```
**********************************************************
   15093 Fri Jun  6 16:24:05 2014
new/usr/src/lib/pkcs11/pkcs11_softtoken/common/softGeneral.c
4912 soft_giant_mutex in pkcs11_softtoken stays held after fork
**********************************************************
/*
 * 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 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright 2014, OmniTI Computer Consulting, Inc. All rights reserved.
 */

#include <strings.h>
#include <errno.h>
#include <cryptoutil.h>
#include <unistd.h> /* for pid_t */
#include <pthread.h>
#include <security/cryptoki.h>
#include "softGlobal.h"
#include "softSession.h"
#include "softObject.h"
#include "softKeystore.h"
#include "softKeystoreUtil.h"

#pragma init(softtoken_init)
#pragma fini(softtoken_fini)

extern soft_session_t token_session; /* for fork handler */

static struct CK_FUNCTION_LIST functionList = {
        { 2, 20 },      /* version */
        C_Initialize,
        C_Finalize,
        C_GetInfo,
        C_GetFunctionList,
        C_GetSlotList,
        C_GetSlotInfo,
        C_GetTokenInfo,
        C_GetMechanismList,
        C_GetMechanismInfo,
        C_InitToken,
        C_InitPIN,
        C_SetPIN,
        C_OpenSession,
        C_CloseSession,
        C_CloseAllSessions,
```



```
        C_GetSessionInfo,
        C_GetOperationState,
        C_SetOperationState,
        C_Login,
        C_Logout,
        C_CreateObject,
        C_CopyObject,
        C_DestroyObject,
        C_GetObjectSize,
        C_GetAttributeValue,
        C_SetAttributeValue,
        C_FindObjectsInit,
        C_FindObjects,
        C_FindObjectsFinal,
        C_EncryptInit,
        C_Encrypt,
        C_EncryptUpdate,
        C_EncryptFinal,
        C_DecryptInit,
        C_Decrypt,
        C_DecryptUpdate,
        C_DecryptFinal,
        C_DigestInit,
        C_Digest,
        C_DigestUpdate,
        C_DigestKey,
        C_DigestFinal,
        C_SignInit,
        C_Sign,
        C_SignUpdate,
        C_SignFinal,
        C_SignRecoverInit,
        C_SignRecover,
        C_VerifyInit,
        C_Verify,
        C_VerifyUpdate,
        C_VerifyFinal,
        C_VerifyRecoverInit,
        C_VerifyRecover,
        C_DigestEncryptUpdate,
        C_DecryptDigestUpdate,
        C_SignEncryptUpdate,
        C_DecryptVerifyUpdate,
        C_GenerateKey,
        C_GenerateKeyPair,
        C_WrapKey,
        C_UnwrapKey,
        C_DeriveKey,
        C_SeedRandom,
        C_GenerateRandom,
        C_GetFunctionStatus,
        C_CancelFunction,
        C_WaitForSlotEvent
};

boolean_t softtoken_initialized = B_FALSE;

static pid_t softtoken_pid = 0;

/* This mutex protects soft_session_list, all_sessions_closing */
pthread_mutex_t soft_sessionlist_mutex;
soft_session_t *soft_session_list = NULL;

int all_sessions_closing = 0;

slot_t soft_slot;
```

```
obj_to_be_freed_list_t obj_delay_freed;
ses_to_be_freed_list_t ses_delay_freed;

/* protects softtoken_initialized and access to C_Initialize/C_Finalize */
pthread_mutex_t soft_giant_mutex = PTHREAD_MUTEX_INITIALIZER;
/*
 * ONLY SET TO TRUE BY softtoken_fork_prepare().
 * ONLY SET TO FALSE BY softtoken_fork_after().
