```
**********************************************************
   21226 Thu Sep 26 12:54:42 2013
new/usr/src/man/man5/resource_controls.5
3830 SIGQUEUE_MAX's limit of 32 is too low
Reviewed by: Cedric Blancher <cedric.blancher@gmail.com>
Reviewed by: John Kennedy <john.kennedy@delphix.com>
Reviewed by: Irek Szczesniak <iszczesniak@gmail.com>
**********************************************************
```

```
   1 '\" te
   2 .\" Copyright (c) 2007, Sun Microsystems, Inc. All Rights Reserved.
   3 .\" The contents of this file are subject to the terms of the Common Development
   4 .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
   5 .\" When distributing Covered Code, include this CDDL HEADER in each file and in
   6 .TH RESOURCE_CONTROLS 5 "Jul 19, 2013"
   6 .TH RESOURCE_CONTROLS 5 "Jul 2, 2007"
   7 .SH NAME
   8 resource_controls \- resource controls available through project database
   9 .SH DESCRIPTION
  10 .sp
  11 .LP
  12 The resource controls facility is configured through the project database. See
  13 \fBproject\fR(4). You can set and modify resource controls through the
  14 following utilities:
  15 .RS +4
  16 .TP
  17 .ie t \(bu
  18 .el o
  19 \fBprctl\fR(1)
  20 .RE
  21 .RS +4
  22 .TP
  23 .ie t \(bu
  24 .el o
  25 \fBprojadd\fR(1M)
  26 .RE
  27 .RS +4
  28 .TP
  29 .ie t \(bu
  30 .el o
  31 \fBprojmod\fR(1M)
  32 .RE
  33 .RS +4
  34 .TP
  35 .ie t \(bu
  36 .el o
  37 \fBrctladm\fR(1M)
  38 .RE
  39 .sp
  40 .LP
  41 In a program, you use \fBsetrctl\fR(2) to set resource control values.
  42 .sp
  43 .LP
  44 In addition to the preceding resource controls, there are resource pools,
  45 accessible through the \fBpooladm\fR(1M) and \fBpoolcfg\fR(1M) utilities. In a
  46 program, resource pools can be manipulated through the \fBlibpool\fR(3LIB)
  47 library.
  48 .sp
  49 .LP
  50 The following are the resource controls are available:
  51 .sp
  52 .ne 2
  53 .na
  54 \fB\fBprocess.max-address-space\fR\fR
  55 .ad
  56 .sp .6
  57 .RS 4n
```



```
  58 Maximum amount of address space, as summed over segment sizes, that is
  59 available to this process, expressed as a number of bytes.
  60 .RE

  62 .sp
  63 .ne 2
  64 .na
  65 \fB\fBprocess.max-core-size\fR\fR
  66 .ad
  67 .sp .6
  68 .RS 4n
  69 Maximum size of a core file created by this process, expressed as a number of
  70 bytes.
  71 .RE

  73 .sp
  74 .ne 2
  75 .na
  76 \fB\fBprocess.max-cpu-time\fR\fR
  77 .ad
  78 .sp .6
  79 .RS 4n
  80 Maximum CPU time that is available to this process, expressed as a number of
  81 seconds.
  82 .RE

  84 .sp
  85 .ne 2
  86 .na
  87 \fB\fBprocess.max-data-size\fR\fR
  88 .ad
  89 .sp .6
  90 .RS 4n
  91 Maximum heap memory available to this process, expressed as a number of bytes.
  92 .RE

  94 .sp
  95 .ne 2
  96 .na
  97 \fB\fBprocess.max-file-descriptor\fR\fR
  98 .ad
  99 .sp .6
 100 .RS 4n
 101 Maximum file descriptor index available to this process, expressed as an
 102 integer.
 103 .RE

 105 .sp
 106 .ne 2
 107 .na
 108 \fB\fBprocess.max-file-size\fR\fR
 109 .ad
 110 .sp .6
 111 .RS 4n
 112 Maximum file offset available for writing by this process, expressed as a
 113 number of bytes.
 114 .RE

 116 .sp
 117 .ne 2
 118 .na
 119 \fB\fBprocess.max-msg-messages\fR\fR
 120 .ad
 121 .sp .6
 122 .RS 4n
 123 Maximum number of messages on a message queue (value copied from the resource
```

```
124 control at \fBmsgget()\fR time), expressed as an integer.
125 .RE

127 .sp
128 .ne 2
129 .na
130 \fB\fBprocess.max-msg-qbytes\fR\fR
131 .ad
132 .sp .6
133 .RS 4n
134 Maximum number of bytes of messages on a message queue (value copied from the
135 resource control at \fBmsgget()\fR time), expressed as a number of bytes.
136 .RE

138 .sp
139 .ne 2
140 .na
141 \fB\fBprocess.max-port-events\fR\fR
142 .ad
143 .sp .6
144 .RS 4n
145 Maximum allowable number of events per event port, expressed as an integer.
146 .RE

148 .sp
149 .ne 2
150 .na
151 \fB\fBprocess.max-sem-nsems\fR\fR
152 .ad
153 .sp .6
154 .RS 4n
155 Maximum number of semaphores allowed per semaphore set, expressed as an
156 integer.
157 .RE

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

171 .sp
172 .ne 2
173 .na
174 \fB\fBprocess.max-sigqueue-size\fR\fR
175 .ad
176 .sp .6
177 .RS 4n
178 Maximum number of outstanding queued signals.
179 .RE

181 .sp
182 .ne 2
183 .na
184 #endif /* ! codereview */
185 \fB\fBprocess.max-stack-size\fR\fR
186 .ad
187 .sp .6
188 .RS 4n
189 Maximum stack memory segment available to this process, expressed as a number
```



```
190 of bytes.
191 .RE

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

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

219 .sp
220 .ne 2
221 .na
222 \fB\fBproject.max-contracts\fR\fR
223 .ad
224 .sp .6
225 .RS 4n
226 Maximum number of contracts allowed in a project, expressed as an integer.
227 .RE

229 .sp
230 .ne 2
231 .na
232 \fB\fBproject.max-crypto-memory\fR\fR
233 .ad
234 .sp .6
235 .RS 4n
236 Maximum amount of kernel memory that can be used for crypto operations.
237 Allocations in the kernel for buffers and session-related structures are
238 charged against this resource control.
239 .RE

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

252 .sp
253 .ne 2
254 .na
255 \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
```

```
520 scaled threshold value:
521 .sp
522 .in +2
523 .nf
524 task.max-lwps=(priv,1K,deny)
525 .fi
526 .in -2

528 .sp
529 .LP
530 In the \fBproject\fR file, the value \fB1K\fR is expanded to \fB1000\fR:
531 .sp
532 .in +2
533 .nf
534 task.max-lwps=(priv,1000,deny)
535 .fi
536 .in -2

538 .sp
539 .LP
540 A second example uses a larger scaled value:
541 .sp
542 .in +2
543 .nf
544 process.max-file-size=(priv,5G,deny)
545 .fi
546 .in -2

548 .sp
549 .LP
550 In the \fBproject\fR file, the value \fB5G\fR is expanded to \fB5368709120\fR:
551 .sp
552 .in +2
553 .nf
554 process.max-file-size=(priv,5368709120,deny)
555 .fi
556 .in -2

558 .sp
559 .LP
560 The preceding examples use the scaling factors specified in the table above.
561 .sp
562 .LP
563 Note that unit modifiers (for example, \fB5G\fR) are accepted by the
564 \fBprctl\fR(1), \fBprojadd\fR(1M), and \fBprojmod\fR(1M) commands. You cannot
565 use unit modifiers in the project database itself.
566 .SS "Resource Control Values and Privilege Levels"
567 .sp
568 .LP
569 A threshold value on a resource control constitutes a point at which local
570 actions can be triggered or global actions, such as logging, can occur.
