Chapter 3

Definitions

3.1 Shared Library and DLL

3.1.1 Shared Library

A *shared library* is a file that is intended to be shared by excutable files and other shared libraries. Shared libraries may be relocated at runtime and may be dynamically loaded at runtime.

3.1.2 Shared Object File

The term used on UNIX and UNIX-based systems to describe a shared library

3.1.3 Dynamic-link Library

The term used on Windows and OS/2 systems to describe a shared library.

3.1.4 Dynamically Loaded Library

A shared library that can be loaded and unloaded at runtime on request by calling the routines `dlopen`, `dlclose`, `dlsym` on UNIX and UNIX-based systems, or `LoadLibrary`, `FreeLibrary`, `GetProcAddress` on Windows systems. Debuggers require the dynamic loading of the MQD shared library to provide MQD support.

3.1.5 DLL

An overloaded term that refers to either a dynamic-link library, dynamically loaded library, or shared library.

3.2 Process and Image Definitions

3.2.1 Image

An *image file* is an executable or shared library file, which may contain symbol definitions needed by the MQD interface.

3.2.2 MPI Process

An *MPI process*, or simply *process* in the scope of this document, is defined to be an operating system (OS) process, which consists of an *address space* and a collection of execution contexts (threads or lightweight processes). The MPI process is part of the MPI application as described in the MPI standard. While the standard does not require that an MPI process be an OS process, this is a requirement for most debuggers and this interface was designed with that assumption.

3.2.3 Address Space

An *address space* is a region of memory that consists of executable code and data, and is partially composed of a collection of image files. The collection of image files may change at any point during the execution of the MPI process, and the image files may be relocated at runtime within the address space at the point they are loaded into memory.

3.2.4 "mqs_image"

An *mqs_image*, or sometimes simply referred to as an image in this document, is an abstract concept that represents the collection of image files loaded into the address space of the MPI process at any given time, and is debugger implementation defined. In static execution environments, where shared libraries are not supported, an *mqs_image* can represent an executable image file. However, in dynamic execution environments, where shared libraries, dynamically loaded shared libraries, and runtime relocation of shared libraries are supported, an *mqs_image* represents the collection of image files loaded into the address space of the MPI process at any given point in time. In this situation *mqs_image* may in fact represent the MPI process itself.

3.3 "Starter" Process

The *starter process* is the process that is responsible for launching the MPI job. The starter process may be a separate process that is not part of the MPI application, or any MPI process may act as a starter process. By definition, the starter process contains functions, data structures, and symbol table information for the MPIR Process Acquisition Interface.

The MPI implementation determines which launch discipline is used, as described in the following subsections.

3.3.1 The MPI Process as the Starter Process

An MPI implementation might implement its launching mechanism such that an MPI process, e.g., the MPI_COMM_WORLD rank 0 process, launches the remaining MPI processes of the MPI application. In such implementations, the MPI process that started the other MPI processes is the starter process.

3.3.2 A Separate mpiexec as the Starter Process

Many MPI implementations use a separate `mpiexec` process that is responsible for launching the MPI processes. In these implementations, the `mpiexec` process is the starter process. Note that the name of the starter process executable varies by implementation; `mpirun` is a name commonly used by several implementations, for example. Other names include (but are not limited to) `srun`, `aprun`, `orterun`, and `prun`.

3.4 MQD Host and Target Node

For the purposes of this document, the *host node* is defined to be the node running the tool process, and a *target node* is defined to be a node running the target application processes the tool is controlling. A target node might also be the host node; that is, the target application processes might be running on the same node as the tool process.