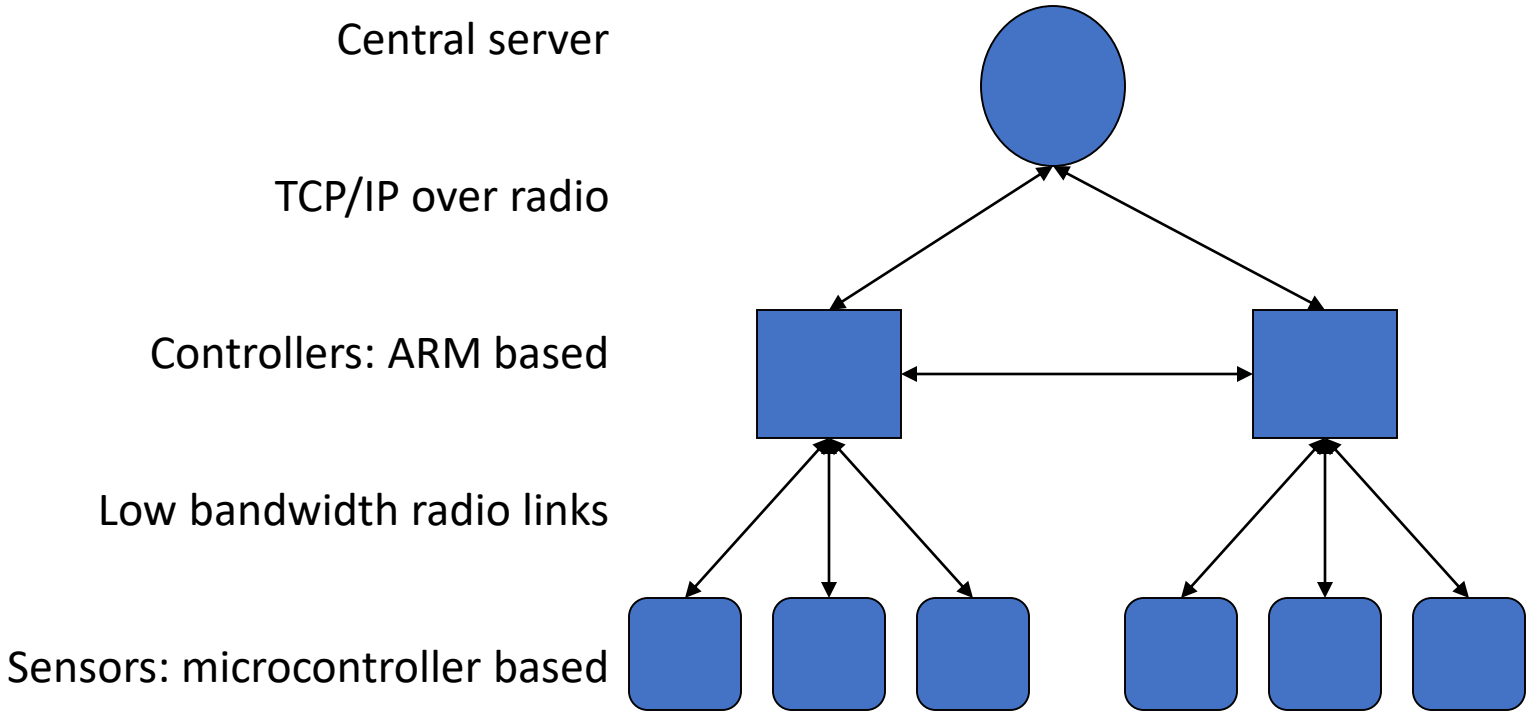
Real Time Operating Systems (RTOS)

EE5182 Microcontrollers and Embedded Systems

Based on content by Subhashis Banerjee, Indian Institute of Technology, Delhi & Akos Ledeczi, Vanderbilt University

Fire alarm system: An example



Based on content by Subhashis Banerjee, Indian Institute of Technology, Delhi & Akos Ledeczi, Vanderbilt University

