GOS2022

GOS2022

Mutex Service Unit Tests

GOS2022

Mutex Service Unit Tests

for GOS2022 version 0.6

Description of the mutex service unit testing

List of test cases

Version history

Version	Date	Author	Change	Released
1.0	2023-11-10	Ahmed Gazar	Initial version of the document.	2023-11-10

Content

1. Introduction

Unit testing serves the purpose of checking small units of code (in this case the API functions of the service) to make sure that they behave as expected.

The unit tests of the mutex service cover its API functions and test the behaviour of those functions.

2. Test cases (specification)

2.1. Mutex initialization test

Test case	MutexInitTest		
Pre-conditions	Local mutex instance		
Step no.	Description	Expected result	Assert point
1.	Call gos_mutexInit passing a NULL pointer	GOS_ERROR	**Yes**
2.	Call gos_mutexInit passing the pointer to the local mutex instance	GOS_SUCCESS	**Yes**
3.	Check the mutex state field of the local mutex instance	GOS_MUTEX_UNLOCKED	**Yes**
4.	Check the owner field of the local mutex instance	GOS_INVALID_TASK_ID	**Yes**

2.2. Mutex lock test

Test case	MutexLockTest		
Pre-conditions	Initialized local mutex instance Elapsed time measurement in [ms]		
Step no.	Description	Expected result	Assert point
1.	Call gos_mutexLock passing the pointer to the local mutex instance with GOS_MUTEX_NO_TMO	GOS_SUCCESS	**Yes**
2.	Check the elapsed time since the lock call	< 5	**Yes**
3.	Call gos_mutexUnlock passing the pointer to the local mutex instance	-	No
4.	Call gos_mutexLock passing the pointer to the local mutex instance with GOS_MUTEX_ENDLESS_TMO	GOS_SUCCESS	**Yes**
5.	Call gos_mutexUnlock passing the pointer to the local mutex instance	-	No
6.	Call gos_mutexLock passing the pointer to the local mutex instance with 500 ms timeout	GOS_SUCCESS	**Yes**
7.	Check the elapsed time since the lock call	< 5	**Yes**
8.	Call gos_mutexUnlock passing the pointer to the local mutex instance	-	No
9.	Call gos_mutexLock passing the pointer to the local mutex instance with GOS_MUTEX_NO_TMO	GOS_SUCCESS	**Yes**

10.	Call gos_mutexLock passing the pointer to the local mutex instance with GOS_MUTEX_NO_TMO	GOS_ERROR	**Yes**
11.	Check the elapsed time since Step 10	< 5	**Yes**
12.	Call gos_mutexLock passing the pointer to the local mutex instance with 500 ms timeout	GOS_ERROR	**Yes**
13.	Check the elapsed time since Step 12	>= 500 **AND** <= 505	**Yes**
14.	Call gos_mutexUnlock passing the pointer to the local mutex instance	-	No
15.	Call gos_mutexLock passing the pointer to the local mutex instance with GOS_MUTEX_NO_TMO	GOS_SUCCESS	**Yes**
16.	Call gos_mutexUnlock passing the pointer to the local mutex instance	-	No
17.	Call gos_mutexLock passing a NULL pointer with GOS_MUTEX_NO_TMO	GOS_ERROR	**Yes**
18.	Call gos_mutexLock passing a NULL pointer with GOS_MUTEX_ENDLESS_TMO	GOS_ERROR	**Yes**
19.	Call gos_mutexLock passing a NULL pointer with 500 ms timeout	GOS_ERROR	**Yes**

2.3. Mutex unlock test

Test case	MutexUnlockTest		
Pre-conditions	Initialized global mutex instance (accessible from multiple tasks) Another task with the same priority as the main test task		
Step no.	Description	Expected result	Assert point
1.	Call gos_mutexLock passing the pointer to the global mutex instance with GOS_MUTEX_NO_TMO	-	No
2.	Call gos_mutexUnlock passing the pointer to the global mutex instance	GOS_SUCCESS	**Yes**
3.	Sleep the main task for 50 ms (let the side task run)	-	No
4.	Call gos_mutexLock passing the pointer to the global mutex instance with GOS_MUTEX_ENDLESS_TMO **in the side task**	-	No
5.	Sleep the side task for 200 ms (let the main task run)	-	No
6.	Check the state of the global mutex instance **in the main task**	GOS_MUTEX_LOCKED	**Yes**
7.	Check the owner of the global mutex instance **in the main task**	Task ID of side task	**Yes**
8.	Call gos_mutexUnlock passing the pointer to the global mutex instance **in the main task**	GOS_ERROR	**Yes**
9.	Sleep the main task for 250 ms (let the side task run)	-	No

