Reference Model and Scheduling Policies for Real-Time Systems

Mayank Agarwal and Ankit Mathur

Dept. of Computer Science and Engineering,
Indian Institute of Technology Delhi
Outline of the Presentation

- Introduction to Real-Time Systems
- A Reference Model of Real-Time Systems
- Priority-driven Scheduling of Real-Time Systems
- An overview of Real-Time Kernel (RtKer)
- Conclusions
Introduction to Real-Time Systems

- Correctness based not only on result but also the time-frame within which delivered
- Hard-Real Time Systems
  - Collision Detection
- Soft-Real Time Systems
  - Multimedia
Hard vs Soft Real-Time Systems

- Hard Real-Time Systems:
  - Require validation that timing constraints are met
  - Guaranteed service
  - No advantage in completing job before deadline

- Soft Real-Time Systems:
  - Emphasis on small response time and high throughput
  - Best Effort service
  - Should meet some *statistical constraints* of performance

- Main emphasis on **soft-real systems** involving Media applications

- Emphasis also on vision applications with hard deadlines - tracking, collision detection, etc.
Outline of the Presentation

- Introduction to Real-Time Systems
- A Reference Model of Real-Time Systems
- Priority-driven Scheduling of Real-Time Systems
- An overview of Real-Time Kernel (RtKer)
- Conclusions
Components of a model:

1. Workload Model - Applications supported by system. It has following the following parameters:
   - Temporal Parameters
   - Functional Parameters
   - Resource Parameters
   - Interconnection Parameters

2. Resource Model - System resources available to applications

3. Scheduling Model - How application system uses resources
Job - A unit of work done by the system

Task - A set of related jobs which jointly provide a system function

**Temporal Parameters of Task T1**

J11 and J12 are Jobs of task T1

- **Period (p)**: Time interval between two consecutive occurrences of a task.
- **Absolute Deadline (d)**: Time by which the task must be completed.
- **Relative Deadline (D)**: Time by which the task must be completed relative to the start of the period.
- **Execution Time (e)**: Time taken to execute the task.

Sample Code for T1
```
{  //compute.......  
  //communicate ....
}
```
These parameters give the timing constraints and behavior of a job

Periodic Task Model
- A Periodic Task $T_i$ is a sequence of jobs with identical parameters
- A periodic task $T_i(\phi_i, p_i, e_i, D_i)$ is described by:
  - Phase $\phi_i$ : Release time of first job in $T_i$
  - Period $p_i$ : Minimum Inter-release interval
  - Execution time $e_i$ : Max execution time of a job in $T_i$
  - Relative Deadline $D_i$

Under Srijan, tasks are purely periodic with well-defined parameters

Role of jitter in periodic task model

Aperiodic and Sporadic Tasks
Workload Model - Interconnection Parameters

- Defines how the job depends on others, how others depend on it
- Relationship '≺' can be defined based on job constraints and dependency
- Precedence Graph/ Task Graph defined on \((J, \prec)\).
- Some types of edge-interconnection parameters:
  - Data dependency
  - Temporal Dependency
- Temporal dependency is an important parameter in applications like LipSync
Workload Model - Functional Parameters

- Used to represent the intrinsic properties of a job.
- Some functional parameters:
  - Pre-emptivity
  - Criticality
  - Optionality of execution
  - Laxity type and laxity function
- In media applications, in case of overloading, we can skip frames (Optionality)
- RtKer doesn’t explicitly support optionality property
Resource Model

- Modeling of resources as
  - Active resources (Servers/Processors)
  - Passive resources (Resources)

- Processors: CPU, Transmission Link, Disks, etc.

- Processors carry out machine instruction, move data, retrieve files, etc.

- Resources: Memory, mutexes, locks, etc.

- Some Characteristics:
  - Context Switch Time
  - Pre-emptivity

- Parameters of Resource Vs Resource Parameters of Job
Application Model in Srijan

- Kahn Process Network (KPN): Interleaved computation and communication
- In original model, non-blocking writes and blocking reads
- However, due to limited buffer size, partially-blocking write

![MPEG Decoding Application Diagram](image-url)
Loop unrolling for analysis:

- Convert to tasks with no interleaving
- However, exact parameters of each job not predictable
- Static scheduling is possible but only assuming worst case behavior
Using modeling, a compact representation is possible

Modeling of the task graph using reference model, involves the following parameters:

- The process network consisting of the Tasks $T = (T_1, \ldots, T_i)$.
- The number of FIFO’s and their sizes
- The deadlines, periods, execution times, etc. for each of the tasks
- The data dependency between the tasks, in particular the partial blocking nature of the FIFO writes

Such a modeling may also be helpful in analysis and estimation
Application Model in Srijan (contd.)

