NAME

cgconfig.conf − libcgroup configuration file

DESCRIPTION

cgconfig.conf is a configuration file used by **libcgroup** to define control groups, their parameters and their mount points. The file consists of *mount* , *group* and *default* sections. These sections can be in arbitrary order and all of them are optional. Any line starting with '#' is considered a comment line and is ignored.

mount section has this form:

```
mount {
        <controller> = <path>;
        ...
}
```

controller

Name of the kernel subsystem. The list of subsystems supported by the kernel can be found in */proc/cgroups* file. Named hierarchy can be specified as controller **"name=<somename>"**. Do not forget to use double quotes around this controller name (see examples below).

Libcgroup merges all subsystems mounted to the same directory (see Example 1) and the directory is mounted only once.

path The directory path where the group hierarchy associated to a given controller shall be mounted. The directory is created automatically on cgconfig service startup if it does not exist and is deleted on service shutdown.

If no *mount* section is specified, no controllers are mounted.

group section has this form:

```
group <name> {
        [permissions]
        <controller> {
                <param name> = <param value>;
                ...
        }
        ...
}
```

name Name of the control group. It can contain only characters, which are allowed for directory names. The groups form a tree, i.e. a control group can contain zero or more subgroups. Subgroups can be specified using '/' delimiter.

The root control group is always created automatically in all hierarchies and it is the base of the group hierarchy. It can be explicitly specified in **cgconfig.conf** by using '.' as group name. This can be used e.g. to set its permissions, as shown in Example 6.

When the parent control group of a subgroup is not specified it is created automatically.

permissions

Permissions of the given control group on mounted filesystem. *root* has always permission to do anything with the control group. Permissions have the following syntax:

```
perm {
```

```
                    task {
                            uid = <task user>;
                            gid = <task group>;
                            fperm = <file permissions>
                    }
                    admin {
                            uid = <admin name>;
                            gid = <admin group>;
                            dperm = <directory permissions>
                            fperm = <file permissions>
                    }
            }
```

task user/group Name of the user and the group, which own the *tasks* file of the control group. Given fperm then specify the file permissions. Please note that the given value is not used as was specified. Instead, current file owner permissions are used as a "umask" for group and others permissions. For example if fperm = 777 then both group and others will get the same permissions as the file owner.

admin user/group Name of the user and the group which own the rest of control group's files. Given fperm and dperm control file and directory permissions. Again, the given value is masked by the file/directory owner permissions.

Permissions are only apply to the enclosing control group and are not inherited by subgroups. If there is no **perm** section in the control group definition, *root:root* is the owner of all files and default file permissions are preserved if fperm resp. dperm are not specified.

controller

Name of the kernel subsystem. The section can be empty, default kernel parameters will be used in this case. By specifying **controller** the control group and all its parents are controlled by the specific subsystem. One control group can be controlled by multiple subsystems, even if the subsystems are mounted on different directories. Each control group must be controlled by at least one subsystem, so that **libcgroup** knows in which hierarchies the control group should be created.

The parameters of the given controller can be modified in the following section enclosed in brackets.

param name

Name of the file to set. Each controller can have zero or more parameters.

param value

Value which should be written to the file when the control group is created. If it is enclosed in double quotes '"', it can contain spaces and other special characters.

If no *group* section is specified, no groups are created.

default section has this form:

```
default {
        perm {
                task {
                        uid = <task user>;
                        gid = <task group>;
                        fperm = <file permissions>
                }
                admin {
                        uid = <admin name>;
                        gid = <admin group>;
                        dperm = <directory permissions>
                        fperm = <file permissions>
                }
        }
}
```

Content of the **perm** section has the same form as in *group* section. The permissions defined here specify owner and permissions of groups and files of all groups, which do not have explicitly specified their permissions in their *group* section.

template section has the same structure as **group** section. Template name uses the same templates string as **cgrules.conf** destination tag (see (**cgrules.conf** (5)). Template definition is used as a control group definition for rules in **cgrules.conf** (5) with the same destination name. Templates does not use **default** section settings.

