FUJITSU

Fujitsu Software
Technical Computing Suite V4.0L20

Development Studio
並列実行デバッガ使用手引書

J2UL-2569-01Z0(02)
2023年3月

まえがき

本書の目的

本書は、富士通製CPU A64FXを搭載したシステム向け並列実行デバッガの機能および使用方法について説明します。

本書の読者

本書は、Fortran、C言語、またはC++で記述され、MPIライブラリを呼び出すプログラム（以降、MPIプログラムと呼びます）のデバッグを行う方を対象に記述しています。本書を読むにあたっては、以下の基本的な知識が必要です。

- MPI
- Fortranプログラム、Cプログラム、およびC++プログラム
- Linuxのコマンド、ファイル操作、およびシェルプログラミング
- GNUデバッガ(以降、GDBと呼びます)を使用したデバッグ

本書の構成

本書は、以下の構成になっています。

第1章 並列実行デバッガの概要

並列実行デバッガの構成および使用するための事前準備について説明します。

第2章 異常終了調査機能

異常終了調査機能について説明します。

第3章 デッドロック調査機能

デッドロック調査機能について説明します。

第4章 コマンドファイルによるデバッガ制御機能

コマンドファイルによるデバッガ制御機能について説明します。

付録A メッセージ一覧

並列実行デバッガが出力するメッセージを説明します。

付録B 調査結果ファイルの詳細

異常終了調査機能およびデッドロック調査機能の調査結果ファイルの詳細について説明します。

本書の位置付け

本書は、以下のマニュアルと関係があります。必要に応じて参照してください。

- “Fortran文法書”
- “Fortran使用手引書”
- “Fortran使用手引書 別冊 COARRAY”
- “Fortran翻訳時メッセージ”
- “C言語使用手引書”
- “C++言語使用手引書”
- “C/C++最適化メッセージ説明書”
- “Fortran/C/C++実行時メッセージ”
- “MPI使用手引書”

上記以外に、以下の関連ソフトウェアのマニュアルも必要に応じて参照してください。

- “ジョブ運用ソフトウェア”

構文表記記号

構文表記記号とは、構文を記述するうえで特別な意味で定められた記号であり、以下のものがあります。

記号名	記号	説明
選択記号	{ }	この記号で囲まれた項目の中から、どれか1つを選択することを表します。
	|	この記号を区切りとして、複数の項目を列挙することを表します。
省略可能記号	[ ]	この記号で囲まれた項目を省略してよいことを表します。また、この記号は選択記号"{ }"の意味を含みます。
反復記号	...	この記号の直前の項目を繰り返して指定できることを表します。

輸出管理規制について

本ドキュメントを輸出または第三者へ提供する場合は、お客様が居住する国および米国輸出管理関連法規等の規制をご確認のうえ、必要な手続きをおとりください。

商標

- Linux(R)は米国及びその他の国におけるLinus Torvaldsの登録商標です。
- そのほか、本マニュアルに記載されている会社名および製品名は、それぞれ各社の商標または登録商標です。
- 本資料に掲載されているシステム名、製品名などには、必ずしも商標表示(TM、(R))を付記しておりません。

出版年月および版数

版数	マニュアルコード
2023年3月 第1.2版	J2UL-2569-01Z0(02)
2020年3月 第1.1版	J2UL-2569-01Z0(01)
2020年2月 初版	J2UL-2569-01Z0(00)

著作権表示

変更履歴

変更内容	変更箇所	版数
項を追加。	1.5.3	第1.2版
“環境変数の指定”を追加。	1.2	第1.1版
“ソースコードの修正”を追加。	1.3	
動的に生成されたプロセスに関する説明を追加。	2.2 2.3.1 3.3 第4章	
説明文の見直し。	-	

目　次

第1章 並列実行デバッガの概要

この章では、並列実行デバッガの構成および使用するための事前準備について説明します。

1.1 並列実行デバッガの構成

本書では、以下を総称して「並列実行デバッガ」と呼びます。並列実行デバッガでは、各種調査に使用する情報の取得とコマンドファイルによるデバッガ制御にGDBを使用しています。

- 異常終了調査機能

  プログラムの異常終了の原因調査を支援します。プログラムの異常終了によりシグナルを受信したタイミングでバックトレースなどの実行情報を取得します。詳細については、“第2章 異常終了調査機能”を参照してください。

- デッドロック調査機能

  プログラムのデッドロック発生の有無およびデッドロック発生の原因調査を支援します。プログラムが終了しない場合や応答しない場合に、ジョブのすべてのプロセスに対してバックトレースなどの実行情報を取得します。詳細については、“第3章 デッドロック調査機能”を参照してください。

- 重複除去機能

  異常終了調査機能およびデッドロック調査機能の調査結果ファイルを対象に、可読性向上のためのデータ加工をします。本機能では専用のコマンド(fjdbg_summary)を使用します。詳細については、“2.3 重複除去機能”を参照してください。

- コマンドファイルによるデバッガ制御機能

  プロセスごとに異なるデバッガ制御を実施するためのデバッガ制御機能を提供します。本機能では、コマンドファイルと呼ばれるGDBのコマンドを記述したファイルを利用します。本機能を使用することで、プログラムの特性に応じた柔軟なデバッグが可能になります。詳細については、“第4章 コマンドファイルによるデバッガ制御機能”を参照してください。

異常終了調査機能、デッドロック調査機能、およびコマンドファイルによるデバッガ制御機能を使用する場合、mpiexecコマンドのオプションで指定します。異常終了調査機能とデッドロック調査機能は同時に使用できますが、コマンドファイルによるデバッガ制御機能はほかの調査機能と同時に使用できません。

1.2 環境変数の指定

並列実行デバッガ使用時に必要な環境変数を指定します。

表1.1 並列実行デバッガ使用時に必要な環境変数

環境変数名	設定値
PATH	/*製品インストールパス*/bin
LD_LIBRARY_PATH	/*製品インストールパス*/lib64

“*製品インストールパス*”については、システム管理者にお問い合わせください。

1.3 ソースコードの修正

MPI_COMM_SPAWNルーチンまたはMPI_COMM_SPAWN_MULTIPLEルーチンを使用して動的に生成したプロセスに対し、異常終了調査機能やデッドロック調査機能を使用する場合は、gdb_wrapperコマンドを使用します。したがって、gdb_wrapperコマンドを使用するためにソースコードを修正する必要があります。

gdb_wrapperコマンドには、mpiexecコマンドと同一の異常終了調査機能やデッドロック調査機能のオプションを指定します。

MPI_COMM_SPAWNルーチンまたはMPI_COMM_SPAWN_MULTIPLEルーチンについてはMPIの規格書や“MPI使用手引書”を、異常終了調査機能のオプションについては“2.1 実行時オプション”を、デッドロック調査機能のオプションについては“3.2 実行時オプション”を参照してください。

例

int MPI_Comm_spawn(const char *command, char *argv[], int maxprocs, MPI_Info info, int root, MPI_Comm comm, MPI_Comm *intercomm, int array_of_errcodes[]) を、gdb_wrapperコマンドを使用するソースコードに修正した例

