ使用時のgcc プレフィックスを設定します。
SMT_LIBDIR	SMT ユーザーランド API ライブラリのあるディレクトリを指定します。
SMT_LIB	ユーザーランド API ライブラリファイル名設定します。
MAKE_CP	make_cp 実行ファイルを設定します。
CPE_WORKSPACE	Cpe で作成したワークスペースを指定します。

Makefile.smt 編集前

```
CROSS_COMPILE =
SMT_LIBDIR=
SMT_LIB =
MAKE_CP =
CPE_WORKSPACE =
CC = $(MAKE_CP) -ctx $(CPE_WORKSPACE) -cc $(CROSS_COMPILE)gcc
```

Makefile.smt 編集後(例)

```
CROSS_COMPILE =arm-none-linux-gnueabi-
SMT_LIBDIR=$(API_LIBRARY)/SoftwareModel/library
SMT_LIB =libsmt.a
MAKE_CP =/home/<username>/make_cp/make_cp
CPE_WORKSPACE =/home/<username>/WorkSpace/myWorkSpace/myWorkSpace.ctx
CC = $(MAKE_CP) -ctx $(CPE_WORKSPACE) -cc $(CROSS_COMPILE)gcc
```

14.3.4. makeの実行

① 一度 clean をしてください。

[HOST]\$ make clean

rm -f helloworld helloworld.o SMTUser.o

② Makefile.smt を指定して make をしてください。

[HOST]\$ make -f Makefile.smt

③ 新しいオブジェクトが生成されたことを確認

make の実行結果にエラーがないことを確認し、helloworld が生成されたことを確認してください。

[HOST]\$ ls helloworld

helloworld

14.4. 出力確認方法

14.4.1. オブジェクトの保存

helloworld を、ユーザーシステムのルートファイルシステムに組み込んでください。 ここでは、/usr/local/binに組み込んだと仮定します。

14.4.2. Linuxシステムの起動

ご利用の開発環境手順に従い、ターゲット Linux システムをブートします。 シリアルコンソール等で起動を確認し、Linux システムにログインしてください。

14.4.3. helloworldの実行

SMT ログファイル出力デーモンを起動してトレースを開始してください。

あらかじめルートファイルシステムに組み込んだ helloworld を実行してください。

[TARGET]# /usr/local/bin/helloworld

14.5. macroTRACE-VIEWERの設定

デバッグ情報を記録する際に、macroTRACE-VIEWER で必要な設定について説明します。

(1) 「ツール」→「パラメータ設定」または、 と を選択し、「パラメータ設定」ダイアログを表示します。



- (2) 左のツリーから「基本」を選択します。
- (3) 「チェックポイントワークスペースファイル」に、「13.2 Cpe ワークスペース新規作成」で作成 した「W:¥Workspace¥myWorkSpace¥myWorkSpace.ctx」を指定します。
- (4) 「OK」をクリックして、変更を保存します。

	ーチェックポイントワークスペー	スファイル		
解析 CPU使用率・ポートモニタ 日・表示 チャート Fャート Real-Time Viewer 日・ プロファイル プロセスプロファイル プロセスプロファイル スレッドプロファイル 関数プロファイル	File myWorkspace.ctx	Checkpoint 1:4	Path W:¥myWo	道加 編集 削除 全削除
□-ツール共通 カラー コマンド 終了 ブックマーク	メッセージタグファイル タイムスタンプ分解能			
4	20 nsec エンディアン リトルエンディアン	~		
× 00 ¥	$\boldsymbol{<}$	ок	キャンセル	適用

