FUJITSU

Fujitsu Software
Technical Computing Suite V4.0L20

Development Studio
Debugger for Parallel Applications
User's Guide

J2UL-2484-02ENZ0(02)
March 2023

Preface

Purpose of This Manual

This guide describes the features and the usage of the Debugger for Parallel Applications function for the system that has the Fujitsu CPU A64FX installed.

Intended Readers

This guide is written for readers who debug programs (called "MPI programs") that are written in Fortran, C, or C++ and call the MPI library.

It is assumed that readers of this guide have the following knowledge:

- MPI (Message Passing Interface)
- Fortran programs, C programs, and C++ programs
- Knowledge of basic Linux(R) and developing shell programming
- Debugging using the GNU debugger (abbreviated as GDB)

Organization of This Manual

This manual consists of the following sections.

Chapter 1 Overview of the Debugger for Parallel Applications

Explains the Debugger for Parallel Applications components and preparation for using the debugger.

Chapter 2 Abnormal Termination Investigative Function

Explains the Abnormal Termination Investigative Function.

Chapter 3 Deadlock Investigative Function

Explains the Deadlock Investigative Function.

Chapter 4 Debugging Control Function with Command Files

Explains the Debugging Control Function with Command Files.

Appendix A Messages

Explains messages output by the Debugger for Parallel Applications.

Appendix B Details of an Investigation Result File

Explains details of investigation result files from the Abnormal Termination Investigative Function and Deadlock Investigative Function.

Related Manuals

This book relates to the following manuals. If necessary, refer also to these manuals.

- "Fortran Language Reference"
- "Fortran User's Guide"
- "Fortran User's Guide Additional Volume COARRAY"
- "Fortran Compiler Messages"
- "C User's Guide"
- "C++ User's Guide"
- "C/C++ Compiler Optimization Messages"
- "Fortran/C/C++ Runtime Messages"
- "MPI User's Guide"

Also, refer to the manuals provided with the following related software:

- "Job Operation Software"

Syntax Description Symbols

A syntax description symbol is a symbol that has specific meaning in syntax. The following symbols are used in this guide.

Symbol name	Symbol	Description
Selection symbols	{ }	Indicates that only one of the enclosed items can be selected
	|	Indicates that it is used as a delimiter in a list of items
Optional symbol	[ ]	Indicates that the enclosed item can be omitted The { } (braces selection symbols) and [ ] (brackets optional symbol) have the same meaning.
Repeat symbol	...	Indicates that the item just before this can be specified repeatedly

Export Controls

Exportation/release of this document may require necessary procedures in accordance with the regulations of your resident country and/or US export control laws.

Trademarks

- Linux(R) is the registered trademark of Linus Torvalds in the U.S. and other countries.
- All other trademarks are the property of their respective owners.
- The trademark notice symbol (TM,(R)) is not necessarily added in the system name and the product name, etc. published in this material.

Date of Publication and Version

Version	Manual code
March 2023, Version 2.2	J2UL-2484-02ENZ0(02)
April 2020, Version 2.1	J2UL-2484-02ENZ0(01)
March 2020, 2nd Version	J2UL-2484-02ENZ0(00)
January 2020, 1st Version	J2UL-2484-01ENZ0(00)

Copyright

Update History

Changes	Location	Version
Added section.	1.5.3	Version 2.2
Added Specifying Environment Variables.	1.2	Version 2.1
Added Source Code Modification.	1.3	
Added description of the dynamically generated processes.	2.2 2.3.1 3.3 Chapter 4	
Improved the explanation.	-	
Added notes on Error Message [ERROR] fjdbg_sig 1010(603) execution failed.	1.3.2 A.2	2nd Version
Changed the look according to product upgrades.	-	

Contents

Chapter 1 Overview of the Debugger for Parallel Applications

Explains the debugger components and preparation for using the Debugger for Parallel Applications.

1.1 Configuration of the Debugger for Parallel Applications

In this manual, a set of the following functions is given a generic term "Debugger for Parallel Applications". The Debugger for Parallel Applications uses GDB to acquire information used in various investigations and control debugging through a command file.

- Abnormal Termination Investigative Function

  This function supports investigations conducted to identify the cause of an abnormal termination of a program. It obtains backtraces or other execution information when a signal is received by an abnormal termination of a program. For details, see "Chapter 2 Abnormal Termination Investigative Function".

- Deadlock Investigative Function

  This function supports investigations conducted to determine whether a program deadlock has occurred and identify the cause of the deadlock. If a program does not end or respond, the function obtains backtraces or other execution information on all job processes. For details, see "Chapter 3 Deadlock Investigative Function".

- Duplication Removal Function

  This function performs data manipulation for readability improvement on an investigation result file from the Abnormal Termination Investigative Function and the Deadlock Investigative Function. It uses a special command (fjdbg_summary). For details, see "2.3 Duplication Removal Function".

