Real Time Operating Systems

RT features in commercial products

Lecturer:
Prof. William Fornaciari
Politecnico di Milano
fornacia@elet.polimi.it
www.elet.polimi.it/~fornacia

Outline

- *Introduction to Real Time Systems*
- *The role of a RTOS*
- *Time constraints*
- *The problem of scheduling RT activities*
- *Classification of scheduling strategies*
- *Review of RT scheduling approaches*
- Linux, Unix SVR4, Win2000
- VxWork real-time features
- Windows CE real-time features

Linux Scheduling

- In addition to traditional UNIX-like scheduling, two scheduling classes for soft-RT are added
  - SCHED_FIFO: First-in-first-out real-time threads
  - SCHED_RR: Round-robin real-time threads
  - SCHED_OTHER: Other, non-real-time threads
- Within each class multiple priorities may be used, with priority for RT classes higher than for SCHED_OTHER class

FIFO threads: rules

- The system can interrupt a FIFO thread only if:
  - Another FIFO thread of higher priority becomes ready
  - The running FIFO thread blocks waiting for an event
  - The running FIFO thread voluntarily gives up the processor (sys call *sched_yield*)
- An interrupted FIFO thread is suspended in the queue with its priority
- When a FIFO thread becomes ready, it can preempt current thread if its priority is higher
- SCHED_RR is similar to FIFO, except for the addition of a time quote associated with each thread

RR and OTHER threads: scheduling rules

- SCHED_RR is similar to FIFO, except for the addition of a time quote associated with each thread
- When SCHED_RR thread executed for its quote, it is suspended and a RT thread of ≥ priority is selected

- A SCHED_OTHER thread can only execute if there are no RT threads ready to execute, using traditional UNIX scheduling

FIFO vs RR: Example

- Three processes with relative priority, where all waiting threads are ready to execute and no higher priority thread is awakened while a thread is running

A	minimum
B	middle
C	middle
D	maximum

(a) Relative thread priorities

D → B → C → A →

(b) Flow with FIFO scheduling

D → B → C → B → C → A →

(c) Flow with RR scheduling

UNIX SVR4 Scheduling

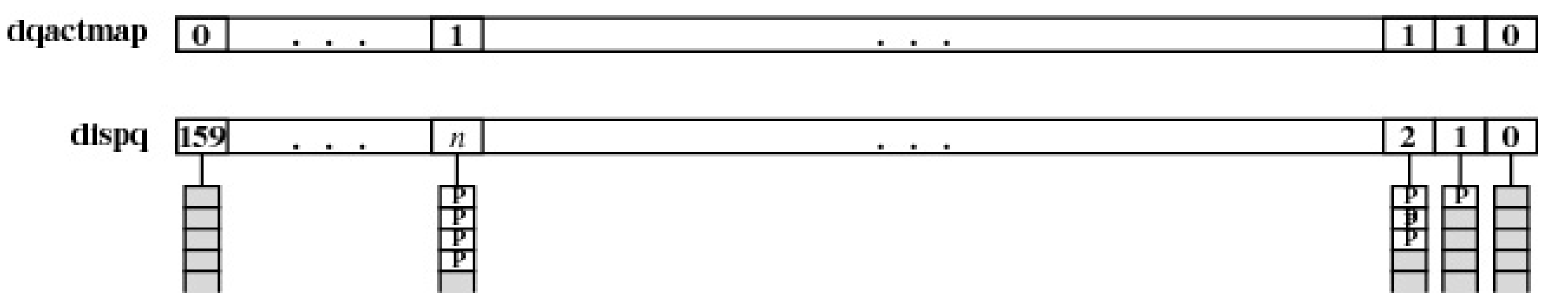
- Complete overhaul of the UNIX scheduler
- Highest preference to real-time processes
- Next-highest to kernel-mode processes
- Lowest preference to other user-mode processes (time shared processes)

