
21226 Thu Sep 26 12:53:19 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 <iszczeniak@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\"
7  .TH RESOURCE_CONTROLS 5 \"Jul 2, 2007\"
8  .SH NAME
9  resource_controls - resource controls available through project database
10 .SH DESCRIPTION
11 .sp
12 .LP
13 The resource controls facility is configured through the project database. See
14 \\fbproject\\fR(4). You can set and modify resource controls through the
15 following utilities:
16 .RS +4
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 \\fbrcctladm\\fR(1M)
38 .RE
39 .sp
40 .LP
41 In a program, you use \\fbsetrcctl\\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 \fBFS\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

```

```

256 .ad
257 .sp .6
258 .RS 4n
259 Maximum number of LWP's 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 LWP's 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) by \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 \fBbrctladm\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=\fR\fIlevel\fR. The possible settings for \fIlevel\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 \fB\fBnone\fR\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 .ad
737 \fB\fBdeny\fR\fR
738 .ad
739 .sp .6
740 .RS 4n
741 You can deny resource requests for an amount that is greater than the
742 threshold. For example, a \fBtask.max-lwps\fR resource control with action deny
743 causes a \fBfork()\fR system call to fail if the new process would exceed the
744 control value. See the \fBfork\fR(2).
745 .RE
746 .sp
747 .ne 2
748 .na
749 .ad
750 \fB\fBsignal=\fR\fR
751 .ad
752 .sp .6
753 .RS 4n
754 You can enable a global signal message action when the resource control is
755 exceeded. A signal is sent to the process when the threshold value is exceeded.
756 Additional signals are not sent if the process consumes additional resources.
757 Available signals are listed below.
758 .RE
759 .sp
760 .LP
761 .LP
762 Not all of the actions can be applied to every resource control. For example, a
763 process cannot exceed the number of CPU shares assigned to the project of which
764 it is a member. Therefore, a deny action is not allowed on the
765 \fBproject.cpu-shares\fR resource control.
766 .sp
767 .LP
768 Due to implementation restrictions, the global properties of each control can
769 restrict the range of available actions that can be set on the threshold value.
770 (See \fBbrctladm\fR(1M).) A list of available signal actions is presented in the
771 following list. For additional information about signals, see
772 \fBsignal\fR(3HEAD).
773 .sp
774 .LP
775 The following are the signals available to resource control values:
776 .sp
777 .ne 2
778 .na
779 \fB\fBSIGABRT\fR\fR
780 .ad
781 .sp .6
782 .RS 4n
783 Terminate the process.

```

```

784 .RE

786 .sp
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 \fB\RCTL_GLOBAL_FILE_SIZE\fR property
846 (\fBprocess.max-file-size\fR). See \fBBrctlblk_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 \fB\RCTL_GLOBAL_CPU_TIME\fR property
858 (\fBprocess.max-cpu-time\fR). See \fBBrctlblk_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 \fBBrctladm\fR(1M) or the
868 \fBsetrctl\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 \fBBrctl\fR command or the \fBsetrctl\fR
878 system call. See \fBsetrctl\fR(2).
879 .sp
880 .LP
881 For the complete list of local flags, global flags, and their definitions, see
882 \fBBrctlblk_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 \fBBrctladm\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\fBlowerable\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 l | l
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), \fBbrctladm\fR(1M), \fBsetrctl\fR(2),
996 \fBbrctlblk_set_value\fR(3C), \fBlibpool\fR(3LIB), \fBproject\fR(4),
997 \fBattributes\fR(5), \fBPFSS\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

1325 Thu Sep 26 12:53:20 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 <iszczeniak@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

new/usr/src/test/os-tests/runfiles/delphix.run

1

671 Thu Sep 26 12:53:21 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 <iszczeniak@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 [DEFAULT]

17 pre =

18 verbose = False

19 quiet = False

20 user = root

20 timeout = 60

21 post =

22 outputdir = /var/tmp/test_results

24 [/opt/os-tests/tests/poll_test]

25 user = root

26 #endif /* ! codereview */

28 [/opt/os-tests/tests/sigqueue]