- Debugging Control Function with Command Files

  This debugger control function varies debugging control according to the process. The function enables flexible debugging according to the program properties, by using a file with GDB commands written in it. The file is called a command file. For details, see "Chapter 4 Debugging Control Function with Command Files".

If the Abnormal Termination Investigative Function, Deadlock Investigative Function, or Debugging Control Function with Command Files is used, an option to the mpiexec command is required to specify the function. You can use the Abnormal Termination Investigative Function and the Deadlock Investigative Function at the same time but cannot use the Debugging Control Function with Command Files and another investigative function at the same time.

1.2 Specifying Environment Variables

Specify environment variables required when using the Debugger for Parallel Applications.

Table 1.1 Environment variables required when using the Debugger for Parallel Applications

Environment Variable	Vaule
PATH	/*installation_path*/bin
LD_LIBRARY_PATH	/*installation_path*/lib64

For details on "*installation_path*", contact the system administrator.

1.3 Source Code Modification

If you want to use Abnormal Termination Investigative Function or Deadlock Investigative Function for dynamically generated processes using the MPI_COMM_SPAWN routine or the MPI_COMM_SPAWN_MULTIPLE routine, use a gdb_wrapper command. Therefore, the source code should be modified so that it uses gdb_wrapper command.

Specify at the gdb_wrapper command the same option for Abnormal Termination Investigative Function and Deadlock Investigative Function as the mpiexec command.

For the MPI_COMM_SPAWN and MPI_COMM_SPAWN_MULTIPLE routines, refer to the MPI standard or the "MPI User's Guide". For information about the options for Abnormal Termination Investigative Function, see "2.1 Runtime Option". For information about the options for Deadlock Investigative Function, see "3.2 Runtime Option".

Example

Example of "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[])" modified to source code that uses the gdb_wrapper command

