REAL-TIME OPERATING SYSTEMS

CONTENTS

General Purpose Operating System

- An interface between users and hardware
- Controlling and allocating memory
- Controlling input and output devices
- Managing file systems
- Facilitating networking

Background: What's a "Real Time" System?

- When correctness of results depend on content and time
- Hard or Soft: indicates how forgiving the system is

What makes an OS Real-Time?

- Predictable (possibly deterministic behavior), that’s all

- By product: mediocre throughputs

How do they work?

- Tasks are scheduled by OS according to fixed priority (typically)
- Tasks can’t directly interact; they use messages or shared memory & semaphores to communicate

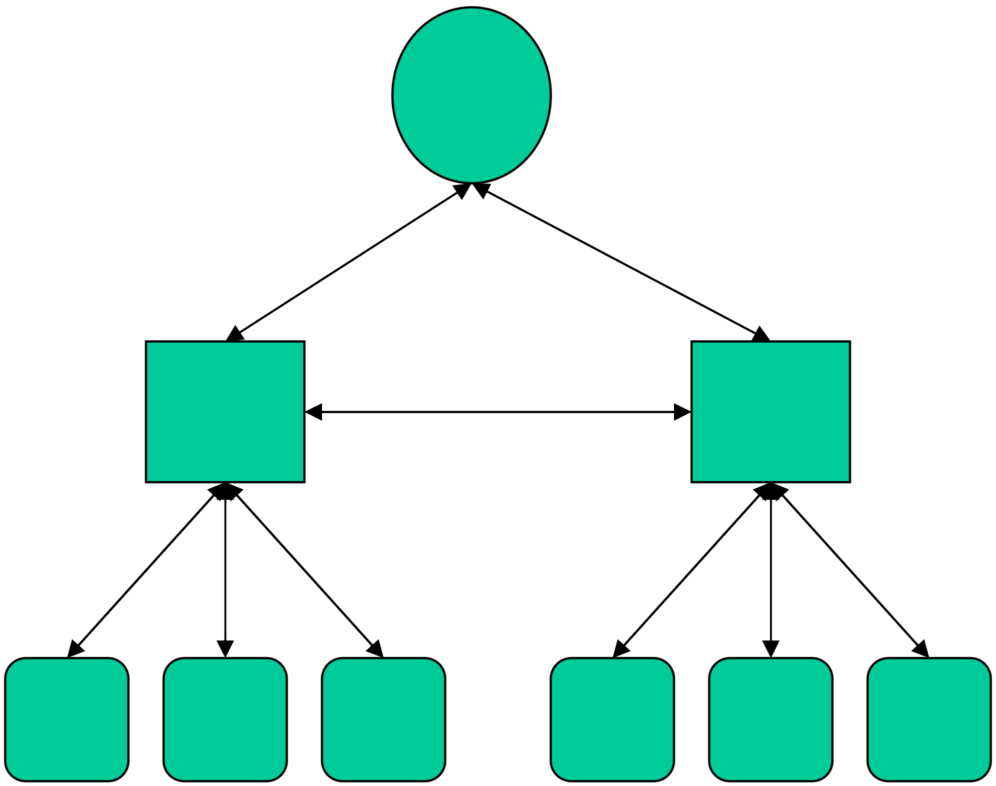
Real Time System

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

Non-Real-Time systems

- Non-Real-Time systems are the operating systems most often used
- No guaranteed worst case scheduling jitter
- System load may result in delayed interrupt response
- System response is strongly load dependent
- System timing is a unmanaged resource

Fire alarm system: an example



Central server

TCP/IP over radio

Controllers: ARM based

Low bandwidth radio links

Sensors: microcontroller based

Fire Alarm System

- 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 no of intermediate nodes
        - Fast Response Time
      - Secure

What is a RTOS ??

- RTOS is a pre-emptive multitasking operating system intended for real-time applications
- Predictable OS timing behavior
- Able to determine task's completion time
- A system of priority inheritance has to exist
- Guarantees task completion at a set **deadline**.

Designing an RTOS: End Goal

- Have known switching & scheduling overhead
- Avoid common problems like priority inversion and deadlock

Designing an RTOS: Common Problems

- Priority inversion
  - High-level task stalled due to low-level using shared resources, then a medium-level task holding up the low-level one
  - Solution: Priority inheritance – give low-level task high-level priority

Designing an RTOS: Common Problems (con't)

- Deadlock
  - Two semaphores locked out of order by two tasks and circularly block each other
  - Solution: "Instant Inheritance" implementation of Priority Ceiling Protocol – semaphores possibly needed by higher processes become priority tokens

Designing with an RTOS: What do you need?

- Task information
  - Priorities for each task
  - Worst-case runtime
  - Best-case period

Solving Problems

- Logs
- Time Machines
- Memory Conservation

Types of RTOS

- Soft Real-Time system
- Hard Real-Time system

Soft Real-Time system

- The soft real-time definition allows for frequently missed deadlines
- If the system fails to meet the deadline, possibly more than once ,the system is not considered to have failed
- Example : Multimedia streaming , Video games

Hard Real-Time system

- A hard real-time system guarantees that real-time tasks be completed within their required deadlines
- Failure to meet a single deadline may lead to a critical system failure
- Examples: air traffic control , vehicle subsystems control, medical systems