UNIX SVR4: major modifications

- Addition of a preemptable static priority scheduler, with 160 priorities partitioned in three classes
  - Real Time (159-100)
    - Processes are selected to run before any Kernel or TS process
  - Kernel (99-60)
    - Processes are selected to run before TS processes, but must defer to RT processes
  - Time Shared (59-0)
    - Intended for user other than no RT applications
- Insertion of preemption points
  - Since basic kernel is not preemptive, safe points to interrupt kernel and schedule new processes have been indetified
  - Safe place: region of code where all kernel data structures are either updated and consistent or locked via a semaphore

SVR4 Scheduling: Dispatch Queues

- A dispatch queue is associated with each priority level and processes at a given level execute RR
- *dqactmap*: bit-map vector, one bit per priority level
  - 1 for nonempty queues
  - The scheduler checks *dqactmap* and dispatch a ready process from the highest priority nonempty queue
- In preemption points, the kernel checks a flag (*kprunrun*), indicating the existence of RT processes in the Ready state. If it exists, a preemption will



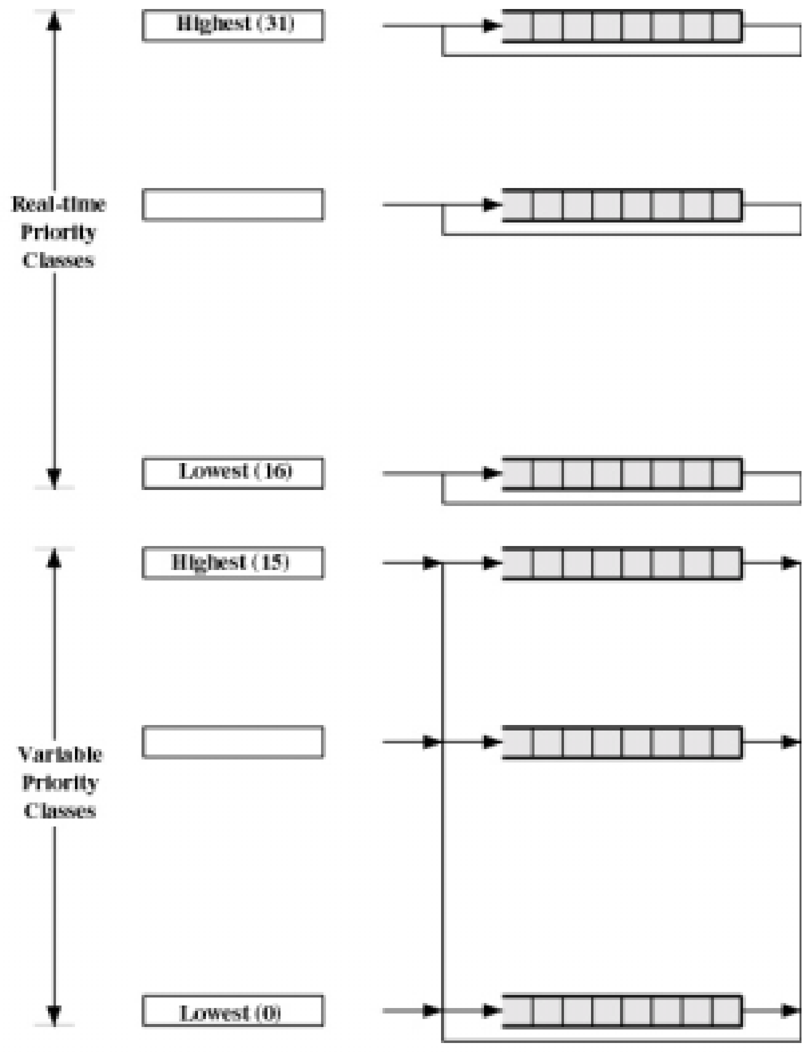
SVR4 Scheduling: Dispatch Queues

- The priority of TS class is variable
  - *Reduce* if it uses time quantum
  - *Rise* if Process is blocked on event
- Variable time quantum for TS class
  - 100 ms for priority 0
  - 10 ms for priority 59
- Each RT process has fixed priority and fixed time quantum

Windows 2000 Scheduling

- Flexible priority-driven preemptive scheduler
  - RR scheduling within each level
  - For some levels variable priority based on current thread activity
- Priorities organized into two bands or classes, each consisting of 16 priority levels
  - Real-time
  - Variable
- RT threads have precedence over other threads
- Lower priority thread is preempted by higher priority