Indicates modifications in red text.

```
MPI_Init( &argc, &argv );
MPI_Info info;
MPI_Info_create(&info);
MPI_Info_set(info, "fjdbg_spawn_dir_name", "spawn_test"); // *1

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

/* Suppress */

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

*1: For information about the info key "fjdbg_spawn_dir_name", see "MPI User's Guide".

1.4 Compiler Option

If you use the Debugger for parallel applications, you should specify the -g option when compiling the MPI program. If you do not specify the -g option when compiling the program, you cannot obtain information on argument variables, values of argument variable, local variables, and values of local variable. For details on compilation options, see the "MPI User's Guide".

1.5 Notes

This section explains what should be noted when using the Debugger for Parallel Applications.

1.5.1 Fortran COARRAY Program

The Debugger for Parallel Applications uses rank numbers instead of an image index in COARRAY programs too. The rank number equals the image index minus 1 because rank numbering starts at 0 and the image index starts with 1. For details on COARRAY programs, see the "Fortran User's Guide Additional Volume COARRAY".

1.5.2 Error Message [ERROR] fjdbg_sig 1010(603) execution failed.

In "Abnormal Termination Investigation Function" or "Deadlock Investigation Function" of Debugger for Parallel Applications, in order to avoid the deadlock of Debugger for Parallel Applications itself, there is a time limit before attaching to the executable. However, some execution environments may exceed the time limit before attaching to the executable. If this occurs, it prints the error message "[ERROR] fjdbg_sig 1010 (603) execution failed.". If you receive this error message, set the following environment variables:

Table 1.2 Environment Variables

Environment Variable Name	Default Value	Description
FJ_TOOL_TIME_LIMIT	60	Specifies the time limit (sec) before attaching to the executable. If you exceed the time limit and do not attach to the executable, it prints an error message and exits.

1.5.3 Compile Command

When using the Debugger for Parallel Applications, use the compile commands of the Fujitsu compiler to compile and link your programs. You cannot use the Debugger for Parallel Applications when using the compile commands of other compilers, such as the GNU compiler.

Chapter 2 Abnormal Termination Investigative Function

Explains the Abnormal Termination Investigative Function.

If Abnormal termination of a program occurs, a signal is received. At this timing, you can use the Abnormal Termination Investigative Function to obtain program execution information (backtraces, frame-specific local and argument variable values, disassembly output including the address where the signal occurred, content of a register information, and a memory map).

2.1 Runtime Option

The options used with the Abnormal Termination Investigative Function are listed below. They are the runtime options of the mpiexec command. On how to use the mpiexec command, see the "MPI User's Guide".

The Abnormal Termination Investigative Function has two options: one begins with a single hyphen (-) and the other begins with two hyphens (--). The two are identical in meaning. The remainder of this chapter follows only the format beginning with a single hyphen (-).

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

This option enables the Abnormal Termination Investigative Function. For *signal*, specify the target signal(s). For specification, you can use ill, abrt, fpe, segv, or bus (multiple items can be specified) or specify all. To specify multiple *signal*, delimit them with a comma (,). If all and another argument are specified for *signal* at the same time, the specification of all takes precedence. You cannot omit *signal*. If it is omitted, program execution will terminate with an error message.

ill

Catches the SIGILL signal.

abrt

Catches the SIGABRT signal.

fpe

Catches the SIGFPE signal.

segv

Catches the SIGSEGV signal.

bus

Catches the SIGBUS signal.

all

Catch the SIGILL, SIGABRT, SIGFPE, SIGSEGV, and SIGBUS signals.

This option must be specified together with the -fjdbg-out-dir option. If this option is specified solely, program execution will terminate with an error message.

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

Specify the storage directory that has investigation result files. For *output-dir*, specify the absolute or relative path of the directory. If the directory specified in this option does not exist, a new directory is created. If the specified directory exists, there must be no file named "signal" immediately under the directory. If there is a directory named "signal" immediately under the directory, it must be empty. If these conditions are not satisfied, outputs an error message to standard error output of process 0 and ends execution of process 0. The job may not end. If so, delete the job manually.

This option must be specified together with the -fjdbg-sig option. If this option is specified solely, program execution will terminate with an error message.

 Example

Example of specifying the options in the mpiexec command

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

2.2 Investigation Result File

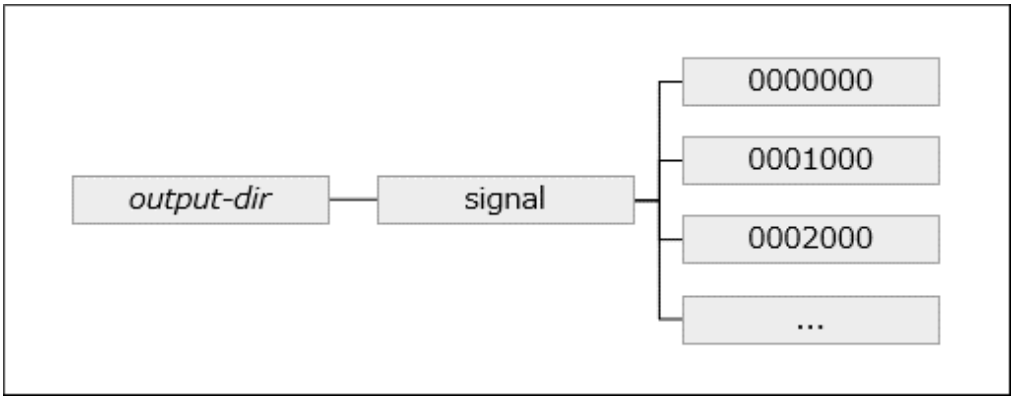
This section explains the investigation result file output by the Abnormal Termination Investigative Function.

The Abnormal Termination Investigative Function creates a storage directory for investigation result files during execution. The storage directory is structured as follows:

- Immediately under *output-dir*, a directory named "signal" (hereinafter called "signal" directory) is created. Only the "signal" directory is created and no more is output when the program ends normally.
- The investigation result file is output into the "signal" directory.
- Immediately under the "signal" directory, a directory is created for each set of 1000 ranks and investigation result files of each rank are output to pertinent directories. Investigation result files are named in the format of "*job-ID.rank-number*".

The figure below shows the directory structure.

Figure 2.1 Directory structure



Example

Execution result of a program with job ID 99999 and rank numbers 0 to 2

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

Note

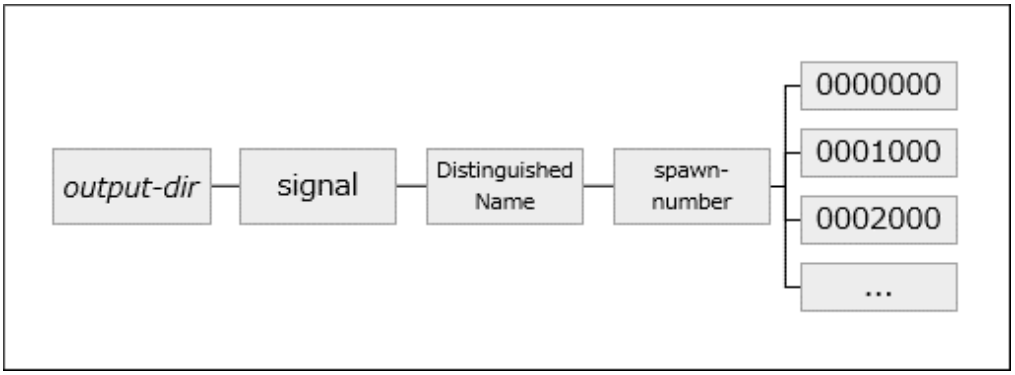
Processes dynamically generated using the MPI_COMM_SPAWN or MPI_COMM_SPAWN_MULTIPLE routines differ from regular processes in the following ways. For more information about using the MPI_COMM_SPAWN or MPI_COMM_SPAWN_MULTIPLE routine, see "1.3 Source Code Modification".

- Directly below the "signal" directory, create a directory named after the value defined in the info key "fjdbg_spawn_dir_name" in the program. If the info key "fjdbg_spawn_dir_name" is not defined, create a directory named "spawn". This directory is hereinafter referred to as the "Distinguished Name" directory.
- Create a directory for each spawn number directly below the "Distinguished Name" directory.

- Directories are created in 1000 rank units directly under each spawn number directory, and investigation result file are output for each rank. The investigation result file names follow the rules in "*job-ID.rank-number@spawn-number*".

  The figure below shows the directory structure.

Figure 2.2 Directory structure



Example

Execution result of a program with job ID 99999, Distinguished Name is none, spawn number 1, and rank numbers 0 to 2

