

```

*****
21226 Fri Jul 19 13:59:25 2013
new/usr/src/man/man5/resource_controls.5
3830 SIGQUEUE_MAX's limit of 32 is too low
*****
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 \fBbrctladm\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 \fBblibpool\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 \fBFS(7)\fR). 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
256 .ad
257 .sp .6
258 .RS 4n

```

```

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

263 .sp
264 .ne 2
265 .na
266 \fB\fBproject.max-msg-ids\fR\fR
267 .ad
268 .sp .6
269 .RS 4n
270 Maximum number of message queue IDs allowed for a project, expressed as an
271 integer.
272 .RE

274 .sp
275 .ne 2
276 .na
277 \fB\fBproject.max-port-ids\fR\fR
278 .ad
279 .sp .6
280 .RS 4n
281 Maximum allowable number of event ports, expressed as an integer.
282 .RE

284 .sp
285 .ne 2
286 .na
287 \fB\fBproject.max-sem-ids\fR\fR
288 .ad
289 .sp .6
290 .RS 4n
291 Maximum number of semaphore IDs allowed for a project, expressed as an integer.
292 .RE

294 .sp
295 .ne 2
296 .na
297 \fB\fBproject.max-shm-ids\fR\fR
298 .ad
299 .sp .6
300 .RS 4n
301 Maximum number of shared memory IDs allowed for a project, expressed as an
302 integer.
303 .RE

305 .sp
306 .ne 2
307 .na
308 \fB\fBproject.max-shm-memory\fR\fR
309 .ad
310 .sp .6
311 .RS 4n
312 Total amount of shared memory allowed for a project, expressed as a number of
313 bytes.
314 .RE

316 .sp
317 .ne 2
318 .na
319 \fB\fBproject.max-tasks\fR\fR
320 .ad
321 .sp .6
322 .RS 4n
323 Maximum number of tasks allowable in a project, expressed as an integer.
324 .RE

```

```

326 .sp
327 .ne 2
328 .na
329 \fB\fBproject.pool\fR\fR
330 .ad
331 .sp .6
332 .RS 4n
333 Binds a specified resource pool with a project.
334 .RE

336 .sp
337 .ne 2
338 .na
339 \fB\fBrcap.max-rss\fR\fR
340 .ad
341 .sp .6
342 .RS 4n
343 The total amount of physical memory, in bytes, that is available to processes
344 in a project.
345 .RE

347 .sp
348 .ne 2
349 .na
350 \fB\fBtask.max-cpu-time\fR\fR
351 .ad
352 .sp .6
353 .RS 4n
354 Maximum CPU time that is available to this task's processes, expressed as a
355 number of seconds.
356 .RE

358 .sp
359 .ne 2
360 .na
361 \fB\fBtask.max-lwps\fR\fR
362 .ad
363 .sp .6
364 .RS 4n
365 Maximum number of LWPs simultaneously available to this task's processes,
366 expressed as an integer.
367 .RE

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

386 .sp
387 .ne 2
388 .na
389 \fB\fBzone.cpu-shares\fR\fR
390 .ad

```

```

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

400 .sp
401 .ne 2
402 .na
403 \fB\fBzone.max-locked-memory\fR\fR
404 .ad
405 .sp .6
406 .RS 4n
407 Total amount of physical locked memory available to a zone.
408 .RE

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

423 .sp
424 .ne 2
425 .na
426 \fB\fBzone.max-msg-ids\fR\fR
427 .ad
428 .sp .6
429 .RS 4n
430 Maximum number of message queue IDs allowed for a zone, expressed as an
431 integer.
432 .RE

434 .sp
435 .ne 2
436 .na
437 \fB\fBzone.max-sem-ids\fR\fR
438 .ad
439 .sp .6
440 .RS 4n
441 Maximum number of semaphore IDs allowed for a zone, expressed as an integer.
442 .RE

444 .sp
445 .ne 2
446 .na
447 \fB\fBzone.max-shm-ids\fR\fR
448 .ad
449 .sp .6
450 .RS 4n
451 Maximum number of shared memory IDs allowed for a zone, expressed as an
452 integer.
453 .RE

455 .sp
456 .ne 2

```

```

457 .na
458 \fB\fBzone.max-shm-memory\fR\fR
459 .ad
460 .sp .6
461 .RS 4n
462 Total amount of shared memory allowed for a zone, expressed as a number of
463 bytes.
464 .RE

466 .sp
467 .ne 2
468 .na
469 \fB\fBzone.max-swap\fR\fR
470 .ad
471 .sp .6
472 .RS 4n
473 Total amount of swap that can be consumed by user process address space
474 mappings and \fBtmpfs\fR mounts for this zone.
475 .RE

477 .sp
478 .LP
479 See \fBzones\fR(5).
480 .SS "Units Used in Resource Controls"
481 .sp
482 .LP
483 Resource controls can be expressed as in units of size (bytes), time (seconds),
484 or as a count (integer). These units use the strings specified below.
485 .sp
486 .in +2
487 .nf
488 Category           Res Ctrl   Modifier  Scale
489 Type String
490 -----
491 Size               bytes      B          1
492                   KB         2^10
493                   MB         2^20
494                   GB         2^30
495                   TB         2^40
496                   PB         2^50
497                   EB         2^60

499 Time               seconds    s          1
500                   Ks         10^3
501                   Ms         10^6
502                   Gs         10^9
503                   Ts         10^12
504                   Ps         10^15
505                   Es         10^18

507 Count              integer    none       1
508                   K          10^3
509                   M          10^6
510                   G          10^9
511                   T          10^12
512                   P          10^15
513                   Es         10^18
514 .fi
515 .in -2

517 .sp
518 .LP
519 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 \fBCTL_BASIC\fR, \fBCTL_PRIVILEGED\fR,
621 or \fBCTL_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 \fBrcldm\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
652 .ie t \(\bu
653 .el o
654 Set global logging actions.

