



Think your future OS strategy includes  
VxWorks? Think again.

Leo Forget, Product Manager  
Pat Shelly, Field Application Engineer



### **Intel to Acquire Wind River Systems for Approximately \$884 Million**

**SANTA CLARA, Calif., June 4, 2009** – Intel Corporation has entered into a definitive agreement to acquire Wind River Systems Inc, under which Intel will acquire all outstanding Wind River common stock for \$11.50 per share in cash, or approximately \$884 million in the aggregate. Wind River is a leading software vendor in embedded devices, and will become part of Intel's strategy to grow its processor and software presence outside the traditional PC and server market segments into embedded systems and mobile handheld devices. Wind River will become a wholly owned subsidiary of Intel and continue with its current business model of supplying leading-edge products and services to its customers worldwide.

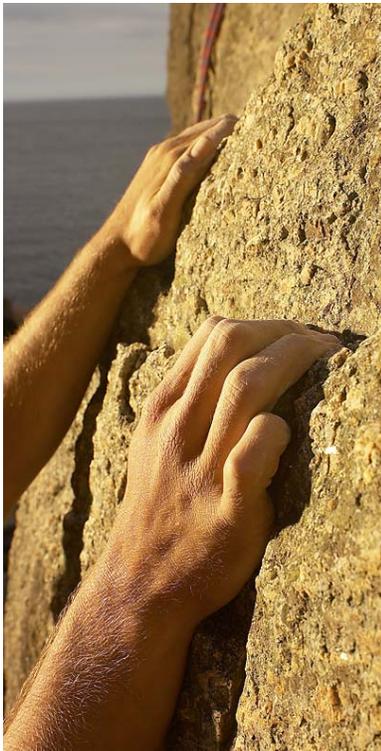
"This acquisition will bring us complementary, market-leading software assets and an incredibly talented group of people to help us continue to grow our embedded systems and mobile device capabilities," said Renee James, Intel vice president and general manager of the company's Software and Services Group. "Wind River has thousands of customers in a wide range of markets, and now both companies will be better positioned to meet growth opportunities in these areas."

## Business risks to Wind River customers



- Will Intel continue investments into Wind River to support the latest technology trends?
- Will Wind River continue to support non-Intel silicon?
- Will other silicon vendors cooperate (roadmaps, technical assistance, funding) with their biggest competitor?
- Will the Wind River software ecosystem shrink as software suppliers desert a perceived Intel-only OS?
- Will VxWorks business or support models change as a result of the acquisition?
- Will Intel decide to change the focus of Wind River (for example, from a general embedded company to phone-only focus)?
- Will VxWorks be discontinued by Wind River and/or Intel?

## Technical risks to Wind River customers



- Will the solution continue to be sufficient in the face of relentlessly advancing hardware and software complexity?
- Is the development team able to prevent an escalating workload with the current tools and OS technologies?
- Will the current OS solution continue to support newly released hardware?
- Is your current software solution easily scalable and modifiable as you expand your product's feature set?
- Can the vast resources of open source be tapped, but in a safe reliable fashion?

## OS migration: QNX is a great fit if you need ...



### QNX has...

- A reliable foundation for critical systems?** → A 30 year track record
- Hard real-time or time/space partitioning?** → Scheduler and adaptive partitioning
- A scalable architecture for the future?** → Modular microkernel architecture
- To build on standards?** → POSIX, OpenGL ES, OpenKODE, Eclipse
- Certifications?** → ISO9001, PSE52, EAL4+, SIL3

## OS migration: QNX is a great fit if you need ...



### QNX has...

**Transparency of source?**

➔ Fully published source code and forums

**Clean intellectual property?**

➔ No viral GPL licensing

**Support through lifetime of the product?**

➔ Guaranteed availability of past versions

**Sophisticated HMI solutions?**

➔ Attractive HMI environment

**Low RAM and flash requirements?**

➔ Solution customizable for smallest size



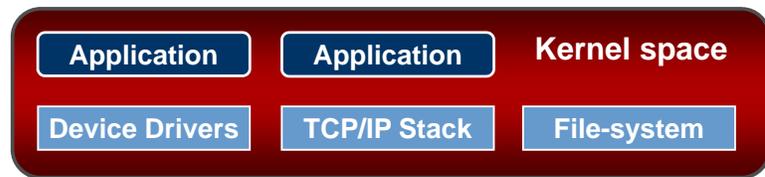
**Technology comparison**

# Microkernel architecture benefits



## Realtime executive

- > No MMU and no protection
- > Applications, drivers, and protocols are all in kernel space