```
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
```

You can view investigation result files as they are. Note however that the use of the Duplication Removal Function (explained later) is assumed here. When you use the Duplication Removal Function, do not change the names of created directories and files and the directory structure. To directly view the investigation result file, see "Appendix B Details of an Investigation Result File".

2.3 Duplication Removal Function

The Duplication Removal Function performs the following processing on the investigation result files output by the Abnormal Termination Investigative Function and the Deadlock Investigative Functions:

- Removal of duplicated backtraces
- Display of formatted program execution information
  - Backtraces
  - Frame-specific local and argument variable values
  - Disassembly output of code before and after the location where the signal was detected
  - Content of registers
  - Memory Map

Hereinafter, the processing performed by the Duplication Removal Function is referred to as "duplication removal processing". For details of deadlock investigative function, see "Chapter 3 Deadlock Investigative Function".

2.3.1 Usage

The Duplication Removal Function uses the fjdbg_summary command. The usage of the fjdbg_summary command is as follows:

fjdbg_summary command format

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

If no runtime option is specified, the command performs "duplication removal in backtrace information with a function name as a key" and "formatting of various types of information" on target code of all ranks.

Runtime Option

-h

The command displays its usage and then terminates. If this option is specified, other options are ignored.

-v

The command displays version information and then terminates. If this option is specified, other options are ignored except the -h option. If this option and the -h option are specified together, only the -h option is valid, and the version information is not displayed.

-n

If this option is specified, the command does not perform duplication removal on backtraces and only performs formatting of various types of information. The -n and -a options cannot be specified together. If the two options are specified together, the -n option takes precedence.

-a

If this option is specified, the command performs duplication removal on backtraces using, in addition to the function name, the address of backtraces as the keys. The -a and -n options cannot be specified together. If the two options are specified together, the -n option takes precedence.

-b

The command changes the conditions for output of backtraces. If this option is not specified, output of backtraces is terminated when the function name for backtraces is not found from symbols in the object file. If the option is specified, backtraces are output until the end regardless of whether the object file contains symbols.

-r *rankspec*

Specify the rank numbers of the duplication removal results that is output. If this option is not specified, all ranks are assumed as the output target. The output information is limited to the rank(s) specified for *rankspec*. You cannot omit *rankspec*. If it is omitted, program execution will terminate with an error message. If *rankspec* includes a rank number whose execution result does not exist, that rank number is ignored.

Specifying only one rank number to *rankspec*

Specify the pertinent rank number. (Example: -r 1)

Specifying a range of rank numbers to *rankspec*

Specify the starting and ending rank numbers of the range by concatenating them with a hyphen(-). (Example: -r 1-10)

Specifying two or more rank numbers to *rankspec*

Specify the rank numbers or ranges by delimiting them with a comma (,). (Example: -r 1,4-6,8)

-p *outrank*

Specify the number of ranks of the frame-specific local and argument variable values that will be output. The command will output frame-specific argument and local variables, beginning with those in the minimum-rank investigation result file, within the range of ranks specified for *outrank*. If this option is not specified, all ranks are assumed as the output target. You cannot omit *outrank*. If it is omitted, program execution will terminate with an error message. Any rank number whose execution result does not exist will not be counted as a rank. If the *outrank* value exceeds the total number of ranks, the duplication removal target of all ranks will be output. If the -r option is effective, the duplication removal target of the ranks specified by that option will be output.

input-dir

Specify the relative or absolute path to the directory that is the target of duplication removal processing. Specify the "signal" or "deadlock" directory within the investigation result directory created by the abnormal termination or deadlock investigative function. However, if the duplication removal target is the dynamically generated processes, specify the "signal/*Distinguished Name*/*spawn-number*" or "deadlock/*Distinguished Name*/*spawn-number*" directory in the investigation result file. If the directory specified in *input-dir* does not exist or is neither the "signal" directory, "deadlock" directory, nor "*spawn-number*" directory program execution will terminate with an error message.

Example

Example

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

2.3.2 Output Content

During execution, the command outputs duplication removal results to standard output. The output content is explained below.

When the Duplication Removal Function is used, the input information is output in the following sequence:

1. Backtraces, frame-specific local and argument variable values
2. Disassembly output of code before and after the location where the signal was detected
3. Content of registers
4. Memory Map

The command outputs information items, item 1 to 3, for each thread and outputs information item 4 at the end. Information items, item 2 to 4, are not included in the target of the Duplication Removal Function. They are always output for each rank. Note that some parts of the items may not be output depending on the investigative function used or problem occurrence status. Example 1 to 4 show information output examples.

Example1: Backtraces, Frame-specific local and argument variable values

