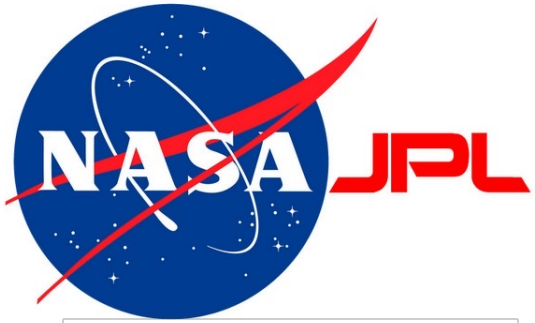
VxWorks

ENRICO MARTINI, MIRCO DE MARCHI

Overview

VxWorks is the industry's most trusted and widely deployed RTOS for critical embedded systems that must be safe and secure.

- Extensive range of board support packages (BSPs)
- Competitive advantage, time-to-market, cost savings, risk reduction, and peace of mind
- Long-term performance, security, and feature support



KUKA

BOSCH

SIEMENS

OLYMPUS

Your Vision, Our Future

Rockwell Automation

AgustaWestland
A Finmeccanica Company

Collins Aerospace

NORTHROP GRUMMAN

Choosing a RTOS

Commercial RTOS

- Organized backing
- Complete documentation
- No glitches
- Supports
- Safety certifications
- BSPs*

Open Source RTOS

- Free, but problems encountered while implementing an open-source system could quickly push deadlines and stretch budgets.

* Wind River® offers the most extensive range of board support packages (BSPs) in the embedded software industry: *68K, ARM/ARM64, Coldfire, MIPS, PPC/PPC64, RISC-V, SH, SPARC, X86/x86-64, Xscale*

History of Wind River Systems

- **1981**: Wind River Systems was formed.
- **1987**: Wind River introduced VxWorks.
- **1995**: VxWorks launches into space on the NASA Clementine moon probe.
- **1997**: VxWorks, the real-time operating system for NASA's Mars Pathfinder mission, lands on Mars.
- **2008**: Wind River was the embedded Linux market leader (30% of total market revenue).
- **2009**: Intel acquires Wind River for approximately $884 million.
- **2010**: Wind River partners with Intel and the Linux Foundation to create the Yocto Project.
- **2012**: Wind River debuts software platform targeted at gateways and hubs for IoT.