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

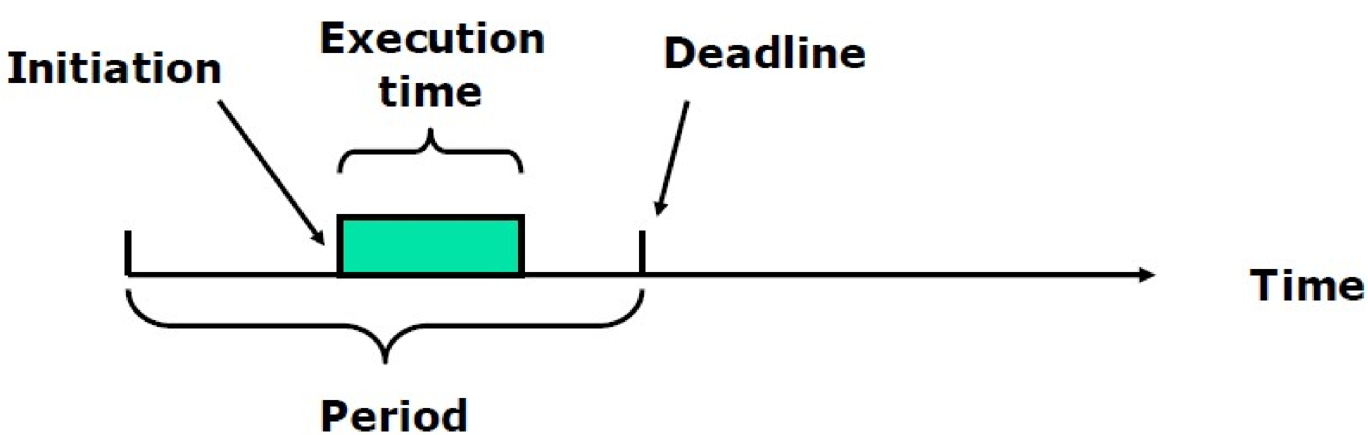
- Standalone Applications
  - Often no OS involved
  - Micro controller based Embedded Systems
- Some Real Time Applications are huge & complex
  - Multiple threads
  - Complicated Synchronization Requirements
  - Filesystem / Network / Windowing support
  - OS primitives reduce the software design time

Basic functions of RTOS kernel

- Task Management
- Interrupt handling
- Memory management
- Exception handling
- Task synchronization
- Task scheduling
- Time management
- Scheduling
- Resource Allocation

Task Management

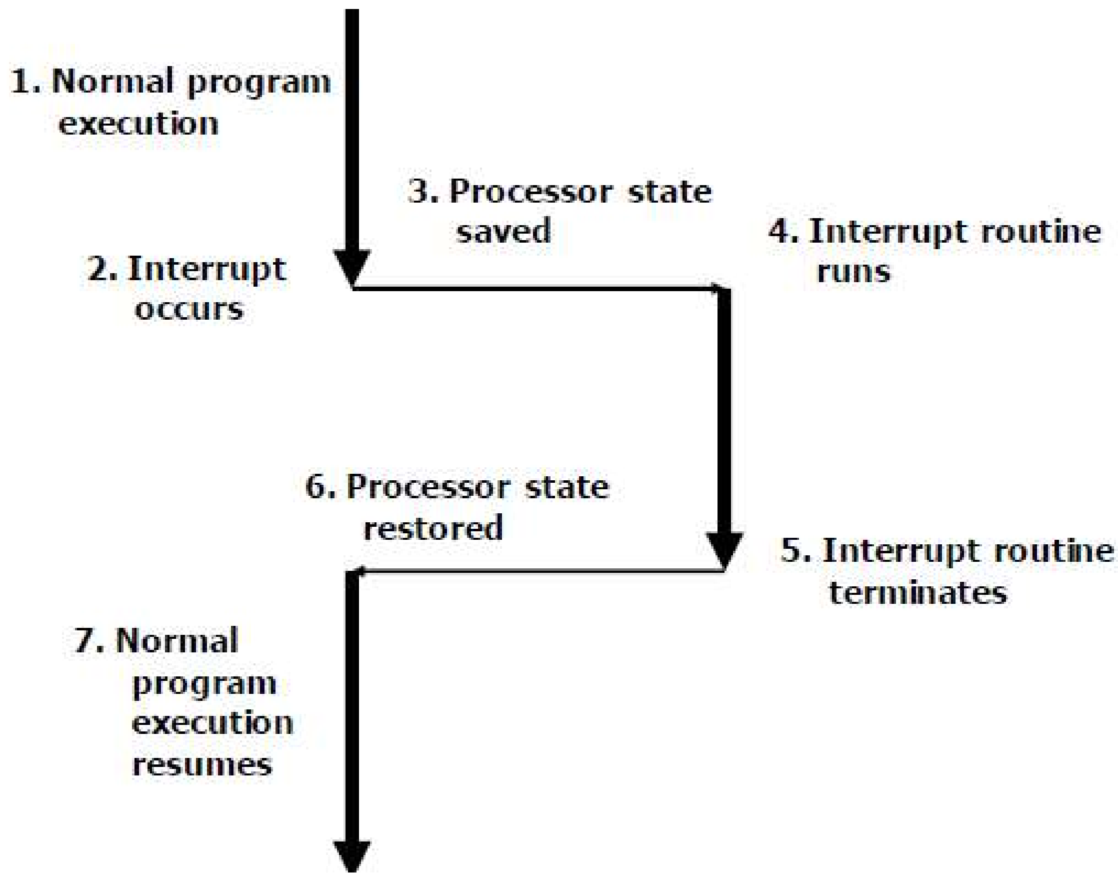
- Tasks are implemented as threads in RTOS
- Have timing constraints for tasks
- Each task a triplet: (execution time, period, deadline)
- Can be initiated any time during the period