- Other possibilities of such a representation:
  - These can be used for systematically validating timing constraints
  - Modeling will be required for specifying the tasks and scheduling policy to the **Scheduler Compiler**
Outline of the Presentation

- Introduction to Real-Time Systems
- A Reference Model of Real-Time Systems
- Priority-driven Scheduling of Real-Time Systems
- An overview of Real-Time Kernel (RtKer)
- Conclusions
Common Scheduling Approaches for Real-Time Systems

- **Clock-Driven approach**
  - All parameters of hard real-time jobs fixed and known
  - Schedule computed off-line and used at run-time
  - Scheduling made at specific time instants
  - At each decision time, scheduler schedules according to pre-computed schedule

- **Weighted Round-robin approach**
  - Each job given some time slices in a round
  - Number of slices proportional to its weight
  - Used mainly for scheduling real-time traffic in high-speed switched networks

- However, our main interest is in **Priority-driven approach**
Priority Scheduling

- Also known as Greedy/List/Work-conserving Scheduling
- Never leave any resource idle intentionally
- On-line scheduling
- Queues ordered by priorities of jobs
- Scheduling decisions taken at event occurrences
- Job with highest priority from relevant queue executed
Dynamic vs Static Partitioning Schemes

- Dynamic System
  - Common priority queue for processors
  - Jobs are migratable

- Static Systems
  - Jobs are statically bound to processors
  - Jobs on each processor are scheduled by themselves
  - Dependent jobs on different processors must follow synchronization and resource-access control schemes

- In worst case, dynamic systems may have poorer response but average performance will be better

- No reliable validation techniques for dynamic systems

- In Srijan, tasks are statically partitioned and are bound to fixed processors
Priority Allocation Schemes

- Based on priority assignment, three categories of algorithms:
  1. Fixed priority
  2. Task-level dynamic priority
  3. Job-level dynamic priority
- Fixed-priority algorithms are more predictable, dynamic more optimal
- Most real-time systems are Task-level dynamic (and job-level fixed)
- RtKer provides two schedulers: EDF and FIFO; both are Task-level dynamic
Rate Monotonic (RM) algorithm: Shorter the period, higher the priority

**Example of RM Scheduling**

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T2</th>
<th>T1</th>
<th>T2</th>
<th>T1</th>
<th>T3</th>
<th>T2</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>0</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

T1 (4,1) T2 (5,2) T3 (20,5)
Fixed Priority Algorithms: RM and DM

- Rate Monotonic (RM) algorithm: Shorter the period, higher the priority

Example of RM Scheduling

- Deadline Monotonic (DM) algorithm: Shorter the deadline, higher the priority

Example of DM Scheduling
Dynamic Priority algorithms: EDF and LST

Example of EDF Scheduling

- EDF and LST are optimal:
  - For single processor, with pre-emptable jobs, and no resource contention
- EDF and LST are not optimal:
  - For multi-processors
  - When preemption is not allowed
  - In overloaded systems
- Thus, we might need good overload and overrun management strategies
Validation of Timing Constraints in Real-Time Systems

The Validation Problem: Given the information:

1. the period $p_i$, execution time $e_i$, and relative deadline $D_i$ of every task $T_i$ in the system $T = \{T_1, \ldots, T_n\}$ of independent periodic tasks
2. a priority-driven algorithm used to schedule the tasks in $T$ preemptively on one processor

determine whether all the deadlines of every task $T_i$, for every $1 \leq i \leq n$ are always met

- Validation algorithms based on specific workload models
- Fast algorithms well suited for on-line acceptance tests
- Efficiency and robustness other desired features in validation algorithms
Validation Example: Workload Model

- Every job is preemptable at any time
- Once a job is released, it never suspends itself and is ready for execution till completion
- Scheduling and context-switch time is negligible
- Scheduler is event driven and acts immediately upon event occurrences
- Every task (or job) has a distinct priority
- Every job in a fixed priority system is scheduled at a constant priority
Validation Example: Validation for EDF

Some parameters:
1. Schedulable utilization $u_k$ of task $T_k$ is given by $e_k / p_k$.
2. Schedulable utilization $U$ of the system is the sum of individual utilizations, $\sum u_k$.
3. Density of task $T_k$ is $e_k / \min(D_k, p_k)$.
4. Density of the system, $\Delta$ is the sum of densities of all tasks.

Theorem: A system $T$ of independent, preemptable processes with relative deadlines equal to their periods can be feasibly scheduled on one processor iff its total utilization, $U \leq 1$.