Wind River Systems solutions

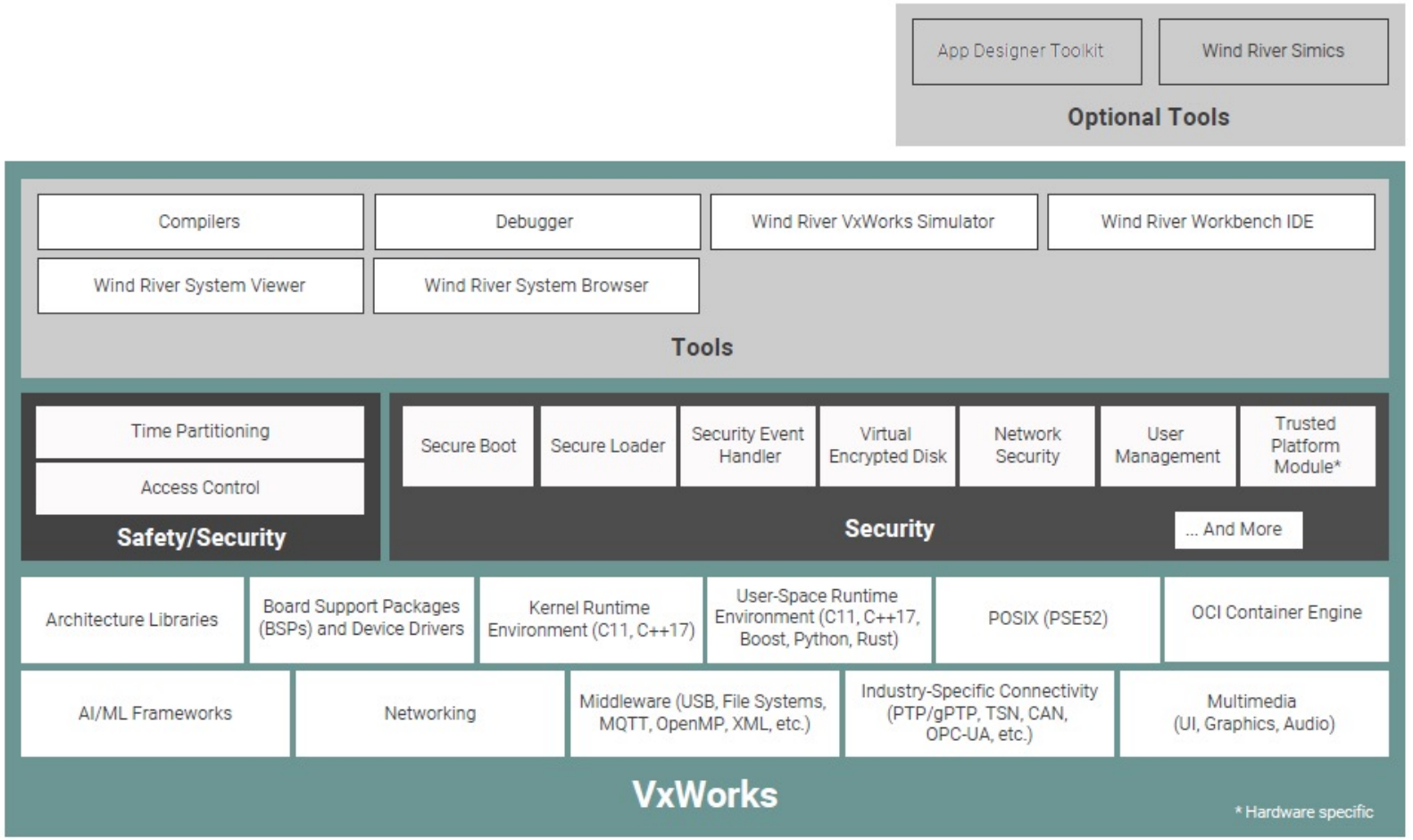
VxWorks

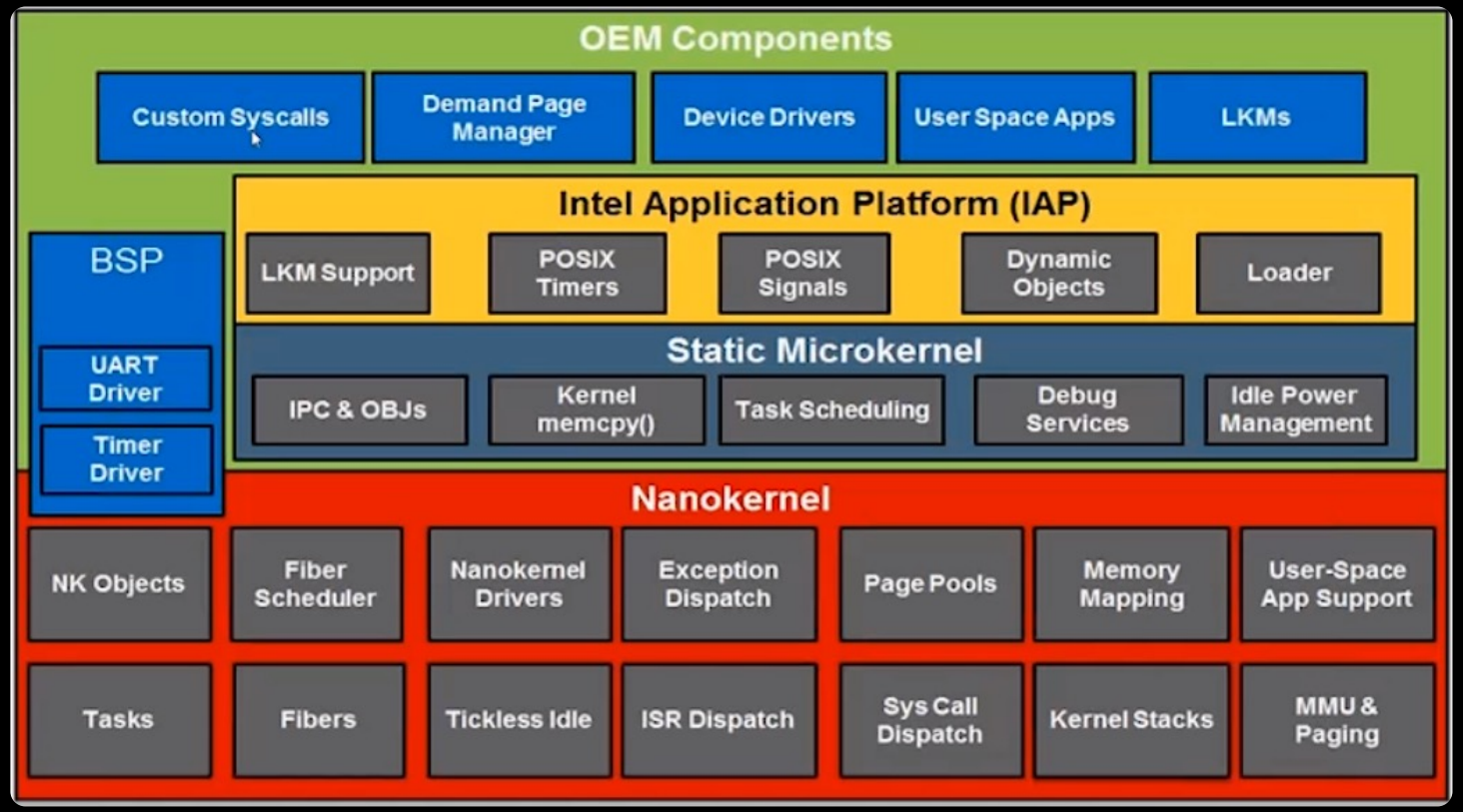
- Real-time deterministic operating system
- For embedded and critical infrastructure devices
- Suited for devices with lower computational capabilities
- VxMicro + VxWorks 7

Wind River Linux

- Source code and a Yocto-based build system that generate runtime images suitable for embedded devices