571 .sp
572 .LP
573 Each threshold value on a resource control must be associated with a privilege
574 level. The privilege level must be one of the following three types:
575 .sp
576 .ne 2
577 .na
578 \fB\fBbasic\fR\fR
579 .ad
580 .sp .6
581 .RS 4n
582 Can be modified by the owner of the calling process.
583 .RE

585 .sp
```



```
586 .ne 2
587 .na
588 \fB\fBprivileged\fR\fR
589 .ad
590 .sp .6
591 .RS 4n
592 Can be modified by the current process (requiring \fBsys_resource\fR privilege)
593 or by \fBprctl\fR(1) (requiring \fBproc_owner\fR privilege).
594 .RE

596 .sp
597 .ne 2
598 .na
599 \fB\fBsystem\fR\fR
600 .ad
601 .sp .6
602 .RS 4n
603 Fixed for the duration of the operating system instance.
604 .RE

606 .sp
607 .LP
608 A resource control is guaranteed to have one \fBsystem\fR value, which is
609 defined by the system, or resource provider. The \fBsystem\fR value represents
610 how much of the resource the current implementation of the operating system is
611 capable of providing.
612 .sp
613 .LP
614 Any number of privileged values can be defined, and only one basic value is
615 allowed. Operations that are performed without specifying a privilege value are
616 assigned a basic privilege by default.
617 .sp
618 .LP
619 The privilege level for a resource control value is defined in the privilege
620 field of the resource control block as \fBRCTL_BASIC\fR, \fBRCTL_PRIVILEGED\fR,
621 or \fBRCTL_SYSTEM\fR. See \fBsetrctl\fR(2) for more information. You can use
622 the \fBprctl\fR command to modify values that are associated with basic and
623 privileged levels.
624 .sp
625 .LP
626 In specifying the privilege level of \fBprivileged\fR, you can use the
627 abbreviation \fBpriv\fR. For example:
628 .sp
629 .in +2
630 .nf
631 task.max-lwps=(priv,1K,deny)
632 .fi
633 .in -2

635 .SS "Global and Local Actions on Resource Control Values"
636 .sp
637 .LP
638 There are two categories of actions on resource control values: global and
639 local.
640 .sp
641 .LP
642 Global actions apply to resource control values for every resource control on
643 the system. You can use \fBrctladm\fR(1M) to perform the following actions:
644 .RS +4
645 .TP
646 .ie t \(bu
647 .el o
648 Display the global state of active system resource controls.
649 .RE
650 .RS +4
651 .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
```

****
1325 Thu Sep 26 12:54:42 2013
new/usr/src/pkg/manifests/system-test-ostest.mf
3830 SIGQUEUE_MAX's limit of 32 is too low
Reviewed by: Cedric Blancher <cedric.blancher@gmail.com>
Reviewed by: John Kennedy <john.kennedy@delphix.com>
Reviewed by: Irek Szczesniak <iszczesniak@gmail.com>
****

```
   1 #
   2 # This file and its contents are supplied under the terms of the
   3 # Common Development and Distribution License ("CDDL"), version 1.0.
   4 # You may only use this file in accordance with the terms of version
   5 # 1.0 of the CDDL.
   6 #
   7 # A full copy of the text of the CDDL should have accompanied this
   8 # source.  A copy of the CDDL is also available via the Internet at
   9 # http://www.illumos.org/license/CDDL.
  10 #

  12 #
  13 # Copyright (c) 2012 by Delphix. All rights reserved.
  14 #

  16 set name=pkg.fmri value=pkg:/system/test/ostest@$(PKGVERS)
  17 set name=pkg.description value="Miscellaneous OS Unit Tests"
  18 set name=pkg.summary value="OS Unit Test Suite"
  19 set name=info.classification \
  20     value=org.opensolaris.category.2008:Development/System
  21 set name=variant.arch value=$(ARCH)
  22 dir path=opt/os-tests
  23 dir path=opt/os-tests/bin
  24 dir path=opt/os-tests/runfiles
  25 dir path=opt/os-tests/tests
  26 dir path=opt/os-tests/tests/sigqueue
  27 #endif /* ! codereview */
  28 file path=opt/os-tests/README mode=0444
  29 file path=opt/os-tests/bin/ostest mode=0555
  30 file path=opt/os-tests/runfiles/delphix.run mode=0444
  31 file path=opt/os-tests/runfiles/openindiana.run mode=0444
  32 file path=opt/os-tests/tests/poll_test mode=0555
  33 file path=opt/os-tests/tests/sigqueue/sigqueue_queue_size mode=0555
  34 #endif /* ! codereview */
  35 license cr_Sun license=cr_Sun
  36 license lic_CDDL license=lic_CDDL
  37 depend fmri=system/test/testrunner type=require
```

****
671 Thu Sep 26 12:54:43 2013
new/usr/src/test/os-tests/runfiles/delphix.run
3830 SIGQUEUE_MAX's limit of 32 is too low
Reviewed by: Cedric Blancher <cedric.blancher@gmail.com>
Reviewed by: John Kennedy <john.kennedy@delphix.com>
Reviewed by: Irek Szczesniak <iszczesniak@gmail.com>
****

```
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright (c) 2012 by Delphix. All rights reserved.
#

[DEFAULT]
pre =
verbose = False
quiet = False
user = root
timeout = 60
post =
outputdir = /var/tmp/test_results

[/opt/os-tests/tests/poll_test]
user = root
#endif /* ! codereview */

[/opt/os-tests/tests/sigqueue]
tests = ['sigqueue_queue_size']
#endif /* ! codereview */
```

```
***********************************************************
     671 Thu Sep 26 12:54:43 2013
new/usr/src/test/os-tests/runfiles/openindiana.run
3830 SIGQUEUE_MAX's limit of 32 is too low
Reviewed by: Cedric Blancher <cedric.blancher@gmail.com>
Reviewed by: John Kennedy <john.kennedy@delphix.com>
Reviewed by: Irek Szczesniak <iszczesniak@gmail.com>
***********************************************************
```

```
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright (c) 2012 by Delphix. All rights reserved.
#

[DEFAULT]
pre =
verbose = False
quiet = False
user = root
timeout = 60
post =
outputdir = /var/tmp/test_results

[/opt/os-tests/tests/poll_test]
user = root
#endif /* ! codereview */

[/opt/os-tests/tests/sigqueue]
tests = ['sigqueue_queue_size']
#endif /* ! codereview */
```

```
**********************************************************
     520 Thu Sep 26 12:54:44 2013
new/usr/src/test/os-tests/tests/Makefile
3830 SIGQUEUE_MAX's limit of 32 is too low
Reviewed by: Cedric Blancher <cedric.blancher@gmail.com>
Reviewed by: John Kennedy <john.kennedy@delphix.com>
Reviewed by: Irek Szczesniak <iszczesniak@gmail.com>
**********************************************************
```

```
   1 #
   2 # This file and its contents are supplied under the terms of the
   3 # Common Development and Distribution License ("CDDL"), version 1.0.
   4 # You may only use this file in accordance with the terms of version
   5 # 1.0 of the CDDL.
   6 #
   7 # A full copy of the text of the CDDL should have accompanied this
   8 # source.  A copy of the CDDL is also available via the Internet at
   9 # http://www.illumos.org/license/CDDL.
  10 #

  12 #
  13 # Copyright (c) 2012 by Delphix. All rights reserved.
  14 #

  16 SUBDIRS = poll sigqueue
  16 SUBDIRS = poll

  18 include $(SRC)/test/Makefile.com
```

```
**********************************************************
    1104 Thu Sep 26 12:54:44 2013
new/usr/src/test/os-tests/tests/sigqueue/Makefile
3830 SIGQUEUE_MAX's limit of 32 is too low
Reviewed by: Cedric Blancher <cedric.blancher@gmail.com>
Reviewed by: John Kennedy <john.kennedy@delphix.com>
Reviewed by: Irek Szczesniak <iszczesniak@gmail.com>
**********************************************************
```

```
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright (c) 2012 by Delphix. All rights reserved.
