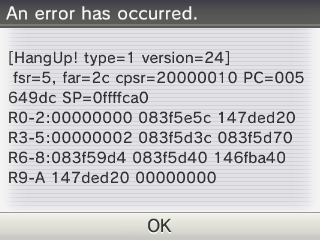
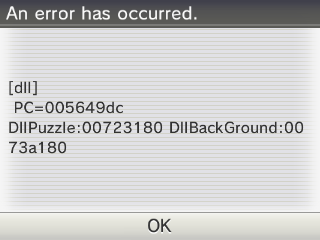
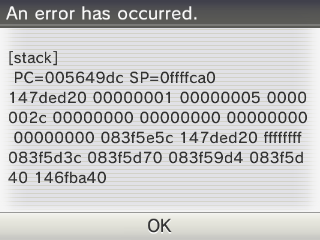
「ハング時PC位置表示アプレット」ドキュメント

2013/07/04

森昭人

**●概要**

このドキュメントはＫｕｊｉｒａプログラムに組み込んだハングアップ時アプレット表示機能について記述したものです。

**●組み込み箇所**

この「ハングアップ時アプレット表示機能」が組込まれているソースコードは「prog/src/system/machine\_use.cpp」内の

KujiraExceptionHandler関数になります。

**●説明**

3DSプログラムは開発実機上でハングアップした際には「プログラム例外ハンドラ」という機能あります。

デバッガーでデバッグ情報込(dbmk,rmk)で実行している間はいいのですが、rommkを開発実機で実行している時にハングアップすると

まったくデバッグ情報がでなくなるため推測材料がありません。

そこで止まった瞬間のPCカウンタやレジスタ・スタック・DLL割り付け情報が割り込みで取得する事ができるというプログラム例外ハンドラの

話を聞き、エラーアプレットで表示するように処理を追加しました。

このアプレットで表示している「PCカウンタ」と「DLL割り付け情報」「スタックダンプ」を入力して、実際にどの関数の中で止まったのか

mapファイルを検索できるようにしたツールも用意しています。

**●アプレット表示の実際**

アプレットは３回に分けて表示されます。

|  |  |
| --- | --- |
| １ページ目 | 1ページ目はプログラム実行情報を主に表示します。  Typeはnn::os::ARM::ExceptionTypeというアボート発生要因の値を示しています（下に資料として添付）  VersionはkujiraのＲＯＭバージョンをしめしています。24がＸ、25がＹです。(pm\_version.h参照)  主に見る数値としてはＰＣがプログラム実行位置。  SPがスタックポインタの値を示しています。  スタックは0x10000000をスタート位置として、積みあがるたびに数値が減っていく動作になっています。  以後、３つずつR10までレジスタ数値を表示しています。 |

|  |  |
| --- | --- |
| ２ページ目 | ２ページ目はＤＬＬ情報を表示しています。  プログラムハングアップ時に読み込まれていたＤＬＬ情報を表示します。  １ページ目でも表示していたＰＣ位置を再度表示。  以降ＤＬＬ名と割り付けアドレスを列記していきます。 |

|  |  |
| --- | --- |
| ３ページ目 | ３ページ目はスタックを表示します。  最初にＰＣ位置の表示。  以降スタックポインタから６４バイト分の値を４バイトずつプリントします。  ３ＤＳのコード領域は「0x00100000」から配置されるため、８番目の数値「00819d28」などはハングアップした関数の呼び出し元の可能性が高いです。 |

Kujiraプロジェクトの静的領域プログラムサイズはだいたい0x0072000で終了しているため、

DllFieldが0x00721600あたりから割り付けられている事が多いようです。

●注意

この「例外ハンドラ機能」はデバッガーでは起きません。開発実機でのみ表示されます。

Kujiraのコードでは開発実機に焼いたrmk、rommkでのみ発生します。

PM\_DEBUGで括っていますので、fmkでは表示されません。

どうしてもfmkで再現しない場合はmachine\_use.cppを改造する事でfmkで表示する事はできるようになるはずです。

以下、CTR-SDKの関数マニュアルのコピーです。

nn::os::ARMに実際のページも存在しています。

[**nn**](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn\Overview.html)**::**[**os**](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn\os\Overview.html)**::**[**ARM**](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn\os\ARM\Overview.html)**::SetUserExceptionHandler 関数**