Task States

- **Idle** : task has no need for computer time
- **Ready** : task is ready to go active, but waiting for processor time
- **Running** : task is executing associated activities
- **Waiting** : task put on temporary hold to allow lower priority task chance to execute
- **suspended**: task is waiting for resource

Interrupt handling



Interrupt handling

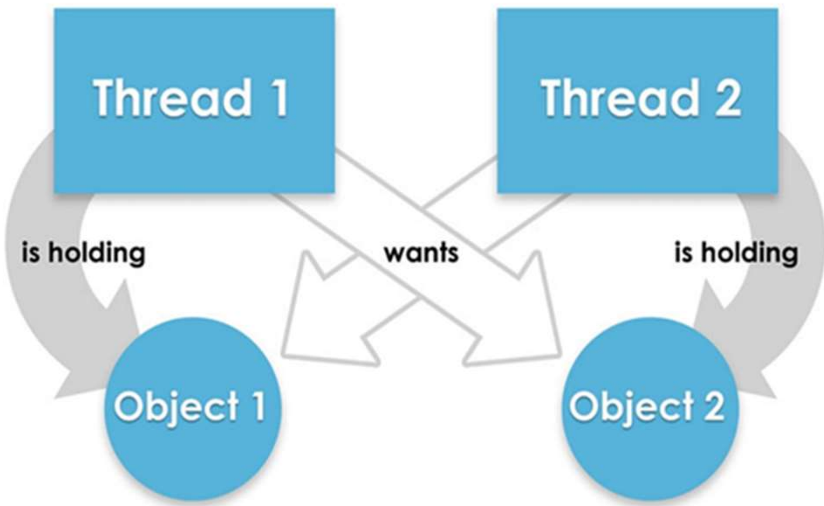
- Types of interrupts
  - **Asynchronous or hardware interrupt**
  - **Synchronous or software interrupt**
- Very low Interrupt latency
- The Interrupt Service Routine (ISR) of a lower-priority interrupt may be blocked by the ISR of a high-priority

Memory management

- RTOS may disable the support to the dynamic block allocation
- When a task is created the RTOS simply returns an already initialized memory location
- when a task dies, the RTOS returns the memory location to the pool
- No virtual memory for hard RT tasks

Exception handling

- Exceptions are triggered by the CPU in case of an error
- E.g. : Missing deadline, running out of memory, timeouts, deadlocks, divide by zero, etc.
  - Error at system level, e.g. deadlock
  - Error at task level, e.g. timeout



Exception handling

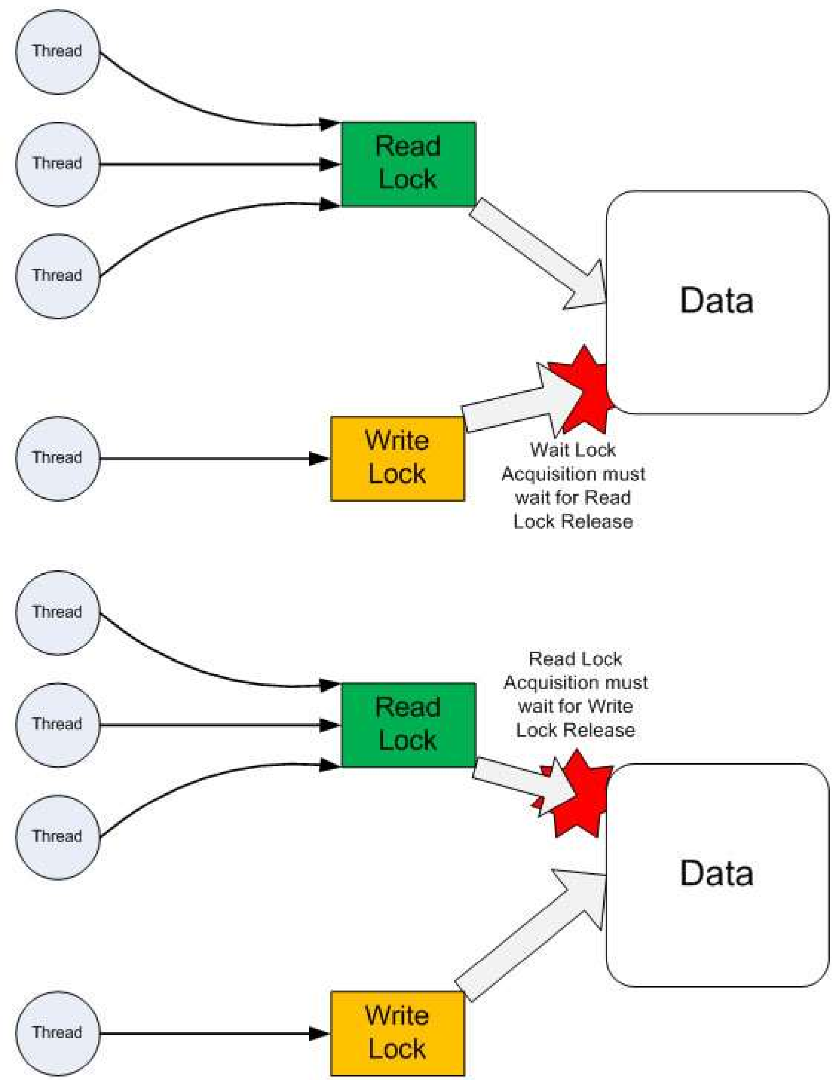
- Standard techniques:
  - System calls with error code
  - Watch dog
  - Fault-tolerance
  - Missing one possible case may result in disaster

