COS786: Parallel and Distributed Computing

MPI: Establishing Communication
Introduction

• Reasons for establishing communication:
  • Two, independent parts of an application.
  • Monitoring tools that attach to an application.
  • A client/server service model.
Overview

- Create an intercommunicator that groups the two sets of original processes that didn't share a communicator before.
- The two groups of processes do not participate in a parent/child relationship.
- The process of creating the intercommunicator is collective and asymmetric.
Overview

- The group of processes that indicates its willingness to accept connections is referred to as the server.

- The group that connects to the server is referred to as the client.

- However, client/server semantics are not necessarily applicable. For example:
  - A client that crashes or doesn't participate in a collective operation may cause the server to fail or hang.
Finding the Server

• Since no communication channel exists, the client must first find the server.

• A rendezvous point is required.

• A third-party may:
  • Provide the rendezvous point;
  • Transmit the rendezvous point from server to client.

• Furthermore, a client may desire only a service and not a specific server.
Finding the Server

- MPI always maintains maximum compatibility to ensure portability. To that end, an implementation should provide compatibility with the following discovery mechanisms:
  - Server resides at well-known Internet host:port.
  - The server prints an address to the terminal.
  - The server places an address on a nameserver.
  - The server is actually a broker.
Finding the Server

- MPI implementations are not required to provide a nameserver.
- However, MPI provides a standard nameserver interface and is compatible with external nameservers.
Finding the Server

- A port_name is a system-supplied string that encodes a low-level network address where a server may be contacted.
- The implementation is free to use any protocol.
- The server establishes a port_name with MPI_OPEN_PORT.
- The server accepts connections to a given port with MPI_COMM_ACCEPT.
- The client uses port_name to connect.
Finding the Server

- An application publishes a service_name.
- It would be efficient if a client could use only a service_name to find a service without having to use a port_name too.
- MPI allows a server to use MPI_PUBLISH_NAME to publish a service_name and port_name.
- A client may retrieve the port_name using MPI_LOOKUP_NAME.
Finding the Server

- Portability implications of publishing:
  - Applications that do not rely on looking up the port_name are most portable.
  - Applications that rely on MPI_PUBLISH_NAME are portable amongst implementations that provide the mechanism but should have a fallback mechanism.
  - Applications that ignore MPI name publishing may provide their own mechanisms. This is not portable.
Server Routines

• MPI_OPEN_PORT(info, port_name)
  • (IN)info : implementation-specific information on how to assign the network address
  • (OUT)port_name : the newly established port. This buffer must be MPI_MAX_PORT_NAME characters long.

• port_name is defined by the implementation so the format may vary.
Server Routines

- MPI_CLOSE_PORT(port_name)
  - (IN)port_name : The port to release.
- Once the port is released by the system it may be re-used.
Server Routines

- **MPI_COMM_ACCEPT**(port_name, info, root, comm, newcomm)
  - *(IN)*port_name : The port, which must have been established using MPI_OPEN_PORT
  - *(IN)*info : System-specific arguments.
  - *(IN)*root : Rank of the root in comm.
  - *(IN)*comm : The communicator over which this call is collective.
  - *(OUT)*newcomm : The intercommunicator between the local (server) group and the remote (client) group.
Client Routines

- **MPI_COMM_CONNECT(port_name, info, root, comm, newcomm)**
  - (IN)port_name : port string – used only on root.
  - (IN)info : system-specific arguments.
  - (IN)root : rank, in comm, of root node.
  - (IN)comm : intracommunicator over which this call is collective.
  - (OUT)newcomm : intercommunicator with the server as the remote group.
Client Routines

- If `port_name` does not exist or was closed then an error of class MPI_ERR_PORT is raised.
- If the client does not encounter a pending MPI_COMM_ACCEPT, the operation will wait until it times out.
- `port_name` must be provided by MPI_OPEN_PORT, called by the server.
- Alternative `port_name` strings may be allowed, e.g. `host:port/ip:port`
Name Publishing

- **MPI_PUBLISH_NAME**(service_name, info, port_name)
  - (IN)service_name: service name to associate with the port
  - (IN)info: system-specific arguments
  - (IN)port_name: a port string
Name Publishing

- The implementation must define the scope of the publication. For example:
  - A job, as defined by a distributed operating system or scheduler.
  - A machine.
  - As determined by the info argument.
Name Publishing

• It is possible to publish multiple service names under the same port.

• However, if the publication was initially done using the info argument then MPI_PUBLISH_NAME is undefined for subsequent calls.
Name Publishing

• A service name has limited scope, as defined by the implementation.

• A port name has global scope – it is globally unique within the communication universe of the implementation.
Name Publishing

- MPI_UNPUBLISH_NAME(service_name, info, port_name)
  - (IN)service_name : service name string
  - (IN)info : system-specific arguments
  - (IN)port_name : port name string

- If the service name does not exist an error of class MPI_ERR_NAME_NOTFOUND is raised.

- If info was used to publish a name then it may be required here too.
Name Publishing

- MPI_LOOKUP_NAME(service_name, info, port_name)
  - (IN)service_name : a service name string
  - (IN)info : system-specific arguments
  - (OUT)port_name : The retrieved port name string.
- If the service name was not published then an error of class MPI_ERR_NAME is raised.
Name Publishing

• If multiple service name entries appear in the directory, then a port_name is chosen according to the implementation.

• If an info argument was used to publish the name then a similar info argument may be required here.
Singleton MPI_INIT

- The specification recommends that a high-quality implementation will make it possible to start an MPI process without the parallel application launcher.
- Any process that calls MPI_INIT should become an MPI process with an MPI_WORLD_COMM of size 1.
- The new process may then interact with other MPI processes using MPI_COMM_ACCEPT and MPI_COMM_CONNECT.
Singleton MPI_INIT

- If the implementation relies on an environment that must be started, for example, daemon processes, then a call to MPI_INIT should do one of two things:
  - Start the daemon processes if necessary or;
  - Raise an error to indicate that the singleton MPI process could not be started.
Releasing Connections

- MPI applications that are connected, become susceptible to each other's errors.
- It is desirable for a client to disconnect from a server or for a parent to disconnect from its child to avoid potential errors.
- Two processes are connected if:
  - They belong to the same communicator
  - They previously belonged to a communicator that was freed with MPI_COMM_FREE instead of MPI_COMM_DISCONNECT.
Releasing Connections

- If A is connected to B and B to C then A is also connected to C.
- Connected processes that don't share the same MPI_COMM_WORLD may become disconnected if the communication path between them is broken by MPI_COMM_DISCONNECT.
- MPI_FINALIZE is collective over a set of connected processes.
Releasing Connections

- MPI_ABORT makes a “best effort” attempt to abort only those processes specified by its comm argument but may abort all processes in MPI_COMM_WORLD as well as connected processes.
Releasing Connections

- MPI_COMM_DISCONNECT(comm)
  - (IN/OUT)comm : communicator handle
- This operation waits for all pending communication on comm to complete.
- comm is deallocated and set to MPI_COMM_NULL.
- It is collective over comm.
Releasing Connections

- MPI_COMM_FREE does the same as MPI_COMM_DISCONNECT except that it does not wait for pending communication to complete.