29 tests = ['sigqueue_queue_size']

30 #endif /* ! codereview */

new/usr/src/test/os-tests/runfiles/openindiana.run

1

671 Thu Sep 26 12:53:21 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 <iszczeniak@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 [DEFAULT]

17 pre =

18 verbose = False

19 quiet = False

20 user = root

20 timeout = 60

21 post =

22 outputdir = /var/tmp/test_results

24 [/opt/os-tests/tests/poll_test]

25 user = root

26 #endif /* ! codereview */

28 [/opt/os-tests/tests/sigqueue]

29 tests = ['sigqueue_queue_size']

30 #endif /* ! codereview */

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

1

520 Thu Sep 26 12:53:21 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 <iszczeniak@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

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

1

1104 Thu Sep 26 12:53:22 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 <iszczeniak@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 #
11 #
12 #
13 # Copyright (c) 2012 by Delphix. All rights reserved.
14 # Copyright (c) 2013 David Hoeppner. All rights reserved.
15 #
16 #
17 include $(SRC)/cmd/Makefile.cmd
18 include $(SRC)/test/Makefile.com
19 #
20 PROG = sigqueue_queue_size
21 OBJS = $(PROG:%=.o)
22 SRCS = $(OBJS:%=.c)
23 #
24 C99MODE = -xc99=%all
25 #
26 ROOTOPTPKG = $(ROOT)/opt/os-tests
27 TESTDIR = $(ROOTOPTPKG)/tests/sigqueue
28 #
29 CMDS = $(PROG:%=$(TESTDIR)/%)
30 $(CMDS) := FILEMODE = 0555
31 #
32 all: $(PROG)
33 #
34 $(PROG): $(OBJS)
35     $(LINK.c) $(OBJS) -o $@ $(LDLIBS)
36     $(POST_PROCESS)
37 #
38 %.o: ../%.c
39     $(COMPILE.c) $<
40 #
41 install: all $(CMDS)
42 #
43 lint: lint_SRCS
44 #
45 clobber: clean
46     -$(RM) $(PROG)
47 #
48 clean:
49     -$(RM) $(OBJS)
50 #
51 $(CMDS): $(TESTDIR) $(PROG)
52 #
53 $(TESTDIR):
54     $(INS.dir)
55 #
56 $(TESTDIR)/%: %
57     $(INS.file)
58 #endif /* ! codereview */
```

new/usr/src/test/os-tests/tests/sigqueue/sigqueue_queue_size.c

1

```
*****
3257 Thu Sep 26 12:53:22 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 <iszczeniak@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 2013 David Hoeppner. All rights reserved.
14 */

16 /*
17  * Queue maximum number of signals and test if we can queue more signals then
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 }
```

new/usr/src/test/os-tests/tests/sigqueue/sigqueue_queue_size.c

2

```
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 /* ARGSUSED */
67 static void
68 maximum_test_handler(int signal, siginfo_t *siginfo, void *context)
69 {
70     nreceived++;
71 }

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

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

85     /*
86      * Get the maximum size of the queue.
87      */
88     sigqueue_max = sysconf(_SC_SIGQUEUE_MAX);
89     if (sigqueue_max == -1) {
90         test_failed(test_name, "sysconf\n");
91     }

93     /*
94      * Put the signal on hold.
95      */
96     error = sighold(SIGQUEUE_SIGNAL);
97     if (error == -1) {
98         test_failed(test_name, "sighold\n");
99     }

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

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

107     error = sigemptyset(&action.sa_mask);
108     if (error == -1) {
109         test_failed(test_name, "sigemptyset\n");
110     }

112     /*
113      * Set signal handler.
114      */
115     error = sigaction(SIGQUEUE_SIGNAL, &action, 0);
116     if (error == -1) {
117         test_failed(test_name, "sigaction\n");
118     }

120     /*
121      * Fill the signal queue to the maximum.
122      */
123     for (i = 0; i < sigqueue_max; i++) {
124         error = sigqueue(pid, SIGQUEUE_SIGNAL, value);
```