修正した部分を赤文字で示します。

```
MPI_Init( &argc, &argv );
MPI_Info info;
MPI_Info_create(&info);
MPI_Info_set(info, "fjdbg_spawn_dir_name", "spawn_test"); // 注

char *command[2]={"gdb_wrapper",NULL};
char *arg[6]={"-fjdbg-sig","all","-fjdbg-out-dir","data","./spawn1.out",NULL};

/*省略*/

MPI_Comm_spawn( command[0], arg, maxprocs, info, root, comm, intercomm, array_of_errcodes );
```

注:infoキー“fjdbg_spawn_dir_name”については“MPI使用手引書”を参照してください。

1.4 翻訳時オプション

並列実行デバッガを使用する場合、MPIプログラムの翻訳時に-gオプションを指定することを推奨します。翻訳時に-gオプションを指定しない場合、引数変数、引数変数値、ローカル変数、およびローカル変数値の情報を取得できません。翻訳時オプションの詳細については、“MPI使用手引書”を参照してください。

1.5 注意事項

並列実行デバッガを使用する際の注意事項について説明します。

1.5.1 COARRAYプログラム使用時の注意事項

並列実行デバッガではCOARRAYプログラムの場合も像番号ではなくランク番号を使用します。ランク番号は0から、像番号は1から始まるため「ランク番号 = 像番号-1」です。

COARRAYプログラムの詳細については、“Fortran使用手引書 別冊 COARRAY”を参照してください。

1.5.2 エラーメッセージ [ERROR] fjdbg_sig 1010(603) execution failed.

並列実行デバッガの“異常終了調査機能”または“デッドロック調査機能”では、並列実行デバッガ自身のデッドロックを避けるために実行プログラムにアタッチするまでの制限時間を設けています。しかし、実行環境によっては、実行プログラムにアタッチするまでに制限時間を超過してしまう場合があります。本現象が発生した場合、エラーメッセージ“[ERROR] fjdbg_sig 1010(603) execution failed.”を出力します。このエラーメッセージが出力された場合、以下の環境変数を設定してください。

表1.2 環境変数

環境変数名	デフォルト値	説明
FJ_TOOL_TIME_LIMIT	60	実行プログラムにアタッチするまでの制限時間(秒)を指定します。制限時間を超過しても実行プログラムにアタッチしない場合、エラーメッセージを出力して処理を終了します。

1.5.3 翻訳コマンド

並列実行デバッガを使用する場合、富士通コンパイラの翻訳コマンドを使用してプログラムの翻訳およびリンクを行ってください。GNUコンパイラなどの他のコンパイラの翻訳コマンドを使用した場合、並列実行デバッガを使用することはできません。

第2章 異常終了調査機能

この章では、異常終了調査機能について説明します。

異常終了調査機能を使用すると、プログラムが異常終了によりシグナルを受信したタイミングでプログラムの実行情報(バックトレース、フレームごとのローカル変数値および引数変数値、シグナル発生時アドレスを含む逆アセンブリ出力、レジスタ内容、メモリマップ)を取得できます。

2.1 実行時オプション

異常終了調査機能で使用するオプションを以下に示します。本オプションはmpiexecコマンドの実行時オプションです。mpiexecコマンドの使用方法については、“MPI使用手引書”を参照してください。

異常終了調査機能のオプションには、先頭のハイフン(-)の数が1つのものと2つのものがありますが、どちらも同じ意味です。以降の説明では、ハイフン(-)が1つのオプションを使用します。

{ -fjdbg-sig *signal* | --fjdbg-sig *signal* }

異常終了調査機能を有効にします。*signal*には捕捉したいシグナルをill、abrt、fpe、segv、busの中から1つ以上またはallで指定します。*signal*を複数指定する場合、コンマ(,)で区切ってください。allとそのほかの*signal*を同時に指定した場合、allが有効になります。*signal*を省略することはできません。*signal*を省略した場合、エラーメッセージを出力しプログラムの実行を終了します。

ill

SIGILLシグナルを捕捉します。

abrt

SIGABRTシグナルを捕捉します。

fpe

SIGFPEシグナルを捕捉します。

segv

SIGSEGVシグナルを捕捉します。

bus

SIGBUSシグナルを捕捉します。

all

SIGILL、SIGABRT 、SIGFPE、SIGSEGV、およびSIGBUSシグナルを捕捉します。

本オプションは -fjdbg-out-dir オプションと同時に指定してください。本オプションのみを指定した場合、エラーメッセージを出力しプログラムの実行を終了します。

{ -fjdbg-out-dir *output-dir* | --fjdbg-out-dir *output-dir* }

調査結果ファイルを格納するディレクトリを指定します。*output-dir*には、格納ディレクトリ名を相対パスまたは絶対パスで指定します。指定したディレクトリが存在しない場合、本オプションに指定したディレクトリを新規に作成します。指定したディレクトリがすでに存在する場合、その直下に“signal”という名前のファイルが存在してはいけません。指定したディレクトリの直下に“signal”という名前のディレクトリが存在する場合、そのディレクトリは空ディレクトリでなければいけません。これらの条件を満たさない場合、プロセス0の標準エラー出力にエラーメッセージを出力し、プロセス0の実行を終了します。なお、ジョブが終了しない場合があります。その場合はジョブを手動で削除してください。

本オプションは -fjdbg-sig オプションと同時に指定してください。本オプションのみを指定した場合、エラーメッセージを出力しプログラムの実行を終了します。

例

mpiexecコマンドへのオプション指定例

```
mpiexec -fjdbg-sig all -fjdbg-out-dir "/fefs/log" -n 4 ./a.out
```

2.2 調査結果ファイル

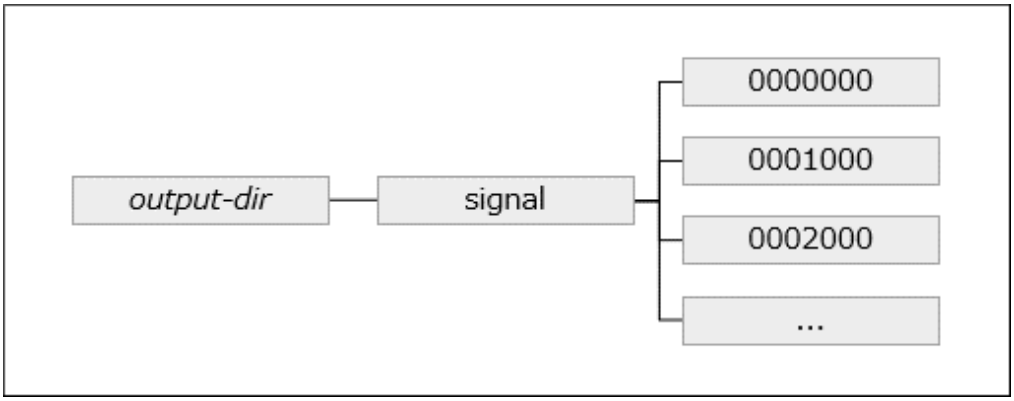
異常終了調査機能が出力する調査結果ファイルについて説明します。

異常終了調査機能は、プログラムの実行中に調査結果ファイルの格納用ディレクトリを生成します。格納用ディレクトリの構成は以下のとおりです。

- *output-dir*直下に“signal”という名前のディレクトリ(以降、“signal”ディレクトリと呼びます)を作成します。プログラムが正常終了した場合は“signal”ディレクトリのみ作成し以降の出力を行いません。
- “signal”ディレクトリ内に調査結果ファイルを出力します。
- “signal”ディレクトリ直下に1000ランク単位でディレクトリを作成し、ランクごとに調査結果ファイルを出力します。調査結果ファイル名は「ジョブID.ランク番号」の規則に従います。