デバッグ情報を記録するための設定が完了しました。

14.6. システムマクロトレース取得例

取得例は下記のようになります。

関数チャートが表示されない場合、

下図の緑枠に示す"関数チャート/スレッドチャートの切り替えボタン"でチャート表示を切り替えて 関数チャートが表示されるかどうか確認してください。

ファイル	DTRACE-\	VIEWER 長示(V)	【閲覧モード】 【smt.tlog】 検索(S) 記録(R) ツール	(T) プロファイル(P)		<i>1</i> (Н)					×
								- 4 4			
				ਦ : 2 ∉ : 7 - ₹	•	:# =	: 🛪 :	3 (* *	2	_	_
チヤ	7-h	_								Ф	× ↓
۹.	19	E –	F - 10 fr (18.703m826u900r	n 緑	18.7	03m826i	u900n 🗦	差分	On	7
)n 704m610u000n	704m620u000n 704m6	630u000)n 704m640u0	00n	704m650	u000n 704m660u000n	704m670u00	
Core	e0 Thread	d>		TID=00000)224(he	lloworld)/PID=000	00224(h	elloworld)	1		
Core	e1 Thread	 d>		+		<unknown></unknown>					
				1					· · · · · · · · · · · · · · · · · · ·		
KUnkr	(nown>										Â.
hello	from tok	νοΩ									
get_n hello_	name() _from_new	wyork()									=
get_n hello_	name() _from_new	vyork()									E
eet_n hello	_rrom_new	vyork()									H F
riello get_n hello	inom_ddx hame() _from_new 	vyork()	F <								T X
rieno get_n hello ; ログ Mark	name() _from_new 	vyork() Sub	► <	DifferenceTime	Core	Thread	Attr	Level	Message	ې بې Information	T X
reno eet n hello ; ログ Mark	name() from_new パリスト Index -10	vyork() Sub		DifferenceTime	Core	Thread (DLE>	Attr	Level	Message	ا ب Information	T X
rieno eet_n hello ; ログ Mark	Imame() from_new 7UZF Index -10 -10	sub 1584 1616	Time 00:00:00 202m514u300n 00:00:00 204m192u780n	Difference Time 1 m678u480n	Core 1	Thread <idle> TID=0000208</idle>	Attr IDLE SW	Level P/T	Message TID=00000208(mmcqd/	ب ب Information	× ×
reno eet_n hello_ ii □⊅ Mark	IIII Index -10 -10 -10	Sub 1584 1616 1728	Time 00:00:00.202m514u300n 00:00:00.204m192/380n 00:00:00.204m23u380n	DifferenceTime 1 m678u480n 30u520n	Core 1 0	Thread (IDLE> TID=00000208 TID=000001 E	Attr IDLE SW SW	Level P/T P/T	Message TID=00000208(mmcqd/ TID=000001E5(kworker	₽ ₽ Information	×
neno eet,n hello i ログ Mark	ImameQ _from_new #UX h Index -10 -10 -10 -10	Sub 1584 1616 1728 1872	Time 00:00:00 202m514u300n 00:00:00 204m23u300n 00:00:00 204m23u300n 00:00:00 204m253u820n	Difference Time 1 m678u480n 30u520n 30u520n	Core 1 0 0	Thread (IDLE> TID=00000208 TID=000001 E TID=00000208	Attr IDLE SW SW SW	Level P/T P/T P/T	Message TID=00000208(mmcqd/ TID=000011E5(kworker TID=00000208(mmcqd/	ے ب ب Information	E V X
reno eet.n hello ii ログ Mark	Imame() _from_new ////////////////////////////////////	Sub 1584 1616 1728 1872 1984	Time 00:00:00 202m514u300n 00:00:00 204m192u780n 00:00:00 204m223u300n 00:00:00 204m253u820n 00:00:00 204m253u820n	Difference Time 1 m678u480n 30u520n 30u520n 0n	Core 1 0 0 0	Thread <idle> TID=000001E TID=0000028 TID=0000028 TID=0000021</idle>	Attr IDLE SW SW SW	Level P/T P/T P/T P/T	Message TID=00000208(mmcqd/ TID=000001E5(kworker TID=00000218(mmcqd/ TID=0000021D(smtex)/	L ► 4	E X
reno eet, hello iii ログ Mark	Image: 0 _from_new ////////////////////////////////////	sub Sub 1584 1616 1728 1872 1984 2112	Time 00:00:00:202m514u300n 00:00:00:204m192u780n 00:00:00:204m23u820n 00:00:00:204m253u820n 00:00:00:204m253u820n	Difference Time 1 m678u480n 30u520n 30u520n 0n 0n	Core 1 0 0 0 0	Thread <idle> TID=0000208 TID=0000028 TID=0000021 TID=00000208 TID=00000208</idle>	Attr IDLE SW SW SW SW SW	Level P/T P/T P/T P/T	Message TID=00000208(mmcqd/ TID=000001E5(kworker TID=00000208(mmcqd/ TID=00000210(smtcqd/ TID=00000208(mmcqd/	↓ Ţ Information	E X
Merilo eet,∩ hello iii □ ⊅ Mark	Imame() from_new #UXh Index -10 -10 -10 -10 -10	sub 1584 1616 1728 1872 1984 2112 2224	► ★ Time 00:00:00 202m514u300n 00:00:00 204m192u780n 00:00:00 204m23u820n 00:00:00 204m253u820n 00:00:00 204m553u820n 00:00:00 204m559u000n 00:00:00 204m559u000n 00:00:00 204m559u000n	DifferenceTime 1 m678u480n 30u520n 30u520n 0n 3051180n 30520n	Core 1 0 0 0 0 0	Thread <idle> TID=0000208 TID=0000021 TID=0000021 TID=00000218 (IDLE> TID=00000218 (IDLE></idle>	Attr IDLE SW SW SW SW SW SW IDLE	Level P/T P/T P/T P/T	Message TID=00000208(mmcqd/ TID=000001E5(kworker TID=00000208(mmcqd/ TID=00000208(mmcqd/ TID=00000208(mmcqd/	a P	E X
Merilo eetin hello Mark	Imame() from_new JUZ h Index -10 -10 -10 -10 -10 -10 -10	Sub 1584 1616 1728 1872 1984 2112 2224 2256	Image: Control of the second secon	DifferenceTime 1 m678u480n 30u520n 30u520n 0n 305u180n 305u180n 30u520n 1 m709u000n	Core 1 0 0 0 0 0 0 0 0 0 0	Thread (IDLE> TID=00000208 TID=00000208 TID=00000208 TID=00000208 (IDLE> TID=00000208	Attr IDLE SW SW SW SW SW IDLE SW	Level P/T P/T P/T P/T P/T P/T	Message TID=00000208(mmcqd/ TID=00000208(mmcqd/ TID=00000208(mmcqd/ TID=00000208(mmcqd/ TID=00000208(mmcqd/	- + - - Information	T X 4
15. 効果的なご利用方法