/etc/cgconfig.d/ directory can be used for additional configuration files. cgrulesengd searches this directory for additional templates.

EXAMPLES

Example 1

The configuration file:

```
mount {
        cpu = /mnt/cgroups/cpu;
        cpuacct = /mnt/cgroups/cpu;
}
```

creates the hierarchy controlled by two subsystems with no groups inside. It corresponds to the following operations:

```
mkdir /mnt/cgroups/cpu
mount -t cgroup -o cpu,cpuacct cpu /mnt/cgroups/cpu
```

Example 2

The configuration file:

```
mount {
        cpu = /mnt/cgroups/cpu;
        "name=scheduler" = /mnt/cgroups/cpu;
        "name=noctrl" = /mnt/cgroups/noctrl;
}

group daemons {
        cpu {
                cpu.shares = "1000";
        }
}
group test {
        "name=noctrl" {
        }
}
```

creates two hierarchies. One hierarchy named **scheduler** controlled by cpu subsystem, with group **daemons** inside. Second hierarchy is named **noctrl** without any controller, with group **test**. It corresponds to following operations:

```
mkdir /mnt/cgroups/cpu
mount -t cgroup -o cpu,name=scheduler cpu /mnt/cgroups/cpu
mount -t cgroup -o none,name=noctrl none /mnt/cgroups/noctrl

mkdir /mnt/cgroups/cpu/daemons
echo 1000 > /mnt/cgroups/cpu/daemons/www/cpu.shares

mkdir /mnt/cgroups/noctrl/tests
```

The *daemons* group is created automatically when its first subgroup is created. All its parameters have the default value and only root can access group's files.

Since both *cpuacct* and *cpu* subsystems are mounted to the same directory, all groups are implicitly controlled also by *cpuacct* subsystem, even if there is no *cpuacct* section in any of the groups.

Example 3

The configuration file:

```
mount {
        cpu = /mnt/cgroups/cpu;
        cpuacct = /mnt/cgroups/cpu;
}

group daemons/www {
        perm {
                task {
                        uid = root;
                        gid = webmaster;
                        fperm = 770;
                }
                admin {
                        uid = root;
                        gid = root;
                        dperm = 775;
                        fperm = 744;
                }
        }
        cpu {
                cpu.shares = "1000";
        }
}

group daemons/ftp {
        perm {
                task {
                        uid = root;
                        gid = ftpmaster;
                        fperm = 774;
                }
                admin {
                        uid = root;
                        gid = root;
                        dperm = 755;
                        fperm = 700;
                }
            }
            cpu {
                cpu.shares = "500";
            }
        }
```