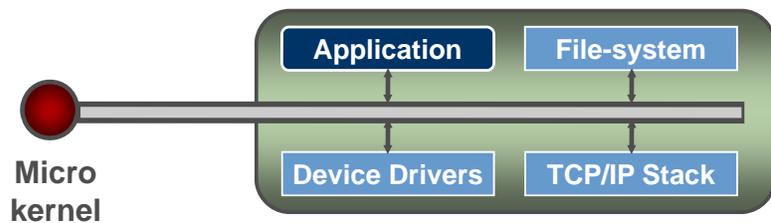
## Monolithic kernel (Microsoft / Unix / etc)

- > MMU with partial protection
- > Applications are protected

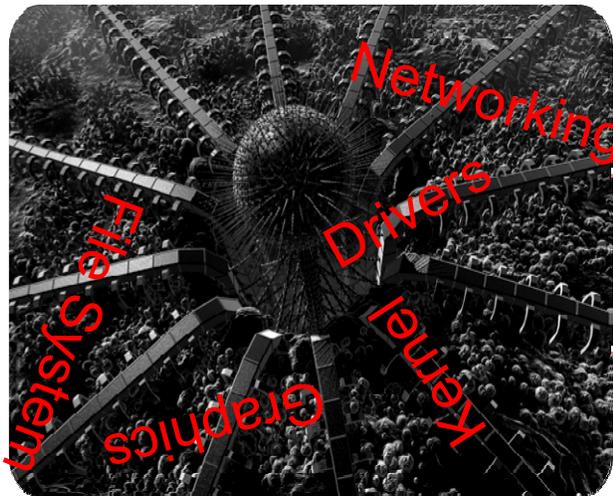


## TRUE microkernel (QNX Neutrino RTOS)

- > MMU with full protection
- > Applications, drivers, and protocols are protected



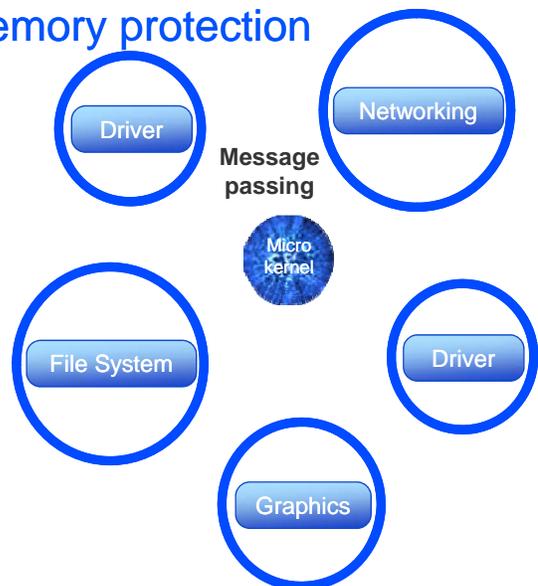
# QNX reliability (when failure is not an option)



## Monolithic kernel

WinCE 3.9 million lines of code  
Linux: 5.76 million lines of code  
XP: 40 million lines of code

## Memory protection



## Microkernel

QNX 0.1 million lines of code

# Microkernel architecture benefits



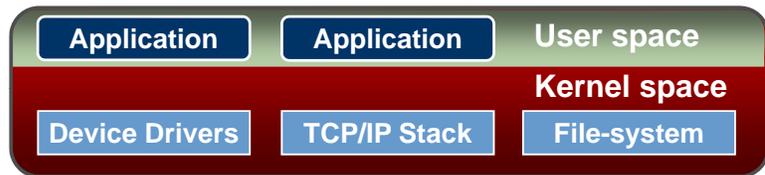
## Realtime executive

- > No MMU and no protection
- > Applications, drivers, and protocols are all in kernel space