W2k: Thread Dispatching Priorities

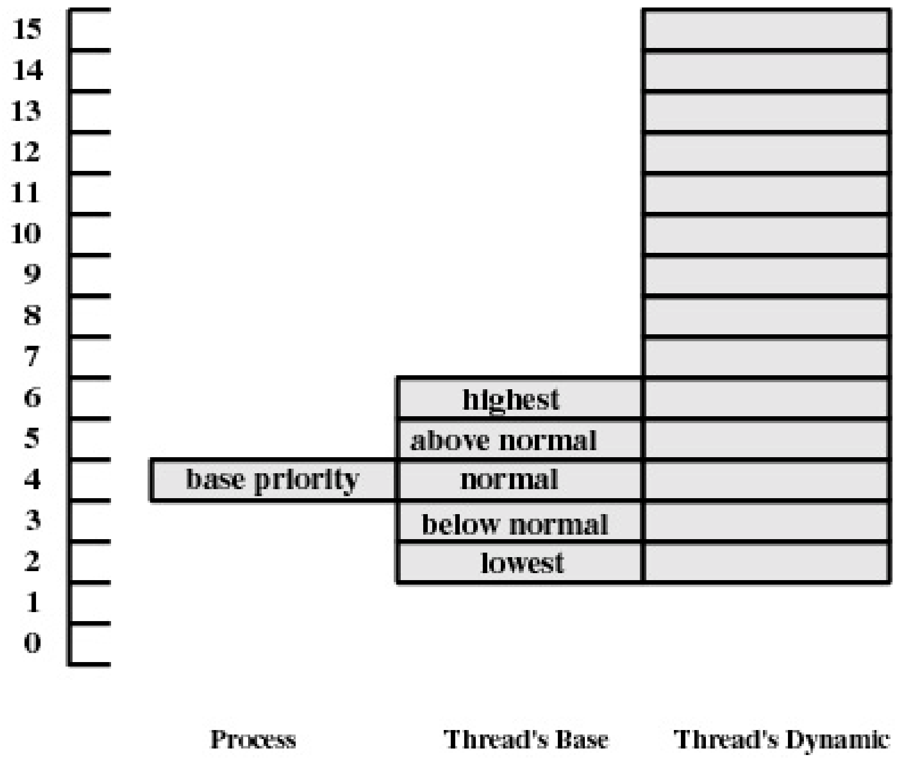


Windows 2000 Priorities

- RT priority class
  - all threads have fixed priority, each level is RR
- Variable priority class
  - Prority can move up/down during thread's lifetime, but not move to RT class
  - Initial priority of a thread
    - Process base priority (0..15) attribute of process object
    - Thread base priority. Relative to that of the process
  - The initial priority of a thread in in a range around that of process (base process + base thread)
  - The dynamic priority may never fall below lower range of thread base prority and over 15
  - W2K executive tends to rise I/O bound (interactive) threads

W2K Priorities Relationship: Example

- Process base priority is 4
- Each threads objects have base priority 2-6
- Dynamic priority may fluctuate from 2 to 15



Outline

- VxWork Real-Time Features
  - Introduction
  - Kernel
  - Memory Architecture
  - Real-Time Measurement Tools

WxWorks Definition

- Unix and Windows are not appropriate for real time systems
- VxWorks has to be used with another operating system: it handles the critical real-time chores, while the host machine is used for program development and non real-time applications

Kernel

- All applications consist of a tasks
- A task is the basic unit to which the operating system allocates processor time
- Tasks are essential to organize applications into independent, though cooperating, programs

Processes

- WxWorks creates a context for each task
- A task's context includes
  - The CPU register
  - A stack for dynamic variables and function calls
  - I/O assignments for standard input, output and error
  - A delay timer
  - A timeslice timer
  - Signal handlers

