

Rialto

- Rialto is a real time system developed at Microsoft Research based on the windows NT architecture.
- The main idea: rewriting the kernel and scheduler so that they support real time deadlines.
- Unlike RTLinux it doesn't require the separation of the hard real time functionality from the rest of the code.
- It uses the `beginConstraint` and `endConstraint` function calls that pass the parameters of the real time portion of task to the scheduler.
- The parameters passed to the scheduler:
 - release time
 - deadline
 - running time estimate
 - criticality (critical=hard constraint , non-critical=soft)
- The scheduler returns the feasibility of scheduling the task. This allows the task to perform load shedding if possible, or fail gracefully if not.

Rialto

- The scheduling in Rialto is quite complicated:
 1. Tasks can be granted CPU time reservations. This is the average amount of CPU time they should get.
 2. The scheduler is based on a variation of preemptive LST which uses the maximal amount of time before a task has to start running (deadline-running time estimate).
 3. Critical tasks always run before non-critical tasks.
- The Rialto system is system-resource aware.
- Tasks can negotiate with the **resource manager** in order to reserve the resources they need.
- The resource manager arbitrates the use of resources, and can cause an existing task to relinquish resources allocated to it.

Rialto

- Rialto doesn't handle the problem of the kernel disabling interrupts.
- The schedulability test was not fully implemented originally.
- Summary:
 - Rialto uses the approach of rewriting an existing kernel in order to allow support for real time performance.
 - built as “academic” work, was never widely used.
 - uses a novel scheduling approach, but because it is complicated it is sensitive to many implementation details.

VxWorks

- VxWorks is a commercial hard real time operating system developed by Wind River Systems.
- The main idea: use a monolithic kernel to schedule user tasks according to user defined priorities. Maximize kernel timing predictability.
- Gives the users maximal control.
- A dedicated real time system, not intended as a general purpose OS.
- Lacks many modern OS features that interfere with real time performance (flat memory model, no paging).

VxWorks

- Scheduling is done using a preemptive priority driven approach, priorities are chosen arbitrarily by the user (0-256).
- Priorities can be changed by the user at runtime but this is discouraged.
- A user can lock a task so that it can't be preempted even by higher priority tasks or interrupts.
- This allows the use of the fixed priority response time analysis to check schedulability offline.

VxWorks

- Is resource sharing aware and has a priority inheritance built in.
- Optimizations in implementation of context switches and the return from interrupts.
- The kernel never disables NMI (non-maskable interrupts) so they are always available to the user.
- Lacks many modern OS features.
- Guaranteeing the deadlines is the responsibility of the user at design time.
- Doesn't support most modern applications and APIs (only a small subset of POSIX).
- Despite the flat memory model, dynamic memory allocation still causes memory fragmenting, which increases timing unpredictability.

VxWorks

- Summary:
 - a dedicated and widely used real time system.
 - offers the user maximal control , but also passes him responsibility for the deadlines.
 - lacks many modern OS features.

Real-Time POSIX

- Overview
 - POSIX *Portable Operating System Interface*
 - an API standard
 - POSIX 1003.1: defines the basic functions of a Unix os
 - POSIX thread and real-time extensions
 - POSIX 1003.1b: real-time extension
 - prioritized scheduling, enhanced signals, IPC primitives, high-resolution timer, memory locking, (a)synchronized I/O, contiguous files, etc
 - POSIX 1003.1c: thread extension
 - creation of threads and management of their execution
- Threads
 - basic units of concurrency
 - functions
 - create/initialize/destroy threads
 - manage thread resources
 - schedule executions of threads
 - read/set attributes of a thread
 - priority, scheduling policy, stack size and address, etc.

Real-Time POSIX

- Clocks and timers
 - time made visible to the application threads
 - the system may have more than one clock
 - functions
 - get/set time of a specified clock
 - create/set/cancel/destroy timers (up to 32 timers)
 - timer resolution: nanosecond
- Scheduling interface
 - support fixed priority scheduling with at least 32 priority levels
 - a thread may
 - (1) set and get its own priority and priorities of other threads
 - (2) choose among FIFO, round-robin and implementation-specific policies
 - in principle, it is possible to support the EDF or other dynamic priority algorithms, but with high implementation overhead
 - different threads within the same process may be scheduled according to different scheduling policies

Real-Time POSIX

- Synchronization
 - semaphores
 - simple; very low overhead
 - unable to control priority inversion
 - mutexes
 - support both priority inheritance and priority ceiling protocols
 - condition variables
 - allow a thread to lock a mutex depending on one or more conditions being true
 - mutexes are associated with a condition variable which defines the waited-for condition
- Interprocess communication
 - messages: prioritized
 - send/receive: nonblocking
 - receive notification: no check necessary for message arrivals
 - signals
 - primarily for event notification and software interrupt
 - at least eight application-defined signals
 - delivered in priority order
 - can carry data
 - queues blocked signals

Real-Time POSIX

- Shared memory and memory locking
 - a process can create a shared memory object
 - in case of virtual memory, applications can control memory residency of their code and data by locking the entire memory or specified range of address space
- File I/O
 - synchronized I/O
 - two levels of sync: data integrity, file integrity
 - asynchronous I/O
 - I/O concurrently with CPU processing