creates the hierarchy controlled by two subsystems with one group and two subgroups inside, setting one parameter. It corresponds to the following operations (except for file permissions which are little bit trickier to emulate via chmod):

```
mkdir /mnt/cgroups/cpu
mount -t cgroup -o cpu,cpuacct cpu /mnt/cgroups/cpu

mkdir /mnt/cgroups/cpu/daemons

mkdir /mnt/cgroups/cpu/daemons/www
chown root:root /mnt/cgroups/cpu/daemons/www/*
chown root:webmaster /mnt/cgroups/cpu/daemons/www/tasks
echo 1000 > /mnt/cgroups/cpu/daemons/www/cpu.shares

 # + chmod the files so the result looks like:
 # ls -la /mnt/cgroups/cpu/daemons/www/
 # admin.dperm = 755:
 # drwxr-xr-x. 2 root webmaster 0 Jun 16 11:51 .
 #
 # admin.fperm = 744:
 # --w-------. 1 root webmaster 0 Jun 16 11:51 cgroup.event_control
 # -r--r--r--. 1 root webmaster 0 Jun 16 11:51 cgroup.procs
 # -r--r--r--. 1 root webmaster 0 Jun 16 11:51 cpuacct.stat
 # -rw-r--r--. 1 root webmaster 0 Jun 16 11:51 cpuacct.usage
 # -r--r--r--. 1 root webmaster 0 Jun 16 11:51 cpuacct.usage_percpu
 # -rw-r--r--. 1 root webmaster 0 Jun 16 11:51 cpu.rt_period_us
 # -rw-r--r--. 1 root webmaster 0 Jun 16 11:51 cpu.rt_runtime_us
 # -rw-r--r--. 1 root webmaster 0 Jun 16 11:51 cpu.shares
 # -rw-r--r--. 1 root webmaster 0 Jun 16 11:51 notify_on_release
 #
 # tasks.fperm = 770
 # -rw-rw----. 1 root webmaster 0 Jun 16 11:51 tasks


mkdir /mnt/cgroups/cpu/daemons/ftp
chown root:root /mnt/cgroups/cpu/daemons/ftp/*
chown root:ftpmaster /mnt/cgroups/cpu/daemons/ftp/tasks
echo 500 > /mnt/cgroups/cpu/daemons/ftp/cpu.shares

 # + chmod the files so the result looks like:
 # ls -la /mnt/cgroups/cpu/daemons/ftp/
 # admin.dperm = 755:
 # drwxr-xr-x. 2 root ftpmaster 0 Jun 16 11:51 .
 #
 # admin.fperm = 700:
 # --w-------. 1 root ftpmaster 0 Jun 16 11:51 cgroup.event_control
 # -r--------. 1 root ftpmaster 0 Jun 16 11:51 cgroup.procs
 # -r--------. 1 root ftpmaster 0 Jun 16 11:51 cpuacct.stat
```

```
    # -rw-------. 1 root ftpmaster 0 Jun 16 11:51 cpuacct.usage
    # -r--------. 1 root ftpmaster 0 Jun 16 11:51 cpuacct.usage_percpu
    # -rw-------. 1 root ftpmaster 0 Jun 16 11:51 cpu.rt_period_us
    # -rw-------. 1 root ftpmaster 0 Jun 16 11:51 cpu.rt_runtime_us
    # -rw-------. 1 root ftpmaster 0 Jun 16 11:51 cpu.shares
    # -rw-------. 1 root ftpmaster 0 Jun 16 11:51 notify_on_release
    #
    # tasks.fperm = 774:
    # -rw-rw-r--. 1 root ftpmaster 0 Jun 16 11:51 tasks
```

The *daemons* group is created automatically when its first subgroup is created. All its parameters have the default value and only root can access the group's files.

Since both *cpuacct* and *cpu* subsystems are mounted to the same directory, all groups are implicitly also controlled by the *cpuacct* subsystem, even if there is no *cpuacct* section in any of the groups.

Example 4

The configuration file:

```
mount {
        cpu = /mnt/cgroups/cpu;
        cpuacct = /mnt/cgroups/cpuacct;
}

group daemons {
        cpuacct{
        }
        cpu {
        }
}
```

creates two hierarchies and one common group in both of them. It corresponds to the following operations:

```
mkdir /mnt/cgroups/cpu
mkdir /mnt/cgroups/cpuacct
mount -t cgroup -o cpu cpu /mnt/cgroups/cpu
mount -t cgroup -o cpuacct cpuacct /mnt/cgroups/cpuacct

mkdir /mnt/cgroups/cpu/daemons
mkdir /mnt/cgroups/cpuacct/daemons
```

In fact there are two groups created. One in the *cpuacct* hierarchy, the second in the *cpu* hierarchy. These two groups have nothing in common and can contain different subgroups and different tasks.

Example 5

The configuration file:

```
mount {
        cpu = /mnt/cgroups/cpu;
        cpuacct = /mnt/cgroups/cpuacct;
}

group daemons {
        cpuacct{
		}
	}

	group daemons/www {
		cpu {
			cpu.shares = "1000";
		}
	}

	group daemons/ftp {
		cpu {
			cpu.shares = "500";
		}
	}
```

creates two hierarchies with few groups inside. One of the groups is created in both hierarchies.