ディレクトリ構成を下図に示します。

図2.1 ディレクトリ構成



例

ジョブIDが99999、ランク番号が0～2のプログラムの実行結果

```
output-dir/signal/0000000/99999.0
output-dir/signal/0000000/99999.1
output-dir/signal/0000000/99999.2
```

注意

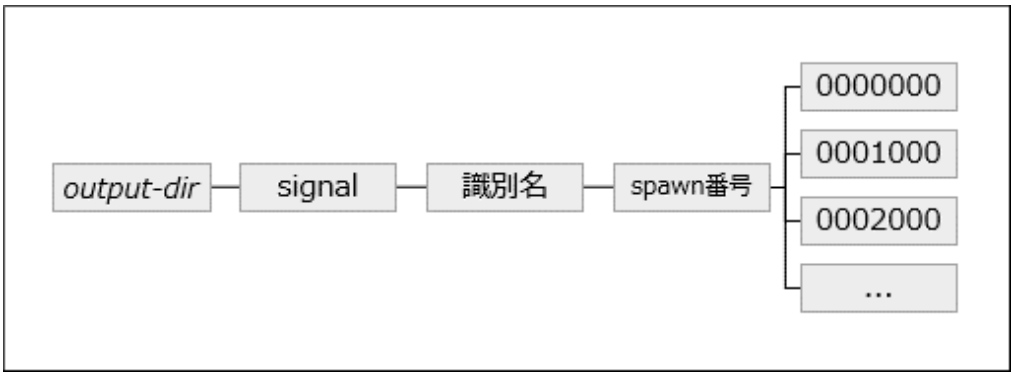
MPI_COMM_SPAWNルーチンまたはMPI_COMM_SPAWN_MULTIPLEルーチンを使用して動的に生成されたプロセスの場合、通常のプロセスとは以下の点が異なります。MPI_COMM_SPAWNルーチンまたはMPI_COMM_SPAWN_MULTIPLEルーチンを使用する場合の詳細については、“1.3 ソースコードの修正”を参照してください。

- “signal”ディレクトリ直下に、プログラム内でinfoキー“fjdbg_spawn_dir_name”に定義した値を名前としたディレクトリを作成します。infoキー“fjdbg_spawn_dir_name”が定義されていない場合、“spawn”という名前のディレクトリを作成します。本ディレクトリを以降“識別名”ディレクトリと呼びます。
- “識別名”ディレクトリ直下にspawn番号ごとにディレクトリを作成します。

- spawn番号ごとのディレクトリ直下に1000ランク単位でディレクトリを作成し、ランクごとに調査結果ファイルを出力します。調査結果ファイル名は“*ジョブID.ランク番号@spawn番号*”の規則に従います。

  ディレクトリ構成を下図に示します。

図2.2 ディレクトリ構成



例

ジョブIDが99999、識別名なし、spawn番号1、ランク番号が0～2のプログラムの実行結果

```
output-dir/signal/spawn/1/0000000/99999.0@1
output-dir/signal/spawn/1/0000000/99999.1@1
output-dir/signal/spawn/1/0000000/99999.2@1
```

調査結果ファイルはそのまま参照できますが、後述の重複除去機能を使用することを前提としています。重複除去機能を使用する場合、生成されたディレクトリ名、ディレクトリ構成、およびファイル名を変更しないでください。調査結果ファイルを直接参照したい場合、“付録B 調査結果ファイルの詳細”を参照してください。

2.3 重複除去機能

重複除去機能は、異常終了調査機能およびデッドロック調査機能の調査結果ファイルに対して、以下の処理をします。

- 重複するバックトレースの除去
- プログラムの実行情報を整形して表示
  - バックトレース
  - フレームごとのローカル変数値および引数変数値
  - シグナルを検出した箇所の前後の逆アセンブリ出力
  - レジスタ内容
  - メモリマップ

以降、重複除去機能が行う処理を重複除去処理と呼びます。

デッドロック調査機能の詳細については、“第3章 デッドロック調査機能”を参照してください。

2.3.1 使用方法

重複除去機能は、fjdbg_summaryコマンドを使用します。fjdbg_summaryコマンドの使用方法について以下に示します。

fjdbg_summaryコマンドの形式

```
fjdbg_summary [ -h | -v ] [ -n ] [ -a ] [ -b ] [ -r rankspec ] [ -p outrank ] input-dir
```

実行時オプションを指定しない場合、全ランクを対象に「関数名をキーとしたバックトレースの重複部分を除去」と「各種情報の整形処理」をします。

実行時オプション

-h

コマンドの使用方法を表示して終了します。本オプションを指定した場合、そのほかのオプションを無視します。

-v

バージョン情報を表示して終了します。本オプションを指定した場合、-h オプション以外のオプションを無視します。-hオプションと本オプションを同時に指定した場合、-hオプションのみ有効となりバージョン情報は表示されません。

-n

本オプションを指定した場合、バックトレースの重複除去を実施せず、各種情報の整形処理だけ実施します。-n オプションと -a オプションを同時に指定することはできません。同時に指定した場合、 -n オプションが有効になります。

-a

本オプションを指定した場合、関数名に加えてバックトレースのアドレス部をキーとしたバックトレースの重複除去を実施します。-a オプションと -n オプションを同時に指定することはできません。同時に指定した場合、-n オプションが有効になります。

-b

バックトレースを出力する条件を変更します。本オプションを指定しない場合、バックトレースの関数名がオブジェクトファイル内のシンボルから見つからなくなったタイミングでバックトレースの出力を打ち切ります。本オプションを指定した場合、オブジェクトファイル内のシンボル有無にかかわらず最後までバックトレースを出力します。

-r *rankspec*

重複除去結果として出力するランク番号を指定します。本オプションを指定しない場合、全ランクを対象にします。*rankspec* で指定したランク番号の情報のみを出力します。*rankspec* を省略することはできません。*rankspec* を省略した場合、エラーメッセージを出力しプログラムの実行を終了します。実行結果が存在しないランク番号が *rankspec* に含まれていた場合、そのランク番号は無視します。

rankspec に1つだけ指定する場合

ランク番号を指定します。(例: -r 1)

rankspec に範囲指定する場合

開始番号と終了番号をハイフン(-)で繋ぎます。(例: -r 1-10)

rankspec に複数指定する場合

ランク番号または範囲指定をコンマ(,)で区切ります。(例: -r 1,4-6,8)

