How to Shield Your Linux Resources

SUSE Linux Enterprise Real Time 12 SP3

How to Shield Your Linux Resources

SUSE Linux Enterprise Real Time 12 SP3

by Alex Tsariounov

Publication Date: September 07, 2018

SUSE LLC
10 Canal Park Drive
Suite 200
Cambridge MA 02141
USA
https://www.suse.com/documentation

Contents

1 Introduction

In the Linux kernel, the cpuset facility provides a mechanism for creating logical entities called "cpusets" that encompass definitions of CPUs and NUMA Memory Nodes (if NUMA is available). Cpusets constrain the CPU and Memory placement of a task to only the resources defined within that cpuset. These cpusets can then be arranged into a nested hierarchy visible in the "cpuset" virtual file system. Sets of tasks can be assigned to these cpusets to constrain the resources that they use. The tasks can be moved from one cpuset to another to use other resources defined in those other cpusets.

The `cset` command is a Python application that provides a command line front-end for the Linux cpusets functionality. Working with cpusets directly can be confusing and slightly complex. The cset tool hides that complexity behind an easy-to-use command line interface.

There are two distinct use cases for cset: the basic shielding use case and the "advanced" case of using raw `set` and `proc` subcommands. The basic shielding function is accessed with the `shield` subcommand and described in the next section. Using the raw `set` and `proc` subcommands allows one to set up arbitrarily complex cpusets and is described in *Chapter 3, Full Featured Cpuset Manipulation Commands*.

Note that in general, one either uses the `shield` subcommand or a combination of the `set` and `proc` subcommands. One rarely, if ever, uses all of these subcommands together. Doing so will likely become too confusing. Additionally, the `shield` subcommand sets up its required cpusets with exclusively marked CPUs. This can interfere with your cpuset strategy. If you find that you need more functionality for your strategy than `shield` provides, go ahead and transition to using `set` and `proc` exclusively. It is straightforward to implement what `shield` does with a few extra `set` and `proc` subcommands.

OBTAINING ONLINE HELP

For a full list of cset subcommands

```
tux > cset help
```

For in-depth help on individual subcommands

```
tux > cset help <subcommand>
```

For options on individual subcommands

```
tux > cset <subcommand> (-h | --help)
```

2 The Basic Shielding Model

Although any setup of `cpuset`s can really be described as *shielding*, there is one prevalent shielding model in use that is so common that cset has a subcommand that is dedicated to its use. This subcommand is called **`shield`**.

The concept behind this model is the use of three `cpuset`s:

- **Root `cpuset`.** is always present in all configurations and contains all CPUs.
- **System `cpuset`.** contains CPUs which are used for system tasks. These are the normal tasks that are not important, but which need to run on the system.
- **User `cpuset`.** "the shield", contains CPUs which are used for important tasks. Only those tasks that are somehow important, usually tasks whose performance determines the overall rating for the machine, are run in the `user` `cpuset`.

The **`shield`** subcommand manages all of these `cpuset`s and lets you define the CPUs and memory nodes that are in the `shielded` and `unshielded` sets. The subcommand automatically moves all movable tasks on the system into the `unshielded` `cpuset` on shield activation, and back into the `root` `cpuset` on shield tear down. The subcommand lets you move tasks into and out of the shield. Additionally, you can move special tasks (kernel threads) which normally run in the `root` `cpuset` into the `unshielded` set. This makes your shield have even less disturbance.

The **`shield`** subcommand abstracts the management of these `cpuset`s away from you. It provides options that drive how the shield is set up, which tasks are to be shielded or not, and the status of the shield. In fact, you need not be bothered with the naming of the required `cpuset`s or even where the `cpuset` file system is mounted. **`cset`** and the **`shield`** subcommand takes care of all that.

If you need to define more `cpuset`s for your application, it is likely that this simple shielding is not rich enough for you. In this case, you should transition to using the **`set`** and **`proc`** subcommands described in *Chapter 3, Full Featured Cpuset Manipulation Commands*.

2.1 A Simple Shielding Example

Assume that we have a 4-core machine that has uniform memory access. This means there are 4 CPUs at your disposal and there is only one memory node available. On such machines, we do not need to specify any memory node parameters to cset, it sets up the only available memory node by default.

Usually, one wants to dedicate as many CPUs to the shield as possible and leave a minimal set of CPUs for normal system processing. The reasoning for this is, the performance of the important tasks will rule the performance of the installation as a whole. These important tasks need as many resources available to them as possible, exclusive of other, unimportant tasks that are running on the system.

Note: Definition of Task

In this document *task* is used to represent either a process or a thread that is running on the system.

2.2 Setup and Teardown of the Shield

To set up a shield of 3 CPUs with 1 CPU left for low priority system processing, issue the following command.

```
tux > cset shield -c 1-3
cset: --> activating shielding:
cset: moving 176 tasks from root into system cpuset...
[==================================================]%
cset: "system" cpuset of CPUSPEC(0) with 176 tasks running
cset: "user" cpuset of CPUSPEC(1-3) with 0 tasks running
```

This command does several things. First, it creates a `user` `cpuset` with what is called a CPUSPEC (CPU specification) from the `-c/--cpu` option. This CPUSPEC specifies to use CPUs 1 through 3 inclusively. Next, the command creates a `system` `cpuset` with a CPUSPEC that is the inverse of the `-c` option for the current machine. On this machine that `cpuset` will only contain the first CPU, CPU0. Next, all user space processes running in the `root` `cpuset` are transferred to the `system` `cpuset`. This makes all those processes run only on CPU0. The effect of this is that the shield consists of CPUs 1 through 3 and they are now idling.

Note that the command did not move the kernel threads that are running in the `root` `cpuset` to the `system` `cpuset`. This is because you may want these kernel threads to use all available CPUs. If you do not, then you can use the `-k`/`--kthread` option as described below.

The shield setup command above outputs the information of which `cpuset`s were created and how many tasks are running on each. To see the current status of the shield again, issue this command:

```
tux > cset shield
```

```
cset: --> shielding system active with
cset: "system" cpuset of CPUSPEC(0) with 176 tasks running
cset: "user" cpuset of CPUSPEC(1-3) with 0 tasks running
```

Which shows us that the shield is set up and that 176 tasks are running in the `system` `cpuset` — the *unshielded* `cpuset`.