- Used to control in-car electronics (with BMW, Intel and Magneti Marelli)
- Suited for devices with higher computational capabilities
- VxMicro + Linux

VxWorks SDK components





VxMicro 2.2 Block Diagram

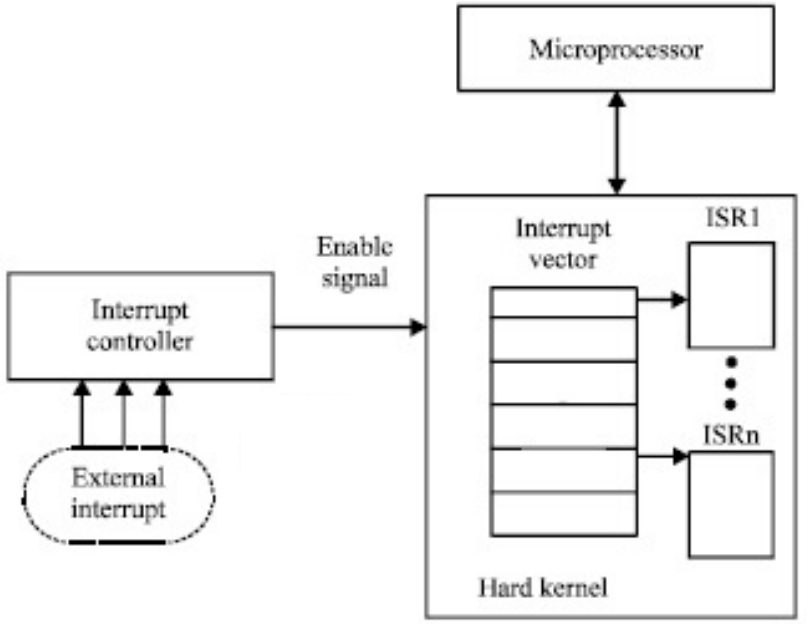
https://learning.windriver.com/building-the-vxworks-microkernel-profile