本章では API ライブラリを効果的に使用するための方法に記載しています。 API ライブラリの詳細については『システムマクロトレース API ライブラリユーザーマニュアル』をご 覧ください。

15.1. システムコール情報の取得

```
「sys_」ではじまるシステムコールディスパッチ関数の入口/出口に埋め込む例です。
カーネルソース編集後ビルドを実施してください。
編集前のサンプルソースファイルを
$(API_LIBRARY)¥SoftwareModel¥sample¥kernel¥exit.org
編集後のサンプルソースファイルを
$(API_LIBRARY)¥SoftwareModel¥sample¥kernel¥exit.c
にあります。
$(KERNEL)/kernel/exit.c
※カーネルバージョンによって編集ソースが異なる場合があります。
```

通常、システムコール番号の定義はカーネルツリーの include/asm/unistd.hファイル内に記載 されています。

```
/*
```

*/

* Linux o32 style syscalls are in the range from 4000 to 4999.

#defineNR_Linux	4000
#defineNR_syscall	(NR_Linux + 0)
#defineNR_exit	$(_NR_Linux + 1)$
#defineNR_fork	(NRLinux + 2)
#defineNR_read	$(_NR_Linux + 3)$
#defineNR_write	$(_NR_Linux + 4)$
#defineNR_open	(NR_Linux + 5)
#defineNR_close	$(_NR_Linux + 6)$
#defineNR_waitpid	(NRLinux + 7)
#defineNR_creat	(NR_Linux + 8)

15.2. 割込み情報の取得

割込み発生時に必ず実行される共通関数に埋め込む例です。場合によってはヘッダファイルになること があります。

カーネルソース編集後ビルドを実施してください。

編集前のサンプルソースファイルを

\$(API_LIBRARY)¥SoftwareModel¥sample¥kernel¥irq.org

編集後のサンプルソースファイルを

\$(API_LIBRARY) ¥SoftwareModel ¥sample ¥kernel ¥irq.c

にあります。

※カーネルバージョンによって編集ソースが異なる事や、ヘッダファイルになる場合があります。

#include <smt/SMTAPI.h>

asmlinkage void __exception asm_do_IRQ(unsigned int irq, struct pt_regs *regs)