It is important to move all possible tasks from the `root` `cpuset` to the unshielded `system` `cpuset` because a task's `cpuset` property is inherited by its children. Since we've moved all running tasks (including init) to the unshielded `system` `cpuset`, that means that any new tasks that are spawned will also run in the unshielded `system` `cpuset`.

Some kernel threads can be moved into the unshielded `system` `cpuset` as well. These are the threads that are not bound to specific CPUs. If a kernel thread is bound to a specific CPU, then it is generally not a good idea to move that thread to the `system` set because at worst it may hang the system and at best it will slow the system down significantly. These threads are usually the IRQ threads on a real time Linux kernel, for example, and you should not move these kernel threads into `system`. If you leave them in the `root` `cpuset`, then they will have access to all CPUs.

However, if your application demands an even "quieter" shield, then you can move all movable kernel threads into the unshielded `system` set with the following command.

```
tux > cset shield -k on
cset: --> activating kthread shielding
cset: kthread shield activated, moving 70 tasks into system cpuset...
[==================================================]%
cset: done
```

You can see that this moved an additional 70 tasks to the unshielded `system` `cpuset`. Note that the `-k`/`--kthread on` parameter can be given at the shield's creation time. You do not need to perform these two steps separately if you know you will want kernel thread shielding as well. Executing **`cset shield`** again shows us the current state of the shield.

```
tux > cset shield
cset: --> shielding system active with
cset: "system" cpuset of CPUSPEC(0) with 246 tasks running
cset: "user" cpuset of CPUSPEC(1-3) with 0 tasks running
```

You can get a detailed listing of what is running in the shield by adding either `-s/--shield` or `-u/--unshield` to the **`shield`** subcommand and using the verbose flag. You will get output similar to the following.

```
tux > cset shield --unshield -v
```

```
cset: "system" cpuset of CPUSPEC(0) with 251 tasks running
USER     PID   PPID  SPPr TASK NAME
-------- ----- ----- ---- ---------
root     1     0     Soth init [5]
root     2     0     Soth [kthreadd]
root     84    2     Sf50 [IRQ-9
]...
alext    31796 31789 Soth less
root     32653 25222 Roth python ./cset shield --unshield -v
```

Note that the listing is abbreviated; we do have 251 tasks running in the `system` set. However, the `SPPr` field may need a little explanation. SPPr stands for `State, Policy and Priority`. You can see that the initial two tasks are Stopped and running in timeshare priority, marked as `oth` (for `other`). The [IRQ-9] task is also stopped, but marked at real time FIFO policy with a priority of 50. The last task in the listing is the **`cset`** command itself and is marked as running. Also note that adding a second `-v/--verbose` option will not restrict the output to fit into an 80 character screen.

Tear down of the shield, stopping the shield in other words, is done with the `-r/--reset` option to the shield subcommand. When this command is issued, both the `system` and `user` `cpuset`s are deleted and any tasks that are running in both of those `cpuset`s are moved to the `root` `cpuset`. Once so moved, all tasks will have access to all resources on the system. For example:

```
tux > cset shield --reset
cset: --> deactivating/reseting shielding
cset: moving 0 tasks from "/user" user set to root set...
cset: moving 250 tasks from "/system" system set to root set...
[==================================================]%
cset: deleting "/user" and "/system" sets
cset: done
```

2.3 Moving Interesting Tasks Into and Out of the Shield

Now that we have a shield running, the objective is to run our “important” processes in that shield. These processes can be anything, but usually they are directly related to the purpose of the machine. There are two ways to run tasks in the shield:

- Exec a process into the shield
- Move an already running task into the shield

2.3.1 Execing a Process Into the Shield

Running a new process in the shield can be done with the `-e/--exec` option to the **`shield`** subcommand. This is the simplest way to get a task to run in the shield. For this example, let's exec a new Bash shell into the shield with the following commands.

```
tux > cset shield -s
cset: "user" cpuset of CPUSPEC(1-3) with 0 tasks running
cset: done


tux > cset shield -e bash
cset: --> last message, executed args into cpuset "/user", new pid is: 13300


tux > cset shield -s -v
cset: "user" cpuset of CPUSPEC(1-3) with 2 tasks running
USER     PID   PPID  SPPr TASK NAME
-------- ----- ----- ---- ---------
root     13300 8583  Soth bash
root     13329 13300 Roth python ./cset shield -s -v


tux > exit


tux > cset shield -s
cset: "user" cpuset of CPUSPEC(1-3) with 0 tasks running
cset: done
```

The first command above lists the status of the shield. You see that the shield is defined as CPUs 1 through 3 inclusive and currently there are no tasks running in it.

The second command execs the Bash shell into the shield with the `-e` option. The last message of **`cset`** lists the PID of the new process.

Note: Separating the Tool Options From the cset Command

cset follows the tradition of separating the tool options from the command to be execed options with a double dash (`--`). This is not shown in this simple example, but if the command you want to exec also takes options, separate them with the double dash as follows:

```
tux > cset shield -e mycommand -- -v
```

The `-v` will be passed to mycommand, and not to cset.

The next command lists the status of the shield again. There are two tasks running shielded: our new shell and the cset status command itself. Remember that the `cpuset` property of a task is inherited by its children. Since we ran the new shell in the shield, its child, which is the status command, also ran in the shield.

Tip: Execing a Shell Into the Shield Is Useful

Execing a shell into the shield is a useful way to experiment with running tasks in the shield since all children of the shell will also run in the shield.

The last command exits the shell after which we request a shield status again and see that, once again, it does not contain any tasks.

You may have noticed in the output above that both the new shell and the status command are running as the `root` user. This is because cset needs to run as `root` and so all it is children will also run as `root`. If you need to run a process under a different user and or group, you may use the `--user` and `--group` options for execution as follows.

```
tux > cset shield --user=alext --group=users -e bash
cset: --> last message, executed args into cpuset "/user", new pid is: 14212


tux > cset shield -s -v
cset: "user" cpuset of CPUSPEC(1-3) with 2 tasks running
USER     PID   PPID  SPPr TASK NAME
-------- ----- ----- ---- ---------
alext    14212 8583  Soth bash
alext    14241 14212 Roth python ./cset shield -s -v
```

2.3.2 Moving a Running Task Into and Out of the Shield

While execing a process into the shield is undoubtedly useful, most of the time, you will want to move already running tasks into and out of the shield. The **`cset`** shield subcommand includes two options for doing this: `-s/--shield` and `-u/--unshield`. These options require a PIDSPEC (process specification) to also be specified with the `-p/--pid` option. The PIDSPEC defines which tasks get operated on. The PIDSPEC can be a single process ID, a list of process IDs separated by commas, and a list of process ID ranges separated by dashes, groups of which are separated by commas. For example:

`--shield --pid` 1234
: This PIDSPEC argument specifies that PID `1234` be shielded.