VxMicro microkernel

- Small footprint (< 50KB)
- 2 KB runtime footprint
- ***Nanokernel***:
  - Interrupts and fibers
  - Quick context switching
  - Process-dependent code
  - Semaphores, LIFO, FIFO, stack
  - Data passing between ISR, driver and applications
- ***Static Microkernel***:
  - Multi-tasking, preemption, synchronization
  - Lower priority than the nanokernel ISR and drivers
  - Memory maps, communication, channels, task groups and messaging

VxMicro Hardware Interrupt Handling

- VxWorks provides routines to handle hardware interrupts (see *sigLib*)
  - `intConnect(ISR_NUM, funcISR, arg)`: connect a routine to an interrupt vector
  - `intLock(), intUnlock()`: disable/enable interrupts
- ISR code does not run in the normal task context, then it must be short and fast:
  - No task can block an ISR
  - ISR's can't block (no malloc, no semTake, no I/O)
  - Typical read and write memory-mapped I/O registers
  - Communication allowed: shared memory, queue, pipe, signals, semaphore (only give)



Memory Management

Addressing protection

The protection issue

- In general, RTOS implement only kernel mode, for more predictability
- However, all applications give access to the whole address space
- On the other hand, virtual memory introduces impredictability
- Memory Management Unit (MMU) necessary

VxWorks solution

- Configurable alternatives:
  - Single Kernel mode (= single address space)
  - VxVMI: library allowing to mark data areas private
  - Tasks running in Kernel-mode and real-time processes (RTPs) running in User-mode

Memory Management

Addressing translation and memory allocation

Addressing mode

- Real-addressing mode: non-overlapping
- Virtual-addressing mode: obtain protection benefits

Real-addressing mode better for predictability

Allocation

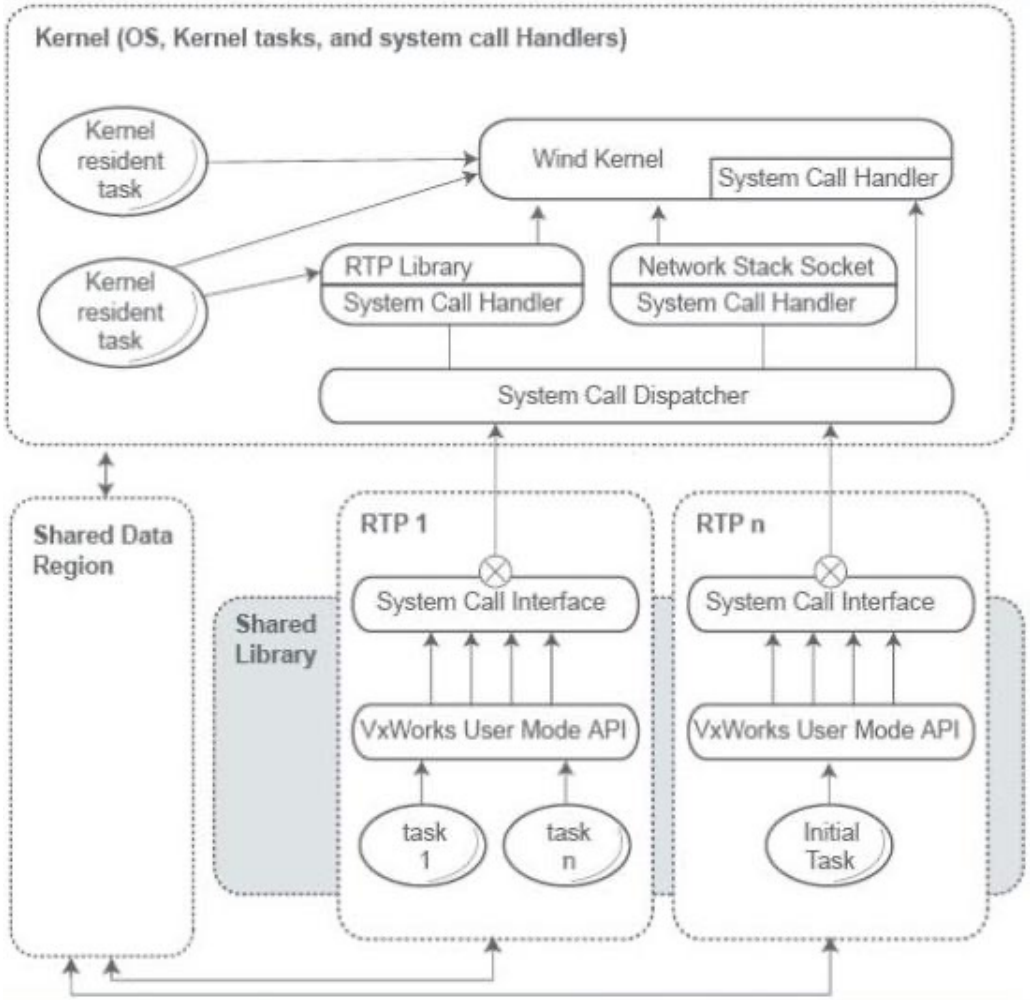
- First-fit allocation algorithm: fragmentation disadvatage but good for static allocation
- Best-fit allocation algorithm: binary tree (O(log(n))), more deterministic