```

```

655 .RE
656 .sp
657 .LP
658 You can disable or enable the global logging action on resource controls. You
659 can set the \fBsyslog\fR action to a specific degree by assigning a severity
660 level, \fBsyslog=\fRilevel\fR. The possible settings for ilevel\fR are as
661 follows:
662 .RS +4
663 .TP
664 .ie t \(\bu
665 .el o
666 \fBdebug\fR
667 .RE
668 .RS +4
669 .TP
670 .ie t \(\bu
671 .el o
672 \fBinfo\fR
673 .RE
674 .RS +4
675 .TP
676 .ie t \(\bu
677 .el o
678 \fBnotice\fR
679 .RE
680 .RS +4
681 .TP
682 .ie t \(\bu
683 .el o
684 \fBwarning\fR
685 .RE
686 .RS +4
687 .TP
688 .ie t \(\bu
689 .el o
690 \fBerr\fR
691 .RE
692 .RS +4
693 .TP
694 .ie t \(\bu
695 .el o
696 \fBcrit\fR
697 .RE
698 .RS +4
699 .TP
700 .ie t \(\bu
701 .el o
702 \fBalert\fR
703 .RE
704 .RS +4
705 .TP
706 .ie t \(\bu
707 .el o
708 \fBemerg\fR
709 .RE
710 .sp
711 .LP
712 By default, there is no global logging of resource control violations.
713 .sp
714 .LP
715 Local actions are taken on a process that attempts to exceed the control value.
716 For each threshold value that is placed on a resource control, you can
717 associate one or more actions. There are three types of local actions:
718 \fBnone\fR, \fBdeny\fR, and \fBsignal=\fR. These three actions are used as
719 follows:
720 .sp

```

```

721 .ne 2
722 .na
723 \fBnone\fR
724 .ad
725 .sp .6
726 .RS 4n
727 No action is taken on resource requests for an amount that is greater than the
728 threshold. This action is useful for monitoring resource usage without
729 affecting the progress of applications. You can also enable a global message
730 that displays when the resource control is exceeded, while, at the same time,
731 the process exceeding the threshold is not affected.
732 .RE
733 .sp
734 .ne 2
735 .na
736 \fBdeny\fR
737 .ad
738 .sp .6
739 .RS 4n
740 You can deny resource requests for an amount that is greater than the
741 threshold. For example, a \fBtask.max-lwps\fR resource control with action deny
742 causes a \fBfork()\fR system call to fail if the new process would exceed the
743 control value. See the \fBfork(2)\fR.
744 .RE
745 .sp
746 .ne 2
747 .na
748 \fBsignal=\fR
749 .ad
750 .sp .6
751 .RS 4n
752 You can enable a global signal message action when the resource control is
753 exceeded. A signal is sent to the process when the threshold value is exceeded.
754 Additional signals are not sent if the process consumes additional resources.
755 Available signals are listed below.
756 .RE
757 .sp
758 .LP
759 Not all of the actions can be applied to every resource control. For example, a
760 process cannot exceed the number of CPU shares assigned to the project of which
761 it is a member. Therefore, a deny action is not allowed on the
762 \fBproject.cpu-shares\fR resource control.
763 .sp
764 .LP
765 Due to implementation restrictions, the global properties of each control can
766 restrict the range of available actions that can be set on the threshold value.
767 (See \fBrcldm(1M)\fR.) A list of available signal actions is presented in the
768 following list. For additional information about signals, see
769 \fBsignal(3HEAD)\fR.
770 .sp
771 .LP
772 The following are the signals available to resource control values:
773 .sp
774 .ne 2
775 .na
776 \fBSIGABRT\fR
777 .ad
778 .sp .6
779 .RS 4n
780 Terminate the process.
781 .RE
782 .sp
783 .ne 2
784 .na
785 \fBSIGABRT\fR
786 .ad