`--shield --pid` 1234,42,1934,15000,15001,15002
: This PIDSPEC argument specifies that this list of PIDs only be moved into the shield.

`--unshield -p` 5000,5100,6010-7000,9232
: This PIDSPEC argument specifies that PIDs `5000`, `5100` and `9232` be unshielded (moved out of the shield) along with any existing PID that is in the range `6010` through `7000` inclusive.

Note: Information About the Range In a PIDSPEC

A range in a PIDSPEC does not need to have tasks running for every number in that range. In fact, it is not even an error if there are no tasks running in that range: none will be moved in that case. The range only specifies to act on any tasks that have a PID or TID that is within that range.

Use of the appropriate PIDSPEC can thus be handy to move tasks and groups of tasks into and out of the shield. Additionally, there is one more option that can help with multi-threaded processes, and that is the `--threads` flag. If this flag is used together with a **`shield`** or **`unshield`** command with a PIDSPEC and if any of the task IDs in the PIDSPEC belong to a thread in a process container, then all the sibling threads in that process container will get shielded or unshielded as well. This flag provides an easy mechanism to shield/unshield all threads of a process by simply specifying one thread in that process.

In the following example, we move the current shell into the shield with a range PIDSPEC and back out with the Bash variable for the current PID.

```
tux > echo $$
```

```
22018


tux > cset shield -s -p 22010-22020
cset: --> shielding following pidspec: 22010-22020
cset: done


tux > cset shield -s -v
cset: "user" cpuset of CPUSPEC(1-3) with 2 tasks running
USER     PID   PPID  SPPr TASK NAME
-------- ----- ----- ---- ---------
root     3770  22018 Roth python ./cset shield -s -v
root     22018 5034  Soth bash
cset: done


tux > cset shield -u -p $$
cset: --> unshielding following pidspec: 22018
cset: done


tux > cset shield -s
cset: "user" cpuset of CPUSPEC(1-3) with 0 tasks running
cset: done
```

3 Full Featured Cpuset Manipulation Commands

While basic shielding as described above is useful and a common use model for **cset**, there comes a time when more functionality will be desired to implement your strategy. To implement this, **cset** provides two subcommands: **set**, which allows you to manipulate cpusets; and **proc**, which allows you to manipulate processes within those cpusets.

3.1 The set Subcommand

To do anything with cpusets, you must be able to create, adjust, rename, move, and destroy them. The **set** subcommand allows the management of cpusets in such a manner.

3.1.1 Creating and Destroying Cpusets with **set**

The basic syntax of **set** for cpuset creation is:

```
tux > cset set -c 1-3 -s my_cpuset1
cset: --> created cpuset "my_cpuset1"
```

This creates a cpuset named `my_cpuset1` with a CPUSPEC of CPU1, CPU2 and CPU3. The CPUSPEC is the same concept as described in the *Section 2.2, “Setup and Teardown of the Shield”*. The **set** subcommand also takes a `-m`/`--mem` option that lets you specify the memory nodes the **set** will use and flags to make the CPUs and MEMs exclusive to the cpuset. If you are on a non-NUMA machine, leave the `-m` option out and the default memory node `0` will be used.

Like with **shield**, you can adjust the CPUs and MEMs with subsequent calls to set. If, for example, you want to adjust the `my_cpuset1` cpuset to only use CPUs 1 and 3 (and omit CPU2), then issue the following command.

```
tux > cset set -c 1,3 -s my_cpuset1
cset: --> modified cpuset "my_cpuset
```

cset will then adjust the CPUs that are assigned to the `my_cpuset1` set to only use CPU1 and CPU3.

To rename a cpuset, use the `-n/--newname` option. For example:

```
tux > cset set -s my_cpuset1 -n super_set
```

```
cset: --> renaming "/cpusets/my_cpuset1" to "super_set"
```

Renames the cpuset called `my_cpuset1` to `super_set`.

To destroy a cpuset, use the `-d/--destroy` option as follows.

```
tux > cset set -d super_set
cset: --> processing cpuset "super_set", moving 0 tasks to parent "/"...
cset: --> deleting cpuset "/super_set"
cset: done
```

This command destroys the newly created cpuset called `super_set`. When a cpuset is destroyed, all the tasks running in it are moved to the parent cpuset. The root cpuset, which always exists and always contains all CPUs, cannot be destroyed. You may also give the `--destroy` option a list of cpusets to destroy.

Note: Information About the Mounted Cpuset File System

The **`cset`** subcommand creates the cpusets based on a mounted cpuset file system. You do not need to know where that file system is mounted, although it is easy to figure out (by default it is on `/cpusets`). When you give the **`set`** subcommand a name for a new cpuset, it is created wherever the cpuset file system is mounted.

To create a cpuset hierarchy, then you must give a path to the **`cset set`** subcommand. This path will always begin with the root cpuset, for which the path is `/`. For example:

```
tux > cset set -c 1,3 -s top_set
cset: --> created cpuset "top_set"


tux > cset set -c 3 -s /top_set/sub_set
cset: --> created cpuset "/top_set/sub_set"
```

These commands created two cpusets: `top_set` and `sub_set`. The `top_set` uses CPU1 and CPU3. It has a subset of `sub_set` which only uses CPU3. Once you have created a subset with a path, then if the name is unique, you do not need to specify the path to affect it. If the name is not unique, then **`cset`** will complain and ask you to use the path. For example:

```
tux > cset set -c 1,3 -s sub_set
cset: --> modified cpuset "sub_set
```

This command adds CPU1 to the `sub_set` cpuset for its use. Note that using the path in this case is optional.

If you attempt to destroy a cpuset which has sub-cpusets, **`cset`** will complain and not do it unless you use the `-r/--recurse` and the `--force` options. If you do use `--force`, then all the tasks running in all subsets of the deletion target cpuset will be moved to the target's parent cpuset and all cpusets.

