```
**********************************************************
   21225 Sat Jul 13 17:57:07 2013
new/usr/src/man/man5/resource_controls.5
3830 SIGQUEUE_MAX's limit of 32 is too low
**********************************************************
'\" te
.\" Copyright (c) 2007, Sun Microsystems, Inc. All Rights Reserved.
.\" The contents of this file are subject to the terms of the Common Development
.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
.\" When distributing Covered Code, include this CDDL HEADER in each file and in
.TH RESOURCE_CONTROLS 5 "Jul 2, 2013"
.TH RESOURCE_CONTROLS 5 "Jul 2, 2007"
.SH NAME
resource_controls \- resource controls available through project database
.SH DESCRIPTION
.sp
.LP
The resource controls facility is configured through the project database. See
\fBproject\fR(4). You can set and modify resource controls through the
following utilities:
.RS +4
.TP
.ie t \(bu
.el o
\fBprctl\fR(1)
.RE
.RS +4
.TP
.ie t \(bu
.el o
\fBprojadd\fR(1M)
.RE
.RS +4
.TP
.ie t \(bu
.el o
\fBprojmod\fR(1M)
.RE
.RS +4
.TP
.ie t \(bu
.el o
\fBrctladm\fR(1M)
.RE
.sp
.LP
In a program, you use \fBsetrctl\fR(2) to set resource control values.
.sp
.LP
In addition to the preceding resource controls, there are resource pools,
accessible through the \fBpooladm\fR(1M) and \fBpoolcfg\fR(1M) utilities. In a
program, resource pools can be manipulated through the \fBlibpool\fR(3LIB)
library.
.sp
.LP
The following are the resource controls are available:
.sp
.ne 2
.na
\fB\fBprocess.max-address-space\fR\fR
.ad
.sp .6
.RS 4n
Maximum amount of address space, as summed over segment sizes, that is
available to this process, expressed as a number of bytes.
.RE
```



```
.sp
.ne 2
.na
\fB\fBprocess.max-core-size\fR\fR
.ad
.sp .6
.RS 4n
Maximum size of a core file created by this process, expressed as a number of
bytes.
.RE

.sp
.ne 2
.na
\fB\fBprocess.max-cpu-time\fR\fR
.ad
.sp .6
.RS 4n
Maximum CPU time that is available to this process, expressed as a number of
seconds.
.RE

.sp
.ne 2
.na
\fB\fBprocess.max-data-size\fR\fR
.ad
.sp .6
.RS 4n
Maximum heap memory available to this process, expressed as a number of bytes.
.RE

.sp
.ne 2
.na
\fB\fBprocess.max-file-descriptor\fR\fR
.ad
.sp .6
.RS 4n
Maximum file descriptor index available to this process, expressed as an
integer.
.RE

.sp
.ne 2
.na
\fB\fBprocess.max-file-size\fR\fR
.ad
.sp .6
.RS 4n
Maximum file offset available for writing by this process, expressed as a
number of bytes.
.RE

.sp
.ne 2
.na
\fB\fBprocess.max-msg-messages\fR\fR
.ad
.sp .6
.RS 4n
Maximum number of messages on a message queue (value copied from the resource
control at \fBmsgget()\fR time), expressed as an integer.
.RE
```

```
.sp
.ne 2
.na
\fB\fBprocess.max-msg-qbytes\fR\fR
.ad
.sp .6
.RS 4n
Maximum number of bytes of messages on a message queue (value copied from the
resource control at \fBmsgget()\fR time), expressed as a number of bytes.
.RE

.sp
.ne 2
.na
\fB\fBprocess.max-port-events\fR\fR
.ad
.sp .6
.RS 4n
Maximum allowable number of events per event port, expressed as an integer.
.RE

.sp
.ne 2
.na
\fB\fBprocess.max-sem-nsems\fR\fR
.ad
.sp .6
.RS 4n
Maximum number of semaphores allowed per semaphore set, expressed as an
integer.
.RE

.sp
.ne 2
.na
\fB\fBprocess.max-sem-ops\fR\fR
.ad
.sp .6
.RS 4n
Maximum number of semaphore operations allowed per \fBsemop\fR call (value
copied from the resource control at \fBsemget()\fR time). Expressed as an
integer, specifying the number of operations.
.RE

.sp
.ne 2
.na
\fB\fBprocess.max-sigqueue-size\fR\fR
.ad
.sp .6
.RS 4n
Maximum number of outstanding queued signals.
.RE

.sp
.ne 2
.na
#endif /* ! codereview */
\fB\fBprocess.max-stack-size\fR\fR
.ad
.sp .6
.RS 4n
Maximum stack memory segment available to this process, expressed as a number
of bytes.
.RE
```



```
.sp
.ne 2
.na
\fB\fBproject.cpu-caps\fR\fR
.ad
.sp .6
.RS 4n
Maximum amount of CPU resources that a project can use. The unit used is the
percentage of a single CPU that can be used by all user threads in a project.
Expressed as an integer. The cap does not apply to threads running in real-time
scheduling class. This resource control does not support the \fBsyslog\fR
action.
.RE

.sp
.ne 2
.na
\fB\fBproject.cpu-shares\fR\fR
.ad
.sp .6
.RS 4n
Number of CPU shares granted to a project for use with the fair share scheduler
(see \fBFSS\fR(7)). The unit used is the number of shares (an integer). This
resource control does not support the \fBsyslog\fR action.
.RE

.sp
.ne 2
.na
\fB\fBproject.max-contracts\fR\fR
.ad
.sp .6
.RS 4n
Maximum number of contracts allowed in a project, expressed as an integer.
.RE

.sp
.ne 2
.na
\fB\fBproject.max-crypto-memory\fR\fR
.ad
.sp .6
.RS 4n
Maximum amount of kernel memory that can be used for crypto operations.
Allocations in the kernel for buffers and session-related structures are
charged against this resource control.
.RE

.sp
.ne 2
.na
\fB\fBproject.max-locked-memory\fR\fR
.ad
.sp .6
.RS 4n
Total amount of physical memory locked by device drivers and user processes
(including D/ISM), expressed as a number of bytes.
.RE

.sp
.ne 2
.na
\fB\fBproject.max-lwps\fR\fR
.ad
.sp .6
.RS 4n
```

```
Maximum number of LWPs simultaneously available to a project, expressed as an
integer.
.RE

.sp
.ne 2
.na
\fB\fBproject.max-msg-ids\fR\fR
.ad
.sp .6
.RS 4n
Maximum number of message queue IDs allowed for a project, expressed as an
integer.
.RE

.sp
.ne 2
.na
\fB\fBproject.max-port-ids\fR\fR
.ad
.sp .6
.RS 4n
Maximum allowable number of event ports, expressed as an integer.
.RE

.sp
.ne 2
.na
\fB\fBproject.max-sem-ids\fR\fR
.ad
.sp .6
.RS 4n
Maximum number of semaphore IDs allowed for a project, expressed as an integer.
.RE

.sp
.ne 2
.na
\fB\fBproject.max-shm-ids\fR\fR
.ad
.sp .6
.RS 4n
Maximum number of shared memory IDs allowed for a project, expressed as an
integer.
.RE

.sp
.ne 2
.na
\fB\fBproject.max-shm-memory\fR\fR
.ad
.sp .6
.RS 4n
Total amount of shared memory allowed for a project, expressed as a number of
bytes.
