Copyright © 2000 Scyld Computing Corporation.

Copyright © 1999, 2000 Erik Hendriks < hendriks@scyld.com >

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided also that the entire resulting derived work is clearly marked marked as a modified version and distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions.

Scyld Beowulf and the Scyld logo are trademarks of Scyld Computing Corporation.

All other trademarks and copyrights referred to are the property of their respective owners.

September, 2000.
Printed in USA.

Scyld Computing Corporation
410 Severn Avenue, Suite 210
Annapolis, MD 21403
http://www.scyld.com
# Table of Contents

Introduction ......................................................... 1

1 Scyld Beowulf System Overview ....................... 2

2 Configuring the Scyld Beowulf Cluster: BeoSetup ........... 5
   2.1 Main Window.................................................. 5
      2.1.1 Apply and Revert buttons......................... 5
      2.1.2 Short Cuts........................................... 5
      2.1.3 Pop-up Menus........................................ 5
   2.2 Menu Items.................................................. 5
      2.2.1 File Menu.............................................. 6
      2.2.2 Settings Menu....................................... 6
         2.2.2.1 Preferences.................................. 6
         2.2.2.2 PCI Table.................................. 7

3 Booting the Slave Nodes: BeoBoot ............... 8
   3.1 Overview..................................................... 8
   3.2 Boot Images................................................. 9
      3.2.1 Remaking Boot Images............................ 9
   3.3 File Systems............................................... 9
   3.4 Program Usage............................................. 9
      3.4.1 beoboot............................................... 10
      3.4.2 beoboot-install.................................... 11
      3.4.3 beoserv............................................. 11
   3.5 Adding A New Network Driver.................. 12
   3.6 Technical Details......................................... 12
      3.6.1 Boot Phases.......................................... 12
         3.6.1.1 Phase 1 - Initial (Floppy) Boot............ 12
         3.6.1.2 Phase 2 - Final Kernel - initrd.......... 12
         3.6.1.3 Phase 3 - Final Kernel - ramdisk Root... 13

4 The Scyld Beowulf Distributed Process Space: BProc .......... 14
   4.1 Installing BProc......................................... 14
      4.1.1 Installation from RPMs............................ 14
      4.1.2 Building BProc from Scratch.................... 14
      4.1.3 Installing........................................... 14
   4.2 Running BProc............................................. 15
      4.2.1 Node States......................................... 15
   4.3 VMADump.................................................... 15
4.3.1 Limitations / Important Details .......................... 16

4.4 Program Usage .................................................. 17
  4.4.1 bpmaster .................................................. 17
  4.4.2 bpslave .................................................. 17
  4.4.3 bpstat .................................................. 18
  4.4.4 bptcl .................................................. 18
  4.4.5 bps .................................................. 19
  4.4.6 bpcp .................................................. 19
  4.4.7 vmadlib ................................................... 19

4.5 BProc Programmer’s Guide ......................................... 21
  4.5.1 Process Migration With BProc ......................... 21
  4.5.2 C Library Interface ........................................ 21
    4.5.2.1 Machine Information Calls ....................... 21
    4.5.2.2 Process Migration Calls ....................... 22
    4.5.2.3 System Management Calls ................... 23

5 Scyld Beowulf Message Passing Interface: BeoMPI ............... 24
  5.1 Overview .................................................. 24
    5.1.1 History of MPI ........................................ 24
    5.1.2 Compatibility with BeoMPI ......................... 24
  5.2 Installing BeoMPI ........................................ 24
    5.2.1 Installation from RPMs ............................. 25
    5.2.2 Building BeoMPI from Scratch ................... 25
    5.2.3 Installing ........................................ 25
    5.2.4 Co-existing with other MPI Implementations ....... 25
  5.3 Running BeoMPI ........................................ 25
    5.3.1 Program Placement .................................. 25
  5.4 mpirun .................................................. 26
    5.4.1 Invoking mpirun .................................. 26
    5.4.2 Inline-mpirun ................................... 27
    5.4.3 Invoking mpirun from Inside an Application .... 27
  5.5 BeoMPI Programmer’s Guide ................................ 29
    5.5.1 Compiling with beompi ............................. 29
    5.5.2 Scheduler Hinting with beompi ................... 29
  5.6 Troubleshooting ........................................ 30
    5.6.1 Troubleshooting with strace ....................... 30
    5.6.2 Troubleshooting mpirun ............................ 30
    5.6.3 Troubleshooting MPI applications ................ 31
6 Intel PPro Performance Counter Support 32
   6.1 C Language Interface ................................. 32
      6.1.1 Return value .................................. 33
      6.1.2 System Calls .................................. 34
   6.2 Counter Configuration ............................... 34
      6.2.1 Countable Events ................................. 34
         6.2.1.1 Data Cache Unit (DCU) ................. 34
         6.2.1.2 Instruction Fetch Unit (IFU) .............. 34
         6.2.1.3 L2 Cache .................................. 35
         6.2.1.4 External Bus Logic ....................... 35
         6.2.1.5 Floating Point Unit ....................... 36
         6.2.1.6 Memory Ordering ......................... 37
         6.2.1.7 Instruction Decoding and Retirement ... 37
         6.2.1.8 Interrupts ................................ 37
         6.2.1.9 Branches .................................. 37
         6.2.1.10 Stalls ................................... 38
         6.2.1.11 Segment Register Loads ................. 38
         6.2.1.12 Clocks .................................. 38
      6.2.2 Counter Flags .................................. 38
   6.3 Known Issues and Bugs ............................... 39
   6.4 More Information .................................. 39

7 Monitoring the Status of the Scyld Beowulf
Cluster: BeoStatus .......................... 40
   7.1 Overview of Options ................................. 40
   7.2 Communication Methods ......................... 40
   7.3 Presentation Modes ................................. 41

Appendix A Scyld Beowulf Configuration
File Reference ................................. 42

Appendix B Scyld Beowulf Boot Configuration
File Reference ................................. 44

Appendix C References ............................... 46

Appendix D GNU General Public License .... 47

Index ............................................. 53
Introduction

This release of the Scyld Beowulf Scalable Computing Distribution contains all the software required for configuring, administering, running and maintaining a Beowulf cluster.

Advances provided by Scyld Beowulf include:
- A single system image provided by the Beowulf distributed process space system (BProc)
- The Beowulf MPI (Message Passing Interface) implementation, optimized for Scyld Beowulf
- A GUI interface for configuring a Beowulf cluster
- Automatic, remote installation of compute nodes
- A GUI interface for monitoring a Beowulf cluster
- Seamless incorporation of additional compute nodes into a running cluster
- Support for multiple, independent front end machines

Scyld Beowulf Overview
For an overview of the main portions of the Scyld Beowulf Scalable Computing Distribution, see Chapter 1 [Scyld Beowulf System Overview], page 2. Additionally, use the Table of Contents and the Index of this manual to find other information of interest.

Hardware Recommendations
Hardware recommendations for building a Scyld Beowulf are contained in Chapter 1 [Scyld Beowulf System Overview], page 2.

Starting the Installation
To launch the "quick start" installation, boot the cluster’s front end machine from the Scyld CD-ROM. See section “Quick Start” in Scyld Beowulf Installation Guide. Alternatively, install Scyld Beowulf from RPM packages. See section “Scyld Beowulf Installation (x86) from RPMs” in Scyld Beowulf Installation Guide.
Chapter 1: Scyld Beowulf System Overview

1 Scyld Beowulf System Overview

The Scyld Beowulf Scalable Computing Distribution streamlines the processes of configuring, administering, running and maintaining a Beowulf cluster computer. This software may be used with a group of off-the-shelf computers connected through a private network to build a Beowulf cluster computer. The front end "master" computer in the cluster distributes computing tasks to the other machines, the slave nodes, in this parallel architecture supercomputer.

For any Beowulf system, hardware selection is based upon the price/performance. Scyld recommends the following hardware for use with this release of the Scyld Beowulf Scalable Computing Distribution:

- **Processors**: Intel Pentium III™, Intel Celeron™, AMD Athlon™ or Alpha AXP™
- **Architecture**: one, two or four processors per motherboard
- **Physical Memory**: 128 MBytes or more (min. 32 MBytes)
- **Operating System**: Scyld Beowulf Scalable Computing Distribution
- **Network Cards**: Fast or Gigabit Ethernet PCI Adapters with existing Linux driver support (see http://www.scyld.com/network for list)
- **Network Hub**: Fast or Gigabit Ethernet switch
- **Drives**: IDE (up to ATA/66) hard drive and floppy drive for each node

- Requires the Scyld Beowulf Scalable Computing Distribution for AXP.

A Beowulf cluster is connected by a private internal network. The front end machine must have two network interfaces, one connected to the cluster internal network and one connecting to the outside network. Each of the slave nodes requires a network interface connected to the cluster network.
Assemble the cluster as shown:

---

| External -------> | Switch | <------------------- |
| Network | ------------ <--- | |
| <-------- | ^ | |
| | | | |
| ----------- ----------- ----------- ----------- |
| | Master | | Slave | | Slave | | Slave | |
| | Node | | Node 0 | | Node 1 | | Node N-1 | |
| (front | | | | | | ... | |
| end) | | | | | | | |
| | | | | | | | |
| ----------- ----------- ----------- ----------- |

The full Scyld Beowulf Scalable Computing Distribution is installed only onto the front end machine. A graphical user interface is available and included to further streamline the cluster configuration processes.

The Beowulf configuration file is created on the front end machine which specifies the master’s cluster interface and a range of IP number assignments for the nodes. The slave configuration file is also created on the front end machine detailing the file system configuration for the slave nodes. A slave node boot image and complete operational kernel image for the slave nodes are created and stored on the front end machine. This completes the installation on the front end machine.

For the slave nodes, N identical slave node boot floppy disks (one for each of the N nodes) must be created containing the slave node boot image. These floppy disks must be inserted into each of the nodes before the nodes are powered up, one at a time. As a node becomes active on the network, the assigned IP address appears in the master configuration file as an "unknown" computer on the network. You may then select which entries to include in the cluster. You may also choose to partition the slave node hard disks over the network. Finally, the complete operational kernel image is downloaded to the slave node over the cluster network.

