Real–Time Operating System Case Studies (Motivation)

Magnetic Resonance Imaging (MRI)

Real–Time Needs:

- Scanner control – Modulating magnetic field to get images (changes axis of rotation of hydrogen nuclei)
- Acquisition and processing of images

Other Needs:

- Network and bus support (Ethernet and SCSI)
- To ride it across several hardware generations (want to be able to think of it as renting hardware)
- Network cross–development for quick and easy downloads and an overall development cycle

Automatic Call Distribution System (ACDs)

Real–Time Needs:

- “Time–critical” response – system has to handle over a million calls a day.

Other Needs:

- Compatibility with UNIX–based existing software
- Convenient upgrade path

Multicomputer Performance Monitoring, Probe Data Acquisition System

Real–Time Needs:

- “Time–critical” response to manage data flow.
- Real–time visualization of selected data

Other Needs:

- UNIX compatible development environment
- Network facilities
- VMEbus support
- Upgrade path
How to met these requirements?
Use real–time operating system (OS–9, Window–CE, VxWorks)

VxWorks Real–Time Operating System and Development Environment

Why has VxWorks gained acceptance?
- Based on 3 key technologies
  1. UNIX workstation development environment
  2. Ethernet (implies bus–based chassis)
  3. Powerful single board computers (> .5MBytes of memory, 68020, 030, 040, Intel 960, MIPS R3000, SPARC, etc.)
- UNIX compatibility – Uses existing UNIX features
- High performance – It out performs existing OSs
- Industry Acceptance – Success builds on success. It has continuity (WindRiver Systems, the VxWorks source, acquiring ISI)

VxWorks Overview
- VxWorks/UNIX Partnership (VxWorks for time–critical tasks and UNIX for program development — best of both worlds)
- Efficient Build/Download/Test Cycle
  1. UNIX tools (editors, make, printers, disks, GNU C compiler, etc.)
  2. VxWorks interactive shell
  3. When finished debugging facilities can be removed and application linked with VxWorks and put in ROM
- Multitasking & Intertask Communications
  1. Multitasking – application constructed as a set of independent tasks each with own thread of execution & set of system resources. VxWorks provides mechanisms to spawn tasks.
  2. Intertask communications – allows task synchronization and communications. (semaphores, message queues, pipes, sockets and signals)
• Network – UNIX network facilities for
  1. Debugging
  2. Final system non–real–time services
  3. remote file access
  4. remote login
  5. remote procedure call
• Module Loader & System Symbol Table – Operating System
  available to application as extensive set of C subroutines. Dynamic
  and even interactive access to all loaded modules based on
  1. System symbol table
  2. Loader with run–time linking
  3. Unloader to allow modules to be removed
• Shell – interactive program opening door to VxWorks facilities
  1. Can interpret and execute almost all C–language expressions
  2. Call functions (including VxWorks system functions)
  3. Define, set or examine variables
  4. History facility with \texttt{vi}–like command–line editing
• Debugging
  1. Symbolic disassembler can disassemble any loaded module
  2. Traceback facility – listing of sequence of nested subroutine
     calls of task
  3. Trapping of hardware exceptions
  4. Breakpoint and single–stepping
  5. Register and memory display and modification
  6. Routines for system and task status
• Performance Evaluation
  1. \texttt{timexLib} to time execution
  2. \texttt{spyLib} for task CPU time utilized
• I/O System – Uniform device–independent access to many kinds
  of devices
  1. Seven basic I/O functions: \texttt{creat()}, \texttt{remove()}, \texttt{open()}, \texttt{close()},
     \texttt{read()}, \texttt{write()}, and \texttt{ioctl()}. Also \texttt{printf()} and \texttt{scanf()} pro-
     vided and build on these functions.
2. \textit{ansiStdio} package includes \texttt{fopen()}, \texttt{fclose()}, \texttt{fread()}, \texttt{fwrite()}, \texttt{getch()}, \texttt{putch()}


- Local File Systems
  1. DOS – disks created with VxWorks DOS and DOS PCs may be freely interchanged
  2. RT–11 – appropriate for real–time since all files are contiguous, lacks hierarchy
  3. Raw disk file system for speed