**構文**

void SetUserExceptionHandler(

[UserExceptionHandler](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn\os\ARM\UserExceptionHandler.html) pHandler,

[uptr](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn_types\uptr.html) stackBottom,

[ExceptionBuffer](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn\os\ARM\ExceptionBuffer\Overview.html) \* pExceptionBuffer

);

**引数**

|  | **名前** | **説明** |
| --- | --- | --- |
| in | **pHandler** | 例外発生時に呼び出される例外ハンドラを指定します。 |
| in | **stackBottom** | 例外ハンドラで使用するスタックを指定します。 スタックの底を指定する必要があります。 |
| in | **pExceptionBuffer** | 例外情報を格納するバッファを指定します。 |

**返り値**

なし。

**説明**

グローバル例外ハンドラを設定します。

**引数**

stackBottom に [HANDLER\_STACK\_USE\_THREAD\_STACK](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn\os\ARM\HANDLER_STACK_USE_THREAD_STACK.html) を指定すると 例外が発生したスレッドのスタックを、 そのまま例外ハンドラのスタックとして使用します。

pHandler に渡される ExceptionInfo と ExceptionContext は pExceptionBuffer で指定されるバッファ上に確保されます。 pExceptionBuffer に [EXCEPTION\_BUFFER\_USE\_HANDLER\_STACK](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn\os\ARM\Overview.html) を指定すると、これらを例外ハンドラのスタック上に確保します。 pExceptionBuffer に [EXCEPTION\_BUFFER\_USE\_THREAD\_STACK](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn\os\ARM\Overview.html) を指定すると、これらを例外が発生したスレッドのスタック上に確保します。

**例外ハンドラの設定**

例外ハンドラの設定はスレッドごとに存在し、 例外が発生した場合には例外が発生したスレッドの設定を用いて 例外ハンドラが呼び出されます。 ただし、例外が発生したスレッドに対して例外ハンドラが設定されていない場合は メインスレッドの設定を用いて例外ハンドラが呼び出されます。

グローバル例外ハンドラとはメインスレッドの例外ハンドラを指します。 この関数はメインスレッドの例外ハンドラを設定します。

例外ハンドラが設定されている場合に例外が発生すると 例外が発生したスレッドのコンテキストで pHandler が呼び出されます。 ただしスタックは stackBottom で指定されるスタックに 挿しかえられています。

例外ハンドラが呼び出される時に、 割り込み禁止などの特別な処理が行われないことに注意してください。 そのため例外ハンドラのスタックや例外情報を格納するバッファとして 複数のスレッドで同じメモリ領域を設定している場合、 それらのスレッドのいずれかで例外ハンドラを処理中に メモリ領域を共有する他のスレッドで例外が発生した場合、 例外ハンドラのスタックや例外情報のバッファが破壊されることになります。

**例外ハンドラの実装**

例外ハンドラには以下の制約があります。

* 例外ハンドラから return することはできません。 return した場合の動作は不定です。
* 例外ハンドラは ARM 命令でコンパイルされている必要があります。 THUMB 命令の例外ハンドラは使用できません。 例外ハンドラから THUMB 命令の関数を呼び出すことはできます。

デバッガ環境では例外ハンドラを呼び出すことは出来ませんので ご注意ください。

# [nn](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn\Overview.html)::[os](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn\os\Overview.html)::[ARM](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn\os\ARM\Overview.html)::ExceptionInfo 構造体