VxWorks Real-Time Process Communication

Task instances created inside an RTP shares the same address space.

RTPs and kernel has to communicate with IPC:

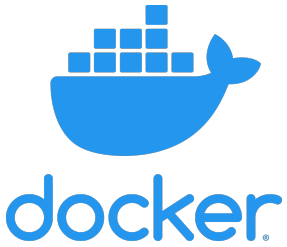
- Shared Memory: *shmLib*
  - `shm_open()`, `shm_unlink()`, `mmap()`, `munmap()`
- Queue: *msgQLib*
  - `msgQCreate()`, `msgQDelete()`, `msgQSend()`, `msgQReceive()`
- Pipe: *pipeDrv*
  - `pipeDrv()`, `pipeDevCreate()`, `open()`, `read()`, `write()`, `close()`
- Semaphore: *semLib*
  - `semGive()`, `semTake()`, `semFlush()`, `semDelete()`



Linux Containers for Operational Technology

From IT to OT containers

- Containers are used in cloud-native computing architecture and IT environments, with the aim to obtain the benefits of flexibility and modularity
- Needs to leverage the use of containers in edge applications for flexibility and scalability

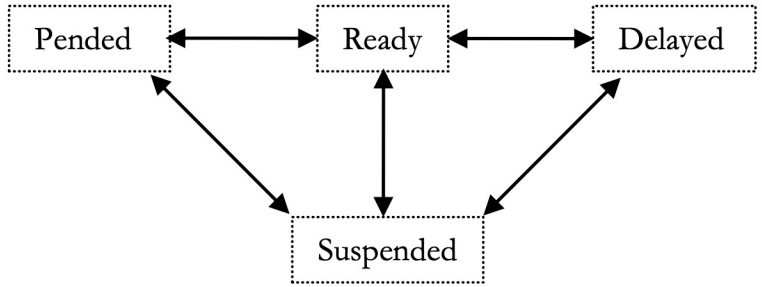


kubernetes

- VxWorks is the first to include Open Container Initiative (OCI)-compatible container technology, named OverC (lighter than Docker)
- Supports development and orchestration frameworks such as Docker and Kubernetes