```
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 = 0xfffffffec88 [1]
         more than 1 distinct values
       locals:
         int rank = 1 [1]
(omitted)
         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
(omitted)
```

Information

- If the backtraces includes a rank that is included in the target of duplication removal, it is indicated at the beginning just like as [1-3] in Example1. If the information held varies depending on the rank as in local variables (locals:) or argument variables (params:), the number enclosed in [] at the end indicates the rank number.
- If debug information was created during debug program compilation, line numbers and other information associated with the source file will be output. If no debug information is created, information associated with symbol names is output.
- The backtraces start frame is one of the following function names:
  - main
  - __lpg_main
  - _lwp_start
  - __jwe_PE
  - __mpc_PE
  - start_thread
- The generated signal information (for the Abnormal Termination Investigative Function) or the position of abnormal termination is indicated by the "=>*" symbol in the output.
- If there is a variable within a rank that is not displayed due to the specification of the -p *outrank* option, a message "more than *outrank* distinct values" is output.

Example2: Disassembly output of code before and after the location where the signal was detected

```
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
```

Information

The disassembly output including the address where the signal occurred is output only when the Abnormal Termination Investigative Function is in use.

Example3: Content of registers

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

(omitted)

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

Example4: Memory map

```
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

(omitted)

    [ 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]
(omitted)
```

2.4 Notes

This section explains what should be noted when using the Abnormal Termination Investigative Function.

2.4.1 Signal Handler Operation

When the Abnormal Termination Investigative Function is in use, it does not start the signal handler configured for the dynamic link library at the time of compilation in order to catch the SIGILL, SIGABRT, SIGFPE, SIGSEGV, and SIGBUS signals. However, if a user has created a program that catches the above signals, the user-configured signal handler is effective. If the target signals are specified in the runtime option of the Abnormal Termination Investigative Function, the Abnormal Termination Investigative Function takes precedence over other functions that use the specified signals.

Chapter 3 Deadlock Investigative Function

Explains the Deadlock Investigative Function.

By using the Deadlock Investigative Function, you can obtain program execution information on all job processes if a program does not end or respond. Program execution information includes backtraces, frame-specific local variable values and argument variable values, and a memory map.

3.1 Usage

Use the Deadlock Investigative Function as follows:

1. Create a job script for program execution. The necessary steps are as follows:

   a. Using the trap command, change the setting when a SIGHUP (hungup) or SIGXCPU (CPU timeout) signal is received.

   b. Execute the mpiexec command. To enable the Deadlock Investigative Function, you need to specify the runtime options, which will be explained later.

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

2. Submit the job script that was created in step 1.
3. Execute the pjstat command to check the job ID obtained in step 2.
4. Execute the following command in a situation where a deadlock could occur.

