About QNX

What is QNX?

- Real-Time Operating System
- POSIX-compliant ~ UNIX-like
- µ-Kernel Operating System
- available for a variety of 32-bit target architectures
  - x86, MIPS, PowerPC, SH-4 and ARM (ARM, StrongARM, xScale)

History of QNX

1980 Idea was born
  - after an OS lecture at Waterloo University
1980 Quantum Software Systems was founded
1982 first version of QNX
  - target architecture: Intel 8088
  - non-embedded OS
mid 1990s QNX 4
  - POSIX-compliance
  - graphical user interface: PHOTON
end 1990s QNX Neutrino
  - Linux-like
2004 QNX Software Systems taken over by Harman International
Library vs. Monolithic vs. \( \mu \)-Kernel

<table>
<thead>
<tr>
<th>Library</th>
<th>Monolithic</th>
<th>( \mu )-Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Diagram" /></td>
<td><img src="image2" alt="Diagram" /></td>
<td><img src="image3" alt="Diagram" /></td>
</tr>
</tbody>
</table>

- **Library**
  - Single address space
  - No isolation or protection
  - Kernel interface: procedure calls
  - Small, efficient kernels
  - Fault-prone

- **Monolithic**
  - User space and kernel space
  - Separated address spaces
  - Isolation and protection
    - Between applications
    - Between applications and the kernel
  - No isolation or protection
    - Kernel and device drivers
  - Kernel interface: traps
  - Efficient
  - Inflexible

- **Microkernels**
  - User space and kernel space
  - Separated address spaces
  - Isolation and protection
    - Between applications and device drivers and the kernel
  - Kernel interface: traps
  - Communication: message passing
  - Flexible
  - Supposed to be inefficient
The Role of the QNX Microkernel

Provides a Software Bus

QNX Architecture

The QNX Microkernel

QNX Microkernel Services

- thread services (POSIX)
- signal services (POSIX)
- message-passing services
- synchronization services (POSIX)
- scheduling services (POSIX)
- timer services (POSIX)
- process management services

Thread Scheduling

- priority-driven (256 priorities)
- full preemptive
- threads are scheduled globally
- threads inherit parent priority
- context switch when a thread
  - blocks
  - is preempted
  - yields
- scheduling algorithms
  - FIFO
  - round-robin
  - sporadic

Thread Scheduling Algorithms

- FIFO
  - A thread runs until it
    - relinquishes control
    - is preempted

- Round-robin
  - A thread runs until it
    - relinquishes control
    - is preempted
    - consumes its timeslice
Sporadic Scheduling

- Sporadic Server
  - initial budget \((C)\)
  - foreground: normal priority \((N)\) ~ budget is available
  - background: low priority \((L)\) ~ budget is exhausted
  - replenishment period \((T)\)
    - budget is replenished in chunks
    - number of pending replenishments can be bounded ~ overhead
  - equivalent to a periodic thread \(T = (T, C)\)

\[
\begin{align*}
\text{Priority N} & \quad \text{Priority L} \\
\end{align*}
\]

\[
\begin{align*}
0 & \quad 50 & \quad 100 \\
\end{align*}
\]

\(T = \frac{50\text{ms}}{15\text{ms}}\)

1. thread blocks after 10 ms ~ replenishment of 10 ms at 50 ms
2. thread runs again at 15 ms ~ replenishment of 5 ms at 65 ms
3. thread exhausts its budget at 20 ms ~ drops to low priority L
4. 10 ms of threads budget are replenished ~ back to priority N
5. thread exhausts its budget at 60 ms ~ drops to low priority L
6. 5 ms of threads budget are replenished ~ back to priority N

Thread Synchronization

- Mutexes (kernel, priority inheritance)
- Condition Variables (kernel)
- Barriers (on top of mutexes, Condition Variables)
- Sleepon Locks (on top of mutexes, Condition Variables)
- Reader/Writer Locks (on top of mutexes, Condition Variables)
- Semaphores (kernel)
- FIFO Scheduling (kernel)
- Send/Receive/Reply (kernel)
- Atomic Operations (processor or emulated by kernel)

Startup

1. Initial programm loader (IPL)
   - configure memory controller, clock
   - setup processor, stack
   - locate OS image
   - load startup program
2. startup program
   - copy and decompress OS image
   - configure hardware
   - determine system configuration
   - establish callout-routines
   - start the OS kernel
3. Operating System Kernel
4. Boot Script
5. Applications
Available modules

- Filesystems (RAM, QNX4, DOS, CD-ROM, FFS3, NFS, CIFS, Ext2, Virtual, ETFS, ...)  
- Character I/O (Console, Serial, Parallel, Pseudo terminals ...)  
- Networking (QNet, TCP/IP)  
- Power Management  
- Photon microGUI  
- XFree86  
- tcsh  
- ...