

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

```
*****  
12538 Fri Jul 12 18:06:18 2013  
new/usr/src/uts/common/os/rctl_proc.c  
3830 SIGQUEUE_MAX's limit of 32 is too low  
*****  
1 /*  
2 * CDDL HEADER START  
3 *  
4 * The contents of this file are subject to the terms of the  
5 * Common Development and Distribution License (the "License").  
6 * You may not use this file except in compliance with the License.  
7 *  
8 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE  
9 * or http://www.opensolaris.org/os/licensing.  
10 * See the License for the specific language governing permissions  
11 * and limitations under the License.  
12 *  
13 * When distributing Covered Code, include this CDDL HEADER in each  
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.  
15 * If applicable, add the following below this CDDL HEADER, with the  
16 * fields enclosed by brackets "[]" replaced with your own identifying  
17 * information: Portions Copyright [yyyy] [name of copyright owner]  
18 *  
19 * CDDL HEADER END  
20 */  
21 /*  
22 * Copyright 2006 Sun Microsystems, Inc. All rights reserved.  
23 * Use is subject to license terms.  
24 */  
  
26 #pragma ident "%Z%%M% %I%     %E% SMI"  
  
26 #include <sys/types.h>  
27 #include <sys/cmn_err.h>  
28 #include <sys/sysmacros.h>  
29 #include <sys/proc.h>  
30 #include <sys/rctl.h>  
31 #include <sys/rctl_impl.h>  
32 #include <sys/port_kernel.h>  
33 #include <sys/signal.h>  
34 #endif /* ! codereview */  
  
36 #include <sys/vmparam.h>  
37 #include <sys/machparam.h>  
  
39 /*  
40 * Process-based resource controls  
41 * The structure of the kernel leaves us no particular place where the process  
42 * abstraction can be declared--it is intertwined with the growth of the Unix  
43 * kernel. Accordingly, we place all of the resource control logic associated  
44 * with processes, both existing and future, in this file.  
45 */  
  
47 rctl_hdl_t rctlproc_legacy[RLIM_NLIMITS];  
48 uint_t rctlproc_flags[RLIM_NLIMITS] = {  
49     RCTL_LOCAL_SIGNAL,          /* RLIMIT_CPU */  
50     RCTL_LOCAL_DENY | RCTL_LOCAL_SIGNAL, /* RLIMIT_FSIZE */  
51     RCTL_LOCAL_DENY,           /* RLIMIT_DATA */  
52     RCTL_LOCAL_DENY,           /* RLIMIT_STACK */  
53     RCTL_LOCAL_DENY,           /* RLIMIT_CORE */  
54     RCTL_LOCAL_DENY,           /* RLIMIT_NOFILE */  
55     RCTL_LOCAL_DENY           /* RLIMIT_VMEM */  
56 };  
57 int rctlproc_signals[RLIM_NLIMITS] = {  
58     SIGXCPU,                  /* RLIMIT_CPU */  
59     SIGXFSZ,                  /* RLIMIT_FSIZE */
```

```
1
```

