```
**********************************************************
   19352 Wed Aug 19 20:52:09 2015
new/usr/src/uts/common/fs/pcfs/pc_node.c
6141 use kmem_zalloc instead of kmem_alloc + bzero/memset
**********************************************************
```

```
/*
 * 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 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

#include <sys/param.h>
#include <sys/t_lock.h>
#include <sys/errno.h>
#include <sys/sysmacros.h>
#include <sys/buf.h>
#include <sys/systm.h>
#include <sys/vfs.h>
#include <sys/vnode.h>
#include <sys/kmem.h>
#include <sys/proc.h>
#include <sys/cred.h>
#include <sys/cmn_err.h>
#include <sys/debug.h>
#include <vm/pvn.h>
#include <sys/fs/pc_label.h>
#include <sys/fs/pc_fs.h>
#include <sys/fs/pc_dir.h>
#include <sys/fs/pc_node.h>
#include <sys/dirent.h>
#include <sys/fdio.h>
#include <sys/file.h>
#include <sys/conf.h>

struct pchead pcfhead[NPCHASH];
struct pchead pcdhead[NPCHASH];

extern krwlock_t pcnodes_lock;

static int      pc_getentryblock(struct pcnode *, struct buf **);
static int      syncpcp(struct pcnode *, int);

/*
 * fake entry for root directory, since this does not have a parent
 * pointing to it.
```



```
 */
struct pcdir pcfs_rootdirentry = {
        "",
        "",
        PCA_DIR
};
_____unchanged_portion_omitted_

struct pcnode *
pc_getnode(
        struct pcfs *fsp,       /* filsystem for node */
        daddr_t blkno,          /* phys block no of dir entry */
        int offset,             /* offset of dir entry in block */
        struct pcdir *ep)       /* node dir entry */
{
        struct pcnode *pcp;
        struct pchead *hp;
        struct vnode *vp;
        pc_cluster32_t scluster;

        ASSERT(fsp->pcfs_flags & PCFS_LOCKED);
        if (ep == (struct pcdir *)0) {
                ep = &pcfs_rootdirentry;
                scluster = 0;
        } else {
                scluster = pc_getstartcluster(fsp, ep);
        }
        /*
         * First look for active nodes.
         * File nodes are identified by the location (blkno, offset) of
         * its directory entry.
         * Directory nodes are identified by the starting cluster number
         * for the entries.
         */
        if (ep->pcd_attr & PCA_DIR) {
                hp = &pcdhead[PCDHASH(fsp, scluster)];
                rw_enter(&pcnodes_lock, RW_READER);
                for (pcp = hp->pch_forw;
                    pcp != (struct pcnode *)hp; pcp = pcp->pc_forw) {
                        if ((fsp == VFSTOPCFS(PCTOV(pcp)->v_vfsp)) &&
                            (scluster == pcp->pc_scluster)) {
                                VN_HOLD(PCTOV(pcp));
                                rw_exit(&pcnodes_lock);
                                return (pcp);
                        }
                }
                rw_exit(&pcnodes_lock);
        } else {
                hp = &pcfhead[PCFHASH(fsp, blkno, offset)];
                rw_enter(&pcnodes_lock, RW_READER);
                for (pcp = hp->pch_forw;
                    pcp != (struct pcnode *)hp; pcp = pcp->pc_forw) {
                        if ((fsp == VFSTOPCFS(PCTOV(pcp)->v_vfsp)) &&
                            ((pcp->pc_flags & PC_INVAL) == 0) &&
                            (blkno == pcp->pc_eblkno) &&
                            (offset == pcp->pc_eoffset)) {
                                VN_HOLD(PCTOV(pcp));
                                rw_exit(&pcnodes_lock);
                                return (pcp);
                        }
                }
                rw_exit(&pcnodes_lock);
        }
        /*
         * Cannot find node in active list. Allocate memory for a new node
         * initialize it, and put it on the active list.