Moving a cpuset from under a certain cpuset to a different location is not implemented.

3.1.2 Listing Cpusets with set

To list cpusets, use the **`set`** subcommand with the `-l/--list` option. For example:

```
tux > cset set -l
cset:
Name         CPUs-X       MEMs-X    Tasks Subs Path
------------ ---------- - ------- - ----- ---- ----------
root         0-3 y        0 y       320   1    /
one          3 n          0 n       0     1    /one
```

This shows that there is currently one cpuset present called `one`. (Of course there is also the `root` set, which is always present.) The output shows that the one cpuset has no tasks running in it. The `root` cpuset has 320 tasks running. The `-X` for CPUs and MEMs fields denotes whether the CPUs and MEMs in the cpusets are marked exclusive to those cpusets. Note that the `one` cpuset has subsets as indicated by a `1` in the `Subs` field. You can specify a cpuset to list with the **`set`** subcommand as follows:

```
tux > cset set -l -s one
cset:
Name         CPUs-X       MEMs-X    Tasks Subs Path
------------ ---------- - ------- - ----- ---- ----------
one          3 n          0 n       0     1   /one
two          3 n          0 n       0     1   /one/two
```

This output shows that there is a cpuset called `two` in cpuset `one` and it also has subset. You can also ask for a recursive listing as follows:

```
tux > cset set -l -r
cset:
Name         CPUs-X       MEMs-X    Tasks Subs Path
------------ ---------- - ------- - ----- ---- ----------
```

```
root         0-3 y        0 y       320   1    /
one          3 n          0 n       0     1    /one
two          3 n          0 n       0     1    /one/two
three        3 n          0 n       0     0    /one/two/three
```

This command lists all cpusets existing on the system since it asks for a recursive listing beginning at the `root` cpuset. Incidentally, should you need to specify the `root` cpuset you can use either `root` or `/` to specify it explicitly—just remember that the `root` cpuset cannot be deleted or modified.

3.2 The **proc** Subcommand

Now that you know how to create, rename and destroy cpusets with the **`set`** subcommand, the next step is to manage threads and processes in those cpusets. The subcommand to do this is called **`proc`** and it allows you to exec processes into a cpuset, move existing tasks around existing cpusets, and list tasks running in specified cpusets. For the following examples, let us assume a cpuset setup of two sets as follows:

```
tux > cset set -l
cset:
Name         CPUs-X       MEMs-X    Tasks Subs Path
------------ ---------- - ------- - ----- ---- ----------
root         0-3 y        0 y       309   2    /
two          2 n          0 n       3     0    /two
three        3 n          0 n       10    0    /three
```

3.2.1 Listing Tasks With **proc**

Operation of the **`proc`** subcommand follows the same model as the **`set`** subcommand. For example, to list tasks in a cpuset, you need to use the `-l/--list` option and specify the cpuset by name or, if the name exists multiple times in the cpuset hierarchy, by path. For example:

```
tux > cset proc -l -s two
cset: "two" cpuset of CPUSPEC(2) with 3 tasks running
USER     PID   PPID  SPPr TASK NAME
-------- ----- ----- ---- ---------
root     16141 4300  Soth bash
root     16171 16141 Soth bash
root     16703 16171 Roth python ./cset proc -l two
```

This output shows us that the cpuset called two has CPU2 only attached to it and is running three tasks: two shells and the **`python`** command to list it. Note that cpusets are inherited so that if a process is contained in a cpuset, then any children it spawns also run within that set. In this case, the **`python`** command to list set `two` was run from a shell already running in set `two`. This can be seen by the PPID (parent process ID) of the **`python`** command matching the PID of the shell.

Additionally, the `SPPr` field needs explanation. SPPr stands for `State, Policy and Priority`. You can see that the initial two tasks are stopped and running in timeshare priority, marked as `oth` (for `other`). The last task is marked as running, `R` and at timeshare priority, `oth`. If any of these tasks would have been at real time priority, the policy would be shown as `f` for FIFO or `r` for round robin. The priority would be a number from 1 to 99. See below for an example.

```
tux > cset proc -l -s root | head -7
cset: "root" cpuset of CPUSPEC(0-3) with 309 tasks running
USER      PID   PPID  SPPr TASK NAME
-------- ----- ----- ---- ---------
root        1     0 Soth init [5]
root        2     0 Soth [kthreadd]
root        3     2 Sf99 [migration/0]
root        4     2 Sf99 [posix_cpu_timer]
```

This output shows the first few tasks in the `root` cpuset. Note that both `init` and `[kthread]` are running at timeshare; however, the `[migration/0]` and `[posix_cpu_timer]` kernel threads are running at real-time policy of FIFO and priority of `99`. Incidentally, this output is from a system running the real-time Linux kernel which runs some kernel threads at real-time priorities. And finally, note that you can use **`cset`** as any other Linux tool and include it in pipelines as in the example above.

Taking a peek into the third cpuset called `three`, we see:

```
tux > cset proc -l -s three
cset: "three" cpuset of CPUSPEC(3) with 10 tasks running
USER      PID   PPID  SPPr TASK NAME
-------- ----- ----- ---- ---------
alext    16165     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
alext    16169     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
alext    16170     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
alext    16237     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
alext    16491     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
alext    16492     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
alext    16493     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
alext    17243     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
alext    17244     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
alext    17265     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
```

This output shows that a lot of `beagled` tasks are running in this cpuset and it also shows an ellipsis (`…`) at the end of their listings. If you see this ellipsis, that means that the command was too long to fit onto an 80 character screen. To see the entire command line, use the `-v/--verbose` flag:

```
tux > cset proc -l -s three -v | head -4
cset: "three" cpuset of CPUSPEC(3) with 10 tasks running
USER     PID   PPID  SPPr TASK NAME
-------- ----- ----- ---- ---------
alext    16165     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg --autostarted
 --indexing-delay 300
```

3.2.2 Execing Tasks with proc

To exec a task into a cpuset, the **`proc`** subcommand needs to be employed with the `-e/--exec` option. Let's exec a shell into the cpuset named `two` in our set. First we check to see what is running that set:

```
tux > cset proc -l -s two
cset: "two" cpuset of CPUSPEC(2) with 0 tasks running

tux > cset proc -s two -e bash
cset: --> last message, executed args into cpuset "/two", new pid is: 20955

tux > cset proc -l -s two
cset: "two" cpuset of CPUSPEC(2) with 2 tasks running
USER     PID   PPID  SPPr TASK NAME
-------- ----- ----- ---- ---------
root     20955 19253 Soth bash
root     20981 20955 Roth python ./cset proc -l two
```