## 構文

struct ExceptionInfo

{

[util::SizedEnum1](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn\util\SizedEnum1\Overview.html)< [ExceptionType](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn\os\ARM\ExceptionType.html) > type;

[bit32](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn_types\bit32.html) fsr;

[bit32](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn_types\bit32.html) far;

[bit32](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn_types\bit32.html) fpexc;

[bit32](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn_types\bit32.html) fpinst;

[bit32](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn_types\bit32.html) fpinst2;

};

## 説明

発生した例外と関連する情報を保持する構造体です。

## メンバ変数

|  |  |  |  |
| --- | --- | --- | --- |
|  | **type** | [util::SizedEnum1](file:///C:\\tools\\ctr_mancollection20120328-DL\\Packages\\SDK\\CTRSDK\\CTR_SDK\\documents\\api\\nn\\util\\SizedEnum1\\Overview.html)<[ExceptionType](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn\os\ARM\ExceptionType.html) > | 発生した例外の種類が格納されます。 |
|  | **fsr** | [bit32](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn_types\bit32.html) | プリフェッチアボートのときには IFSR の値が、データアボートのときには DFSR の値が格納されます。 |
|  | **far** | [bit32](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn_types\bit32.html) | プリフェッチアボートのときには [ExceptionContext::r](file:///C:\\tools\\ctr_mancollection20120328-DL\\Packages\\SDK\\CTRSDK\\CTR_SDK\\documents\\api\\nn\\os\\ARM\\ExceptionContext\\Overview.html)[15] と同じ値が、データアボートのときには DFAR の値が格納されます。 |
|  | **fpexc** | [bit32](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn_types\bit32.html) | VFP 例外の時に FPEXC の値が格納されます。 |
|  | **fpinst** | [bit32](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn_types\bit32.html) | VFP 例外の時に FPINST の値が格納されます。 |
|  | **fpinst2** | [bit32](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn_types\bit32.html) | VFP 例外の時に FPINST2 の値が格納されます。 |

# [nn](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn\Overview.html)::[os](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn\os\Overview.html)::[ARM](file:///C:\tools\ctr_mancollection20120328-DL\Packages\SDK\CTRSDK\CTR_SDK\documents\api\nn\os\ARM\Overview.html)::ExceptionType 列挙型

## 構文

enum ExceptionType;

## 値

| **値** | **説明** |
| --- | --- |
| **EXCEPTION\_TYPE\_PABT** | プリフェッチアボート例外が発生したことを表します。 |
| **EXCEPTION\_TYPE\_DABT** | データアボート例外が発生したことを表します。 |
| **EXCEPTION\_TYPE\_UNDEF** | 未定義命令例外が発生したことを表します。 |
| **EXCEPTION\_TYPE\_VFP** | VFP 例外が発生したことを表します。 |
| **EXCEPTION\_TYPE\_MAX\_BIT** |  |

●プリフェッチアボート例外について

ARMはプログラム命令を実際に実行するために前もってコードを読み込んでおくという処理を

行いながらできるだけ高速動作させるようにしているが、その事前読み込みが失敗する事がプリフェッチアボートである。

コードの実行先がNULLポインタだった場合などにジャンプしようとして失敗した時に発生する。

●データアボート例外について

ARMはアドレス空間をどこが有効で、どこが無効かを定義する事ができる。

例えば32bitだと0x0～0xffffffffまでアドレス空間が存在するが、このすべてのアドレスから数値を参照できるわけではない。

実際3DSでは0x20000000よりも大きいのアドレスは無効空間になっている。そういう場所からデータを読み込もうとすると間違ったデータが取得されるわけでは無く「何もデータがとれない」状態となる。

このようにポインタでデータを参照しようとして、有効でないアドレスからデータを読み込もうとすると発生するのがデータアボートである。

●未定義命令例外について

これはプログラムでジャンプした先が配列データだったときなどに発生する。

つまり、命令を実行しようとしたがアセンブラとして実行できない命令を実行しようとしてしまった時に発生する。