```

```

787 .ne 2
788 .na
789 \fB\fBSIGHUP\fR\fR
790 .ad
791 .sp .6
792 .RS 4n
793 Send a hangup signal. Occurs when carrier drops on an open line. Signal sent to
794 the process group that controls the terminal.
795 .RE

797 .sp
798 .ne 2
799 .na
800 \fB\fBSIGTERM\fR\fR
801 .ad
802 .sp .6
803 .RS 4n
804 Terminate the process. Termination signal sent by software.
805 .RE

807 .sp
808 .ne 2
809 .na
810 \fB\fBSIGKILL\fR\fR
811 .ad
812 .sp .6
813 .RS 4n
814 Terminate the process and kill the program.
815 .RE

817 .sp
818 .ne 2
819 .na
820 \fB\fBSIGSTOP\fR\fR
821 .ad
822 .sp .6
823 .RS 4n
824 Stop the process. Job control signal.
825 .RE

827 .sp
828 .ne 2
829 .na
830 \fB\fBSIGXRES\fR\fR
831 .ad
832 .sp .6
833 .RS 4n
834 Resource control limit exceeded. Generated by resource control facility.
835 .RE

837 .sp
838 .ne 2
839 .na
840 \fB\fBSIGXFSZ\fR\fR
841 .ad
842 .sp .6
843 .RS 4n
844 Terminate the process. File size limit exceeded. Available only to resource
845 controls with the \fBRCCTL_GLOBAL_FILE_SIZE\fR property
846 (\fBprocess.max-file-size\fR). See \fBrcctlblk_set_value\fR(3C).
847 .RE

849 .sp
850 .ne 2
851 .na
852 \fB\fBSIGXCPU\fR\fR

```

```

853 .ad
854 .sp .6
855 .RS 4n
856 Terminate the process. CPU time limit exceeded. Available only to resource
857 controls with the \fBRCCTL_GLOBAL_CPU_TIME\fR property
858 (\fBprocess.max-cpu-time\fR). See \fBrcctlblk_set_value\fR(3C).
859 .RE

861 .SS "Resource Control Flags and Properties"
862 .sp
863 .LP
864 Each resource control on the system has a certain set of associated properties.
865 This set of properties is defined as a set of flags, which are associated with
866 all controlled instances of that resource. Global flags cannot be modified, but
867 the flags can be retrieved by using either \fBrcctladm\fR(1M) or the
868 \fBsetrcctl\fR(2) system call.
869 .sp
870 .LP
871 Local flags define the default behavior and configuration for a specific
872 threshold value of that resource control on a specific process or process
873 collective. The local flags for one threshold value do not affect the behavior
874 of other defined threshold values for the same resource control. However, the
875 global flags affect the behavior for every value associated with a particular
876 control. Local flags can be modified, within the constraints supplied by their
877 corresponding global flags, by the \fBprctl\fR command or the \fBsetrcctl\fR
878 system call. See \fBsetrcctl\fR(2).
879 .sp
880 .LP
881 For the complete list of local flags, global flags, and their definitions, see
882 \fBrcctlblk_set_value\fR(3C).
883 .sp
884 .LP
885 To determine system behavior when a threshold value for a particular resource
886 control is reached, use \fBrcctladm\fR to display the global flags for the
887 resource control. For example, to display the values for
888 \fBprocess.max-cpu-time\fR, enter:
889 .sp
890 .in +2
891 .nf
892 $ rctladm process.max-cpu-time
893 process.max-cpu-time syslog=off [ lowerable no-deny cpu-time inf seconds ]
894 .fi
895 .in -2

897 .sp
898 .LP
899 The global flags indicate the following:
900 .sp
901 .ne 2
902 .na
903 \fB\fBblowerable\fR\fR
904 .ad
905 .sp .6
906 .RS 4n
907 Superuser privileges are not required to lower the privileged values for this
908 control.
909 .RE

911 .sp
912 .ne 2
913 .na
914 \fB\fBno-deny\fR\fR
915 .ad
916 .sp .6
917 .RS 4n
918 Even when threshold values are exceeded, access to the resource is never

```

```

919 denied.
920 .RE

922 .sp
923 .ne 2
924 .na
925 \fB\fBcpu-time\fR\fR
926 .ad
927 .sp .6
928 .RS 4n
929 \fBSIGXCPU\fR is available to be sent when threshold values of this resource
930 are reached.
931 .RE

933 .sp
934 .ne 2
935 .na
936 \fB\fBseconds\fR\fR
937 .ad
938 .sp .6
939 .RS 4n
940 The time value for the resource control.
941 .RE

943 .sp
944 .LP
945 Use the \fBprctl\fR command to display local values and actions for the
946 resource control. For example:
947 .sp
948 .in +2
949 .nf
950 $ prctl -n process.max-cpu-time $$
951   process 353939: -ksh
952   NAME      PRIVILEGE  VALUE   FLAG  ACTION      RECIPIENT
953   process.max-cpu-time
954   privileged 18.4Es  inf   signal=XCPU  -
955   system    18.4Es  inf   none
956 .fi
957 .in -2

959 .sp
960 .LP
961 The \fBmax\fR (\fBRCCTL_LOCAL_MAXIMAL\fR) flag is set for both threshold values,
962 and the \fBinf\fR (\fBRCCTL_GLOBAL_INFINITE\fR) flag is defined for this
963 resource control. An \fBinf\fR value has an infinite quantity. The value is
964 never enforced. Hence, as configured, both threshold quantities represent
965 infinite values that are never exceeded.
966 .SS "Resource Control Enforcement"
967 .sp
968 .LP
969 More than one resource control can exist on a resource. A resource control can
970 exist at each containment level in the process model. If resource controls are
971 active on the same resource at different container levels, the smallest
972 container's control is enforced first. Thus, action is taken on
973 \fBprocess.max-cpu-time\fR before \fBtask.max-cpu-time\fR if both controls are
974 encountered simultaneously.
975 .SH ATTRIBUTES
976 .sp
977 .LP
978 See \fBattributes\fR(5) for a description of the following attributes:
979 .sp

981 .sp
982 .TS
983 box;
984 c | c