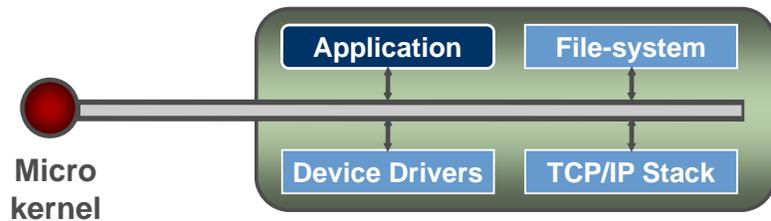
## Monolithic kernel (Microsoft / Unix / etc)

- > MMU with partial protection
- > Applications are protected



## TRUE microkernel (QNX Neutrino RTOS)

- > MMU with full protection
- > Applications, drivers, and protocols are protected



# Microkernel architecture benefits



## Realtime executive

- > No MMU and no protection
- > Applications, drivers, and protocols are all in kernel space



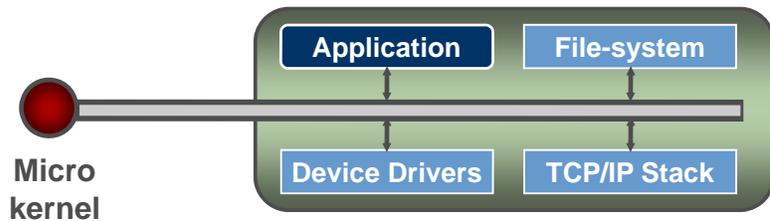
## Monolithic kernel (Microsoft / Unix / etc)

- > MMU with partial protection
- > Applications are protected



## TRUE microkernel (QNX Neutrino RTOS)

- > MMU with full protection
- > Applications, drivers, and protocols are protected



# Microkernel architecture benefits



## Realtime executive

- > No MMU and no protection
- > Applications, drivers, and protocols are all in kernel space



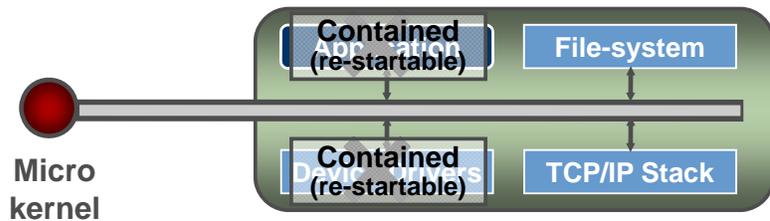
## Monolithic kernel (Microsoft / Unix / etc)

- > MMU with partial protection
- > Applications are protected



## TRUE microkernel (QNX Neutrino RTOS)

- > MMU with full protection
- > Applications, drivers, and protocols are protected



## Weighing technical risks



	VxWorks	QNX Software Systems
<b>Multicore</b>	✓ Yes	✓ QNX Neutrino RTOS supports three types of multi-core operation; Symmetric, Bound and Asymmetric.
<b>Scalable</b>	✗ Distribution code is not transparent and requires application rewriting	✓ QNX Neutrino RTOS supports scaling up or down easily: TDP for distributed processing, and significant support for multicore silicon
<b>High availability</b>	✓ VxWorks support a 3 <sup>rd</sup> party High Availability solution.	✓ QNX Neutrino RTOS provides features like Adaptive Partitioning and High Availability Monitor (HAM), designed for reliability and robustness
<b>Real-time</b>	✓ VxWorks is hard real-time	✓ QNX Neutrino RTOS is hard real-time
<b>System architecture</b>	✗ While sometimes dubbed a microkernel by Wind River, it is not.	✓ The QNX Neutrino microkernel and its accompanying Process Manager (procnto) contain only the most fundamental OS services (signals, scheduling, messaging, etc.). Virtually all other OS services and user-written code — file systems, device drivers, GUIs, protocol stacks, applications — run in memory-protected user space.
<b>Standards</b>	✓ VxWorks solution adheres to many industry standards including POSIX, Eclipse	✓ QNX pursues and participates in standards; standards promote freedom of choice: POSIX, OpenGL-ES, OpenVG, OpenKODE, Eclipse