# Copyright (c) 2013 David Hoeppner. All rights reserved.
#

include $(SRC)/cmd/Makefile.cmd
include $(SRC)/test/Makefile.com

PROG = sigqueue_queue_size
OBJS = $(PROG:%=%.o)
SRCS = $(OBJS:%.o=%.c)

C99MODE = -xc99=%all

ROOTOPTPKG = $(ROOT)/opt/os-tests
TESTDIR = $(ROOTOPTPKG)/tests/sigqueue

CMDS = $(PROG:%=$(TESTDIR)/%)
$(CMDS) := FILEMODE = 0555

all: $(PROG)

$(PROG): $(OBJS)
        $(LINK.c) $(OBJS) -o $@ $(LDLIBS)
        $(POST_PROCESS)

%.o: ../%.c
        $(COMPILE.c) $<

install: all $(CMDS)

lint: lint_SRCS

clobber: clean
        -$(RM) $(PROG)

clean:
        -$(RM) $(OBJS)

$(CMDS): $(TESTDIR) $(PROG)

$(TESTDIR):
        $(INS.dir)

$(TESTDIR)/%: %
        $(INS.file)
#endif /* ! codereview */
```

```
**********************************************************
    3257 Thu Sep 26 12:54:44 2013
new/usr/src/test/os-tests/tests/sigqueue/sigqueue_queue_size.c
3830 SIGQUEUE_MAX's limit of 32 is too low
Reviewed by: Cedric Blancher <cedric.blancher@gmail.com>
Reviewed by: John Kennedy <john.kennedy@delphix.com>
Reviewed by: Irek Szczesniak <iszczesniak@gmail.com>
**********************************************************
```

```
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */

/*
 * Copyright 2013 David Hoeppner.  All rights reserved.
 */

/*
 * Queue maximum number of signals and test if we can queue more signals then
 * allowed.
 */

#include <sys/types.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

#define SIGQUEUE_SIGNAL         SIGRTMIN        /* Signal used for testing */

int nreceived = 0;

static void
test_start(const char *test_name, const char *format, ...)
{
        va_list args;

        (void) printf("TEST STARTING %s: ", test_name);

        va_start(args, format);
        (void) vprintf(format, args);
        va_end(args);
        (void) fflush(stdout);
}

static void
test_failed(const char *test_name, const char *format, ...)
{
        va_list args;

        (void) printf("TEST FAILED %s: ", test_name);

        va_start(args, format);
        (void) vprintf(format, args);
        va_end(args);

        (void) exit(-1);
}
```



```
static void
test_passed(const char *test_name)
{
        (void) printf("TEST PASS: %s\n", test_name);
        (void) fflush(stdout);
}

/* ARGSUSED */
static void
maximum_test_handler(int signal, siginfo_t *siginfo, void *context)
{
        nreceived++;
}

static void
sigqueue_maximum_test(void)
{
        const char *test_name = __func__;
        struct sigaction action;
        long sigqueue_max, i;
        pid_t pid;
        union sigval value;
        int error;

        test_start(test_name, "queue maximum number of signals\n");

        /*
         * Get the maximum size of the queue.
         */
        sigqueue_max = sysconf(_SC_SIGQUEUE_MAX);
        if (sigqueue_max == -1) {
                test_failed(test_name, "sysconf\n");
        }

        /*
         * Put the signal on hold.
         */
        error = sighold(SIGQUEUE_SIGNAL);
        if (error == -1) {
                test_failed(test_name, "sighold\n");
        }

        pid = getpid();
        value.sival_int = 0;

        action.sa_flags = SA_SIGINFO;
        action.sa_sigaction = maximum_test_handler;

        error = sigemptyset(&action.sa_mask);
        if (error == -1) {
                test_failed(test_name, "sigemptyset\n");
        }

        /*
         * Set signal handler.
         */
        error = sigaction(SIGQUEUE_SIGNAL, &action, 0);
        if (error == -1) {
                test_failed(test_name, "sigaction\n");
        }

        /*
         * Fill the signal queue to the maximum.
         */
        for (i = 0; i < sigqueue_max; i++) {
                error = sigqueue(pid, SIGQUEUE_SIGNAL, value);
```

```
125                 if (error == -1) {
126                         test_failed(test_name, "sigqueue\n");
127                 }
128         }

130         /*
131          * Send a further signal and test if we get the expected
132          * error.
133          */
134         error = sigqueue(pid, SIGQUEUE_SIGNAL, value);
135         if (error != -1) {
136                 test_failed(test_name, "sigqueue\n");
137         }

139         /*
140          * Unblock the signals and check if we received all messages
141          * from the signal queue.
142          */
143         error = sigrelse(SIGQUEUE_SIGNAL);
144         if (error == -1) {
145                 test_failed(test_name, "sigrelse\n");
146         }

148         if (nreceived != sigqueue_max) {
149                 test_failed(test_name, "nreceived != sigqueue_max\n");
150         }

152         test_passed(test_name);
153 }

155 static void
156 run_tests(void)
157 {
158         sigqueue_maximum_test();
159 }

161 /* ARGSUSED */
162 int
163 main(int argc, char *argv[])
164 {
165         run_tests();

167         return (EXIT_SUCCESS);
168 }
169 #endif /* ! codereview */
```

```
**********************************************************
   30414 Thu Sep 26 12:54:45 2013
new/usr/src/test/test-runner/cmd/run.py
3830 SIGQUEUE_MAX's limit of 32 is too low
Reviewed by: Cedric Blancher <cedric.blancher@gmail.com>
Reviewed by: John Kennedy <john.kennedy@delphix.com>
Reviewed by: Irek Szczesniak <iszczesniak@gmail.com>
**********************************************************
```

```
#!/usr/bin/python2.6

#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright (c) 2012 by Delphix. All rights reserved.
#

import ConfigParser
import os
import logging
from datetime import datetime
from optparse import OptionParser
from pwd import getpwnam
from pwd import getpwuid
from select import select
from subprocess import PIPE
from subprocess import Popen
from sys import argv
from sys import exit
from threading import Timer
from time import time

BASEDIR = '/var/tmp/test_results'
KILL = '/usr/bin/kill'
TRUE = '/usr/bin/true'
SUDO = '/usr/bin/sudo'


class Result(object):
    total = 0
    runresults = {'PASS': 0, 'FAIL': 0, 'SKIP': 0, 'KILLED': 0}

    def __init__(self):
        self.starttime = None
        self.returncode = None
        self.runtime = ''
        self.stdout = []
        self.stderr = []
        self.result = ''

    def done(self, proc, killed):
        """
        Finalize the results of this Cmd.
        """
        Result.total += 1
        m, s = divmod(time() - self.starttime, 60)
        self.runtime = '%02d:%02d' % (m, s)
        self.returncode = proc.returncode
```



```
        if killed:
            self.result = 'KILLED'
            Result.runresults['KILLED'] += 1
        elif self.returncode is 0:
            self.result = 'PASS'
            Result.runresults['PASS'] += 1
        elif self.returncode is not 0:
            self.result = 'FAIL'
            Result.runresults['FAIL'] += 1


class Output(object):
    """
    This class is a slightly modified version of the 'Stream' class found
    here: http://goo.gl/aSGfv
    """
    def __init__(self, stream):
        self.stream = stream
        self._buf = ''
        self.lines = []

    def fileno(self):
        return self.stream.fileno()

    def read(self, drain=0):
        """
        Read from the file descriptor. If 'drain' set, read until EOF.
        """
        while self._read() is not None:
            if not drain:
                break

    def _read(self):
        """
        Read up to 4k of data from this output stream. Collect the output
        up to the last newline, and append it to any leftover data from a
        previous call. The lines are stored as a (timestamp, data) tuple
        for easy sorting/merging later.