.RE

.sp
.ne 2
.na
\fB\fBproject.max-tasks\fR\fR
.ad
.sp .6
.RS 4n
Maximum number of tasks allowable in a project, expressed as an integer.
.RE
```



```
.sp
.ne 2
.na
\fB\fBproject.pool\fR\fR
.ad
.sp .6
.RS 4n
Binds a specified resource pool with a project.
.RE

.sp
.ne 2
.na
\fB\fBrcap.max-rss\fR\fR
.ad
.sp .6
.RS 4n
The total amount of physical memory, in bytes, that is available to processes
in a project.
.RE

.sp
.ne 2
.na
\fB\fBtask.max-cpu-time\fR\fR
.ad
.sp .6
.RS 4n
Maximum CPU time that is available to this task's processes, expressed as a
number of seconds.
.RE

.sp
.ne 2
.na
\fB\fBtask.max-lwps\fR\fR
.ad
.sp .6
.RS 4n
Maximum number of LWPs simultaneously available to this task's processes,
expressed as an integer.
.RE

.sp
.LP
The following zone-wide resource controls are available:
.sp
.ne 2
.na
\fB\fBzone.cpu-cap\fR\fR
.ad
.sp .6
.RS 4n
Sets a limit on the amount of CPU time that can be used by a zone. The unit
used is the percentage of a single CPU that can be used by all user threads in
a zone. Expressed as an integer. When projects within the capped zone have
their own caps, the minimum value takes precedence. This resource control does
not support the \fBsyslog\fR action.
.RE

.sp
.ne 2
.na
\fB\fBzone.cpu-shares\fR\fR
.ad
```

```
.sp .6
.RS 4n
Sets a limit on the number of fair share scheduler (FSS) CPU shares for a zone.
CPU shares are first allocated to the zone, and then further subdivided among
projects within the zone as specified in the \fBproject.cpu-shares\fR entries.
Expressed as an integer. This resource control does not support the
\fBsyslog\fR action.
.RE

.sp
.ne 2
.na
\fB\fBzone.max-locked-memory\fR\fR
.ad
.sp .6
.RS 4n
Total amount of physical locked memory available to a zone.
.RE

.sp
.ne 2
.na
\fB\fBzone.max-lwps\fR\fR
.ad
.sp .6
.RS 4n
Enhances resource isolation by preventing too many LWPs in one zone from
affecting other zones. A zone's total LWPs can be further subdivided among
projects within the zone within the zone by using \fBproject.max-lwps\fR
entries. Expressed as an integer.
.RE

.sp
.ne 2
.na
\fB\fBzone.max-msg-ids\fR\fR
.ad
.sp .6
.RS 4n
Maximum number of message queue IDs allowed for a zone, expressed as an
integer.
.RE

.sp
.ne 2
.na
\fB\fBzone.max-sem-ids\fR\fR
.ad
.sp .6
.RS 4n
Maximum number of semaphore IDs allowed for a zone, expressed as an integer.
.RE

.sp
.ne 2
.na
\fB\fBzone.max-shm-ids\fR\fR
.ad
.sp .6
.RS 4n
Maximum number of shared memory IDs allowed for a zone, expressed as an
integer.
.RE

.sp
.ne 2
```



```
.na
\fB\fBzone.max-shm-memory\fR\fR
.ad
.sp .6
.RS 4n
Total amount of shared memory allowed for a zone, expressed as a number of
bytes.
.RE

.sp
.ne 2
.na
\fB\fBzone.max-swap\fR\fR
.ad
.sp .6
.RS 4n
Total amount of swap that can be consumed by user process address space
mappings and \fBtmpfs\fR mounts for this zone.
.RE

.sp
.LP
See \fBzones\fR(5).
.SS "Units Used in Resource Controls"
.sp
.LP
Resource controls can be expressed as in units of size (bytes), time (seconds),
or as a count (integer). These units use the strings specified below.
.sp
.in +2
.nf
Category             Res Ctrl      Modifier  Scale
                     Type String
-----------          -----------   --------  -----
Size                 bytes         B         1
                                   KB        2^10
                                   MB        2^20
                                   GB        2^30
                                   TB        2^40
                                   PB        2^50
                                   EB        2^60

Time                 seconds       s         1
                                   Ks        10^3
                                   Ms        10^6
                                   Gs        10^9
                                   Ts        10^12
                                   Ps        10^15
                                   Es        10^18

Count                integer       none      1
                                   K         10^3
                                   M         10^6
                                   G         10^9
                                   T         10^12
                                   P         10^15
                                   Es        10^18
.fi
.in -2

.sp
.LP
Scaled values can be used with resource controls. The following example shows a
scaled threshold value:
.sp
.in +2
```

```
.nf
task.max-lwps=(priv,1K,deny)
.fi
.in -2

.sp
.LP
In the \fBproject\fR file, the value \fB1K\fR is expanded to \fB1000\fR:
.sp
.in +2
.nf
task.max-lwps=(priv,1000,deny)
.fi
.in -2

.sp
.LP
A second example uses a larger scaled value:
.sp
.in +2
.nf
process.max-file-size=(priv,5G,deny)
.fi
.in -2

.sp
.LP
In the \fBproject\fR file, the value \fB5G\fR is expanded to \fB5368709120\fR:
.sp
.in +2
.nf
process.max-file-size=(priv,5368709120,deny)
.fi
.in -2

.sp
.LP
The preceding examples use the scaling factors specified in the table above.
.sp
.LP
Note that unit modifiers (for example, \fB5G\fR) are accepted by the
\fBprctl\fR(1), \fBprojadd\fR(1M), and \fBprojmod\fR(1M) commands. You cannot
use unit modifiers in the project database itself.
.SS "Resource Control Values and Privilege Levels"
.sp
.LP
A threshold value on a resource control constitutes a point at which local
actions can be triggered or global actions, such as logging, can occur.
.sp
.LP
Each threshold value on a resource control must be associated with a privilege
level. The privilege level must be one of the following three types:
.sp
.ne 2
.na
\fB\fBbasic\fR\fR
.ad
.sp .6
.RS 4n
Can be modified by the owner of the calling process.
.RE

.sp
.ne 2
.na
\fB\fBprivileged\fR\fR
```



```
.ad
.sp .6
.RS 4n
Can be modified by the current process (requiring \fBsys_resource\fR privilege)
or by \fBprctl\fR(1) (requiring \fBproc_owner\fR privilege).
.RE

.sp
.ne 2
.na
\fB\fBsystem\fR\fR
.ad
.sp .6
.RS 4n
Fixed for the duration of the operating system instance.
.RE

.sp
.LP
A resource control is guaranteed to have one \fBsystem\fR value, which is
defined by the system, or resource provider. The \fBsystem\fR value represents
how much of the resource the current implementation of the operating system is
capable of providing.
.sp
.LP
Any number of privileged values can be defined, and only one basic value is
allowed. Operations that are performed without specifying a privilege value are
assigned a basic privilege by default.
.sp
.LP
The privilege level for a resource control value is defined in the privilege
field of the resource control block as \fBRCTL_BASIC\fR, \fBRCTL_PRIVILEGED\fR,
or \fBRCTL_SYSTEM\fR. See \fBsetrctl\fR(2) for more information. You can use
the \fBprctl\fR command to modify values that are associated with basic and
privileged levels.
