Lecture 9
Real-Time Application Interface for Linux

Asst. Prof. Tolga Ayav, Ph.D.

Department of Computer Engineering
İzmir Institute of Technology
RTOS

- Operating system (OS) is a program that acts as an intermediary between a user and the computer hardware. The primary goal of an OS is to make the computer system convenient to use and the secondary goal is to use the hardware in efficient manner.
Requirements of a RTOS

- Interrupt handling, guaranteed interrupt response
- Process management (Support for scheduling of real-time processes and preemptive scheduling)
- Interprocess communication and synchronization.
- Time management.
- Memory management
- I/O support (Support for communication with peripheral devices via drivers)
- High speed data acquisition
- Resource management (User control of system resources)
- Error and exception handling
Operating systems must provide the following specific functions:

- CPU Management
- Task management
- I/O management
- Memory management
- Shared resource management
# Main RTOS functions and their occurrence ratios

<table>
<thead>
<tr>
<th>Function</th>
<th>Occurrence (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task scheduler</td>
<td>47</td>
</tr>
<tr>
<td>I/O management</td>
<td>16</td>
</tr>
<tr>
<td>Task management</td>
<td>15</td>
</tr>
<tr>
<td>Timer control</td>
<td>6</td>
</tr>
<tr>
<td>Memory management</td>
<td>4</td>
</tr>
<tr>
<td>Shared resource management</td>
<td>12</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>100</strong></td>
</tr>
</tbody>
</table>
### Main RTOS products and their vendors

<table>
<thead>
<tr>
<th>OS name</th>
<th>Vendor</th>
<th>OS name</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chorus</td>
<td>Chorus</td>
<td>VxWorks</td>
<td>WindRiver</td>
</tr>
<tr>
<td>ECOS</td>
<td>Cygnus</td>
<td>Windows CE</td>
<td>Microsoft</td>
</tr>
<tr>
<td>EPOC32</td>
<td>Symbian</td>
<td>MicroC/OS</td>
<td>White Horse Design</td>
</tr>
<tr>
<td>OSE</td>
<td>Enea</td>
<td>AMX</td>
<td>Kadak</td>
</tr>
<tr>
<td>JavaOS</td>
<td>Sun Microsystems</td>
<td>Ariel</td>
<td>Micoware</td>
</tr>
<tr>
<td>LynxOS</td>
<td>Lynx</td>
<td>ARTOS</td>
<td>Locamation</td>
</tr>
<tr>
<td>NucleusPlus</td>
<td>Acc. Tech</td>
<td>ASP6x</td>
<td>DAN enterprise</td>
</tr>
<tr>
<td>OS/9</td>
<td>Microware</td>
<td>Chimera</td>
<td>Robotics</td>
</tr>
<tr>
<td>PalmOS</td>
<td>3com</td>
<td>CMX</td>
<td>CMX company</td>
</tr>
<tr>
<td>PSOS+</td>
<td>ISI</td>
<td>Diamond</td>
<td>3L</td>
</tr>
<tr>
<td>QNX</td>
<td>QNX</td>
<td>DOS6-XL</td>
<td>General Software</td>
</tr>
<tr>
<td>RTEMS</td>
<td>OAR corp.</td>
<td>EOS</td>
<td>Etnoteam</td>
</tr>
<tr>
<td>VRTX</td>
<td>Microtec</td>
<td>EUROS</td>
<td>Kanet</td>
</tr>
<tr>
<td>RT-Linux</td>
<td>…</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The standard LINUX kernel isn't a real-time kernel.
Some patches should be applied to fulfill the real-time requirements.
Linux Kernel Core

Linux, offers to the applications at least:

- HW management layer dealing with event polling or Processor/peripheral Interrupts
- Scheduler classes dealing with process activation, priorities, time slice, soft real-time
- Communications means among Applications (at least FIFO).
- Hardware abstraction layer
- real-time interrupt dispatcher, real-time scheduler
- Inter processes communication services
- The patches available are: RTLinux, xenomai and RTAI
RT-Linux