        """
        fd = self.fileno()
        buf = os.read(fd, 4096)
        if not buf:
            return None
        if '\n' not in buf:
            self._buf += buf
            return []

        buf = self._buf + buf
        tmp, rest = buf.rsplit('\n', 1)
        self._buf = rest
        now = datetime.now()
        rows = tmp.split('\n')
        self.lines += [(now, r) for r in rows]


class Cmd(object):
    verified_users = []

    def __init__(self, pathname, outputdir=None, timeout=None, user=None):
        self.pathname = pathname
        self.outputdir = outputdir or 'BASEDIR'
        self.timeout = timeout or 60
        self.user = user or ''
        self.killed = False
        self.result = Result()
```

```
125     def __str__(self):
126         return "Pathname: %s\nOutputdir: %s\nTimeout: %s\nUser: %s\n" % (
127                 self.pathname, self.outputdir, self.timeout, self.user)
128
129     def kill_cmd(self, proc):
130         """
131         Kill a running command due to timeout, or ^C from the keyboard. If
132         sudo is required, this user was verified previously.
133         """
134         self.killed = True
135         do_sudo = len(self.user) != 0
136         signal = '-TERM'
137
138         cmd = [SUDO, KILL, signal, str(proc.pid)]
139         if not do_sudo:
140             del cmd[0]
141
142         try:
143             kp = Popen(cmd)
144             kp.wait()
145         except:
146             pass
147
148     def update_cmd_privs(self, cmd, user):
149         """
150         If a user has been specified to run this Cmd and we're not already
151         running as that user, prepend the appropriate sudo command to run
152         as that user.
153         """
154         me = getpwuid(os.getuid())
155
156         if not user or user is me:
157             return cmd
158
159         ret = '%s -E -u %s %s' % (SUDO, user, cmd)
160         return ret.split(' ')
161
162     def collect_output(self, proc):
163         """
164         Read from stdout/stderr as data becomes available, until the
165         process is no longer running. Return the lines from the stdout and
166         stderr Output objects.
167         """
168         out = Output(proc.stdout)
169         err = Output(proc.stderr)
170         res = []
171         while proc.returncode is None:
172             proc.poll()
173             res = select([out, err], [], [], .1)
174             for fd in res[0]:
175                 fd.read()
176         for fd in res[0]:
177             fd.read(drain=1)
178
179         return out.lines, err.lines
180
181     def run(self, options):
182         """
183         This is the main function that runs each individual test.
184         Determine whether or not the command requires sudo, and modify it
185         if needed. Run the command, and update the result object.
186         """
187         if options.dryrun is True:
188             print self
189             return
```



```
191         privcmd = self.update_cmd_privs(self.pathname, self.user)
192         try:
193             old = os.umask(0)
194             if not os.path.isdir(self.outputdir):
195                 os.makedirs(self.outputdir, mode=0777)
196             os.umask(old)
197         except OSError, e:
198             fail('%s' % e)
199
200         try:
201             self.result.starttime = time()
202             proc = Popen(privcmd, stdout=PIPE, stderr=PIPE)
203             t = Timer(int(self.timeout), self.kill_cmd, [proc])
204             t.start()
205             self.result.stdout, self.result.stderr = self.collect_output(proc)
206         except KeyboardInterrupt:
207             self.kill_cmd(proc)
208             fail('\nRun terminated at user request.')
209         finally:
210             t.cancel()
211
212         self.result.done(proc, self.killed)
213
214     def skip(self):
215         """
216         Initialize enough of the test result that we can log a skipped
217         command.
218         """
219         Result.total += 1
220         Result.runresults['SKIP'] += 1
221         self.result.stdout = self.result.stderr = []
222         self.result.starttime = time()
223         m, s = divmod(time() - self.result.starttime, 60)
224         self.result.runtime = '%02d:%02d' % (m, s)
225         self.result.result = 'SKIP'
226
227     def log(self, logger, options):
228         """
229         This function is responsible for writing all output. This includes
230         the console output, the logfile of all results (with timestamped
231         merged stdout and stderr), and for each test, the unmodified
232         stdout/stderr/merged in it's own file.
233         """
234         if logger is None:
235             return
236
237         user = ' (run as %s)' % self.user if len(self.user) else ''
238         msga = 'Test: %s%s ' % (self.pathname, user)
239         msgb = '[%s] [%s]' % (self.result.runtime, self.result.result)
240         pad = ' ' * (80 - (len(msga) + len(msgb)))
241
242         # If -q is specified, only print a line for tests that didn't pass.
243         # This means passing tests need to be logged as DEBUG, or the one
244         # line summary will only be printed in the logfile for failures.
245         if not options.quiet:
246             logger.info('%s%s%s' % (msga, pad, msgb))
247         elif self.result.result is not 'PASS':
248             logger.info('%s%s%s' % (msga, pad, msgb))
249         else:
250             logger.debug('%s%s%s' % (msga, pad, msgb))
251
252         lines = self.result.stdout + self.result.stderr
253         for dt, line in sorted(lines):
254             logger.debug('%s %s' % (dt.strftime("%H:%M:%S.%f ")[:11], line))
255
256         if len(self.result.stdout):
```

```
257             with open(os.path.join(self.outputdir, 'stdout'), 'w') as out:
258                 for _, line in self.result.stdout:
259                     os.write(out.fileno(), '%s\n' % line)
260         if len(self.result.stderr):
261             with open(os.path.join(self.outputdir, 'stderr'), 'w') as err:
262                 for _, line in self.result.stderr:
263                     os.write(err.fileno(), '%s\n' % line)
264         if len(self.result.stdout) and len(self.result.stderr):
265             with open(os.path.join(self.outputdir, 'merged'), 'w') as merged:
266                 for _, line in sorted(lines):
267                     os.write(merged.fileno(), '%s\n' % line)


270 class Test(Cmd):
271     props = ['outputdir', 'timeout', 'user', 'pre', 'pre_user', 'post',
272              'post_user']

274     def __init__(self, pathname, outputdir=None, timeout=None, user=None,
275                  pre=None, pre_user=None, post=None, post_user=None):
276         super(Test, self).__init__(pathname, outputdir, timeout, user)
277         self.pre = pre or ''
278         self.pre_user = pre_user or ''
279         self.post = post or ''
280         self.post_user = post_user or ''

282     def __str__(self):
283         post_user = pre_user = ''
284         if len(self.pre_user):
285             pre_user = ' (as %s)' % (self.pre_user)
286         if len(self.post_user):
287             post_user = ' (as %s)' % (self.post_user)
288         return "Pathname: %s\nOutputdir: %s\nTimeout: %s\nPre: %s%s\nPost: " \
289                "%s%s\nUser: %s\n" % (self.pathname, self.outputdir,
290                 self.timeout, self.pre, pre_user, self.post, post_user,
291                 self.user)

293     def verify(self, logger):
294         """
295         Check the pre/post scripts, user and Test. Omit the Test from this
296         run if there are any problems.
297         """
298         files = [self.pre, self.pathname, self.post]
299         users = [self.pre_user, self.user, self.post_user]

301         for f in [f for f in files if len(f)]:
302             if not verify_file(f):
303                 logger.info("Warning: Test '%s' not added to this run because"
304                             " it failed verification." % f)
305                 return False

307         for user in [user for user in users if len(user)]:
308             if not verify_user(user, logger):
309                 logger.info("Not adding Test '%s' to this run." %
310                             self.pathname)
311                 return False

313         return True

315     def run(self, logger, options):
316         """
317         Create Cmd instances for the pre/post scripts. If the pre script
318         doesn't pass, skip this Test. Run the post script regardless.