.sp
.LP
In specifying the privilege level of \fBprivileged\fR, you can use the
abbreviation \fBpriv\fR. For example:
.sp
.in +2
.nf
task.max-lwps=(priv,1K,deny)
.fi
.in -2

.SS "Global and Local Actions on Resource Control Values"
.sp
.LP
There are two categories of actions on resource control values: global and
local.
.sp
.LP
Global actions apply to resource control values for every resource control on
the system. You can use \fBrctladm\fR(1M) to perform the following actions:
.RS +4
.TP
.ie t \(bu
.el o
Display the global state of active system resource controls.
.RE
.RS +4
.TP
.ie t \(bu
.el o
Set global logging actions.
```

```
.RE
.sp
.LP
You can disable or enable the global logging action on resource controls. You
can set the \fBsyslog\fR action to a specific degree by assigning a severity
level, \fBsyslog=\fR\fIlevel\fR. The possible settings for \fIlevel\fR are as
follows:
.RS +4
.TP
.ie t \(bu
.el o
\fBdebug\fR
.RE
.RS +4
.TP
.ie t \(bu
.el o
\fBinfo\fR
.RE
.RS +4
.TP
.ie t \(bu
.el o
\fBnotice\fR
.RE
.RS +4
.TP
.ie t \(bu
.el o
\fBwarning\fR
.RE
.RS +4
.TP
.ie t \(bu
.el o
\fBerr\fR
.RE
.RS +4
.TP
.ie t \(bu
.el o
\fBcrit\fR
.RE
.RS +4
.TP
.ie t \(bu
.el o
\fBalert\fR
.RE
.RS +4
.TP
.ie t \(bu
.el o
\fBemerg\fR
.RE
.sp
.LP
By default, there is no global logging of resource control violations.
.sp
.LP
Local actions are taken on a process that attempts to exceed the control value.
For each threshold value that is placed on a resource control, you can
associate one or more actions. There are three types of local actions:
\fBnone\fR, \fBdeny\fR, and \fBsignal=\fR. These three actions are used as
follows:
.sp
```



```
.ne 2
.na
\fB\fBnone\fR\fR
.ad
.sp .6
.RS 4n
No action is taken on resource requests for an amount that is greater than the
threshold. This action is useful for monitoring resource usage without
affecting the progress of applications. You can also enable a global message
that displays when the resource control is exceeded, while, at the same time,
the process exceeding the threshhold is not affected.
.RE

.sp
.ne 2
.na
\fB\fBdeny\fR\fR
.ad
.sp .6
.RS 4n
You can deny resource requests for an amount that is greater than the
threshold. For example, a \fBtask.max-lwps\fR resource control with action deny
causes a \fBfork()\fR system call to fail if the new process would exceed the
control value. See the \fBfork\fR(2).
.RE

.sp
.ne 2
.na
\fB\fBsignal=\fR\fR
.ad
.sp .6
.RS 4n
You can enable a global signal message action when the resource control is
exceeded. A signal is sent to the process when the threshold value is exceeded.
Additional signals are not sent if the process consumes additional resources.
Available signals are listed below.
.RE

.sp
.LP
Not all of the actions can be applied to every resource control. For example, a
process cannot exceed the number of CPU shares assigned to the project of which
it is a member. Therefore, a deny action is not allowed on the
\fBproject.cpu-shares\fR resource control.
.sp
.LP
Due to implementation restrictions, the global properties of each control can
restrict the range of available actions that can be set on the threshold value.
(See \fBrctladm\fR(1M).) A list of available signal actions is presented in the
following list. For additional information about signals, see
\fBsignal\fR(3HEAD).
.sp
.LP
The following are the signals available to resource control values:
.sp
.ne 2
.na
\fB\fBSIGABRT\fR\fR
.ad
.sp .6
.RS 4n
Terminate the process.
.RE

.sp
```

```
.ne 2
.na
\fB\fBSIGHUP\fR\fR
.ad
.sp .6
.RS 4n
Send a hangup signal. Occurs when carrier drops on an open line. Signal sent to
the process group that controls the terminal.
.RE

.sp
.ne 2
.na
\fB\fBSIGTERM\fR\fR
.ad
.sp .6
.RS 4n
Terminate the process. Termination signal sent by software.
.RE

.sp
.ne 2
.na
\fB\fBSIGKILL\fR\fR
.ad
.sp .6
.RS 4n
Terminate the process and kill the program.
.RE

.sp
.ne 2
.na
\fB\fBSIGSTOP\fR\fR
.ad
.sp .6
.RS 4n
Stop the process. Job control signal.
.RE

.sp
.ne 2
.na
\fB\fBSIGXRES\fR\fR
.ad
.sp .6
.RS 4n
Resource control limit exceeded. Generated by resource control facility.
.RE

.sp
.ne 2
.na
\fB\fBSIGXFSZ\fR\fR
.ad
.sp .6
.RS 4n
Terminate the process. File size limit exceeded. Available only to resource
controls with the \fBRCTL_GLOBAL_FILE_SIZE\fR property
(\fBprocess.max-file-size\fR). See \fBrctlblk_set_value\fR(3C).
.RE

.sp
.ne 2
.na
\fB\fBSIGXCPU\fR\fR
```



```
.ad
.sp .6
.RS 4n
Terminate the process. CPU time limit exceeded. Available only to resource
controls with the \fBRCTL_GLOBAL_CPUTIME\fR property
(\fBprocess.max-cpu-time\fR). See \fBrctlblk_set_value\fR(3C).
.RE

.SS "Resource Control Flags and Properties"
.sp
.LP
Each resource control on the system has a certain set of associated properties.
This set of properties is defined as a set of flags, which are associated with
all controlled instances of that resource. Global flags cannot be modified, but
the flags can be retrieved by using either \fBrctladm\fR(1M) or the
\fBsetrctl\fR(2) system call.
.sp
.LP
Local flags define the default behavior and configuration for a specific
threshold value of that resource control on a specific process or process
collective. The local flags for one threshold value do not affect the behavior
of other defined threshold values for the same resource control. However, the
global flags affect the behavior for every value associated with a particular
control. Local flags can be modified, within the constraints supplied by their
corresponding global flags, by the \fBprctl\fR command or the \fBsetrctl\fR
system call. See \fBsetrctl\fR(2).
.sp
.LP
For the complete list of local flags, global flags, and their definitions, see
\fBrctlblk_set_value\fR(3C).
.sp
.LP
To determine system behavior when a threshold value for a particular resource
control is reached, use \fBrctladm\fR to display the global flags for the
resource control . For example, to display the values for
\fBprocess.max-cpu-time\fR, enter:
.sp
.in +2
.nf
$ rctladm process.max-cpu-time
process.max-cpu-time  syslog=off [ lowerable no-deny cpu-time inf seconds ]
.fi
.in -2

.sp
.LP
The global flags indicate the following:
.sp
.ne 2
.na
\fB\fBlowerable\fR\fR
.ad
.sp .6
.RS 4n
Superuser privileges are not required to lower the privileged values for this
control.
.RE

.sp
.ne 2
.na
\fB\fBno-deny\fR\fR
.ad
.sp .6
.RS 4n
Even when threshold values are exceeded, access to the resource is never
```

```
denied.
.RE

.sp
.ne 2
.na
\fB\fBcpu-time\fR\fR
.ad
.sp .6
.RS 4n
\fBSIGXCPU\fR is available to be sent when threshold values of this resource
are reached.