- RT-Linux is an operating system, in which a small real-time kernel co-exists with standard Linux kernel
  - The real-time kernel sits between *standard Linux kernel* and the *h/w*.
  - The standard Linux kernel sees this real-time layer as actual h/w
  - The real-time kernel *intercepts all hardware interrupts*.
    - Only for those RTLinux-related interrupts, the appropriate ISR is run.
    - All other interrupts are held and passed to the standard Linux kernel as software interrupts when the standard Linux kernel runs.
  - The real-time kernel assigns the *lowest priority* to the *standard Linux kernel*. Thus the realtime tasks will be executed in real-time
  - user can create realtime tasks and achieve correct timing for them by deciding on scheduling algorithms, priorities, execution freq, etc.
  - Realtime tasks are *privileged* (that is, they have direct access to hardware), and they do *NOT use virtual memory*. 
RT-Linux
RT-Linux
Real-Time Application Interface

*It is a module in dormant state ready to overtake Linux*

- Not a RTOS.
- Makes Linux kernel fully pre-emptable.
- Adds the features of RTOS to Linux.
- *interrupt dispatcher*: traps the peripherals interrupts and if necessary re-routes them to Linux.
- *Hardware abstraction layer (HAL)*: Gets information from Linux and traps fundamental functions. Provides few dependencies to Linux Kernel.
- Minimizes intrusion on the standard Linux kernel
- Localizes interrupt handling and emulation code
- Linux is a background task for RTAI
Real-Time Application Interface

Offers some services related to:

- HW management layer dealing with peripherals.
- Scheduler classes dealing with tasks, priorities, hard real-time.
- Communications means among tasks & processes (at least FIFO).
The software architecture of RTAI is made of:

- 1 I/F to Linux HW Management (HAL): basically a data structure.
- 3 basic components (dispatcher, scheduler, fifo's).
- 1 I/F (set of functions) used in user tasks to initialize and start the components.

From a Linux point of view these entities populate modules.
Control flow in a RTAI/Linux system

Interrupt Occurrence

RTAI Dispatcher

RT Int Handler

Linux Dispatcher

SRQ Dispatcher

Linux Intr Ret

Interrupt Return

Return to Program Execution
Virtual Interrupt Control

- Cli () and Sti () RTAI functions set flags recording incoming and ignored interrupts
- Sti () records incoming interrupts
- Cli () records ignored interrupts
- RTAI registers all interrupts and signals them at an appropriate time
Real-Time Application Interface

- HAL supports **five** core loadable modules
  - Rtai -> provides basic framework
  - Rtai_sched -> provides periodic or one-shot scheduling
  - Rtai_mups -> provides simultaneous one-shot and periodic schedulers
  - Rtai_shm -> allows memory sharing (both inter-linux and intra-linux)
  - Rtai_fifos -> adaptation of the RTLinux FIFO's
Scheduler

- RT-Linux contains a dynamic scheduler

- RT-Linux has many kinds of schedulers
  - The EDF (Earliest Deadline First) scheduler
  - Rate-monotonic scheduler

- Real-time FIFOs
  - RT-FIFOs are used to pass information between real-time process and ordinary Linux process.
  - RT-FIFOs are designed to never block the real-time tasks.
  - RT-FIFOs are, like realtime tasks, never page out. This eliminates the problem of unpredictable delay due to paging.
**Time Resolution**

- If the kernel was patched with UTIME, we could schedule processes with microsecond resolution.
- Running rtlinx-V3.0 Kernel 2.2.19 on the 486 allows stable hard real-time operation. Giving:
  - 15 microseconds worst case jitter.
    - 10 microseconds event resolution.
    - 17 nanoseconds timer resolution.
    - 6 microseconds interrupt response time. (This value was measured on interrupts on the parallel port)
- High resolution timing functions give nanosecond resolution (limited by the hardware only)
Linux v.s. RTLinux

- **Linux Non-real-time Features**
  - Linux scheduling algorithms are not designed for real-time tasks
    - Provide good *average* performance or throughput
  - Unpredictable delay
    - Uninterruptible system calls, the use of interrupt disabling virtual memory support (context switch may take hundreds of microsecond).
    - Linux Timer resolution is coarse, 10ms
  - Linux Kernel is Non-preemptible.

- **RTLinux Real-time Features**
  - Support real-time scheduling
  - Predictable delay (by its small size and limited operations)
  - Finer time resolution
  - Preemptible kernel
  - No virtual memory support