```
125         if (error == -1) {
126             test_failed(test_name, "sigqueue\n");
127         }
128     }
129
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     }
138
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     }
147
148     if (nreceived != sigqueue_max) {
149         test_failed(test_name, "nreceived != sigqueue_max\n");
150     }
151
152     test_passed(test_name);
153 }
154
155 static void
156 run_tests(void)
157 {
158     sigqueue_maximum_test();
159 }
160
161 /* ARGSUSED */
162 int
163 main(int argc, char *argv[])
164 {
165     run_tests();
166
167     return (EXIT_SUCCESS);
168 }
169 #endif /* ! codereview */
```

new/usr/src/test/test-runner/cmd/run.py

1

39414 Thu Sep 26 12:53:22 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 <iszczeniak@gmail.com>

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

3 #
4 # This file and its contents are supplied under the terms of the
5 # Common Development and Distribution License ("CDDL"), version 1.0.
6 # You may only use this file in accordance with the terms of version
7 # 1.0 of the CDDL.
8 #
9 # A full copy of the text of the CDDL should have accompanied this
10 # source. A copy of the CDDL is also available via the Internet at
11 # http://www.illumos.org/license/CDDL.
12 #

14 #
15 # Copyright (c) 2012 by Delphix. All rights reserved.
16 #

18 import ConfigParser
19 import os
20 import logging
21 from datetime import datetime
22 from optparse import OptionParser
23 from pwd import getpwnam
24 from pwd import getpwuid
25 from select import select
26 from subprocess import PIPE
27 from subprocess import Popen
28 from sys import argv
29 from sys import exit
30 from threading import Timer
31 from time import time

33 BASEDIR = '/var/tmp/test_results'
34 KILL = '/usr/bin/kill'
35 TRUE = '/usr/bin/true'
36 SUDO = '/usr/bin/sudo'

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

43     def __init__(self):
44         self.starttime = None
45         self.returncode = None
46         self.runtime = ''
47         self.stdout = []
48         self.stderr = []
49         self.result = ''

51     def done(self, proc, killed):
52         """
53         Finalize the results of this Cmd.
54         """
55         Result.total += 1
56         m, s = divmod(time() - self.starttime, 60)
57         self.runtime = '%02d:%02d' % (m, s)
58         self.returncode = proc.returncode
```

new/usr/src/test/test-runner/cmd/run.py

2

```
59     if killed:
60         self.result = 'KILLED'
61         Result.runresults['KILLED'] += 1
62     elif self.returncode is 0:
63         self.result = 'PASS'
64         Result.runresults['PASS'] += 1
65     elif self.returncode is not 0:
66         self.result = 'FAIL'
67         Result.runresults['FAIL'] += 1

70 class Output(object):
71     """
72     This class is a slightly modified version of the 'Stream' class found
73     here: http://goo.gl/aSgfv
74     """
75     def __init__(self, stream):
76         self.stream = stream
77         self._buf = ''
78         self.lines = []

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

83     def read(self, drain=0):
84         """
85         Read from the file descriptor. If 'drain' set, read until EOF.
86         """
87         while self._read() is not None:
88             if not drain:
89                 break

91     def _read(self):
92         """
93         Read up to 4k of data from this output stream. Collect the output
94         up to the last newline, and append it to any leftover data from a
95         previous call. The lines are stored as a (timestamp, data) tuple
96         for easy sorting/merging later.
97         """
98         fd = self.fileno()
99         buf = os.read(fd, 4096)
100         if not buf:
101             return None
102         if '\n' not in buf:
103             self._buf += buf
104             return []

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

114 class Cmd(object):
115     verified_users = []

117     def __init__(self, pathname, outputdir=None, timeout=None, user=None):
118         self.pathname = pathname
119         self.outputdir = outputdir or 'BASEDIR'
120         self.timeout = timeout or 60
121         self.user = user or ''
122         self.killed = False
123         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)

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'

138     cmd = [SUDO, KILL, signal, str(proc.pid)]
139     if not do_sudo:
140         del cmd[0]

142     try:
143         kp = Popen(cmd)
144         kp.wait()
145     except:
146         pass

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())

156     if not user or user is me:
157         return cmd

159     ret = '%s -E -u %s %s' % (SUDO, user, cmd)
160     return ret.split(' ')

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)

179     return out.lines, err.lines

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)

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()

212     self.result.done(proc, self.killed)

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'

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

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)))

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))

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))

256     if len(self.result.stdout):