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

```
60     0, 0, 0, 0, 0, 0                                     /* remainder do not signal */  
61 };  
  
63 rctl_hdl_t rc_process_msgrnb;  
64 rctl_hdl_t rc_process_msqtql;  
65 rctl_hdl_t rc_process_semsl;  
66 rctl_hdl_t rc_process_semopm;  
67 rctl_hdl_t rc_process_portev;  
68 rctl_hdl_t rc_process_sigqueue;  
69 #endif /* ! codereview */  
  
71 /*  
72 * process.max-cpu-time / RLIMIT_CPU  
73 */  
74 /*ARGSUSED*/  
75 static int  
76 proc_cpu_time_test(struct rctl *rctl, struct proc *p, rctl_entity_p_t *e,  
77     rctl_val_t *rval, rctl_qty_t inc, uint_t flags)  
78 {  
79     return (inc >= rval->rcv_value);  
80 }  
  
82 static rctl_ops_t proc_cpu_time_ops = {  
83     rcop_no_action,  
84     rcop_no_usage,  
85     rcop_no_set,  
86     proc_cpu_time_test  
87 };  
  
89 /*  
90 * process.max-file-size / RLIMIT_FSIZE  
91 */  
92 static int  
93 proc_filesize_set(rctl_t *rctl, struct proc *p, rctl_entity_p_t *e,  
94     rctl_qty_t nv)  
95 {  
96     if (p->p_model == DATAMODEL_NATIVE)  
97         nv = MIN(nv, rctl->rc_dict_entry->rcd_max_native);  
98     else  
99         nv = MIN(nv, rctl->rc_dict_entry->rcd_max_ilp32);  
100    ASSERT(e->rcep_t == RCENTITY_PROCESS);  
101    e->rcep_p.proc->p_fsz_ctl = nv;  
102    return (0);  
103 }  
104  
107 static rctl_ops_t proc_filesize_ops = {  
108     rcop_no_action,  
109     rcop_no_usage,  
110     proc_filesize_set,  
111     rcop_no_test  
112 };  
114 /*  
115 * process.max-data / RLIMIT_DATA  
116 */  
118 /*  
119 * process.max-stack-size / RLIMIT_STACK  
120 */  
121 static int  
122 proc_stack_set(rctl_t *rctl, struct proc *p, rctl_entity_p_t *e,  
123     rctl_qty_t nv)  
124 {  
125     klwp_t *lwp = ttolwp(curthread);
```

```
2
```

```

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

```

```

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

```

```

258     (void) rctl_rlimit_set(rctlproc_legacy[RLIMIT_STACK], initp, &rlp64, gp,
259      RCTL_LOCAL_DENY, 0, kcred);
260
261     /*
262      * RLIMIT_CORE: deny always, no default action.
263      */
264     rlp64.rlim_cur = rlp64.rlim_max = RLIM64_INFINITY;
265     (void) rctl_rlimit_set(rctlproc_legacy[RLIMIT_CORE], initp, &rlp64, gp,
266      RCTL_LOCAL_DENY, 0, kcred);
267
268     /*
269      * RLIMIT_NOFILE: deny always, no action.
270      */
271     rlp64.rlim_cur = rlim_fd_cur;
272     rlp64.rlim_max = rlim_fd_max;
273     (void) rctl_rlimit_set(rctlproc_legacy[RLIMIT_NOFILE], initp, &rlp64,
274      gp, RCTL_LOCAL_DENY, 0, kcred);
275
276     /*
277      * RLIMIT_VMEM
278      */
279     rlp64.rlim_cur = rlp64.rlim_max = RLIM64_INFINITY;
280     (void) rctl_rlimit_set(rctlproc_legacy[RLIMIT_VMEM], initp, &rlp64, gp,
281      RCTL_LOCAL_DENY, 0, kcred);
282 }
283
284 /* void rctlproc_init()
285 *
286 * Overview
287 * Register the various resource controls associated with process entities.
288 * The historical rlim_infinity_map and rlim_infinity32_map are now encoded
289 * here as the native and ILP32 infinite values for each resource control.
290 */
291 void
292 rctlproc_init(void)
293 rctlproc_init()
294 {
295     rctl_set_t *set;
296     rctl_alloc_gp_t *gp;
297     rctl_entity_p_t e;
298
299     rctlproc_legacy[RLIMIT_CPU] = rctl_register("process.max-cpu-time",
300      RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_NEVER |
301      RCTL_GLOBAL_CPU_TIME | RCTL_GLOBAL_INFINITE | RCTL_GLOBAL_SECONDS,
302      UINT64_MAX, UINT64_MAX, &proc_cpu_time_ops);
303     rctlproc_legacy[RLIMIT_FSIZE] = rctl_register("process.max-file-size",
304      RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
305      RCTL_GLOBAL_FILE_SIZE | RCTL_GLOBAL_BYTES,
306      MAXOFFSET_T, MAXOFFSET_T, &proc_filesize_ops);
307     rctlproc_legacy[RLIMIT_DATA] = rctl_register("process.max-data-size",
308      RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
309      RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
310      ULONG_MAX, UINT32_MAX, &rctl_default_ops);
311 #ifdef _LP64
312 #ifdef __sparc
313     rctlproc_legacy[RLIMIT_STACK] = rctl_register("process.max-stack-size",
314      RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
315      RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
316      LONG_MAX, INT32_MAX, &proc_stack_ops);
317 #else /* __sparc */
318     rctlproc_legacy[RLIMIT_STACK] = rctl_register("process.max-stack-size",
319      RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
320      RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
321      MAXSSIZ, USRSTACK32 - PAGESIZE, &proc_stack_ops);
322 #endif /* __sparc */

