MPI_Init

Initialize the MPI execution environment

Synopsis

```
#include "mpi.h"
int MPI_Init(argc,argv)
int  *argc;
char ***argv;
```

Input Parameters

argc - Pointer to the number of arguments

argv - Pointer to the argument vector

Command line arguments

MPI specifies no command-line arguments but does allow an MPI implementation to make use of them.

-mpiqueue - print out the state of the message queues when `MPI_FINALIZE` is called. All processors print; the output may be hard to decipher. This is intended as a debugging aid.

-mpiversion - print out the version of the implementation (*not* of MPI), including the arguments that were used with configure.

-mpinice nn - Increments the nice value by `nn` (lowering the priority of the program by `nn`). `nn` must be positive (except for root). Not all systems support this argument; those that do not will ignore it.

-mpedbg - Start a debugger in an xterm window if there is an error (either detected by MPI or a normally fatal signal). This works only if MPICH was configured with `-mpedbg`.

-mpipktsize nn - Set the message length where the ADI changed to the long message protocol to `nn`. This only works if MPICH was configured with `-var_pkt`.

The following options are available only on the Chameleon device and devices built with debugging code. Normally, these should only be used by MPICH implementors when debugging new ports.

-mpichdebug - Print out the Chameleon device operations

-mpidbfile filename - Like mpichdebug, but sends the output to the specified file. If the filename contains a `%d`, then that part of the filename is replaced with the rank in `MPI_COMM_WORLD`. For example,

```
-mpidbfile log.%d
```

writes to `log.0`, `log.1`, etc.

-mpichmemdebug - (Chameleon device only) Print out a list of unreclaimed memory. This requires that MPI be built with the `-DMPIR_DEBUG_MEM` switch. This is intended for debugging the MPI implementation itself.

-mpimem - If MPICH was built with `-DMPIR_DEBUG_MEM`, this checks all malloc and free operations (internal to MPICH) for signs of injury to the memory allocation areas.

-mpichmsg - Print out the number of messages received, by category, when the program exits.

-mpidb options - Activate various debugging options. Some require that MPICH have been built with special options. These are intended for debugging MPICH, not for debugging user programs. The available options include:

```
mem      - Enable dynamic memory tracing of internal MPI objects
memall   - Generate output of all memory allocation/deallocation
ptr      - Enable tracing of internal MPI pointer conversions
rank n   - Limit subsequent -mpidb options to on the process with
           the specified rank in MPI_COMM_WORLD.  A rank of -1
           selects all of MPI_COMM_WORLD.
ref      - Trace use of internal MPI objects
reffile filename - Trace use of internal MPI objects with output
           to the indicated file
trace    - Trace routine calls
```

Notes

Note that the Fortran binding for this routine has only the error return argument (`MPI_INIT(ierror)`)

Because the Fortran and C versions of `MPI_Init` are different, there is a restriction on who can call `MPI_Init`. The version (Fortran or C) must match the main program. That is, if the main program is in C, then the C version of `MPI_Init` must be called. If the main program is in Fortran, the Fortran version must be called.

On exit from this routine, all processes will have a copy of the argument list. This is *not required* by the MPI standard, and truely portable codes should not rely on it. This is provided as a service by this implementation (an MPI implementation is allowed to distribute the command line arguments but is not required to).

Command line arguments are not provided to Fortran programs. More precisely, non-standard Fortran routines such as getarg and iargc have undefined behavior in MPI and in this implementation.

The MPI standard does not say what a program can do before an `MPI_INIT` or after an `MPI_FINALIZE`. In the MPICH implementation, you should do as little as possible. In particular, avoid anything that changes the external state of the program, such as opening files, reading standard input or writing to standard output.

Signals used

The MPI standard requires that all signals used be documented. The MPICH implementation itself uses no signals, but some of the software that MPICH relies on may use some signals. The list below is partial and should be independantly checked if you (and any package that you use) depend on particular signals.

IBM POE/MPL for SP2

SIGHUP, SIGINT, SIGQUIT, SIGFPE, SIGSEGV, SIGPIPE, SIGALRM, SIGTERM, SIGIO

-mpedbg switch

SIGQUIT, SIGILL, SIGFPE, SIGBUS, SIGSEGV, SIGSYS

Meiko CS2

SIGUSR2

ch_p4 device

SIGUSR1

Intel Paragon (ch_nx and nx device)

SIGUSR2

Shared Memory (ch_shmem device)

SIGCHLD

Note that if you are using software that needs the same signals, you may find that there is no way to use that software with the MPI implementation. The signals that cause the most trouble for applications include `SIGIO`, `SIGALRM`, and `SIGPIPE`. For example, using `SIGIO` and `SIGPIPE` may prevent X11 routines from working.

Errors

All MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error value; C routines as the value of the function and Fortran routines in the last argument. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. The error handler may be changed with `MPI_Errhandler_set`; the predefined error handler `MPI_ERRORS_RETURN` may be used to cause error values to be returned. Note that MPI does *not* guarentee that an MPI program can continue past an error.

MPI_SUCCESS - No error; MPI routine completed successfully.

MPI_ERR_OTHER - This error class is associated with an error code that indicates that an attempt was made to call `MPI_INIT` a second time. `MPI_INIT` may only be called once in a program.

Location:init.c