Scheduling

- VxWorks uses, by default, a priority-base preemptive scheduling algorithm
- 256 priority levels:
  - 0: highest priority level for time critical activities
  - 1-254
  - 255: lower priority level
- Higher priority levels are assigned to real-time applications
- Preemptive scheduling can be augmented with round-robin scheduling

Scheduling

- Threads with higher priority are scheduled to run first
- Threads with the same priority run in a round-robin fashion
- Lower priority threads do not run until higher priority threads have finished
- Lower priority threads can be suspended by higher priority threads

Scheduling

- Time Slice:
  - Threads run for a time slice
  - A run-time counter is kept for each task, and incremented on every clock cycle
  - When the time slice is completed, the task is placed at the tail of the queue of tasks at its priority
  - If a task is preempted by a higher priority tasks during its interval, its run-time count is saved

Interrupts

- VxWorks supplies routines for handling hardware interrupts and software traps without having to resort to assembly language coding
- Routines are provided to connect C routines to hardware interrupt vectors, and to manipulate the processor interrupt level

Kernel:

Interrupts

- For the fastest possible response to interrupts, interrupt service routines (ISRs) in VxWorks run in a special context outside of any task's context
- Interrupt handling involves no task context switch
- All ISRs use the same interrupt stack

Interrupts

- When an interrupt occurs, the connected C function is called at interrupt level with the specified argument
- When the interrupt handling is finished, the connected function returns
- A routine connected to an interrupt in this way is called an interrupt service routine (ISR)

Intertask Communications

- Intertask communications can be realized through the following mechanism:
  - Shared Data Structures (global variables, linear buffers, ring buffers, linked lists, and pointers)
  - Mutual exclusion
  - Semaphores
  - Message Queues
  - Pipes
  - Signals

Memory Architecture

- VxWorks supplies management facility for dynamically allocating, freeing, and reallocating blocks of memory from a memory pool
- Blocks of arbitrary size can be allocated
- The size of the memory pool can be specified
- VxWorks manages several separate memory pools

Virtual Memory

- VxWorks provides two levels of virtual memory support
- The basic level is bundled with VxWorks and provides caching on a per-page basis
- The full level is unbundled, and requires an optional component, VxVMI
- VxVMI provides:
  - Write protection of text segments
  - VxWorks exception vector table
  - Architecture-independent interface to the CPU's memory management unit (MMU)

Real-Time Measurement Tools

- VxWorks provides various timing facilities to understand and optimize the performance of a real-time system
- VxWorks execution timer can time any subroutine or group of subroutines
- The timer can also repeatedly execute a group of routines until the time of a single iteration is known to a reasonable accuracy

Spy utility

- There is also a spy utility which provides CPU utilization information for each task:
  - CPU time consumed
  - Time spent at interrupt level
  - Amount of idle time
- Time is displayed in ticks and in percentages

Outline

- Windows CE Real-Time Features
  - Introduction
  - Kernel
  - Memory Architecture
  - Real-Time Measurement Tools

Windows CE Definition

- Windows CE is the modular real-time embedded operating system for small footprint and mobile 32-bit intelligent and connected device. (Microsoft)

Kernel

- All applications consist of a process and one or more threads
- Each process starts with a single thread: primary thread
- A thread is the basic unit to which the operating system allocates processor time

Processes

- Windows CE can run up to 32 processes at one time
- When a Windows CE-based device starts:
  - The OS creates a single 4-GB virtual address space
  - It is divided into 33 slots of 32 MB each
  - When a process start the OS assigns it one slot
  - Slot zero is for the currently running process

Threads

- Threads are implemented in kernel space
- Problems to avoid in multi-thread applications:
  - *Deadllocks*: when all threads are in the blocked state, Windows CE enters in the idle mode
  - *Race conditions*: thread sharing common resources must coordinate their work by using a method of syncronization

Scheduling

- Windows CE uses a priority-base time-slice algorithm
- 256 priority levels:
  - 0: highest priority level for time critical activities
  - 1-254
  - 255: lower priority level
- Higher priority levels are assigned to real-time applications
- Process priority classes are not supported

Scheduling

- Threads with higher priority are scheduled to run first
- Threads with the same priority run in a round-robin fashion
- Lower priority threads do not run until higher priority threads have finished
- Lower priority threads can be suspended by higher priority threads