319         """
320         pretest = Cmd(self.pre, outputdir=os.path.join(self.outputdir,
321                       os.path.basename(self.pre)), timeout=self.timeout,
322                       user=self.pre_user)
```



```
323         test = Cmd(self.pathname, outputdir=self.outputdir,
324                    timeout=self.timeout, user=self.user)
325         posttest = Cmd(self.post, outputdir=os.path.join(self.outputdir,
326                        os.path.basename(self.post)), timeout=self.timeout,
327                        user=self.post_user)

329         cont = True
330         if len(pretest.pathname):
331             pretest.run(options)
332             cont = pretest.result.result is 'PASS'
333             pretest.log(logger, options)

335         if cont:
336             test.run(options)
337         else:
338             test.skip()

340         test.log(logger, options)

342         if len(posttest.pathname):
343             posttest.run(options)
344             posttest.log(logger, options)


347 class TestGroup(Test):
348     props = Test.props + ['tests']

350     def __init__(self, pathname, outputdir=None, timeout=None, user=None,
351                  pre=None, pre_user=None, post=None, post_user=None,
352                  tests=None):
353         super(TestGroup, self).__init__(pathname, outputdir, timeout, user,
354                                         pre, pre_user, post, post_user)
355         self.tests = tests or []

357     def __str__(self):
358         post_user = pre_user = ''
359         if len(self.pre_user):
360             pre_user = ' (as %s)' % (self.pre_user)
361         if len(self.post_user):
362             post_user = ' (as %s)' % (self.post_user)
363         return "Pathname: %s\nOutputdir: %s\nTests: %s\nTimeout: %s\n" \
364                "Pre: %s%s\nPost: %s%s\nUser: %s\n" % (self.pathname,
365                 self.outputdir, self.tests, self.timeout, self.pre, pre_user,
366                 self.post, post_user, self.user)

368     def verify(self, logger):
369         """
370         Check the pre/post scripts, user and tests in this TestGroup. Omit
371         the TestGroup entirely, or simply delete the relevant tests in the
372         group, if that's all that's required.
373         """
374         # If the pre or post scripts are relative pathnames, convert to
375         # absolute, so they stand a chance of passing verification.
376         if len(self.pre) and not os.path.isabs(self.pre):
377             self.pre = os.path.join(self.pathname, self.pre)
378         if len(self.post) and not os.path.isabs(self.post):
379             self.post = os.path.join(self.pathname, self.post)

381         auxfiles = [self.pre, self.post]
382         users = [self.pre_user, self.user, self.post_user]

384         for f in [f for f in auxfiles if len(f)]:
385             if self.pathname != os.path.dirname(f):
386                 logger.info("Warning: TestGroup '%s' not added to this run. "
387                             "Auxiliary script '%s' exists in a different "
388                             "directory." % (self.pathname, f))
```

```
389                 return False

391             if not verify_file(f):
392                 logger.info("Warning: TestGroup '%s' not added to this run. "
393                             "Auxiliary script '%s' failed verification." %
394                             (self.pathname, f))
395                 return False

397         for user in [user for user in users if len(user)]:
398             if not verify_user(user, logger):
399                 logger.info("Not adding TestGroup '%s' to this run." %
400                             self.pathname)
401                 return False

403         # If one of the tests is invalid, delete it, log it, and drive on.
404         for test in self.tests:
405             if not verify_file(os.path.join(self.pathname, test)):
406                 del self.tests[self.tests.index(test)]
407                 logger.info("Warning: Test '%s' removed from TestGroup '%s' "
408                             "because it failed verification." % (test,
409                             self.pathname))

411         return len(self.tests) is not 0

413     def run(self, logger, options):
414         """
415         Create Cmd instances for the pre/post scripts. If the pre script
416         doesn't pass, skip all the tests in this TestGroup. Run the post
417         script regardless.
418         """
419         pretest = Cmd(self.pre, outputdir=os.path.join(self.outputdir,
420                       os.path.basename(self.pre)), timeout=self.timeout,
421                       user=self.pre_user)
422         posttest = Cmd(self.post, outputdir=os.path.join(self.outputdir,
423                        os.path.basename(self.post)), timeout=self.timeout,
424                        user=self.post_user)

426         cont = True
427         if len(pretest.pathname):
428             pretest.run(options)
429             cont = pretest.result.result is 'PASS'
430             pretest.log(logger, options)

432         for fname in self.tests:
433             test = Cmd(os.path.join(self.pathname, fname),
434                        outputdir=os.path.join(self.outputdir, fname),
435                        timeout=self.timeout, user=self.user)
436             if cont:
437                 test.run(options)
438             else:
439                 test.skip()

441             test.log(logger, options)

443         if len(posttest.pathname):
444             posttest.run(options)
445             posttest.log(logger, options)


448 class TestRun(object):
449     props = ['quiet', 'outputdir']

451     def __init__(self, options):
452         self.tests = {}
453         self.testgroups = {}
454         self.starttime = time()
```



```
455         self.timestamp = datetime.now().strftime('%Y%m%dT%H%M%S')
456         self.outputdir = os.path.join(options.outputdir, self.timestamp)
457         self.logger = self.setup_logging(options)
458         self.defaults = [
459             ('outputdir', BASEDIR),
460             ('quiet', False),
461             ('timeout', 60),
462             ('user', ''),
463             ('pre', ''),
464             ('pre_user', ''),
465             ('post', ''),
466             ('post_user', '')
467         ]

469     def __str__(self):
470         s = 'TestRun:\n    outputdir: %s\n' % self.outputdir
471         s += 'TESTS:\n'
472         for key in sorted(self.tests.keys()):
473             s += '%s%s' % (self.tests[key].__str__(), '\n')
474         s += 'TESTGROUPS:\n'
475         for key in sorted(self.testgroups.keys()):
476             s += '%s%s' % (self.testgroups[key].__str__(), '\n')
477         return s

479     def addtest(self, pathname, options):
480         """
481         Create a new Test, and apply any properties that were passed in
482         from the command line. If it passes verification, add it to the
483         TestRun.
484         """
485         test = Test(pathname)
486         for prop in Test.props:
487             setattr(test, prop, getattr(options, prop))

489         if test.verify(self.logger):
490             self.tests[pathname] = test

492     def addtestgroup(self, dirname, filenames, options):
493         """
494         Create a new TestGroup, and apply any properties that were passed
495         in from the command line. If it passes verification, add it to the
496         TestRun.
497         """
498         if dirname not in self.testgroups:
499             testgroup = TestGroup(dirname)
500             for prop in Test.props:
501                 setattr(testgroup, prop, getattr(options, prop))

503             # Prevent pre/post scripts from running as regular tests
504             for f in [testgroup.pre, testgroup.post]:
505                 if f in filenames:
506                     del filenames[filenames.index(f)]

508             self.testgroups[dirname] = testgroup
509             self.testgroups[dirname].tests = sorted(filenames)

511             testgroup.verify(self.logger)

513     def read(self, logger, options):
514         """
515         Read in the specified runfile, and apply the TestRun properties
516         listed in the 'DEFAULT' section to our TestRun. Then read each
517         section, and apply the appropriate properties to the Test or
518         TestGroup. Properties from individual sections override those set
519         in the 'DEFAULT' section. If the Test or TestGroup passes
520         verification, add it to the TestRun.
```

```
521         """
522         config = ConfigParser.RawConfigParser()
523         if not len(config.read(options.runfile)):
524             fail("Coulnd't read config file %s" % options.runfile)
525
526         for opt in TestRun.props:
527             if config.has_option('DEFAULT', opt):
528                 setattr(self, opt, config.get('DEFAULT', opt))
529         self.outputdir = os.path.join(self.outputdir, self.timestamp)
530
531         for section in config.sections():
532             if 'tests' in config.options(section):
533                 testgroup = TestGroup(section)
534                 for prop in TestGroup.props:
535                     try:
536                         setattr(testgroup, prop, config.get('DEFAULT', prop))
537                         setattr(testgroup, prop, config.get(section, prop))
538                     except ConfigParser.NoOptionError:
539                         pass
540
541                 # Repopulate tests using eval to convert the string to a list
542                 testgroup.tests = eval(config.get(section, 'tests'))
543
544                 if testgroup.verify(logger):
545                     self.testgroups[section] = testgroup
546             else:
547                 test = Test(section)
548                 for prop in Test.props:
549                     try:
550                         setattr(test, prop, config.get('DEFAULT', prop))
551                         setattr(test, prop, config.get(section, prop))
552                     except ConfigParser.NoOptionError:
553                         pass
554                 if test.verify(logger):
555                     self.tests[section] = test
556
557     def write(self, options):
558         """
559         Create a configuration file for editing and later use. The
560         'DEFAULT' section of the config file is created from the
561         properties that were specified on the command line. Tests are
562         simply added as sections that inherit everything from the
563         'DEFAULT' section. TestGroups are the same, except they get an
564         option including all the tests to run in that directory.
