Subject: [RFC] libcg: design and plans
Posted by Dhaval Giani on Tue, 04 Mar 2008 15:23:41 GMT
View Forum Message <> Reply to Message

Hi,

We have been working on a library for control groups which would provide
simple APIs for programmers to utilize from userspace and make use of
control groups.

We are still designing the library and the APIs. I've attached the
design (as of now) to get some feedback from the community whether we
are heading in the correct direction and what else should be addressed.

We have a project on sourceforge.net at
https://sourceforge.net/projects/libcg and the mailing list (cc'd here)
can be found at https://lists.sourceforge.net/lists/listinfo/libcg-devel

Thanks,
--

libcg
1. Aims/Requirements
2. Design
3. APIs
4. Configuration Scheme

1. Aims/Requirements

1.1 What are Control Groups

Control Groups provide a mechanism for aggregating/partitioning sets of
tasks, and all their future children, into hierarchical groups with
specialized behaviour [1]. It makes use of a filesystem interface.

1.2 Aims of libcg

libcg aims to provide programmers easily usable APIs to use the control
group file system. It should satisfy the following requirements

1.2.1. Provide a programmable interface for cgroups

This should allow applications to create cgroups using something like
create_cgroup() as opposed to having to go the whole filesystem route.

1.2.2. Provide persistent configuration across reboots

Control Groups have a lifetime of only one boot cycle. The configuration

is lost at reboot. Userspace needs to handle this issue. This is handled
by libcg

1.2.3. Provide a programmable interface for manipulating configurations

This should allow libcg to handle changing application requirements. For
example, while gaming, you might want to reduce the cpu power of other
groups whereas othertimes you would want greater CPU power for those
groups.

2. Design

2.1 Architecture

2.1.1 Global overview

libcg will be consumed in the following fashion

```
   ---------------------------
   |     applications   |
   ---------------------------
   |   libcg    |
   ---------------------------
   |   kernel   |
   ---------------------------
```

A more detailed example would be as follows. Consider various applications
running at the same time on a system. A typical system would be running
a web browser, a mail client, a media player and office software. libcg
could be used to group these applications into various groups and give
them various resources. A possible example would be three groups, Internet,
Entertainment and Office. A daemon could attach tasks to these groups according
to some rules and the adminstrator can control the resources attached to each
group via the configuration manager.i

```
      Internet           Office       Entertainment
  ----------------------       --------------  -----------
  | firefox mutt |       | openoffice |  | mplayer |
  ----------------------       --------------  -----------
    \       |        /
     \       |       /
   ------------------------------------------------------
   |        Some daemon       |
   ------------------------------------------------------
   |                      libcg                    |
   ------------------------------------------------------
   |                      kernel                    |
   ------------------------------------------------------
```

2.1.2

libcg will consist of two main parts. The configuration manager and the
library.

The configuration manager will used to maintain the configurations, to load
and unload the configurations, to set the bootup configurations and so on.
This is similar to the network configuration. A configuration file is used
to setup the networking at bootup. Similarly a configuration file will be
used to setup the default control groups (and maybe the top level control
groups) at bootup.

The administrator can directly access the configuration files, and
applications can access it through the library. The configuration manager
is used to provide the persistence.

```
 Application  Administrator
     |        |
     V         |
 library APIs              |
      \       /
       \     /
  \     /
   V    V
     libcg configuration files
   |
   |
   V
    libcg configuration manager
```

The configuration manager has to provide isolation between various users of
libcg. That is, if two different users A and B are making use of libcg, then
the configuration manager has to ensure that user A does not affect user B's
settings/configurations.

The top level limits and permissions for A and B are to be provided by the
administrator. The permissions are filesystem permissions as cgroup is
filesystem based.

With this architecure in mind, we expect two levels of configuration files.
One would be the global configuration which the administrator would control
and setup the groups, and a local configuration which the group owner will
control.

A simple example could be that the administrator could split the top level
according to uid, and then each user could control the resources available
to him and group those applications accordingly.

```
    root
     |
       ------------------------------------------------
 |      |
 A      B
 |- browsers     |- compilers
 |- games     |- internet
 |- office     |- dev-environment
 |- entertainment    |- others
```

In this example, we have an example cgroup filesystem configuration.

The administrator decides the resources available to "A" and "B". Both "A" and "B" have followed grouping according to their usage. They decide the resources availble to their groups (which is dependent of the resources alloted to them by the adminstrator).

libcg will be written mainly in C with lex and yacc for parsing the configuration files.

3. APIs

The APIs are envisaged to be of two main types

3.1. Manipulating Control Groups

3.1.1. Create Control Group: This API is proposed to create control groups. It should take care of the following scenarios

3.1.1.1 Create non persistent control groups: These groups should exist for just duration of this run. They should not stick across different sessions.

3.1.1.2 Create persistent control groups: These groups should stick across different sessions.

3.1.2 Delete Control Group: This API is proposed to delete control groups. It would have the same scenarios as expected for Create Control Group.

3.1.2 Modify Control Group: This API proposed to modify an already existing group's control files. It too should handle the persistence issue as like Create Control Group does.

More details about configuration are available in sections 2 and 4.

3.2. Manipulating Configurations

3.2.1. Generate Configuration File: If a cgroup filesystem hierarchy already exists,

it should be possible to generate a configuration file which can create it. This
is proposed to be provided by this API.

3.2.2. Change Configuration File: If one configuration is currently loaded in
memory, it is possible for it to be replaced with the new file. This API proposes
to implement that.

3.2.3. Manipulate Configuration File: This API proposes to allow the configuration
file to be modified.

We should also plan on taking care of statistics once its available in mainline.

4. Configuration Scheme

There are multiple configuration levels. The basic wlm.conf file will provide
the mount points and the controller details. This can only be manipulated by
the adminstrator. No APIs will be provided to modify this file.

There will be group specific configuration files as well. The exact details
of the same still need to be worked out.

4.1. Sample configuration files

4.1.1. Sample wlm.conf

```
#
# controller file
#

group ca1 {
 perm {
  task {
   uid = balbir;
   gid = cgroup;
  }
  admin {
   uid = root;
   gid = cgroup;
  }
 }

 cpu {
  cpu.shares = 500;
 }
}

mount {
 cpu = /container;
```

}

This is an example of a top level group. The mount{} block is used to provide
the mount point of the various controllers. For eg, the cpu controller is
mounted at /container. Next we have the group ca1. This is the top level group
and its permissions are given by the uid and gid fields for tasks and admin. The
next is the individual controller block. For the mount point of cpu, the cpu.shares
value is provided. Thus the above file can be represented as the following script

```
mkdir /container
mount -t cgroup -o cpu none /container
mkdir /container/ca1
/bin/echo 500 > /container/ca1/cpu.shares
chown -R root /container/ca1
chgrp -R cgroup /container/ca1
chown balbir /container/ca1/tasks
chgrp cgroup /container/ca1/tasks
```

5. References

1. Documentation/cgroups.txt in kernel sources.
--
regards,
Dhaval