10.	Call gos_mutexUnlock passing the pointer to the global mutex instance **in the side task**	GOS_SUCCESS	**Yes**
11.	Terminate the side task	-	No
12.	Check the state of the global mutex instance **in the main task**	GOS_MUTEX_UNLOCKED	**Yes**
13.	Call gos_mutexLock passing the pointer to the global mutex instance with 100 ms timeout **in the main task**	-	No
14.	Call gos_mutexUnlock passing the pointer to the global mutex instance **in the main task**	GOS_SUCCESS	**Yes**
15.	Call gos_mutexUnlock passing a NULL pointer **in the main task**	GOS_ERROR	**Yes**

3. Test environment

For the mutex service unit testing, the GOS Test Framework is used. The framework is implemented in a single header and source file. For more information, please read the documentation of the GOS Test Framework.

4. Test cases (implementation)

Based on Test cases (specification), the following test cases are implemented:

4.1. Mutex initialization test

```
TEST_CASE(MutexInitTest)
{
    /*
     * Local variables.
     */
    gos_mutex_t testMutex;

    /*
     * Function code.
     */
    TEST_BEGIN
    {
            // Nothing to prepare.
    }

    TEST
    {
            /*
             * Initialization with NULL pointer.
             */
            TEST_ASSERT(gos_mutexInit(NULL) == GOS_ERROR);

            /*
             * Normal initialization.
             */
            TEST_ASSERT(gos_mutexInit(&testMutex) == GOS_SUCCESS);

            /*
             * Mutex state after initialization shall be unlocked.
             */
            TEST_ASSERT(testMutex.mutexState == GOS_MUTEX_UNLOCKED);

            /*
             * Mutex owner shall be invalid task ID.
             */
            TEST_ASSERT(testMutex.owner == GOS_INVALID_TASK_ID);

    }
}
TEST_END
```

4.2. Mutex lock test

```
TEST_CASE(MutexLockTest)
{
    /*
     * Local variables.
     */
    gos_mutex_t testMutex;
    u32_t       sysTicks;

    /*
     * Function code.
     */
    TEST_BEGIN
    {
            (void_t) gos_mutexInit(&testMutex);
    }

    TEST
    {
            /*
             * Try zero timeout.
             * Timeout value shall also be checked:
             * - Elapsed time shall be less than mutex sleep time -> 5ms.
             */
            sysTicks = gos_kernelGetSysTicks();

            TEST_ASSERT(gos_mutexLock(&testMutex, GOS_MUTEX_NO_TMO) == GOS_SUCCESS);
            TEST_ASSERT((gos_kernelGetSysTicks() - sysTicks) < 5);

            /*
             * Unlock mutex.
             */
            (void_t) gos_mutexUnlock(&testMutex);

            /*
             * Try endless timeout.
             */
            TEST_ASSERT(gos_mutexLock(&testMutex, GOS_MUTEX_ENDLESS_TMO) == GOS_SUCCESS);

            /*
             * Unlock mutex.
             */
            (void_t) gos_mutexUnlock(&testMutex);

            /*
             * Try normal timeout.
             */
            sysTicks = gos_kernelGetSysTicks();

            TEST_ASSERT(gos_mutexLock(&testMutex, 500u) == GOS_SUCCESS);
            TEST_ASSERT((gos_kernelGetSysTicks() - sysTicks) < 5);

            /*
             * Unlock mutex.
             */
            (void_t) gos_mutexUnlock(&testMutex);

            /*
             * Lock mutex once.
             */
            TEST_ASSERT(gos_mutexLock(&testMutex, GOS_MUTEX_NO_TMO) == GOS_SUCCESS);

            /*
             * Try locking it again with zero timeout.
             * Timeout value shall also be checked:
             * - Elapsed time shall be less than mutex sleep time -> 5ms.
             */
            sysTicks = gos_kernelGetSysTicks();

            TEST_ASSERT(gos_mutexLock(&testMutex, GOS_MUTEX_NO_TMO) == GOS_ERROR);
            TEST_ASSERT((gos_kernelGetSysTicks() - sysTicks) < 5);

            /*
             * Try locking it again with normal timeout.
             * Timeout value shall also be checked:
             * - Elapsed time shall be at least 500ms.
             * - Elapsed time shall be less than or equal to timeout + 1 x mutex sleep time ->
505ms.
             */
            sysTicks = gos_kernelGetSysTicks();

            TEST_ASSERT(gos_mutexLock(&testMutex, 500u) == GOS_ERROR);
            TEST_ASSERT((gos_kernelGetSysTicks() - sysTicks) >= 500 &&
(gos_kernelGetSysTicks() - sysTicks) <= 505);
```

```
            /*
             * Unlock mutex.
             */
            (void_t) gos_mutexUnlock(&testMutex);

            /*
             * Try to lock it again.
             */
            TEST_ASSERT(gos_mutexLock(&testMutex, GOS_MUTEX_NO_TMO) == GOS_SUCCESS);

            /*
             * Unlock mutex.
             */
            (void_t) gos_mutexUnlock(&testMutex);

            /*
             * Try with NULL pointer and no timeout.
             */
            TEST_ASSERT(gos_mutexLock(NULL, GOS_MUTEX_NO_TMO) == GOS_ERROR);

            /*
             * Try with NULL pointer and endless timeout.
             */
            TEST_ASSERT(gos_mutexLock(NULL, GOS_MUTEX_ENDLESS_TMO) == GOS_ERROR);

            /*
             * Try with NULL pointer and normal timeout.
             */
            TEST_ASSERT(gos_mutexLock(NULL, 500u) == GOS_ERROR);
    }
}
TEST_END
```

