Connect. Accelerate. Outperform.™

Mellanox NPS-400 Demo Application

Rev 2.2

Software Version 18.0300.00

www.mellanox.com

NOTE:
THIS HARDWARE, SOFTWARE OR TEST SUITE PRODUCT ("PRODUCT(S)") AND ITS RELATED DOCUMENTATION ARE PROVIDED BY MELLANOX TECHNOLOGIES "AS-IS" WITH ALL FAULTS OF ANY KIND AND SOLELY FOR THE PURPOSE OF AIDING THE CUSTOMER IN TESTING APPLICATIONS THAT USE THE PRODUCTS IN DESIGNATED SOLUTIONS. THE CUSTOMER'S MANUFACTURING TEST ENVIRONMENT HAS NOT MET THE STANDARDS SET BY MELLANOX TECHNOLOGIES TO FULLY QUALIFY THE PRODUCT(S) AND/OR THE SYSTEM USING IT. THEREFORE, MELLANOX TECHNOLOGIES CANNOT AND DOES NOT GUARANTEE OR WARRANT THAT THE PRODUCTS WILL OPERATE WITH THE HIGHEST QUALITY. ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT ARE DISCLAIMED. IN NO EVENT SHALL MELLANOX BE LIABLE TO CUSTOMER OR ANY THIRD PARTIES FOR ANY DIRECT, INDIRECT, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES OF ANY KIND (INCLUDING, BUT NOT LIMITED TO, PAYMENT FOR PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY FROM THE USE OF THE PRODUCT(S) AND RELATED DOCUMENTATION EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Mellanox Technologies
350 Oakmead Parkway Suite 100
Sunnyvale, CA 94085
U.S.A.
www.mellanox.com
Tel: (408) 970-3400
Fax: (408) 970-3403

Table of Contents

List of Figures

List of Tables

Release Update History

Release	Date	Description
Rev 2.2	Sept. 15, 2016	Relates to EZdk version 18.0300.00.
2.1 Open	Jul. 4, 2016	Relates to EZdk version 2.1a Open.

1 Introduction

The Demo Application is a self-contained sample of a software application for the NPS-400 network processor supplied in the EZdk software development kit. The sample was designed as a reference for software developers who plan to develop both control plane and data plane software for products using the NPS-400 network processor.

The application demonstrates the following control plane features:

- Add, delete and modify of the forwarding rules contained in a hash data structure.
- Connecting to an Agent port.
- Compiling the application using the makefile.
- Initialization of the control plane application.
- Use of the EZcp API routines to configure and work with the network processor.

In the data plane, the application receives, modifies and forwards Ethernet packets from one port to another based on their source port and C-VLAN configuration. It retrieves information required for sending out the frame by performing a lookup in the Forwarding Hash data structure. The new C-VLAN and destination interface are determined by the lookup result.

1.1 Folder Structure and Contents

The Demo Application is part of the EZdk installation and is located under the \samples\demo\ folder. Its folder structure is shown below:

Table 1 - Demo Application Folder Structure

Folder	Contents
\demo_target	Demo target root directory. Used by both the target launches (demo_cp and demo_dp).
\frames	Demo target frames input.
\launches	Demo target launches. Also includes the run_demo_cp_target.sh script to run demo_cp.
\demo_cp	Control plane launch root directory. Also includes the Makefile.
\src	Source files for the control plane launch (*.c and *.h).
\demo_dp	Data plane launch root directory. Also includes the Makefile.
\src	Source files for the data plane launch (*.c and *.h).

2 Getting Started

This section describes how to operate the Demo Application.

- Compiling the Application explains how to compile the application.
- Starting the Application explains how to run the application.

2.1 Compiling the Application

This section provides detailed information on how to compile the Demo Application.

The application can be compiled for the Linux x86 platform.

2.1.1 Compiling with Makefile

Step 1. Go to the directory /samples/demo/demo_cp

Step 2. Build the demo_cp application using the command:

```
EZDK_BASE={base EZdk path} EZDK_PLATFORM=linux_x86_64 make
```

2.2 Starting the Application

This section describes various methods for running the Demo Application.

2.2.1 Launch from Linux Shell on a Real Chip

2.2.1.1 Using demo_cp as the Control Plane

Step 1. Open a shell on your Linux system.

