NAME

systemd.resource-control - Resource control unit settings

SYNOPSIS

slice.slice, *scope*.scope, *service*.service, *socket*.socket, *mount*.mount, *swap*.swap

DESCRIPTION

Unit configuration files for services, slices, scopes, sockets, mount points, and swap devices share a subset of configuration options for resource control of spawned processes. Internally, this relies on the Linux Control Groups (cgroups) kernel concept for organizing processes in a hierarchical tree of named groups for the purpose of resource management.

This man page lists the configuration options shared by those six unit types. See **systemd.unit(5)** for the common options of all unit configuration files, and **systemd.slice(5)**, **systemd.scope(5)**, **systemd.service(5)**, **systemd.socket(5)**, **systemd.mount(5)**, and **systemd.swap(5)** for more information on the specific unit configuration files. The resource control configuration options are configured in the [Slice], [Scope], [Service], [Socket], [Mount], or [Swap] sections, depending on the unit type.

In addition, options which control resources available to programs *executed* by systemd are listed in **systemd.exec(5)**. Those options complement options listed here.

See the **New Control Group Interfaces**[1] for an introduction on how to make use of resource control APIs from programs.

AUTOMATIC DEPENDENCIES

Units with the *Slice=* setting set automatically acquire *Requires=* and *After=* dependencies on the specified slice unit.

UNIFIED AND LEGACY CONTROL GROUP HIERARCHIES

The unified control group hierarchy is the new version of kernel control group interface, see **cgroup-v2.txt**[2]. Depending on the resource type, there are differences in resource control capabilities. Also, because of interface changes, some resource types have separate set of options on the unified hierarchy.

CPU

Due to the lack of consensus in the kernel community, the CPU controller support on the unified control group hierarchy requires out-of-tree kernel patches. See **cgroup-v2-cpu.txt**[3].

CPUWeight= and *StartupCPUWeight=* replace *CPUShares=* and *StartupCPUShares=*, respectively.

The "cpuacct" controller does not exist separately on the unified hierarchy.

Memory

MemoryMax= replaces *MemoryLimit=*. *MemoryLow=* and *MemoryHigh=* are effective only on unified hierarchy.

IO

IO prefixed settings are superset of and replace *BlockIO* prefixed ones. On unified hierarchy, IO resource control also applies to buffered writes.

To ease the transition, there is best-effort translation between the two versions of settings. For each controller, if any of the settings for the unified hierarchy are present, all settings for the legacy hierarchy are ignored. If the resulting settings are for the other type of hierarchy, the configurations are translated before application.

Legacy control group hierarchy (see **cgroups.txt**[4]), also called cgroup-v1, doesn't allow safe delegation of controllers to unprivileged processes. If the system uses the legacy control group hierarchy, resource control is disabled for systemd user instance, see **systemd(1)**.

OPTIONS

Units of the types listed above can have settings for resource control configuration:

CPUAccounting=

Turn on CPU usage accounting for this unit. Takes a boolean argument. Note that turning on CPU accounting for one unit will also implicitly turn it on for all units contained in the same slice and for all its parent slices and the units contained therein. The system default for this setting may be controlled with *DefaultCPUAccounting=* in **systemd-system.conf(5)**.

CPUWeight=weight, *StartupCPUWeight=weight*

Assign the specified CPU time weight to the processes executed, if the unified control group hierarchy is used on the system. These options take an integer value and control the "cpu.weight" control group attribute. The allowed range is 1 to 10000. Defaults to 100. For details about this control group attribute, see **cgroup-v2.txt**[2] and **sched-design-CFS.txt**[5]. The available CPU time is split up among all units within one slice relative to their CPU time weight.

While *StartupCPUWeight=* only applies to the startup phase of the system, *CPUWeight=* applies to normal runtime of the system, and if the former is not set also to the startup phase. Using *StartupCPUWeight=* allows prioritizing specific services at boot-up differently than during normal runtime.

Implies "CPUAccounting=true".

These settings replace *CPUShares=* and *StartupCPUShares=*.

CPUQuota=

Assign the specified CPU time quota to the processes executed. Takes a percentage value, suffixed with "%". The percentage specifies how much CPU time the unit shall get at maximum, relative to the total CPU time available on one CPU. Use values > 100% for allotting CPU time on more than one CPU. This controls the "cpu.max" attribute on the unified control group hierarchy and "cpu.cfs_quota_us" on legacy. For details about these control group attributes, see **cgroup-v2.txt**[2] and **sched-design-CFS.txt**[5].