## Weighing commercial risks



	VxWorks	QNX Software Systems
<b>Focus</b>	? Wind River has a long history of discontinuing products and changing strategies. How will Intel acquisition affect company focus?	✓ QNX has a 29 year track record in the embedded software industry.
<b>Reliability</b>	? VxWorks can execute user-written code in kernel space. While this model enables the programmer to optimize performance, it makes the kernel vulnerable to poorly written code. The programmer can easily introduce potential kernel faults, creating the need for a significant amount of kernel testing prior to deployment.	✓ QNX Neutrino RTOS field-proven in many life-critical applications: nuclear power plants, military and space applications, medical equipment, etc.
<b>Tools</b>	✓ Wind River provides an Eclipse based development environment.	✓ QNX uses industry standard Eclipse based IDE, tightly integrated with many sophisticated QNX-specific features
<b>Support</b>	✓ Wind River has a solid support organization.	✓ QNX support focused on direct customer problem resolution and customer involvement
<b>Licensing terms</b>	☒ Licensing models are complex and expensive.	✓ Simpler, more flexible, lower barrier to entry.
<b>Hardware</b>	✓ VxWorks currently runs on a wide variety of hardware platforms.	✓ QNX Neutrino RTOS runs on a wide variety of hardware platforms



**OS migration – what is involved?**

## OS migration phases



### 1. Assess

Thoroughly understand your product's design and software components and identify potential porting issues

### 2. Port

Make code changes as necessary to build and run your product on QNX Neutrino RTOS

### 3. Optimize

Take advantage of the QNX Neutrino RTOS's modularity and isolation to increase system reliability and facilitate trouble shooting

```
01. /* Fig. 2-5: fig02_05.c
02. Addition program */
03. #include <stdio.h>
04.
05. /* function main begins program execution */
06. int main( void )
07. {
08.     int integer1; /* first number to be input by user */
09.     int integer2; /* second number to be input by user */
10.     int sum;      /* variable in which sum will be stored */
11.
12.     printf( "Enter first integer\n" ); /* prompt */
13.     scanf( "%d", &integer1 ); /* read an integer */
14.
15.     printf( "Enter second integer\n" ); /* prompt */
16.     scanf( "%d", &integer2 ); /* read an integer */
17.
18.     sum = integer1 + integer2; /* assign total to sum */
19.     printf( "Sum is %d\n", sum ); /* print sum */
20.     return 0; /* indicate that program ended successfully */
}
```

- Interface analysis
- Task analysis
- Hardware access analysis
- Third party software analysis
- API analysis

1. Assess

- **Memory accessibility**
- **Porting from VxWorks API to QNX Neutrino API**
- **Exception handling**
- **Modularity enforcement**
- **Device driver infrastructure**

1. Assess

## Porting phase – elements of porting

A typical porting project will involve effort in the following areas:

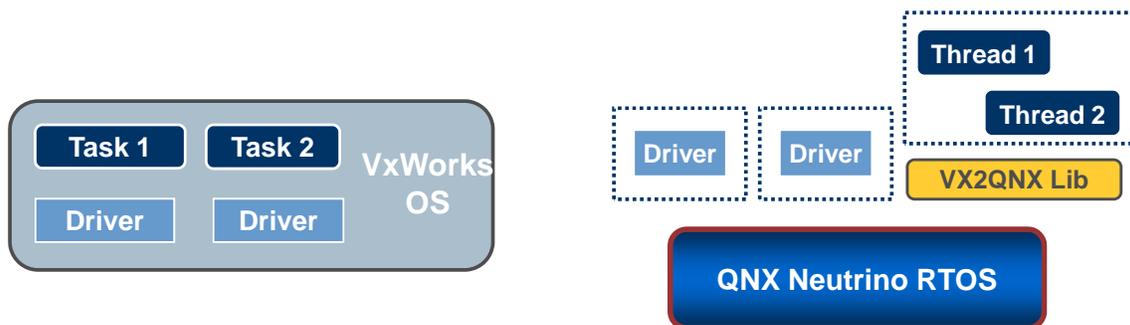
2. Port

- Board bring up / startup
- Hardware input/output (drivers)
- Networking
- Applications
- Build environment