-p *outrank*

フレームごとのローカル変数値および引数変数値を出力するランク数を指定します。調査結果ファイルの最小ランクを先頭に *outrank* 数分のフレームごとの引数変数およびローカル変数を出力します。本オプションを指定しない場合、全ランクを出力対象とします。*outrank* を省略することはできません。*outrank* を省略した場合、エラーメッセージを出力しプログラムの実行を終了します。実行結果が存在しないランク番号はランク数としてカウントしません。*outrank* がランクの総数を超える場合、重複除去対象の全ランクを対象とします。-rオプションが有効な場合、その範囲を対象とします。

input-dir

重複除去処理対象のディレクトリを相対パスまたは絶対パスで指定します。異常終了調査機能またはデッドロック調査機能で作成した調査結果ディレクトリ内の、“signal”ディレクトリまたは“deadlock”ディレクトリを指定します。ただし、重複除去対象が動的に生成されたプロセスの場合、調査結果ディレクトリ内の“signal/*識別名*/*spawn番号*”ディレクトリまたは“deadlock/*識別名*/*spawn番号*”ディレクトリを指定します。*input-dir* に指定したディレクトリが存在しない、または“signal”ディレクトリ、“deadlock”ディレクトリ、および“*spawn番号*”ディレクトリ以外を指定した場合、エラーメッセージを出力しプログラムの実行を終了します。

例

コマンド実行例

```
fjdbg_summary -a -r 1-10 ./dbg_result/signal
```

2.3.2 出力内容

コマンド実行時、標準出力に重複除去結果を出力します。以下に、出力内容について説明します。

重複除去機能を使用した場合、入力情報を以下の順番で出力します。

1. バックトレース、フレームごとのローカル変数値および引数変数値
2. シグナルを検出した箇所の前後の逆アセンブリ出力
3. レジスタ内容
4. メモリマップ

スレッドごとに1.～3.の情報を出力し、最後に4.を出力します。2.～4.の情報は重複除去機能の対象とならず、常に1ランクごとに情報を出力します。なお、使用する調査機能や問題の発生状況などによっては一部項目を出力しません。各情報の出力例を例1から例4に示します。

例1: バックトレース、フレームごとのローカル変数値および引数変数値 出力例

```
Thread : 1     Signal: SIGABRT [1,3]
               Signal: SIGSEGV [2]
 ------------------------------
   [ 1-3 ] <3 processes>
 ------------------------------
 0:    0x0000000000401070 in main () at /tmp/sample.c:47
       params:
         int argc = 1 [1]
         char ** argv = 0xfffffffffec88 [1]
         more than 1 distinct values
       locals:
         int rank = 1 [1]
(中略)
         int res = 0 [1]
         more than 1 distinct values
 1:        0x00000000004010f0 in sub () at /tmp/lib.c:10
           params:
             int a = 1 [1]
             more than 1 distinct values
           locals:
             int res = 0 [1]
             int b = 100 [1]
             more than 1 distinct values
 2:            0x00000000004011c0 in func () at /tmp/lib.c:24
               params:
                 int b = 100 [1]
                 int a = 1 [1]
                 more than 1 distinct values
               locals:
                 int res = 100 [1]
                 more than 1 distinct values
             ------------------------------
               [ 1,3 ] <2 processes>
             ------------------------------
 3:                0x0000400001125940 in abort () from=/lib64/libc.so.6
 4:=>*                 0x00004000011242c8 in raise () from=/lib64/libc.so.6
 5:                        0x000040000005066c in <signal handler called> ()
 6:                      0x0000400000063554 in intergdbHandler () from=/opt/FJSVxtclanga/tcsds-1.1.1/lib64/libsigdbg.so.1
 7:                              0x0000400000fd039c in wait () from=/lib64/libpthread.so.0
(後略)
```

参考

- バックトレースに重複除去対象となったランクが存在する場合、例1の[1-3]のように先頭に表示します。ローカル変数(locals:)、引数変数(params:)のようにランクごとに異なる情報を持つ場合、末尾の[]内の数字がランク番号を示します。
- プログラムの翻訳時にデバッグ情報を生成している場合、行番号などソースファイルに対応した情報を出力します。デバッグ情報を生成していない場合、シンボル名に対応した情報を出力します。
- バックトレースの先頭フレームは、以下のいずれかの関数名です。
  - main
  - __lpg_main
  - _lwp_start
  - __jwe_PE
  - __mpc_PE
  - start_thread
- 発生したシグナル情報(異常終了調査機能時)や異常終了箇所は、「 =>* 」という記号で出力します。
- -p *outrank* オプションの指定によって非表示となったランク内に変数が存在する場合、“ more than *outrank* distinct values ”というメッセージを出力します。

例2:シグナルを検出した箇所の前後の逆アセンブリ出力 出力例

```
Disassemble:
     [ 1 ]
          0xffffffff01ba1c74 <+>:         nop
          0xffffffff01ba1c78 <+>:         nop
          0xffffffff01ba1c7c <+>:         nop
          0xffffffff01ba1c80 <strlen+0>:     mov  %o0, %o1
          0xffffffff01ba1c84 <strlen+4>:     andn  %o0, 7, %o0
       =>* 0xffffffff01ba1c88 <strlen+8>:     ldx  [ %o0 ], %o5
          0xffffffff01ba1c8c <strlen+12>:     and  %o1, 7, %g1
          0xffffffff01ba1c90 <strlen+16>:     mov  -1, %g5
          0xffffffff01ba1c94 <strlen+20>:     sethi  %hi(0x1010000), %o2
          0xffffffff01ba1c98 <strlen+24>:     sll  %g1, 3, %g1
     [ 2 ]
          0xffffffff01ba1c74 <+>:         nop
          0xffffffff01ba1c78 <+>:         nop
          0xffffffff01ba1c7c <+>:         nop
          0xffffffff01ba1c80 <strlen+0>:     mov  %o0, %o1
          0xffffffff01ba1c84 <strlen+4>:     andn  %o0, 7, %o0
       =>* 0xffffffff01ba1c88 <strlen+8>:     ldx  [ %o0 ], %o5
          0xffffffff01ba1c8c <strlen+12>:     and  %o1, 7, %g1
          0xffffffff01ba1c90 <strlen+16>:     mov  -1, %g5
          0xffffffff01ba1c94 <strlen+20>:     sethi  %hi(0x1010000), %o2
          0xffffffff01ba1c98 <strlen+24>:     sll  %g1, 3, %g1
```

