Portability Made Possible:
Creating Reusable Software Assets
Through POSIX

Steve Furr, QNX Software Systems
Jason Clarke, QNX Software Systems
QNX Highlights

QNX: provider of realtime operating system (RTOS) software, development tools and services for mission-critical embedded applications

- 24 years of realtime embedded experience
- Millions of installations worldwide
- Reputation for reliability and scalability

Leader in innovative embedded technology

- First multitasking RTOS running with MMU support
- First RTOS to implement distributed processing
- First RTOS to implement symmetric multi-processing
- First POSIX certified RTOS
- First microGUI windowing system for embedded systems
Agenda

POSIX: Standard APIs
- Divergent Environments
- Portability vs. Conformance
- POSIX Overview and Evolution
- POSIX Comparison
  - QNX Neutrino, Linux & VxWorks

Application Portability
- VxWorks & QNX Neutrino
- Migration Roadmap
- Examples and Q&A
Today’s Environment

- Typical company has multiple product lines and limited interoperability

- Vendors are locked into a single OS solution or

- Applications need to be recoded or ported to deploy on different product lines
  - Takes time, adds costs
  - Increases delays to product deployment

<table>
<thead>
<tr>
<th>Application</th>
<th>Product Line A</th>
<th>Product Line B</th>
<th>Product Line C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OS 1</td>
<td>OS 2</td>
<td>OS 3</td>
</tr>
</tbody>
</table>

Recoding
Standards & Portability

- Standard APIs preserve software investments
- Portability lowers time, cost and risk associated with integrating new technology across product lines
- Common standard maximizes application base for development environments
- Developers familiar with standard become productive more quickly
Portability vs. Conformance

Portability

- Degree to which a software/application base is reusable
  - Between different versions of the same vendor’s environment
  - Between different vendors environments
- Measurement
  - Difficult to verifiably measure
  - Portability from one environment to another is not a reliable metric of how portable it will be to other environments, except under constrained circumstances

Conformance

- Provides verifiable metric of portability on an application by application basis (pass/fail)
- Two Sides:
  - Vendor conformance: conformant implementation
  - Consumer conformance: conforming application
The POSIX Specification
POSIX Overview

- Portable Operating System Interface

- Family of standards that define an interface, using the C programming language, a command interpreter, and common utility programs

- Developed by industry organizations
  - IEEE
  - ISO/IEC
  - The Open Group

- Introduced in 1980s to define standard way to interact with multiple UNIX derivatives

- POSIX1003.1-2001: current version of standard
  - Used by Linux Standard Base and Embedded Linux Consortium
Where Is it Used?

POSIX can be broadly implemented across a wide range of systems, including:

- Current major systems that are ultimately derived from the original UNIX system code (Version 7 or later)
- Compatible systems that are not derived from the original UNIX system code
- Emulations hosted on entirely different operating systems
- Networked systems
- Distributed systems
- Systems running on a broad range of hardware
API Evolution

- Single UNIX v3: 1742
- Single UNIX v2: 1434
- POSIX 1003.1-1996: 390
- POSIX 1003.2: 130
- 1003.1-90 FIPS 151-2: 199
Key Benefits

➡️ Source-level compatibility of applications
   ➢ Can choose the best OS for the job at hand, without having to rewrite entire code base or change programming models

➡️ Portability of applications and programmers
   ➢ Lowers the time, cost and risks associated with integrating new technology across the enterprise

➡️ Shifts focus from incompatible system product (RTOS) implementations to compliance to single set of APIs

➡️ If an OS meets the specification and commonly available applications run on it then it is open
   ➢ Which specification (i.e. profile) do I need?
# POSIX Feature Matrix

<table>
<thead>
<tr>
<th>Feature</th>
<th>PSE 51</th>
<th>PSE 52</th>
<th>PSE 53</th>
<th>PSE 54</th>
<th>POSIX 1003.1-2001</th>
</tr>
</thead>
<tbody>
<tr>
<td>1003.1-90 Processes</td>
<td>-</td>
<td>-</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>1003.1-90 Pipes</td>
<td>-</td>
<td>-</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>1003.1-90 Files &amp; Directories</td>
<td>-</td>
<td>x</td>
<td>-</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>1003.1-90 Users &amp; Groups</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>1003.1b-93 Memory Protection</td>
<td>-</td>
<td>-</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>1003.1b-93 Hi Res. Clocks &amp; Timers</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>1003.1b-93 Realtime Signals</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>1003.1b-93 Semaphores</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>1003.1b-93 Shared Memory</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>1003.1b-93 IPC Message Passing</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Check for Advanced Realtime
## POSIX Feature Matrix