.RE

.sp
.ne 2
.na
\fB\fBseconds\fR\fR
.ad
.sp .6
.RS 4n
The time value for the resource control.
.RE

.sp
.LP
Use the \fBprctl\fR command to display local values and actions for the
resource control. For example:
.sp
.in +2
.nf
$ prctl -n process.max-cpu-time $$
    process 353939: -ksh
    NAME    PRIVILEGE    VALUE    FLAG   ACTION              RECIPIENT
 process.max-cpu-time
         privileged   18.4Es    inf   signal=XCPU               -
         system       18.4Es    inf   none
.fi
.in -2

.sp
.LP
The \fBmax\fR (\fBRCTL_LOCAL_MAXIMAL\fR) flag is set for both threshold values,
and the \fBinf\fR (\fBRCTL_GLOBAL_INFINITE\fR) flag is defined for this
resource control. An \fBinf\fR value has an infinite quantity. The value is
never enforced. Hence, as configured, both threshold quantities represent
infinite values that are never exceeded.
.SS "Resource Control Enforcement"
.sp
.LP
More than one resource control can exist on a resource. A resource control can
exist at each containment level in the process model. If resource controls are
active on the same resource at different container levels, the smallest
container's control is enforced first. Thus, action is taken on
\fBprocess.max-cpu-time\fR before \fBtask.max-cpu-time\fR if both controls are
encountered simultaneously.
.SH ATTRIBUTES
.sp
.LP
See \fBattributes\fR(5) for a description of the following attributes:
.sp

.sp
.TS
box;
c | c
```



```
l | l .
ATTRIBUTE TYPE  ATTRIBUTE VALUE
_
Interface Stability     Evolving
.TE

.SH SEE ALSO
.sp
.LP
\fBprctl\fR(1), \fBpooladm\fR(1M), \fBpoolcfg\fR(1M), \fBprojadd\fR(1M),
\fBprojmod\fR(1M), \fBrctladm\fR(1M), \fBsetrctl\fR(2),
\fBrctlblk_set_value\fR(3C), \fBlibpool\fR(3LIB), \fBproject\fR(4),
\fBattributes\fR(5), \fBFSS\fR(7)
.sp
.LP
\fISystem Administration Guide:  Virtualization Using the Solaris Operating
System\fR
```

```
**********************************************************
   12538 Sat Jul 13 17:57:08 2013
new/usr/src/uts/common/os/rctl_proc.c
3830 SIGQUEUE_MAX's limit of 32 is too low
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/types.h>
#include <sys/cmn_err.h>
#include <sys/sysmacros.h>
#include <sys/proc.h>
#include <sys/rctl.h>
#include <sys/rctl_impl.h>
#include <sys/port_kernel.h>
#include <sys/signal.h>
#endif /* ! codereview */

#include <sys/vmparam.h>
#include <sys/machparam.h>

/*
 * Process-based resource controls
 *   The structure of the kernel leaves us no particular place where the process
 *   abstraction can be declared--it is intertwined with the growth of the Unix
 *   kernel.  Accordingly, we place all of the resource control logic associated
 *   with processes, both existing and future, in this file.
 */

rctl_hndl_t rctlproc_legacy[RLIM_NLIMITS];
uint_t rctlproc_flags[RLIM_NLIMITS] = {
        RCTL_LOCAL_SIGNAL,                      /* RLIMIT_CPU   */
        RCTL_LOCAL_DENY | RCTL_LOCAL_SIGNAL,    /* RLIMIT_FSIZE */
        RCTL_LOCAL_DENY,                        /* RLIMIT_DATA  */
        RCTL_LOCAL_DENY,                        /* RLIMIT_STACK */
        RCTL_LOCAL_DENY,                        /* RLIMIT_CORE  */
        RCTL_LOCAL_DENY,                        /* RLIMIT_NOFILE */
        RCTL_LOCAL_DENY                         /* RLIMIT_VMEM  */
};
int rctlproc_signals[RLIM_NLIMITS] = {
        SIGXCPU,                                /* RLIMIT_CPU   */
        SIGXFSZ,                                /* RLIMIT_FSIZE */
```



```
        0, 0, 0, 0, 0                           /* remainder do not signal */
};

rctl_hndl_t rc_process_msgmnb;
rctl_hndl_t rc_process_msgtql;
rctl_hndl_t rc_process_semmsl;
rctl_hndl_t rc_process_semopm;
rctl_hndl_t rc_process_portev;
rctl_hndl_t rc_process_sigqueue;
#endif /* ! codereview */

/*
 * process.max-cpu-time / RLIMIT_CPU
 */
/*ARGSUSED*/
static int
proc_cpu_time_test(struct rctl *rctl, struct proc *p, rctl_entity_p_t *e,
    rctl_val_t *rval, rctl_qty_t inc, uint_t flags)
{
        return (inc >= rval->rcv_value);
}

static rctl_ops_t proc_cpu_time_ops = {
        rcop_no_action,
        rcop_no_usage,
        rcop_no_set,
        proc_cpu_time_test
};

/*
 * process.max-file-size / RLIMIT_FSIZE
 */
static int
proc_filesize_set(rctl_t *rctl, struct proc *p, rctl_entity_p_t *e,
    rctl_qty_t nv)
{
        if (p->p_model == DATAMODEL_NATIVE)
                nv = MIN(nv, rctl->rc_dict_entry->rcd_max_native);
        else
                nv = MIN(nv, rctl->rc_dict_entry->rcd_max_ilp32);

        ASSERT(e->rcep_t == RCENTITY_PROCESS);
        e->rcep_p.proc->p_fsz_ctl = nv;

        return (0);
}

static rctl_ops_t proc_filesize_ops = {
        rcop_no_action,
        rcop_no_usage,
        proc_filesize_set,
        rcop_no_test
};

/*
 * process.max-data / RLIMIT_DATA
 */

/*
 * process.max-stack-size / RLIMIT_STACK
 */
static int
proc_stack_set(rctl_t *rctl, struct proc *p, rctl_entity_p_t *e,
    rctl_qty_t nv)
{
        klwp_t *lwp = ttolwp(curthread);
```

```
127         if (p->p_model == DATAMODEL_NATIVE)
128                 nv = MIN(nv, rctl->rc_dict_entry->rcd_max_native);
129         else
130                 nv = MIN(nv, rctl->rc_dict_entry->rcd_max_ilp32);

132         /*
133          * In the process of changing the rlimit, this function actually
134          * gets called a number of times. We only want to save the current
135          * rlimit the first time we come through here. In post_syscall(),
136          * we copyin() the lwp's ustack, and compare it to the rlimit we
137          * save here; if the two match, we adjust the ustack to reflect
138          * the new stack bounds.
139          *
140          * We check to make sure that we're changing the rlimit of our
141          * own process rather than on behalf of some other process. The
142          * notion of changing this resource limit on behalf of another
143          * process is problematic at best, and changing the amount of stack
144          * space a process is allowed to consume is a rather antiquated
145          * notion that has limited applicability in our multithreaded
146          * process model.