```



```

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("Couldn'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 = '/' + testfile.split('/')[components:].rstrip('/')
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)
718         logger.info("Warning: user '%s' cannot use passwordless 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:])

```

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

1

```
*****
12757 Thu Sep 26 12:53:23 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 <iszczeniak@gmail.com>
*****

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 #include <sys/var.h>
35 #endif /* ! codereview */

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

40 /*
41  * Process-based resource controls
42  * The structure of the kernel leaves us no particular place where the process
43  * abstraction can be declared--it is intertwined with the growth of the Unix
44  * kernel. Accordingly, we place all of the resource control logic associated
45  * with processes, both existing and future, in this file.
46 */

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

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

2

```
57 };
58 int rctlproc_signals[RLIM_NLIMITS] = {
59     SIGXCPU,          /* RLIMIT_CPU */
60     SIGXFSZ,          /* RLIMIT_FSIZE */
61     0, 0, 0, 0, 0    /* remainder do not signal */
62 };

64 rctl_hdl_t rc_process_msgmb;
65 rctl_hdl_t rc_process_msgtql;
66 rctl_hdl_t rc_process_semmsl;
67 rctl_hdl_t rc_process_semopm;
68 rctl_hdl_t rc_process_portev;
69 rctl_hdl_t rc_process_sigqueue;
70 #endif /* ! codereview */

72 /*
73  * process.max-cpu-time / RLIMIT_CPU
74 */
75 /* ARGSUSED */
76 static int
77 proc_cpu_time_test(struct rctl *rctl, struct proc *p, rctl_entity_p_t *e,
78     rctl_val_t *rval, rctl_qty_t inc, uint_t flags)
79 {
80     return (inc >= rval->rcv_value);
81 }

83 static rctl_ops_t proc_cpu_time_ops = {
84     rcop_no_action,
85     rcop_no_usage,
86     rcop_no_set,
87     proc_cpu_time_test
88 };

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

102     ASSERT(e->rcep_t == RENTITY_PROCESS);
103     e->rcep_p.proc->p_fsz_ctl = nv;

105     return (0);
106 }

108 static rctl_ops_t proc_filesize_ops = {
109     rcop_no_action,
110     rcop_no_usage,
111     proc_filesize_set,
112     rcop_no_test
113 };

115 /*
116  * process.max-data / RLIMIT_DATA
117 */

119 /*
120  * process.max-stack-size / RLIMIT_STACK
121 */
122 static int
```

```

123 proc_stack_set(rctl_t *rctl, struct proc *p, rctl_entity_p_t *e,
124   rctl_qty_t nv)
125 {
126     klpw_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;

```