You can see that initially, `two` had nothing running in it. After the completion of the second command, we list `two` again and see that there are two tasks running: the shell which we execed and the **`python cset`** command that is listing the cpuset. The reason for the second task is that the cpuset property of a running task is inherited by all its children. Since we executed the listing command from the new shell which was bound to cpuset two, the resulting process for the listing is also bound to cpuset `two`. Let's test that by running a new shell with no prefixed **`cset`** command.

```
tux > bash
```

```
tux > cset proc -l -s two
cset: "two" cpuset of CPUSPEC(2) with 3 tasks running
USER     PID   PPID  SPPr TASK NAME
-------- ----- ----- ---- ---------
root     20955 19253 Soth bash
root     21118 20955 Soth bash
root     21147 21118 Roth python ./cset proc -l two
```

Here again we see that the second shell, PID `21118`, has a parent PID of `20955` which is the first shell. Both shells, and the listing command, are running in the two cpuset.

Note: Separating the Tool Options From the cset Command

`cset` follows the tradition of separating the tool options from the command to be execed options with a double dash (`--`). This is not shown in this simple example, but if the command you want to exec also takes options, separate them with the double dash as follows:

```
tux > cset proc -s myset -e mycommand -- -v
```

The `-v` will be passed to *`mycommand`*, and not to `cset`.

Tip: Execing a Shell Into the Shield Is Useful

Execing a shell into a cpuset is a useful way to experiment with running tasks in that cpuset since all children of the shell will also run in the same cpuset. Finally, if you misspell the command to be execed, the result may be puzzling. For example:

```
tux > cset proc -s two -e blah-blah
cset: --> last message, executed args into cpuset "/two", new pid is: 21655
cset: **> [Errno 2] No such file or directory
```

The result is no new process even though a new PID is output. The reason for the message is of course that the **`cset`** process forked in preparation for exec, but the command **`blah-blah`** was not found to execute it.

3.2.3 Moving Tasks with **proc**

Although the ability to exec a task into a cpuset is fundamental, you will most likely be moving tasks between cpusets more often. Moving tasks is accomplished with the `-m/--move` and `-p/--pid` options to the **`proc`** subcommand of **`cset`**. The `move` option tells the **`proc`** subcommand that a task move is requested. The `-p/--pid` option takes an argument called a PIDSPEC (PID Specification). The PIDSPEC defines which tasks get operated on.

The PIDSPEC can be a single process ID, a list of process IDs separated by commas, and a list of process ID ranges also separated by commas. For example:

`--pid` 1234
: This PIDSPEC argument specifies that PID `1234` will be moved.

`--pid` 1234,42,1934,15000,15001,15002
: This PIDSPEC argument specifies that only listed tasks will be moved.

`-p` 5000,5100,6010-7000,9232
: This PIDSPEC argument specifies that tasks `5000`, `5100` and `9232` will be moved along with any existing task with PID in the range `6010` through `7000` inclusive.

Note: Information About the Range In a PIDSPEC

A range in a PIDSPEC does not need to have running tasks for every number in that range. In fact, it is not even an error if there are no tasks running in that range; none will be moved in that case. The range simply specifies to act on any tasks that have a PID or TID that is within that range.

In the following example, we move the current shell into the cpuset named `two` with a range PIDSPEC and back out to the `root` cpuset with the Bash variable for the current PID.

```
tux > cset proc -l -s two
cset: "two" cpuset of CPUSPEC(2) with 0 tasks running


tux > echo $$
19253


tux > cset proc -m -p 19250-19260 -t two
cset: moving following pidspec: 19253
```

```
cset: moving 1 userspace tasks to /two
cset: done


tux > cset proc -l -s two
cset: "two" cpuset of CPUSPEC(2) with 2 tasks running
USER     PID   PPID  SPPr TASK NAME
-------- ----- ----- ---- ---------
root     19253 16447 Roth bash
root     29456 19253 Roth python ./cset proc -l -s two


tux > cset proc -m -p $$ -t root
cset: moving following pidspec: 19253
cset: moving 1 userspace tasks to /
cset: done


tux > cset proc -l -s two
cset: "two" cpuset of CPUSPEC(2) with 0 tasks running
```

Use of the appropriate PIDSPEC can thus be handy to move tasks and groups of tasks. Additionally, there is one more option that can help with multi-threaded processes, and that is the `--threads` flag. If this flag is used together with the **`proc`** move command with a PIDSPEC and if any of the task IDs in the PIDSPEC belongs to a thread in a process container, then *all* the sibling threads in that process container will also get moved. This flag provides an easy mechanism to move all threads of a process by simply specifying one thread in that process. In the following example, we move all the threads running in cpuset `three` to cpuset `two` by using the `--threads` flag.

```
tux > cset set two three
cset:
Name         CPUs-X       MEMs-X    Tasks Subs Path
------------ ---------- - ------- - ----- ---- ----------
two          2 n          0 n       0     0    /two
three        3 n          0 n       10    0    /three


tux > cset proc -l -s three
cset: "three" cpuset of CPUSPEC(3) with 10 tasks running
USER     PID   PPID  SPPr TASK NAME
-------- ----- ----- ---- ---------
alext    16165     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
alext    16169     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
alext    16170     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
```

```
alext    16237     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
alext    16491     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
alext    16492     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
alext    16493     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
alext    17243     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
alext    17244     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
alext    27133     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...


tux > cset proc -m -p 16165 --threads -t two
cset: moving following pidspec:
 16491,16493,16492,16170,16165,16169,27133,17244,17243,16237
cset: moving 10 userspace tasks to /two
[==================================================]%
cset: done


tux > cset set two three
cset:
Name         CPUs-X       MEMs-X    Tasks Subs Path
------------ ---------- - ------- - ----- ---- ----------
two          2 n          0 n       10    0    /two
three        3 n          0 n       0     0    /three
```

3.2.3.1 Moving All Tasks From One Cpuset to Another

There is a special case for moving all tasks currently running in one cpuset to another. This can be a common use case, and when you need to do it, specifying a PIDSPEC with `-p` is not necessary so long as you use the `-f/--fromset` *and* the `-t/--toset` options.