147          */
148         ASSERT(e->rcep_t == RCENTITY_PROCESS);
149         if (lwp != NULL && lwp->lwp_procp == e->rcep_p.proc &&
150             lwp->lwp_ustack && lwp->lwp_old_stk_ctl == 0) {
151                 lwp->lwp_old_stk_ctl = (size_t)e->rcep_p.proc->p_stk_ctl;
152                 curthread->t_post_sys = 1;
153         }

155         e->rcep_p.proc->p_stk_ctl = nv;

157         return (0);
158 }

160 static rctl_ops_t proc_stack_ops = {
161         rcop_no_action,
162         rcop_no_usage,
163         proc_stack_set,
164         rcop_no_test
165 };

167 /*
168  * process.max-file-descriptors / RLIMIT_NOFILE
169  */
170 static int
171 proc_nofile_set(rctl_t *rctl, struct proc *p, rctl_entity_p_t *e, rctl_qty_t nv)
172 {
173         ASSERT(e->rcep_t == RCENTITY_PROCESS);
174         if (p->p_model == DATAMODEL_NATIVE)
175                 nv = MIN(nv, rctl->rc_dict_entry->rcd_max_native);
176         else
177                 nv = MIN(nv, rctl->rc_dict_entry->rcd_max_ilp32);

179         e->rcep_p.proc->p_fno_ctl = nv;

181         return (0);
182 }

184 static rctl_ops_t proc_nofile_ops = {
185         rcop_no_action,
186         rcop_no_usage,
187         proc_nofile_set,
188         rcop_absolute_test
189 };

191 /*
```



```
192  * process.max-address-space / RLIMIT_VMEM
193  */
194 static int
195 proc_vmem_set(rctl_t *rctl, struct proc *p, rctl_entity_p_t *e, rctl_qty_t nv)
196 {
197         ASSERT(e->rcep_t == RCENTITY_PROCESS);
198         if (p->p_model == DATAMODEL_ILP32)
199                 nv = MIN(nv, rctl->rc_dict_entry->rcd_max_ilp32);
200         else
201                 nv = MIN(nv, rctl->rc_dict_entry->rcd_max_native);

203         e->rcep_p.proc->p_vmem_ctl = nv;

205         return (0);
206 }

208 static rctl_ops_t proc_vmem_ops = {
209         rcop_no_action,
210         rcop_no_usage,
211         proc_vmem_set,
212         rcop_no_test
213 };

215 /*
216  * void rctlproc_default_init()
217  *
218  * Overview
219  *   Establish default basic and privileged control values on the init process.
220  *   These correspond to the soft and hard limits, respectively.
221  */
222 void
223 rctlproc_default_init(struct proc *initp, rctl_alloc_gp_t *gp)
224 {
225         struct rlimit64 rlp64;

227         /*
228          * RLIMIT_CPU: deny never, sigtoproc(pp, NULL, SIGXCPU).
229          */
230         rlp64.rlim_cur = rlp64.rlim_max = RLIM64_INFINITY;
231         (void) rctl_rlimit_set(rctlproc_legacy[RLIMIT_CPU], initp, &rlp64, gp,
232             RCTL_LOCAL_SIGNAL, SIGXCPU, kcred);

234         /*
235          * RLIMIT_FSIZE: deny always, sigtoproc(pp, NULL, SIGXFSZ).
236          */
237         rlp64.rlim_cur = rlp64.rlim_max = RLIM64_INFINITY;
238         (void) rctl_rlimit_set(rctlproc_legacy[RLIMIT_FSIZE], initp, &rlp64, gp,
239             RCTL_LOCAL_SIGNAL | RCTL_LOCAL_DENY, SIGXFSZ, kcred);

241         /*
242          * RLIMIT_DATA: deny always, no default action.
243          */
244         rlp64.rlim_cur = rlp64.rlim_max = RLIM64_INFINITY;
245         (void) rctl_rlimit_set(rctlproc_legacy[RLIMIT_DATA], initp, &rlp64, gp,
246             RCTL_LOCAL_DENY, 0, kcred);

248         /*
249          * RLIMIT_STACK: deny always, no default action.
250          */
251 #ifdef __sparc
252         rlp64.rlim_cur = DFLSSIZ;
253         rlp64.rlim_max = LONG_MAX;
254 #else
255         rlp64.rlim_cur = DFLSSIZ;
256         rlp64.rlim_max = MAXSSIZ;
257 #endif
```

```
258         (void) rctl_rlimit_set(rctlproc_legacy[RLIMIT_STACK], initp, &rlp64, gp,
259             RCTL_LOCAL_DENY, 0, kcred);

261         /*
262          * RLIMIT_CORE: deny always, no default action.
263          */
264         rlp64.rlim_cur = rlp64.rlim_max = RLIM64_INFINITY;
265         (void) rctl_rlimit_set(rctlproc_legacy[RLIMIT_CORE], initp, &rlp64, gp,
266             RCTL_LOCAL_DENY, 0, kcred);

268         /*
269          * RLIMIT_NOFILE: deny always, no action.
270          */
271         rlp64.rlim_cur = rlim_fd_cur;
272         rlp64.rlim_max = rlim_fd_max;
273         (void) rctl_rlimit_set(rctlproc_legacy[RLIMIT_NOFILE], initp, &rlp64,
274             gp, RCTL_LOCAL_DENY, 0, kcred);

276         /*
277          * RLIMIT_VMEM
278          */
279         rlp64.rlim_cur = rlp64.rlim_max = RLIM64_INFINITY;
280         (void) rctl_rlimit_set(rctlproc_legacy[RLIMIT_VMEM], initp, &rlp64, gp,
281             RCTL_LOCAL_DENY, 0, kcred);
282 }

284 /*
285  * void rctlproc_init()
286  *
287  * Overview
288  *   Register the various resource controls associated with process entities.
289  *   The historical rlim_infinity_map and rlim_infinity32_map are now encoded
290  *   here as the native and ILP32 infinite values for each resource control.