Task Synchronization

- Semaphore
- Mutex
- Event flag
- Read/write locks

Task Synchronization

- Semaphore
- Mutex
- Event flag
- Read/write locks



Task scheduling

- Scheduler is responsible for time-sharing of CPU among tasks
  - Priority-based Preemptive Scheduling
  - Rate Monotonic Scheduling
  - Earliest Deadline First Scheduling
  - Round robin scheduling

Task scheduling

- **Priority-based Preemptive Scheduling**
  - Assign each process a priority
  - At any time, scheduler runs highest priority process ready to run

Task scheduling

- **Rate Monotonic Scheduling**
  - A priority is assigned based on the inverse of its period
  - Shorter execution periods = higher priority
  - Longer execution periods = lower priority

Task scheduling

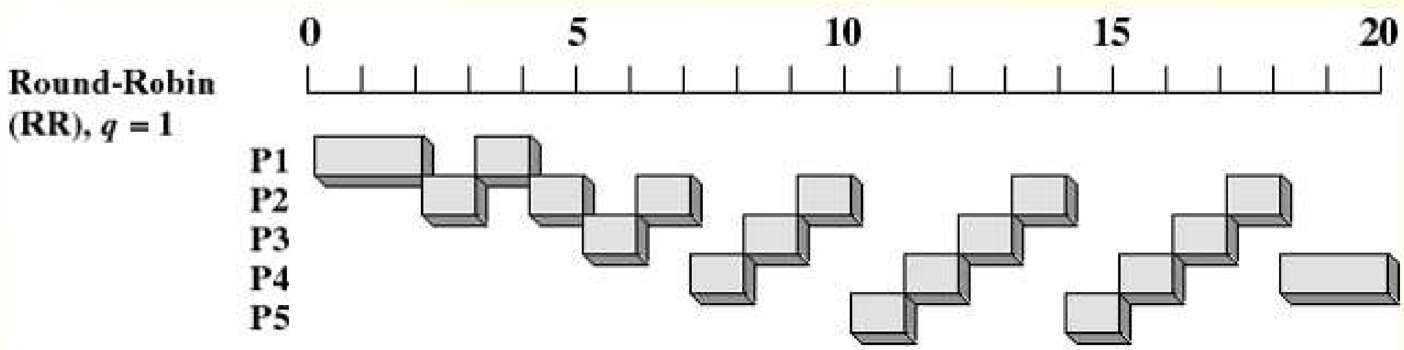
- **Earliest Deadline First Scheduling**
  - Priorities are assigned according to deadlines
  - The earlier the deadline, the higher the priority
  - Priorities are dynamically chosen

Task scheduling

- **Round robin scheduling**
  - Designed for time-sharing systems
  - Jobs get the CPU for a fixed time
  - Ready queue treated as a circular buffer
  - Process may use less than a full time slice

Example of Task scheduling (RR)

Process	Arrival Time	Service Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



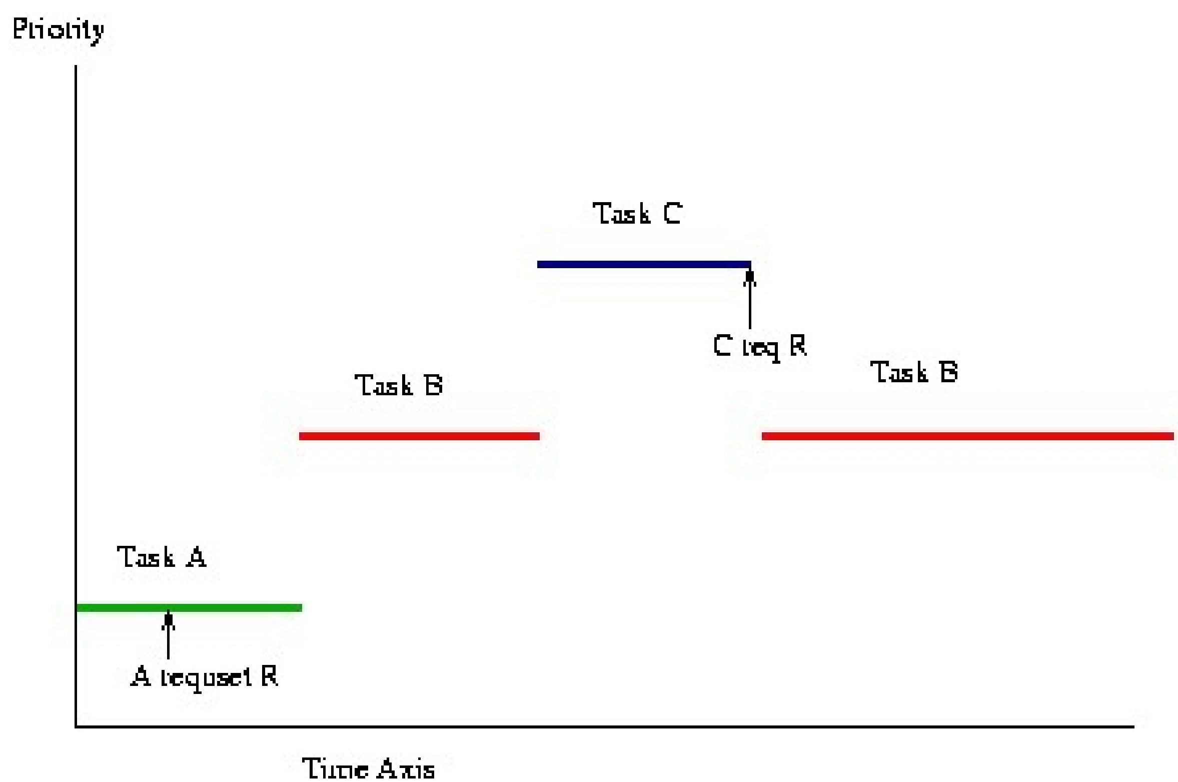
Time management

- **Time interrupt** : A high resolution hardware timer is programmed to interrupt the processor at fixed rate
- Each time interrupt is called a system tick
- The tick may be chosen according to the given task parameters