Software to implement the Beowulf Distributed Process Space (BProc) is an integral part of the Scyld Beowulf Scalable Computing Distribution. This allows you to start processes on slave nodes in the cluster and track them in the process table on the front end machine. It also provides process migration mechanisms to help in the creation of remote processes. This removes the need for most binaries on the remote nodes.

Turn to the next page for a brief description of the major portions of the Scyld Beowulf Scalable Computing Distribution.
A brief description of the major portions of the Scyld Beowulf Scalable Computing Distribution is given below:

**BeoSetup**  A GUI interface for configuring the cluster

**beofdisk**  A utility for remote partitioning of the slave node hard disks

**BeoBoot**  A set of utilities for booting the slave nodes

**beoserv**  The BeoBoot boot server: it responds to slave nodes and serves the boot image

**BProc**  The Beowulf distributed process space: the integral part of Scyld Beowulf

**bpmaster**  The BProc master daemon: it runs on the front end machine

**bpslave**  The BProc slave daemon: it runs on the slave nodes

**bpstat**  A part of BProc: it displays various pieces of status information

**bpctl**  A part of BProc: it is used to apply commands to referenced slave nodes

**bpsh**  A part of BProc: it is an rsh replacement

**bpcp**  A part of BProc: it copies files between machines

**BeoMPI**  The Message Passing Interface, optimized for Scyld Beowulf

**BeoStatus**  A GUI interface for monitoring the status of the Scyld Beowulf cluster

Refer to the Table of Contents and Index of this manual to obtain detailed information for each of these components.
2 Configuring the Scyld Beowulf Cluster: BeoSetup

The beosetup program is a graphical front-end for controlling a Beowulf cluster using the BPproc system. It is intended to be used by the cluster system administrator; configuration file write permission is required for most actions.

2.1 Main Window

The main window contains three lists of Ethernet hardware addresses. The first list contains unknown addresses, those not yet assigned to either of the other two lists. The second list contains nodes that are to be active in the cluster. They are ordered by node number (ID). The third list contains nodes or other machines that are to be ignored, even though they produce RARP (reverse address request protocol) requests.

Addresses may be moved between lists by dragging an address with the left (first) mouse button or by right (third button) clicking on the address with the mouse and choosing the appropriate pop-up menu item.

2.1.1 Apply and Revert buttons

After moving addresses between lists, the Apply button must be clicked for changes to take effect. Clicking on the Apply button saves the changes to the configuration file and signals the Beowulf daemons to re-read the configuration file.

Revert will re-read the existing Beowulf configuration file. This has the effect of undoing any undesired changes that have not yet been applied or synchronizing beosetup with any changes that have been made to the configuration file by an external editor.

2.1.2 Short Cuts

Next to the Apply and Revert buttons are two short-cut buttons for generating a Node Floppy ("slave node boot floppy") and setting Preferences. These items are also accessed through the File Menu and Settings Menu, respectively.

2.1.3 Pop-up Menus

Each list has a pop-up menu associated with it that can be accessed by right clicking on a list item. Insert a new address by choosing Insert from the pop-up menu on the active node (middle) list. Delete (forget about) addresses by selecting Delete in the pop-up menu on the active node list.

Any active hardware address may be edited by choosing Edit from the pop-up menu.

2.2 Menu Items

This section explains the functionality of the menu items in beosetup.
2.2.1 File Menu

*Boot Configuration File* and *Configuration File* allow non-default filenames to be used for the output configuration files. The *boot configuration file* is used for the *beoboot* floppy. The *configuration file* must be the same one that the *beoserv* daemon is currently reading, for the Beowulf Server software to work properly with *beosetup*.

*Create Node Boot Floppy* creates a *beoboot* floppy disk (or image) for booting a node in the cluster. *Create BeoBoot file* creates the network boot file, which is downloaded from the server to each node during node boot. This *beoboot* file contains the kernel image, kernel flags, and ramdisk image that start each node.

*Exit* will quit the *beosetup* program (not the *beoserv* daemon).

2.2.2 Settings Menu

Choosing *Preferences* from the *Settings* menu brings up the beosetup configuration dialog box. *PCI Table* brings up the PCI table dialog. *Restart Daemons* sends a signal to the Beowulf daemons to re-read the configuration file. It doesn’t actually kill the daemons.

2.2.2.1 Preferences

The first tab of the Configuration dialog box contains network configuration items that appear in the configuration file:

- **Interface**: specifies the internal network interface of the Beowulf server (connection to the rest of the cluster).
- **Ports**: specifies the TCP/IP network socket port number used to connect with the nodes – current convention is 1555, and the Bproc network socket port number – current convention is 2223.
- **Boot File**: specifies the Beowulf boot file. Note that this is not a plain Linux kernel, but a combination of a Linux kernel and a ramdisk created using the Beowulf tools.
- **IP Address Range**: specifies the range of TCP/IP addresses available for the cluster nodes. The maximum number of nodes is defined by this (inclusive) range.

The second tab accesses the settings for the following GUI options:

- **Automatically apply all changes**: If ‘on’, automatically apply all address changes in the main window (without having to click on the *Apply* button in the main window).
- **Automatic new node assignment**: Off by default; automatically assigns new nodes to the bottom of either the configured node list or the ignored node list.

The third tab contains file system options for the later stages of booting. During a normal boot, the server will attempt to configure the filesystems on the node by running some combination of a filesystem check and a filesystem create. The radio buttons in this tab determine the default global policy:
Safe filesystem check
   gives up if it encounters bad errors.

Full filesystem check
   will try to answer \texttt{y} to all the \texttt{should I fix?} questions.

If check fails
   indicates that it’s OK to re-create a blank filesystem if the filesystem check fails.

Always make filesystem
   re-creates the filesystem on every boot (the filesystem check will be skipped and thus that selection is greyed).

2.2.2.2 PCI Table

The PCI Table dialog is used to add PCI vendor/device/driver entries to the boot configuration file. Use it when you know that a new version of an old card is supported by a certain driver, but is not in the Beowulf PCI table (thus not getting recognized and loaded properly).
3 Booting the Slave Nodes: BeoBoot

BeoBoot is a set of utilities to simplify booting of slave nodes in a Beowulf cluster. BeoBoot generates initial boot images which allow a slave node to boot and download its kernel over the network, from the cluster master node.

BeoBoot:
- Effectively allows Linux to netboot from any network hardware that Linux supports.
- Uses a ‘Two Kernel Monte’ to start the downloaded kernel from memory.
- Allows safe kernel testing and upgrades of kernels on clusters.
- Provides scripts for easy start/stop/restart of Beowulf services on the master node.
- Provides utilities for creating all the boot images needed to boot cluster nodes.
- Provides node setup scripts to automatically configure and setup nodes.
- Utilizes BProc at all phases of cluster node setup.

3.1 Overview

BeoBoot is a collection of programs and scripts which allows easy booting of slave nodes in the Scyld Beowulf cluster. On the master node, there is a boot server daemon and a collection of scripts for setting up slave nodes.

The following events occur while booting a slave node with BeoBoot.
1. The node loads the BeoBoot initial image from the designated boot medium (the local floppy drive OR the BeoBoot partition on the slave node hard disk OR the CD-ROM).
2. The node (running the BeoBoot initial image) scans the PCI bus to auto-detect network hardware and install network drivers.
3. The node sends out RARP requests on all detected interfaces.
4. The node receives a RARP response on one of those interfaces. Using that interface, the node contacts the machine (the master) which responded to the RARP request to get the final kernel and ramdisk.
5. The node loads the new kernel and ramdisk image into memory. Using this ramdisk image, the node reboots. This is all done with a ‘Two Kernel Monte’ which means that nothing is written out to permanent storage during this process.
   Note that using this process to load a second kernel allows safe experimentation with new kernel images.
6. After the new kernel starts up, the node repeats the network driver detection and RARP steps.
7. The node now contacts the front end to become a BProc slave node.
8. On the master node, a new slave connection is received. The master daemon runs the node setup script on the master node.
9. The setup script performs the operations necessary to finish configuring the node. This includes configuring additional network interfaces, installing additional modules, mounting file systems (including the root file system), copying over files, etc.
10. When that script completes successfully, the node is finally tagged as up on the master node and it is available to users. If any of the slave node hard disks have not been previously partitioned, they will remain in the unavailable state. To remotely partition any of the slave node hard disks use beofdisk. See section “Disk Partitioning” in Scyld Beowulf Installation Guide.

3.2 Boot Images

There are two sets of boot images involved in booting a slave node with BeoBoot. The first set is copied onto the slave node boot floppy disk and into the BeoBoot partition of the slave node hard disk, if using the Scyld default partitioning scheme, see section “Disk Partitioning” in Scyld Beowulf Installation Guide. These are known as the phase 1 or initial images, composed of a minimal kernel image and an initial ramdisk image. These are generated from kernels and modules that are included with the Scyld Beowulf BeoBoot distribution. To add a network driver to a slave node boot floppy image, you must compile the driver against the kernel headers which match the BeoBoot kernels. See Section 3.5 [Adding A New Network Driver], page 12.

The second boot image contains the final kernel and modules that the slave node will use. This image is usually generated from the kernel images that the master node is running.

3.2.1 Remaking Boot Images

You should never have to regenerate the BeoBoot initial image unless you make some kind of hardware change to the cluster or you have some other kind of problem which forces you to make a change.

The second phase boot image should be updated whenever you upgrade the kernel or any other modules on the front end. Running the same kernel on the master node and the slave nodes is highly recommended.

The second phase boot image should also be updated if you upgrade your C library or any of the other shared libraries residing in ‘/lib’. If these libraries, which exist on both the master and slave nodes, do not match, you may have problems the next time the slave nodes boot. Re-run beoboot -2 -k kernel image on the master node. Then, re-boot the slave nodes from the BeoBoot partition on the hard drive OR the original slave node boot floppies (leaving these floppies in the floppy drives is recommended to simplify rebooting) using bpctl -S all -s reboot on the master node. BeoBoot will download the new libraries during the network boot portion of the re-boot.

3.3 File Systems

The file system table for slave nodes is stored in ‘/etc/beowulf/fstab’.

3.4 Program Usage

This section contains basic usage information for the binaries that are included with Beo-Boot.
3.4.1 beoboot

beoboot [ -o outputfile ] -1
beoboot [ -o outputfile ] -2 [ -k kernelimage ] [ -c commandline ]

beoboot generates Beowulf boot images. There are two sets of images: phase 1 and phase 2. Phase 1 images are placed on the hard disk or a floppy disk and are used to boot the machine. The phase 2 image is downloaded from the cluster front end by the phase 1 image. The phase 2 image is placed on the front end in a place where beoserv can find it.

In the -2 mode, beoboot will detect the version of the kernel given as its argument and look for the matching modules in `/lib/modules/kernelversion`

Options:

'-h' Display a help message and exit.

'-v' Display version information and exit.

'-1' Create a phase 1 (initial) boot image.

'-i' Create phase 1 kernel and ramdisk images.

'-2' Create a phase 2 boot image (see Options for phase 2, below).

'-o output_file' Write the output to output_file.

'-r dir' Use dir as the root directory (somewhat like chroot).

'-L dir' Find beoboot files and programs in dir instead of the default location (`/usr/lib/beoboot`).

Options for phase 2:

'-k kernelimage' Create a phase 2 boot image using kernelimage instead of the image given in the configuration file.

'-c cmdline' Use cmdline instead of the commandline given in the configuration file.

'-m dir' Look for modules matching the kernel image in dir instead of the default, which is `/lib/modules/kernelversion`. 
3.4.2 beoboot-install

beoboot-install -h
beoboot-install -v
beoboot-install node device
beoboot-install -a device

beoboot-install installs the beoboot initial slave node boot image onto the hard disk of a cluster node. This will allow booting the node without using a slave node boot floppy disk or CD-ROM.

Options:

`-h` Display a help message and exit.
`-v` Display version information and exit.
`-a device` Install on all nodes device hard disk instead of a particular node. device = hda, hdb, ..., sda, sdb, ...

Requirement: a small partition (minimum 2MB) must be set aside for beoboot on the hard disk. This partition should be tagged as type 89. And, this partition should exist near the beginning of the disk to avoid problems with large disks. See section “Disk Partitioning” in Scyld Beowulf Installation Guide.

3.4.3 beoserv

beoserv -h
beoserv -v
beoserv [-f file] [-n file]

beoserv is the BeoBoot boot server. It responds to RARP requests from slave nodes in a cluster and also serves a boot image (via TCP) to the nodes.

Options:

`-h` Display a help message and exit.
`-v` Display version information and exit.
`-f file` Read configuration from file instead of the default (`/etc/beowulf/config`).
`-n file` Write new slave node addresses to file instead of the default (`/var/beowulf/unknown_addresses`).

Configuration information is normally read from `/etc/beowulf/config`. Beoserv will listen on the interface specified by the interface line. The range of IP addresses for assignment to slave nodes are defined in the iprange directive. Beoserv will respond to addresses given on the node lines. IP addresses are assigned to slave nodes in the order that these node lines appear in the configuration file.

The server will ignore requests from addresses that are listed on ignore lines.

When a request comes in from an unknown address, the server will append an unknown line to the configuration file. This allows the setup tools to see new nodes as they appear on the network.
Sending a HUP signal to the daemon will cause it to re-read its configuration file, thus implementing any updates to the file.

### 3.5 Adding A New Network Driver

It is possible to build the BeoBoot kernel (and generate the slave node boot floppy) for hardware which is not supported by the BeoBoot system as shipped. You must have the driver for the hardware (This section does not include instructions on how to build kernel modules).

The Linux kernel include files to build against are located in ‘/usr/lib/beosboot/include’. Use ‘/usr/lib/beosboot’ as the location of the Linux source.

After building the module, place the resulting kernel module binary in ‘/usr/lib/beosboot/kernel/module_binary_name’. The next time you generate a BeoBoot image it will be included.

If the driver is for new hardware, the vendor and device IDs for the hardware should be included in the driver list. The driver list is stored in ‘/etc/beowulf/config.boot’. If your driver is composed of multiple modules, dependencies are automatically generated via ‘depmod’. If the driver merely replaces an old driver and doesn’t add support for new hardware, this step may be skipped.

After these steps are completed, re-run BeoBoot to generate a new slave node boot floppy image.

### 3.6 Technical Details

#### 3.6.1 Boot Phases

##### 3.6.1.1 Phase 1 - Initial (Floppy) Boot

Phase 1 is the initial boot up of the machine from the initial (floppy) image. This image may be stored either on a floppy disk or in the BeoBoot partition of the node’s hard drive. See section “Disk Partitioning” in Scyld Beowulf Installation Guide. First, the BIOS loads a sector from the slave node boot image. Next, the boot loader on the floppy (or hard disk) takes over and loads the rest of the data stored in the initial image.

The slave node initial boot image contains a minimal kernel image and an initial ramdisk image. These images probe the PCI for network hardware, configure network interfaces and download the final kernel image and ramdisk that the machine will run.

The final image and ramdisk will be started via a ‘Two Kernel Monte’.

##### 3.6.1.2 Phase 2 - Final Kernel - initrd

In phase 2, the node is running the final kernel image, which was downloaded in phase 1. The root file system is the ramdisk image downloaded during phase 1. This image contains all the kernel modules for this final kernel. The PCI probe will load all relevant drivers at this time. The file system also contains a selection of shared libraries from the master node.
This file system is rather large for a ramdisk (approx 15MB), thus it shouldn’t be stored indefinitely. The ramdisk image contains a smaller image which will be used as the permanent root file system. The boot program for this phase takes this smaller ramdisk image and copies it into one of the /dev/ramX devices.

3.6.1.3 Phase 3 - Final Kernel - ramdisk Root

In phase 3, the linuxrc has exited and the new smaller root file system has been mounted. The init program used is "boot". In this capacity, it starts the BProc slave daemon and waits for it to exit. If the slave daemon dies for any reason, the init program will reboot the system.

The old root file system (the large ramdisk from phase 2) will be mounted on /initrd. This is done so that the shared libraries on that image will still be available throughout node setup time. The slave node setup script will unmount and free this ram disk when appropriate.
4 The Scyld Beowulf Distributed Process Space: BProc

The Scyld Beowulf Distributed Process Space (BProc) is a set of kernel modifications, utilities and libraries which allow a user to start processes on other machines in a Beowulf-style cluster. Remote processes started with this mechanism appear in the process table of the front end machine in a cluster. This allows remote process management using the normal UNIX process control facilities. Signals are transparently forwarded to remote processes and exit status is received using the usual `wait()` mechanisms.

BProc also provides process migration mechanisms for the creation of remote processes. These mechanisms remove the need for most binaries on the remote nodes.

4.1 Installing BProc

BProc requires a number of kernel modifications and modules to be installed. It is much simpler to install pre-built kernel packages rather than build kernel images from scratch. To simplify managing the nodes in a BProc style cluster, use of the BeoBoot cluster management package is highly recommended.

4.1.1 Installation from RPMs

RPMs for Scyld Beowulf are available via FTP from: `ftp://ftp.scyld.com/pub/beowulf`. Note that you may have to modify `'/etc/lilo.conf'` to point to the new kernel. Re-run lilo to make these changes take effect.

4.1.2 Building BProc from Scratch

Building BProc from scratch means building a kernel that includes the BProc modifications. Apply the `bproc` patch to your kernel. When configuring the new kernel, select "Yes" to 'Beowulf Distributed Process Space'. See the documentation included with the Linux kernel for more information about configuring and compiling Linux kernels.

After patching the kernel, it is possible to build the rest of the BProc package by running `make` in the top level `bproc` directory. The Makefile presumes that the kernel tree to build against resides in `'/usr/src/linux'`. If this is not accurate, provide `make` with the `LINUX=/path/to/linux` argument.

4.1.3 Installing

See the instructions with the Linux kernel or your Linux distribution for instructions on how to install a new kernel.

First, install the BProc kernel modules. There are three modules which must be loaded in the following order: `ksyscall.o`, `vmadump.o` and `bproc.o`. After running `depmod`, `modprobe bproc` should load them all. These modules must be loaded on both the front end and the slave nodes.

If using pre-built kernel packages, run the following to install all the programs and modules to their proper locations.
# make install
# depmod -a
# modprobe bproc

Note: BProc daemons require ‘/dev/bproc’ to communicate with the kernel layer. This is a character device with major number 10, minor number 226.

### 4.2 Running BProc

The master daemon, \texttt{bpmaster} is the central part of BProc system. It runs on the front end machine. Once it is running, the slave nodes run the slave daemon, \texttt{bpslave} to connect to the front end machine.

\texttt{bpmaster} runs on the front end machine and handles all the details of running BProc.

# bpmaster

#### 4.2.1 Node States

\textbf{down} Nodes are ‘down’ when they are NOT connected to the BProc master daemon. It is impossible to do anything to nodes via BProc when they are in this state.

\textbf{unavailable} When a node is ‘unavailable’, it is connected to the BProc master but has not yet been tagged as ready for users. While in this state, the system makes no guarantees about the state of the node.

Unavailable usually means that the node is in some transitional state. The node may be booting and setting up or it may be shutting down.

It is also possible that the system administrator has manually set the node state to unavailable to indicate that it should not be used for some other reason.

Nodes also remain unavailable after booting if their hard disks have not been partitioned. See section “Disk Partitioning” in \textit{Scyld Beowulf Installation Guide}.

\textbf{up} When a node is tagged as ‘up’, it is available for use by users.

\textbf{reboot} It is possible to ‘reboot’ a node.

\textbf{halt} It is possible to ‘halt’ a node (suspend processing, but not power off).

\textbf{pwroff} It is possible to remotely power off a node using ‘pwroff’.

Node states may be viewed and manually manipulated using the \texttt{bpctl} program.

### 4.3 VMADump

VMADump is the system used by BProc to take a running process and copy it to a remote node. VMADump saves or restores a process’s memory space to or from a stream. In the case of BProc, the stream is a TCP socket to the remote machine. VMADump implements an optimization which greatly reduces the size of the memory space.
Most programs on the system are dynamically linked. At run time, they will use `mmap` to get copies of various libraries in their memory spaces. Since they are demand paged, the entire library is always mapped even if most of it will never be used. These regions must be included when copying a process’s memory space and again when the process is restored. This is expensive since the C library dwarfs most programs in size.

Here is an example memory space for the program `sleep`. This is taken directly from ‘/proc/pid/maps’.

```
08048000-08049000 r-xp 00000000 03:01 288816 /bin/sleep
08049000-0804a000 rw-p 00000000 03:01 288816 /bin/sleep
40012000-40013000 rw-p 00000000 03:01 911381 /lib/ld-2.1.2.so
00170000-00170000 r-xp 00000000 03:01 911434 /lib/libc-2.1.2.so
40102000-40106000 rw-p 000ea000 03:01 911434 /lib/libc-2.1.2.so
bffe0000-c0000000 rwxp fffff000 00:00 0
```

The total size of the memory space for this trivial program is 1089536 bytes. All but 32K of that comes from shared libraries - VMADump takes advantage this. Instead of storing the data contained in each of these regions, it stores a reference to the regions. When the image is restored, that files will be `mmap`ed to the same memory locations.

In order for this optimization to work, VMADump must know which files it can expect to find in the location where they are restored. VMADump has a list of files which it presumes are present on remote systems. The `vmadlib` utility exists to manage this list. See Section 4.4.7 [vmadlib], page 19.

### 4.3.1 Limitations / Important Details

Note that VMADump will correctly handle regions mapped with `MAP_PRIVATE`, which have been written.

VMADump does not specially handle shared memory regions. A copy of the data within the region will be included in the dump. No attempt to re-share the region will be made at restoration time. The process will get a private copy.

VMADump does not save or restore any information about file descriptors.

VMADump will only dump a single thread of a multi-threaded program. There is currently no way to dump a multi-threaded program in a single dump.
4.4 Program Usage

This section contains basic usage information for the binaries that are included with BProc.

4.4.1 bpmaster

bpmaster -h
bpmaster -v
bpmaster [ -c c_file ] [ -m m_file ]

bpmaster is the BProc master daemon. It runs on the front end machine of a cluster running BProc. It listens on a TCP port and accepts connections from slave daemons. Configuration information comes from the Beowulf configuration file. The BProc master daemon reads interface, iprange, bprocport, allowinsecureports and logfacility. See Appendix A [Scyld Beowulf Config File Reference], page 42.

Options:

‘-h’ Display a help message and exit.
‘-v’ Display version information and exit.
‘-d’ Increase debugging (verbose)
‘-c c_file’ Read configuration information from c_file instead of the default file (/etc/beowulf/config).
‘-m m_file’ Dump a message trace to m_file. This is only useful for debugging and slows down the daemons.

4.4.2 bpslave

bpslave -h
bpslave -v
bpslave [ -l facility ] [ -r ] [ -m m_file ] masterIPaddr port

bpslave is the BProc slave daemon. It runs on slave nodes in a cluster and connects to the front end machine (masterIPaddr) to accept jobs through masterport (port).

Options:

‘-h’ Display a help message and exit.
‘-v’ Display version information and exit.
‘-l <log>’ Specify the log facility to which the messages should be sent (default=daemon).
‘-r’ Automatically reconnect if the connection to the master daemon is lost.
‘-d’ Do not daemonize self.
‘-m m_file’ Dump a message trace to m_file. This is only useful for debugging and slows down the daemons.
‘-v’ Increase verbose level (implies -d)
Chapter 4: The Scyld Beowulf Distributed Process Space: BProc

4.4.3 **bpstat**


bpstat displays various pieces of status information about a BProc cluster. This program also includes a number of options intended to be useful for scripts.

Options:

`-h`  Display a help message and exit.

`-v`  Display version information and exit.

`-n`  Print the number of nodes in the machine. Note that this is the number of nodes configured (via iprange) not the number of nodes that are up.

`-u`  Print the number of nodes that are up.

`-a node`  Print the IP address of node, where node is value 0 through N-1 or -1 (the front end machine).

`-s node`  Print the status address of node, where node is value 0 through N-1 or -1 (the front end machine).

`-m`  Display machine state. (This is the default mode of operation.)

`-p`  Display process state.

`-P`  Read output from ps from standard input and add a column containing the node which the process exists on.

4.4.4 **bpctl**

bpctl -h
bpctl -v
bpctl -M [-a]
bpctl -S node [-a] [-r dir] [-s state]

bpctl is bproc control. Used to apply commands to referenced nodes.

Options:

`-h`  Display a help message and exit.

`-v`  Display version information and exit.

`-M`  Apply the following commands to the front end machine.

`-S nodenum`  where nodenum is value, 0 through N-1 or all. Apply the following arguments to slave node, nodenum or all nodes.

`-a`  Print the IP address of the front end machine (when used with the -M option). Print the IP address of the slave node (when used with the -S option).
`-r dir`  Ask the slave daemon to perform a chroot() to dir. After doing this, all processes started on a node via BProc will see dir as their root directory. This command is only usable on slave nodes.

`-s state`  Set slave state to state. The valid node states are 'down', 'unavailable', 'error', 'up', 'reboot', 'halt' and 'pwroff'. The state of nodes in the 'down' state cannot be changed. Setting the state of a node to 'down' will cause a node to be disconnected from the master daemon.

### 4.4.5 bpsh

bpsh [-n] nodenumber command  
bpsh -a [-n] command  
bpsh -A [-n] command  
bpsh is a rsh replacement. Runs command on node.  
Options:  
`-a`  Run the command on all available nodes.  
`-A`  Run the command on all nodes which are 'up'.  
`-h`  Display a help message and exit.  
`-v`  Display version information and exit.  
`-n`  Redirect stdin from /dev/null.

### 4.4.6 bpcp

bpcp [-p] f1 f2  
bpcp [-r] [-p] f1 ... fn dir  
bpcp copies files between machines. Each file or directory argument is either a remote file name of the form node:path, or a local file name (containing no '::' characters).  
Options:  
`-p`  Preserve file timestamps.  
`-r`  Copy directories recursively.

### 4.4.7 vmadlib

vmadlib -c  
vmadlib -a [ libs ... ]  
vmadlib -d [ libs ... ]  
vmadlib -l  
This program is a utility to manage the VMADump in-kernel library list.  
Options:
‘-c’ Clear the library list.
‘-a [ libs ... ]’
   Add libs to the library list. If ‘-’ is given as an argument, newline separated library file names will read from standard input.
‘-d [ libs ... ]’
   Delete libs to the library list. If ‘-’ is given as an argument, newline separated library file names will read from standard input.
‘-l’ List the libraries in the library list.
4.5 BProc Programmer’s Guide

BProc currently includes a C Library interface only.

4.5.1 Process Migration With BProc

Bproc provides a number of mechanisms for creating processes on remote nodes. It is instructive to think of these mechanisms as moving processes from the front end to the remote node. The rexec mechanism is like doing a move then exec with lower overhead. The rfork mechanism is implemented as an ordinary fork on the front end and then a move to the remote node before the system call returns. Execmove does an exec and then move before the exec returns to the new process.

Movement to another machine on the system is voluntary and is not transparent. Once a process has been moved all its open files are lost except for STDOUT and STDERR. These two are replaced with a single socket (their outputs are combined). There is an IO daemon which will forward from the other end of that connection to whatever the original STDOUT was connected. No pseudo tty operations are done.

The move is completely visible to the process after it has moved except for process ID space operations. Process ID space operations include fork, wait, kill, etc. All file operations will operate on files local to the node to which the process has been moved. Memory that was shared on the front end will no longer be shared.

4.5.2 C Library Interface

Programs that use the BProc library should contain the line #include <sys/bproc.h> and be linked against the BProc library by adding -lbproc to the linker command line.

4.5.2.1 Machine Information Calls

The BProc library provides the following interfaces for finding information about the configuration of the machine. These interfaces may be used from any node on the cluster.

int bproc_numnodes(void)
  Returns the number of nodes in the system. This is the number of slave nodes (not including the front end). The nodes are numbered 0 through N-1. This function returns -1 on error.

int bproc_currnnode(void)
  This call returns the node number on which a process is currently running. -1 indicates that the process is running on the front end.

int bproc_nodestatus(int node)
  This function is for use on SLAVE NODES ONLY - not the front end machine, since it is always ‘up’.
  Returns the status of node number given node. This function returns -1 on error and errno will be set appropriately. The value returned is one of the following:
**Chapter 4: The Scyld Beowulf Distributed Process Space: BProc**

**bproc_node_down**
The node is not connected to the master daemon. It may be off or crashed or not far enough along in its boot process to connect to the master daemon.

**bproc_node_unavailable**
The node is running but is currently unavailable to users (this is not enforced). Nodes are in this state while booting or shutting down.

**bproc_node_error**
There is a problem with the node.

**bproc_node_up**
The node is up and ready to accept processes.

```c
int bproc_nodeaddr(int node, struct sockaddr *addr, int *size)
```
This call saves the IP address of `node` in the sockaddr pointed to by `addr`. The `size` parameter should be initialized to indicate the amount of space pointed to by `addr`. On return it contains the actual size of the `addr` returned (in bytes).

This function returns 0 on success and -1 on failure.

```c
int bproc_masteraddr(struct sockaddr *addr, int *size)
```
This call is equivalent to `bproc_nodeaddr(-1, addr, size)`

### 4.5.2.2 Process Migration Calls

```c
int bproc_rexec(int node, char *cmd, char **argv, char **envp)
```
This call has semantics similar to `execve`. It replaces the current process with a new one. The new process is created on `node` and the local process becomes the ghost representing it. All arguments are interpreted on the remote machine. The binary and all libraries it needs must be present on the remote machine. Currently, if remote process creation is successful but exec fails, the process will just exit with status 1. If remote process creation fails, the function will return -1.

```c
int bproc_move(int node)
```
This call will move the current process to the remote node number given by `node`. Returns 0 on success, -1 on failure.

```c
int bproc_rfork(int node)
```
The semantics of this function are designed to mimic `fork` except that the child process created will end up on the node given by the `node` argument. The process forks a child and that child performs a `bproc_move` to move itself to the remote node.

Combining these two operations in a system call, prevents zombies and SIGCHLD’s in the case that the fork is successful but the move is not. On success, this function returns the process ID of the new child process to the parent and zero to the child. On failure it returns -1.
int bproc_execmove(int node, char *cmd, char **argv, char **envp)
   This function allows execution of local binaries on remote nodes. BProc will
   start the binary on the current node and then move it to a remote node, before
   the binary gets running.
   NOTE: This migration mechanism will move the binary image but not any
dynamically loaded libraries that the application might need. Therefore any
libraries that the application uses must be present on the remote system.

4.5.2.3 System Management Calls

The system management calls are made by programs like bpctl to control the machine
state. These calls are privledged and not useful to normal applications.

int bproc_slave_chroot(int node, char *path)
   This call requests the slave daemon to perform a chroot. This call returns 0
   on success and -1 on failure.

int bproc_setnodestatus(int node, int status)
   This call sets the status of a node. See bproc_nodestatus for information
   regarding permissible node states. It is not possible to change the status of a
   node which is marked as down.
5 Scyld Beowulf Message Passing Interface: BeoMPI

MPI, or Message Passing Interface, is a de facto-standard interface for message-based parallel computing that is maintained by a forum of members drawn from academia and the remnants of the traditional supercomputing industry.

5.1 Overview

5.1.1 History of MPI

The MPI forum was self-tasked with creating a standard that could loosely accommodate the existing systems for message-passing on multi-computers in a way that could be implemented on contemporary machines with reasonable performance.

MPI, unlike earlier systems such as PVM, was to be a standard instead of software itself. Furthermore, MPI was to be an API standard. This meant that implementors were granted wide latitude to implement MPI in ways that need not have runtime interoperability with other platforms or implementations.

At the present time, there are at least a dozen such implementations of MPI under active maintenance – the Scyld Computing implementation, BeoMPI is one.


5.1.2 Compatibility with BeoMPI

Scyld distributes BeoMPI, an implementation of MPI drawn directly from the MPICH project at Argonne National Laboratory. Scyld has made only those changes necessary to allow MPICH to take advantage of the special system features provided by our Beowulf system software (notably the features provided by the BProc system).

In general, if you have an application which can take advantage of MPI, you can make it run on Beowulf. In particular, applications which already run on MPICH should have no problems on Beowulf.

Scyld has simplified the deployment of MPI applications in a number of ways – applications which take advantage of these simplifications may experience porting pains when backporting to more primitive systems. Fortunately, our improvements to the system are not provided at the expense of compliance with the MPI standard.


5.2 Installing BeoMPI

BeoMPI is built against the Scyld BProc system. Your system must have the BProc dynamic libraries installed to install BeoMPI. Additionally, your system must have the BProc header files installed to successfully build BeoMPI. NOTE: You do not need to have a BProc-enabled kernel to build, install, or run BeoMPI, but you will not be able to take advantage of many of the multiprocessing features of a Beowulf system.
5.2.1 Installation from RPMs


5.2.2 Building BeoMPI from Scratch

Make BeoMPI from scratch by running `make` in the top level `beompi` directory.

5.2.3 Installing

Install BeoMPI by running `make install` in the top level `beompi` directory.

After installing, run `ldconfig`.

5.2.4 Co-existing with other MPI Implementations

NOTE: As `beompi` is designed for installation as a system-wide MPI resource for Beowulf systems, the `beompi` installation process creates a number of files which may create collisions with other MPI implementations you may intend to install. In particular, you should be aware of:

- man pages in `/usr/man`
- header files in `/usr/include` (including `mpi.h`)
- libraries in `/usr/lib` (including `libmpi.so` and `libmpi.a`)
- the binary program `mpirun` in `/usr/bin`

(a complete list of files is available through the rpm system)

You should try to install alternate MPI implementations in non-conflicting locations as some Beowulf utilities may depend on features present in Scyld’s BeoMPI.

If you wish to install BeoMPI on an existing system, you may specify alternate file locations when installing a scratch-built system. Do this by running `USRDIR=/usr/beowulf make -e install` in the top level `beompi` directory (where `'/usr/beowulf'` is your intended target path).

5.3 Running BeoMPI

There are no configuration files or daemons which require configuration to use the BeoMPI subsystem for Beowulf. Information about the state of the system and the nodes is gathered from the BProc system at runtime.

Instructions on running BeoMPI therefore relate only to starting MPI-enabled applications on a Beowulf system.

5.3.1 Program Placement

Simply preparing a job for execution has long been a weak point on loosely-coupled MPPs. It has typically been a multi-stage process that required careful system configuration by a skilled administrator.
Job creation on a typical first-generation (pre-BProc) Beowulf cluster had a number of prerequisite steps:
- Ensure that the user had an account on all the target nodes accomplished manually, via script, or via a system like NIS
- Ensure that the user could spawn jobs on the target nodes accomplished via ‘hosts.allow’, PAM, or remote server-daemons running on behalf of the user (ala PVM)
- Ensure that every node could find a copy of the binary accomplished manually, via script, or via NFS-like shared namespaces
- Ensure that every node could dynamically link the binary against the necessary shared libraries, accomplished manually, via script, or via NFS-shared namespaces.
- Ensure that the application had knowledge of the state of the machine accomplished manually, via configuration file, or via scheduler system.

The BProc system eliminates or simplifies many of these steps.
- Users no longer require accounts on remote nodes.
- Users no longer require explicit authentication mechanisms to run programs on remote nodes.
- Neither binaries nor libraries need to be available on(from) remote nodes.
- The BProc system tracks the state of the system for the user.

Given the features offered by the BProc system, installing and running a parallel program can be as simple as running a serial one.

5.4 mpirun

The MPI standard does not extend to job creation (exception: see MPI_Comm_spawn() in MPI-2) However, a convention does exist: most MPI implementations support an external program called, ‘mpirun’ that is responsible for running an MPI application.

While beompi does not require the use of such an external link, beompi makes it available for those applications which expect it.

5.4.1 Invoking mpirun

mpirun --mpi-help
mpirun --mpi-version
mpirun [options] [options] <command> [command options]

Options:
- `-np <int>`
  spawn a program with MPI size int
- `--all-nodes`
  MPI job shall run on all available nodes
- `--all-cpus`
  MPI job shall run on all available cpus
`--nodes <int>`
MPI job shall run on `int` nodes

`--cpus <int>`
MPI job shall run on `int` cpus

`--local` MPI job shall run exclusively on the front end node

In addition to the above command-line options, mpirun responds to several environment variables:

Variables:

- `NP=<int>` spawn a program with MPI size `int`
- `NODES=<int>` MPI job shall run on `int` nodes
- `CPUS=<int>` MPI job shall run on `int` cpus
- `LOCAL` MPI job shall run exclusively on the front end node

Command-line arguments override conflicting values supplied by the environment.

### 5.4.2 Inline-mpirun

Instead of relying on an external program to spawn MPI jobs, beompi makes an inline interface available to applications which link dynamically against the MPI library. Users may directly supply any of the command-line arguments, environment variables, or compile-time hints accepted by mpirun directly to the MPI-enabled application.

These arguments are processed and a job schedule is created before the application’s `main()` function is even called. This feature allows for the construction of a parallelized application which behaves and can be invoked transparently to the user.

The inline mpirun features may be accessed with the same command-line options and environment variables as the stand-alone version of mpirun, however mpirun arguments may now be mixed freely with options belonging to the command. For example:

```bash
> mpifrob --mode=deathray --np 16 --outputfile=/dev/null
```
may be used in place of

```bash
> mpirun --np 16 mpifrob --mode=deathray --outputfile=/dev/null
```

The inline mpirun may be disabled by:

- setting the scheduler hint, `MPIRUN_INLINE` to 0.
- setting the environment variable, `NO_INLINE_MPIRUN` to non-empty.
- supplying the command-line argument, `--no-inline-mpirun` to the application.

### 5.4.3 Invoking mpirun from Inside an Application

beompi supports one other model of MPI job creation to address the special needs of applications with defined dynamic-link interfaces to executable ‘plug-ins’.

beompi’s in-place job creation system allows an application of this type to run an MPI-enabled plug-in without itself having to be MPI-aware. Provided is a fragment plug-in that
is MPI-aware. Note that `mpirun()` will generate an argc, argv pair for you that contains the arguments needed by `MPI_Init()` — even if you were not passed an argc, argv as part of your plug-in API.

```c
#include <mpi.h>
#include <mpirun.h>

int
plugin_init()
{
int retval;
int module_argc;
char **module_argv;
int rank,size;

/* schedule this job -- ask for size==8 */
retval=mpirun(&module_argc,&module_argv,MSH_SIZE,8,MSH_END);

MPI_Init(&module_argc,&module_argv);

/* From here, all of the jobs are running from this *
* point in the code -- no need for them to go through *
* the body of the parent application to get here. */

MPI_Comm_size(MPI_COMM_WORLD,&size);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);

/* Do parallel processing here */

MPI_Finalize();

/*
* Children should never exit back into the parent application
*/

if(rank!=0) exit(0); else return 0; }
```
5.5 BeoMPI Programmer’s Guide

BeoMPI features language bindings for C, C++, and Fortran.

5.5.1 Compiling with beompi

beompi places the MPI header files and libraries in standard locations. Compiling and linking an MPI application is often as simple as:

```bash
> cc -lmpi foo.c -o foo
```

To compile a fortran code, try:

```bash
f77 -lmpif foo.f -o foo
```

Notice that the MPI library for fortran is 'mpif'. In the future, these libraries may be merged – in which case the 'mpif' library will be maintained for backwards compatibility with beompi and with other MPI MPI implementations.

5.5.2 Scheduler Hinting with beompi

While beompi supports the defacto mpirun interface for scheduling and spawning MPI-enabled programs, Scyld has created an extra mechanism for an application to directly provide scheduler cues to the system without needing external ‘schema’ files or enormous mpirun command lines.

This ‘hinting’ technique involves placing harmless macro calls inside an MPI-enabled application (as shown below) that generate specially-named common symbols in the resulting application. These symbols are available both to the beompi MPI library and to external programs which process the application’s symbol table.

An example:

```c
#include <mpi.h> /* generally necessary for mpi applications */
#include <mpirun.h> /* necessary to use the library interface to mpirun */

#ifdef MIPRUN_GLOBAL_HINT
MPIRUN_GLOBAL_HINT(MPIRUN_NP,16) /* this code likes at least 16 jobs */
#endif

int main(int argc, char **argv)
{
    MPI_Init(&argc,&argv);
    /* do parallel processing */
    MPI_Finalize();

    return 0;
}
```

In the above example, the application hints that it wants to run as a 16-way job. These hints may be overridden by both command-line arguments and environment variables, but may be convenient for applications that have particular knowledge about the way they perform.
A number of hints are defined:

**MPIRUN_INLINE <flag>**
- determines whether the inlined mpirun should be called

**MPIRUN_NP <int>**
- program will spawn with MPI size int

**MPIRUN_NODES <int>**
- program shall run on int nodes

**MPIRUN_CPUS <int>**
- program shall run on int cpus

**MPIRUN_LOCAL <flag>**
- determines whether the job shall run exclusively on the front end.

## 5.6 Troubleshooting

### 5.6.1 Troubleshooting with strace

`strace` and other `ptrace()` based tools are not currently well supported under the BProc system when running on multiple machines. These tools may be used, however, if the target MPI application is run as a ‘local’ job. Example:

```
> LOCAL=true strace -f mpi-application
```

`strace` and `ltrace` both accept `-f` which instructs them to follow `fork()` calls and print calls for children. You must supply this option to see the system calls for the entire MPI application.

### 5.6.2 Troubleshooting mpirun

`mpirun` contains a built-in facility for logging and debugging. You can access this facility by supplying the `MFT_LOG_THRESH` environment variable to any of the `mpirun` forms described here.

`MFT_LOG_THRESH` may take on one of the following values:

- **none**  
  no logging will be performed

- **fatal**  
  messages that correspond to program termination will be logged

- **error**  
  messages that correspond to program errors will be logged

- **info**  
  messages that are normal but informative will be logged

- **branch**  
  messages that occur at conditional logic points will be logged

- **progress**  
  messages that indicate "Got this far!" will be logged

- **entryexit**  
  messages that correspond to function entry and exit will be logged

Logging levels are cumulative. Setting `MFT_LOG_THRESH` to `info` will cause log messages for `error` and `fatal` levels to also be emitted.
5.6.3 Troubleshooting MPI applications

beompi is constructed from MPICH on P4. MPI applications built on top of beompi may use the debugging features built into P4. Example:

```
> mpi-application -p4dbg 100
```

-p4dbg accepts an integer from 0 to 100; 100 is maximum logging.
6 Intel PPro Performance Counter Support

The Pentium Pro Performance counter package adds support for the hardware performance counters present in the Intel Pentium Pro, Pentium II, Celeron and Pentium III CPUs. The Pentium Pro provides two counters which can programmed to count a wide variety of system events. (See Countable Events, below.)

The counters are virtualized so many different processes can safely use the counters at the same time. Processes will only count when they are scheduled. Since the counter values and configurations are saved and restored at context switch time, the counters are safe to use on SMP machines where processes may move from one CPU to another. When counting in the system-wide mode on an SMP machine, individual counts are returned for each CPU in the system.

6.1 C Language Interface

The C language interface is provided via ‘libperf.a’. The included header file (‘perf.h’) defines the following interfaces. Note that this requires ‘asm/perf.h’ from the kernel source to be present at compile time.

PERF_COUNTERS

PERF_COUNTERS is the number of performance counters supported by this performance counter library. Currently, 2 counters are supported.

int perf_reset(void);
The perf_reset function clears the configuration and counter registers. If counting was started, it will be stopped.

int perf_get_config(int counter, int *config);
The perf_get_config function reads back counter configurations. counter is the counter whose configuration is to be read and config points to the location where the value will be stored. The value read back may not always be the same as the value that was written. (See PERF_OS and PERF_USR.)

int perf_set_config(int counter, int config);
The perf_set_config function is used to select which events will be counted in a counter. The config argument is one of the countable events (see below) and may be OR’ed with zero or more flags. Note that some values can only be counted in certain counters. This function has the side effect of stopping the counters and resetting them back to zero.

int perf_start(void);
int perf_stop(void);
The perf_start and perf_stop functions start and stop the counters. These should be used after configuring the counters. Note that these functions start and stop all the counters.
int perf_read(int counter, unsigned long long *dest);

The `perf_read` function reads the value of a single performance counter. `counter` is the counter to be read and the value will be stored in the memory location pointed to by `dest`.

int perf_write(int counter, unsigned long long *src);

The `perf_write` function writes the value of a single performance counter. `counter` is the counter to be written and the value will be read from the memory location pointed to by `src`.

int perf_wait(pid_t pid, int *status, int options, struct rusage *ru, unsigned long long *counts);

The `perf_wait` function is an extension of the `wait(4)` function. Its operation is identical except that it can also return the values of the performance counters at the time that the process exited. The counts argument should be an array of length `PERF_COUNTERS`.

There are versions of these functions that may be used for system wide counting. Normally, the counter configurations are switched at task switch time so that each process appears to have its own set of counters. Counters can also be used on a system-wide basis. In this mode, counting is unaffected by task switches. Every CPU also produces its own counting results.

The system-wide counters are only available to the super user. While using the system-wide counters, users will receive an EBUSY error if they attempt to use ‘per-process’ counters. Calling any of the `perf_sys` functions (except `perf_sys_reset`) will cause system-wide counting to start. System-wide counting will not stop until `perf_sys_reset` is called again. Note that system-wide counting does NOT stop if the process that started system-wide counting terminates.

int perf_sys_reset(void);

Calling `perf_sys_reset` clears the counter configuration and frees the performance counters for per-process use.

int perf_sys_set_config(int cpu, int counter, int event, int flags);
int perf_sys_get_config(int cpu, int counter, int *event, int *flags);
int perf_sys_start(void);
int perf_sys_stop(void);
int perf_sys_read(int cpu, int counter, unsigned long long *dest);
int perf_sys_write(int cpu, int counter, unsigned long long *src);

6.1.1 Return value

All of the following functions return 0 on success and -1 on failure. On failure, `errno` will also be set.

The perf syscalls can produce the following errors:

EBUSY The counters are being used for system-wide counters are not available for per-process counting.
**EPERM**  A non-root user attempted to use the system-wide profiling functions.

**EFAULT**  A bad pointer was given as an argument to the system call.

### 6.1.2 System Calls

In general the `sys_perf` system call is the only system call that will affect counter configurations.

**fork**  Counting configuration is not inherited by the child process. Counter configuration in the parent process is unaffected.

**exec**  The counter configuration is unchanged after an exec syscall. If counting was started before the exec call, it will continue after the exec call. This allows for counting on processes which do not support performance counters when used in conjunction with `perf_wait()`.

### 6.2 Counter Configuration

Counter configurations are stored in integers. Valid configurations are generated by picking one of the countable events and doing a bitwise OR with zero or more of the counter flags.

#### 6.2.1 Countable Events

##### 6.2.1.1 Data Cache Unit (DCU)

**PERF_DATA_MEM_REFS**  All memory references, both cacheable and non-cacheable.

**PERF_DCU_LINES_IN**  Total lines allocated in the DCU.

**PERF_DCU_M_LINES_IN**  Number of M state lines allocated in the DCU.

**PERF_DCU_M_LINES_OUT**  Number of M state lines evicted from the DCU. This includes evictions via snopp HITM, intervention or replacement.

**PERF_DCU_MISS_STANDING**  Weighted number of cycles while a DCU miss is outstanding.

##### 6.2.1.2 Instruction Fetch Unit (IFU)

**PERF_IFU_IFETCH**  Number of instruction fetches, both cacheable and non-cacheable.

**PERF_IFU_IFETCH_MISS**  Number of instruction fetch misses.

**PERF_ITLB_MISS**  Number of ITLB misses.
PERF_IFU_MEM_STALL
   Number of cycles that the instruction fetch pipe stage is stalled including cache
   misses, ITLB misses, ITLB faults, and victim cache evictions.

PERF_ILDSTALL
   Number of cycles that the instruction length decoder is stalled.

### 6.2.1.3 L2 Cache

PERF_L2_IFETCH
   Number of L2 instruction fetches. Requires MESI flags.

PERF_L2_LD
   Number of L2 data loads. Requires MESI flags.

PERF_L2_ST
   Number of L2 data stores. Requires MESI flags.

PERF_L2_LINES_IN
   Number of lines allocated in the L2.

PERF_L2_LINES_OUT
   Number of lines removed from the L2 for any reason.

PERF_L2_LINES_INM
   Number of modified lines allocated in the L2.

PERF_L2_LINES_OUTM
   Number of modified lines removed from the L2 for any reason.

PERF_L2_RQSTS
   Number of L2 requests. Requires MESI flags.

PERF_L2_ADS
   Number of L2 address strobes.

PERF_L2_DBUS_BUSY
   Number of cycles during which the data bus was busy.

PERF_L2_DBUS_BUSY_RD
   Number of cycles during which the data bus was busy transferring data from
   the L2 to the processor.

### 6.2.1.4 External Bus Logic

PERF_BUS_DRDY_CLOCKS
   Number of clocks during which DRDY is asserted. Requires SELF/ANY flags.

PERF_BUS_LOCK_CLOCKS
   Number of clocks during which LOCK is asserted. Requires SELF/ANY flags.

PERF_BUS_REQ_OUTSTANDING
   Number of bus requests outstanding.
PERF_BUS_TRAN_BRD
Number of burst read transactions. Requires SELF/ANY flags.

PERF_BUS_TRAN_RFO
Number of read for ownership transactions. Requires SELF/ANY flags.

PERF_BUSTRANS_WB
Number of write back transactions. Requires SELF/ANY flags.

PERF_BUS_TRAN_IFETCH
Number of instruction fetch transactions. Requires SELF/ANY flags.

PERF_BUS_TRAN_INVAL
Number of invalidate transactions. Requires SELF/ANY flags.

PERF_BUS_TRAN_PWR
Number of partial write transactions. Requires SELF/ANY flags.

PERF_BUS_TRAN_P
Number of partial transactions. Requires SELF/ANY flags.

PERF_BUS_TRAN_IO
Number of IO transactions. Requires SELF ANY flags.

PERF_BUS_TRAN_DEF
Number of deferred transactions. Requires SELF/ANY flags.

PERF_BUS_TRAN_BURST
Number of burst transactions. Requires SELF/ANY flags.

PERF_BUS_TRAN_ANY
Number of all transactions. Requires SELF ANY flags.

PERF_BUS_TRAN_MEM
Number of memory transactions. Requires SELF/ANY flags.

PERF_BUS_DATA_RCV
Number of bus clock cycles during which this processor is receiving data.

PERF_BUS_BNR_DRV
Number of bus clock cycles during which this processor is driving the BNR pin.

PERF_BUS_HIT_DRV
Number of bus clock cycles during which this processor is driving the HIT pin.

PERF_BUS_HITM_DRV
Number of bus clock cycles during which this processor is driving the HITM pin.

PERF_BUS_SNOOP_STALL
Number of clock cycles during which the bus is snoop stalled.

6.2.1.5 Floating Point Unit

PERF_FLOPS
Number of computational floating-point operations retired. Counter 0 only.
PERF_FP_COMP_OPS_EXE
Number of computational floating-point operations executed. Counter 0 only.

PERF_FP_ASSIST
Number of floating-point exception cases handled by microcode. Counter 1 only.

PERF_MUL
Number of multiplies. Counter 1 only.

PERF_DIV
Number of divides. Counter 1 only.

PERF_CYCLES_DIV_BUSY
Number of cycles during which the divider is busy. Counter 0 only.

6.2.1.6 Memory Ordering

PERF_LD_BLOCK
Number of store buffer blocks.

PERF_SB_DRAINS
Number of store buffer drain cycles.

PERF_MISALIGN_MEM_REF
Number of misaligned data memory references.

6.2.1.7 Instruction Decoding and Retirement

PERF_INST_RETIRED
Number of instructions retired.

PERF_UOPS_RETIRED
Number of UOPS retired.

PERF_INST_DECODER
Number of instructions decoded.

6.2.1.8 Interrupts

PERF_HW_INT_RX
Number of hardware interrupts received.

PERF_CYCLES_INST_MASKED
Number of processor cycles for which interrupts are disabled.

PERF_CYCLES_INT_PENDING_AND_MASKED
Number of processor cycles for which interrupts are disabled and interrupts are pending.

6.2.1.9 Branches

PERF_BR_INST_RETIRED
Number of branch instructions retired.
PERF_BR_MISS_PRED_RETIRE
  Number of mispredicted branches retired.

PERF_BR_TAKEN_RETIRE
  Number of taken branches retired.

PERF_BR_MISS_PRED_TAKEN_RET
  Number of taken mispredicted branched retired.

PERF_BR_INST_DECODED
  Number of branch instructions decoded.

PERF_BR_BTB_MISSES
  Number of branches that miss the BTB.

PERF_BR_BOGUS
  Number of bogus branches.

PERF_BACLEARS
  Number of times BACLEAR is asserted.

6.2.1.10 Stalls

PERF_RESOURCESTALLS
  Number of cycles during which there are resource related stalls.

PERF_PARTIAL_RATSTALLS
  Number of cycles or event for partial stalls.

6.2.1.11 Segment Register Loads

PERF_SEGMENT_REG_LOADS
  Number of segment register loads.

6.2.1.12 Clocks

PERF_CPU_CLK_UNHALTED
  Number of cycles during which the processor is not halted.

6.2.2 Counter Flags

Many of the external bus logic events can be further qualified with either the PERF_SELF or PERF_ANY flags.

PERF_SELF
  Count events for this processor only.

PERF_ANY
  Count events for any processor.

Many of the L2 cache events to be counted can be further qualified with the following flags. These flags can be OR’ed together to count more than one cache state.
PERF_CACHE_M
Count events for modified cache lines.

PERF_CACHE_E
Count events for exclusive cache lines.

PERF_CACHE_S
Count events for shared cache lines.

PERF_CACHE_I
Count events for invalid cache lines.

PERF_CACHE_ALL
Count events for all cache states.

The flags PERF_OS and PERF_USR flags allow you to control when counting should occur. These two flags can be combined. The default (when no flag is specified) for per-process counting is PERF_USR only and the default for system-wide counting is PERF_OS only.

PERF_OS Count events only when the processor is operating in system mode (privilege level 0).

PERF_USR Count events only when the processor is operating in user mode (privilege levels 1, 2 or 3).

6.3 Known Issues and Bugs
When system-wide counting is used, the other processes get no indication that their monitoring has been corrupted.

6.4 More Information
Most of the information provided here is derived from the Intel architecture manuals available at http://developer.intel.com .
See also the wait(4) manual page for more information on wait semantics.
Chapter 7: Monitoring the Status of the Scyld Beowulf Cluster: BeoStatus

7 Monitoring the Status of the Scyld Beowulf Cluster: BeoStatus

BeoStatus is the Scyld Beowulf Status program. It displays CPU usage, memory usage, swap usage, and root partition disk usage. These outputs may be displayed in four different formats: two GTK+ formats, Curses format and line output format. Beowstatus works on either a Beowulf BProc system or on a simple cluster of Linux machines using rsh.

7.1 Overview of Options

This is an overview of available options to BeoStatus:

(Shorthand, explicit)

- 

-r, --rsh
Use rsh to communicate with nodes.

-s, --ssh
Use ssh to communicate with nodes.

-b, --bpsh
Use bpsh to communicate with nodes.

-c, --curses
Use curses mode output instead of Gnome/GTK+.

-t, --text
Use plain text mode output instead of Gnome/GTK+.

-d, --dots
Use compact dot mode output instead of full size. Gnome/GTK+.

-u, --update=secs
Rate at which statistics are reported; value of secs in seconds (affects cpu load); nominally, 4 seconds.

-v, --version
Display version information and exit.

7.2 Communication Methods

There are three different methods for communicating with the nodes in the cluster: ssh, rsh, and Beowulf/BProc. The default communication method is currently ssh. Rsh mode is selected with the --r or --rsh option; Beowulf/BProc mode is selected using --b or --bpsh option. Only one of these should be specified at a time. While ssh and rsh modes use machine names, Beowulf/BProc mode uses node numbers (OR, if no numbers are specified, then all nodes defined by IP Address range are implied).

beostatus -b 0 1 2 3

In Beowulf mode, the up and available flags correspond directly to the BProc states of the same name. In rsh or ssh modes, up means that beostatus is successfully pinging the machine with ICMP packets; available means that beostatus is receiving status packets from that host.

If while running in rsh or ssh mode, node status is up but not available, manually use rsh or ssh to transfer and run the grabstats program on the remote machine. In order to
avoid the password challenge on the remote machine, you must list your local machine in the ‘.rhosts’ file (rsh) or ‘.ssh/authorized_keys’ (ssh) file on the remote machine.

7.3 Presentation Modes

There are currently four presentation modes. The default mode is GTK+ mode, which uses a progress bar to represent usage.

Dots mode is a compact GTK+ format which uses colored dots to represent each node’s status. The dot color represents the status. The default color scheme is as follows:

- Red: node down, unreachable.
- Yellow: node up, but not responding (unavailable).
- Green: node up and mostly idle.
- Blue: node up and busy.

Curses mode should be used when an X server connection is not available for beostatus. It is automatically selected if the DISPLAY environment variable is not set or is manually selected with the --curses flag.

There is also a line output mode, selected with the --text flag in case a terminal doesn’t support Curses control characters.
Appendix A Scyld Beowulf Configuration
File Reference

The Beowulf configuration file is used by all the Beowulf daemons and normally resides in ‘/etc/beowulf/config’ on the front end machine of the Beowulf cluster.

address ipaddress
This sets the address of the internal cluster interface on the front end machine to ipaddress. This address should not fall in the IP range used for slave nodes.

allowinsecureports
The allowinsecureports directive causes the BProc master daemon to accept connections from non-privileged ports.

bootfile file
The bootfile directive specifies the path to the boot image for slave nodes in the cluster.

bootport port
The bootport directive controls from which TCP port the boot server will serve the boot image.

bprocport port
The bprocport directive controls from which TCP port the bproc server will run.

fsck policy
The fsck command sets the default policy for file system checks. The possible values of policy are ‘never’, ‘safe’ and ‘full’.
Note: this configuration item interacts with mkfs.
‘never’ The file system will never be checked.
‘safe’ A safe file system check will be used to check the file system. This check is similar to the check which is done automatically by a UNIX system at boot time. (NOTE: It is possible for a recoverable file system to fail this check).
‘full’ A full file system check will be used to check the file system. This is similar to the check executed when fsck is run on the command line. Note: Any recoverable file system should be repaired by this check.

ignore macaddress
The ignore tag specifies MAC addresses which will be ignored on the network. This should be used to have the Beowulf boot server ignore RARP requests from devices other than Beowulf nodes.

interface eth
The interface directive tells the Beowulf servers which interface is used as the internal cluster interface. The Beowulf server daemons will listen on this interface.
### Appendix A: Scyld Beowulf Configuration File Reference

#### iprange w.x.y.z w.x.y.z

The `iprange` directive defines for the Beowulf daemons the range of IP addresses for nodes in the cluster. The range given by the two IP addresses is inclusive. Node numbers and the maximum number of nodes are specified by this IP range. Note that these **MUST** be IP addresses and **NOT** host names.

#### libraries library ...

The `libraries` tag lists the libraries that should be copied to remote nodes at boot time. Entries on this list can be either specific file names or directory names. If a directory name is given, all the ELF shared objects (.so files) in that directory will be copied.

#### logfacility facilityname


#### mkfs policy

The `mkfs` command sets the default policy for creating file systems at boot time. The possible values for `policy` are ‘never’, ‘if_needed’ and ‘always’.

- **‘never’**  
  Never create file systems. Node setup will fail if file systems are not usable.

- **‘if_needed’**  
  Create file systems if the existing file systems fail the file system check. See `fsck`.

- **‘always’**  
  Always create file systems. In this case the file system checks will be skipped at startup.

#### netmask mask

The `netmask` sets the netmask on the internal cluster network.

#### node macaddress

The `node` tag specifies that the MAC address belongs to a node in the cluster. The ordering of nodes depends on the order of the node lines. The first node directive specifies node 0, the second one specifies node 1 and so on. If you wish to leave a gap in the sequence, preface the MAC address with "off", instead of "node".

#### unknown macaddress

The unknown tag specifies an unknown MAC address for which the boot server has received a RARP request. These lines are automatically added by the boot server.
Appendix B Scyld Beowulf Boot Configuration File Reference

The Scyld Beowulf boot configuration file is used by the `beoboot` script when creating new boot images. A copy of this configuration file is included in the boot images and actually used at boot time. This file is located in `'/etc/beowulf/config.boot'` on the front end machine of the Scyld Beowulf cluster.

**bootmodule modules** ...

The bootmodule directive controls which kernel modules will be included in phase 2 boot images. Only include network drivers in this list. Once the network is up and BProc is started, the front end will then download and install more modules for other types of devices such as SCSI controllers.

There may be more than one bootmodule directive and each one may contain several modules. You should not include the ".o" extension in the module names.

**bootport port**

The bootport directive controls from which TCP port the boot server will serve the boot image.

**bprocport port**

The bprocport directive controls from which TCP port the bproc server will run.

**insmod module args...**

The insmod directive causes a module to be loaded without dependency checking. Do not include the ".o" extension in the module name.

**modarg module args...**

The modarg tag specifies arguments for modules. These arguments are used when doing a `modprobe` on a module without arguments. This is the way to get module arguments for modules loaded during the PCI scan.

**moddep module dependencies...**

The moddep tag specifies module dependencies. The first argument is a module name. The remaining arguments are a list of modules which should be loaded before attempting to load the first module. If loading of any of the other modules fails, then loading of the first module will not be attempted.

Do not include the ".o" extension as part of the module names. Normally, module dependency information is automatically generated by the `beoboot` script.

**modprobe module args...**

The modprobe directive causes a module to be loaded with dependency checking. Do not include the ".o" extension in the module name.

**pci vendor device driver**

The pci tag specifies which driver supports a particular PCI device. The vendor and device ID numbers can be either decimal or hexadecimal with the "0x" notation. The driver is the name of the module which supports the device.
You should not include the "\,.o" extension in the driver name. If the driver requires arguments or has dependencies, these are specified with ‘modarg’ and ‘moddep’.
Appendix C References

How to Build A Beowulf: A Guide to the Implementation and Application of PC Clusters
Thomas L. Sterling, John Salmon, Donald J. Becker, et al
1999, MIT Press

Parallel Programming with MPI
Peter S. Pacheco
1997, Morgan Kaufman

MPI: The Complete Reference
Marc Snir, Steve W. Otto, Steven Huss-Lederman, et al
1996, MIT Press
ISBN 0-262-69184-1

Using MPI: Portable Parallel Programming with the Message-Passing Interface
William Gropp, Ewing Lusk, Anthony Skjellum
1995, MIT Press
ISBN 0-262-57104-8
Appendix D GNU General Public License

Version 2, June 1991
Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation’s software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author’s protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors’ reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone’s free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program. You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions
for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you
indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.
10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.
<one line to give the program’s name and a brief idea of what it does.> Copyright (C) 19yy <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

    Gnomovision version 69, Copyright (C) 19yy name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’. This is free software, and you are welcome to redistribute it under certain conditions; type ‘show c’ for details.

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than ‘show w’ and ‘show c’; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

    Yoyodyne, Inc., hereby disclaims all copyright interest in the program ‘Gnomovision’ (which makes passes at compilers) written by James Hacker.

    <signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.
Index

C
C Library Interface ........................................ 21
config ..................................................... 42
config.boot ................................................ 44

B
beoboot ..................................................... 10
beoboot-install ............................................. 11
beoserv ..................................................... 11
bootfile ..................................................... 42
bootmodule .................................................. 44
bootport ...................................................... 42, 44
bpcp .......................................................... 19
bpctl .......................................................... 18
bpmaster ...................................................... 17
bpproc currnode ............................................. 21
bpproc execmove ........................................... 22
bpproc_masteraddr .......................................... 22
bpproc_move ................................................ 22
bpproc_node_down .......................................... 21
bpproc_node_error ......................................... 22
bpproc_node_unavailable .................................. 22
bpproc_node_up ............................................. 22
bpproc_nodeaddr ............................................ 22
bpproc_nodestatus ......................................... 21
bpproc_numnodes ........................................... 21
bpproc_rexec ................................................. 22
bpproc_rfork ............................................... 22
bpproc_setnodelist ......................................... 23
bpproc_slave_chroot ....................................... 23
bpprocport .................................................. 42, 44
bpsch ........................................................ 19
bpslave ........................................................ 17
bpstat ........................................................ 18

A
adding a network driver .................................. 12
address ....................................................... 42
allowinsecureports ......................................... 42

D
down ........................................................... 15

F
fsck ............................................................ 42

H
halt ............................................................. 15

I
ignore ........................................................ 42
Index .......................................................... 53
insmod ........................................................ 44
Intel PPro Performance Counter Support ............... 31
interface ..................................................... 42
iprange ....................................................... 42

L
libraries .................................................... 43
logfacility ................................................... 43

M
migration ..................................................... 21
mkfs .......................................................... 43
modarg ....................................................... 44
moddep ....................................................... 44
modprobe .................................................... 44
mpirun ....................................................... 26

N
netmask ....................................................... 43
network driver, adding to beoboot ....................... 12
node .......................................................... 43
node states ................................................... 15

Countable Events ........................................... 34
<table>
<thead>
<tr>
<th>Index</th>
<th>54</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>P</strong></td>
<td></td>
</tr>
<tr>
<td>pci</td>
<td>44</td>
</tr>
<tr>
<td>pentium pro performance counter support</td>
<td>31</td>
</tr>
<tr>
<td>PERF_ANY</td>
<td>38</td>
</tr>
<tr>
<td>PERF_CACHE_ALL</td>
<td>39</td>
</tr>
<tr>
<td>PERF_CACHE_E</td>
<td>39</td>
</tr>
<tr>
<td>PERF_CACHE_I</td>
<td>39</td>
</tr>
<tr>
<td>PERF_CACHE_M</td>
<td>38</td>
</tr>
<tr>
<td>PERF_CACHE_S</td>
<td>39</td>
</tr>
<tr>
<td>PERF_COUNTERS</td>
<td>32</td>
</tr>
<tr>
<td>perf_get_config</td>
<td>32</td>
</tr>
<tr>
<td>PERF_OS</td>
<td>39</td>
</tr>
<tr>
<td>perf_read</td>
<td>32</td>
</tr>
<tr>
<td>perf_reset</td>
<td>32</td>
</tr>
<tr>
<td>PERF_SELF</td>
<td>38</td>
</tr>
<tr>
<td>perf_set_config</td>
<td>32</td>
</tr>
<tr>
<td>perf_start</td>
<td>32</td>
</tr>
<tr>
<td>perf_stop</td>
<td>32</td>
</tr>
<tr>
<td>perf_sys_get_config</td>
<td>33</td>
</tr>
<tr>
<td>perf_sys_read</td>
<td>33</td>
</tr>
<tr>
<td>perf_sys_reset</td>
<td>33</td>
</tr>
<tr>
<td>perf_sys_set_config</td>
<td>33</td>
</tr>
<tr>
<td>perf_sys_start</td>
<td>33</td>
</tr>
<tr>
<td>perf_sys_stop</td>
<td>33</td>
</tr>
<tr>
<td>perf_sys_write</td>
<td>33</td>
</tr>
<tr>
<td>PERF_US</td>
<td>39</td>
</tr>
<tr>
<td>PERF_USR</td>
<td>39</td>
</tr>
<tr>
<td>perf_wait</td>
<td>33</td>
</tr>
<tr>
<td>perf_write</td>
<td>33</td>
</tr>
<tr>
<td>performance counter support</td>
<td>31</td>
</tr>
<tr>
<td>process migration</td>
<td>21</td>
</tr>
<tr>
<td>pwroff</td>
<td>15</td>
</tr>
<tr>
<td><strong>R</strong></td>
<td></td>
</tr>
<tr>
<td>reboot</td>
<td>15</td>
</tr>
<tr>
<td><strong>U</strong></td>
<td></td>
</tr>
<tr>
<td>unavailable</td>
<td>15</td>
</tr>
<tr>
<td>unknown</td>
<td>43</td>
</tr>
<tr>
<td>up</td>
<td>15</td>
</tr>
<tr>
<td><strong>V</strong></td>
<td></td>
</tr>
<tr>
<td>vmadlib</td>
<td>19</td>
</tr>
<tr>
<td>VMADump</td>
<td>15</td>
</tr>
</tbody>
</table>