- Utility Libraries
  1. Interrupt handling support – handles interrupts and traps without having to resort to assembly language coding. Routines to connect C routine to vectors and control interrupt level.
  2. Watchdog times – caller can schedule execution of routines after time delay unless canceled first
  3. Message logging – error or status messages can be sent to task for formatting and passing to system console, disk, and accessible memory
  4. Memory allocation – dynamic allocating, freeing and reallocating blocks of memory from a “memory pool”
  5. String formatting and scanning – ANSI C library string formatting and scanning subroutines
  6. Linear and ring buffer manipulation – copying, filling, comparing functions optimized for speed
  7. Linked–list manipulations – complete set of routines for creating and manipulating doubly–linked lists
Libraries (VxWorks Reference Manual)
A selected sampling of VxWorks libraries (not all subroutines are implemented)

ansiCtype
Testing character type (alphanumeric, letter, control character, decimal digit, etc.) and upper/lower case conversion

ansiMath
Math library (sin, cos, pow, exp, log, floor, ceil, fmod, modf, etc.). Functions take double arguments and return double values.

ansiStdio
Numerous input/output functions (fopen, fclose, fgetc, fputc, fread, fwrite, fprintf, fscanf, getchar, putchar, rewind, scanf, gets, puts, etc.)

ansiStdlib
Assorted routines to take absolute value, convert strings to other data types, integer division, quick sort, random number generation, etc.

ansiString
Assorted routines for copying, comparing, searching, concatenating and finding the length of strings

ansiTime
Many functions dealing with calendar time (date and time) and some deal with local time (e.g., specific time zone). Functions to acquire time and do conversion of time to string, etc.

bLib
A buffer manipulation library allowing comparison, order of bytes inversion, swapping contents, copying, filling, finding first or last occurrence of string in buffer, etc.

cacheLib
Architecture–independent routines for managing instruction and data caches. Functions include cacheEnable, cacheDisable, cacheLock, cacheUnlock, cacheFlush, cacheInvalidate, and routines to do mapping between physical and virtual addresses.

clockLib
A clock interface, as defined in the IEEE standard, POSIX 1003.4, Draft 12. Allows resolution to be determined and getting and setting of time.

dbgLib
The VxWorks debugging facilities used from the shell (b, bd, d, cret, s, l, etc)
**dosFsLib**
Provides services for file-oriented drivers to use the MS–DOS file standard, taking care of buffering, directory maintenance, and file system details.

**errnoLib**
Get or set error status. Most VxWorks functions return ERROR when they detect an error, or NULL in the case of functions returning pointers. UNIX uses a global variable `errno`, however, in VxWorks there are many talk and interrupt contexts that share common memory space and therefore conflict in their use of this global variable must be resolved. For tasks, VxWorks maintains the `errno` for each context separately (saved and restored at each task switch). The value of `errno` for a non-executing task is stored in the task’s TCB and is always available. For interrupt service routines `errno` is saved on the interrupt stack and can be examined or modified by the interrupt service routine.

**fppShow**
Provides interface for viewing floating-point coprocessor registers. Allows display of coprocessor registers on a per-task basis using `fppTaskRegsShow`.

**ioLib**
Provides interface to basic I/O system. Includes `create`, `open`, `close`, `read`, `write`, etc.

**ledLib**
Provides a `vi`–like line editing layer on top of a tty device.

**mathALib**
More math, include floats (e.g., `sinf`)

**memLib**
Facilities for managing the allocation of blocks of memory from ranges of memory called memory partitions. The system memory partition is created when the kernel is initialized and is controlled by a configuration file.

**memPartLib**
Facilities for creating and changing allocation to a memory partition.

**memShow**
Allows viewing of memory partition information.

**moduleLib**
VxWorks module management allowing the loading and unloading of modules. Shows the current status of all loaded modules with `moduleShow`.

**pipeDrv**
Provides mechanism that lets tasks communicate with each other through the standard I/O interface. VxWorks pipes are “message oriented” and differ significantly from UNIX pipes which are stream oriented and do not preserve message boundaries.
**scsiLib**
Implements the SCSI protocol in a controller-independent manner.

**semBLib**
Provides interface to VxWorks binary semaphores.

**semCLib**
Provides interface to VxWorks counting semaphores (if count is non-zero, it is decremented and the calling task continues executing, if the count is zero, the task will be blocked).

**semLib**
Allows taking or giving of semaphore: *semTake* acquires a specific semaphore, blocking the calling task or making the semaphore unavailable and *semGive* relinquishes a specified semaphore, unblocking a pended task or making the semaphore available.