```

```
140          */
141         pcp = kmem_zalloc(sizeof (struct pcnode), KM_SLEEP);
143         pcp = kmem_alloc(sizeof (struct pcnode), KM_SLEEP);
144         bzero(pcp, sizeof (struct pcnode));
142         vp = vn_alloc(KM_SLEEP);
143         pcp->pc_vn = vp;
144         pcp->pc_entry = *ep;
145         pcp->pc_eblkno = blkno;
146         pcp->pc_eoffset = offset;
147         pcp->pc_scluster = scluster;
148         pcp->pc_lcluster = scluster;
149         pcp->pc_lindex = 0;
150         pcp->pc_flags = 0;
151         if (ep->pcd_attr & PCA_DIR) {
152                 vn_setops(vp, pcfs_dvnodeops);
153                 vp->v_type = VDIR;
154                 if (scluster == 0) {
155                         vp->v_flag = VROOT;
156                         blkno = offset = 0;
157                         if (IS_FAT32(fsp)) {
158                                 pc_cluster32_t ncl = 0;

160                                 scluster = fsp->pcfs_rdirstart;
161                                 if (pc_fileclsize(fsp, scluster, &ncl)) {
162                                         PC_DPRINTF1(2, "cluster chain "
163                                             "corruption, scluster=%d\n",
164                                             scluster);
165                                         pcp->pc_flags |= PC_INVAL;
166                                 }
167                                 pcp->pc_size = fsp->pcfs_clsize * ncl;
168                         } else {
169                                 pcp->pc_size =
170                                     fsp->pcfs_rdirsec * fsp->pcfs_secsize;
171                         }
172                 } else {
173                         pc_cluster32_t ncl = 0;

175                         if (pc_fileclsize(fsp, scluster, &ncl)) {
176                                 PC_DPRINTF1(2, "cluster chain corruption, "
177                                     "scluster=%d\n", scluster);
178                                 pcp->pc_flags |= PC_INVAL;
179                         }
180                         pcp->pc_size = fsp->pcfs_clsize * ncl;
181                 }
182         } else {
183                 vn_setops(vp, pcfs_fvnodeops);
184                 vp->v_type = VREG;
185                 vp->v_flag = VNOSWAP;
186                 fsp->pcfs_frefs++;
187                 pcp->pc_size = ltohi(ep->pcd_size);
188         }
189         fsp->pcfs_nrefs++;
190         VFS_HOLD(PCFSTOVFS(fsp));
191         vp->v_data = (caddr_t)pcp;
192         vp->v_vfsp = PCFSTOVFS(fsp);
193         vn_exists(vp);
194         rw_enter(&pcnodes_lock, RW_WRITER);
195         insque(pcp, hp);
196         rw_exit(&pcnodes_lock);
197         return (pcp);
198 }
```

_____***unchanged_portion_omitted_***

****
190710 Wed Aug 19 20:52:09 2015
new/usr/src/uts/common/inet/ip/spd.c
6141 use kmem_zalloc instead of kmem_alloc + bzero/memset
****
_____***unchanged_portion_omitted_***

```
4520 ipsec_latch_t *
4521 iplatch_create()
4522 {
4523         ipsec_latch_t *ipl = kmem_zalloc(sizeof (*ipl), KM_NOSLEEP);
4523         ipsec_latch_t *ipl = kmem_alloc(sizeof (*ipl), KM_NOSLEEP);
4524         if (ipl == NULL)
4525                 return (ipl);
4526         bzero(ipl, sizeof (*ipl));
4526         mutex_init(&ipl->ipl_lock, NULL, MUTEX_DEFAULT, NULL);
4527         ipl->ipl_refcnt = 1;
4528         return (ipl);
4529 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   16061 Wed Aug 19 20:52:10 2015
new/usr/src/uts/common/io/pckt.c
6141 use kmem_zalloc instead of kmem_alloc + bzero/memset
**********************************************************
```

```
   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, Version 1.0 only
   6  * (the "License").  You may not use this file except in compliance
   7  * with the License.
   8  *
   9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
  10  * or http://www.opensolaris.org/os/licensing.
  11  * See the License for the specific language governing permissions
  12  * and limitations under the License.
  13  *
  14  * When distributing Covered Code, include this CDDL HEADER in each
  15  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  16  * If applicable, add the following below this CDDL HEADER, with the
  17  * fields enclosed by brackets "[]" replaced with your own identifying
  18  * information: Portions Copyright [yyyy] [name of copyright owner]
  19  *
  20  * CDDL HEADER END
  21  */
  22 /*
  23  * Copyright 2004 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   */


  31 #pragma ident   "%Z%%M% %I%     %E% SMI"        /* from S5R4 1.10 */

  31 /*
  32  * Description: The pckt module packetizes messages on
  33  *              its read queue by pre-fixing an M_PROTO
  34  *              message type to certain incoming messages.
