

Freescal e MQX RTOS Example Guide

Semaphore_lite example

This document explains the semaphore_lite example, what to expect when running it and a brief introduction to the API.

The example

The semaphore example code shows how semaphore works. The code is written in a way that three different semaphores are synchronized to ensure mutual exclusion of a common memory space.

Running the example

The user only needs to do compilation of MQX libraries, ksdk library and the example without any further step.

In <MQX_folder>\rtos\mqx\config\mcu\<board>\mqx_sdk_config.h please set

```
#define MQX_USE_SEMAPHORES      1
#define MQX_USE_NAME            1
```

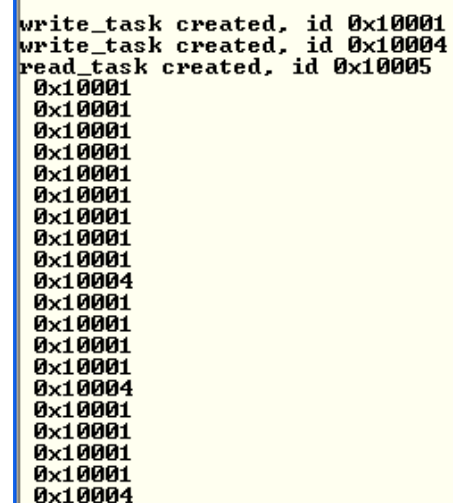
If the platform supports floating point, you have to disable floating point:

```
#define MQXCFG_ENABLE_FP        0
```

To run the example the corresponding IDE, compiler, debugger and a terminal program are needed.

Start a terminal application on your PC and set the serial connection for 115200 baud, 8 data bits, 1 stop bit, no parity and no flow control.

After running, the results will be as the picture below.

A screenshot of a terminal window with a yellow background and a blue border. The text is black and shows the output of a program. It starts with three lines: 'write_task created, id 0x10001', 'write_task created, id 0x10004', and 'read_task created, id 0x10005'. This is followed by a series of hexadecimal values: 0x10001, 0x10001, 0x10001, 0x10001, 0x10001, 0x10001, 0x10001, 0x10001, 0x10001, 0x10001, 0x10004, 0x10001, 0x10001, 0x10001, 0x10001, 0x10004, 0x10001, 0x10001, 0x10001, 0x10001, 0x10001, 0x10004.

```
write_task created, id 0x10001
write_task created, id 0x10004
read_task created, id 0x10005
0x10001
0x10001
0x10001
0x10001
0x10001
0x10001
0x10001
0x10001
0x10001
0x10001
0x10004
0x10001
0x10001
0x10001
0x10001
0x10004
0x10001
0x10001
0x10001
0x10001
0x10001
0x10004
```

Explaining the example

The application example creates three tasks with same priority and FIFO policy.

Main task

This task initializes three semaphores (write_sem, read_sem, index_sem), creates NUM_WRITERS write_task's, and creates one read_task.

Write task

This task opens a connection to all three semaphores then waits for sem.write and sem.index.

If the write_sem and index_sem are available, the write_task writes one entry in the data array (id of active write task) and posts sem.index and sem.read.

Read task

This task opens a connection to all three semaphores then waits for sem.read and sem.index.

If the read_sem and index_sem are available, the read_task will displays element in the data array. Sem.index and sem.write are then posted.

Flow chart