Resource Allocation in RTOS

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

Priority Inversion



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

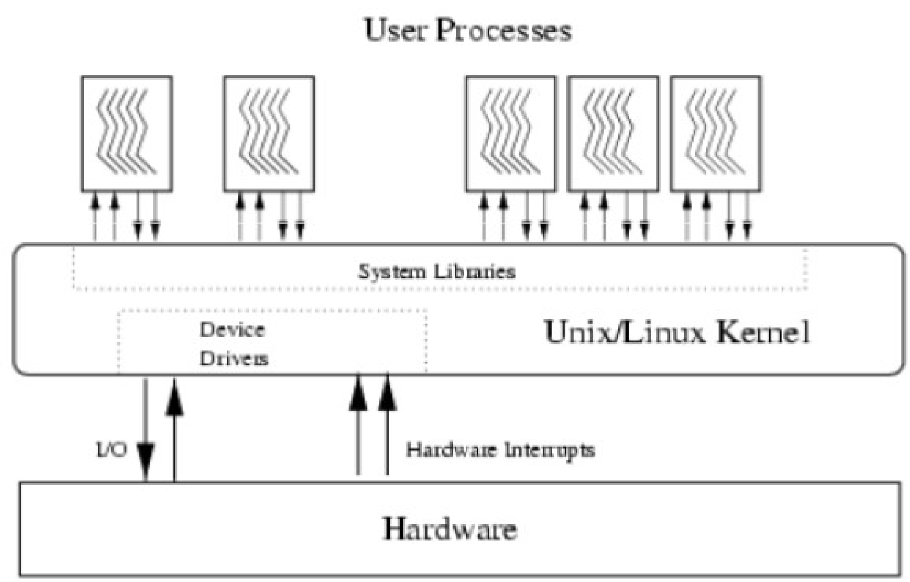
- Interrupt Latency should be very small
  - Kernel has to respond to real time events
  - Interrupts should be disabled for minimum possible time
- For embedded applications Kernel Size should be small
  - Should fit in ROM
- Sophisticated features can be removed
  - No Virtual Memory
  - No Protection

Existing RTOS categories

- Priority based kernel for embbeded applications
  - VxWorks, OSE, QNX
- Real Time Extensions of existing time-sharing OS
  - Real time Linux , Real time NT
- Research RT Kernels
  - MARS, Spring