{

. . .

struct pt_regs *old_regs = set_irq_regs(regs); struct irq_desc *desc = irq_desc + irq;

/*

* Some hardware gives randomly wrong interrupts. Rather * than crashing, do something sensible. */

```
if (irq>=NR_IRQS)
```

desc = &bad_irq_desc;

_SMT_OsSwitch_Irq_in((unsigned long)irq);

irq_enter(); #ifdef CONFIG_CODETEST ct_isr_enter(irq); #endif/* CONFIG_CODETEST */

desc_handle_irq(irq, desc);

/* AT91 specific workaround */ irq_finish(irq);

irq_exit();

_SMT_OsSwitch_Irq_out((unsigned long)irq);

#ifdef CONFIG_CODETEST

ct_isr_exit(irq); #endif/* CONFIG_CODETEST */ set_irq_regs(old_regs);

15.3. printk情報の取得

printk で出力時に必ず実行される共通関数に埋め込む例です。 カーネルソース編集後ビルドを実施してください。 編集前のサンプルソースファイルを \$(API_LIBRARY)¥SoftwareModel¥sample¥kernel¥printk.org 編集後のサンプルソースファイルを \$(API_LIBRARY)¥SoftwareModel¥sample¥kernel¥printk.c

\$(KERNEL)/kernel/printk.c ※カーネルバージョンによって編集ソースが異なる場合があります。

#include <smt/SMTAPI.h> #define YDC_SMT_NAME_LENGTH 124 char printk_output[YDC_SMT_NAME_LENGTH] = ""; (中略)

}

static void __call_console_drivers(unsigned start, unsigned end)

```
struct console *con;
```

```
for (con = console_drivers; con; con = con->next) {
    if ((con->flags & CON_ENABLED) && con->write &&
        (cpu_online(smp_processor_id()) ||
```

(con->flags & CON_ANYTIME)))

con->write(con, &LOG_BUF(start), end - start);

```
if( (end - start) < YDC_SMT_NAME_LENGTH){
    strncpy(printk_output,&LOG_BUF(start), end - start);
    printk_output[end - start] = '¥0';
} else {
    strncpy(printk_output,&LOG_BUF(start), YDC_SMT_NAME_LENGTH);
    printk_output[YDC_SMT_NAME_LENGTH-1] = '¥0';
    _SMT_Printf(0, ''smt over buffer. over len=%d'', end - start - YDC_SMT_NAME_LENGTH);
}
_SMT_Puts(0, printk_output);</pre>
```

16. 付録

16.1.実施項目の確認事項

実施項目の確認をまとめて記載します。

16.1.1. カーネルランドからの関数トレース出力関連

番号	確認項目	チェック
1	Cpe でパス変換設定は正しく実施しましたか?	
	「13.2.6 パス変換設定」	
2	Linux ホストでライブラリサーチパス設定はしましたか	
	「14.2 ライブラリサーチパス設定の実施」	
3	Makefile を編集して make_cp 経由でビルドするようにしましたか?	
	「13.4 MakeFile の編集」	
4	macroTRACE-VIEWER で作成したワークスペースファイルの設定をしましたか?	
	「13.6 macroTRACE-VIEWER の設定」	

16.1.2. ユーザーランドからの関数トレース出力関連

番号	確認項目	チェック
1	ユーザーランド API ライブラリのビルドは実施しましたか?	
	「8 ユーザーランド API ライブラリのビルド」	
2	Linux ホストでライブラリサーチパス設定はしましたか	
	「14.2 ライブラリサーチパス設定の実施」	
3	トレース対象の Makefile を編集してユーザーランド API ライブラリをリンクするよ	
	うにしましたか?	
	「14.1.2 Makefile を編集」	
4	トレース対象の Makefile を編集してユーザーランド API ライブラリヘッダファイル	
	のサーチパスを設定しましたか?	
	「14.1.2 Makefile を編集」	
5	Makefile を編集して make_cp 経由でビルドするようにしましたか?	
	「14.3.3 Makefile.smt を編集」	
6	macroTRACE-VIEWER で作成したワークスペースファイルの設定をしましたか?	
	「14.5 macroTRACE-VIEWER の設定」	