Task Scheduling

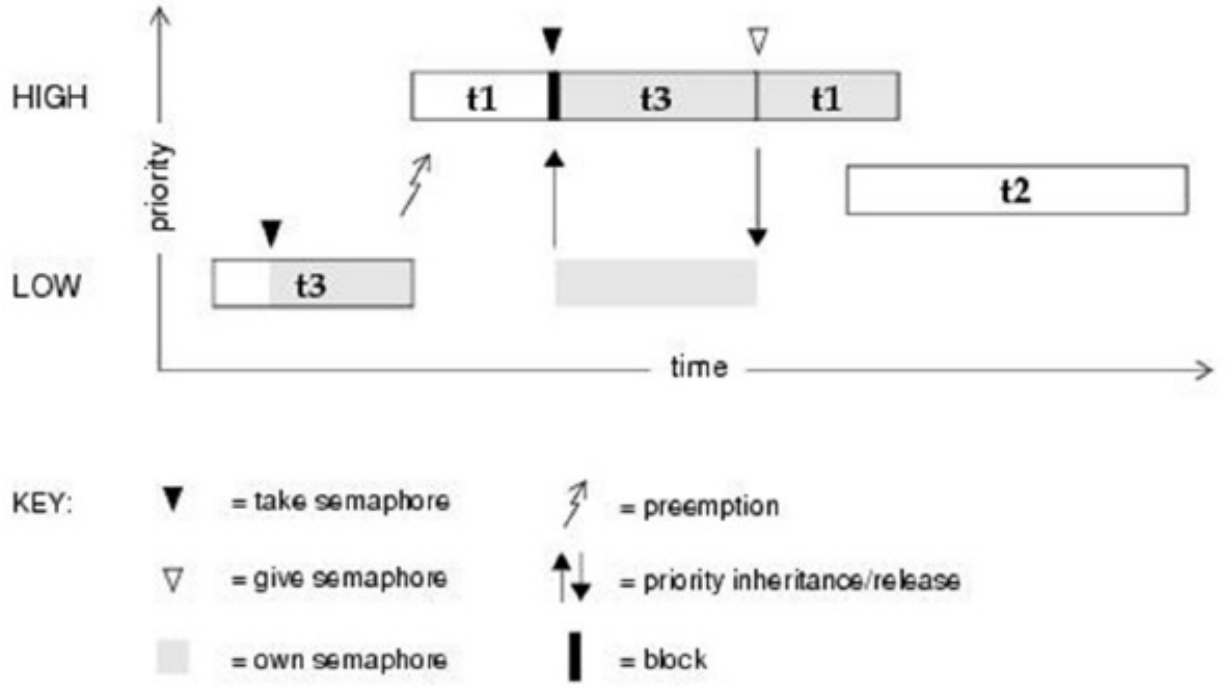
- Wind task scheduling uses a priority based preemptive scheduling algorithm (VxWorks currently support 256 priority levels).



- In case of there are two processes in ready queue with the same priority:
  - FCFS: non-preemptive first-come first-served (set as default)
  - Round Robin (RR): every task is run for the length of a user-defined time slice
  - Dynamic Priority change with EDD (discouraged because programmer is supposed to guarantee the tasks' deadline by analyzing the schedulability)
- The interrupt latency is smaller than others because of FCFS and RR does not require a frequent resorting compared to EDF.

Mutual-Exclusion semaphores and priority inheritance

- Specialized binary semaphore designed to address issues inherent in mutual exclusion
- Mutex attributes (POSIX 1003.1c) are held in a data type called **pthread_mutexattr_t**, which can be of two types:
  - PTHREAD_PRIO_INHERIT: priority inheritance
  - PTHREAD_PRIO_PROTECT: priority ceiling to avoid elevating a lower-priority thread to a too-high priority



Security enhancements (1/2)

- Arm® TrustZone OP-TEE support®:
  - *Trusted Execution Environment (TEE)*
  - *Additional layer of security where code/data running cannot be accessed or tampered*

- GE Digital® Achilles System Level II Certification:
  - *Compliant with the IEC 62443-3-3 standard's cybersecurity requirements*

- Secure Boot/Secure Loader:
  - *Images and applications of all types can be digitally signed and, optionally, encrypted.*

Security enhancements (2/2)

- Kernel page-table isolation (KPTI)
  - *Technique to mitigate the Meltdown security vulnerability*
- OpenSSL FIPS 140-2 Module
  - *Conform to the Federal Information Processing Standard (FIPS) Publication 140-2.*
- SSH client and server
  - *Enable secure management of remote servers that run embedded systems*

Low Power Design

VxWorks provides a feasible way to achieve low power dissipation with its kernel scheduler and hook mechanism that can maximize the power efficiency under embedded system and can be used as a reference for low-power design-related occasions under Vxworks.