Example: *CPUQuota=20%* ensures that the executed processes will never get more than 20% CPU time on one CPU.

Implies "CPUAccounting=true".

MemoryAccounting=

Turn on process and kernel memory accounting for this unit. Takes a boolean argument. Note that turning on memory accounting for one unit will also implicitly turn it on for all units contained in the same slice and for all its parent slices and the units contained therein. The system default for this setting may be controlled with *DefaultMemoryAccounting=* in **systemd-system.conf(5)**.

MemoryLow=bytes

Specify the best-effort memory usage protection of the executed processes in this unit. If the memory usages of this unit and all its ancestors are below their low boundaries, this unit's memory won't be reclaimed as long as memory can be reclaimed from unprotected units.

Takes a memory size in bytes. If the value is suffixed with K, M, G or T, the specified memory size is parsed as Kilobytes, Megabytes, Gigabytes, or Terabytes (with the base 1024), respectively. Alternatively, a percentage value may be specified, which is taken relative to the installed physical memory on the system. This controls the "memory.low" control group attribute. For details about this control group attribute, see **cgroup-v2.txt**[2].

Implies "MemoryAccounting=true".

This setting is supported only if the unified control group hierarchy is used and disables *MemoryLimit=*.

MemoryHigh=bytes

Specify the high limit on memory usage of the executed processes in this unit. Memory usage may go above the limit if unavoidable, but the processes are heavily slowed down and memory is taken away aggressively in such cases. This is the main mechanism to control memory usage of a unit.

Takes a memory size in bytes. If the value is suffixed with K, M, G or T, the specified memory size is parsed as Kilobytes, Megabytes, Gigabytes, or Terabytes (with the base 1024), respectively.

Alternatively, a percentage value may be specified, which is taken relative to the installed physical memory on the system. If assigned the special value "infinity", no memory limit is applied. This controls the "memory.high" control group attribute. For details about this control group attribute, see **cgroup-v2.txt**[2].

Implies "MemoryAccounting=true".

This setting is supported only if the unified control group hierarchy is used and disables *MemoryLimit=*.

MemoryMax=bytes

Specify the absolute limit on memory usage of the executed processes in this unit. If memory usage cannot be contained under the limit, out-of-memory killer is invoked inside the unit. It is recommended to use *MemoryHigh=* as the main control mechanism and use *MemoryMax=* as the last line of defense.

Takes a memory size in bytes. If the value is suffixed with K, M, G or T, the specified memory size is parsed as Kilobytes, Megabytes, Gigabytes, or Terabytes (with the base 1024), respectively. Alternatively, a percentage value may be specified, which is taken relative to the installed physical memory on the system. If assigned the special value "infinity", no memory limit is applied. This controls the "memory.max" control group attribute. For details about this control group attribute, see **cgroup-v2.txt**[2].

Implies "MemoryAccounting=true".

This setting replaces *MemoryLimit=*.

MemorySwapMax=bytes

Specify the absolute limit on swap usage of the executed processes in this unit.

Takes a swap size in bytes. If the value is suffixed with K, M, G or T, the specified swap size is parsed as Kilobytes, Megabytes, Gigabytes, or Terabytes (with the base 1024), respectively. If assigned the special value "infinity", no swap limit is applied. This controls the "memory.swap.max" control group attribute. For details about this control group attribute, see **cgroup-v2.txt**[2].

Implies "MemoryAccounting=true".

This setting is supported only if the unified control group hierarchy is used and disables *MemoryLimit=*.

TasksAccounting=

Turn on task accounting for this unit. Takes a boolean argument. If enabled, the system manager will keep track of the number of tasks in the unit. The number of tasks accounted this way includes both kernel threads and userspace processes, with each thread counting individually. Note that turning on tasks accounting for one unit will also implicitly turn it on for all units contained in the same slice and for all its parent slices and the units contained therein. The system default for this setting may be controlled with *DefaultTasksAccounting=* in **systemd-system.conf(5)**.

TasksMax=N

Specify the maximum number of tasks that may be created in the unit. This ensures that the number of tasks accounted for the unit (see above) stays below a specific limit. This either takes an absolute number of tasks or a percentage value that is taken relative to the configured maximum number of tasks on the system. If assigned the special value "infinity", no tasks limit is applied. This controls the "pids.max" control group attribute. For details about this control group attribute, see **pids.txt**[6].