291  */
292 void
293 rctlproc_init(void)
 35 rctlproc_init()
294 {
295         rctl_set_t *set;
296         rctl_alloc_gp_t *gp;
297         rctl_entity_p_t e;

299         rctlproc_legacy[RLIMIT_CPU] = rctl_register("process.max-cpu-time",
300             RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_NEVER |
301             RCTL_GLOBAL_CPU_TIME | RCTL_GLOBAL_INFINITE | RCTL_GLOBAL_SECONDS,
302             UINT64_MAX, UINT64_MAX, &proc_cpu_time_ops);
303         rctlproc_legacy[RLIMIT_FSIZE] = rctl_register("process.max-file-size",
304             RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
305             RCTL_GLOBAL_FILE_SIZE | RCTL_GLOBAL_BYTES,
306             MAXOFFSET_T, MAXOFFSET_T, &proc_filesize_ops);
307         rctlproc_legacy[RLIMIT_DATA] = rctl_register("process.max-data-size",
308             RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
309             RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
310             ULONG_MAX, UINT32_MAX, &rctl_default_ops);
311 #ifdef _LP64
312 #ifdef __sparc
313         rctlproc_legacy[RLIMIT_STACK] = rctl_register("process.max-stack-size",
314             RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
315             RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
316             LONG_MAX, INT32_MAX, &proc_stack_ops);
317 #else   /* __sparc */
318         rctlproc_legacy[RLIMIT_STACK] = rctl_register("process.max-stack-size",
319             RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
320             RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
321             MAXSSIZ, USRSTACK32 - PAGESIZE, &proc_stack_ops);
322 #endif  /* __sparc */
```



```
323 #else   /* _LP64 */
324         rctlproc_legacy[RLIMIT_STACK] = rctl_register("process.max-stack-size",
325             RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
326             RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
327             USRSTACK - PAGESIZE, USRSTACK - PAGESIZE, &proc_stack_ops);
328 #endif
329         rctlproc_legacy[RLIMIT_CORE] = rctl_register("process.max-core-size",
330             RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
331             RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
332             MIN(MAXOFFSET_T, ULONG_MAX), UINT32_MAX, &rctl_default_ops);
333         rctlproc_legacy[RLIMIT_NOFILE] = rctl_register(
334             "process.max-file-descriptor", RCENTITY_PROCESS,
335             RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
336             RCTL_GLOBAL_COUNT, INT32_MAX, INT32_MAX, &proc_nofile_ops);
337         rctlproc_legacy[RLIMIT_VMEM] =
338             rctl_register("process.max-address-space", RCENTITY_PROCESS,
339             RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
340             RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
341             ULONG_MAX, UINT32_MAX, &proc_vmem_ops);

343         rc_process_semmsl = rctl_register("process.max-sem-nsems",
344             RCENTITY_PROCESS, RCTL_GLOBAL_DENY_ALWAYS | RCTL_GLOBAL_COUNT,
345             SHRT_MAX, SHRT_MAX, &rctl_absolute_ops);
346         rctl_add_legacy_limit("process.max-sem-nsems", "semsys",
347             "seminfo_semmsl", 512, SHRT_MAX);

349         rc_process_semopm = rctl_register("process.max-sem-ops",
350             RCENTITY_PROCESS, RCTL_GLOBAL_DENY_ALWAYS | RCTL_GLOBAL_COUNT,
351             INT_MAX, INT_MAX, &rctl_absolute_ops);
352         rctl_add_legacy_limit("process.max-sem-ops", "semsys",
353             "seminfo_semopm", 512, INT_MAX);

355         rc_process_msgmnb = rctl_register("process.max-msg-qbytes",
356             RCENTITY_PROCESS, RCTL_GLOBAL_DENY_ALWAYS | RCTL_GLOBAL_BYTES,
357             ULONG_MAX, ULONG_MAX, &rctl_absolute_ops);
358         rctl_add_legacy_limit("process.max-msg-qbytes", "msgsys",
359             "msginfo_msgmnb", 65536, ULONG_MAX);

361         rc_process_msgtql = rctl_register("process.max-msg-messages",
362             RCENTITY_PROCESS, RCTL_GLOBAL_DENY_ALWAYS | RCTL_GLOBAL_COUNT,
363             UINT_MAX, UINT_MAX, &rctl_absolute_ops);
364         rctl_add_legacy_limit("process.max-msg-messages", "msgsys",
365             "msginfo_msgtql", 8192, UINT_MAX);

367         rc_process_portev = rctl_register("process.max-port-events",
368             RCENTITY_PROCESS, RCTL_GLOBAL_DENY_ALWAYS | RCTL_GLOBAL_COUNT,
369             PORT_MAX_EVENTS, PORT_MAX_EVENTS, &rctl_absolute_ops);
370         rctl_add_default_limit("process.max-port-events", PORT_DEFAULT_EVENTS,
371             RCPRIV_PRIVILEGED, RCTL_LOCAL_DENY);

373         rc_process_sigqueue = rctl_register("process.max-sigqueue-size",
374             RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
375             RCTL_GLOBAL_COUNT, _SIGQUEUE_SIZE_MAX, _SIGQUEUE_SIZE_MAX,
376             &rctl_absolute_ops);
377         rctl_add_default_limit("process.max-sigqueue-size",
378             _SIGQUEUE_SIZE_BASIC, RCPRIV_BASIC, RCTL_LOCAL_DENY);
379         rctl_add_default_limit("process.max-sigqueue-size",
380             _SIGQUEUE_SIZE_PRIVILEGED, RCPRIV_PRIVILEGED, RCTL_LOCAL_DENY);

382 #endif /* ! codereview */
383         /*
384          * Place minimal set of controls on "sched" process for inheritance by
385          * processes created via newproc().
386          */
387         set = rctl_set_create();
388         gp = rctl_set_init_prealloc(RCENTITY_PROCESS);
```

```
389         mutex_enter(&curproc->p_lock);
390         e.rcep_p.proc = curproc;
391         e.rcep_t = RCENTITY_PROCESS;
392         curproc->p_rctls = rctl_set_init(RCENTITY_PROCESS, curproc, &e,
393             set, gp);
394         mutex_exit(&curproc->p_lock);
395         rctl_prealloc_destroy(gp);
396 }
```

```
**********************************************************
   73742 Sat Jul 13 17:57:08 2013
new/usr/src/uts/common/os/sig.c
3830 SIGQUEUE_MAX's limit of 32 is too low
**********************************************************
______unchanged_portion_omitted_

#ifndef INT_MAX
#define INT_MAX 2147483647
#ifndef UCHAR_MAX
#define UCHAR_MAX       255
#endif

#define _SIGQUEUE_PREALLOC      32      /* XXX: log scale? */

#endif /* ! codereview */
/*
 * The pre-allocated pool (with _SIGQUEUE_PREALLOC entries) is
 * allocated at the first sigqueue/signotify call.
 * The entire pool (with maxcount entries) is pre-allocated at
 * the first sigqueue/signotify call.
 */
sigqhdr_t *
sigqhdralloc(size_t size, uint_t maxcount)
{
        size_t i;
        sigqueue_t *sq, *next;
        sigqhdr_t *sqh;

        i = (_SIGQUEUE_PREALLOC * size) + sizeof (sigqhdr_t);
        ASSERT(maxcount <= INT_MAX);
        i = (maxcount * size) + sizeof (sigqhdr_t);
        ASSERT(maxcount <= UCHAR_MAX && i <= USHRT_MAX);
        sqh = kmem_alloc(i, KM_SLEEP);
        sqh->sqb_count = maxcount;
        sqh->sqb_maxcount = maxcount;
        sqh->sqb_size = i;
        sqh->sqb_count = (uchar_t)maxcount;
        sqh->sqb_maxcount = (uchar_t)maxcount;
        sqh->sqb_size = (ushort_t)i;
        sqh->sqb_pexited = 0;
        sqh->sqb_sent = 0;
        sqh->sqb_free = sq = (sigqueue_t *)(sqh + 1);
        for (i = _SIGQUEUE_PREALLOC - 1; i != 0; i--) {
        for (i = maxcount - 1; i != 0; i--) {
                next = (sigqueue_t *)((uintptr_t)sq + size);
                sq->sq_next = next;
                sq = next;
        }
        sq->sq_next = NULL;
        cv_init(&sqh->sqb_cv, NULL, CV_DEFAULT, NULL);
        mutex_init(&sqh->sqb_lock, NULL, MUTEX_DEFAULT, NULL);
        return (sqh);
}

static void sigqrel(sigqueue_t *);

/*
 * Allocate a sigqueue/signotify structure from the per process
 * pre-allocated pool or allocate a new sigqueue/signotify structure
 * if the pre-allocated pool is exhausted.
 * allocate a sigqueue/signotify structure from the per process
 * pre-allocated pool.
 */
sigqueue_t *
sigqalloc(sigqhdr_t *sqh)
```



```
{
        sigqueue_t *sq = NULL;

        ASSERT(MUTEX_HELD(&curproc->p_lock));

        if (sqh != NULL) {
                mutex_enter(&sqh->sqb_lock);
                if (sqh->sqb_count > 0) {
                        sqh->sqb_count--;
                        if (sqh->sqb_free == NULL) {
                                /*
                                 * The pre-allocated pool is exhausted.