565         """
566
567         defaults = dict([(prop, getattr(options, prop)) for prop, _ in
568                          self.defaults])
569         config = ConfigParser.RawConfigParser(defaults)
570
571         for test in sorted(self.tests.keys()):
572             config.add_section(test)
573
574         for testgroup in sorted(self.testgroups.keys()):
575             config.add_section(testgroup)
576             config.set(testgroup, 'tests', self.testgroups[testgroup].tests)
577
578         try:
579             with open(options.template, 'w') as f:
580                 return config.write(f)
581         except IOError:
582             fail('Could not open \'%s\' for writing.' % options.template)
583
584     def complete_outputdirs(self, options):
585         """
586         Collect all the pathnames for Tests, and TestGroups. Work
```



```
587         backwards one pathname component at a time, to create a unique
588         directory name in which to deposit test output. Tests will be able
589         to write output files directly in the newly modified outputdir.
590         TestGroups will be able to create one subdirectory per test in the
591         outputdir, and are guaranteed uniqueness because a group can only
592         contain files in one directory. Pre and post tests will create a
593         directory rooted at the outputdir of the Test or TestGroup in
594         question for their output.
595         """
596         done = False
597         components = 0
598         tmp_dict = dict(self.tests.items() + self.testgroups.items())
599         total = len(tmp_dict)
600         base = self.outputdir
601
602         while not done:
603             l = []
604             components -= 1
605             for testfile in tmp_dict.keys():
606                 uniq = '/'.join(testfile.split('/')[components:]).lstrip('/')
607                 if not uniq in l:
608                     l.append(uniq)
609                     tmp_dict[testfile].outputdir = os.path.join(base, uniq)
610                 else:
611                     break
612             done = total == len(l)
613
614     def setup_logging(self, options):
615         """
616         Two loggers are set up here. The first is for the logfile which
617         will contain one line summarizing the test, including the test
618         name, result, and running time. This logger will also capture the
619         timestamped combined stdout and stderr of each run. The second
620         logger is optional console output, which will contain only the one
621         line summary. The loggers are initialized at two different levels
622         to facilitate segregating the output.
623         """
624         if options.dryrun is True:
625             return
626
627         testlogger = logging.getLogger(__name__)
628         testlogger.setLevel(logging.DEBUG)
629
630         if options.cmd is not 'wrconfig':
631             try:
632                 old = os.umask(0)
633                 os.makedirs(self.outputdir, mode=0777)
634                 os.umask(old)
635             except OSError, e:
636                 fail('%s' % e)
637             filename = os.path.join(self.outputdir, 'log')
638
639             logfile = logging.FileHandler(filename)
640             logfile.setLevel(logging.DEBUG)
641             logfilefmt = logging.Formatter('%(message)s')
642             logfile.setFormatter(logfilefmt)
643             testlogger.addHandler(logfile)
644
645         cons = logging.StreamHandler()
646         cons.setLevel(logging.INFO)
647         consfmt = logging.Formatter('%(message)s')
648         cons.setFormatter(consfmt)
649         testlogger.addHandler(cons)
650
651         return testlogger
```

```
653     def run(self, options):
654         """
655         Walk through all the Tests and TestGroups, calling run().
656         """
657         try:
658             os.chdir(self.outputdir)
659         except OSError:
660             fail('Could not change to directory %s' % self.outputdir)
661         for test in sorted(self.tests.keys()):
662             self.tests[test].run(self.logger, options)
663         for testgroup in sorted(self.testgroups.keys()):
664             self.testgroups[testgroup].run(self.logger, options)

666     def summary(self):
667         if Result.total is 0:
668             return

670         print '\nResults Summary'
671         for key in Result.runresults.keys():
672             if Result.runresults[key] is not 0:
673                 print '%s\t% 4d' % (key, Result.runresults[key])

675         m, s = divmod(time() - self.starttime, 60)
676         h, m = divmod(m, 60)
677         print '\nRunning Time:\t%02d:%02d:%02d' % (h, m, s)
678         print 'Percent passed:\t%.1f%%' % ((float(Result.runresults['PASS']) /
679             float(Result.total)) * 100)
680         print 'Log directory:\t%s' % self.outputdir


683 def verify_file(pathname):
684     """
685     Verify that the supplied pathname is an executable regular file.
686     """
687     if os.path.isdir(pathname) or os.path.islink(pathname):
688         return False

690     if os.path.isfile(pathname) and os.access(pathname, os.X_OK):
691         return True

693     return False


696 def verify_user(user, logger):
697     """
698     Verify that the specified user exists on this system, and can execute
699     sudo without being prompted for a password.
700     """
701     testcmd = [SUDO, '-n', '-u', user, TRUE]
702     can_sudo = exists = True

704     if user in Cmd.verified_users:
705         return True

707     try:
708         _ = getpwnam(user)
709     except KeyError:
710         exists = False
711         logger.info("Warning: user '%s' does not exist.", user)
712         return False

714     p = Popen(testcmd)
715     p.wait()
716     if p.returncode is not 0:
717         logger.info("Warning: user '%s' cannot use passwordless sudo.", user)
717         logger.info("Warning: user '%s' cannot use paswordless sudo.", user)
```



```
718         return False
719     else:
720         Cmd.verified_users.append(user)

722     return True


725 def find_tests(testrun, options):
726     """
727     For the given list of pathnames, add files as Tests. For directories,
728     if do_groups is True, add the directory as a TestGroup. If False,
729     recursively search for executable files.
730     """

732     for p in sorted(options.pathnames):
733         if os.path.isdir(p):
734             for dirname, _, filenames in os.walk(p):
735                 if options.do_groups:
736                     testrun.addtestgroup(dirname, filenames, options)
737                 else:
738                     for f in sorted(filenames):
739                         testrun.addtest(os.path.join(dirname, f), options)
740         else:
741             testrun.addtest(p, options)


744 def fail(retstr, ret=1):
745     print '%s: %s' % (argv[0], retstr)
746     exit(ret)


749 def options_cb(option, opt_str, value, parser):
750     path_options = ['runfile', 'outputdir', 'template']

752     if option.dest is 'runfile' and '-w' in parser.rargs or \
753         option.dest is 'template' and '-c' in parser.rargs:
754         fail('-c and -w are mutually exclusive.')

756     if opt_str in parser.rargs:
757         fail('%s may only be specified once.' % opt_str)

759     if option.dest is 'runfile':
760         parser.values.cmd = 'rdconfig'
761     if option.dest is 'template':
762         parser.values.cmd = 'wrconfig'

764     setattr(parser.values, option.dest, value)
765     if option.dest in path_options:
766         setattr(parser.values, option.dest, os.path.abspath(value))


769 def parse_args():
770     parser = OptionParser()
771     parser.add_option('-c', action='callback', callback=options_cb,
772                       type='string', dest='runfile', metavar='runfile',
773                       help='Specify tests to run via config file.')
774     parser.add_option('-d', action='store_true', default=False, dest='dryrun',
775                       help='Dry run. Print tests, but take no other action.')