参考

「シグナル発生時アドレスを含む逆アセンブリ出力」は、異常終了調査機能使用時のみを出力します。

例3:レジスタ内容 出力例

```
Registers:
      [ 1 ]
        g0        0x3
        g1        0x1
        g2        0x1
        g3        0xfffffffffffffef8
        g4        0xfffff802e1bfd7c8
        g5        0xffffffff01b6a6e4
        g6        0x8101010101010100
        g7        0xfffff802e1bff890

(中略)

      [ 2 ]
        g0        0x5
        g1        0x8e
        g2        0x2
        g3        0x0
        g4        0x101975
        g5        0x2525252525252525
        g6        0x8101010101010100
        g7        0xfffff80200026610
(後略)
```

例4:メモリマップ 出力例

```
Mapped address spaces:
    [ 1 ]
          process 3472
                Start Addr             End Addr                 Size       Offset objfile
                  0x100000             0x102000               0x2000              0x0 /tmp/a.out
                 0x20000000           0x20400000              0x400000         0x100000 /tmp/a.out
                 0x20400000           0x20800000              0x400000              0x0 [heap]
                 0xff802000         0x7feffc00000         0x7fe003fe000              0x0 [stack]
         0xfffff80200000000    0xfffff80200002000               0x2000              0x0
         0xfffff80200002000    0xfffff80200004000               0x2000              0x0

(中略)

    [ 2 ]
          process 3473
                Start Addr             End Addr                 Size       Offset objfile
                  0x100000             0x102000               0x2000              0x0 /tmp/a.out
                 0x20000000           0x20400000              0x400000         0x100000 /tmp/a.out
                 0x20400000           0x20800000              0x400000              0x0 [heap]
                 0xff802000         0x7feffc00000         0x7fe003fe000              0x0 [stack]
(後略)
```

2.4 注意事項

異常終了調査機能を使用する際の注意事項について説明します。

2.4.1 シグナルハンドラの動作について

異常終了調査機能を使用する場合、SIGILL、SIGABRT、SIGFPE、SIGSEGV、およびSIGBUSシグナルを捕捉するため、翻訳時に動的リンクライブラリで設定されるシグナルハンドラを起動しません。ただし、利用者が上記のシグナルを捕捉するプログラムを作成している場合、利用者の設定したシグナルハンドラが有効になります。異常終了調査機能の実行時オプションで捕捉対象のシグナルを指定した場合、指定したシグナルを利用するほかの機能よりも異常終了調査機能が優先されます。

第3章 デッドロック調査機能

この章では、デッドロック調査機能について説明します。

デッドロック調査機能を使用すると、プログラムが終了しない場合や応答がない場合に、ジョブのすべてのプロセスに対してプログラムの実行情報(バックトレース、フレームごとのローカル変数値および引数変数値、メモリマップ)を取得できます。

3.1 使用方法

デッドロック調査機能の使用方法は以下のとおりです。

1. プログラム実行用のジョブスクリプトを作成します。必要な処理は以下のとおりです。
   a. trapコマンドを使用して、SIGHUP(ハングアップ)およびSIGXCPU(CPUタイムアウト)シグナル受信時の設定を変更します。
   b. mpiexecコマンドを実行します。デッドロック調査機能を有効にするため、後述の実行時オプションを指定する必要があります。

```
trap 'echo SIGHUP/SIGXCPU received.' HUP XCPU
mpiexec -fjdbg-dlock -fjdbg-out-dir "/fefs/log" -n 4 ./a.out
```

2. 1.のジョブスクリプトを投入します。
3. pjstatコマンドを実行して、2.のジョブIDを確認します。
4. デッドロックが疑われる状態で、以下のコマンドを投入します。

```
pjsig -s SIGHUP 2.のジョブID
```

注意

pjsigコマンド投入後、調査結果ファイルを格納してジョブが終了します。

参照

trapコマンドの使用方法については、オンラインマニュアルを参照してください。

pjstatコマンドおよびpjsigコマンドの使用方法については、“ジョブ運用ソフトウェア”のマニュアルを参照してください。

3.2 実行時オプション

デッドロック調査機能で使用するオプションを以下に示します。本オプションはmpiexecコマンドの実行時オプションです。mpiexecコマンドの使用方法については、“MPI使用手引書”を参照してください。

デッドロック調査機能のオプションには、先頭のハイフン(-)の数が1つのものと2つのものがありますが、どちらも同じ意味です。以降の説明では、ハイフン(-)が1つのオプションを使用します。

{ -fjdbg-dlock | --fjdbg-dlock }

デッドロック調査機能を有効にします。本オプションは -fjdbg-out-dir オプションと同時に指定してください。本オプションのみを指定した場合、エラーメッセージを出力しプログラムの実行を終了します。

{ -fjdbg-out-dir *output-dir* | --fjdbg-out-dir *output-dir* }

調査結果ファイルを格納するディレクトリを指定します。*output-dir* には、格納ディレクトリ名を相対パスまたは絶対パスで指定します。指定したディレクトリが存在しない場合、本オプションに指定したディレクトリを新規に作成します。指定したディレクトリがすでに存在する場合、その直下に“deadlock”という名前のファイルが存在してはいけません。指定したディレクトリの直下に“deadlock”という名前のディレクトリが存在する場合、そのディレクトリは空ディレクトリでなければいけません。これらの条件を満たさない場合、プロセス0の標準エラー出力にエラーメッセージを出力し、プロセス0の実行を終了します。なお、ジョブが終了しない場合があります。その場合はジョブを手動で削除してください。

本オプションは -fjdbg-dlock オプションと同時に指定してください。本オプションのみを指定した場合、エラーメッセージを出力しプログラムの実行を終了します。

例

mpiexecコマンドへのオプション指定例

```
mpiexec -fjdbg-dlock -fjdbg-out-dir "/fefs/log" -n 4 ./a.out
```

3.3 調査結果ファイル

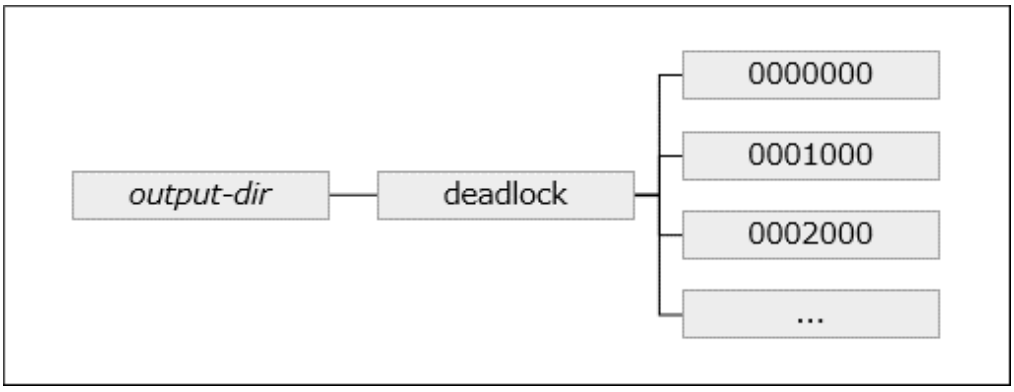
デッドロック調査機能が出力する調査結果ファイルについて説明します。

デッドロック調査機能は、プログラムの実行中に調査結果ファイルの格納用ディレクトリを生成します。その後、pjsigコマンドを投入したタイミングで格納用ディレクトリ内に調査結果ファイルを出力します。格納用ディレクトリの構成は以下のとおりです。

