```
**********************************************************
   12507 Mon Jun 24 12:53:16 2013
new/usr/src/uts/common/os/rctl_proc.c
3830 SIGQUEUE_MAX's limit of 32 is too low
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

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

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

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

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

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

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

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

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

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

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

        return (0);
}

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

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

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

```c
	if (p->p_model == DATAMODEL_NATIVE)
		nv = MIN(nv, rctl->rc_dict_entry->rcd_max_native);
	else
		nv = MIN(nv, rctl->rc_dict_entry->rcd_max_ilp32);

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

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

	return (0);
}

static rctl_ops_t proc_stack_ops = {
	rcop_no_action,
	rcop_no_usage,
	proc_stack_set,
	rcop_no_test
};

/*
 * process.max-file-descriptors / RLIMIT_NOFILE
 */
static int
proc_nofile_set(rctl_t *rctl, struct proc *p, rctl_entity_p_t *e, rctl_qty_t nv)
{
	ASSERT(e->rcep_t == RCENTITY_PROCESS);
	if (p->p_model == DATAMODEL_NATIVE)
		nv = MIN(nv, rctl->rc_dict_entry->rcd_max_native);
	else
		nv = MIN(nv, rctl->rc_dict_entry->rcd_max_ilp32);

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

	return (0);
}

static rctl_ops_t proc_nofile_ops = {
	rcop_no_action,
	rcop_no_usage,
	proc_nofile_set,
	rcop_absolute_test
};

/*
```

```c
 * process.max-address-space / RLIMIT_VMEM
 */
static int
proc_vmem_set(rctl_t *rctl, struct proc *p, rctl_entity_p_t *e, rctl_qty_t nv)
{
	ASSERT(e->rcep_t == RCENTITY_PROCESS);
	if (p->p_model == DATAMODEL_ILP32)
		nv = MIN(nv, rctl->rc_dict_entry->rcd_max_ilp32);
	else
		nv = MIN(nv, rctl->rc_dict_entry->rcd_max_native);

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

	return (0);
}

static rctl_ops_t proc_vmem_ops = {
	rcop_no_action,
	rcop_no_usage,
	proc_vmem_set,
	rcop_no_test
};

/*
 * void rctlproc_default_init()
 *
 * Overview
 *   Establish default basic and privileged control values on the init process.
 *   These correspond to the soft and hard limits, respectively.
 */
void
rctlproc_default_init(struct proc *initp, rctl_alloc_gp_t *gp)
{
	struct rlimit64 rlp64;

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

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

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

	/*
	 * RLIMIT_STACK: deny always, no default action.
	 */
#ifdef __sparc
	rlp64.rlim_cur = DFLSSIZ;
	rlp64.rlim_max = LONG_MAX;
#else
	rlp64.rlim_cur = DFLSSIZ;
	rlp64.rlim_max = MAXSSIZ;
#endif
```

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

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

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

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

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

        rctlproc_legacy[RLIMIT_CPU] = rctl_register("process.max-cpu-time",
            RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_NEVER |
            RCTL_GLOBAL_CPU_TIME | RCTL_GLOBAL_INFINITE | RCTL_GLOBAL_SECONDS,
            UINT64_MAX, UINT64_MAX, &proc_cpu_time_ops);
        rctlproc_legacy[RLIMIT_FSIZE] = rctl_register("process.max-file-size",
            RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
            RCTL_GLOBAL_FILE_SIZE | RCTL_GLOBAL_BYTES,
            MAXOFFSET_T, MAXOFFSET_T, &proc_filesize_ops);
        rctlproc_legacy[RLIMIT_DATA] = rctl_register("process.max-data-size",
            RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
            RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
            ULONG_MAX, UINT32_MAX, &rctl_default_ops);
#ifdef _LP64
#ifdef __sparc
        rctlproc_legacy[RLIMIT_STACK] = rctl_register("process.max-stack-size",
            RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
            RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
            LONG_MAX, INT32_MAX, &proc_stack_ops);
#else   /* __sparc */
        rctlproc_legacy[RLIMIT_STACK] = rctl_register("process.max-stack-size",
            RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
            RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_BYTES,
            MAXSSIZ, USRSTACK32 - PAGESIZE, &proc_stack_ops);
#endif  /* __sparc */
```

