VxWorks - An Overview

WindRiver's VxWorks distributed real-time operating system includes integrated networking facilities and a complete software development environment for Windows® and UNIX® hosts.

Major features of VxWorks include a fast, multitasking kernel with pre-emptive scheduling and fast interrupt response, extensive intertask communications and synchronization facilities, efficient UNIX-compatible memory management, multiprocessor facilities, a shell for user interface, symbolic and source level debugging capabilities, performance monitoring and an I/O file system.

The full range of VxWorks facilities are available to users, with Concurrent Technologies' VxWorks Board Support Packages (BSPs) for a wide range of target Multibus CPU card types. All BSPs have been fully validated by Concurrent Technologies for the greatest confidence and ease of use.

VxWorks, from Wind River Systems, is a networked real-time operating system designed to be used in a distributed environment. It runs on a wide variety of hardware, including MC680x0, MC683xx, Intel i960, Intel i386, R3000, SPARC, Fujitsu SPARClite, and TRON Gmicro, based systems. It requires a host workstation for program development; supported host platforms include Sun3, Sun4, HP9000, IBM RS-6000, DEC, SGI, and MIPS.

It does not run development systems software such as compiler, linker and editor on the target machine. The development environment is based on cross-development or remote-development method. You will need a UNIX machine of some sort (e.g. SUN's) to run the compilers and debuggers. The compiled application code can be downloaded to the target and runs as part of the VxWorks image. During the development phase or thereafter, individual object code (.o files) can be downloaded dynamically to running target system.

Features:
In Version 5.1:

- wind kernel unlimited tasks
- 256 priorities
- binary, counting mutex semaphores
- message q's
- POSIX pipes
- sockets
- shared memory
- profiling utilities
- ethernet support (i596, LANCE, NIC, SONIC)
- SLIP (no PPP yet)
- backplane driver for network
- rlogin (server & client)
- telnet (server only)

- rpc (no rpcgen)
- nfs (client)
- ftp (server & client)
- tftp (client & server)
- rsh
- bootp
- proxyarp
- C-interpreter shell (incomplete but useful)
- symbolic debugging
- disassembly
- performmance monitoring tools
- exception handling
- signal handling (not quite UNIX compatible)
- dynamic object image loader
- system symbol table
- system info utilities
- libraries of 600+ utility routines
- remote source level debugger(VxGDB)
- SCSI support
- DOS & RT11 & Raw filesystems.
- "full ANSI"
- "POSIX I/O"

What are differences between traditional UNIX and VxWorks?
VxWorks does have a lot of UNIX "compatible" routines in the user libraries. So porting a UNIX application is not that hard. But there are enough differences to make such a port take longer than normally expected.

VxWorks runs in one mode. No protected vs. user mode switching is done. Running in supervisor mode on most processors, and not using traps for system calls, VxWorks can achieve minimal overhead on a given piece of hardware than UNIX. Programming on VxWorks can be more tricky than UNIX for the same reason.

UNIX provides resource reclamation; by default, VxWorks does not. Instead programmers write what they need as needed. As a result, the context switch time in VxWorks is on the order of a few micro-seconds (since there is a lot smaller context to save and restore). VxWorks does not have full "process"; it only has tasks, or "threads", or light weight processes as some people like to call them.

Like any other multi-threaded environments (or MP environments), care should be taken when writing multi-tasking code. Each routine should be written carefully to be re-entrant (if it is going to be called from multiple contexts), semaphores are used a lot for this. And static variables are frowned upon. Sometimes, when porting a UNIX application, you may need to add "task variables" for this reason (as done for rpcLib in VxWorks).

VxWorks: minimal interrupt latency (e.g. spl's are quasi-implemented as semaphores). Traditional UNIX: high interrupt latency (e.g. spl's are implemented as interrupt lock and unlock calls).

VxWorks: priority interrupt-driven preemption, optional round-robin time-slicing.
Traditional UNIX: prioritized round-robin preemptive time-slicing. Since VxWorks is just a glorified "program" it can be changed and customized pretty easily. Task scheduling can be customized as desired, for example.

VxWorks networking code, however, is very UNIX compatible. It is relatively easy to port socket based code to VxWorks

VxWorks most definitely is not a "realtime UNIX", or a varient of UNIX as often misunderstood by some people. The confusion perhaps is due to the fact that UNIX hosts are used most widely to develop applications for VxWorks (and VxWorks itself).