```

```

323 #else /* _LP64 */
324     rctlproc_legacy[RLIMIT_STACK] = rctl_register("process.max-stack-size",
325      RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
326      RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
327      USRSTACK - PAGESIZE, USRSTACK - PAGESIZE, &proc_stack_ops);
328 #endif
329     rctlproc_legacy[RLIMIT_CORE] = rctl_register("process.max-core-size",
330      RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
331      RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
332      MIN(MAXOFFSET_T, ULONG_MAX), UINT32_MAX, &rctl_default_ops);
333     rctlproc_legacy[RLIMIT_NOFILE] = rctl_register(
334      "process.max-file-descriptor", RCENTITY_PROCESS,
335      RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
336      RCTL_GLOBAL_COUNT, INT32_MAX, INT32_MAX, &proc_nofile_ops);
337     rctlproc_legacy[RLIMIT_VMEM] =
338     rctl_register("process.max-address-space", RCENTITY_PROCESS,
339      RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
340      RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
341      ULONG_MAX, UINT32_MAX, &proc_vmem_ops);
342
343     rc_process_semsmsl = rctl_register("process.max-sem-nsems",
344      RCENTITY_PROCESS, RCTL_GLOBAL_DENY_ALWAYS | RCTL_GLOBAL_COUNT,
345      SHRT_MAX, SHRT_MAX, &rctl_absolute_ops);
346     rctl_add_legacy_limit("process.max-sem-nsems", "semssys",
347      "seminfo_semsmsl", 512, SHRT_MAX);
348
349     rc_process_semopm = rctl_register("process.max-sem-ops",
350      RCENTITY_PROCESS, RCTL_GLOBAL_DENY_ALWAYS | RCTL_GLOBAL_COUNT,
351      INT_MAX, INT_MAX, &rctl_absolute_ops);
352     rctl_add_legacy_limit("process.max-sem-ops", "semssys",
353      "seminfo_semopm", 512, INT_MAX);
354
355     rc_process_msgrnmb = rctl_register("process.max-msg-qbytes",
356      RCENTITY_PROCESS, RCTL_GLOBAL_DENY_ALWAYS | RCTL_GLOBAL_BYTES,
357      ULONG_MAX, ULONG_MAX, &rctl_absolute_ops);
358     rctl_add_legacy_limit("process.max-msg-qbytes", "msgssys",
359      "msginfo_msgrnmb", 65536, ULONG_MAX);
360
361     rc_process_msgrql = rctl_register("process.max-msg-messages",
362      RCENTITY_PROCESS, RCTL_GLOBAL_DENY_ALWAYS | RCTL_GLOBAL_COUNT,
363      UINT_MAX, UINT_MAX, &rctl_absolute_ops);
364     rctl_add_legacy_limit("process.max-msg-messages", "msgssys",
365      "msginfo_msgrql", 8192, UINT_MAX);
366
367     rc_process_portev = rctl_register("process.max-port-events",
368      RCENTITY_PROCESS, RCTL_GLOBAL_DENY_ALWAYS | RCTL_GLOBAL_COUNT,
369      PORT_MAX_EVENTS, PORT_MAX_EVENTS, &rctl_absolute_ops);
370     rctl_add_default_limit("process.max-port-events", PORT_DEFAULT_EVENTS,
371      RCPRIV_PRIVILEGED, RCTL_LOCAL_DENY);
372
373     rc_process_sigueue = rctl_register("process.max-sigueue-size",
374      RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
375      RCTL_GLOBAL_COUNT, _SIGQUEUE_SIZE_MAX, _SIGQUEUE_SIZE_MAX,
376      &rctl_absolute_ops);
377     rctl_add_default_limit("process.max-sigueue-size",
378      _SIGQUEUE_SIZE_BASIC, RCPRIV_BASIC, RCTL_LOCAL_DENY);
379     rctl_add_default_limit("process.max-sigueue-size",
380      _SIGQUEUE_SIZE_PRIVILEGED, RCPRIV_PRIVILEGED, RCTL_LOCAL_DENY);
381
382 #endif /* ! codereview */
383 /*
384  * Place minimal set of controls on "sched" process for inheritance by
385  * processes created via newproc().
386  */
387 set = rctl_set_create();
388 gp = rctl_set_init_prealloc(RCENTITY_PROCESS);