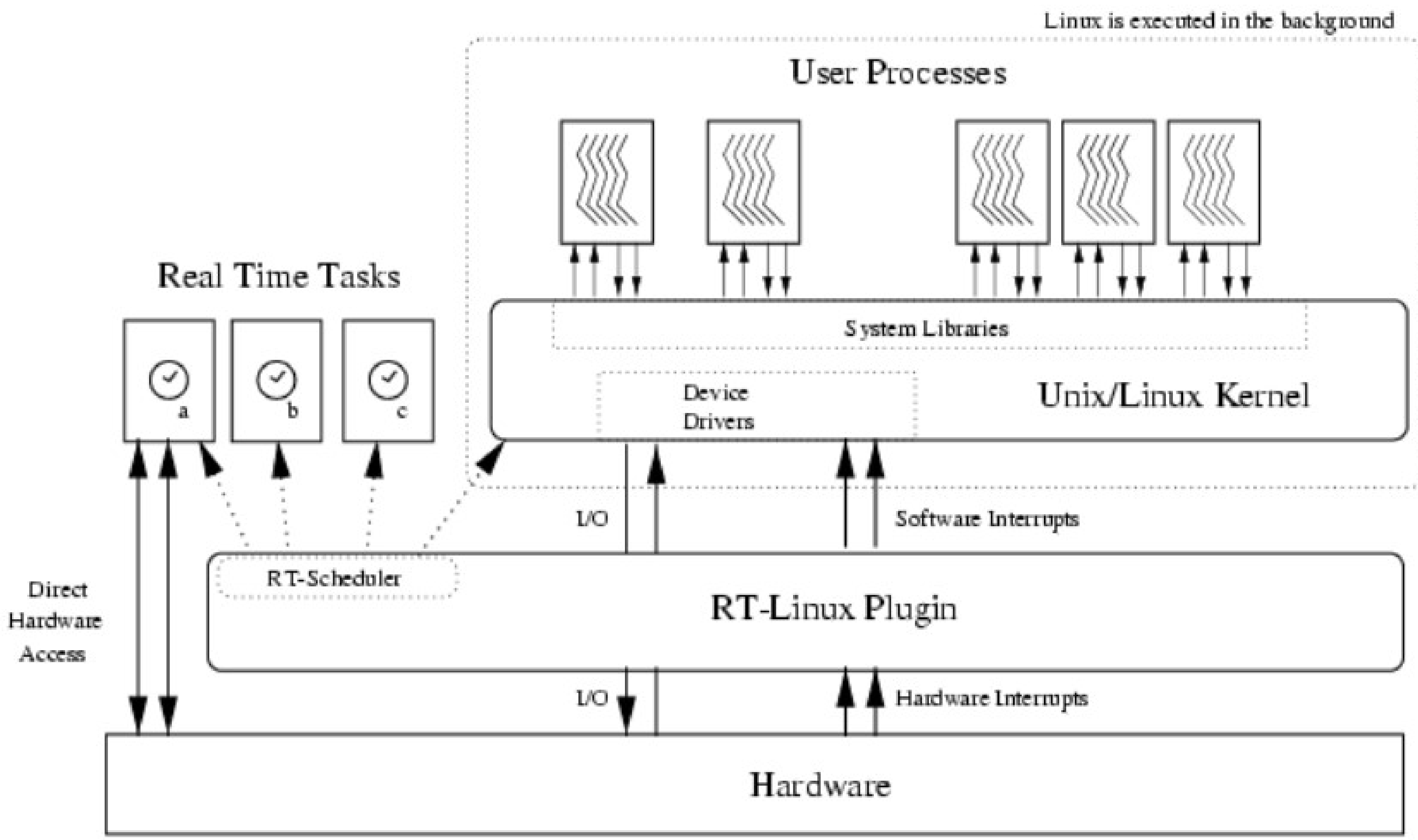
RT Linux: an example

- RT-Linux is an operating system, in which a small real-time kernel co-exists with standard Linux kernel