Adaptive Low Power Method:

- Create a daemon idle task: when is scheduled in, the system can be safely put into the low power state by setting corresponding PLL (Phase Lock Loop) registers.
- Configure the VxWorks hook method that recover the system back to the normal stat when daemon idle task is scheduled out.

GuiPing Cao, KeZhu Song and JunFeng Yang, "Adaptive low power design based on Vxworks kernel scheduler and hook machanism," 2010 Second IITA International Conference on Geoscience and Remote Sensing, 2010, pp. 59-62, doi: 10.1109/IITA-GRS.2010.5603226.

Low Power Design

Implementation

```
void powerManagerInit() {
    if (powerMgrInit) return;

    // Spawn deamon task.
    powerManagerTaskId = taskSpawn(
        "PowerMgr", 255,
        …, (FUNCPTR) powerMgr, …);


    // Create task switch hook
    taskSwitchHookAdd((FUNCPTR)checkPowerState);

    powerMgrInit = true;
}
```

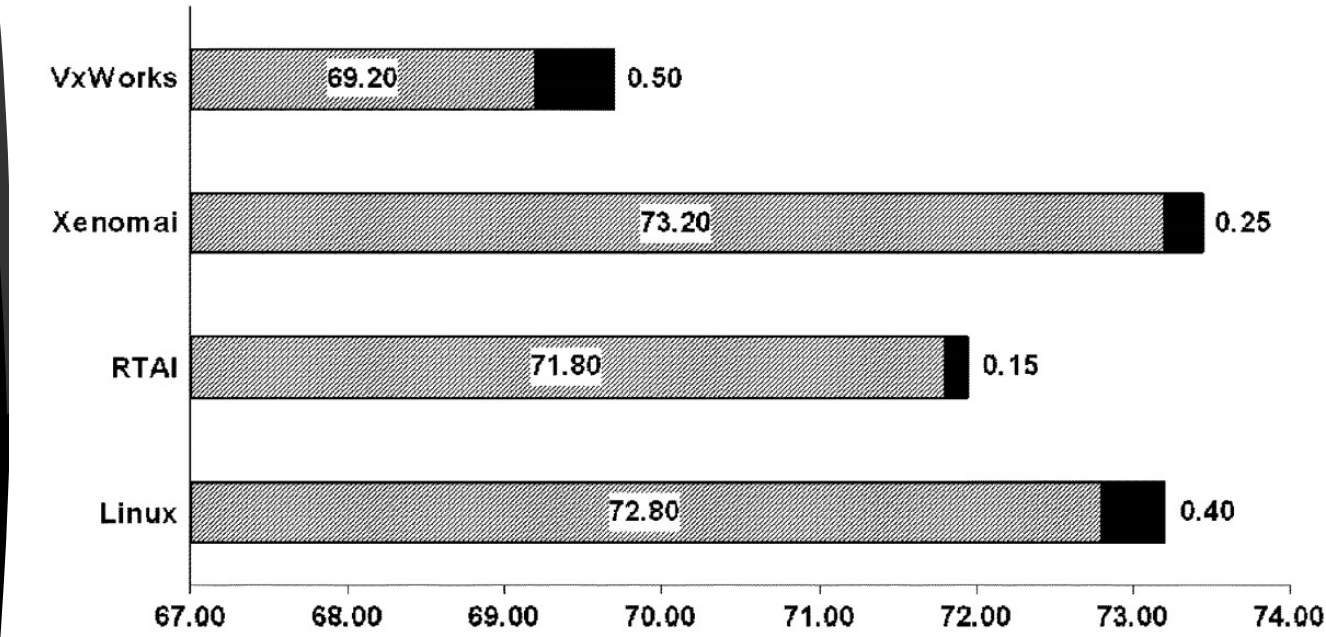
```
void powerMgr(void) {
    while(1) {
        if (isLowEnergy == false) {
            // Set PLL registers and put system in low energy
            isLowEnergy = true;
        }
    }
}
```

```
void checkPowerState(WIND_TCB *pOldTcb, WIND_TCB *pNewTcb)
{
    // Deamon task is scheduled out -> Low Energy OFF
    if(taskTcb(powerManagerTaskId) == pOldTcb) {
        // Reset PLL registers and restore normal power
        enterLowEnergy = false;
    }
}
```

Performance analysis: interrupt latency and rescheduling time

It is used a test program, *acqloop*, which acquires 64 cannels from an ADC converter and produces a single output on a board, corresponding to the first acquired signal.