```

```

985 1 | 1 .
986 ATTRIBUTE TYPE  ATTRIBUTE VALUE
987 -
988 Interface Stability    Evolving
989 .TE

991 .SH SEE ALSO
992 .sp
993 .LP
994 \fBprctl\fR(1), \fBpooladm\fR(1M), \fBpoolcfg\fR(1M), \fBprojadd\fR(1M),
995 \fBprojmod\fR(1M), \fBrcctladm\fR(1M), \fBsetrcctl\fR(2),
996 \fBrcctlblk_set_value\fR(3C), \fBlibpool\fR(3LIB), \fBproject\fR(4),
997 \fBattributes\fR(5), \fBFBSS\fR(7)
998 .sp
999 .LP
1000 \fISystem Administration Guide: Virtualization Using the Solaris Operating
1001 System\fR

```


new/usr/src/pkg/manifests/system-test-ostest.mf

1

1279 Fri Jul 19 13:59:26 2013

new/usr/src/pkg/manifests/system-test-ostest.mf

3830 SIGQUEUE_MAX's limit of 32 is too low

```
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 file path=opt/os-tests/README mode=0444
27 file path=opt/os-tests/bin/ostest mode=0555
28 file path=opt/os-tests/runfiles/delphix.run mode=0444
29 file path=opt/os-tests/runfiles/openindiana.run mode=0444
30 file path=opt/os-tests/tests/poll_test mode=0555
31 file path=opt/os-tests/tests/sigqueue_queue_size mode=0555
32 #endif /* ! codereview */
33 license cr_Sun license=cr_Sun
34 license lic_CDDL license=lic_CDDL
35 depend fmri=system/test/testrunner type=require
```

new/usr/src/test/os-tests/tests/Makefile

1

520 Fri Jul 19 13:59:26 2013

new/usr/src/test/os-tests/tests/Makefile

3830 SIGQUEUE_MAX's limit of 32 is too low

```
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 #
11 #
12 #
13 # Copyright (c) 2012 by Delphix. All rights reserved.
14 #
```

```
16 SUBDIRS = poll sigqueue
```

```
16 SUBDIRS = poll
```

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

new/usr/src/test/os-tests/tests/sigqueue/Makefile

1

1015 Fri Jul 19 13:59:27 2013

new/usr/src/test/os-tests/tests/sigqueue/Makefile

3830 SIGQUEUE_MAX's limit of 32 is too low

```
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 #
11 #
12 #
13 # Copyright (c) 2012 by Delphix. All rights reserved.
14 #
15 #
16 include $(SRC)/cmd/Makefile.cmd
17 include $(SRC)/test/Makefile.com
18 #
19 PROG = sigqueue_queue_size
20 OBJS = $(PROG:%=%.o)
21 SRCS = $(OBJS:%.o=%.c)
22 #
23 ROOTOPTPKG = $(ROOT)/opt/os-tests
24 TESTDIR = $(ROOTOPTPKG)/tests
25 #
26 CMDS = $(PROG:%=$(TESTDIR)/%)
27 $(CMDS) := FILEMODE = 0555
28 #
29 all: $(PROG)
30 #
31 $(PROG): $(OBJS)
32     $(LINK.c) $(OBJS) -o $@ $(LDLIBS)
33     $(POST_PROCESS)
34 #
35 %.o: ../%.c
36     $(COMPILE.c) $<
37 #
38 install: all $(CMDS)
39 #
40 lint: lint_SRCS
41 #
42 clobber: clean
43     -$(RM) $(PROG)
44 #
45 clean:
46     -$(RM) $(OBJS)
47 #
48 $(CMDS): $(TESTDIR) $(PROG)
49 #
50 $(TESTDIR):
51     $(INS.dir)
52 #
53 $(TESTDIR)/%: %
54     $(INS.file)
55 #endif /* !codereview */
```

```

*****
3195 Fri Jul 19 13:59:27 2013
new/usr/src/test/os-tests/tests/sigqueue/sigqueue_queue_size.c
3830 SIGQUEUE_MAX's limit of 32 is too low
*****
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 2013 David Hoepfner. All rights reserved.
14 */

16 /*
17  * Queue maximum number of signals and test if we can queue more signals than
18  * allowed.
19 */

21 #include <sys/types.h>
22 #include <stdarg.h>
23 #include <stdio.h>
24 #include <stdlib.h>
25 #include <unistd.h>
26 #include <signal.h>

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

30 int nreceived = 0;

32 static void
33 test_start(const char *test_name, const char *format, ...)
34 {
35     va_list args;

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

39     va_start(args, format);
40     (void) vprintf(format, args);
41     va_end(args);
42     (void) fflush(stdout);
43 }

45 static void
46 test_failed(const char *test_name, const char *format, ...)
47 {
48     va_list args;

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

52     va_start(args, format);
53     (void) vprintf(format, args);
54     va_end(args);

56     (void) exit(-1);
57 }

59 static void
60 test_passed(const char *test_name)
61 {