Implies "TasksAccounting=true". The system default for this setting may be controlled with *DefaultTasksMax=* in **systemd-system.conf(5)**.

IOAccounting=

Turn on Block I/O accounting for this unit, if the unified control group hierarchy is used on the system. Takes a boolean argument. Note that turning on block I/O accounting for one unit will also implicitly turn it on for all units contained in the same slice and all for its parent slices and the units contained therein. The system default for this setting may be controlled with *DefaultIOAccounting=* in **systemd-system.conf(5)**.

This setting replaces *BlockIOAccounting=* and disables settings prefixed with *BlockIO* or *StartupBlockIO*.

IOWeight=weight, *StartupIOWeight=weight*

Set the default overall block I/O weight for the executed processes, if the unified control group hierarchy is used on the system. Takes a single weight value (between 1 and 10000) to set the default block I/O weight. This controls the "io.weight" control group attribute, which defaults to 100. For details about this control group attribute, see **cgroup-v2.txt**[2]. The available I/O bandwidth is split up among all units within one slice relative to their block I/O weight.

While *StartupIOWeight=* only applies to the startup phase of the system, *IOWeight=* applies to the later runtime of the system, and if the former is not set also to the startup phase. This allows prioritizing specific services at boot-up differently than during runtime.

Implies "IOAccounting=true".

These settings replace *BlockIOWeight=* and *StartupBlockIOWeight=* and disable settings prefixed with *BlockIO* or *StartupBlockIO*.

IODeviceWeight=device weight

Set the per-device overall block I/O weight for the executed processes, if the unified control group hierarchy is used on the system. Takes a space-separated pair of a file path and a weight value to specify the device specific weight value, between 1 and 10000. (Example: "/dev/sda 1000"). The file path may be specified as path to a block device node or as any other file, in which case the backing block device of the file system of the file is determined. This controls the "io.weight" control group attribute, which defaults to 100. Use this option multiple times to set weights for multiple devices. For details about this control group attribute, see **cgroup-v2.txt**[2].

Implies "IOAccounting=true".

This setting replaces *BlockIODeviceWeight=* and disables settings prefixed with *BlockIO* or *StartupBlockIO*.

IOReadBandwidthMax=device bytes, *IOWriteBandwidthMax=device bytes*

Set the per-device overall block I/O bandwidth maximum limit for the executed processes, if the unified control group hierarchy is used on the system. This limit is not work-conserving and the executed processes are not allowed to use more even if the device has idle capacity. Takes a space-separated pair of a file path and a bandwidth value (in bytes per second) to specify the device specific bandwidth. The file path may be a path to a block device node, or as any other file in which case the backing block device of the file system of the file is used. If the bandwidth is suffixed with K, M, G, or T, the specified bandwidth is parsed as Kilobytes, Megabytes, Gigabytes, or Terabytes, respectively, to the base of 1000. (Example: "/dev/disk/by-path/pci-0000:00:1f.2-scsi-0:0:0:0 5M"). This controls the "io.max" control group attributes. Use this option multiple times to set bandwidth limits for multiple devices. For details about this control group attribute, see **cgroup-v2.txt**[2].

Implies "IOAccounting=true".

These settings replace *BlockIOReadBandwidth=* and *BlockIOWriteBandwidth=* and disable settings prefixed with *BlockIO* or *StartupBlockIO*.

IOReadIOPSMax=device IOPS, *IOWriteIOPSMax=device IOPS*

Set the per-device overall block I/O IOs-Per-Second maximum limit for the executed processes, if the unified control group hierarchy is used on the system. This limit is not work-conserving and the executed processes are not allowed to use more even if the device has idle capacity. Takes a space-separated pair of a file path and an IOPS value to specify the device specific IOPS. The file path may be a path to a block device node, or as any other file in which case the backing block device of the file system of the file is used. If the IOPS is suffixed with K, M, G, or T, the specified IOPS is parsed as KiloIOPS, MegaIOPS, GigaIOPS, or TeraIOPS, respectively, to the base of 1000. (Example:

"/dev/disk/by-path/pci-0000:00:1f.2-scsi-0:0:0:0 1K"). This controls the "io.max" control group attributes. Use this option multiple times to set IOPS limits for multiple devices. For details about this control group attribute, see **cgroup-v2.txt**[2].