Non RT Kernel

RT Linux Kernel



Why Linux

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

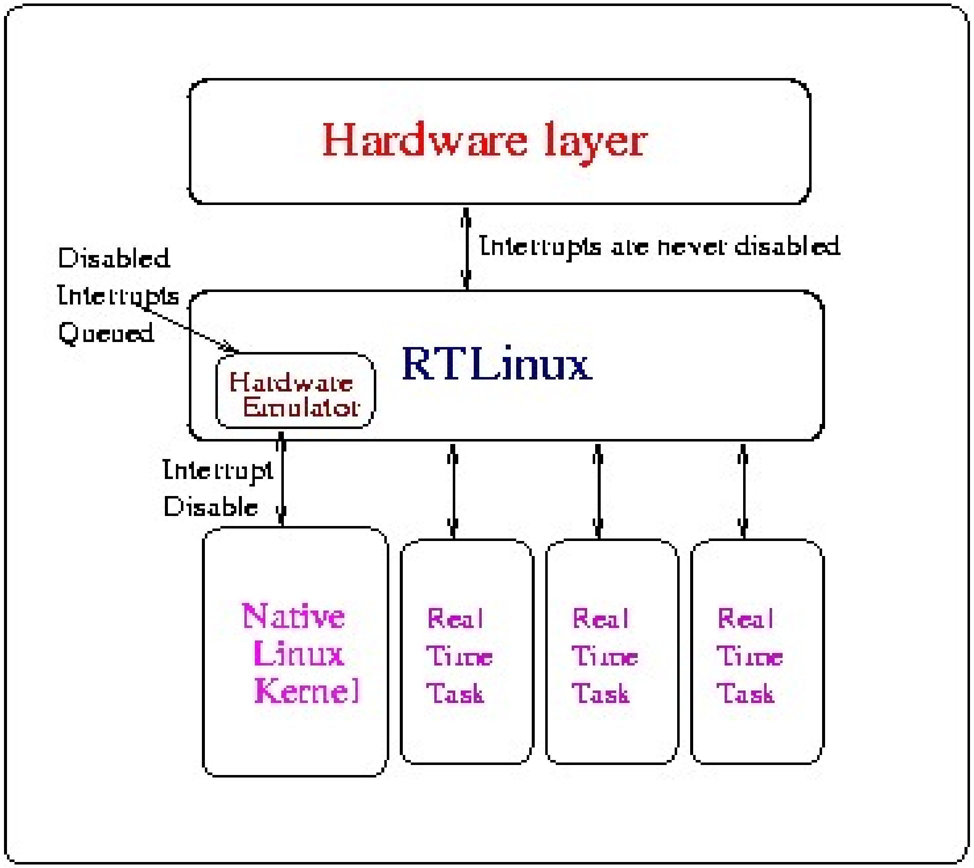
RTLinux

- 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



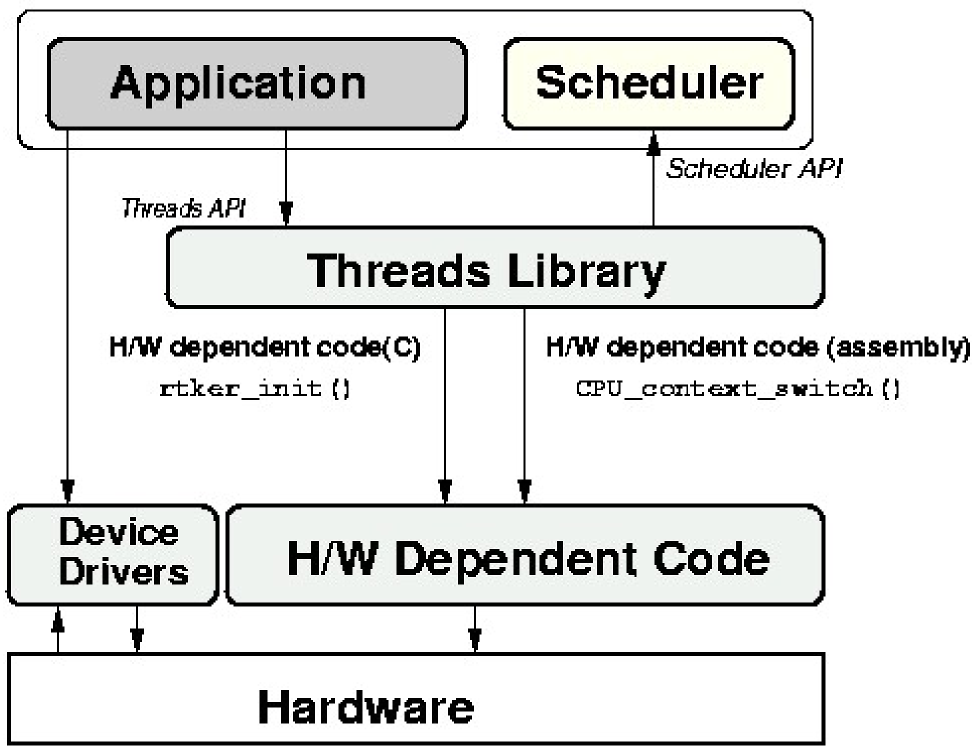
rtker – Our RTOS

- Motivation
  - Our own OS
    - Full grasp over source code – Easily modifiable, portable
- Features
  - Modular Design
    - Isolation of Architecture/CPU dependent and independent code – Easy to Port
  - Pluggable Scheduler
  - Two level Interrupt Handling
  - Small footprint
  - Oskit’s Device Driver Framework

Pluggable Scheduler

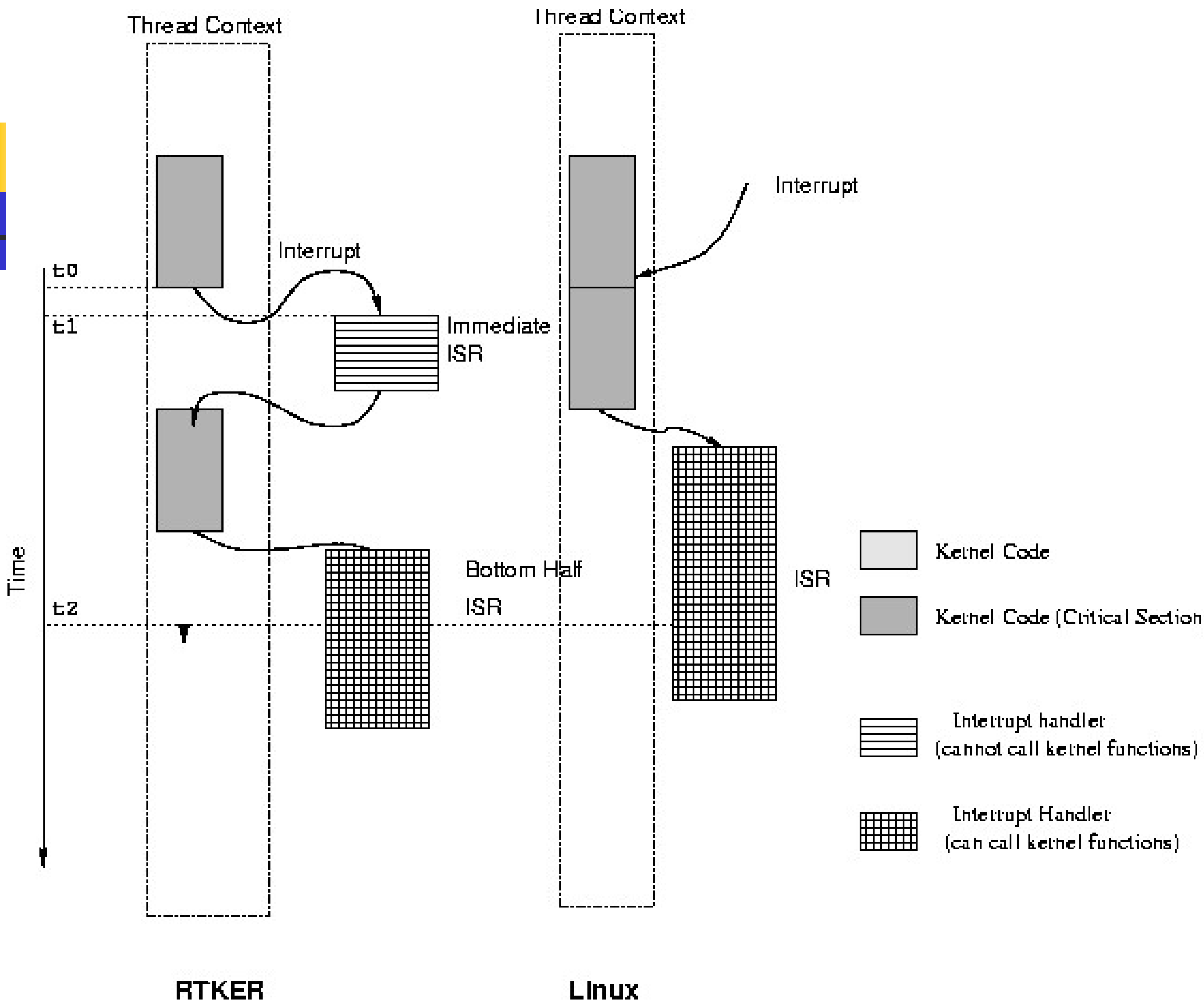
- Scheduler - part of the Application
- Kernel interacts with the scheduler through an API
- Application developer needs to implement the scheduler API
  - Can optimize on Data Structures & Algorithms for implementing the scheduler