4.3. Mutex unlock test

```
TEST_CASE(MutexUnlockTest)
{
    /*
     * Function code.
     */
    TEST_BEGIN
    {
            (void_t) gos_mutexInit(&sideTestMutex);
    }

    TEST
    {
            /*
             * At this point the main task has the CPU (side task has not
             * been scheduled yet).
             * Lock and unlock from the same task.
             */
            (void_t) gos_mutexLock(&sideTestMutex, GOS_MUTEX_NO_TMO);
            TEST_ASSERT(gos_mutexUnlock(&sideTestMutex) == GOS_SUCCESS);

            /*
             * With the sleep call here we give the CPU to the side task.
             * Lock from side task and try to unlock from here.
             */
            (void_t) gos_taskSleep(50);

            TEST_ASSERT(sideTestMutex.mutexState == GOS_MUTEX_LOCKED);
            TEST_ASSERT(sideTestMutex.owner == mutexTestSideTaskDesc.taskId);
            TEST_ASSERT(gos_mutexUnlock(&sideTestMutex) == GOS_ERROR);

            /*
             * Try to lock again after side task has finished its operation.
             */
            (void_t) gos_taskSleep(250);
            TEST_ASSERT(sideTestMutex.mutexState == GOS_MUTEX_UNLOCKED);
            (void_t) gos_mutexLock(&sideTestMutex, 100);
            TEST_ASSERT(gos_mutexUnlock(&sideTestMutex) == GOS_SUCCESS);

            /*
             * Try with NULL pointer.
             */
            TEST_ASSERT(gos_mutexUnlock(NULL) == GOS_ERROR);
    }
}
TEST_END
```

4.4. Side task

```
GOS_STATIC gos_taskDescriptor_t mutexTestSideTaskDesc =
{
    .taskFunction       = MutexTestSideTask,
    .taskName           = "mutex_test_side_task",
    .taskPriority       = 0,
    .taskStackSize      = 0x300,
    .taskPrivilegeLevel = GOS_TASK_PRIVILEGE_SUPERVISOR
};

GOS_STATIC void_t MutexTestSideTask (void_t)
{
    /*
     * Function code.
     */
    (void_t) gos_mutexLock(&sideTestMutex, GOS_MUTEX_ENDLESS_TMO);
    (void_t) gos_taskSleep(200);
    TEST_ASSERT(gos_mutexUnlock(&sideTestMutex) == GOS_SUCCESS);
    (void_t) gos_taskDelete(mutexTestSideTaskDesc.taskId);

    for (;;);
}
```

4.5. Running the tests

```
void_t MutexTestRun (void_t)
{
    /*
     * Function code.
     */
    (void_t) gos_traceTrace(GOS_FALSE, "\r\nMutex test section\r\n");

    TEST_RUN(MutexInitTest);
    TEST_RUN(MutexLockTest);

    (void_t) gos_taskRegister(&mutexTestSideTaskDesc, NULL);
    TEST_RUN(MutexUnlockTest);
}
```

5. Test report

Report date: 2023-11-10 15:55
Run by: Ahmed Gazar

```
Mutex test section
--------------------------------
Test: MutexInitTest
Time: 00:00:00.172
Time taken: 00:00:00.000
Test [ PASS ]
--------------------------------
--------------------------------
Test: MutexLockTest
Time: 00:00:00.182
Time taken: 00:00:00.506
Test [ PASS ]
--------------------------------
--------------------------------
Test: MutexUnlockTest
Time: 00:00:00.693
Time taken: 00:00:00.300
Test [ PASS ]
--------------------------------
--------------------------------
TEST RESULT:
Time taken:     00:00:00.806
Test cases:     3
Passed:         3
Failed:         0
--------------------------------
```