                                 */
                                sq = kmem_alloc(sizeof (sigqueue_t), KM_SLEEP);
                                sq->sq_func = NULL;
                        } else {
#endif /* ! codereview */
                                sq = sqh->sqb_free;
                                sq->sq_func = sigqrel;
#endif /* ! codereview */
                                sqh->sqb_free = sq->sq_next;
                        }
#endif /* ! codereview */
                        mutex_exit(&sqh->sqb_lock);
                        bzero(&sq->sq_info, sizeof (k_siginfo_t));
                        sq->sq_backptr = sqh;
                        sq->sq_func = sigqrel;
                        sq->sq_next = NULL;
                        sq->sq_external = 0;
                } else {
                        mutex_exit(&sqh->sqb_lock);
                }
        }
        return (sq);
}
______unchanged_portion_omitted_
```

****
10030 Sat Jul 13 17:57:09 2013
new/usr/src/uts/common/sys/signal.h
3830 SIGQUEUE_MAX's limit of 32 is too low
****
_____***unchanged_portion_omitted_***

```
 303 typedef struct sigqhdr {                /* sigqueue pool header         */
 304         sigqueue_t      *sqb_free;      /* free sigq struct list        */
 305         int             sqb_count;      /* sigq free count              */
 306         uint_t          sqb_maxcount;   /* sigq max free count          */
 307         size_t          sqb_size;       /* size of header+free structs  */
 306         uchar_t         sqb_count;      /* sigq free count              */
 307         uchar_t         sqb_maxcount;   /* sigq max free count          */
 308         ushort_t        sqb_size;       /* size of header+free structs  */
 308         uchar_t         sqb_pexited;    /* process has exited           */
 309         uint_t          sqb_sent;       /* number of sigq sent          */
 310         uchar_t         sqb_sent;       /* number of sigq sent          */
 310         kcondvar_t      sqb_cv;         /* waiting for a sigq struct    */
 311         kmutex_t        sqb_lock;       /* lock for sigq pool           */
 312 } sigqhdr_t;

 314 #define _SIGQUEUE_SIZE_BASIC            128     /* basic limit */
 315 #define _SIGQUEUE_SIZE_PRIVILEGED       512     /* privileged limit */
 316 #define _SIGQUEUE_SIZE_MAX              8192    /* maximum limit */

 315 #define _SIGQUEUE_MAX   32
 318 #define _SIGNOTIFY_MAX  32

 320 extern  void    setsigact(int, void (*)(int), const k_sigset_t *, int);
 321 extern  void    sigorset(k_sigset_t *, const k_sigset_t *);
 322 extern  void    sigandset(k_sigset_t *, const k_sigset_t *);
 323 extern  void    sigdiffset(k_sigset_t *, const k_sigset_t *);
 324 extern  void    sigintr(k_sigset_t *, int);
 325 extern  void    sigunintr(k_sigset_t *);
 326 extern  void    sigreplace(k_sigset_t *, k_sigset_t *);

 328 extern  int     kill(pid_t, int);

 330 #endif /* _KERNEL */

 332 #ifdef  __cplusplus
 333 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    5578 Sat Jul 13 17:57:09 2013
new/usr/src/uts/common/syscall/sigqueue.c
3830 SIGQUEUE_MAX's limit of 32 is too low
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/* Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/param.h>
#include <sys/types.h>
#include <sys/sysmacros.h>
#include <sys/systm.h>
#include <sys/errno.h>
#include <sys/proc.h>
#include <sys/procset.h>
#include <sys/fault.h>
#include <sys/signal.h>
#include <sys/siginfo.h>
#include <sys/debug.h>

extern rctl_hndl_t rc_process_sigqueue;

#endif /* ! codereview */
static int
sigqkill(pid_t pid, sigsend_t *sigsend)
{
        proc_t *p;
        int error;

        if ((uint_t)sigsend->sig >= NSIG)
                return (EINVAL);

        if (pid == -1) {
                procset_t set;

                setprocset(&set, POP_AND, P_ALL, P_MYID, P_ALL, P_MYID);
                error = sigsendset(&set, sigsend);
        } else if (pid > 0) {
                mutex_enter(&pidlock);
```



```
                if ((p = prfind(pid)) == NULL || p->p_stat == SIDL)
                        error = ESRCH;
                else {
                        error = sigsendproc(p, sigsend);
                        if (error == 0 && sigsend->perm == 0)
                                error = EPERM;
                }
                mutex_exit(&pidlock);
        } else {
                int nfound = 0;
                pid_t pgid;

                if (pid == 0)
                        pgid = ttoproc(curthread)->p_pgrp;
                else
                        pgid = -pid;

                error = 0;
                mutex_enter(&pidlock);
                for (p = pgfind(pgid); p && !error; p = p->p_pglink) {
                        if (p->p_stat != SIDL) {
                                nfound++;
                                error = sigsendproc(p, sigsend);
                        }
                }
                mutex_exit(&pidlock);
                if (nfound == 0)
                        error = ESRCH;
                else if (error == 0 && sigsend->perm == 0)
                        error = EPERM;
        }

        return (error);
}


/*
 * for implementations that don't require binary compatibility,
 * the kill system call may be made into a library call to the
 * sigsend system call
 */
int
kill(pid_t pid, int sig)
{
        int error;
        sigsend_t v;

        bzero(&v, sizeof (v));
        v.sig = sig;
        v.checkperm = 1;
        v.sicode = SI_USER;
        if ((error = sigqkill(pid, &v)) != 0)
                return (set_errno(error));
        return (0);
}

/*
 * The handling of small unions, like the sigval argument to sigqueue,
 * is architecture dependent.  We have adopted the convention that the
 * value itself is passed in the storage which crosses the kernel
 * protection boundary.  This procedure will accept a scalar argument,
 * and store it in the appropriate value member of the sigsend_t structure.