```

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

```
*****  
73380 Fri Jul 12 18:06:19 2013  
new/usr/src/uts/common/os/sig.c  
3830 SIGQUEUE_MAX's limit of 32 is too low  
*****  
_____unchanged_portion_omitted_____  
  
2376 #ifndef INT_MAX  
2377 #define INT_MAX 2147483647  
2376 #ifndef UCHAR_MAX  
2377 #define UCHAR_MAX 255  
2378 #endif  
  
2380 /*  
2381 * The entire pool (with maxcount entries) is pre-allocated at  
2382 * the first sigqueue/signotify call.  
2383 */  
2384 sigqhdr_t *  
2385 sigqhdralloc(size_t size, uint_t maxcount)  
2386 {  
2387     size_t i;  
2388     sigqueue_t *sq, *next;  
2389     sqghdr_t *sqh;  
  
2391     i = (maxcount * size) + sizeof (sigqhdr_t);  
2392     ASSERT(maxcount <= INT_MAX);  
2392     ASSERT(maxcount <= UCHAR_MAX && i <= USHRT_MAX);  
2393     sqh = kmalloc(i, KM_SLEEP);  
2394     sqh->sqb_count = maxcount;  
2395     sqh->sqb_maxcount = maxcount;  
2396     sqh->sqb_size = i;  
2394     sqh->sqb_count = (uchar_t)maxcount;  
2395     sqh->sqb_maxcount = (uchar_t)maxcount;  
2396     sqh->sqb_size = (ushort_t)i;  
2397     sqh->sqb_pextited = 0;  
2398     sqh->sqb_sent = 0;  
2399     sqh->sqb_free = sq = (sigqueue_t *) (sqh + 1);  
2400     for (i = maxcount - 1; i != 0; i--) {  
2401         next = (sigqueue_t *) ((uintptr_t)sq + size);  
2402         sq->sq_next = next;  
2403         sq = next;  
2404     }  
2405     sq->sq_next = NULL;  
2406     cv_init(&sqh->sqb_cv, NULL, CV_DEFAULT, NULL);  
2407     mutex_init(&sqh->sqb_lock, NULL, MUTEX_DEFAULT, NULL);  
2408     return (sqh);  
2409 }  
_____unchanged_portion_omitted_____
```