<table>
<thead>
<tr>
<th>Feature</th>
<th>PSE 51</th>
<th>PSE 52</th>
<th>PSE 53</th>
<th>PSE 54</th>
<th>POSIX 1003.1-2001</th>
</tr>
</thead>
<tbody>
<tr>
<td>1003.1c-95 Threads</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>1003.1c-95 Thread Priority Protection</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>1003.1c-95 Thread Priority Scheduling</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Process Shared Attribute Stack Size</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>1003.1c-95 Thread Priority Inheritance</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>1003.1c-95 Thread Priority Protection</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>POSIX2_SW_DEV</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
POSIX Profiles – OS Compliance

<table>
<thead>
<tr>
<th>POSIX Standard</th>
<th>QNX Neutrino</th>
<th>Linux</th>
<th>VxWorks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification Base</td>
<td>1003.1-2001</td>
<td>1003.1-1996*</td>
<td>PSE 51/PSE 52</td>
</tr>
<tr>
<td>Realtime (.1b)</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Realtime Threads (.1c)</td>
<td>x</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>1003.1d-1999 Additional Realtime</td>
<td>x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(Sporadic server scheduling, execution timers, …)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1003.1j-200x Advanced Realtime</td>
<td>x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(Barriers, spin-locks, …)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Best practices (development)</td>
<td>Configure, GCC, perl, …</td>
<td>Configure, GCC, perl, …</td>
<td>-</td>
</tr>
</tbody>
</table>

*Newer versions of the Linux kernel are moving toward conformance with the 2001 specification.*
Linux-QNX Portability

- Application portability between Linux and QNX Neutrino can be easily accomplished
  - Both Linux and QNX Neutrino share large POSIX feature set

- Linux developers can retain programming model and existing APIs while porting applications to QNX Neutrino

Bottom line:

- Porting applications between Linux and QNX Neutrino is relatively simple
- Standard POSIX APIs are key
<table>
<thead>
<tr>
<th>Untar the source:</th>
<th>tar xz lynx2.8.4.tar.gz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Go to the source directory.</td>
<td>cd lynx2-8-4</td>
</tr>
<tr>
<td>Make a host directory:</td>
<td>mkdir x86-pc-nto-qnx</td>
</tr>
<tr>
<td>Go to the host directory.</td>
<td>cd x86-pc-nto-qnx</td>
</tr>
<tr>
<td>Configure the source:</td>
<td>../configure</td>
</tr>
<tr>
<td>Make.</td>
<td>make</td>
</tr>
<tr>
<td>Install the browser:</td>
<td>make install</td>
</tr>
</tbody>
</table>
## Sample 3rd-Party Applications

<table>
<thead>
<tr>
<th>TAO — CORBA ORB</th>
<th>PVM — Distributed processing system</th>
<th>mySQL — Open-source database</th>
<th>Postgres — Open-source database</th>
</tr>
</thead>
<tbody>
<tr>
<td>GDBM — Database</td>
<td>Apache — Web server</td>
<td>LibXML2 — XML Database</td>
<td>LibXSLT — XSL Processor</td>
</tr>
<tr>
<td>Xerces — XML Processor</td>
<td>Python — Scripting language</td>
<td>Perl — Scripting language</td>
<td>Ruby — Scripting language</td>
</tr>
<tr>
<td>Open SSH/SSL — Secure sockets and shells</td>
<td>Zebra router — Manages TCP/IP based protocols</td>
<td>Samba — Shared access to resources on Windows networks</td>
<td>Mozilla — Web browser based on Netscape source code</td>
</tr>
<tr>
<td>Doxygen — Source code documentation tool</td>
<td>Open LDAP — Light weight Directory Access Protocol</td>
<td>Sendmail — Email server</td>
<td>GNU EMACS — Programmers’ editor</td>
</tr>
<tr>
<td>GDB — GNU debugger</td>
<td>GCC — GNU C/C++ compiler</td>
<td>CVS — Source code Version-control System</td>
<td>VIM — Vi IMproved, a programmers’ editor</td>
</tr>
</tbody>
</table>
POSIX Summary

➤ Standard interface increases software portability for all embedded systems

➤ Some markets, such as military, moving toward using POSIX as their base specification