```

255 #else
256     rlp64.rlim_cur = DFLSSIZ;
257     rlp64.rlim_max = MAXSSIZ;
258 #endif
259     (void) rctl_rlimit_set(rctlproc_legacy[RLIMIT_STACK], initp, &rlp64, gp,
260         RCTL_LOCAL_DENY, 0, kcred);
261
262     /*
263      * RLIMIT_CORE: deny always, no default action.
264      */
265     rlp64.rlim_cur = rlp64.rlim_max = RLIM64_INFINITY;
266     (void) rctl_rlimit_set(rctlproc_legacy[RLIMIT_CORE], initp, &rlp64, gp,
267         RCTL_LOCAL_DENY, 0, kcred);
268
269     /*
270      * RLIMIT_NOFILE: deny always, no action.
271      */
272     rlp64.rlim_cur = rlim_fd_cur;
273     rlp64.rlim_max = rlim_fd_max;
274     (void) rctl_rlimit_set(rctlproc_legacy[RLIMIT_NOFILE], initp, &rlp64,
275         gp, RCTL_LOCAL_DENY, 0, kcred);
276
277     /*
278      * RLIMIT_VMEM
279      */
280     rlp64.rlim_cur = rlp64.rlim_max = RLIM64_INFINITY;
281     (void) rctl_rlimit_set(rctlproc_legacy[RLIMIT_VMEM], initp, &rlp64, gp,
282         RCTL_LOCAL_DENY, 0, kcred);
283 }
284
285 /*
286  * void rctlproc_init()
287  */
288 * Overview
289 * Register the various resource controls associated with process entities.
290 * The historical rlim_infinity_map and rlim_infinity32_map are now encoded
291 * here as the native and ILP32 infinite values for each resource control.
292 */
293 void
294 rctlproc_init(void)
295 {
296     rctl_set_t *set;
297     rctl_alloc_gp_t *gp;
298     rctl_entity_p_t e;
299
300     rctlproc_legacy[RLIMIT_CPU] = rctl_register("process.max-cpu-time",
301         RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_NEVER |
302         RCTL_GLOBAL_CPU_TIME | RCTL_GLOBAL_INFINITE | RCTL_GLOBAL_SECONDS,
303         UINT64_MAX, UINT64_MAX, &proc_cpu_time_ops);
304     rctlproc_legacy[RLIMIT_FSIZE] = rctl_register("process.max-file-size",
305         RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
306         RCTL_GLOBAL_FILE_SIZE | RCTL_GLOBAL_BYTES,
307         MAXOFFSET_T, MAXOFFSET_T, &proc_filesize_ops);
308     rctlproc_legacy[RLIMIT_DATA] = rctl_register("process.max-data-size",
309         RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
310         RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
311         ULONG_MAX, UINT32_MAX, &rctl_default_ops);
312 #ifdef _LP64
313 #ifdef __sparc
314     rctlproc_legacy[RLIMIT_STACK] = rctl_register("process.max-stack-size",
315         RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
316         RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
317         LONG_MAX, INT32_MAX, &proc_stack_ops);
318 #else
319     rctlproc_legacy[RLIMIT_STACK] = rctl_register("process.max-stack-size",

```