It corresponds to the following operations:

```
	mkdir /mnt/cgroups/cpu
	mkdir /mnt/cgroups/cpuacct
	mount -t cgroup -o cpu cpu /mnt/cgroups/cpu
	mount -t cgroup -o cpuacct cpuacct /mnt/cgroups/cpuacct

	mkdir /mnt/cgroups/cpuacct/daemons
	mkdir /mnt/cgroups/cpu/daemons
	mkdir /mnt/cgroups/cpu/daemons/www
	echo 1000 > /mnt/cgroups/cpu/daemons/www/cpu.shares
	mkdir /mnt/cgroups/cpu/daemons/ftp
	echo 500 > /mnt/cgroups/cpu/daemons/ftp/cpu.shares
```

Group *daemons* is created in both hierarchies. In the *cpuacct* hierarchy the group is explicitly mentioned in the configuration file. In the *cpu* hierarchy the group is created implicitly when *www* is created there. These two groups have nothing in common, for example they do not share processes and subgroups. Groups *www* and *ftp* are created only in the *cpu* hierarchy and are not controlled by the *cpuacct* subsystem.

Example 6

The configuration file:

```
	mount {
		cpu = /mnt/cgroups/cpu;
		cpuacct = /mnt/cgroups/cpu;
	}

	group . {
		perm {
			task {
				uid = root;
				gid = operator;
			}
			admin {
				uid = root;
				gid = operator;
			}
		}
		cpu {
		}
```

```
}

group daemons {
        perm {
                task {
                        uid = root;
                        gid = daemonmaster;
                }
                admin {
                        uid = root;
                        gid = operator;
                }
        }
        cpu {
        }
}
```

creates the hierarchy controlled by two subsystems with one group having some special permissions. It corresponds to the following operations:

```
mkdir /mnt/cgroups/cpu
mount -t cgroup -o cpu,cpuacct cpu /mnt/cgroups/cpu

chown root:operator /mnt/cgroups/cpu/*
chown root:operator /mnt/cgroups/cpu/tasks

mkdir /mnt/cgroups/cpu/daemons
chown root:operator /mnt/cgroups/cpu/daemons/*
chown root:daemonmaster /mnt/cgroups/cpu/daemons/tasks
```

Users which are members of the *operator* group are allowed to administer the control groups, i.e. create new control groups and move processes between these groups without having root privileges.

Members of the *daemonmaster* group can move processes to the *daemons* control group, but they can not move the process out of the group. Only the *operator* or root can do that.

Example 7

The configuration file:

```
mount {
        cpu = /mnt/cgroups/cpu;
        cpuacct = /mnt/cgroups/cpuacct;
}

group students {
        cpuacct{
        }
        cpu {
        }
}

template students/%u {
        cpuacct{
        }
        cpu {
        }
}

mkdir /mnt/cgroups/cpu/daemons
mkdir /mnt/cgroups/cpuacct/daemons
```

The situation is the similar as in Example 4. The only difference is template, which is used if some rule uses "/students/%u" as a destination.

RECOMMENDATIONS

Keep hierarchies separated

Having multiple hierarchies is perfectly valid and can be useful in various scenarios. To keeps things clean, do not create one group in multiple hierarchies. Examples 4 and 5 show how unreadable and confusing it can be, especially when reading somebody elses configuration file.

Explicit is better than implicit

libcgroup can implicitly create groups which are needed for the creation of configured subgroups. This may be useful and save some typing in simple scenarios. When it comes to multiple hierarchies, it’s better to explicitly specify all groups and all controllers related to them.

FILES

/etc/cgconfig.conf
default libcgroup configuration file

/etc/cgconfig.d/
default libcgroup configuration files directory

SEE ALSO

cgconfigparser (8)

BUGS

Parameter values must be single strings without spaces. Parsing of quoted strings is not implemented.