Implies "IOAccounting=true".

These settings are supported only if the unified control group hierarchy is used and disable settings prefixed with *BlockIO* or *StartupBlockIO*.

DeviceAllow=

Control access to specific device nodes by the executed processes. Takes two space-separated strings: a device node specifier followed by a combination of **r**, **w**, **m** to control *r*eading, *w*riting, or creation of the specific device node(s) by the unit (*m*knod), respectively. This controls the "devices.allow" and "devices.deny" control group attributes. For details about these control group attributes, see **devices.txt**[7].

The device node specifier is either a path to a device node in the file system, starting with /dev/, or a string starting with either "char-" or "block-" followed by a device group name, as listed in /proc/devices. The latter is useful to whitelist all current and future devices belonging to a specific device group at once. The device group is matched according to file name globbing rules, you may hence use the "*" and "?" wildcards. Examples: /dev/sda5 is a path to a device node, referring to an ATA or SCSI block device. "char-pts" and "char-alsa" are specifiers for all pseudo TTYs and all ALSA sound devices, respectively. "char-cpu/*" is a specifier matching all CPU related device groups.

DevicePolicy=auto|closed|strict

Control the policy for allowing device access:

strict

means to only allow types of access that are explicitly specified.

closed

in addition, allows access to standard pseudo devices including /dev/null, /dev/zero, /dev/full, /dev/random, and /dev/urandom.

auto

in addition, allows access to all devices if no explicit *DeviceAllow=* is present. This is the default.

Slice=

The name of the slice unit to place the unit in. Defaults to system.slice for all non-instantiated units of all unit types (except for slice units themselves see below). Instance units are by default placed in a subslice of system.slice that is named after the template name.

This option may be used to arrange systemd units in a hierarchy of slices each of which might have resource settings applied.

For units of type slice, the only accepted value for this setting is the parent slice. Since the name of a slice unit implies the parent slice, it is hence redundant to ever set this parameter directly for slice units.

Special care should be taken when relying on the default slice assignment in templated service units that have *DefaultDependencies=no* set, see **systemd.service(5)**, section "Automatic Dependencies" for details.

Delegate=

Turns on delegation of further resource control partitioning to processes of the unit. For unprivileged services (i.e. those using the *User=* setting), this allows processes to create a subhierarchy beneath its control group path. For privileged services and scopes, this ensures the processes will have all control group controllers enabled.

DEPRECATED OPTIONS

The following options are deprecated. Use the indicated superseding options instead:

CPUShares=weight, *StartupCPUShares=weight*

Assign the specified CPU time share weight to the processes executed. These options take an integer value and control the "cpu.shares" control group attribute. The allowed range is 2 to 262144. Defaults to 1024. For details about this control group attribute, see **sched-design-CFS.txt**[5]. The available CPU time is split up among all units within one slice relative to their CPU time share weight.

While *StartupCPUShares=* only applies to the startup phase of the system, *CPUShares=* applies to normal runtime of the system, and if the former is not set also to the startup phase. Using *StartupCPUShares=* allows prioritizing specific services at boot-up differently than during normal runtime.

Implies "CPUAccounting=true".

These settings are deprecated. Use *CPUWeight=* and *StartupCPUWeight=* instead.

MemoryLimit=bytes

Specify the limit on maximum memory usage of the executed processes. The limit specifies how much process and kernel memory can be used by tasks in this unit. Takes a memory size in bytes. If the value is suffixed with K, M, G or T, the specified memory size is parsed as Kilobytes, Megabytes, Gigabytes, or Terabytes (with the base 1024), respectively. Alternatively, a percentage value may be specified, which is taken relative to the installed physical memory on the system. If assigned the special value "infinity", no memory limit is applied. This controls the "memory.limit_in_bytes" control group attribute. For details about this control group attribute, see **memory.txt**[8].

Implies "MemoryAccounting=true".

This setting is deprecated. Use *MemoryMax=* instead.

BlockIOAccounting=

Turn on Block I/O accounting for this unit, if the legacy control group hierarchy is used on the system. Takes a boolean argument. Note that turning on block I/O accounting for one unit will also implicitly turn it on for all units contained in the same slice and all for its parent slices and the units contained therein. The system default for this setting may be controlled with *DefaultBlockIOAccounting=* in **systemd-system.conf(5)**.