```
pjsig -s SIGHUP job-ID
```

Note

After executing the pjsig command, the investigation result file is output then the job finishes.

See

On how to use the trap command, see the online manuals.

On how to use the pjstat command and the pjsig command, see the "Job Operation Software" manual.

3.2 Runtime Option

The options used with the Deadlock Investigative Function are listed below. They are the runtime options of the mpiexec command. On how to use the mpiexec command, see the "MPI User's Guide".

The Deadlock Investigative Function has two options: one begins with a single hyphen (-) and the other begins with two hyphens (--). The two are identical in meaning. The remainder of this chapter follows only the format beginning with a single hyphen (-).

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

This option enables the Deadlock Investigative Function. This option must be specified together with the -fjdbg-out-dir option. If this option is specified solely, program execution will terminate with an error message.

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

Specify the storage directory that has investigation result files. For *output-dir*, specify the absolute or relative path of the directory. If the directory specified in this option does not exist, a new directory is created. If the specified directory exists, there must be no file named "deadlock" immediately under the directory. If there is a directory named "deadlock" immediately under the directory, it must be empty. If these conditions are not satisfied, outputs an error message to standard error output of process 0 and ends execution of process 0. The job may not end. If so, delete the job manually.

This option must be specified together with the -fjdbg-dlock option. If this option is specified solely, program execution will terminate with an error message.

Example

Example of specifying the options in the mpiexec command

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

3.3 Investigation Result File

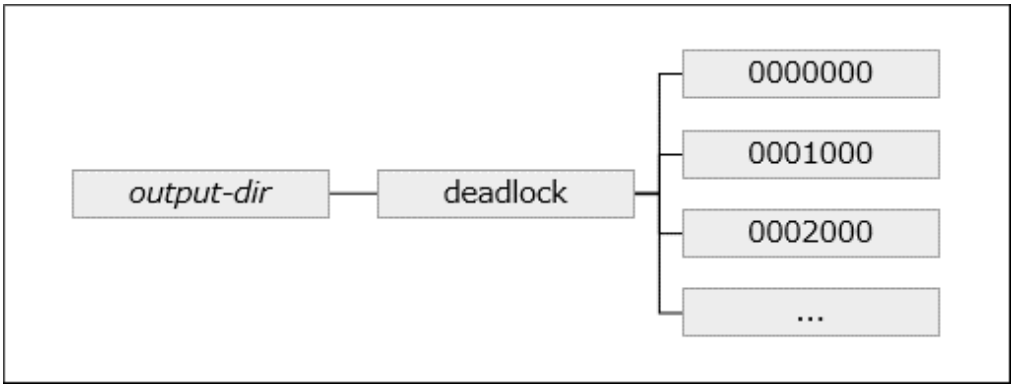
This section explains the investigation result file output by the Deadlock Investigative Function.

The Deadlock Investigative Function creates a storage directory for investigation result files during program execution. Then, when the pjsig command is entered, the investigation result files are output to the storage directory. The structure of the storage directory is as follows:

- Immediately under *output-dir*, a directory named "deadlock" (hereinafter called "deadlock" directory) is created. Only the "deadlock" directory is created and no more is output when the program ends normally.
- The investigation result file is output into the "deadlock" directory.
- Immediately under the "deadlock" directory, a directory is created for each set of 1000 ranks and investigation result files of each rank are output to pertinent directories. Investigation result files are named in the format of "*job-ID.rank-number*".

The figure below shows the directory structure.

Figure 3.1 Directory structure



Example

Execution result of a program with job ID 99999 and rank numbers 0 to 2

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

Note

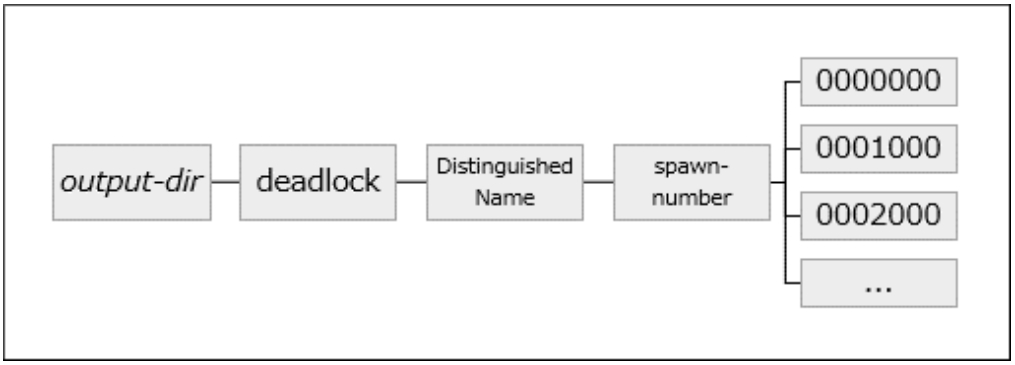
Processes dynamically generated using the MPI_COMM_SPAWN or MPI_COMM_SPAWN_MULTIPLE routines differ from regular processes in the following ways. For more information about using the MPI_COMM_SPAWN or MPI_COMM_SPAWN_MULTIPLE routine, see "1.3 Source Code Modification".

- Directly below the "deadlock" directory, create a directory named after the value defined in the info key "fjdbg_spawn_dir_name" in the program. If the info key "fjdbg_spawn_dir_name" is not defined, create a directory named "spawn". This directory is hereinafter referred to as the "Distinguished Name" directory.

- Create a directory for each spawn number directly below the "Distinguished Name" directory.
- Directories are created in 1000 rank units directly under each spawn number directory, and investigation result file are output for each rank. The investigation result file names follow the rules in "*job-ID.rank-number@spawn-number*".

  The figure below shows the directory structure.

Figure 3.2 Directory structure



Example

Execution result of a program with job ID 99999, Distinguished Name is none, spawn number 1, and rank numbers 0 to 2