Fire alarm system: An example

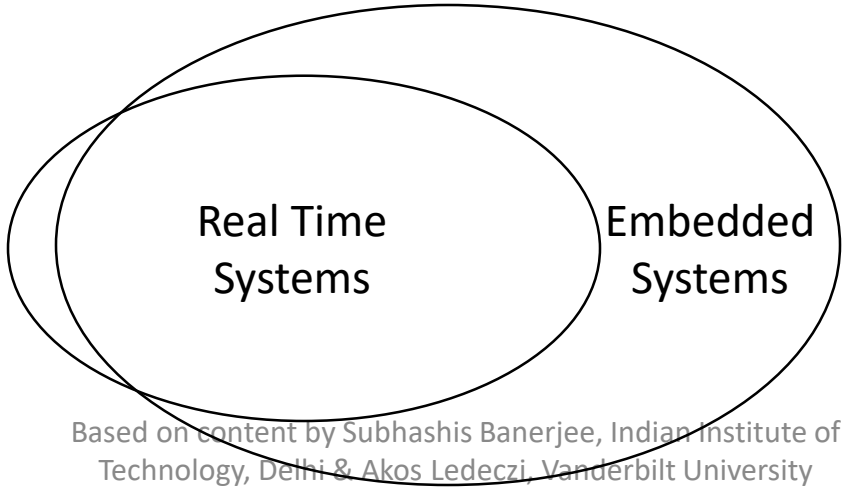
- Problem
  - Hundreds of sensors, each fitted with Low Range Wireless
    - Sensor information to be logged in a server & appropriate action initiated
- Possible Solution
  - Collaborative Action
    - Routing
      - Dynamic – Sensors/controllers may go down
      - Auto Configurable – No/easy human intervention.
      - Less Collision/Link Clogging
      - Less number of intermediate nodes – Fast Response Time
      - Secure

Real Time Systems

- A system is said to be Real Time if it is required to complete it's work and deliver it's services on time.
- Example – Flight Control System
  - All tasks in that system must execute on time.
- Non Example – PC system

Embedded vs. Real Time Systems

- Embedded system:
  - A computer system that performs a limited set of specific functions.
  - It often interacts with its environment.
- Real Time Systems:
  - Correctness of the system depends not only on the logical results, but also on the time in which the results are produced.



Examples?

Based on content by Subhashis Banerjee, Indian Institute of Technology, Delhi & Akos Ledeczi, Vanderbilt University

Examples

- Real Time Embedded:
  - Nuclear reactor control; Flight control; Basically any safety critical system
  - GPS; Mobile phone
- Real Time, but not Embedded:
  - Stock trading system
  - Skype
- Embedded, but not Real Time:
  - Home temperature control
  - Sprinkler system
  - Washing machine, refrigerator, etc.
  - Blood pressure meter

Characteristics of Real Time Systems

- Event-driven, reactive
- High cost of failure
- Concurrency/multiprogramming
- Stand-alone/continuous operation
- Reliability/fault-tolerance requirements
- Predictable behavior

What's Important in Real Time Systems

- Metrics for real-time systems differ from that for time-sharing systems.

	Time-Sharing Systems	Real-Time Systems
Capacity	High throughput	Schedulability
Responsiveness	Fast average response	Ensured worst-case response
Overload	Fairness	Stability

- schedulability is the ability of tasks to meet all hard deadlines
- latency is the worst-case system response time to events
- stability in overload means the system meets critical deadlines even if all deadlines cannot be met

Hard and Soft Real Time Systems (Qualitative Definition)

- Hard Real Time System
  - Failure to meet deadlines is fatal
  - example : Flight Control System
- Soft Real Time System
  - Late completion of jobs is undesirable but not fatal.
  - System performance degrades as more & more jobs miss deadlines
  - Online Databases

Hard and Soft Real Time Systems (Operational Definition)

- Hard Real Time System
  - Validation by provably correct procedures or extensive simulation that the system always meets the timings constraints
- Soft Real Time System
  - Demonstration of jobs meeting some statistical constraints suffices.
- Example – Multimedia System
  - 25 frames per second on an average

Role of an OS in Real Time Systems

- Standalone Applications
  - Often no OS involved
  - Micro controller based Embedded Systems
- Some Real Time Applications are huge & complex
  - Multiple threads
  - Complicated Synchronization Requirements
  - Filesystem / Network / Windowing support
  - OS primitives reduce the software design time
- Real Time Operating System (RTOS)

Features of RTOS

- Scheduling
- Resource Allocation
- Interrupt Handling
- Other issues like kernel size

Scheduling in RTOS

- More information about the tasks are known
  - No of tasks
  - Resource Requirements
  - Release Time
  - Execution time
  - Deadlines
- Being a more deterministic system better scheduling algorithms can be devised.

Scheduling Algorithms in RTOS

- Clock Driven Scheduling
- Weighted Round Robin Scheduling
- Priority Scheduling
  - (Greedy / List / Event Driven)

Scheduling Algorithms in RTOS (contd)

- Clock Driven
  - All parameters about jobs (release time/ execution time/deadline) known in advance.
  - Schedule can be computed offline or at some regular time instances.
  - Minimal runtime overhead.
  - Not suitable for many applications.

Scheduling Algorithms in RTOS (contd)

- Weighted Round Robin
  - Jobs scheduled in FIFO manner
  - Time quantum given to jobs is proportional to it’s weight
  - Example use : High speed switching network
    - QOS guarantee.
  - Not suitable for precedence constrained jobs.
    - Job A can run only after Job B. No point in giving time quantum to Job B before Job A.