  35  */

  37 #include <sys/types.h>
  38 #include <sys/param.h>
  39 #include <sys/stream.h>
  40 #include <sys/stropts.h>
  41 #include <sys/kmem.h>
  42 #include <sys/errno.h>
  43 #include <sys/ddi.h>
  44 #include <sys/sunddi.h>
  45 #include <sys/debug.h>

  47 /*
  48  * This is the loadable module wrapper.
  49  */
  50 #include <sys/conf.h>
  51 #include <sys/modctl.h>

  53 static struct streamtab pcktinfo;

  55 /*
  56  * Per queue instances are single-threaded since the q_ptr
  57  * field of queues need to be shared among threads.
  58  */
  59 static struct fmodsw fsw = {
```



```
  60         "pckt",
  61         &pcktinfo,
  62         D_NEW | D_MTPERQ | D_MP
  63 };
_____unchanged_portion_omitted_

 183 /*
 184  * Dummy qbufcall callback routine used by open and close.
 185  * The framework will wake up qwait_sig when we return from
 186  * this routine (as part of leaving the perimeters.)
 187  * (The framework enters the perimeters before calling the qbufcall() callback
 188  * and leaves the perimeters after the callback routine has executed. The
 189  * framework performs an implicit wakeup of any thread in qwait/qwait_sig
 190  * when it leaves the perimeter. See qwait(9E).)
 191  */
 192 /* ARGSUSED */
 193 static void
 194 dummy_callback(void *arg)
 195 {}

 197 /*
 198  * pcktopen - open routine gets called when the
 199  *          module gets pushed onto the stream.
 200  */
 201 /*ARGSUSED*/
 202 static int
 203 pcktopen(
 204         queue_t *q,             /* pointer to the read side queue */
 205         dev_t   *devp,          /* pointer to stream tail's dev */
 206         int     oflag,          /* the user open(2) supplied flags */
 207         int     sflag,          /* open state flag */
 208         cred_t  *credp)         /* credentials */
 209 {
 210         struct pckt_info        *pip;
 211         mblk_t                  *mop; /* ptr to a setopts msg block */
 212         struct stroptions       *sop;

 214         if (sflag != MODOPEN)
 215                 return (EINVAL);

 217         if (q->q_ptr != NULL) {
 218                 /* It's already attached. */
 219                 return (0);
 220         }

 222         /*
 223          * Allocate state structure.
 224          */
 225         pip = kmem_zalloc(sizeof (*pip), KM_SLEEP);
 227         pip = kmem_alloc(sizeof (*pip), KM_SLEEP);
 228         bzero(pip, sizeof (*pip));

 227 #ifdef _MULTI_DATAMODEL
 228         pip->model = ddi_model_convert_from(get_udatamodel());
 229 #endif /* _MULTI_DATAMODEL */

 231         /*
 232          * Cross-link.
 233          */
 234         pip->pi_qptr = q;
 235         q->q_ptr = pip;
 236         WR(q)->q_ptr = pip;

 238         qprocson(q);

 240         /*
```

```
241          * Initialize an M_SETOPTS message to set up hi/lo water marks on
242          * stream head read queue.
243          */

245         while ((mop = allocb(sizeof (struct stroptions), BPRI_MED)) == NULL) {
246                 bufcall_id_t id = qbufcall(q, sizeof (struct stroptions),
247                     BPRI_MED, dummy_callback, NULL);
248                 if (!qwait_sig(q)) {
249                         qunbufcall(q, id);
250                         kmem_free(pip, sizeof (*pip));
251                         qprocsoff(q);
252                         return (EINTR);
253                 }
254                 qunbufcall(q, id);
255         }


258         /*
259          * XXX: Should this module really control the hi/low water marks?
260          * Is there any reason in this code to do so?
261          */
262         mop->b_datap->db_type = M_SETOPTS;
263         mop->b_wptr += sizeof (struct stroptions);
264         sop = (struct stroptions *)mop->b_rptr;
265         sop->so_flags = SO_HIWAT | SO_LOWAT;
266         sop->so_hiwat = 512;
267         sop->so_lowat = 256;

269         /*
270          * Commit to the open and send the M_SETOPTS off to the stream head.