 */
boolean_t fork_prepared = B_FALSE;

static CK_RV finalize_common(boolean_t force, CK_VOID_PTR pReserved);
static void softtoken_init();
static void softtoken_fini();
static void softtoken_fork_prepare();
static void softtoken_fork_after();
static void softtoken_fork_after_droplocks(void);

CK_RV
C_Initialize(CK_VOID_PTR pInitArgs)
{

        int initialize_pid;
        boolean_t supplied_ok;
        CK_RV rv;

        /*
         * Get lock to insure only one thread enters this
         * function at a time.
         */
        (void) pthread_mutex_lock(&soft_giant_mutex);

        initialize_pid = getpid();

        if (softtoken_initialized) {
                if (initialize_pid == softtoken_pid) {
                        /*
                         * This process has called C_Initialize already
                         */
                        (void) pthread_mutex_unlock(&soft_giant_mutex);
                        return (CKR_CRYPTOKI_ALREADY_INITIALIZED);
                } else {
                        /*
                         * A fork has happened and the child is
                         * reinitializing.  Do a finalize_common to close
                         * out any state from the parent, and then
                         * continue on.
                         */
                        (void) finalize_common(B_TRUE, NULL);
                }
        }

        if (pInitArgs != NULL) {
                CK_C_INITIALIZE_ARGS *initargs1 =
                    (CK_C_INITIALIZE_ARGS *) pInitArgs;

                /* pReserved must be NULL */
                if (initargs1->pReserved != NULL) {
                        (void) pthread_mutex_unlock(&soft_giant_mutex);
                        return (CKR_ARGUMENTS_BAD);
                }

                /*
                 * ALL supplied function pointers need to have the value
                 * either NULL or non-NULL.
                 */
```



```
                supplied_ok = (initargs1->CreateMutex == NULL &&
                    initargs1->DestroyMutex == NULL &&
                    initargs1->LockMutex == NULL &&
                    initargs1->UnlockMutex == NULL) ||
                    (initargs1->CreateMutex != NULL &&
                    initargs1->DestroyMutex != NULL &&
                    initargs1->LockMutex != NULL &&
                    initargs1->UnlockMutex != NULL);

                if (!supplied_ok) {
                        (void) pthread_mutex_unlock(&soft_giant_mutex);
                        return (CKR_ARGUMENTS_BAD);
                }

                /*
                 * When the CKF_OS_LOCKING_OK flag isn't set and mutex
                 * function pointers are supplied by an application,
                 * return an error.  We must be able to use our own primitives.
                 */
                if (!(initargs1->flags & CKF_OS_LOCKING_OK) &&
                    (initargs1->CreateMutex != NULL)) {
                        (void) pthread_mutex_unlock(&soft_giant_mutex);
                        return (CKR_CANT_LOCK);
                }
        }

        /* Initialize the session list lock */
        if (pthread_mutex_init(&soft_sessionlist_mutex, NULL) != 0) {
                (void) pthread_mutex_unlock(&soft_giant_mutex);
                return (CKR_CANT_LOCK);
        }

        /*
         * token object related initialization
         */
        soft_slot.authenticated = 0;
        soft_slot.userpin_change_needed = 0;
        soft_slot.token_object_list = NULL;
        soft_slot.keystore_load_status = KEYSTORE_UNINITIALIZED;

        if ((rv = soft_init_token_session()) != CKR_OK) {
                (void) pthread_mutex_destroy(&soft_sessionlist_mutex);
                (void) pthread_mutex_unlock(&soft_giant_mutex);
                return (rv);
        }

        /* Initialize the slot lock */
        if (pthread_mutex_init(&soft_slot.slot_mutex, NULL) != 0) {
                (void) pthread_mutex_destroy(&soft_sessionlist_mutex);
                (void) soft_destroy_token_session();