```
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
```

You can view investigation result files as they are. Note however that the use of the duplication removal function (explained later) is assumed here. When you use the duplication removal function, do not change the names of created directories and files and the directory structure. To directly view the investigation result file, see "Appendix B Details of an Investigation Result File".

3.4 Duplication Removal Function

For information on the duplication removal function, see "2.3 Duplication Removal Function".

3.5 Notes

This section explains what should be noted when using the Deadlock Investigative Function.

3.5.1 Signal Handler Operation

When the Deadlock Investigative Function is in use, it does not start the signal handler configured for the dynamic link library at the time of compilation in order to catch the SIGHUP and SIGXCPU signals. However, if a user has created a program that catches the above signals, the user-configured signal handler is effective. If the target signals are specified in the runtime option of the Deadlock Investigative Function, the Deadlock Investigative Function takes precedence over other functions that use the specified signals.

Chapter 4 Debugging Control Function with Command Files

Explains the Debugging Control Function with Command Files.

The Debugging Control Function with Command Files controls debugging by using the command files at the job submission time. This enables debugging to vary by process and debugging targeted at only a specific process. The Debugging Control Function with Command Files uses GDB batch mode. GDB batch mode uses files with GDB commands written in them, called command files. For details on the batch mode and command files, see the GDB online manuals. Debugging Control Function with Command Files cannot be used in the dynamically generated processes.

4.1 Runtime Option

The options used with the Debugging Control Function with Command Files are listed below. They are the runtime options of the mpiexec command. On how to use the mpiexec command, see the "MPI User's Guide".

The Debugging Control Function with Command Files has two options: one begins with a single hyphen (-) and the other begins with two hyphens (--). The two are identical in meaning. The remainder of this chapter follows only the format beginning with a single hyphen (-).

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

This option enables the Debugging Control Function with Command Files. Specify the rank number for *rank-no*. The rank number is associated with the execution of the Debugging Control Function with Command Files.

Specifying only one rank number to *rank-no*

Specify the pertinent rank number. (Example: -gdbx "1:command.txt")

Specifying a range of rank numbers to *rank-no*

Specify the starting and ending rank numbers of the range by concatenating them with a hyphen (-). (Example: -gdbx "1-10:command.txt")

Specifying two or more rank numbers to *rank-no*

Specify the rank numbers or ranges by delimiting them with a comma (,). (Example: -gdbx "1,4-6,8:command.txt")

Not specifying rank number to *rank-no*

The Debugging Control Function with Command Files is executed on all ranks.

Specify the relative or absolute path of a command file for *command-file*. If you specify *rank-no*, separate *rank-no* and *command-file* with a colon (:). If you do not specify *rank-no*, no colon is required. By combining *rank-no* and *command-file*, you can specify a command file and its debugging target rank. You can also specify multiple combinations of *rank-no* and *command-file* by separating the combinations with a semicolon (;). When specifying multiple combinations, enclose the entire combinations in double quotations ("). If double quotations are not used, the part subsequent to a semicolon (;) is treated as another command.

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

Specify the relative path or absolute path of the storage directory for debugging result files in *output-dir*. If this option is omitted, the storage directory is that specified in the mpiexec command. If the storage directory does not exist, a new *output-dir* directory is created. If the storage directory exists, there must be no file named "gdbx" immediately under it. If there is a directory named "gdbx" immediately under the storage directory, it must be empty. If these conditions are not satisfied, outputs an error message to standard error output of process 0 and ends execution of process 0. The job may not end. If so, delete the job manually.

Example

Example of specifying the options in the mpiexec command

```
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 Debugging Result File

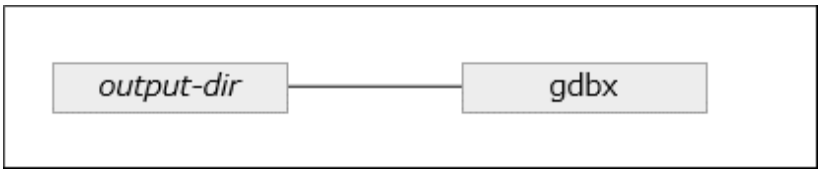
During execution, the command outputs Debugging Control Function with Command Files results (hereinafter called as debugging result file) to standard output.

If -fjdbg-out-dir option is specified, immediately under *output-dir*, a directory named "gdbx" (hereinafter called "gdbx" directory) is created and the debugging result file is output into the "gdbx" directory.

Debugging result files are named in the format of "*job-ID.rank-number*".

The figure below shows the directory structure.

Figure 4.1 Directory structure



Example

Execution result of a program with job ID 99999 and rank numbers 0 to 2

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

Information

In the case of output to standard output, you can add the "--ofprefix *data,rank,nid*" option to the mpiexec command to determine which rank and which node ID are the source of the output. For details, see the "MPI User's Guide".

Appendix A Messages

This appendix explains messages output by the Debugger for Parallel Applications Messages are output to standard output.

A.1 Message Format

Messages output by the Debugger for Parallel Applications are in the following format:

Message Format

```
[Message level] category number (return code) message
```

Message Level

ERROR: Serious error. Processing will be suspended after the message output.

WARN: Minor error or warning. Processing will continue after the message output.

Category

fjdbg_sig: The message concerns the Abnormal termination investigative function or Deadlock investigative function.

fjdbg_summary: The message concerns the Duplication removal function.

fjdbg: The message concerns the debugging control function with command files or a general Debugger for parallel applications matter.

Number

Message management number. A unique number is assigned to each message.

Return code

Maintenance engineers use a return code. Some messages do not have it.

A.2 Messages

Messages output by the Debugger for Parallel Applications are as follows:

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

[Message Explanation]

Failed to acquire the work area.

[Parameters Explanation]

cat: Category

ret: Return code

[System Behavior]

Ends Debugger for Parallel Applications processing.

[User Response]

Notify the maintenance engineer (SE) of the output message number and return code.