➤ OS conformance a matter of degree
  > QNX Neutrino provides conformance with 1003.1-2001
  > Linux moving toward 2001.3-2001 with latest versions
  > VxWorks only conforms with minimal profiles – PSE 51/PSE 52

➤ Migration of legacy VxWorks code to POSIX RTOS increases software portability
OS Migration: VxWorks to QNX Neutrino

Jason Clarke, QNX Software Systems
OS Migration – Business Drivers

➡️ Supplier limitations
  ➤ Proprietary API locks customer to OS vendor
  ➤ High cost of developer training
  ➤ Limited software choices

➡️ Product capabilities
  ➤ Product stability – OS reliability, performance
  ➤ Support for latest technologies – SMP, HA, 3D graphics
  ➤ Dynamic upgradeability – modularity, software hotswap

➡️ Development costs
  ➤ High cost of new feature development and deployment
  ➤ Soaring bug identification and bug fix costs
  ➤ Third-party software porting and integration costs
  ➤ Need to employ specialized kernel experts
Porting Issues

➤ Software architecture
  ➤ Memory accessibility (process vs. single-address model)
  ➤ Tight coupling between OS, system, and user tasks

➤ Differences at system and application level API
  ➤ POSIX vs. minimal profile + proprietary
  ➤ Physical memory vs. virtual memory addressing
Architectural Comparison

Realtime Executive

Microkernel

Application

Device Drivers

TCP/IP Stack

File-system

Application

Device Drivers

TCP/IP Stack

File-system

QNX Confidential. All content copyright QNX Software Systems.
→ POSIX is bred in the bone

→ Applications, drivers, stacks coded to the same APIs
The Migration Goal:
- Re-use existing legacy software
- Future-proof, unified, scalable software architecture

The Migration Challenge:
- Poorly defined coupling between components
- Implicit sharing of memory
- Several assumptions about legacy architecture creep into code
Memory Accessibility

Realtime executive model: All tasks have access to complete system memory
- Tight coupling between OS, system and user tasks
- Difficult to separate components without re-design
- Re-architecting parts of the system is non-trivial

Process model: Threads within a process have access to the same memory
- Separate processes can share memory using explicitly defined shared memory regions
Differences at system and application level API

- Legacy applications often use proprietary APIs
- Interface to system is also proprietary (possibly different)
- While porting, a mapping may be necessary between the proprietary API and the underlying POSIX API of the new OS
  - Can encapsulate most of mapping in a “porting library”
Physical vs. Virtual Memory

- OS uses hardware to translate virtual addresses to physical addresses using a maintained table.
- Must ensure all memory accesses performed via “mapped-in” variables instead of `#define`.
Software Builds

- **Build scripts/makefiles must be re-worked**
  - Adapt/adopt build infrastructure
    - Re-work macros to point to QNX tools + re-write link sections
    - Import code base using the QNX Momentics IDE
      (automatically sets up make infrastructure)

- **Compiler differences**
  - Different compiler vendors (Diab, Metrowerks, GCC, etc.)
  - Different GCC variants: 2.7.2 (VxWorks 5.4) vs. 2.95 (QNX / VxWorks 5.5)
  - Code changes required to remove compiler errors

- **Linker**
  - Different linkers/linker options (change makefiles / macros)
  - “main” function for each separate process: Equivalent to VxWorks
    “usrApplInit()” function
  - Shared library concept used to reduce memory footprint
Other Considerations

- Significant legacy software base
  - Millions to tens of millions of LOC
  - Large number of protocols and applications to be re-used

- Most software written for real-time executives
  - Specific assumptions about underlying RTOS

- Code may depend on specific tools
  - C++ especially fragile

- Modularity not always enforced
  - May complicate “from the ground up” re-architecting

- Device driver infrastructure

- Software build infrastructure, capabilities
Porting Strategies
A phased approach
Porting Process

Two main aspects: porting driver/hardware related code, and porting application code

- Typically porting driver code will be done manually by inspection ("do-once" operation)
- Porting application code would likely be most significant portion of effort associated with porting
Two main strategies to deal with legacy application code

- Develop porting library that provides legacy API while implementing it using underlying API calls of new OS
- Replace legacy functions with appropriate native OS calls for the new OS. Can be done manually or automatically through use of code parsing tools
Mapping tasks – Option #1

Run application as single process under the new OS. Every task in the original legacy application becomes a thread in the new application process. Drivers run in their own protected memory spaces. Application is protected from driver and OS.
Option #2: Break application down into separate processes that communicate using process IPC mechanisms and shared memory to share data (far more robust)
Porting Roadmap