776     parser.add_option('-g', action='store_true', default=False,
777                       dest='do_groups', help='Make directories TestGroups.')
778     parser.add_option('-o', action='callback', callback=options_cb,
779                       default=BASEDIR, dest='outputdir', type='string',
780                       metavar='outputdir', help='Specify an output directory.')
781     parser.add_option('-p', action='callback', callback=options_cb,
782                       default='', dest='pre', metavar='script',
783                       type='string', help='Specify a pre script.')
```

```
784     parser.add_option('-P', action='callback', callback=options_cb,
785                       default='', dest='post', metavar='script',
786                       type='string', help='Specify a post script.')
787     parser.add_option('-q', action='store_true', default=False, dest='quiet',
788                       help='Silence on the console during a test run.')
789     parser.add_option('-t', action='callback', callback=options_cb, default=60,
790                       dest='timeout', metavar='seconds', type='int',
791                       help='Timeout (in seconds) for an individual test.')
792     parser.add_option('-u', action='callback', callback=options_cb,
793                       default='', dest='user', metavar='user', type='string',
794                       help='Specify a different user name to run as.')
795     parser.add_option('-w', action='callback', callback=options_cb,
796                       default=None, dest='template', metavar='template',
797                       type='string', help='Create a new config file.')
798     parser.add_option('-x', action='callback', callback=options_cb, default='',
799                       dest='pre_user', metavar='pre_user', type='string',
800                       help='Specify a user to execute the pre script.')
801     parser.add_option('-X', action='callback', callback=options_cb, default='',
802                       dest='post_user', metavar='post_user', type='string',
803                       help='Specify a user to execute the post script.')
804     (options, pathnames) = parser.parse_args()

806     if not options.runfile and not options.template:
807         options.cmd = 'runtests'

809     if options.runfile and len(pathnames):
810         fail('Extraneous arguments.')

812     options.pathnames = [os.path.abspath(path) for path in pathnames]

814     return options


817 def main(args):
818     options = parse_args()
819     testrun = TestRun(options)

821     if options.cmd is 'runtests':
822         find_tests(testrun, options)
823     elif options.cmd is 'rdconfig':
824         testrun.read(testrun.logger, options)
825     elif options.cmd is 'wrconfig':
826         find_tests(testrun, options)
827         testrun.write(options)
828         exit(0)
829     else:
830         fail('Unknown command specified')

832     testrun.complete_outputdirs(options)
833     testrun.run(options)
834     testrun.summary()
835     exit(0)


838 if __name__ == '__main__':
839     main(argv[1:])
```

```
**********************************************************
   12757 Thu Sep 26 12:54:45 2013
new/usr/src/uts/common/os/rctl_proc.c
3830 SIGQUEUE_MAX's limit of 32 is too low
Reviewed by: Cedric Blancher <cedric.blancher@gmail.com>
Reviewed by: John Kennedy <john.kennedy@delphix.com>
Reviewed by: Irek Szczesniak <iszczesniak@gmail.com>
**********************************************************
```

```
/*
 * 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>
#include <sys/var.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
```

```
123 proc_stack_set(rctl_t *rctl, struct proc *p, rctl_entity_p_t *e,
124     rctl_qty_t nv)
125 {
126         klwp_t *lwp = ttolwp(curthread);
127
128         if (p->p_model == DATAMODEL_NATIVE)
129                 nv = MIN(nv, rctl->rc_dict_entry->rcd_max_native);
130         else
131                 nv = MIN(nv, rctl->rc_dict_entry->rcd_max_ilp32);
132
133         /*
134          * In the process of changing the rlimit, this function actually
135          * gets called a number of times. We only want to save the current
136          * rlimit the first time we come through here. In post_syscall(),
137          * we copyin() the lwp's ustack, and compare it to the rlimit we
138          * save here; if the two match, we adjust the ustack to reflect
139          * the new stack bounds.
140          *
141          * We check to make sure that we're changing the rlimit of our
142          * own process rather than on behalf of some other process. The
143          * notion of changing this resource limit on behalf of another
144          * process is problematic at best, and changing the amount of stack
145          * space a process is allowed to consume is a rather antiquated
146          * notion that has limited applicability in our multithreaded
147          * process model.
148          */
149         ASSERT(e->rcep_t == RCENTITY_PROCESS);
150         if (lwp != NULL && lwp->lwp_procp == e->rcep_p.proc &&
151             lwp->lwp_ustack && lwp->lwp_old_stk_ctl == 0) {
152                 lwp->lwp_old_stk_ctl = (size_t)e->rcep_p.proc->p_stk_ctl;
153                 curthread->t_post_sys = 1;
154         }
155
156         e->rcep_p.proc->p_stk_ctl = nv;
157
158         return (0);
159 }
160
161 static rctl_ops_t proc_stack_ops = {
162         rcop_no_action,
163         rcop_no_usage,
164         proc_stack_set,
165         rcop_no_test
166 };
167
168 /*
169  * process.max-file-descriptors / RLIMIT_NOFILE
170  */
171 static int
172 proc_nofile_set(rctl_t *rctl, struct proc *p, rctl_entity_p_t *e, rctl_qty_t nv)
173 {
174         ASSERT(e->rcep_t == RCENTITY_PROCESS);
175         if (p->p_model == DATAMODEL_NATIVE)
176                 nv = MIN(nv, rctl->rc_dict_entry->rcd_max_native);
177         else
178                 nv = MIN(nv, rctl->rc_dict_entry->rcd_max_ilp32);
179
180         e->rcep_p.proc->p_fno_ctl = nv;
181
182         return (0);
183 }
184
185 static rctl_ops_t proc_nofile_ops = {
186         rcop_no_action,
187         rcop_no_usage,
188         proc_nofile_set,
```



```
189         rcop_absolute_test
190 };
191
192 /*
193  * process.max-address-space / RLIMIT_VMEM
194  */
195 static int
196 proc_vmem_set(rctl_t *rctl, struct proc *p, rctl_entity_p_t *e, rctl_qty_t nv)
197 {
198         ASSERT(e->rcep_t == RCENTITY_PROCESS);
199         if (p->p_model == DATAMODEL_ILP32)
200                 nv = MIN(nv, rctl->rc_dict_entry->rcd_max_ilp32);
201         else
202                 nv = MIN(nv, rctl->rc_dict_entry->rcd_max_native);
203
204         e->rcep_p.proc->p_vmem_ctl = nv;
205
206         return (0);
207 }
208
209 static rctl_ops_t proc_vmem_ops = {
210         rcop_no_action,
211         rcop_no_usage,
212         proc_vmem_set,
213         rcop_no_test
214 };
215
216 /*
217  * void rctlproc_default_init()
218  *
219  * Overview
220  *   Establish default basic and privileged control values on the init process.
221  *   These correspond to the soft and hard limits, respectively.
222  */
223 void
224 rctlproc_default_init(struct proc *initp, rctl_alloc_gp_t *gp)
225 {
226         struct rlimit64 rlp64;
227
228         /*
229          * RLIMIT_CPU: deny never, sigtoproc(pp, NULL, SIGXCPU).
230          */
231         rlp64.rlim_cur = rlp64.rlim_max = RLIM64_INFINITY;
232         (void) rctl_rlimit_set(rctlproc_legacy[RLIMIT_CPU], initp, &rlp64, gp,
233             RCTL_LOCAL_SIGNAL, SIGXCPU, kcred);
234
235         /*
236          * RLIMIT_FSIZE: deny always, sigtoproc(pp, NULL, SIGXFSZ).
237          */
238         rlp64.rlim_cur = rlp64.rlim_max = RLIM64_INFINITY;
239         (void) rctl_rlimit_set(rctlproc_legacy[RLIMIT_FSIZE], initp, &rlp64, gp,
240             RCTL_LOCAL_SIGNAL | RCTL_LOCAL_DENY, SIGXFSZ, kcred);
241
242         /*
243          * RLIMIT_DATA: deny always, no default action.