Legacy system



Ported system



# Porting phase – board bring up/ startup



2. Port



(BIOS/bootloader)



romInit.s



romStart.c



sysAlib.S



prjConfig.c

VxWorks



IPL/ROM Monitor



Startup



Boot Script



Init Scripts/  
Programs

QNX Neutrino RTOS

## Porting phase – hardware input/output



### VxWorks:

- ISRs are dealt via the intArchLib.
- Device drivers tightly bound into operating system

### QNX Software Systems:

2. Port

- Hardware access in ISRs needs to be properly setup in terms of privity/permissions, memory addressing and memory accessing before an ISR can be used
- Device drivers can be started and stopped as standard processes
- Drivers can be developed and debugged just like any other application

### Conclusions:

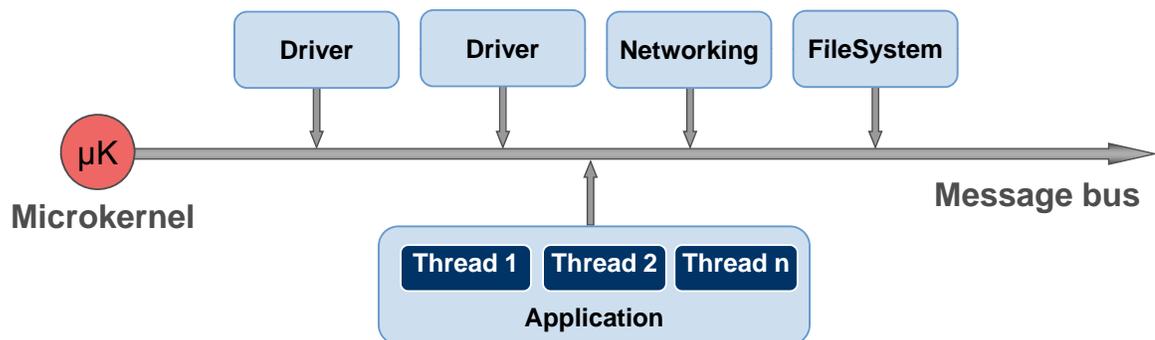
2. Port

- Simple VxWorks ISRs that clear hardware interrupt source and perform a semGive are relatively easy to port to QNX Neutrino RTOS
- At the application layer, the ANSI-C/POSIX compatible APIs are nearly identical between VxWorks and QNX Neutrino RTOS
- Device drivers operation is very different between VxWorks and QNX Neutrino RTOS, and drivers must be rewritten

### Execution model: mapping tasks

2. Port

- Run the applications a 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 protected memory spaces. Application is protected from driver and OS.