rtker – Block Diagram

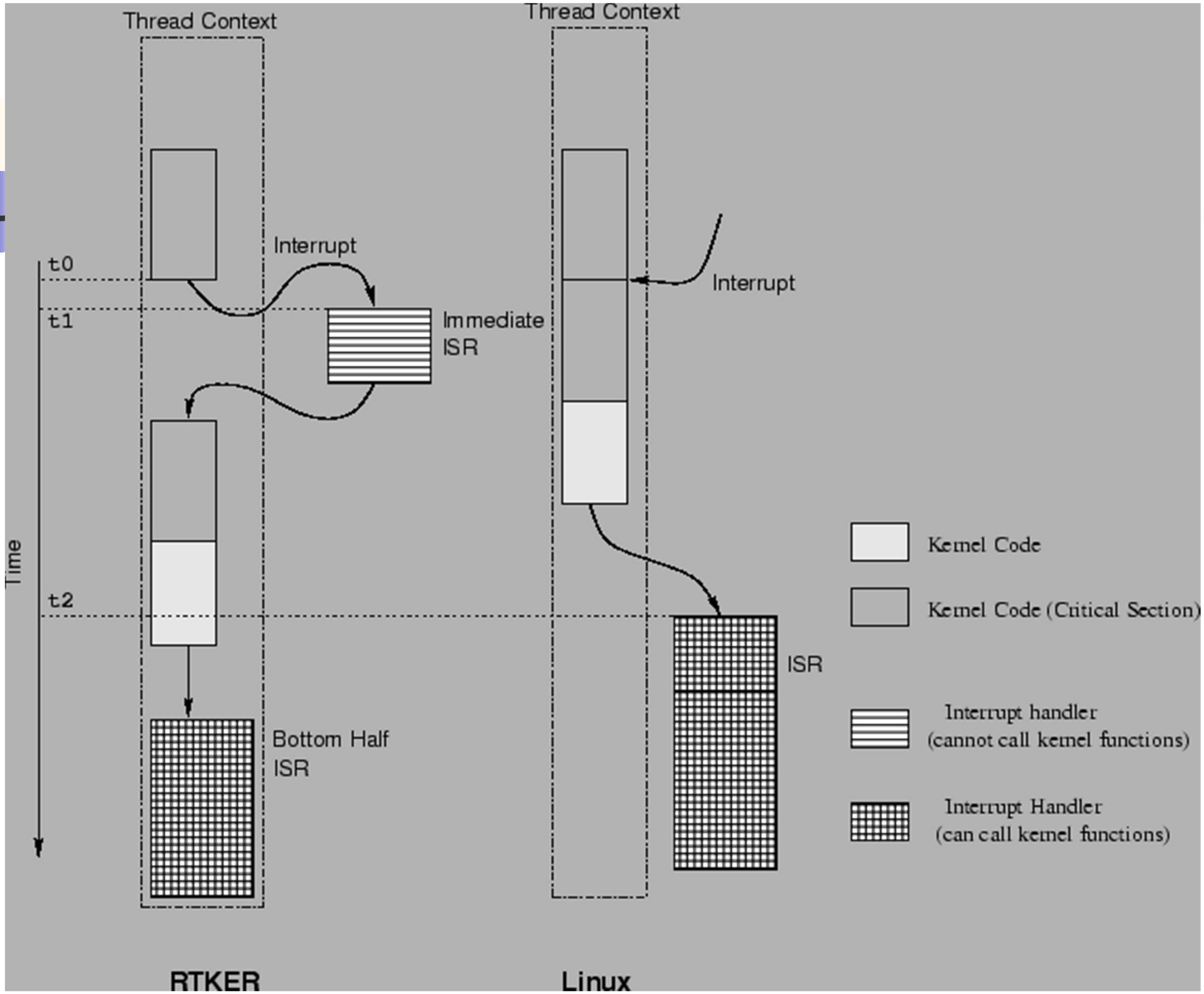


Two Level Interrupt Handling

- Two level Interrupt Handling
  - Top Half Interrupt Handler
    - Called Immediately – Kernel never disables interrupts
    - Cannot invoke thread library functions - Race Conditions
  - Bottom Half Interrupt Handler
    - Invoked when kernel not in Critical Section
    - Can invoke thread library functions
- Very Low Response time (as compared to Linux)



Thread Context
Thread Context
Interrupt
Interrupt
t0
t1
t2
Time
Immediate
ISR
Bottom Half
ISR
ISR
Kernel Code
Kernel Code (Critical Section)
Interrupt handler
(cannot call kernel functions)
Interrupt Handler
(can call kernel functions)
RTKER
Linux



Thread Context
Thread Context
Interrupt
t0
t1
Immediate
ISR
Interrupt
Time
t2
Kernel Code
Kernel Code (Critical Section)
ISR
Bottom Half
ISR
Interrupt handler
(cannot call kernel functions)
Interrupt Handler
(can call kernel functions)
RTKER
Linux

Other Features

- Footprint
  - Small footprint (~50kb)
- Oskit’s Device Driver Framework
  - Allows direct porting of existing drivers from Linux.
  - Example – Ethernet Driver of Linux

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

Peripheral devices and protocols

- Interfacing

  Serial/parallel ports, USB, I2C, PCMCIA, IDE
- Communication

  Serial, Ethernet, Low bandwidth radio, IrDA, 802.11b based devices
- User Interface

  LCD, Keyboard, Touch sensors, Sound, Digital pads, Webcams
- Sensors

  A variety of sensors using fire, temperature, pressure, water level, seismic, sound, vision

Conclusion

- RTOS is an OS for response time controlled and event controlled processes. The processes have predicable latencies and execute by pre-emptive scheduling
- An RTOS is an OS for the systems having the hard or soft real timing constraints and deadline on the tasks

Any questions and queries???

Thankyou...!!!

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