Test for EDF algorithm: $\Delta \leq 1$. 
Validation - Conclusions

- Numerous validation algorithms are available for different scheduling methods
- Some techniques are fast and suitable for on-line admission tests, others more robust
- Formal validation techniques used for providing Hard Real-Time guarantees
- In Srijan, our main emphasis is on soft real-time systems; thus validation will be mostly by testing and benchmarking
- However, since our focus is also on Vision-based applications, we might also deal with some formal validation techniques
Factors to be considered in workload model

- Non-preemptability
  - Due to usage of a limited resource that must be mutually exclusiven

- Self-Suspension
  - Invoking some external operation: I/O, remote procedure, etc.

- Context Switches
  - In SPARC architecture (on which Leon is based), context switch time is significant due to register windows
  - A crude estimate of Context Switch Time on Leon:
    \[
    (64 \times 3 + 10) / 25 \mu\text{sec} = 8 \mu\text{sec}
    \]
Factors to be considered (contd)

- **Limited-Priority Levels**
  - RTOS may have limited priority levels so, tasks (or jobs) may have nondistinct priorities
  - However, in RtKer, we can define up to $\text{Maxint}$ priority levels

- **Tick Scheduling**
  - Events may not be incorporated instantaneously, since scheduler might be clock driven.

- **Varying Priority in Fixed-Priority Systems**
  - Job might require different priorities in different segments (contention for resource etc.)
Outline of the Presentation

- Introduction to Real-Time Systems
- A Reference Model of Real-Time Systems
- Priority-driven Scheduling of Real-Time Systems
- An overview of Real-Time Kernel (RtKer)
- Conclusions
Introduction to RTKer

The Basic RTKer Kernel supports

- Threads
- Semaphores
- Real-time interrupt handling
- Context switching
- Minimal C library for basic functions (printf/malloc)
The salient features of RTKer are:

- Two Stage Interrupt Handling - For minimum interrupt response time
- Pluggable Scheduler - User can add custom scheduling
- Modular design - Easy modification and porting
Interrupt Handling

- Interrupt Latency - Delay in execution of ISR from the time interrupt is raised

- Arises due to:
  - Kernel’s disabling of interrupts to protect data structures from ISRs
  - Kernel’s processing time to call the ISR
  - Device driver’s disabling of interrupts because of device specification requirement

- RTKer tries to reduce the latency arising out of need for protection of data structures
RTKer supports two-stage interrupt handling routine to be registered

- **Immediate ISR**
- **Bottom Half ISR**

- Immediate ISR provides a mechanism for an immediate response
- Kernel library related functions are then postponed for the Bottom Half ISR
Interrupt Handling: RTKer vs Linux

- **Interrupt Handling Routine** with system calls
- **Immediate Interrupt Handler** w/o system calls
- **Kernel Critical Section**
- **Kernel Code**

Diagram:
- **t0** (Immediate ISR)
- **t1** (Interrupt)
- **t2** (Bottom Half ISR)

**RTKer** vs **Linux**
Acceptability of any OS including RTOSes depends on the support for various peripherals.

RTKer solves the problem of support for devices as:
- RTKer uses device drivers from popular OSes like Linux
- Uses OSKIT device driver framework
- Interface of the OSKIT framework to device driver is through glue code
struct tcb {
    
    struct reg_context thread_context;

    struct tcb *recoveryTask;

    u32_t magic_key;

};

- *thread_context* field is architecture specific and scheduler doesn’t need to access it
- *magic_key* field is available for debugging purposes
Semaphores Implementation

- Integrity is maintained through spin locks
- Kernel maintains global queue for threads with timeouts

![Diagram of semaphores implementation](image-url)
Functions defined within struct `sched_info` have to be implemented by user

```c
struct sched_info {
    sched_lock
    init
    ::
    heir_thread
    set_mode
    set_priority
    get_priority
    ::
}
```
Implementation for Portability

- Application
- Scheduler
- Threads Library
- Device Drivers
- Hardware Dependent Code
- Hardware
Implementation for Portability (Cont’d)

- Directory-wise separation for hardware dependent and hardware independent code with a clear API
- Only the hardware dependent functions need to be modified for porting to different architecture
  - CPU Context Initialize
  - CPU Context Switch
  - rtker Init function
  - C Library
  - Boot strap
  - Device Drivers
Outline of the Presentation

- Introduction to Real-Time Systems
- A Reference Model of Real-Time Systems
- Priority-driven Scheduling of Real-Time Systems
- An overview of Real-Time Kernel (RtKer)
- Conclusions
Outline of the Presentation

- Important issues in a Multiprocessor environment:
  - Partitioning
  - Scheduling
  - Inter-process communication

- Work to be done on RtKer and in SPN