10030 Fri Jul 12 18:06:19 2013
new/usr/src/uts/common/sys/signal.h
3830 SIGQUEUE_MAX's limit of 32 is too low

_____unchanged_portion_omitted_____

```
303 typedef struct sigghdr {          /* sigqueue pool header      */  
304     sigqueue_t    *sqb_free;      /* free sigq struct list    */  
305     int           sqb_count;     /* sigq free count          */  
306     uint_t        sqb_maxcount;  /* sigq max free count      */  
307     size_t        sqb_size;      /* size of header+free structs */  
308     uchar_t       sqb_count;     /* sigq free count          */  
309     ushort_t     sqb_maxcount;  /* sigq max free count      */  
310     uchar_t       sqb_size;      /* size of header+free structs */  
311     uchar_t       sqb_pexited;   /* process has exited       */  
312     uint_t        sqb_sent;      /* number of sigq sent      */  
313     uchar_t       sqb_sent;      /* number of sigq sent      */  
314     kcondvar_t   sqb_cv;        /* waiting for a sigq struct */  
315     kmutex_t     sqb_lock;      /* lock for sigq pool       */  
  
314 #define _SIGQUEUE_SIZE_BASIC      128    /* basic limit */  
315 #define _SIGQUEUE_SIZE_PRIVILEGED 512    /* privileged limit */  
316 #define _SIGQUEUE_SIZE_MAX        8192   /* maximum limit */  
  
315 #define _SIGQUEUE_MAX    32  
318 #define _SIGNOTIFY_MAX  32  
  
320 extern void    setsigact(int, void (*)(int), const k_sigset_t *, int);  
321 extern void    sigorset(k_sigset_t *, const k_sigset_t *);  
322 extern void    sigandset(k_sigset_t *, const k_sigset_t *);  
323 extern void    sigdiffset(k_sigset_t *, const k_sigset_t *);  
324 extern void    sigintt(k_sigset_t *, int);  
325 extern void    sigunintr(k_sigset_t *);  
326 extern void    sigreplace(k_sigset_t *, k_sigset_t *);  
  
328 extern int     kill(pid_t, int);  
  
330 #endif /* _KERNEL */  
332 #ifdef __cplusplus  
333 }
```

_____unchanged_portion_omitted_____

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

```
*****
5578 Fri Jul 12 18:06:20 2013
new/usr/src/uts/common/syscall/sigqueue.c
3830 SIGQUEUE_MAX's limit of 32 is too low
*****
```

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

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

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

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

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

41 extern rctl_hdlr_t rc_process_sigqueue;

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

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

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

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

1

```
new/usr/src/uts/common/syscall/sigqueue.c
```

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_pglink) {
80 if (p->p_stat != SIDL) {
81 nfound++;
82 error = sigsendproc(p, sigsend);
83 }
84 }
85 mutex_exit(&pidlock);
86 if (nfound == 0)
87 error = ESRCH;
88 else if (error == 0 && sigsend->perm == 0)
89 error = EPERM;
90 }
92 return (error);
93 }

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

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

116 /*
117 * The handling of small unions, like the sigval argument to sigqueue,
118 * is architecture dependent. We have adopted the convention that the
119 * value itself is passed in the storage which crosses the kernel
120 * protection boundary. This procedure will accept a scalar argument,
121 * and store it in the appropriate value member of the sigsend_t structure.
122 */
123 int
124 sigqueue(pid_t pid, int sig, /* union sigval */ void *value,
125 int si_code, int block)