```

```

62     (void) printf("TEST PASS: %s\n", test_name);
63     (void) fflush(stdout);
64 }

66 static void
67 maximum_test_handler(int signal, siginfo_t *siginfo, void *context)
68 {
69     nreceived++;
70 }

72 static void
73 sigqueue_maximum_test(void)
74 {
75     const char *test_name = __func__;
76     struct sigaction action;
77     long sigqueue_max, i;
78     pid_t pid;
79     union sigval value;
80     int error;

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

84     /*
85      * Get the maximum size of the queue.
86      */
87     sigqueue_max = sysconf(_SC_SIGQUEUE_MAX);
88     if (sigqueue_max == -1) {
89         test_failed(test_name, "sysconf\n");
90     }

92     /*
93      * Put the signal on hold.
94      */
95     error = sighold(SIGQUEUE_SIGNAL);
96     if (error == -1) {
97         test_failed(test_name, "sighold\n");
98     }

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

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

106     error = sigemptyset(&action.sa_mask);
107     if (error == -1) {
108         test_failed(test_name, "sigemptyset\n");
109     }

111     /*
112      * Set signal handler.
113      */
114     error = sigaction(SIGQUEUE_SIGNAL, &action, 0);
115     if (error == -1) {
116         test_failed(test_name, "sigaction\n");
117     }

119     /*
120      * Fill the signal queue to the maximum.
121      */
122     for (i = 0; i < sigqueue_max; i++) {
123         error = sigqueue(pid, SIGQUEUE_SIGNAL, value);
124         if (error == -1) {
125             test_failed(test_name, "sigqueue\n");
126         }
127     }

```

```
129     /*
130     * Send a further signal and test if we get the expected
131     * error.
132     */
133     error = sigqueue(pid, SIGQUEUE_SIGNAL, value);
134     if (error != -1) {
135         test_failed(test_name, "sigqueue\n");
136     }
137
138     /*
139     * Unblock the signals and check if we received all messages
140     * from the signal queue.
141     */
142     sigrelse(SIGQUEUE_SIGNAL);
143
144     if (nreceived != sigqueue_max) {
145         test_failed(test_name, "nreceived != sigqueue_max\n");
146     }
147
148     test_passed(test_name);
149 }
150
151 static void
152 run_tests(void)
153 {
154     sigqueue_maximum_test();
155 }
156
157 int
158 main(int argc, char *argv[])
159 {
160     run_tests();
161
162     exit(EXIT_SUCCESS);
163 }
164 #endif /* ! codereview */
```

```

*****
12538 Fri Jul 19 13:59:27 2013
new/usr/src/uts/common/os/rctl_proc.c
3830 SIGQUEUE_MAX's limit of 32 is too low
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

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

26 #include <sys/types.h>
27 #include <sys/cmn_err.h>
28 #include <sys/sysmacros.h>
29 #include <sys/proc.h>
30 #include <sys/rctl.h>
31 #include <sys/rctl_impl.h>
32 #include <sys/port_kernel.h>
33 #include <sys/signal.h>
34 #endif /* ! codereview */

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

39 /*
40  * Process-based resource controls
41  * The structure of the kernel leaves us no particular place where the process
42  * abstraction can be declared--it is intertwined with the growth of the Unix
43  * kernel. Accordingly, we place all of the resource control logic associated
44  * with processes, both existing and future, in this file.
45  */

47 rctl_hdl_t rctlproc_legacy[RLIM_NLIMITS];
48 uint_t rctlproc_flags[RLIM_NLIMITS] = {
49     RCTL_LOCAL_SIGNAL,          /* RLIMIT_CPU */
50     RCTL_LOCAL_DENY | RCTL_LOCAL_SIGNAL, /* RLIMIT_FSIZE */
51     RCTL_LOCAL_DENY,          /* RLIMIT_DATA */
52     RCTL_LOCAL_DENY,          /* RLIMIT_STACK */
53     RCTL_LOCAL_DENY,          /* RLIMIT_CORE */
54     RCTL_LOCAL_DENY,          /* RLIMIT_NOFILE */
55     RCTL_LOCAL_DENY           /* RLIMIT_VMEM */
56 };
57 int rctlproc_signals[RLIM_NLIMITS] = {
58     SIGXCPU,                   /* RLIMIT_CPU */
59     SIGXFSZ,                   /* RLIMIT_FSIZE */