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

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

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

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

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

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

        rc_process_sigqueue = rctl_register("process.max-sigqueue-size",
            RCENTITY_PROCESS, RCTL_GLOBAL_LOWERABLE | RCTL_GLOBAL_DENY_ALWAYS |
            RCTL_GLOBAL_COUNT, _SIGQSZ_MAX, _SIGQSZ_MAX, &rctl_absolute_ops);
        rctl_add_default_limit("process.max-sigqueue-size", _SIGQSZ_DEFAULT,
            RCPRIV_BASIC, RCTL_LOCAL_DENY);
        rctl_add_default_limit("process.max-sigqueue-size", _SIGQSZ_PRIVILEGED,
            RCPRIV_PRIVILEGED, RCTL_LOCAL_DENY);

#endif /* ! codereview */
        /*
         * Place minimal set of controls on "sched" process for inheritance by
         * processes created via newproc().
         */
        set = rctl_set_create();
        gp = rctl_set_init_prealloc(RCENTITY_PROCESS);
        mutex_enter(&curproc->p_lock);
```

```
	e.rcep_p.proc = curproc;
	e.rcep_t = RCENTITY_PROCESS;
	curproc->p_rctls = rctl_set_init(RCENTITY_PROCESS, curproc, &e,
	    set, gp);
	mutex_exit(&curproc->p_lock);
	rctl_prealloc_destroy(gp);
}
```

```
**********************************************************
   73380 Mon Jun 24 12:53:17 2013
new/usr/src/uts/common/os/sig.c
3830 SIGQUEUE_MAX's limit of 32 is too low
**********************************************************
______unchanged_portion_omitted_

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

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

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

```
**********************************************************
   10094 Mon Jun 24 12:53:17 2013
new/usr/src/uts/common/sys/signal.h
3830 SIGQUEUE_MAX's limit of 32 is too low
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright (c) 1988, 2012, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

/*
 * University Copyright- Copyright (c) 1982, 1986, 1988
 * The Regents of the University of California
 * All Rights Reserved
 *
 * University Acknowledgment- Portions of this document are derived from
 * software developed by the University of California, Berkeley, and its
 * contributors.
 */

#ifndef _SYS_SIGNAL_H
#define _SYS_SIGNAL_H

#include <sys/feature_tests.h>
#include <sys/iso/signal_iso.h>

#ifdef  __cplusplus
extern "C" {
#endif

#if defined(__EXTENSIONS__) || defined(_KERNEL) || !defined(_STRICT_STDC) || \
        defined(__XOPEN_OR_POSIX)

#if defined(__EXTENSIONS__) || defined(_KERNEL) || \
        (!defined(_STRICT_STDC) && !defined(__XOPEN_OR_POSIX)) || \
        (_POSIX_C_SOURCE > 2) || defined(_XPG4_2)
/*
 * We need <sys/siginfo.h> for the declaration of siginfo_t.
 */
#include <sys/siginfo.h>
#endif
```

```
/* Duplicated in <sys/ucontext.h> as a result of XPG4v2 requirements */
#ifndef _SIGSET_T
#define _SIGSET_T
typedef struct {                /* signal set type */
        unsigned int    __sigbits[4];
} sigset_t;
______unchanged_portion_omitted_

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

/*
 * Limits on maximum number of sigqueue(3C) entries per process.
 */
#define _SIGQSZ_DEFAULT         128     /* default number */
#define _SIGQSZ_PRIVILEGED      512     /* privilege required to set abouve */
#define _SIGQSZ_MAX             8192    /* max even with privilege */

#define _SIGQUEUE_MAX   32
#define _SIGNOTIFY_MAX  32

extern  void    setsigact(int, void (*)(int), const k_sigset_t *, int);
extern  void    sigorset(k_sigset_t *, const k_sigset_t *);
extern  void    sigandset(k_sigset_t *, const k_sigset_t *);
extern  void    sigdiffset(k_sigset_t *, const k_sigset_t *);
extern  void    sigintr(k_sigset_t *, int);
extern  void    sigunintr(k_sigset_t *);
extern  void    sigreplace(k_sigset_t *, k_sigset_t *);

extern  int     kill(pid_t, int);

#endif /* _KERNEL */

#ifdef  __cplusplus
}
______unchanged_portion_omitted_
```

```
**********************************************************
    5578 Mon Jun 24 12:53:18 2013
new/usr/src/uts/common/syscall/sigqueue.c
3830 SIGQUEUE_MAX's limit of 32 is too low
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

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

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

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

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

extern rctl_hndl_t rc_process_sigqueue;

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

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

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

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

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

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

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

        return (error);
}


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

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

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