 */
int
sigqueue(pid_t pid, int sig, /* union sigval */ void *value,
    int si_code, int block)
```

```
126 {
127         int error;
128         sigsend_t v;
129         sigqhdr_t *sqh;
130         proc_t *p = curproc;

132         /* The si_code value must indicate the signal will be queued */
133         if (pid <= 0 || !sigwillqueue(sig, si_code))
134                 return (set_errno(EINVAL));

136         if ((sqh = p->p_sigqhdr) == NULL) {
137                 rlim64_t sigqsz_max;

139                 mutex_enter(&p->p_lock);
140                 sigqsz_max = rctl_enforced_value(rc_process_sigqueue,
141                     p->p_rctls, p);
142                 mutex_exit(&p->p_lock);

144 #endif /* ! codereview */
145                 /* Allocate sigqueue pool first time */
146                 sqh = sigqhdralloc(sizeof (sigqueue_t), (uint_t)sigqsz_max);
 43                 sqh = sigqhdralloc(sizeof (sigqueue_t), _SIGQUEUE_MAX);
147                 mutex_enter(&p->p_lock);
148                 if (p->p_sigqhdr == NULL) {
149                         /* hang the pool head on proc */
150                         p->p_sigqhdr = sqh;
151                 } else {
152                         /* another lwp allocated the pool, free ours */
153                         sigqhdrfree(sqh);
154                         sqh = p->p_sigqhdr;
155                 }
156                 mutex_exit(&p->p_lock);
157         }

159         do {
160                 bzero(&v, sizeof (v));
161                 v.sig = sig;
162                 v.checkperm = 1;
163                 v.sicode = si_code;
164                 v.value.sival_ptr = value;
165                 if ((error = sigqkill(pid, &v)) != EAGAIN || !block)
166                         break;
167                 /* block waiting for another chance to allocate a sigqueue_t */
168                 mutex_enter(&sqh->sqb_lock);
169                 while (sqh->sqb_count == 0) {
170                         if (!cv_wait_sig(&sqh->sqb_cv, &sqh->sqb_lock)) {
171                                 error = EINTR;
172                                 break;
173                         }
174                 }
175                 mutex_exit(&sqh->sqb_lock);
176         } while (error == EAGAIN);

178         if (error)
179                 return (set_errno(error));
180         return (0);
181 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    5280 Sat Jul 13 17:57:10 2013
new/usr/src/uts/common/syscall/sysconfig.c
3830 SIGQUEUE_MAX's limit of 32 is too low
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
/*        All Rights Reserved   */

#include <sys/param.h>
#include <sys/types.h>
#include <sys/sysmacros.h>
#include <sys/systm.h>
#include <sys/tuneable.h>
#include <sys/errno.h>
#include <sys/var.h>
#include <sys/signal.h>
#include <sys/time.h>
#include <sys/sysconfig.h>
#include <sys/resource.h>
#include <sys/ulimit.h>
#include <sys/unistd.h>
#include <sys/debug.h>
#include <sys/cpuvar.h>
#include <sys/mman.h>
#include <sys/timer.h>
#include <sys/zone.h>
#include <sys/vm_usage.h>

extern rctl_hndl_t rc_process_sigqueue;

#endif /* ! codereview */
long
sysconfig(int which)
{
        switch (which) {

        /*
         * if it is not handled in mach_sysconfig either
         * it must be EINVAL.
         */
```



```
        default:
                return (mach_sysconfig(which)); /* 'uname -i'/os */

        case _CONFIG_CLK_TCK:
                return ((long)hz);      /* clock frequency per second */

        case _CONFIG_PROF_TCK:
                return ((long)hz);      /* profiling clock freq per sec */

        case _CONFIG_NGROUPS:
                /*
                 * Maximum number of supplementary groups.
                 */
                return (ngroups_max);

        case _CONFIG_OPEN_FILES:
                /*
                 * Maximum number of open files (soft limit).
                 */
                {
                        rlim64_t fd_ctl;
                        mutex_enter(&curproc->p_lock);
                        fd_ctl = rctl_enforced_value(
                            rctlproc_legacy[RLIMIT_NOFILE], curproc->p_rctls,
                            curproc);
                        mutex_exit(&curproc->p_lock);
                        return ((ulong_t)fd_ctl);
                }

        case _CONFIG_CHILD_MAX:
                /*
                 * Maximum number of processes.
                 */
                return (v.v_maxup);

        case _CONFIG_POSIX_VER:
                return (_POSIX_VERSION); /* current POSIX version */

        case _CONFIG_PAGESIZE:
                return (PAGESIZE);

        case _CONFIG_XOPEN_VER:
                return (_XOPEN_VERSION); /* current XOPEN version */

        case _CONFIG_NPROC_CONF:
                return (zone_ncpus_get(curproc->p_zone));

        case _CONFIG_NPROC_ONLN:
                return (zone_ncpus_online_get(curproc->p_zone));

        case _CONFIG_NPROC_MAX:
                return (max_ncpus);

        case _CONFIG_STACK_PROT:
                return (curproc->p_stkprot & ~PROT_USER);

        case _CONFIG_AIO_LISTIO_MAX:
                return (_AIO_LISTIO_MAX);

        case _CONFIG_AIO_MAX:
                return (_AIO_MAX);

        case _CONFIG_AIO_PRIO_DELTA_MAX:
                return (0);

        case _CONFIG_DELAYTIMER_MAX:
```

```
128                 return (INT_MAX);

130         case _CONFIG_MQ_OPEN_MAX:
131                 return (_MQ_OPEN_MAX);

133         case _CONFIG_MQ_PRIO_MAX:
134                 return (_MQ_PRIO_MAX);

136         case _CONFIG_RTSIG_MAX:
137                 return (_SIGRTMAX - _SIGRTMIN + 1);

139         case _CONFIG_SEM_NSEMS_MAX:
140                 return (_SEM_NSEMS_MAX);

142         case _CONFIG_SEM_VALUE_MAX:
143                 return (_SEM_VALUE_MAX);

145         case _CONFIG_SIGQUEUE_MAX:
146                 /*
147                  * Maximum number of outstanding queued signals.
148                  */
149                 {
150                         rlim64_t sigqsz_max;
151                         mutex_enter(&curproc->p_lock);
152                         sigqsz_max = rctl_enforced_value(rc_process_sigqueue,
153                             curproc->p_rctls, curproc);
154                         mutex_exit(&curproc->p_lock);
155                         return ((uint_t)sigqsz_max);
156                 }
 50                 return (_SIGQUEUE_MAX);

158         case _CONFIG_SIGRT_MIN:
159                 return (_SIGRTMIN);

161         case _CONFIG_SIGRT_MAX:
162                 return (_SIGRTMAX);

164         case _CONFIG_TIMER_MAX:
165                 return (timer_max);

167         case _CONFIG_PHYS_PAGES:
168                 /*
169                  * If the non-global zone has a phys. memory cap, use that.
170                  * We always report the system-wide value for the global zone,
171                  * even though rcapd can be used on the global zone too.
172                  */
173                 if (!INGLOBALZONE(curproc) &&
174                     curproc->p_zone->zone_phys_mcap != 0)
175                         return (MIN(btop(curproc->p_zone->zone_phys_mcap),
176                             physinstalled));

178                 return (physinstalled);

180         case _CONFIG_AVPHYS_PAGES:
181                 /*
182                  * If the non-global zone has a phys. memory cap, use
183                  * the phys. memory cap - zone's current rss.  We always
184                  * report the system-wide value for the global zone, even
185                  * though rcapd can be used on the global zone too.
186                  */
187                 if (!INGLOBALZONE(curproc) &&
188                     curproc->p_zone->zone_phys_mcap != 0) {
189                         pgcnt_t cap, rss, free;
190                         vmusage_t in_use;
191                         size_t cnt = 1;
```



```
193                         cap = btop(curproc->p_zone->zone_phys_mcap);
194                         if (cap > physinstalled)
195                                 return (freemem);

197                         if (vm_getusage(VMUSAGE_ZONE, 1, &in_use, &cnt,
198                             FKIOCTL) != 0)
199                                 in_use.vmu_rss_all = 0;
200                         rss = btop(in_use.vmu_rss_all);
201                         /*
202                          * Because rcapd implements a soft cap, it is possible
203                          * for rss to be temporarily over the cap.
204                          */
205                         if (cap > rss)
206                                 free = cap - rss;
207                         else
208                                 free = 0;
209                         return (MIN(free, freemem));
210                 }

212                 return (freemem);

214         case _CONFIG_MAXPID:
215                 return (maxpid);

217         case _CONFIG_CPUID_MAX:
218                 return (max_cpuid);

220         case _CONFIG_EPHID_MAX:
221                 return (MAXEPHUID);

223         case _CONFIG_SYMLOOP_MAX:
224                 return (MAXSYMLINKS);
225         }
226 }
```

_____***unchanged_portion_omitted_***