Legacy System

Driver

APP-1

APP-2

OS

Application Blob

Driver

Driver

APP-1

APP-2

Legacy API

VX2QNX Lib

OS

Application Blob

Driver

APP-1

APP-2

Driver

APP-2

Shared Memory

IPC

Decoupled Application

POSIX API

OS

POSIX API

OS
Porting Library

- Implements key VxWorks functions
  - Functionally equivalent implementation for the VxWorks API calls
  - Provides code compatibility with legacy code at the application layer

- Complete VxWorks system is encapsulated inside one process under the QNX® Neutrino® RTOS
  - Task in VxWorks → Thread in QNX Neutrino

- taskLib, msgQLib, semLib, semCLib, semBLib, semMLib, wdLib, errnoLib, taskInfoLib, kernelLib, lstLib, schedPxLib, mqPxLib, clockLib, semPxLib, sigLib, timerLib, ...

- Covers majority of core VxWorks API

- Networking API coverage also being added into library

- Library provided as source:
  - Use as reference for porting and/or deployment as a compatibility layer
Migration Scope

- Adapt/adopt build infrastructure
  - Examine on per-environment basis

- Port hardware-related code
  - Drivers and OS

- Port application code
  - Porting library solves a major issue (API compatibility)

- Phased effort lets you evolve system over time
  - No system downtime
  - More manageable
Porting Examples
Inter-task Synchronization

VxWorks Using Semaphores

```c
SEM_ID mSem;
void func1(void) {
    sem_take(mSem,...);
    ...critical section
    sem_give(mSem);
}
void func2(void){
    sem_take(mSem,...);
    ...critical section
    sem_give(mSem);
}
void init(void) {
    mSem = semMCreate(...)
taskSpawn("Task1", ..., func1, ...);
taskSpawn("Task2", ..., func2, ...);
}
```

QNX Neutrino Using Mutexes

```c
pthread_mutex_t *mmtx;
void func1(void) {
    pthread_mutex_lock(mmtx,...);
    ...critical section
    pthread_mutex_unlock(mmtx);
}
void func2(void){
    pthread_mutex_lock(mmtx, ...);
    ...critical section
    pthread_mutex_unlock(mmtx);
}
void init(void) {
    mmtx = malloc(sizeof(pthread_mutex_t));
    pthread_mutex_init(mmtx, ...);
    pthread_create(..., func1, ...);
    pthread_create(..., func2, ...);
}
```

QNX Neutrino Using Semaphores

```c
sem_t *msem;
void func1(void) {
    sem_wait(msem,...);
    ...critical section
    sem_post(msem);
}
void func2(void){
    sem_wait(msem, ...);
    ...critical section
    sem_post(msem);
}
void init(void) {
    msem = malloc(sizeof(sem_t));
    sem_init(msem, ...);
    pthread_create(..., func1, ...);
    pthread_create(..., func2, ...);
}
```
Memory Accessibility

**VxWorks**

```c
#define DEVICEADDR 0x8000abcd
void *daddr;

void func1(void) {
    int val;
    ...
    // read value
    val = *daddr;
    // modify value
    ...
    // write value
    *daddr = val;
}

void init(void) {
    ...
    daddr = DEVICEADDR;
    taskSpawn("Task1", ..., func1, ...);
    ...
}
```

**QNX Neutrino**

```c
#define DEVICEADDR 0x8000abcd
void *daddr;

void func1(void) {
    int val;
    ...
    // read value
    val = *daddr;
    // modify value
    ...
    // write value
    *daddr = val;
}

void init(void) {
    ...
    daddr = mmap(... len, ..., DEVICEADDR);
    pthread_create(..., func1, ...);
    ...
}
```
Migration Summary

- Business and technical needs drive migration
  - Reliability, modularity, cost to add new features, software reuse…

- Moving to POSIX RTOS provides greater application portability, increases software ROI

- Phased approach to migration enables continued revenue stream and manageable migration path

- QNX offers program designed to accelerate migration
  - Porting tutorial and library
  - Extended QNX Momentics evaluation
  - Migration support and services packages
Thanks for Your Time!

Steve Furr (furr@qnx.com)
Jason Clarke (jclarke@qnx.com)

Porting Tutorial: “Migrating Legacy Code from WindRiver’s VxWorks to the QNX Neutrino RTOS”
http://www.qnx.com/mailings/vxexporting/