Scheduling

- Time Quantum:
  - Threads run for a time slice
  - By setting the time quantum of a thread to zero, a round-robin scheduling algorithm can change to a run-to-completion algorithm
  - Threads set to run-to-completion can be preempted only by higher priority threads
  - The default quantum is 100 milliseconds

Scheduling

- Priority Inversion:
  - Thread priorities are fixed and do not change
  - Priority inversion occurs when a high priority thread is blocked by a lower priority thread
  - Windows CE guarantees the handling of priority inversion only to a depth of one level

Interrupts

- The use of interrupts requires balancing performance against ease of use
- Interrupt processing is split into two steps:
  - **ISR**: interrupt service routine
    - It directs the kernel to launch the appropriate IST
  - **IST**: interrupt service thread
    - It contains the interrupt handling

Interrupts

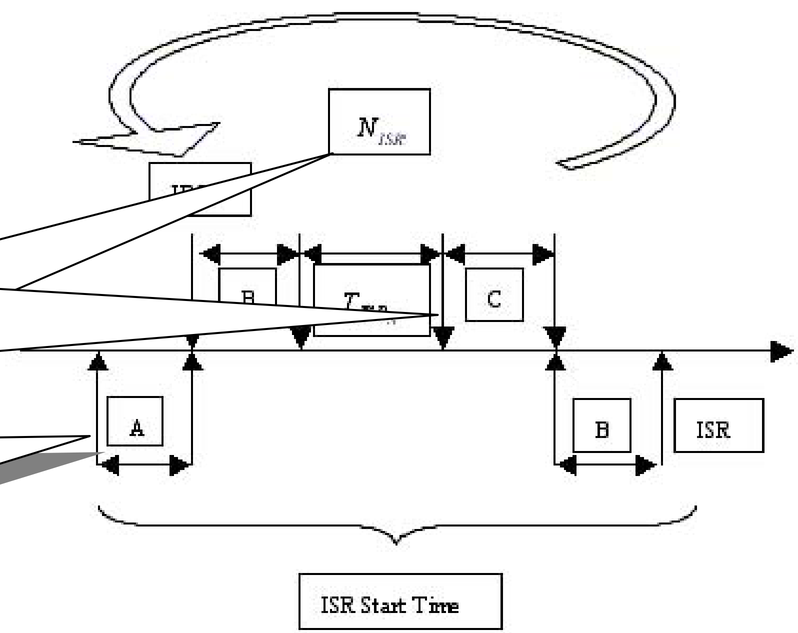
- Nested Interrupts:
  - Windows CE handling of nested ISR calls:
    - The kernel disables all IRQ lines for the same and lower priorities
    - If a higher priority IRQ arrives, the current ISR state is stored
    - The kernel calls the higher-priority ISR
    - When the request is completed, the kernel loads the original ISR

Interrupts

- Interrupt latencies:
  - They are bounded for threads locked in memory if paging does not occur
  - It is possible to calculate the worst-case latencies

Interrupts

- Interrupt latencies:
  - ISR latency
    - Time from when an IRQ is set at the CPU to when the ISR begins to run