2

```

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

```
*****
5280 Fri Jul 12 18:06:20 2013
new/usr/src/uts/common/syscall/sysconfig.c
3830 SIGQUEUE_MAX's limit of 32 is too low
*****
1 /*
2 * CDDL HEADER START
3 *
4 * The contents of this file are subject to the terms of the
5 * Common Development and Distribution License (the "License").
6 * You may not use this file except in compliance with the License.
7 *
8 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
22 /*
23 * Copyright 2008 Sun Microsystems, Inc. All rights reserved.
24 * Use is subject to license terms.
25 */
27 /* Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
28 /* All Rights Reserved */
30 #include <sys/param.h>
31 #include <sys/types.h>
32 #include <sys/sysmacros.h>
33 #include <sys/sysm.h>
34 #include <sys/tunable.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/znode.h>
48 #include <sys/vm_usage.h>
50 extern rctl_hdlr_t rc_process_sigqueue;
52 #endif /* ! codereview */
53 long
54 sysconfig(int which)
55 {
56     switch (which) {
58     /*
59     * if it is not handled in mach_sysconfig either
60     * it must be EINVAL.
61     */
62     default:
63         return (mach_sysconfig(which)); /* 'uname -i' /os */
65     case _CONFIG_CLK_TCK:
66         return ((long)hz); /* clock frequency per second */
68     case _CONFIG_PROF_TCK:
69         return ((long)hz); /* profiling clock freq per sec */
71     case _CONFIG_NGROUPS:
72         /*
73         * Maximum number of supplementary groups.
74         */
75         return (ngroups_max);
77     case _CONFIG_OPEN_FILES:
78         /*
79         * Maximum number of open files (soft limit).
80         */
81         {
82             rlim64_t fd_ctl;
83             mutex_enter(&curproc->p_lock);
84             fd_ctl = rctl_enforced_value(
85                 rctlproc_legacy[RLIMIT_NOFILE], curproc->p_rctl,
86                 curproc);
87             mutex_exit(&curproc->p_lock);
88             return ((ulong_t)fd_ctl);
89         }
91     case _CONFIG_CHILD_MAX:
92         /*
93         * Maximum number of processes.
94         */
95         return (v.v_maxup);
97     case _CONFIG_POSIX_VER:
98         return (_POSIX_VERSION); /* current POSIX version */
100    case _CONFIG_PAGESIZE:
101        return (PAGESIZE);
103    case _CONFIG_XOPEN_VER:
104        return (_XOPEN_VERSION); /* current XOPEN version */
106    case _CONFIG_NPROC_CONF:
107        return (zone_ncpus_get(curproc->p_zone));
109    case _CONFIG_NPROC_ONLN:
110        return (zone_ncpus_online_get(curproc->p_zone));
112    case _CONFIG_NPROC_MAX:
113        return (max_ncpus);
115    case _CONFIG_STACK_PROT:
116        return (curproc->p_stkprot & ~PROT_USER);
118    case _CONFIG_AIO_LISTIO_MAX:
119        return (_AIO_LISTIO_MAX);
121    case _CONFIG_AIO_MAX:
122        return (_AIO_MAX);
124    case _CONFIG_AIO_PRIO_DELTA_MAX:
125        return (0);
127    case _CONFIG_DELAYTIMER_MAX:
```

1

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

```
62     default:
63         return (mach_sysconfig(which)); /* 'uname -i' /os */
65     case _CONFIG_CLK_TCK:
66         return ((long)hz); /* clock frequency per second */
68     case _CONFIG_PROF_TCK:
69         return ((long)hz); /* profiling clock freq per sec */
71     case _CONFIG_NGROUPS:
72         /*
73         * Maximum number of supplementary groups.
74         */
75         return (ngroups_max);
77     case _CONFIG_OPEN_FILES:
78         /*
79         * Maximum number of open files (soft limit).
80         */
81         {
82             rlim64_t fd_ctl;
83             mutex_enter(&curproc->p_lock);
84             fd_ctl = rctl_enforced_value(
85                 rctlproc_legacy[RLIMIT_NOFILE], curproc->p_rctl,
86                 curproc);
87             mutex_exit(&curproc->p_lock);
88             return ((ulong_t)fd_ctl);
89         }
91     case _CONFIG_CHILD_MAX:
92         /*
93         * Maximum number of processes.
94         */
95         return (v.v_maxup);
97     case _CONFIG_POSIX_VER:
98         return (_POSIX_VERSION); /* current POSIX version */
100    case _CONFIG_PAGESIZE:
101        return (PAGESIZE);
103    case _CONFIG_XOPEN_VER:
104        return (_XOPEN_VERSION); /* current XOPEN version */
106    case _CONFIG_NPROC_CONF:
107        return (zone_ncpus_get(curproc->p_zone));
109    case _CONFIG_NPROC_ONLN:
110        return (zone_ncpus_online_get(curproc->p_zone));
112    case _CONFIG_NPROC_MAX:
113        return (max_ncpus);
115    case _CONFIG_STACK_PROT:
116        return (curproc->p_stkprot & ~PROT_USER);
118    case _CONFIG_AIO_LISTIO_MAX:
119        return (_AIO_LISTIO_MAX);
121    case _CONFIG_AIO_MAX:
122        return (_AIO_MAX);
124    case _CONFIG_AIO_PRIO_DELTA_MAX:
125        return (0);
127    case _CONFIG_DELAYTIMER_MAX:
```

2