```

320         RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
321         RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
322         MAXSSIZ, USRSTACK32 - PAGE_SIZE, &proc_stack_ops);
323 #endif /* __sparc */
324 #else /* _LP64 */
325     rctlproc_legacy[RLIMIT_STACK] = rctl_register("process.max-stack-size",
326         RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
327         RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
328         USRSTACK - PAGE_SIZE, USRSTACK - PAGE_SIZE, &proc_stack_ops);
329 #endif
330     rctlproc_legacy[RLIMIT_CORE] = rctl_register("process.max-core-size",
331         RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
332         RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
333         MIN(MAXOFFSET_T, ULONG_MAX), UINT32_MAX, &rctl_default_ops);
334     rctlproc_legacy[RLIMIT_NOFILE] = rctl_register(
335         "process.max-file-descriptor", RCENTITY_PROCESS,
336         RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
337         RCTL_GLOBAL_COUNT, INT32_MAX, INT32_MAX, &proc_nofile_ops);
338     rctlproc_legacy[RLIMIT_VMEM] =
339         rctl_register("process.max-address-space", RCENTITY_PROCESS,
340         RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
341         RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
342         ULONG_MAX, UINT32_MAX, &proc_vmem_ops);
343
344     rc_process_semmsl = rctl_register("process.max-sem-nsems",
345         RCENTITY_PROCESS, RCTL_GLOBAL_DENY_ALWAYS | RCTL_GLOBAL_COUNT,
346         SHRT_MAX, SHRT_MAX, &rctl_absolute_ops);
347     rctl_add_legacy_limit("process.max-sem-nsems", "semsys",
348         "seminfo_semmsl", 512, SHRT_MAX);
349
350     rc_process_semopm = rctl_register("process.max-sem-ops",
351         RCENTITY_PROCESS, RCTL_GLOBAL_DENY_ALWAYS | RCTL_GLOBAL_COUNT,
352         INT_MAX, INT_MAX, &rctl_absolute_ops);
353     rctl_add_legacy_limit("process.max-sem-ops", "semsys",
354         "seminfo_semopm", 512, INT_MAX);
355
356     rc_process_msgmb = rctl_register("process.max-msg-qbytes",
357         RCENTITY_PROCESS, RCTL_GLOBAL_DENY_ALWAYS | RCTL_GLOBAL_BYTES,
358         ULONG_MAX, ULONG_MAX, &rctl_absolute_ops);
359     rctl_add_legacy_limit("process.max-msg-qbytes", "msgsys",
360         "msginfo_msgmb", 65536, ULONG_MAX);
361
362     rc_process_msgttl = rctl_register("process.max-msg-messages",
363         RCENTITY_PROCESS, RCTL_GLOBAL_DENY_ALWAYS | RCTL_GLOBAL_COUNT,
364         UINT_MAX, UINT_MAX, &rctl_absolute_ops);
365     rctl_add_legacy_limit("process.max-msg-messages", "msgsys",
366         "msginfo_msgttl", 8192, UINT_MAX);
367
368     rc_process_portev = rctl_register("process.max-port-events",
369         RCENTITY_PROCESS, RCTL_GLOBAL_DENY_ALWAYS | RCTL_GLOBAL_COUNT,
370         PORT_MAX_EVENTS, PORT_MAX_EVENTS, &rctl_absolute_ops);
371     rctl_add_legacy_limit("process.max-port-events", "PORT_DEFAULT_EVENTS",
372         RCPRIV_PRIVILEGED, RCTL_LOCAL_DENY);
373
374     /*
375      * We set the upper limit to the maximum number of processes per user
376      * to make it theoretical possible to deliver all SIGCHLD signals on
377      * child termination, but at least to 8k.
378      */
379     rc_process_sigqueue = rctl_register("process.max-sigqueue-size",
380         RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
381         RCTL_GLOBAL_COUNT, MAX(v.v_maxup, 8192), MAX(v.v_maxup, 8192),
382         &rctl_absolute_ops);
383     rctl_add_default_limit("process.max-sigqueue-size",
384         SIGQUEUE_SIZE_BASIC, RCPRIV_BASIC, RCTL_LOCAL_DENY);
385     rctl_add_default_limit("process.max-sigqueue-size",

```