The ADC generates an interrupt when a new set of digitized data is ready.
The delays are measured with an oscilloscope.



	Linux	RTAI	Xenomai	VxWorks
Jitter (µs)	0.40	0.15	0.25	0.50
Delay (µs)	72.80	71.80	73.20	69.20

A. Barbalace, A. Luchetta, G. Manduchi, M. Moro, A. Soppelsa and C. Taliercio, "*Performance Comparison of VxWorks, Linux, RTAI, and Xenomai in a Hard Real-Time Application*," in IEEE Transactions on Nuclear Science, vol. 55, no. 1, pp. 435-439, Feb. 2008, doi: 10.1109/TNS.2007.905231.

Analysis and Testing of Key Performance Indexes

The standard deviation of each performance index shows that the fluctuations of test results are small. Comparing the maximum value with the average value, we can see that real-time operating system performance index will be larger than the average value in some cases.

Overall, the test results show that VxWorks has strong real-time and meets the performance requirements.

Key Performance Indicators	Average Value (μs)	Standard Deviation (us)	Maximum Value (μs)
Task Switching Time	6.9	0.04	6.97
Preemption Time	8.7	0.18	9.11
Interrupt Latency Time	65.5	0.55	66.69
Message Communication Time	58.2	0.91	60.86
Semaphore Shuffling Time	7.1	0.25	7.59

M. Li, Z. Shang, Q. Hu, G. Yang, Y. Li and F. Sun, "Analysis and Testing of Key Performance Indexes of Vxworks in Real-Time System," *2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*

Development Tools

- **VxWorks SDK**: APIs for in-kernel and user-level real-time-process (RTP)
- **Wind River Workbench**: complete suite to develop, debug, analyse and tune the entire system
- **Diab Compiler**: tiny footprint, high performance, large processors and architectures support (ARM, MIPS, Intel…)
- **Virtual Simulator**: QEMU with GDB, Simics
- **Wind River Studio**: *cloud-native platform for the development, deployment, operations, and servicing*

Implement a Simple Kernel Module

```
int taskSpawn(
        char * name,        // name of new task
        int priority,       // priority of new task
        int options,        // task option word
        int stackSize,      // size (bytes) of stack
        FUNCPTR entryPt,    // entry point of new task
        int arg1,
        int arg2,
        int arg3,
        int arg4,
        int arg5,
        int arg6,
        int arg7,
        int arg8,
        int arg9,
        int arg10
)
```

```c
#include <taskLib.h>
#include <stdio.h>
#include <kernelLib.h>

int task_periods[] = {18, 25, 30};

void task(int task_idx)
{
    printf("Task %i has been started\n",task_idx);
    while (1)
    {
        printf("task %d: running\n", task_idx);

        // Task period operations.

        printf("task %d: stopped\n", task_idx);
        taskDelay(task_stop[task_idx]);
    }
}

void main(void)
{
    id1=taskSpawn("Task0", 210, 0, 4096, (FUNCPTR) task, 0, ...);
    id2=taskSpawn("Task1", 210, 0, 4096, (FUNCPTR) task, 1, ...);
    id3=taskSpawn("Task2", 210, 0, 4096, (FUNCPTR) task, 2, ...);
}
```

VxWorks and QNX Neutrino

both commercial RTOS for Embedded Systems, but with several differences

- Both proprietary RTOS
- Widely used in industrial and academic environments
- In business from 1980
- Both MicroKernel

QNX:

- Message passing architecture
- Designed for embedded platforms
- Limited set of primitives and software dependency
- Preemptive priority-based and Round-Robin scheduling

VxWorks:

- Shared Memory architecure
- Designed for distributed systems
- All the kernel and user services exist in the same address space
- FIFO, Round-Robin, and Sporadic Scheduling