                (void) pthread_mutex_unlock(&soft_giant_mutex);
                return (CKR_CANT_LOCK);
        }

        /* Initialize the keystore lock */
        if (pthread_mutex_init(&soft_slot.keystore_mutex, NULL) != 0) {
                (void) pthread_mutex_destroy(&soft_slot.slot_mutex);
                (void) pthread_mutex_destroy(&soft_sessionlist_mutex);
                (void) soft_destroy_token_session();
                (void) pthread_mutex_unlock(&soft_giant_mutex);
                return (CKR_CANT_LOCK);
        }

        /* Initialize the object_to_be_freed list */
        if (pthread_mutex_init(&obj_delay_freed.obj_to_be_free_mutex, NULL)
            != 0) {
```

```
260                 (void) pthread_mutex_destroy(&soft_slot.keystore_mutex);
261                 (void) pthread_mutex_destroy(&soft_slot.slot_mutex);
262                 (void) pthread_mutex_destroy(&soft_sessionlist_mutex);
263                 (void) soft_destroy_token_session();
264                 (void) pthread_mutex_unlock(&soft_giant_mutex);
265                 return (CKR_CANT_LOCK);
266         }
267         obj_delay_freed.count = 0;
268         obj_delay_freed.first = NULL;
269         obj_delay_freed.last = NULL;

271         if (pthread_mutex_init(&ses_delay_freed.ses_to_be_free_mutex, NULL)
272             != 0) {
273                 (void) pthread_mutex_destroy(
274                     &obj_delay_freed.obj_to_be_free_mutex);
275                 (void) pthread_mutex_destroy(&soft_slot.keystore_mutex);
276                 (void) pthread_mutex_destroy(&soft_slot.slot_mutex);
277                 (void) pthread_mutex_destroy(&soft_sessionlist_mutex);
278                 (void) soft_destroy_token_session();
279                 (void) pthread_mutex_unlock(&soft_giant_mutex);
280                 return (CKR_CANT_LOCK);
281         }
282         ses_delay_freed.count = 0;
283         ses_delay_freed.first = NULL;
284         ses_delay_freed.last = NULL;

286         if (rv != CKR_OK) {
287                 (void) pthread_mutex_destroy(
288                     &ses_delay_freed.ses_to_be_free_mutex);
289                 (void) pthread_mutex_destroy(
290                     &obj_delay_freed.obj_to_be_free_mutex);
291                 (void) pthread_mutex_destroy(&soft_slot.keystore_mutex);
292                 (void) pthread_mutex_destroy(&soft_slot.slot_mutex);
293                 (void) pthread_mutex_destroy(&soft_sessionlist_mutex);
294                 (void) soft_destroy_token_session();
295                 (void) pthread_mutex_unlock(&soft_giant_mutex);
296                 return (CKR_FUNCTION_FAILED);
297         }

299         softtoken_pid = initialize_pid;
300         softtoken_initialized = B_TRUE;
301         (void) pthread_mutex_unlock(&soft_giant_mutex);

303         return (CKR_OK);
304 }
```

*unchanged_portion_omitted_*

```
415 /*
416  * softtoken_fini() function required to make sure complete cleanup
417  * is done if softtoken is ever unloaded without a C_Finalize() call.
418  */
419 static void
420 softtoken_fini()
421 {
422         int rc;

424         rc = pthread_mutex_trylock(&soft_giant_mutex);
425         /*
426          * Check to see if it was acquired by softtoken_fork_prepare().
427          */
428         if (rc == EBUSY && fork_prepared)
429                 rc = 0;

431         if (rc != 0) {
432                 /*
433                  * I don't know WHAT just happened, but it's pretty bad.  Grab
```



```
434                  * the lock for real, even if it means deadlocking here.
435                  */
436                 (void) pthread_mutex_lock(&soft_giant_mutex);
437         } /* Else we acquired the lock and life is good. */

439         /* Only finalize if we're initialized. */
440         if (softtoken_initialized) {
441                 /*
442                  * We also have to DROP all of the other locks that
443                  * softtoken_fork_prepare did IF it was prepared, because
444                  * finalize_common() is going to pick them right back up.