Step 2. Run the ./samples/demo/demo_target/launches/run_demo_cp_target_rc.sh script using the following command line:

```
./samples/demo/demo_target/launches/run_demo_cp_target_rc.sh .  [base EZdk path]
[sample_dp][host_ip][chip ip]
```

Step 3. The script will start logging to screen the action it performs and open 3 new terminal windows:

- demo_cp – running the driver – do not close this windows until finish testing
- HOST – a terminal with ssh connection to the host
- CHIP – a terminal with telnet connection to the chip

Step 4. In the CHIP terminal, the sample bin file is located under the /tmp folder. The sample is not automatically run and the user needs to manually run it, e.g. /tmp/demo_dp.

3 Sample Input and Configuration

3.1 Core Configuration

The core is configured with:

- 640 bytes of private CMEM
- 256 bytes of shared CMEM
- 256 bytes of caches
- 1K of stack
- Two FMT slots (slots #5 and #11) mapped to x1 cluster code and x16 cluster code, accordingly.

3.2 Control Plane Configuration

The sample configures two ports and transmits 48 frames which enter port 0 on side 0 and exit from port 0 on side 1.

The configuration is defined by the control plane application.

3.3 Input Frames

When working with the real chip, frames should be sent by a traffic generator.

The input frames are represented in *.pcap files, one file per port, located in the /frames directory.

frames_in_side0_eng0_10GE_00.pcap

The file contains 48 frames for side 0 engine 0 10GE interface number 0.
24 frames in the file are simple UDP packets of IPv4 Ethernet type (untagged packet).
Another 24 frames in the file are TCP packets of different sizes with a C-VLAN tag.

Input frame files supplied use the naming convention:
frames_in_side<side number>_eng<engine number>_<interface type>_<interface number>
where
chip <side number> is 0 or 1
interface <engine number> is 0-3
<interface type> is 10GE, 40GE or 100GE
<interface number> is a two digit value representing the interface number 0-11.

3.4 Control Plane Logging

The logging system can be available by calling the following EZenv library API sets:

Step 1. Configure the variable that will hold the file name: **EZc8 acFullFileName[1024 + 256];**

Step 2. Call **EZlog_SetFileName(acFullFileName);**

Step 3. Call **EZlog_OpenLogFile();**

Step 4. Call **EZlog_SetForceFlush(TRUE);**

Step 5. Set all the logs you want to open through:
EZlog_SetLog(EZlog_OUTPUT_FILE, EZlog_COMP_ALL, EZlog_SUB_COMP_LOG_ALL, EZlog_LEVEL_DEBUG);

3.5 Formats

3.5.1 Lookup Table Definition

The Forwarding Hash structure serves for retrieving the transmit information required for forwarding the incoming frame.

The key for the lookup table is constructed from the input port's logical ID and C-VLAN ID. The result of the lookup contains the C-VLAN ID for the outbound frame and 16 bytes of Tx Info used for writing to the job descriptor when transmitting the frame to the TM.

Table 2 - Forwarding Hash Result Format

Name	Byte Offset	Note
Control	0	The first 4 bits are for lookup control. The rest of the bits are reserved (ignored).
C-VLAN	2..3	The first 12 bits are the C-VLAN ID of the outbound frame. The rest of the bits are reserved (ignored).
Destination	4..7	20 bits of flow ID (bits 0..18) and side (bit 19)

3.5.2 CMEM Data

This section describes the structure of the CMEM (core local memory), as used in the application. The structure was specifically designed to match the application's behavior and requirements.

Table 3 - CMEM Format

Name	Size in Bytes	Note
Frame	84	The frame data structure.
Frame data	256	256 byte data buffer working area – used to load first buffer data for processing and modification.
Decode result	16	The result of the MAC addresses decoding and parsing.
Lookup key	4	The key for the Forwarding Hash lookup (input port's logical ID and C-VLAN ID).
Work area	144	Working area for the Forwarding Hash lookup.

Table 4 - Shared CMEM Format

Name		Note
Forward hash struct descriptor	20	Hash structure descriptor for forwarding hash database

4 Control Plane Details

4.1 Supported Platform

The control plane includes support for the Linux x86 platform.

The control plane may be run on an NPS evaluation board.

4.2 Implementation

The control plane is divided into the following main parts:

Table 5 - Main Parts of the Control Plane

Part	Description
Environment Initialization	Initialization of the NPS host environment, including the various software components in the host.
NPS Initialization	Creation and initialization of the NPS channel, including definition of search structures and loading of the search memory partition.
Operation Execution	Execution of the Control Plane sample operations based on the user selection in the operations menu.

The following sections provide more details on each of these topics.