In the following example, we move all 10 `beagled` threads back to cpuset `three` with this method.

```
tux > cset proc -l two three
cset: "two" cpuset of CPUSPEC(2) with 10 tasks running
USER      PID   PPID  SPPr TASK NAME
-------- ----- ----- ---- ---------
alext     16165     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -…
alext     16169     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
alext     16170     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
alext     16237     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
alext     16491     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
alext     16492     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
alext     16493     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
alext     17243     1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
```

```
alext      17244      1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
alext      27133      1 Soth beagled /usr/lib64/beagle/BeagleDaemon.exe --bg -...
cset: "three" cpuset of CPUSPEC(3) with 0 tasks running


tux > cset proc -m -f two -t three
cset: moving all tasks from two to /three
cset: moving 10 userspace tasks to /three
[==================================================]%
cset: done


tux > cset set two three
cset:
Name          CPUs-X       MEMs-X    Tasks Subs Path
------------ ---------- - ------- - ----- ---- ----------
two           2 n          0 n       0     0    /two
three         3 n          0 n       10    0    /three
```

3.2.3.2 Moving Kernel Threads With **proc**

Kernel threads are special and **cset** detects tasks that are kernel threads and will refuse to move them unless you also add a `-k/--kthread` option to your **proc** move command. Even if you include `-k`, **cset** will still refuse to move the kernel thread if they are bound to specific CPUs. The reason for this is system protection.

Several kernel threads, especially on the real-time Linux kernel, are bound to specific CPUs and depend on per-CPU kernel variables. If you move these threads to a different CPU than what they are bound to, you risk at best that the system will become horribly slow, and at worst a system hang. If you must move those threads (after all, **cset** needs to give the knowledgeable user access to the keys), then you also need to use the `--force` option.

Warning: Use `--force` With Care

Overriding a task move command with `--force` can have dire consequences for the system. Be sure of the command before you force it.

In the following example, we move all unbound kernel threads running in the root cpuset to the cpuset named two by using the `-k` option.

```
tux > cset proc -k -f root -t two
```

```
cset: moving all kernel threads from / to /two
cset: moving 70 kernel threads to: /two
cset: --> not moving 76 threads (not unbound, use --force)
[==================================================]%
cset: done
```

You will note that we used the fromset→toset facility of the **`proc`** subcommand and we only specified the `-k` option (not the `-m` option). This has the effect of moving all kernel threads only.

Note that only 70 actual kernel threads were moved and 76 were not. The reason that 76 kernel threads were not moved was because they are bound to specific CPUs. Now, let's move those kernel threads back to `root`.

```
tux > cset proc -k -f two -t root
cset: moving all kernel threads from /two to /
cset: ** no task matched move criteria
cset: **> kernel tasks are bound, use --force if ok


tux > cset set -l -s two
cset:
Name         CPUs-X       MEMs-X    Tasks Subs Path
------------ ---------- - ------- - ----- ---- ----------
two          2 n          0 n       70    0    /two
```

`cset` refused to move the kernel threads back to `root` because it says that they are "bound". Let's check this with the **`taskset`** command.

```
tux > cset proc -l -s two | head -5
cset: "two" cpuset of CPUSPEC(2) with 70 tasks running
USER     PID   PPID  SPPr TASK NAME
-------- ----- ----- ---- ---------
root         2     0 Soth [kthreadd]
root        55     2 Soth [khelper]


tux > taskset -p 2
pid 2's current affinity mask: 4


tux > cset set -l -s two
cset:
Name         CPUs-X       MEMs-X    Tasks Subs Path
------------ ---------- - ------- - ----- ---- ----------
two          2 n          0 n       70    0    /two
```

Of course, since the cpuset named `two` only has CPU2 assigned to it, once we moved the unbound kernel threads from `root` to `two`, their affinity masks got automatically changed to only use CPU2. This is evident from the **`taskset`** output which is a hex value. To really move these threads back to root, we need to force the move as follows.

```
tux > cset proc -k -f two -t root --force
cset: moving all kernel threads from /two to /
cset: moving 70 kernel threads to: /
[==================================================]%
cset: done
```

3.2.4 Destroying Tasks

There actually is no **`cset`** subcommand or option to destroy tasks—it's not really needed. Tasks exist and are accessible on the system as normal, even if they happen to be running in one cpuset or another. To destroy tasks, use the usual Ctrl–C method or by using the **`kill(1)`** command.

3.3 Implementing “Shielding” With `set` and `proc`

With the preceding material on the **`set`** and **`proc`** subcommands, we now have the background to implement the basic shielding model, like the **`shield`** subcommand.

One may pose the question why we want to do this, especially since **`shield`** already does it? The answer is that sometimes one needs more functionality than **`shield`** need to implement one's shielding strategy. In those cases you need to first stop using **`shield`** since that subcommand will interfere with the further application of **`set`** and **`proc`**. However, you will still need to implement the functionality of **`shield`** to implement successful shielding.

Remember from the above sections describing **`shield`**, that shielding has at minimum three cpusets: `root`, which is always present and contains all CPUs; `system` which is the *non-shielded* set of CPUs and runs unimportant system tasks; and `user`, which is the *shielded* set of CPUs and runs your important tasks. Remember also that **`shield`** moves all movable tasks into `system` and, optionally, moves unbound kernel threads into `system` as well.