- *output-dir* 直下に“deadlock”という名前のディレクトリ(以降、“deadlock”ディレクトリと呼びます)を作成します。
- “deadlock”ディレクトリ内に調査結果ファイルを格納します。
- “deadlock”ディレクトリ直下に1000ランク単位でディレクトリを作成し、ランクごとに調査結果ファイルを出力します。調査結果ファイル名は「ジョブID.ランク番号」の規則に従います。

ディレクトリ構成を下図に示します。

図3.1 ディレクトリ構成



例

ジョブIDが99999、ランク番号0～2のプログラムの実行結果

```
output-dir/deadlock/0000000/99999.0
output-dir/deadlock/0000000/99999.1
output-dir/deadlock/0000000/99999.2
```

注意

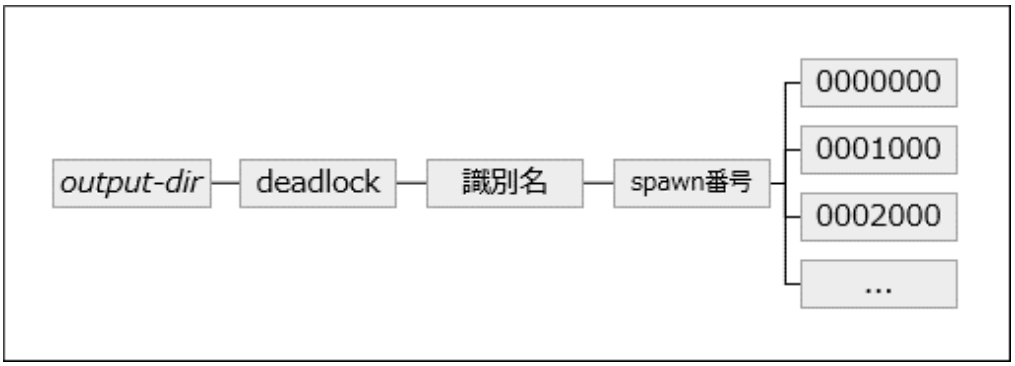
MPI_COMM_SPAWNルーチンまたはMPI_COMM_SPAWN_MULTIPLEルーチンを使用して動的に生成されたプロセスの場合、通常のプロセスとは以下の点が異なります。MPI_COMM_SPAWNルーチンまたはMPI_COMM_SPAWN_MULTIPLEルーチンを使用する場合の詳細については、“1.3 ソースコードの修正”を参照してください。

- “deadlock”ディレクトリ直下に、プログラム内でinfoキー“fjdbg_spawn_dir_name”に定義した値を名前としたディレクトリを作成します。infoキー“fjdbg_spawn_dir_name”が定義されていない場合、“spawn”という名前のディレクトリを作成します。本ディレクトリを以降“識別名”ディレクトリと呼びます。
- “識別名”ディレクトリ直下にspawn番号ごとにディレクトリを作成します。

- spawn番号ごとのディレクトリ直下に1000ランク単位でディレクトリを作成し、ランクごとに調査結果ファイルを出力します。調査結果ファイル名は“*ジョブID.ランク番号@spawn番号*”の規則に従います。

  ディレクトリ構成を下図に示します。

  図3.2 ディレクトリ構成



例

ジョブIDが99999、識別名なし、spawn番号1、ランク番号が0～2のプログラムの実行結果

```
output-dir/deadlock/spawn/1/0000000/99999.0@1
output-dir/deadlock/spawn/1/0000000/99999.1@1
output-dir/deadlock/spawn/1/0000000/99999.2@1
```

調査結果ファイルはそのままでも参照できますが、後述の重複除去機能を使用することを前提としています。重複除去機能を使用する場合、生成されたディレクトリ名、ディレクトリ構成、およびファイル名を変更しないでください。調査結果ファイルを直接利用したい場合、“付録B 調査結果ファイルの詳細”を参照してください。

3.4 重複除去機能

重複除去機能については、“2.3 重複除去機能”を参照してください。

3.5 注意事項

デッドロック調査機能を使用する際の注意事項について説明します。

3.5.1 シグナルハンドラの動作について

デッドロック調査機能を使用する場合、SIGHUPおよびSIGXCPUシグナルを捕捉するため、翻訳時に動的リンクライブラリで設定されるシグナルハンドラを起動しません。ただし、利用者が上記のシグナルを捕捉するプログラムを作成している場合には、利用者の設定したシグナルハンドラが有効になります。デッドロック調査機能を指定した場合、指定したシグナルを利用するほかの機能よりもデッドロック調査機能が優先されます。

第4章 コマンドファイルによるデバッガ制御機能

この章では、コマンドファイルによるデバッガ制御機能について説明します。

コマンドファイルによるデバッガ制御機能を使用すると、ジョブ投入時にコマンドファイルを利用したデバッガ制御を行います。これにより、プロセスごとに異なるデバッグを実行したり特定のプロセスだけをデバッグの対象にしたりすることができます。コマンドファイルによるデバッガ制御機能ではGDBのバッチモードを利用します。GDBのバッチモードではコマンドファイルと呼ばれるGDBのコマンドを記述したファイルを使用します。バッチモードおよびコマンドファイルの詳細についてはGDBのオンラインマニュアルを参照してください。動的に生成されたプロセスでは、コマンドファイルによるデバッガ制御機能は使用できません。

4.1 実行時オプション

コマンドファイルによるデバッガ制御機能で使用するオプションを以下に示します。本オプションはmpiexecコマンドの実行時オプションです。mpiexecコマンドの使用方法については、“MPI使用手引書”を参照してください。

コマンドファイルによるデバッガ制御機能のオプションには、先頭のハイフン(-)の数が1つのものと2つのものがありますが、どちらも同じ意味です。以降の説明では、ハイフン(-)が1つのオプションを使用します。

{ -gdbx "[*rank-no*:] *command-file* [;...]" | --gdbx "[*rank-no*:] *command-file* [;...]" }

コマンドファイルによるデバッガ制御機能を有効にします。*rank-no*には、コマンドファイルによるデバッガ制御機能を実行するランク番号を指定します。

rank-no に1つだけを指定する場合

ランク番号を指定します。(例: -gdbx "1:command.txt")

rank-no に範囲指定する場合

開始番号と終了番号をハイフン(-)で繋ぎます。(例: -gdbx "1-10: command.txt")

rank-no に複数指定する場合

ランク番号または範囲指定をコンマ(,)で区切ります。(例: -gdbx "1,4-6,8:command.txt")

rank-no を省略した場合

全ランクを対象にコマンドファイルによるデバッガ制御機能を実行します。(例: -gdbx "command.txt")