4.2.1 Environment Initialization

The first portion of the application provide a reference for initial/common portions of the host application, such as the initialization and shutdown sequence of the control plane SDK host components.

init_board( ): Initialize the EZdev SDK host components to work with an NPS evaluation board.

init_cp( ): Initialize the control plane SDK host components.

delete_board( ): Delete the EZdev SDK host component.

delete_cp( ): Delete the control plane SDK host components.

In many cases, this reference implementation can be used as is for customer host applications.

4.2.2 NPS Initialization

Once the environment initialization has been performed, the application demonstrates the creation and initialization of the NPS channel, including definition of search structures and loading of the search memory partition.

This is done by invoking the function **setup_chip()**.

4.2.3 Operation Execution

The final portion of the application displays the control plane operations menu (Figure 1) and performs the execution of the various operations based on the developer's selection.

The scenarios are executed via a CLI menu (see routine **cp_menu_CLI()** located in /demo_cp/src/cli.h). The operation menu is displayed, and the operation start functions are invoked based on the developer's selection. Each operation opens a new CLI menu with the desired operation.

Figure 1: Integrated Control Plane Operations Menu

```
**********************************************
* Menu                                       *
**********************************************

Please enter what would you like to do


0. Exit.

1. Add rule.

2. Delete rule.

3. Update rule.

4. Show all rules.

5. Enable/disable AGT debug agent interface.
```

Table 6 - Operations Available in the Integrated Control Plane

Menu Value	Scenario Name	Operation Description
0	Exit	Exit the application.
1	Add Rule (Operation #1)	Add a new forwarding rule to the hash structure.
2	Delete Rule (Operation #2)	Delete an existing forwarding rule from the hash structure.
3	Update Rule (Operation #3)	Update an existing forwarding rule in the hash structure.
4	Show All Rules (Operation #4)	Show all rules configured in the hash structure.
5	Enable/Disable AGT Debug Agent Interface (Operation #5)	Enable/disable agent interface with a specific port number.

To run an operation enter the number of the desired scenario on the keyboard followed by the Enter key (e.g. press "1 + Enter" to begin the Add rule operation).

Wait for the operations to run and display on screen.

To terminate the Control Plane sample press "0+ Enter".

The following sections provide detailed information on each of the supported operation.

Screenshots are shown in Appendix A: Screenshots from the Control Plane.

4.2.3.1 Add Rule (Operation #1)

The Add Rule operation demonstrates how a new forwarding rule is added to the hash structure using the EZcp library APIs.

Operation Flow

- The developer will enter the source port of the packet.
- The developer will enter the source VLAN of the packet.
- The developer will enter the destination port of the packet.
- The developer will enter the destination VLAN of the packet.
- A new rule will be added to the hash structure according to the input parameters.

Screenshot is shown in Figure 4,“Add a New Forwarding Rule Screen”.

Operation Code

- Read source port entered by the developer.
- Read source VLAN entered by the developer.
- Read destination port entered by the developer.
- Read destination VLAN entered by the developer.
- Build the search_key and search_result for the entry to insert into the hash structure.
- Add the entry to hash structure using the API routine EZapiStruct_AddEntry().
- Print the added rule to the screen.

4.2.3.2 Delete Rule (Operation #2)

The Delete Rule operation demonstrates how to delete existing forwarding rule using the Control Plane library APIs.

Operation Flow

- The developer will enter the source port of the packet.
- The developer will enter the source VLAN of the packet.
- The developer will enter the destination port of the packet.
- The developer will enter the destination VLAN of the packet.
- The existing rule will be deleted from the hash structure according to the input parameters.

Screenshot is shown in Figure 5,“Delete a Forwarding Rule Screen”.

Operation Code

- Read source port entered by the developer.
- Read source VLAN entered by the developer.
- Read destination port entered by the developer.
- Read destination VLAN entered by the developer.
- Build the search_key and search_result for the entry to remove from the hash structure.
- Delete the entry from hash structure using the API routine EZapiStruct_DeleteEntry().
- Print the deleted rule.

4.2.3.3 Update Rule (Operation #3)

The Update Rule operation demonstrates how to update an existing forwarding rule with a new rule using the EZcp library APIs.

Operation Flow

- The developer will enter the source port of the packet.
- The developer will enter the source VLAN of the packet.
- The developer will enter the new destination port of the packet.
- The developer will enter the new destination VLAN of the packet.
- An existing rule will be updated in the hash structure according to the input parameters.

Screenshot is shown in Figure 6,“Update a Forwarding Rule Screen”.

Operation Code

- Read source port entered by the developer.
- Read source VLAN entered by the developer.
- Read new destination port entered by the developer.
- Read new destination VLAN entered by the developer.
- Build the search_key and search_result for the entry to update the hash structure.
- Update the entry from hash structure using the API routine EZapiStruct_UpdateEntry().
- Print the updated rule.