271          */
272         putnext(q, mop);

274         return (0);
275 }
```

*unchanged_portion_omitted_*

****
49447 Wed Aug 19 20:52:10 2015
new/usr/src/uts/common/rpc/sec_gss/svc_rpcsec_gss.c
6141 use kmem_zalloc instead of kmem_alloc + bzero/memset
****
_____***unchanged_portion_omitted_***

```
 918 static enum auth_stat
 919 rpcsec_gss_init(
 920         struct svc_req          *rqst,
 921         struct rpc_msg          *msg,
 922         rpc_gss_creds           creds,
 923         bool_t                  *no_dispatch,
 924         svc_rpc_gss_data        *c_d) /* client data, can be NULL */
 925 {
 926         svc_rpc_gss_data        *client_data;
 927         int ret;
 928         svcrpcsec_gss_taskq_arg_t *arg;

 930         if (creds.ctx_handle.length != 0) {
 931                 RPCGSS_LOG0(1, "_svcrpcsec_gss: ctx_handle not null\n");
 932                 ret = AUTH_BADCRED;
 933                 return (ret);
 934         }

 936         client_data = c_d ? c_d : create_client();
 937         if (client_data == NULL) {
 938                 RPCGSS_LOG0(1,
 939                     "_svcrpcsec_gss: can't create a new cache entry\n");
 940                 ret = AUTH_FAILED;
 941                 return (ret);
 942         }

 944         mutex_enter(&client_data->clm);
 945         if (client_data->stale) {
 946                 ret = RPCSEC_GSS_NOCRED;
 947                 RPCGSS_LOG0(1, "_svcrpcsec_gss: client data stale\n");
 948                 goto error2;
 949         }

 951         /*
 952          * kgss_accept_sec_context()/gssd(1M) can be overly time
 953          * consuming so let's queue it and return asap.
 954          *
 955          * taskq func must free arg.
 956          */
 957         arg = kmem_alloc(sizeof (*arg), KM_SLEEP);

 959         /* taskq func must free rpc_call_arg & deserialized arguments */
 960         arg->rpc_call_arg = kmem_zalloc(sizeof (*arg->rpc_call_arg), KM_SLEEP);
 960         arg->rpc_call_arg = kmem_alloc(sizeof (*arg->rpc_call_arg), KM_SLEEP);

 962         /* deserialize arguments */
 963         bzero(arg->rpc_call_arg, sizeof (*arg->rpc_call_arg));
 963         if (!SVC_GETARGS(rqst->rq_xprt, __xdr_rpc_gss_init_arg,
 964             (caddr_t)arg->rpc_call_arg)) {
 965                 ret = RPCSEC_GSS_FAILED;
 966                 client_data->stale = TRUE;
 967                 goto error2;
 968         }

 970         /* get a xprt clone for taskq thread, taskq func must free it */
 971         arg->rq_xprt = svc_clone_init();
 972         svc_clone_link(rqst->rq_xprt->xp_master, arg->rq_xprt, rqst->rq_xprt);
 973         arg->rq_xprt->xp_xid = rqst->rq_xprt->xp_xid;
```



```
 976         /* set the appropriate wrap/unwrap routine for RPCSEC_GSS */
 977         arg->rq_xprt->xp_auth.svc_ah_ops = svc_rpc_gss_ops;
 978         arg->rq_xprt->xp_auth.svc_ah_private = (caddr_t)client_data;

 980         /* get a dup of rpc msg for taskq thread */
 981         arg->msg = rpc_msg_dup(msg);  /* taskq func must free msg dup */

 983         arg->client_data = client_data;
 984         arg->cr_version = creds.version;
 985         arg->cr_service = creds.service;

 987         /* We no longer need the xp_xdrin, destroy it all here. */
 988         XDR_DESTROY(&(rqst->rq_xprt->xp_xdrin));

 990         /* should be ok to hold clm lock as taskq will have new thread(s) */
 991         ret = ddi_taskq_dispatch(svcrpcsec_gss_init_taskq,
 992             svcrpcsec_gss_taskq_func, arg, DDI_SLEEP);
 993         if (ret == DDI_FAILURE) {
 994                 cmn_err(CE_NOTE, "rpcsec_gss_init: taskq dispatch fail");
 995                 ret = RPCSEC_GSS_FAILED;
 996                 rpc_msg_free(&arg->msg, MAX_AUTH_BYTES);
 997                 svc_clone_unlink(arg->rq_xprt);
 998                 svc_clone_free(arg->rq_xprt);
 999                 kmem_free(arg, sizeof (*arg));