You start first by creating the `system` and `user` cpusets as follows. Let's assume that the machine is a four-CPU machine without NUMA memory features. The system cpuset should hold only CPU0 while the user cpuset should hold the rest of the CPUs.

```
tux > cset set -c 0 -s system
```

```
cset: --> created cpuset "system"


tux > cset set -c 1-3 -s user
cset: --> created cpuset "user"


tux > cset set -l
cset:
Name         CPUs-X     MEMs-X   Tasks Subs Path
------------ ---------- - ------- - ----- ---- ----------
root         0-3 y      0 y      333   2    /
user         1-3 n      0 n      0     0    /user
system       0 n        0 n      0     0    /system
```

Now, we need to move all running user processes into the `system` cpuset.

```
tux > cset proc -m -f root -t system
cset: moving all tasks from root to /system
cset: moving 188 userspace tasks to /system
[==================================================]%
cset: done


tux > cset set -l
cset:
Name         CPUs-X     MEMs-X   Tasks Subs Path
------------ ---------- - ------- - ----- ---- ----------
root         0-3 y      0 y      146    2   /
user         1-3 n      0 n      0      0   /user
system       0 n        0 n      187    0   /system
```

We now have the basic shielding setup. Since all user space tasks are running in `system`, anything that is spawned from them will also run in `system`. The `user` cpuset has nothing running in it unless you put tasks there with the **`proc`** subcommand as described above. If you also want to move movable kernel threads from `root` to `system` (to achieve a form of "interrupt shielding" on a real time Linux kernel, for example), you would execute the following command as well:

```
tux > cset proc -k -f root -t system
cset: moving all kernel threads from / to /system
cset: moving 70 kernel threads to: /system
cset: --> not moving 76 threads (not unbound, use --force)
[==================================================]%
cset: done
```

```
tux > cset set -l
cset:
Name         CPUs-X       MEMs-X    Tasks Subs Path
------------ ---------- - ------- - ----- ---- ----------
root         0-3 y        0 y       76    2    /
user         1-3 n        0 n       0     0    /user
system       0 n          0 n       257   0    /system
```

At this point, you have achieved the simple shielding model that the **`shield`** subcommand provides. You can now add other cpuset definitions to expand your shielding strategy beyond that simple model.

3.4 Implementing Hierarchy With set and proc

One popular extended *shielding* model is based on hierarchical cpusets, each with diminishing numbers of CPUs. This model is used to create *priority cpusets* that allow assignment of CPU resources to tasks based on some arbitrary priority definition. The idea is that a higher priority task will get access to more CPU resources than a lower priority task.

The example provided here once again assumes a machine with four CPUs and no NUMA memory features. This base serves to illustrate the point well; however, note that if your machine has (many) more CPUs, then strategies such as this and others get more interesting.

We define a shielding setup as in the previous section where we have a `system` cpuset with only CPU0 that takes care of "unimportant" system tasks. You will usually require this type of cpuset since it forms the basis of shielding. We modify the strategy to not use a `user` cpuset; instead we create several new cpusets each holding one more CPU than the other. These cpusets will be called `prio_low` with one CPU, `prio_med` with two CPUs, `prio_high` with three CPUs, and `prio_all` with all CPUs.

Note: The Sense Behind Creating a prio_all Cpuset With All CPUs

You may ask, why create a `prio_all` with all CPUs when that is substantially the definition of the `root` cpuset? The answer is that it is best to keep a separation between the `root` cpuset and everything else, even if a particular cpuset duplicates `root` exactly. Usually, automation is build on top of a cpuset strategy. In these cases, it is best to avoid using invariant names of cpusets, such as `root` for example, in this automation.

All of these `prio_*` cpusets can be created under root, in a flat way; however, it is advantageous to create them as a hierarchy. The reasoning for this is twofold: first, if a cpuset is destroyed, all its tasks are moved to its parent; second, one can use exclusive CPUs in a hierarchy.

If a cpuset has CPUs that are exclusive to it, then other cpusets may not use those CPUs unless they are children of that cpuset. This has more relevance to machines with many CPUs and more complex strategies.

Now, we start with a clean slate and build the appropriate cpusets as follows.

```
tux > cset set -r
cset:
Name         CPUs-X       MEMs-X    Tasks Subs Path
------------ ---------- - ------- - ----- ---- ----------
root         0-3 y        0 y       344   0    /


tux > cset set -c 0-3 prio_all
cset: --> created cpuset "prio_all"


tux > cset set -c 1-3 /prio_all/prio_high
cset: --> created cpuset "/prio_all/prio_high"


tux > cset set -c 2-3 /prio_all/prio_high/prio_med
cset: --> created cpuset "/prio_all/prio_high/prio_med"


tux > cset set -c 3 /prio_all/prio_high/prio_med/prio_low
cset: --> created cpuset "/prio_all/prio_high/prio_med/prio_low"


tux > cset set -c 0 system
cset: --> created cpuset "system"


tux > cset set -l -r
cset:
Name         CPUs-X       MEMs-X    Tasks Subs Path
------------ ---------- - ------- - ----- ---- ----------
root         0-3 y        0 y       344   2    /
system       0 n          0 n       0     0    /system
prio_all     0-3 n        0 n       0     1    /prio_all
prio_high    1-3 n        0 n       0     1    /prio_all/prio_high
prio_med     2-3 n        0 n       0     1    /prio_all/prio_high/prio_med
```

```
prio_low     3 n          0 n       0     0     /prio_all/pr...rio_med/prio_low
```

Note: Why -r/--recurse Is Needed in This Case

The option `-r` / `--recurse` lists all the sets in the last command above. If you execute that command without `-r/--recurse`, `prio_med` and `prio_low` cpusets would not appear.

The strategy is now implemented and we now move all user space tasks and all movable kernel threads into the `system` cpuset to activate the shield.

```
tux > cset proc -m -k -f root -t system
cset: moving all tasks from root to /system
cset: moving 198 userspace tasks to /system
cset: moving 70 kernel threads to: /system
cset: --> not moving 76 threads (not unbound, use --force)
[===================================================]%
cset: done


tux > cset set -l -r
cset:
Name         CPUs-X       MEMs-X    Tasks Subs Path
------------ ---------- - ------- - ----- ---- ----------
root         0-3 y        0 y       76    2    /
system       0 n          0 n       268   0    /system
prio_all     0-3 n        0 n       0     1    /prio_all
prio_high    1-3 n        0 n       0     1    /prio_all/prio_high
prio_med     2-3 n        0 n       0     1    /prio_all/prio_high/prio_med
prio_low     3 n          0 n       0     0    /prio_all/pr...rio_med/prio_low
```

The shield is now active. Since the `prio_*` cpuset names are unique, you can assign tasks to them either via their simple name, or their full path (as described in *Section 3.2.2, “Execing Tasks with `proc`”*).