4.2.3.4 Show All Rules (Operation #4)

The Show All Rules operation demonstrates how to show all forwarding rules configured in the hash structure using the EZcp APIs. Initially the hash structure is configured with no rules.

Operation Code

- Iterate over all the entries in the hash structure using the API routine EZapiStruct_Iterate().
- Print all the rules to the screen.

Screenshot is shown in Figure 7,“Show All Forwarding Rules Screen”.

4.2.3.5 Enable/Disable AGT Debug Agent Interface (Operation #5)

The Enable/disable AGT debug agent interface operation demonstrates how to open a socket to the agent interface with a specific port using the EZagt Agent group APIs. This selection may also be used to close such a socket once it has been opened.

- Operation Flow
- The developer will enter the port of the Agent.
- A new socket will be opened for listening in the specific port.

Screenshot is shown in Figure 8,“Enable/disable AGT Debug Agent Interface Screen”.

Operation Code

- Read agent port from user.
- Create a new server with the specific port to listen using the API routine EZagtRPC_CreateServer().
- Register standard control plane commands on the specific port using the API routine EZagt_RegisterStandardCmdFunctions().
- Create the task for run rpc-json commands using the API routine EZosTask_Spawn().
- Print the agent port opened to the screen.

5 Data Plane Details

5.1 Operation

The main flow of the data plane is as follows (see Figure 2):

Step 1. Receive a new frame.

Step 2. Load current frame buffer to CMEM (after frame receive, the user can assume current buffer = first).

Step 3. Parse and decode the Ethernet header (Layer 2 header) to ensure validity of the Layer 2 data.

Step 4. Build a lookup key with the source port's logical ID and C-VLAN ID. (If the Ethernet packet is untagged, the default VLAN ID is used.)

Step 5. Perform a lookup in the Forwarding Hash search structure.

Step 6. If C-VLAN already exists:

Step a. Update the C-VLAN data based on the result of the lookup.