244          */
245         rlp64.rlim_cur = rlp64.rlim_max = RLIM64_INFINITY;
246         (void) rctl_rlimit_set(rctlproc_legacy[RLIMIT_DATA], initp, &rlp64, gp,
247             RCTL_LOCAL_DENY, 0, kcred);
248
249         /*
250          * RLIMIT_STACK: deny always, no default action.
251          */
252 #ifdef __sparc
253         rlp64.rlim_cur = DFLSSIZ;
254         rlp64.rlim_max = LONG_MAX;
```

```
#else
        rlp64.rlim_cur = DFLSSIZ;
        rlp64.rlim_max = MAXSSIZ;
#endif
        (void) rctl_rlimit_set(rctlproc_legacy[RLIMIT_STACK], initp, &rlp64, gp,
            RCTL_LOCAL_DENY, 0, kcred);

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

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

        /*
         * RLIMIT_VMEM
         */
        rlp64.rlim_cur = rlp64.rlim_max = RLIM64_INFINITY;
        (void) rctl_rlimit_set(rctlproc_legacy[RLIMIT_VMEM], initp, &rlp64, gp,
            RCTL_LOCAL_DENY, 0, kcred);
}

/*
 * void rctlproc_init()
 *
 * Overview
 *   Register the various resource controls associated with process entities.
 *   The historical rlim_infinity_map and rlim_infinity32_map are now encoded
 *   here as the native and ILP32 infinite values for each resource control.
 */
void
rctlproc_init(void)
 rctlproc_init()
{
        rctl_set_t *set;
        rctl_alloc_gp_t *gp;
        rctl_entity_p_t e;

        rctlproc_legacy[RLIMIT_CPU] = rctl_register("process.max-cpu-time",
            RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_NEVER |
            RCTL_GLOBAL_CPU_TIME | RCTL_GLOBAL_INFINITE | RCTL_GLOBAL_SECONDS,
            UINT64_MAX, UINT64_MAX, &proc_cpu_time_ops);
        rctlproc_legacy[RLIMIT_FSIZE] = rctl_register("process.max-file-size",
            RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
            RCTL_GLOBAL_FILE_SIZE | RCTL_GLOBAL_BYTES,
            MAXOFFSET_T, MAXOFFSET_T, &proc_filesize_ops);
        rctlproc_legacy[RLIMIT_DATA] = rctl_register("process.max-data-size",
            RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
            RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
            ULONG_MAX, UINT32_MAX, &rctl_default_ops);
#ifdef _LP64
#ifdef __sparc
        rctlproc_legacy[RLIMIT_STACK] = rctl_register("process.max-stack-size",
            RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
            RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
            LONG_MAX, INT32_MAX, &proc_stack_ops);
#else   /* __sparc */
        rctlproc_legacy[RLIMIT_STACK] = rctl_register("process.max-stack-size",
```



```
            RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
            RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
            MAXSSIZ, USRSTACK32 - PAGESIZE, &proc_stack_ops);
#endif  /* __sparc */
#else   /* _LP64 */
        rctlproc_legacy[RLIMIT_STACK] = rctl_register("process.max-stack-size",
            RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
            RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
            USRSTACK - PAGESIZE, USRSTACK - PAGESIZE, &proc_stack_ops);
#endif
        rctlproc_legacy[RLIMIT_CORE] = rctl_register("process.max-core-size",
            RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
            RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
            MIN(MAXOFFSET_T, ULONG_MAX), UINT32_MAX, &rctl_default_ops);
        rctlproc_legacy[RLIMIT_NOFILE] = rctl_register(
            "process.max-file-descriptor", RCENTITY_PROCESS,
            RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
            RCTL_GLOBAL_COUNT, INT32_MAX, INT32_MAX, &proc_nofile_ops);
        rctlproc_legacy[RLIMIT_VMEM] =
            rctl_register("process.max-address-space", RCENTITY_PROCESS,
            RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
            RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
            ULONG_MAX, UINT32_MAX, &proc_vmem_ops);

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

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

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

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

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

        /*
         * We set the upper limit to the maximum number of processes per user
         * to make it theoretical possible to deliver all SIGCHILD signals on
         * child termination, but at least to 8k.
         */
        rc_process_sigqueue = rctl_register("process.max-sigqueue-size",
            RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
            RCTL_GLOBAL_COUNT, MAX(v.v_maxup, 8192), MAX(v.v_maxup, 8192),
            &rctl_absolute_ops);
        rctl_add_default_limit("process.max-sigqueue-size",
            _SIGQUEUE_SIZE_BASIC, RCPRIV_BASIC, RCTL_LOCAL_DENY);
        rctl_add_default_limit("process.max-sigqueue-size",
```

```
386             _SIGQUEUE_SIZE_PRIVILEGED, RCPRIV_PRIVILEGED, RCTL_LOCAL_DENY);

388         /*
389 #endif /* ! codereview */
390          * Place minimal set of controls on "sched" process for inheritance by
391          * processes created via newproc().
392          */
393         set = rctl_set_create();
394         gp = rctl_set_init_prealloc(RCENTITY_PROCESS);
395         mutex_enter(&curproc->p_lock);
396         e.rcep_p.proc = curproc;
397         e.rcep_t = RCENTITY_PROCESS;
398         curproc->p_rctls = rctl_set_init(RCENTITY_PROCESS, curproc, &e,
399             set, gp);
400         mutex_exit(&curproc->p_lock);
401         rctl_prealloc_destroy(gp);
402 }
```

```
**********************************************************
   73723 Thu Sep 26 12:54:45 2013
new/usr/src/uts/common/os/sig.c
3830 SIGQUEUE_MAX's limit of 32 is too low
Reviewed by: Cedric Blancher <cedric.blancher@gmail.com>
Reviewed by: John Kennedy <john.kennedy@delphix.com>
Reviewed by: Irek Szczesniak <iszczesniak@gmail.com>
**********************************************************
______unchanged_portion_omitted_

#ifndef UCHAR_MAX
#define UCHAR_MAX       255
#endif

/*
 * 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;

        /*
         * Before the introduction of process.max-sigqueue-size
         * _SC_SIGQUEUE_MAX had this static value.
         */
#define _SIGQUEUE_PREALLOC      32

        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_
```

****
9977 Thu Sep 26 12:54:46 2013
new/usr/src/uts/common/sys/signal.h
3830 SIGQUEUE_MAX's limit of 32 is too low
Reviewed by: Cedric Blancher <cedric.blancher@gmail.com>
Reviewed by: John Kennedy <john.kennedy@delphix.com>
Reviewed by: Irek Szczesniak <iszczesniak@gmail.com>
****
_____***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 */

 315 #define _SIGQUEUE_MAX   32
 317 #define _SIGNOTIFY_MAX  32

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

 327 extern  int     kill(pid_t, int);

 329 #endif /* _KERNEL */

 331 #ifdef  __cplusplus
 332 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    5578 Thu Sep 26 12:54:46 2013
new/usr/src/uts/common/syscall/sigqueue.c
3830 SIGQUEUE_MAX's limit of 32 is too low
Reviewed by: Cedric Blancher <cedric.blancher@gmail.com>
Reviewed by: John Kennedy <john.kennedy@delphix.com>
Reviewed by: Irek Szczesniak <iszczesniak@gmail.com>
**********************************************************
/*
 * 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.
 */
```

```
123 int
124 sigqueue(pid_t pid, int sig, /* union sigval */ void *value,
125     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 Thu Sep 26 12:54:47 2013
new/usr/src/uts/common/syscall/sysconfig.c
3830 SIGQUEUE_MAX's limit of 32 is too low
Reviewed by: Cedric Blancher <cedric.blancher@gmail.com>
Reviewed by: John Kennedy <john.kennedy@delphix.com>
Reviewed by: Irek Szczesniak <iszczesniak@gmail.com>
**********************************************************
/*
 * 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:
```

```
125                 return (0);

127         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_***