Freescale MQX RTOS Example Guide
mutex example

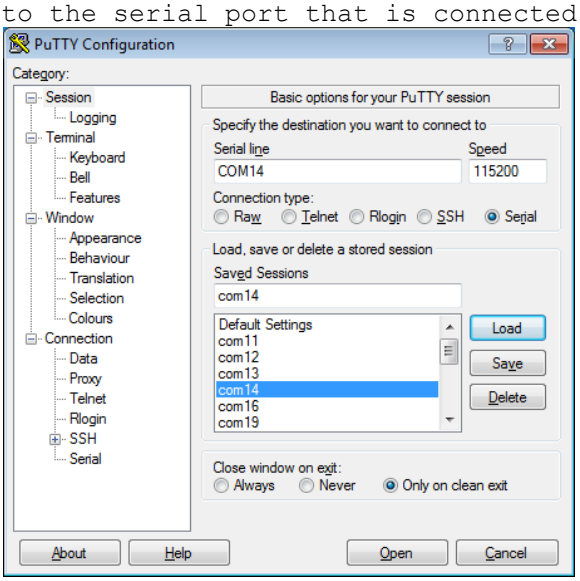
This document explains the mutex example, what to expect from the example and a brief introduction to the API.

The example

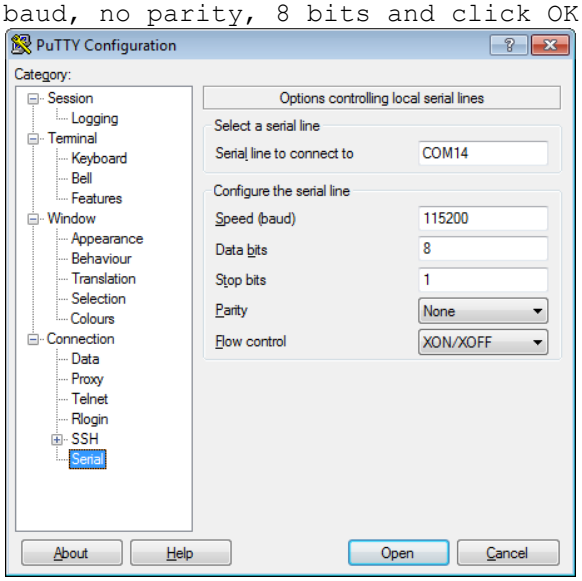
The mutex example code is used to demonstrate how to use a mutex to synchronize two tasks. It creates a mutex and two tasks. Both tasks use STDOUT to print out messages. Each task will lock the mutex before printing and unlock it after printing to ensure that the outputs from tasks are not mixed together.

Running the example

Start PuTTY on the.
Make a connection to the serial port that is connected to the board.



Set it for 115200 baud, no parity, 8 bits and click OK.



Then we compile the project mutex. When compiling there may be compiling error like this:

```
Error : preprocessor #error directive
main.c line 51 #error This application requires MQX_HAS_TIME_SLICE
defined non-zero in user_config.h Please recompile kernel with this
option.
```

In C:\Program Files\Freescale\Freescale MQX 3.x\config\common\small_ram_config.h please change

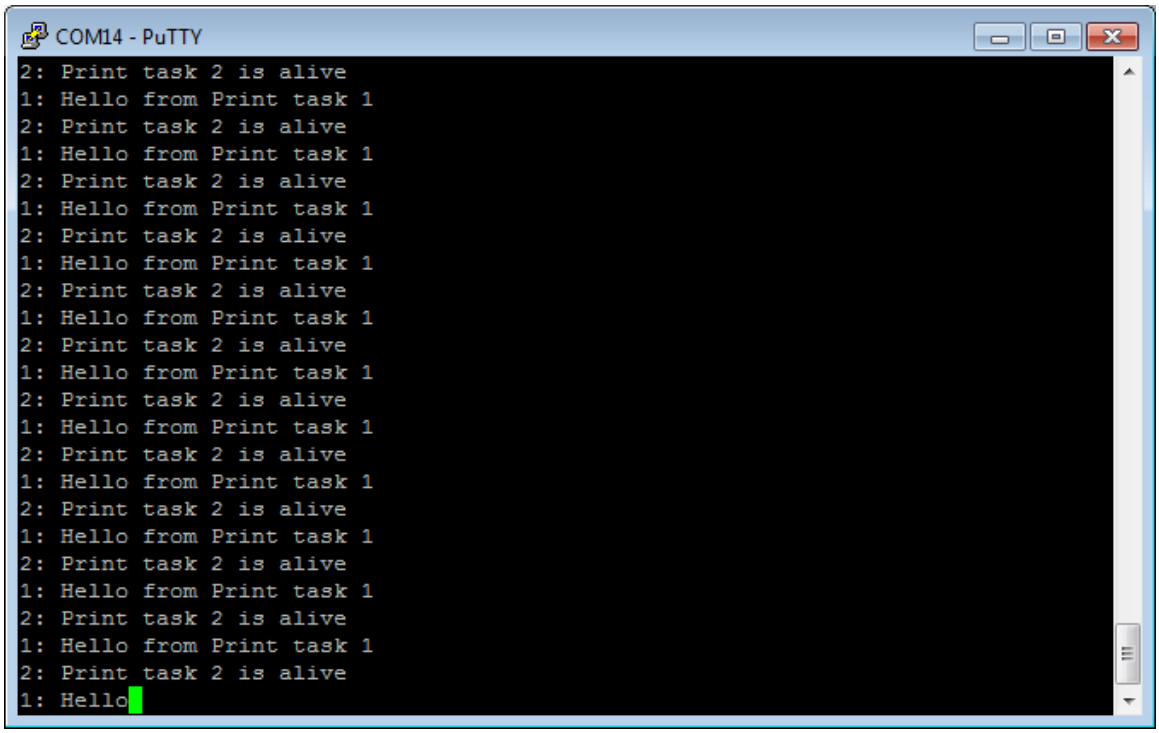
```
#define MQX_HAS_TIME_SLICE                  0
```