```

128         return (INT_MAX);
130
131     case _CONFIG_MQ_OPEN_MAX:
132         return (_MQ_OPEN_MAX);
133
134     case _CONFIG_MQ_PRIO_MAX:
135         return (_MQ_PRIO_MAX);
136
137     case _CONFIG_RTSIG_MAX:
138         return (_SIGRTMAX - _SIGRTMIN + 1);
139
140     case _CONFIG_SEM_NSEMS_MAX:
141         return (_SEM_NSEMS_MAX);
142
143     case _CONFIG_SEM_VALUE_MAX:
144         return (_SEM_VALUE_MAX);
145
146     case _CONFIG_SIGQUEUE_MAX:
147         /*
148          * Maximum number of outstanding queued signals.
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
157     return (_SIGQUEUE_MAX);
158
159     case _CONFIG_SIGRT_MIN:
160         return (_SIGRTMIN);
161
162     case _CONFIG_SIGRT_MAX:
163         return (_SIGRTMAX);
164
165     case _CONFIG_TIMER_MAX:
166         return (timer_max);
167
168     case _CONFIG_PHYS_PAGES:
169         /*
170          * If the non-global zone has a phys. memory cap, use that.
171          * We always report the system-wide value for the global zone,
172          * even though rcapd can be used on the global zone too.
173          */
174         if (!INGLOBALZONE(curproc) &&
175             curproc->p_zone->zone_phys_mcap != 0)
176             return (MIN(bttop(curproc->p_zone->zone_phys_mcap),
177                         physinstalled));
178
179         return (physinstalled);
180
181     case _CONFIG_AVPHYS_PAGES:
182         /*
183          * If the non-global zone has a phys. memory cap, use
184          * the phys. memory cap - zone's current rss. We always
185          * report the system-wide value for the global zone, even
186          * though rcapd can be used on the global zone too.
187          */
188         if (!INGLOBALZONE(curproc) &&
189             curproc->p_zone->zone_phys_mcap != 0) {
190             pgcnt_t cap, rss, free;
191             vmsusage_t in_use;
192             size_t cnt = 1;

```

```

193         cap = bttop(curproc->p_zone->zone_phys_mcap);
194         if (cap > physinstalled)
195             return (freemem);
196
197         if (vm_getusage(VMUSAGE_ZONE, 1, &in_use, &cnt,
198                         FKIOCTL) != 0)
199             in_use.vmu_rss_all = 0;
200         rss = bttop(in_use.vmu_rss_all);
201
202         /*
203          * Because rcapd implements a soft cap, it is possible
204          * for rss to be temporarily over the cap.
205          */
206         if (cap > rss)
207             free = cap - rss;
208         else
209             free = 0;
210         return (MIN(free, freemem));
211
212     return (freemem);
213
214     case _CONFIG_MAXPID:
215         return (maxpid);
216
217     case _CONFIG_CPUID_MAX:
218         return (max_cpuid);
219
220     case _CONFIG_EPHID_MAX:
221         return (MAXEPHUID);
222
223     case _CONFIG_SYMLOOP_MAX:
224         return (MAXSYMLINKS);
225
226 }

```

unchanged portion omitted