QNX Priorities - You Aren’t in UNIX Anymore

Authored by: **Chris McKillop**
Updated by: **Thomas Fletcher**

Unlike many general purpose systems (like Linux, BSD, Windows), the Neutrino kernel uses a fixed priority system for determining when processes get to use the CPU(s). The more general systems tend to use a heuristic for figuring out when to run a process. It is usually based on such things as a process's current "nice" level, how long it ran in the past before blocking, how often it has been blocking and other internal measures. Now, this can provide a nice interactive system for multiple users. Unfortunately it does fails to provide consistent response time under heavy load or in circumstances not considered by their algorithms.

The Neutrino Kernel's fixed priority system, however, uses a process's priority alone to determine when it gets to run*****. This means that the highest priority process that is ready to run is the one that will be running, and it will continue to run until it blocks. A process blocks when it needs to wait for something to happen, or it gives up the processor on its own. When a process blocks it will be put at the end of the line of the processes waiting to run at the same priority.

So what does this mean to you? Under QNX Neutrino, processes will run at priority 10 by default (set up from the initial build files). So when you open a terminal and run a program, unless it takes steps to modify its priority, it will be running at priority 10. Now this is good for processes that behave themselves, but let’s look at what happens when a process doesn't behave. The following source code can be compiled into a program that will simply eat CPU time forever, printing out a dot every 10 million iterations.

```
/* cpuhog.c - chew up cpu time at its priority level */
#include <stdio.h>
int main( void )
{
    int count;
    count = 0;
    while( 1 )
    {
        if( ( count % 10000000 ) == 0 )
        {
            printf( "." );
            fflush( stdout );
        }
        count++;
    }
}
```

If you compile this little program and run it, you will find that pretty much everything in the system starts to run slow. If you are running in Photon, then you’ll notice that redraws will take longer and things will be generally less "responsive". In fact, the only reason you are able to interact with the system at all is that services like Photon and hardware drivers all run above priority 10, which can force the **cpuhog** to the end of the ready-to-run queue for priority 10.

Now, if you try running **cpuhog** at priority 9 by invoking...

```
"on -p 9 ./cpuhog"
```

Now the system will run normally and **cpuhog** will still print out dots at about the same rate as before. This is because now **cpuhog** will only run when the processes at priority >9 are not wanting to use the CPU. You can also try raising the priority of **cpuhog**, but it might take a while to kill it off. You will be saved by the fact that there is a **printf**(), which will cause **cpuhog** to become blocked once in a while as it waits for the output to print! That should be enough time to get a ctrl-c accepted by Photon and the **pterm**.

Armed with this information you can now start playing with the priority of different processes running in your system. For example, you could lower the priority of the shell running in a terminal by running:

```
"renice 1 $$"
```

This will lower the priority of the shell (**$$** is the shell **pid**) by 1 (making it run at priority 9). Now all the programs you start from that shell will also run at priority 9. You can also change priorities by name, using the slay utility. For example,

```
"slay -P 11 voyager vserver"
```

This will boost your web browser above other processes, making it much more responsive under heavy load. If you want to launch your programs from the shell with custom priorities then you should also check out the **on** utility that can spawn programs with custom priorities.

Finally, be sure to read the *QNX Neutrino System Architecture Guide* in the Neutrino help system.

*__Note__: This assumes you are using FIFO and not Round Robin scheduling. Otherwise, there is also time slicing that occurs within each priority level. It also assumes that you aren't using APS to partition the CPU of your system, although prioritization within an APS partition still occurs.