1000                 goto error2;
1001         }

1003         mutex_exit(&client_data->clm);
1004         *no_dispatch = TRUE;
1005         return (AUTH_OK);

1007 error2:
1008         ASSERT(client_data->ref_cnt > 0);
1009         client_data->ref_cnt--;
1010         mutex_exit(&client_data->clm);
1011         cmn_err(CE_NOTE, "rpcsec_gss_init: error 0x%x", ret);
1012         return (ret);
1013 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   28925 Wed Aug 19 20:52:10 2015
new/usr/src/uts/common/syscall/rctlsys.c
6141 use kmem_zalloc instead of kmem_alloc + bzero/memset
**********************************************************
______unchanged_portion_omitted_

 202 /*
 203  * static long rctlsys_get(char *name, rctl_opaque_t *old_rblk,
 204  *   rctl_opaque_t *new_rblk, int flags)
 205  *
 206  * Overview
 207  *   rctlsys_get() is the implementation of the core logic of getrctl(2), the
 208  *   public system call for fetching resource control values.  Three mutually
 209  *   exclusive flag values are supported: RCTL_USAGE, RCTL_FIRST and RCTL_NEXT.
 210  *   When RCTL_USAGE is presented, the current usage for the resource control
 211  *   is returned in new_blk if the resource control provides an implementation
 212  *   of the usage operation.  When RCTL_FIRST is presented, the value of
 213  *   old_rblk is ignored, and the first value in the resource control value
 214  *   sequence for the named control is transformed and placed in the user
 215  *   memory location at new_rblk.  In the RCTL_NEXT case, the value of old_rblk
 216  *   is examined, and the next value in the sequence is transformed and placed
 217  *   at new_rblk.
 218  */
 219 static long
 220 rctlsys_get(char *name, rctl_opaque_t *old_rblk, rctl_opaque_t *new_rblk,
 221     int flags)
 222 {
 223         rctl_val_t *nval;
 224         rctl_opaque_t *nblk;
 225         rctl_hndl_t hndl;
 226         char *kname;
 227         size_t klen;
 228         rctl_dict_entry_t *krde;
 229         int ret;
 230         int action = flags & (~RCTLSYS_ACTION_MASK);

 232         if (flags & (~RCTLSYS_MASK))
 233                 return (set_errno(EINVAL));

 235         if (action != RCTL_FIRST && action != RCTL_NEXT &&
 236             action != RCTL_USAGE)
 237                 return (set_errno(EINVAL));

 239         if (new_rblk == NULL || name == NULL)
 240                 return (set_errno(EFAULT));

 242         kname = kmem_alloc(MAXPATHLEN, KM_SLEEP);
 243         krde = kmem_alloc(sizeof (rctl_dict_entry_t), KM_SLEEP);

 245         if (copyinstr(name, kname, MAXPATHLEN, &klen) != 0) {
 246                 kmem_free(kname, MAXPATHLEN);
 247                 kmem_free(krde, sizeof (rctl_dict_entry_t));
 248                 return (set_errno(EFAULT));
 249         }

 251         if ((hndl = rctl_hndl_lookup(kname)) == -1) {
 252                 kmem_free(kname, MAXPATHLEN);
 253                 kmem_free(krde, sizeof (rctl_dict_entry_t));
 254                 return (set_errno(EINVAL));
 255         }

 257         if (rctl_global_get(kname, krde) == -1) {
 258                 kmem_free(kname, MAXPATHLEN);
 259                 kmem_free(krde, sizeof (rctl_dict_entry_t));
 260                 return (set_errno(ESRCH));
```



```
 261         }

 263         kmem_free(kname, MAXPATHLEN);

 265         if (action != RCTL_USAGE)
 266                 nval = kmem_cache_alloc(rctl_val_cache, KM_SLEEP);

 268         if (action == RCTL_USAGE) {
 269                 rctl_set_t *rset;
 270                 rctl_t *rctl;
 271                 rctl_qty_t usage;

 273                 mutex_enter(&curproc->p_lock);
 274                 if ((rset = rctl_entity_obtain_rset(krde, curproc)) == NULL) {
 275                         mutex_exit(&curproc->p_lock);
 276                         kmem_free(krde, sizeof (rctl_dict_entry_t));