*command-file*にはコマンドファイルのパスを相対パスまたは絶対パスで指定します。*rank-no*の指定がある場合、*rank-no*と*command-file*をコロン(:)で区切ってください。*rank-no*の指定がない場合、コロン(:)は不要です。*rank-no*と*command-file*を組み合わせることで、コマンドファイルとそのデバッグ対象ランクを指定できます。セミコロン(;)で区切ることで、*rank-no*と*command-file*の組み合わせを複数指定することもできます。複数の組み合わせを指定する場合、全体をダブルクォーテーション(")で囲ってください。ダブルクォーテーション(")で囲まない場合、セミコロン(;)以降を別のコマンドとして処理します。

[-fjdbg-out-dir *output-dir* | --fjdbg-out-dir *output-dir*]

デバッグ結果ファイルの格納先ディレクトリを、相対パスまたは絶対パスで*output-dir*に指定します。本オプションを省略した場合、格納先ディレクトリはmpiexecコマンドの指定に従います。格納先ディレクトリが存在しない場合、*output-dir*を新規に作成します。格納先ディレクトリが存在する場合、その直下に“gdbx”という名前のファイルが存在してはいけません。格納先ディレクトリに“gdbx”という名前のディレクトリが存在する場合、そのディレクトリは空ディレクトリでなければいけません。これらの条件を満たさない場合、プロセス0の標準エラー出力にエラーメッセージを出力し、プロセス0の実行を終了します。なお、ジョブが終了しない場合があります。その場合はジョブを手動で削除してください。

例

mpiexecコマンドへのオプション指定例

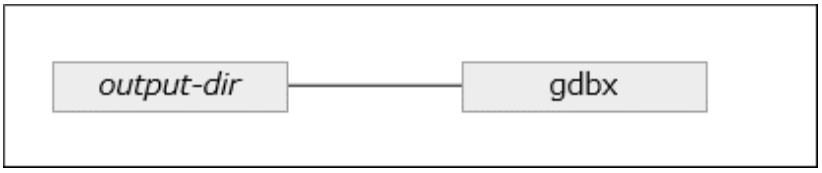
```
mpiexec -gdbx "0,1:/tmp/command.txt" -n 2 ./a.out arg1 arg2 arg3
```

```
mpiexec -gdbx "0,1:/tmp/command1.txt;2:/tmp/command2.txt" -fjdbg-out-dir "/fefs/log" -n 2 ./a.out arg1 arg2 arg3
```

4.2 デバッグ結果ファイル

コマンドファイルによるデバッガ制御機能実行時、標準出力にデバッグ結果を出力します。-fjdbg-out-dir オプションを指定した場合、指定ディレクトリ直下に“gdbx”という名前のディレクトリを作成し、デバッグ結果ファイルを出力します。デバッグ結果ファイル名は「ジョブID.ランク番号」の規則に従います。ディレクトリ構成を下図に示します。

図4.1 ディレクトリ構成



例

ジョブIDが99999、ランク番号が0～2のプログラムの実行結果

```
output-dir/gdbx/99999.0
output-dir/gdbx/99999.1
output-dir/gdbx/99999.2
```

参考

標準出力に出力する場合、mpiexecに“ --ofprefix *data,rank,nid*”オプションを付加することで、どのランクとノードIDから出力されたのか判断することが可能です。詳細については“MPI 使用手引書”を参照してください。

付録A　メッセージ一覧

ここでは、並列実行デバッガが出力するメッセージについて説明します。メッセージは標準エラー出力に出力します。

A.1　メッセージの形式

並列実行デバッガが出力するメッセージの形式は以下のとおりです。

メッセージ形式

［*メッセージレベル*］ *カテゴリー 番号*（*復帰コード*） *メッセージ本文*

メッセージレベル

ERROR：重度のエラーです。メッセージ出力後に処理を中断します。

WARN：軽度のエラー、または警告です。メッセージ出力後に処理を継続します。

カテゴリー

fjdbg_sig：異常終了調査機能またはデッドロック調査機能のメッセージです。

fjdbg_summary：重複除去機能のメッセージです。

fjdbg：コマンドファイルによるデバッガ制御機能または並列実行デバッガ全般のメッセージです。

番号

メッセージ管理番号です。メッセージごとに一意の番号が割り当てられます。

復帰コード

担当保守員が使用します。復帰コードがないメッセージもあります。

A.2　メッセージ一覧

並列実行デバッガが出力するメッセージを以下に記します。

[ERROR] *cat* 1000 (*ret*) allocation error.

[メッセージの説明]

作業用領域の獲得に失敗しました。

[パラメーターの説明]

cat: カテゴリー

ret: 復帰コード

[システムの処理]

並列実行デバッガの処理を終了します。

[利用者の処置]

出力されたメッセージの番号と復帰コードを担当保守員(SE)にご連絡ください。

[ERROR] *cat* 1010 (*ret*) execution failed.

[メッセージの説明]

実行に失敗しました。

[パラメーターの説明]

cat: カテゴリー

ret: 復帰コード

[システムの処理]

並列実行デバッガの処理を終了します。

[利用者の処置]

出力されたメッセージの番号と復帰コードを担当保守員(SE)にご連絡ください。ただし、復帰コード(*ret*)が603の場合は“1.5.2 エラーメッセージ [ERROR] fjdbg_sig 1010(603) execution failed.”を参照してください。

[ERROR] fjdbg_sig 2012 (*ret*) *dirpath* is not a directory.

[メッセージの説明]

-fjdbg-out-dir オプションに指定された*dirpath* はディレクトリではありません。

[パラメーターの説明]

ret: 復帰コード

dirpath: ディレクトリパス

[システムの処理]

並列実行デバッガの処理を終了します。

[利用者の処置]

-fjdbg-out-dir オプションの引数に、ファイル名を指定していないか確認してください。

[ERROR] *cat* 3010 option invalid.(*opt*)

[メッセージの説明]

実行時オプション*opt* の指定に誤りがあります。

[パラメーターの説明]

cat: カテゴリー

opt: 実行時オプション

[システムの処理]

並列実行デバッガの処理を終了します。

[利用者の処置]

実行時オプション *opt* の指定に誤りがないか確認してください。

[ERROR] fjdbg 3011 when -gdbx option is specified, -fjdbg-sig and -fjdbg-dlock option cannot be specified.

[メッセージの説明]

-gdbx オプションが指定されている場合、-fjdbg-sig オプションおよび-fjdbg-dlock オプションは指定できません。

[システムの処理]

並列実行デバッガの処理を終了します。

[利用者の処置]

-gdbx オプションを指定する場合、-fjdbg-sig オプションまたは-fjdbg-dlock オプションは指定しないでください。-fjdbg-sig オプションまたは-fjdbg-dlock オプションを指定する場合、-gdbx オプションは指定しないでください。

[ERROR] fjdbg 3012 when -fjdbg-sig option or -fjdbg-dlock option is specified, -fjdbg-out-dir option is necessary.

