Performance Engineering of Real-Time and Embedded Systems

QNX Primitives
QNX provides some very good documentation which you get to via Help → Help Contents.
You will probably want to read the Getting Started guide which comes from the Krten book.
If you need to write an interrupt handler look at a section in QNX Neutrino RTOS.

There is also a student tutorial available.
A section in the IDE info will help if you want to reduce the number of kernel events captured.
QNX is a POSIX-conformant operating system.

- Any POSIX tutorial that you find on the web can provide examples and information for you.
- QNX scheduling centers around processes and threads
  - You build an application that runs as a process in its own address space
  - Threads execute in the process’ address space
  - Each thread gets its own stack allocation
pthread_create is the function you use to create a new thread

```c
#include <pthread.h>

int pthread_create( pthread_t* thread,
                    const pthread_attr_t* attr,
                    void* (*start_routine)(void* ),
                    void* arg );
```

- There is a student tutorial on working with threads in an object-oriented environment.
- Naming threads is convenient for tracing

```c
int pthread_setname_np(pthread_t tid,
                      const char* newname);
```
QNX provides a standard set of POSIX-conformant synchronization primitives.

- Semaphores, mutual exclusion
- Message queues
- Timers, alarms
- nano_spin – kill CPU cycles without sleeping
If you need to change task priorities for the dynamic priority scheduling algorithms…

- This primitive function will be of interest to you.
  - `pthread_setschedparam()`
    Changes a number of scheduling parameters for a thread.

If you think you need to change the default timing resolution in QNX…

- These functions will be of interest to you.
  - `clock_getres()` Returns the current clock resolution.
  - `ClockPeriod(), ClockPeriod_r()` Get or set a clock period.
Timers will be the way that you can wake up a thread at a specific time.

- You can setup the timers to send a pulse to you, or to send you a SIGALRM signal
  - *The pulse you receive by checking MsgReceive*
  - *To receive a signal you specify a callback function as the signal handler*

- There is a student tutorial available for working with timers in QNX.
When working with threads, interrupt handlers, and signal handlers be aware of safety concerns.

- In this case, safety is from other threads or just whether it is legal to execute an operation in that context.
- The description for each OS primitive specifies its safe usage.
When you setup your kernel event logging you can control what data is captured.

Filter an entire class of events

Filter one type of event
The TraceEvent() function allows you to insert user events into the kernel event stream.

```
#include <sys/neutrino.h>
#include <sys/trace.h>

TraceEvent(_NTO_TRACE_INSERTSUSEREVENT,
           int event, int data0, int data1)
TraceEvent(_NTO_TRACE_INSERTCUSEREVENT,
           int event, unsigned * buf, unsigned len)
TraceEvent(_NTO_TRACE_INSERTUSRSTREVENT,
           int event, const char * str)
```

*event must be between _NTO_TRACE_USERFIRST and _NTO_TRACE_USERLAST*

Look at the examples in the System Analysis Toolkit guide.

You can add these statements to your application even if you set up kernel logging through Momentics.