UNIVERSITÀ
DEGLI STUDI
FIRENZE

DINFO
Dipartimento di
Ingegneria dell'Informazione

Software Engineering for Embedded Systems

# The real-time operating system VxWorks

Laura Carnevali, Imad Zaza

*Software Technologies Lab*
*Department of Information Engineering*
*University of Florence*
`http://stlab.dinfo.unifi.it`
{laura.carnevali,imad.zaza}@unifi.it

# Outline

1. Credits
2. Introduction to VxWorks

# Credits

# Credits

- Part of this material is taken from the course ”Real-Time Systems for Automation” given by Prof. Paolo Torroni (University of Bologna)
- These slides are authorized for personal use only
- Any other use, redistribution, and profit sale of the slides (in any form) requires the consent of the copyright owners

# Introduction to VxWorks

# VxWorks features

- Commercial Real-Time Operating System (RTOS) developed by Wind River (which is a subsidiary of Intel Corporation)
- UNIX-based RTOS
- Native 64-bit RTOS (x86-64 is the only 64-bit supported architecture)
- Multi-core support (symmetric/asymmetric)
- VxWorks 7: latest version since 2014 (used in our lab session)
- Probably the most widespread commercial RTOS
    - Safety certification: RTCA DO-178, ISO 26262, IEC 61508

COMAU

KUKA

SIEMENS

MITSUBISHI ELECTRIC

ABB

NASA

YASKAWA
NXC100 robot controller offers high performance, open communication, and integrated cell control
NXC100
ROBOT CONTROLLER

BOSCH

Schneider Electric

# Flat memory model

- No virtual memory or page swapping
- VXWorks Virtual Memory Interface (VxVMI)[1]
  - Each task has its own context
  - Single common address space
- Prior to VxWorks 6.x
  - In VxWorks 5.5, all the tasks can access all the memory area visible to the OS, notwithstanding access protection provided by the VxVMI optional feature
- Since VxWorks 6.x
  - Real Time Processes (RTPs) are isolated from the kernel and from each other

[1] See vmLib in reference of VxWorks IDE

# More on the memory model since VxWorks 6.x

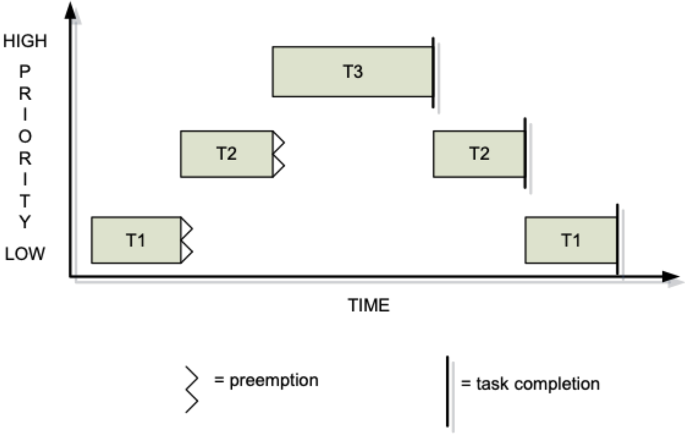
- Memory divided into the kernel space and the user space
  - Tasks in the kernel space execute in supervisor mode (aka kernel mode)
  - Tasks in the user space (RTPs) execute in user mode
  - The Memory Management Unit (MMU) of the CPU enforces the isolation between the kernel space and the user space and also among the RTPs
- System call interface: the only "gate" mechanism . . .
  - . . . allowing an RTP to call out to the kernel code
  - . . . promoting an RTP from running in user mode to running in kernel mode
  - . . . allowing an RTP to access the kernel space

# Typical components of RTOS kernels

- Scheduler: determines which task executes on each core at each time
- Objects: special kernel constructs that help developers create applications for real-time embedded systems, e.g., tasks, semaphores, message queues
- Services: operations performed on an object or general operations, e.g., timing, interrupt handling, and resource management

# VxWorks kernel

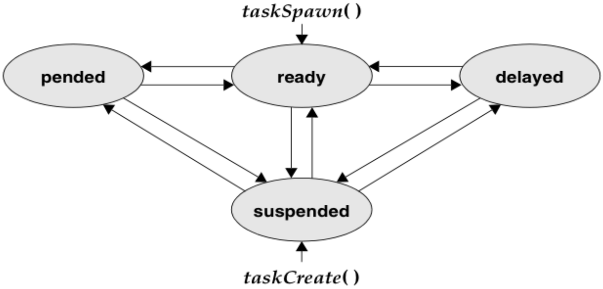
- Multitasking kernel based on a microkernel with preemptive scheduling (priority-based or round-robin) and fast interrupt response
  - Priority-based scheduler with 256 priority levels and unlimited number of tasks



# VxWorks objects: tasks

- Each task has its own context, i.e., CPU environment and system resources
- On a context switch, task context is saved in the Task Control Block (TCB)
- Task context includes:
    - Tasks run in a single common address space
    - a thread of execution (the task's program counter)
    - the tasks virtual memory context (if process support is included)
    - the CPU registers and (optionally) coprocessor registers
    - stacks for dynamic variables and function calls
    - I/O assignments for standard input, output, and error
    - ...