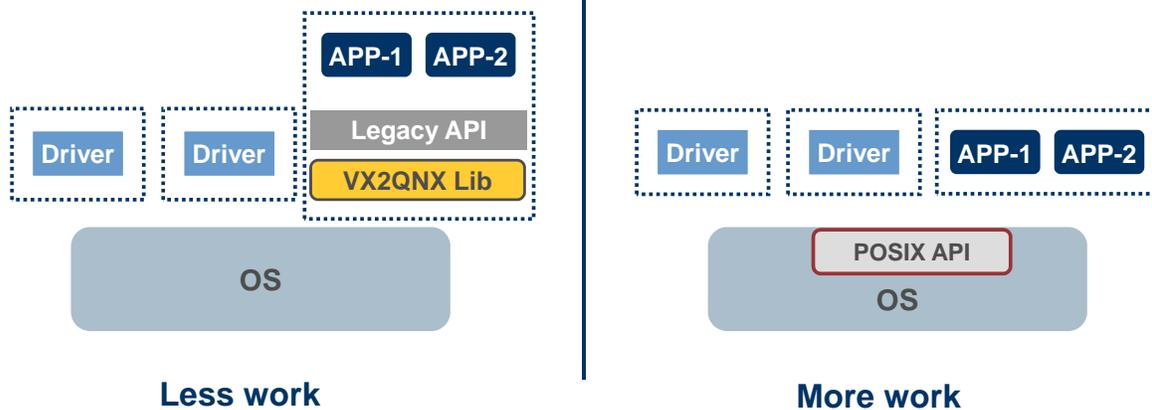
## Porting phase – applications

### API model : mapping APIs

2. Port

#### ● Two main strategies for 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



## Porting phase – build environment



### 2. Port

C/C++ compiler and tool chain	<b>WIND RIVER</b> Diab	 GNU
OSes available	vxWorks only	QNX Neutrino RTOS and vxWorks
Type checking and warnings	Medium	High
Library support	May have STL or other library incompatibilities	Standard
Build result	App or OS Image	Application Image

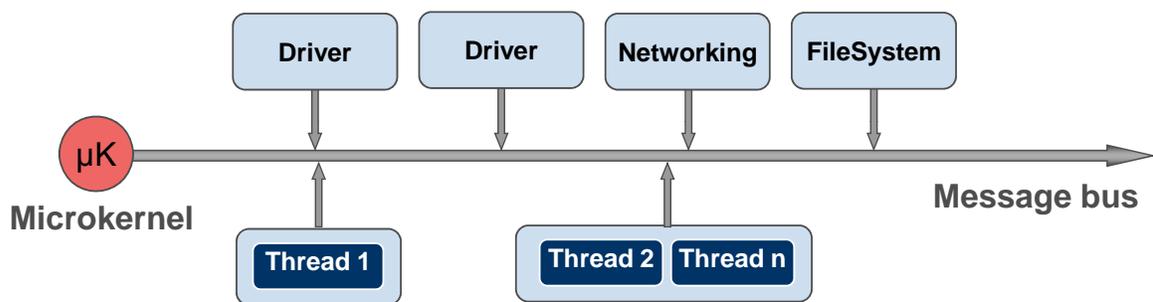
### 3. Optimize

- Refactoring for increased modularity
- Memory usage optimization
- Performance optimization

## Optimization phase 1) modularity

- Group logical threads into separate processes to increase system modularity and reliability.

3. Optimize



- Use IPC or shared data to communicate between processes

# Optimization phase 2) memory usage



3. Optimize

Track memory allocations as they occur.

Immediately detect buffer over-runs and jump into the debugger at the offending line

View the number of bytes free, allocated, and in use, both overall and per byte range

Find undetected memory leaks

View changes in memory usage over time.

The screenshot displays the QNX Memory Analysis tool interface. At the top, a code editor shows a C function with a `while(1)` loop containing `buffer = malloc(10); strcpy(buffer, "1234567890");` and a conditional `free(buffer);` statement. Below the code, the 'Memory Events' window shows a list of events, including a warning: 'Pointer within malloc region, but outside of malloc data bounds'. The 'Allocation Trace' window provides a detailed log of memory allocation calls, such as `malloc` and `a_bad_function`. On the right, the 'Malloc Information' panel shows a bar chart for 'Total Heap 16K' with 'used: 1K' and 'over' indicators. Below this, a table shows 'Distribution' of memory usage by byte range, with columns for 'Counts (malloc/free)' and 'Allocated'. At the bottom right, a 'History' bar chart shows changes in memory usage over time.

Byte Range	Counts (malloc/free)	Allocated
0 - 16	21 / 8	13
17 - 24	6 / 0	6
25 - 32	0 / 0	0
33 - 48	1 / 0	1
49 - 64	0 / 0	0

# Optimization phase 3) performance optimization



## 3. Optimize

**Determine which threads are busiest**

Thread ID	Processor Time (s)	% Time Usage
Thread #1	427.775	[Progress bar]