```
{
        int error;
        sigsend_t v;
        sigqhdr_t *sqh;
        proc_t *p = curproc;

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

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

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

#endif /* ! codereview */
                /* Allocate sigqueue pool first time */
                sqh = sigqhdralloc(sizeof (sigqueue_t), (uint_t)sigqsz_max);
                sqh = sigqhdralloc(sizeof (sigqueue_t), _SIGQUEUE_MAX);
                mutex_enter(&p->p_lock);
                if (p->p_sigqhdr == NULL) {
                        /* hang the pool head on proc */
                        p->p_sigqhdr = sqh;
                } else {
                        /* another lwp allocated the pool, free ours */
                        sigqhdrfree(sqh);
                        sqh = p->p_sigqhdr;
                }
                mutex_exit(&p->p_lock);
        }

        do {
                bzero(&v, sizeof (v));
                v.sig = sig;
                v.checkperm = 1;
                v.sicode = si_code;
                v.value.sival_ptr = value;
                if ((error = sigqkill(pid, &v)) != EAGAIN || !block)
                        break;
                /* block waiting for another chance to allocate a sigqueue_t */
                mutex_enter(&sqh->sqb_lock);
                while (sqh->sqb_count == 0) {
                        if (!cv_wait_sig(&sqh->sqb_cv, &sqh->sqb_lock)) {
                                error = EINTR;
                                break;
                        }
                }
                mutex_exit(&sqh->sqb_lock);
        } while (error == EAGAIN);

        if (error)
                return (set_errno(error));
        return (0);
}
```
______***unchanged_portion_omitted_***

```
**********************************************************
    5280 Mon Jun 24 12:53:18 2013
new/usr/src/uts/common/syscall/sysconfig.c
3830 SIGQUEUE_MAX's limit of 32 is too low
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

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

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

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

extern rctl_hndl_t rc_process_sigqueue;

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

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

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

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

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

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

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

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

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

        case _CONFIG_PAGESIZE:
                return (PAGESIZE);

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

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

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

        case _CONFIG_NPROC_MAX:
                return (max_ncpus);

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

        case _CONFIG_AIO_LISTIO_MAX:
                return (_AIO_LISTIO_MAX);

        case _CONFIG_AIO_MAX:
                return (_AIO_MAX);

        case _CONFIG_AIO_PRIO_DELTA_MAX:
                return (0);

        case _CONFIG_DELAYTIMER_MAX:
                return (INT_MAX);

        case _CONFIG_MQ_OPEN_MAX:
                return (_MQ_OPEN_MAX);

        case _CONFIG_MQ_PRIO_MAX:
                return (_MQ_PRIO_MAX);

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

        case _CONFIG_SEM_NSEMS_MAX:
                return (_SEM_NSEMS_MAX);

        case _CONFIG_SEM_VALUE_MAX:
                return (_SEM_VALUE_MAX);

        case _CONFIG_SIGQUEUE_MAX:
                /*
                 * Maximum number of outstanding queued signals.
                 */
                {
                        rlim64_t sigqsz_max;
                        mutex_enter(&curproc->p_lock);
                        sigqsz_max = rctl_enforced_value(rc_process_sigqueue,
                            curproc->p_rctls, curproc);
                        mutex_exit(&curproc->p_lock);
                        return ((uint_t)sigqsz_max);
                }
                return (_SIGQUEUE_MAX);

        case _CONFIG_SIGRT_MIN:
                return (_SIGRTMIN);

        case _CONFIG_SIGRT_MAX:
                return (_SIGRTMAX);

        case _CONFIG_TIMER_MAX:
                return (timer_max);

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

                return (physinstalled);

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

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

                        if (vm_getusage(VMUSAGE_ZONE, 1, &in_use, &cnt,
                            FKIOCTL) != 0)
                                in_use.vmu_rss_all = 0;
                        rss = btop(in_use.vmu_rss_all);
                        /*
                         * Because rcapd implements a soft cap, it is possible
                         * for rss to be temporarily over the cap.
                         */
                        if (cap > rss)
                                free = cap - rss;
                        else
                                free = 0;
                        return (MIN(free, freemem));
                }

                return (freemem);

        case _CONFIG_MAXPID:
                return (maxpid);

        case _CONFIG_CPUID_MAX:
                return (max_cpuid);

        case _CONFIG_EPHID_MAX:
                return (MAXEPHUID);

        case _CONFIG_SYMLOOP_MAX:
                return (MAXSYMLINKS);
        }
}
______unchanged_portion_omitted_
```