[メッセージの説明]

-fjdbg-sig オプションまたは-fjdbg-dlock オプションのどちらかを指定した場合、-fjdbg-out-dir オプションの指定が必要です。

[システムの処理]

並列実行デバッガの処理を終了します。

[利用者の処置]

-fjdbg-out-dir オプションを指定してください。

[ERROR] fjdbg 3013 *libpath* is not found.

[メッセージの説明]

ライブラリ*libpath* が存在しません。

[パラメーターの説明]

libpath: ライブラリパス

[システムの処理]

並列実行デバッガの処理を終了します。

[利用者の処置]

出力されたメッセージの番号と*libpath* の情報を担当保守員(SE)にご連絡ください。

[ERROR] fjdbg 3014 one or more files exist in the specified directory. (*dirpath*)

[メッセージの説明]

デバッグ結果出力先ディレクトリ(*dirpath*)配下にファイルが存在します。

[パラメーターの説明]

dirpath: ディレクトリパス

[システムの処理]

並列実行デバッガの処理を終了します。

[利用者の処置]

デバッグ結果出力先ディレクトリ(*dirpath*)配下に存在するファイルを削除してください。

[ERROR] fjdbg 3015 making the specified directory failed.

[メッセージの説明]

デバッグ結果出力先ディレクトリの作成に失敗しました。

[システムの処理]

並列実行デバッガの処理を終了します。

[利用者の処置]

指定したディレクトリパスに、デバッグ結果出力先ディレクトリが作成できるか確認してください。

[ERROR] fjdbg 3016 specified rank number is not correct.(*parm*)

[メッセージの説明]

-gdbx オプションに指定したランク番号*parm* に誤りがあります。

[パラメーターの説明]

parm: ユーザーが指定した引数

[システムの処理]

並列実行デバッガの処理を終了します。

[利用者の処置]

-gdbx オプションに指定したランク番号が正しいか確認してください。

[ERROR] fjdbg 3017 rank number is not a numerical value.(*parm*)

[メッセージの説明]

-gdbx オプションに指定したランク番号*parm* が数値ではありません。

[パラメーターの説明]

parm: ユーザーが指定した引数

[システムの処理]

並列実行デバッガの処理を終了します。

[利用者の処置]

オプションに指定したランク番号が正しいか確認してください。

[WARN] fjdbg 3018 specified command file does not exist, the program is executed without the debugger.

[メッセージの説明]

-gdbx オプションに指定したコマンドファイルが存在しません、デバッガを使用せず実行します。

[システムの処理]

処理を継続します。

[利用者の処置]

オプションに指定したコマンドファイルへのパスが正しいか確認してください。

[ERROR] fjdbg_summary 4011 (*ret*) option invalid.(*opt*)

[メッセージの説明]

重複除去機能の実行時オプション*opt* の指定に誤りがあります。

[パラメーターの説明]

ret: 復帰コード

opt: 実行時オプション

[システムの処理]

処理を中断します。

[利用者の処置]

実行時オプション *opt* の指定に誤りがないか確認してください。

[ERROR] fjdbg_summary 4012 (*ret*) directory open error.

[メッセージの説明]

重複除去機能対象ディレクトリのオープンに失敗しました。

[パラメーターの説明]

ret: 復帰コード

[システムの処理]

処理を中断します。

[利用者の処置]

実行時オプションに指定した重複除去機能対象ディレクトリが正しいか確認してください。

[WARN] fjdbg_summary 4013 (*ret*) because -a and -n were specified at the same time, -n is made effective.

[メッセージの説明]

-a オプションと-n オプションが同時に指定されたため、-n オプションを有効にしました。

[パラメーターの説明]

ret: 復帰コード

[システムの処理]

処理を継続します。

[利用者の処置]

ありません。

付録B　調査結果ファイルの詳細

注意

ここでは、調査結果ファイルを直接利用する必要のある方のみ対象とします。

通常は重複除去機能を使用してください。重複除去機能については“2.3 重複除去機能”を参照してください。

調査結果ファイルは、以下の3つの情報で構成されています。

- 重複除去機能で使用する情報
- GDBのマシンインターフェースインタプリタ(GDB/MI)が出力する情報
- メモリマップ

調査結果ファイル内の各行がどの情報に属するかに関しては、以下の方法で分類できます。

- # で始まる行は、重複除去機能で使用する情報です。
- それ以外の結果行のうち先頭から1つ目のタブ以降は、GDBが出力する情報です。
- &"info proc map¥n" と^doneで囲まれた範囲は、メモリマップです。

注意

異常終了発生からジョブが強制終了するまでの間にGDBの情報やメモリマップの出力が完了しない場合があります。ファイルの終端の文字列が“#end”以外の場合、出力は完了していません。出力されているデータの妥当性が保証できないためご注意ください。

B.1　出力例

```
#start debug: 2019/11/19 13:22:54
#thread_max="3"
#sig_num="6"
1,3     ^done,stack=[frame={level="0",addr="0x0000400000fd039c",func="wait",from="/lib64/libpthread.so.
0"},frame={level="1",addr="0x0000400000063554",func="intergdbHandler",from="/opt/FJSVxtclanga/tcsds-1.1.1/lib64/
libsigdbg.so.1"},frame={level="2",addr="0x000040000005066c",func="<signal handler
called>"},frame={level="3",addr="0x00004000011242c8",func="raise",from="/lib64/libc.so.
6"},frame={level="4",addr="0x0000400001125940",func="abort",from="/lib64/libc.so.
6"},frame={level="5",addr="0x00000000004011c0",func="func",file="tmp_lib.c",fullname="/tmp/
lib.c",line="24"},frame={level="6",addr="0x00000000004010f0",func="sub",file=" tmp _lib.c",fullname="/tmp/
lib.c",line="10"},frame={level="7",addr="0x0000000000401070",func="main",file="sample.c",fullname="/tmp/
sample.c",line="47"}]
1,3     ^done,stack-args=[frame={level="3",args=[]}]
1,3     ^done,locals=[]
1,4     ^done,stack-args=[frame={level="4",args=[]}]
1,4     ^done,locals=[]

(中略)

3,4     ^done,locals=[]
3,5     ^done,stack-args=[frame={level="5",args=[]}]
3,5     ^done,locals=[]
&"info proc exe¥n"
~"process 138¥n"
~"exe = '/tmp/sample.out'¥n"
^done
&"info proc map¥n"
~"process 138¥n"
~"Mapped address spaces:¥n¥n"
```

```
~"          Start Addr           End Addr       Size     Offset objfile¥n"
~"            0x400000           0x410000     0x10000        0x0 /tmp/sample.out¥n"
~"            0x410000           0x420000     0x10000        0x0 /tmp/sample.out¥n"
~"            0x420000           0x6e0000    0x2c0000        0x0 [heap]¥n"

(中略)

~"      0x40000fa30000     0x400010110000     0x6e0000        0x0 ¥n"
~"      0x400010110000     0x400011310000    0x1200000        0x0 ¥n"
~"      0x400011310000     0x400011320000     0x10000        0x0 ¥n"
~"      0xfffffffd0000  0x1000000000000     0x30000        0x0 [stack]¥n"
^done
#end
```