 277                         return (set_errno(ESRCH));
 278                 }
 279                 mutex_enter(&rset->rcs_lock);
 280                 if (rctl_set_find(rset, hndl, &rctl) == -1) {
 281                         mutex_exit(&rset->rcs_lock);
 282                         mutex_exit(&curproc->p_lock);
 283                         kmem_free(krde, sizeof (rctl_dict_entry_t));
 284                         return (set_errno(ESRCH));
 285                 }
 286                 if (RCTLOP_NO_USAGE(rctl)) {
 287                         mutex_exit(&rset->rcs_lock);
 288                         mutex_exit(&curproc->p_lock);
 289                         kmem_free(krde, sizeof (rctl_dict_entry_t));
 290                         return (set_errno(ENOTSUP));
 291                 }
 292                 usage = RCTLOP_GET_USAGE(rctl, curproc);
 293                 mutex_exit(&rset->rcs_lock);
 294                 mutex_exit(&curproc->p_lock);

 296                 nblk = kmem_zalloc(sizeof (rctl_opaque_t), KM_SLEEP);
 296                 nblk = kmem_alloc(sizeof (rctl_opaque_t), KM_SLEEP);
 297                 bzero(nblk, sizeof (rctl_opaque_t));
 297                 nblk->rcq_value = usage;

 299                 ret = copyout(nblk, new_rblk, sizeof (rctl_opaque_t));
 300                 kmem_free(nblk, sizeof (rctl_opaque_t));
 301                 kmem_free(krde, sizeof (rctl_dict_entry_t));
 302                 return (ret == 0 ? 0 : set_errno(EFAULT));
 303         } else if (action == RCTL_FIRST) {

 305                 mutex_enter(&curproc->p_lock);
 306                 if (ret = rctl_local_get(hndl, NULL, nval, curproc)) {
 307                         mutex_exit(&curproc->p_lock);
 308                         kmem_cache_free(rctl_val_cache, nval);
 309                         kmem_free(krde, sizeof (rctl_dict_entry_t));
 310                         return (set_errno(ret));
 311                 }
 312                 mutex_exit(&curproc->p_lock);
 313         } else {
 314                 /*
 315                  * RCTL_NEXT
 316                  */
 317                 rctl_val_t *oval;
 318                 rctl_opaque_t *oblk;

 320                 oblk = kmem_alloc(sizeof (rctl_opaque_t), KM_SLEEP);

 322                 if (copyin(old_rblk, oblk, sizeof (rctl_opaque_t)) == -1) {
 323                         kmem_cache_free(rctl_val_cache, nval);
 324                         kmem_free(oblk, sizeof (rctl_opaque_t));
```

```
325                         kmem_free(krde, sizeof (rctl_dict_entry_t));
326                         return (set_errno(EFAULT));
327                 }

329                 oval = kmem_cache_alloc(rctl_val_cache, KM_SLEEP);

331                 rctlsys_rblk_xfrm(oblk, NULL, oval, RBX_FROM_BLK | RBX_VAL);
332                 mutex_enter(&curproc->p_lock);
333                 ret = rctl_local_get(hndl, oval, nval, curproc);
334                 mutex_exit(&curproc->p_lock);

336                 kmem_cache_free(rctl_val_cache, oval);
337                 kmem_free(oblk, sizeof (rctl_opaque_t));

339                 if (ret != 0) {
340                         kmem_cache_free(rctl_val_cache, nval);
341                         kmem_free(krde, sizeof (rctl_dict_entry_t));
342                         return (set_errno(ret));
343                 }
344         }

346         nblk = kmem_alloc(sizeof (rctl_opaque_t), KM_SLEEP);

348         rctlsys_rblk_xfrm(nblk, krde, nval, RBX_TO_BLK | RBX_VAL | RBX_CTL);

350         kmem_free(krde, sizeof (rctl_dict_entry_t));
351         kmem_cache_free(rctl_val_cache, nval);

353         if (copyout(nblk, new_rblk, sizeof (rctl_opaque_t)) == -1) {
354                 kmem_free(nblk, sizeof (rctl_opaque_t));
355                 return (set_errno(EFAULT));
356         }

358         kmem_free(nblk, sizeof (rctl_opaque_t));

360         return (0);
361 }
```

_____***unchanged_portion_omitted_***