```
386     _SIGQUEUE_SIZE_PRIVILEGED, RCPRIV_PRIVILEGED, RCTL_LOCAL_DENY);
388
389 /* ! codereview */
390 #endif
391 * Place minimal set of controls on "sched" process for inheritance by
392 * processes created via newproc().
393 */
394 set = rctl_set_create();
395 gp = rctl_set_init_prealloc(RCENTITY_PROCESS);
396 mutex_enter(&curproc->p_lock);
397 e.rcep_p.proc = curproc;
398 e.rcep_t = RCENTITY_PROCESS;
399 curproc->p_rctls = rctl_set_init(RCENTITY_PROCESS, curproc, &e,
400     set, gp);
401 mutex_exit(&curproc->p_lock);
402 rctl_prealloc_destroy(gp);
403 }
```

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

1

73723 Thu Sep 26 12:53:23 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 <iszczeniak@gmail.com>

_____unchanged_portion_omitted_____

2374 #ifndef UCHAR_MAX
2375 #define UCHAR_MAX 255
2376 #endif

2374 /*
2375 * The pre-allocated pool (with SIGQUEUE_PREALLOC entries) is
2376 * allocated at the first sigqueue/signotify call.
2379 * The entire pool (with maxcount entries) is pre-allocated at
2380 * the first sigqueue/signotify call.
2377 */
2378 sigqhdr_t *
2379 sigqhdralloc(size_t size, uint_t maxcount)
2380 {
2381 size_t i;
2382 sigqueue_t *sq, *next;
2383 sigqhdr_t *sqh;

2385 /*
2386 * Before the introduction of process.max-sigqueue-size
2387 * _SC_SIGQUEUE_MAX had this static value.
2388 */
2389 #define SIGQUEUE_PREALLOC 32

2391 i = (SIGQUEUE_PREALLOC * size) + sizeof (sigqhdr_t);
2392 ASSERT(maxcount <= INT_MAX);
2393 i = (maxcount * size) + sizeof (sigqhdr_t);
2394 ASSERT(maxcount <= UCHAR_MAX && i <= USHRT_MAX);
2395 sqh = kmem_alloc(i, KM_SLEEP);
2396 sqh->sqb_count = maxcount;
2397 sqh->sqb_maxcount = maxcount;
2398 sqh->sqb_size = i;
2399 sqh->sqb_count = (uchar_t)maxcount;
2400 sqh->sqb_maxcount = (uchar_t)maxcount;
2401 sqh->sqb_size = (ushort_t)i;
2402 sqh->sqb_pexited = 0;
2403 sqh->sqb_sent = 0;
2404 sqh->sqb_free = sq = (sigqueue_t *) (sqh + 1);
2405 for (i = SIGQUEUE_PREALLOC - 1; i != 0; i--) {
2406 for (i = maxcount - 1; i != 0; i--) {
2407 next = (sigqueue_t *) ((uintptr_t) sq + size);
2408 sq->sq_next = next;
2409 sq = next;
2410 }
2411 }
2412 sq->sq_next = NULL;
2413 cv_init(&sqh->sqb_cv, NULL, CV_DEFAULT, NULL);
2414 mutex_init(&sqh->sqb_lock, NULL, MUTEX_DEFAULT, NULL);
2415 return (sqh);
2416 }

2411 static void sigqrel(sigqueue_t *);

2413 /*
2414 * Allocate a sigqueue/signotify structure from the per process
2415 * pre-allocated pool or allocate a new sigqueue/signotify structure
2416 * if the pre-allocated pool is exhausted.
2417 * allocate a sigqueue/signotify structure from the per process

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

2

2413 * pre-allocated pool.
2417 */
2418 sigqueue_t *
2419 sigqalloc(sigqhdr_t *sqh)
2420 {
2421 sigqueue_t *sq = NULL;

2423 ASSERT(MUTEX_HELD(&curproc->p_lock));

2425 if (sqh != NULL) {
2426 mutex_enter(&sqh->sqb_lock);
2427 if (sqh->sqb_count > 0) {
2428 sqh->sqb_count--;
2429 if (sqh->sqb_free == NULL) {
2430 /*
2431 * The pre-allocated pool is exhausted.
2432 */
2433 sq = kmem_alloc(sizeof (sigqueue_t), KM_SLEEP);
2434 sq->sq_func = NULL;
2435 } else {
2436 #endif /* ! codereview */
2437 sq = sqh->sqb_free;
2438 sq->sq_func = sigqrel;
2439 #endif /* ! codereview */
2440 sqh->sqb_free = sq->sq_next;
2441 }
2442 #endif /* ! codereview */
2443 mutex_exit(&sqh->sqb_lock);
2444 bzero(&sq->sq_info, sizeof (k_siginfo_t));
2445 sq->sq_backptr = sqh;
2446 sq->sq_func = sigqrel;
2447 sq->sq_next = NULL;
2448 sq->sq_external = 0;
2449 } else {
2450 mutex_exit(&sqh->sqb_lock);
2451 }
2452 }
2453 return (sq);
2454 }

_____unchanged_portion_omitted_____

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

1

```
*****
9977 Thu Sep 26 12:53:24 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 <iszczeniak@gmail.com>
*****
_____unchanged_portion_omitted_____
```