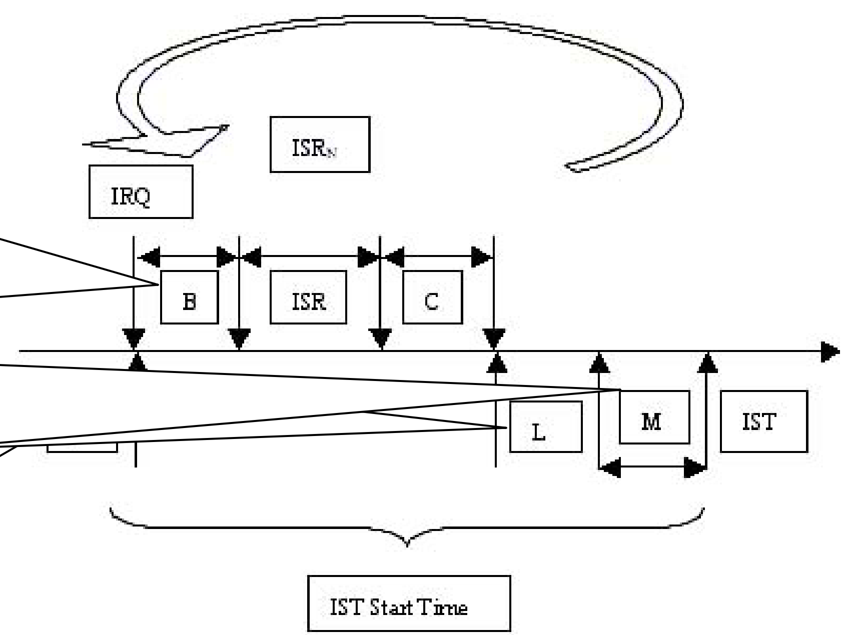
C = Time between when an ISR returns to the kernel and the kernel stops processing

Interrupts

- Interrupt latencies:
  - IST latency:
    - Time from when an ISR finishes execution to when IST begins execution

M = Time to schedule a thread

interrupt.



Interrupts

- Interrupt latencies:
  - ISR latency:
    - Windows CE is in control of the variables A, B, C, all of which are bounded
  - IST latency:
    - Windows CE is in control of the variables B, C, L, and M all of which are bounded.

Synchronization

- Synchronization solutions:
  - Wait functions
  - Interlocked functions
  - Synchronization objects:
    - Event objects
    - Mutex
    - Critical sections
    - Semaphores
  - Messages

Memory Architecture

- The RAM is divided into two areas:
  - The **object store** resembles a permanent, virtual RAM disk
  - The **program memory** works like a RAM, storing heaps and stacks
- The maximum size for the RAM file system is 256 MB
- The boundary between the two areas is movable

Virtual Memory

- Windows CE implements a paged virtual memory management system
- Each process has 32 MB of virtual address space.
- This space is used for: processes, DLLs, heaps, stacks and virtual memory allocation
- Allocating virtual memory may affect real-time performance

Heap Memory

- Windows CE only supports allocating fixed blocks in the heap
- It can lead to the heap becoming fragmented
- Fragmentation can affect performance
- To reduce the impact on allocating heap memory, a process should allocate it before proceeding with normal processing

Stack Memory

- When a thread is created, Windows CE reserves enough memory space to accomodate the maximum stack size
- If no memory space is available, the thread is suspended until the space is granted
- To prevent the initial commitment from affecting performance, ensure that the thread is scheduled at least once before real-time processing

Real-Time Measurement Tools

- Two kernel-level tools to test real-time performance:
  - Interrupt Timing Analysis (IntrTime)
    - IntrTime Command Prompt
    - Extrnal Interrupt Response
  - Scheduler Timing Analysis (CEBench)

Interrupt Timing Analysis

- IntrTime Command Prompt:
  - The aim is the measurements of ISR and IST latencies
  - The measurements are done using the system clock timer
  - The following variations can be introduced:
    - Setting the IST to run on various priorities
    - Flushing or not flushing the cache after each interrupt
    - Changing the ISR rate and number of interrupts captured
    - Printing or outputting to file the collected results

Interrupt Timing Analysis

- External Interrupt Response
  - The previous method relies on the timer on the device itself, which may affect the measurements
  - In this analysis two machine can be set up:
    - A workstation that generates an external interrupt and measures the time it takes to receive acknowledgements from ISR and IST routines
    - Device-under-test that receives the external interrupt and toggles output lines when ISR and IST routines are reached

Scheduler Timing Analysis

- It measures the time required to perform basic kernel operations
- Timing samples are collected for the following performance metrics:
  - Acquire/release critical section
  - Wait/signal an event
  - Semaphores
  - Mutexes
  - Voluntary yield using **Sleep(0)**

Scheduler Timing Analysis

- The **QueryPerformanceCounter** is used to obtain timing information
- Using the function call to get time stamps isnot free
- In cases where the operation takes a very short time to complete, the overhead of the function call becomes significant