# Task execution states



| | | | |
|---|---|---|---|
| | → | ready | taskSpawn( ) |
| | → | suspended | taskCreate( ) |
| ready | → | pended | semTake( ) / msgQReceive( ) |
| ready | → | delayed | taskDelay( ) |
| ready | → | suspended | taskSuspend( ) |
| pended | → | ready | semGive( ) / msgQSend( ) |
| pended | → | suspended | taskSuspend( ) |
| delayed | → | ready | expired delay |
| delayed | → | suspended | taskSuspend( ) |
| suspended | → | ready | taskResume( ) / taskActivate( ) |
| suspended | → | pended | taskResume( ) |
| suspended | → | delayed | taskResume( ) |

# Task execution states

- READY: the state of a task not waiting for any resource other than the CPU
  - VxWorks does not distinguish whether task is running (assigned CPU) or not
  - `taskSpawn()` creates and activates a task
- PENDED: the state of a task blocked due to unavailability of some resource
  - Task is blocked, waiting for some resource to be assigned to it
    e.g., waiting for a semaphore, reading from an empty message queue, ...
  - `semTake()` and `msqQReceive()` may move task from READY to PENDED
- SUSPENDED: the state of a task unavailable for (inhibited from) execution
  - `taskCreate()` creates a task, which enters the SUSPENDED state
  - `taskActivate()` activates a created task, which enters the READY state
  - `taskSuspend()` moves a task fro READY to SUSPENDED
- DELAYED: the state of a task that is asleep for some duration
  - The task waits for some time interval to elapse
  - Usually caused by calling `taskDelay()` or `nanosleep()`
- Mixed states, e.g., PEND + T (task pended with a timeout value)

# VxWorks objects: inter-task/inter-process communication

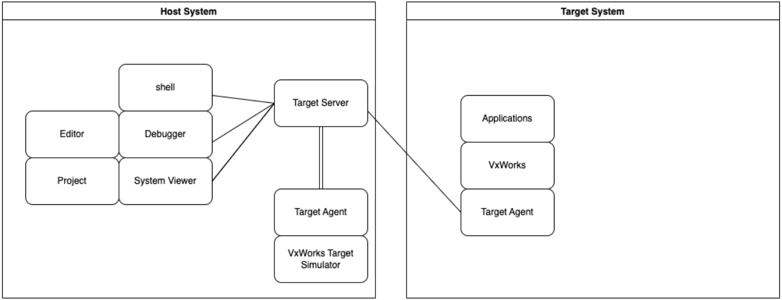
- Three types of semaphores
  - Binary semaphores
    - Let a task "pend" until an event (e.g., an interrupt) occurs
    - Used to implement precedence relations among tasks
  - Counting semaphores
    - Track resources with capacity
    - Let a task wait until an instance of the required resource is available
    - Used to implement bounded buffer-style synchronization
  - Mutex semaphores
    - Let a task acquire s lock for exclusive access to a shared resource
    - Used to implement critical sections
    - Used to solve the priority inversion problem and the secure delete problem
- Message queues, Signals, Pipes, Sockets
- Shared memory

# VxWorks services

- File systems (FSs)
  - FAT FS, raw FS, TrueFFS (for flash memories), fault-tolerant FS HRFS
- Multimedia: OpenGL (ES), OpenCV
- Security: OpenSSL, Secure boot
- Connectivity
  - IPv4, IPv6
  - Time-sensitive networking (TSN)
    - Time-sensitive transmission of data over deterministic Ethernet networks
  - Universal Serial Bus (USB)
  - Controller Area Network (CAN)
    - Message-based protocol vehicle bus standard designed to allow microcontrollers and devices to communicate without a host computer

# Workbench: VxWorks developing environment

- Reference IDE to develop and debug VxWorks applications
  - Custom distribution of the Eclipse environment, to develop real-time and embedded applications with minimal intrusion on the target system
- Includes an integrated source-code editor, C/C++ compiler and linker, the source code level debugger CrossWind, and project management facilities
- Includes SystemViewer, a visual monitoring tool to analyze the target system
- Includes command and C shell supporting control of the target
- Includes VxSim, a simulator of the target system



# Where can VxWorks be deployed?

- Target: (real or simulated) environment where VxWorks is deployed
  - Target can be a simulated environment (VxSim)
  - Target can be a prototyping hardware, e.g., Raspberry PI
- Host: environment where Workbench runs
- Once created, a target environment can be started and connected to the host
- At that point, it is possible to control the target directly from the host IDE