Scheduling Algorithms in RTOS (contd)

- Priority Scheduling
  - (Greedy/List/Event Driven)
  - Processor never left idle when there are ready tasks
  - Processor allocated to processes according to priorities
  - Priorities
    - static - at design time
    - Dynamic - at runtime

Priority Scheduling

- Earliest Deadline First (EDF)
  - Process with earliest deadline given highest priority
- Least Slack Time First (LSF)
  - slack = relative deadline – execution left
- Rate Monotonic Scheduling (RMS)
  - For periodic tasks
  - Tasks priority inversely proportional to it’s period

Resource Allocation in RTOS

- Resource Allocation
  - The issues with scheduling applicable here.
  - Resources can be allocated in
    - Weighted Round Robin
    - Priority Based
- Some resources are non preemptible
  - Example : semaphores
- Priority Inversion if priority scheduling is used

Solutions to Priority Inversion

- Non Blocking Critical Section
  - Higher priority Thread may get blocked by unrelated low priority thread
- Priority Ceiling
  - Each resource has an assigned priority
  - Priority of thread is the highest of all priorities of the resources it's holding
- Priority Inheritance
  - The thread holding a resource inherits the priority of the thread blocked on that resource

Other RTOS issues

- Interrupt Latency should be very small
  - Kernel has to respond to real time events
  - Interrupts should be disabled for minimum possible time
- For embedded applications Kernel Size should be small
  - Should fit in ROM
- Sophisticated features can be removed
  - No Virtual Memory
  - No Protection

Linux for Real Time Applications

- Scheduling
  - Priority Driven Approach
    - Optimize average case response time
  - Interactive Processes Given Highest Priority
    - Aim to reduce response times of processes
  - Real Time Processes
    - Processes with high priority
    - No notion of deadlines
- Resource Allocation
  - No support for handling priority inversion

Interrupt Handling in Linux

- Interrupts are disabled in ISR/critical sections of the kernel
- No worst case bound on interrupt latency avaliable
  - eg: Disk Drivers may disable interrupt for few hundred milliseconds
- Not suitable for Real Time Applications
  - Interrupts may be missed

Other Problems with Linux

- Processes are non preemtible in Kernel Mode
  - System calls like fork take a lot of time
  - High priority thread might wait for a low priority thread to complete it's system call
- Processes are heavy weight
  - Context switch takes several hundred microseconds

Why Linux

- Coexistence of Real Time Applications with non Real Time Ones
  - Example http server
- Device Driver Base
- Stability