445                  */
446                 if (fork_prepared)
447                         softtoken_fork_after_droplocks();
448                 (void) finalize_common(B_TRUE, NULL_PTR);
449                 /* ASSERT(!softtoken_initialized); */
416         /* if we're not initilized, do not attempt to finalize */
417         if (!softtoken_initialized) {
418                 (void) pthread_mutex_unlock(&soft_giant_mutex);
419                 return;
450         }

452         /*
453          * If we don't want to drop the lock, we assume softtoken_fork_after()
454          * will get called.
455          */
456         if (!fork_prepared)
422         (void) finalize_common(B_TRUE, NULL_PTR);

457                 (void) pthread_mutex_unlock(&soft_giant_mutex);
458 }
```

*unchanged_portion_omitted_*

```
518 /*
519  * Take out all mutexes before fork.
520  *
521  * Order:
522  * 1. soft_giant_mutex
523  * 2. soft_sessionlist_mutex
524  * 3. soft_slot.slot_mutex
525  * 4. soft_slot.keystore_mutex
526  * 5. token_session mutexes via soft_acquire_all_session_mutexes()
527  * 6. all soft_session_list mutexes via soft_acquire_all_session_mutexes()
528  * 7. obj_delay_freed.obj_to_be_free_mutex;
529  * 8. ses_delay_freed.ses_to_be_free_mutex
530  */
531 void
532 softtoken_fork_prepare(void)
499 softtoken_fork_prepare()
533 {
534         (void) pthread_mutex_lock(&soft_giant_mutex);
535         if (softtoken_initialized) {
536                 (void) pthread_mutex_lock(&soft_sessionlist_mutex);
537                 (void) pthread_mutex_lock(&soft_slot.slot_mutex);
538                 (void) pthread_mutex_lock(&soft_slot.keystore_mutex);
539                 soft_acquire_all_session_mutexes(&token_session);
540                 soft_acquire_all_session_mutexes(soft_session_list);
541                 (void) pthread_mutex_lock(
542                     &obj_delay_freed.obj_to_be_free_mutex);
543                 (void) pthread_mutex_lock(
544                     &ses_delay_freed.ses_to_be_free_mutex);
545         }
546         fork_prepared = B_TRUE;
547 }

549 static void
```

```
550 softtoken_fork_after_droplocks(void)
551 {
552         /* ASSERT(softtoken_initialized); */
553         (void) pthread_mutex_unlock(&ses_delay_freed.ses_to_be_free_mutex);
554         (void) pthread_mutex_unlock(&obj_delay_freed.obj_to_be_free_mutex);
555         soft_release_all_session_mutexes(soft_session_list);
556         soft_release_all_session_mutexes(&token_session);
557         (void) pthread_mutex_unlock(&soft_slot.keystore_mutex);
558         (void) pthread_mutex_unlock(&soft_slot.slot_mutex);
559         (void) pthread_mutex_unlock(&soft_sessionlist_mutex);
560 }

562 /*
563  * Release in opposite order to softtoken_fork_prepare().
564  * Function is used for parent and child.
565  */
566 void
567 softtoken_fork_after(void)
520 softtoken_fork_after()
568 {
569         if (softtoken_initialized)
570                 softtoken_fork_after_droplocks();
571         fork_prepared = B_FALSE;
522         if (softtoken_initialized) {
523                 (void) pthread_mutex_unlock(
524                     &ses_delay_freed.ses_to_be_free_mutex);
525                 (void) pthread_mutex_unlock(
526                     &obj_delay_freed.obj_to_be_free_mutex);
527                 soft_release_all_session_mutexes(soft_session_list);
528                 soft_release_all_session_mutexes(&token_session);
529                 (void) pthread_mutex_unlock(&soft_slot.keystore_mutex);
530                 (void) pthread_mutex_unlock(&soft_slot.slot_mutex);
531                 (void) pthread_mutex_unlock(&soft_sessionlist_mutex);
532         }
572         (void) pthread_mutex_unlock(&soft_giant_mutex);
573 }
```

*_____unchanged_portion_omitted_*