to

```
#define MQX_HAS_TIME_SLICE                  1
```

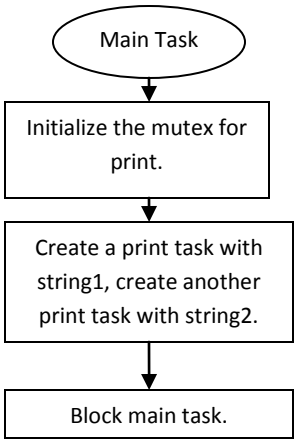
And rebuild the libraries bsp and psp. C:\Program Files\Freescale\Freescale MQX 3.7\mqx\build\cw10
This may take several minutes depending on the speed of the PC.

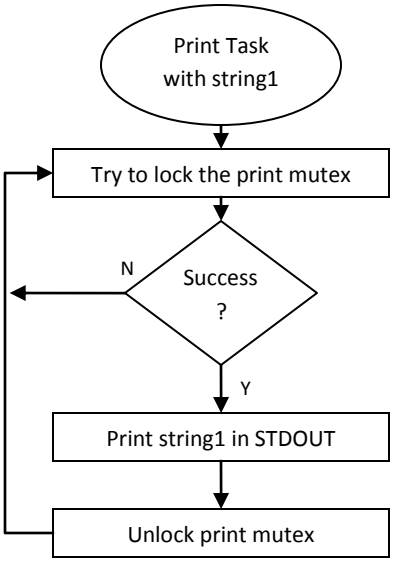
Then the mutex example can be compiled and downloaded to target board. We can see the running results displayed in the PuTTY:

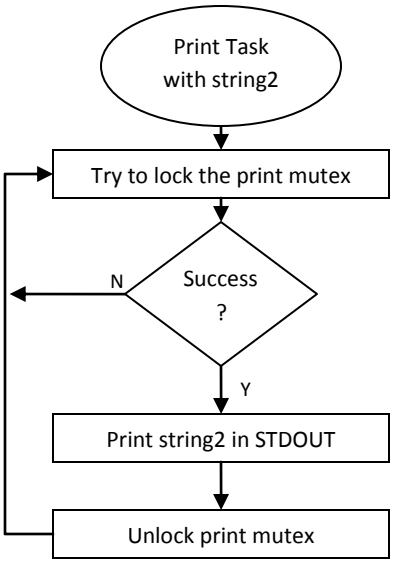


Explanation of the example

The application demo creates the main task first. The main task then creates two print tasks. The flows of the tasks are described in the next figure.







The main task first initializes the mutex with the following line:

```
_mutex_init(&print_mutex, &mutexattr)
```

Then the main task creates a print task with parameter sting1, and creates a second print task with parameter strings withthe following line:

```
_task_create(0, PRINT_TASK, (uint_32)string1);
_task_create(0, PRINT_TASK, (uint_32)string2);
```

Then the main task blocks itself.

The two print tasks both use the STDOUT. If they used it in the same time, there would be conflicts, so a mutex is used to synchronize the two tasks.

Each print task will try to lock the mutex before printing the message; it will wait for the mutex as long as needed. Once the mutex is locked it prints the message and then unlocks the mutex so that the other task can lock it. This process is repeated indefinitely.