**semMLib**
Provides interface to VxWorks mutual-exclusion semaphores (like binary semaphore but tailored to address issues inherent in mutual exclusion).

**semShow**
Allows showing of semaphore statistics such as type, queuing method, task pended, etc.

**sigLib**
Provides a signal interface for tasks. Signals are used to alter the flow control of task by communicating asynchronous events within or between task contexts. Any task of an interrupt service routine can “raise” (or send) a signal to a particular task. The task being signaled will immediately suspend its current thread of execution and invoke a task-specific, user supplied, “signal handler” routine. Signals are most appropriate for error and exception handling, rather than as a general purpose intertask communications mechanism. Functions include: initialization, addition, deletion, handler identification, task suspension until signal sent, signal handler installation, etc.

**sockLib**
Provides UNIX BSD 4.3 compatible socket calls. These calls may be used to open, close, read, and write sockets, either on the same CPU or over a network. In VxWorks, sockets are the basis for intertask communications. A socket is an endpoint for communications between tasks; data is sent from one socket to another. Included in the creation of a socket is the specification of the Internet communications protocol to be used.
spiLib
Allows the monitoring of tasks’ use of the CPU. Provides display of CPU time by task, the amount of time spent at interrupt level, the amount of time spent in the kernel, and the amount of idle time.

taskInfo
Provides task information such as options, register contents, name, ID, state, etc. Useful for debugging.

taskLib
Interface to task management facilities. Allows task control including spawning, initialization, activation, deletion, suspension, resumption, priority, delay, etc.

taskShow
Allows showing of task–related information, such as register values, task status, etc.

timexLib
Used for timing programs or functions. The VxWorks system clock is used as a time base.

usrLib
Provides routines meant to be executed from the VxWorks shell. Examples include: help, period, repeat, sp, i, show, ts, tr, td, m, d, cd, ld, mRegs, logout, h, etc.

wdLib
Provides a general watchdog timer facility. Task can create a watchdog timer, then use it to provide events that can happen after the specified delay, outside the context of the task itself.

wdShow
Allows showing of watchdog statistics.

Host Tools
Executable commands available at host prompt.
Examples:

hex
Convert an a.out format object file into Motorola hex records.

compress
Tool reduces the size of a file using the modified Lempel–Ziv method. Also used with –d switch to decompress file.
VxWorks Real–Time OS Basics

Modern real–time operating systems are based on two concepts:

1. Multitasking – applications constructed as a set of independent tasks, and
2. Intertask communications – fast semaphores, message queues, UNIX–like pipes and sockets allow these tasks to synchronize and communicate to coordinate activities.

Another Key Facility: high performance interrupt handling. VxWorks runs in a special context of its own (outside context of any task).

Multitasking

Creates appearance of many programs executing concurrently by using kernel to interleave their execution on the basis of a scheduling algorithm. Each apparently independent program called a task. Each task has a context. On a context switch, context info is saved in the task control block (TCB). Info saved:

- task’s program counter
- cpu registers and optionally the floating point regs.
- a stack for dynamic variables and function calls
- I/O assignments for standard input, output, and error
- a delay timer
- a timeslice timer
- kernel control structures
- signal handlers
- debugging and performance monitoring values
Task State Transition

ready – the state of a task that is not waiting for any resource other than the CPU.

pended – the state of a task that is blocked due to the unavailability of some resource.

delayed – the state of a task that is asleep for some duration.

suspended – when created the task enters this state. Activation is needed to enter ready state (taskResume/ taskActivate). Also used for debugging.

Task Scheduling

Two types used:

1. Preemptive Priority Scheduling – task assigned priority and kernel ensures that highest priority task runs, even preempting one of lower priority already running.

2. Round–Robin Scheduling – At a priority level each task given an equal time slice on a rotating basis.

kernelTimeslice() used to control round–robin scheduling.

taskPrioritySet() used to change priority of a task.
Preemption Locks

Scheduler disabled using `tasklock()`, priority disabling will not occur, but if task blocks or suspends, the scheduler will select the next highest priority eligible task until it unblocks. Note that interrupt handling is not blocked.

`taskUnlock` disables locking feature.

Preemption locks useful for mutual exclusion, but duration should be kept to a minimum.