```

```

60     0, 0, 0, 0, 0           /* remainder do not signal */
61 };

63 rctl_hdl_t rc_process_msgmnb;
64 rctl_hdl_t rc_process_msgtql;
65 rctl_hdl_t rc_process_semml;
66 rctl_hdl_t rc_process_semopm;
67 rctl_hdl_t rc_process_portev;
68 rctl_hdl_t rc_process_sigqueue;
69 #endif /* ! codereview */

71 /*
72  * process.max-cpu-time / RLIMIT_CPU
73  */
74 /*ARGSUSED*/
75 static int
76 proc_cpu_time_test(struct rctl *rctl, struct proc *p, rctl_entity_p_t *e,
77     rctl_val_t *rval, rctl_qty_t inc, uint_t flags)
78 {
79     return (inc >= rval->rcv_value);
80 }

82 static rctl_ops_t proc_cpu_time_ops = {
83     rcop_no_action,
84     rcop_no_usage,
85     rcop_no_set,
86     proc_cpu_time_test
87 };

89 /*
90  * process.max-file-size / RLIMIT_FSIZE
91  */
92 static int
93 proc_filesize_set(rctl_t *rctl, struct proc *p, rctl_entity_p_t *e,
94     rctl_qty_t nv)
95 {
96     if (p->p_model == DATAMODEL_NATIVE)
97         nv = MIN(nv, rctl->rc_dict_entry->rcd_max_native);
98     else
99         nv = MIN(nv, rctl->rc_dict_entry->rcd_max_ilp32);

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

104     return (0);
105 }

107 static rctl_ops_t proc_filesize_ops = {
108     rcop_no_action,
109     rcop_no_usage,
110     proc_filesize_set,
111     rcop_no_test
112 };

114 /*
115  * process.max-data / RLIMIT_DATA
116  */

118 /*
119  * process.max-stack-size / RLIMIT_STACK
120  */
121 static int
122 proc_stack_set(rctl_t *rctl, struct proc *p, rctl_entity_p_t *e,
123     rctl_qty_t nv)
124 {
125     klpw_t *lwp = ttolwp(curthread);

```

```

127     if (p->p_model == DATAMODEL_NATIVE)
128         nv = MIN(nv, rctl->rc_dict_entry->rcd_max_native);
129     else
130         nv = MIN(nv, rctl->rc_dict_entry->rcd_max_ilp32);
131
132     /*
133     * In the process of changing the rlimit, this function actually
134     * gets called a number of times. We only want to save the current
135     * rlimit the first time we come through here. In post_syscall(),
136     * we copyin() the lwp's ustack, and compare it to the rlimit we
137     * save here; if the two match, we adjust the ustack to reflect
138     * the new stack bounds.
139     *
140     * We check to make sure that we're changing the rlimit of our
141     * own process rather than on behalf of some other process. The
142     * notion of changing this resource limit on behalf of another
143     * process is problematic at best, and changing the amount of stack
144     * space a process is allowed to consume is a rather antiquated
145     * notion that has limited applicability in our multithreaded
146     * process model.
147     */
148     ASSERT(e->rcep_t == RCENTITY_PROCESS);
149     if (lwp != NULL && lwp->lwp_proc == e->rcep_p.proc &&
150         lwp->lwp_ustack && lwp->lwp_old_stk_ctl == 0) {
151         lwp->lwp_old_stk_ctl = (size_t)e->rcep_p.proc->p_stk_ctl;
152         curthread->t_post_sys = 1;
153     }
154
155     e->rcep_p.proc->p_stk_ctl = nv;
156
157     return (0);
158 }
159
160 static rctl_ops_t proc_stack_ops = {
161     rcop_no_action,
162     rcop_no_usage,
163     proc_stack_set,
164     rcop_no_test
165 };
166
167 /*
168 * process.max-file-descriptors / RLIMIT_NOFILE
169 */
170 static int
171 proc_nofile_set(rctl_t *rctl, struct proc *p, rctl_entity_p_t *e, rctl_qty_t nv)
172 {
173     ASSERT(e->rcep_t == RCENTITY_PROCESS);
174     if (p->p_model == DATAMODEL_NATIVE)
175         nv = MIN(nv, rctl->rc_dict_entry->rcd_max_native);
176     else
177         nv = MIN(nv, rctl->rc_dict_entry->rcd_max_ilp32);
178
179     e->rcep_p.proc->p_fno_ctl = nv;
180
181     return (0);
182 }
183
184 static rctl_ops_t proc_nofile_ops = {
185     rcop_no_action,
186     rcop_no_usage,
187     proc_nofile_set,
188     rcop_absolute_test
189 };
190
191 /*

```

```

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

```

```

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

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

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

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

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

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

```

```

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

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

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

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

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

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

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

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