```
303 typedef struct sigqhdr {
304     sigqueue_t    *sqb_free;      /* sigqueue pool header */
305     int           sqb_count;      /* free sigq struct list */
306     uint_t        sqb_maxcount;   /* sigq free count */
307     size_t        sqb_size;      /* sigq max free count */
308     uchar_t       sqb_count;     /* size of header+free structs */
309     uchar_t       sqb_maxcount;  /* size of header+free structs */
310     ushort_t      sqb_size;      /* sigq free count */
311     uchar_t       sqb_pexited;   /* sigq max free count */
312     uint_t        sqb_sent;      /* size of header+free structs */
313     uchar_t       sqb_sent;      /* process has exited */
314     kcondvar_t    sqb_cv;        /* number of sigq sent */
315     kmutex_t      sqb_lock;      /* number of sigq sent */
316 } 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 12:53:24 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 <iszczeniak@gmail.com>
*****

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"

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

42 extern rctl_hdl_t rc_process_sigqueue;

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

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

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

57         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 = ttoproc(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_glink) {
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 }

95 /*
96  * for implementations that don't require binary compatibility,
97  * the kill system call may be made into a library call to the
98  * sigsend system call
99 */
100 int
101 kill(pid_t pid, int sig)
102 {
103     int error;
104     sigsend_t v;

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

115 /*
116  * The handling of small unions, like the sigval argument to sigqueue,
117  * is architecture dependent. We have adopted the convention that the
118  * value itself is passed in the storage which crosses the kernel
119  * protection boundary. This procedure will accept a scalar argument,
120  * and store it in the appropriate value member of the sigsend_t structure.
121  */

```

```

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;
131
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));
135
136     if ((sqh = p->p_sigqhdr) == NULL) {
137         rlim64_t sigqsz_max;
138
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);
143
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);
158
159     do {
160         bzero(&v, sizeof (v));
161         v.sig = sig;
162         v.checkperm = 1;
163         v.si_code = 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);
177
178     if (error)
179         return (set_errno(error));
180     return (0);
181 }

```

unchanged_portion_omitted

new/usr/src/uts/common/syscall/sysconfig.c

1

```
*****
5280 Thu Sep 26 12:53:25 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 <iszczeniak@gmail.com>
*****
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 /*
23  * Copyright 2008 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25 */
26
27 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
28 /*      All Rights Reserved */
29
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) {
57
58         /*
```

new/usr/src/uts/common/syscall/sysconfig.c

2

```
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 */
64
65     case _CONFIG_CLK_TCK:
66         return ((long)hz); /* clock frequency per second */
67
68     case _CONFIG_PROF_TCK:
69         return ((long)hz); /* profiling clock freq per sec */
70
71     case _CONFIG_NGROUPS:
72         /*
73          * Maximum number of supplementary groups.
74          */
75         return (ngroups_max);
76
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_rctl,
86                 curproc);
87             mutex_exit(&curproc->p_lock);
88             return ((ulong_t)fd_ctl);
89         }
90
91     case _CONFIG_CHILD_MAX:
92         /*
93          * Maximum number of processes.
94          */
95         return (v.v_maxup);
96
97     case _CONFIG_POSIX_VER:
98         return (_POSIX_VERSION); /* current POSIX version */
99
100    case _CONFIG_PAGESIZE:
101        return (PAGESIZE);
102
103    case _CONFIG_XOPEN_VER:
104        return (_XOPEN_VERSION); /* current XOPEN version */
105
106    case _CONFIG_NPROC_CONF:
107        return (zone_ncpus_get(curproc->p_zone));
108
109    case _CONFIG_NPROC_ONLN:
110        return (zone_ncpus_online_get(curproc->p_zone));
111
112    case _CONFIG_NPROC_MAX:
113        return (max_ncpus);
114
115    case _CONFIG_STACK_PROT:
116        return (curproc->p_stkprot & ~PROT_USER);
117
118    case _CONFIG_AIO_LISTIO_MAX:
119        return (_AIO_LISTIO_MAX);
120
121    case _CONFIG_AIO_MAX:
122        return (_AIO_MAX);
123
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_rctl, curproc);
154             mutex_exit(&curproc->p_lock);
155             return ((uint_t)sigqsz_max);
156         }
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 (MAXEPHID);
223     case _CONFIG_SYMLINK_MAX:
224         return (MAXSYMLINKS);
225     }
226 }

```

unchanged_portion_omitted