You may have noted that there is an ellipsis in the path of the `prio_low` cpuset in the listing above. This is done to fit the output onto an 80 character screen. To see the entire line, use the `-v` / `--verbose` flag as follows:

```
tux > cset set -l -r -v
cset:
Name         CPUs-X       MEMs-X    Tasks Subs Path
------------ ---------- - ------- - ----- ---- ----------
root         0-3 y        0 y       76    2    /
system       0 n          0 n       268   0    /system
```

```
prio_all     0-3 n        0 n       0     1    /prio_all
prio_high    1-3 n        0 n       0     1    /prio_all/prio_high
prio_med     2-3 n        0 n       0     1    /prio_all/prio_high/prio_med
prio_low     3 n          0 n       0     0    /prio_all/prio_high/prio_med/prio_low
```

4 Using Shortcuts

The commands listed in the previous sections always used all the required options. However, **cset** does have a shortcut facility that will execute certain commands without specifying all options. An effort has been made to do this with the “principle of least surprise”. This means that if you do not specify options, but do specify parameters, then the outcome of the command should be intuitive as possible.

Using shortcuts is not necessary. In fact, you can use either shortcuts or long options. However, using long options instead of shortcuts does have a use case: when you write a script intended to be self-documenting, or perhaps when you generate **cset** documentation.

To begin, the subcommands **shield**, **set** and **proc** can themselves be shortened to the fewest number of characters that are unambiguous. For example, the following commands are identical:

Long method	Short method
`tux > cset shield -s -p 1234`	`tux > cset sh -s -p 1234`
`tux > cset set -c 1,3 -s newset`	`tux > cset se -c 1,3 -s newset`
`tux > cset proc -s newset -e bash`	`tux > cset p -s newset -e bash`

The **proc** command can be shortened to **p**, while **shield** and **set** need two letters to disambiguate.

4.1 shield Subcommand Shortcuts

The **shield** subcommand supports two areas with shortcuts: the short method (when there are no options given and where to shield is the common use case), and the long method (which makes `-p/--pid` optional for the `-s/--shield` and `-u/--unshield` options).

For the common use case of actually shielding either a PIDSPEC or execing a command into the shield, the following **cset** commands are equivalent.

Long method	Short method
`tux > cset shield -s -p 1234,500-649`	`tux > cset sh 1234,500-649`

Long method	Short method
`tux > cset shield -s -e bash`	`tux > cset sh bash`

When using the `-s` or `-u` shield/unshield options, it is optional to use the `-p` option to specify a PIDSPEC. For example:

Long method	Short method
`tux > cset shield -s -p 1234`	`tux > cset sh -s 1234`
`tux > cset shield -u -p 1234`	`tux > cset sh -u 1234`

4.2 set Subcommand Shortcuts

The **`set`** subcommand has a limited number of shortcuts. The option `--set` is optional usually and the `--list` option is also optional to list sets. For example, these commands are equivalent:

Long method	Short method
`tux > cset set -l -s myset`	`tux > cset se -l myset`
`tux > cset se -l myset`	`tux > cset se myset`
`tux > cset set -c 1,2,3 -s newset`	`tux > cset se -c 1,2,3 newset`
`tux > cset set -d -s newset`	`tux > cset se -d newset`
`tux > cset set -n newname -s oldname`	`tux > cset se -n newname oldname`

In fact, if you want to apply either the list or the destroy options to multiple `cpuset`s with one **`cset`** command, you will not need to use the `-s` option. For example:

```
cset se -d myset yourset ourset
--> destroys cpusets: myset, yourset and ourset


cset se -l prio_high prio_med prio_low
```

```
--> lists only cpusets prio_high, prio_med and prio_low
--> the -l is optional in this case since list is default
```

4.3 **proc** Subcommand Shortcuts

For the **proc** subcommand, the `-s`, `-t` and `-f` options to specify the `cpuset`, the origination `cpuset` and the destination `cpuset` can sometimes be optional. For example, the following commands are equivalent. To list tasks in `cpuset` s:

Long method	**Short method**
`tux > cset proc -l -s myset` or `tux > cset proc -l -f myset` or `tux > cset proc -l -t myset`	`tux > cset p -l myset`
`tux > cset p -l myset`	`tux > cset p myset`
`tux > cset proc -l -s one two`	`tux > cset p -l one two`
`tux > cset p -l one two`	`tux > cset p one two`

To execute a process into a `cpuset`:

Long method	**Short method**
`tux > cset proc -s myset -e bash`	`tux > cset p myset -e bash`

Moving tasks into and out of `cpuset` s have the following shortcuts. To move a PIDSPEC into a `cpuset`:

Long method	**Short method**
`tux > cset proc -m -p 4242,4243 -s myset`	`tux > cset p -m 4242,4243 myset`

Long method	Short method
`tux > cset proc -m -p 12 -t myset`	`tux > cset p -m 12 myset`

To move all tasks from one `cpuset` to another:

Long method	Short method

`tux > cset proc -m -f set1 -t set2`
or
`tux > cset proc -m -s set1 -t set2`
or
`tux > cset proc -m -f set1 -s set2`
	`tux > cset p -m set1 set2`

5 What to Do If There Are Problems

If you encounter any issues with the `cset` application, you can file a bug report on our `cset` Bugzilla product found here:

https://code.google.com/p/cpuset/issues/list

If you are using `cset` on a supported operating system such as SUSE Linux Enterprise Server 12 SP3 or SUSE Linux Enterprise Real Time 12 SP3, then should use the following Bugzilla product listing here:

https://bugzilla.suse.com

`cset` contains a logging application that is invaluable for our developers to diagnose problems and find quick solutions. To create a log of your issue, use the `--log` option with a file name as an argument to the main cset application. For example:

```
tux > cset -l logfile.txt set -n newname oldname
```

If the issue persists and is reproducible, Including this report in your bug submission greatly reduces development time. This command saves debugging information within the file `logfile.txt`.

A GNU Licenses

This appendix contains the GNU Free Documentation License version 1.2.

GNU Free Documentation License

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

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

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/ ↗.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

```
  Copyright (c) YEAR YOUR NAME.
  Permission is granted to copy,
 distribute and/or modify this document
  under the terms of the GNU Free
 Documentation License, Version 1.2
  or any later version published by the
 Free Software Foundation;
  with no Invariant Sections, no Front-
Cover Texts, and no Back-Cover Texts.
  A copy of the license is included in
 the section entitled &ldquo;GNU
  Free Documentation License&rdquo;.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

```
  with the Invariant Sections being LIST
 THEIR TITLES, with the
  Front-Cover Texts being LIST, and with
 the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.