```



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

73779 Fri Jul 19 13:59:27 2013

new/usr/src/uts/common/os/sig.c

3830 SIGQUEUE_MAX's limit of 32 is too low

unchanged portion omitted

```

2376 #ifndef UCHAR_MAX
2377 #define UCHAR_MAX      255
2378 #endif

2376 /*
2377  * The pre-allocated pool (with SIGQUEUE_PREALLOC entries) is
2378  * allocated at the first sigqueue/signotify call.
2381  * The entire pool (with maxcount entries) is pre-allocated at
2382  * the first sigqueue/signotify call.
2379  */
2380 sigqhdr_t *
2381 sigqhdralloc(size_t size, uint_t maxcount)
2382 {
2383     size_t i;
2384     sigqueue_t *sq, *next;
2385     sigqhdr_t *sqh;

2387     /*
2388      * Before the introduction of process.max-sigqueue-size
2389      * _SC_SIGQUEUE_MAX had this static value.
2390      */
2391     #define _SIGQUEUE_PREALLOC      32

2393     i = (_SIGQUEUE_PREALLOC * size) + sizeof (sigqhdr_t);
2394     ASSERT(maxcount <= INT_MAX);
2395     i = (maxcount * size) + sizeof (sigqhdr_t);
2396     ASSERT(maxcount <= UCHAR_MAX && i <= USHRT_MAX);
2397     sqh = kmem_alloc(i, KM_SLEEP);
2398     sqh->sqb_count = maxcount;
2399     sqh->sqb_maxcount = maxcount;
2400     sqh->sqb_size = i;
2401     sqh->sqb_count = (uchar_t)maxcount;
2402     sqh->sqb_maxcount = (uchar_t)maxcount;
2403     sqh->sqb_size = (ushort_t)i;
2404     sqh->sqb_pexited = 0;
2405     sqh->sqb_sent = 0;
2406     sqh->sqb_free = sq = (sigqueue_t *) (sqh + 1);
2407     for (i = _SIGQUEUE_PREALLOC - 1; i != 0; i--) {
2408         for (i = maxcount - 1; i != 0; i--) {
2409             next = (sigqueue_t *) ((uintptr_t) sq + size);
2410             sq->sq_next = next;
2411             sq = next;
2412         }
2413         sq->sq_next = NULL;
2414         cv_init(&sqh->sqb_cv, NULL, CV_DEFAULT, NULL);
2415         mutex_init(&sqh->sqb_lock, NULL, MUTEX_DEFAULT, NULL);
2416         return (sqh);
2417     }

```

```

2413 static void sigqrel(sigqueue_t *);

```

```

2415 /*
2416  * Allocate a sigqueue/signotify structure from the per process
2417  * pre-allocated pool or allocate a new sigqueue/signotify structure
2418  * if the pre-allocated pool is exhausted.
2419  * allocate a sigqueue/signotify structure from the per process
2420  * pre-allocated pool.
2419  */
2420 sigqueue_t *

```

```

2421 sigqalloc(sigqhdr_t *sqh)
2422 {
2423     sigqueue_t *sq = NULL;

2425     ASSERT(MUTEX_HELD(&curproc->p_lock));

2427     if (sqh != NULL) {
2428         mutex_enter(&sqh->sqb_lock);
2429         if (sqh->sqb_count > 0) {
2430             sqh->sqb_count--;
2431             if (sqh->sqb_free == NULL) {
2432                 /*
2433                  * The pre-allocated pool is exhausted.
2434                  */
2435                 sq = kmem_alloc(sizeof (sigqueue_t), KM_SLEEP);
2436                 sq->sq_func = NULL;
2437             } else {
2438                 #endif /* ! codereview */
2439                 sq = sqh->sqb_free;
2440                 sq->sq_func = sigqrel;
2441             #endif /* ! codereview */
2442             sqh->sqb_free = sq->sq_next;
2443         }
2444         #endif /* ! codereview */
2445         mutex_exit(&sqh->sqb_lock);
2446         bzero(&sq->sq_info, sizeof (k_siginfo_t));
2447         sq->sq_backptr = sqh;
2448         sq->sq_func = sigqrel;
2449         sq->sq_next = NULL;
2450         sq->sq_external = 0;
2451     } else {
2452         mutex_exit(&sqh->sqb_lock);
2453     }
2454     return (sq);
2455 }

```

unchanged portion omitted

10030 Fri Jul 19 13:59:28 2013

new/usr/src/uts/common/sys/signal.h

3830 SIGQUEUE_MAX's limit of 32 is too low

_____unchanged_portion_omitted_____

```

303 typedef struct sigqhdr {
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 */
308     uchar_t       sqb_count;      /* sigq free count */
309     uchar_t       sqb_maxcount;   /* sigq max free count */
310     ushort_t      sqb_size;      /* size of header+free structs */
311     uchar_t       sqb_pexited;    /* process has exited */
312     uint_t        sqb_sent;      /* number of sigq sent */
313     uchar_t       sqb_sent;      /* number of sigq sent */
314     kcondvar_t    sqb_cv;        /* waiting for a sigq struct */
315     kmutex_t      sqb_lock;      /* lock for sigq pool */
316 } sigqhdr_t;

```

```

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

```

```

315 #define _SIGQUEUE_MAX      32
318 #define _SIGNOTIFY_MAX    32

```

```

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

```

```

328 extern int    kill(pid_t, int);

```

```

330 #endif /* _KERNEL */

```

```

332 #ifdef __cplusplus
333 }

```

_____unchanged_portion_omitted_____

```

*****
5578 Fri Jul 19 13:59:29 2013
new/usr/src/uts/common/syscall/sigqueue.c
3830 SIGQUEUE_MAX's limit of 32 is too low
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25  */

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

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

29 #include <sys/param.h>
30 #include <sys/types.h>
31 #include <sys/sysmacros.h>
32 #include <sys/system.h>
33 #include <sys/errno.h>
34 #include <sys/proc.h>
35 #include <sys/procset.h>
36 #include <sys/fault.h>
37 #include <sys/signal.h>
38 #include <sys/siginfo.h>
39 #include <sys/debug.h>

41 extern rctl_hdl_t rc_process_sigqueue;

43 #endif /* ! codereview */
44 static int
45 sigqkill(pid_t pid, sigsend_t *sigsend)
46 {
47     proc_t *p;
48     int error;

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

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

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