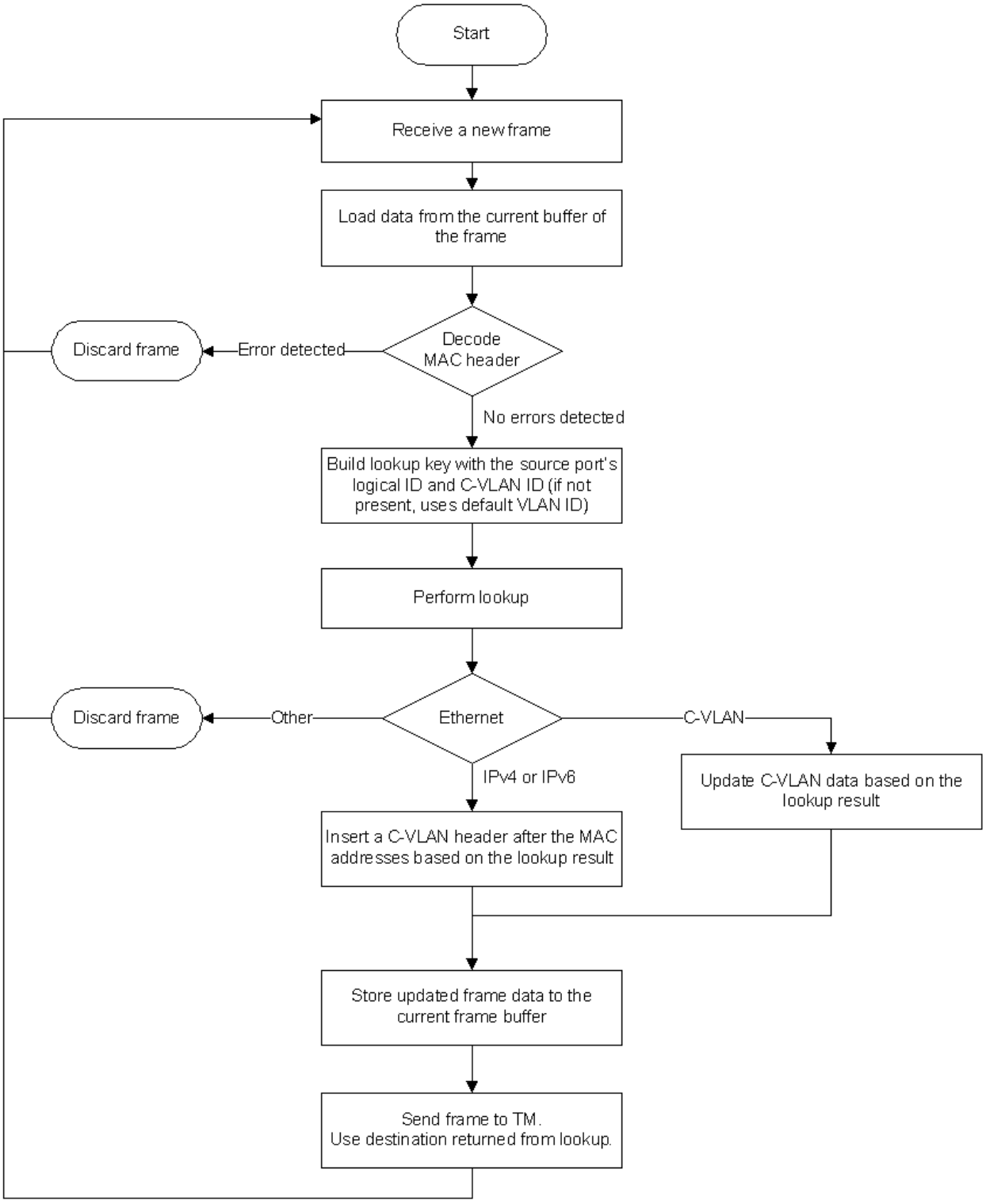
Step 7. Else

Step a. Add C-VLAN to the packet, while the VLAN ID is based on the lookup result.

Step 8. Store the updated buffer data in CMEM back to the current frame buffer.

Step 9. Send the frame to the TM. Use the destination received from lookup.
Use MAC decode hash to select the link for LAG.

Figure 2: Flow Chart



6 Appendix A: Screenshots from the Control Plane

Provided below are sample screen shots when running the integrated control plane.

Figure 3: Integrated Control Plane Operations Menu

```
**********************************************
* Menu                                       *
**********************************************

Please enter what would you like to do


0. Exit.

1. Add rule.

2. Delete rule.

3. Update rule.

4. Show all rules.

5. Enable/disable AGT debug agent interface.
```

Figure 4: Add a New Forwarding Rule Screen

```
**********************************************************
* Rule selection 0.                                      *
* (To Quit press 'q' any time)                            *
**********************************************************

Please enter the source port

1

Please enter the source vlan ID.

1

Please enter the destination port.

1

Please enter the destination vlan ID.

1

Rule added with
   source port: 1,
   vlan: 1,
   destination port: 1,
   vlan: 1.
```

Figure 5: Delete a Forwarding Rule Screen

```
**********************************************************
* Delete rule                                            *
* (To Quit press 'q' any time)                           *
**********************************************************

Please enter the source port

2

Please enter the source vlan ID.

2

Please enter the destination port.

3

Please enter the destination vlan ID.

3
```

Figure 6: Update a Forwarding Rule Screen

```
**********************************************************
* Update rule                                            *
* (To Quit press 'q' any time)                           *
**********************************************************

Please enter the source port

1

Please enter the source vlan ID.

1

Please enter the new destination port.

2

Please enter the new destination vlan ID.

3

Rule update with
   source port: 1,
   vlan: 1,
   destination port: 2,
   vlan: 3.
```

Figure 7: Show All Forwarding Rules Screen

```
************************************************************
* All Rules:                                               *
************************************************************

Rule
   source port: 1,
   vlan: 1,
   destination port: 2,
   vlan: 3.
```

Figure 8: Enable/disable AGT Debug Agent Interface Screen

```
************************************************************
* Agent selection                                          *
* (To Quit press 'q' any time)                             *
************************************************************

Please enter the Agent port

1234

Agent created on port 1234
```

7 Appendix B: Data Plane Command Line

The data plane command line is as follows:

```
data_plane_demo_app –run_cpus <cpus_string>
```

Option	Description
-run_cpus <cpus_string>	Specifies the processors to run the application on. <cpu_string> is a comma-delimited list of separate processors, or a range of processors. Usage example: 0,5,7-9 - use cpus: 0,5,7,8,9 Default value: If a “run_cpus” argument is not supplied, the application is run only on processor 16. Note: The father application spawns the needed child processes and passes each child the processor ID it should run on. Each child process then binds itself to the matching processor, and obtains its own private resources and mappings. After the work processes creation, the father process waits indefinitely. When the father process receives a shutdown signal, it kills all the child processes and terminates the application. If only one processor is passed, the application binds itself to that processor without creating any child processes.