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

[Message Explanation]

Failed execution.

[Parameters Explanation]

cat: Category

ret: Return code

[System Behavior]

Ends Debugger for Parallel Applications processing.

[User Response]

Notify the maintenance engineer (SE) of the output message number and return code. If the return code (*ret*) is 603, refer to "1.5.2 Error Message [ERROR] fjdbg_sig 1010(603) execution failed.".

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

[Message Explanation]

dirpath, which is specified as the -fjdbg-out-dir option, is not a directory.

[Parameters Explanation]

ret: Return code

dirpath: Directory path

[System Behavior]

Ends Debugger for Parallel Applications processing.

[User Response]

Check if a file name is specified for the argument of the -fjdbg-out-dir option.

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

[Message Explanation]

The specification of the runtime option *opt* is incorrect.

[Parameters Explanation]

cat: Category

opt: Runtime option

[System Behavior]

Ends Debugger for Parallel Applications processing.

[User Response]

Check that the specification of the runtime option *opt* is correct.

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

[Message Explanation]

If the -gdbx option is specified, you cannot specify the -fjdbg-sig option and -fjdbg-dlock option.

[System Behavior]

Ends Debugger for Parallel Applications processing.

[User Response]

When specifying the -gdbx option, do not specify the -fjdbg-sig or -fjdbg-dlock option. When specifying the -fjdbg-sig or -fjdbg-dlock option, do not specify the -gdbx option.

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

[Message Explanation]

If the -fjdbg-sig option or -fjdbg-dlock option is specified, you need to specify the -fjdbg-out-dir option.

[System Behavior]

Ends Debugger for Parallel Applications processing.

[User Response]

Specify the -fjdbg-out-dir option.

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

[Message Explanation]

The library *libpath* does not exist.

[Parameters Explanation]

libpath: Library path

[System Behavior]

Ends Debugger for Parallel Applications processing.

[User Response]

Notify the maintenance engineer (SE) of the output message number and the *libpath* information.

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

[Message Explanation]

A file exists in the debugging result output directory (*dirpath*).

[Parameters Explanation]

dirpath: Directory path

[System Behavior]

Ends Debugger for Parallel Applications processing.

[User Response]

Delete the file that exists in the debugging result output directory (*dirpath*).

[ERROR] fjdbg 3015 making the specified directory failed.

[Message Explanation]

Failed to create a debugging result output directory.

[System Behavior]

Ends Debugger for Parallel Applications processing.

[User Response]

Check if you can create a debugging result output directory at the specified directory path.

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

[Message Explanation]

The rank number *parm* specified for the -gdbx option is incorrect.

[Parameters Explanation]

parm: Argument(s) specified by the user

[System Behavior]

Ends Debugger for Parallel Applications processing.

[User Response]

Check that the rank number specified for the -gdbx option is correct.

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

[Message Explanation]

The rank number *parm,* which is specified for the -gdbx option, is not a numeric value.

[Parameters Explanation]

parm: Argument(s) specified by the user

[System Behavior]

Ends Debugger for Parallel Applications processing.

[User Response]

Check that the rank number specified for the option is correct.

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

[Message Explanation]

The command file specified by the -gdbx option does not exist. The program is executed without using the debugger.

[System Behavior]

Continues processing.

[User Response]

Check if the path to the command file specified by the option is correct.

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

[Message Explanation]

The specification of the runtime option *opt* for the duplication removal function is incorrect.

[Parameters Explanation]

ret: Return code

opt: Runtime option

[System Behavior]

Suspends processing.

[User Response]

Check that the specification of the runtime option *opt* is correct.

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

[Message Explanation]

Failed to open the target directory of the duplication removal function.

[Parameters Explanation]

ret: Return code

[System Behavior]

Suspends processing.

[User Response]

Check that the target directory specified with the runtime option for the duplication removal function is correct.

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

[Message Explanation]

Because the -a option and -n option were specified at the same time, the -n option is validated.

[Parameters Explanation]

ret: Return code

[System Behavior]

Continues processing.

[User Response]

None.

Appendix B Details of an Investigation Result File

Note

This appendix is intended only for those who need to directly use investigation result files.

Usually, use the Duplication removal function. For information on the Duplication removal function, see "2.3 Duplication Removal Function".

An investigation result file consists of the following three types of information:

- Information that is used by the Duplication removal function
- Information output by the GDB man-machine interface interpreter (GDB/MI)
- Memory map

You can judge each line of an investigation result file as belonging to one of the above types as follows:

- A line that begins with a # symbol is information that is used by the Duplication removal function.
- The part after the first tab on other result lines is information that is output by the GDB.
- The range between &"info proc map\n" and ^done is a memory map.

Note

During the period from abnormal termination to forced job termination, the output of GDB information or a memory map may be incomplete. If the string at the end of the file is not #end, the output is not completed. Note that the validity of output data cannot be guaranteed in this case.

B.1 Example

```
#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=[]

(omitted)

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"

(omitted)

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