This setting is deprecated. Use *IOAccounting=* instead.

BlockIOWeight=weight, *StartupBlockIOWeight=weight*

Set the default overall block I/O weight for the executed processes, if the legacy control group hierarchy is used on the system. Takes a single weight value (between 10 and 1000) to set the default block I/O weight. This controls the "blkio.weight" control group attribute, which defaults to 500. For details about this control group attribute, see **blkio-controller.txt**[9]. The available I/O bandwidth is split up among all units within one slice relative to their block I/O weight.

While *StartupBlockIOWeight=* only applies to the startup phase of the system, *BlockIOWeight=* applies to the later runtime of the system, and if the former is not set also to the startup phase. This allows prioritizing specific services at boot-up differently than during runtime.

Implies "BlockIOAccounting=true".

These settings are deprecated. Use *IOWeight=* and *StartupIOWeight=* instead.

BlockIODeviceWeight=device weight

Set the per-device overall block I/O weight for the executed processes, if the legacy control group hierarchy is used on the system. Takes a space-separated pair of a file path and a weight value to specify the device specific weight value, between 10 and 1000. (Example: "/dev/sda 500"). The file path may be specified as path to a block device node or as any other file, in which case the backing block device of the file system of the file is determined. This controls the "blkio.weight_device" control group attribute, which defaults to 1000. Use this option multiple times to set weights for multiple devices. For details about this control group attribute, see **blkio-controller.txt**[9].

Implies "BlockIOAccounting=true".

This setting is deprecated. Use *IODeviceWeight=* instead.

BlockIOReadBandwidth=device bytes, *BlockIOWriteBandwidth=device bytes*

Set the per-device overall block I/O bandwidth limit for the executed processes, if the legacy control group hierarchy is used on the system. Takes a space-separated pair of a file path and a bandwidth value (in bytes per second) to specify the device specific bandwidth. The file path may be a path to a block device node, or as any other file in which case the backing block device of the file system of the file is used. If the bandwidth is suffixed with K, M, G, or T, the specified bandwidth is parsed as Kilobytes, Megabytes, Gigabytes, or Terabytes, respectively, to the base of 1000. (Example: "/dev/disk/by-path/pci-0000:00:1f.2-scsi-0:0:0:0 5M"). This controls the "blkio.throttle.read_bps_device" and "blkio.throttle.write_bps_device" control group attributes. Use this option multiple times to set bandwidth limits for multiple devices. For details about these control group attributes, see **blkio-controller.txt**[9].

Implies "BlockIOAccounting=true".

These settings are deprecated. Use *IOReadBandwidthMax=* and *IOWriteBandwidthMax=* instead.

SEE ALSO

systemd(1), **systemd.unit(5)**, **systemd.service(5)**, **systemd.slice(5)**, **systemd.scope(5)**, **systemd.socket(5)**, **systemd.mount(5)**, **systemd.swap(5)**, **systemd.exec(5)**, **systemd.directives(7)**, **systemd.special(7)**, The documentation for control groups and specific controllers in the Linux kernel: **cgroups.txt**[4], **cpuacct.txt**[10], **memory.txt**[8], **blkio-controller.txt**[9].

NOTES

1. New Control Group Interfaces
   http://www.freedesktop.org/wiki/Software/systemd/ControlGroupInterface/

2. cgroup-v2.txt
   https://www.kernel.org/doc/Documentation/cgroup-v2.txt

3. cgroup-v2-cpu.txt
   https://git.kernel.org/cgit/linux/kernel/git/tj/cgroup.git/tree/Documentation/cgroup-v2-cpu.txt?h=cgroup-v2-cpu

4. cgroups.txt
   https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt

5. sched-design-CFS.txt
   https://www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt

6. pids.txt
   https://www.kernel.org/doc/Documentation/cgroup-v1/pids.txt

7. devices.txt
   https://www.kernel.org/doc/Documentation/cgroup-v1/devices.txt

8. memory.txt
   https://www.kernel.org/doc/Documentation/cgroup-v1/memory.txt

9. blkio-controller.txt
   https://www.kernel.org/doc/Documentation/cgroup-v1/blkio-controller.txt

10. cpuacct.txt
    https://www.kernel.org/doc/Documentation/cgroup-v1/cpuacct.txt