**Find optimization bottle-necks: functions that execute the longest, called most frequently, etc.**

Function	Total Time (s)	Time since last reset (s)	Call Count	usec/Call	% Time Usage
dix	175.594	175.594	6991	25117.151	[Progress bar]
straight	89.925	89.925	6991	12862.967	[Progress bar]
ins	11.039	11.039	6991	1579.030	[Progress bar]
quick	0.459	0.459	6991	65.656	[Progress bar]
merge	13.352	13.352	6991	1909.884	[Progress bar]
lookup	0.003	0.003			
shl	0.246	0.246			
heap	1.920	1.920			
fill	2.717	2.717			
rand	0.678	0.678			
rand r	0.415	0.415			

**Call pairing identifies your programs dynamic execution structure; use that information to make it more efficient**

```
graph TD
    start --> main
    main --> radix
    main --> quick
    main --> merge
    main --> bubble
    main --> shl
    main --> ins
```

**Pinpoint individual lines of source code that consume the most CPU**

```
L3: p=R[0]-
    if (Kt<
    q=p, p-
    R[q]->L
}
void bubble(RECORD **R, int N) {
    int j, t, BOUND;
    RECORD *Rt = (RECORD *)malloc(sizeof(RECOR
    BOUND=N;
    t=0; for (j=1; j<=BOUND-1
    if (R[j]->K>R[j+1]->K
    if (t!=0) { BOUND=t; goto
    free(Rt);
}
void merge(RECORD **R, int N) {
    int i, j, d, p, q, r, t;
    RECORD *Rt = (RECORD *)ma
    t=lg(N, HIGH); for (j=1.
    q=pow(2, t-1); r=0; d
    for (i=0; i<N-d; i+=1) if ((i&p)==r) {
    if (R[i+1]->K>R[i+d+1]->K) swap(R[i
    if (q!=p) { d=q-p; q=q/2; r=p; goto M3
    p=p/2; if (p>0) goto M2;
    free(Rt);
}
void quick(RECORD **R, int N) {
    int i, j, l, r, K, S;
```

# Optimization phase 3) performance optimization



**3. Optimize**

**Log activity of all processes in system to determine process behavior and interaction; eliminate systemic performance problems**

**Measure exact CPU and thread use for single and multicore systems**

**View detailed log with labelled events and precise timestamps**

Event	Time	Owner	Type	Data
922044	18s 814ms 899us	pidin Thread 1	Send Message	pid 1
922045	18s 814ms 904us	pidin Thread 1	State Reply	pid 3
922046	18s 814ms 905us	procnto-smp-instr Thread 12	State Running	pid 1
922047	18s 814ms 908us	procnto-smp-instr Thread 12	Receive Message	pid 1
922048	18s 814ms 909us	procnto-smp-instr Thread 12	MsgReceivev Exit	rcvid 0x00000012 rmsg 0x00020100
922049	18s 814ms 912us	Interrupt 0x0	Entry	ip 0xF0002ACA interrupt 0x00000000
922050	18s 814ms 913us	Interrupt 0x0 - procnto-smp-instr	Handler Entry	pid 1 area 0x00000000 ip 0xF001D668 interrupt 0x0
922051	18s 814ms 918us	procnto-smp-instr Thread 12	KER RING0 Enter	ara 0xE7039E7C func 0xF0039E68

## VxWorks migration Foundry27 project



### Foundry27

The community portal for QNX software developers

- **Comprehensive VxWorks porting guide provides**
  - Detailed OS comparison
  - Porting guidelines, tips and tricks
  
- **Vx2QNX Porting Library**
  - Covers majority of core VxWorks API
  - Provides code compatibility with legacy code at the application layer.
  - Complete VxWorks system is encapsulated inside one process under QNX Neutrino RTOS
    - Task in VxWorks → Thread in QNX Neutrino RTOS
  - Provided as source - Use as reference for porting and/or deployment as a compatibility layer
  
- **Forum support**
  - Interact with QNX engineers

## Contact QNX



### ⇒ North America

- T: + 1-800-676-0566
- F: + 1-613-591-3579

### ⇒ International

- T: + 1-613-591-0931
- F: + 1-613-591-3579

### ⇒ Online

- [info@qnx.com](mailto:info@qnx.com)
- [www.qnx.com](http://www.qnx.com)
- [www.foundry27.com](http://www.foundry27.com)

Thank you for joining us.

# Questions?