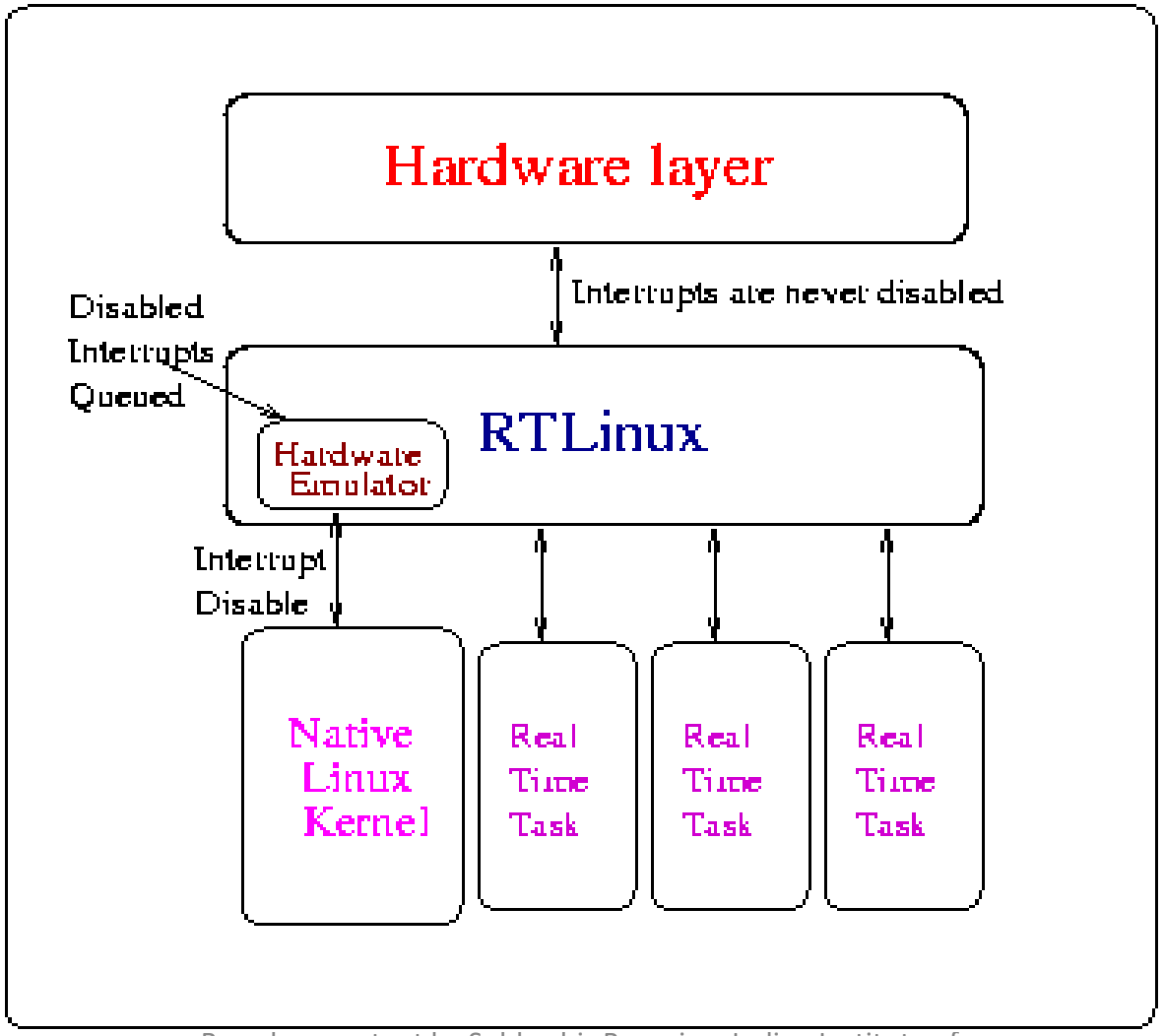
RTLinux

- Real Time Kernel at the lowest level
- Linux Kernel is a low priority thread
  - Executed only when no real time tasks
- Interrupts trapped by the Real Time Kernel and passed onto Linux Kernel
  - Software emulation to hardware interrupts
    - Interrupts are queued by RTLinux
    - Software emulation to disable_interrupt()

RTLinux (contd)

- Real Time Tasks
  - Statically allocate memory
  - No address space protection
- Non Real Time Tasks are developed in Linux
- Communication
  - Queues
  - Shared memory

RTLinux Framework



Based on content by Subhashis Banerjee, Indian Institute of Technology, Delhi & Akos Ledeczi, Vanderbilt University

Other RTOS's

- LynxOS
  - Microkernel Architecture
    - Kernel provides scheduling/interrupt handling
  - Additional features through Kernel Plug Ins(KPIs)
    - TCP/IP stack , Filesystem
    - KPI's are multithreaded
  - Memory Protection/ Demand Paging Optional
  - Development and Deployment on the same host
    - OS support for compilers/debuggers

Other RTOS's (contd)

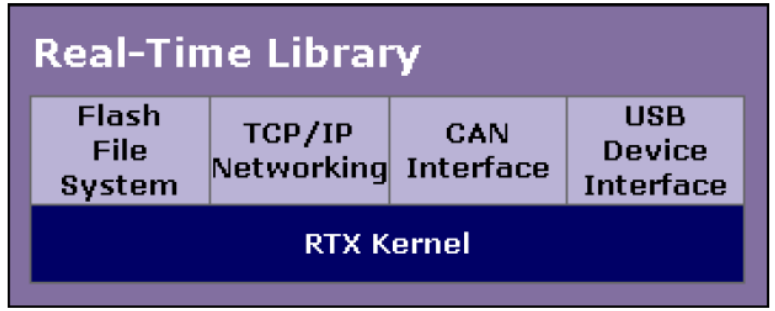
- VxWorks
  - Monolithic Architecture
  - Real Time Posix compliant
  - Cross development System
- pSOS
  - Object Oriented OS

RTX

- Royalty-free, deterministic, open source RTOS
- High-Speed real-time operation with low interrupt latency
- Flexible Scheduling: round-robin, pre-emptive, and collaborative
- Small footprint for resource constrained systems
- Compatible with ARM cores (from ARM7, ARM9 to Cortex-M processors) and software tools (Keil MDK-ARM)
- Support for multithreading and thread-safe operation
- Kernel aware debug support in Keil MDK-ARM
- Dialog-based setup using µVision Configuration Wizard

RTX Structure

- Keil Real-Time Library (RTL)
  - RTX Kernel
  - Flash file system
  - Networking
  - CAN interface
  - USB device interface



- RTX Kernel
  - Scheduler is the core of the RTX kernel
  - Supports for mutex, memory pool, mailbox, timing functions, events and semaphores