```

```

60         if ((p = prfind(pid)) == NULL || p->p_stat == SIDL)
61             error = ESRCH;
62         else {
63             error = sigsendproc(p, sigsend);
64             if (error == 0 && sigsend->perm == 0)
65                 error = EPERM;
66         }
67         mutex_exit(&pidlock);
68     } else {
69         int nfound = 0;
70         pid_t pgid;

72         if (pid == 0)
73             pgid = ttproc(curthread)->p_pgrp;
74         else
75             pgid = -pid;

77         error = 0;
78         mutex_enter(&pidlock);
79         for (p = pgfind(pgid); p && !error; p = p->p_pglink) {
80             if (p->p_stat != SIDL) {
81                 nfound++;
82                 error = sigsendproc(p, sigsend);
83             }
84         }
85         mutex_exit(&pidlock);
86         if (nfound == 0)
87             error = ESRCH;
88         else if (error == 0 && sigsend->perm == 0)
89             error = EPERM;
90     }

92     return (error);
93 }

96 /*
97  * for implementations that don't require binary compatibility,
98  * the kill system call may be made into a library call to the
99  * sigsend system call
100 */
101 int
102 kill(pid_t pid, int sig)
103 {
104     int error;
105     sigsend_t v;

107     bzero(&v, sizeof (v));
108     v.sig = sig;
109     v.checkperm = 1;
110     v.sicode = SI_USER;
111     if ((error = sigqkill(pid, &v)) != 0)
112         return (set_errno(error));
113     return (0);
114 }

116 /*
117  * The handling of small unions, like the sigval argument to sigqueue,
118  * is architecture dependent. We have adopted the convention that the
119  * value itself is passed in the storage which crosses the kernel
120  * protection boundary. This procedure will accept a scalar argument,
121  * and store it in the appropriate value member of the sigsend_t structure.
122 */
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);
147         sqh = sigqhdralloc(sizeof (sigqueue_t), _SIGQUEUE_MAX);
148         mutex_enter(&p->p_lock);
149         if (p->p_sigqhdr == NULL) {
150             /* hang the pool head on proc */
151             p->p_sigqhdr = sqh;
152         } else {
153             /* another lwp allocated the pool, free ours */
154             sigqhdrfree(sqh);
155             sqh = p->p_sigqhdr;
156         }
157         mutex_exit(&p->p_lock);

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 Fri Jul 19 13:59:29 2013
new/usr/src/uts/common/syscall/sysconfig.c
3830 SIGQUEUE_MAX's limit of 32 is too low
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright 2008 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25  */

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

30 #include <sys/param.h>
31 #include <sys/types.h>
32 #include <sys/sysmacros.h>
33 #include <sys/system.h>
34 #include <sys/tuneable.h>
35 #include <sys/errno.h>
36 #include <sys/var.h>
37 #include <sys/signal.h>
38 #include <sys/time.h>
39 #include <sys/sysconfig.h>
40 #include <sys/resource.h>
41 #include <sys/ulimit.h>
42 #include <sys/unistd.h>
43 #include <sys/debug.h>
44 #include <sys/cpuvar.h>
45 #include <sys/mman.h>
46 #include <sys/timer.h>
47 #include <sys/zone.h>
48 #include <sys/vm_usage.h>

50 extern rctl_hdl_t rc_process_sigqueue;

52 #endif /* ! codereview */
53 long
54 sysconfig(int which)
55 {
56     switch (which) {

58     /*
59      * if it is not handled in mach_sysconfig either
60      * it must be EINVAL.
61      */

```

```

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

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

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

71     case _CONFIG_NGROUPS:
72         /*
73          * Maximum number of supplementary groups.
74          */
75         return (ngroups_max);

77     case _CONFIG_OPEN_FILES:
78         /*
79          * Maximum number of open files (soft limit).
80          */
81         {
82             rlim64_t fd_ctl;
83             mutex_enter(&curproc->p_lock);
84             fd_ctl = rctl_enforced_value(
85                 rctlproc_legacy[RLIMIT_NOFILE], curproc->p_rctls,
86                 curproc);
87             mutex_exit(&curproc->p_lock);
88             return ((ulong_t)fd_ctl);
89         }

91     case _CONFIG_CHILD_MAX:
92         /*
93          * Maximum number of processes.
94          */
95         return (v.v_maxup);

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

100    case _CONFIG_PAGESIZE:
101        return (PAGESIZE);

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

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

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

112    case _CONFIG_NPROC_MAX:
113        return (max_ncpus);

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

118    case _CONFIG_AIO_LISTIO_MAX:
119        return (_AIO_LISTIO_MAX);

121    case _CONFIG_AIO_MAX:
122        return (_AIO_MAX);

124    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         }
157     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_SYMLINK_MAX:
224         return (MAXSYMLINKS);
225     }
226 }

```

unchanged portion omitted