Chapter 5

Debugging and Error Handling

This chapter contains information on

- debugging
- error handling
- safety routines
- error files
- Fortran traceback files

5.1 Debugging

On the CM-5, message-passing programs are debugged using the Prism parallel programming environment.

Prism tracks the application across the entire partition, allowing the programmer to follow a partition-wide overview or to zoom in on the activities of a single node or group of nodes.

Applications written in C, C*, Fortran 77, and CM Fortran can make full use of Prism. See the description of Node Prism in the *Prism User's Guide* or the on-line Prism documentation for details.

There is no Prism support for applications written in C++.

Copyright © 1995 Thinking Machines Corporation

5.2 Error Handling

CMOST provides two error handlers:

`CM_panic ("error_message")`	for host programs
`CMPN_panic ("error_message")`	for node programs

CMMD provides an equivalent function:

```
int CMMD_error("error_format", [args] ...)
```

The syntax of this function is the same as the syntax of the UNIX function `printf`. The "*error_format*" argument can contain a character string (that is, an error message) that is copied into the output stream; it can also contain conversion specification for the optional arguments. (See the on-line man page for `printf` for details regarding `CMMD_error`'s arguments.) If successful, `CMMD_error` does not return.

The `CMMD_error` function can be used on either host or node programs; it calls the appropriate OS function in either case. Users are encouraged to use `CMMD_error`, rather than the OS functions. It provides the cleanest method for terminating a program in case of error.

Both the OS functions and the CMMD function allow you to supply your own error message. You can word your error messages to be as helpful as possible. Using different prefixes for different routines, for example, or otherwise identifying the source (and, to the extent possible, the cause) of the error is often useful, especially if you are writing code for others to use.

NOTE

Having any node call the exit routine (`EXIT` or `STOP` in Fortran, `exit` in C) does not terminate the program; it merely halts that particular node, which then waits for all other nodes to exit.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

5.2.1 Default Error Handling

The default behavior for the panic routines is to abort the currently running process after printing the specified error message to the user's `stderr` and producing core dumps for the node and host processes. Both routines use the PM and the timesharing daemon to do this. (For details of the default behavior, see the `CM_panic`(1) man page.)

If you are running your code in Prism, Prism's debugger will trap the error signal and halt your code. This allows you to examine and analyze the state of the failed program.

5.2.2 Customized Error Handling

You can alter the default behavior of `CM_panic` and `CMPN_panic` (and therefore of `CMMD_error`) in a number of ways:

- You can set the environment variable `CM_NO_PN_CORE`, to disable the creation of the errors file and the node core dumps, stack file, and heap file. (The command line `setenv CM_NO_PN_CORE` accomplishes this.)
- The default behavior of both routines culminates in the host process receiving a `SIGTERM` signal. You may choose to install a different error handler for `SIGTERM`, or, alternatively, to have your program ignore the `SIGTERM` signal. (If the signal is ignored, the `CM_panic` routine simply returns; the program may or may not be able to recover.)

5.3 CMMD Safety Routines

CMMD includes a pair of routines that can be used to enable and disable safety checking in your program:

```
CMMD_enable_safety()
CMMD_disable_safety()
```

When these functions are invoked, they set a global variable and return immediately, with no return value. When safety mode is enabled, CMMD send and receive functions, channel functions, and associated utility routines

Copyright © 1995 Thinking Machines Corporation

signal errors if the user either passes recognizably illegal arguments or calls the functions under illegal circumstances. For a list of the errors signaled by safety checking, see the *CMMD Reference Manual*.

5.4 When Your Program Is Terminated

If your program is terminated by a `SIGTERM` signal, you will usually get at least two things:

- PN (node) core dump(s) (see Section 5.4.3)
- a PN errors file (see Section 5.4.2)

In addition to those files, Fortran programs may also get a Fortran traceback (see Section 5.5). The data contained in these three files, combined with some well-placed `printf` statements in your code, plus the host core file, if any, should help you to track down the cause of the error.

5.4.1 Using printf

Any node can call the `printf` routine to print out data. The output is printed to `stdout`. It is important that the user set the I/O mode for `stdout` and `stderr` (unit 6 and unit 1 in Fortran 77) to `independent` while debugging. In fact, it is always a good idea to have `stderr` set to independent mode.

NOTE: The `printf` routine, like all I/O functions, polls, so if your CMMD program depends on explicit polling for its operation, incautiously inserting `printf` statements may have unexpected effects.

5.4.2 The Errors File

In the directory from which you executed your program, you should find a file called `CMTSD_errors.`*`pid`*, where *`pid`* is the process ID. This file is generated by the timesharing daemon when a user program crashes; it lists the status of each node (and of the PM, if an error was detected there). The errors file will tell you which nodes crashed, and give you some information

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

about the crash, such as what memory address the node was trying to reference, whether the error was caused by a segmentation fault, and so on.

5.4.3 Core Files

You should also find one or more node core files. These files are named `CMTSD_core_pn`*`X`*`.`*`pid`*, where *`pid`* is again the process ID, and *`X`* is the node identifier. In some circumstances, you may also see a regular core file, from the host process.

To save disk space, only unique cores are dumped. Thus, if several nodes have the same error, only the core for the first node with that type of error is dumped. The first node with no error (if there is such a node) will also dump core.

If you don't want PN core files generated (generating them does take time and disk space), set the environment variable `CM_NO_PN_CORE` to a non-null value.

You can look at PN core files with Prism. See the Prism documentation for details.

5.4.4 CMTSD Files

You may also see two files called `CMTSD_heap.`*`pid`* and `CMTSD_stack.`*`pid`*. These files contain the contents of the parallel stack and heap for the failed process. They are unlikely to be of much use to you. You can simply delete them, if you wish.

5.5 Fortran Tracebacks: A Warning about Synchronization

When a Fortran program dies, it may generate a traceback. The traceback file will be called *`prog`*`.trace`, where *`prog`* is the name of your host program. The file is appended to every time your Fortran program dies, so if you crash multiple times, there will be multiple traces in the file. The last trace in the file is the newest one.

Copyright © 1995 Thinking Machines Corporation

The traceback may give you an indication of which routine the code died in. However, the information may not be reliable. Remember that the host and the nodes are not necessarily synchronized. If a node has an error, the host may continue working for a while before the error status is propagated to it and your program halts. Therefore, the routine or instruction that is executing on the host when the nodes die may have nothing to do with the error.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation