Reference Guide

Scyld ClusterWare Release 7.4.1-741g0000

October 31, 2017

Reference Guide: Scyld ClusterWare Release 7.4.1-741g0000; October 31, 2017

Revised Edition

Published October 31, 2017

Table of Contents

Preface

Welcome to the Scyld ClusterWare *Reference Guide*. This document describes Scyld ClusterWare commands that can be invoked by the ordinary user and the maintenance utilities that are intended for the cluster administrator. It also provides in-depth information on the configuration file `/etc/beowulf/config` and the cluster node file system table `/etc/beowulf/fstab`. The document also includes an extensive library/function reference for the **beostat** Beowulf Status library and the **BProc** Beowulf Process Control library.

The Scyld Beowulf functionality is implemented through several packages, notably the following:

- The **BProc** package, which implements the Scyld **BProc** unified process space functionality.
- The **libbeostat** package, which monitors the state of the compute nodes and gathers performance metrics, making these metrics available through a library API

This *Reference Guide* is written with the assumption that the reader has a background in a Linux or Unix operating environment. Therefore, this document does not cover basic Linux system use, administration, or application development.

Feedback

We welcome any reports on errors or difficulties that you may find. We also would like your suggestions on improving this document. Please direct all comments and problems to support@penguincomputing.com.

When writing your email, please be as specific as possible, especially with errors in the text. Please include the chapter and section information. Also, please mention in which version of the manual you found the error. This version is *Scyld ClusterWare, Revised Edition*, published October 31, 2017.

Scyld ClusterWare Commands

This section of the *Reference Guide* describes the Scyld ClusterWare commands that are intended to be invoked by the ordinary user. Most of the commands are found in the directory `/opt/scyld/bin`.

bdate

Name

bdate — Set the time on a compute node

Synopsis

/usr/lib/beoboot/bin/bdate *node*

Description

This program sets the time on a compute node to the time currently on the master. The program only takes one option, the number of the node you wish to set the time on.

This program is usually run from the node_up script so that the time gets set on the compute nodes at boot.

Options

The following options are available to the **bdate** program.

node

The number of the node to set the time on.

Examples

Updating the time on node 1:

```
[user@cluster user] $ /usr/lib/beoboot/bin/bdate 1
```

Note that **bdate** is not normally in the path, and the full executable path must be specified.

beoboot

Name

beoboot — Generate Scyld ClusterWare boot images

Synopsis

/usr/bin/beoboot [-h] [-v] [-2] [-a] [-i] [-n] [-o `output_file`] [-L `dir`, --libdir `dir`] [-k `kernimg`, --kernel `kernimg`] [-c `cmdline`, --cmdline `cmdline`] [-m `dir`, --modules `dir`]

Description

beoboot is a booting system for Scyld clusters. The **beoboot** program creates boot images that implement the compute node side of the cluster boot system.

The final boot image is provided by one or more master nodes designated as "boot masters". This final boot image has the run-time kernel and initial information needed to contact an operational master.

A final boot image may be created for use on a floppy disk or CD. This eliminates an extra phase during boot, but requires updating the image each time the kernel, network device drivers, or other process creation components are updated on the master.

Options

-h

Display a help message and exit.

-v

Display version information and exit.

-2

Create a phase 2 image. This image contains the final kernel to run.

-i

Create stand-alone images (kernel and ramdisk). These images will be appropriate for use with other boot mechanisms. The kernel and ramdisk image will be stored in: outfile and outfile.initrd

-n

Create a netboot image. If no output_file argument is specified, then the image file will be named `/var/beowulf/boot.img`.

-o `output_file`

Set output filename to `output_file`.

-L `dir`, --libdir `dir`

Find beoboot files in `dir` instead of `/usr/lib/beoboot/`

-k `kernimg`, --kernel `kernimg`

Use `kernimg` as the kernel image instead of the image given in the configuration file (final boot image only).

If this is not specified on the command line, the default is taken out of `/etc/beowulf/config`. If it is not specified there, `/boot/vmlinuz` is used.

-c `cmdline`, --cmdline `cmdline`

Use the command line cmdline instead of the command line given in the configuration file.

If this is not specified on the command line, the default is taken out of `/etc/beowulf/config`.

-m `dir`, --modules `dir`

Look for modules matching the kernel image in `dir` instead of `/lib/modules/<kernelversion>`, which is the default.

PXE-Media Package

The old Phase 1 boot images are no longer supported. If you need to boot from removeable media, the PXE-media package includes support for booting from a floppy or CD using PXE.

The PXE-media images are located in `/usr/share/pxe-media`. There are two image files, `pxe-media.floppy` (for copying to floppy disk) and `pxe-media.iso` (for copying to CD-R). Note the following:

- Creating a boot CD: Use **cdrecord** or a similar program to burn the .iso image to a boot CD-R (one for each compute node). Then boot each compute node with one of the CDs inserted in the CD-ROM drive. You must have your compute nodes configured to boot from CD for this to work.
- Creating a boot floppy: Use the **dd** command to copy the floppy image to a floppy disk (one for each compute node). Then boot each compute node with one of the floppies inserted in the floppy drive.

> **Caution**
>
> When you are making a final boot image, you must be running the kernel you are putting in the image, whether this kernel is specified on the command line or in `/etc/beowulf/config`.

Examples

Creating a final boot image:

```
[user@cluster user] $ beoboot -2 -n
  Building phase 2 file system image in /tmp/beoboot.6684...
  ram disk image size (uncompressed): 1888K
  compressing...done
  ram disk image size (compressed): 864K
  Kernel image is:    "/tmp/beoboot.6684".
  Initial ramdisk is: "/tmp/beoboot.6684.initrd".
  Netboot image is in: /var/beowulf/boot.img
```

beoconfig

Name

beoconfig — Operate on Scyld ClusterWare cluster configuration files.

Synopsis

beoconfig [-a, --all *"string"*] [-c, --config *file*]
[-i, --insert *"string"*] [-r, --replace *"string 1" "string 2"*]
[-d, --delete *"string"*] [-D, --deleteall *"string"*]
[-n, --node *nodes*] [-w, --withcomments]
[-l, --syslog] [-h, --help] [-u, --usage] [-V, --version]

Description

beoconfig manipulates a ClusterWare cluster configuration file to insert, replace, or delete *"string"* entries. A *"string"* consists of an initial keyword, plus zero or more parameters, plus an optional comment. This utility is commonly used in script files to retrieve parameters from the config file.

Options

The following options are available to the **beoconfig** program.

-a, --all

Return all entries with specified keyword (default).

-c *file*, --config *file*

Read configuration file *file*. Default is `/etc/beowulf/config`.

-d, --delete *"delete string"*

Delete the specified string from the config file

-D, --deleteall *"delete string"*

Delete ALL instances of specified string from the config file.

-h, --help

Show a usage message.

-i, --insert *"insert string"*

Append the specified string to the config-file if it does not already exist. When inserting a "node" entry, you must also specify a **--node** *nodes* argument.

-l, --syslog

Log error messages to syslog

-n, --node *nodes*

Perform action on specified node(s) only. Nodes can be specified individually, as ranges, or as a comma-separated list of individual nodes and/or ranges.

-r, --replace *"search string" "replacement string"*

Replace "search string" with "replacement string".

-u, --usage

Show a usage summary.

-V, --version

Show this version number.

-w, --withcomments

Show entries including comments.

Examples

```
[user@cluster user] $ export CLUSTERDEV="beoconfig interface"
[user@cluster user] $ $CLUSTERDEV
eth1
```

View MAC addresses for ALL nodes in the node-db

```
[user@cluster user] $ beoconfig -a "node"
00:50:45:01:03:68 00:50:45:01:03:69
00:50:45:5C:29:F6 00:50:45:5C:29:F7
00:50:45:BB:A6:EA 00:50:45:BB:A6:EB
off
off
00:50:45:CD:BE:61
```

Demonstrate --withcomments

```
[root@cluster ~]#  beoconfig --insert "newkeyword param1 # with comment"
[root@cluster ~]#  beoconfig newkeyword
param1
[root@cluster ~]#  beoconfig -w newkeyword
param1 # with comment
# Now remove the unnecessary "newkeyword" entry:
[root@cluster ~]#  beoconfig -d newkeyword
```

Enable only Nodes 1, 2 and 8, to use IPMI

```
[root@cluster ~]#  beoconfig --node 1 --insert "ipmi enabled"
[root@cluster ~]#  beoconfig --node 8 --insert "ipmi enabled"
```

```
[root@cluster ~]#  beoconfig --node 2 --insert "ipmi enabled"
[root@cluster ~]#  beoconfig --node 1 "ipmi"
enabled
[root@cluster ~]#  beoconfig --node 3 "ipmi"
disabled
```

Looking in `/etc/beowulf/config`, one will see that the 'ipmi' keyword has a global value of 'disabled'. However, there is another 'ipmi' keyword entry, and this one has an embedded node-set.

```
[root@cluster ~]# cat /etc/beowulf/config | grep ipmi
ipmi disabled
ipmi 1-2,8 enabled
```

Replace functionality: Suppose the config file has a 'nodewake' line to invoke an IPMI version 2.0 script for nodes 1 through 10, but node 5 has been replaced with a machine that supports only IPMI version 1.5. An admin must now replace node 5's nodewake script.

```
[root@cluster ~]#  cat /etc/beowulf/config | grep nodewake
#nodewake /usr/lib/beoboot/bin/node_wake_ipmi
nodewake 1-10 /nfs/support/scripts/nodeup_ipmiv2.0
[root@cluster ~]#  beoconfig --node 5 --replace "nodewake *" "nodewake /nfs/support/scripts/nodeup_ipmiv1.5"
[root@cluster ~]#  cat /etc/beowulf/config | grep nodewake
#nodewake /usr/lib/beoboot/bin/node_wake_ipmi
nodewake 1-4,6-10 /nfs/support/scripts/nodeup_ipmiv2.0
nodewake 5 /nfs/support/scripts/nodeup_ipmiv1.5
```

Insert node-holes for node's 0 through 9, while adding a MAC address for node 10

```
[root@cluster ~]#  beoconfig -a node
[root@cluster ~]#  beoconfig -i "node 00:11:22:33:44:55 " --node 10
[root@cluster ~]#  beoconfig -a node
off ##node number 000
off ##node number 001
off ##node number 002
off ##node number 003
off ##node number 004
off ##node number 005
off ##node number 006
off ##node number 007
off ##node number 008
off ##node number 009
node 00:11:22:33:44:55
```

Get the MAC address for node 10

```
[root@cluster ~]# beoconfig -n 10 node
00:11:22:33:44:55
```

See Also

beowulf-config(5)

beomap

Name

beomap — Show a job map from the **beomap** scheduler.

Synopsis

beomap [-h, --help] [-V, --version]
[--all-cpus] [--all-nodes] [--all-local] [--no-local]
[--map *nodelist*] [--exclude *nodelist*] [--np *processes*]

Description

This program retrieves a job map from the currently installed **beomap** scheduler. This is the same job map that would be used by an integrated application (such as **beorun**, **mpirun**, or **mpprun**) started with the same scheduling parameters at that instant in time.

The **beomap** command may be used to generate a job map for applications that do not have their own scheduler interface, in scripts, or to examine the current scheduling state of the system.

You can influence the job map either by setting environment variables or by entering command line options. Note that command-line options take precedence over the environment variable settings.

Options

The following general command line options are available to **beomap**. Also see the next section, which describes the job map parameters.

-h, --help

Print the command usage message and exit. If **-h** is in the option list, all other options will be ignored.

-V, --version

Print the command version number and exit. Any other options will be parsed and handled.

Job Map Parameters

You can influence the **beomap** job map either by entering command line options or by setting environment variables. Following are the available command line options, together with their equivalent environment variables. Note that the command line options take precedence over the environment variables.

All of the **beomap** job map parameters listed below can also be used directly with **beorun**, **mpirun**, and **mpprun**.

--all-cpus

Create a process map consisting of all "up" nodes, with each node number repeated to represent the number of CPUs on that node. This parameter is not allowed in conjunction with the **--map** parameter.

The equivalent environment variable is **ALL_CPUS**.

--all-nodes

Create a process map consisting of all "up" nodes, with 1 CPU mapped on each of the "up" nodes. This parameter is not allowed in conjunction with the **--map** parameter.

The equivalent environment variable is **ALL_NODES**.

--all-local

Create a process map consisting entirely of master node entries. This option eliminates everything except node -1 from the pool of candidate node numbers, thus forcing the map to use node -1 for everything. This parameter is not allowed in conjunction with the **--map** parameter.

This option is a handy shortcut for troubleshooting connectivity problems or testing on an isolated head node before running a job on a "live" cluster.

The equivalent environment variable is **ALL_LOCAL**.

--no-local

Exclude the master in the process map. This option is essentially a syntactic shortcut for including -1 in the **--exclude *nodelist*** option. For MPI jobs, this option puts the "rank 0" job on a compute node instead of the master. This parameter is not allowed in conjunction with the **--map** parameter.

The equivalent environment variable is **NO_LOCAL**.

--exclude *nodelist*

Do not include the listed nodes in the process map; the nodes to be excluded are stated as a colon-delimited list. This parameter is not allowed in conjunction with the **--map** parameter.

The equivalent environment variable is **EXCLUDE=*nodelist***.

--map *nodelist*

Specify a process map consisting of a colon-delimited list of nodes. Each node in the list indicates where one process will be assigned. The number of entries in the job map implies the number of ranks in the job.

Listing a node more than once in the list will assign multiple processes to that node. Typically, this is done to assign one process to each processor or core on a node, but this can also be used to assign more processes to a node than it has processors or cores.

The **--all-cpus**, **--all-nodes**, **--np *processes***, **--all-local**, **--no-local**, and **--exclude** parameters are not allowed in conjunction with the **--map** parameter.

The equivalent environment variable is **BEOWULF_JOB_MAP=*nodelist***.

--np *processes*

Specify the number of processes to run. The **beomap** command attempts to place one process per processor or core, but will assign multiple processes per processor or core if there are not enough individual processors or cores available. This parameter is not allowed in conjunction with the **--map** parameter.

The equivalent environment variable is **NP=*processes***.

Tip: The environment variables have an order of priority. The **BEOWULF_JOB_MAP** variable acts as a "master override" for the other environment variables. If **BEOWULF_JOB_MAP** is not set, the following priorities apply:

- Three of the environment variables determine *how many ranks* to schedule in the map: **ALL_CPUS** (priority 1), **ALL_NODES** (priority 2), and **NP** (priority 3). If none of these are set explicitly by the user, then a usage of NP=1 is assumed.
- Three of the environment variables determine *what node numbers* are candidates for being mapped: **ALL_LOCAL** (priority 1), **NO_LOCAL** (priority 2), and **EXCLUDE** (priority 3).

> **Caution**
>
> It is an error to use **NO_LOCAL** and **ALL_LOCAL** together. If both are used, **ALL_LOCAL** will take precedence.

Examples

Find the set of machines available for use:

```
[user@cluster ~] $ beomap --all-cpus
  -1:0:1:2:3:4:5:6:7:8:9:10:11:12:13:14:15
```

Create a process map to run 20 processes on a cluster with 10 idle dual-processor compute nodes:

```
[user@cluster user] $ beomap --np 20
  -1:0:0:1:1:2:2:3:3:4:4:5:5:6:6:7:7:8:8:9
```

Note: Since **--no-local** was not specified, the master node (listed as "-1") is included in the map, and node 9 is listed only once.

Select an available machine to start up an application, while handling application termination or machine failure; note that the following works only for the sh family of shells (bash):

```
[user@cluster user] $ while :; do export NODE=`beomap --no-local -np 1`; bpsh $NODE application-to-run; done
```

Provide an explicit map to run 5 processes on node 0:

```
[user@cluster user] $ beomap --np 5 --map 0:0:0:0:0
```

Special Notes

The underlying **beomap** system calls pluggable schedulers, which may use arbitrary scheduling inputs. The command line options replace and delete environment variables used by the Scyld-provided default schedulers/mappers, but other schedulers are free to ignore these advisory settings. Specifically, the **beomap** command does not confirm that the parameters, such as **--no-local**, are true in the resulting job map.

See Also

beorun(1), bpsh(1), beostatus(1), beonpc(1), mpprun(1), and the User's Guide.

beonpc

Name

beonpc — Show the count of all user processes started by this master running on the specified compute node.

Synopsis

beonpc [-h, --help, -u, --usage] [-V, --version]
[-p, --pids] [-s, --sum] *node*

Description

The **beonpc** program prints the count of running processes on the specified cluster node. The count includes only the processes started by the current machine and running on the specified node.

The **beonpc** program prints "-1" for nodes that are not controlled by this master or are otherwise inaccessible.

The **beonpc** command is typically used to make and observe scheduling and job mapping decisions.

Options

The following options are available to the **beonpc** program.

-h, --help, -u, --usage

Print the command usage message on `stdout` and exit. When one of these options is recognized in the option list, all following options will be ignored.

-V

Print the command version number on `stdout` and exit. Any following options will be ignored.

-p, --pids

Show the process IDs in a process list.

-s, --sum

Emit only a total cluster process count.

node

Optionally, show for the specific node number, or *all* (the default) for all nodes, or *list* for nodes with a nonzero count.

Example

Find the number of jobs this master is running on cluster compute node 23:

```
[user@cluster user] $ beonpc 23
3
```

See Also

beomap(1), bpsh(1), beostatus(1)

beorun

Name

beorun — Run a job on a Scyld cluster using dynamically selected nodes.

Synopsis

beorun [-h, --help] [-V, --version]
[--all-cpus] [--all-nodes] [--all-local] [--no-local]
[--map *nodelist*] [--exclude *nodelist*] [--np *processes*]
command [*command-args...*]

Description

The **beorun** program runs the specified program on a dynamically selected set of cluster nodes. It generates a job map from the currently installed **beomap** scheduler, and starts the program on each node specified in the map. The scheduling parameters from the command line and environment are the same as for **beomap**, and the resulting job map is identical to the job map that **beomap** would if generate at that instant in time for that program name.

The **beorun** command may be used to start applications that are not cluster-aware or do not have their own scheduler interface.

Options

The following general command line options are available to **beorun**. Also see the next section, which describes the **beomap** job map parameters.

-h, --help, -u, --usage

Print the command usage message on `stdout` and exit. When one of these options is recognized in the option list, all following options will be ignored.

-V

Print the command version number on `stdout` and exit. Any following options will be ignored.

Unrecognized options and invalid option formats are reported on `stderr` and the command exits with exit status 1 (invalid option) or 2 (no command specified or invalid command).

NOTE: **beorun** does not pass information from `stdin` to applications. In cases where an application must read data from `stdin`, it is suggested that **bpsh** be used instead. Please see the **bpsh** man page for usage information; command line options for **bpsh** are similar to those for **beorun**, but not exactly the same.

Job Map Parameters

You can influence the **beomap** job map either by entering command line options or by setting environment variables. Following are the available command line options, together with their equivalent environment variables. Note that the command line options take precedence over the environment variables.

All of the **beomap** job map parameters listed below can also be used directly with **beorun**, **mpirun**, and **mpprun**.

--all-cpus

Create a process map consisting of all "up" nodes, with each node number repeated to represent the number of CPUs on that node. This parameter is not allowed in conjunction with the **--map** parameter.

The equivalent environment variable is **ALL_CPUS**.

--all-nodes

Create a process map consisting of all "up" nodes, with 1 CPU mapped on each of the "up" nodes. This parameter is not allowed in conjunction with the **--map** parameter.

The equivalent environment variable is **ALL_NODES**.

--all-local

Create a process map consisting entirely of master node entries. This option eliminates everything except node -1 from the pool of candidate node numbers, thus forcing the map to use node -1 for everything. This parameter is not allowed in conjunction with the **--map** parameter.

This option is a handy shortcut for troubleshooting connectivity problems or testing on an isolated head node before running a job on a "live" cluster.

The equivalent environment variable is **ALL_LOCAL**.

--no-local

Exclude the master in the process map. This option is essentially a syntactic shortcut for including -1 in the **--exclude *nodelist*** option. For MPI jobs, this option puts the "rank 0" job on a compute node instead of the master. This parameter is not allowed in conjunction with the **--map** parameter.

The equivalent environment variable is **NO_LOCAL**.

--exclude *nodelist*

Do not include the listed nodes in the process map; the nodes to be excluded are stated as a colon-delimited list. This parameter is not allowed in conjunction with the **--map** parameter.

The equivalent environment variable is **EXCLUDE=*nodelist***.

--map *nodelist*

Specify a process map consisting of a colon-delimited list of nodes. Each node in the list indicates where one process will be assigned. The number of entries in the job map implies the number of ranks in the job.

Listing a node more than once in the list will assign multiple processes to that node. Typically, this is done to assign one process to each processor or core on a node, but this can also be used to assign more processes to a node than it has processors or cores.

The **--all-cpus**, **--all-nodes**, **--np *processes***, **--all-local**, **--no-local**, and **--exclude** parameters are not allowed in conjunction with the **--map** parameter.

The equivalent environment variable is **BEOWULF_JOB_MAP=*nodelist***.

--np *processes*

Specify the number of processes to run. The **beomap** command attempts to place one process per processor or core, but will assign multiple processes per processor or core if there are not enough individual processors or cores available. This parameter is not allowed in conjunction with the **--map** parameter.

The equivalent environment variable is **NP=*processes***.

Tip: The environment variables have an order of priority. The **BEOWULF_JOB_MAP** variable acts as a "master override" for the other environment variables. If **BEOWULF_JOB_MAP** is not set, the following priorities apply:

- Three of the environment variables determine *how many ranks* to schedule in the map: **ALL_CPUS** (priority 1), **ALL_NODES** (priority 2), and **NP** (priority 3). If none of these are set explicitly by the user, then a usage of NP=1 is assumed.
- Three of the environment variables determine *what node numbers* are candidates for being mapped: **ALL_LOCAL** (priority 1), **NO_LOCAL** (priority 2), and **EXCLUDE** (priority 3).

Caution

It is an error to use **NO_LOCAL** and **ALL_LOCAL** together. If both are used, **ALL_LOCAL** will take precedence.

Examples

Run **uptime** on any two available cluster compute nodes:

```
[user@cluster user] $ beorun --np 2 --no-local uptime
  11:05am  up 2 days, 11:16,  0 users,  load average: 0.05, 0.24, 0.65
  11:05am  up 2 days, 11:16,  0 users,  load average: 0.01, 0.07, 0.37
```

See Also

beomap(1), bpsh(1), mpprun(1), beonpc(1), and the User's Guide.

beosi

Name

beosi — Collects or extracts cluster configuration information.

Synopsis

beosi [-m] [-n] [-I] [-d *file*] [-h] [-v]

Description

The primary function of the **beosi** utility is to collect configuration and state information from the master node and/or compute nodes on a Scyld ClusterWare cluster, organize the information into ASCII files within a new directory in the current working directory, and finally **tar** that directory and **uuencode** the gzipped tarball into a compressed, portable archive file that can be saved locally or transmitted (e.g., by ftp or email). **beosi** can also be used to **uudecode** a previously assembled archive to reform the tarball for later extraction as desired.

beosi should be executed with root access, since much of the interesting information can only be accessed as root. The *-m* and *-n* options are typically used together to produce a directory named `conf-`*date*, where *date* indicates the current year, month, and day. The *-m* option creates ASCII files in the subdirectory `conf-`*date*`/master/`, and the *-n* option creates ASCII files in per-node subdirectories, e.g., `conf-`*date*`/Node0/` and `conf-`*date*`/Node1/`. The **beosi** default end product is an archive file named `conf-`*date*`.encoded`.

beosi can be used to capture configuration information for later retrieval and comparison. For example, if the current configuration is working, you can execute **beosi** and store the archive file for safekeeping. If a subsequent configuration change causes your cluster to stop working, then you can create another archive, extract both the new archive and the previous archive, and examine the differences between the two configurations (e.g. using **diff**) to determine which change caused the problem.

Options

The following options are available to the **beosi** utility.

-m

Collect information about the master node, typically used together with the *-n* option. By default, produces a uuencoded gzipped tar file in the current directory called `conf-`*date*`.encoded`

-n

Collect information about individual (compute) nodes, typically used together with the *-m* option. By default, produces a uuencoded gzipped tar file in the current directory called `conf-`*date*`.encoded`

-I

The *-I* option overrides the default action of converting the `conf-`*date* directory of captured information into a uuencoded gzipped tar file. Thus, the directory of information is retained in its fully "exploded" form, `conf-`*date*, which allows the cluster administrator to view all the information that would otherwise be bundled into the uuencoded gzipped tar file. This is especially useful to discern why beosi might be producing an unexpectedly large uuencoded file.

-d *file*

Decodes information from an archive created previously by **beosi**. The result is a gzipped tar file with the same root filename.

-h

Display a summary of beosi command arguments.

-v

Display program version information and exit.

Examples

To inspect the configuration information on the master node, first run:

```
[root@cluster ~]# beosi -m
[root@cluster ~]# ls
  conf-15-07-22.encoded
```

Then extract the information:

```
[root@cluster ~]# beosi -d conf-15-07-22.encoded
[root@cluster ~]# tar -zxvf conf-15-07-22.tar.gz
  conf-15-07-22/
  conf-15-07-22/master/
  conf-15-07-22/master/dmesg
  conf-15-07-22/master/lsmod
  conf-15-07-22/master/syslog
  ...
```

Alternatively, avoid producing a uuencoded tar file and thus retain the fully explorable directory of information:

```
[root@cluster ~]# beosi -m -I
[root@cluster ~]# ls
  conf-15-07-22
```

Use a prior configuration to identify individual files that differ:

```
[root@cluster ~]# diff -r --brief conf-15-06-30 conf-15-07-22
  Files conf-15-06-30/master/ifconfig and conf-15-07-22/master/ifconfig differ
  Files conf-15-06-30/master/lsmod and conf-15-07-22/master/lsmod differ
  Files conf-15-06-30/master/network and conf-15-07-22/master/network differ
  Files conf-15-06-30/master/proc_buddyinfo and conf-15-07-22/master/proc_buddyinfo differ
  ...
```

Use a prior configuration to compare individual files:

```
[root@cluster ~]# diff -u conf-15-06-30/master/lsmod conf-15-07-22/master/lsmod
```

```
--- conf-15-06-30/master/lsmod 2009-12-14 09:19:45.000000000 -0800
+++ conf-15-07-22/master/lsmod 2015-07-21 18:27:31.000000000 -0800
@@ -1,11 +1,12 @@
 Module                  Size  Used by
+iptable_filter          7745  0
 bproc                 181208  2
 task_packer            24708  1 bproc
 filecache              28220  2 bproc,task_packer
 ipt_MASQUERADE          9025  1
 iptable_nat            34149  2 ipt_MASQUERADE
 ip_conntrack           57369  2 ipt_MASQUERADE,iptable_nat
-ip_tables              25537  2 ipt_MASQUERADE,iptable_nat
+ip_tables              25537  3 iptable_filter,ipt_MASQUERADE,iptable_nat
 nfsd                  274657  17
 exportfs               10945  1 nfsd
 lockd                  82833  2 nfsd
```

To gather complete information about the cluster (i.e., master and all **up** compute nodes):

```
[root@cluster ~]# beosi -m -n
[root@cluster ~]# ls
  conf-15-07-22.encoded
```

See Also

bpsh(1), beostat(1), beoconfig(1), **uuencode(1p)**, **tar(1)**

beostatus

Name

beostatus — Display status information about the cluster.

Synopsis

beostatus [--classic] [-c, --curses] [-H, --html]
[-C, --combined-spider] [-d, --dots] [-l, --levometer]
[-p, --pie] [-s, --spider] [-S, --stripchart]
[-r, --remote=*host*] [-P, --port=*port*] [-U, --user=*name*]
[--disable-ssl] [-u *seconds*, --update=*seconds*]
[-v, --version] [-h, --help]

Description

Beostatus is a utility that displays status information for the master node and all compute nodes in the cluster.

The default display is a graphical user interface (GUI) known as the "Classic" mode, which is a tabular format, one row per node, showing per-node specific state and resource usage information. An optional non-GUI "Curses" display mode is also available that shows the same per-node information as the "Classic" mode: the assigned number for each node, the node state, CPU usage, memory usage, swap space usage, root filesystem usage, and network bandwidth usage.

Alternate GUI display modes can be selected by **beostatus** command line option or by using a pulldown menu within each of the GUI displays.

Various filtering options are available in the "Curses" display mode that limit the displayed information to nodes that are being currently utilized by a specified user. Note: filtering functionality requires that TORQUE be installed on the cluster.

Beostatus can also be used to access cluster state information for a remote node. This requires the presence of **beoweb** functionality on the remote node.

See the Administrator's Guide for additional information about **beostatus**.

Options

The following options are available to the **beostatus** utility:

--classic

Display output in GUI "Classic" mode. This is the default display mode.

-c, --curses

Display output in non-GUI "Curses" mode. It displays the same information as the GUI "Classic" mode. It is appropriate for simple terminal windows and when X is unavailable.

-C --combined-spider

Display output in GUI "Combined Spider" mode.

-d --dots

Display output in GUI "Dots" mode. Each node is represented by a colored box. The user selects the status element that the box represents (e.g., node state or CPU utilization), and different colors indicate different status values for that element (e.g., node state "up" vs. "down", or gradations of CPU loading).

-H, --html

Display output in HTML format.

-l --levometer

Display output in GUI "Levometer" mode.

-p --pie

Display output in GUI "Piechart" mode.

-P *port*, --port=*port*

When retrieving remote beostatus information, override the default port number 5000 with another *port* value.

-r *host*, --remote=*host*

Retrieve beostatus information from a remote *host*.

-s --stripchart

Display output in GUI "Stripchart" mode.

-u *seconds*, --update=*seconds*

Override the default update rate of 5 seconds. Units are integer seconds.

-U *name*, --user=*name*

When retrieving remote beostatus information, authenticate as user *name*.

--disable-ssl

When retrieving remote beostatus information, don't use SSL encryption.

-v, --version

Show version information.

-h, --help

Show usage information and exit. If *-h* is one of the first two options, all other options will be ignored. If *-h* is not one of the first two options, it will be ignored.

FILTERING OPTIONS

Various filtering options are available when in "Curses" mode. Each is enabled by a single lowercase letter keystroke, and is disabled by a matching uppercase letter keystroke:

f, F

Limit the display to only those nodes that are running TORQUE jobs for a specific user. If the current user is root, then **beostatus** prompts for a username; otherwise, the username defaults to the current user.

j, J

Limit the display to only those nodes that are running TORQUE jobs for a specific user, and yet those jobs have no processes actually executing. If the current user is root, then **beostatus** prompts for a username; otherwise, the username defaults to the current user.

p, P

Limit the display to only those nodes that are running processes (irrespective of TORQUE) for a specific user. If the current user is root, then **beostatus** prompts for a username; otherwise, the username defaults to the current user.

z, Z

Limit the display to only those nodes that are running TORQUE jobs for any user, and yet those jobs have no processes actually executing.

q, Q

Terminate the **beostatus** utility.

Examples

Print cluster status. Use "q" to exit continuously updating display:

```
[user@cluster user] $ beostatus -c
       BeoStatus - 3.0
Node  State  CPU 0  CPU 1  CPU 2  CPU 3  Memory   Swap   Disk   Network
 -1      up   0.2%  11.5%   0.0%   0.0%   12.4%   0.0%  38.2%     36 kBps
  0      up   0.0% 100.0%                   9.6%   0.0%   2.2%     23 kBps
  1      up  90.0%   0.0%                  22.1%   0.0%   2.2%     13 kBps
  2    down
  3    down
```

beostat

Name

beostat — Display raw data from the Beostat system.

Synopsis

beostat [-h, --help] [-v, --verbose] [-N NUM, --node=NUM]
[-c, --cpuinfo] [-m, --meminfo] [-l, --loadavg]
[-n, --net] [-s, --stat] [-f, --file] [-C, --cpupercent]
[-D, --diskusage] [-R, --netrate] [-I, --idle=thres]
[-b, --brief] [--version]

Description

The **beostat** command is a low-level utility that displays data being managed by the **Beostat** package. This data is collected on each compute node and sent by the **sendstats** daemon to the **recvstats** daemon on the master node. Other commands, such as **beostatus**, present a more user-friendly higher-level picture of the cluster.

The default **beostat** command shows all available information on all "up" nodes. The *--verbose* option shows all nodes, including "down" nodes. Various other options constrain the output to show more specific information. You may present multiple options to see multiple data classifications.

Options

The following options are available to the **beostat** command.

-v, --verbose

Verbose, display information on all nodes, even if those nodes are down. Normally, **beostat** displays only "up" nodes.

-N NUM, --node=NUM

Show only the data for node NUM rather than for all nodes.

-c, --cpuinfo

Display the CPU model information.

-m --meminfo

Display memory statistics.

-l, --loadavg

Display load average information.

-n, --net

Display network interface information.

-s, --stat

Display CPU statistics information.

-f, --file

Display root filesystem information.

-h, --help

Display brief help and exit.

-C, --cpupercent

Convenience option to display CPU info.

-D, --diskusage

Convenience option to display disk usage of root filesystem.

-R, --netrate

Convenience option to display network rate of all interfaces.

-I, --idle=THRES

Convenience option to display number of CPUs more idle than THRES.

-b, --brief

Display convenience values (`cpupercent`, `diskusage`, `netrate`, or `idle`) with no extra text. This eliminates the need to parse the output to obtain the specific values. Only valid when used with one of the convenience options (-C, -D, -R, or -I).

--version

Display program version information and exit.

bpcp

Name

bpcp — Copies files and/or directories between cluster machines.

Synopsis

bpcp [-h] [-v] [-p] [-r] [*host1:* {*file1* }] [*host2:* {*file2* }]

Description

This utility is part of the **BProc** package and is installed by default. It is similar to the Linux **rcp** command. **bpcp** will copy files and/or directories between machines. Each file or directory is either a remote file name of the form `rhost:path` or a local file name. You must have read permission on the source and write permission on the destination. **bpcp** also handles node-to-node copies, where neither the source nor destination files are on the current node.

Options

The following options are available to the **bpcp** program.

-h

Print the **bpcp** usage message and exit. If *-h* is the first option, all other options will be ignored. If *-h* is not the first option, the other options will be parsed up to the *-h* option, but no action will be taken.

-v

Print the **bpcp** version number and exit. If *-v* is the first option, all other options will be ignored. If *-v* is not the first option, the other options will be parsed up to the *-v* option, but no action will be taken.

-p

Preserve the attributes of the source files, ignoring the umask. By default, **bpcp** will modify the time, permission bits, user and group information when the file is copied. This parameter will cause time and permission bits to be unchanged, but the user and group will change to reflect the new user.

-r

Descend source directory tree recursively and copy files and tree to destination. In this case, the destination must be a directory.

[host1:]file1

The name of the file to be copied (and optionally the name of the host it resides on if other than the local host). *file1* can be the directory name when used with the *-r* option.

[host2:]file2

The name of the file and/or host where the specified file should be copied. *file2* can be a directory name.

Examples

Copy *file1* from the master node to compute node 1 *file2* in `/home/user`. Like **cp**, the directory will not be created if it does not exist.

```
[user@cluster user] $ bpcp /home/user/file1 1:/home/user/file2
```

Copy all files and sub-directories from compute node 2 in `/home/user` to compute node 1 in `/home/user`. The directory tree will be created if it does not exist.

```
[user@cluster user] $  bpcp -r 2:/home/user/ 1:/home/user/
```

Using node 1 as an intermediary, copy `file1.txt` on node 0 to `file1.txt` on node 2.

```
[user@cluster user] $ bpsh 1 bpcp 0:/tmp/file1.txt 2:/tmp/
```

Copy `/tmp/file.txt` from the master node to the `/tmp` directory on every node in the cluster that is "up".

```
[user@cluster user] $ bpsh -a bpcp master:/tmp/file1.txt /tmp
```

Note: **bpcp** will give an "rfork: Invalid argument" message when the node is unreachable.

bpsh

Name

bpsh, bprsh — Run the program on the indicated node(s).

Synopsis

bpsh [-h] [-v]
[-a] [-A] [-L] [-p] [-s] [-d] [-b *num*]
[-n] [-N] [-I *file*, --stdin *file*]
[-O *file*, --stdout *file*] [-E *file*, --stderr *file*]
nodenumber command [*command-args*]

Description

This utility is part of the **BProc** package and is installed by default on Scyld ClusterWare systems. It is the basic mechanism for running programs on nodes, and it is patterned after the **rsh** and **ssh** commands.

Nodenumber can range from -1 (the master) to one less than the number of accessible nodes. **bpsh** will also accept a delimited list of nodes; use **-a** for all nodes that are "up" and **-A** for all nodes that are communicating (this includes the states "up", "error" and "unavailable").

bpsh will forward `stdin`, `stdout` and `stderror` for the remote processes, unless directed otherwise by **-n** or **-N** arguments. `stdin` will be duplicated for every process on each remote node selected. For a single remote process, the exit status of **bpsh** will be the exit status of that process. Non-normal exit status will also be captured and displayed. For multiple processes, **bpsh** exits with the highest exit status.

The **bprsh** utility is a variant of **bpsh**. See the EXAMPLES.

Options

The following options are available to the **bpsh** program.

-h

Print the command usage message and exit. If *-h* is the first option, all other options will be ignored. If *-h* is not the first option, the other options will be parsed up to the *-h* option, but no action will be taken.

-v

Print the command version number and exit. If *-v* is the first option, all other options will be ignored. If *-v* is not the first option, the other options will be parsed up to the *-v* option, but no action will be taken.

-a

Specifies that the command will be run on all nodes in the "up" state.

-A num

Specifies that the command will be run on all nodes in either the "up", "error", and "unavailable" states. Note that non-root users may get "BProc move failed" errors, since they are only allowed to run on "up" nodes, regardless of other node permissions.

-L *state*

Line buffer output from nodes.

-p

Prefix the node number on each output line from the node that sent it.

-s

List sequentially all the output from each node.

-d

Print a divider line between the sequential output from each node.

-b num

Set the IO line buffer size to the number of bytes. The default is 4096.

-n

Get `stdin` from `/dev/null`. On any read from `stdin`, `/dev/null` will return EOF. This is useful for any program that you background or daemonize. Like **rsh**, **bpsh** will not exit immediately if `stdin` is left open and the program has not completed. **bpsh** assumes the program may want input from `stdin`.

-N

No IO forwarding.

-I *file*, --stdin *file*

Redirect standard input from the specified file on the remote node.

-O *file*, --stdout *file*

Redirect standard output to the specified file on the remote node.

-E *file*, --stderr *file*

Redirect standard error to the specified file on the remote node.

nodenumber

The node to run the command on.

command

The command/program to run.

command-args

The arguments for *command*.

Examples

Run the **ls** command on nodes -1, 0 and 2, and prefix the node number to each line. Note, due to the way *getopt* works, the master (node -1) cannot be first in the node list.

```
[user@cluster user] $ bpsh 0,-1,2 -p ls /tmp
  -1: f1.txt
  -1: foo.txt
  0: f3.txt
  2: newfoo.txt
  2: oops.txt
```

Run the **uptime** command on nodes in the "up" state.

```
[user@cluster user] $ bpsh -a -d uptime
  0 --------------------------------------------------------------------
    2:42pm  up 2 days, 23:51,  0 users
  1 --------------------------------------------------------------------
    2:41pm  up 3 days,  5:38,  0 users
  3 --------------------------------------------------------------------
    2:42pm  up 3 days,  5:38,  0 users
```

Run a single instance of the **uptime** command on a node chosen by the scheduler, displaying the node number before the output.

```
[user@cluster user] $ bpsh `beomap --nolocal` -d uptime
  1 --------------------------------------------------------------------
    22:19:27 up 21:28,  0 users,  load average: 0.00, 0.00, 0.00
```

Run a complex *command* that consists of multiple commands that displays all the "up" nodes that have been up for less than 24 hours. Note: the **bpsh** utility expects *command* to be a single command, so to execute multiple commands you must use **bash -c** with the desired *command* in quotes:

```
[user@cluster user] $ bpsh -sap bash -c "uptime | grep -v days"
```

Alternatively, **bprsh** accepts the more complex *command* as-is, just as **rsh** would do:

```
[user@cluster user] $ bprsh -sap "uptime | grep -v days"
```

See Also

beorun(1), mpprun(1), and the User's Guide.

bpstat

Name

bpstat — Show cluster node status and cluster process mapping.

Synopsis

bpstat [-h, --help] [-V, --version] [-U, --update]
[-c, --compact] [-l, --long] [-a, --address]
[-s, --status] [-n, --number] [-t, --total]
[-N, --sort-number] [-S, --sort-status] [-O, --keep-order]
[-R, --sort-reverse]
[-p] [-P [*nodes*]]
[-A *hostname*] [-M] [*nodes* | all*state*]

Description

This utility displays the **BProc** status of cluster nodes, and processes running on those nodes. Node information includes the node's IP address, state, user ownership, group ownership, and running node user processes.

Options

The following options are available to the **bpstat** program.

-a, --address

Prints the IP address of the indicated node.

-A *hostname*

Prints the node number that corresponds to the specified hostname or IP address.

-c, --compact

Print compacted listing of nodes (default).

-h, --help

Print the command usage message and exit. If *-h* is the first option, all other options will be ignored. If *-h* is not the first option, the other options will be parsed up to the *-h* option, and those options will be processed.

-l, --long

Print long list of node status. This includes IP address, status, mode, user and group information.

-M

Prints the status of the master node, in addition to the specified compute node(s), for the default case where no specific *nodes* are specified.

-n, --number

Prints the node numbers that are being used and/or are available for the nodes in the cluster.

-N, --sort-number

Prints the node list sorted by node number.

-O, --keep-order

Prints the nodes in the order returned by the system (no sorting is done).

-p

Prints a list of processes (by PID) that are currently running on the specified node.

-P [*nodes*]

Postprocesses the output from the **ps** command, prepending the node number that **BProc**-controlled processes are running on. This is typically used as **ps aux | bpstat -P**. Processes not controlled by the **BProc** system will not have a number appended. If *nodes* is supplied, then the **ps** output is filtered to show only the specified node(s). Node(s) can be identified by names, numbers, or a list of numbers.

-R, --sort-reverse

Prints the node list in reverse sorted order.

-s, --status

Prints the state for the indicated node. The **BProc** states are "down", "boot", "error", "unavailable", "up", "reboot", "halt", and "pwroff".

-S, --sort-status

Prints the node list sorted by node status.

-t, --total

Prints the total number of compute nodes configured for the cluster. The number is calculated from the cluster configuration in the `/etc/beowulf/config` file. Note that this is the potential maximum size of the cluster, not the current number of available nodes or the count of machines assigned node numbers.

-U, --update

Continuously update the status; otherwise, print status once and exit.

-V, --version

Print the command version number and exit. If *-V* is the first option, all other options will be ignored. If *-V* is not the first option, the other options will be parsed up to the *-V* option, and those options will be processed.

[*nodes* | all*state*]

Optionally, specify the nodes for which information is to be displayed. Nodes can be specified individually, as ranges, or in a comma-separated list of individual nodes and/or ranges. Alternatively, **all***state* specifies all nodes that are in a particular state, e.g., allup, alldown, allboot, allerror. Note: **allup** does not include the master node, even if **-M** is present.

Examples

Print the number of available nodes:

```
[user@cluster user] $ bpstat --total allup
  9
```

Generate a list of all usable nodes:

```
[user@cluster user] $ bpstat --number allup
  0 1 2 4 5 10 16 17 20
[user@cluster user] $ bpstat --number allup | awk '{ print "."$1 }'
  .0 .1 .2 .4 .5 .10 .16 .17 .20
```

Print status for all nodes, including the master node:

```
[user@cluster user] $ bpstat
 Node(s)                        Status       Mode       User       Group
 8,11,22-31                     down         ---------- root       root
 3,6-7,9,12-15,18-19,21         error        ---x------ root       root
 -1,0-2,4-5,10,16-17,20         up           ---x--x--x root       root
```

Print the PIDs and associated node number of currently running processes:

```
[user@cluster user] $ bpstat -p
 PID Node
  7503 0
  8262 1
```

Print status for specific nodes:

```
[user@cluster user] $ bpstat 0-2,3,8
 Node(s)                        Status       Mode       User       Group
  8                              down         ---------- root       root
  3                              error        ---x------ root       root
  0-2                            up           ---x--x--x root       root
```

Augment **ps aux** for node numbers:

```
[user@cluster user] $ ps aux | bpstat -P
NODE    USER       PID %CPU %MEM   VSZ  RSS TTY      STAT START   TIME COMMAND
        root         1  0.0  0.0  4756  552 ?        S    10:58   0:02 init [5]
        root         2  0.0  0.0     0    0 ?        S    10:58   0:00 [migration/0]
  (etc.)
```

Filter **ps aux** for nodes n1 and n2:

```
[user@cluster user] $ ps aux | bpstat -P n1,n2
NODE    USER       PID %CPU %MEM   VSZ  RSS TTY      STAT START   TIME COMMAND
1       root      1328  0.0  0.0  6864  692 ?        Ss   12:45   0:00 [portmap]
2       root     32397  0.0  0.0  6864  692 ?        Ss   12:45   0:00 [portmap]
```

See Also

beomap(1), bproc_nodestatus(3), bproc_nodeaddr(3), bproc_nodeinfo(3), bproc_pidnode(3)

mpprun

Name

mpprun — Run a series of commands on a Scyld cluster using a dynamically generated job map.

Synopsis

mpprun [-h, --help] [-V, --version] [-p,--prefix]
[--all-cpus] [--all-nodes] [--all-local] [--no-local]
[--map *nodelist*] [--exclude *nodelist*] [--np *processes*]
command [*command-args...*]

Description

The **mpprun** program sequentially runs the specified program on a dynamically selected set of cluster nodes. It generates a job map from the currently installed **beomap** scheduler, and runs the program on each node specified in the map. The scheduling parameters from the command line and environment are the same as for **beomap**, and the resulting job map is identical to the job map that **beomap** would if generate at that instant in time for that program name.

mpprun is similar to the **beorun** program, but **beorun** starts the job simultaneously on the cluster nodes, whereas **mpprun** starts the job sequentially.

Options

The following general command line options are available to **mpprun**. Also see the next section, which describes the **beomap** job map parameters.

-h, --help, -u, --usage

Print the command usage message on `stdout` and exit. When one of these options is recognized in the option list, all following options will be ignored.

-V

Print the command version number on `stdout` and exit. Any following options will be ignored.

-p,--prefix

Prefix each line of output the node number.

Unrecognized options and invalid option formats are reported on `stderr` and the command exits with exit status 1 (invalid option) or 2 (no command specified or invalid command).

Job Map Parameters

You can influence the **beomap** job map either by entering command line options or by setting environment variables. Following are the available command line options, together with their equivalent environment variables. Note that the command line options take precedence over the environment variables.

All of the **beomap** job map parameters listed below can also be used directly with **beorun**, **mpirun**, and **mpprun**.

--all-cpus

Create a process map consisting of all "up" nodes, with each node number repeated to represent the number of CPUs on that node. This parameter is not allowed in conjunction with the **--map** parameter.

The equivalent environment variable is **ALL_CPUS**.

--all-nodes

Create a process map consisting of all "up" nodes, with 1 CPU mapped on each of the "up" nodes. This parameter is not allowed in conjunction with the **--map** parameter.

The equivalent environment variable is **ALL_NODES**.

--all-local

Create a process map consisting entirely of master node entries. This option eliminates everything except node -1 from the pool of candidate node numbers, thus forcing the map to use node -1 for everything. This parameter is not allowed in conjunction with the **--map** parameter.

This option is a handy shortcut for troubleshooting connectivity problems or testing on an isolated head node before running a job on a "live" cluster.

The equivalent environment variable is **ALL_LOCAL**.

--no-local

Exclude the master in the process map. This option is essentially a syntactic shortcut for including -1 in the **--exclude *nodelist*** option. For MPI jobs, this option puts the "rank 0" job on a compute node instead of the master. This parameter is not allowed in conjunction with the **--map** parameter.

The equivalent environment variable is **NO_LOCAL**.

--exclude *nodelist*

Do not include the listed nodes in the process map; the nodes to be excluded are stated as a colon-delimited list. This parameter is not allowed in conjunction with the **--map** parameter.

The equivalent environment variable is **EXCLUDE=*nodelist***.

--map *nodelist*

Specify a process map consisting of a colon-delimited list of nodes. Each node in the list indicates where one process will be assigned. The number of entries in the job map implies the number of ranks in the job.

Listing a node more than once in the list will assign multiple processes to that node. Typically, this is done to assign one process to each processor or core on a node, but this can also be used to assign more processes to a node than it has processors or cores.

The **--all-cpus**, **--all-nodes**, **--np *processes***, **--all-local**, **--no-local**, and **--exclude** parameters are not allowed in conjunction with the **--map** parameter.

The equivalent environment variable is **BEOWULF_JOB_MAP=*nodelist***.

--np *processes*

Specify the number of processes to run. The **beomap** command attempts to place one process per processor or core, but will assign multiple processes per processor or core if there are not enough individual processors or cores available. This parameter is not allowed in conjunction with the **--map** parameter.

The equivalent environment variable is **NP=*processes***.

Tip: The environment variables have an order of priority. The **BEOWULF_JOB_MAP** variable acts as a "master override" for the other environment variables. If **BEOWULF_JOB_MAP** is not set, the following priorities apply:

- Three of the environment variables determine *how many ranks* to schedule in the map: **ALL_CPUS** (priority 1), **ALL_NODES** (priority 2), and **NP** (priority 3). If none of these are set explicitly by the user, then a usage of NP=1 is assumed.
- Three of the environment variables determine *what node numbers* are candidates for being mapped: **ALL_LOCAL** (priority 1), **NO_LOCAL** (priority 2), and **EXCLUDE** (priority 3).

Caution

It is an error to use **NO_LOCAL** and **ALL_LOCAL** together. If both are used, **ALL_LOCAL** will take precedence.

Examples

Run **uptime** on any two available cluster compute nodes.

```
[user@cluster user] $ mpprun --np 2 --no-local uptime
  11:05am  up 2 days, 11:16,  0 users,  load average: 0.05, 0.24, 0.65
  11:05am  up 2 days, 11:16,  0 users,  load average: 0.01, 0.07, 0.37
```

See Also

beomap(1), bpsh(1), mpprun(1), beonpc(1), and the User's Guide.

Scyld ClusterWare Maintenance Commands

This section of the *Reference Guide* describes the Scyld ClusterWare maintenance utilities. These commands can be used by the cluster administrator, and are not intended for use by the ordinary user.

beofdisk

Name

beofdisk — Query and modify hard drive partitions on compute nodes.

Synopsis

/usr/sbin/beofdisk [-h, --help] [-v, --version] [-q, --query]
[-w, --write] [-d, --default] [-M, --mbr]
[-n *num*, --node *num*]

Description

This script allows you to partition the hard drives on compute nodes.

When you query, it will create files in `/etc/beowulf/fdisk/`, one for each device/drive geometry it finds. These files can then be modified by hand, or with the defaults options, then written back to the hard drives.

Options

-h, --help

Display a help message and exit.

-v, --version

Display version information and exit.

-q, --query

Queries the hard drives and writes their current partition tables into files in `/etc/beowulf/fdisk/`. If no -n *num* node is specified, then all nodes are queried.

-w, --write

Matches the files in `/etc/beowulf/fdisk/` with the hard drives and changes the partition tables on the compute nodes to match what is in the files. If no -n *num* node is specified, then all nodes are written.

WARNING: This option is potentially dangerous. It modifies partition tables, and incorrect partition tables can cause problems.

-d, --default

This will cause beofdisk to go through the files in `/etc/beowulf/fdisk/` and set them all to contain default partitioning schemes that include a beoboot partition, a swap partition, and the rest as `/`.

-M, --MBR

Write a simple Master Boot Record to the hard drive that directs the BIOS to "boot next device" after each failure to boot. Typically, this ultimately results in a PXE boot. If no -n *num* node is specified, then all nodes are written with this new MBR.

-n *num*, --node *num*

By default, the apply the specified operation to all nodes. Optionally, apply the operation only to node *num*.

Examples

Creating default partition schemes:

```
[root@cluster ~] # beofdisk -d
 Creating a default partition table for hda:2495:255:63
 Creating a default partition table for hda:1222:255:63
```

Writing the defaults to node 0's hard drive:

```
[root@cluster ~] #  beofdisk -w -n 0

Disk /dev/hda: 2495 cylinders, 255 heads, 63 sectors/track
Old situation:
Units = cylinders of 8225280 bytes, blocks of 1024 bytes, counting from 0

   Device Boot Start     End   #cyls   #blocks   Id  System
/dev/hda1   *      0+      0       1-     8001   89  Unknown
/dev/hda2          1      32      32    257040   82  Linux swap
/dev/hda3         33    2494    2462  19776015   83  Linux
/dev/hda4          0       -       0         0    0  Empty
New situation:
Units = sectors of 512 bytes, counting from 0

   Device Boot    Start       End  #sectors  Id  System
/dev/hda1   *        63     16064     16002  89  Unknown
/dev/hda2         16065    546209    530145  82  Linux swap
/dev/hda3        546210  40082174  39535965  83  Linux
/dev/hda4             0         -         0   0  Empty
Successfully wrote the new partition table

Re-reading the partition table ...

If you created or changed a DOS partition, /dev/foo7, say, then use dd(1)
to zero the first 512 bytes:  dd if=/dev/zero of=/dev/foo7 bs=512 count=1
(See fdisk(8).)
Node partition tables have been modified.
You must reboot each affected node for changes to take effect.
```

Query the disks on the compute nodes to determine how they are partitioned:

```
[root@cluster ~] # beofdisk -q
```

The following creates a partition file in `/etc/beowulf/fdisk`, with a name similar to `sda:512:128:32` and containing lines similar to the following:

```
[root@cluster ~] # cat sda:512:128:32
 /dev/sda1  :  start=    32,  size=  8160,   id=89,  bootable
 /dev/sda2  :  start=    8192,   size=    1048576,  Id=82
 /dev/sda3  :  start=    1056768,    size=    1040384,  Id=83
 /dev/sda4  :  start=    0, size=  0,  Id=0
```

beorsync

Name

/usr/sbin/beorsync — Sync files between two servers in an HA configuration.

Synopsis

beorsync *syncfiles*

Description

beorsync is a perl script used to synchronize individual files and the contents of entire directories between master nodes in a High Availability master node failover environment.

The script has one required argument: *syncfiles*, which is the name of a file containing a list of files and directories to be synchronized.

beorsync expects to execute on the passive master node of a passive-active pair. Both the source and target nodes must be running heartbeat. The script pulls only those files that have changed on the active master node.

Diagnostic messages are logged to /var/log/beorsync.log.

Errors

If beorsync is invoked on the active master node, then the script exits with an error message.

Examples

A typical *syncfiles* contains the following list of files and directories to be synchronized:

```
/etc/hosts
/etc/resolv.conf
/etc/ntp.conf
/root/bin/
/var/spool/cron/
/etc/beowulf/
/etc/passwd
/etc/shadow
/etc/group
/etc/nsswitch.conf
/etc/exports
/etc/services
/etc/ha.d/haresources
/var/spool/torque/mom_priv/config
/var/spool/torque/server_priv/jobs/
/var/spool/torque/server_priv/serverdb
```

Commonly, a cron job should be set up that periodically executes the **beorsync** script. For example, the following cron entry executes the script every 5 minutes, syncing all of the files and directories listed in the *syncfiles* file named /etc/beowulf/beorsyncfiles:

```
*/5 * * * * beorsync /etc/beowulf/beorsyncfiles
```

beoserv

Name

beoserv — The daemon that serves IP addresses and boot files to compute nodes

Synopsis

/usr/sbin/beoserv [-h] [-V, --version] [-v] [-f `file`] [-n `file`]

Description

This daemon responds to DHCP, PXEboot, and TFTP requests from compute node clients. The daemon is started by `/etc/init.d/clusterware` when the clusterware starts.

Options

-h, --help

Display a help message and exit.

-V, --version

Display version information and exit.

-v

Increase verbosity level. Each additional 'v' increases verbosity.

-f `file`

Read configuration from `file` instead of `/etc/beowulf/config`.

-n `file`

Write unrecognized nodes to `file` instead of `/var/beowulf/unknown_addresses`.

Examples

Start daemon using file `myconfig`.

```
[root@cluster ~] # /usr/sbin/beoserv -f /etc/beowulf/myconfig
```

bpctl

Name

bpctl — Control the operational state and ownership of compute nodes.

Synopsis

bpctl [-h, --help] [-v, --version] [-f]
[-M, --master] [-S *num*, --slave *num*]
[-s *state*, --state *state*] [-m *mode*, --mode *mode*]
[-u *user*, --user *user*] [-g *group*, --group *group*]
[-H, --halt] [-P, --poweroff, --pwroff] [-R, --reboot]
[-O, --orphan] [-C *r2c-state*, --completion *r2c-state*]
[-I *idle-threshold*, --idle *idle-threshold*]

Description

This utility is part of the **BProc** package and is installed by default. It allows the root user to modify the state of the compute nodes. Compute nodes may be in one of eight states: **down**, **boot**, **up**, **error**, **unavailable**, **reboot**, **halt**, **poweroff**. The states are described as follows:

down

No communication with compute node, and prior node state is unknown.

boot

Node has initialized communication and started but not completed the node_up script. This state is not commandable. It is status information only.

up

Node is communicating and has completed the node_up script without errors.

error

Node is communicating and encountered an error while running the node_up script.

unavailable

Node is communicating and the cluster administrator has marked the node as unavailable to non-root users.

reboot

Node will do a software reboot. Node status will show reboot through start of machine shutdown until node_up script has begun.

halt

Node has been commanded to halt. This command causes the node CPUs to execute the halt machine instruction. Once halted the node must be reset by external means to resume normal operations.

poweroff, pwroff

Node will power off. This command is valid for nodes that meet the ATX specification. This command requires BIOS support. Non-ATX machines may reboot on this command.

Normally the node will transition from **down** to **boot** to **up**, and will remain **up** until commanded otherwise. **up** is the operational state for user programs. User **BProc** commands will be rejected if the node is not **up**.

BProc supports a simplified user and group compute node access scheme. Before any action is taken on a node, **BProc** checks if the user or group match. If either is matched the user action is processed. Note, normal file permissions are still in affect on each node. **BProc** permissions simply allow users to execute a program on a node. Root bypasses the check and always has access.

User and group changes made with **bpctl** remain in effect until the node or the ClusterWare daemons are restarted. After a restart, the user and group information is read from the `/etc/beowulf/config` file. For persistent changes, you must edit the config file. Changes to the config file take effect when you SIGHUP the daemons via **systemctl reload clusterware** or reboot the nodes via **systemctl restart clusterware**. With **reload**, running jobs will not be affected unless they start a new process and are denied node access based on the file changes.

Whenever the ClusterWare daemons are restarted, all nodes are initialized to the **down** state and node history is lost. When this occurs, previously communicating nodes will reboot and attempt to re-establish communication after the "ping timeout", which by default is 30 seconds.

Options

The following options are available to **bpctl**:

-h

Print the command usage message and exit. If -h is the first option, all other options will be ignored. If -h is not the first option, the other options will be parsed up to the -h option, but no action will be taken.

-v

Print the command version number and exit. If -v is the first option, all other options will be ignored. If -v is not the first option, the other options will be parsed up to the -v option, but no action will be taken.

-f

Fast mode. Whenever possible, do not wait for acknowledgment from compute nodes.

-M

Specifies that the remaining options apply to the master node.

-S *num*

Specifies that the remaining options apply to the specified compute node. The *num* can range from 0 to the total number of nodes minus one.

-s *state*

Set the node to the indicated *state*. Valid *state* values are **down**, **up**, **error**, **unavailable**, **reboot**, **halt**, or **pwroff**. Setting *state* to **down** causes the node to reboot due to a communications timeout after the "ping timeout" interval, which by default is 30 seconds.

-m *mode*

Set the permission bits for the indicated node. Only the Execute mode bits are recognized, i.e., a logical or'ing of octal values 001, 010, and/or 100.

-u *user*

Set the user id for the indicated node. Will reject invalid users. Numbers or strings may be used. A numeric user id will be converted to a name if the name is known.

-g *group*

Set the group id for the indicated node. Will reject invalid groups. Numbers or strings may be used. A numeric group id will be converted to a names if the name is known.

-H, --halt

Halt the indicated node.

-P, --poweroff, --pwroff

Power off the indicated node.

-R, --reboot

Reboot the indicated node.

-O, --orphan

Direct the indicated node to become an immediate orphan.

-C *r2c-state*, --completion *r2c-state*

Turn run-to-completion mode on or off for the nodes specified by **-S** *num*. Acceptable *r2c-state* values are **on** (an "orphaned" node stays up indefinitely, until manually rebooted), **off** (an "orphaned" node reboots immediately), or a positive number of seconds of "effectively idle" time that an orphaned node will wait until rebooting.

-I *idle-threshold*, --idle *idle-threshold*

Override the default cpu usage percentage threshold that an "orphaned" compute node uses to determine whether or not the node is "effectively idle".

When a compute node becomes an "orphan" and the *r2c-state* specifies that the node reboot after the specified number of "effectively idle" seconds, BProc periodically determines how much cpu usage has occurred during the preceding interval (which is nominally 10 seconds). If the cpu usage is above the *idle-threshold* percentage, then the time-until-reboot is reset back to *r2c-state* seconds. The *idle-threshold* value must be a positive numeric value, and it may be an integer or a floating-point number. A too-low value means BProc will mistakenly interpret trivial amounts of cpu usage (e.g., executed by daemons that wake up and check for work) as being significant, and thus the node may never reboot. A too-high value means BProc will mistakenly interpret significant cpu usage as being insignificant, and thus the node may reboot while a low-usage process is doing important work.

Examples

This command will cause all nodes to reboot:

```
[root@cluster ~] # bpctl -S all -s reboot
```

This command returns an error, because boot is not commandable:

```
[root@cluster ~] # bpctl -S 4 -s boot
 Non-commandable node state: boot
```

The following sets nodes 3 and 4 ownership to user "foo", which must be a valid user:

```
[root@cluster ~] # bpctl -S 3-4 -u foo
```

The following sets permission on the master node to allow only user root to execute on the master node, e.g., to disallow a non-root user to execute on a compute node and **bpsh** a command to execute on the master:

```
[root@cluster ~] # bpctl -M -m 100
```

And this resets permission on the master node to allow any user to execute on the master node:

```
[root@cluster ~] # bpctl -M -m 111
```

This command resets the run-to-completion timeout to five minutes, and sets the "effectively idle" cpu usage percent to 1.5%:

```
[root@cluster ~] # bpctl -C 300 -I 1.5
```

Return Values

Upon successful completion, **bpctl** returns 0. On failure, an error message is printed to `stderr` and **bpctl** returns 1.

bplib

Name

bplib — bplib manages the VMAdump in-kernel library list and individual file list of cached files .

Synopsis

bplib [-h, --help] [-v, --version] [-c] [-l] [-a [*libs...*]] [-d [*libs...*]]

Description

This utility is part of the **BProc** package and is installed by default. It is used to modify entries of the in-kernel cache list

Options

The following options are available to the bplib program.

-h

Print the command usage message and exits success.

-v

Print the command version number and exits success.

-c

Clears ALL cached entries in the in-kernel cache list

-a libs

Adds the specified file or directory to the in-kernel cache list

-d lib

Deletes the specified file or directory to the in-kernel cache list

-l

Lists all entries known by the in-kernel cache list

bpmaster

Name

bpmaster — Daemon for cluster control and communication.

Synopsis

bpmaster [options] [-h] [-V] [-d] [-v] [-i] [-c file] [-m file]

Description

This daemon is part of the **BProc** package and is installed by default. It is the controller and message/IO manager for all the compute nodes and must be running for the cluster to function.

It is started in the /etc/init.d/clusterware script, along with other **BProc** daemons, and forks a copy of itself for IO forwarding. With normal cluster operation there should be 2 PIDs for **bpmaster**. Type **ps -x |grep bpmaster** to check.

The daemons may be restarted at any time using **systemctl restart clusterware**, but note that this will cause all nodes to reboot. During normal operations, you should use **systemctl reload clusterware** to enable cluster configuration changes. **Bpmaster** uses `/var/log/messages` to report cluster events such as configuration changes and errors.

Options

The following options are available to the bpmaster program.

-h

Print the command usage message and exit. If `-h` is the first option, all other options will be ignored. If `-h` is not the first option, the other options will be parsed up to the `-h` option, but no action will be taken.

-V

Print the command version number and exit. If `-v` is the first option, all other options will be ignored. If `-v` is not the first option, the other options will be parsed up to the `-v` option, but no action will be taken.

-d

Start the program in debug (verbose) mode. **bpmaster** will not daemonize, and all information and error messages will go to `stdout`. This information is useful when the daemon exits abnormally during operation as the information is not mixed in with the normal `/var/log/messages`.

-v

Increase verbosity level. This may be specified multiple times.

-i

Ignore interface version mismatch. This can be dangerous.

-c file

Specifies a different configuration file is to be used. The default is set to `/etc/beowulf/config`. This option is for debug and development. This option is not recommended for normal use.

-m file

Log master and node **BProc** messages to the indicated file. This information is intended for **BProc** debugging, and should not be enabled unless requested by a Scyld support engineer. This file grows in size rapidly depending of the number of nodes, approximately 2 megabytes/minute with six nodes.

Examples

Don't start as daemon.

```
[root@cluster ~] # bpmaster -d
```

Start the daemon using the startup script.

```
[root@cluster ~] # systemctl start clusterware
 Configuring network interface (eth1):                    [  OK  ]
 Loading modules:                                         [  OK  ]
 Setting up libraries:                                    [  OK  ]
 Starting bpmaster:
 Starting beoserv:
 Starting recvstats:
 Starting sendstats:
```

bpslave

Name

bpslave — This program is the BProc distributed process space slave daemon. It runs on each compute node.

Synopsis

bpslave [options] [-h] [-V] [-l *logfacility*] [-r] [-i]
[-d] [-s *addr*] [-c *dir*] [-p *port*] [-m *file*] [-v]

Description

The **bpslave** daemon is part of the **BProc** package, and is installed by default. It is the controller and message and I/O manager run on each compute node, and must be running for the node to function.

bpslave is started by the Scyld compute node init process, which sets parameters based on what is passed in through the kernel command line option in the `/etc/beowulf/config` file. All parameters of the **bpslave** daemon are not accessible via the kernel command line keyword in `/etc/beowulf/config`.

The **bpslave** daemon is not intended to be run from the command line nor started, except implicitly by the compute node init process.

Options

The following options are available to the **bpslave** program. These options are mainly intended for using **BProc** in a standard linux environment where the master and compute nodes both have full system installs.

-h

Show this message and exit. If *-h* is the first option, all other options will be ignored. If *-h* is not the first option, the other options will be parsed up to the *-h* option, but no action will be taken.

-V

Print version information and exit.

-l *logfacility*

Log to this log facility (default=daemon).

-r

Automatic reconnect on error or lost connection.

-i

Ignore **BProc** version mismatches (dangerous).

-d

Do not daemonize self.

-s *addr*

Connect from source address *addr*.

-c *dir*

Set library cache to *dir*. \n.

-p *port*

Set library cache file request port to *port*. The default is port 932, which can be overridden by a directive **server beofs2** ***port*** in `/etc/beowulf/config`.

Debugging options:

-m file

Enable message trace to file.

-v

Increase verbose level (implies `-d`).

Masterhostname [[port]]

The host name and (optionally) the port number of the **bpmaster** daemon. The default is port 933, which can be overridden by a directive **server bproc** ***port*** in `/etc/beowulf/config`.

node_down

Name

node_down — Bring a compute node down cleanly.

Synopsis

/usr/lib/beoboot/bin/node_down node [state]

Description

This script can be used to bring a node down ("reboot", "halt", "pwroff") in such a way that the local filesystems on the compute node remain in a constant state.

node_down works by first changing the node's state to "unavailable", then remounting all of the filesystems read-only, followed by using **bpctl** to perform the actual state change you requested.

Options

node

The node number of the node to bring down.

state

The state to put the node in after remounting all the filesystems. The state defaults to "reboot" if unspecified.

Examples

Cleanly rebooting node 3:

```
[root@cluster ~] #  /usr/lib/beoboot/bin/node_down 3
 Remounting / readonly...
 Remounting /proc readonly...
 Remounting /home readonly...
 Remounting /dev/pts readonly...
 Syncing disks on node 3.
 Shutting down node 3 (reboot)
```

recvstats

Name

recvstats — receive status from cluster nodes

Synopsis

recvstats [-p *port*] [-N *initial-max-num-nodes*] [-f]

Description

The **recvstats** daemon is part of the **beostat** package. It executes on the master node, receives periodic per-node status data sent by each cluster node's **sendstats** daemon, and makes the data available to various commands and services on the master node.

The **recvstats** daemon parses the received data to ensure basic validity. The exact content and format of the **sendstats** messages is version specific, though it typically includes a unique node number identifying the sender, plus the dynamic values of the following **proc** file system files: `/proc/cpuinfo`, `/proc/meminfo`, `/proc/loadavg`, `/proc/net/dev`, and `/proc/stat`.

The **recvstats** daemon stores the incoming data in shared memory in the `/dev/shm` filesystem, which should be readable by everyone. If that filesystem doesn't exist or the file permissions are not set correctly, then **recvstats** and the consumers of that data will not function correctly. Note: most consumers of this **recvstats** data access it using various commands (e.g., **beostat(1)**, **beostatus(1)**, **ganglia**, **Scyld IMF**) or use the **libbeostat** abstracted library interface.

The **recvstats** daemon is started by the `/etc/init.d/clusterware` script when the clusterware service starts, and **sendstats** is started by the node initialization script `/etc/beowulf/init.d/95sendstats`.

Options

The following options are available to the **recvstats** daemon.

-p *port*

Override the default listen port of 5545. Only use this option if there is a conflict with the default port, and use the same non-default *port* when executing **sendstats** on each cluster node.

-N *initial-max-num-nodes*

Start **recvstats** with an explicit non-default guess about how many cluster nodes will be sending data. If a node number above this initial maximum sends data, then **recvstats** dynamically expands the shared memory structure to accommodate it.

Examples

Start the **recvstats** daemon on the master node:

```
[root@cluster ~] # /usr/bin/recvstats -p 5545 -N `beoconfig nodes`
```

See Also

sendstats(8), beostat(1), beostatus(1)

sendstats

Name

sendstats — transmit cluster node status

Synopsis

sendstats [-h] [*nodenumber*] [*IPaddress*[:*port*] ...]

Description

The **sendstats** daemon is part of the **beostat** package. Typically, the daemon executes on each node in the cluster and periodically transmits status data to a **recvstats** daemon that executes on the master node. In a cluster with multiple master nodes, **sendstats** typically sends status data to every master node.

The optional *nodenumber* is unnecessary for normal uses of **sendstats**. The **recvstats** daemon is normally able to discern the sender's node number from the sender's IP address. If *nodenumber* is specified, then it must be seen by the receiving **recvstats** as being unique to one and only one sending node in the cluster.

The exact content and format of the **sendstats** messages is version specific, though it typically includes a unique identifying *nodenumber* plus the dynamic values of the following **proc** file system files: `/proc/cpuinfo`, `/proc/meminfo`, `/proc/loadavg`, `/proc/net/dev`, and `/proc/stat`.

The *port* number is optional, defaulting to port 5545. In the event of a collision with another preexisting service, which would typically be defined in `/etc/services`, you must override the default. Choose a new value that is not currently employed on the system, then add a **server beostats *port*** directive to `/etc/beowulf/config`.

The **recvstats** daemon is started by the `/etc/init.d/clusterware` script when ClusterWare services start, and **sendstats** for BProc nodes is started by the node initialization script `/etc/beowulf/init.d/13sendstats`.

The **sendstats** daemon may be used on machines outside of the **BProc** cluster management domain. In any case, the port number must match the port on which **recvstats** listens.

Examples

Start the daemon on ClusterWare node n0, sending stats to the master at 10.20.30.1 using the default port:

```
[root@cluster ~] # bpsh 0 /usr/sbin/sendstats 10.20.30.1
```

Start the daemon on ClusterWare node n0, sending stats to the master at 10.20.30.1 and a second master at 10.20.30.2, using a non-default port:

```
[root@c ~] # bpsh 0 /usr/sbin/sendstats 10.20.30.1:939 10.20.30.2:939
```

Start the daemon on a non-ClusterWare node n1, using the default port:

```
[root@c ~] # ssh n1 /usr/sbin/sendstats 10.20.30.1:5545
```

See Also

recvstats(8), beowulf-config(5).

Scyld ClusterWare Special Files

This section of the *Reference Guide* describes `/etc/beowulf/config` and `/etc/beowulf/fstab`, the configuration files that are used by the Scyld ClusterWare system.

beowulf-config

Name

`/etc/beowulf/config` — Configuration file for BProc and beoboot

Description

The Beowulf config file `/etc/beowulf/config` defines the structure of a Scyld cluster and provides a central location for many of the operational parameters. The file contains the settings for **beoboot**, node initialization, **BProc** communication parameters, and other aspects of cluster operation.

The syntax of the Scyld configuration files is standardized and is intended for human editing with embedded comments. Tools are provided for reading and writing from common programming and scripting languages, with writing retaining comments and formatting.

> **Tip:** Care must be taken when editing or otherwise modifying `/etc/beowulf/config`, e.g., avoid editing while new compute nodes are coming online and ClusterWare itself is adding or modifying 'node' lines. Also note that incorrect editing may leave the cluster unuseable.

Scyld Config File Format

The config file is a line-oriented sequence of configuration entries. Each configuration entry starts with a keyword followed by parameters. A line is terminated by a newline or '#'. The latter character starts a comment.

The keyword and following parameters have the same syntax rules: they may be preceded by whitespace and continue to the next whitespace or the end of the line.

Keywords and following parameters may include whitespace by quoting between a matching pair of '"' (double quote) or "' (single quote) characters. A '\' (backslash) removes the special meaning of the following quote character.

Note that comments and newlines take precedence over any other processing, thus a '#' may not be used in a keyword or embedded in a parameter, and a backslash followed by a newline does not join lines.

Each configuration option is contained on a single line, with a keyword and optional parameters. Blank lines are ignored. Comments begin with an unquoted '#' and continue to the end of the line.

Keywords

bootmodule *`modulename`*

> The **bootmodule** keyword specifies that the kernel binary module *`modulename`* be included in the compute nodes' initrd image. These are typically network drivers needed to fully initialize a booting node. At node startup, the **beoclient** daemon on a compute node scans the node's `/proc/bus/pci/devices` list and automatically executes a **modprobe** for every *`modulename`* driver named by a PCI device so discovered. However, note that if the PCI scan does not find a need for a particular driver, then no automatic **modprobe** occurs. Add an additional **modprobe** keyword to forcibly load the *`modulename`*.

firmware *firmfile*

The **firmware** keyword specifies that the *firmfile* file, which typically resides on the master node in /lib/firmware/*firmfile*, be included in the compute nodes' initrd image, if known to be needed by a particular **bootmodule** *modulename*. Adding one or more **firmware** keywords significantly increases the size of the initrd image. See the Administrator's Guide for details.

fsck *fsck-policy*

The **fsck** keyword specifies the file system checking policy to be used at node boot time. The valid policies are "never", "safe" or "full".

never

The file system on the compute nodes will not be checked on boot.

safe

The file system on the compute nodes will go through a safe check every time the compute node boots.

full

The file system on the compute nodes will go through a full check every time the compute node boots. The full check might possibly remove files from the filesystem if they cannot be repaired.

host *MACaddress IPaddress* [*hostname(s)*]

The **host** keyword assigns an IP addresses to a specific client device identified by its MAC address, if and when that client makes a DHCP request to the master. The IP addresses must be in dotted notation (e.g., 192.168.1.100), and it must be within the range of one of the **hostrange** IP address ranges. These **host** clients are not Scyld nodes, which are identified by **node** keywords and are assigned IP addresses from the **iprange** range. Rather, typically they are devices like smart Ethernet switches that connect to the cluster private network and issue a DHCP request to obtain an IP address. Up to six optional *hostname* names may be assigned to a client, and these names are recognized by the Beo NSS service.

hostrange [*name*] *IPaddress-lwb IPaddress-upb*

The **hostrange** is used in conjunction with the **host** keyword. It declares a range of IP addresses that may later be used for **host** clients doing DHCP requests. An optional *name*] may be associated with this range. Multiple **hostrange** keywords may be present.

ignore *MACaddress*

The **ignore** keyword specifies a MAC address (e.g., 00:11:22:AA:BB:CC) that **beoserv** should ignore DHCP and PXE requests from. Multiple **ignore** keywords are allowed.

initrdimage [*noderange*] *imagename*

The **initrdimage** keyword specifies the full path to the initrd image that should be used when creating the final boot images for the compute nodes. If *noderange* is specified, then this *imagename* applies only to the specified range of nodes; otherwise, *imagename* applies to all nodes.

insmod *module-name* [*options*]

The **insmod** keyword specifies a kernel module to be loaded (usually a network driver). Options for the module may be specified as well.

interface *interfacename*

The **interface** keyword specifies the name of the interface that connects the master node to the compute nodes. This is used by the cluster services and management tools such as the **bpmaster** daemon and the **beoserv** daemon. Common values are "eth0" or "eth1". If present, entries after the interface name specify the IP address and netmask that the interface should be configured to.

iprange [*nodenumber*] *IPaddress1 IPaddress2*

The **iprange** keyword specifies the range of IP addresses to be assigned to nodes. If the optional *nodenumber* is given, the first address in the range will be assigned to that node, the second address to the next node, etc. If no node number is given, the address assignment will begin with the node following the node that was last assigned. If no nodes have been assigned, the assignment will begin with node 0.

kernelcommandline [*noderange*] *options*

The **kernelcommandline** keyword specifies any options you wish to have passed to the kernel on the compute nodes. These are the same options that are normally passed with "append=" in **lilo**, or on the **lilo** prompt while the machine is booting (e.g., "kernelcommandline apm=power-off"). If *noderange* is specified, then these *options* apply only to the specified range of nodes; otherwise, *options* apply to all nodes.

kernelimage [*noderange*] *imagename*

The **kernelimage** keyword specifies the full path to the kernel that should be used when creating the final boot images for the compute nodes. If *noderange* is specified, then this *imagename* applies only to the specified range of nodes; otherwise, *imagename* applies to all nodes.

libraries *librarypath1* [, *librarypath2*, ...]

The **libraries** keyword specifies a list of libraries that should be cached on the compute nodes when an application on the node references the library. The library path can be a directory or file. If a file name is specified, then that specific file may be cached, if needed. If a directory name is specified, then every file in that directory may be cached. If the directory name ends with "/", then subdirectories under the specified directory may be cached.

logfacility *facility*

The **logfacility** keyword specifies the log facility that the **BProc** master daemon should use. Some example log facility names are "daemon", "syslog", and "local0" (see the **syslog** documentation for more information). The default log facility is "daemon".

masterdelay *SECS*

The **masterdelay** keyword specifies the timeout value in seconds for a non-primary master node to delay sending a response to an incoming dhcp request. The default value is 15 seconds.

masterorder *nodes IPaddress_primary IPaddress_secondary*

The **masterorder** keyword specifies the cluster IP addresses of the primary master node and the secondary master node(s) for a given set of *nodes*. This is used by the **beoserv** daemon for Master-Failover (cold reparenting). A compute node's PXE request broadcasts across the cluster network. The primary master node is given **masterpxedelay** seconds to respond, after which the first secondary master node will respond. If multiple secondary master nodes are specified, then each waits in turn for **masterpxedelay** seconds for a preferred master to respond. Similarly, the compute node's subsequent DHCP broadcast gets serviced in the same order, with each secondary master waiting **masterdelay** seconds for a preferred master to respond.

Example:

```
masterorder 0,5,10-20 10.1.0.1 10.2.0.1
masterorder 1-4,21-30 10.2.0.1 10.1.0.1
```

If master 10.1.0.1 is down or fails to respond to PXE/DHCP requests to compute node 10, then master 10.2.0.1 becomes the primary parent for compute node 10.

masterpxedelay *SECS*

The **masterpxedelay** keyword specifies the timeout value in seconds for a non-primary master node to delay sending a response to an incoming PXE request. The default value is 5 seconds.

mcastbcast *interface*

The **mcastbcast** keyword directs the **beoserv** daemon to use broadcast instead of multicast when transmitting files over the *interface*. This is useful when network equipment has trouble with heavy multicast traffic.

mcastthrottle *interface rate*

The **mcastthrottle** keyword controls the rate at which data is transmitted over the specified interface. The rate is given in megabits per second. This is useful when the compute node interfaces cannot keep up with the master interface when sending large files.

mkfs *mkfs-policy*

The **mkfs** keyword specifies the policy to use when building a Linux file system on the compute nodes. The valid policies are "never", "if_needed", or "always".

never

The filesystem on the compute nodes will never be recreated on boot.

if_needed

The filesystem on the compute nodes will only be recreated if the filesystem check fails.

always

The filesystem on the compute nodes will be recreated on every boot. **fsck** will be assumed to be set to "never" when this is set.

modarg *options*

The **modarg** keyword specifies options to be used for modules that are loaded during the boot process without options. This is useful for specifying options to modules that get loaded during the PCI scan.

moddep *module-list*

The **moddep** keyword is used to specify module dependencies. The first module listed is dependent on the remaining modules in the space separated list. The first module will be loaded after all other listed modules. Module dependency information is normally automatically generated by the **beoboot** script.

modprobe *modulename* [*options*]

The **modprobe** keyword specifies the name of the kernel module to be loaded with dependency checking, along with any specified module options. Note that the *modulename* must also be named by a **bootmodule** keyword.

node [*nodenumber*] *MACaddress*

The **node** keyword is used to assign MAC addresses to node numbers. There should be one of these lines for each node in your cluster. Note the following:

- If a value is not provided for the *nodenumber* argument, the first node entry is node 0, the second is node 1, the third is node 2, etc.
- The value "off" can be used for the *MACaddress* argument to leave a place holder for that node number.
- To skip a node number, use the value "node" or "node off" for the *MACaddress* argument.
- To skip a node number and make sure it will never be automatically filled in by something later in the future, use the value "node reserved" for the *MACaddress* argument.

nodeacceses [-M | -S *nodenumber* | all] *arglist*

The **nodeaccess** keyword overrides the default access permissions for the master node (**-M**), for all compute nodes (**all**), or for a specific compute node (*nodenumber*). The remaining *arglist* is passed directly to the **bpctl** command for parsing and execution. See the *Administrator's Guide* for details about node access permissions.

Example:

```
nodeaccess -M -m 0110
nodeaccess -S 5 -g physics
nodeaccess -S 6 -g physics
```

nodeassign *nodeassign-method*

The **nodeassign** keyword specifies the node assignment strategy used when the **beoserv** daemon receives a new, unknown MAC address from a computer that is not currently entered in the node database. The total number of entries in the node database is limited to the number specified with the **nodes** keyword (see above).

The valid node assignment methods are "append", "insert", "manual", or "locked". Note the following:

- "Append" and "insert" are the only two choices that allow new nodes to be automatically given node numbers and welcomed into the cluster.
- Any failures of automatic node assignment through "append" or "insert" (such as when the node table is full) will cause the node assignment to be treated as "manual".

Following are the definitions of the valid node assignment methods:

append

This is the default setting. The system will append new MAC addresses to the end of the node list in the `/etc/beowulf/config` file. This is done by seeking out the highest already-assigned node number and attempting to go one number beyond it. If the highest node number in the cluster has already been assigned, the "append" method will fail and the "manual" method will take precedence.

insert

The system will insert new MAC addresses into the node list in the `/etc/beowulf/config` file, starting with the lowest vacant node number. If no spaces are available, the "append" method will be used instead. Typically, a user would choose "insert" when replacing a single node if they want the new node entry to appear in the same place as the old node entry. If the node table is full, the "insert" method will fail and the "manual" method will take precedence.

manual

The system will enter new MAC addresses in the `/var/beowulf/unknown_addresses` file, and require the user to manually assign the new nodes. The node entries will appear in the "Unknown" list in the BeoSetup GUI, which simplifies the node assignment process. An alternative to using the BeoSetup GUI is to manually edit the `/etc/beowulf/config` file and copy in the new MAC addresses from the `/var/beowulf/unknown_addresses` file.

locked

The system will ignore DHCP requests from any MAC addresses not already listed in the `/etc/beowulf/config` file. This prevents nodes from getting added to the cluster accidentally. This is particularly useful in a cluster with multiple masters, because it enables the Cluster Administrator to control which master responds to a new node request. When you are troubleshooting issues related to the cluster not "seeing" new nodes, one of the first things to check is whether **nodeassign** is set to "locked".

See the *Administrator's Guide* for additional information on configuring nodes with the BeoSetup GUI and on manual node configuration.

nodename *name-format* [*IPv4 Offset or Base*] [*netgroup*]

The **nodename** keyword defines the primary hostname, as well as additional hostname-aliases for compute nodes. It can also be used to define hostnames and hostname-aliases for non-compute node entities with a per compute node relationship (e.g., to define a hostname and IP address for the IPMI management interface on each compute node). The presence of the (optional) IPv4 parameter determines if the entry is for compute nodes or for non-compute node entities. If no 'nodename' keyword is defined for compute nodes, then compute nodes' primary hostname is of the 'dot-number' format (e.g., node 10's primary hostname is '.10').

name-format

Define a hostname or hostname-alias. The first instance of the nodename keyword with no IPv4 parameter defines the primary hostname format for compute nodes. While the user may define the primary hostname, the FIRST hostname alias shall always be of the 'dot-number' format. This allows compute nodes to always resolve their address from the 'dot-number' notation. Additional nodename entries without an IPv4 parameter define additional hostname aliases.

The name-format string must contain a conversion specification for node number substitution. The conversion specification is introduced by a percent sign (the '%' symbol). An optional following digit in the range 1..5 specifies a zero-padded minimum field width. The specification is completed with an 'N'. An unspecified or zero field width allows numeric interpretation to match compute node host names. For example, "n%N" will match "n23", "n+23", and "n0000023". By contrast, "n%3N" will only match "n001" or "n023", but not "n1" or "n23".

IPv4 Offset or Base

The presence of the optional IPv4 argument defines if the entry is for "compute nodes" (i.e. the entry will resolve to the 'dot-number' name) or if the entry is for non-cluster entities that are loosely associated with the compute node. If the argument has a leading zero, then the parameter specifies an IPv4 Offset. If the argument does not lead with a zero, then the argument specifies a 'base' from which IP addresses are computed, by adding the 'node-number' associated with the non-compute node entity.

Netgroup

The netgroup parameter specifies a netgroup that contains all the entries generated by the nodename entry

nodes *numnodes*

The **nodes** keyword specifies the total possible number of nodes in the cluster. This should normally be set to match the *iprange*. However, if multiple *ipranges* are specified, then this value should represent the total number of nodes in all the *iprange* entries.

pingtimeout *SECS*

The **bpmaster** daemon that executes on the master node sends periodic "ping" messages to the **bpslave** daemon that executes on each compute node, and each bpslave dutifully responds. This interaction serves as mutual bpmaster<->bpslave assurance that the other daemon and the network link is still alive and well. If bpslave does not see this "ping" message for *SECS* seconds, then the bpslave goes into "orphan mode". If run-to-completion is enabled (see the *Administrator's Guide* for details), then the node attempts to remain alive and functioning, despite its apparent inability to communicate with the master node. If run-to-completion is not enabled (which is the default), then the node reboots immediately. If bpmaster does not see a ping reply for *SECS* seconds, then it syslogs this event and breaks its side of the network connection to the compute node.

The default **pingtimeout** value is 32 seconds. In rare cases, a particular workload may trigger such a "ping timeout" and its associated spontaneous reboot, and using a **pingtimeout** keyword to increase the timeout value may stop the spontaneous rebooting.

pci *vendorid deviceid drivername*

The **pci** keyword specifies what driver should be used in support of the specified PCI device. A device is identified by a unique vendor ID and device ID pair. The vendor and device ID's can be either in decimal or hexadecimal with the "0x" notation. You should have one of these lines for each PCI ID (a vendor ID combined with a device ID) for each device on your compute nodes that is not already recognized. Any module dependencies or arguments should be specified with *moddep* and *modarg*.

prestage *pathname*

The **prestage** keyword names a specific file that each compute node pulls from the master at node boot time. Multiple instances of **prestage** can be used. If the *pathname* is a file in one of the **libraries** directories, then the *pathname* gets pulled into the compute node's library cache. Otherwise, the file (and its directory hierarchy) is copied from the master to the compute nodes.

server *transport-protocol port*

The **server** keyword specifies the port numbers that ClusterWare uses for specified transport protocols. Each transport protocol uses a unique default port number. In the event that a default *port* value conflicts with a port number used by another service (typically, specified in `/etc/services`), a **server** keyword must specify an override value. The allowable *transport-protocol* keywords are "beofs2" (default port 932), "bproc" (default port 933), "beonss" (default port 3045), and "beostats" (default port 5545). (The keyword "tcp" is deprecated - use "beofs2" instead.)

Examples

```
iprange 192.168.1.0 192.168.1.50
nodename ipmi-n%N 0.0.1.0
```

In the above example, the hostname "ipmi-n0" has an address of 192.168.2.50. That is, the compute node's address (192.168.1.50 for compute node 0) plus the IPv4 Offset of 0.0.1.0. The hostname "ipmi-n12" has an address of 192.168.2.12, which is compute node 12's address plus the IPv4 Offset of 0.0.1.0.

```
nodename ib0-n%N 0.1.0.0 infiniband
```

In the above example, define a hostname for the infiniband interface for each compute node. Using the **iprange** values in the previous example, the infiniband interface for compute node 0 has a primary hostname of "ib-n0" and resolves to the address 192.169.1.0: node 0's basic **iprange** IP address, plus the increment 0.1.0.0. The infiniband interface for compute node 10 has a primary hostname of "ib-n10" and resolves to the address 192.169.1.10. Each of the "ib0-n%N" hostnames belong to the "infiniband" netgroup.

```
nodename computenode%N
nodename cnode%3N
```

In the above example, the primary hostname for compute node 0 is "computenode0", and the primary hostname for compute node 12 is "computenode12". The second nodename entry defines additional hostname aliases. The FIRST hostname alias will always be the 'dot-number' notation, so compute node 12's first hostname alias is ".12", and the second hostname alias will be "cnode012". The '%' followed by a three specifies a three-digit field width format for the entry.

The following is an example of a complete Beowulf Configuration File

```
# Beowulf Configuration file

# Network interface used for Beowulf
# Only first argument to interface is important
interface eth1 192.168.1.1 255.255.255.1

# These two should probably agree for most users
iprange 192.168.1.100 192.168.1.107
nodes 8

# Default location of boot images
bootfile /var/beowulf/boot.img
kernelimage /boot/vmlinuz-2.4.17-0.18.12.beo
kernelcommandline apm=power-off

# Default libraries
libraries /lib /usr/lib

# Default file system policies.
fsck full
mkfs if_needed

# beoserv settings
server beofs2 932

# Default Modules
bootmodule 3c59x 8139too dmfe eepro100 epic100 hp100 natsemi
bootmodule ne2k-pci pcnet32 sis900 starfire sundance tlan
bootmodule tulip via-rhine winbond-840 yellowfin

# Non-kernel integrated drivers
bootmodule e100 bcm5700 # gm

# Node assignment method
nodeassign append

# PCI Gigabit Ethernet.
```

```
#  * AceNIC and SysKonnect firmwares are very large.
#  * Some of these are distributed separate from the kernel
bootmodule dl2k hamachi e1000 ns83820 # acenic sk98lin

node 00:50:8B:D3:25:4D
node 00:50:8B:D3:07:8B
ignore 00:50:8B:D3:31:FB
node 00:50:8B:D3:62:A0
node 00:50:8B:D3:00:66
node 00:50:8B:D3:30:42
node 00:50:8B:D3:98:EA
```

See Also

beoconfig(1), beowulf-fstab(5).

Administrator's Guide

beowulf-fstab

Name

/etc/beowulf/fstab — cluster node filesystem control table

Description

The /etc/beowulf/fstab file on the master node contains a list of filesystems to be mounted on compute nodes at boot time. Its purpose, format, and contents are similar to the traditional /etc/fstab, plus a few additional cluster-specific features.

The Scyld fstab system is designed to keep all configuration information on a master node. The /etc/beowulf/fstab file is the default for all compute nodes. Any optional node-specific /etc/beowulf/fstab.*N* file overrides this default file for node number *N*.

The root filesystem on each compute node is a tmpfs filesystem that is automatically sized for the available RAM. In earlier versions of Scyld ClusterWare, this root filesystem was explicitly declared in fstab, but this is no longer done.

The compute node's root filesystem is used to dynamically cache binaries and libraries from the master, to provide space for /tmp and /var/tmp, to provide mountpoints for NFS mounts, etc. Although Scyld ClusterWare does not require a harddrive on a compute node, some clusters employ harddrive(s) for node-local persistent storage, for "scratch" storage to avoid having /tmp and /var/tmp consume tmpfs RAM, or for swap space to expand the available virtual memory space and thus reduce Out-of-Memory conditions.

The Scyld fstab interacts with the *mkfs* and *fsck* directives in the /etc/beowulf/config file (see **man beowulf-config**) to control automatic creation or boot-time checking (and potentially repairing) of compute node filesystems on node-local harddrives.

A directive *mkfs always* specifies to rebuild at boot time every harddrive partition specified in /etc/beowulf/fstab, and thus should be used with *great* care so as to not automatically rebuild a partition and thus destroy data that is expected to survive across compute node reboots. Normally the default directive *mkfs never* is used.

A directive *fsck full* specifies to check and potentially repair at boot time every harddrive partition. Alternatively, *fsck safe* specifies to perform an fsck, but to not attempt any repairs; after boot, the cluster administrator may manually perform repairs as needed. A directive *fsck never* is the default, which specifies that no checking be done at boot time.

Syntax

The syntax and layout is identical to the master node's /etc/fstab file. The file contents are processed line by line. All blank lines and lines that begin with a "#" are ignored. All other lines should have six fields, separated by tabs or spaces.

The first field is the device to mount. For filesystems on local harddrives, this should point to a /dev entry, such as /dev/hda2. If mounting an NFS filesystem, the device should be specified as *hostname:directory*, where *hostname* is the IP address of the NFS server, and *directory* is the path on the NFS server you want to mount. If the NFS server is the master node, you can use "$MASTER" as the hostname. Currently, *hostname* cannot be an actual alphanumeric host name because /etc/beowulf/fstab is evaluated at boot time before the compute node's name service is initialized. For some special filesystems, such as **proc** and **devpts**, the *hostname* can be set to "none".

The second field is the mount point. For a swap partition, this should be "swap", but for all other filesystems, this must be a path that begins with "/". Any paths that you specify as mount points will be automatically created by the **node_up** script before it tries to mount the filesystem. Ensure that you do not specify the same mount point on more than one line, because this can cause problems. You can have multiple lines that use "swap" as the mount point, but that is the only exception to the rule.

The third field is the filesystem type. This should be "swap" for swap partitions, or a standard Linux filesystem type (e.g., "ext2", "ext3", "xfs"), or "nfs" for an NFS file system, or particular pseudo filesystem types (e.g., "proc" for the **proc** filesystem, "devpts" for the **devpts** filesystem). Any filesystem that can normally be used by Linux can also be specified here, but you must also take steps to create the harddrive filesystems on the compute nodes before attempting to mount them.

The fourth field lists the mount options for the filesystem. All options should be comma-separated with no spaces. If you do not know of any specific options to use, then you should use the "defaults" keyword.

In addition to the mount options normally supported by Linux, one additional option is supported by Scyld: "nonfatal". Normally, any mount failure results in an immediate abort of the node boot, and the node state transitions from "boot" to "error". Adding "nonfatal" to the options overrides this behavior and allows the node boot to continue, potentially to a node "up" state. However, because filesystem mounts have in fact failed, the node may not actually have full functionality. When using the "nonfatal" option, the cluster administrator is encouraged to view the Scyld boot log files found in directory `/var/log/beowulf/` to discover potential mount failures and other warnings or soft error conditions. The "nonfatal" option is useful for harddrive filesystems when not all compute nodes share the same number and partitioning of drives, or when NFS mounts might fail because an NFS server is temporarily unavailable or the specified filesystem is not currently exported.

The fifth and sixth fields are left there for compatibility with the standard `fstab` format. These fields are not used at the moment, but are required to be there. We recommend they both be set to "0".

Examples

```
# This file is the fstab for nodes.
# One difference is that we allow for shell variable expansions...
#
# Variables that will get substituted:
#  MASTER = IP address of the master node.  (good for doing NFS mounts)

# This is the default setup from beofdisk, once you setup your disks.
#/dev/hda2 swap  swap defaults,nonfatal 0 0
#/dev/hda3 /  ext2 defaults,nonfatal 0 0

# These should always be added
none  /proc  proc defaults 0 0
none  /dev/pts devpts gid=5,mode=620 0 0

# NFS (for example and default friendliness)
$MASTER:/home /home  nfs nolock,nonfatal  0 0
```

Files

`/etc/beowulf/fstab`, `/etc/beowulf/fstab.<`*node-num*`>`, `/etc/beowulf/config`

See Also

beowulf-config(5)

Scyld ClusterWare Beostat Libraries

This part of the *Reference Guide* describes the functions included in the Scyld ClusterWare C libraries for **Beostat**, the Beowulf Status library. The functions in this library can be used for retrieving performance information about the nodes on the cluster, such as CPU and memory utilization.

beostat_count_idle_cpus

Name

`beostat_count_idle_cpus` — count number of idle CPUS in cluster

Synopsis

```
#include <sys/beostat.h>
int beostat_count_idle_cpus (float threshold);
```

Arguments

threshold

The value of CPU usage below which the CPU will be considered idle.

Description

`beostat_count_idle_cpus` executes on the master node and counts the number of CPUs in the entire cluster that are available to the current user/group and have CPU usage below a given threshold. Note that an easy way to count the total number of CPUs available to a user independent of the usage is to use an arbitrarily large threshold value.

Examples

```
int max, fif;
max = beostat_count_idle_cpus (9999.0);
fif = beostat_count_idle_cpus (0.5);
printf ("%d of the %d CPUs available are busy.\n", (max-fif), max);
```

Return Value

Returns the number of CPUs that are both available for the caller and have usage below the threshold. If an error occurs, it will return -1.

Errors

This function relies on the **Beostat** subsystem, which consists of the **proc** filesystem on the remote node, the **sendstats** daemon on the remote node, the **recvstats** daemon on the master node, and two shared memory files in the directory `/var/shm`. If any part of the system breaks down, this function could fail.

beostat_count_idle_cpus_on_node

Name

beostat_count_idle_cpus_on_node — count number of idle CPUS on a given node

Synopsis

```
#include <sys/beostat.h>
int beostat_count_idle_cpus_on_node (int node, float cpu_idle_threshold);
```

Arguments

node

The node of interest

cpu_idle_threshold

The value of CPU usage below which the CPU will be considered idle.

Description

beostat_count_idle_cpus_on_node executes on the master node and counts the number of CPUs on a given node that have CPU usage below a given threshold.

Examples

```
int cnt;
cnt = beostat_count_idle_cpus_on_node (3, 0.5);
printf ("Node 3 has %d CPUs below 50% usage.\n", cnt);
```

Return Value

Returns the number of CPUs on the give node that have usage below the threshold. If an error occurs, it will return -1.

Errors

This function relies on the **Beostat** subsystem, which consists of the **proc** filesystem on the remote node, the **sendstats** daemon on the remote node, the **recvstats** daemon on the master node, and two shared memory files in the directory /var/shm. If any part of the system breaks down, this function could fail.

beostat_get_avail_nodes_by_id

Name

beostat_get_avail_nodes_by_id — get a list of available nodes for a given identity

Synopsis

```
#include <sys/beostat.h>
int beostat_get_avail_nodes_by_id (int **node_list, uid_t uid, gid_t *gid_list, int gid_size);
```

Arguments

node_list

A handle that will have memory allocated and filled with the array of nodes. This memory must be freed by the caller.

uid

The user identifier number

gid_list

A pointer to a list of group identifier numbers

gid_size

The number of elements in the previous arguments array

Description

beostat_get_avail_nodes_by_id executes on the master node and returns a list of nodes that are available to the given user identifier number who also is a member of the group identifier numbers listed. Memory allocated by the function for *node_list* must be freed by the caller.

Examples

```
int cnt, *node_list, gid_size, i;
uid_t uid;
gid_t *gid_list;
uid = getuid();
gid_size = getgroups (0, gid_list);
gid_list = malloc (sizeof (gid_t) * gid_size);
getgroups (gid_size, gid_list);
cnt = beostat_get_avail_nodes_by_id (&node_list, uid,
gid_list, gid_size);
printf ("You may run jobs on nodes: ");
for (i = 0; i < cnt; i++)
```

```
printf ("%d ", node_list[i]);
printf ("\n");
free (gid_list);
free (node_list);
```

Return Value

Returns the number of nodes in *node_list*. If an error occurs, it will return -1.

Errors

This function relies on the **Beostat** subsystem, which consists of the **proc** filesystem on the remote node, the **sendstats** daemon on the remote node, the **recvstats** daemon on the master node, and two shared memory files in the directory `/var/shm`. If any part of the system breaks down, this function could fail.

beostat_get_cpu_count

Name

`beostat_get_cpu_count` — return the number of CPUs on the specified node

Synopsis

```
#include <sys/beostat.h>
int beostat_get_cpu_count (int node, size_t *ncpus);
```

Arguments

node

The node to query.

ncpus

A pointer to a size_t, which upon successful completion will contain the number of CPUs on the node specified by the *node* parameter.

Description

`beostat_get_cpu_count` executes on the master node and returns the number of CPUs on a specified compute node. A CPU count of zero means that the *node*'s **sendstats** daemon is not executing.

Return Value

Return 0 on success. If an error occurs, it will return -1.

Errors

This function relies on the **Beostat** subsystem, which consists of the **proc** filesystem on the remote node, the **sendstats** daemon on the remote node, the **recvstats** daemon on the master node, and two shared memory files in the directory `/var/shm`. If any part of the system breaks down, this function could fail.

beostat_get_cpu_percent

Name

`beostat_get_cpu_percent` — get the CPU usage on a node

Synopsis

```
#include <sys/beostat.h>
float beostat_get_cpu_percent (int node, int cpu);
```

Arguments

node

The node to query

cpu

The CPU index on the particular node

Description

`beostat_get_cpu_percent` executes on the master node and returns the current CPU usage as a floating-point value between 0.0 and 1.0.

Examples

```
printf ("CPU 0 on node 3 is %f percent busy.\n", beostat_get_cpu_percent (3, 0));
```

Return Value

Return a float between 0.0 and 1.0. If an error occurs, it will return -1.0.

Errors

This function relies on the **Beostat** subsystem, which consists of the **proc** filesystem on the remote node, the **sendstats** daemon on the remote node, the **recvstats** daemon on the master node, and two shared memory files in the directory `/var/shm`. If any part of the system breaks down, this function could fail.

beostat_get_cpuinfo_x86

Name

beostat_get_cpuinfo_x86 — get the time of the last update for node

Synopsis

```
#include <sys/beostat.h>
int beostat_get_cpuinfo_x86 (int node, struct cpuinfo_x86*cpuinfo);
```

Arguments

node

The node to query

cpuinfo

A pointer to a *struct beostat_cpuinfo_x86*, which is defined as follows (names in comments are entries from cpuinfo_x86 in asm/processor.h):

```
struct beostat_cpuinfo_x86
{
  int processor;            /* [which cpu (SMP)] */
  char vendor_id[16];       /* x86_vendor_id */
  int family;               /* x86 */
  int model;                /* x86 model */
  char name[64];            /* x86 model ID */
  int stepping;             /* x86 mask */
  float MHz;                /* derived from bogomips */
  int cache_size_KB;        /* x86_cache_size */
  boolean fdiv_bug;         /* same */
  boolean hlt_bug;          /* ~hlt_works_ok */
  boolean sep_bug;          /* [Derived] */
  boolean f00f_bug;         /* same */
  boolean coma_bug;         /* same */
  boolean fpu;              /* hard_math */
  boolean fpu_exception;    /* based on exception 16 */
  int cpuid_level;          /* same */
  boolean wp;               /* wp_works_ok */
  float bogomips;           /* loops_per_sec derived */
};
```

Description

beostat_get_cpuinfo_x86 executes on the master node and returns a structure describing information about the CPU on the host node. The information in this structure parallels the output seen in /proc/cpuinfo. Note that since this information is architecture specific, this function has "x86" in its name.

Examples

```
struct beostat_cpuinfo_x86 cpuinfo;
beostat_get_cpuinfo_x86 (3, &cpuinfo);
printf ("Node 3 has a %f MHz processor\n", cpuinfo.MHz);
```

Return Value

Return 0 on success. If an error occurs, it will return -1.

Errors

This function relies on the **Beostat** subsystem, which consists of the **proc** filesystem on the remote node, the **sendstats** daemon on the remote node, the **recvstats** daemon on the master node, and two shared memory files in the directory `/var/shm`. If any part of the system breaks down, this function could fail.

beostat_get_disk_usage

Name

`beostat_get_disk_usage` — get the disk usage on root partition of a node

Synopsis

```
#include <sys/beostat.h>
int beostat_get_disk_usage (int node, int *max, int *curr);
```

Arguments

node

The node to query

max

A pointer to an *int*. Upon successful completion, will contain the capacity of the root partition of the node's disk in megabytes.

curr

A pointer to an *int*. Upon successful completion will contain the current usage of the root partition of the node's disk in megabytes.

Description

`beostat_get_disk_usage` executes on the master node and returns the current disk usage, as well as the total capacity of the disk in megabytes.

Examples

```
int max, curr;
beostat_get_disk_usage (3, &max, &curr)
printf ("CPU 0 on node 3's disk is %f percent full.\n",
  (double) curr / (double) max );
```

Return Value

Returns 0 upon successful completion. If an error occurs, it will return -1.

Errors

This function relies on the **Beostat** subsystem, which consists of the **proc** filesystem on the remote node, the **sendstats** daemon on the remote node, the **recvstats** daemon on the master node, and two shared memory files in the directory `/var/shm`. If any part of the system breaks down, this function could fail.

beostat_get_last_multicast

Name

`beostat_get_last_multicast` — get file system statistics for the root file system on a node

Synopsis

```
#include <sys/beostat.h>
time_t beostat_get_last_multicast (void);
```

Description

`beostat_get_last_multicast` executes on the master node and returns the time of the last multicast request sent to the nodes. It is usually reserved for internal use.

Return Value

Returns the time in seconds since Epoch (00:00:00 UTC, January 1, 1970) of the last multicast request. If an error occurs, it will return -1.

Errors

This function relies on the **Beostat** subsystem, which consists of the **proc** filesystem on the remote node, the **sendstats** daemon on the remote node, the **recvstats** daemon on the master node, and two shared memory files in the directory `/var/shm`. If any part of the system breaks down, this function could fail.

beostat_get_loadavg

Name

`beostat_get_loadavg` — get load average on a node

Synopsis

```
#include <sys/beostat.h>
int beostat_get_loadavg (int node, struct beostat_loadavg *loadavg);
```

Arguments

node

The node to query

loadavg

A pointer to a *struct beostat_loadavg*, which is defined as follows:

```
struct beostat_loadavg
{
  float load[3];
  int num_active_procs;
  int total_procs;
  int last_pid;
};
```

Description

`beostat_get_loadavg` executes on the master node and returns the load average information of a node in the cluster. The three values returned are averages over increasing time durations.

Examples

```
struct beostat_loadavg loadavg;
beostat_get_loadavg (3, &loadavg);
printf ("The load process ID on node 3 was %d.\n", loadavg.last_pid);
```

Return Value

Return 0 on success. If an error occurs, it will return -1.

Errors

This function relies on the **Beostat** subsystem, which consists of the **proc** filesystem on the remote node, the **sendstats** daemon on the remote node, the **recvstats** daemon on the master node, and two shared memory files in the directory `/var/shm`. If any part of the system breaks down, this function could fail.

beostat_get_meminfo

Name

`beostat_get_meminfo` — get information about the memory usage on a node

Synopsis

```
#include <sys/beostat.h>
int beostat_get_meminfo (int node, struct beostat_meminfo *meminfo);
```

Arguments

node

The node to query

meminfo

A pointer to a *struct beostat_meminfo*, which is defined as follows:

```
struct beostat_meminfo
{
  struct beostat_memusage mem;
  struct beostat_memusage swap;
  unsigned long long shared;
  unsigned long long buffers;
  unsigned long long cached;
};
```

where *struct beostat_memusage* is defined as follows:

```
struct beostat_memusage
{
  unsigned long long used;
  unsigned long long free;
};
```

Description

`beostat_get_meminfo` executes on the master node and returns the memory usage of a node in the cluster. All values are in bytes.

Warning: Since Linux aggressively caches the hard disk into memory it will often appear to always be about 90% used. Some have suggested that the values of `buffers` and `cached` added together should be subtracted from the reported memory usage. However, these values may not be mutually exclusive.

Examples

```
meminfo_t meminfo;
beostat_get_meminfo (3, &meminfo);
printf ("The node 3 has %s bytes free\n", meminfo.mem.free);
```

Return Value

Return 0 on success. If an error occurs, it will return -1.

Errors

This function relies on the **Beostat** subsystem, which consists of the **proc** filesystem on the remote node, the **sendstats** daemon on the remote node, the **recvstats** daemon on the master node, and two shared memory files in the directory `/var/shm`. If any part of the system breaks down, this function could fail.

beostat_get_MHz

Name

`beostat_get_MHz` — get the speed of the processor on a node

Synopsis

```
#include <sys/beostat.h>
int beostat_get_MHz (int node, float *MHz);
```

Arguments

node

The node to query

MHz

A pointer to a float, which will contain the speed of processor on the node in megahertz upon successful completion.

Description

`beostat_get_MHz` executes on the master node and returns the speed of CPU(s) on a given node in units of megahertz. On multi-CPU (SMP) machines it is assumed that all CPUs are the same speed. This is currently a hardware requirement on all known SMP machines.

Examples

```
float speed;
beostat_get_MHz (3, &speed);
printf ("The node 3 has a %f MHz processor\n", speed);
```

Return Value

Return 0 on success. If an error occurs, it will return -1.

Errors

This function relies on the **Beostat** subsystem, which consists of the **proc** filesystem on the remote node, the **sendstats** daemon on the remote node, the **recvstats** daemon on the master node, and two shared memory files in the directory `/var/shm`. If any part of the system breaks down, this function could fail.

beostat_get_name

Name

`beostat_get_name` — get the name of node

Synopsis

```
#include <sys/beostat.h>
int beostat_get_name (int node, char **name);
```

Arguments

node

The node to query

name

A handle to a *char*, which will be allocated with an appropriate amount of memory and then set to the name of a node. The caller must free the allocated memory when it is done with the memory.

Description

`beostat_get_name` executes on the master node and returns the name of a given node.

Examples

```
char *name;
beostat_get_name (3, &name);
printf ("The name for node 3 is %s\n", name);
free (name);
```

Return Value

Return 0 on success. If an error occurs, it will return -1.

Errors

This function relies on the **Beostat** subsystem, which consists of the **proc** filesystem on the remote node, the **sendstats** daemon on the remote node, the **recvstats** daemon on the master node, and two shared memory files in the directory `/var/shm`. If any part of the system breaks down, this function could fail.

beostat_get_net_dev

Name

beostat_get_net_dev — get the network interface statistics on a node

Synopsis

```
#include <sys/beostat.h>
int beostat_get_net_dev (int node, struct beostat_net_dev *devs, int size);
```

Arguments

node

The node to query

devs

A pointer to a array of structures of the type struct beostat_net_dev, which is defined as follows:

```
struct beostat_net_dev
{
  char name[16];
  struct beostat_net_stat recv;
  unsigned long frame;
  unsigned long multicast;
  struct beostat_net_stat trans;
  unsigned long colls;
  unsigned long carrier;
};
```

where beostat_net_stat is defined as follows:

```
struct beostat_net_stat
{
  unsigned long bytes;
  unsigned long packets;
  unsigned long errs;
  unsigned long drop;
  unsigned long fifo;
  unsigned long compressed;
};
```

size

The number of beostat_net_dev structures allocated by the caller.

Description

`beostat_get_net_dev` executes on the master node and returns the network interface statistics of a node. The caller must allocate the memory for the array of structures, and a maximum of `MAX_NET_DEV` or *`size`* entries will be filled (whichever is smaller). Unused space in the structure(s) are filled with zeros.

Examples

```
int i;
struct beostat_net_dev net_dev[MAX_NET_DEV];
beostat_get_net_dev (3, net_dev, MAX_NET_DEV);
for (i = 0; i < MAX_NET_DEV; I++)
  if (net_dev[i].name[0] != 0)
    printf ("%ld bytes of interface %s has been received on node 3.\n",
      net_dev[i].recv.bytes, net_dev[i].name
```

Return Value

Return 0 on success. If an error occurs, it will return -1.

Errors

This function relies on the **Beostat** subsystem, which consists of the **proc** filesystem on the remote node, the **sendstats** daemon on the remote node, the **recvstats** daemon on the master node, and two shared memory files in the directory `/var/shm`. If any part of the system breaks down, this function could fail.

beostat_get_net_rate

Name

`beostat_get_net_rate` — get the cumulative network interface on a node

Synopsis

```
#include <sys/beostat.h>
unsigned long beostat_get_net_rate (int node);
```

Arguments

node

The node to query

Description

`beostat_get_net_rate` executes on the master node and returns the current network usage rate in bytes per second across all interfaces on that node.

Examples

```
 printf ("Node 3 is currently transferring %d bytes / second.\n", beostat_get_net_rate (3));
```

This function can give erroneous results for its transfer counts during the moment of rollover of each interface.

Return Value

Returns an unsigned long, which represents the network transfer rate. If an error occurs, it will return -1.

Errors

This function relies on the **Beostat** subsystem, which consists of the **proc** filesystem on the remote node, the **sendstats** daemon on the remote node, the **recvstats** daemon on the master node, and two shared memory files in the directory `/var/shm`. If any part of the system breaks down, this function could fail.

beostat_get_stat_cpu

Name

beostat_get_stat_cpu — get the statistics of CPU utilization

Synopsis

```
#include <sys/beostat.h>
int beostat_get_stat_cpu (int node, int cpu, struct beostat_stat_cpu *stat_cpu);
```

Arguments

node

The node to query

cpu

The CPU index on the particular node

stat_cpu

A pointer to a *struct beostat_stat_cpu*, which will be filled upon successful completion. *struct beostat_stat_cpu* is defined as follows:

```
struct beostat_stat_cpu
{
  long user;
  long system;
  long nice;
  long idle;
};
```

The members of this structure have the following meanings:

user

The number of CPU ticks spend processing normal priority (0) user level instructions.

nice

The number of CPU ticks spend processing nice priority (>0) user level instructions.

system

The number of CPU ticks spend processing system (kernel) level instructions.

idle

The number of CPU ticks spend idle.

Description

`beostat_get_stat_cpu` executes on the master node and returns the cpu ticks counts on a given node/CPU. These ticks just keep incrementing over time until they overflow and wrap back around. To get actual CPU usage over some time period, you must either take the derivative of these values or use the **beostat** convenience function `beostat_get_cpu_percent`.

Examples

```
struct beostat_stat_cpu stat_cpu;
beostat_get_stat_cpu (3, 0, &stat_cpu);
printf ("There have been %ld idle ticks on cpu 0 for node 3 is %s\n", stat_cpu.idle);
free (name);
```

Return Value

Return 0 on success. If an error occurs, it will return -1.

Errors

This function relies on the **Beostat** subsystem, which consists of the **proc** filesystem on the remote node, the **sendstats** daemon on the remote node, the **recvstats** daemon on the master node, and two shared memory files in the directory `/var/shm`. If any part of the system breaks down, this function could fail.

beostat_get_statfs_p

Name

`beostat_get_statfs_p` — get file system statistics for the root file system on a node

Synopsis

```
#include <sys/beostat.h>
int beostat_get_statfs_p (int node, struct statfs *statfs);
```

Arguments

node

The node to query

statfs

A pointer to a statfs structure that will be filled upon successful completion. See the man page for statfs(2) for a description of the fields.

Description

`beostat_get_statfs_p` executes on the master node and returns the filesystem statistics for the root filesystem on a given node.

Warning: Since Linux aggressively caches the hard disk into memory it will often appear to always be about 90% used. Some have suggested that the values of `buffers` and `cached` added together should be subtracted from the reported memory usage. However, these values may not be mutually exclusive.

Examples

```
statfs_p_t statfs_p;
beostat_get_statfs_p (3, &statfs_p);
printf ("The node 3 has %s bytes free\n", statfs_p.mem.free);
```

Return Value

Return 0 on success. If an error occurs, it will return -1.

Errors

This function relies on the **Beostat** subsystem, which consists of the **proc** filesystem on the remote node, the **sendstats** daemon on the remote node, the **recvstats** daemon on the master node, and two shared memory files in the directory `/var/shm`. If any part of the system breaks down, this function could fail.

beostat_get_time

Name

`beostat_get_time` — get the time of the last update for node

Synopsis

```
#include <sys/beostat.h>
int beostat_get_time (int node, struct node_time *node_time);
```

Arguments

node

The node to query

node_time

A pointer to a *struct node_time*, which is defined as follows:

```
struct node_time {
  time_t time;
};
```

Description

`beostat_get_time` executes on the master node and returns the time of the last update to the **Beostat** system by a given node. The **Beostat** functionality works by the having the **sendstats** daemon on each compute node periodically send node status information to the master node's **recvstats** daemon. This function provides the time of the last update from a given node. It is useful when timely information is required and old information should be disregarded. The time is measured in seconds since the standard UNIX Epoch (00:00:00 UTC, January 1, 1970). Use functions like `ctime()` to convert to a human readable string.

Examples

```
time_t time;
beostat_get_time (3, &time);
printf ("The time of the last update for node 3 is %s\n", ctime(&time));
```

Return Value

Return 0 on success. If an error occurs, it will return -1.

Errors

This function relies on the **Beostat** subsystem, which consists of the **proc** filesystem on the remote node, the **sendstats** daemon on the remote node, the **recvstats** daemon on the master node, and two shared memory files in the directory `/var/shm`. If any part of the system breaks down, this function could fail.

beostat_is_node_available

Name

beostat_is_node_available — determine if a given user/group can run on a given node

Synopsis

```
#include <sys/beostat.h>
int beostat_is_node_available (int node, uid_t uid, gid_t *gid_list, int gid_size);
```

Arguments

node

The node of interest

uid

The user identifier number

gid_list

A pointer to a list of group identifier numbers

gid_size

The number of elements in the previous arguments array

Description

beostat_is_node_available executes on the master node and determines if the given user with specified UID and belonging to the groups in *gid_list* has permission to run on a given node.

See the manual page for beostat_get_avail_nodes_by_id for a example of a similar function.

Return Value

Returns 1 if the node can be used, 0 if not, and -1 if an error occurs.

Errors

This function relies on the **Beostat** subsystem, which consists of the **proc** filesystem on the remote node, the **sendstats** daemon on the remote node, the **recvstats** daemon on the master node, and two shared memory files in the directory /var/shm. If any part of the system breaks down, this function could fail.

beostat_set_last_multicast

Name

`beostat_set_last_multicast` — set the time of the last multicast transmission to the compute nodes

Synopsis

```
#include <sys/beostat.h>
int beostat_set_last_multicast (time_t last);
```

Arguments

last

The time in seconds since the Epoch (00:00:00 UTC, January 1, 1970)

Description

`beostat_set_last_multicast` executes on the master node and returns the time that the last multicast event occurred. This should not be used unless you really know what you are doing. It is for internal use.

Return Value

Return 0 on success. If an error occurs, it will return -1.

Errors

This function relies on the **Beostat** subsystem, which consists of the **proc** filesystem on the remote node, the **sendstats** daemon on the remote node, the **recvstats** daemon on the master node, and two shared memory files in the directory `/var/shm`. If any part of the system breaks down, this function could fail.

Scyld ClusterWare BProc Libraries

This part of the *Reference Guide* describes the functions included in the Scyld ClusterWare C libraries for **BProc**, the Beowulf Process Control library. The functions in this library are used to control jobs running on the cluster.

bproc_access

Name

`bproc_access` — Check if the current user may use a cluster node.

Synopsis

```
#include <sys/bproc.h>
int bproc_access (int node, int mode);
int _bproc_access (struct bproc_node_info_t *nodeinfo, int mode);
```

Arguments

node

The node number to check.

mode

The mode bits to check against.

nodeinfo

A filled-in bproc_node_info structure to check against.

Description

The current user's ability to execute processes on the specified cluster *node* is checked. The *mode* parameter specifies the mode bits to check.

See the Administrator's Guide for details of the semantics of node ownership and how the settings interact with schedulers.

Return Value

If a process may be started on the node, 0 is returned.

If the node is not available or there is an error, -1 is returned and `errno` is set.

Errors

ENOSYS

The **BProc** system is not available.

EIO

The **BProc** system is loaded but is not configured or active.

EACCES

This user does not have permission to start jobs on the node.

ENOMEM

Insufficient kernel memory was available.

See Also

bproc_chown(3), bproc_chgrp(3), bproc_chmod(3)

bproc_chown

Name

`bproc_chown` — Change the ownership for a cluster node.

Synopsis

```
#include <sys/bproc.h>
int bproc_chown (int node, int user);
int bproc_chgrp (int node, int group);
int bproc_chown (int node, int user, int group);
```

Arguments

node

The node to change ownership of

user

The numeric user ID to assign to the node

group

The numeric group ID to assign to the node

Description

The owner of the cluster *node* is changed. The *user* specifies the desired user ID (UID). The second form, available only with the **BProc** v2 compatibility library, sets the group owner to *group*.

Previous **BProc** versions used the values `BPROC_USER_ANY` and `BPROC_GROUP_ANY`. The same effect can be achieved by setting world-execute permission using bproc_chmod(3)

See the Administrator's Guide for details of the semantics of node ownership and how the settings interact with schedulers.

Return Value

Returns 0 on success.

Returns -1 on error, and sets `errno`.

Errors

ENOSYS

The **BProc** system is not available.

EIO

The **BProc** system is loaded but not configured or active.

EPERM

This process does not have permission to change node ownership.

ENOMEM

Insufficient kernel memory was available to change ownership.

See Also

bproc_access(3), bproc_chgrp(3), bproc_chmod(3)

bproc_currnode

Name

`bproc_currnode` — Get the current node number

Synopsis

```
#include <sys/bproc.h>
int bproc_currnode (void);
```

Return Value

Returns the node number of the machine on which this process is currently running. The value `BPROC_NODE_MASTER`, -1, indicates that the process is running on the master.

Bugs

This function will return -1 if there is an error in processing, or if you are on the master node. If there is the possibility of ambiguity, the `errno` variable should be initialized to 0 before the call and checked for errors after the call.

Errors

ENOSYS

The **BProc** system is not available.

EIO

The **BProc** system is loaded, but is not configured or active.

ENOMEM

Insufficient kernel memory was available to return a value.

bproc_detach

Name

bproc_detach — Remove the current process from the BProc process space.

Synopsis

```
#include <sys/bproc.h>
int bproc_detach(long code);
```

Description

bproc_detach removes the current process from the global BProc process space. After bproc_detach succeeds, the process continues to execute on its node, but is no longer visible from other nodes. From the viewpoint of other processes in the BProc system, the effect is as if the process had executed exit(*code*). See exit(2) for a description of the effects seen by the parent process.

Return Value

On success, bproc_detach returns zero.

On error, bproc_detach returns -1 and sets *errno* appropriately.

Errors

EPERM

The caller does not have root permissions.

ENOSYS

The **BProc** system is not loaded in the current kernel.

In addition, any errors listed in fork(2) may occur.

Notes

In the current release, bproc_detach may change the PID and PPID of the current process, but is not guaranteed to do so. The PPID may change at some point after bproc_detach has returned, but such behavior is not guaranteed.

In future releases, bproc_detach may be implemented to remove the current process from the BProc system, or it may be implemented to make a copy of the current process outside of the BProc system; other options are also possible. The user should make no assumptions beyond those documented in this man page.

Some implementations of GNU libc cache the value of getpid in userspace; thus, getpid may return inaccurate values if called both before and after bproc_detach.

See Also

exit(2), fork(2)

bproc_execmove

Name

`bproc_execmove` — Exec a local binary on a remote node

Synopsis

```
#include <sys/bproc.h>
int _bproc_execmove_io (int node, int port, const char * cmd, char * const argv[], char *
const envp[]);
int bproc_execmove (int node, const char * cmd, char * const argv[], char * const envp[]);
```

Arguments

node

The destination node for the child process.

port

The IP port **BProc** should connect back to for I/O forwarding.

cmd

The program to execute

argv

The argument list

envp

The environment

Description

This function allows execution of local binaries on remote nodes. **BProc** will load the binary image on the current node and then move it to a remote node, prior to executing the binary image.

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. Function does not return on success. On failure, it returns -1 and sets `errno` appropriately.

port is the TCP port **BProc** should connect back to to handle I/O forwarding. A *port* value of 0 means it assumes I/O forwarding is being done on the existing socket for `stdout` and `stderr` only. Any other value and it will try to connect back to that port and open three connections, one for `stdout`, one for `stderr`, and one for `stdin`.

If you use `bproc_execmove`, *port* has a default value of 0.

Return Value

Does not return on success.

Returns on error, and sets `errno`.

Errors

EPERM

The filesystem where *cmd* resides is mounted nosuid and the program is suid or sgid

ENOMEM

Out of memory

EBUSY

No Master

EFAULT

cmd, *envp*, or *argv* points to memory that is not accessible the by the program.

EACCES

The program does not have execute permission on *cmd*

E2BIG

Argument list is too big

ENOEXEC

cmd is not in a recognized executable format or is for the wrong architecture

ENAMETOOLONG

cmd is too long

ENOENT

cmd does not exist.

ENOTDIR

Part of the path to *cmd* is not a directory.

ELOOP

Too many symbolic links were encountered when resolving *cmd*.

ETXTBSY

cmd is open for writing by another program.

EIO

An I/O error occurred.

ENFILE

The limit on open files has been reached.

EINVAL

An ELF executable had more than one `PT_INTERP` segment.

bproc_getnodebyname

Name

`bproc_getnodebyname` — Get a node number from a node name.

Synopsis

```
#include <sys/bproc.h>
int bproc_getnodebyname (const char * name);
```

Arguments

name

A machine name

Description

This function returns the node number associated with the string *name*. Valid strings include "master", "self", a string representation of a decimal number, and the string representation of a decimal number prepended with a "."

Note that this function duplicates some of the functionality of the BeoNSS system, but with a limited set of names. Note also that this function does not use **BProc** kernel information: it returns a value even when the cluster system is not active, and it may return a node number that is outside the valid range of nodes.

Return Value

Returns the node number represented by *name*.

May return `BPROC_NODE_SELF` if the string is "self".

Returns `BPROC_NODE_NONE` if a valid string was not passed.

Errors

No errors

bproc_masteraddr

Name

`bproc_masteraddr` — Get the private cluster network IP address for the master node.

Synopsis

```
#include <sys/bproc.h>
int bproc_masteraddr (struct sockaddr * addr, int * size);
```

Arguments

addr

pointer to a struct sockaddr

size

The size of *addr*

Description

Save the master node's IP address in the struct sockaddr pointed to by *addr*. *size* 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).

Return Value

Returns 0 on success.

Returns -1 on error, and sets `errno`.

Errors

EFAULT

addr or *size* points to memory that is not accessible by the program.

EIO

There was an I/O error.

ENOMEM

Out of memory error.

bproc_move

Name

bproc_move — Move the running process to another node

Synopsis

```
#include <sys/bproc.h>
int _bproc_move_io (int node, int flags, int port);
int _bproc_move (int node, int flags);
int bproc_move (int node);
```

Arguments

node

The node to move to

flags

Flags for VMAdump.

port

The IP port **BProc** should connect back to for I/O forwarding.

Description

This call will move the current process to the remote node number given by *node*. It returns 0 on success, -1 on failure. errno is set on failure.

node is the node to move to.

flags can be one of the following: BPROC_DUMP_LIBS, BPROC_DUMP_EXEC, BPROC_DUMP_OTHER or any combination of them binary OR'd together. A binary OR of all three, BPROC_DUMP_ALL, is also provided as a shortcut. These flags tell **VMAdump** how much of the running process to dump and send to the compute node.

port is the port **BProc** should connect back to, to handle I/O forwarding. A *port* value of 0 means it assumes I/O forwarding is being done on the existing socket for stdout and stderr only. Any other value and it will try to connect back to that port and open three connections, one for stdout, one for stderr, and one for stdin.

If you use _bproc_move or bproc_move, *port* has a default value of 0. If you use bproc_move, *flags* takes a default value that is BPROC_DUMP_EXEC|BPROC_DUMP_OTHER if you're trying to move to an up node or the master, otherwise it is BPROC_DUMP_EXEC|BPROC_DUMP_LIBS|BPROC_DUMP_OTHER

Return Value

Returns 0 on success.

Returns -1 on error, and sets `errno`.

Errors

EBUSY

No master?

ENOMEM

Out of memory.

EIO

An I/O error occurred.

bproc_nodeaddr

Name

`bproc_nodeaddr` — Get the IP address for a node.

Synopsis

```
#include <sys/bproc.h>
int bproc_nodeaddr (int node, struct sockaddr * addr, int * size);
```

Arguments

node

The node number

addr

pointer to a struct sockaddr

size

The size of *addr*

Description

Save the node's IP address in the struct sockaddr pointed to by *addr*. The *size* element 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).

Return Value

Returns 0 on success.

Returns -1 on error, and sets `errno`.

Errors

EFAULT

addr or *size* points to memory that is not accessible by the program.

EIO

There was an I/O error.

ENOMEM

Out of memory error.

bproc_nodeinfo

Name

`bproc_nodeinfo` — Get general status information for a node

Synopsis

```
#include <sys/bproc.h>
int bproc_nodeinfo (int node, struct bproc_node_info_t * info);
```

Arguments

node

The node you want information on.

info

Pointer to a struct bproc_node_info_t.

Description

This function will get information about the node and fill that information into the struct bproc_node_info_t.

```
struct bproc_node_info_t {
   int      node;         /* Same as bproc_currnode()
   int      status;       /* Same as bproc_nodestatus() */
   int      mode;         /* The node's access permissions */
   uid_t    user;         /* The uid and gid of the user    */
   gid_t    group;        /*  to which the node is assigned */
   uint32_t addr;         /* The node's 32-bit struct sockaddr */
}
```

See the Administrator's Guide for more information on the user and group.

Return Value

Returns 0 on success.

Returns -1 on error, and sets `errno`.

Errors

EFAULT

info points to memory that is inaccessible by the program.

EIO

I/O Error

ENOMEM

Out of Memory

bproc_nodenumber

Name

`bproc_nodenumber` — Get the node number based on the given IP address.

Synopsis

```
#include <sys/bproc.h>
int bproc_nodenumber (struct sockaddr * addr, int size);
```

Arguments

addr

pointer to a struct sockaddr, that has the IP filled in

size

The size of *addr*

Description

Retrieves the IP address from the sockaddr structure and provides the number of the node with that address. There is a direct one-to-one mapping of node number to IP address as given in the `/etc/beowulf/config` file. Node numbering starts at 0 with the first IP address in the range and increments by 1 up to the last IP address in the range.

Return Value

Returns the node number associated with the IP address.

Returns `BPROC_NODE_NONE` if no valid node was found.

Errors

EFAULT

addr points to memory that is not accessible by the program.

EIO

There was an I/O error.

ENOMEM

Out of memory error.

bproc_nodestatus

Name

`bproc_nodestatus` — Returns the status of the given node.

Synopsis

```
#include <sys/bproc.h>
int bproc_nodestatus (int node);
```

Arguments

node

The node number.

Description

This node argument should list one of the compute nodes, not the master. The master is considered to be always up.

Return Value

On error, it will return -1 and set `errno` appropriately.

The possible states are:

`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. Nodes are only in this state if set that way explicitly by the administrator.

`bproc_node_error`

There is a problem with the node. Nodes are assigned this state if booting is unsuccessful.

`bproc_node_up`

The node is up and ready to accept processes. This is the only state in which non-root users can send jobs to the node.

`bproc_node_reboot`

The node was told to reboot and has not come back up yet.

`bproc_node_halt`

The node was told to halt and is still down.

`bproc_node_pwroff`

The node was told to power off and is still down.

`bproc_node_boot`

The node is in the process of coming up (running the **node_up** script).

Errors

There was an I/O error.

ENOMEM

Out of memory error.

bproc_numnodes

Name

`bproc_numnodes` — Get the count of cluster nodes.

Synopsis

```
#include <sys/bproc.h>
int bproc_numnodes (void );
```

Description

This function returns the number of nodes the cluster is configured to support. Note that this is the potential size of the cluster, not the current number of available nodes or the count of machines assigned node numbers.

Return Value

Returns the number of compute nodes the system is configured to support. If the **BProc** system is not loaded, returns 0 and sets `errno` to `ENOSYS`.

Returns -1 on error, and sets `errno`.

Errors

EIO

The **BProc** system is loaded but not configured or active.

ENOMEM

Insufficient kernel memory was available.

See Also

bproc_currnode(3), bproc_nodeinfo(3), bpstat(1)

bproc_pidghostnode

Name

`bproc_pidghostnode` — Get the node a PID started on.

Synopsis

```
#include <sys/bproc.h>
int bproc_pidghostnode (int pid);
```

Arguments

pid

The process id

Description

Retrieves the node number associated with starting the given Process ID in the **BProc** unified process space. Typically, this will be the master node, but may be another node if the process called a **BProc** move function.

Return Value

Return the node number that the PID started on.

Returns `BPROC_NODE_NONE` if the PID isn't one that was moved through **BProc** or isn't a valid PID.

BUGS

This functions returns `BPROC_NODE_NONE` if there was an error accessing the **BProc** status file, or if *pid* was not found in the status file. There is currently no way to tell if `BPROC_NODE_NONE` resulted from an error or from *pid* not being masqueraded by **BProc**.

Errors

EACCES

The program does not have access to read the **BProc** status file.

ENOENT

The **BProc** status file does not exist.

ENOMEM

Insufficient kernel memory.

EMFILE

The limit of files that can be opened by the process has been reached.

ENFILE

The limit of files that can be opened by the system has been reached.

EAGAIN

The **BProc** status file has been locked.

bproc_pidnode

Name

`bproc_pidnode` — Get the node a PID is running on.

Synopsis

```
#include <sys/bproc.h>
int bproc_pidnode (int pid);
```

Arguments

pid

The process id

Description

Retrieves the node number associated with the given Process ID from the **BProc** process space. Note that only user processes ghosted on the master are available in this way. Node kernel and internal **BProc** PIDs are not accessible.

Return Value

Return the node number that the PID is running on.

Returns `BPROC_NODE_NONE` if the PID is running on the master node or isn't a valid PID.

Bugs

This functions returns `BPROC_NODE_NONE` if there was an error accessing the **BProc** status file, or if *pid* was not found in the status file. There is currently no way to tell if `BPROC_NODE_NONE` resulted from an error or from *pid* not being masqueraded by **BProc**.

Errors

EACCES

The program does not have access to read the **BProc** status file.

ENOENT

The **BProc** status file does not exist.

ENOMEM

Insufficient kernel memory.

EMFILE

The limit of files that can be opened by the process has been reached.

ENFILE

The limit of files that can be opened by the system has been reached.

EAGAIN

The **BProc** status file has been locked.

bproc_rexec

Name

bproc_rexec — exec a program on a remote node

Synopsis

```
#include <sys/bproc.h>
int _bproc_rexec_io (int node, int port, const char * cmd, char * const argv[], char * const
envp[]);
int bproc_rexec (int node, const char * cmd, char * const argv[], char * const envp[]);
```

Arguments

node

The node the child should be on.

port

The port to **BProc** should connect back to for I/O forwarding.

cmd

The program to execute

argv

The argument list

envp

The environment

Description

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 and errno is set appropriately.

port is the TCP port **BProc** should connect back to to handle I/O forwarding. A *port* value of 0 means it assumes I/O forwarding is being done on the existing socket for stdout and stderr only. Any other value and it will try to connect back to that port and open three connections, one for stdout, one for stderr, and one for stdin.

If you use bproc_execmove, *port* has a default value of 0.

Return Value

Does not return on success.

Returns -1 on error, and sets `errno`.

Errors

EPERM

The filesystem where *cmd* resides is mounted nosuid and the program is suid or sgid

ENOMEM

Out of memory

EBUSY

No Master

EFAULT

cmd, *envp*, or *argv* points to memory that is not accessible by the program.

EACCES

The program does not have execute permission on *cmd*

E2BIG

Argument list is too big

ENOEXEC

cmd is not in a recognized executable format or is for the wrong architecture

ENAMETOOLONG

cmd is too long

ENOENT

cmd does not exist.

ENOTDIR

Part of the path to *cmd* is not a directory.

ELOOP

Too many symbolic links were encountered when resolving *cmd*.

ETXTBSY

cmd is open for writing by another program.

EIO

An I/O error occurred.

ENFILE

The limit on open files has been reached.

EINVAL

An ELF executable had more than one PT_INTERP segment.

bproc_rfork

Name

bproc_rfork — fork, with the child ending up on a remote node.

Synopsis

```
#include <sys/bproc.h>
int _bproc_rfork_io (int node, int flags, int port);
int _bproc_rfork (int node, int flags);
int bproc_rfork (int node);
```

Arguments

node

The node the child should be on.

flags

Flags for **VMAdump**.

port

The port **BProc** should connect back to for I/O forwarding.

Description

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 SIGCHLDs 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 0 to the child. On failure it returns -1, and errno is set appropriately.

node is the node the child should be on.

flags can be one of the following: BPROC_DUMP_LIBS, BPROC_DUMP_EXEC, BPROC_DUMP_OTHER or any combination of them binary OR'd together. If you with to use all of them, you can also use BPROC_DUMP_ALL as a shortcut. These flags tell **VMAdump** how much of the running process to dump and send to the compute node.

port is the port **BProc** should connect back to for I/O forwarding. A port value of zero means it assumes I/O forwarding is being done on the existing socket for stdout and stderr only. Any other value and it will try to connect back to that port and open three connections, one for stdout, one for stderr, and one for stdin.

If you use _bproc_rfork or bproc_rfork, port has a default value of 0. If you use bproc_rfork, *flags* takes a default value that is BPROC_DUMP_EXEC|BPROC_DUMP_OTHER, if you are trying to move to an up node or the master, otherwise it is BPROC_DUMP_EXEC|BPROC_DUMP_LIBS|BPROC_DUMP_OTHER.

Return Value

For the parent process, this will return the PID of the child process.

For the child process, this will return 0.

If there is an error, -1 will be returned to the parent process and there will be no child process.

Errors

EBUSY

No Master

bproc_setnodestatus

Name

bproc_setnodestatus — Change the status of a node

Synopsis

```
#include <sys/bproc.h>
int bproc_setnodestatus (int node, int status);
```

Arguments

node

The node to change the status of

status

The new status for the node

Description

This call sets the status of a node. Note that it is not possible to change the status of a node that is marked as "down", "pwroff", or "halt".

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. Nodes are only in this state if set that way explicitly by the administrator.

bproc_node_error

There is a problem with the node. Nodes are assigned this state if booting is unsuccessful.

bproc_node_up

The node is up and ready to accept processes. This is the only state in which non-root users can send jobs to the node.

bproc_node_reboot

Setting a node to this state will tell it to reboot.

bproc_node_halt

Setting a node to this state will tell it to halt.

`bproc_node_pwroff`

Setting a node to this state will tell it to power off.

`bproc_node_boot`

The node is in the process of coming up (running the **node_up** script). A node should only be put in this state by the **BProc** master daemon.

Return Value

Returns 0 on success.

Returns -1 on error, and sets `errno`.

Errors

EPERM

You do not have root access

ENOMEM

Out of memory

EIO

I/O error

bproc_slave_chroot

Name

`bproc_slave_chroot` — Request the slave daemon to perform a `chroot`

Synopsis

```
#include <sys/bproc.h>
int bproc_slave_chroot (int node, const char * path);
```

Arguments

`node`

The node you want to perform the `chroot`

`path`

The path to `chroot` to

Description

This call requests the slave daemon to perform a `chroot` to the path specified by *`path`*. It is the **BProc** equivalent to the standard `chroot`, which sets the base path for filename lookup.

Return Value

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

Errors

EPERM

You do not have root access

ENOMEM

Out of memory

E2BIG

`path` is too long.

EIO

I/O error